

Proceedings of CGAIDE'2004 : 5th Game-on international conference on computer games: Artificial intelligence, design and education, 8-10 November, 2004

| | |
|---------------|--|
| Item Type | Edited book |
| Citation | Medhi, Q., Gough, N., Natkin, S. and Al-Dabass, D. (eds.) Proceedings of CGAIDE'2004. 5th Game-On International Conference on Computer Games: Artificial Intelligence, Design and Education 8-10 November, 2004, Microsoft Academic Campus Reading UK. |
| Publisher | University of Wolverhampton, School of Computing and Information Technology |
| Download date | 2025-07-19 05:08:46 |
| Link to Item | http://hdl.handle.net/2436/625533 |

Proceedings of CGAIDE'2004

5th Game-On International Conference on Computer Games: Artificial Intelligence, Design and Education

8-10 November, 2004

Hosted by

**Microsoft Academic Campus
Reading UK**

Organised by

University of Wolverhampton

in association with

Microsoft Academic, UK

and

**Society for Modelling and Simulation (SCS-Europe)
Institution of Electrical Engineers (IEE)
British Computer Society (BCS)
Digital Games Research Association (DiGRA)**

Edited by:

Quasim Mehdi and Norman Gough

Guest Editors:

Stéphane Natkin and David Al-Dabass

**Published by The University of Wolverhampton
School of Computing and Information Technology
Printed in Wolverhampton, UK**

©2004 The University of Wolverhampton

Responsibility for the accuracy of all material appearing in the papers is the responsibility of the authors alone. Statements are not necessarily endorsed by the University of Wolverhampton, members of the Programme Committee or associated organisations. Permission is granted to photocopy the abstracts and other portions of this publication for personal use and for the use of students providing that credit is given to the conference and publication. Permission does not extend to other types of reproduction nor to copying for use in any profit-making purposes. Other publications are encouraged to include 300-500 word abstracts or excerpts from any paper, provided credits are given to the author, conference and publication. For permission to publish a complete paper contact Quasim Mehdi, SCIT, University of Wolverhampton, Lichfield Street, Wolverhampton, WV1 1EL, UK, q.h.mehdi@wlv.ac.uk.

All author contact information in these Proceedings is covered by the European Privacy Law and may not be used in any form, written or electronic without the explicit written permission of the author and/or the publisher.

Published by The University of Wolverhampton, School of Computing and Information Technology

ISBN 0-9549016-0-6

Contents

| | |
|---|------------|
| Programme Committee | 4 |
| Preface | 5 |
| Proceedings | 6 |
| Keynote Presentations | 7 |
| Tools and Systems for Games | 19 |
| Graphics Developments and Simulation | 65 |
| Games Art, Design, Modelling and Development | 95 |
| Mobile and Multi-user Games | 127 |
| Neural Networks in Games | 161 |
| Intelligent Agents | 201 |
| AI Tools for Games | 257 |
| Social/Humanities Aspects of Games | 299 |
| Education for Games Design and Development | 337 |
| Learning and Adaptation in Games | 370 |
| Author Index | 415 |

Programme Committee

General Conference Chair

Quasim Mehdi
RIATec, School of Computing &
Information Technology
University of Wolverhampton, UK

General Programme Chair

Norman Gough
RIATec, School of Computing &
Information Technology
University of Wolverhampton, UK

General Programme Co-Chair for Student Presentations

Stuart Slater
School of Computing & Information Technology
University of Wolverhampton, UK

Microsoft Coordinator

Gavin King

International Programme Committee

Don Anderson
Intellas Group LLC, Quantum Int.
Corp., USA

Professor David Al-Dabass (Guest
Editor), Nottingham-Trent University,
UK

Professor Marc Cavazza
University of Teesside, UK

Dr Darryl Charles
University of Ulster, UK

Professor Peter Cowling
University of Bradford, UK

Dr Johannes Fuernkranz
Technical University of Darmstadt,
Germany

Dr. John Funge
iKuni Inc, Palo Alto, USA

Dr. William Godfried
University of East London, UK

Professor Ian Marshall
University of Coventry, UK

Dr Stefan Grünvogel
NOMADS Lab, Cologne, Germany

Dr Daniel Livingstone
University of Paisley, UK

Dr Steve Maddock, University of
Sheffield, UK

Dr Stephen McGlinchey
University of Paisley, UK

Professor Stéphane Natkin (Guest
Editor), CNAM Paris, France

Professor Yoshihiro Okada
Kyushu University, Fukuoka, Japan

Dr Jon Purdy
University of Hull, UK

Professor Leon Rothkrantz
University Delft. The Netherlands

Dr Jason Rutter
University of Manchester, UK

Pieter Spronck
University of Maastricht, The
Netherlands

Dr Ian Wright
iKuni Inc, Palo Alto, USA

Conference Administrator: Tarvinder Kaur

Preface

On behalf of the Programme Committee we welcome all delegates to our 5th Game-On International Conference on Computer Games. Our previous Game-On events were held in London and this year we are privileged to bring the conference to the Microsoft Academic Campus at Reading. In addition, the theme has been expanded to include Artificial Intelligence, Design and Education, which reflects the wishes of last years' delegates. Since its inception, the conference has enjoyed a steady growth in popularity and the number of participants has doubled each year.

This year we have also benefited by being associated with *SCS*, *IEE*, *BCS* and *DiGRA*: The *Society for Modelling and Simulation (SCS-Europe)* was the original sponsor of the conference and we are pleased to be working with this organisation again; the *British Computer Society* has kindly donated prizes for the Best Student Demo/Poster sessions; and the *Digital Games Research Association (DiGRA)* - a fast-growing organisation for promoting games research internationally - has been associated with our *European Network for Digital Games Research* from the outset.

An important aspect of this conference is to provide a forum for MSc/MPhil/PhD students to present their work to their peers and to experts in the field. All papers have been reviewed by at least two eminent members of the Programme Committee who were delighted with the standard attained. Special thanks are due to all of these reviewers who have been most diligent in their task by providing detailed and useful feedback to authors. The best papers will be reviewed for possible inclusion in the *International Journal of Intelligent Games & Simulation* and as possible candidates for the Best Research Student Paper at the *Imagina Festival*, Monaco 2005.

The conference has been organised into 10 themes and you will find the following papers grouped into these themes. As previously, we have added an extra (optional session) so that we can review progress made in our *European Network for Digital Games Research*.

A big vote of thanks goes to Microsoft for so generously making available their excellent facilities for this conference and for providing valuable support. We particularly wish to thank Gavin King for his excellent cooperation and coordination and Joanna Smail for her assistance. Thanks are also due to Stuart Slater and Caroline Phillips for organising the student activities. It is especially pleasing that we have been able to provide places at no cost for MSPs (Microsoft Student Partners), thereby assuring greater student participation.

Last, but not least, the assistance of the School of Computing & Information Technology is appreciated for its endless support and particularly our Administrator Tarvinder Kaur.

We trust that you will all enjoy your stay in Reading and benefit from this conference.

Quasim Mehdi, General Conference Chair
Norman Gough, General Programme Chair
Wolverhampton, November 2004

Proceedings

Keynote Presentations

**Grand theft academic: Stealing from six disciplines to
build a model of interactive narrative in computer
games**

**Professor R. Michael Young
Head of the Liquid Narrative Group,
North Carolina State University**

**Microsoft at the Game Developers' Conference:
Talking and listening**

**Mike Pelton
Developer Platforms Group, Microsoft UK**

Game Design in Education

**Professor Mark H. Overmars
Institute of Information and Computing Sciences,
Utrecht University**

GRAND THEFT ACADEMIC: STEALING FROM SIX DISCIPLINES TO BUILD A MODEL OF INTERACTIVE NARRATIVE IN COMPUTER GAMES

R. Michael Young
Department of Computer Science
North Carolina State University
Raleigh, NC, 27695, USA
E-mail: young@csc.ncsu.edu

KEYWORDS

Interactive Narrative, Planning, Storytelling.

ABSTRACT

In this paper, we set out a basic approach to the modeling of narrative in games. This approach adopts a bipartite model taken from narrative theory, in which narrative is composed of *story* and *discourse*. In our approach, story elements – plot and character – are defined in terms of plans that drive the dynamics of a virtual environment. Discourse elements – the narrative’s communicative actions – are defined in terms of discourse plans whose communicative goals include conveying the story world plan’s structure.

INTRODUCTION

The number and type of interactive 3D games continue to grow as the processing power of commercial graphics cards increases. Many of these environments exploit informal adaptations of narrative techniques drawn from conventional narrative media in their design. Much of that work, however, conflates two central aspects of narrative structure that limit a) the range of techniques that can be brought to bear on the narrative’s generation and b) the range of narrative structures that can be generated for a given environment. These two aspects of narrative are the structure of story and the structure of narrative discourse.

In this paper, I describe an approach to the generation of narrative-oriented interaction within games that treats story and discourse as its two foundational elements. In this approach, I adapt models of narrative from narrative theory, computational linguistics and cognitive psychology, integrating these approaches with techniques from artificial intelligence in order to create intelligent narrative-oriented games.

BACKGROUND: STORY AND DISCOURSE

The work described here adapts and extends existing work in artificial intelligence to account for specific story-oriented applications within 3D virtual environments. This approach is based on concepts and methods first developed in narrative theory. Narratologists have provided an extensive characterization of narrative and its elements, describing the fundamental building blocks used by an author to create a compelling story (Chatman 1990; Rimmon-Keenan 2002). Narrative-theoretic approaches, however, are analytic in nature and do not directly lend themselves to a computational model capable of being used in a generative capacity. A central challenge of any computational approach that seeks to operationalize concepts from narrative theory is to determine appropriate methods to translate concepts derived from analysis into concrete, formal models capable of being put to use in the creation of a computer game.

While a broad range of approaches to the analysis of narrative exists, our work makes use of a structure that divides a narrative into two fundamental parts -- the *story* and the *discourse* (Chatman 1990; Emmot 1999) – and we construct distinct representations and tools to manage each. From a narratological perspective, a *story* consists of a complete conceptualization of the world in which the narrative is set. This includes all the characters, locations, conditions and actions or events that take place during the story’s temporal extent. Two fundamental components of a narrative – its plot and its characters – are defined within the story itself.

Distinct from the story, but closely tied to it, is the narrative *discourse*. Our discourse model represents those elements responsible for the *telling* of the story, rather than containing the story

elements themselves. This notion of discourse differs from its conventional meaning. Specifically, the discourse we are generating is not communication between the user and the characters within the story. Rather, it is concerned with that communication between the system and the user that conveys the storyline (which may include character dialog as individual elements).

In my approach, the construction of a narrative discourse can itself be divided into two conceptual aspects. One aspect is the determination of both the content of the discourse and its organization. To compose a narrative discourse, an author makes choices about those elements from the story to include in the story's telling and those elements to leave out. Further, the author determines additional information about the story-world to convey to the reader. Finally, the author must organize the discourse, determining what is to be told first, what second, etc, and how the sub-parts of the discourse should be arranged so as to achieve the intended communicative effects on a reader.

A second aspect to the generation of narrative discourse is the selection of the specific communicative resources to be used to convey the story's elements to the reader. In a 3D virtual environment, these resources include a range of media, from voice-over narration to 3D camera control to background music. The work that we describe here focuses on the generation of coherent, cinematic camera control, though our results are applicable to aspects of communicative actions across media.

GENERATING STORY AND DISCOURSE

Action and change are central to narrative. In most narratives, story-world action is initiated by characters as they attempt to achieve their individual and collective goals. Goals play a role at the discourse level as well; in film narratives a cinematographer acts in a goal-directed manner to build the cinematic discourse, intentionally composing shots to effectively communicate

unfolding story action. The goal-oriented focus in operation at both the story- and discourse-levels in conventional narrative media motivates us to use a *plan-based* model of the control of activity within games; we have constructed an architecture, called Mimesis, that uses this model to generate plans for controlling characters operating within a narrative as well as for controlling media resources used for telling the narrative. We briefly describe the Mimesis architecture here. More details can be found in (Young, et al 2004).

The Mimesis system integrates a suite of intelligent control tools with a number of existing virtual world environments and conventional programming environments. In this paper, we will restrict our discussion to applications built using Unreal Tournament (UT), a commercially available 3D graphical game engine. Mimesis overrides UT's default mechanisms for controlling its virtual environment, using instead a client/server architecture in which low-level control of the game environment is performed by a customized version of the game engine (called the *MWorld*) and high-level reasoning about narrative structure and user interaction is performed remotely by a suite of intelligent agents called *Mimesis Components* or *MCs* (see Figure 1).

Within Mimesis, the MCs act collectively as a narrative server, determining the narrative elements of the user's experience within the virtual world. The MCs are responsible for the generation of a story (in the form of a *story-world plan* characterizing all character actions that are to be performed within the environment) the generation of a *discourse plan* characterizing the media-specific communicative actions used to convey the story to the user, and the maintenance of a coherent narrative experience in the face of unanticipated user activity. At start-up, the MWorld sends a message to the MCs requesting a story. This request identifies a goal state for the story, the MWorld's current world state and the library of actions that are available for characters in the MWorld's world. The MCs then generate a the characters will execute in the story world. It

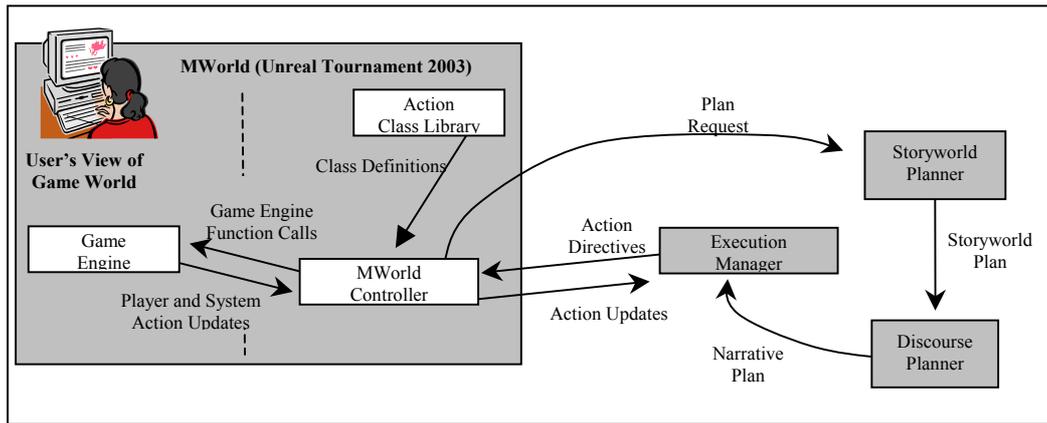


Figure 1. The Mimesis system architecture shown with an MWorld built using Unreal Tournament 2003 as a sample game engine. Individual gray boxes indicate components described below. Within the MWorld component, the vertical dashed line represents the boundary between code created by the Mimesis developer (to the right of the line) and that created by the game engine developer (to the left of the line).

sends this plan as input to the discourse planner, which generates a specification of the communicative action (in our case, 3D camera shot specifications) that will convey the elements of the story plan to the user. These two plans are integrated and passed to the *Execution Manager*, the component responsible for driving the story's action. The Execution Manager builds a directed acyclic graph whose nodes represent individual actions in the plans and whose arcs define temporal constraints between actions' orderings. The Execution Manager encodes nodes from the graph into XML messages and transmits these messages to the MWorld for execution as the action corresponding to each node becomes ready for execution. The MWorld translates the XML messages by using a one-to-one mapping from the action types of the nodes in the Execution Manager's graph to game engine functions and from the parameters of each action to instances of game engine objects in order to construct function calls that will drive the appropriate animations and state changes within the virtual world.

Creating the Storyworld Plan

The plan structures that we employ are produced by the DPOCL (Decompositional Partial Order Causal Link) planner (Young, et al 1994b). DPOCL plans contain many of the structures common to least-commitment planners (Penberthy and Weld 1994): steps and ordering constraints and causal connections between them. Further, DPOCL plans contain information about the

hierarchical structure of a plan, similar to the representation used by hierarchical task network (HTN) planners (Sacerdotti 1977). Because action sequences within narratives are often *episodic* that is, because they follow common patterns of action, hierarchical structures are highly amenable to representing story fragments.

Adopting a plan-based model of story structure allows the system to compose new stories in response to novel starting states or goal specifications, or to customize a story based on a user's interests and knowledge. An additional benefit of using the DPOCL plan representation to drive a narrative is in the plan's structural correspondence to a user's mental model of the story it defines. Recent research (Ratterman, et al 2002; Young 1999) suggests that hierarchical causal link plans like DPOCLs, as well as the techniques used by the DPOCL algorithm to create them, make for effective models of human plan reasoning. Our empirical studies indicate that the core elements of DPOCL plans match up with the models of narrative structure defined and validated by psychologists (Christian and Young 2004).

Creating the Narrative's Discourse

A narrative system must not only create engaging story-world plans, it must use its resources to tell the story effectively. In this paper we discuss one particular strategy used in the effective creation of a narrative: building narrative discourse involves the central task of determining the content and

organization of a sequence of camera shots that film the action unfolding within a story world.

We build on our previous research on the generation of natural language discourse to generate discourse plans for controlling an automated camera that is filming the unfolding action within a 3D story-world. To create these discourse plans, we use a discourse planning system named Longbow (Young, et al 1994a) The Longbow planner is built on the core DPOCL algorithm, and so the two planners' representations are quite similar. In our approach, 3D camera shots and shot sequences are viewed as planned, intentional action whose effects obtain in the cognitive state of the user. Individual camera shots are treated as primitive communicative actions, multi-shot sequences and cinematographic idioms are characterized using hierarchical plan operators, and, as in conventional discourse planning, plan structure that specifies the communicative content of a discourse is created to achieve particular effects upon the mental state of the user.

Conventional discourse planners take as input a set of propositions intended to be conveyed to the user of a system, along with a model of the user's existing knowledge of the domain of discourse and a library of plan operators describing both the primitive communication actions available to the planner (e.g., typically speech acts such as INFORM or REQUEST) and definitions for a set of abstract actions and their sub-plan schemas, sometimes referred to as *recipes*. Abstract operators often specify rhetorical structure (Mann and Thompson 1987) in a discourse (e.g., when one part of a discourse stands as evidence for the claim set forth in a second part of a discourse) and their sub-plan schemas specify how more primitive collections of communicative actions can be combined to achieve the abstract act's communicative effects.

There are several important ways that the task of narrative discourse generation – and our approach to it – differ from the task of discourse generation in conventional contexts. In our approach, the propositional content that the narrative discourse planner receives as input refers not just to relations that hold in the domain of discourse, but also to propositions describing the structure of the story-world plan. For instance, in addition to generating discourse that conveys the fact that a character has a gun, the narrative

discourse must also convey the action of the character using the gun to rob a bank. The task of the discourse planner is, in part, to generate camera action sequences that convey the execution of story-world plan actions to the user.

Beyond the requirement to communicate a different type of content in narrative discourse, our approach to the generation of plans for 3D narrative discourse addresses two key problems. First, the narrative discourse that we generate must contain structure beyond that which simply mirrors the structure of the actions executed in the story world. Cinematic discourse contains both rhetorical structure, aimed at conveying propositions about the story world to a user and idiomatic structure mirroring the use of patterns for shot composition used in film (Arijon 1976). Our plan operators capture these aspects of discourse structure and combine them effectively to tell the story.

A second key problem addressed by our approach to discourse planning is the temporal integration of the story-world and discourse-level plans. The actions in discourse plans for narrative in virtual worlds, unlike actions in plans for textual narrative, must themselves execute. Camera actions for panning, tracking, fading, etc, all require time to play out, a physical location from which the camera films, physical objects that must be included or excluded from the field of view, etc. A particularly complicating aspect of this is that these camera actions must execute in the same temporal and spatial environment as the objects of the story that they must convey to the user. A knowledge representation for narrative discourse must take this shared environment into account or risk creating suboptimal plans.

In order to allow the operator writer to specify the temporal relationships between the execution of camera actions and the story-world actions that they must film, primitive camera actions in the discourse planner can be annotated with temporal constraints between the two plans. These constraints relate the start and end times of the camera actions to the start and end times of the actions that they film.

SUMMARY AND CONCLUSIONS

In this paper, we have set out a basic approach to the modeling of narrative in interactive virtual worlds. This approach adopts a bipartite model of

narrative as story and discourse in which story elements – plot and character – are defined in terms of plans that drive the dynamics of a virtual environment. Narrative discourse elements – the narrative’s communicative actions – are defined in terms of discourse plans whose communicative goals include conveying the story world plan’s structure. While there are many possible means to approach a story-and-discourse model of interactive narrative, our goal is to demonstrate the effectiveness of this model using the Mimesis system as a test bed; our initial results, mentioned in the work we cite, are encouraging.

ACKNOWLEDGEMENTS

We are grateful to the many students that have contributed to the Liquid Narrative Group and the development of the Mimesis system. This work was supported by National Science Foundation CAREER award #0092586.

REFERENCES

Arijon, D, 1976. *Grammar of the Film Language*, Hastings House: New York.

Cavazza, M., Charles, F., Mead, S., 2002. Planning characters' behaviour in interactive storytelling. *Journal of Visualization and Computer Animation*, 13(2): 121-131.

Chatman, S. 1990. *Story and Discourse: Narrative Structure in Fiction and Film*, Cornell University Press.

Christian, D. and Young, R. M. 2004. Comparing Cognitive and Computational Models of Narrative Structure. To appear in the *Proceedings of the National Conference on Artificial Intelligence*.

Emmot, K. (1999) *Narrative comprehension: A Discourse Perspective*, Oxford.

Mann and Thompson 1987. Rhetorical structure theory: A theory of text organization. In Livia Polyani, editor, *The Structure of Discourse*. Ablex Publishing Corporation.

Rattermann, M. J., Spector, L., Grafman, J., Levin, H. and Harward, H. 2002. Partial and total-order planning: evidence from normal and prefrontally damaged populations, *Cognitive Science* 25(6); 941-975.

Rimmon-Keenan, S. 2002. *Narrative Fiction: Contemporary Poetics*, Routledge.

Sacerdoti, E. 1977. *A Structure for Plans and Behavior*. Elsevier, New York.

Young, R.M., 1999. Cooperative Plan Identification: Constructing Concise and Effective Plan Descriptions. *Proceedings of the National Conference of the American Association for Artificial Intelligence*, pp. 597-604. Orlando, FL.

Young, R. M., Moore, J. D., and Pollack, M., 1994a. Towards a Principled Representation of Discourse Plans, In the proceedings of the Sixteenth Conference of the Cognitive Science Society, Atlanta, GA.

Young, R. M., Pollack, M. E. and Moore, J.D., 1994b. Decomposition and causality in partial order planning, in *Proceedings of the Second International Conference on AI and Planning Systems*, Chicago, IL, pages 188-193.

Young, R. M, Riedl, M, Branly, M, Jhala, A, Martin, R.J. and Saretto, C.J., 2004. An architecture for integrating plan-based behavior generation with interactive game environments, in *The Journal of Game Development*, vol.1 issue 1, pages 51-70.

MICROSOFT AT THE GAME DEVELOPERS' CONFERENCE: TALKING AND LISTENING

Mike Pelton
Developer Platforms Group
Microsoft, Reading, UK
mpelton@microsoft.com

ABSTRACT

At the recent Games Developers' Conference, Microsoft outlined its plans for the XNA initiative, and outlined short- and longer-term plans for developer tools, talking too about the future of DirectX. But that was only the first day – the main body of the conference focused on the state of the gaming industry, and future trends, including a look at whether mobile devices will fulfil their potential as a gaming platform. This session will provide an overview of Microsoft's announcements, and will also offer an (inevitably subjective!) insight into the highlights of the event, with a look at the most surprising issues, observations, and discussions.

GAME DESIGN IN EDUCATION

Mark Overmars
Institute of Information and Computing Sciences
Utrecht University
3584 CH Utrecht, The Netherlands
E-mail: markov@cs.uu.nl

KEYWORDS

Game design, Education, Game Maker.

ABSTRACT

Computer games play a very important role in the life of most youth. Games offer many possibilities in education. Various people have studied the use of existing games or specially designed educational games. In this paper we consider the use of game design itself in an educational context, with a focus on high schools. Game design can be used in many different subjects, ranging from languages and arts, to mathematics and computer sciences. We also introduce the *Game Maker* software package that can be used by students to easily create their games.

INTRODUCTION

Most kids, students and young adults love playing computer games. It has become an important part of their life. They are willing to spend vast amounts of time on improving their skills for playing these games and the span of attention they have for these game is very long. This is a clear reason for educators to investigate how games can be use in education. This has led to a range of educational games, the formation of a sub discipline of game design, named *serious gaming*, and an increasing number of researchers studying these phenomena and their implications for the education systems, see e.g. (Prensky 2001).

One approach is to use standard games in an educational context. Games like *SimCity* can create understanding of economic systems, *Super Monkey Ball* can teach kids about certain physics principles, and *Roman Total War* can give historic insight in the Roman Empire (see Figure 1). Unfortunately such games are not tuned toward the educational practice and, hence, are not easy to employ in the classroom. Some teachers also use



Figure 1. *Rome Total War* can be used to obtain insight in the way the Roman Empire expanded itself.

games to reward kids for their work but that seems a poor approach.

The more common approach is to create games that are particularly written for certain educational goals. Unfortunately, most of these games are of poor quality. There are many reasons for this. One is that the budgets for educational games are normally orders of magnitude smaller than those for normal games. As a result, kids will easily be disappointed with the result. But more importantly, game principles and educational goals are often conflicting. Without going in detail, there is on one hand the educational demand that often wants to put the control over what is learned in the hands of the teacher, while a crucial ingredient of interesting game play is that the player should have control over the action. While games can be good in providing insight, letting the player explore some domain, understanding intricate mechanisms, and making motivated choices, most educational games focus on learning specific facts and skills without providing an adequate motivation within the game world. There is still a long way to go before the educational system has changed such that games will play an important role there.

In this paper I want to explore a third way of using games in an educational context. Rather than using games I want to focus on the creation of games. Creating games requires a large number of skills which can easily relate to certain subjects in schools. Also, creating a game about some topic can be a good and motivating way of understanding certain material. Game design can be used in language education, arts education, computer science, physics, geography, and many other areas. I will indicate some of these possibilities and then concentrate on the *Game Maker* program that I wrote for this.

ASPECTS OF GAME DESIGN

Creating a computer games involves many different aspects. The game play must be defined, the story must be written, the characters must be designed, levels must be created, and interaction and behavior of computer controlled entities must be programmed. Usability tests are required to make sure the game satisfies the player's demands, and a marketing and promotion plan is required to actually sell the game. All these aspects can be used in an educational context.

Language classes for example could study the important aspects of stories in games. See for example (Glassner 2004) for an overview of the role of stories in games. Students could write a game story, for example for an adventure game. This would be highly motivating, in particular when their stories would be used in other classes to be turned into an actual game.

In art classes, rather than e.g. drawing portraits, students could design game characters that express a certain archetype, or they can paint game backgrounds that add the correct atmosphere to the game. They can design the 2-dimensional or 3-dimensional levels in which the game takes place. And, going a step further, they could design animations and even introductory movies for the games.

In music classes students could study game music and the effect it has on the player. They could investigate adaptive music that smoothly adapts itself to the game situation and they could

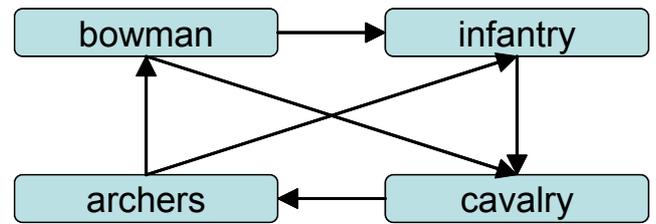


Figure 2. The relation between different units in a strategy game. An arrow indicated the relative strength, for example, the archers are stronger than the (horse) bowman.

compose their own game music using the various packages available.

MATHEMATICS

The core of a game consists of the rules that define the game play. Rules on one hand describe the inner mechanisms inside the game (for example how much damage a particular weapon does to an enemy) and on the other hand describe the moves the player can make. See (Salen and Zimmerman 2004) for an extensive study of rules in games. This applies equally well to board games, sport games, and computer games. A prime aspect of designing a good game is to come up with a set of rules that is consistent, balanced, and meaningful. This obviously requires creativity but also mathematical skills, in particular in logic and probability calculus.

Such systems of rules are very suitable for an educational context. For example, students could investigate the well-known rock-paper-scissors principle in which paper defeats rock, rock defeats scissors, and scissors defeats paper. Such a simple cyclic relation unfortunately leads to rather boring gameplay (the best strategy is to make random choices) but more complex relations can be studied as well, like the one depicted in Figure 2 that is rather common in real-time strategy games (Rollings and Morris 2003). Here students could deduce that for example infantry is not required to win a game. Schemes can be made more interesting by adding costs to entities (often called shadow costs in game design terminology) and calculations can be made to determine the correct relative costs between units. This leads to various aspects of the mathematical field of game theory. See e.g. (Dutta 2001) for an easy introduction.



Figure 3. Some high school students, during a workshop, are designing a game with some dice and colored stones.

Probability also plays an important role in games. Many games contain events that happen with a certain chance, for example the appearance of a special bonus. The game designer must choose such chances correctly to obtain interesting game play. This can be used to introduce probability theory to students. For example, students could be asked to design some sort of casino game in which they have to determine the correct height of the prices in relation to the chance that certain combinations of dice or playing cards appear. Even letting students design a simple game involving a few dice already leads to a fun challenge (see Figure 3).

COMPUTER SCIENCE

When I was taking my first steps in programming I was very excited when I could write a program to compute the first 100 prime numbers. Nowadays, computer applications provide access to music, video, and games, and novice programmers want to create similar programs. Unfortunately, such programs are difficult to create. Teachers have tried to raise student interest by using languages such as *Logo* (www.logosurvey.co.uk) that can create interesting drawings, or robots, such as the *Lego MindStorms* (www.legomindstorms.com). But using *Logo* to make drawings is no longer flashy enough, and robots are rather expensive and limited in their functionality.

Creating computer games on the other hand is a challenge that many students want to take on. Developing computer games involves many aspects of computing, including computer graphics, artificial intelligence, human-computer

interaction, security, distributed programming, simulation, and software engineering. It can be used as a vehicle to teach students about these topics.

Many teachers indicate that it is difficult for students to understand object-oriented programming. This is somewhat surprising because object-oriented design is very natural. In real-life we think in terms of objects with certain properties and behavior. Still, once people write a program they tend to adopt the traditional view of instructions being executed and control structures.

But when you are creating computer games, the situation changes. In a computer game, everything is an object: the monsters, wall segments, coins, bonuses, power-ups, and the guns and bullets. Thinking about creating games means thinking about objects and how they react to one another and to the player's input. So the game creator naturally thinks in an object-oriented way.

Also inheritance, a powerful but sometime difficult to grasp object-oriented programming concept, becomes much easier to understand in a game design context. Take, for example, the well-known class of games based on *Breakout*, in which the user must destroy stones by hitting them with a bouncing ball. All stones exhibit similar behavior but will appear in a variety of shapes and colors. These characteristics make it logical and efficient to create one stone object and specify its behavior, then create similar objects with different colors that inherit the original stone object's behavior. Making modified stones that override certain behavior also becomes an easy concept. For a more extended description see (Overmars 2004).

Finally games can be used to let students understand how to plan and execute a larger project. Creating games can involve different people (even from different courses) that must work together as a team. Writing design documents, performing usability studies, and setting up a good testing environment are crucial for the success of games.

GAME PROJECTS

Game design can also play a role in other subjects. For example, in areas like physics and chemistry students could create games to investigate or explain concepts like gravity, electricity, or chemical reactions. In economics they could design their own economic system for a simulated city. And in geography they could make games about the location of certain towns. With a bit of imagination you can come up with many more possibilities. In this way students can create their own educational games.

GAME MAKER

Creating computer games is not an easy process. Commercial games are the combined work of teams of 10 to 50 people and require budgets of millions of dollars. And even creating smaller, simpler games from scratch would be a complicated task requiring advanced programming skills. Fortunately, there are a number of software packages available that make it way easier to create computer games, replacing (part of) the programming by mechanism in which games are constructed from simple building block. Examples are *StageCast* (www.stagecast.com), that is particularly aimed at young kids, and the products by *ClickTeam* (www.clickteam.com). Many similar packages exist, several of which can be found at www.ambrosine.com/resource.html.

In this paper we will concentrate on *Game Maker* (www.gamemaker.nl), written by the author. Game Maker, is a rapid-application development tool currently used worldwide by young people at home and in schools to create two-dimensional and isometric games. Figure 4 shows the Game Maker interface, which uses an object-oriented, event-driven approach. With Game Maker's drag-and-drop techniques, users can create games without writing a single line of code, but it also includes an interpreted programming language. The program produces stand-alone games that can be distributed freely; a version of Game Maker itself is available for free as well.

Game Maker has become extremely popular during the past few years. In 2003, users downloaded over a million copies of the program. An active user community exists with many

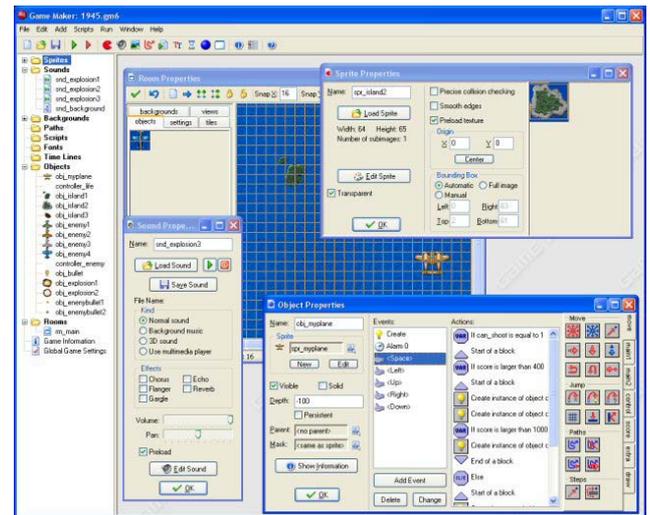


Figure 4. The Game Maker interface. The left side displays resources such as sprites and sounds, while the right side shows the object editor, room editor, and other property forms.

forums (see <http://forums.gamemaker.nl>). The youngest users, 8-year-olds, receive their introduction to computer programming through Game Maker. The oldest users are 80-year-old senior citizens.

Game Maker uses an object-oriented design concept as described above. Creating a game consists of defining objects. Some objects have a visual representation, such as an animated sprite. Others, like those that control game flow or maintain the score, might lack this feature. Multiple instances of the same object can appear in the game at the same moment.

Instances have properties. Some are built-in, like the speed with which the instance moves and the sprite used to represent it. Others can be defined, manipulated, and checked using actions or code. The user must define each object's behavior. While some objects, like wall segments, will have no behavior, others, like the avatar representing the player, will most likely have complicated behavior.

Game Maker defines behavior in event-driven terms. Events occur for objects, and the designer specifies actions that the game must execute when these events occur. Typical events include object creation or destruction, user input, collisions between instances, and alarm clocks. To achieve this, the game designer can simply drag and drop

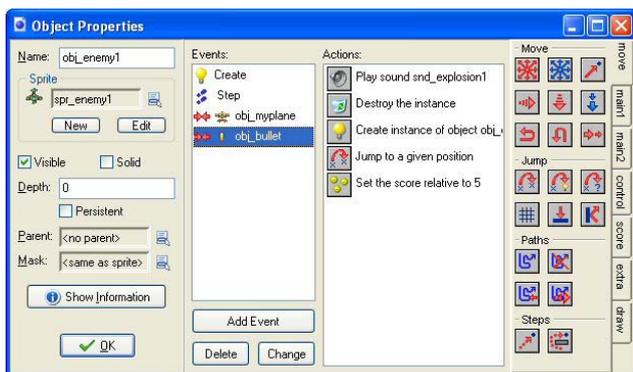


Figure 5. Object property form. The list of defined events for the enemy appears in the left center, while the actions that the game must perform when the enemy collides with a bullet appear to the right.



Figure 6. Some games created with Game Maker.

actions into events, as Figure 5 shows. Inheritance is achieved by simply setting the Parent field in an object. This indicates that behavior is inherited from another (parent) object.

Game Maker has more than 100 built-in actions, ranging from moving the object in a particular direction to playing a sound or displaying a high-score list. For more advanced tasks, the designer uses a code action to type in pieces of code that are executed when the event occurs. Within this code are close to 1,000 possible functions that can control all aspects of the game, including a particle system, network play functionality, and routines for 3D graphics. Students in general will start using the drag-and-drop actions but soon realize the use of writing pieces of code. In this way they are naturally introduced into the concept of programming.

Once the objects are defined (and the required sprites and sounds are added to the game) the designer can create rooms (or levels) using the room editor. Instances of objects are placed in the rooms and when the game is executed these instances come to life because of the actions in their creation events and they start reacting to each other and to the user input through the actions in their collision events and keyboard or mouse events.

Creating games with Game Maker is very efficient. After some experience with the program, a typical *Pacman* clone takes less than an hour to create. As a result the students can concentrate more on the design aspects of the games rather

than on all the details of getting the game to work. People have created all sorts of games with Game Maker, ranging from simple maze games and scrolling shooters to adventure games and strategy games. See Figure 6 for some examples of their creations.

CONCLUSIONS

Creating games appeals to all ages and to both males and females. It involves a lot more than programming, bringing together aspects of liberal arts, mathematics, social sciences and computer science. By using game design in an educational context you create an enthusiastic group of students that are eager to learn and who will find out that creating games can be even more fun than playing them.

REFERENCES

- Dutta, P.K., 2001. *Strategies and Games*, The MIT Press.
- Glassner, A., 2004. *Interactive Storytelling: Techniques for 21st Century Fiction*, A K Peters, Ltd.
- Overmars, M.H., 2004. "Teaching computer science through game design." *IEEE Computer* 37, no.4 (Apr): 81-83.
- Prensky, M., 2001. *Digital Game-Based Learning*, McGraw-Hill.
- Rollings A. and D. Morris, 2003. *Game Architecture and Design*, New Riders Publishing.
- Salen K. and E. Zimmerman, 2004. *Rules of Play, Game Design Fundamentals*, The MIT Press.

Tools and Systems For Games

| | |
|---|-----------|
| Hiromichi Fukutake, Yoshihiro Okada and Koichi Nijima 3D visual component based voice input/output interfaces for interactive 3D graphics applications | 20 |
| Burley, M. A., Gough, N. E., Mehdi, Q. H. and Natkin, S. Encoding sound by polynomial interpolation for intelligent dynamic music in computer games | 25 |
| Charles, D. and Black, M. Dynamic player modelling: A framework for player-centric digital games | 29 |
| Cheng, D. C. and Thawonmas, R. Case-based plan recognition for real-time strategy games | 36 |
| Hastings, E. J., Mesit, J., Guha, R. K. T-Collide: A temporal, real-time collision detection technique for bounded objects | 41 |
| Mesit, J., Guha, R. K. and Hastings, E. J. Optimized collision detection for flexible object in a large environment | 49 |
| Hartley, T., Mehdi, Q. H. and Gough, N. E. Applying Markov decision processes to 2D real-time games | 55 |
| Bancroft, M. and Al-Dabass, D. A combat simulator for dungeons and dragons | 60 |

3D VISUAL COMPONENT BASED VOICE INPUT/OUTPUT INTERFACES FOR INTERACTIVE 3D GRAPHICS APPLICATIONS

Hiomichi Fukutake¹, Yoshihiro Okada^{1,2} and Koichi Nijima¹

¹Graduate School of Information Science and Electrical Engineering, Kyushu University
6-1 Kasuga-koen, Kasuga, Fukuoka, 816-8580 JAPAN

²*Intelligent Cooperation and Control, PRESTO, JST*
{h-fuku, okada, nijima}@i.kyushu-u.ac.jp

KEYWORDS

3D Graphics software, *IntelligentBox*, Voice Commands, Voice Input Interface, Voice Output Interface, 3D Games.

ABSTRACT

A keyboard and a mouse device are still standard input devices used in 3D graphics software as well as in 2D software. However, the use of a keyboard or a mouse device is not suitable for wearable computer and augmented reality applications so that voice input/output interfaces are more significant. Especially for physically handicapped persons, voice input/output interfaces are very convenient. Moreover for the development of multimodal interaction games, voice input/output interfaces are necessary.

The authors have been studying component based 3D graphics software development systems and they have proposed *IntelligentBox* as their research prototype system. One application of *IntelligentBox* is the development of interactive 3D games. The authors have already introduced a video based motion input interface into *IntelligentBox* to enhance its ability for the development of various interactive 3D games. This time, the authors also introduced voice input/output interfaces into *IntelligentBox* for the development of multimodal interactive 3D games. This paper explains software components for voice input/output interfaces introduced into *IntelligentBox*. The authors also describe the usefulness of those components by showing their actual 3D graphics application examples of entertainment fields, e.g., 3D games.

INTRODUCTION

Advances in recent computer hardware technology have made possible 3D rendering images in real time. Consequently, 3D software has become in great demand although its development is more laborious work than 2D software development. For this reason, Okada and Tanaka developed a 3D prototype system called *IntelligentBox* (Okada and Tanaka 1995, 1998). *IntelligentBox* provides various 3D reactive objects called *boxes*. *Boxes* are manually operable objects, which have a 3D visible shape and a unique functionality. *IntelligentBox* also provides a dynamic data linkage mechanism called 'slot connection' so that users can construct interactive 3D graphics applications by combining already existing *boxes* through direct manipulations on a computer screen.

Application fields of *IntelligentBox* include 3D animation creation, virtual reality software development, 3D interactive simulator development and so on (Okada, et al. 2000). *IntelligentBox* would be also useful for the development of

wearable computer and augmented reality applications. For these kinds of applications, voice input/output interfaces are more significant. Especially for physically handicapped persons, voice input/output interfaces are strongly recommended. Moreover for the development of multimodal interactive games, voice input/output interfaces are necessary. Another application of *IntelligentBox* is the development of interactive 3D games. Therefore we introduced voice input/output interfaces into *IntelligentBox* to enhance its ability for the development of multimodal interactive 3D games. This paper explains software components for voice input/output interfaces introduced into *IntelligentBox*. We also describe their usefulness by showing their actual 3D graphics application examples of entertainment fields, e.g., 3D games. There are some researches about voice input interfaces. Igarashi and Hughes proposed the use of non-verbal features in voice, like pitch, volume, and continuation, to directly control interactive applications (Igarashi and Hughes 2001). The SUITEKeys system is a speech user interface for physically handicapped users. This interface provides accesses to all available functionalities of a computer by modeling interactions at the physical keyboard and mouse level (Manaris and Harkreader 1998; Manaris et al. 2001). Our research purpose is to propose component based software architecture that makes it easier to develop various 3D graphics applications including interactive 3D games, wearable computer and augmented reality applications. In this paper, we wish to insist the availability of component based approach to provide voice input/output interfaces for the development of multimodal interactive 3D games.

The remainder of this paper is organized as follows: First of all, next Section explains essential mechanisms of *IntelligentBox*. After that, we introduce component-based voice input/output interfaces. Furthermore, we show application examples of entertainment fields those use the voice input/output interfaces. Finally we conclude the paper in the last Section.

ESSENTIAL MECHANISMS OF INTELLIGENTBOX

IntelligentBox employs the following essential mechanisms, i.e., an *MD* structure and a slot-connection. In this section, we explain these mechanisms briefly.

Model-DisplayObject (MD) Structure

As shown in Figure 1, each *box* consists of two objects, a *model* and a *display object*. This structure is called an *MD (Model-Display object)* structure. Indeed a *display object* consists of two objects, a *view* and a *controller*. Therefore,

this structure is called an *MVC* structure. A *model* holds state values of a *box*. They are stored in variables called *slots*. A *view* defines how the *box* appears on a computer screen. A *controller* defines how the *box* reacts to user operations.

Figure 1 also shows messages between a *display object* and a *model*. This is an example of a *RotationBox*. A *RotationBox* has a *slot* named 'ratio' that holds a double precision number, which means a rotation angle. This value is normalized between zero and one. One means one rotation. Through direct manipulations on a *box*, its associated *slot* value changes. Furthermore, its visual image simultaneously changes according to the *slot* value change. Then a *box* reacts to user's manipulations according to its functionality.

Message-Sending Protocol for Slot Connections

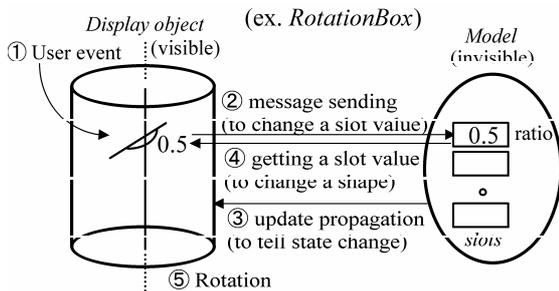


Figure 1. An MD structure of a *box* and its internal messages

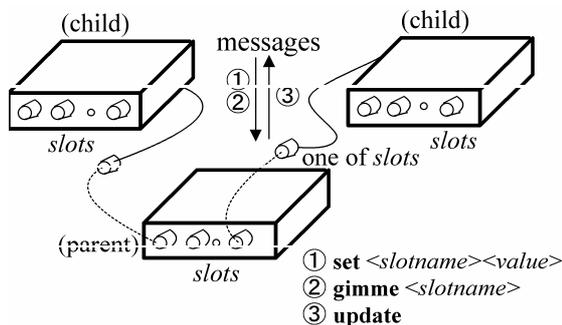


Figure 2. Standard messages between *boxes*

Figure 2 illustrates a data linkage concept among *boxes*. As shown in the figure, each *box* has multiple *slots*. Each *slot* can

be connected to one of the *slots* of an other *box*. This connection is called a *slot connection*. The *slot connection* is carried out by some messages when there is a parent-child relationship between two *boxes*. There are three standard messages, i.e., a *set* message, a *gimme* message and an *update* message. These messages have the following formats:

- (1) Parent box *set* <slotname> <value>.
- (2) Parent box *gimme* <slotname>.
- (3) Child box *update*.

A <value> in a format (1) represents any value, and a <slotname> in formats (1) and (2) represents a user-selected *slot* of the parent *box* that receives these two messages.

A *set* message writes a child *box slot* value into its parent *box slot*. A *gimme* message reads a parent *box slot* value and sets it into its child *box slot*. *Update* messages are issued from a parent *box* to all of its child *boxes* to tell them that the parent *box slot* value has changed.

Each *box* has three main flags that control the above message flow, i.e., a *set* flag, a *gimme* flag, and an *update* flag. These flags are properties of a *display object*. A *box* works as an input device if its *set* flag is set to true. Contrarily a *box* works as an output device if its *gimme* flag is set to true. A *box* sends *update* messages if its *update* flag is set to true. Then child *boxes* take an action depending upon the states of the *set* flag and the *gimme* flag after they receive an *update* message or after they individually change their *slot* values.

COMPONENT-BASED VOICE INPUT/OUTPUT INTERFACES OF INTELLIGENTBOX

Since *IntelligentBox* does not have any voice recognition and text-to-speech functionalities, we employed *Microsoft Speech API* and developed one server program for that. As shown in Figure 3, our server program called *VoiceServer* provides voice dictation and text-to-speech functionalities using *Microsoft Speech API*. *IntelligentBox* connects to the server using a standard TCP/IP socket communication by specifying the IP address and port number of the server. In the following subsections, we explain the functionality of *VoiceServer* and introduce voice input/output components of *IntelligentBox*. Among them are *VoiceInputBox*, *VoiceCommandBox* and *SpeechBox*.

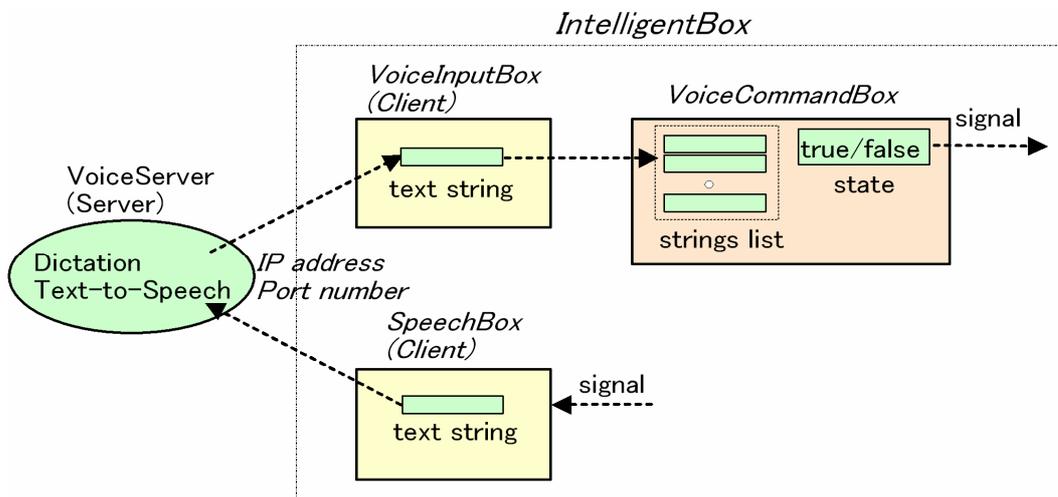


Figure 3. System configuration for voice input/output interfaces of *IntelligentBox*

VoiceServer

Microsoft Speech API has mainly two functionalities, i.e., voice dictation functionality and text-to-speech functionality. Microsoft also provides Speech SDK Ver. 5.1 (Microsoft SAPI) that includes development tools, libraries and sample programs. We modified one of the sample programs to make it work as a server and named it *VoiceServer*. Since this works as a server, it is possible to use both voice dictation and text-to-speech functionality from any other application programs. In the case of *IntelligentBox*, we developed two new components, *VoiceInputBox* as the client for voice input interface and *SpeechBox* as the client for voice output interface.

Voice Input Interface

For voice input interface, we developed two components, i.e., *VoiceInputBox* and *VoiceCommandBox*. *VoiceInputBox* works as the client that connects to *VoiceServer* and sends a certain message to ask the server to send a current dictated text string to the *VoiceInputBox*. The received text string is also sent to *VoiceCommandBox*. *VoiceCommandBox* has two modes, i.e., training mode and execution mode. Even if the user makes voices of the same text string, *VoiceServer* sometimes recognizes them as different strings. *VoiceCommandBox* must keep several text strings those are dictated as different strings when the user makes his/her voices as the same text string. This means a training mode. In the

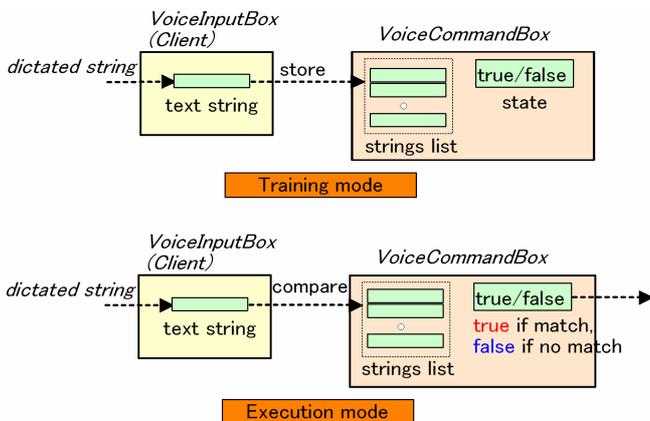


Figure 4. Components for voice command input and their message flow

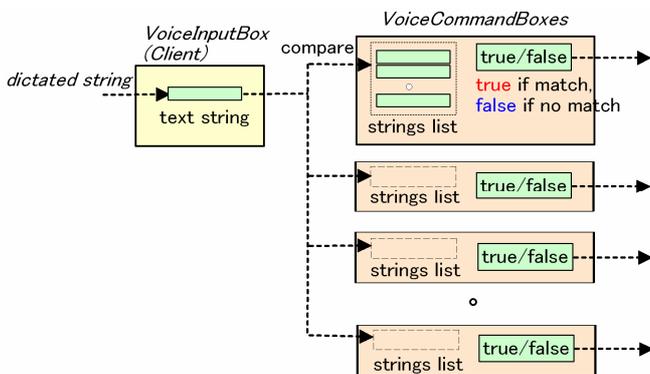


Figure 5. Components composition for multiple voice commands input and their message flow

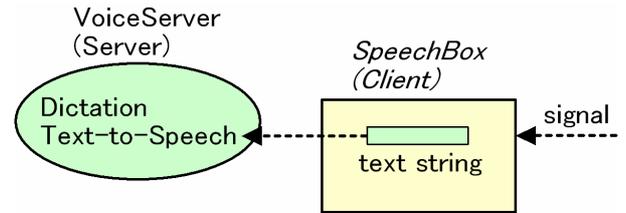


Figure 6. Component for voice output and its message flow

training mode, *VoiceCommandBox* always receives a dictated text string from *VoiceInputBox* and store it in a text strings list as shown in the upper figure of Figure 4. On the other hand, in the execution mode, *VoiceCommandBox* receives a dictated text string from *VoiceInputBox* and checks that it is the same as one of the already stored text strings in the strings list. When it is the same as one item of the strings list, 'state' slot of *VoiceCommandBox* becomes true as shown in the lower figure of Figure 4. Furthermore this 'state' slot value is sent to another composite box as a trigger signal to invoke a required action of the box. In this way, our voice command input interface is realized as the combination of software components, i.e., *VoiceInputBox* and *VoiceCommandBox*. Then, the user can realize his/her required voice command input interface for the already developed applications by adding the software components of the interface to the applications. Furthermore, as shown in Figure 5, multiple voice commands input interface can also be made using multiple *VoiceCommandBoxes*.

Voice Output Interface

For voice output interface, we developed only one component called *SpeechBox*. *SpeechBox* also works as the client that connects to *VoiceServer*. As show in Figure 6, *SpeechBox* has 'text' slot in which the text string the user want to *VoiceServer* to speech is stored.

APPLICATION EXAMPLES

In this section, we show two application examples of component based voice input interfaces. Those are a tank battle game and a toy for interactions with a CG dog.

Tank Battle Game

We applied voice input interface components to the tank battle game already developed using *IntelligentBox*. In this game, players move a tank on the ground using a mouse device as shown in Figure 7 and attack other player's tank by shooting a bullet. The bullet is shot by the mouse click operation.

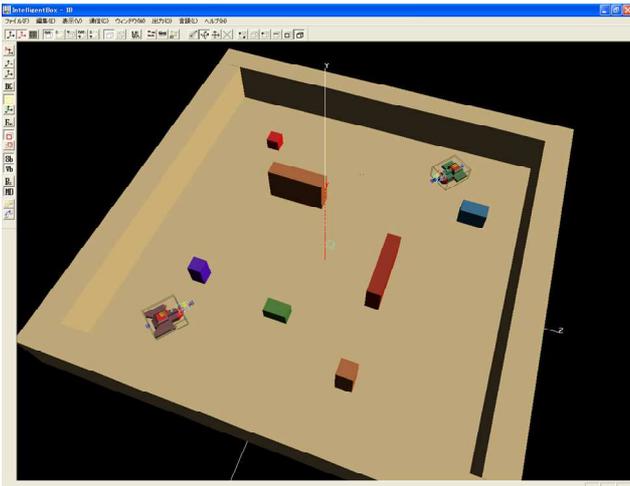


Figure 7: Bird-eye view of a tank game

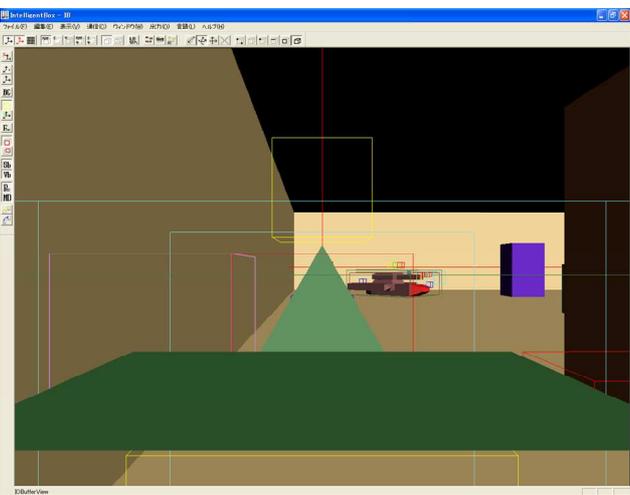


Figure 8: View image seen from the player of a green tank

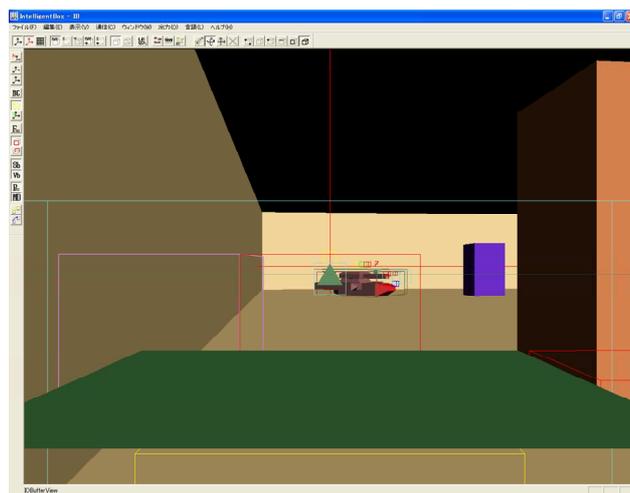


Figure 9: View image when shooting a bullet

Figure 8 shows the view seen from the player of a green tank. There are three wire-frame boxes those are controllers for shooting a bullet, moving a tank and changing user's viewpoint. In fact, it was difficult to play this tank game smoothly because *IntelligentBox* does not allow the user to operate multiple boxes simultaneously. This means that the user needs additional mouse-click operations whenever changing the

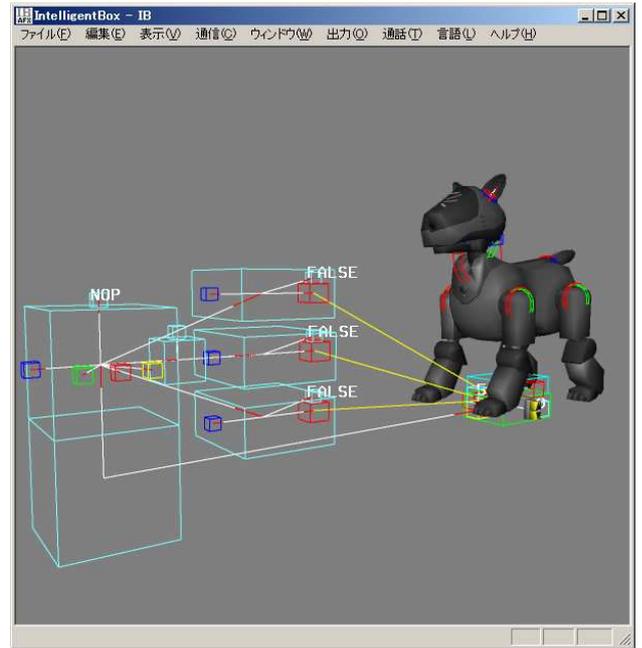


Figure 10. Components for interaction with *AIBO*.

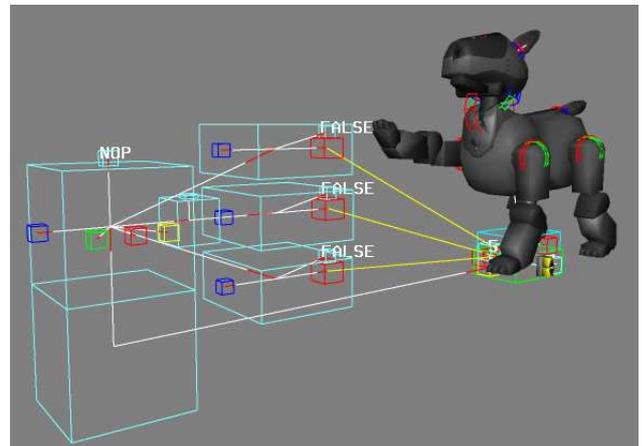


Figure 11. *AIBO* with raising his right front leg.

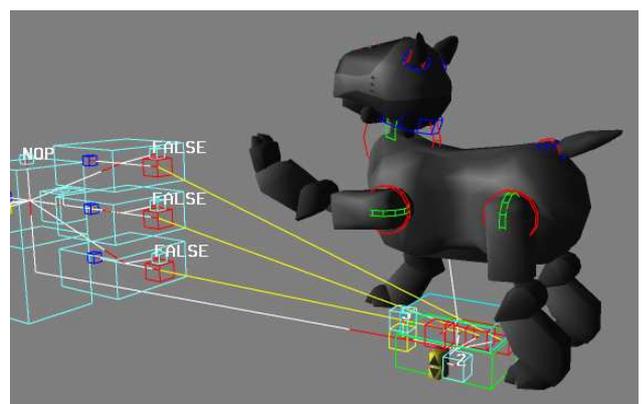


Figure 12. *AIBO* with raising his left front leg.

operation on the controller of shooting a bullet into the controller of moving a tank and its opposite. We attached voice command input interface to the box which is the controller for shooting a bullet and the box which is the controller for changing user's viewpoint. Thus, it is possible to shoot a bullet and to change user's viewpoint with voice commands so that a player can concentrate on moving a tank

using a mouse device. Figure 9 shows the view image seen from the player of a green tank when he/she has shot a bullet by a voice command. In this way, it has become smooth to play the game.

Interaction with *AIBO*

As another application example of component based voice input interface, we made a toy for interactions with a CG dog. In fact, this CG dog is the model of *AIBO*, which is an entertainment robot produced by *Sony Corp.* (*AIBO*).

As shown in Figure 10, in this example, we used three *VoiceCommandBoxes* to accept three different voice commands. Those data flow is the same as that shown in Figure 5. If the user says “right leg” in Japanese, *AIBO* raises his right front leg as shown in Figure 11. If the user says “left leg” in Japanese, *AIBO* raises his left front leg as shown in Figure 12. Moreover if the user says “return” in Japanese, *AIBO* moves his legs into their original positions. With using *IntelligentBox*, we could make this example in less than two hours. We are supposed to add voice output interface to make this example more entertaining.

CONCLUDING REMARKS

This paper proposed component based voice input/output interfaces. We have been studying component based 3D graphics software development systems. We have already proposed a prototype system called *IntelligentBox*. Application fields of *IntelligentBox* include various topics, one of them is the development of 3D games, and *IntelligentBox* would be also useful for the development of wearable computer and augmented reality applications. For these kinds of applications, voice input/output interfaces are more useful. Especially for physically handicapped persons, voice input/output interfaces are strongly required. Moreover for the development of multimodal interactive games, voice input/output interfaces are necessary. Then we introduced voice input/output interfaces as software components into *IntelligentBox*. This paper explained the functionalities and the usage of those software components. We also shown practical 3D graphics application examples of entertainment fields, a tank battle game and a toy for interactions with a CG dog, to clarify the usefulness of component based voice input/output interfaces. These application examples, as speech controlled 3D games, may not be interesting for the readers. We will also develop more interesting speech controlled 3D games.

In this paper, we did not show any examples of augmented reality and wearable computer applications. We will make such examples to clarify the usefulness of *IntelligentBox* for those applications fields. There is another voice recognition tool called *Julius* (Lee, et al. 2001). Since *Julius* is open source software and is easy to be customized, we have been trying to use *Julius* in order to change native operations of *IntelligentBox* performed by a keyboard and a mouse device into those performed by voice commands. These are our future works.

REFERENCES

- Igarash, T. and Hughes, J. F., 2001. “Voice as Sound: Using Non-verbal Voice Input for Interactive Control”, 14th Annual Symposium on User Interface Software and Technology, ACM UIST'01, Orlando, FL, pp.155-156.
- Lee, A., Kawahara, T. and Shikano, K., 2001. “Julius --- an open source real-time large vocabulary recognition engine”, In Proc. European Conference on Speech Communication and Technology (EUROSPEECH), pp. 1691-1694.
- Manaris, B. and Harkreader, A., 1998. “SUITEKeys: A Speech Understanding Interface for the Motor-Control Challenged”, Proc. of The Third International ACM Conference on Assistive Technologies (ASSETS '98), Marina del Ray, pp. 108-115.
- Manaris, B., McCauley, R. and MacGyvers, V., 2001. “An Intelligent Interface for Keyboard and Mouse Control, Providing Full Access to PC Functionality via Speech, Proceedings of 14th International Florida AI Research Symposium (FLAIRS-01), pp. 182-188.
- Okada, Y. and Tanaka, Y., 1995. “IntelligentBox: A Constructive Visual Software Development System for Interactive 3D Graphic Applications”, Proc. of Computer Animation '95, IEEE Computer Society Press, pp.114-125.
- Okada, Y. and Tanaka, Y., 1998. “Collaborative Environments of IntelligentBox for Distributed 3D Graphics Applications. The Visual Computer, Vol. 14, No. 4, pp.140-152.
- Okada, Y., Itoh, E. and Hirokawa, S. 2000, “IntelligentBox: Its Aspects as a Rapid Construction System for Interactive 3D Games, Proc. of First International Conference on Intelligent Games and Simulation (GAME-ON2000), SCS Publication, pp. 22-26.
- AIBO*, <http://www.sony.net/Products/aibo/index.html>
- Microsoft SAPI,
<http://www.microsoft.com/speech/download/sdk51/>

ENCODING SOUND BY POLYNOMIAL INTERPOLATION FOR INTELLIGENT DYNAMIC MUSIC IN COMPUTER GAMES

M. A. Burley, N. E. Gough, Q. H. Mehdi¹ and S. Natkin²

¹ Games, Simulation and AI Centre
Research Institute for Advanced Technologies
University of Wolverhampton, Wolverhampton, UK
E-mail: M.Burley@wlv.ac.uk

² Centre De Recherche en Informatique du CNAM
Conservatoire National des Arts et Métiers, Paris, France
E-mail: natkin@cnam.fr

KEYWORDS

AI, Polynomial interpolation, sound, music, computer games

ABSTRACT

Current research in computer music composition almost exclusively involves the manipulation of music stored as MIDI data. While this allows direct access to the structure of music, it creates limitations in realism for the end result of such techniques. This paper describes a method designed to represent music in a form that facilitates the use of existing processing techniques while conserving the ‘real-world’ attributes of music recorded in PCM format giving computer-game developers a facility for the production of variations on a pre-recorded theme, whatever the original source. Experimental results are presented to demonstrate that polynomial interpolation is a viable technique.

INTRODUCTION

This paper explores the use of polynomial interpolation to improve the generation of audio tracks for computer games. Traditionally, there is a recurring tendency for computer music research to tackle the processing of music at a grammatical level. Music is often described as a language and, indeed, can be quite legitimately thought of as so. There is however evidence to suggest that working at a higher-level than that of the note-sequence has considerable potential for analysis and composition. As far back as 1979, it was becoming apparent that simply applying techniques similar to those used in Natural Language Processing (NLP) was falling short of the mark in unlocking the secret of what makes music sound musical (Meehan 1979). The concept of ‘Shenkerism’, whereby initial parsing of a piece of music is performed at the note-group level rather than delving into every facet of its structure, was a hint that being ‘less-precise’ could in fact make the task of instilling creativity in computer music easier. This was also the case with the later POD system of Truax (1977) that introduced the concept of ‘Digital Sound Objects’. A survey by Roads (1985) is quick to criticise many of the automatic-composition systems

developed around the middle of the twentieth century for their rigidity – something that could reasonably be seen as a necessary compromise to achieve the required degree of success when using an abstracted representation of music.

Nevertheless, computer music research seems to be anchored to the concept of musical notes whether the technique in use is a Neural Network, Genetic Algorithm, Stochastic or Grammatical algorithm or an Iterative Formula (with some exceptions in the latter case). A cursory glance through core texts in computer music such as Roads (1995) and Miranda (2001) will make this apparent. One possible reason for this is the fact that, when working at a higher syntactic level, one faces the choice of either being limited by having to work with note-groups or relative pitch structures as atomic components or, if these high-level symbols are made more flexible, losing some of the very information one is trying to process.

The computer-games industry largely ignores existing automatic composition techniques. The game ‘Halo’ (O’Donnell, 2002) which is recognised as having one of the most advanced dynamic-music systems currently in existence only stretches to event-driven transitions between manually-composed segments of music whilst other landmark games such as Quake seem to treat background music as a technical afterthought.

A major driving factor behind the technique presented here, was the desire to preserve as much data as possible when improvising around an existing composition. Of paramount importance in this respect is the issue of timbre. Usually defined as the characteristic of a sound that allows us to identify it as emanating from a particular source (a musical instrument for instance), timbre is an issue for any composition stored in MIDI format as the composer is restricted to whatever sounds the synthesis module of the sound-card can generate. This issue becomes augmented when working with existing compositions recorded from real instruments in PCM form. A conversion to MIDI format allowing computer improvisations destroys all of the original timbre information resulting in (despite the use of advanced

synthesis techniques) an artificial sounding end-product that fails to preserve any of the nuances and idiosyncrasies of the original performers. The aim of the work described here was to allow any recording to be used as the basis for background music in a game so as to create the same emotional effects that a film-soundtrack, which is tailored to a predefined script, creates for its audience. Consequently, MIDI was discarded as a viable option.

The disadvantage of working with wave-data as opposed to MIDI is that while it might be possible to decipher and work on the regular, amplitude samples provided in wave-data files (using say a neural network), this involves a complex procedure just to obtain a single note that can be identified from the mass of fundamentals, partials and general background noise. This limitation was the initial hurdle in the process of developing a professional dynamic music system for use in computer-games. While the idea of a musical-improvisation system composing the soundtrack for a game in real-time and in response to environmental and narrative factors present in the gaming environments is not unresearched (Casella & Paiva 2001), there seems to be an automatic choice of MIDI as the format to work with, presumably for the reasons already mentioned. It is felt that a compromise is possible if some of the complexity of such a representation system were to be handled by Artificial Intelligence (AI).

The aims of this research are thus to produce variations on an existing sound track by means of AI; to limit the representation of that theme by defining only at a conceptual level; to segment the track and represent each segment parametrically; and to use the parameters to generate new instantiations of the sound.

The paper is organised as follows: In the next section we examine the possible use of AI techniques to solve this problem and outline the use of polynomial interpolation as a basic data representation for this process. This is followed by a description of the experimental methodology and the results obtained. The paper concludes by examining the limitations and possible improvements for the proposed technique.

METHODOLOGY

AI as a Facilitator

Artificial Intelligence (AI) provides a way of tackling problems without having to think about the fine detail. As an eventual aim of the work is to produce variations on a theme by means of AI, the representation of that theme needs only to be defined at a conceptual level. What is required is a level of quantisation whereby each code represents not just pitch information but rhythmic and timbral information as well. Our approach is based around the theory of wavelets and is designed to allow segments of a soundtrack to be categorised as instantiations of dynamically identified generic waveforms. While these 'waveform-objects' are extremely difficult to work with manually, it is believed that a stochastic technique such as a Markov Model (Russell & Norvig 1995) or an AI technique such as the Kohonen Self Organising Map (Kohonen 1989) will be able to identify the relationships

between them in the context of specific musical tracks. These relationships can then be manipulated in order to induce variations on the original theme, theoretically producing music that sounds as though performed by the original artists.

Segmentation and Polynomial Interpolation

In order to identify segments of PCM data as specialisations of generic waveforms, it is necessary to find a representation of those segments that allows rigorous comparison. The approach taken here is a functional one. Lagrange Interpolation is applied to successive segments of the waveform representation of a track resulting in a sequence of polynomial equations, each of which representing a particular segment. As the Lagrange formula allows determination of the degree of a polynomial before performing calculations, the only parts of an equation that need to be stored are the coefficients of the various terms.

The Lagrange formula adopted is as follows (Butler & Kerr 1962):

$$l_j(x) = \frac{(x-x_0)(x-x_1) \dots (x-x_{j-1})(x-x_{j+1}) \dots (x-x_n)}{(x_j-x_0)(x_j-x_1) \dots (x_j-x_{j-1})(x_j-x_{j+1}) \dots (x_j-x_n)} \quad (1)$$

$j \in [0, n]$, $j \neq n$, where $x_0 \dots x_n$ represent a series of values for the independent variable (in this case, instants in time) and $l_j(x)$ is the polynomial for the wave segment at instant x . A complete approximating polynomial for a given segment is obtained by summing the products of the various $l_j(x_j)$ and their corresponding $f(x_j)$ (the amplitude at instant x_j):

$$L(x) = l_0(x)f_0 + l_1(x)f_1 + \dots + l_j(x)f_j + \dots + l_n(x)f_n \quad (2)$$

It was discovered pragmatically that, in the context of this paper, Lagrange polynomials generated from large sets of PCM data are generally unreliable in terms of accuracy. Another important issue affecting the choice of parameters was that of sampling frequency. The danger is that by taking PCM values in close proximity to each other, little variation is picked up, with the result that each 'wavelet' reduces effectively to a simple curve or, in extreme cases, a line. In practice this would lead to identical classifications of most segments giving no significant outcome.

Currently the sampling rate (11,025 Hz), number of interpolation points (6 per segment) and distance between interpolation points (5 samples) are fixed at values chosen through informal experimentation. While this configuration is sufficient to demonstrate the potential of the technique, it is unlikely that this approach will be sufficient to take the work forward. As there is no relation between the aforementioned parameters and the structure of the music being processed, it is purely a matter of chance as to whether or not the more (structurally) significant parts of a wave are picked up or missed by the interpolating-quantiser. The next step is therefore to add a degree of 'intelligence' to the algorithm, taking into account the structure of the music on which it is working in both the time and frequency domains.

EXPERIMENTS

Various recordings of a musical soundtrack were made in PCM format using the low-level wave functions provided by the Win32 API in a modified version of the simple buffered recording program provided by Petzold (1998) in order to reduce development time. This approach provided memory-buffers of wave data that were then processed by the interpolating-quantiser. The Lagrange coefficients were written to a file. The quality of this representation was then validated by reconstructing the waveform and comparing the result with the original. While the technique is not designed to be an alternative storage-format for music due to an inevitable loss in sound-quality caused by the geometric properties of the Lagrange polynomials, this exercise was necessary in order to verify that the generated wavelets bore sufficient relation to the original wave-segments. Once it had been determined that the LIP data, when played back, was recognisable as the original PCM recording, graphical representations of some of the wavelets were made with their associated wave-segments. These graphs clearly illustrate the potential for success of this approach to sound representation while simultaneously highlighting areas for improvement. A low sampling rate was deliberately chosen in order to determine the maximum 'strain' that the system could deal with.

RESULTS

The following graphs illustrate the effect of interpolative-quantisation using the Lagrange-based technique described above on a piano rendition of the C Major scale sampled at 11,025 Hz (CD audio is generally recorded at 44.1KHz). The start and end samples are given in the titles. Also, note the differing ranges of the Y axes.

The reproduction in Fig. 1 is very close in form to the original, however the Lagrange technique is at the mercy of two factors. We will discuss the most innate of these shortly, but an evident side effect of taking points at fixed intervals is the fact that, by missing a peak or trough, the resulting polynomial will flatten out that part of the waveform as the subsequent group of samples shown in Fig. 2 demonstrates.

One should also be aware that making even a slight change to a waveform can introduce many new partials (component waves that, when added together, form the complex wave seen) and that because of the way the brain interprets sound waves (Zotkin et al, 2003) this can have unwanted side effects such as single notes being turned into chords and pseudo-random timbres replacing the sounds of the original instruments.

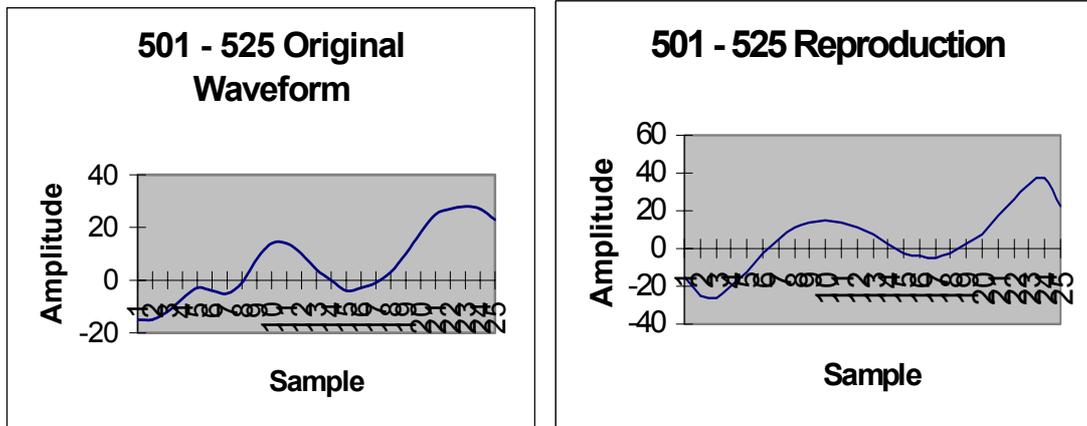


Fig. 1 Acceptable interpolated reproduction of wave-segment

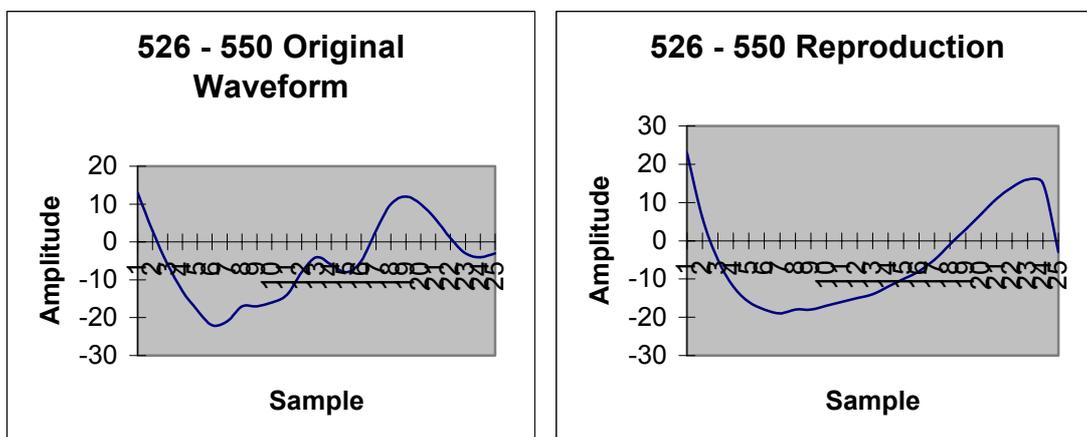


Fig. 2 Inaccurate reproduction

LIMITATIONS AND POSSIBLE SOLUTIONS

The primary risk in using Lagrange Interpolation is that the formula does not offer a way to determine how well a polynomial fits the original waveform (Hosking *et al*, 1986). One technique which is less than satisfactory and high on the priority list for replacement is that of overlapping the regenerated wavelets to compensate for a tendency whereby (at least with the current, empirically determined operating parameters of the system) the Lagrange Polynomials usually become very inaccurate after the last reference point (I.e. the last point given to the Lagrange formula from the original PCM data). This may be resolved by the measures described below, but if not will warrant the replacement of the Lagrange formula by another polynomial technique.

As has already been stated, the final aim of this work is not to develop an alternative sound storage technique. It is to create, in parallel to the PCM data, a meaningful digest of a piece of music that can be used to alter it without losing timbral information. We must therefore take into account the fact that some parts of a piece of music are more significant than others. Immediately, fixing the interval between interpolation points is highlighted as a problem. While Lagrangian Interpolation can implicitly deal with varying distances between these points, the problem again arises that we have no way of determining the accuracy of a polynomial and, with so many polynomials resulting from even a one second sample of PCM data at 11,025 Hz, no way of flagging 'bad' polynomials for treatment by a corrective algorithm.

On the other hand, other interpolation techniques tend to require data tabulated at equal intervals (Hosking *et al*, 1986). This may turn out to be unacceptable for reasons already mentioned. Also, the order at which these constant intervals are found often determines the order of the resultant polynomial. This makes the storage structure required more complex but may provide a payoff in terms of accuracy.

With some options clearly available, we then face the task of identifying significant events in a piece of music. While the efficacy of techniques with which to accomplish this is yet to be investigated, it is felt that a protocol-analytic study of composers and audience members will be of value.

Assuming for a moment the worst case scenario; it may become apparent that interpolating polynomials will be sufficient for the manipulation of music by AI techniques but not sufficient to completely replace PCM as a recording **and** playback format. It will therefore be necessary to map changes made to the approximating polynomials to the raw PCM data that will actually form the output of the dynamic-music system. It is here that a more significant overlap with Computer Sound Synthesis occurs. Slaney, *et al* (1996) have conducted research into

the problem of "Automatic Audio Morphing", a means of smoothly transitioning from one sound to another. They achieve this by isolating each different aspect of a sound-wave to its own dimension. These dimensions are then warped according to freely-definable rules governing the relationships between them. This approach may well provide a satisfactory Polynomial-PCM bridge in our dynamic-music system.

CONCLUSION

While currently in its infancy, we have demonstrated a way of representing sound that has the capacity to facilitate the manipulation of any music stored in PCM format while preserving all of the original data not concerned with the piece's 'grammatical' structure. This is almost an opposite approach to that taken by systems using the MIDI standard or similar. Any pitfalls currently inherent in the technique indicate their own solutions and have enabled us to construct a solid methodology with which to take the work forward.

References / Bibliography

- Butler, R. and Kerr, E. (1962) *An Introduction to Numerical Methods*. 1st ed., London: Sir Isaac Pitman & Sons Ltd.
- Casella, P. and Paiva, A. (2001) MAgentA: an architecture for real time automatic composition of background music. *Intelligent Virtual Agents'2001*, Springer.
- Hosking, R. J., Joyce, D. C. and Turner, J. C. (1986) *First steps in numerical analysis*. 1st ed., Kent: Hodder and Stoughton Educational.
- Kohonen, T. (1989) *Self-Organisation and Associative Memory*, 3rd Ed. Springer-Verlag.
- Meehan, J. (1979) An Artificial Intelligence Approach to Tonal Music Theory. in ACM: Association for Computing Machinery *Proc. 1979 Annual Conference, 1979*. New York: ACM Press, pp.116-120.
- O'Donnell, M. (2002) *Producing audio for Halo* [online]. [accessed 7th September 2004]. Available from: <http://www.gdconf.com/archives/2002/marty_odonnell.doc>
- Petzold, C. (1998) *Programming windows*. 5th ed., Microsoft Press International.
- Russell S. and Norvig P. (1995), *Artificial Intelligence a Modern Approach*, Prentice Hall Inc.
- Slaney, M., Covell, M. and Lassiter, B. (1996) Automatic Audio Morphing. In *Proc. IEEE int. Conf. Acoust., Speech and Signal Processing, 1996, Atlanta*. pp.1-4.
- Truax (1977) The POD System of Interactive Composition Programs. in *Computer Music Journal*, 1(3), 1977, pp. 30-39
- Zotkin, D., Shamma, S., Ru, P., Duraiswami, R. and Davis, L. (2003) Pitch and timbre manipulations using cortical representation of sound. in *Proc. ICASSP, 2003, April, 2003, Hong Kong*. vol.5, pp.517-520.

DYNAMIC PLAYER MODELLING: A FRAMEWORK FOR PLAYER-CENTRIC DIGITAL GAMES

Darryl Charles¹ and Michaela Black²

School of Computing and Information Engineering,

University of Ulster, Northern Ireland.

¹dk.charles@ulster.ac.uk

²mm.black@ulster.ac.uk

KEYWORDS

Adaptive Games, Dynamic Player Modelling, Neural Networks.

ABSTRACT

In this paper we outline a framework for creating player-centred digital games. At the core of our proposal is the requirement for games to be more responsive to different player types and their individual needs; for games to have the capability to adapt so as to provide an appropriate level of challenge for each player, to smooth the learning curve, and enhance the gameplay experience for each player individually. This is not an easy objective to achieve and adaptive game technology, although a popular ideal in some quarters, is still fraught with difficulties and some controversy. We address some of the most well known issues and outline a proposal for dealing with two of the more recent issues: that of monitoring the effectiveness of game adaptation on the basis of player intention and/or frustration, and dealing with dynamic player profiles – because players learn in different ways and at a different speed.

INTRODUCTION

All game players are different; each has a different preference for the pace and style of gameplay within a game, and the range of game playing capabilities between players can vary widely. Even players with a similar level of game playing ability will often find separate aspects of a game to be more difficult to them individually and the techniques that each player focuses on to complete separate challenges can also be very different. For these reasons and others it can be very difficult to design a game that caters for a wide range of player capability and preference. Game developers have traditionally dealt with the range of player abilities in a very straightforward manner, for example, by allowing the player to select a difficulty level at the beginning of the game, as with the classic first person shooter “Doom”. Once a player selects their level of difficulty for a game designed in this way, then there is usually no attempt within the game to monitor how a player is performing in order to adjust the level of challenge or gameplay experience. Recent games are better at allowing a player to set up preferences for their gameplay experience, e.g. as with “DeusEx” where a player can tailor their own avatar’s characteristics, but this relates more to setting up the gameplay experience before starting the game than intelligently recognizing and adapting to the

needs of the player in-game. While the concept of an adaptive game is a controversial topic among some gamers and developers, there are clear benefits to tailoring the game experience to particular player types – especially for educational games (Beal et al 2002). Catering for the individual more effectively could help attract a wider participation, if for no other reason that it will be easier for players to get started, progress and complete a game. In a recent edition of the Edge (Edge magazine 2004) Poole provides an insightful discussion on the problem of “beginnings”, teaching the player, and lack of game completions by most players, while in the same issue Redeye highlights the niche quality to current games and their lack of accessibility to a wider group of people.

Adaptivity within games may primarily be implemented by auto dynamic difficulty technologies (Miller, 2004) but there are a number of other ways in which adaptivity can be advantageous. For example, in helping players avoid getting stuck, adapting the gameplay more to the player’s preference/taste, or perhaps detecting deviant player behaviour and modifying the game in response. What we mean by deviant player behaviour is, for example, when a player uses or abuses an oversight in the game design to their advantage. Often this means that the player finds it easier to succeed in the game but their enjoyment of the game is lessened because the challenge that they face is reduced and they are not encouraged to explore the full features of the game – i.e. players will often repeat a successful strategy over and over again because it leads to a predictable win, even if it is boring and somewhat ruins the game. This happens frequently in real-time strategy games such as “Warcraft” or “Command and Conquer”. Bungie, the creators of “Halo 2” – a game much praised for its AI – acknowledged the importance of this when they designed the AI deliberately to prevent the player using “boring” tactics but positively reinforced the player when they used imaginative or adventurous tactics (Griesemer & Butcher, 2002).

In this paper we propose and discuss a novel framework incorporating advanced ideas about player-centred game design. This comprises of four key aspects: player modelling, adaptive game environments in response to player needs, monitoring the effectiveness or appropriateness of any adaptation, and dynamic player remodelling or classification.

PLAYER-CENTRED GAME DESIGN

Most game design is, of course, already centred on the player but it tends to focus on large groups of players rather than catering for individual players – in this paper we hope to persuade the reader that games that are adaptive in catering for the individual will be one of the key innovations in future games. One of the novel aspects of the framework that we propose in this paper for player-centric games is the ability of a game to dynamically model, remodel, or reclassify a player as they play the game. Players differ not only in their characteristics and ability as they begin to play the game, but every player will learn at a different rate and each player will excel in (or just simply enjoy) different aspects of the game.

The most common game model for differentiating between player – and even this is quite rare – is shown in Fig. 1. A player may set their difficulty preference (and perhaps make a few other choices relating to their ability or preference) before beginning the game. Within the gameplay itself there may be a simple hinder/help mechanism, as in the racing game “Mario Kart” where a player who is doing well will not receive good power-ups or weapon bonuses while a player who is struggling will gain a lot of help through a discreet speed up or by receiving more powerful item drops. Most of the simpler methods used – and often most effective – are straightforward help mechanisms, for example in the “Crash Bandicoot” series if a player repeatedly fails at the same point in the game then a mask is provided to the player character which acts as a shield. This essentially allows the player to make one mistake and still be able to progress, e.g. the character may hit a land mine once without losing a life.

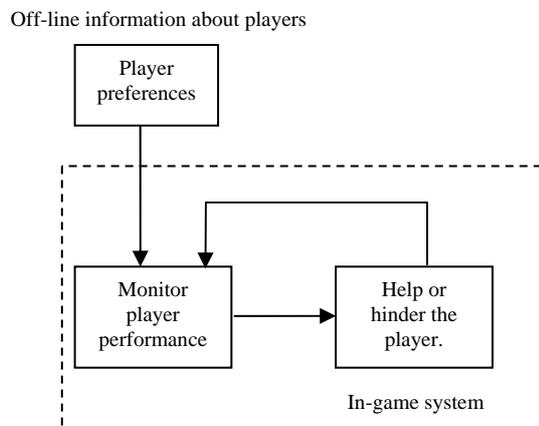


Fig. 1 A typical current game system that changes in response to the player.

Often such systems are “life” based as with “Maximo” where a player is provided with a coin by the angel of death character “Grim” in order to buy another go when they die – when a character fails at a challenge they may go back to a save point within the game level if, and only if, they have a life/death coin. In this way a weaker player still has an opportunity to progress while a stronger player is encouraged to play sensibly – because they have a limited number coins. However, as well designed as this mechanism is, the game can still be prohibitively difficult at times for the novice player. When a player runs out of coins then he/she has to reload a save and restart the level, and this inevitably is one of the reasons that many people will never finish this game. “Prince of Persia: The Sands of Time” provides another mechanism which operates in a similar manner by allowing

the player to press a button that “rewinds” a sequence back in time by up to approximately 10 seconds. This is particularly useful for dealing with mistakes, accidents, or misjudgements by a player, for example, let’s say a player makes his/her character jump across a gap and the jump is miss-judged so that the character falls to his death. Usually in a game – particularly with game consoles – this would mean that the level would have to be restarted or the player would have to go back to a previous save point, but with “Prince of Persia” a player simply rewinds that mistake and tries it again (up to a limited number of times obviously). This mechanism proves to be excellent in reducing frustration simply by adding a quality game design feature.

A different type of help mechanism used in 3rd person view games is to have a game character look at areas that are interesting as with “Eternal Darkness” where the player character will turn his or her head to look at pictures etc. that perhaps should be examined. “Ico” is even more impressive in this regard in that the non player character “Yorda” who accompanies the main player character will often wander around independently, looking and pointing at things that the player should examine after he/she has been stuck in an area for a while.

These approaches supplement clear, directional level design but are not particularly dynamic. So the basis that we propose for a player-centred framework should build on recent research within the AI community with methods such as intelligent interfaces (Rogers & Iba, 2002, and Livingstone & Charles, 2004). Mainstream AI research is relatively unused within the game development world yet progress in this area for games has the potential to

revolutionise gameplay (Charles, 2003) as much as 3D game technology has in the past. AI can provide a perceptual and functional interface between the player and game (Charles & Livingstone, 2004) to enhance the experience for an individual player.

PLAYER MODELLING AND ADAPTIVE GAMES

A few game developers and researchers are now considering player modelling (Houlette, 2004) and adaptive games (Charles, 2003, and Charles & Livingstone, 2004), though work in this area is still relatively rare. Fig 2 illustrates our view of how a basic adaptive game system could be set up. Two sources of information can be used to identify the

player-type for a game: firstly the information that a player provides when they begin the game by setting basic preferences and inputting information about themselves. The second source of information should be taken from the player's gameplay habits and performance in-game. Together this information can be used to match the player to pre-defined models and the game can then be adapted to cater specifically to their needs and abilities.

Players may not necessary need to be modelled by one single object but several object models may be used to model them that cover different aspects of the gameplay and their relationship with the player. A more refined object model is obviously better because player modelling can be complex, for example one player may be excellent at combat but terrible at problem solving, while for another the opposite may be true. In this case it is clearly better to model both aspects separately, rather than try to fit them into a coarser single model.

as a starting point for the dynamic modelling process in-game or to help label player groups. For example, we know that there are certain differences, in general, between some of these groups in terms of reaction time and in game play deliberation. Of course, caution must be taken when adopting this approach, because this initial classification process will be quite coarse, e.g. girls may generally like games like "The Sims" and "Everquest" due to the pace of the game and other factors but many prefer action/adventure or sports games. Identification of which type of information produces the most informative profiles is a very important initial task. Key fields of data can be identified as attributes of information, for example gender attribute with values: male, female, and once the necessary attributes have been identified and the information collected, some pre-analysis can be done. If predetermined profiles are not obvious we can use unsupervised machine learning techniques such as clustering to partition groups of players. We demonstrate how both may be achieved with neural networks in the next

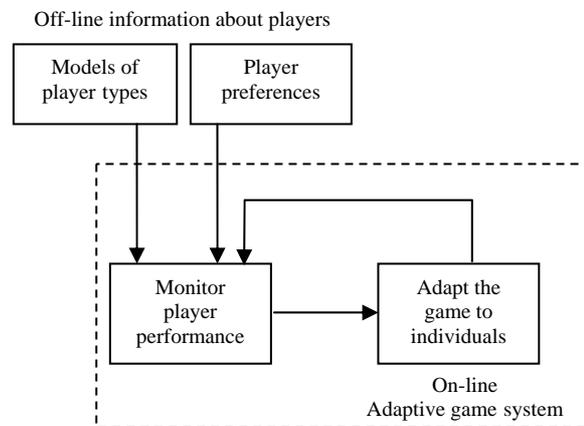


Fig 2. A basic adaptive game system.

Player Modelling

It could be said that there are two main reasons for player modelling in digital games. Firstly, modelling a player in order to instil human-like qualities into a non-player character, as was demonstrated by an example in a recent paper (McGlinchey, 2003) where it was shown that the characteristics of an individual player could be captured while playing a game of "Pong" by a Self Organising Map (SOM) neural network. The SOM could then be used as the "AI" for an artificial computer opponent in subsequent games. The second reason for player modelling and the approach that we are interested in within this paper, is modelling players – or perhaps classifying typical player types or behaviour – so that we may recognise predefined player types or behaviour within the game. The reason that we want to recognise the type of player currently playing is because we wish the game to adapt the needs of the player.

To enable the creation of initial user profiles, some monitoring of game players is required to attain information. Additionally, information about the player – provided by the player themselves – such as whether they are a novice or advanced, male or female, young or old, and other basic general factors may be used as part of the player modelling or clustering process. This information can be used as part of the initial modelling or classification process and it can serve

chapter.

Once the most appropriate attributes have been identified we then may produce our separate player profiles where each cluster group represents a different profile of player. If we wish to be able to interpret the properties of these individual groups, they can be labelled and the individual examples of each group provided to a supervised machine learning technique such as a tree induction classifier (Quinlan, 1986). This common inference task consists of making discrete predictions about a concept, in this case each profile, and this prediction problem is referred to as the classification problem. The task of a classification algorithm is to accept a set of training examples which will depict the current state of knowledge for that concept/profile. These training examples are a set of descriptive attributes with an associated class, and this class represents a value for the concept. The algorithm will induce a knowledge structure to distinguish between the values of the concept. A tree induction algorithm will produce a classifier in the form of a tree from which rules can be interpreted as one for each path from the root of the tree to each leaf. These rules depict knowledge which represents the concept. As will be demonstrated in the next chapter we can also use other supervised learning algorithms such as neural networks.

Adaptive games

Adaptation can have two related meanings: one meaning that relates simply to change, and another related to learning and transformation. In the first case the adaptation from one form to another has been predetermined and the adaptive states are known in advance, and in the second case the adaptation occurs after some learning from experience and the transformed state may be previously unknown. Both forms of adaptation are relevant for games, but adaptation from learning is the most interesting and also the more controversial. The reason for this controversy is that mechanisms within games that have online learning are unpredictable and therefore are very difficult to test thoroughly. Scepticism (or even anger!) is also often expressed by gamers and developers with regard to games that change according to player performance. “Mario Kart” provides one of the most well-known examples; in this game if a player is winning then he/she does not get any of the powerful power-ups and the computer controlled cars often speed up – and the opposite is the case if a player is losing. While this can be annoying if a player is dominating a race, it does even out player ability disparity in a multiplayer competition and thus the race may be more evenly matched and thus exciting. However, if players are aware of “cheating” AI they may alter their gameplay accordingly; i.e. a player may decide to remain in second or third place until near the end of the race so that they may receive a significant power-up or weapon to unleash on the leader on the last corner of the race – thus a new (perhaps unpredicted) gameplay mechanic is introduced. There is some evidence that adaptive game technology is more effective when the player is unaware that it is happening, for example, the primary author of this paper played and completed “Max Payne” without realizing that it incorporated “auto-dynamic difficulty” technology (Miller, 2004).

There are two opposing desires in players that we need to take into account: the desire of a player to learn the rules so as to master the game, and the requirement to avoid “sameness” or lack of variety of gameplay. Thus, while we believe that there is a clear need for player-centric adaptive technology within games to cater for individual players needs, to help them learn and play the game, to enhance their playing experience, to recognise when the player is stuck or frustrated and help out. There is also a requirement that the rules of the game do not change significantly, which would frustrate many players, and ideally either the player should not be aware of the adaptive nature of the game or they should have the option to switch it off.

NEURAL NETWORKS FOR THE MODELLING PROCESS

The use of neural networks for the player modelling process is quite an obvious approach but the authors are not aware of them having been used much for this purpose in games yet and so we provide an overview to a few possible supervised and unsupervised approaches below. Neural networks are good at detecting patterns and clustering data (depending on the method) and so we can use a variety of neural network techniques in different ways to identify or understand different players. Additionally, as neural networks are essentially learning machines they hold a number of

possibilities with regard to our ideas about adapting to individual players and the dynamic re-modelling of players.

Supervised Approaches

In-game data is very valuable in the process of tailoring a game to the individual player and building accurate player models. For example, we can use reaction times, choices made, styles of play, accuracy of shots/hits, how often a stage needs to be repeated before completing, average health, number of deaths per level, kills per level per possible kills as with “Max Payne” Auto-dynamic Difficulty technology (Miller, 2004). This data may be used directly to decide how to change the parameters of the game environment, attributes of the player character, or non-player character behaviour dynamically through the training of a neural network such as the Backpropagation network. With this approach player entered game data may also be used alongside the in-game player data to moderate the response of the network. This aspect could be important because it may provide a clue to how rapidly or how much the game should be adapted to the player. For example, if an advanced game player is currently playing then they may be less frustrated by not completing a challenge after a few attempts than a novice and therefore the game adaptation may be by a smaller amount or not at all. There are problems with using user-entered profile data (or perhaps any type of profiling), for example, profiling may become frustrating or even redundant if more than one player plays the game at the same time (taking turns) and thus sharing the same profile, in this case it would be impossible for the profiling and adaptation to be accurate. Also, every type of game would require a different approach and the technology may not be appropriate for many types multi-player games because players would be playing against each other on an uneven playing field. For example, in “Soul Calibur II” it is possible for a weaker player to increase their “life bar” relative to their opponent but it is actually unusual for this to occur in practice because players like to feel that they are competing on a level playing field.

We can also take another neural network approach to player modelling by using a clustering algorithm. In this way we use the neural networks to cluster player types according to out-of-game and in-game data, grouping player with a similar profile into the same group type. There is a wide range of ways in which this may be done, for example we could use a radial basis network with fixed cluster centres to classify the players, with the centres fixed on different areas of the data space that we believe to provide a good “centre” for our player classification. By monitoring and adapting the player profile throughout the game then the player may achieve a new classification, and thus the game would respond differently. Radial basis networks may also have moving “centres” and so the centres can be moved automatically during training to fit the data more appropriately. It is also possible to retrain the full network during gameplay on the basis of new data, although this is not necessarily an easy thing to do. For example, a single player, depending on the method, may only provide one new data point and so re-training may be futile. This is generally an issue with online learning in games; it is not only slow but often there is not enough new data to significantly impact the training of the network, and needs to be taken into account

when choosing which method to use and how to implement it.

Unsupervised Approaches

There is very little digital game research going on that involves unsupervised learning, perhaps because of a lack of expertise in this area. However, we would like to demonstrate here that there are a few very positive and promising uses for unsupervised neural networks for forming a statistical understanding of player data. Unsupervised neural networks are generally used to explore or investigate structure or patterns in data on the basis of statistics or information theory (or similar). It is not known, a priori (though we may have an idea), what the relationship is between the data variables and we would like to investigate this. This is similar to data mining the player data and we can use this approach to help us understand the difference between player styles or capabilities then use this knowledge in our player modelling process. Once the neural network has been trained to our satisfaction then it may be used directly in-game to identify player types or behaviours. Many of these algorithms are also quick to train and so may be more suitable than other approaches for on-line re-training. The techniques that we focus on in the examples below are known as projection methods. With projection methods we typically want to explore the relationship between the input variables but with clustering approaches we treat each data example as a data point (e.g. a player description) and attempt to group data points together based on some similarity measure.

Let us say that we wish to explore the relationship between the variables that we have chosen to uniquely describe a player in a game, e.g. average health, times shot, enemies shot, enemies killed, etc. Then using statistical neural network approaches such as Principal Component Analysis or Factor Analysis we may explore the data so as to identify the correlational (or high order statistical) relationship between the variables. Factor analysis is particularly interesting in this regard because it is frequently used by statisticians in an exploratory mode. A well known example of the use of this method is where the statistical relationship for different forms crime in different cities are explored, e.g. murder, theft, robbery etc. Factor analysis can decipher which input variables have the strongest correlation and the statistician can interpret what this means. It may be found that there is a strong link between robbery and murder and so the output of the network that identifies this relationship may be said to have identified a correlational link which can be explained because these are violent crimes. Similarly, a non-violent crime correlational may be discovered. Using this method to explore player data we may have an advantage in our interpretation of the data because we can also collect information additional about the player that can help us interpret the statistical relationships, e.g. how old are they, sex, what type of games they like to play, how often do they play etc. These values could also be used in the statistical analysis but we would suggest that they may be better served in helping us interpret the correlations discovered by the outputs of the network. Whereas a clustering method would group players together so that we can label these groups as novice, normal or advanced, on the basis of the complete data point. Factor Analysis can identify

relationships between sub-sets of the data variables that may be used to identify more refined aspects of player behaviour, e.g. output one could identify the overall capability of the player and output two may identify whether the player is cautious or just dashes in etc. Being able to identify more subtle or complex aspects of player behaviour could be very valuable in tailoring the game experience to the player, and it also potentially opens up new possibilities for dynamic gameplay. For example, if we are able to discover patterns that relate more to player emotion or motivation then this may be used with other sensory devices to discern the needs or desires of the player and the game can be adapted to account for this.

AN ADVANCED FRAMEWORK FOR PLAYER-CENTRED GAMES

Two particular novel technology aspects that we discuss in this paper are monitoring adaptation through sensory equipment and dynamic player modelling and we explore these in more detail within this section. Detection of the need for the game to adapt based, for example, on measuring player frustration (Gilleade & Dix, 2004) is one approach for game adaptation but we propose a slightly different model, one in which the game is adapted on the basis of detecting player type coupled with game performance. The effectiveness of adaptation can then be measured by a reduction in the level of frustration and other measures. If adaptation does not improve player performance or their frustration levels then perhaps this is because the player has been classified incorrectly, or more likely as they have progressed through the game the model that fitted the player initially is no longer applicable. Therefore in this scenario it may make sense to reclassify or dynamically remodel the player – Fig 3 illustrates how this advanced framework may be executed.

Measuring the Effectiveness of Adaptation

We need to know when to adapt the game to a player (Gilleade & Dix, 2004) but also we should monitor if our adaptation has been effective or appropriate. If we make a change based on the game data coupled with the player profile and this frustrates, or hinders the player more (or vice versa) then we may make one of two conclusions: our adaptation is inappropriate or our model of the player is inaccurate. In either case this is a good reason to have the feedback loop in our model illustrated in Fig. 3.

Assuming that there are discrete changes to the adaptation of the game and that these have tested these thoroughly before game release, we then can focus on making sure that we classify the player correctly so that the state of the game is appropriate to them. This is especially important because players learn at different rates and so we need to take account of concept drift (Black & Hickey, 1999) in the classification process (see next section).

The manner in which we measure the requirement or appropriateness of adaptation may be most effectively achieved using affective computing techniques by monitoring a player's emotional state through input devices, coupled with in-game data. It seems clear by initial research that attempts to detect a player's emotion through input devices that it is not very straightforward. For example, the

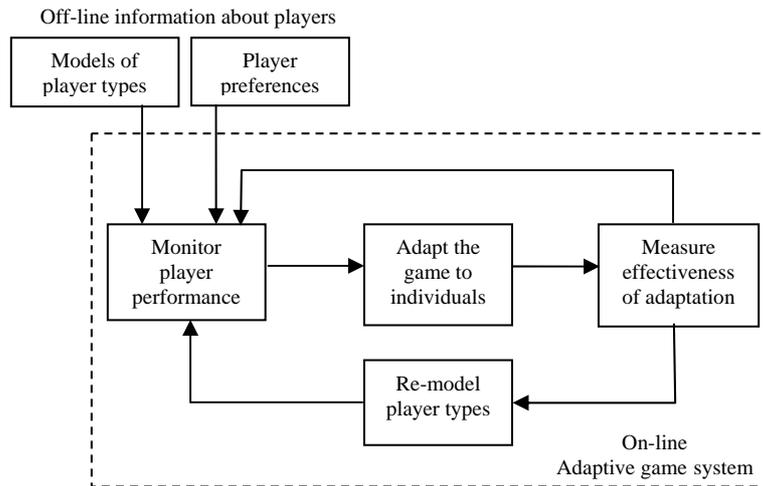


Fig. 3 Adaptive game system diagram illustrating the three phases that takes account of errors in adaptation.

emotional affect detected in the player through a gamepad analog button (Sykes & Brown, 2003) will vary with each player, game type and even perhaps when it's played. The information may be corrupted by interaction stress in, for example, playing an action game by altering physiological factors that would normally infer emotion, such as skin conductance (when using an appropriate sensor). Facial expressions or body movement may be used to infer the emotional state of the player – whether they are happy, content or frustrated – and game cameras such as the PS2 EyeToy is becoming more popular and widely incorporated into games. The difficulty with using a camera for facial expression though is that, to some degree, there is an expectation that the player will roughly maintain their position relative to the camera (Gilleade & Dix, 2004) – this is particularly an issue with game consoles.

With simple modification of existing input devices temperature or pulse (i.e. heart rate) sensors may be added like those on a typical exercise bicycle. These would not be expensive to implement but could potentially revolutionize game design with respect to a games' responsiveness to an individual players' needs. Even in casual way this information may introduce interesting new directions in gameplay – if you imagine a game from the horror genre such as "Silent Hill" or "Resident Evil". In this example the game could wait until a player seems at their more relaxed before landing that shocking surprise on him or her. Normally, games of this type must craft the levels and script events very cleverly to achieve the same effect, and it is very difficult to perfect. It will probably prove to be the case that one method alone will not be enough to accurately gauge a player's mood. That a mix information from standard sources such as the mouse or joy pad, along with more advanced sources of information provided by cameras or other sensory devices along with the player's profile, will be necessary to make decisions that tailor the game to individual players on the basis of their emotional state. Statistical methods such as neural networks will then be necessary to decipher the structural relationships in the data.

Dynamic Player Modelling and Reclassification

The idea that a player's model needs to be adapted has been recognised recently (Houlette, 2004) but this is still a very

new area for digital game research. On a basic level a player model may be thought of as a statistical representation of the player based on the frequency of repeated actions or average values of the parameters of their player character etc. It should be obvious then that an individual player's profile is likely to change throughout the progress of a game. This can be for all sorts of reasons, for example they are learning the action aspects of game more quickly than adventure aspects or perhaps they have reached a new gameplay dynamic in the game that they can't quite get to grips with – all players will be different so these things are very difficult to predict.

Because of the nature of game playing there will be new examples available about the player's profile as they play the game, hence the requirement for on-line learning. These on-line learning systems will receive examples on a continual basis and are required to induce and maintain a basis for classification and thus may have to deal with concept drift (Black & Hickey, 1999). Game players will adapt their strategy to survive or win as the game adapts to suit their profile. This change in the player behaviour, as discussed previously, may be part of their learning process: i.e. they get better at the game over time, or a may be forced into a strategic change of tactics. This adaptation, known as concept drift, can therefore be an immediate change in tactic or a slow progression to another. By concept drift we mean that some, or all, of the basis for defining a profile is changing as a function of time.

Typically there are a number of sub-tasks involved in the handling of drift within incremental classification learning. In increasing order of difficulty these are:

1. Identifying that drift is occurring;
2. Updating classification in the light of drift;
3. Tracking and modeling/analysing the pattern of drift over a period of time.

Machine learning techniques have been used with this form of user profiling/modelling in other domains such as cellular fraud in telecommunications (Fawcett & Provost, 1999). The aim was to analyse calling behaviour and detect anomalies. It also highlighted that patterns of fraud are dynamic; bandits constantly change their strategies to avoid detection. This links very well into game players having

profiles which change/evolve through the life of a game. Game players may be thought of as behaving like the fraudsters; they adapt and change their strategies as a mechanism to win/survive and so profiles can be monitored and adapted using existing machine learning techniques. For example, recent work (Black & Hickey, 1999) has demonstrated that profiles may be induced from telecommunication customers in relation to using a product, and that changes may be detected in the customers who are currently using the product.

As already indicated, player's profiles may change in many ways. We can break these down into two aspects of change: a progressive move – referred to as evolutionary adaptation, or immediate change – referred to as revolutionary adaptation (Black & Hickey, 1999). This work also introduces a methodology called TSAR (Time Stamp Attribute Relevance) which has been used successively to adapt to concept drift in telecommunication customer data (Black & Hickey, 2002). This methodology can be applied to neural network approaches, as discussed earlier, so as to deal with concept drift in online learning within digital games.

CONCLUSIONS

Modern digital games are extraordinarily good at many things but even the best examples of these games are still not very capable at monitoring players, distinguishing between different player groups and altering the game state to meet individual players' needs. In this paper we described a framework for dealing with this issue and providing more adaptable games, and in particular approaches for dealing with two particularly current issues: that of monitoring the effectiveness of adaptation through affective and statistical computing approaches, and the dynamic remodelling of players based on ideas from concept drift. We proposed several neural network approaches as part of the realisation of this framework and in future work intend to test these ideas further. The improvements that may come from positive developments in this area could be as straightforward as helping the player in learning how to play the game, through to encouraging gameplay innovation in digital games.

References

Beal C, Beck J, Westbrook D, Atkin M & Cohen P, "Intelligent Modeling of the User in Interactive Entertainment", pp 8 – 12, AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment, Stanford, 2002.

Black, MM, Hickey RJ, "Maintaining the Performance of a Learned Classifier under Concept Drift", *Intelligent Data Analysis* 3 (1999) 453-474.

Black MM & Hickey RJ, "Classification of Customer Call Data in the Presence of Concept Drift and Noise" *Soft-Ware 2002: Computing in an Imperfect World*, Belfast, 8-10 April 2002, in "LNCS 2311", Springer-Verlag (Berlin), ISBN 3-540-43481-X, Pages 74-87.

Charles D, "Enhancing Gameplay: Challenges for Artificial Intelligence in Digital Games", 1st World Conference on Digital Games 2003, University of Utrecht, The Netherlands, 4-6th November 2003.

Charles D, Livingstone D, "AI: the Missing Link in Digital Game Interface Design?", 3rd International Conference on Entertainment Computing, September 1st-3rd, 2004, Eindhoven, The Netherlands. *Edge Magazine* (UK) #139, Trigger Happy and Redeye articles, August 2004.

Fawcett T. & Provost F, "Activity Monitoring: Noticing Interesting Changes in Behavior", 5th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining, San Diego, pp 53 – 62, 1999.

Gilleade K. & Dix A, "Using Frustration in the Design of Adaptive Videogames", *Proceedings of ACE 2004, Advances in Computer Entertainment Technology*, ACM Press, 3-5 June, 2004.

Griesemer J. & Butcher C, "The Illusion of Intelligence", *Game Developer Conference*, San Jose, 2002, slides available from (last accessed 11/07/2004), <http://halo.bungie.org/misc/gdc.2002.haloai/talk.html>

Houlette R, "Player Modelling for Adaptive Games", pp 557-566, *AI Game Programming Wisdom II*, Charles River Media, 2004.

Livingstone D. & Charles D, "Intelligent Interfaces for Digital Games", AAAI-04 Workshop on Challenges in Game AI, 25-26th July 2004.

McGlinchey S, "Learning of AI Players from Game Observation Data", *GAME-ON 2003, 4th International Conference on Intelligent Games and Simulation*, pp. 106-110, Nov. 2003.

Miller S, "Auto-dynamic Difficulty", website forum debate, last accessed 11/07/2004, http://dukenukem.typepad.com/game_matters/2004/01/autoadjusting_g.html

Quinlan, J.R., *Induction of Decision Trees*, *Machine Learning*, 1(1), 81-106, 1986.

Rogers S. & Iba W, "Adaptive User Interfaces", 2000 AAAI Spring Symposium, Technical Report SS-00-01.

Sykes J. & Brown S, "Affective Gaming: Measuring Emotion through the Gamepad", *Human Factors in Computing*, pp 732-733 *CHI 2003*.

CASE-BASED PLAN RECOGNITION FOR REAL-TIME STRATEGY GAMES

Danny C. Cheng¹ and Ruck Thawonmas²
College of Computer Studies De La Salle University¹
2401 Taft Ave. Manila Philippines 1004
Email: chengd@dlsu.edu.ph

Intelligent Computer Entertainment Laboratory²
Department of Human and Computer Intelligence Ritsumeikan University
Kusatsu, Shiga 525-8577, Japan
Email: ruck@ci.ritsumeik.ac.jp

KEYWORDS

Case-based Plan Recognition, User modelling,
Real-time strategy game

ABSTRACT

The current game industry around the world is one of the fastest growing industries. One gaming genre that is very popular is the real-time strategy games. However, current implementations of games apply extensive usage of FSM that makes them highly predictable and provides less replayability. Thus, this paper looks at the possibility of employing case-based plan recognition for NPCs so as to minimize their predictability. The paper also looks into possible representation adaptations to minimize the resource requirement to maintain the possibility of deployment in mobile devices.

INTRODUCTION

Applying artificial intelligence in computer games has long been in use starting from the earliest days of such systems (Firclough et. al. 2002). With the recent rise in popularity of real-time strategy games, it can be noticed that most players of such games prefer to play against human opponents in a multi-player environment as opposed to playing against the computer player. This is due to the fact that minimal effort has been invested into the development and improvement of artificial intelligence in this field due to the enormous amount of overhead both financially and on computing power needed. (Buro and Furtak 2003)

One of the challenges in RTS games is the fact that in RTS games, the worlds normally feature numerous objects, incomplete information, micro-actions, and fast paced actions. Several currently

existing works focus mainly on slow-paced, or turn-based games that includes a lot of actions with global effect that would simply overwhelm the human player. (Buro and Furtak 2003)

In this paper, we present some existing works that could be adapted into the area of real-time strategy games. Issues and recommendations are stated at the end of this paper for further development.

REAL-TIME STRATEGY GAMES

Several fields of application and game genres currently exist wherein artificial intelligence research can be applied to. However, this paper focuses on RTS games specifically due to the numerous variety of research problems that exists within the aforementioned game genre. Some research problems would include the following (Buro and Furtak 2003):

- Adversarial real-time planning – planning can take place in several levels namely, strategic, tactical, and operational. Strategic planning would refer to what should be done, tactical refers to how to carry out such plans, while operational would refer to specific actions for each tactical decision. (Kaukoranto et.al. 2003) The problem here is that the environment is dynamic hence pre-defined rules and less than applicable hence alternative approaches have to be investigated.
- Decision making under uncertainty – human players are able to decide on specific plans or strategies even with the lack of information. They are also able to proactively determine the need to look for such information to gain an advantage. Such things might be interesting if they were incorporated to a computer player. (Kaukoranto et.al. 2003)

- Opponent learning and modeling – the ability to determine the players strategies and find ways to react to it in the proper level has been an ideal situation that has been sought after but not yet reached. Most games right now still follow a pre-determined plan of action.
- Spatial and Temporal reasoning – In an ever changing environment, strategies and plans have to be constantly reevaluated for applicability. Understanding of the environment should also be added so as smart decisions can be made.
- Resource management – another task that human players perform is the balancing of allotments for resources. Though recent games are fairly efficient with regards to this, again they are pre-programmed responses to a fixed world environment.
- Collaboration – this is clearly lacking in computer players wherein they never collaborate against a common enemy when attacking it. In contrast, human players usually form teams to fight against a stronger enemy.
- Path Finding – this has always been part of game research since most existing work deal with path finding. The ability to rapidly determine a path in a 2D terrain with moving objects and changing environments has always been a challenge.

In this paper, we look at CBPR (Case-based plan recognition) and some tweaking of the approach as a possible solution to some of the aforementioned research problems focusing more on the user modelling rather than path finding.

CASE-BASED PLAN RECOGNITION

Plan recognition refers to the act of an agent observing the actions of another agent whether it be human or computer-based with the intent of predicting its future actions, intentions, or goals. Several approaches can be used to perform plan recognition namely deductive, abductive, probabilistic, and case-based. It can also be classified as either intended or keyhole. An intended case-based plan recognition system assumes that the agent or the user is actively giving signals or input to the sensing again to denote plans and intensions. In the case of a real-

time strategy game, the user or player is focused on playing the game and not focused on trying to convey his or her intention to the sensing agent, hence for this scenario, we would be classified as keyhole plan recognition wherein predictions are based on indirect observations about the users actions in a certain scenario.(Fagan and Cunnigham 2003)

One specific attempt or implementation of case-based plan recognition(CBPR) in games used the game space-invaders as the target platform. (Fagan and Cunnigham 2003) Although this work demonstrated the applicability of CBPR to a certain extent, it also has made several assumptions in its work. First, the set of states are fixed to three, namely Safe(S), Unsafe(U), and Very Unsafe(VU), in a more complex game scenario or genre such as an RTS game, such states may not be finite or defined at the start as they may represent world states at a certain time.

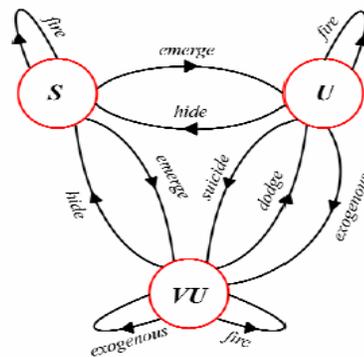


Figure 1. State transition diagram used in the implementation (Fagan and Cunnigham 2003)

EXISTING WORK AND THEIR PROBLEMS

One of the existing work that is applicable to the target domain of real-time strategy games is the work of (Kerkez 2003). The contribution of this work is the presentation of an approach on how to discover and locate plans from incomplete plan libraries.

Most existing work assume that there is a complete plan library to serve as a basis for plan-recognition. However, construction of such a plan-library may not only be not feasible, but the additional or extraneous libraries may affect the performance of the recognizer. (Lesh and Etzioni

1996) Another existing limitation is that “most traditional recognition systems reason in terms of planning actions and do not explicitly keep track of the world states visited during the execution of a plan, except for the initial and the goal states”(Kerkez and Cox 2002).

As illustrated in figure 2, the contribution of (Kerkez 2003)(Kerkez and Cox 2002) is that in the traditional blocksworld problem, the plan representation only incorporated the initial and the end state. The actions that are taken in between and the intermediate states that resulted during the execution was not recorded. This is compared with the representation in figure 2b which is the proposed representation. In contrast, here, the intermediate states are stored for future reference.

Although this approach provides more information and basis for plan-recognition, it also at this point introduced the problem of having too many states to manage and use during recognition. A possible solution as proposed by (Kerkez 2003)(Kerkez and Cox 2002) is illustrated in figure 3.

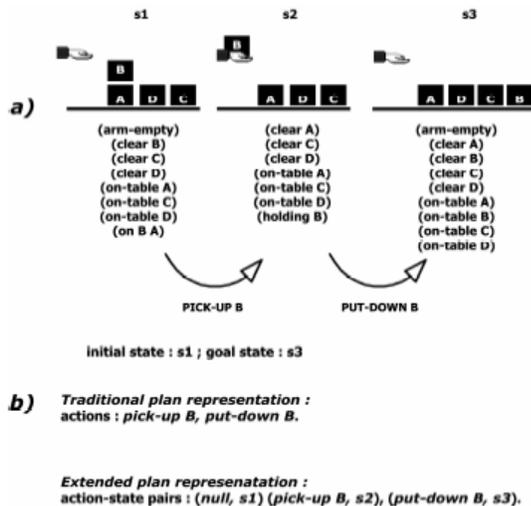


Figure 2. a) An example of a simple planning episode from the blocksworld planning domain. b) Two different views of the observed plan. (Kerkez and Cox 2002)

APPLICABILITY TO RTS GAMES

Based on the aforementioned works from various authors, we believe that there are several considerations needed to be added. The goal is to assist the human player in management tasks in an RTS game such as Warcraft, but not play the game

for the user. Although the concepts may be applicable for the opponent NPC, it is not our initial focus.

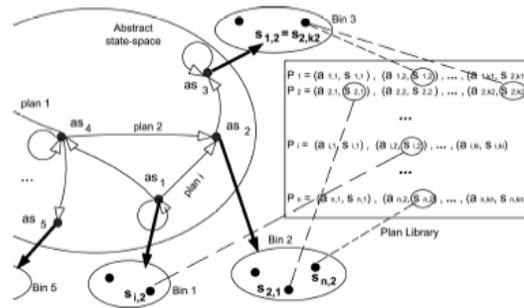


Figure 3. Indexing and storage structures. Abstract states (asi) point to bins (bold lines), containing world states (sj). World states in turn point (dashed lines) to past plans (Pj) in which they are contained. (Kerkez and Cox 2002)

In attempting to apply such methodologies to RTS games, our suggestion is that we limit first the scope of the environment being monitored and controlled by an NPC. This is in order to minimize the possible build up of states that will affect storage and its retrieval specially in limited environments such as a mobile device. An example of a possible limited scope would be in the case of a specific part of the map that contains establishments such as bases rather than the entire map. Hence, strategically, we would be looking at for example defense, or enrichment of resources and fortification rather than plans of attack. Although in (Kerkez 2003), an optimization scheme was suggested based on abstract states indexing and concrete states, it is still not determined if it will be applicable to an RTS game. This is mainly due to the fact that the algorithm may be NP Complete depending on the resulting graph representation of the states. The requirement being that the graph should be either planar or is a circular-arch graph.

Another issue that has to be considered in RTS games is that aside from changing states in the environment, the pieces available or in play can also change depending on the stage of the game being played. As new units are discovered or come into play at higher levels or stages of the game, actions monitored before and subsequent actions taken should be mapped to not only different environment but also different units. A

similarity measurement or mapping function may be provided so as to form correlations between what has been monitored before and what to apply now. In this respect we look at hybrid systems like (Schiaffino and Amandi 2000) wherein case-based reasoning which is used to build the cases or plans is combined with bayesian networks to determine the likelihood that a certain action would be performed. In RTS, this can be viewed as using CBR at strategic and tactical levels to determine similarity features for comparison with other players and bayesian networks can be used to predict the transitions from strategic planning to tactical actions, and then eventually to operational details of the task. Initial set of features for the case-base at each levels is listed in table 1 while table 2 shows a specific example of the case to be stored. The assumption here is that the different lower or more detailed levels are happening based on as a direct consequence of the higher levels decision much like the concept of a chain of command. Temporal information like continuous attacks from the opponent that may signify a certain strategy in use though important, is currently not considered in the case-base so as to maintain a level of simplicity at the start. Bayesian networks will be used to determine the subsequent node to be chosen in the next level (figure 4). It will also be used to account for the dynamic world states that could happen in the game such as will the same action be taken for the same scenario and same user profile given the history of the specific user and the variation in the existing types of objects or units present in the current situation.

Plan's determined can be executed at a local or isolated scope or domain so that it would more manageable. The system should also have a means of learning from erroneous predictions. Explicit corrections being made by the player to computer predicted plans and actions taken should be noted so that these can be considered in future attempts at predicting the players possible responses. Much of the considerations here in terms of abstraction and localization is mainly for the purpose of minimizing the resource requirements of the approach. The assumption here is that the client will be able to cache basic abstracted information for initial computation while additional information can be acquired or retrieved as the need arises.

Given all the considerations, the research work aims to perform comparisons based on prototyping and user testing to determine differences or improvements with traditional methods if any. We expect to implement these concepts initially on desktop platforms and then eventually port them to mobile devices such as Palms or PocketPCs. After which, both qualitative and quantitative analysis will be performed to gauge the performance of the system and its scalability and resource requirements. Test deployments on student population would also be included in the testing and evaluation of the results of this work

Table 1. Initial set of sample features at each level of the cases

| <i>Levels</i> | <i>Feature and Description</i> |
|---------------|--|
| Strategic | <ul style="list-style-type: none"> • Subset of the map for the vicinity in question • Map position • Race and Terrain information • Set of opponents within the vicinity • Set of buildings in the vicinity • Set of friendly units within the vicinity • Gameplay duration |
| Tactical | <ul style="list-style-type: none"> • Friendly units available and their capabilities • Resources available • Actions taken |
| Operational | <ul style="list-style-type: none"> • Command received • Position of Unit |

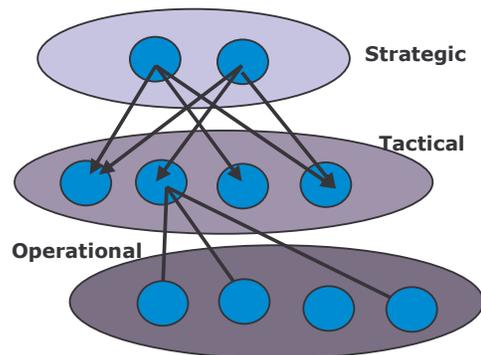


Figure 4. Relationship between the various levels of consideration

Table 2. Sample case entry based on the descriptions using Warcraft as a domain

| <i>Levels</i> | <i>Feature and Description</i> |
|---------------|---|
| Strategic | <ul style="list-style-type: none"> • 32x32 map unit (size of map) • Lower left corner of map • Human Island Town Terrain • Town hall, Peasants • 0 Enemy Units • 1:30 mins. Into game play |
| Tactical | <ul style="list-style-type: none"> • Town Hall Gold: 385 Wood:185 Hit Points: 1500/2400 Armor Type: Fort Armor:5/8 Sight:90/60 Build time:180 • Peasant Gold:75 Wood:0 Food:1 Hit points:220 Armor type: Medium Armor: 0 Sight:80/60 Speed:190 Build time:15 Attack type: Normal Weapon type: Normal Ground attack: 5.5 Air attack:None Cool down:2 Range:Melee • Build Peasant, Militia, Barracks • Set rally point • Scout out enemy position • Send continuous attacking militia |
| Operational | <ul style="list-style-type: none"> • Peasant: mine gold • Peasant: chop trees • Town hall: build militia • Militia: Wait at rally point • Militia: Move to upper right corner • Militia: Move to upper left corner • Militia: Move to lower right corner • Town hall: build barracks • Barracks: build footman |

CONCLUSIONS AND RECOMMENDATION

Currently, the proposed modifications and adaptation have yet to be implemented and tested empirically to determine the appropriateness of the suggestions. However, this research work does present several possibilities that would help improve game play on RTS games on both the desktop and mobile platform. There are additional considerations that are deemed to be ideal inclusions to the research. These would include the detailed study of temporal considerations and the concept of chain of events. In adding this to the research, it would greatly improve the accuracy of the predictions in terms of the plans of

a specific user. Also, unlike other games such as an adventure game wherein the goal is either constant or the change is predictable based on the game itself, in a real-time strategy game, the possibility of a change in strategy in the middle of game play is very possible and there is no support structure within the game itself that will aid in the identification of such changes. Hence, issues such as how often should re-evaluation happen comes into view. Also, the current assumption of a strategy is based on a subset map of a certain stage or world in an RTS game. In such events, issues such as complementing or supplementing strategies have yet to be researched on.

REFERENCES

- Fagan M., Cunnigham P. 2003, "Case-based plan recognition in computer games". Technical Report TCD-CS-2003-01 Trinity College Dublin Computer Science Department
- Fairclough C., Fagan M., Namee B. M., Cunnigham P. 2002. *Research Directions for AI in Computer Games*. Department of Computer Science Trinity College Dublin Ireland.
- Kaukoranto T., Smed J., Hakonen H., 2003 "Role of Pattern Recognition in Computer Games". *Proceedings of the 2nd International Conference on Application and Development of Computer Games* p. 189-194
- Kerkez, B, & Cox. M. T. 2002. "Case-based plan recognition with incomplete plan libraries". In B. Bell & E. Santos (Eds.), *Proceedings of the AAAI Fall 2002 Symposium on Intent Inference* (pp. 52-54). Menlo Park, CA: AAAI Press / The MIT Press.
- Kerkez, B. 2003. "Incremental Case-based Keyhole Plan Recognition." Technical Report #WSU-CS-01-01. Wright State University
- Lesh, N., & Etzioni, O. 1996. "Scaling up goal recognition". In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning* (pp 178-189).
- M. Buro & T. Furtak, 2003 "RTS Games as Test-Bed for Real-Time Research", *Invited Paper at the Workshop on Game AI, JCIS*
- Schiaffino S., Amandi A., 2000 "User profiling with case-based reasoning and bayesian networks". *International Joint Conference IBERAMIA-SBIA* pp. 12-21

T-Collide: A Temporal, Real-Time Collision Detection Technique for Bounded Objects

Erin J. Hastings, Jaruwan Mesit, Ratan K. Guha
College of Engineering and Computer Science, University of Central Florida
4000 Central Florida Blvd. Orlando, FL 32816
hasting@cs.ucf.edu, jmesit@cs.ucf.edu, guha@cs.ucf.edu

Abstract

This paper presents T-Collide, a fast, low memory-overhead, low execution-cost, time-based collision detection scheme. It is intended for real-time systems such as games or simulations to optimize collision detection between large numbers of mobile objects. Nearly all aspects of T-Collide are fully customizable to application specifics or implementer preference. T-Collide is based upon Spatial Subdivision, Bounding Volumes, Spatial Hashing, Line Raster Algorithms, and Continuous, or Time-Based Collision.

Keywords

temporal, continuous, real-time collision detection, uniform spatial subdivision, spatial hashing, bounding volumes, T-Collide

Introduction

T-Collide is a collision detection scheme for mobile objects in real time systems and is especially suited to computer simulations or games. It is based upon:

- **Spatial Subdivision:** A “Grid” divides the world into smaller areas.
- **Bounding Volumes:** Each complex object has a simple bounding volume.
- **Spatial Hashing:** A function that determines where an object is in the Grid.
- **Line Raster Algorithms:** When an object passes multiple grid cells in the same frame, grid location detection becomes similar to a line raster problem.
- **Collision over Time:** Collision is a function of position and time, not just position. The “T” signifies temporal collision.

T-Collide requires little memory overhead and requires no significant additional data structures to be integrated into an existing application. Virtually all

aspects of the algorithm are customizable to application or implementer preference. Use of the T-Collide algorithm imparts no arbitrary restrictions on application parameters such as object size or movement rate per frame.

Related Work

First, “required reading” regarding collision detection includes: the Lin-Canny closest features algorithm (Lin 1992), V-Clip (Mirtich 1998), I-COLLIDE (Cohen 1994), OBB-Trees (Gottshalk 1996), Q-Collide (Chung 1996), and QuickCD (Klosowski 1998). Another interesting approach to real-time collision is by Monte-Carlo Method (Guy 2004). Some excellent resources on general collision detection with an emphasis on games are: (Blow 1997), (Bobic 2000), (Dopterchouk 2000), (Gomez 1999), (Gross 2002), (Heuvel 2002), (Lander 1999), (Nettle 2000), and (Policarpo 2001). The recent endeavor most related to this one is (Gross 2002), which also utilizes spatial hashing. The hashing methods differ considerably however, and it does not consider collision over time.

Algorithm Overview

In order to implement T-Collide, the following are required:

- The Grid
- Bounding Volumes for all collision objects
- A Hash Function for each bounding volume type

First we will discuss assumptions regarding application setup. Then we will cover specifics of the Grid, Bounding Volumes, and Hash Function in turn, followed by the T-Collide algorithm itself. Finally we will analyze the algorithm’s time/space complexity and conclude the paper.

Assumptions and Restrictions

Regarding general application setup, the following is assumed:

- Animation involves a typical update/draw loop.
- Collision detection is performed every frame.
- Updates are based on some `TIME_ELAPSED` variable representing time elapsed since the previous frame.
- Object-environment collision is performed prior to object-object collision.
- Object position at the beginning and end of each frame is saved - `position_initial` and `position_final` respectively.
- Objects and their bounding volumes are significantly smaller than grid cells. As we will see, grid cells may be any arbitrary size, however being larger than the bounded objects increases performance.

The Grid

The grid divides the entire scene into distinct cells (uniform spatial subdivision). The grid may be 2D or 3D. A 2D grid is faster and is a better choice for applications where many objects are spread over a wide area, but rarely above each other. For example – flight simulations, naval simulations, isometric (“top down”) real-time strategy games, or space shooters with many units.

The grid requires no explicit storage of cells or content. Only a single variable is required: `CELL_SIZE`. Note that `CELL_SIZE` should be significantly larger than most objects in the scene to reduce chances of objects spanning multiple cells. While a single object spanning multiple cells is acceptable (for example, objects on a cell boundary) it does slow the algorithm somewhat. Thus `CELL_SIZE` will affect performance. Figure 1 depicts a grid.

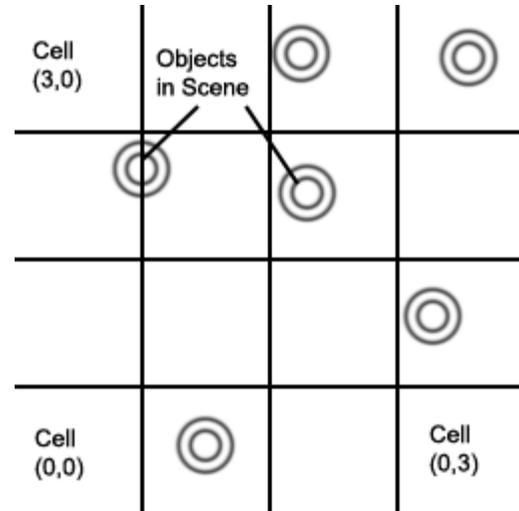


Figure 1 – The Grid

In a 2D grid, each cell is uniquely represented by two integers generated by the hash function. A 3D grid requires three integers. Neither grid cells, nor their contents, are explicitly stored. Each object in the scene has a parameter denoting the grid current cells it occupies.

Bounding Volumes

Each complex model is surrounded by a simple bounding volume. Bounding volumes may serve as either:

- the object’s collision model, where collision with the bounding volume signifies collision with the object
- a “first pass” indicator signifying possible collision with the object’s complex model, where more detailed collision detection is performed later

In the former case, detailed collision is not required and the object’s bounding volume is the object’s collision volume. The latter case is obviously required where complex hit detection is required and the bounding volume serves to signify “possible collision”. Either method can be employed in T-Collide.

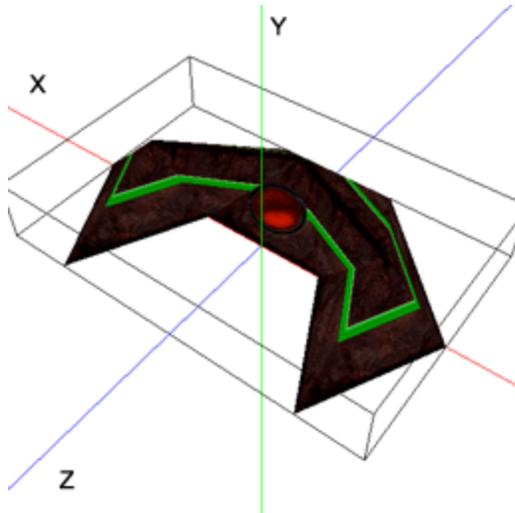


Figure 2 – Object within Axis-Aligned Bounding Box

Any bounding volume may be employed in T-Collide. In the following hashing examples we will use axis-aligned bounding boxes (AABB), Oriented-Bounding Boxes(OBB), and Spheres as bounding volumes for objects. An AABB can be represented by two points – min and max. An AABB never tilts; it is always aligned with the axis thus streamlining collision computation. OBBs are boxes as well, but tilt or rotate with the object. Figure 2 depicts an object enclosed within a bounding box.

The Hash Function

The hash function determines which grid cell an object is in and is run every frame on all objects. The hash function is based on:

- bounding volume type
- object's initial position (beginning of the frame)
- object's final position (end of the frame)

Note that an object may be in more than one grid cell, such as when crosses a cell boundary. For this reason, a list of grid cells is kept with each object called CELLS_OCCUPIED. Now let us examine a hash function.

Assume the following global variables or structures:

- Cell Size: the size of each grid cell

- Grid Cell(x, z): structure to uniquely identify each grid cell denoted by x,z since the grid is 2D, therefore y-axis values may be ignored
- Min, Max: the axis aligned bounding box

Suppose each object in the scene has the following variables:

- position_initial(x,y,z): position at the beginning of the frame
- position_final(x,y,z): position at the end of the frame
- boundingBox (min,max): the object's AABB
- cellsOccupied[...]: list of cells the object spans

Note that “cellsOccupied” is a list, since an object may span multiple cells if it is on a boundary. A majority of the time however it will be a single cell, since grid cells should be much larger than individual objects.

Hashing a Single Point

First, how can we determine what grid cell a single point is in? One obvious choice for a hash function is to simply divide the object's position by “cellSize”. Unfortunately division is slow and should be avoided. A better choice is to define a conversion factor that when multiplied by an object's position yields a unique grid cell. We can define the conversion factor as follows:

- `conversion_factor = 1/cellSize;`

So our hash function for a single point might look something like this.

```
gridCell hash(point p)
{
    gridCell g;
    g.x = p.x*conversion_factor;
    g.z = p.z*conversion_factor;
    return g;
}
```

Given point p a grid cell is returned. As an example, with a cell size of 10.0 hash(x=105.6, y=30.3, z=55.2) would return 2D grid cell (x=10, z=5).

Hashing a Bounding Box

We need to hash entire objects to the grid though, not just single points. Note that there are three cases where an object's bounding box may span either: 1, 2, or 4 grid cells. Figure 3 illustrates this. To determine what cells are spanned by an object at a certain position, we may have to consider the four corners of the object's bounding box.

The function for determining which set of grid cells a bounded object spans is to the right. The function takes a bounded object as a parameter and fills in the object's "cellsOccupied" list. Notice that the function short-circuit evaluates based on the fact that: if min and max hash to the same cell, no further evaluation is required (which will usually be the case).

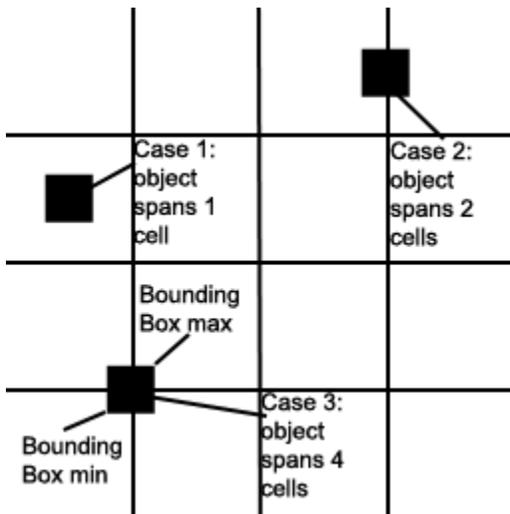


Figure 3: AABB Spanning Multiple Grid Cells

```
void determineGridCellAABB(Object obj)
{
    obj.cellsOccupied.clear(); // clear old
                                // values
    gridCell a = hash(max); // hash max
    obj.cellsOccupied.add(a); // add a to list
    gridCell b = hash(min); // hash min
    if(a!=b) // if a==b, we are done, else...
    {
        obj.cellsOccupied.add(b); // add b to
                                    // list
        point p = max; // compute first corner
        p.x = min.x;
        gridCell c = hash(p);
    }
}
```

```
if(c != b) // if c==b we are done, else...
{
    obj.cellsOccupied.add(c); //add c to
                                // list
    // add last point, doesn't need to be
    // hashed it is adjacent to min and
    // max
    gridCell d;
    d.x = a.x;
    d.z = b.z;
    obj.cellsOccupied.add(d); // add d
                                // to list
}
}
```

If OBBs are used, the min and max must be computed every frame due to box rotation. The max(x,y,z) point simply becomes the maximum x, y, and z values of all the OBB corners (see Figure 4). The min(x,y,z) is computed with min values, then hashing proceeds as with AABBs.

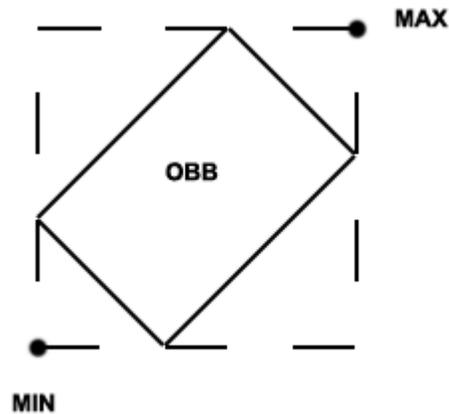


Figure 4 – Max and Min of an OBB

If bounding spheres are used then sphere center + radius values must be hashed. Namely: center.x + radius, center.x - radius, center.z + radius, and center.z - radius. Spheres will be slower to hash however since four points must always be considered (see Figure 5), whereas boxes may short-circuit evaluate if min and max hash to the same cell.

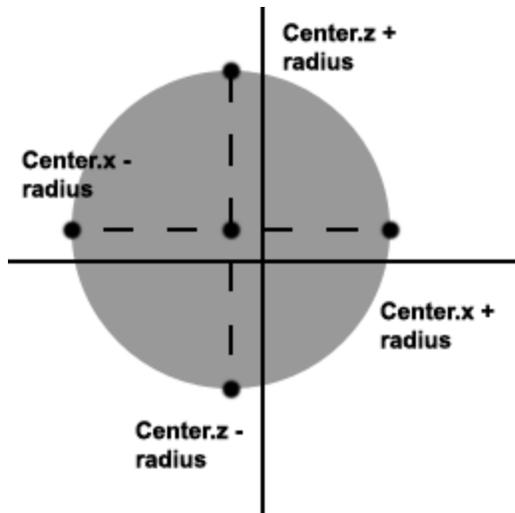


Figure 5 – Hashing a Sphere

Hashing Objects Over Time

Since collision must be determined over the course of the frame there is another factor to consider – the object’s position at the start and the end of the frame. For example, suppose at time T1 Object 1 and Object 2 are in positions designated by Figure 6. Then suppose after update their new positions are as shown at time T2. If collision detection is based only on position at time T2, then this collision will go undetected. Clearly collision is a function of position and time, not just position.

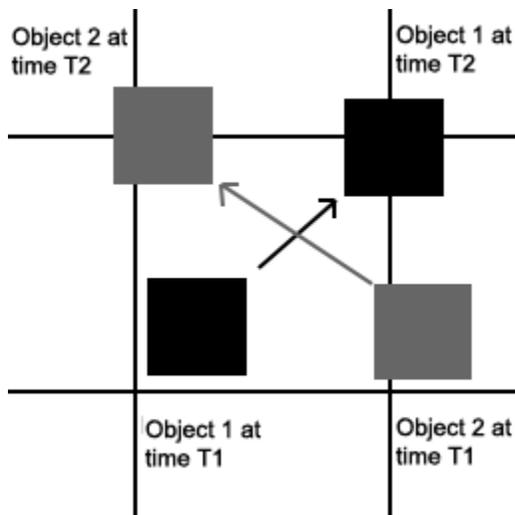


Figure 6 – Collision Over Time

Thus, we must determine all grid cells the object has passed through during the frame (cellsOccupied). Determining which cells an object traverses is done as follows.

- Determine the initial cell the object occupies
- Determine the final cell the object occupies
- Find the cells traversed between them

The issue is somewhat similar to a line raster problem – where the endpoints of the line are the initial and final positions and the “pixels” are the cells traversed. Figure 7 illustrates this, where shaded cells are added to the object’s “cellsOccupied” list.

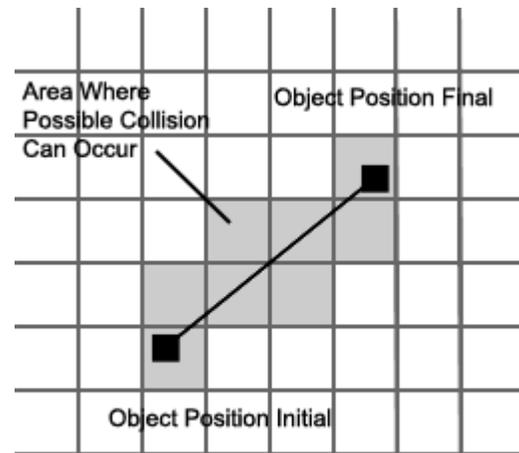


Figure 7 – Cells Traversed by an Object

There are various ways to approach this problem. One issue to note is that simply tracing a line between the two objects will yield a margin of error, since objects are obviously much wider than lines. Figure 8 illustrates this, where the object would clearly intersect the lower right cell but does not.

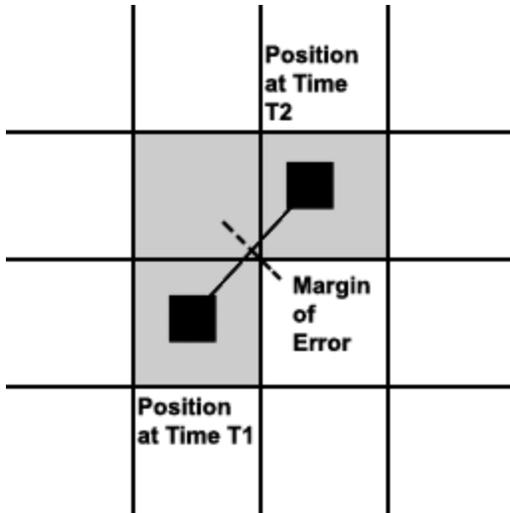


Figure 8 – Determining Cells Traversed During the Frame

We will take a simplistic approach with no margin of error. Suppose an object hashes to cell A at the beginning of the frame and cell B at the end of the frame. We will simply “draw a box” that encompasses the cells from A to B. The algorithm works as follows where A and B are Grid Cells (x,z), A is the lower valued cell (i.e $A.x < B.x$ and $A.z < B.z$), and $A \neq B$.

```
for(i=A.x; i<(B.x-A.x); i++)
  for(j=A.z; j<(B.z-A.z); j++)
    cellOccupied.add( i,j );
```

This may seem a sub-optimal or “brute force” approach but consider:

- Cells are significantly larger than objects
- Time between frames is fairly small in real-time applications

From this we can conclude that an object will almost always move no more than one cell from its current position during the frame. Clearly a complex algorithm for traversing cells is not needed, and may actually slow computation somewhat. In almost all situations cell traversal over the course of the frame will be similar to one of the three cases presented in Figure 9. Thus our simple nested “for” loop is adequate.

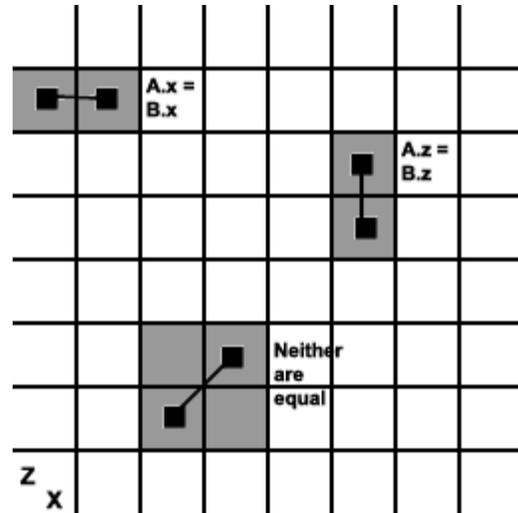


Figure 9 – Hashing Objects Crossing Cell Boundaries

There is one case however, where tracing a line through the grid is entirely appropriate – when the object is represented simply by an origin and a vector. Bullets and other “instant hit” weapons are often implemented this way in games. A vector from the origin of the projectile out to its range can be traversed through the grid. All objects within those traversed cells are possible targets of the bullet. Note that the grid optimizes target computation in this case – if the cells are traversed in order from origin along the vector, the closest possible targets will be found first, allowing early exit.

The T-Collide Algorithm

Given the following:

- Cell Size: size of each cell
- Grid Cell (x,z): data structure uniquely identifying each cell
- Bounded Objects: each object stores a bounding volume, a list of the grid cells it currently occupies, an initial position, and a final position

Collision detection with the T-Collide algorithm proceeds as follows:

- For each object in the scene:
 - Hash object to a set of “cells occupied” (Hash Function)
- For each object in the scene:
 - Compare “cells occupied” list to all other object “cells occupied” lists
 - If any cell matches, do collision (Collision Function)

The Hash Function for AABB or OBB proceeds as follows:

- A = hashed cell at beginning of frame
- B = hashed cell at end of frame
- If A == B we are done, else
 for(i=A.x; i<(B.x-A.x); i++)
 for(j=A.x; j<(B.z-A.z); j++)
 cellOccupied.add(i,j);

Actual collision and response will vary greatly from application to application. Some suggested resources that consider collision over time: (Dopterchouk 2000) and (Heuvel 2002) for spheres, (Lander 1999) and (Policarpo 2001) for AABBs, and (Blow 1997), (Bobic 2000), (Gomez 1999), and (Nettle 2000) for ellipsoids, rays, OBBs, or other shapes.

As a final note, one common mistake in writing collision loop algorithms that compare “all objects in the scene” is to proceed as follows:

```
for(i=0; i< num_objects; i++)  
  for(j=0; j<num_objects; j++)  
    if(i!=j) doCollision(object[i], object[j]);
```

This can be abbreviated to the following, which avoids the check for whether or not an object is being compared to itself (i!=j) but still compares all objects:

```
for(i=0; i< num_objects-1; i++)  
  for(j=i+1; j<num_objects; j++)  
    doCollision(object[i], object[j]);
```

Analysis of the Algorithm

With regard to space complexity T-Collide requires only a list of grid cells (2 integers per cell) per object in the scene:

- **Best case:** 2 integers per object in the scene. Since a majority of objects will hash to only one cell, this case will be the most common.
- **Worst Case:** 16 integers (8 grid cells) or possibly more, but this will be exceedingly rare.

With regard to time complexity the hash function breaks down as follows, assuming use of OBB or AABB for a single object bounding volume:

- **Best Case:** 2 floating point multiplies (to hash min and max at position_final) and 1 integer comparison (if A==B). This case will be the most common where each object hashes to only 1 grid cell.
- **Worst Case:** 6 floating point multiplies (hash 3 points) and 2 integer comparisons. Also, possibly a small nested double “for” loop of integer comparisons when an object travels across multiple cells. As noted above, this case will be exceedingly rare.

With regard to time complexity, comparison of two objects A and B to detect possible collision breaks down as follows:

- **Best Case:** 2 integer comparisons. Specifically, if (A.GridCell.x != B.GridCell.x && A.GridCell.z != B.GridCell.z) which will short-circuit if the first case is not true. Since a majority of objects should hash to a single cell, and most objects won't hash to the same cell, the majority of collisions can be safely resolved with either 1 or 2 integer comparisons.
- **Worst Case:** comparison of 8 or more grid cells (16 integer comparisons). Fortunately, this will almost never happen since: 1) individual grid comparisons short-circuit as noted in best case above, and 2) once a matching grid cell is found the comparison stops.

Summary

What T-Collide does provide with respect to performance and output is:

- **Low Memory Cost** – T-Collide requires no additional large storage structures to be added to an application. A few integers per collision object are all that is required. This makes T-Collide suitable for applications deployed on limited resource clients.
- **Low Execution Cost** – This is obviously required for any real-time application. A majority of object-object collisions can be resolved with the comparison of a few integers.

- Real Time Collision Detection of Bounded Objects: T-Collide is meant to be fast, and therefore ideally suited for real-time applications. It can, of course, be used in non-real time applications as well with little modification.
- Flexibility: All aspects of T-Collide are fully customizable – object bounding volumes, Grid cell size, level of detail, or the hash functions. Experimentation with variables is encouraged since they will significantly affect performance.
- Ease of Implementation: T-Collide is designed to be simple enough to add to any existing application with and provide good results with as little hassle as possible.

What T-Collide does not provide with respect to performance and output is:

- Perfect Physical Accuracy: As with most real-time graphics algorithms, the objective of T-Collide is not to model real world physics perfectly. Its objective is to provide “good” or fairly realistic results very quickly.
- Perfect Optimization: T-Collide may not necessarily be optimal in some cases. However, its simplicity provides an excellent effort/results ratio, and its low memory cost and speed provide an excellent overhead/results ratio.
- Object-Environment Collision: T-Collide does not provide object-environment collision which is better left to rendering algorithms such as BSP-trees, oct-trees, quad-trees, etc... that are optimized for huge amounts of static data. T-Collide is for collision detection between many highly mobile objects.

References

Blow, J. 1997. “Practical Collision Detection”. Proceedings of the Game Developers Conference 1997.

Bobic, N. 2000. “Advanced Collision Detection Techniques”. GamDev.net feature article.

Chung, K. 1996. “Quick Collision Detection of Polytopes in Virtual Environments”. ACM Symposium on Virtual Reality Software and Technology. July 1996.

Cohen, et. al. 1995. “I-Collide: An Interactive and Exact Collision Detection System for Large Scale Environments” ACM Interactive 3D Graphics Conference 1995.

Dopferchouk, O. 2000. “Simple Bounding-Sphere Collision Detection”. GamDev.net feature article.

Eberly, David. 2001. *3D Game Engine Design: A Practical Approach to Real-Time Graphics*. San Diego: Academic Press.

Gomez, M. 1999. “Simple Intersection Tests for Games”. GamDev.net feature article.

Gottshalk, S. et. al. 1996. “OBB-Tree: A Heirarchical Structure for Rapid Interference Detection”. Proceeding of the ACM SIGGRAPH 1996.

Gross, M. et. al. 2002. “Optimized Spatial Hashing for Collision Detection of Deformable Models”. *Vision, Modeling, and Visualization 2003*. (Munich, Germany, Nov. 19-21).

Guy, S. and Debunne, G. 2004. “Monte-Carlo Collision Detection”. PHD thesis paper.

Heuvel, J. and Jackson, M. 2002. “Pool Hall Lessons: Fast, Accurate Collision Detection between Circles or Spheres”. GamDev.net feature article.

Klosowski, J. 1998. “Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments”.

Lander, J. 1999. “When Two Hearts Collide: Axis Aligned Bounding Boxes”. Game Developer Magazine. February, 1999.

Lin, M. 1992. “Efficient Collision Detection for Animation and Robotics”. PHD thesis paper.

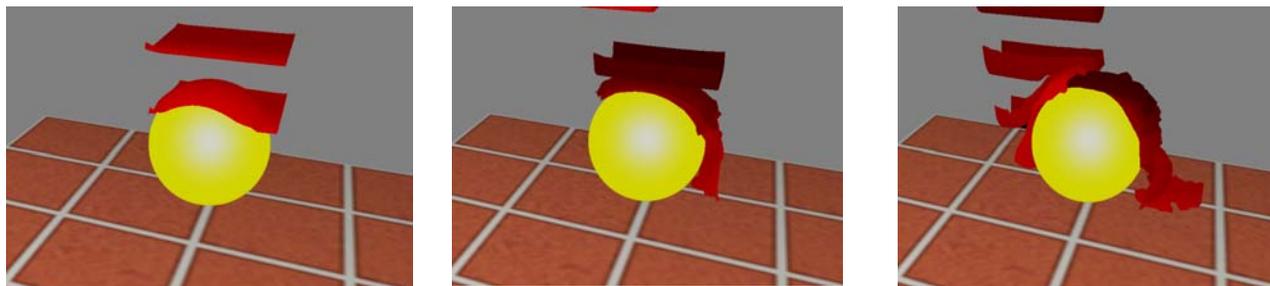
Mirtich, B. 1998. “V-Clip: Fast and Robust Polyhedral Collision Detection”. ACM Transactions on Graphics. July, 1998.

Nettle, P. 2000. “General Collision Detection for Games Using Ellipsoids”. GamDev.net feature article.

Policarpo, F. and Watt, A. 2001. “Real Time Collision Detection and Response with AABB”. SIBGRAPI 2001.

OPTIMIZED COLLISION DETECTION FOR FLEXIBLE OBJECT IN A LARGE ENVIRONMENT

Jaruwan Mesit Ratan K. Guha Erin J. Hastings
Department of computer science, University of Central Florida
4000 Central Florida Blvd., Orlando, Florida 32826
jmesit@cs.ucf.edu guha@cs.ucf.edu hastings@cs.ucf.edu



Abstract

We present a scheme for collision detection of flexible models. This scheme relies on a multi-resolution technique of: 1) object location tracking to decrease time complexity, 2) the combination of bounding boxes (AABBs) and a hash grid function, and 3) computation of distance domain to determine collision with a contact surface. Experiments show that this approach can efficiently detect collisions in a large set of flexible objects.

Keywords: collision detection, multi-resolution method, hash grid function, location tracking, flexible body

Introduction

There are many collision detection problems addressed by many different techniques. Bounding volume hierarchies have been developed to speed up intersection tests of close bodies. For example: bounding spheres (Hubbard 1995, James and Pai 2004), axis-aligned bounding boxes (AABBs) (Bergen 1997, Teschner et al. 2003), oriented bounding boxes (OBBs) (Gottschalk et al. 1996), quantized orientation slabs with primary orientations (QuOSPOs) (He 1999), and discrete-oriented polytopes (K-DOPs) (Klosowski et al. 1998).

VCLIP (Mirtich 1998) and CLOD with dual hierarchy (Otaduy and Lin 2003) have been also proposed for polyhedral objects collision detection. Separation-sensitive collision detection (Erickson et al. 1999) was presented for convex objects while

Kinetic collision detection (Basch et al. 2004) is designed for simple polygon.

For many interactive 3D graphics, animation and visualization applications, many papers have also employed spatial decomposition for both screen and input models to reduce time complexity. The simple idea of screen spatial decomposition is to subdivide the screen into grid cells and determine the grid cell to animated objects from the positions, similarly for input models. The spatial decomposition techniques that have been proposed are octree (Moore and Wilhelms. 1988), BSP tree (Naylor et al. 1990), brep-indices (Bouma and Vanecek, Jr 1991), k-d tree (Held et al. 1995), bucket tree (Ganovelli 2000), hybrid tree (Larsson et al. 2001), BVIT (Otaduy and Lin 2003), and uniform space subdivision (Teschner et al. 2003). Moreover, there are some collision detection algorithms for a large environment such as I-COLLIDE (Cohen et al. 1995) and CULLIDE (Govindaraju et al. 2003).

Most efforts have been in solving the collision detection problem for rigid bodies. This usually entails large sets of static data. Flexible objects however, have dynamically changing geometry. There may be few, if any, fixed points in the scene. Therefore, collision detection techniques optimized for rigid bodies are often not applicable, or less optimal for collision between flexible bodies. We propose a collision detection technique that can efficiently deal with a large set of flexible models.

The remainder of this paper proceeds as follows: first, an overview of the proposed algorithm, then a set of the experiments with the algorithm and results,

and finally, conclusions and discuss possibilities of further work.

Algorithm Overview

For collision detection with flexible objects in a large environment, we present the notion of tracking with bounding boxes, collision grid domain and contact surfaces by following:

1) Tracking with Bounding box (TB): all flexible objects are surrounded by a bounding box (AABB) for tracking. This tracking depends on the velocity gap between time periods. Two points, a minimum and a maximum, define the bounding box. The technique may be modified to make use of any bounding volume however.

2) Collision Grid Domain (CGD): we also present the constraint of animation that can make structural objects move smoothly in a 3-dimensional environment. We then determine the hash index to an object by using a hash function. If the objects are in the same hash index, they are in the same collision grid domain.

3) Contact surface (CS): to compute contact surface, distances of the flexible objects in same collision grid domain (CGD) are calculated. If the distances are less than the collision tolerance, collision or collisions will be detected.

Tracking with Bounding box (TB)

For computation in a linear system at each step time, we use location tracking. We put boundary boxes (AABBs) to determine the location of object in this step. We use the following notation:

Geometry

x^i is the geometric component in object i .
 $v_0^i \dots v_n^i$ are the positions of vertices for object i .
 v_{kx}^i is the position of vertex v_k^i in x direction.
 v_{ky}^i is the position of vertex v_k^i in y direction.
 v_{kz}^i is the position of vertex v_k^i in z direction.
 m^i is the mass of object i .

Direction

${}_nve_k^i$ is the velocity of v_k^i at time n .
 ${}_nn_k^i$ is the normal vector of v_k^i at time n .
 ${}_nf_k^i$ is the force of v_k^i at time n .
 ${}_nVN_k^i$ is dot product of ve_k^i and nn_k^i at time n .

To animate the object, we have:

1) Rigid body has one external force.

fg is the external force (gravity)
 2) Flexible body has one external force and one internal force.

fg is the external force (gravity)
 fs is the internal force (mass spring)

${}_nx^i$ indicates the time beginning with an arbitrary time t_0 .

$${}_nx^i = x^i (t_0 + n dt)$$

${}_nd(x^i, x^j)$ is the distance between observed object i and j at time n .

$${}_nd(x^i, x^j) = \text{sqrt} \left(\sum_{n^p \rightarrow n^x, n^y, n^z} (n^p - n^j)^2 \right)$$

Δx , Δy , and Δz denote the distance of moving object in x direction, y direction, and z direction in each step time.

$$\begin{aligned} |{}_n-1X_x^i - {}_nX_x^i| &= \Delta x \\ |{}_n-1X_y^i - {}_nX_y^i| &= \Delta y \\ |{}_n-1X_z^i - {}_nX_z^i| &= \Delta z \end{aligned}$$

To display objects moving smoothly, we offer the constraint, $\Delta x + \Delta y + \Delta z \leq 1$ for each time step.

Euler integration

We use Euler 1st order integration to animate an object. Euler integration consists of the following steps. First, set time to the initial value t_0 . Next, compute the level of time step by using an interval between frame to frame. Then, contribute the rate of change from time n to time $n+1$. Euler integration is appropriate for ease of implementation (Desbrun et al 1999).

From Euler 1st integration we have

$$\begin{aligned} dve_k^i/dt &= f_k^i / m^i \\ dve_k^i &= f_k^i dt / m^i \end{aligned}$$

Next, we have

$${}_{n+1}ve_k^i = {}_nve_k^i + (f_k^i / m^i) dt$$

Then, we have

$${}_{n+1}v_k^i = {}_nv_k^i + {}_{n+1}ve_k^i dt$$

From this equation we get the position of each vertex in the flexible objects for each time step. We bounded the object with a bounding box (AABB) by finding the minimum point and maximum points in each flexible object. Then we can find the overlapping between the bounding boxes (AABBs). Next, we put the position of each vertex of object i to hash grid function.

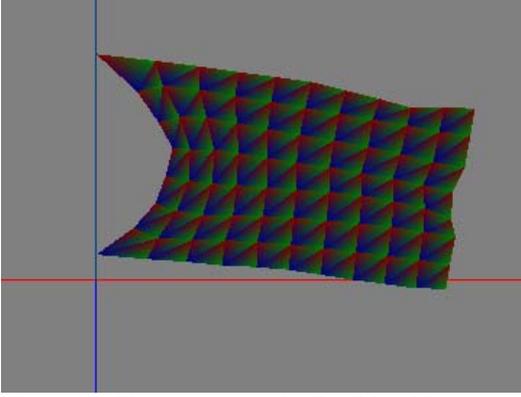


Figure 1: The structure of a flexible object composed of a set of vertices, velocities and forces.

Lemma 1: An object x^i is not colliding with any object if they are not overlapping of bounding box volume.

Proof : This is obvious by definition of bounding box. The objects bounded with a box will collide if there is the intersection between bounding boxes.

Collision Grid Domain (CGD)

It is very efficient to use a hash grid function for spatial subdivision (Teschner et. al 2003). For each time step, we applied hash grid function to each vertex in the overlapping objects to determine the hash index for the animated objects with respect to a user-defined cell size. we can present it by following

$$\begin{aligned}
 xl &= \text{size of grid cells in x axis} \\
 yl &= \text{size of grid cells in y axis} \\
 zl &= \text{size of grid cells in z axis} \\
 h &= \text{hash}(nv_k^i, xl, yl, zl) \\
 h &= (\text{floor}(v_{kx}^i/xl) * xconstant + \\
 &\quad \text{floor}(v_{ky}^i/yl) * yconstant + \\
 &\quad \text{floor}(v_{kz}^i/zl) * zconstant) / \text{hash index size}
 \end{aligned}$$

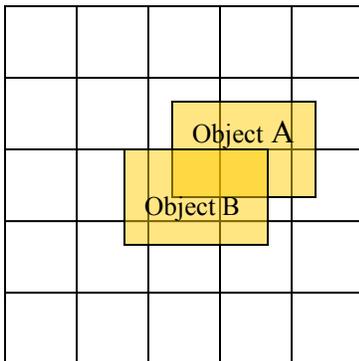


Figure 2: The combination of Grid hash function and bounding box AABB

Suppose there exist m objects (x^0, \dots, x^m) in the simulation frame. Also, suppose each object is composed of vertices v^0, v^1, \dots, v^n where $0 \leq i \leq m$.

Lemma 2: Given n vertices v^0, \dots, v^n . v_k^i does not belong to CGD if object x^i does not intersect with $x^0, \dots, x^{i-1}, x^{i+1}, \dots, x^m$ where $0 \leq i \leq m$ and v_k^i is not in the considering grid cell.

Proof: Follows trivially from the definition of CGD. We use Lemma 2 to check if a vertex belongs to the CGD.

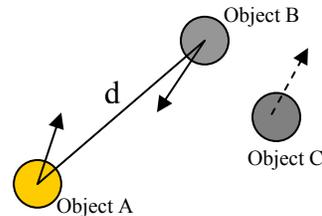
After the hash index and CGD have been determined, this technique computes the distance of vertices in the CGD. Using the distance, the algorithm can detect touching, hitting or throwing.

Contact surface (CS)

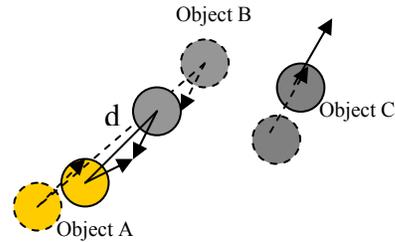
This is the step that we find the distances of vertices in CGD and then we compare with collision tolerance. This step returns true or false. True is colliding while false is not colliding.

Lemma 3: Given v_k^i as a point p^i and v_k^j as a point p^j . v_k^i and v_k^j belong to CGD. v_k^i and v_k^j are collide if $_n d(p^i, p^j) < \text{collision tolerance}$ and $_n VN_k^i < \text{velocity tolerance}$ at time n .

Proof: By the definition of distance computation, we determine collision tolerance. If the distance between p^i and p^j is less than the collision tolerance, the vertices are going to touch each other.



Time $t = n$, $d > \text{collision tolerance}$, they don't collide.



Time $t = n+1$, $d < \text{collision tolerance}$ and $VN < \text{velocity tolerance}$, they collide.

Our algorithm uses three passes to compute collision detection in each time step.

- First pass: we check if x^i overlaps with at least one of the objects x^0, \dots, x^{i-1} or x^{i+1}, \dots, x^m by using intersection of bounding boxes.
- Second pass: In case bounding box of object x^i overlaps with bounding box of object x^j , we check if vertices $v^i_1, v^i_2, \dots, v^i_n$ and $v^j_1, v^j_2, \dots, v^j_n$ are in the same grid cell or not. If so, we determine those vertices are in the CGD.
- Third pass: We calculate to see if a surface is contacted by another object.

In this algorithm, we can consider more than one pair of collision so that multiple flexible objects are detected for a large environment of simulation.

Experiments and results

We compared our algorithm with two other algorithms using five different simulations (A, B, C, D and E) to exam the relative performance of our algorithm. Simulation A, B, and C are composed of 315 vertices in each flexible object whereas D and E are created from 1189 vertices. For simulation A and D, we made 10 flexible objects in simulation, presented in Figure 2. Each object can be touching each other and colliding with a sphere. Simulation B is created with 100 flexible objects with one sphere while C and D are made from 100 flexible objects with 100 spheres shown in Figure 3 and 4 respectively. In case of D and E, we present how these features can affect our algorithms if there are more vertices in the flexible objects. Our scenarios are performed in an update method and the results of our algorithms are compared to the results of other algorithms.

| Model | Number of flexible objects | Number of spheres | Number of vertices |
|-------|----------------------------|-------------------|--------------------|
| A | 10 | 1 | 3150 |
| B | 100 | 1 | 31500 |
| C | 100 | 100 | 31500 |
| D | 10 | 1 | 11890 |
| E | 100 | 100 | 118900 |

Table 1: The specification of each test model

In our experiments, we used our simulations with grid hash function algorithm, bounding Box (AABB) algorithm and our algorithm.

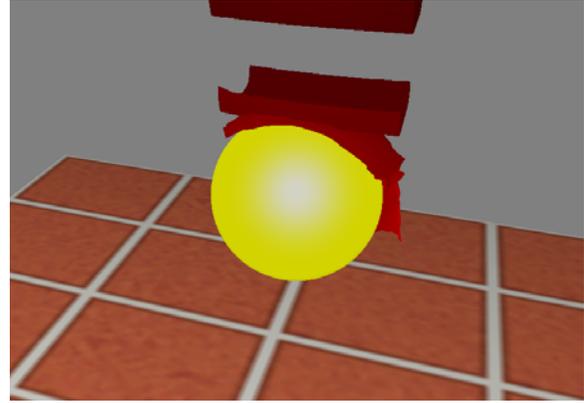


Figure 2: The simulation A and D

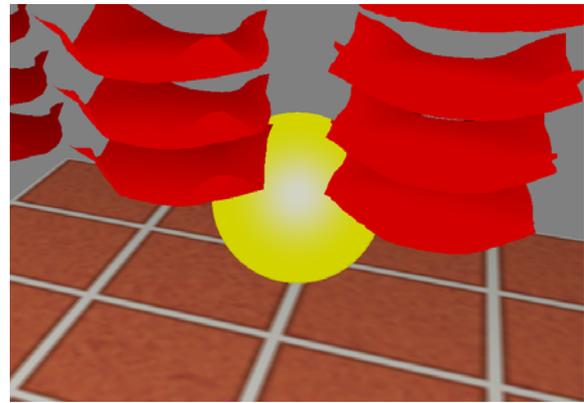


Figure 3: The simulation B

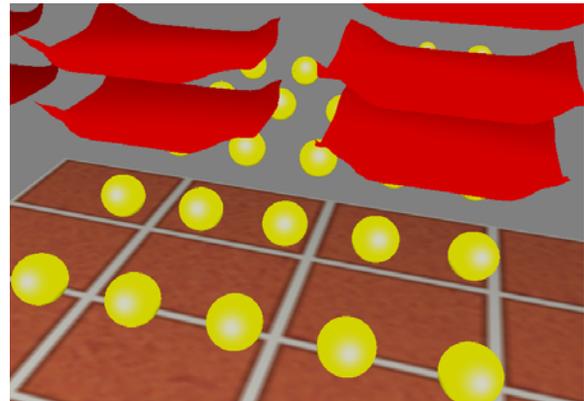


Figure 4: the simulation C and D

| Model | Grid Hash function (ms) | Bounding (ms) | Our algorithm (ms) |
|-------|-------------------------|---------------|--------------------|
| A | 380.78 | 95.5067 | 80.55 |
| B | >3000 | 419.737 | 407.183 |
| C | >3000 | 560.94 | 556.867 |
| D | >3000 | 1254.77 | 652.44 |
| E | >3000 | 2168.25 | 1786.07 |

Table 2: The running time of each algorithm

| Model | Best Case (ms) | Average Case (ms) | Worse case (ms) |
|-------|----------------|-------------------|-----------------|
| A | 58.1 | 65.314 | 80.55 |
| B | 391.313 | 396.978 | 407.183 |
| C | 520.75 | 533.569 | 556.867 |
| D | 149.587 | 313.253 | 652.44 |
| E | 1485.6 | 1583.048 | 1786.07 |

Table 3: The running time of each model in this simulation

Table 2 presents the time spent for each algorithm. If we used only a hash grid function, it spent more than 3000 millisecond in the simulation B, C, D and E which is not acceptable in real time animation. Then we applied the bounding boxes (AABBs), it spent less time. However, the best performance is our algorithm. Similarly, we can see the average time spent for the best case, average case, and worse case from Table 3. The best case meant there was no collision in the simulation frame and the worse case meant there was the most collision during this simulated time period.

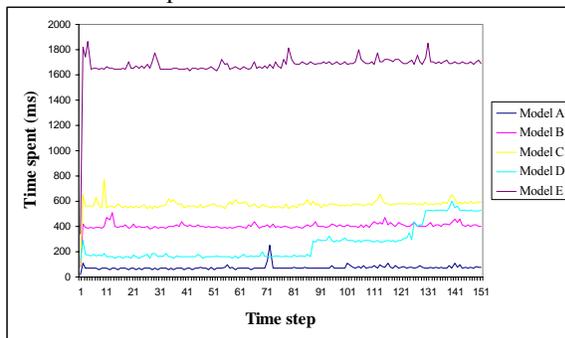


Figure 5: Collision detection performance per time step

Figure 5 shows the time spend in milliseconds when we applied our approach to each simulation. The graphs of simulation A, B, and C are fairly non-fluctuating or increasing while D and E's are increasing because the flexible objects composed of more vertices are colliding. The collision can be detected in the acceptable time when we consider object creation and collision detection time spends.

The results show that our algorithm can reduce time complexity even though there are more flexible objects or vertices in them.

Conclusion and Future work

In this research, we have introduced the concept of tracking with bounding box (TB), collision grid domain (CGD), and contact surface (CS). Our method essentially extends the benefits of two leading existing approaches with concepts of contact surface. All flexible objects are surrounded by a bounding box with maximum point and maximum point. Then, vertices in overlapping bounding boxes are determined into collision grid domain by hash grid function. The vertices in same collision grid domain are, finally, observed for surface contact. The result of this work performed with 100k vertices showed that this algorithm is efficient collision detection for flexible bodies in real-time considering the time spent for animation.

This research can be extended to several ways:

- Simplify the operation process to optimize the algorithm that has been implemented in this paper.
- Simulate the structure of tree to determine the area of collision in the object body and perform the collision detection in the overlapped area to reduce time complexity.
- Create the suitable test cases to compare with different algorithm.

References

- Basch, J., Erickson, J., Guibas, L.J., Hershberger, J., and Zhang, L. 2004. "Kinetic Collision Detection between Two Simple Polygons." *Computational Geometry* 27 2004, 211-235.
- Bergen, G.V.D. 1997. "Efficient Collision Detection of Complex Deformable Models Using AABB Trees." *Journal of Graphics Tools* 1997, vol. 2, Issue 4 (Apr.): 1-13.
- Bouma, W. and Vanecsek, G. Jr. 1991 "Collision Detection and Analysis in a Physical Based Simulation." *Eurographics Workshop on Animation and Simulation 1991* (Vienna) 191-203.
- Cohen, J.D., Lin, M.C., Manocha, D., and Ponamgi, M. 1995. "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments." *Proceedings of the 1995 symposium on Interactive 3D graphics 1995* (Monterey, CA, United States) 189-196.
- Desbrun, M., Schröder, P., and Barr, A. 1999. "Interactive Animation of Structured Deformable Objects." *In Graphics Interface 1999* (Kingston, Canada, Jun.).
- Erickson, J., Guibas, L.J., Stolfi, J., and Zhang, L. 1999 "Separation-Sensitive Collision Detection for Convex Objects."

Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms 1999, Baltimore, Maryland , 327 – 336.

Ganovelli, F., Dingliana, J., and O'Sullivan, C. 2000. "Buckettree: Improving Collision Detection between Deformable Objects." *In Spring Conference in Computer Graphics SCCG 2000* , Bratislava, 156-163.

Govindaraju, N.K., Redon, S., Lin, M.C., and Manocha, D. 2003. "CULLIDE: Interactive Collision Detection Between Complex Models in Large Environments using Graphics Hardware." *Siggraph Eurographics Graphics Hardware 2003* (San Diego, CA, Jul. 26-27).

Gottschalk, S., Lin, M.C., and Manocha, D. 1996. "OBB Tree: A Hierarchical Structure for Rapid Interference Detection." *Proceeding of ACM Siggraph 1996* (New Orleans, Louisiana, Aug. 4-6), 171-180.

Grabner, G. and Kecskeméthy, A. 2003. "Reliable Multibody Collision Detection using Runge-kutta Integration Polynomials." *CD Proceedings of the International Conference on Advances in Computational Multibody Dynamics 2003*, (Lisbon, July 1-4).

He, T. 1999. "Fast Collision Detection Using QuOSPO Trees." *Proceedings of the 1999 symposium on Interactive 3D graphics*, (Atlanta, Georgia, Apr. 26-29) , ACM 1999 55-62.

Held, M., Klosowski, J.T., Mitchell, J.S.B. 1995. "Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs." *Proceedings Seventh Canadian Conference on Computational Geometry 1995*, 205–210.

Hubbard, P.M. 1995. "Collision Detection for Interactive Graphics Applications." *IEEE Transactions on Visualization and Computer Graphics 1995*, 1(3):218–230.

Hughes, M., Dimattia, C., Lin, M.C., Manocha, D., "Efficient and Accurate Interference Detection for Polynomial Deformation and Soft Object Animation." *Proceeding of Computer Animation 1996*, 155-166.

James, D.L. and Pai, D.K. 2004. "BD-tree: Output-Sensitive Collision Detection for Reduced Deformable Models." in *Proceedings of ACM SIGGRAPH 2004*, (Los Angeles, CA ,Aug 8-12).

Klosowski, J.T., Held, M., Mitchell, J., Sowizral, H., and Zikan, K. 1998. "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs." *IEEE Transactions on Visualization and Computer Graphics*, vol4 issue 1(Jan.) : 21-36.

Larsson, T. and Akenine-Möller, T. 2001. "Collision Detection for Continuously Deforming Bodies" *In Eurographics 2001*, Manchester, UK, 325-333.

Lin, M.C. 1994 *Efficient Algorithm Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley, March 1994.

Mirtich, B. 1998. "VClip: Fast and Robust Polyhedral Collision Detection." *ACM Transactions on Graphics 1998*, Vol. 17, no. 3(Jul.) : 177 – 208

Moore, M. and Wilhelms, J. 1988. "Collision Detection and Response for Computer Animation" *In proceedings Computer Graphics SIGGRAPH 1988* , 22(4):289-298.

Naylor, B., Amatodes, J.A., Thibault, W. 1990. "Merging BSP Trees Yields Polyhedral Set Operations." *In proceedings Computer Graphics SIGGRAPH 1990* ,24(4):115–124, 1990.

Otaduy, M.A. and Lin, M.C. 2003. "CLODs: Dual Hierarchies for Multiresolution Collision Detection." *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing 2003* (Aachen, Germany, Jul. 27-31), 94-101.

Quinlan, S. 1994. "Efficient Computation between Non-Convex Objects." *IEEE International Conference on Robotics and Automation 1994* (San Diego, CA, May 8-13).

Remion, Y., Nourrit, J., and Nocent, O. 2000. "Dynamic Animation of N-dimensional Deformable Objects." *WSCG proceedings- International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2000*, 147-154.

Teschner, M., Heidelberger, B., Müller, M., Pomeranets, D., and Gross, M. 2003. "Optimized Spatial Hashing for Collision Detection of Deformable Objects." *Vision, Modeling, and Visualization 2003*. (Munich, Germany, Nov. 19-21).

APPLYING MARKOV DECISION PROCESSES TO 2D REAL TIME GAMES

Thomas Hartley, Quasim Mehdi, Norman Gough
The Research Institute in Advanced Technologies (RIATec)
School of Computing and Information Technology
University Of Wolverhampton, UK, WV1 1EL
E-mail: T.Hartley2@wlv.ac.uk

KEYWORDS

Markov decision processes, value iteration, artificial intelligence (AI), AI in computer games.

ABSTRACT

This paper presents the outcomes of a research project into the field of artificial intelligence (AI) and computer game AI. The project considered the problem of applying AI techniques to computer games. Current commercial computer games tend to use complex scripts to control AI opponents. This can result in poor and predictable gameplay. The use of academic AI techniques is a possible solution to overcome these shortcomings. This paper describes the process of applying Markov decision processes (MDPs) using the value iteration algorithm to a 2D real time computer game. We also introduce a new stopping criterion for value iteration, which has been designed for use in computer games and we discuss results from experiments conducted on the MDPs AI engine. This paper also outlines conclusions about how successful MDPs are in relation to a real computer game AI engine and how useful they might be to computer games developers.

INTRODUCTION

Artificial Intelligence (AI) is required in some form or another for practically all commercially released computer games today. Current commercial AI is almost exclusively controlled by scripts or finite state machines (Cass, 2002; Nareyek, 2004; Spronck *et al.*, 2002). The use of these techniques can result in poor AI, which is predictable, less believable (Thurau *et al.*, 2003) and can be exploited by the human player (Spronck *et al.*, 2002). This reduces the enjoyment for the human player and makes them prefer human controlled opponents (Schaeffer, 2001 in Spronck *et al.*, 2003).

The game industry is however constantly involved in employing more sophisticated AI techniques for non player characters (NPCs) (Kellis, 2002), especially in light of the increase in personal PC power, which enables more time to be spent processing AI. Recent games, such as Black & White (Lionhead, 2001) use learning techniques to create unpredictable and unscripted actions. However most games still do rely on scripts and would benefit from an improvement in their AI. This makes computer games an ideal platform to experiment with academic AI; in fact the researcher John Laird states that interactive computer games are the killer application for human level AI (Laird and Lent, 2000).

These and other observations formed the basis of a research project into the field of AI and computer game AI. The

objectives of the project were the delivery of a computer game AI engine that demonstrated how an AI technique could be implemented as a game AI engine and a basic computer game that demonstrated the engine. The computer game was in the style of a researched computer game, which had been identified as needing improvement.

In our previous work Hartley *et al.* (2004) we presented our implementation of Markov decision processes (MDPs) and the value iteration (VI) algorithm in a computer game AI engine. In this paper we are presenting an implementation of the MDP AI engine we developed in a 2D Pac-man (Namco, 1980) style real time computer game. We are also going to answer a number of the questions raised in our previous work; namely can the AI engine we developed operate successfully in larger game environments and will the use of a less than optimal policy still produce a viable solution in larger environments.

PAC-MAN – A REAL TIME 2D COMPUTER GAME

The Pac-man computer game is a classic arcade game, which was developed by Namco in 1980. The premise of the game is simple; a player must guide Pac-man, which is a yellow ball with an eye and a mouth, around a maze, while eating dots and avoiding four ghosts, each of which has different levels of hunting ability (Hunter, 2000). The ghosts form the AI element of the Pac-man game. They start in a cage in the middle of the maze, they then have to escape from the cage and get to the Pac-man character. If Pac-man and a ghost touch, the player will lose a life. However there are four large dots in each corner of the maze. If Pac-man eats one of them, for a brief period he will be able to eat the ghost. If this happens the ghost's eye moves back to the cage in the centre and the ghost is reincarnated (Hunter, 2000). Figure 1 demonstrates a Pac-man game that is about to finish because Pac-man is about to be touched by one of the four ghosts and has no more lives left.



Figure 1: A screenshot of Microsoft Return of the Arcade Pac-man game.

The Pac-man computer game was identified as needing improvement primarily because the ghosts' movement in the game appeared predictable. This assertion is backed up by Luke and Spector (1996), who state, "...that Pac-man would be a good application for genetic algorithms to improve the

abilities of the ghosts.” However this is also contradicted by Bonet and Stauffer (2001) who state that the “...core gamer has no problem with a game like Pac-man, because even though Pac-man is a deterministic game that behaves exactly the same way every time you play it, it offers a huge amount of gameplay.” This statement indicates that even though the Pac-man game might have poor AI, it is not necessarily a problem because there are hundreds of levels and this type of AI fits well with this type of game. Also this type of AI is what the game player expects.

This is true to a certain extent; it is reasonable to say that predictability can be fun or not fun depending on how the game plays. But the Pac-man game was designed quite a few years ago for arcade machines (Hunter, 2000), so the Pac-man AI is quite simplistic and is orientated so that players will eventually die and put more money in the machine in order to continue. As a result of this it still would be useful to improve the Pac-man game AI and games like it for the home computer market, because the monetary aspect of arcade machines is not a factor any more and game players today expect more from a computer game’s AI. In addition the Pac-man game also offers an excellent real time environment in which to test the MDPs AI engine because the game environment can easily be divided into a finite set of states.

GAME DESCRIPTION

In order to experiment with and demonstrate how the Pac-man computer game could use and benefit from the AI engine, a simple Pac-man style action game, called Gem Raider was developed. Figure 2 contains a screenshots of the game in action. The purpose of the Gem Raider game was to demonstrate the features of the MDP AI engine we developed in a real time 2D Pac-man style environment. The Gem Raider game is not a direct clone of the Pac-man game, but general comparisons of the abilities of their AI’s can still be made.



Figure 2: A Screenshot of the Gem Raider game.

The Gem Raider game revolves around a Pac-man style character called Shingo. The user controls Shingo in order to collect gems, which are dotted about the game environment. Each game environment (or map) is a square area that contains environment properties such as water or obstacles, such as walls. The player also has to avoid 3 guards. The guards represent the Pac-man ghosts and their job is to pursue and kill Shingo before he collects all the gems in the level. In addition to the guards there are also enemies that take the form of sticks of dynamite and have to be avoided by both Shingo and the gem guards.

Every time a gem is collected the player scores some points. When all gems have been collected the player moves onto the next level. The game is over when all the human players lives (of which there are 3) are lost.

IMPLEMENTATION

In this section we present the implementation of the proposed computer game using the MDP AI engine. We also discuss some of the questions raised in our previous work; in particular can the AI engine operate successfully in larger game environments.

Markov Decision Processes

Markov decision processes (MDPs) are a mathematical framework for modelling sequential decision tasks / problems (Bonet, 2002) under uncertainty. According to Russell and Norvig, (1995), Kristensen (1996) and Pashenkova and Rish (1996) early work conducted on the subject was by R. Bellman (1957) and R. A. Howard (1960). The subject is discussed in detail in our previous work Hartley *et al.* (2004). Here we give a description of the framework in relation to the computer game we developed.

The technique works by splitting an environment into a set of states. An NPC moves from one state to another until a terminal state is reached (i.e. a goal or an enemy). All information about each state in the environment is fully accessible to the NPC. Each state transition is independent of the previous environment states or agent actions (Kaelbling and Littman, 1996). An NPC observes the current state of the environment and chooses an action. Nondeterministic effects of actions are described by the set of transition probabilities (Pashenkova and Rish, 1996). These transition probabilities or a transition model (Russell and Norvig, 1995) are a set of probabilities associated with the possible transitions between states after any given action (Russell and Norvig, 1995). For example the probability of moving in one direction could be 0.8, but there is a chance of moving right or left, each at a probability of 0.1. There is a reward value for each state (or cell) in the environment. This value gives an immediate reward for being in a specific state.

A policy is a complete mapping from states to actions (Russell and Norvig, 1995). A policy is like a plan, because it is generated ahead of time, but unlike a plan it’s not a sequence of actions the NPC must take, it is an action that an NPC can take in all states (Yousof, 2002). The goal of MDPs is to find an optimal policy, which maximises the expected utility of each state (Pashenkova and Rish, 1996). The utility is the value or usefulness of each state. Movement between states can be made by moving to the state with the maximum expected utility (MEU).

In order to determine an optimal policy, algorithms for learning to behave in MDP environments have to be used (Kaelbling and Littman, 1996). There are two algorithms that are most commonly used to determine an optimal policy, value iteration (Bellman, 1957) and policy iteration (Howard, 1960). However other algorithms have been developed, such as the Modified Policy Iteration (MPI) algorithm (Puterman and Shin, 1978) and the Combined

Value-Policy Iteration (CVPI) algorithm (Pashenkova and Rish, 1996).

In our previous work Hartley *et al.* (2004) we also implemented a variation on the VI algorithm, which was designed to reduce the execution time of the process and still produce a valid policy, which an NPC can use to navigate the environment. We called the new stopping criterion “Game Value Iteration” (GVI) and it works as follows: we simply wait for each state to have been affected by the home state at least once. This is achieved by checking if the number of states, with utilities that are equal to or less than 0 (zero) are the same after 2 successive iterations. All non-goal states have a reward (cost), which is slightly negative depending on their environment property (i.e. land, water etc.). Since utilities initially equal rewards, a state’s utility will be negative until it has been affected by the positive influence of the home (goal) state.

As a result the number of cells with negative utilities will decrease after each iteration. However some states may always retain a negative utility, because they have larger negative rewards due to their environment property and they may be surrounded by states with similar environment properties. Consequently when the number of states with negative utilities stays the same for 2 successive iterations we can say that the states are not optimal, but they should be good enough for a workable policy to exist, which the NPC can use to navigate the map. Before this point it is likely that no workable policy for the entire environment would exist. This stopping criterion assumes rewards can only be negative and there is a positive terminal state which is equal to 1. Also note that, checking if a state’s utility is greater than 0 is not required for the terminal states, because their utilities never change.

The AI Engine

In addition to developing the computer game we also experimented with increasing the size of the grids used by the AI engine, in order to assess how VI convergence and the GVI algorithm are affected. The AI engine can now represent game worlds at the size of 10x10, 20x20 and 40x40. Due to the processor intensive nature of MDP we decided to experiment with learning partial sections of the game world, which are around an NPC’s location. This should significantly reduce the number of iterations required to learn an instance of a game map, which is an important factor in real time games.

The Game

The game was developed in Microsoft Visual Basic specifically for this application. The relevant sections of the AI engine source code were imported into the game and a game engine was developed. The game runs in real time and it allows the user to swap between the VI and GVI algorithms and specify whether the algorithms should use the whole map or a partial section of the map.

The game world is split up into a grid of 20x20 cells. Each cell in the grid has a property associated with it, such as land, water or an obstacle. The human player can move freely

through the game world, but they cannot move through obstacles and water slows them down. The NPCs in the game can also move freely around the game world, but not through obstacles and only slowly through water. Each NPC in the game has a set of reward values, which it associates, with each cell property. The properties of the environment are used by each NPC (i.e. the AI engine) as the reward value for each cell. For example water means slower movement for the NPC, so by giving cells with the water property a more negative reward value it would mean that the reward for being in that cell is slightly less than cells with no water property. When the utility value of each cell is created the utility values of cells with the water property will be less than those with no water property. So when an NPC makes a choice of which cell to move to it will be less likely to move to the cell that has the water property. This means the NPCs in the game should produce more humanlike movement because they take into account environment variables, which could be considered as having a poor effect on their progress.

EXPERIMENTAL RESULTS

In the introduction we stated that we were going to answer a number of questions raised in our previous work and demonstrate MDPs being applied to a 2D Pac-man style computer game. The first phase of experiments involved increasing the size of the environments used by the AI engine, to 20x20 and 40x40 cells.

For all experiments in this phase of testing the following things were kept the same: there were two terminal states, +1 (goal) and -1 (enemy), and there was a slight cost of -0.00001 for all non-goal states. The probability of moving in the intended direction was set to 0.8. The maps used for these experiments attempt to represent a limited maze like world that you would expect to see in a Pac-man style game. However they were kept relatively simple so their size could be increased, while the map itself remained the same. We also experimented with simpler and more complex maps.

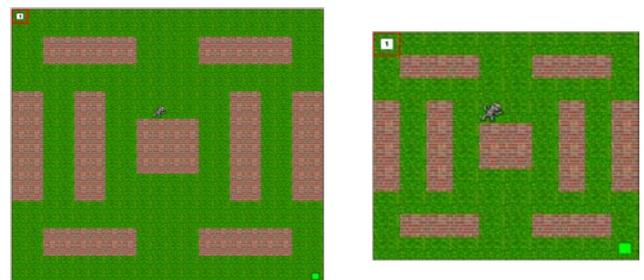


Figure 3: Screenshots of the 20x20 and 10x10 maps used to produce the results in table 1.

| Grid Size | No. of iterations to Convergence | | |
|-----------|----------------------------------|-----|----|
| | VI (OP) | GVI | HD |
| 10x10 | 63 | 18 | 0 |
| 20x20 | 83 | 38 | 0 |
| 40x40 | 130 | 79 | 0 |

Table 1: Results from experiments conducted on different size environments.

The HD column in table 1 stands for hamming distance between the GVI generated policy and the optimal policy. The optimal policy (OP) is the policy obtained by running the VI algorithm with the same initial data and maximum

precision (Pashenkova and Rish, 1996). The use of hamming to determine the difference between a policy and an optimal policy is based on that used in Pashenkova and Rish (1996).

From table 1 we can see that as the size of the grid increases the number of iterations required for convergence also increases, but it appears that the HD distance of the GVI algorithm is not affected by the increase in the grid size. This demonstrates that the GVI algorithm still produces viable policies, even in larger environments. Similar results were also found on different variations of maps.

The second phase of experiments looked at learning only partial sections of maps, around the NPCs location instead of the whole environment. It can be seen from the first experiments that as the size of the map increases the number of iterations required to converge the utility values also increases. In real time games this may prove processor intensive, especially if there is more than one NPC with their own set of rewards and utilities. We experimented with two different sized partial sections (5x5 and 7x7) around the NPCs location. Each cell within the partial area was treated normally, except if the home cell was not in the selection. If this was the case a new home cell was set.

In order to achieve this, for each accessible state within the region, the Euclidean metric between it and the goal was calculated. The state with the smallest value was set as the temporary home cell. If the Euclidean metric values were equal then a state was randomly selected. After the NPC moved one step the region and goal position were re-calculated and the new utilities generated. The map used for these experiments (figure 4) was a recreation of a map found in the Pac-man game.

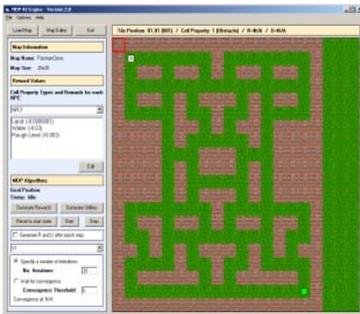


Figure 4: Screenshot of the map used to produce the results for the region experiments.

| Region Size | Average No. of iterations to Convergence | |
|-------------|--|-----|
| | VI (OP) | GVI |
| 5x5 | 43.1 | 6.8 |
| 7x7 | 48.1 | 9.7 |

Table 2: Results from experiments conducted on different region sizes around the NPCs location.

The results from the second phase of experiments were initially promising. Table 2 shows that the use of regions speeds up convergence quite considerably. However this seems to be at the expenses of NPC behaviour. After analysing the movement of the NPC it generally appeared to be less direct than when the whole map was taken into account and environment properties had very little affect on its behaviour. This decrease in movement quality is extremely undesirable because one of the objectives of this

research is to produce better quality, more humanlike behaviour in computer game NPCs. We also found that in complex maps the technique tends to need more information than that provided by the region area to solve which state should be selected as the temporary home state. For example a U-shape area in an environment may trap an NPC if it cannot view states outside the area. We overcame this problem to a certain extent, by only allowing the use of Pac-man style maps which do not have dead ends and by recording the NPCs last position in order to prevent it from moving back on itself.

The third phase of experiments looked at how successful MDPs were when applied to a real time 2D computer game. When running the VI algorithm in the Gem Raider game we experimented with a relatively conservative threshold of 0.001 and 0 (zero). We varied the rewards for each gem guard in order to achieve different types of behaviour.

The experiments conducted on the Gem Raider game were focused on confirming that the AI engine would work in a game environment similar to Pac-man and produce intelligent behaviour. It is however difficult to determine whether the Gem Raider game specifically offers an improvement over the Pac-man game AI for a number of reasons, in particular as discussed above, because the Gem Raider game is not an exact clone of Pac-man and the quality of the Pac-man AI is a subjective issue. From our observations the NPCs in the Gem Raider game appeared to produce intelligent behaviour, which reacted to the dynamic, constantly changing game environments. Some examples of this behaviour include the gem guards appearing to take into account and react to environment properties. In our opinions the behaviour produced by this game appeared to be more intelligent than the NPC behaviour produced by the Pac-man game. However further work, such as a survey, would be necessary to confirm this assertion.

In terms of performance the GVI algorithm performed much better than the VI algorithm. When running the VI algorithm there was a noticeable slow down while the utilities were being calculated. To a certain extent this problem could be overcome by implementing the technique in a more optimised programming language such as C++, however it highlights the problem that this approach is processor intensive.

DISCUSSION

The results presented in this work show that on different size grids the GVI algorithm can be used to produce intelligent NPC behaviour in less iterations that the VI algorithm. After observing the movement produced by the AI engine we found that when only partial sections of an environment were used NPC movement was less effective and took less account of the environment properties. This is an undesirable consequence as it results in the NPC appearing less intelligent.

We have also shown how MDPs using the VI and GVI algorithm can be applied to 2D real time computer games, such as Pac-man. Applying this technique to these types of games has proved relatively successful. The GVI algorithm offers a more suited stopping criterion for computer games,

as it automatically converges to a usable policy in a very small number of iterations. The VI algorithm is less suited to this type of environment because it requires a fixed number of iterations or a convergence threshold, both of which can be problematic in dynamic and constantly changing environments.

When implementing these algorithms in a 2D real time computer game the key factor to consider is processing constraints. If each NPC within the game requires individual rewards and utilities, then the time it takes to generate the utilities could be inhibitive to the playing of the game. A solution to this problem would be to determine only a part of the game environment, however as we have demonstrated here this impacts greatly on the advantages of the technique. Other approaches such as fixed regions or a number of NPCs making use of the same set utilities may solve this problem. However as long as processing power increases the use of this technique will become a more interesting proposition for the development of unscripted NPC navigation in real time computer games.

CONCLUSIONS AND FUTURE WORK

In our previous work we examined how MDPs using the VI and GVI algorithms could be applied to an AI engine that was suitable for use in a 2D real time computer game. This paper has continued this work and shown that these algorithms can be relatively successfully applied to 2D style games. The results from the experiments conducted here are very promising and show that even though the GVI algorithm produces a less than optimal utilities it still is effective, even in large game environments. The results from the Gem Raider game experiments have shown that it is indeed feasible to apply MDPs using GVI to real time computers, such as Pac-man. However the technique is computationally expensive and as a result may prove unfeasible for some computer games.

Further work in this area will involve exploring other algorithms such as policy iteration, modified policy iteration and reinforcement learning (Sutton and Barto, 2000) in relation to computer games and comparing them to the GVI algorithm. Other interesting areas to investigate include applying the technique to other types of games and applying the technique to other types of problems, for example state machines (Sutton and Barto, 2000). The work conducted here could benefit the application of MDPs to other types of problems, for example state machines in 3D shoot-em ups and real time strategy games (Tozour, 2002). However applying these techniques to NPC navigation in these types of games may only offer limited use.

ACKNOWLEDGMENTS

The approach used to create map graphics in the Gem Raider game was inspired by Nick Meuir's Simple Map Editor. URL: <http://www.Planet-Source-Code.com/vb/scripts/ShowCode.asp?txtCodeId=46635&lngWId=1>.

REFERENCES

- Bellman R. (1957) *Dynamic Programming*, Princeton University Press, Princeton, New Jersey.
- Bonet B. (2002) An e-Optimal Grid-Based Algorithm for Partially Observable Markov Decision Processes. in Proc. 19th Int. Conf. On Machine Learning. Sydney, Australia, 2002. Morgan Kaufmann. Pages 51-58.
- Bonet J. and Stauffer C. (2001) Learning to play Pac-man using incremental Reinforcement Learning. <<http://www.ai.mit.edu/people/stauffer/Projects/PacMan/>> (accessed 20 April 2003).
- Cass, S. (2002) Mind Games, IEEE Spectrum pp. 40-44, December 2002.
- Hartley, T., Mehdi, Q., Gough N. (2004) Using Value Iteration to Solve Sequential Decision Problems in Games. *CGAIDE 2004 5th International Conference on Computer Games: Artificial Intelligence, Design and Education* (eds. Quasim Medhi and Norman Gough).
- Howard R. A. (1960). *Dynamic Programming and Markov Processes*. Cambridge, MA: The MIT Press.
- Hunter W. (2000) The history of video games from 'pong' to 'pac-man', <<http://www.designboom.com/eng/education/pong.html>> (accessed 18 April 2003).
- Kaelbling L. and Littman, M. (1996) Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-285.
- Kellis E. (2002). An Evaluation of the Scientific Potential of Evolutionary Artificial Life God-Games: Considering an Example Model for Experiments and Justification. MSc. Thesis, University of Sussex.
- Kristensen A. (1996), Textbook notes of herd management: Dynamic programming and Markov decision processes <<http://www.prodstyr.ihh.kvl.dk/pdf/notat49.pdf>> (accessed 24 April 2003).
- Laird, J., van Lent, M. (2001) Human Level AI's Killer Application. *Interactive Computer Games. Artificial Intelligence Magazine*, 22(2), pp. 15-25.
- Lionhead Studios / Electronic Arts (2001) Black & White. <<http://www.eagames.com/>>.
- Luke, S. and Spector, L. (1996) Evolving Teamwork and Coordination with Genetic Programming. in 1st Annual Conference on Genetic Programming (GP-96). The MIT Press, Cambridge MA, pp. 150-156.
- Namco (1980), Pac-man. <<http://www.namco.co.uk/>>.
- Nareyek. A. (2004) AI in Computer Games. *ACM Queue* [Online]. 10 Feb 2004 [cited 20 Feb 2003], ACM Queue vol. 1, no. 10. Available from: <<http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=117>>.
- Pashenkova E. and Rish I. (1996) Value iteration and Policy iteration algorithms for Markov decision problem. <http://citeseer.nj.nec.com/cache/papers/cs/12181/ftp:zSzzSzfpt.ics.uc.edu:zSzpubzSzCSP-repositoryzSzpaperszSzmdp_report.pdf/value-iteration-and-policy.pdf> (accessed 23 April 2003).
- Puterman M. and Shin M. (1978) Modified policy iteration algorithms for discounted Markov decision processes. *Management Science*, 24:1127-1137.
- Russell S. and Norvig P. (1995). *Artificial Intelligence A modern Approach*, Prentice-Hall: New York.
- Spronck P., Sprinkhuizen-Kuyper I. and Postma E. (2002). Evolving Improved Opponent Intelligence. *GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation* (eds. Quasim Medhi, Norman Gough and Marc Cavazza), pp. 94-98.
- Spronck P., Sprinkhuizen-Kuyper I. and Postma E. (2003). Online Adaptation of Game Opponent AI in Simulation and in Practice. *GAME-ON 2003 4th International Conference on Intelligent Games and Simulation* (eds. Quasim Medhi, Norman Gough and Stephane Natkin), pp. 93-100.
- Sutton R. and Baro A. (2000) *Reinforcement Learning An Introduction*. London: The MIT Press.
- Thurau C. Bauckhage C. and Sagerer G. (2003). Combining Self Organising Maps and Multilayer Perceptrons to Learn Bot-Behaviour from a Commercial Game. *GAME-ON 2003 4th International Conference on Intelligent Games and Simulation* (eds. Quasim Medhi, Norman Gough and Stephane Natkin), pp. 119-123.
- Tozour P. (2002) The Evolution of Game AI in Steve Rabin (ed) *AI Game Programming Wisdom*, Charles River Media, pp. 3-15.
- Yousof S. (2002) MDP Presentation CS594 Automated Optimal Decision Making, <<http://www.cs.uic.edu/~piotr/cs594/Sohail.ppt>> (accessed 27 April 2003).

A COMBAT SIMULATION AID FOR DUNGEON AND DRAGONS

Matthew Bancroft, David Al-Dabass

School of Computing and Informatics
Nottingham Trent University
Nottingham NG1 4BU, UK.
david.al-dabass@ntu.ac.uk

KEYWORDS

Dungeon & Dragons, combat simulator.

ABSTRACT

This paper introduces the idea of using a computer application to aid in the combat sequence for a popular paper-based role-playing system called Dungeons & Dragons. The project aimed to reduce the complexity of the combat sequence and the workload of the Dungeon Master in control of the game. Following a survey of current systems and technologies, five objectives were formulated for the project. The first 3 objectives were successfully completed leaving the last two for future work.

INTRODUCTION

The paper focuses on the development of a Dungeons & Dragons system that aids the players whilst playing the paper-based game. It does this by looking at different ways in which to store the data, the graphical user interface, and a battlefield visualisation for combatants.

Dungeons and Dragons: The following is an extract from 3rdEdition.org, a website dedicated to the latest incarnation, describing what D&D is: "Dungeons and Dragons is a Fantasy Role-Playing Game wherein the participants take on the roles of characters that they create and define, and use those characters to create fantastic adventures in an imaginary world. Such elements may be as simple as the existence of magic, a force by which characters can make incredible things happen, or the existence of fantastic creatures like fairies, ogres, and of course, dragons." (Jason Ward 2000). User evaluation and informal feedback was used to test the system. Users commented on the good ideas used throughout the system. The GUI was seen as aesthetically good and the system was seen as a good foundation upon which to build.

LITERATURE SURVEY

There are little or no books specifically aimed at simulating a paper-based RPG (role-playing game) so our work was extended to include similar solutions in computer games, combat simulators, and computer graphics.

Why Dungeons & Dragons: D&D role-play system heavily revolves around combat that is complicated. Large-scale combat is even trickier as it involves the Dungeon Master controlling several entities. Providing a solution to these problems is one reason for the project. The following five objectives are listed along with their intended benefits.

- The first objective is to provide a way of managing individual characters. It needs to allow players to set up their characters which are to be used in the system. It removes the need for sheets of paper that note down the characters' abilities, which can be lost easily.

- The combat manager, objective two, aims to improve awareness of what is going on in combat by providing a central place to store combat information and guide the players through the combat sequence.

- The third objective is to provide a user-friendly system. The system is intended for use **in conjunction** with the paper-based system so it must be simple and easy to use so it does not impose itself on the game. It is here to speed things up and solve problems not to create new ones and slow it down.

- The fourth objective is to create a tool which can be used to manage groups of characters that will allow players to fight large-scale battles using the D&D system. The aim is to simplify this process and allow the players to utilise the computer's ability to calculate lots of numbers very quickly but NOT to produce a game.

- The final objective is to attribute artificial intelligence to the monsters. It might be more appropriate for DM's to set the monsters AI when controlling large numbers of them. The AI would act as a set of behavioural attributes that would guide the actions of each monster. This would remove some of the strain and complexity from the DM but leave the monsters firmly under their control.

Limitations of the Existing System: The existing system, with regards to this project, is D&D (3rd Edition). To understand the problems within this existing system first the scope of the problem must be narrowed down. The focus of this project is going to be guiding players through the combat sequence. Looking at the combat sequence highlighted several problems.

As the number of combatants grows the encounters become increasingly difficult for the DM to control. There are several reasons for this. Firstly, the DM has to control the actions of all the combatants that are not being controlled by players. This includes player's opponents, but also their allies. Not only does the DM have to control other characters but also the environment and special effects caused by spells. The second reason is misunderstanding, and misinterpretation. It is difficult for the DM to keep track of all the events that happen during combat and displaying this information clearly to the players. If players have a poor understanding of the combat they are in then they cannot make the right decisions. If they

misinterpret the situation they can slow the game play down, as things need to be explained again.

The Dungeon Master's Guide (Cook et al. 2000b, pg17) suggests that the DM have a pad of paper and to note down all the combatants in order to keep track of them. The DM also draws rough maps to show the players where everything is situated. The pad of paper notes the order in which combatants fight and any useful information the DM decides will aid in a smooth combat such as spells in effect and player character's (PC's) hit points. PC's actions can be modified by many things; such as cover or being invisible, remember this is a fantasy world, and the DM needs to keep all of these modifiers to hand when calculating results.

The PHB (Cook et al. 2000a, pg126 and 130) and DMG (Cook et al. 2000b, pg13 and 67) suggest the use of metal or card figures to represent the players on a board. The opponents are numerous and having a model to represent each one would be a tall order. Most players prefer to leave the figures to war gaming and stick to pen and paper.

Limitations of Existing Solutions: There are no commercially available solutions that are aimed at aiding the D&D system. There is a Character Generator, available from Wizards of the Coast, which guides the user through creating a character and updating it as they advance.

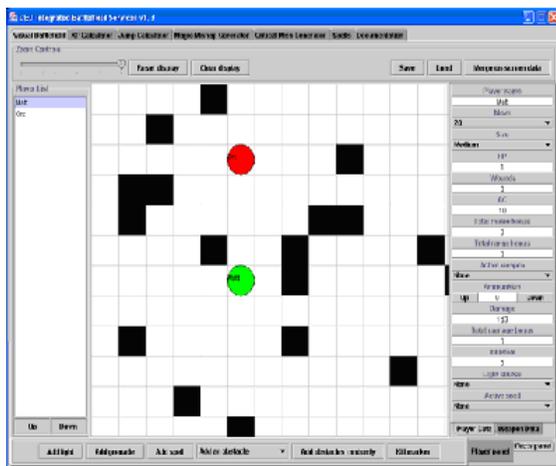


Figure 1 Battleboard's Main View

There are, however, several applications aimed at simulating combat that can be found on the Internet. A lot of time and effort was put into finding examples to include in the literature survey and most came from the same website. It was the only source that regularly popped up from several searches and references from other sites. Figure 1, above, shows an example of one of the combat visualisation aids (SB21@PACIFIC.NET.SG 2003). The website was <http://www.naparpg.us/downloads.htm> and is home to the Napa Valley RPG group. This website was the only source of any usable software aimed at aiding the combat sequence. Other websites that offered role-playing tools offered databases of information found in the three core rulebooks, name generators and many tools to help create fantasy worlds. The area of RPG tools has many applications and few had attempted a combat aid. The following examples are all

available from www.naparpg.org and range from simple working solutions, such as Combat Duel, to unusable failures.

Computer Games: Role-playing computer games have become a very popular genre and there are countless titles now available on most platforms, such as the PS2, X-Box and PC. More specifically there have been many based around D&D; Pool of Radiance, Baldur's Gate, Icewind Dale and Neverwinter Nights are just a few of the more recent titles. The Baldur's Gate series are the most popular D&D versions of this type of game. They provide the option of using the game's core rules making them very close to being a simulator.



Figure 2 Baldur's Gate 2 screenshot

More recently with the release of Neverwinter Nights comes a 3rd Edition game. Similar in idea to the Baldur's Gate series, which was using the 2nd Edition rules, it is much more advanced. The graphics have moved from a pseudo three-dimensional view to fully 3D and a full toolset is included in order for DM's to be able to add their own levels. The toolset is complicated and requires many hours to develop small adventures. E.g. try to imagine listing all the possible things a wise sage would say. It would be impossible to list them all. This is the DM's job; to play the non-player characters. RPG games are not the only source of ideas that can be used for this project. Part of this project is to design a battlefield visualisation and what better place to look than strategy games. These games have also been around since the beginning of gaming with games like Dune: The Battle for Arrakis.

Combat Simulators: These usually exist within the area of the military and are therefore difficult to research. There are several examples of computer games that fall under this category. Command & Conquer, Red Alert, Dune: The Battle for Arrakis, Battle Isle: The Andosia War and the Total War series are all combat based strategy games. The Total War series is the closest to a simulation and a slightly modified version has been used on television shows as just that, a medieval combat simulator.

DESIGN & DEVELOPMENT

The following section looks at the choice of programming language, design techniques and implementation.

Programming Language (PL): There are many programming languages available today of which Visual C++, Java, and Visual Basic 6.0 are the most common. There were two PL's chosen, one for the underlying simulation and one for the GUI. First of all let us look at the available PL's. The .net languages (c#, VB.net etc) were not considered as using them was not an option. – EDIT.

VC++ is a powerful object oriented (OO) language that is commonly used for simulation style projects. VC++ has several advantages over the other PL's. Familiarity with VC++ and visual studios is a large factor that can affect the project, and the visual studios environment provides this. VC++ allows the application to be developed as a console application, windows application (MFC's or standard Win API), or as a dynamic library link (DLL) that provides a good choice of GUI's. "The Microsoft Foundation Classes and Templates (MFC&T) provide tools to create small, lightweight controls, database applications, and full-featured, Windows desktop applications" (MSDN 2004).

Java is also a strong OO language, even more so than VC++, and also offers a powerful GUI through the use of Java Foundation Classes. "The Java Foundation Classes (JFC) are a set of Java class libraries provided as part of Java 2 Platform Standard Edition (J2SE) to support building graphics user interface (GUI) and graphics functionality for client applications that will run on popular platforms such as Microsoft Windows, Linux, and Mac OSX" (Sun Microsystems 2004).

Visual Basic 6.0 is the weakest of the three PL's with regards to OO. On the other hand it is the strongest at rapidly developing a GUI. VB also provides easy access to a simple database such as MS Access.

VC++ was chosen as the programming language based on several points. The familiarity and technical knowledge with both VC++ and MS Visual Studios is far and above any of the others, and this is likely to give the best solution in this situation. It also gave the project a good range of GUI API (application programming interface) to choose from within the developer's scope of experience and considered best for further development of the project with regards to 3D graphics being added. VC++ is the most common language for developing 3D graphics. Java or VB games are less common but are more generally aimed at mobile phone and other non PC based systems.

Graphical User Interface (GUI): The GUI is an important part of this project, as it needs to provide the user with quick and easy access to the program's functions. With this in mind the project needed to get a working GUI up and running quickly to allow users to evaluate its use. It was decided that a prototype, as described by Bowden (1992 cited Pressman, 2003 p.23), be designed in VB. VB is a strong choice for rapid application development (RAD).

After designing the GUI in Visual Basic 6.0 it was found that a dynamic library link (DLL) could be used to connect the VB GUI to the VC++ simulation code. The VB was developed as the front end to the application and would handle all high level graphics. The VC++ code handles all low level functions such

as moving a character or attacking another character. The VC++ code includes an export file that lists the functions that can be called from outside of it. The code also includes a layer between the two PL's, such as the conversion of data types. The VB needs to include a module that acts as function prototype for each of the exported functions. The DLL provides an easy layer between the two sets of code.

VB data types do not match C++ data types so it is necessary to convert the VB types into VC++ types before it can handle the call. In order to keep the VB as a dumb front end all of the data passed to the DLL is just to locate the right object in memory through VC++ code. For example, when a character is created the user enters the data in a VB form. VB does not test to see if it is the right data or if the character already exists, only that each field needed to create the object exists.

Design Methodology: The program was designed and subsequently developed using the spiral methodology (Sommerville 2001, p.53).

The typical waterfall model could have been used but the project is very large and the design of the complete system would take too much of the development time. Any problems could have left the project as just a design at the end of the course. The project was undertaken as part of a BSc Software Engineering course and it was decided that the spiral model would produce a useable (viewable) application at almost any stage of development even if not entirely finished. VB was used as RAD (Rapid Application Development) to develop a useable prototype of the front end to the system. This was developed alongside the core code. This allowed early testing of the GUI to provide an interface that players would find easy to use. The C++ and VB were then modified to communicate with each other, using a DLL for the communication layer. The application was designed with modularity in mind. The core classes can all be independently used in other systems and the DLL layer allows other graphics to be developed at a later date and bolted on (replacing the simple graphics developed for this project).

SOFTWARE DEVELOPMENT

This section follows the development of the character manager and combat simulation in VC++, followed by the prototyping of the GUI in VB. The chapter ends with a look at how the code was converted in to a DLL to use the VB as the actual GUI and the conversion of data between VB and VC++.

Core Classes: The core classes were developed as a starting point to the system.

Character: The character object is the main focus of the system. Essentially a character is a list of variables: height, weight, age, race, experience level, strength, dexterity, constitution, intelligence, wisdom, charisma are among the most basic attributes needed.

Monster: This class is a special type of character. They have all the same attributes as characters but they are always the same for a particular type of monster, e.g. Orcs always have the same ability scores. Certain attributes can and do vary,

such as hit points. A standard Orc wields a great axe and has four hit points (Cook et al. 2000c, pg 146).

Equipment: An equipment class would be very useful for holding all the different objects that a character can hold, such as a weapon, a piece of armour, a backpack.

Weapons & Armour: The weapons class includes attributes to create all types of weapons from the simple club to the composite long bow and, similarly, so does the armour class.

Army & Unit: The army class was written as a simple container of units and the unit class a simple container of characters or monsters.

Manager: The manager class handles all of the instances of object; it encapsulates the object containers and the associated functions.

Combat Simulation: At this stage of development the combat sequence is added to the code. This involved adding new functionality to the manager class to allow encounters and objects to be set up.

Battlefield: Before a character can fight a monster they need an arena to fight in. It was arbitrarily decided that the class be called battlefield, instead of dungeon or arena. The battlefield class is very much a container to begin with; holding a number of squares.

As can be seen from the following code it is simple to find a square within the battlefield; the function requires the x and y co-ordinates and uses a simple algorithm to convert them into a single position within the vector. The vectors direct access allows the square to be accessed and the nature of vectors makes the look up constant, which means no matter how big the battlefield is, it will always take the same amount of time to find a specific square.

```
1 bool Battlefield::isOccupied( int xPos, int yPos )
2 {
3     int pos = 0;
4     pos = xPos + ( yPos * width );
5     return gridSquares[pos].isOccupied();
6 }
```

The only other function needed by the battlefield is a query to see if it is occupied; the function will ask a specific square if it is occupied. The battlefield has to ask the square because it is a square's attribute.

Additional Functionality: Now that the characters have a battlefield in which to fight new functions can be added to the other objects to allow combat to happen. E.g. a character needs to be able to perform actions such as attack, move or draw a weapon. The standard attack function was looked at first.

Data Storage: It was decided early on that data was to be stored in simple text files. This allows initial development to be spent on core aspects of the system. The data could be moved to a database at a later date to provide better storage but early on in development there are no extra advantages.

The GUI: The GUI needed to provide the user with an easy to use interface which could access all the screens they need to. An MDI (multiple document interface) was chosen to keep the tool together and give the user the flexibility to choose what they can see. The GUI was simple to design and provided a neat clean interface to the data. The system is controlled via standard GUI menus (Fig 3).



Figure 3 Menu Choices

Figure 4 is typical of the kind of form created for each object. The form shown displays all the attributes of a weapon. The drop down menu, currently highlighted, allows the user to select any of the weapons loaded into the system.

The create weapon form looks exactly the same, except for the inclusion of a create button. It is accessed by selecting | Weapon | Add New Weapon. It is further to the right on the menu because it is not accessed as often and is grouped on the Custom menu choice with other similar options. The blank space of figure 4 is included to aid layout and aesthetics. It could be used as an image place holder in future expansions.



Figure 4 GUI screenshot showing Weapon List

Battlefield Visualisation: This was developed in VB alongside the GUI with the main focus being a 2D grid. The easiest and simplest way to visualise the combat is to use a 2D grid. Each grid square represents five feet of battlefield. The battlefield can potentially be larger than the screen size, so a way of navigating the battlefield is needed. The battlefield, at maximum zoom, shows a 25 foot square area, a typical sized room. The other zoom levels are 50 square, 75 foot square and 100 foot square.

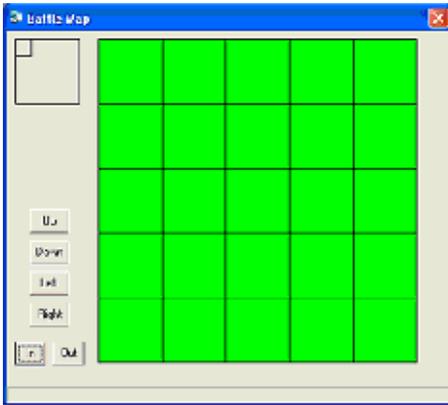


Figure 5 Battlefield Visualisation

DLL: Now that the simulation code and prototype have been completed to a suitable point we need to consider how the two sets of code were modified to use a DLL. The introduction of the DLL meant that the VB prototype could be used as the GUI for the VC++ simulation code. It was possible for the VC++ code to be recompiled as a DLL with few changes.

VC++ Communication Layer: The VC++ code needed several additions; an extern file, a communication layer and type conversion. The extern file lists all of the functions that can be used by other programs. These are the functions that the GUI is going to use. The communication layer provides the interface between the VB and VC++.

RESULTS & DISCUSSIONS

At this stage the programming has reached a level where it needs testing. The core functionality required for its use has been completed and before any new functions are added it is best to make sure what is written is sound.

Usability Test: The usability test asks the users to carry out a series of predefined tasks and then to mark on a likert scale how easy the task was to complete. The aim here was to test both the GUI and functionality of the program.

Limitations of the Design: There are several functions not included in the design: i) Advanced Combat Actions, ii) Feats, iii) Skills, iv) Magic, v) Monsters with special abilities, vi) 3D graphics.

CONCLUSIONS & FUTURE WORK

The modern D&D computer games provide a multiplayer aspect that is almost as involving as the paper based game but is lacking in variety. "An RPG with a human referee ... is a lot more capable of variety than any computer game" (Rollings & Morris 2004). So there will always be paper-based RPG's until such time that computer games have progressed to a stage where an almost endless number of possibilities can be written into any single game.

This project could be seen as a starting point to providing such a game. It is a hybrid of computer game and interactive role-play. The human referee supplies the plot and intrigue whilst the computer supplies the fantastical graphics and game play.

Artificially Intelligent Monsters: To some people role-play is all about rolling dice and bashing lots of monsters. To others it is about intrigue and in depth plots. It becomes very difficult to get involved in the story line when every time you play you end up spending 90% of your time rolling dice to see who wins the combat. One of the reasons its takes so long is that the DM controls so many aspects of the encounters, e.g. the DM controls the 5 Orc warriors, the Orc Shaman, the 5 Spell effects that the shaman has cast and the weather.

Future Work: The idea of AI is to set behavioural attributes of the combatants, e.g. each round all elves will attack the nearest orc. If the orc is adjacent to the elf then the elf will use a sword else a bow will be used. This idea of large scale combat also suggests the need for some sort of army management and this was the idea for objective four. Finally, a set of 3D graphics could be used to visualise the fantastical elements of the game.

References

- BIOWARE CORP, 2000. Baldur's Gate II [PC CD ROM]. Avalon Interactive.
- BIOWARE CORP, 2002. Neverwinter Nights [PC CD ROM]. Atari UK Ltd.
- BLACK ISLE, 2001. Icewind Dale [PC DVD ROM]. Avalon Interactive.
- BOWDEN, P., 2003. Software Quality, notes to accompany the lectures. [Online]. Available as Notes1.doc from: - <http://student.doc.ntu.ac.uk/modules/soft30051/files/Notes/Notes1.doc>
- CAULDREN, 2000. Battle Isle: the Andosia War [PC CD ROM]. Blue Byte Software Ltd.
- CREATIVE ASSEMBLY, 2002. Shogun Total War: Classic Range [PC CD ROM]. Electronic Arts.
- COOK et al., 2000a. Dungeons & Dragons Players Hand Book Core Rulebook I. Second Printing. Wizards of the Coast.
- COOK et al., 2000b. Dungeons & Dragons Dungeon Master's Guide Core Rulebook II. First Printing. Wizards of the Coast.
- COOK et al., 2000c. Dungeons & Dragons Monster Manual Core Rulebook III. First Printing. Wizards of the Coast.
- GEBHARDT, 2004.
- MSDN, 2004. Microsoft Foundation Class Library and Templates [Online] Available at: - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/html/mfc_about_the_microsoft_foundation_classes.asp
- NAPA ROLE-PLAYING GUILD, 2004. Napa Role-playing Guild – Downloads. [Online]. Available at <http://www.naparg.us/downloads.htm>
- ROLLINGS and MORRIS, 2004. Game Architecture & Design. New Riders Publishing.
- SB21@PACIFIC.NET.SG, 2003. D&D Battlefield Services (Battleboard) [Online]. Available as Battleboard.zip from: - <http://enworld.cyberstreet.com/news/modules.php?op=modload&name=Downloads&file=index&req=getit&lid=151>
- SOMERVILLE, I., 2001. Software Engineering. 6th Edition. Addison-Wesley.
- SSI, 2001. Pool of Radiance: Ruins of Myth Drannor [PC CD ROM]. Ubisoft.
- SUN MICROSYSTEMS, 2004. Java Foundation Classes (JFC/Swing) [Online]. Available at <http://java.sun.com/products/jfc/index.jsp>
- WARD, J., 2000. Dungeons & Dragons - 3rd edition - Articles - What Is Dungeons & Dragons? [Online]. Available at <http://www.3rdedition.org/articles/viewer.asp?ID=10>
- WESTWOOD, 1992. Dune 2: Battle for Arrakis [PC CD ROM/FLOPPY DISC]. Virgin Interactive.
- WESTWOOD, 2002. Command & Conquer Red Alert 2 Classic [PC CD ROM]. Electronic Arts.
- WIZARDS OF THE COAST, 2004. Dungeons & Dragons. [Online]. Available at http://www.wizards.com/dnd/DnDArchives_History.asp

Graphics Developments and Simulation

| | |
|---|-----------|
| Rhodes, D., Cant, R., Langensiepen, C. and Al-Dabass, D. Programmable GPUs and shading languages: past, present and future | 66 |
| Langensiepen, C., Foster, G. and Cant, R. GPU implementation of high quality bump mapping | 71 |
| Marshall, D., Delaney, D., McLoone, S. and Ward, T. Representing random terrain on resource limited devices | 76 |
| Meredith, M. and Maddock, S. Using a half-Jacobian for real-time inverse kinematics | 81 |
| Arokiasamy, A., Al-Dabass, D., and Periyamayagam, R. Simulating animation by regular deformation using OpenGL | 89 |

PROGRAMMABLE GPU'S AND SHADING LANGUAGES: PAST, PRESENT AND FUTURE

DANIEL RHODES, RICHARD CANT, CAROLINE.LANGENSIEPEN, DAVID AL-DABASS

*School of Computing & Technology
Nottingham Trent University
Nottingham NG1 4BU
richard.cant@ntu.ac.uk*

KEYWORDS

Cg, HLSL, GLSLang, RenderMonkey, FX Composer, DirectX, OpenGL, Vertex Shaders, Pixel Shaders, Shader Model 3, Programmable GPU.

ABSTRACT

We review the current state of play with regards to the latest tool in the graphics programmers arsenal; the programmable GPU. We discuss the various merits and pitfalls of each currently available high level shader language; designed to help programmers get the most out of these new GPU's with minimal effort. We also look into the other tools available to aid Shader development, along with the latest hardware features and the new Shader Model 3 standard.

INTRODUCTION

Over recent years development of computer graphics hardware for the home market has progressed at a rate well above that stated by Moore's law [Intel, 2004]. This is due largely to the push towards what NVIDIA term as "Cinematic Computing" [NVIDIA, 2003]; essentially what they mean is that real-time computer graphics are beginning to approach the type of effects only previously available with pre-rendered systems such as ray-tracing.

The current efforts for more realistic end products are leading to the requirement for more advanced techniques in computer graphics systems. This has led to many new developments within the hardware itself, these developments have come particularly rapidly over the last 5 years; we look at the reasons for this in the next section. This has led to more complicated and advanced techniques being possible and in some cases being commonly used and developed for consumer level hardware.

One such hardware development is the new generation of programmable GPU's (Graphics/ Graphical Processing Unit also known as the VPU or Visual Processing Unit). This programmability comes in the form of Pixel Shaders and Vertex Shaders. These are a relatively new development still well in their infancy; it has yet to be seen just how far they can

be pushed and what types of graphical algorithms best suit the new hardware structure they provide.

BREIF HISTORY OF SHADERS

The invention of Vertex and Pixel shaders was a direct result of a panel discussion about the future of graphics hardware as part of SIGGRAPH '99 [Dempski, 2002]. Previously, consumer level hardware was restricted to a fixed function based system. This meant the only real control the programmer had over the effects used within an application was by selecting the appropriate API calls to activate a hard-wired algorithm which is pre-programmed onto the hardware itself. This obviously severely restricted what effects were possible to whatever the card manufacturers deemed necessary and allowed very little freedom for the programmer.

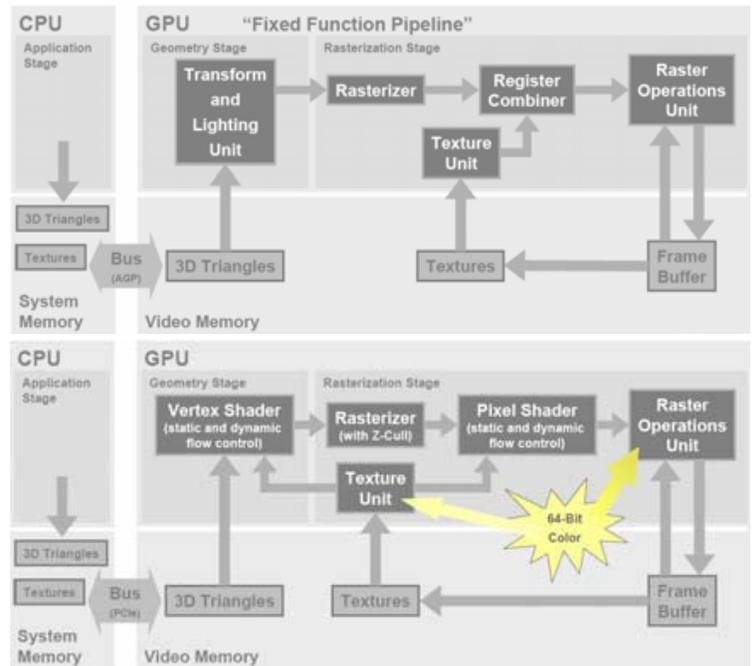


Fig. 1: Fixed Function Pipeline (top) Vs Programmable Pipeline (Bottom, shows current Shader Model 3 GPU architecture) [NVIDIA, 2004c]

Shaders present more flexibility to the programmer by allowing them to define vertex and/or pixel shader programs which can be run on any supporting hardware. For example they allow the programmer to replace the old style fixed function lighting with shader programs. So where previously the only options available may have been for example flat shading or Gouraud shading the programmer now has the option to create their own effects such as anisotropic lighting or cartoon rendering.

NVIDIA were the first to develop this new direction in graphics on consumer level hardware. This was in the form of the GeForce3 [NVIDIA, 2001]; which supported shader model 1.1 (pixel shader 1.1 and vertex shader 1.1). Shaders essentially replace the fixed function pipeline of a standard pre-GeForce3 graphics card by providing programmability as shown in Figure 1. Although fixed functions can still be used in conjunction with most, but not all shader generations. Shader model 3 for example cannot be used in conjunction with fixed function elements [Microsoft, 2004]. Figure 2 shows the progression between shader version 1.1 and 2.0 as illustrated by NVIDIA within the GeForce series. The latest incarnation: Shader Model 3 is covered in more detail later.

| | GeForce4 Ti | GeForce FX |
|-------------------------------|-------------|------------|
| Higher Order Surfaces | | |
| Geometry Displacement Mapping | - | ✓ |
| Vertex Shaders | 1.1 | 2.0+ |
| Max Instructions | 128 | 65536 |
| Max Static Instructions | 128 | 256 |
| Max Constants | 96 | 256 |
| Temporary Registers | 12 | 16 |
| Max Loops | 0 | 256 |
| Static Control flow | - | ✓ |
| Dynamic Control flow | - | ✓ |
| Pixel Shaders | 1.1 | 2.0+ |
| Texture Maps | 4 | 16 |
| Max Texture Instructions | 4 | 1024 |
| Max Color Instructions | 8 | 1024 |
| Max Temp Storage | - | 64 |
| Data Type | INT | FP |
| Data Precision | 32-bit | 128-bit |

Fig. 2: GeForce4 Vs GeForceFX [NVIDIA, 2002]

While this did provide a much greater level of control to the programmer than ever before possible on consumer level graphics hardware it did cause the problem of increased development times. This was due mainly to the fact that until recently Shaders had to be coded directly in Shader Assembly language. This limited the scope of possible applications due to the specialised and complex nature vertex and pixel shader programs and their assembly like language. This caused long development times and other complications inherent with assembly language programming in general. Considered with the faster than Moore's law progression of GPU technology [Hexus, 2002] this meant developers simply couldn't keep up

with the rapid pace of technology changes. This posed quite a problem, particularly when the complexities inherent with assembly language programming are compounded by the complexity of many graphical algorithms; yet these shaders were designed to encourage realism on consumer level hardware.

In an attempt to alleviate this problem NVIDIA, Microsoft and SGI (Silicon Graphics, Inc) have all developed their own High Level Shader Languages (HLSL's); Cg, MSHLSL, and GLSLang respectively. These HLSL's aim to have the same impact on Shader Assembly language that C and other high-level languages had in replacing assembler as the development tool of choice. HLSL's are designed to make it easier to map algorithms into code as they provide a much more logical way of viewing the operations of a shader.

SHADER MODEL 3

Shader model 3 is the latest standard which is unfortunately currently only supported by NVIDIA hardware (the NVIDIA GeForce 6800 series). The standard requires many benefits over previous incarnations including but not exclusive to more available instructions and greater floating point precision (as shown in figure 3); see [NVIDIA, 2004] and [Microsoft, 2004] for a more comprehensive look at Shader model 3.

| | DX 9.0 | Shader Model 3.0 |
|-----------------------------------|--------|--------------------------|
| Vertex Shader Model | 2.0 | 3.0 |
| Vertex Shader Instructions | 256 | 2 ¹⁶ (65,535) |
| Displacement Mapping | - | ✓ |
| Vertex Texture Fetch | - | ✓ |
| Geometry Instancing | - | ✓ |
| Dynamic Flow Control | - | ✓ |
| Pixel Shader Model | 2.0 | 3.0 |
| Required Shader Precision | fp24 | fp32 |
| Pixel Shader Instructions | 96 | 2 ¹⁶ (65,535) |
| Subroutines | - | ✓ |
| Loops & Branches | - | ✓ |
| Dynamic Flow Control | - | ✓ |

Fig. 3: Shader Model 2.0 Vs Shader Model 3.0 [NVIDIA, 2004]

Shader Model 3 is required by DirectX 9c and as such any card wishing to claim DirectX 9c compliance will have to support all the features defined in the Shader Model 3 standard. Previous versions of DirectX 9 only required Shader Model 2. NVIDIA, 2004 shows comparisons between Vertex Shader 2, Vertex Shader 2a and Vertex Shader 3 as well as providing comparisons between Pixel Shader 2, Pixel Shader 2a, Pixel Shader 2b and Pixel Shader 3 (note there is no Vertex Shader 2b BUT there is a Pixel Shader 2b).

CG, HLSL AND GLSLANG

Cg (C for Graphics) is NVIDIA's high level shader language and was developed in partnership with Microsoft; this partnership also spawned Microsoft HLSL (Although Microsoft simply like to call it HLSL we shall refer to it as MSHLSL to avoid confusion). The two languages are virtually identical; in fact Cg is actually a superset of MSHLSL [Case 2002]. The major difference between Cg and HLSL is the fact that the MSHLSL is DirectX specific whereas Cg will happily work with both DirectX and OpenGL. Both languages are based on C and incorporate some elements of C++, conversely due to the constraints inherent with shader programming a lot of the functionality of C has been lost.

As of OpenGL 1.5 there is a new contender for high level shader language of choice. GLSLang (OpenGL shader language) was included as part of OpenGL 1.5 in the form of official ARB extensions. Where previously it had only been available as unofficial extensions. GLSLang or OpenGL Shading Language is OpenGL's answer to MSHLSL, this is again based on ANSI C and again also steals elements from C++. See [Kessenich et al. 2002] for a detailed look at GLSLang.

Cg

```
...  
float3 cSpec = pow(max(0, dot(Nf, H)), phongExp).xxx;  
float3 cPlastic = Cd * (cAmbi + cDiff) + Cs * cSpec;
```

Assembly

```
...  
RSQR R0.x, R0.x;  
MULR R0.xyz, R0.xxxx, R4.xyzz;  
MOVR R5.xyz, - R0.xyzz;  
MOVR R3.xyz, - R3.xyzz;  
DP3R R3.x, R0.xyzz, R3.xyzz;  
SLTR R4.x, R3.x, {0.000000}.x;  
ADDR R3.x, {1.000000}.x, - R4.x;  
MULR R3.xyz, R3.xxxx, R5.xyzz;  
MULR R0.xyz, R0.xyzz, R4.xxxx;  
ADDR R0.xyz, R0.xyzz, R3.xyzz;  
DP3R R1.x, R0.xyzz, R1.xyzz;  
MAXR R1.x, {0.000000}.x, R1.x;  
LG2R R1.x, R1.x;  
MULR R1.x, {10.000000}.x, R1.x;  
EX2R R1.x, R1.x;  
MOVR R1.xyz, R1.xxxx;  
MULR R1.xyz, {0.900000, 0.800000, 1.000000}.xyzz,  
R1.xyzz;  
DP3R R0.x, R0.xyzz, R2.xyzz;  
MAXR R0.x, {0.000000}.x, R0.x;  
MOVR R0.xyz, R0.xxxx;  
ADDR R0.xyz, {0.100000, 0.100000, 0.100000}.xyzz,  
R0.xyzz;  
MULR R0.xyz, {1.000000, 0.800000,  
0.800000}.xyzz, R0.xyzz;  
ADDR R1.xyz, R0.xyzz, R1.xyzz;  
...
```

Fig. 4 Cg Vs Shader Assembler [NVIDIA, 2002]

Cg was the first high level shader language to become widely available to developers and was introduced to work with DirectX 8.1 and OpenGL 1.3, Cg is now almost at version 1.3 (Cg 1.3 is at the beta 2 stage at the time of writing). This latest

version is designed to incorporate the latest features provided by the new NVIDIA GeForce 6 series graphics cards.

Figure 4 shows an example of some Cg code taken from a Phong shader, as can be seen two lines of Cg code equates to twenty-five lines of shader assembly. This illustrates the huge difference in development and maintenance time that can be made by switching to a HLSL.

ALTERNATIVES

There are alternatives to shader assembly other than programming directly in HLSL's, for example ATI's RenderMonkey. This is essentially an IDE specifically for shaders, it provides a method to edit a shader and view the results within the same workspace.

RenderMonkey is designed to allow both artists and programmers to be able to create and tweak shaders relatively simply; Figure 5 shows an example screenshot taken from RenderMonkey.

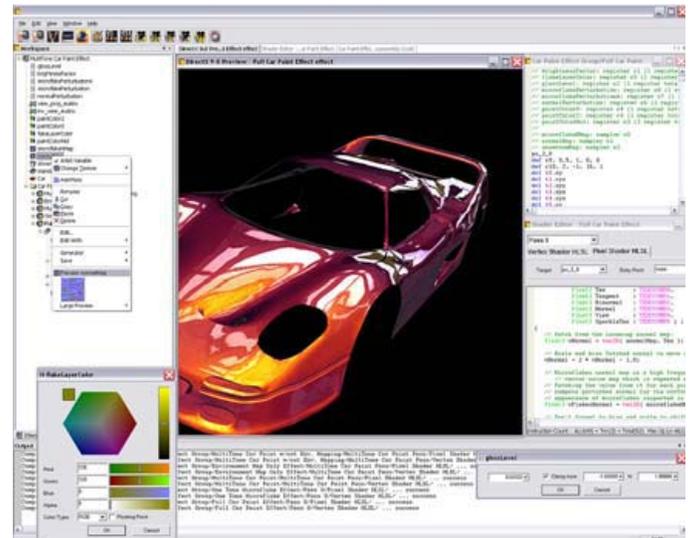


Fig. 5 RenderMonkey

While there are alternatives now available Render-Monkey deserves special mention as it was the first IDE of its type and was ATI's answer to Cg. Where Cg was still simply a shader language, RenderMonkey provided an IDE to encompass the whole shader development process and an interface onto currently existing API's.

RenderMonkey in its current incarnation supports both MSHLSL and GLSLang but doesn't claim Cg compatibility, no surprises there as ATI's and NVIDIA's rivalry is very well publicised. As such ATI are rather concerned about the possibility of Cg becoming a standard, as this would put a lot of power in the hands of its rival particularly as the current Cg compiler has been shown by [ATI, 2004] to perform poorly in comparison to MSHLSL on ATI based systems; when exactly the same code on an NVIDIA system is much more reliable. Not to be outdone NVIDIA now has an answer to

RenderMonkey in the form of the NVIDIA FX Composer which unfortunately is specific to MSHLSL development and perhaps surprisingly currently provides no support for Cg. See [NVIDIA, 2004b] for more details on the FX Composer

HLSL's ADVANTAGES AND DISADVANTAGES

All the HLSL's discussed like to claim a certain amount of platform independence. For example MS HLSL should have no problems running on either NVIDIA, ATI or anyone else's hardware (within MS Windows) and Cg will happily compile up to run on Windows or Linux based systems due to its compatibility with OpenGL and DirectX.

Surely all this so called 'platform independence' can only be a good thing? Shouldn't it simply do what it's claimed to and allow the same code to run on virtually any hardware you care to throw it and drastically reduce potential development times?

Well in reality it's not quite that simple; HLSL's hide a lot of the complexities of the shaders. Pixel shaders in particular carry a lot of restrictions that are not inherently obvious while coding in a HLSL; restrictions that could easily be picked up on while working in shader assembly language. For example Cg's loop statements allow the programmer to write statements which will be executed a constant number of times on the hardware. This can cause unforeseen results; as in the case of DirectX 8 Vertex Shaders (Vertex Shader 1.1) which don't allow looping. What actually happens is that when the code is compiled the Cg compiler unrolls the loop so the shader simply contains the same set of instructions repeated however many times the loop specified [Penfold, 2002]. This means a loop which contains say 2 instructions, repeated 10 times in Cg suddenly becomes 20 instructions in the shader itself. In this manner it would be very easy to exceed the instruction limits (or worse) without the programmer realising at least until the code is compiled obviously wasting valuable development time.

Debugging is another interesting topic of discussion within the world of HLSL's at the moment. Microsoft provide debugging support for MSHLSL in the form of an add-on for Microsoft Visual Studio .NET. Both ATI's RenderMonkey and NVIDIA's FX Composer also provide forms of debug support for MSHLSL including for example a disassembler within RenderMonkey and a jump to error feature in FX Composer. Both also provide the advantage of being able to see the results of shader changes virtually instantly within the IDE unlike within Visual Studio.

Microsoft seems to be leading the way with debugging with NVIDIA lagging far behind by providing no such tools for Cg. Not even NVIDIA's own FX Composer supports Cg, perhaps indicating a switch of focus away from Cg?

But all that said if you're careful HLSL's and take advantage of the debugging tools available (where possible) they do make the whole process of developing shaders much easier and quicker. HLSL's are the future of shader development and as

intended have begun to replace shader assembly as development language of choice in a similar way to how C and other high-level languages replaced assembly language as the most common way to write computer programs.

THE HARDWARE

The current batch of hardware available on the consumer market has at the top of the range the NVIDIA GeForce 6800 and the ATI Radeon X800, these two cards take decidedly different approaches in attempting to top the sales chart.

NVIDIA with the 6800 have chosen to focus on providing more functionality and as such have opted to include Shader Model 3 Support on the 6800. ATI however have decided to stick with their roots and focus on image quality and pure power, at the expense of Shader Model 3 and DirectX 9c support.

In short in the current market if you want you take advantage of Shader Model 3 you have to design for NVIDIA hardware. But at the very least you should provide Shader Model 2 support within your application to allow operation on other manufacturers hardware. Unfortunately this will require either producing different versions of the application for each shader model you wish to support; or limiting your functionality to that of an older shader model (e.g. 1.1), thus enabling your application to operate on older hardware as well.

Although some good news is that Shader models are by design backwards compatible with previous versions meaning pixel shader 1.1 code should happily run on the latest Shader Model 3 hardware. Also the API's try to make the task of working with several different shader versions within an application easier, this is done by allowing the programmer to check what is and is not supported within the hardware before deciding which shader would be most appropriate to run.

THE FUTURE

The greater than Moore's law progression of graphics hardware looks likely to continue. Intel's PCI Express architecture looks set to take off in a big way. In the case of graphics the 16x PCI Express slot provides twice the downstream bandwidth of the current standard AGP 8x (4Gb/s compared to AGP's 2Gb/s) and allows simultaneous up and downstream communication at a total of 8Mb/s (4Mb/s in each direction). This should allow the likes of NVIDIA and ATI to continue to develop their cards at a high pace without fear of hitting a bottleneck there (although this may not necessarily make any difference to overall speed as currently all games are CPU limited anyway [ATI, 2004]).

The software side of things also looks promising as Shader Model 3 continues to gather support; for example ATI having chosen not to provide Shader Model 3 support with the X800 no doubt already have plans to support this in their next generation of cards and NVIDIA should continue to be strong supporters.

The next steps for the APIs will be DirectX 10 and OpenGL 2.0, the latter of the two having surfaced in early September 2004. DirectX 10 on the other hand could be years away as all the major card manufacturers seem in agreement that DirectX 9 will be able to support all the foreseeable next generation graphics features.

REFERENCES

ATI, 2004. **D3D Tutorial Optimisations** [online]. CANADA: ATI. Available at: <URL:ftp://www.ati.com/developer/gdc/D3DTutorial_Optimisations.pdf>

Dempski, K. 2002. **Real-time rendering tips and techniques in DirectX**. 1st ed. USA: Premier Press.

Case, L. 2002. **Making the Right Graphic Choice** [online]. USA: ExtremeTech. Available at: <URL:ftp://www.extremetech.com/article2/0,1558,727608,00.asp>

Kessenich, J. Baldwin, D. Rost, R. **THE OPENGL® SHADING LANGUAGE** [online]. USA: SGI. Available at: <URL:ftp://oss.sgi.com/projects/ogl-sample/registry/ARB/GLSLangSpec.Full.1.10.59.pdf>

HEXUS 2002. **Cg** [online]. HEXUS.NET. Available at: <URL:ftp://www.hexus.net/content/reviews/review.php?dXJsX3Jldmld19JRD0zNzkmdXJsX3BhZ2U9MQ==>.

Intel ®, 2003. **Moore's law** [online]. Santa Clara, USA: Intel ®. Available at: <http://www.intel.com/research/silicon/mooreslaw.htm>

Microsoft ® Corporation, 2004. **Shader Model 3** [online]. USA: Microsoft ® Corporation. Available at: <URL:ftp://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/ProgrammingGuide/ProgrammablePipeline/HLSL/ShaderModel3/ShaderModel3.asp>

NVIDIA® Corporation, 2001. **THE INFINITE EFFECTS GPUs** [online]. Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:ftp://www.nvidia.co.uk/docs/lo/1050/SUPP/gf3ti_overview.pdf>.

NVIDIA® Corporation, 2002. **Getting Started with Cg** [online]. Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:ftp://developer.nvidia.com/docs/IO/3974/ATT/GettingStarted.pdf> [Accessed 23rd March 2003].

NVIDIA® Corporation, 2003. **GeForceFX** [online]. Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:ftp://www.nvidia.co.uk/view.asp?PAGE=geforcefx>

NVIDIA® Corporation, 2004. **Shader Model 3 Unleashed** [online]. Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:ftp://download.nvidia.com/developer/presentations/2004/SIGGRAPH/Shader_Model_3_Unleashed.pdf>

NVIDIA® Corporation, 2004b. **FX Composer** [online]. Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:ftp://developer.nvidia.com/object/fx_composer_home.html>

NVIDIA® Corporation, 2004c. **Programming Graphics Hardware** [online]. Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:ftp://download.nvidia.com/developer/presentations/2004/Eurographics/EG_04_IntroductionToGPU.pdf>

Penfold, D. 2002. **HLSL's, Cg and the RenderMonkey** [online]. Westlake Village, USA: Tom's Hardware Guide. Available at: <URL:ftp://graphics.tomshardware.com/graphic/20021004/>

GPU IMPLEMENTATION OF HIGH QUALITY BUMP MAPPING

Caroline Langensiepen, Gareth Foster and Richard Cant
School of Computing & Informatics
Nottingham Trent University
Nottingham NG1 4BU, United Kingdom
richard.cant@ntu.ac.uk

KEYWORDS

Bump mapping, Cg, Hardware Accelerated Graphics.

ABSTRACT

This work addresses the removal of aliasing issues in bump and environment mapping via the use of multiple vectors in MIP mapping using current 3D graphics acceleration hardware.

Superficially the current generation of 3D graphics APIs are incompatible with the use of multiple vectors in MIP maps. Two approaches were undertaken in an attempt to try to overcome this problem. The first was based around the manipulation of texture coordinates to fix the multiple vector paradigm. It proved unsuccessful due to accuracy limitations, but future graphics hardware may well be able to employ it.

The second approach was based around the use of multi-texturing and limited to the use of two simultaneous vectors. Its implementation was a success, and resulted in a hardware rendered image where there was a noticeable difference between the single and double vector versions. Memory requirements of bump mapping with multiple vectors have been analysed, and found not to be taxing.

We therefore conclude that though the use of multiple vectors in MIP mapping on hardware accelerated platforms is currently somewhat awkward, the technique definitely has a future given the rapid and ongoing development of hardware programmability.

INTRODUCTION

In the mid 1990's the cutting edge of graphics was defined by fixed pipeline hardware accelerated graphics, brought to the mass market by 3DFX. But 3DFX are long gone, and we have come full circle. We are told that we are witnessing "the dawn of cinematic computing" by NVIDIA. As before the rise of 3DFX, the emphasis is again on programmability, rather than the use of a fixed 3D pipe-line. This programmability is the crux of the new revolution, and perhaps the phrase "the dawn of programmable graphics" would have been more apt than "the dawn of cinematic computing".

On the back of the new wave of programmability, the implementation of advanced custom lighting algorithms becomes feasible. Indeed, the web sites of ATI and NVIDIA are littered with example applications ready to prove this fact to the unbeliever.

In this paper an attempt is made to assess the degree to which this rediscovered programmability has matured, by implementing bump mapping with multiple vectors on current 3D acceleration hardware.

THEORY

Bump mapping, like many other techniques in audio and visual processing suffers from artefacts caused by aliasing when implemented, and MIP mapping is the technique most often applied to remove this problem. Details of MIP mapping theory are available from many sources, Watt (Watt 2000) provides a good background.

Texture Aliasing

In both texture mapping, and bump mapping, an array of texture values (texels) is mapped to the pixels within a polygon by way of a texture co-ordinate placed at each vertex. In texture mapping these values are linearly related to the actual shades that will be displayed. Aliasing problems arise due to the fact that the pixel size changes relative to the texels as the viewpoint in the scene is altered. For distant polygons there may be many texels within each pixel.

MIP mapping solves this problem by creating a series of copies of the top level array of values, each a quarter of the size of the last. The values in each successive layer are obtained by filtering the layer above. When a polygon is subsequently rendered, it will be possible to select a level such that the texels in the filtered MIP map match the pixel footprint.

Bump Mapping

In Bump mapping the texel values are actually normal vectors that will be used to compute the shades via a reflection calculation. Although the polygon is scan

converted as if it was flat, the use of normal vectors derived from a (fictional) undulating surface provides a convincing shaded effect as if the surface was in fact rough. Thus if Phong type specular reflection is being modelled with a geometric factor $(\mathbf{n} \cdot \mathbf{h})^n$ the normal vector \mathbf{n} will have been derived from the texture map. (In this expression \mathbf{h} is the so called “halfway vector” – representing the average of the incident and viewing directions whilst the index “ n ” is an arbitrary number (usually an integer) controlling the shininess of the surface.

An essential feature of this effect is that the transformation from vectors to shades, whether via a Phong type calculation or environment mapping, is non-linear. It follows that filtering the vectors a priori, as in MIP mapping, will not give the correct result since it will not commute with the non-linear lighting calculation. There is no simple route around this problem and in practice one has to be satisfied with an approximate solution. Several authors have attempted such an approach for example: (Fournier 1992, Schilling 1997, 2001, Cant and Langensiepen in preparation). In most of these methods the usual MIP map structure is retained with modifications. The two key changes are:

1. A modification to the reflection calculation – determined by the distribution of N vectors \mathbf{x}_i within the filtered texel. This distribution may be parametrised by the “spread” Q where:

$$Q = \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{x}_i \cdot \hat{\mathbf{m}}$$

$$\hat{\mathbf{m}} = \left(\sum_{i=0}^{N-1} \mathbf{x}_i \right) / \left(\left\| \sum_{i=0}^{N-1} \mathbf{x}_i \right\| \right)$$

In the references mentioned above the spread is taken into consideration by modifying the shading calculation. In both (Fournier 1992 and Cant and Langensiepen, in preparation) one approach is to modify the index “ n ” in the specular reflection calculation quoted above. Alternatively if MIP mapped environment mapping is used then Q would determine the MIP map level. Please see the references quoted for full technical details.

2. The use of multiple vectors to account for the situation where the vectors within the filtered texel are distributed such that the average vector $\hat{\mathbf{m}}$ is not representative of any actual vector. In this case :

$$\hat{\mathbf{m}} \cdot \mathbf{x}_i \approx 0 \quad \forall i$$

making it inappropriate to use $\hat{\mathbf{m}}$ in the reflection calculations. To overcome this the vectors need to be placed in groups G_0, G_1, \dots, G_{M-1} where $M \ll N$ (for efficient rendering), the j th group contains N_j vectors and

$$\hat{\mathbf{m}}_j = \left(\sum_{\mathbf{x}_i \in G_j} \mathbf{x}_i \right) / \left(\left\| \sum_{\mathbf{x}_i \in G_j} \mathbf{x}_i \right\| \right)$$

$$Q_j = \frac{1}{N_j} \sum_{\mathbf{x}_i \in G_j} \mathbf{x}_i \cdot \hat{\mathbf{m}}_j$$

The overall lighting calculation will now be:

$$L_{total} = \sum_{j=0}^{M-1} c_j L(\hat{\mathbf{m}}_j, Q_j)$$

Where L represents whatever lighting calculation is used and the c_j are weights determined by the number of vectors in each group. Details of how these mechanisms can be set up are to be found in (Fournier 1992 and Cant and Langensiepen, in preparation).

The advent of Cg makes the modification to the calculation comparatively straightforward – however the storage of the extra vectors, and the extra information needed to control the modified calculation is not easy to arrange within the current architecture – as we shall see later.

In the remainder of the paper we will concentrate on the problem of fitting these structures into the Cg architecture. We will use the multiple vector sets that we have generated in the course of our related research (Cant and Langensiepen in preparation) and in particular the super bump map (.SBM) file format used in that work. However the results here would be almost equally applicable to other implementations. In particular the methods used by Fournier (Fournier 1992) impose similar requirements. The SBM file format used in our software only system supports up to 2 vectors per texel at level 1 (level 0 being the original bump map), 4 at level 2, 8 at level 3 and so on. In practice we usually cap the number of vectors used as we have found that there is little advantage in using large numbers of vectors except for pathological bump maps. As reported below the hardware supported system is more restrictive and we have been limited to 2 vectors per texel so far.

IMPLEMENTATION

The platform chosen for development was an Intel® P4 with an NVIDIA GeForce FX 5900 3D accelerator running the Fedora Core 1 Linux operating system. Shaders were developed in NVIDIA's Cg (C for graphics) shader language while the core of the program made use of SDL (simple direct-media layer) and Open GL .

SDL and Open GL were chosen for their portability, a port to Apple® OSX® or Microsoft® Windows® should require minimal effort. Cg was the only realistic choice for cross platform shader development at the time of writing: that said, the glslang (Open GL shading language) is an integral part of the Open GL 1.5 standard and preliminary support was beginning to appear at the consumer level, see (Kessenich et al 2003) for more information. The NVIDIA hardware used was chosen since it is well supported on every major OS and works effectively with the Cg toolkit.

Figure 1 shows the simplified C/C++ signature of the Open GL MIP mapping interface with the important aspects of it highlighted in bold. It will no doubt have been noticed that this interface is provided for texture mapping, however, this same mechanism is used for bump mapping and any other situation where a shader requires a large volume of data to be passed to the GPU.

In the case of Direct X, the major competitor for Open GL, the interface is similar, but less effective for the purposes of highlighting the issue at hand.

```
void glTexImage2D( target, level,  
internalFormat, width, height, border, format,  
type, pixels )
```

Figure 1. C/C++ signature of the Open GL MIP mapping interface

Hawkins (Hawkins et al 2002) makes the following comment on this interface:

“When creating your MIP-map texture, you need to make sure you keep all the different sizes of the texture in powers of 2 from the largest size of the texture to a 1x1 texture map. For example, if the largest texture map is 64x64, then you also need to create sizes 32x32, 16x16, 8x8, 4x4, 2x2, and 1x1.”

It became immediately obvious from the interface signature shown, and comments associated with it in the relevant texts, that the interface was incompatible with the data that would need to pass through it for the purposes of this project.

Whilst the normal MIP map data size decreases by a factor of four at each level, (since each of the two dimensions is halved) the MIP map with multiple vectors only reduces by a factor of two because of the larger number of vectors present. The incompatibility is slight, but exists nevertheless, and presented a major problem.

The provisional solution was as follows. The bump map, with all its component MIP map levels, would be managed as a whole. MIP map level selection and lookup would then be done manually. Consequently the Open GL mechanism for the use of MIP mapping would have to be ignored, and the required functionality provided by making use of the shader languages.

The SBM data had to be modified in accordance with the limitations of the Open GL API. As the format and internalFormat parameter of the glTexImage2D function show (Figure 1), the data passed to the GPU must be homogeneous, furthermore, this data must be normalised, that is to say, in the range 0.0 to 1.0 inclusive. Additionally, the glTexImage2D interface only allows for RGB, RGBA and Luminance data to be passed to the GPU. The actual meaning of these terms is irrelevant, for the purposes of this project this simply meant that the only acceptable data would be an array of single values, or a three or four component vector. (Shreiner et al 2004) provides a complete list of the values accepted by the glTexImage2D function. It was decided that each bump map vector would be split into a three component vector (x, y, z), leaving the power component and the width component as separate entities. The basic premise here was to group common data types, signed byte, unsigned byte and float. By making use of Open GL multi-texturing, the program would then be able to pass four textures to the GPU, the fourth for the lookup table. It

was fortunate that only four were required, since even on a modern graphics card, four is the limit for simultaneous textures. It was also discovered, that by using NVIDIA extensions, the requirement for textures with power of two dimensions and for normalised texture data was removed.

The simple vertex and fragment shaders described by Kilgard (Kilgard 2003) were taken as a starting point since they had already been proven to work. After some investigation, it became clear that the vertex shader would require only a small modification at this stage, and that most of the extra functionality would need to be added to the fragment shader. The vertex shader had to be modified to pass the world space vertex position to the fragment shader as a texture coordinate. The world space vertex coordinate would be needed in order to perform standard lighting calculations with the normals in the bump map, see the book by Lengyel (Lengyel 2002) for lighting theory. This value had to be passed as a texture coordinate because that is how Cg manages parameters that are to be interpolated across a scan-line.

In the case of the fragment shader, the first requirement was that it be modified to accept all four of the textures that would be passed from the main program, making up the complete SBM normal. Other values required for lighting were passed directly to the fragment program from the main application since they did not require modification at the vertex stage of the pipe-line, these values included the camera or view position, and the light position.

When this test code was executed, some very strange visual results were produced. Some debugging was required in order to discover what was amiss. At this stage, the immense difficulty of debugging shader programs became apparent. There was no way to debug Cg code in the manner C++ code is debugged, it was not possible for example, to follow the execution of the Cg code, or to watch the values in variables. In shaders, it is not even possible to output values to the terminal window, or to a file for analysis. The only way to assess the value a variable contains was to write that value out to the fragment as the RGB colour value. Debugging the program in this manner was difficult and tedious. We hope that future implementations of Cg will provide better facilities and we understand that this is already true for alternatives such as Microsoft's HLSL.

Despite fixing some minor faults, the visual results were still not as expected, and it became clear that something was wrong on a fundamental level. In order to test this suspicion, the Cg code that calculated the offsets into the data tables was carefully evaluated. It was strongly suspected that the code as run in the fragment shader was not sufficiently accurate to produce the correct texture coordinates, although of course, this was difficult to prove. The 'blocky' nature of the resultant texture on the cube suggested that this might have been the problem.

An easy solution to this problem was available if Open GL's MIP mapping facilities could be used but this would require a different organisation of the SBM data. The new strategy was to use multiple standard MIP mapped textures.

The first texture would contain the first vector for all MIP map levels. The second texture would contain the second vector for all levels. Since the top level has only one vector it would be empty in this case. The third texture would then contain the third vector for all levels – and its top two levels would be empty. In principle this process can continue until all the vectors in the super bump map format are included. However this is not possible within Open GL since only four textures are allowed. In fact each SBM vector and its associated parameters (the c_j and Q or equivalent) would require two textures, limiting the number of vectors that could be included to two.

Consequently we decided to temporarily abandon the full SBM format since even the use of two vectors is a noticeable improvement over standard bump mapping. This also limited the system to Phong type lighting, because environment mapping requires additional textures.

At this stage in the development, there was a requirement for heavy reorganisation and reformatting of the vector data to be sent to the GPU.

```
for i to number of MIP map levels
    glTexImage2D(i, SBM[0][i][0])
for i to number of MIP map levels
    glTexImage2D(i, SBM[1][i][0])
```

Figure 2 Pseudo code showing how SBM data was used with the Open GL MIP mapping interface.

By setting the format parameter to `GL_FLOAT` in the call to `glTexImage2D`, the data could be passed directly to the GPU without Open GL making any effort to modify the values. Making the call in this way appeared to be the only way to pass the data without it being modified. This is the real reason for a floating point type being used in the 3D STL vector in the SBM class, obviously the input data had to match the format parameter used in the `glTexImage2D` call. In addition to this change, the call had to take account of the fact that the Open GL MIP mapping facilities were to be used.

Textures one and two were used for vector one, and textures three and four were used for vector two, this meant that columns one and two of Figure 5 were to be sent to the GPU. Figure 2 shows this in pseudo code, the diagram shows only the important aspects of the call, in an effort to illustrate how well the arrangement of SBM data suited the Open GL API.

After considerable efforts to try to make the shader perform MIPmapping, it was found that MIP mapping is already enabled, and simply making a texture lookup on a texture where MIP mapping has been used is sufficient. The fact that this was not mentioned in the NVIDIA documentation seemed astounding: the problem highlighted by this regrettable waste of resources was that it is hard when dealing with Cg to know where exactly the boundary is

drawn between a programmer's control, and the fixed portions of the graphics pipe-line.

RESULTS AND DISCUSSION

In terms of hardware, the memory requirements of the additional vectors did not appear to be unacceptable. The use of multiple vectors for bump maps amounts to an increase in memory use of around 50% over standard Mip mapping.

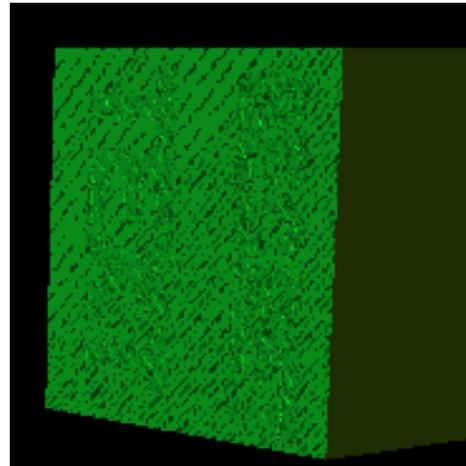


Figure 3 The 'secret text' super bump map displayed with one vector

With respect to execution speed, the frames per second reported was consistently above 100, reaching 300 at times. So it would appear that the use of an extra vector was not damaging to the application performance. That said, the application was not optimised, and was of course, built as more proof of concept than production system. Nevertheless, it is interesting to note that the frame rate is so high.

Figure 3 shows a screen-shot from the program. In this illustration, one normal is used to light each fragment. Figure 4 shows the same scene - however in this illustration, two normals have been used to perform the lighting calculation. The light source was positioned so as to be at 135 degrees to each edge connected to the lower left corner of the cube. This bump map was produced specially to include the words "secret text" in such a way as to ensure that both vectors are required for it to be displayed.. Therefore, the text is only visible when the the lighting is correct and both vectors are enabled. These screen-shots prove that the goals set were met, in that the GPU was successfully programmed to use MIP mapping with multiple vectors. In Figures 5 and 6 the full lighting effects (including specular highlights) can be seen.

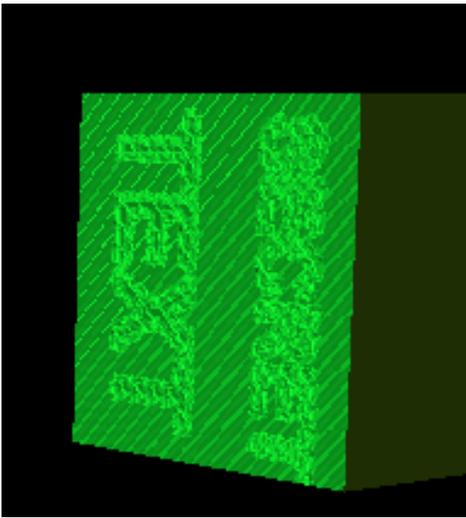


Figure 4 The 'secret text' super bump map displayed with two vectors

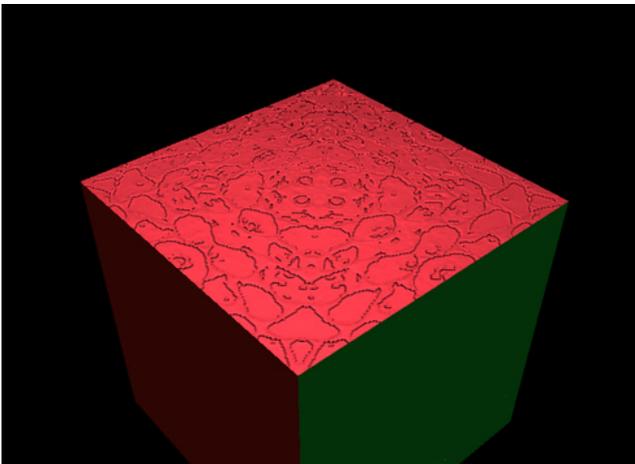


Figure 5 An image that shows the lighting effects in the program

CONCLUSIONS AND FURTHER WORK

We have shown that the programmability of high end consumer graphics accelerators, combined with the Cg language, has now reached the stage where it is possible to implement an algorithm which has been developed independently of the hardware. However there is considerable awkwardness in doing this and our implementation is of necessity quite restricted. We note that at least some of the of the problems relate to features of the hardware/software platform that could be changed without incurring prohibitive cost. We hope that these results can inform the future direction of hardware development to make such implementation easier.

Even so the performance of the system is quite impressive, indicating that the future of programmable graphics acceleration is bright. We anticipate repeating this exercise with future improved graphics cards to exploit the ever improving capabilities of hardware accelerated graphics.

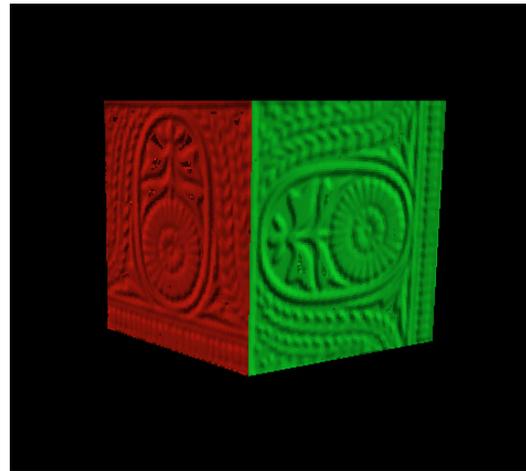


Figure 6 Alternative test image using 2 vectors

REFERENCES

- Cant, R.J., C.S Langensiepen (In preparation) Generating Multiple Surfaces using Competitive Learning. To be submitted to Computers and Graphics.
- Fournier, A. 1992. Filtering Normal Maps and Creating Multiple Surfaces, Department of Computer Science, University of British Columbia, Technical report TR-92-41.
- Hawkins, K., et al 2002, Open GL Game Programming. Portland, Premier Press.
- Kessenich, J. et al 2003 Open GL Shading Language. Sunnyvale, 3DLabs, Inc. Ltd.
- Kilgard, M., J. 2003 The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics. Boston, Addison Wesley.
- Lengyel, E. 2002 Mathematics for 3D Game Programming and Computer Graphics. Massachusetts, Charles River Media, Inc.
- Schilling, A.G. 1997. Towards real-time photorealistic rendering: challenges and solutions. Proceedings of the 1997 SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, Los Angeles, California, pp. 7-15, August 1997.
- Schilling, A.G. 2001. Antialiasing of Environment Maps. Computer Graphics Forum, 20(1), pp5-11.
- Shreiner, S., et al 2004 OpenGL(R) 1.4 Reference Manual (4th Edition). Boston, Addison Wesley.
- Watt,A.H. 2000. 3D Computer Graphics, Addison Wesley.

REPRESENTING RANDOM TERRAIN ON RESOURCE LIMITED DEVICES

DAMIEN MARSHALL[‡], DECLAN DELANEY[‡], SEAMUS MCLOONE*, TOMAS WARD*

[‡]Department of Computer Science,
NUI Maynooth
Maynooth, Co. Kildare
E-mail: damienm@cs.may.ie

* Department of Electronic Engineering
NUI Maynooth,
Maynooth, Co. Kildare
E-mail: tomas.ward@eeng.may.ie

KEYWORDS

Random Terrain, Perlin Noise, Mobile Devices, Game Boy Advance

ABSTRACT

Random terrain generation is the procedural creation of a set of data that represents closely a believable landscape. Common techniques of achieving this include the use of fractals and noise. Such techniques usually require a large volume of memory, as the geometry of the terrain needs to be calculated and stored at run time. Given the limited memory available on mobile devices, such as mobile telephones, the storage of the data required to represent massive terrains can be difficult. In this paper, we propose a novel method of storing terrain data on devices with limited memory. This method involves placing pre-computed blocks of terrain, known as terrain tiles, together in a pseudo-random manner as governed by a noise function known as Perlin noise. This allows large amounts of terrain data to be represented while still giving the appearance of a randomly generated terrain. Traditionally, Perlin noise is used in the procedural generation of textures and the modeling of naturally occurring phenomena. Using Perlin noise, only a subset of the overall data generated by the function needs to be stored at any one time. The approach outlined in this paper associates a tile of terrain with each value generated by the Perlin Noise function, meaning that only a subsection of the total terrain is stored in memory at any one time. We show how this process can be executed in real time on a resource limited device known as the Game Boy Advance, and also illustrate a significant reduction in the memory requirements of terrain storage when compared with traditional methods.

INTRODUCTION

A random terrain is a group of procedurally calculated values that represents closely a believable landscape. They are utilised in all areas of computer science, especially computer graphics and games (Pickover 1995), where they allow for the creation of game content with minimal time overhead. Also, as terrain maps can be quite large, techniques of generating random terrain, such as fractals (Dudgeon and Gopalakrishnan 1996), and models of naturally occurring phenomena (Kelley et al. 1988), help to minimise fixed storage requirements. However, this means that such methods usually require a voluminous amount of Random Access Memory (RAM).

With recent advances in processing power, computer games are becoming more popular on handheld devices (Ritter et

al. 2003). However, there are still limitations to the amount of available RAM for such devices, so that the creation of randomly generated terrain for computer games on such devices can be problematic.

In this paper, a novel method based on the use of terrain tiles and a pseudo random noise known as Perlin Noise is described (Perlin 1985; Rabinovich and Gotsman 1997). Traditionally, Perlin noise is used in the generation of procedural textures and the modeling of naturally occurring phenomena such as clouds and smoke (Ye and Lewis 1999; Holtkämper 2003) – see Figure 7. Here, we propose an approach that associates a tile of terrain with each Perlin noise value. A terrain tile describes a square block of terrain data. Using Perlin noise, only a subset of the overall data generated by the function needs to be stored at any one time. Therefore, large terrain maps can be represented in real time with a small memory footprint, as only the terrain tiles of interest to the user are stored at any given time.

The proposed method is implemented on a limited device known as a Game Boy Advance (www.nintendo.com), and results are presented for the implementation. We show that significant saving can be made in terms of memory space required when compared to traditional methods of storing and representing terrain. As this method makes no use of floating point arithmetic it is very efficient, making it suitable for use on a wide variety of devices where no dedicated floating-point unit is available. It would be particularly suitable for mobile telephones, as it allows for the generation of massive terrain maps with a minimal over the air download. This is of significant importance as large downloads have been identified as a limiting factor in the next generation of three-dimensional games on mobile telephones (EDGE Magazine 2004).

The rest of the paper is laid out as follows. The next section introduces Perlin Noise and outlines the suitability of the Game Boy Advance as a testbed. The Implementation section details the realisation of the proposed method, followed by a selection of performance values in the Results section. Finally, the paper concludes with suggestions for future work.

BACKGROUND

Perlin Noise

Perlin noise is one of the more important noise functions. Created by Ken Perlin (Perlin 1985), this noise has the special property of appearing random to the perceiver, yet remaining

entirely controllable. It works by defining a number of key points at set distances apart, and defining the values of the intermediate points procedurally using interpolation. Any subsection of the data created by the Perlin noise function can be reconstructed, without the need to store and generate the entire data set. In this sense, Perlin noise can be considered to be quasi-random noise. Further detail regarding an implementation of Perlin Noise is given in the Implementation section.

Game Boy Advance

Released in 2001, the Game Boy Advance (GBA) is the successor to the multi-million selling Game Boy and Game Boy Color. The most important specifications of the Game Boy Advance are detailed in Table 1.

| | |
|--------------------------|------------------|
| Processor | 16Mhz |
| Memory | 384 kilobytes |
| Screen Resolution | 240 * 160 pixels |
| Available Colours | 32,768 |

Table 1 Important specifications of the GBA

From Table 1, it can be seen that the GBA is very limited in terms of processing power and available Random Access Memory (RAM). Coupled with the ease of programming for the system, this makes the GBA a perfect platform on which to test our implementation.

In the next section, the implementation details of the proposed method are described, and example screenshots of the application in action on the GBA are given.

IMPLEMENTATION

In this section, the method by which massive terrains can be constructed with a minimal memory footprint is discussed. The standard Perlin Noise implementation is explained, followed by details of our novel use of the algorithm.

Perlin Noise Map of Terrain Tiles

The basis of the technique described here is what is known as a terrain tile. A tile describes a square block of terrain. Terrain tiles are used extensively in many popular game engines such as Torque (Marshall et al. 2004). Each tile is typically defined by two groups of data – a height map, which details the geometry of the terrain, and a texture map, which details the colours and shading of the terrain (Rabinovich and Gotsman 1997). Figure 1 (a) and (b) details examples of both category of maps. Both images in Figure 1 were created using the Wilbur random terrain generation application (Slayton 2001).

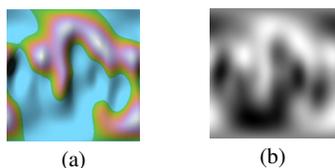


Figure 1 (a) Texture map (b) Grayscale Height Map.

By placing these tiles together in a random order, a large area of terrain can be defined – see Figure 2(a). However, as the tile size decreases, the storage cost of the order of the tiles can become expensive, and can eventually begin to approach that of the traditional method of storing each height of terrain in an array – see Figure 2 (b) and (c). This can make such an approach unsuitable for a limited memory device.

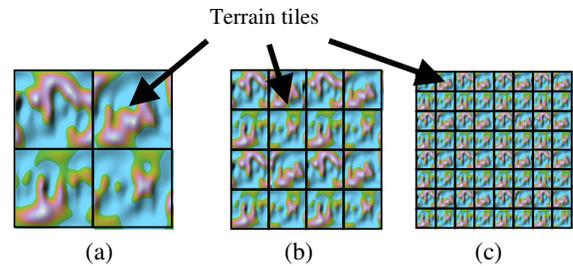


Figure 2 (a) Four 128*128 tiles (b) Eight 64 *64 tiles. (c) Sixty four 32 * 32 tiles. A decrease in tile size increases detail, but leads to an increase in memory requirements.

This problem can be alleviated using Perlin noise to govern the placement of tiles. As mentioned in the previous section, any subsection of the data generated by a Perlin noise function can be recreated given the same input values, without the need to store the complete data set generated by the function. Therefore, only the section of data of interest to the viewer needs to be calculated and stored at any one time, and it is guaranteed that other subsections of the data will be generated consistently with the same values.

The main tenet of the approach outlined in this paper is that by defining a number of tiles with small dimensions, and associating each value generated by the Perlin noise function with a terrain tile, then only the small portion of a massive terrain map needs to be generated and stored at any one time, thus providing dramatic saving in terms of computation time and memory - see Figure 3. In Figure 3, each tile is represented by a different shade of gray. As the user moves about the large terrain map, the subset of terrain tile data is repopulated to represent the users immediate environment.

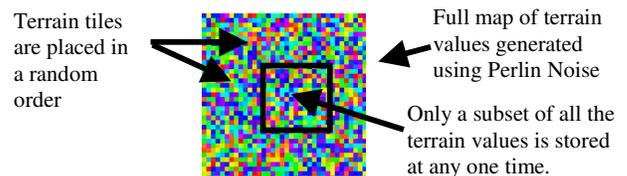


Figure 3 Only a small subset of all the tile values is stored at any one time using a Perlin noise function. This is repopulated as the user moves about the terrain.

Perlin Noise Implementation

The first aspect of the approach explained in this paper is the manner by which the Perlin noise map of terrain values is generated. The primary step of this process is the definition of a grid that defines the overall layout of the map. Each value in this grid is known as a terrain structure value. These terrain structure values are simply random numbers, and are not related to any particular terrain tile. Rather, they are used in conjunction with adjacent terrain structure values in order to define intermediate terrain tile values procedurally. Each

of these terrain tile values is a reference to a particular terrain tile. Although this grid of terrain structure values will have tiny dimensions, it can be thought of as being virtually large, as each element represents a point along a larger map – see Figure 4. So, for example, a 4 * 4 array, with a virtual distance of 1024 pixels between elements of the array, represents a 4096 * 4096 pixel terrain map.

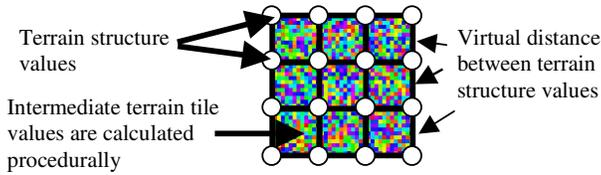


Figure 4 A grid of terrain structure values defines the overall layout of the terrain. Intermediate values are calculated procedurally.

Next, the terrain structure values are used to procedurally define the intermediate terrain tile references that describe the terrain immediately surrounding the user.

Firstly, the area within the grid of terrain structure values that currently houses the user is located, and the points that define the boundaries of that area are recorded. So, taking the example from above, if the player was at point (1056,3042) on a 4*4 map with a virtual distance of 1024 pixels between each element, then the player would be located between the terrain structure elements (1,2), (1,3), (2,2) and (2,3), as detailed in Figure 5 below.

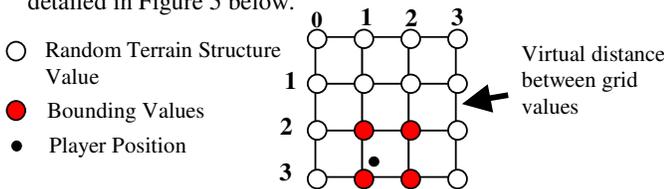


Figure 5 Location of a player’s position within the terrain structure grid.

By linearly interpolating between the two top and the two bottom bounding values using the player’s horizontal position, and interpolating between the results of these operations using the player’s vertical position, a value which represents the tile in which the user resides can be defined – see Figure 6.

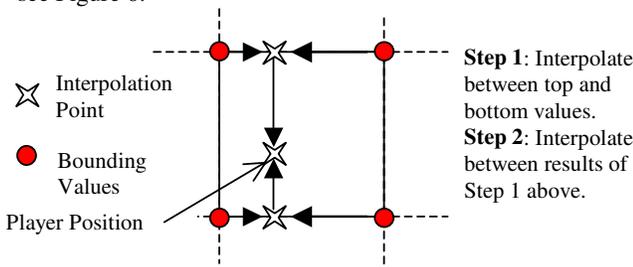


Figure 6 Resolving the terrain tile value based on the players position and the calculated bounding values.

Map of Terrain Tiles

Traditionally, the value generated by the algorithm described above would be used in conjunction with other generated values in order to represent a texture or naturally occurring phenomena procedurally, as in Figure 7.



Figure 7 An example of a traditional output from the Perlin noise function

However, for this implementation, each value is actually a reference to a particular terrain tile. As there will be only a limited number of terrain tiles, and the generated value may exceed the total number of terrain tiles, the result may have to be scaled. This can be achieved using a simple modulus operation.

This process is repeated using the coordinates surrounding the player. The number of repetitions required will vary, depending on the size of each individual tile and the overall map size required. This results in a temporary array of data containing references to the terrain tiles that immediately surround the user. An example of such an array is given in Figure 8. For clarity, each separate map is represented by a different shade of gray. Each colour represents a map similar to that in Figure 1(a).

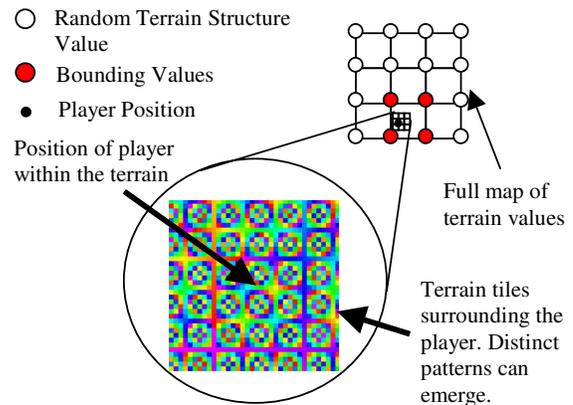


Figure 8 Only the terrain tiles that immediately surround the player are calculated. The player is located at the centre of these tiles.

However, as can be seen in Figure 8, through the use of this method alone, a series of distinct patterns can be observed in the placement of terrain tiles. Such patterns are exactly what Perlin noise is celebrated for. However, in the technique under discussion, more randomness is desired to generate truly random landscapes. This is achieved via the use of another array of random values, which is populated along with the terrain structure grid. The value acquired by the interpolation process is used to index into this random array, allowing for another layer of randomness, with little computation overhead. An example of the temporary array of terrain tile values produced using this method is given in Figure 9. Again, for clarity, each separate map is represented by a different shade of gray.

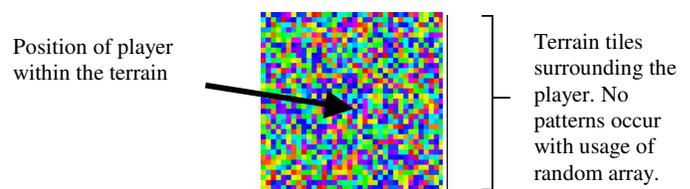


Figure 9 No discernable patterns appear when a random number look-up table is utilised

This temporary array represents only a small subsection of the overall terrain data. However, this fact is not noticeable to the user, as the temporary array of terrain tile references is repopulated when the player reaches a certain distance from the perimeter of the map. This array is then used as input to the rendering process.

Rendering Process

The rendering process uses a method known as ray-casting and voxels in order to draw the relevant terrain tiles to the screen in a timely fashion (Kreeger et al. 1998; Steinbach et al. 2000). Although this method has its limitations, such as only two planes of movement, it allows for a pseudo three-dimensional effect on resource-limited hardware such the GBA, at a stable frame-rate. Figure 10 details screenshots of the rendering application in action on the GBA. Superimposed on the images is a grid defining the boundaries of each terrain tile. Figure 10 also reinforces the relationship between terrain tiles, as seen in Figure 1 (a) and (b), and the temporary array of tile references as seen in Figure 8 and 9.

Each value of the temporary array is a reference to a particular terrain tile

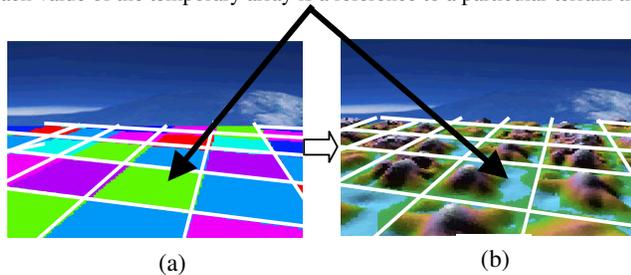


Figure 10 (a) The temporary array of tile values rendered to the screen. Each tile is represented by a separate colour. (b) The associated height and colour maps rendered to the screen.

RESULTS

Computation Time

As a preliminary exercise, the computation time required to calculate the temporary array of terrain tiles surrounding the player was recorded. This is achieved by using an in-built timer on the GBA that is incremented on every clock tick. A running average of the amount of time in milliseconds taken to populate the tile map is noted. In each case, the tile map required was 1024 * 1024 pixels. The tester navigated the environment for 60 seconds, and at the end of this time, the average computation time was recorded. The results of this experiment are presented in Table 2.

As can be seen from Table 2, the computation time required is minimal, and can be calculated in real time on the GBA. As no floating-point arithmetic is used in the calculation of the terrain tile values, this process is suitable for a wide range of devices where no hardware support for floating point values is available. To further demonstrate the efficiency of this approach, the computation time required for the creation of a completely new random map was also investigated. In each case, random maps were generated for 60 seconds, and a running average of the computation time was recorded

using the in-built GBA timers. As this process simply involves the regeneration of the terrain structure grid, the computation time required is minimal, as detailed in Table 3.

| Tile Size | Tiles required | Time |
|------------------|----------------|--------|
| 128 * 128 pixels | 8 * 8 tiles | <1 ms |
| 64 * 64 pixels | 16 * 16 tiles | 10 ms |
| 32 * 32 pixels | 32 * 32 tiles | 67 ms |
| 16 * 16 pixels | 64 * 64 tiles | 165 ms |

Table 2 The computation time required for the generation of a 1024 * 1024 pixel map using terrain tiles of varying size.

| Terrain Structure grid dimension | Time |
|----------------------------------|----------|
| 4 * 4 grid elements | 1.25µs |
| 8 * 8 grid elements | 3.5µs |
| 12 * 12 grid elements | 6.625 µs |
| 16 * 16 grid elements | 16.5µs |

Table 3 The computation time required for the generation of a new random map is minimal as the terrain structure grid size is insignificant.

Memory Usage

The most critical aspect of implementing the proposed technique in mobile devices is how it exploits the available memory. We therefore consider memory usage to be the most important metric by which to assess our proposed approach. This will be done by performing a comparative analysis with the traditional technique of storing a single large terrain geometry map and texture map.

In Table 4 the memory usage of a traditional terrain geometry map and texture map is analysed. Every element of each map is assumed to be a 16 bit unsigned integer.

| Individual Map Size | RAM |
|---------------------|----------|
| 256 * 256 pixels | 256 kb |
| 512 * 512 pixels | 1024 kb |
| 1024 * 1024 pixels | 4096 kb |
| 2048 * 2048 pixels | 16384 kb |

Table 4 Values representing cost of storing full maps in memory.

As can be seen from Table 4, the storage of a single randomly generated map utilises a large volume of memory resources. The generation of a small 256 * 256 pixel map requires approximately 66% of the entire 384 KB of RAM on a GBA. As the map dimensions double, the memory requirements quadruple.

Next, the storage cost of the approach detailed in this paper was analysed – see Table 5. Results are given for the cost of storing one hundred terrain tiles both in fixed storage and in RAM. Each tile consists of a height and texture map, and every element of a map is an unsigned 16-bit integer. The RAM columns display the cost of storing the temporary array of terrain tile references that is procedurally generated. The dimensions of this array vary depending on the individual terrain tile size. If, for example, a temporary map of dimensions 1024 * 1024 pixels is required using terrain tiles of dimensions 8 * 8 pixels, then a 128 * 128 temporary terrain tile reference array is required.

| Terrain Tile Dimensions | Cost to store 100 Terrain Tiles | RAM 256 * 256 map | RAM 512 * 512 map | RAM 1024 * 1024 map | RAM 2048 * 2048 map |
|-------------------------|---------------------------------|-------------------|-------------------|---------------------|---------------------|
| 8 * 8 pixels | 25 kb | 2 kb | 8 kb | 32 kb | 128 kb |
| 16 * 16 pixels | 100 kb | 0.5 kb | 2 kb | 8 kb | 32 kb |
| 32 * 32 pixels | 400 kb | 0.125 kb | 0.5 kb | 2 kb | 8 kb |
| 64 * 64 pixels | 1600 kb | 0.03125 kb | 0.125 kb | 0.5 kb | 2 kb |
| 128 * 128 pixels | 6400 kb | 0.0078125 kb | 0.03125 kb | 0.125 kb | 0.5 kb |

Table 5 Values representing memory cost of storing the terrain tiles and the cost of storing temporary tile value array in Random Access Memory (RAM).

Table 5 shows that the amount of RAM employed by the proposed approach is minimal. If the individual map dimensions are increased in size by 100%, the amount of RAM required reduces by 75% whereas the fixed storage requirements increase by 400%. However, the cost of storing the terrain tiles is nominal at smaller sizes, thus making them suitable for download over low bandwidth connections, such as on a mobile phone network.

Regardless of the dimensions of the overall terrain map required, the figures presented in the RAM columns of Table 5 remain constant. This is due to the fact that, as discussed in the Implementation section, only a subset of the overall terrain map is ever stored, regardless of variation in the dimensions of the overall terrain map. Therefore, any size map could be potentially represented on a resource-limited device such as the Game Boy Advance, with no implications for the overall memory requirements.

CONCLUSIONS

In this paper we have described a method of representing the large volumes of data required to store a random terrain on a device with a limited amount of Random Access Memory known as the Game Boy Advance. By using terrain tiles and Perlin Noise, we have shown how this technique is particularly suitable to devices with limited processing power, as it makes no use of floating point arithmetic. Therefore, the terrain tile positions can be calculated easily at run time. In addition, a totally new random terrain can be generated rapidly, as only the small terrain structure grid needs to be repopulated.

It has also been shown that a massive terrain can be represented using a minimal amount of both fixed and dynamic memory. The cost required to store the necessary terrain tiles can be small, thus making this approach suitable for devices with limited storage capacity, and slow download speeds such as mobile telephones. As only a portion of the overall terrain is stored at any one time, the volume of memory used is fixed, regardless of variation in the dimensions of the overall terrain.

Future work will involve investigating methods of seamless blending between adjacent terrain tiles, such as inter-tile interpolation, in order to provide a more uniform visual experience to the user.

ACKNOWLEDGEMENT

This work was funded by Enterprise Ireland Basic Research Grant SC/2002/129/.

REFERENCES

- Dudgeon, J. E. and R. Gopalakrishnan (1996). "Fractal-based modeling of 3D terrain surfaces". In *Bringing Together Education, Science and Technology*, (Tampa, Florida, USA, 11-14 April 1996), IEEE, 246 - 252.
- EDGE Magazine (2004). "Mobiles prepare for 3D revolution." In *EDGE Magazine Issue 135, April 2004*: 7-9.
- Holtkämper, T. (2003). "Real-time gaseous phenomena: a phenomenological approach to interactive smoke and steam". In *Computer graphics, virtual reality, visualisation and interaction in Africa*, (Cape Town, South Africa, February 03 - 05), ACM Press, New York, NY, USA, 25 - 30.
- Kelley, A. D., M. A. Casey and G. M. Nielson (1988). "Terrain simulation using a model of stream erosion". In *15th Annual Conference on Computer graphics and interactive techniques 1988*, ACM Press New York, NY, USA, 263 - 268.
- Kreeger, K., I. Bitter, F. Dachille, B. Chen and A. Kaufman (1998). "Adaptive perspective ray casting". In *1998 IEEE symposium on Volume visualization*, (Research Triangle Park, North Carolina, United States, ACM Press, New York, NY, USA, 55 - 62.
- Marshall, D., A. McCoy, D. Delaney, S. McLoone and T. Ward (2004). "A Realistic Distributed Interactive Application Testbed for Static and Dynamic Entity State Data Acquisition". In *Irish Systems and Signals Conference*, (Belfast, Ireland, 30 June - 2 July), IEE, 83-88.
- Perlin, K. (1985). "An image synthesizer." In *ACM SIGGRAPH Computer Graphics Issue 3, July*: 287 - 296.
- Pickover, C. A. (1995). "Generating extraterrestrial terrain." In *Computer Graphics and Applications, IEEE Issue 2, March 1995*: 18 - 21.
- Rabinovich, B. and C. Gotsman (1997). "Visualization of large terrains in resource-limited computing environments". In *8th conference on Visualization '97*, (Phoenix, Arizona, United States, 19-24 October), IEEE Computer Society Press, Los Alamitos, CA, USA, 95 - 102.
- Rittern, H., T. Voigt, M. Tian and J. Schiller (2003). "Experiences using a dual wireless technology infrastructure to support ad-hoc multiplayer games". In *Proceedings of the 2nd workshop on Network and system support for games*, (Redwood City, California, May 22 - 23), ACM Press, New York, NY, USA, 101 - 105.
- Slayton, J. (2001). Wilbur Terrain Generator <http://www.ridgecrest.ca.us/~jslayton/index.html>.
- Steinbach, E., B. Girod, P. Eisert and A. Betz (2000). "3-D reconstruction of real-world objects using extended voxels". In *2000 International Conference on Image Processing*, (Vancouver, BC Canada, 10-13 September 2000), 569 - 572.
- Ye, A. G. and D. M. Lewis (1999). "Procedural texture mapping on FPGAs". In *1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, (Monterey, California, United States, February 21 - 23), ACM Press, New York, NY, USA, 112 - 120.

USING A HALF-JACOBIAN FOR REAL-TIME INVERSE KINEMATICS

Michael Meredith & Steve Maddock
Department of Computer Science
University of Sheffield
United Kingdom

E-mail: M.Meredith@dcs.shef.ac.uk, S.Maddock@dcs.shef.ac.uk

KEYWORDS

Inverse Kinematics, Computer Character Animation, Real-time

ABSTRACT

Due to their scalability, numerical techniques often form part of an inverse kinematics (IK) solver. However, because of their iterative nature, such methods can be slow. So far research into the field of kinematics has failed to find a general non-numerical solution to the problem. Many researchers have proposed hybrid techniques yet these still rely on a numerical aspect. It is therefore important to find ways of using numerical techniques as efficiently as possible. In this paper we take a look at the Jacobian-based IK solver and techniques that allow this method to be used as an efficient real-time IK solver. We demonstrate how the half-Jacobian can be used effectively where normally the full Jacobian would be considered the principal technique. The result of this is much reduced computational costs when applying IK to articulated characters.

INTRODUCTION

The problem domain that is tackled by inverse kinematics solvers was first formulated in the mechanical engineering literature (Craig 1955) and more specifically research into the field of robotics. We are interested in its application in computer character animation. The issue that inverse kinematics attempts to resolve is to find a set of joint configurations of an articulated structure based upon a desirable end-effector location. This is expressed mathematically in Equation 1.1 where θ represents the set of orientation values for a structure and X is the global position of a given limb in the hierarchy.

$$\theta = f^{-1}(X) \quad (1.1)$$

There have been many varied techniques used as an inverse kinematics solver. The fastest techniques, analytical algorithms, tend to suffer from poor scalability, whereas the scalable techniques, such as numerical iteration, suffer from poor solver times. Many techniques that have been proposed to offer speed advantages utilise numerical solvers therefore it is important to consider ways that such techniques can be used efficiently. A review of many of the present IK techniques is given in the following section.

We then present an analytical look at the iterative Jacobian approach to inverse kinematics and discuss techniques that allow the method to be used effectively. Following this we present a real-time application that drives a walking character

around rough terrain to demonstrate the effectiveness of our Jacobian interpretation.

RELATED WORK

We can identify 4 different categories of IK solver: geometric/analytical algorithms, cyclic co-ordinate descent (CCD) techniques, differential techniques, and hybrid methods (Tolani *et al.* 2000) which mix together various aspects of the first three techniques.

The geometric/analytical algorithms (Chin 1996, Kwang-Jin and Hyeong-Seok 2000, Paul and Shimano 1988) tend to be very quick because they reduce the IK problem to a mathematical equation that need only be evaluated in a single step to produce a result. However, for large chains of links the task of reducing the problem to a single-step mathematical equation is impractical. Therefore geometric/analytical techniques tend to be less useful in the field of character animation.

IK solvers that are based on CCD (Eberly 2001, Wang and Chen 1991, Welman 1993) use an iterative approach that takes multiple steps towards a solution. The steps that the solver takes are formed heuristically, therefore this step can be performed relatively quickly. An example of a possible heuristic would be to minimise the angle between pairs of vectors created when projecting lines through the current node and end-effector and current node and desired location. However, because the iterative step is heuristically driven, accuracy is normally the price paid for speed. Another issue with this technique is that only one joint angle is updated at a time, which has the unrealistic result of earlier joints moving much more than later limbs in the IK chain.

As with the CCD technique, differential-based techniques (Watt and Watt 1992, Zhao and Badler 1994) utilise an iterative approach that requires multiple steps to find a solution. The steps that the algorithm makes are determined via the use of the system Jacobian that relates small changes in joint configurations to positional offsets. Since all the joint angles are updated in a single step, the movements are dissipated over the whole chain which results in a more realistic looking posture.

By their nature, iterative-based techniques are generally slower at producing a desirable result when compared to their analytical counterparts. However the problem with the analytical methods is their lack of scalability. Fedor (Fedor 2003) explores this trade-off between speed, accuracy and scalability in an IK solver. One of the results from this work demonstrates that differential-based numerical solutions,

although slower than both CCD and analytical techniques, provide better results for larger chains. This highlights the importance of refining numerical techniques such that we maintain accuracy and scalability but drive solution time down.

One such solution proposed by Tang *et al.* (Tang *et al.* 1999) makes use of the SHAKE algorithm (Ryckaert *et al.* 1997) to achieve a fast iterative-based IK solver. This technique treats a hierarchical structure as point masses that are related by system constraints. This is in contrast to the Jacobian-based technique that encapsulates the articulated information and thereby provides us the cohesion between links for free.

In order to achieve a desired end-effector location, the mass points of the SHAKE system are adjusted per cycle until a global goal has been reached. This includes meeting a threshold of acceptable error on the constraints. However because of the lack of node dependency of the algorithm, normally the points will lose their distance relationships between each other. To counter this issue, correcting forces are iteratively applied to each point to reassert cohesion between links therefore the accuracy of parent-child distances directly effects solver time. Without a reasonable level of accuracy at this point, the appearance of rigid links moving about each other would occur. This is an issue that the Jacobian-based techniques are not affected by.

The time complexity of the SHAKE algorithm is suggested by Tang *et al.* to be $O(n^2)$ with respect to the number of constraints. However since each link in a hierarchical chain requires a constraint to impose cohesion, the time to solve a system is also minimally $O(n^2)$ with respect to the number of links in the chain. The inclusion of additional system constraints such as joint angle limits has a further detrimental effect on solution time therefore making the algorithm less applicable to real-time applications as the number of links increases.

Another real-time IK technique proposed by Shin *et al.* (Shin *et al.* 2001) that is used for computer puppetry makes use of a hybrid solution. This technique attempts to use analytical solutions where possible, except in cases where a large amount of body posturing is required, where a numerical implementation is invoked. The numerical solver only acts upon the IK chain defined between the root and the upper body while the analytical solver is used for the limbs of the character.

The hybrid use of IK solvers used by Shin *et al.* demonstrates a good method for performing real-time IK. However the analytical aspect assumes some knowledge about the character's structure (Lee and Shin 1999, Tolani *et al.* 1996). This means that the overall IK technique is not a general one that can be applied to arbitrary IK chains. The other potential problem with the hybrid technique is similar to the CCD techniques in that not all joint angles are updated simultaneously which means unrealistic and unproportional posture configurations could result.

From the research done in the field of real-time IK, it is apparent that analytical solutions by themselves are not scalable enough to meet the demands of modern

computer-based IK problems. Therefore numerical techniques are used as either a substitute or in serial with an analytical solution, which serves to highlight the importance of having fast numerical solutions. Furthermore these solutions should operate on the whole hierarchical structure equally to avoid unrealistic postures. These are the issues we address with our Jacobian-based approach for real-time IK.

OUR INVERSE KINEMATICS SOLUTION

Jacobian Inverse Kinematics

Our implementation of inverse kinematics is based upon the well-established Jacobian technique. The objective of this technique is to incrementally change joint orientations from a stable starting position towards a configuration state that will result in the required end-effector being located at the desired position in absolute space. The amount of incremental change on each iteration is defined by the relationship between the partial derivatives of the joint angles, θ , and the difference between the current location of the end effector, X , and the desired position, X_d . The link between these two sets of parameters leads to the system Jacobian, J . This is a matrix that has dimensionality $(m \times n)$ where m is the spatial dimensional of X and n is the size of the joint orientation set, θ . The Jacobian is derived from the equation for forward kinematics, Equation 1.2, as follows:

$$X = f(\theta) \quad (1.2)$$

Taking partial derivatives of Equation 1.2:

$$dX = J(\theta)d\theta \quad (1.3)$$

where

$$J_{ij} = \frac{\partial f_j}{\partial x_i} \quad (1.4)$$

Rewriting Equation 1.3 in a form similar to inverse kinematics (Equation 1.1) results in Equation 1.5. This form of the problem transforms the under-defined system into a linear one that can be solved using iterative steps.

$$d\theta = J^{-1}dX \quad (1.5)$$

The problem now is that Equation 1.5 requires the inversion of the Jacobian matrix. However because of the under-defined problem that the inverse kinematics technique suffers from, the Jacobian is very rarely square. Therefore, in our implementation we have used the right-hand generalised pseudo-inverse to overcome the non-square matrix problem, as given in equation 1.6.

Generating the pseudo-inverse of the Jacobian in this way can lead to inaccuracies in the resulting inverse that need to be reduced. Any inaccuracies of the inverse Jacobian can be detected by multiplying it with the original Jacobian then subtracting the result from the identity matrix. A magnitude error can be determined by taking the second norm of the resulting matrix multiplied by dX , as outlined in Equation 1.7.

If the error proves too big then dX can be decreased until the error falls within an acceptable limit.

An overview of the algorithm we used to implement an iterative inverse kinematics solution is as follows:

- 1) Calculate the difference between the goal position and the actual position of the end-effector:

$$dX = X_g - X$$

- 2) Calculate the Jacobian matrix using the current joint angles: (using Equation 1.4)

- 3) Calculate the pseudo-inverse of the Jacobian:

$$J^{-1} = J^T (JJ^T)^{-1} \quad (1.6)$$

- 4) Determine the error of the pseudo-inverse

$$\text{error} = \|(I - JJ^{-1})dX\| \quad (1.7)$$

- 5) If $\text{error} > e$ then

$$dX = dX / 2$$

restart at step 4

- 6) Calculate the updated values for the joint orientations and use these as the new current values:

$$\theta = \theta + J^{-1}dX$$

- 7) Using forward kinematics determine whether the new joint orientations position the end-effector close enough to the desired absolute location. If the solution is adequate then terminate the algorithm otherwise go back to step 1.

The computational demand of the algorithm is relatively high over a number of iterations, so well-defined character hierarchies are advantageous. This means that each node in the articulation is defined by the minimum number of degrees of freedom (DOF) required thereby making θ as small as possible. For example, pivot joints such as an elbow would only be modelled using a single DOF whereas a ball and socket joint like the shoulder would need 3 Euler DOFs to represent the range of possible movements.

The use of well-defined hierarchies further helps to prevent the inverse kinematics solver from producing unnatural-looking postures. However this still does not cover all of the potential unnatural poses the solver can return. In order to restrict the IK solver to the orientation space of only possible character configurations, joint orientation restrictions can be enforced within the scope of the existing algorithm. The simplest way of incorporating such constraints is to crop the joint angles. This requires Step 6 of the algorithm to be modified in the following way:

- 6) Calculate the updated values for the joint orientations and use these as the new current values:

$$\theta = \begin{cases} \text{lowerbound} & \text{if } \theta + J^{-1}dX < \text{lowerbound} \\ \text{upperbound} & \text{if } \theta + J^{-1}dX > \text{upperbound} \\ \theta + J^{-1}dX & \text{otherwise} \end{cases}$$

The time to complete the IK algorithm for a given end-effector is an unknown quantity due to an arbitrary number of iterations required. However the time to complete a single iteration is constant with respect to the dimensionality of X and θ which is unchanged under a

complete execution of the algorithm. Therefore by placing an upper limit on the number of iterations we can set a maximum time boundary for the algorithm to return in. If the solver reaches the limit then the algorithm returns the closest result it has seen.

In 3-dimensional space, the dimensionality of X in a Jacobian-based inverse kinematics solver is generally either 3 or 6. The 6-dimensional X vector is normally used as it contains both positional and orientation information whereas a 3-dimensional vector only contains positional information for an end-effector.

From the inverse kinematics algorithm outlined above, it is clear that the 3-dimensional X vector is quicker over its counterpart and should always be used when orientation is not required. However there are times that orientation is required but it is still possible to use the 3-dimensional vector which is demonstrated in our application of the algorithm present later.

To see how much of a cost difference there is between the two sizes of X vector, the corresponding complexity analysis of them is illustrated in the following section.

Complexity Analysis Of The X Vector

We need a technique to perform the inverse of the square matrix JJ^T . For the 3-dimensional X vector, which uses a (3 x 3) matrix we use an analytical solution whereas for the 6-dimensional X vector, which uses a (6 x 6) matrix, we use LU Decomposition.

LU Decomposition and Analytical Inversion

LU decomposition can be used to determine the inverse of a square matrix by using the matrix identity, $AA^{-1} = I$, where I is an identity matrix (and in this case it has dimensionality (6 x 6)). The application of LU decomposition to this equation requires matrix A to be split into 2 further matrices that have the form of lower and upper matrices as illustrated in Equation 1.8.

$$A = LU = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ * & 1 & 0 & \dots & 0 \\ * & * & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \dots & 1 \end{bmatrix} \begin{bmatrix} * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & * \end{bmatrix} \quad (1.8)$$

The decomposition of A into the two matrices allows the original matrix identity to be rewritten into the form of Equation 1.9, which can be solved using forward and backward substitution.

$$LUA^{-1} = I \quad \Rightarrow \quad L(UA^{-1}) = I \quad (1.9a)$$

$$\Rightarrow LY = I \quad (1.9a)$$

$$\wedge UA^{-1} = Y \quad (1.9b)$$

Our algorithm for performing a (6 x 6) LU Decomposition inverse gives a complexity of 619 flops (Meredith and Maddock 2004).

In comparison, the analytical inversion of a (3 x 3) matrix is given in Equation 1.10. This equation can be directly encoded. The complexity of calculation is 51 flops (Meredith and Maddock 2004): 36 multiplications, 1 division and 14 additions & subtractions.

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}^{-1} = \frac{\begin{bmatrix} r_{11}r_{22} - r_{21}r_{12} & r_{21}r_{02} - r_{01}r_{22} & r_{01}r_{12} - r_{11}r_{02} \\ r_{20}r_{12} - r_{10}r_{22} & r_{00}r_{22} - r_{20}r_{02} & r_{10}r_{02} - r_{00}r_{12} \\ r_{10}r_{21} - r_{20}r_{11} & r_{20}r_{01} - r_{00}r_{21} & r_{00}r_{11} - r_{10}r_{01} \end{bmatrix}}{r_{00}(r_{11}r_{22} - r_{12}r_{21}) + r_{01}(r_{12}r_{20} - r_{10}r_{22}) + r_{02}(r_{10}r_{21} - r_{11}r_{20})} \quad (1.10)$$

The decision to use an analytical solver for the smaller matrix and LU decomposition for the larger one is demonstrated in Figure 1.1. The results given in Figure 1.1 were obtained using a matrix with all elements non-zero so the analytical technique was unable to make use of zeros to cut off the co-factor expansions. This is a valid assumption because it would be most unlikely that the (6 x 6) matrix that needs to be inverted in the pseudo-inverse would actually contain any zeros.

Figure 1.1 shows that the analytical approach to solving matrix inversion is only better for matrices that have dimensionality equal to or less than 3. After this size, the number of flops required to solve an analytical inverse increases in a cubic fashion with respect to dimensionality whereas the LU technique increases at the lower squared rate. This analysis justifies the use of an analytical solution for the (3 x 3) matrix while using LU decomposition for the inversion of the larger (6 x 6) matrix.

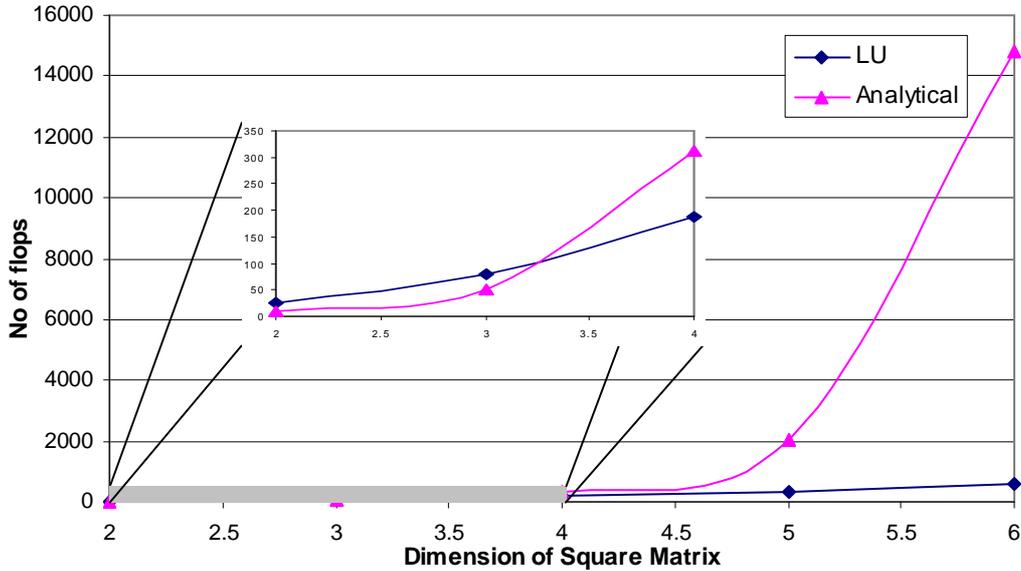


Figure 1.1: Demonstration of the complexity of solving a square matrix using an analytical and LU decomposition technique.

Calculating The Jacobian

If the Jacobian definition of Equation 1.4 is divided by a differential time element, the resulting equivalence provides a mapping between angular velocities in state space, $\dot{\theta}$, and linear velocities in Cartesian space, \dot{X} . This result is illustrated in equation 1.11.

$$\dot{X} = J(\theta)\dot{\theta} \quad (1.11)$$

In the case of a 6-dimensional X vector, \dot{X} consists of linear velocity, V , and angular velocity, Ω , components, whereas the 3-dimensional X vector only includes the linear velocity. Both the linear velocity and angular velocity are with respect to a global frame of reference as too are the partial derivatives of the Jacobian. The Jacobian linking the linear and angular velocity of the end-effector, with the intermediary local angular velocities, is given in equation 1.12, where there are i DOFs in the IK chain.

$$\begin{bmatrix} V \\ \Omega \end{bmatrix} = \begin{bmatrix} b_1, b_2, \dots, b_i \\ a_1, a_2, \dots, a_i \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \vdots \\ \dot{\theta}_i \end{bmatrix} \quad (1.12)$$

In Equation 1.12, the a components are of the local axes for a given link transformed into the global frame of reference. The b elements of the Jacobian are the cross products of the corresponding a axis with the spatial difference between the global origin of the current limb and the absolute location of the end of the articulation, P_e (Equation 1.14). The DOFs within the state space are normally ordered such that limbs from the root are considered first followed by their children, following this pattern to the end of the chain. Using this pattern, the orientation values of the required axes for each limb can be obtained from a transformation matrix, 0T_j , that converts points defined in the limb's local orientation into a global position. Equation 1.13 illustrates this for the j^{th} limb

in the IK chain (note that this assumes a right-handed coordinate system):

$${}^0T_j = \begin{bmatrix} a_{xj} & a_{yj} & a_{zj} & P_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.13)$$

The parameter P_j in Equation 1.13 also gives the global position of the origin of the limb thereby aiding in the determination of the b components in Equation 1.14.

$$b = a_i \times (P_e - P_j) \quad (1.14)$$

Using the chaining principle for calculating the transforms of the local axes into a global reference frame (${}^0T_j = {}^0T_1 \times {}^1T_2 \times \dots \times {}^{j-1}T_j$), the direct implementation of this subsection in 3-dimensional space yields a constant complexity. Assuming that each link has 3 DOFs, the complexity associated with each limb in the IK chain is given by 162 flops: 98 multiplications and 64 additions & subtractions. The assumption of 3 DOFs does not add a great deal of complexity if it is an overestimate since each DOF contributes only 9 flops to the overall result (where the 9 flops is the calculation of the cross product).

Determining The Pseudo-Inverse Of The Jacobian

Using the complexity derivation of the inversion of a square matrix from the above sections, the complexity of the pseudo-inverse of the Jacobian, as given in Equation 1.6, can be calculated. Table 1.1 outlines the number of flops required to calculate the pseudo-inverse depending on the size of the X vector. The variable n is the size of the state space, θ (i.e. the sum of all the links' DOFs). It should also be noted that there is no inclusion of complexity to calculate the transpose of matrices when they are required as this can be handled at no extra cost by simply swapping out indexing parameters.

Complexity Of The Whole IK Solver

The complexity of a single loop of the IK algorithm described above can be derived using the complexity analyses of the smaller parts of the algorithm already determined. This is shown in Table 1.2 where m is the number of inner loops executed at stage 5 of our algorithm.

As Table 1.2 illustrates, the use of a 3-dimensional X vector appears to be about 2½ times less computationally demanding

| Size of Matrix Operation | (3 x n) 3D X Vector | (6 x n) 6D X Vector |
|----------------------------|------------------------|------------------------|
| $A = JJ^T$ | 18n - 9 | 72n - 36 |
| $B = A^{-1}$ | 51 | 619 |
| $J^T B$ | 15n | 33n |
| $J^{-1} = J^T (JJ^T)^{-1}$ | 33n + 42 | 105n + 583 |

Table 1.1: Number of flops required to calculate the pseudo-inverse of a non-square matrix.

than its 6-dimensional counterpart. Considering only the major factor of the complexity, which is the size of the state space, n , the 3-dimensional X vector should be 238.9% quicker than the alternative. However, the complexity of each algorithm is not only dependent on the size of the state space but also on the number of inner loops which are required to make the inversion of the Jacobian stable enough to provide meaningful results. Therefore it needs to be shown that the use of a smaller Jacobian in the 3-dimensional X vector case does not adversely affect the pseudo-inverse. This does not appear to be the case as illustrated with the empirical dataset present in the following section.

Since the smaller X vector can be shown to be less computationally demanding by a significant factor, it raises the issue of whether the smaller X vector can be used even when orientation is important. The following section gives a brief discussion of possible application areas for using the half-Jacobian over the full-Jacobian. Thereafter we illustrate an example for which we have used the half-size Jacobian in an application that would normally be considered a full-sized Jacobian problem domain.

USING THE HALF- OVER THE FULL-JACOBIAN

An obvious application of the half-Jacobian is in applications that do not discriminate against the orientation of the final link in an inverse kinematics chain. In applications of inverse kinematics where the orientation of the end-effector has little consequence, the 3-dimensional X vector should always be used to reduce the computation effort required. For example, when configuring a spider's legs using IK, because the spider effectively walks on the tips of its legs, the orientation of this end point is immaterial. Therefore only the 3-dimensional X vector would be required. As illustrated in Table 1.2, using the full-sized Jacobian in such cases would be less efficient than the half-sized Jacobian.

Another, more subtle, application of the half-sized X vector is

| Size of X Vector | 3 | 6 |
|--------------------------|-----------------------|------------------------|
| Algorithm Stage | | |
| 1. Calc. increment | 3 flops | 6 flops |
| 2. Calc. Jacobian | 162 flops | 162 flops |
| 3. Calc. Pseudo-Inverse | 33n + 42 flops | 105n + 583 flops |
| 4. Check for convergence | 18n + 15 flops | 72n + 66 flops |
| 5. Reduce dX | 18m flops | 72m flops |
| 6. Update joint angles | 6n flops | 12n flops |
| 7. Calc. new position | 38n flops | 38n flops |
| Total | 95n + 18m + 252 flops | 227n + 72m + 817 flops |

Table 1.2: Complexity analysis of our Jacobian based IK solver

in situations where the penultimate link in the IK chain has unlimited and full use of all 3 DOFs (in 3 dimensional space). In this scenario the first step is to calculate the position of the penultimate link based on the desired position and orientation of the final node. The 3-dimensional X vector can then be used to position the penultimate node in the chain. Once this is done the desired orientation of the final node can be specified thereby allowing the correct end configuration of the chain.

Other applications where the half-size Jacobian would prove a better technique to employ over the full-size version is in situations of low resolution modelling. For example, if a complex articulated model is being animated as a background entity in a scene, it would be advantageous to switch to the quicker half-Jacobian to solve its configuration. This means that more avatars can be animated in the background of a scene.

There are many other applications where the half-sized Jacobian could substitute for the traditional full-sized version. Currently we have applied the quick half-Jacobian inverse kinematic solver to motion capture retargetting and IK-driven character walking. Both of these applications can easily run in real-time as demonstrated in the following section which describes the latter of our applications.

IK-GENERATED HUMANIOD WALKING

The coupling of a procedural model and an inverse kinematics solver provides the basic building blocks needed to generate the walking motion of a computer character. The procedural model describes the path through which the foot travels during a stride while the IK solver positions (and orientates) the foot along this path over time. The task of tracing the foot along the path would initially appear to require the full-sized Jacobian, inherently requiring the foot to be orientated in a forward facing direction. Without the orientation of the foot taken into account, there are an infinite number of anatomically correct positions the heel could take in order to meet a simple positional constraint. This is possible because the hip joint for a leg can rotate about the axis of the femur approximately ± 90 degrees from the forward facing pose, as illustrated in Figure 1.2.

From the evidence of Figure 1.2, it would seem that the full-sized Jacobian is the only choice of IK solver to drive the walking motion of a humanoid character. However, by realising that in the course of a walking motion, any large hip joint rotations result in unnatural postures, additional constraints can be added to restrict movement to only plausible ranges. This would allow the half-sized Jacobian to be used to calculate the position of the heel and thus simultaneously reduce the potential for orientation error and increase the performance of the solver.

This approach has been used in our implementation of an IK-driven humanoid character, *MovingIK* (Meredith and Maddock 2004) which has the ability to walk over uneven terrain in real-time.



Figure 1.2: Infinite number of positional solutions to fixing a heel plant without regard to the orientation of the foot. The purple ring shows the location of all possible knee positions.

MovingIK, makes use of a procedural stride model to define how the foot moves over time as the character is walking. The source for the procedural stride model in our application comes from a simple mathematical equation whose form is illustrated in Figure 1.3 where a complete cycle ranges between 0 and 3π .

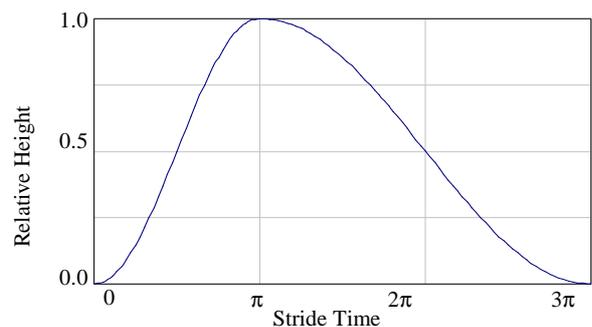


Figure 1.3: Graph of procedural stride used in *MovingIK*

Along with the procedural model used to drive the character's foot through the air, we have a pre-flight stage that rolls the foot from a heel supporting phase to a complete foot supporting phase. This uses the inverse kinematics algorithm to simultaneously plant the heel of the character and gravitate the toes towards the ground. This extra bit of the walking cycle increases the realistic-looking nature of the resulting animation and gives us the ability to model the complete foot as opposed to just the heel.

The character is driven around an uneven terrain in real-time using an analogue joystick that determines parameters such as stride length, stride speed and direction of travel.

Using MovingIK we are able to compare the half-sized and full-sized Jacobian techniques for both performance and realism.

Empirical Results

The results in Table 1.4 are obtained from running *MovingIK* on a Pentium 4 1.4GHz processor with a GeForce2 Ultra. There was a maximum iteration count imposed on the IK solver for the outer loop of 200 cycles while the inner loop was subject to a 20-cycle ceiling. These limits were determined by the empirical running of the IK solver to determine over what limits a solution was very rarely found. The character driven by the user is made up of 18 hierarchical segments where only naturally-occurring DOFs within the human body were permitted. The constraints on each remaining DOF were further limited to joint angles within the scope of normal human movement.

The results given were obtained by driving the character around both flat and uneven terrains. In the case of the uneven terrain, several randomly-generated surfaces were used (including a flight of steps) and the overall results were obtained by averaging the results. Each of the uneven terrains had the same number of vertices and polygons in the model (64,082 polygons compared to 2 polygons for even terrain). The character displayed was that of either a stick figure or a 3D model consisting of 11,101 polygons.

MovingIK was not optimised to use either the half- or full Jacobian but instead provided the ability to switch between the two techniques at run-time. There are three different configurations possible to switch between. The first two modes use only the half- or full Jacobian respectively to calculate the configuration of the character to position the leading foot and trailing toes. The third mode uses a hybrid approach that uses the full Jacobian to determine the configuration of the leading foot and the half-Jacobian to anchor the trailing toes.

The empirical results of driving the computer character within *MovingIK* are illustrated in Table 1.4. It should be noted that during a single frame, *MovingIK* solves two IK chains – one for each leg. An illustration of *MovingIK* is given in Figure 1.4.

The speed-up factor between the full Jacobian and the half-Jacobian, based on the empirical average time per iteration, is 238.5% which when compared to the analytical computed result of 238.9% reinforces the advantages of using the half-Jacobian over the full Jacobian whenever possible.

A further conclusion that can be obtained from these results is that the use of the full Jacobian does not necessarily make the IK solver any more stable. This logical conclusion comes from the fact that the analytical speed-up factor calculated assumes that the inner loop is executed an equal number of times for both algorithms. If this were not the case then the empirical results would show a larger difference in speed up factor due to one algorithm executing the inner loop more times than the other.

CONCLUSIONS & FUTURE WORK

From this analysis of the empirical and analytical results, there is no proven stability advantage from using the full Jacobian compared to that of the half-Jacobian. Therefore there is a definite argument for using the half-sized Jacobian when only the position of an end-effector is needed.

As we have shown, there is also scope for using the quicker half-Jacobian for limited domains when orientation is required as well as position. Although we have only demonstrated this for a walking motion, this represents one of the most fundamental movements in computer character animation. In other work we have also applied this technique to the field of motion capture retargeting with similarly successful results in both speed and visual accuracy. There are many other conceivable domains in which this application can be used by placing extra dynamic constraints on joint angles to prevent the orientation from deviating too much from a natural-looking configuration. An arm, for example, would prove just as suitable a subject for the technique.

| <i>Measurement</i> | <i>IK Mode</i> | All Half Jacobian (3D X Vector) | All Full Jacobian (6D X Vector) | Hybrid Method |
|--|----------------|------------------------------------|------------------------------------|---------------|
| Flat Floor with Stick Character | | 260 fps | 95 fps | 115 fps |
| Flat Floor with Skeleton | | 140 fps | 69 fps | 83 fps |
| Uneven Terrain with Stick Character | | 97 fps | 54 fps | 64 fps |
| Uneven Terrain with Skeleton Character | | 75 fps | 42 fps | 53 fps |
| Average time to execute each IK solver | | 0.24 ms | 5.5 ms | ---- |
| Average Number of iterations | | 18.15 | 180 | ---- |
| Average time per iteration | | 0.013 ms | 0.031 ms | ---- |

Table 1.4: Empirical Results from *MovingIK*

The advantages of using dynamic constraints to transform an orientation and positional IK problem into a position-only task are a speed-up factor of about 238%. There is also no extra cost to adding in constraints to the half-sized Jacobian algorithm because its framework already operates using joint restrictions. Effectively you get the dynamic constraints for free in the Jacobian-based IK solver.

We have already integrated our quick real-time inverse kinematics solver into a motion capture retargeting application where the next step will be to use the solver to simultaneously individualise the character. For this we are looking into the application of weighted IK chains such that different parts of the articulation change with a varying rate to the others. This would give rise to the very simple and quick production of injuries or even varying character builds in computer figures.

REFERENCES

Chin, K.W., "Closed-form and generalized inverse kinematic solutions for animating the human articulated structure.", *Bachelor's Thesis in Computer Science, Curtin University of Technology*, 1996

Craig, J. J., "Introduction to Robotics: Mechanics and Control", *Addison-Wesley*, 1955

Eberly, D. H., "3D Game Engine Design", *Morgan Kaufmann*, 2001

Fedor, M., "Application of Inverse Kinematics for Skeleton Manipulation in Real-time", *International Conference on Computer Graphics and Interactive Techniques*, p.203-212, 2003

Kwang-Jin, C., Hyeong-Seok, K., "On-line Motion Retargeting", *The Journal of Visualization and Computer Animation*, Vol. 11, p.223-235, 2000

Lee, J., Shin, S. Y., "A Hierarchical Approach to Interactive Motion

Editing for Human-Like Figures", *Siggraph 99*, p.39-48, 1999

Meredith, M., Maddock, S., "Real-Time Inverse Kinematics: The Return of the Jacobian", Technical Report No. CS-04-06, *Department of Computer Science, The University of Sheffield*, 2004

Paul, R. P., Shimano, B., Mayer, G. E., "Kinematic Control Equations for Simple Manipulators", *IEEE Transactions on System, Man & Cybernetics*, Vol. 11, No. 6, 1988

Ryckaert, J. P., Ciccotti, G., Berendsen, H. J. C., "Numerical Integration of the Cartesian Equations of Motions of a System with Constraints: Molecular Dynamics of n-Alkanes", *Journal of Computational Physics*, Vol. 23 p.327-341, 1977

Shin, H. J., Lee, J., Gleicher, M., Shin, S. Y., "Computer Puppetry: An Importance-Based Approach", *ACM Transactions On Graphics*, Vol. 20, No. 2, p.67-94, April 2001

Tang, W., Cavazza, M., Mountain, D., Earnshaw, R., "A Constrained Inverse Kinematics Technique for Real-time Motion Capture Animation", *The Visual Computer*, Vol. 15, p.413-425, 1999

Tolani, D., Badler, N. I., "Real-time Inverse Kinematics of the Human Arm", *Presence*, Vol. 5, No. 4, p.393-401, 1996

Tolani, D., Goswami, A., Badler, N., "Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs", *Graphics Models*, Vol. 62, No. 6, p.353-388, 2000

Wang, L., Chen, C., "A Combined Optimisation Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators", *IEEE Transactions on Robotics & Applications*, Vol. 7, No. 4, p.489-499, 1991

Welman, C., "Inverse kinematics and geometric constraints for articulated figure manipulation", Master of Science Thesis, *School of Computing Science, Simon Fraser University*, 1993

Watt, A., Watt, M., "Advanced animation and rendering techniques", *Addison-Wesley*, 1992

Zhao, J., Badler, N. I., "Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures", *ACM Transactions on Graphics*, Vol. 13, No. 4, p.313-336, 1994

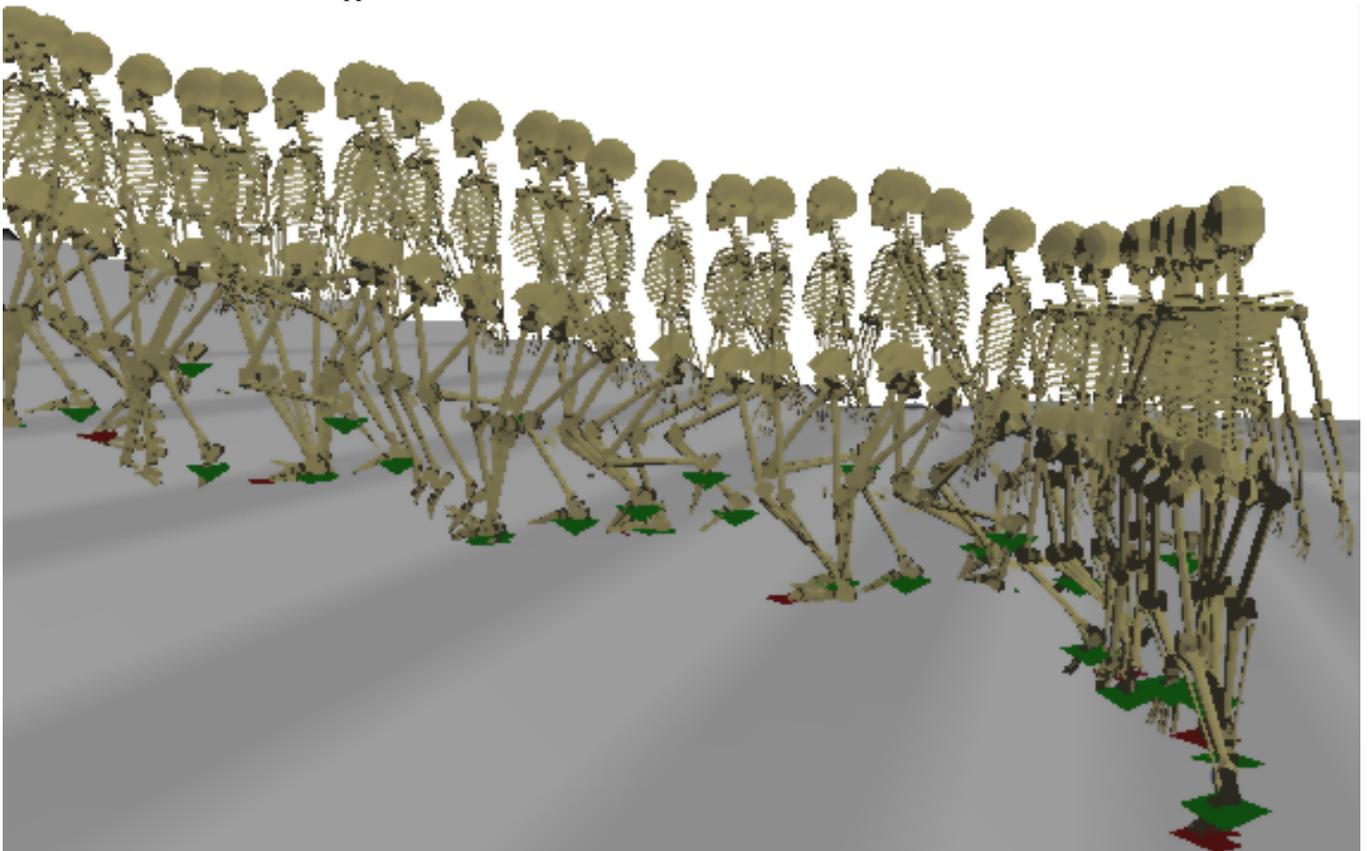


Figure 1.4: Analogue joystick-controlled real-time IK over uneven terrain; green pyramids represent the intended position of the leading foot while the red pyramids indicate desired location of the training toes.

SIMULATING ANIMATION BY REGULAR DEFORMATION USING OPENGL

A.Arokiasamy¹

David Al-Dabass²

Rajan Periyannayagam³

¹ Faculty of Engineering, Multimedia University, 63100, Cyberjaya, Malaysia

Tel: +603-83125386, Fax: +603-8318-3029 Email: a.arokiasamy@mmu.edu.my

² School of Computing and Technology, Nottingham Trent University, NG1 4BU, UK

³ Centre for Computer Graphics and Virtual Reality, Multimedia University, Cyberjaya, Malaysia

KEYWORDS

Animation, Regular deformation, tapering, twisting, bending, solid modeling

ABSTRACT

Images convey a lot of information because the human visual system is a sophisticated information processor. It follows, then, that moving images have the potential to convey much more information and hence animation is treated as an important factor in computer graphics. Shape modification can be a flexible means for controlling animation, for example, the tail of a dog wagging, the swimming motion of a fish, or even human figure animation. One of the most important problems of available solid modeling systems is that the range of shapes generated is limited. It is not easy to model an irregular object in a conventional solid modeling system. But it is easy to model irregular objects by deforming regular objects that in turn can be generated using parametric equations. Without deformation, to simulate an irregular object, one has to save every point on the object. Then it will become a tedious process. However, one can create a regular object and then apply deformation to it to create an irregular object with much less data. The main aim of this paper is to demonstrate how to achieve animation by regular deformation such as tapering, twisting, and bending. This work is done in OpenGL as it provides more flexibility to implement this work.

INTRODUCTION

Animation means “to bring to life”. By providing still images that change a number of times per second we can provide the illusion of movement, and thus of life. That is rapid display of slightly different images create the illusion of motion called animation. Animation is one of the most interesting and important part of computer graphics. The application of graphics is increased significantly by being able to animate objects.

Animation adds to graphics the dimension of time, which vastly increases the amount of information which can be transmitted. In order to animate something, the animator has to be able to specify, either directly or indirectly, how the ‘thing’ is to move

through time and space. The basic problem is to select or design animation tools that are expressive enough for the animator to specify what he/she wants to specify while at the same time are powerful or automatic enough that the animator doesn't have to specify the details that he/she is not interested in. Obviously, there is no one tool that is going to be right for every animator, for every animation, or even for every scene in a single animation. The appropriateness of a particular animation tool depends on the effect desired by the animator. An artistic piece of animation will probably require different tools than an animation intended to simulate reality.

The most popular type of animation is computer animation. The use of computer has brought about a new way of approaching animations. It started when computer experts saw the possibility of computers as a way of developing wonderfully good pictures that would have taken a lot more effort in producing or even impossible to produce otherwise. The use of computers also came in where a dangerous stunt needs to be done. Now, the simple flicks done to a computer can save so much more effort and resources by the use of computers, we now let it do the more menial work that used to be done by junior cartoonist. Before this, the chief cartoonist would draw out the key frames and others would draw the pages in between. Now, we can use the computers to create these scenes and do the in 'betweening' (or known as tweening) scenes. This saved the animators a lot of effort and time, which is much appreciated. At the same time one of the major problems in computer animation is that the computer can produce missing drawings based on extreme drawings produced by animators (Catmull 1978).

There are several different (general) types of animation available such as real time animation, keyframe animation, character animation, motion path animation, hierarchical animation, shape animation, procedural animation, simulation, camera animation etc. Animation adds a lot to a computer graphics application. Adding interactivity to animation gives the user control. Games, interactive visualizations and learning tools all provide users with the ability to control the presentation – driving a car, “flying” through a human body, etc.

OPENGL

OpenGL is a low level, real-time 3D graphics API. It is a software interface for applications to generate interactive 2D and 3D computer graphics. OpenGL was originally developed in 1992 by Silicon Graphics, as a descendant of an API known as Iris GL for Unix. It was created as an open standard and is available on many different platforms. That is where the word “Open” in OpenGL derived from. Meanwhile, GL stands for Graphics Library. OpenGL is designed to be independent of operating system, window system, and hardware operations and many vendors support it.

OpenGL is available on PCs, Macintosh and workstations. It provides a wide range of graphics functions from rendering a simple geometric point, line, or filled polygon, to texture mapping, NURBS, or curved surfaces. Developer driven advantages of OpenGL are its industry standard, stable, reliable, portable, evolving, scalable, easy to use and well-documented API. This API is designed to work efficiently even if the computer that displays the graphics is not the computer that runs the OpenGL program (Angel 2000, Angel 2002, Woo et al 1999, Wright1999). With any compiler, the programmer will be able to create any 3D graphics within imagination with the help of OpenGL libraries.

In OpenGL, animation is achieved by using variables for the parameters like position and rotation. The user or the program can then update these variables and when OpenGL redraws the screen, the image will be different.

DEFORMATION

One of the most important problems of available solid modeling systems is that the range of shapes generated is limited. It is not easy to model an irregular object in a conventional solid modeling system. Without deformation, to simulate an irregular object, one has to save every point on the object. Then it will become a tedious process. However, one can create a regular object and then apply deformation to it to create an irregular object with much less data. Deformations are easily combined in a hierarchical structure, creating complex objects from simpler ones. Deformations are important and highly intuitive operations that ease the control and rendering of large families of three dimensional geometric shapes. Deformations can be incorporated into traditional CAD/CAM solid modeling and surface patch methods, reducing the data storage requirements for simulating flexible geometric objects, such as objects made of metal, fabric or rubber.

In the real world changes of shape are very common. When a flower is brushed by a breath of air, the stem bends and the petals and leaves may also bend and change shape subtly. When a dog curls up and lies down on a rug, its body bends into a semicircular arc. The expressions of the human face are almost exclusively a matter of the face changing shape. Shape modification can be a flexible means for controlling animation, for example, the tail of a dog wagging, the swimming motion of

a fish, or even human figure animation. The first question, which needs to be answered, is 'what is shape'? Or 'what transformations change the shape of an object'?

We may be able to change an actual shape of the object to reform or deform the shape of the surface of the object. Although we might think that a change of scale is change of shape, but a change of scale does not constitute a change of shape in the sense meant here (O'Rourke 1995). Scaling an object makes it uniformly longer or shorter in one or more directions, but it does not alter the basic configuration of the surface. It does not alter the “bumps and hollows” of the surface (O'Rourke 1995). Shape deformation is a deformation between the outlines-either 2D or 3D.

Deformation, first introduced by (Barr 1984) is a highly intuitive and easily visualized set of operations. Deformations allow the user to treat a solid as if it were constructed from a special type of topological putty or clay, which may be bent, twisted, tapered, compressed, expanded, and otherwise transformed repeatedly into a final shape. They are highly intuitive and easily visualized operations that simulate some important manufacturing processes for fabricating objects, such as the bending of bar stock and sheet metal. Deformations can be incorporated into traditional CAD/CAM solid modeling and surface patch methods, reducing the data storage requirements for simulating flexible geometric objects, such as objects made of metal, fabric or rubber.

Without deformation, to simulate an irregular object, one has to save every point on the object. However, one can create a regular object and then apply deformation to it to create an irregular object with much less data (Gudukbay, et al 1990). It saves time and it will become an easy task too.

One long-term goal of computer graphics and numerical methods for three-dimensional design is a unified mathematical formalism. Such a unified mathematical formalism for geometric representation and computation provides a natural base for a geometric modeler of considerable versatility and robustness (Farouki and Hinds, 1985). The system uses quadric objects that are collections of smooth parametric objects producing a new spectrum of flexible forms. The chief advantage is that they allow complex solids and surfaces to be constructed and altered easily by changing a few interactive parameters. The quadrics family consists of sphere, ellipsoid, torus, hyperboloids etc. Quadrics can be defined by either nonparametric or parametric equations. Our implementation uses parametric equations to generate quadrics since generation of surface points is easier with this method. We may also deform super-quadric objects using regular deformation technique (Gudukbay et al 1990).

The deformation technique used in this implementation is regular deformation such as twisting, bending, tapering of geometric objects. Regular deformations are well defined and their results are straightforward but it lacks generality. Regular deformation is very fast when compared to the other

deformation techniques like Free-Form Deformation (FFD) [Barr 1984, Gudukbay 1990). A globally specified deformation of a three dimensional solid is a mathematical function \underline{F} which explicitly modifies the global coordinates of points in space. Mathematically, it can be represented by the equation $\underline{X} = \underline{F}(x)$ where \underline{x} represents the point in the un-deformed solid, and \underline{X} represents the points in the deformed solid (Barr 1984, Gudukbay 1990).

ANIMATION BY REGULAR DEFORMATION

Using regular deformation technique such as tapering, twisting and bending we may be able to animate the objects. Tapering, twisting and bending functions are clearly explained in (Barr 1984, Gudukbay 1990). Each of the regular deformation is explained in detail in the following section.

Scaling

One of the simplest deformations is a change in the length of the three global components parallel to the coordinate axes. This produces an orthogonal scaling operation:

$$\begin{aligned} X &= a_1x \\ Y &= a_2y \\ Z &= a_3z \end{aligned}$$

Global Tapering

Tapering is similar to scaling, by differentially changing the length of two global components without changing the length of the third. To do tapering operation along the z-axis one should choose a tapering function depending on the z-coordinates of the points. When the tapering function $f(z) = 1$, the portion of the deformed object is unchanged; the object increases in size as a function of z when $f'(z) > 0$ and decreases in size when $f'(z) < 0$. The object passes through a singularity at $f(z) = 0$ and becomes everted when $f(z) < 0$. Global tapering along the z-axis is given by the following equations

$$\begin{aligned} r &= f(z) \\ X &= rx \\ Y &= ry \\ Z &= z \end{aligned}$$

Global Axial Twists

For some applications, it is useful to simulate global twisting of an object. A twist can be approximated, as differential rotation

just as tapering is a differential scaling of the global basis vectors. We rotate one pair of global basis vectors as a function of height, without altering the third global basis vector. The deformation can be demonstrated by twisting a deck of cards, in which each card is rotated somewhat more than the card beneath it.

The global twist around the z-axis is produced by the following equations:

$$\begin{aligned} \theta &= f(z) \\ C_\theta &= \cos(\theta) \\ S_\theta &= \sin(\theta) \\ X &= xC_\theta - yS_\theta \\ Y &= xS_\theta + yC_\theta \\ Z &= z \end{aligned}$$

The twist proceeds along the z-axis at a rate of $f'(z)$ radians per unit length in the z direction.

Global Bending

Bending simulates an important manufacturing process for fabricating objects. An example of this operation is the bending of a bar stock or sheet metal. For other applications, it is useful to have a simple simulation of bending. To make a bending along the y-axis, one has to specify a bent region along the y-axis. The following equations represent an isotropic bend along a centerline parallel to the y-axis; the length of the centerline does not change during the bending process.

$$\begin{aligned} \theta &= k(\hat{y} - y_0) \\ C_\theta &= \cos(\theta) \\ S_\theta &= \sin(\theta) \end{aligned}$$

where

$$\hat{y} = \left\{ \begin{array}{ll} y_{\min}, & \text{if } y \leq y_{\min} \\ y, & \text{if } y_{\min} < y < y_{\max} \\ y_{\max}, & \text{if } y \geq y_{\max} \end{array} \right\}$$

The following relations give the formula for this type of bending along the y-axis centerline:

$X = x$

$$Y = \left\{ \begin{array}{ll} -S_{\theta} \left(z - \frac{1}{k} \right) + y_0, & y_{\min} \leq y \leq y_{\max} \\ -S_{\theta} \left(z - \frac{1}{k} \right) + y_0 + C_{\theta} (y - y_{\min}), & y < y_{\min} \\ -S_{\theta} \left(z - \frac{1}{k} \right) + y_0 + C_{\theta} (y - y_{\max}), & y > y_{\max} \end{array} \right\}$$

$$Z = \left\{ \begin{array}{ll} C_{\theta} \left(z - \frac{1}{k} \right) + \frac{1}{k}, & y_{\min} \leq y \leq y_{\max} \\ -C_{\theta} \left(z - \frac{1}{k} \right) + \frac{1}{k} + S_{\theta} (y - y_{\min}), & y < y_{\min} \\ -C_{\theta} \left(z - \frac{1}{k} \right) + \frac{1}{k} + S_{\theta} (y - y_{\max}), & y > y_{\max} \end{array} \right\}$$

The bending angle θ , is constant at the extremities, but changes linearly in the central region. In the bent region, the bending rate k , measured in radians per unit length, is constant, and the differential basis vectors are simultaneously rotated and translated around the third local basis vector. Outside the bent region, the deformation consists of a rigid body rotation and translation. The range of the bending deformation is controlled by y_{\min} and y_{\max} with the bent region corresponding to values of y such that $y_{\min} < y < y_{\max}$. The bending axis is located along $[s, y_0, 1/k]^T$, where s is the parameter of the line. The center of the bend occurs at $y = y_0$ i.e., where one would "put one's thumbs" to create the bend. The radius of curvature of the bend is $1/k$.

These functions have continuous values at the boundaries of each of the three regions for y , and in the limit, for $k = 0$. However, there is a jump in the derivative of the bending angle θ at the $y = y_{\min}$ and $y = y_{\max}$ boundaries. The discontinuities may be eliminated by using a smooth function for θ as a function of y .

Regular deformation explained above can be combined with rotation around some axis so that these operations can be performed around other axes. The mathematical details of the regular deformations can be found in (Barr 1984, Gudukbay 1990). By combining these above deformations we can get unimaginable beautiful outputs that are listed below.

By varying the different regular deformation functions we can animate the geometric objects by varying the different scaling values to these deformation functions. This work is done in OpenGL because it provides two different modes for animation. Especially the double buffer mode is specially meant for smoother and flicker free animation. In our program first we select the type of regular deformation in which the object is to be deformed and animated using the menu option included in the program. Then we have to fix one of the deformation functions that is simple, sine, cosine and tan function options are provided and then the object is animated. OpenGL also

reduces over work by providing lot of built in commands like menu facility.

CONCLUSION

Computer animation techniques have achieved a very high level of physical realism and nowadays provide a wide character control capability. Despite the high quality of the resulting animations, their creation still requires the intervention of good programmers and graphic designers to build a custom character controller or to manually generate the character movements and behaviors.

Thus in this paper we have simulated animation with the help of regular deformation techniques. Some the snapshots of this program are also added in the appendix. We may also able to change the program for different twisting and bending functions. We can also animate other quadric objects like sphere, ellipsoid, hyperboloid etc. Real time applications of this work are it can be used in cartoon film making, game design, education and training, presentation graphics.

REFERENCES

- Barr, A H, 1984, Global and local deformation of solid primitives, *Computer Graphics* 18(3), 21-30, July 1984.
- Angel, 2000, *Interactive Computer Graphics - A Top-Down Approach with OpenGL* (Third Edition, Addison – Wesley, 2000).
- Angel, 2002, *OpenGL - A premier* (Addison – Wesley, 2002).
- Catmull, E, 1978, The Problems of Computer-Assisted Animation, *SIGGRAPH' 78*, pp.348-353.
- O'Rourke, M, 1995, *Principles of Three-Dimensional Computer Animation* (W.W. Norton and Company, New York, 1995).
- Farouki, R T, and J. K. Hinds, 1985, A hierarchy of geometric forms , *IEEE Computer Graphics and Applications* 5(5), 51-78, May 1985.
- Gudukbay, U, and Bulent Ozguc, 1990, Free-Form solid modeling using deformation, *Computer Graphics*, Vol. 14, Nos. 3/4, pp491-500, 1990.
- Woo, Neider, Davis, Shreiner, 1999, *OpenGL Programming Guide*(Third Edition, Addison – Wesley, 1999).
- Wright, Sweet, 1999, *OpenGL SuperBible* (Second Edition, Waite Group Press, 1999).

APPENDIX: Snapshots

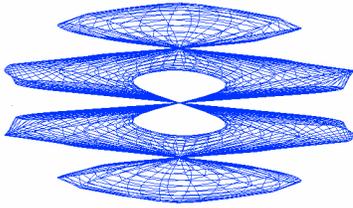


Fig.1
Tapering a torus using a sine function

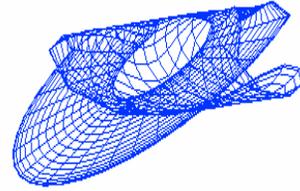


Fig.4
Bending the torus along y-axis using a bending function

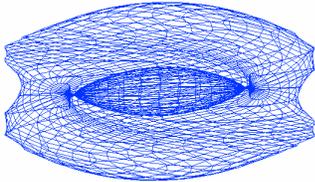


Fig.2
Twisting a torus along y-axis using a cosine function

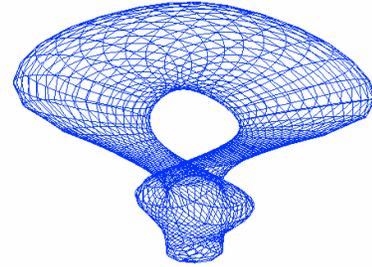


Fig.5
Tapered and twisted torus along y-axis using a sine function

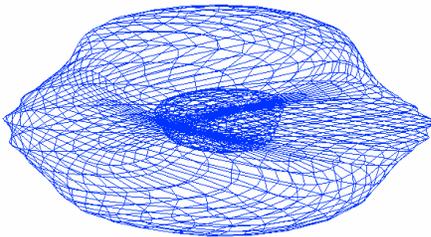


Fig.3
Twisting a torus along y-axis using a sine function

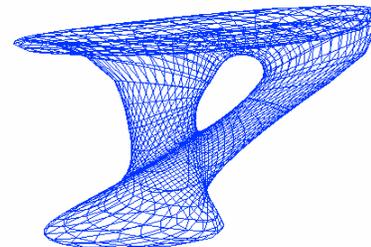


Fig.6
Tapered and bent torus along y-axis using a tan function

Games Art, Design, Modelling and Development

| | |
|--|------------|
| Bangsø, O., Jensen, O. G., Jensen, F. V., Andersen, P. B. and Kocka, T. Non-linear interactive storytelling using object-oriented Bayesian networks | 96 |
| Fairclough, C.R. and Cunningham, P. AI structuralist storytelling in computer games | 104 |
| Natkin, S., Vega, L. and Grünvogel, S. M. A new methodology for spatiotemporal game design | 109 |
| Röber, N. and Masuch, M. Auditory game authoring from virtual worlds to auditory environments | 114 |
| Zeng, X., Mehdi, Q. H. and Gough, N. E. Implementation of VRML and Java for Story Visualization Tasks | 122 |

NON-LINEAR INTERACTIVE STORYTELLING USING OBJECT-ORIENTED BAYESIAN NETWORKS

Olav Bangsø¹, Ole G. Jensen¹, Finn V. Jensen¹, Peter B. Andersen², and Tomas Kocka³

¹Department of Computer Science, Aalborg University,
Frederik Bajers Vej 7E, DK-9220 Aalborg Øst, Denmark,
E-mail: <bangsy | guttorm | fvj>@cs.aau.dk

²Department of Information and Media Studies, University of Aarhus,
Helsingforsgade 14, DK-8200 Aarhus N, Denmark,
E-mail: pba@imv.au.dk

³ADASTRA, s.r.o.,
Beneovská 10, 101 00 Praha 10, Czech Republic,
E-mail: tomas.kocka@adastracorp.com

KEYWORDS

Interactive narrative, interactive drama, object-oriented Bayesian networks.

ABSTRACT

Narration and interaction are often viewed as contrary properties in computer games. Games with a high degree of interaction fail to provide a coherent narration and the player's interaction seldom has any direct impact on the narrative. Games with a high degree of narration often tell a linear story similar to books or movies with little room for the player to interact. We propose *non-linear interactive storytelling* (NOLIST) as a first step towards developing games with a high degree of interaction and a coherent narrative. The main idea is that the narrative is not fixed from the beginning but instead constructed as the game progresses based on the player's interaction. We provide a simple model that allows writers to specify a NOLIST as a set of *actions* which the game engine then combines to create the narrative. Finally, we propose to develop a game engine using Bayesian networks to model the probability of the possible narratives that can be created from the actions, and use this knowledge to create better narratives.

INTRODUCTION

Narration and interaction are often considered as two incompatible properties in computer games. This has led to the distinction between games of progression and games of emergence [Juil, 2004]. Games of progression have been compared to movies that stop at certain points to allow the player to choose among a set of options which determine how the narrative progresses from that point on. In these

games, narration is highly linear with limited player interaction, so playing the game more than once rarely provides the player with a new gaming experience. Games of emergence define a set of simple and deliberate rules which when combined emerge into more complex patterns, and thus motivate the player to develop more advanced strategies for playing the game. The high level of player interaction is achieved at the expense of a coherent narrative. Players rarely experience events later in the game as direct consequences of earlier events, and much of the interaction has no impact on the narrative [Mallon and Webb, 2000].

This paper proposes *non-linear interactive storytelling* (NOLIST) as a first step towards the development of games with both a high degree of player interaction and a coherent narrative. The main idea is that the narrative is not fixed from the outset, but instead constructed as the game progresses. The outcome of events that occur in the game whether caused by the player's direct interaction or by agents in the game world change the likelihood of past events (not observed by the player) having occurred and the probability that certain future events can occur. At any point in the game, the narrative consists of the events observed by the player. These events determine the probability and possibility of different pasts and futures for the narrative. The player's interaction is restricted to events that are consistent with the possible pasts and futures in order to ensure a coherent narrative. As more events are observed by the player, the set of possible pasts and futures narrows. Consequently, the player's choices become more and more restricted as the game progresses until all the events of the narrative are determined.

We propose a framework to develop NOLIST in computer games. A NOLIST game engine specifies a set of actions which are the building blocks of the narrative. An

action could, e.g., be a small movie clip, a sentence in a dialog, or the description of an event. The game engine maintains a model of the possible pasts and the possible futures for the narrative. The game is played in rounds, where in each round the game engine first determines which actions are possible. These actions must be part of a possible future and be consistent with a possible past. The player then chooses one of them. The chosen action is played out in the game and then part of the narrative. The chosen action influences the possible pasts and futures for the narrative. E.g., in a murder story the investigator (the player) finds a smoking gun close to the spot where someone was killed. This action influences the past by making it more probable that the gun was used to kill the victim and less probable that the victim was stabbed to death. In turn, all suspects with access to the gun are more likely to be the murderer. Changes to the possible pasts influence the possible futures. E.g., the suspects with access to the gun are now more likely to try to conceal their actions and mislead the investigator. The action also influences the possible futures directly. E.g., it is now more probable that a bullet will be found in the victim's body if it is examined. When the game engine has determined how the chosen action influences the possible pasts and futures, a new round can commence. Some actions are possible endings. The game can only end immediately after the game engine has played out such an action, however, this is not mandatory.

NOLIST is related to interactive drama. The narrative engine, IDtension, calculates the set of all possible actions of the characters based on the current state in the world of the story and ranks them according to a user model for their narrative effects [Szilas, 2003]. The NOLIST game engine ensures consistency between past events observed by the user and future events as well as prevents stories with no endings. The narrative quality of stories is not considered in this paper, however, with NOLIST it is possible to evaluate the narrative quality based on the probability of the possible futures of each action. In Facade interactive dramas are divided into beats and semi-autonomous agents with a drama manager are used to choose among the possible beats [McKee, 1997, Mateas and Stern, 2003]. Each beat has a tension value and beats are chosen to best fit the Aristotelian story tension value arc. It is the responsibility of the author to ensure that all states have possible beats. In NOLIST atomic actions resemble beats, and the engine ensures that the story never reach a state where no actions are possible.

NON-LINEAR INTERACTIVE STORY-TELLING (NOLIST)

A story is usually divided into a number of smaller parts, we use the term chapter, but any subdivision is valid and supported. The chapters are supposed to be read in sequence, so the first chapter begins the story while the last chapter ends it. Before reading each chapter, the reader is required to know about certain characters and events in the story (those described in all chapters preceding this chap-

ter). Reading the chapters in any other order often makes the story confusing and incoherent. We say that such a story is linear because all the chapters must be read in a specific order. In contrast, a non-linear story allows chapters to be read in different orders and not all chapters have to be read. We cannot expect a coherent narrative to result from any random order of chapters, so each chapter has some prerequisites that must be satisfied by the preceding chapters. E.g., to identify the murderer, a motive and an opportunity must have been established. However, the details of what the motive and opportunity are or how they were established are not relevant and can be established by different sequences of preceding chapters. Therefore, with much fewer chapters a non-linear story can represent many different linear stories. In a non-linear interactive story the reader can influence the order of chapters.

We introduce actions as the building blocks of a non-linear interactive story. Some actions are possible endings (denoted by a suffix '**'). An *action* is characterized by its content, prerequisites, and effects. The *content* encapsulates a set of actions, analogical to a chapter being divided into sub-chapters. If the content is empty, then the action is *atomic*. The *prerequisites* are the events that must have occurred before the action can be performed and the *effects* are the events that occur as a result of performing the action. An *event* is a simple statement such as *the gun is the murder weapon* or *if John has a motive it is not known*. The content of an action may encapsulate actions which in turn encapsulate other actions. This leads to an *action hierarchy* such as the one illustrated in Figure 1. In the figure the action *Examine the crime scene* encapsulates the two actions *X found at crime scene* and *Y found at crime scene*, both of which are atomic actions (since their content is empty) and possible endings (denoted by the '**').

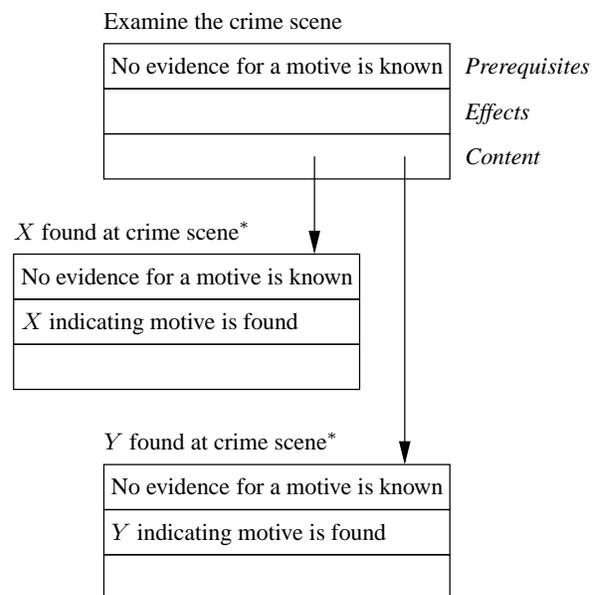


Figure 1: Action hierarchy for *Examine the crime scene*.

A *non-linear interactive story* consists of an action and a past. The *past* is a set of events that is believed to hold at a particular point in the narrative, and only actions whose prerequisites are satisfied by the past can be performed. Whenever an action is performed, the past is changed to reflect the effects of that action.

An Example of a Non-Linear Interactive Story

To help illustrate non-linear interactive stories and actions we provide an example. The example is a murder story where the player takes the role of an investigator trying to solve a murder case by investigating the crime scene and interviewing the three suspects named *A*, *B*, and *C*. Figure 2 depicts the action hierarchy for the murder story. At the top level (1) is the Murder story action with a content of five actions: Investigate the crime scene (1.1), Interview *A*, *B*, and *C* (1.2-1.4), and Reveal the murderer (1.5). Actions at the lowest level are atomic.

The prerequisites and effects of the atomic actions are in the appendix. The effects of all non-atomic actions in the murder story are empty, and the prerequisites of any non-atomic action is the disjunction of the prerequisites of the actions in its content.

When the game begins, the past is initialized as follows:

| |
|--|
| No evidence for a motive is known No evidence for an opportunity is known It is not known who found the victim It is not known if <i>A</i> had a motive It is not known if <i>B</i> had a motive It is not known if <i>C</i> had a motive It is not known if <i>A</i> had an opportunity It is not known if <i>B</i> had an opportunity It is not known if <i>C</i> had an opportunity |
|--|

In order to identify the murderer, the investigator must establish a strong motive and a clear opportunity for at least one of the suspects. Note that at this point, the game has not decided who actually murdered the victim. This will be determined as the game progresses based on the player's interaction with the game.

A Game Example using the Murder NOLIST

In this section we describe an example gaming session using the murder story described in the previous section. When using a NOLIST for a game, some actions are chosen by the player or others by the game engine. In this example, all non-atomic actions are chosen by the player while the game engine chooses among the atomic actions. Table 1 summarizes the gaming session.

The game begins at the top level of the action hierarchy, where only the Murder story action is available. To perform this action, we must perform actions in its content whose prerequisites are satisfied by the (initial) past until a possible ending is performed. All actions at this level of the action hierarchy are available except 1.5, and the player chooses 1.1. Again, the player must choose among the actions in the content of action 1.1, and chooses 1.1.1.

| | |
|---------|---|
| 1 | Murder story* |
| 1.1 | Investigate the crime scene |
| 1.1.1 | Examine the victim* |
| 1.1.1.1 | <i>M</i> found on victim* |
| 1.1.1.2 | <i>N</i> found on victim* |
| 1.1.2 | Examine the crime scene* |
| 1.1.2.1 | <i>X</i> found at crime scene* |
| 1.1.2.2 | <i>Y</i> found at crime scene* |
| 1.1.3 | Ask who found the victim* |
| 1.1.3.1 | <i>A</i> found the victim* |
| 1.1.3.2 | <i>B</i> found the victim* |
| 1.1.3.3 | <i>C</i> found the victim* |
| 1.2 | Interview <i>A</i> |
| 1.2.1 | Ask <i>A</i> about <i>B</i> * |
| 1.2.1.1 | <i>A</i> indicates weak motive for <i>B</i> * |
| 1.2.1.2 | <i>A</i> indicates vague opportunity for <i>B</i> * |
| 1.2.1.3 | <i>A</i> and <i>B</i> have an alibi* |
| 1.2.2 | Ask <i>A</i> about <i>C</i> * |
| 1.2.2.1 | <i>A</i> indicates strong motive for <i>C</i> * |
| 1.2.2.2 | <i>A</i> indicates vague opportunity for <i>C</i> * |
| 1.3 | Interview <i>B</i> |
| 1.3.1 | Ask <i>B</i> about <i>A</i> * |
| 1.3.1.1 | <i>B</i> indicates weak motive for <i>A</i> * |
| 1.3.1.2 | <i>B</i> indicates clear opportunity for <i>A</i> * |
| 1.3.2 | Present evidence to <i>B</i> * |
| 1.3.2.1 | <i>B</i> acknowledges clear opportunity* |
| 1.3.3 | Ask <i>B</i> about <i>C</i> * |
| 1.3.3.1 | <i>B</i> indicates weak motive for <i>C</i> * |
| 1.3.3.2 | <i>B</i> indicates clear opportunity for <i>C</i> * |
| 1.3.3.3 | <i>B</i> and <i>C</i> have an alibi* |
| 1.4 | Interview <i>C</i> |
| 1.4.1 | Ask <i>C</i> about <i>A</i> * |
| 1.4.1.1 | <i>C</i> provides evidence indicating motive* |
| 1.4.1.2 | <i>C</i> indicates strong motive for <i>A</i> * |
| 1.4.1.3 | <i>C</i> indicates vague opportunity for <i>A</i> * |
| 1.4.2 | Ask <i>C</i> about <i>B</i> * |
| 1.4.2.1 | <i>C</i> indicates weak motive for <i>B</i> * |
| 1.4.2.2 | <i>C</i> indicates clear opportunity for <i>B</i> * |
| 1.5 | Reveal the murderer* |
| 1.5.1 | <i>A</i> is the murderer* |
| 1.5.2 | <i>B</i> is the murderer* |
| 1.5.3 | <i>C</i> is the murderer* |

Figure 2: Action hierarchy for the murder story. Actions marked with * are possible endings.

| | |
|------------|---|
| P: 1 | Murder story |
| P: 1.1 | Investigate the crime scene |
| P: 1.1.1 | Examine the victim |
| G: 1.1.1.2 | <i>N</i> found on victim |
| P: 1.1.2 | Examine the crime scene |
| G: 1.1.2.1 | <i>X</i> found at crime scene |
| P: 1.1.3 | Ask who found the victim |
| G: 1.1.3.1 | <i>A</i> found the victim |
| P: 1.2 | Interview <i>A</i> |
| P: 1.2.2 | Ask <i>A</i> about <i>C</i> |
| G: 1.2.2.2 | <i>A</i> indicates vague opportunity for <i>C</i> |
| G: 1.2.2.1 | <i>A</i> indicates strong motive for <i>C</i> |
| P: 1.3 | Interview <i>B</i> |
| P: 1.3.3 | Ask <i>B</i> about <i>C</i> |
| G: 1.3.3.3 | <i>B</i> and <i>C</i> have an alibi |
| P: 1.3.1 | Ask <i>B</i> about <i>A</i> |
| G: 1.3.1.1 | <i>B</i> indicates weak motive for <i>A</i> |
| G: 1.3.1.2 | <i>B</i> indicates clear opportunity for <i>A</i> |
| P: 1.4 | Interview <i>C</i> |
| P: 1.4.1 | Ask <i>C</i> about <i>A</i> |
| G: 1.4.1.2 | <i>C</i> indicates strong motive for <i>A</i> |
| P: 1.5 | Reveal the murderer |
| G: 1.5.1 | <i>A</i> is the murderer |

Table 1: An example gaming session of the murder story.

The contents of action 1.1.1 contains two atomic actions both with satisfied prerequisites. The game engine chooses among atomic actions and performs 1.1.1.2. The action is performed immediately (since it is atomic) and its effects update our belief in which events occurred in the past by replacing the statement *No evidence for an opportunity is known* with *N indicating opportunity is found*. Since 1.1.1.2 is a possible ending, the enclosed action (1.1.1) can be completed. Action 1.1.1 is also a possible ending, so the player may choose to complete action 1.1 as well. However, the player chooses to complete the two remaining actions in its content instead before completing action 1.1. At this point, the player chooses to interview *A* and establish a vague opportunity for *C* to commit the murder and a strong motive. The player now questions *B* about *C* hoping to establish a clear opportunity for *C* and thus reveal *C* as the murderer (action 1.3.3). The game engine has two atomic actions with satisfied prerequisites to choose from: 1.3.3.2 and 1.3.3.3. Either a clear opportunity is established for *C* or *B* provides an alibi for both *B* and *C*. In the example, the game engine selects the latter atomic action. Consequently, neither *B* nor *C* are likely suspects, and the player starts to question them about *A* (who apparently tried to frame *C*). This line of questioning leads to a strong motive and a clear opportunity for *A*, and *A* is finally revealed as the murderer by action 1.5.1. This action is a possible ending and action 1.5 can now be completed. Action 1.5 is also a possible ending, so the top level action can finally be completed, and the game ends.

In the example we did not specify *how* the game engine chooses which atomic story part to play next. Satisfaction of the prerequisites of an atomic story part is not sufficient to ensure a coherent and interesting story. E.g., in the mur-

der story it is entirely possible for more than one suspect to be the murderer or for all suspects to have an alibi. In the latter case, the prerequisites for the chapter marked as an ending will never be satisfied, so the story continues indefinitely. In the former case, only one of the suspects will be revealed before the story ends. To avoid such inconsistent stories, the game engine has to consider the possible futures for each of the atomic story parts to be played next. Only atomic story parts with acceptable futures and satisfied prerequisites can be played next. E.g., in the murder story, atomic story parts leading to everyone having an alibi or more than one suspect having a clear opportunity and a strong motive will never be played. As a consequence some actions may become unavailable to the player although they are associated with atomic story parts with satisfied prerequisites.

By evaluating the stories produced by the possible futures for an atomic story part, the game engine can avoid the least interesting stories. E.g., in round six of the murder story the game engine could have selected the atomic story part with the narrative: *B* indicates clear opportunity for *C* if evidence is known. This leads to *C* being the murderer. However, since all evidence points towards *C* from the beginning, the story is a trivial detective story. Clearly, what constitutes an interesting story varies a lot between genres. However, we believe that writers often have a fairly clear idea about how their own story should develop and that this can be modeled, at least in part, by considering how the past develops.

In the example we have only considered which atomic story parts are possible after each round of the game. In general, we would also like to know how probable they are. A story where all atomic story parts played are highly improbable might be fun to read but can easily become confusing and incoherent. Similarly, playing only the most probable atomic story parts produces a predictable and most likely boring story. Knowing the probability of each possible future helps the game engine to progress the story according to the stated intentions of the writer.

In the next section we introduce Bayesian networks for as they seem fit for modeling a NOLIST game engine that considers the probability of possible futures and avoids the pitfalls of stories without endings.

BAYESIAN NETWORKS

A Bayesian network is a graphical structure, which is used to represent cause-effect relations in a domain ([Pearl, 1988] and [Jensen, 2001]). In particular, they are widely used in domains with uncertainty attached to the impact of a cause. For example, if a investigator asks a suspect where he was at the time of the crime, the fact whether the suspect is guilty of the crime has a causal impact on the answer. However, even if the suspect was not at the crime scene at the time of the crime (and not guilty), you cannot be sure that he will tell the truth.

A Bayesian network consists of a structural part and a quantitative part. The structural part is a directed acyclic

graph (DAG), where nodes represent particular events, and the directed links represent cause-effect relations. A node has a finite set of states representing possibilities for this event. In Figure 3 the situation above is represented.

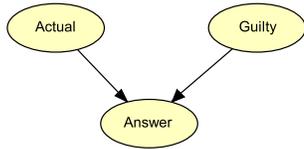


Figure 3: A DAG representing the situation where a investigator asks a suspect of his whereabouts at the time of the crime.

The node *Actual* represents the suspect’s possible whereabouts, and it may have the states *home*, *crime scene*, *mistress*, and *work*. The node *Guilty* has the states *yes* and *no*, and the node *Answer* has the same states as *Actual* plus the state *no answer*. We refer to the relations in a DAG using family terms. For example, *Actual* is a parent of *Answer*, and *Answer* is a child of *Guilty*.

The strength of the cause-effect relations is represented through conditional probabilities. For each node you have to specify a probability distribution for its states given all possible configurations of its parent nodes. For the model in Figure 3 you for example specify the probability distribution for *Answer* given *Actual*= *home* and *Guilty*=*no*. This could be

$$P(\textit{Answer} | \textit{Actual} = \textit{home}, \textit{Guilty} = \textit{no}) = (0.1, 0.1, 0.2, 0.3, 0.3)$$

You have to specify eight distributions of that kind. Furthermore, you also have to specify (prior) distributions for *Actual* and *Guilty*.

When Bayesian networks are used they are more complicated than the example in Figure 3. Figure 6 shows a slightly more complex model.

Basically, Bayesian networks are used to determine new probabilities given evidence. For example, when you know the answer of the question it is inserted in the network and used to determine the posterior probabilities for *Actual* and *Guilty*.

Object-Oriented Bayesian Networks

We utilize an extension to Bayesian networks called Object Oriented Bayesian Networks (OOBN) [Koller and Pfeffer, 1997, Bangsø and Wuillemin, 2000]. In the object oriented paradigm the basic component is an object; an entity with identity, state and behavior. Objects are grouped into classes. A class which is a description of a set of objects with the same structure, behavior and attributes. Whenever an object of a class is needed, an instance of that class is created. Note that each instance has a unique encapsulating class, the class in which they are instantiated.

A class is a Bayesian network fragment containing three sets of nodes:

- *O*: the set of output nodes; nodes that can be parents of nodes outside instances of the class.
- *I*: the set of input nodes; represents nodes that are not in the class, but are used as parents of nodes inside instances of the class. Input nodes cannot have parents in the class.
- *P*: the set of protected nodes; nodes that can only have parents and children inside the class.

The input and output nodes constitute the *interface*; the part of instances of a class that interfaces with the surroundings of the instance.

When an instance of a class is created, it can be linked to the rest of the network through the interface. To be able to do this linking a new type of link needs to be defined; the reference link. The child node in a reference link must be an input node and the parent is the node which is used as parent of the children of the input node, the parent and child must have the same number of states.

To allow the presence of input nodes without a parent, a default potential is introduced; a probability distribution over the states of the input node, used to create a regular node if the input node is not a child in a reference link. It is worth noting that the interface nodes are part of both the instance where they are defined and the class encapsulating that instance. This means that links from output nodes of an instance to nodes not in that instance (be it nodes in the encapsulating class or input nodes of other instances in the encapsulating class) are part of the specification for the encapsulating class.

By construction, instances of classes are inside a unique encapsulating class, ensuring that a tree of instances can be constructed. We will call such a tree an Instance Tree (IT). This tree will have the encapsulating class as the root, and each instance inside this class will define a sub tree with that instance as the root, and so on.

CREATING AN OOBN FOR A NOLIST

The specification of a NOLIST can be used to make a translation of the story into an OOBN. This OOBN will be used to choose actions for the game engine. The translation can be done automatically by letting :

- Prerequisites and effects for actions be variables.
- Actions be classes where the prerequisites are input nodes and the effects are output nodes.
- The action hierarchy be reflected in the IT.

In the following the translation will be outlined, and some problems and their solution described.

Atomic Actions

A class for an atomic action will consist of the prerequisites as input nodes and the effects as output nodes.

A class for one of the atomic actions under the “Examine the crime scene” action can be seen in Figure 4. e_m is the event modeling what evidence on motive has been found, it has the states *no evidence*, *M*, and *N*. This event is both a prerequisite and an effect, so it is present as both an input node (the name is prefixed with *i_*) and an output node (prefixed with *o_*. We need another event stating whether the prerequisites are fulfilled or not, modeled by the node *Prereq*. As we will show later, this will be used by the encapsulating action class to determine if the atomic action can be performed or not, so this must be an output of the atomic action class.

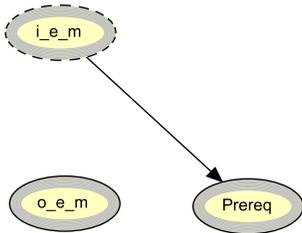


Figure 4: A class for the atomic action “M found on the victim” The special event output node *Prereq* has two states; *yes* and *no* for implying that the prerequisites for the atomic story are fulfilled or not. i_{e_m} is the input prerequisite event and o_{e_m} is the effect of the atomic story, and it is an output node.

Non-Atomic Actions

The representation of Non-atomic actions must ensure that any possible action it contains (called sub-actions) can be performed and also that sub-actions are performed until an end action has been performed. When an end sub-action has been performed, sub-actions can still be performed, as long as the last sub-action performed in the action is an end action. To handle this we make a representation of an action that will perform one of the possible sub-actions and make several instances of it to perform several of the sub-actions. In the murder story all atomic actions are marked as endings of the action they are in, but several atomic actions can be performed in sequence, e.g. P4 “Ask *A* about *C*”, where *A* first indicates a vague opportunity for *C* and then indicates a strong motive for *C*.

A Non-atomic action class has to make sure that one and only of its sub-actions is performed in each instance and that only possible sub-actions can be performed. This is done by using an instance of each sub-action class, i.e., the class representing the sub-action, an instance of the constraint class in Figure 5 for each sub-action. Evidence will be entered in the *Ok* nodes to make sure that only legal sub-actions can be performed. To ensure that at most one sub-action is performed, a variable with a state for each

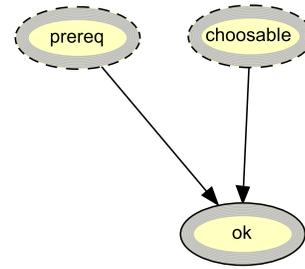


Figure 5: The constraint class that is instantiated once for each sub-action of an action

sub-action and one for nothing happening is created, and a variable for each sub-action monitoring if that sub-action can be performed created as a child of this. *Sc* is the variable describing possible sub-actions, each *Choosable* variable monitors if a sub-action is possible, they are *true*, if the corresponding sub-actions is possible. Note that *Sc* has *Sc_old* as a parent, this is to ensure that if nothing happens in an action instance, nothing will also happen in subsequent action instances, of the current action. Also, *nothing happening* can only occur if the last sub-action performed was an end sub-action or *nothing happening*. Making sure that an action ends with an end sub-action is done by introducing the variable *End* with two states, *y* and *n* as a child of *Sc* and letting the conditional probability table be constructed so *End* is in the state *y* if the sub-action performed in *Sc* is an end sub-action and *n* otherwise, and entering evidence on *y* in the last instance of the action class. Note that if *Sc* is in the state *nothing happening*, the last sub-action actually performed was an end sub-action, so *nothing happening* is also an end. In Figure 6 the OOBN unfolded to a Bayesian network for an action with two sub-actions, and one event, e_m , is shown, evidence will be entered in the two *Ok* variables to ensure that the *choosable* variables are only *true* if the prerequisites for the corresponding sub-action are met.

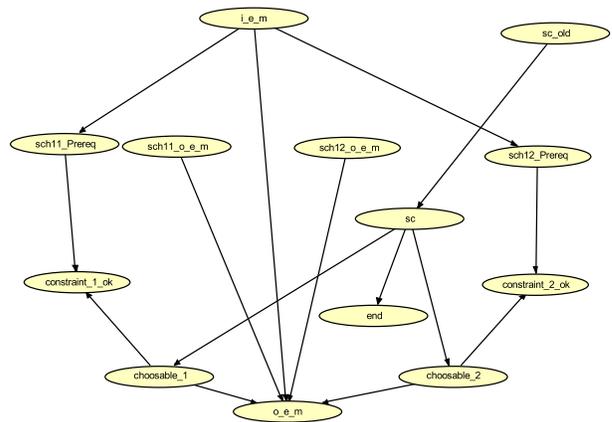


Figure 6: The part of an action class making sure that at most one sub-action is performed, and that only possible sub-actions can be performed. Note that the two atomic actions are called *Sch1.1* and *Sch1.2*.

Inputs and Outputs of an Action Class

All that is left for the action class is to make sure that the appropriate variables are available for the sub-actions, and that the past is updated according to the actions performed. This is done by having the union of inputs and outputs of the sub-actions as input to the action class, and the union of the outputs of these as output. The input of the action class ensures that any information that may be needed by a sub-action is available. The output of an action class ensures that any changes to events is available outside the action class instances. As only one sub-actions are performed in each instance, some of the events that can be changed may not necessarily be changed. In order for the action class to have the correct distributions in its outputs we need to know what these events are before the sub-action is performed. In Figure 7 a chapter class can be seen. Each output has the corresponding input as a parent to make sure that if the sub-action performed does not change the event, the events is not changed. Furthermore each output has the choosable variables corresponding to the sub-actions where they may be changed as parents, and the appropriate output of those sub-action instances are also parents. We have used parent divorcing to simplify the conditional probability tables, but as an example look at o_{e_m} , the output of the action class for the event e_m . It can be changed in sub-action 1.1 and 1.2 ($Sch1_1$ and $Sch1_2$), so $choosable_1$ and $choosable_2$ are parents along with the e_m outputs of these instances. The conditional probability table of o_{e_m} is the same as i_{e_m} if none of the $choosable$ variables are true, otherwise it will be the same as the e_m variable of the instance corresponding to the one that is. The structure and the evidence on the Ok variables ensures that at most on $choosable$ variable can be true in each instance of the action class. Note that the action class should be instantiated several times where each output is is referenced by the corresponding inputs in the next instance.

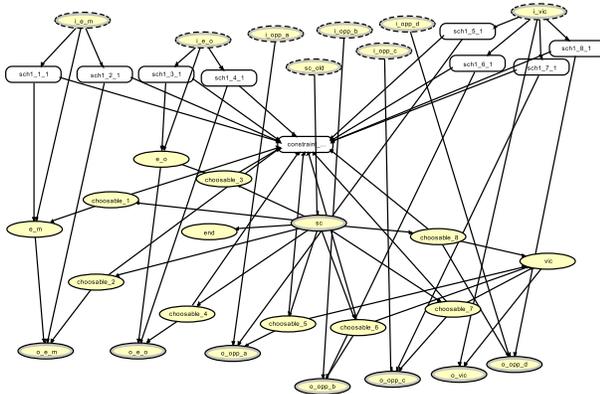


Figure 7: An action class for an action with eight sub-actions. The instances have not been expanded for overview.

Note that the sub-actions may themselves be non-atomic actions, and that the overall story will also be an action.

An OOBN for the example can be generated automatically from the specification. We will call an OOBN generated in this way a NOOBN.

FUTURE RESEARCH DIRECTIONS

The work presented in this paper is still preliminary and subject to ongoing research. Given an NOOBN, we envision a game engine that, whenever it has to choose an action, ensures that the game generated will have an ending, regardless of the players actions. It furthermore has to ensure that the possible continuations of the game will be interesting, and that two games will not be identical even if the player always performs the same actions. There are three problems in constructing this game engine:

1. Ensuring diversity of games.
2. Guaranteeing that the last action of the game is marked as an ending of the game.
3. Generating interesting games.

Sampling the Actions

Whenever the game engine chooses an action it will start by entering evidence on the actions already performed in the Sc nodes of the NOOBN. We will then sample possible continuations of the game to make it likely that we get different games, even if everything so far has been the same. Sampling is done by starting with the first part and hence the first Sc variable that hasn't received evidence in this yet. There are different ways this sampling can be done, one is to incorporate the probability distribution in the variable by weighting the possible states with their probability. We will then sample the next Sc variable given the previous actions. This however requires a propagation of the network as we have evidence both before this point of the game (the previous actions) and later (we want all actions to end with an action that is marked as an ending. We will continue to sample Sc variables as long as we have not reached the end of the overall action. We have reached the end if there are no more unsampled Sc nodes. All the Sc nodes will now be a complete game, with an ending. Note that we are also sampling the player actions.

Rating the Game

When choosing the next action, we want to ensure that it leads to interesting games. This requires a method of scoring a game.

A simplistic approach is to associate a score value with each atomic action. The measure the value of a story we can either accumulate the values or compare the values to some distribution (such as the tension value arc in [Mateas and Stern, 2003]. Such a simplistic approach will not capture all the intricacies of a good game, e.g. in a murder story it is usually good form to let the evidence point

strongly at one of the subjects, only to reveal that this subject could not have done the crime. Developing useful algorithms to score games based on the narrative content and other factors is a subject of future research.

Assuming games can be scored, the NOLIST game engine facilitates the rating of each possible next action by adding the weighting the score of each sampled game resulting from that action with its probability. The next action can then be chosen among the possible next actions based on their rating. The chosen action is then performed and the past updated by its effects, and the process is repeated until the story ends.

REFERENCES

- [Bangsø and Wuillemin, 2000] Bangsø, O. and Wuillemin, P.-H. (2000). Top-down construction and repetitive structures representation in Bayesian networks. In *Proceedings of the Thirteenth International FLAIRS Conference*, Florida, USA.
- [Jensen, 2001] Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. Springer Verlag, New York.
- [Juul, 2004] Juul, J. (2004). *Half-real. Video games between real rules and fictional worlds*. PhD thesis, IT-Universitetet.
- [Koller and Pfeffer, 1997] Koller, D. and Pfeffer, A. (1997). Object-oriented Bayesian networks. In Geiger, D. and Shenoy, P. P., editors, *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 302–313, San Francisco. Morgan Kaufmann Publishers, San Francisco.
- [Mallon and Webb, 2000] Mallon, B. and Webb, B. (2000). Structure, causality, visibility, and interaction: propositions for evaluating engagement in narrative multimedia. *International Journal of Human-Computer Studies*, 53:269–287.
- [Mateas and Stern, 2003] Mateas, M. and Stern, A. (2003). Facade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference*.
- [McKee, 1997] McKee, R. (1997). *Story: Substance, Structure, Style and The Principles of Screenwriting*. HarperCollins, New York.
- [Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers.
- [Szilas, 2003] Szilas, N. (2003). IDtension: a narrative engine for interactive drama. In *1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE 2003)*, March 24-26, 2003, Darmstadt, Germany.

APPENDIX

In the following table each number defines an atomic action. The last line of each entry contains the effects while the remaining lines are the prerequisites.

| | |
|---------|--|
| 1.1.1.1 | No evidence for an opportunity is known <i>M</i> indicating opportunity is found |
| 1.1.1.2 | No evidence for an opportunity is known <i>N</i> indicating opportunity is found |
| 1.1.2.1 | No evidence for a motive is known <i>X</i> indicating motive is found |
| 1.1.2.2 | No evidence for a motive is known <i>Y</i> indicating motive is found |
| 1.1.3.1 | It is not known who found the victim <i>A</i> found the victim, <i>A</i> had vague opportunity |
| 1.1.3.2 | It is not known who found the victim <i>B</i> found the victim, <i>B</i> had vague opportunity |
| 1.1.3.3 | It is not known who found the victim <i>B</i> found the victim, <i>C</i> had vague opportunity |
| 1.2.1.1 | It is not known if <i>B</i> had a motive <i>B</i> has a motive |
| 1.2.1.2 | It is not known if <i>B</i> had an opportunity <i>B</i> had vague opportunity |
| 1.2.1.3 | <i>B</i> had vague or clear opportunity <i>A</i> had no opportunity, <i>B</i> had no opportunity |
| 1.2.2.1 | <i>X</i> indicating motive is found <i>C</i> had strong motive |
| 1.2.2.2 | It is not known if <i>C</i> had an opportunity <i>C</i> had vague opportunity |
| 1.3.1.1 | It is not known if <i>A</i> had a motive <i>A</i> had weak motive |
| 1.3.1.2 | <i>A</i> had vague opportunity, <i>N</i> indicating opportunity is found <i>A</i> had clear opportunity |
| 1.3.2.1 | <i>B</i> had vague opportunity, <i>M</i> indicating opportunity is found, it is not known if <i>B</i> had a motive <i>B</i> had clear opportunity |
| 1.3.3.1 | It is not known if <i>C</i> had a motive <i>C</i> had weak motive |
| 1.3.3.2 | <i>C</i> had vague opportunity, <i>N</i> indicating opportunity is found <i>C</i> had clear opportunity |
| 1.3.3.3 | <i>C</i> had vague or clear opportunity <i>B</i> had no opportunity, <i>C</i> had no opportunity |
| 1.4.1.1 | <i>X</i> indicating motive is not found <i>Y</i> indicating motive is found |
| 1.4.1.2 | <i>A</i> had weak motive, <i>X</i> indicating motive is found <i>A</i> had strong motive |
| 1.4.1.3 | It is not known if <i>A</i> had an opportunity <i>A</i> had vague opportunity |
| 1.4.2.1 | It is not known if <i>B</i> had a motive <i>B</i> had weak motive |
| 1.4.2.2 | <i>B</i> had vague opportunity <i>B</i> had clear motive |
| 1.5.1.1 | <i>A</i> had strong motive, <i>A</i> had clear opportunity |
| 1.5.1.2 | <i>B</i> had strong motive, <i>B</i> had clear opportunity |
| 1.5.1.3 | <i>C</i> had strong motive, <i>C</i> had clear opportunity |

AI STRUCTURALIST STORYTELLING IN COMPUTER GAMES

Chris R. Fairclough and Pádraig Cunningham,
ML Group,
Computer science dept.,
Trinity College Dublin,
chris.fairclough@cs.tcd.ie, padraig.cunningham@cs.tcd.ie

KEYWORDS

Interactive story, case-based planning, computer games.

ABSTRACT

This paper is a description of our work in creating a story director agent which utilises AI techniques. The story director controls the storyline in an adventure computer game, with the player controlling the hero character, and the story director reacting to the player's actions. The story is told through subplot-level plans being formulated with a case-based planner, and a social simulation system that the story director is 'plugged in to', allowing consistent logical stories while allowing for player freedom. The system has been named OPIATE – Open-ended Proppian Interactive Adaptive Tale Engine.

INTRODUCTION

This paper follows (Fairclough & Cunningham 2002), and (Fairclough & Cunningham 2003); the former proposes the system, and the latter describes its development for use in multi-player games. After some background, this paper describes in detail the AI algorithms used in the system, the limitations of the approach, and possible future improvements.

BACKGROUND

Computer games are currently going through a number of contradictory trends. There is a new outcrop of mobile and internet-based games that emphasise short bursts of fun that are used for advertising, promoting websites and services, and even political messages. On the other hand, games that people invest more time in, such as adventure PC and console games, are becoming larger-scale, more complex affairs. This schism is serving to generate a wide range of new genres that borrow game concepts from each other, and from the older genres, through the short evolution of the computer game.

Genres that emphasise story and adventure are very popular, with 'Spiderman 2' currently topping charts around the globe. The current successful model for storytelling in games, popularised by GTA3, but initiated with Mario64, is to have a series of 'story missions' that advance the plot, with a selection of optional missions that enable a feeling of freedom of choice in a player. The variability of this model is based on the character abilities that the player has, so each game seems different, while this basic gameplay model is common to a lot of current games. This paper

proposes a possible next step for this storytelling model, abolishing the more traditional pre-scripted main plot for a more open-ended, procedural, view of stories themselves. This approach has been developed based on previous work in the fields of structuralist analysis, and was inspired by such contemporary practitioners as Chris Crawford (Crawford 2002), Michael Mateas (Mateas 1999), Nicholas Szilas (Szilas 1999), Nikitas Sgouros (Sgouros 1999), Norbert Braun (Braun & Grasbon 2001), and many others.

Some Previous Work

AI in storytelling was first concerned with story generation as text. In the seventies, Meehan's Talespin (Meehan 1977) generated much interest as a simple computer storyteller that utilised character-level planning. Later, Turner's Minstrel (Turner 1992) expanded on this to include author-level goals in a case-based planner. Turner's biggest success was in formulating a complete set of rules and paradigms for author and character-level planning using what he called 'imaginative memory', and analogical reasoning, to generate novel situations and plans for the characters and author model.

Storytelling for computer games has always been faced with the problems that occur when a player is given choices that could affect the plot. For real-time story generation, it has been assumed that these problems can lead to combinatorial explosions in complexity for a computer story teller, yet our approach demonstrates that this is avoidable. Since the OZ project in CMU (Smith & Bates 1989) began, more and more interest in AI real-time storytelling has surfaced, although it is not a technology that has been in much use commercially, although Braun's work on the Geist project (Braun 2002) has been used in a tourist attraction, utilising augmented reality headsets for display of characters.

Every system that has been developed is necessarily focused on a particular genre of story. Mateas focuses on a small location with only three characters, for a dramatic story experience. Crawford focuses on using a large number of interaction types (verbs) with short story segments that can relate to each other. He also provides an author tool-kit that enables creation of new storyworlds using his technology, but this does not allow for the emotional expressiveness of Mateas's approach, and is notoriously difficult to use. Nevertheless, his work has shown some of the possibilities and promise of interactive stories.

The challenge of creating a mechanism, whereby a player is both engrossed in a story and immersed in a world, is one that has been steadily overcome over the course of the evolution of computer games. Simulation

techniques, such as cellular automata, can enable a greater feeling of involvement and freedom in a living world, but the traditional concept of a story is incompatible with a world like this. Players are seen to create their own stories from their experiences in the world, as has been observed in 'The Sims'.

However, a story is not merely a series of causal events. Stories have their own innate structures and processes, independent of the characters they portray. This was asserted in the 19th century by Adolf Bastian (Koepping 1984), and emphasised by Vladimir Propp (Propp 1982) and Claude Levi-Strauss as structuralist theory was developed. To enable a simulation-level model of a story, these common structures of stories must be simulated using rules of dynamics based on the structuralist theory, and they must be active in the interactions of the storyworld. A believable gameworld can thus be augmented to create events that fit into the rules of world dynamics, but that also fit into a suitable story structure.

The story structures that we have elected to use are those of Vladimir Propp, who analysed Russian Folktales in 1928 and came up with an extremely empirical methodology for classifying his corpus. The applicability of folktale analysis to computer game storytelling is compelling, as the nature of folktales is ever-changing, allowing for an analysis that extrapolates the nonvariant elements of the tales. This has enabled the discovery of skeletal structures that can be fleshed out differently for each storyworld.

DESIGN

This section will deal with the storytelling architecture and detail the AI methodology that was used in the OPIATE system. The game architecture is detailed in our previous work, but consists of a 3D adventure game, with characters, objects, and locations being the most important components. The characters use a layered architecture and feature collision detection, idle behaviours, social simulation, attitudes, and goal-directed behaviours. The use of objects is how characters perform interactions, which generate events. The game engine handles some game events, but the independent story director agent initiates most events by being aware of the storyworld and giving relevant goals to the NPCs (non-player-characters). The most important element of the gameworld is the gossip system, which provides a dynamic social simulation where knowledge of game events is disseminated throughout the characters. This allows the player to effect the unfolding of the story, as the story director bases its decisions on these character dynamics. See (Fairclough & Cunningham 2003) for a more detailed description of the current testbed game engine.

Story Modelling

Stories are modelled as an interplay between autonomous character actions and story director- initiated story actions. The autonomous character actions occur as a result of a social simulation system, whereby each character builds up a set of attitudes for other characters, based on a memory of the actions that have happened directly concerning them, and actions that they have heard about or

witnessed. Characters have a gossiping system, which propagates information about game events through the cast of characters.

The story director agent queries the game world about character attitudes and locations, and player feedback, and bases planning decisions on this information. The plans it creates are sequences of character actions, each of which can be enacted by any character that fulfills the criteria for that action. These are equivalent to Propp's 'character functions', defined as 'an act of character, defined from the point of view of its importance to the course of action'. The system has a case library of plans that were authored based on the corpus of Propp's analysis in (Propp 1968). This case library encodes the expert knowledge that does not represent Propp's expertise, or any one expert's knowledge, but the expert knowledge encoded in the folk tales themselves, concerning the skeletal structures that define the different types of stories Propp analysed.

Case-based planning encodes knowledge as a library of cases, and deals with new problems through the mechanisms of recalling previous similar cases, adapting them for reuse, and assessing and storing the resulting new solution. Thus, a learning, adaptive system can efficiently solve problems similar to old ones. The story director (SD) in OPIATE uses the scheme shown below (Figure 1) to plan and cast story goals to characters. Each component of this process will be detailed in the following sections.

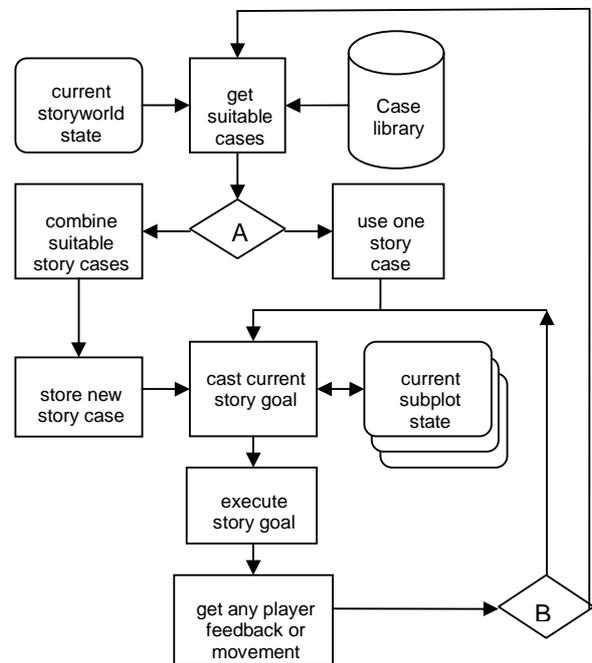


Figure 1. A flowchart showing the planning process

Suitability of Sub-Plots

The case based planning system uses a k-nearest neighbour algorithm to find suitable cases based on the heuristic shown below (Equation 1). The heuristic can be

termed a *suitability* metric, instead of the normal similarity metric used in case based systems. It finds the most suitable sub-plot to be enacted given the current state of the characters and storyworld, taking into account attitudes of the characters to each other, and to the player character. The core features that are used in this metric concern *roles* and *actions*. Roles are occupied by characters when they are enacting story functions, and the relevance of a character to a certain role is calculated based on past and present attitudes and memories concerning the player/hero character, and the other characters. Actions are enabled by *actionObjects* that occupy the storyworld, and allow characters to perform distinct types of interactions. They can all be picked up, given to other characters, and gossiped about.

$$\text{Eqn 1: } Sn = \left(\sum_{i=1}^{Ln} (Wr * Sri) + (Wa * Sai) \right) / Ln$$

Where Sn is the suitability of case n , Ln is the length in functions of case n , Wr and Wa are the relative weights attached to roles and actions, and Sri and Sai are the suitabilities of the roles and action(s) present in function i . Sri is given by Eqn 2, and Sai is given by Eqn 3.

$$\text{Eqn 2: } Sri = \sum_{j=1}^{\#C} Raj$$

Where $\#C$ is the number of characters currently available to the SD, and Raj is the relevance of character j to the role given by function i .

$$\text{Eqn 3: } Sai = \sum_{k=1}^{\#A} Rak$$

Where $\#A$ is the number of actions currently available to the characters, and Rak is the relevance of those actions to the actions required by function i . The relevance values are binary, as an action object either fulfils the action given in function i , or it doesn't.

Case Combination

Once an ordered list of suitable cases is found, using the quicksort algorithm, a decision is made to use the most suitable case (decision diamond A in Fig. 1), or combine cases to get a new one. If a hardcoded suitability threshold is reached, the former choice will occur, but if a combination of cases gets a better suitability, the latter will occur. Combination of cases is done on a per-function basis. As each function has its own suitability rating, the most suitable can be interchanged with less suitable functions in the target case. This is done by taking the most suitable case and replacing its less suitable functions with equivalent, but better scoring ones from the second or third ranking cases.

An important element in combining cases is to maintain integrity of the structures when they are transferred, so Propp's *groupings* of functions are used to facilitate this. If a function is selected for transfer, and it has associated functions from the source case, these are also transferred to the target case. This can entail replacement of target case

functions, so when the new case is constructed, it is reassessed for suitability. The groups are only of two or three functions, so this is not a difficult operation.

Casting

Once a suitable subplot plan is selected, it must be converted from a list of abstract story functions into a series of events in the gameworld, interpretable by the player as a storyline. To this end, the story director uses a casting system which dynamically casts the game characters into eight of the nine possible roles. Propp defined the seven roles: Hero, Villain, Mediator, Donor, Helper, False Hero, and Princess, and these have been augmented with two roles that he mentions, yet in his schema fall into the other categories. These are the roles of Family, and King. The hero character is always occupied by the player, even if they don't act particularly heroic. The usefulness of Propp's schema would be reduced if this was not the case.

These roles are cast as needed by a subplot. I will mention here that the term 'subplot' has been used in this paper where in Propp's work and our own previous papers, the term 'move' is used. This is to aid readability, as the general understanding of 'subplot' is roughly equivalent to the sense of Propp's 'move'. The roles required of the current subplot are dynamically cast as the subplot is being enacted, so that for example, a character can take the role of a Donor, and later can be the False Hero if the player/hero character falls out of favour with that character.

Casting is done using a set of criteria for each role. The villain role is filled by the character that opposes the hero the most, or else is a character close to that character. Opposition to the hero can come out of an attitude developed from author-defined backstory, or from events that occur in the course of the game. In this way, acts of villainy can be carried out by 'henchmen', depending on availability of characters. A Mediator can be any character that is available and nearby, even if the character is antagonistic to the hero. The Donor role can be filled by an available character that has not met the hero or has a slightly positive attitude. The Helper is filled by a previously met character that is fulfilling a positive previous encounter. The False Hero character must be a character with a previous positive attitude to the hero, who has either developed a negative attitude, or else has developed a positive attitude to the villain. The princess role is one that a character close to the hero can occupy, or a character that has not met the hero, but has been pre-authored as a possible princess character. The characters with positive attitudes to the hero can all take the Family role, and the King is taken by a powerful character, that a large number of characters have positive attitudes to.

The specificity of these roles and rules was formulated using a familiarity with Propp's work and its applicability to the game that has been developed, yet they could be editable through a toolkit if this system were to be used for other games. The rules are not arbitrary, and have been designed to maximise a sense of believability of the characters in their enactment of subplots.

Once a character is selected for a given function, the means of carrying it out is selected through a search of all actions available to the character. A character can be given

a sub-goal to find and pick up the object, or it can be given by another character. The enactment of the story function consists of finding the target of the function, animation of the actionobject, and the generation of suitable text for dialogue. The dialogue is generated with simple verb-noun structures, with characters capable of talking about characters, objects, events, and attitudes in a simple manner. Descriptive or emotional text is not used, and syntax is kept extremely simple. Despite this, a story can be seen to emerge based on the simple dialogue.

Because the system presents stories with animation, and is less dialogue-based, it is not suitable to present the output of the system here as a listing of dialogue. However, a presentation and some video files that illustrate the output of the system are available at www.cs.tcd.ie/faircloc/.

Player Feedback and Numerous Subplots

The player can elect to do what is asked of him in certain functions, e.g. the Donor function where a character tests the hero's worth with some challenge or request, or can ignore the request, whereby a recasting of the Donor goal is done. If the hero ventures into a new area, with new actions and characters available, or if new elements enter his current situation as a result of the simulation, an entirely new subplot can be selected for enactment (decision diamond B in Fig. 1). If this happens, the old subplot is not forgotten, but can come back into play if it is found to be again suitable for enactment. A set of active subplots is maintained, and the player can choose which ones to follow. These are the chief mechanisms which allow for player freedom of choice in the game, yet because the whole case-based planning mechanism works from data that is directly alterable by the player, the plot can also be directly influenced by player action in this way. For instance, if the player is 'liked' by a character, but performs some action that alters that character's attitudes towards them negatively, a plot with the Falsehero role would be more likely to be selected.

LIMITATIONS AND IMPROVEMENTS

The OPIATE system is limited by the home-grown game engine that is the current testbed. A more believable game world would help in evaluating the system's usefulness. It was decided not to use an available commercial game engine, such as the 'Unreal' engine, due to the need for flexibility and the required presence of the omniscient story director agent. The game has been developed to the point where a player testing scheme is possible, for a more objective analysis of the 'storyness' of the game experience. This is necessary for evaluation of the system, as the assumptions that were made in building the system are in question. Some of these assumptions are:

- That Propp's classification system is correct and shows structures that are actually present in tales.
- That a story can be 'reverse engineered' using these structures, and incorporated into a set of character dynamics.
- That sophisticated dialogue is not necessary to convey a story, but is used to enhance its quality.

The last assumption indicates an improvement that could be made to the system by incorporating a more advanced natural language module into the characters, which can be customised to each character. This would involve an author defining 'turns of phrase', colloquialisms, and typical adjectives that a character uses. This could form part of the social simulation, where characters can grow to use the pet phrases of the characters closest to them, and serve as an implicit indication of social connections.

An important component of creating a story in this system is the authoring of the game world and the interactions present in it. This is the chief method of authoring the high-level 'flow' of the story, defining the paths of movement and interaction through the game environment. By placing certain actionobjects in key locations or with certain characters, an author dictates the sort of interactions that will occur in certain stages of the game. An author also defines a backstory for the characters by giving them attitudes and memories of events. These attitudes and events are equivalent to the in-game ones, and effect the character dynamics in the same way.

The gameplay in our demonstration game is quite limited, and poses no real challenge to a player. If the player chooses to ignore a puzzle or challenge, then they can simply pursue a different subplot. However, there are only a limited number of puzzles authored, and no real possibility for emergent puzzles in the game. Puzzles are authored as specific problems a character wants solved, or the required use of a certain object to progress. As there are not a large number of locations (22), the player's options are quite limited. There are 28 characters and 18 types of actionobjects, however, so the player's choices primarily lie in their interactions with other characters. Characters can develop desires for certain types of object as a result of a subplot requiring it, but these are not authored puzzles, and play out somewhat artificially, as the desire is not based on any internal drives on the part of the character model. In a more large-scale, fully simulated world, the OPIATE system should perform better, with characters' problems and desires emerging from more fully realised character simulations.

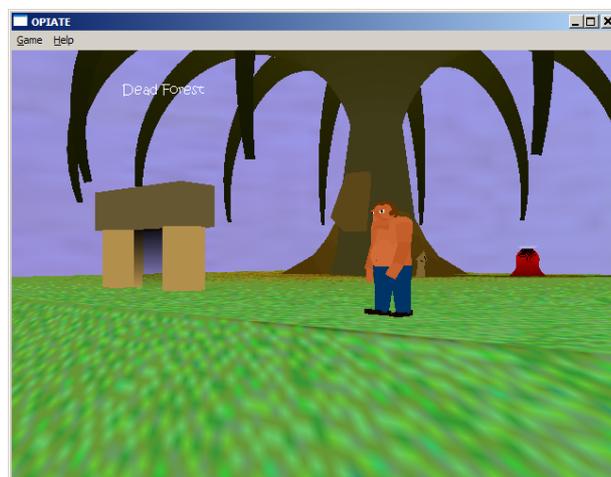


Figure 2: A screenshot of the demo game.

The OPIATE demonstration game is called 'Bonji's Adventures in Calabria' (Figure 2) and features three distinct 'locales' with about 8 locations in each. Progress from locale to locale entails solving a number of puzzles that are incorporated into the dynamic storytelling, with characters taking different roles depending on the player's interactions. However, the variability of the stories is limited by the initial setup of the storyworld, so the first subplot to be selected is always the same. Once a locale has been reached, the player can go back through the locales, revisiting characters and prompting new subplots to be selected based on the history of interactions.

Non-player characters can move around locales, but not between them. However, every time a subplot is successfully concluded, and as long as the subplot finishes with the 'Wedding' character function, a new character is available for player control. Propp's Wedding function is used as the hero's final reward at the end of folktales, and the decision was made to use this function to concurrently reward the player, by allowing them to control a new character. When a new character is selected, the previous hero character behaves like all the other NPCs, gossiping about attitudes and events, and is available for story goal enactment, consistent with previous interactions. This character can then be re-selected for use at any time. It would be theoretically possible for the player to gain control over every character in the game.

Overall, the system has turned out to be a success, blending ideas from a number of different projects to achieve an attractive option for a storytelling paradigm in computer games. The approach is experimental and not fully realised yet, but could help in developing more flexible story experiences for players. Future work on the system could help in its applicability to other game engines. This will necessitate the building of an author toolkit to allow for greater author control of the processes that the SD uses to direct the story, and a plot script editor for designing new subplots and new types of character function, outside the ones Propp defined which were used in this work. One serious limitation of the approach is that it does not seem to be incompatible with the current trend of pre-recorded speech in games. However, some games, notably ICO by SONY, manage to tell a story with almost no dialogue at all, so the more action-based storytelling approach of OPIATE could be useful in this type of game.

REFERENCES

- Braun, N. 2002, "Narrative Semiotics for Computer Games" Challenge of Computer Games, Lodz.
- Crawford, C. 2002, *The Art of Interactive Design* No Starch Press
- Fairclough, C. and Cunningham, P. 2002. "An interactive Story Engine", AICS 2002, LNAI 2464, p 171-176
- Fairclough, C. and Cunningham, P. 2003. "A Multiplayer Case-Based Story Engine", *Proceedings of the 4th International conference on Intelligent games and Simulation*. Eurosis. pp 41-46
- Grasbon, D. & Braun, N. 2001 "A Morphological Approach to Interactive Storytelling" Proceedings of the Conference on artistic, cultural and scientific aspects of experimental media spaces.
- Koepping, K. P. 1983 *Adolf Bastian and the Psychic Unity of Mankind* University of Queensland Press
- Magerko, B. 2002. "A Proposal for an Interactive Drama Architecture", AAAI Spring Symposium on interactive entertainment.
- Mateas, M., Stern, A 2000. "Towards Integrating Plot and Character for Interactive Drama." *Working notes of the Social Intelligent Agents: The Human in the Loop Symposium*. AAAI Fall Symposium Series.
- Meehan, J. 1977. *The Metanovel: writing stories on computer*. University microfilms international.
- Propp, V. 1968. *Morphology of the Folktale*. University of Texas Press
- Propp, V. 1984 *Theory and History of Folklore* University of Minnesota Press
- Sgouros, N. M. 1999, "Dynamic Generation, Management and Resolution of Interactive Plots" *Artificial Intelligence*, vol. 107, no.1, pp 29-62, Elsevier Science
- Smith, S. and Bates, J. 1989. "Towards a Theory of Narrative for Interactive Fiction", CMU-CS
- Szilas, N. 1999, "Interactive Drama on Computer: Beyond Linear Narrative" *Working notes of the Narrative Intelligence Symposium AAAI Fall Symposium Series*. Menlo Park, CA: AAAI Press.
- Turner, S. R. 1992, "Minstrel: A Computer Model of Creativity and Storytelling" University of California Tech Report CSD920057

A NEW METHODOLOGY FOR SPATIOTEMPORAL GAME DESIGN

Stéphane Natkin and Liliana Vega
Centre De Recherche en Informatique du CNAM
Conservatoire National des Arts et Métiers
292, rue St Martin - 75003 Paris, France
E-mails: {natkin, lvega}@cnam.fr

Stefan M. Grünvogel*
NOMADS Lab
Nonlinear Media: Art, Development and Science
Piusstr. 40, D-50823 Köln, Germany
E-mail: gruenvogel@nomadslab.org

KEYWORDS

Game design, spatiotemporal relations, Petri net, reachability tree, hypergraph, hyperedge replacement, connection

ABSTRACT

We propose a new formal approach for the design process of computer games, which involves the modeling of spatiotemporal relationships. Logical and temporal transactions are modeled using Petri Nets and topological relationships of the game universe by hypergraphs. For splicing these structures, we introduce *connections* which relate hyperedge replacement with the reachability tree of the Petri Net. By using these constructs flexible changes and the validation of certain properties of the missions can be accomplished.

INTRODUCTION

Game design is a difficult task that combines artistic and technical processes. Considering the interactive nature of computer games and, especially, their main goal (to entertain), the author must leave controlled freedom to the player (Bates 2001). The player has to be confronted with complex problems which are neither too easy nor too difficult to solve, making sure that game experience leads to a succession of goals within a reasonable time (Gal et al. 2002). In addition, the player must have a feeling of freedom in this interactive world, even when he is guided towards a solution in an unconscious way. To accomplish this, the game designer has to create “*one or more causally linked series of challenges in a simulated environment*” (Rollings 2003). This is not an easy job, as the following example shows.

Example: (room-key error) Two rooms *A* and *B* are only connected by a closed door, the avatar of the player is located in room *A* and its task is to get into room *B*. To achieve this he has to unlock the door with the help of a key. But the key is located in room *B*, thus the avatar will never be able to open the door.

Although this seems to be a trivial example, in real games different tasks can change the topology of the virtual space in a quite complex way, such that it is not easy to prevent the game designer from these or similar errors.

In this article, we concentrate on the modeling of spatiotemporal relationships of games. We start by giving a short overview of common design practices. The differentiation into game and level design is reflected by the two following sections. Special Petri Nets are used to define the missions of a game, and hypergraphs for characterizing the topology of the virtual space. To bring these two structures together we define after that *connections* and describe their basic applications.

GAME AND LEVEL DESIGN

In this article we consider the design process of a computer game from industry’s classical point of view. This process is decomposed into two phases: Game Design and Level Design. For the description of both a common method is to use game design documents. These documents define the different elements of game design and illustrate the game concept: scenario, game and level missions, character description, etc. (Bates 2000, Rollings 2003). It becomes *the* reference document for all production team members.

Game creation starts in most of the cases by an original idea and the development of its scenario. Thus in a first conceptual stage the principal aspects of the game universe are defined (Gal et al. 2002): Epoch and style, context, goals to be reached, main type of objects involved, users game perception, etc. The definition of the game at this stage is known as the *Game Design*.

Level Design is the next specification stage. A game level consists of a virtual space, puzzles, main actions and a set of objects to interact with in order to complete a given goal. The difficulty of puzzles can be defined by the geometry and topology of space, logic, action sequences and objects localization. Each level has to be meaningful for the game, and goals have to be the central element unifying the level theme (Bates 2001). With this in mind, a game designer’s task is to motivate the player by balancing objects behavior and game rules. Thus he has to take into account at each stage the space description, the positioning of objects in the virtual world, the logic of actions sequences relating these objects and the constraints caused by the topology of the virtual world.

* The author wants to express his sincere gratitude to Prof. Natkin and the CEDRIC-CNAM for the invitation to a research visit as *Professeur Invité* at the CEDRIC-CNAM during summer 2004.

PETRI NETS

We use Petri nets to model the ordering of action sequences in a game. This approach allows us to describe the logic structure of the missions in the game. The advantage of using Petri Nets (PN) is that they can be represented either by graphical or mathematical models, depending on model complexity and application context. (Peterson 1981). The following discussion is based on (Natkin and Vega 2003) approach.

PN game models are composed of transactions (cf. Figure 1). Transactions represent the atomic actions of the player, i.e. only one transaction can be executed at a time. Transactions can be in one of tree states: not started, executing or finished.

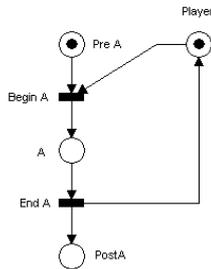


Figure 1. Basic Model of a Transaction.

A transaction net $N = (T, G)$ is a set of transactions T , which are combined by three basic constructs. A relation between two transactions is denoted by G . Transaction nets describe possible choices the player has during the game. Given two transactions a and b , they can be combined as shown in Figure 2. Each construct represents the possible sequence in which a and b can be executed. In Figure 2, the left construct show the case where a and b are not related (i.e. they can be executed in any order). In the center construct a is before b , which means that b can only be executed if a has been executed. In the right construct if a is executed b cannot be executed.

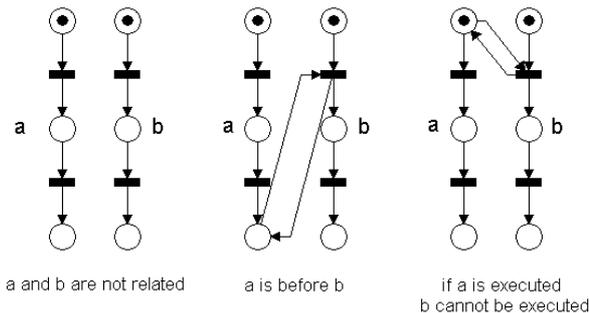


Figure 2. Relations between two Transactions a and b .

By combination of constructs, more complex semantics can be generated e.g. mutual exclusion between two transactions (i.e. if one transaction is executed, then the other one can not).

The reachability tree of a Petri Net generates a Petri Net language (cf. Diaz 2001). A sequence of firing transitions generates the corresponding strings. The reachability tree of a transaction net describes all possible sequences of atomic

player-choices. In transaction nets, transactions are atomic, thus we can merge the *Begin* and *End* transition of a transaction into one letter, generating the language $L(N)$. Applying these concepts to the transaction net on Figure 2, the left construct in Figure 2 generates the strings $\{a, b, ab, ba\}$, the middle construct $\{a, ab\}$ and the right construct $\{a, b, ba\}$.

Example: The former approach is applied to Silent Hill 2 (Konami 2002), a horror-adventure game that takes place in a mysterious and almost deserted town. The player controls an avatar (James) with unclear goals at the beginning. To accomplish his mission, he must explore the environment, fight against enemies and collect objects (keys, notes, information, etc.) to solve puzzles. The game starts at a parking bathroom outside town.

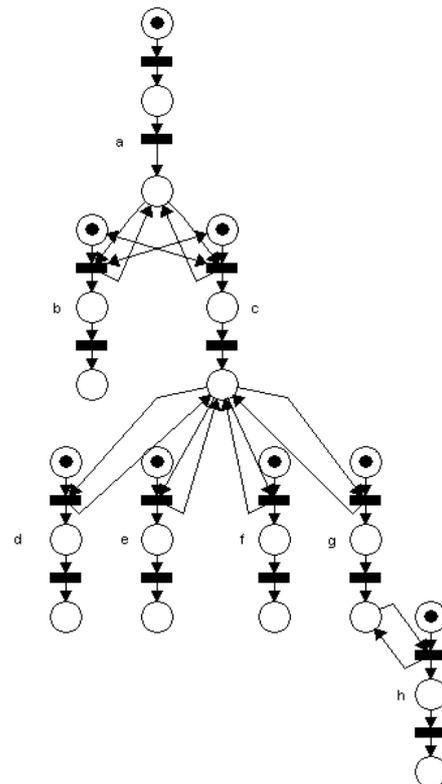


Figure 3. Silent Hill 2 First Level Transaction Net.

The main action sequence the player needs to execute to complete the first level is explained next (cf. Figure 3):

- (a) get the map from James car and hit the road to Silent Hill. Once in the town, (c) win the fight against a creature in order to be able to continue. (b) Loosing the fight means that the game is over. After winning the fight, James can execute the next four actions in arbitrary order:
- (d) Recover an unclear inscription from a weird monument.
- (e) Look at the map in the trailer.
- (f) Find a second map in Neely's Bar.
- (g) Find the apartment key on a dead body.
- (d), (e) and (f) are optional actions that can be repeated an infinite number of times with out changing the course of the game. (g) is mandatory and marks the end of the first level. After this, James is able to (h) enter the apartment to begin the next level.

SPATIAL RELATIONSHIPS

Besides describing the mission's logical structure (cf. the previous section), describing the game world topological properties and its evolution is also important.

There are different approaches to describe spatial relationships. One is to use visual languages (Haarslev, 1996) where description logic is used to combine topological and spatial relations and applications can be found in geographical information systems. A common way for describing topological relationships is to use the usual point-set topology with open and closed sets. In (Haarslev, 1996) an overview of different concepts is given, where often the interior and the boundary of sets are also taken into account. We do not follow this fine grained typology, but use instead a much simpler concept where we do not take into account the boundaries of sets.

In (Flury et al. 2003) an overview of location models in pervasive computing is given, where they establish the term *locus / loci* for location entities (e.g. regions of physical space) and *locants* for locatable entities, which can be passive or active objects (like a user). We adopt these terms and will denote a region of the virtual space as *locus*. In our approach, we do not model locants explicitly, but we concentrate instead only on the connections between different loci. The representation we introduce is a *locus graph*, where loci represent some subsets of space and their structural relationships are described by edges. These relationships are expressed here by (directed) hypergraphs.

Hypergraphs are generalizations of graphs, for a formal description and a thorough discussion of hypergraphs cf. (Drewes et al. 1997). For the presentation in the article, we take the somewhat simpler definition of (Gallo et al. 1992). A *hypergraph* H is a pair $H=(V,E)$, where V is a finite set of *vertices* (or *nodes*) and $E = \{E_1, K, E_m\}$ with $E_i \subseteq V$ for $i=1 \dots m$ is the set of *hyperedges*. Clearly, when $|E_i| = 2$ for all i , then the hypergraph is a standard graph. Additionally, by a finite nonempty set C of *labels* and a function $lab : E \rightarrow C$ the hyperedges can be labeled.

A directed hyperedge is an ordered pair $E=(X,Y)$ of (possibly empty) disjoint subsets of vertices. X is the *tail* of E and Y its *head*. A *directed hypergraph* is a hypergraph with directed hyperedges, which we will denote hypergraph in the following for simplicity. Petri Nets can be modeled by hypergraphs and also (ordinary) directed graphs. Given a set of vertices V and labels C the class of all directed hypergraphs over C with these vertices is denoted by $H(C)$.

To construct the topological relationships, the virtual space of a game is divided into loci, which are represented by nodes V in a hypergraph. Each locus is a pathconnected subset of the virtual space, which means that the avatar can move to every point in the locus at any time of the game if he is within the locus. Typical examples for loci are the rooms of a house, or special zones in an outdoor area. The

set of loci does not change during the game, it is fixed. Therefore the partitioning of the virtual world into loci defines all the possible places of interest which have to be taken into account for the game. Note, that loci do not have to be maximal with respect of the pathconnection property, which means that it is e.g. possible to divide a room into two loci (the left and the right half) while the avatar is able to move freely within the room.

The ability for an avatar to move from one locus to another at a certain state of the game is modeled by an hyperedge E_i .

Because the hypergraph we use is directed, it is possible to model one-way connections, where it is only possible to move an object from one locus (the tail of the hyperedge) to another (the head), but not the same way back. Note also, that the hypergraph is not unique because there may be several ways of defining the corresponding hyperedges. We use hyperedges because they have more possibilities for presenting spatial relationships than edges from ordinary graphs, which can only connect two nodes.

While playing the game, the hypergraphs may change, i.e. certain hyperedges can be replaced by others or even vanish. Formally this can be described by hyperedge replacement cf. (Drewes et al. 1997). As mentioned before, we assume that the decomposition of the virtual space into loci is fixed and does not change during the game.

Example: In Figure 4 a map of the first part of Silent Hill 2 is given (non-real scale).

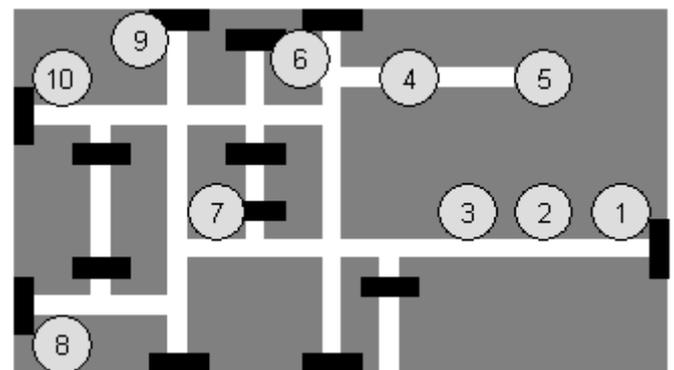


Figure 4. Partial Map of Silent Hill 2. White: Street, Grey: Building/Area, Black: Roadblock, 1 Parking Place, 2 Fountain, 3 Church, 4 Backyard, 5 Tunnel, 6 Swamp Monument, 7 Neely's Bar, 8 Trailer, 9 Apartment Gate Key, 10 Apartment.

The white stripes represent the streets, the grey areas buildings or open areas. The circled numbers represent special subsets of the virtual world where the player experiences special actions or has to fulfill specific tasks. These are taken as nodes in the hypergraph of Figure 5, where the corresponding spatial relationships between these loci at the beginning of the game are represented. Hyperedges are represented as arrows with the label of the hyperedge within a square. Note, that the hyperedge A and A' are not connected with any nodes (especially not with node 1), which reflects that at the beginning of the game, the

protagonist can not leave the parking place. For graphical representation, we also gather several hyperedges into the edge D , which means that you can move freely between any of the nodes 3 and 4.

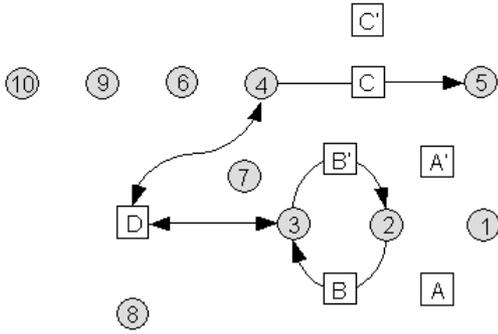


Figure 5. Spatial Relationships at the Start.

CONNECTIONS

The change of the topology of the virtual world can be modeled by hyperedge replacement. For a formal definition of general hyperedge replacement, we refer to (Drewes et al. 1997). For the sake of simplicity we only introduce in this article the replacement of one hyperedge at a time. But all of the following constructions are also possible for general hyperedge replacements on hypergraphs.

Let the set of vertices V be fixed, C a set of labels and $H(C)$ denote the corresponding set of hypergraphs over C , let $A \subseteq V$ be a hyperedge to be replaced by another hyperedge $B \subseteq V$. Then the replacement $[A/B]$ is a function $[A/B]$ from H to H , where a hypergraph H is mapped to the hypergraph $H[A/B]$ by removing A from H and adding the hyperedge B to H . If the hyperedge A is labeled, then the new hyperedge B inherits this label.

As an example, the hyperedge $e = (\{1\}, \{1\})$ with $lab(e) = A$ in Figure 5 does not connect any node. It is replaced by the hyperedge $e' = (\{1\}, \{2\})$ with $lab(e') = A$ in Figure 6, which actually connects now the nodes 1 and 2.

In the following we introduce *connections* relating hypergraphs with transaction nets.

Definition: Let V be a set of vertices, C as set of labels, $H(C)$ be the corresponding set of hypergraphs and $N = (T, G)$ a transaction net with a corresponding language $L(N)$. Let $\hat{T} = T \cup \{*, ?\}$ be the augmented set of transactions. Then a *connection* is a pair $([A/B], p)$ where $[A/B]$ is a replacement and $p \in \hat{T}^*$. Here \hat{T}^* is the set of all strings of \hat{T} including the empty string. We call $[A/B]$ the *replacement* and p the *pattern* of the connection.

Now the semantics of a connection $([A/B], p)$ is defined as follows. Given the overall mission by a transaction net N and the initial spatial relationship of the mission by a hypergraph H . By walking through the transaction net, a string s of increasing length is created. If the string s contains the string p then the corresponding replacement

$[A/B]$ is applied on the hypergraph. The wildcard symbols “*” and “?” have the usual semantics known from common pattern matching programs, where “*” means “arbitrary symbols” and “?” “at most one symbol”.

Example: We consider again the Silent Hill example. After having examined the car, the avatar is able to leave the parking lot. This can be modeled in the following way. Let $W = (\{1\}, \{1\})$ and $X = (\{1\}, \{2\})$ with $lab(W) = lab(X) = A$ and $Y = (\{2\}, \{2\})$ and $Z = (\{2\}, \{1\})$ with $lab(Y) = lab(Z) = A'$ be hyperedges. Then the connectors are defined as

$$([W/X], a) \text{ and } ([Y/Z], a),$$

where a is the transaction from Figure 3.

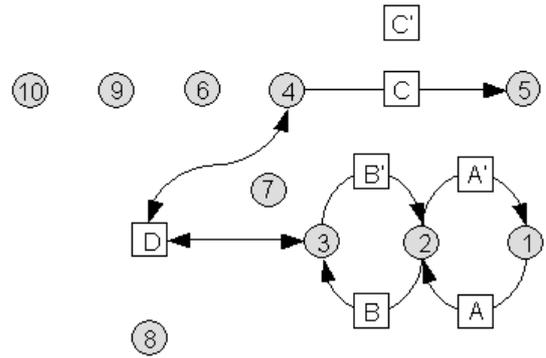


Figure 6. After investigating car in (1), the player can leave the parking place.

In the same way, that only after killing the monster in the tunnel the avatar is able to leave the tunnel is modeled by

$$([K/L], c) \text{ and } ([M/N], g),$$

with $K = (\{1\}, \{1\})$, $L = (\{5\}, \{4\})$ and $lab(K) = lab(L) = C'$ and $M = (\{3, 4\}, \{3, 4\})$, $N = (\{3, 4, 6, 7, 8, 9\}, \{3, 4, 6, 7, 8, 9\})$ and $lab(M) = lab(N) = D$. (cf. Figure 7).

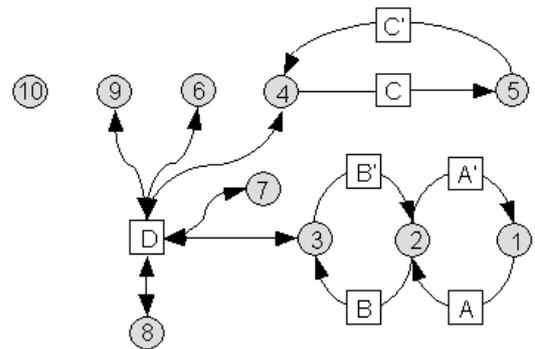


Figure 7. After the fight with the monster is won.

Finally the accessibility of the apartments after receiving the key (cf. Figure 8) is expressed by

$$([P/Q], g)$$

with $P = (\{3, 4, 6, 7, 8, 9\}, \{3, 4, 6, 8, 9\})$, and $Q = (\{3, 4, 6, 7, 8, 9, 10\}, \{3, 4, 6, 8, 9, 10\})$ and $lab(M) = lab(N) = D$. (Note that for

AUDITORY GAME AUTHORIZING

FROM VIRTUAL WORLDS TO AUDITORY ENVIRONMENTS

Niklas Röber and Maic Masuch
Games Research Group, Department of Simulation and Graphics
University of Magdeburg, D-39106, Magdeburg, Germany
E-mail: nroeber|masuch@isg.cs.uni-magdeburg.de

ABSTRACT

The majority of information that we perceive from our real-world environment is of audio-visual nature. Virtual worlds, which are utilized in computer games to line out the story's stage, are composed of visual and auditory environments. These environments are designed to provide sufficient information for the interaction and exploration of these worlds. The authoring – or content creation – of such environments can be a very tedious and time consuming task. In this paper we focus on a specific chapter of game authoring: The authoring of auditory environments for virtual worlds. Many of the tools available for auditory authoring focus on visual cues rather than on auditory cues and common hearing behaviour. We compare several existing programs towards their applicability for authoring audio-visual as well as audio-only applications. In addition, we propose a new system, that allows the authoring of auditory environments through a non-visual interface by solely utilizing sound and specially designed interaction techniques. Our work is motivated by the current development of such an authoring system for virtual, auditory spaces. The implementation is work in progress and currently exists as prototypic application.

KEYWORDS

Authoring, sound, auditory environments, audio-only applications, sonification, interaction.

INTRODUCTION

The design and development of computer games is a very time and resource consuming venture. Unlike during the earlier days, where computer games could be created by single persons or small groups, today's games easily can cost several million dollars and take numerous employees and years to complete. The small fringe market of electronic games has evolved into a huge, global business. Today's games are not only bigger and more complex, they also feature unseen graphic and sound effects, making these games more realistic than ever before.

To develop such games, many groups are working collaboratively for years on various aspects of the game, ranging from 3D game programming, computer graphics and design over concept-art to game authoring (Watt and Policarpo 2000). The authoring of a game can be seen as the final assembly step where all the different parts are put together. The graphics and assets are imported into the game engine and connected with story related game events. The same procedure applies to sound, AI and all the other game elements. Many proprietary tools have been developed for game authoring and are often shipped with freely available game engines (Fly3D).

In this paper we concentrate on the auditory authoring of virtual, 3-dimensional worlds. This authoring includes the setup of auditory environments for audio-visual and with a particular focus also on audio-only applications. The acoustics of many games in the earlier days was often limited to beeps of varying length and frequency. Later, with the introduction of home computers like the Atari, Amiga or the Commodore C64, midi music and the playback of more complex sounds were possible. With the advent of additional sound hardware in the beginning of the 90s, stereo sound and advanced midi music using wavetable synthesis were possible and warmly welcomed by the gaming industry. Aureal and Creative Labs added a new dimension with the introduction of 3D sound in the end of the 90s (IASIG 1997). Three-dimensional sound moved quickly into the focus of game developers and players and is now a well established standard in many computer games (Menshikov 2003).

Nowadays, the sound engines, which were designed to handle all the audio processing, have evolved by a large magnitude and many game developers devote to audio processing more and more attention. Several technologies have emerged that support software or hardware accelerated rendering of 3D sounds and room acoustics (Gardner 1999). While playing computer games, the most active senses are vision and hearing. But the visual and the auditory field of view are two independent sensory systems, that respond to different activations and highlight different parts of our local

environment (Goldstein 2001). Through these differences, hearing provides us with information that is often distinct from the visual perception and used to enhance the cognition provided by the eyes. Hearing is often considered to drive the attention: *The ears are steering the eyes*. Adopting this understanding to game authoring, methods that highlight on auditory cues rather than visual cues can be used to model acoustic spaces more intuitively.

So far, many of the existing auditory authoring systems used in the industry for film and computer games are based on visual cues, where sound sources are selected through mouse interaction and parameters are defined through visual interfaces (EAGLE 2004). Various programs arrange sounds in virtual tracks, in which they can be composed together for surround sound rendering (Maven3D 2004).

Although, some of the games developed still utilize only stereophonic sound, in this paper we explicitly focus on three-dimensional sound and acoustic rendering. We provide an overview of common auditory authoring software with the focus on creating entertaining audio-visual and audio-only applications. At the end we will concentrate on our new approach for authoring auditory worlds through sound alone.

The paper is organized as follows: In the next section we give an introduction to auditory environments and compare them with their visual counterpart. As the focus is slightly on non-visual applications, special assumptions are made to provide additional information for navigation and orientation within these auditory spaces. The following section discusses state-of-the-art audio authoring applications as they are used throughout the game and film industry. We highlight their advantages as well as show their limitations towards authoring non-visual worlds. In the following section, we explain how the auditory authoring process can be improved and how non-visual authoring tools can be designed to aid in the authoring process. In the end we will summarize the work and state possible directions for future improvements.

AUDITORY ENVIRONMENTS

Auditory Environments are much like visual environments. The only difference is that these environments are perceived by hearing instead of vision. The environmental information conveyed through sound is usually different than the one through vision (Goldstein 2001). Auditory elements, such as object sounds, music and speech can be used to model a scene in virtual, auditory worlds and to describe the environment through sound alone (Röber and Masuch 2004). These auditory scenes can vary as much as visual scenes and can also be grouped to form a larger

surrounding with smooth transitions from one auditory environment to the other. Special care has to be taken with the modelling of these transitions. For non-interactive environments, like movies, these transitions can be previously defined and setup in advance to precisely match their desired effect. For interactive environments, like computer games, these transitions have to be fine-tuned during the game play depending on the games status and the players actions. By employing a smart blending between the environments, smooth transitions are possible within the auditory display for both, object-sounds and music (Hämäläinen 2002).

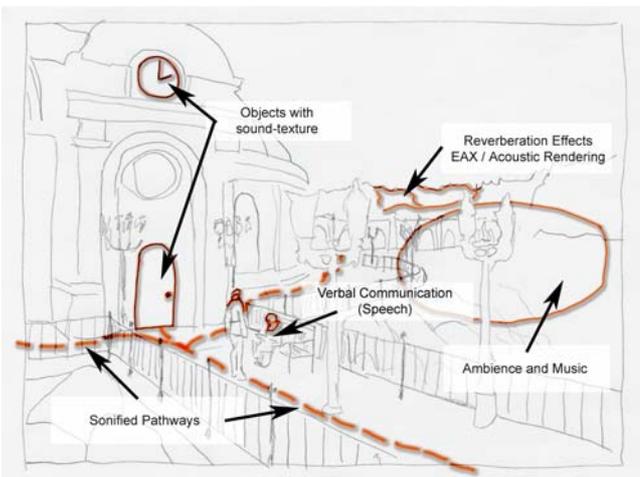
We define auditory environments as the audible analogue of a visual scene which is composed of auditory elements describing the objects in this scenery. Similar to the real world, virtual worlds are constructed of different perceptible environments that are intertwined. In audio-visual applications, each scene consists of definitions for the visual and the auditory part.

As the visual and the auditory field of view are diverse, the information conveyed through auditory channels is not necessarily the same as the one which is depicted in the visual part. While most applications rely mainly on visual cues to convey the majority of information, many of them utilize sound as additional attractor to increase the efficiency of the display. Some applications have been developed that only use non-speech sound to transmit information. Many of these applications originate from data sonification (Speeth 1961) and applications to aid in the navigation and orientation of the visually impaired (Strothotte 1995).

The focus of so-called auditory displays is to convey information through sound alone. Although most of the applications are developed for the blind, many of the techniques can also be used by sighted operators to utilize sound as enhancement for visual perception or for mobile applications where vision and the demand of free eyes are mandatory. Examples are audio based guiding systems and audio-only games. The Audio Games website (AudioGames) contains some good introductory articles about this topic. Most of these games use the narrative to develop an underlying story and the most successful genre are Adventures. Drewes (Drewes 2000) developed an immersive audio game, employing wearable computing and augmented reality technology. Targett et. al. (Targett 2003) examine the possibilities of using audio-only games for therapeutic applications. They discovered that playing audio-only games is not only fun for both, sighted and visually impaired, but can also be used to develop certain skills in auditory perception.



(a) Visual scene (Syberia 2002).



(b) Auditory scene.

Figure 1: Visual and auditory environments.

For the rendering of acoustic scenes, it is more important to focus on the clarity and effectivity of the display rather than on physically accurate rendering. Special sonification and interaction techniques have been developed to convey the information and to provide the player with enough intelligence for orientation, navigation and object interaction (Röber and Masuch 2004).

An auditory environment is composed of object sounds, music and speech. These auditory elements are sufficient to convey all the information needed. Figure 1 shows an example scene which is represented visually (Figure 1a) and through auditory elements (Figure 1b). In general, less information can be transmitted through acoustic channels. To compensate for this deficiency in audio-only applications, the auditory environment has to be enriched and special sonification and interaction techniques have to be employed. Every object that is part of such an acoustic environment is audible and can be described through auditory textures. Each of these sound textures is a collection of different sounds that describes an object

in different situations or under varying conditions. Sound types that are used for the composing of auditory textures are:

- A general object sound,
- Several action or status changed sounds,
- A call sign for the radar, and
- A verbal description.

The general object sound is the acoustic representation for this object in its usual state. This sound is audible if the player is within a pre-defined range for this object. Action and status changed sounds are used to characterize a current activity or changing situation for this object. These sounds are played only once, e.g. clicking a button, but can also activate a different general representation for this object, e.g. switching on a radio. The radar call sign and the verbal description are two additional auditory icons, that are used to describe the object and to identify its position. These are activated on request and only played once.

Auditory Texture

| | |
|--------------------|-----------------------------------|
| General Sound | Silent or auditory Icon (ringing) |
| Status changed | Incoming call (loud ringing) |
| Status changes | Broken (auditory icon) |
| Action | Pick up phone (auditory icon) |
| Action | Dialing (beeps) |
| Action | Talking (silent) |
| Action | Ring off (auditory icon) |
| Radar call | Auditory icon (ringing) |
| Verbal description | Speech: "Telephone" |



Figure 2: Auditory texture (Telephone).

Figure 2 sketches the idea of auditory textures using a telephone as an example. The sound texture consists of a total of 9 sounds including silence, four action sounds and three status changed sounds. Some objects may only have one general object sound, like a wall we can bump into, while other objects can have many layers of different sounds. Which sounds are selected for the object representation depends on the underlying story engine and the player's interaction. The selections of sounds can be managed through a story engine to assist the player in the game play and to push the story forward. Therefore, these acoustic textures can also be classified as *story-related* sound textures. Similar to story-related sound objects are environmental sound textures, which are mentioned here for completeness, but are not further discussed in this paper. These textures describe an objects physical state and the interaction between several objects. Story- and environment-related sound textures share many

similarities. The environmental audio texture is composed of sounds that characterize the object in different situations. Depending on the physical state of this object, one or more of these sounds are selected and composed together. An example would be the difference in sound at varying weather or road conditions from the tires of a moving car. The authoring of these textures works similar as with the story-related textures, except that here several sounds can be composed together to precisely meet the current environmental settings. Which sounds are used depends on the underlying physics engine which models the environment.

The field of auralization is concerned with the correct rendering of the acoustics in a virtual scene. Often used techniques involve 3D sound rendering through binaural differences and HRTFs¹ as well as an acoustic simulation model for environmental reverberation (Begault 1994). For the latter, statistical approximations like EAX are commonly used to simulate the reverberation for a specific environment. These approximations are often sufficient enough to convey general information about the environment, but fail in the recognition of special places where information is acquired through the analysis of reflected sound.

Another – still game related – interface are in-game or shell menus, which are used to adjust game specific parameters, like loudness, or allow to load a previously saved game. In audio-visual applications, especially in computer games, many commands are additionally commented through an auditory feedback sign. With the focus on non-visual applications, auditory displays and auditory user interfaces (AUI) are utilized for these settings (Begault 1994). For the interaction with these menus, similar methods apply as for the non-visual authoring, see Section 4.

Depending on the type of application (audio-visual or audio-only) different auditory environments have to be created to communicate the information that is necessary for the later orientation, navigation and object interaction. In the following sections, a rough overview of state-of-the-art audio authoring tools and a definition of requirements for non-visual authoring is presented. In the section after, we focus on our new approach, highlighting a non-visual authoring environment for audio-only applications.

AUDITORY AUTHORIZING OF GAMES

Auditory authoring is the process of designing, editing and integrating auditory information into a virtual

environment. This is done in two steps, where first the required sounds and music are arranged and composed and later integrated into the virtual environment. This integration includes the definition and the connection of sounds and music to gameplay events.

For the auditory authoring we mainly focus on the second part, the integration of sounds into the virtual environment. Music and speech are both placed at areas within the environment and activated through the story engine. Additionally, the narrator can be triggered through the player for assistance and for verbal scene descriptions. Based on virtual, auditory environments, two groups can be identified for the authoring process:

- Environment-related and
- Story-related authoring.

The integration of sound in the environment includes the specification of the sounds location and the possible connection to geometrical objects or regions. Additionally, it can include the setup of parameters like loudness, range or directional performance. For the integration into the story line, events are defined that act as switches between the different layers of the objects story-related auditory texture. If an object additionally uses an environment-related auditory texture, these events are defined by the physics engine of the environment.

For the authoring we assume that the sounds are already engineered and we have a large variety of sounds which *only* need to be integrated into the game and connected with the game play events. With the focus on audio-only environments, we distinguish between authoring for audio-visual and audio-only applications.

Some basic functionalities which have to be supported by the authoring environment include the import and a proper sonification of the underlying geometrical data of the virtual environment, as well as standard preview and exploration techniques that allow for orientation, navigation and object interaction. For the sound authoring, we have to be able to perform the following tasks:

- Select, create and remove sound sources,
- Placement of sound sources (connection to objects or areas),
- Setup of sound parameters (loudness, range, attenuation etc.), and
- Setup of background sound sources.

Sound sources are created within the geometrical representation of the environment and can be either autonomous or connected with scene objects. Sounds

¹ Head-related Transfer Functions

or auditory textures are selected and assigned to these sound sources and parameters with sound typical properties like volume, range or attenuation can be adjusted. Many programs additionally allow the setup of music and the integration of narrator's comments, as well as simple settings for general background and environmental sounds.

The object sound authoring is performed in combination with the environmental sound setup and includes the following actions:

- Definition of sound textures,
- Selection of sounds,
- Setup of sound parameters (loudness, range, attenuation etc.),
- Definition of events and actions, and
- Switch on/off sound sources or sound groups.

This is basically the construction of story-related auditory textures and the definition of events that switch between these different object sounds. This includes parameter specifications for the individual sound layers as well as the group behaviour of sound objects. These audio textures are now assigned to sound sources created during the environmental authoring step.

In most existing applications, the sound setup is handled with the help of a 3D modelling program that displays the game's geometrical data and shows features of the game play. The sound sources are arranged and eventually connected to *real* objects and events. An auditory preview of the scene allows a verification of the designed auditory environment. Some programs combine the design and the sound authoring in one convenient environment.

Tools that are solely used for design and composition are Sonic Foundry's Soundforge for wave data (SoundForge 2004) and the Cubase (Cubase 2004) and Cakewalk (Cakewalk 2004) program families for midi editing. These programs are used ahead of the final authoring step to design the sound and music samples. The functionality of the authoring programs varies depending on the target platform and the sound engine used. Professional tools are the new XACT system for auditory game authoring which is part of Microsoft's new XNA game development platform (XNA 2004). Other professional authoring applications include Sony's SCREAM for the PS/2 (SCREAM 2004), Sensaura's gameCODA2 API and authoring environment (gameCODA 2004) as well as Creative Lab's EAGLE for EAX environmental authoring (EAGLE 2004).

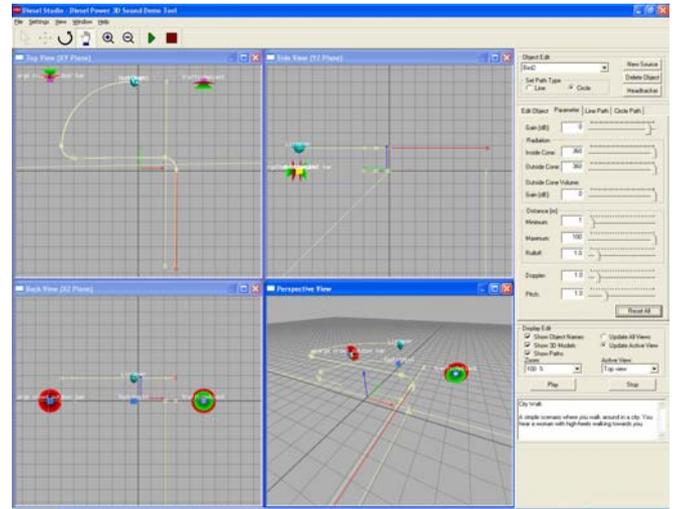


Figure 3: Screenshot of DieselStudio from AM:3D (AM:3D 2004).

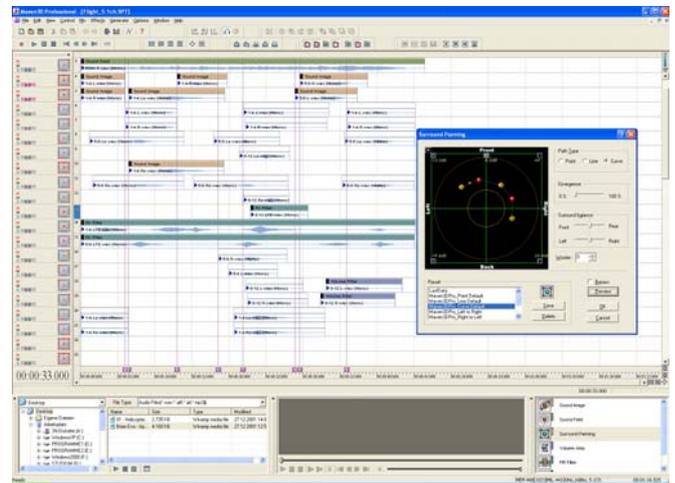


Figure 4: Screenshot of Maven3D (Maven3D 2004).

A popular API for software rendering of spatialized sounds is AM:3D's DieselStudio (AM:3D 2004). It includes an API for programming 3D sound, as well as a simple editor, that can be used to author simple scenarios, Figure 3. It allows the animation of sounds along pathways and the inclusion of geometry to find the correct sound placement within the environment.

An example for a track based authoring system is Maven3D from Emerging Systems (Maven3D 2004) and the Soundfactory from CRI Middleware (Soundfactory 2004). Both allow the design and setup of 3D sounds and their integration into the game environment. Figure 4 shows a screenshot of Maven3D Professional, mixing several tracks for a 5.1 audio project. Maven3D also allows for cross-talk cancellation and some specifications in acoustic rendering.

Interesting and unconventional ideas for user interfaces to perform sound setup, design or authoring can be found in art and research projects. An overview of

alternative music instruments can be found in the Master's Thesis of Jörg Piringer (Piringer 2001). Although, many of these interfaces abjure visual interaction (Flür 1976), most of the authoring systems designed for the authoring of virtual environments use visual metaphors (Väänänen 2003).

Most of these authoring tools have evolved over many years and are appropriate for the authoring of audio-visual applications, but do not meet the requirements for the setup of audio-only environments. For these applications, special techniques have to be used to integrate additional information, that is required for the successful sonification of these environments. The next sections focus on these problems and introduces a new authoring idea that uses the same media: sound and a non-visual interface for the authoring of acoustic spaces.

NON-VISUAL AUDITORY AUTHORIZING

We propose that for a more immersive and realistic setup of acoustic environments, especially with the focus on audio-only applications, the authoring should take place in and with the same media: sound and a non-visual interface. This is both challenging and rewarding. Challenging, as we have to express non-auditory information through acoustics, and rewarding, as we do not break the illusion of being immersed in a virtual, auditory world. For the authoring of non-visual environments, we have to include additional information for non-visual perception. This includes auditory textures that describe scene objects as well as data to aid in navigation and orientation. Furthermore, music and a narrator's voice have to be integrated into the environment to support the story and to deliver extra information.

The non-visual authoring of auditory environments works similar to the authoring of audio-visual worlds. The only difference is the media of interaction and the utilization of specific interaction and data sonification techniques (Röber and Masuch 2004). The user is provided with the geometrical representation of the virtual environment that is acoustically displayed using data sonification techniques. Within this environment, the user is able to use a 3D auditory pointing device to select locations or objects from this data on which sound sources can be placed. Parameters and sound textures are defined through gestures and audio widgets. Additionally, head-tracking is used to allow for an easier navigation and orientation within the data sets.

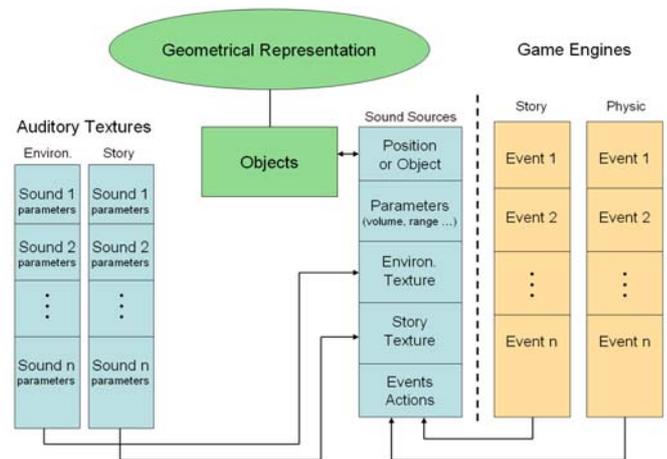


Figure 5: System overview.

Figure 5 shows an overview of the system. The authoring itself is split into two parts: the construction of auditory textures and the integration of sound sources into the environment. At desired places, sound sources can be constructed and parameters are setup for these sources. Both auditory textures, environmental and story-related, can be assigned to the sound source, and their internal sounds are linked with the events from the story, respectively the physics engine.

Most of the interaction techniques used for the authoring can also be applied to playing audio-only games (Röber and Masuch 2004). Supported techniques are:

- Head-tracking,
- 3D auditory pointing devices,
- Auditory widgets,
- Gestures and a
- Gamepad.

Head-tracking and other tracking devices allow intuitive operations with the environment by using natural hearing behaviour and gestures as interactive media. A 3-dimensional pointing device, that is part of the tracking system, is used for sound positioning and the specification of parameters. These parameters are defined through a combination of auditory widgets and gestures. A gamepad is utilized for navigation and listener orientation as well as for object and sound selection to construct auditory textures and to set up sound sources. Figure 6 shows a simple scene which shall be used to exemplify the authoring process. The player is located inside a locked room and has to find the keys to open the door. Basically three main sound groups are present in this scene. Sounds “S1” and “S2” represent ambient sounds from the outside of the building and are used as landmark for orientation purposes. Other sounds can be added to make the scenery more realistic, but care has to be taken to not clutter the auditory display with too much information.

Through interaction the player will notify that the door is locked “S4”. While looking around, the player can find the keys “S3” and open the door to proceed. For the moment, the focus is not on the interaction to accomplish this task, rather on how to model and authorize this scene. Sounds “S1” and “S2” can easily be defined as a mixture of ambient sounds representing noise on the street. If no further interaction is aimed, the associated sound texture consists of only one sound. The auditory texture of the door “S4” has at least three sounds, one that represents a locked door, one for an open one and another one for unlocking the door. The key object “S3” on the other hand only needs a prominent sound that identifies itself as *key* object. The story engine can control the effectiveness of this sound and make it more distinctive or more ambient, depending on the player’s intuition.

The authoring itself is straight forward. The mesh of the environment is imported and sonified and the author can walk around freely. Mesh objects are identified through their names and groups, but can also be selected from an auditory menu. During the 3D modelling of the mesh, dummy objects (simple boxes) can be placed at positions which are later used to identify sound object positions. The author can now select these objects and connect them with an auditory texture.

These auditory textures are created using an auditory menu in which gestures and real 3D pointing devices play an important role. As the sounds are readily composed, one only needs to select the number of slots for the auditory texture and fill them with data. After this, each sound and the sound object as a whole can be fine tuned by specifying parameters, like direction, range etc. These parameters are again specified using an auditory menu. The development of a story engine and the connection of events with this story engine is currently under development.

The system is currently work in progress and at the time of writing existent as prototypic implementation. This prototype will be extended by additional sonification and interaction techniques and the resulting application examined in user studies.

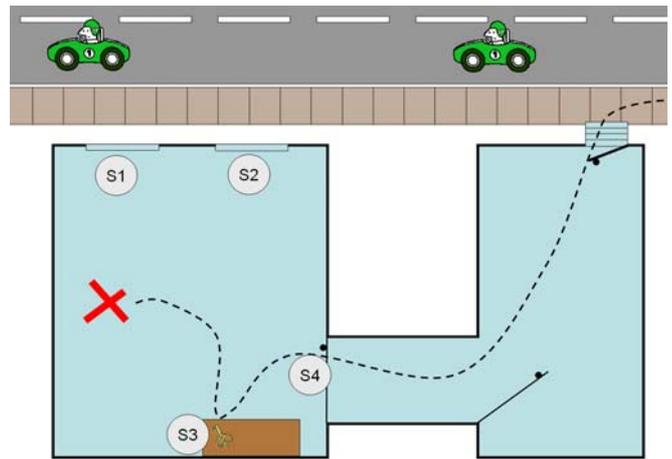


Figure 6: Example Scene.

The applications for such an authoring tool include audio-visual as well as audio-only applications. The main benefits are that it allows the creation of additional sound sources that provide enough information for non-visual applications, such as auditory displays or audio-only games. Other advantages are that the authoring is accomplished within the same environment and using the same media. This simplifies the preview process and the auditory compositing. A challenging task still is to provide enough information for the navigation and orientation within this auditory world. Sighted users need to rethink their auditory perception and gather some experience prior using this system most efficiently.

CONCLUSIONS AND FUTURE RESEARCH

With the increased demand of mobile applications in the near future, other senses come in the range of interest besides vision, VR and augmented reality. Mobile auditory applications have the advantage that they are easy to build, affordable and with the appropriate interaction techniques easy and intuitive to use. The biggest benefit is that no vision is required, making these systems usable by visually impaired as well as by sighted users which need their vision to observe other tasks.

Computer Games are often cutting edge applications and lead the way for new technology. Mobile games already play a large role, including handheld gaming platforms like the GameBoy or cell phone based games. All of them have in common that they implement the classic way of audio-visual gaming. Some audio-only games have already been developed (AudioGames), but mainly with the focus for the visually impaired and are not widespread.

When comparing the information that is perceived by visual and auditory senses, less and different

information is conveyed through hearing than through vision. For real audio-only applications, techniques have to be developed that focus on auditory cues and allow the integration of additional information in the *natural* auditory environment. The authoring software has to support these techniques and allow for an authoring of such environments. Although many auditory authoring systems exist, most of them are developed for the authoring of audio-visual computer games. The authoring within these tools is often limited to the integration of various sounds and music into the game environment. They are insufficient for the increased authoring of non-visual environments as they do not allow the integration of additional information, which is essential for most audio-only applications.

The system that was introduced in Section 4 allows both, authoring for audio-visual as well as for non-visual applications. As an additional benefit, the authoring takes place using the same media and in the same environment. This is a huge advantage, as the authoring process does not break the illusion of being immersed in a virtual, auditory world. This fact should not be underestimated, as the visual authoring of an auditory world is different from non-visual auditory authoring. This is of course different for audio-visual applications. Given an adequate interface, non-visual auditory authoring is superior for audio-only applications. One limitation is that sighted users have to practice this new form of authoring, as the only form of perceiving information is through sound. Beneficial is that non-sighted users are able to use these tools as well. As the system is currently under development, extensive tests and user studies have to be performed in the near future as well as the integration of additional sonification and interaction techniques to improve the non-visual authoring.

ACKNOWLEDGEMENTS

The authors would like to thank Daniel Bergner and Daniel Walz for fruitful discussions and their help in the implementation of parts of this project.

REFERENCES

- (AM:3D 2004) Sven Vestergaard and Preben S. Nielsen, "Website AM:3D", <http://www.am3d.com>, 2004
- (AudioGames) Richard van Tol and Sander Huiberts, "Website AudioGames", <http://www.audiogames.net>, 2004
- (Begault 1994) Durand R Begault, "3D Sound - For Virtual Reality and Multimedia", *AP Professional*, 1994.
- (Cakewalk 2004) Twelve Tone Systems, "Cakewalk", PC.

- (Cubase 2004) Steinberg, "Cubase", PC, 2004
- (Drewes 2000) T. M. Drewes, Elizabeth D. Mynatt, M. Gandy, "Sleuth: An Audio Experience", *ICAD*, 2000.
- (EAGLE 2004) CreativeLabs, "EAGLE" PC, 2004
- (Flür 1976) Wolfgang Flür and Florian Schneider, „Der Käfig“, KRAFTWERK, Germany, 1976
- (Fly3D) Fabio Policarpo. "Fly3D Game Engine", <http://www.fly3d.com.br>, 2004
- (gameCODA 2004) Sensaura, "gameCODA", <http://www.gamecoda.com>, 2004
- (Gardner 1999) William G. Gardner, "3D Audio and Acoustic Environment Modeling", *Wave Arts Inc.*, 1999
- (Goldstein 2001) E. Bruce Goldstein, "Sensation and Perception" *Wadsworth Publishing Company*, 2001
- (Hämäläinen 2002) Perttu Hämäläinen, "3D Sound Rendering and Cinematic Sound in Computer Games", *Technical Report HUT, Telecommunications, Software and Multimedia Laboratory*, 2002
- (IASIG 1997) Interactive Audio Special Interest Group - 3D Working Group, "3Dxp DirectSound 3.0 Extension API", <http://www.iasig.org/pubs/3dxc.pdf>, 1997
- (Klante 2003) P. Klante, "Visually supported design of auditory user Interfaces", *Volume 2 of the Proceedings of the 10th HCI International*, Page 696-700, 2003
- (Maven3D 2004) Emerging Systems, "Maven3D Professional", PC, 2004
- (Menshikov 2003) Aleksei Menshikov, "Modern Audio Technologies in Games", *Game Developers Conf.*, 2003
- (Piringer 2001) Jörg Piringer, „Elektronische Musik und Interaktivität: Prinzipien, Konzepte, Anwendungen“, Master's Thesis, TU Wien, Austria, October 2001
- (Röber and Masuch 2004) Niklas Röber and Maic Masuch, "Interacting with Sound: An interaction Paradigm for virtual auditory Worlds", *Proceedings of 10th ICAD*, 2004
- (SCREAM 2004) Sony Corporation, "SCREAM", *Game Developers Conference*, 2003
- (Soundfactory 2004) CRI Middleware, "CRI Sound Factory", PC, 2004
- (SoundForge 2004) sonicfoundry, "Sound Forge", PC, 2004
- (Speeth 1961) S. D. Speeth, "Seismometer Sounds", *J. of the Acoustical Society of America*. Vol.33, pp.909-916, 1961
- (Strothotte 1995) Thomas Strothotte and Helen Petrie and Valerie Johnson and Lars Reichert. "MoBIC: An aid to increase the independent Mobility of Blind and elderly Travellers" *2nd TIDE Congress*. 1995
- (Syberia 2002) Microids Adventure, "Syberia", PC, 2002
- (Targett 2003) Sue Targett Mikael Fernström, "Audio Games: Fun for all? All for fun?", *ICAD*, 2003
- (Väänänen 2003) R. Väänänen. "User interaction and authoring of 3D sound scenes in the Carrouso EU project", 114th Convention of the Audio Engineering Society (AES), Amsterdam, March 2003
- (Watt and Policarpo 2000) Alan Watt and Fabio Policarpo, "3D Games Vol.1", *Addison Wesley*, 2000
- (XNA 2004) Microsoft Corporation, "XNA development platform", <http://www.microsoft.com/xna>, 20

IMPLEMENTATION OF VRML AND JAVA FOR STORY VISUALIZATION TASKS

X, Zeng, Q. H. Mehdi and N. E. Gough
Games Simulation and AI (GSAI) Centre, RIATec
University of Wolverhampton, Wolverhampton, WV1 1EQ, UK
E-Mail: x.zeng@wlv.ac.uk

KEYWORDS

Story visualisation, VRML, Java, External Authoring Interface, XML

Abstract

VRML (Virtual Reality Modeling Language) is a high level graphic language for describing 3D virtual objects and worlds on Internet. The paper considers the implementation of VRML and Java for story visualization tasks. It focuses on creating and manipulating the properties of 3D virtual objects by using natural language input. We first explore the potential of the binding between VRML and Java technologies for generating the interactive virtual worlds. We then investigate the possibilities and limitations of these methods. Finally, we present an inside look into our Story Visualiser graphic engine, discussing its internal architecture and some of the insights of Java, XML and VRML technology we have gained during its development.

INTRODUCTION

The development of interactive 3D applications is a difficult task. Unlike OpenGL and Direct3D, which are high performance graphic techniques with low-level functionality libraries that can be directly used for rendering, VRML, X3D and Java3d higher-level graphic techniques are based on scene graphs for representing a 3D interactive scene on the Web. The scene and other visual elements are described in a hierarchical structure. Apart from the rendering of a scene graph the interaction of the user with a scene graph is an important part of a 3D application (Wetering 2001). VRML is the ISO standard and has been widely accepted as a central metaphor for presentation, visualization and simulation purposes. It already has extensive usage in medicine, engineering and scientific visualization, entertainment and education. One of the strengths of VRML is it can provide interactivity in real time and there is not an excessive rendering delay. Another powerful feature of VRML is its easy extensibility and flexibility to add new node types and capabilities to

the base language (Ames *et al* 1997; McCarthy and Descartes 1998) The VRML Specification defines a set of 54 built-in nodes that not only define the contents of a virtual world, but also dynamically change their properties using event sending and processing.

The work described here is part of ongoing research into a 3D story visualization authoring system (Zeng *et al.* 2002, 2003; Mehdi *et al.* 2003). Story based natural language was used as the primary input source in this system to construct a 3D virtual environment (3DVE). The natural language processing (NLP) and 3D graphic presentation techniques were integrated to allow construction and manipulation of a VRML-based scene graph in real time. In this paper, we explore the potential of the binding between VRML and Java technologies for generating the interactive virtual worlds. We also investigate the possibilities and limitations of these methods and focus on how to integrate Java, VRML and XML technologies to construct our graphic representation system.

DIFFERENT WAYS OF INTERACTION BETWEEN VRML AND JAVA

The universal acceptance of VRML as the world's first widely used non-proprietary file format for the deployment of behaviour-rich, real-time 3D applications has profound implications for how we envision information (Marrin *et al* 1997b). But VRML is very limited in terms of interactivity and it is not a general purpose programming language, and Java is not a 3D presentation language. However both were designed as web technologies and serve different goals. While Java has the ability to access VRML worlds and has dramatically changed the nature of the VRML itself while enriching and deepening the meaning of the data it encapsulates. The VRML and Java link provides a standardized, portable and platform independent way to render dynamic, interactive 3D scenes across the Internet (Brutzman 1998). For the majority of today's web projects, the marriage of Java and VRML provides the perfect solution (Marrin *et al* 1997b; Lea *et al* 1996). There are two popular approaches to using Java to extend VRML

world: one is internal Java Script Authoring Interface (JSAI) and the other is External Authoring Interface (EAI).

Java Script Authoring Interface

A VRML file consists of nodes that mimic real-world objects and concepts such as various geometries and material descriptions *etc.* *Script nodes* defined within the VRML file are used to program more complex behaviour for a VRML scene. A node can signify or receive events from the user or other nodes from the scene and effect changes in the scene by using ROUTE to send events. The script is the way VRML communicates with outside world, encapsulating the Java code and providing naming conventions for interconnecting Java variables with field values in the scene. Program scripts are miniature applications that contain the logic and interfaced Java classes import the *vrml.**, *vrml.field.** and *vrml.node.** class libraries to provide type conversion between Java and VRML. In this way, we can establish a link between a VRML scene and a Java application by making a handle to a Java.class file in a script node.

In this approach, the SAI provides a suite of classes and methods that enable Java to access its interface Fields and EventOuts, converting between data types and initialise the Java program and to respond to the events when the VRML browser run. In spite of its strengths, JSAI is unable to link with external data because it runs entirely within its plug-ins environment, a universe unto itself (Pesce 1997). As Kimen (1997) claimed: "VRML is- Java does- and the EAI can help".

External Authoring Interface

The EAI is a set of language-independent bindings that provides a conceptual interface between VRML scenes and an external environment. The virtual scene can be controlled via external programs or applets in the case of a web browser hosting the VRML browser. A typical web application consists of VRML browser window and additional controls in a Java applet on a same HTML page. The EAI not only allows manipulation all of the entities that internal scripts can modify. The true power of EAI is in the development of applications that incorporate VRML as only one of the elements in a presentation (Marrin et al 1997a). The creation of custom GUIs, the linking of VRML representations to external data, and even the possibility of multi-user interaction are all made possible in part because of the EAI. These are exactly the types of applications that developers have always wanted from VRML (Larsson 1999). The role and use of

EAI lies in bridging and linking those Java applications to the 3D VRML scene. By using EAI to bind the data in the Java applet to the VRML world, developers are able to create a compelling cost efficient, cross-platform solution.

EAI firstly creates an object reference to the browser, using the methods of that object to locate various nodes which are named using a VRML statement DEF within a scene, once the pointer to a node is obtained. It then creates objects that encapsulate the EventIn and EventOut constructs of that node. Once that is accomplished, manipulating values within a VRML world is accomplished in the same way as SAI. Another feature of EAI is being notified when events are sent from eventOuts of nodes inside the scene. In this case the applet must subclass the EventOutObserver class, and implement the callback() method. The advice() method of EventOut is then passed the EventOutObserver. It permits handling of events from multiple sources, the source being distinguished just according to this value (Marrin 1997). Thus the applet can be notified once something has happened in the VRML world.

DISCUSSION

There is no doubt that Java offers sophisticated behaviours and enormous functionality to VRML worlds. However, whilst JSAI provides a flexible rule-based knowledge representation to handle the internal events of the virtual worlds easily, it has the disadvantage of being entirely contained within the plug-in, and references to Java also remain entirely constrained within the VRML world. This means we must be concerned with programming both Java and VRML aspects, and the performance of this combination is slow. There are few ways to bring the outside data in, so this approach is not suitable for programming control system. In contrast, the EAI has overcome these shortcomings and is useful for systemic node and dynamic outside event handling (Marrin 1998). This conceptual integration allows a Java applet to "query" a plug-in and establish a real-time event-based communication stream with it. This means that potentially any Java applet could invoke and use VRML as a visualization interface, and that VRML worlds can be manipulated through a Java-based interface (Pesce 1997).

Experience has been gained using the EAI approach the course of this research project. However, although EAI offers compelling capabilities to interact with VRML world, a number of limitations have also been encountered. EAI allows new objects to be created on the fly; the new objects can be added as child nodes to any node in the VRML world. But some methods such

as `CreateVrmlFromX(String, URL)` are clumsy (although this also applies to the Scripting references as well) (Campbell 1998). When objects have been created through these methods, the nodes return an array of nodes and are resident in the memory. The DEF names are not accessible and `getNode` is unavailable on nodes that are created dynamically. We can use `TouchSensor` to solve the problem by asking the user to click on the object and then manipulate them, but this solution does not work well for complex objects (e.g. objects which are assembled from many parts). Furthermore, this unfriendly interface is not convenient in this work which uses natural language input. The second solution is to pre-code all of the possible DEFed nodes in a whole Java class and connect them to the corresponding Java variables. Also pre-define all the `eventIn` and `eventOut` for each node and attach them to Java objects with corresponding types. This method can be used for simulations that contain unchanged

this extends the limitation of EAI. This is to protect from a malicious applet storing a virus on the computer. Even a signed applet can access local system resources as allowed by the local systems security policy. But it is not a good practice for the program extension, especially for the program that is still in the early development stage.

GRAPHIC PRESENTATION SYSTEM IMPLEMENTATION

To address the limitations of the binding between the EAI and VRML worlds we have developed a graphic presentation system. This user-extensible authoring system – termed a *Story Visualiser* – is implemented and based on VRML, Java, EAI and XML cooperation. There are three main processing layers: File Layer, DOM (Document Object Model) Layer and Control Layer. Each layer includes several modules and the architecture of the system is shown in detail in Figure 1.

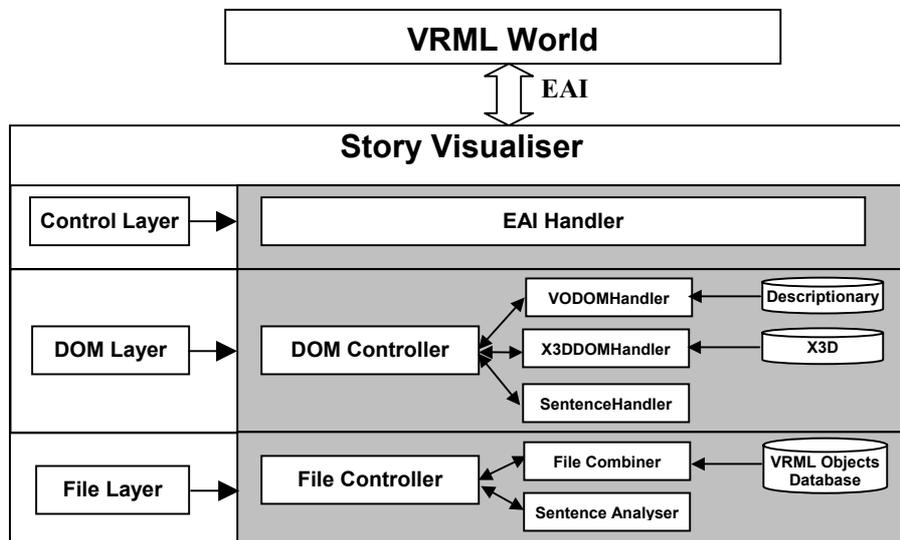


Figure 1. Story Visualiser System Architecture

objects in the scene during whole operation. Clearly, this is another clumsy approach for our purposes because it introduces a massive programming overhead and makes future extension difficult.

Another limitation is that once `getNode` overloads DEF and any particular name might have been multiple used, then only the last occurrence of a given name is accessible (Marrin 1997). The program may thus be confused as more than one identical objects appear in the scene, so the user would fail to manipulate through being unable to get hold of the specific object.

EAI allows Java to interact with VRML world by using a Java applet. However the applet is restricted to the security sandbox, i.e. it is not possible to write to files on a local hard drive, and

The system works as follows. Beginning with the File Layer, an input sentence is analysed by the Sentence Analyser module, the XML formatted semantic representation output is extracted and a call is made to the corresponding objects from the VRML objects database. The objects are combined through a File Combiner module and converted into a set of XML files through the File Controller Layer. Then the output XML data can be operated on by the DOM Layer and passed to the EAI Handle module in Control Layer. This finally enables the Story Visualiser to construct a virtual environment. Next we describe some detail of the modular approach to discover how each of the limitations of using EAI is overcome.

The File Combiner module integrates the objects that correspond to the nouns of the semantic

presentation output from the input sentence. It first reads the base VRML file *root.wrl* from the VRML database, and then read the remaining VRML objects into the memory and outputs a new combinative VRML file which contains all the objects. Once the objects have been merged into a new single VRML file, the problem of accessing the DEF names of the nodes has been addressed. However, given an increase in the number of objects and bearing in mind limitations of the natural language interface, it is necessary to find a way to restrict the DEF names of the objects. Currently, we have tackled this by generating a rule which allows sending and processing events for the specific nodes, *i.e.* change attributes of the object, such as Material node, Texture node and special related nodes (Transform node, etc). For example, the File Merge module automatically change the DEF ball as ball_1, ball_material as ball_1_material, and ball_text as ball_1_texture. If the ball appears again, the DEF name will increment by 1, *e.g.* ball_2 *etc.* Our approach is not only to formulate the DEF names of objects that provide solutions for data accessibility, consistency and efficiency, but also to define the interaction between a VRML world and a database by use of VRML and Java. Another advantage of this approach avoids the problem of possible multiple DEF names. Furthermore, because this module was written by a Java application, this means we can now read the source files, write a new file back to the local drive and address the Java applet's limitation.

XML is used for data presentation and can be used as middleware to integrate legacy systems with other applications. XML defines a standard format for representing and exchanging structured data and can be extended or embedded in Java through the use of a standard API, the Document Object Model (DOM), for managing that data, and the deployment of standard services for generating and viewing XML content. In our system, XML has been used as a major data structure for the data exchange between the different layers, *e.g.* output of the XML formatted semantic representation of the sentence to enable the system to match VRML objects and transport object depictions (*e.g.* adjectives, prepositions) to the Story Visualiser to manipulate VRML scene.

To enable the Java Applet to interact with a VRML scene through EAI, it is necessary to pre-code all the DEFed nodes and related events in the Java file in advance. This approach results in data redundancy and it is also impossible to include all the DEFed nodes in a single file. However, to extract the DEFed nodes from a VRML file is a difficult task. So in this instance, we use X3D (Extensible 3D) as the solution because it is

relative easy to obtain the DEFed nodes. X3D is a 3D standard for the Web that expresses the geometry and behavior capabilities of VRML 97 using XML. It has been proposed by the Web3D Consortium since 1999 and represents the next-generation VRML. X3D is an Open Standard XML-enabled 3D file format that enables real-time communication of 3D data across all applications and network applications. However, the standard is still immature and under review by the ISO. In our system the output of the combined VRML file is converted into an X3D file using the VRML2X3D translator. We then use X3D DOM Handler to extract and store the DEFed names as a tree structure in memory to enable the EAI Handler to access and send or receive events to/from them.

COMPILATION AND EXAMPLE

JavaSoft's JDK 1.3.1 has been used to compile the Java classes. JAXP 1.0.1 is used to parse the XML file. The EAI is using blaxxunClientSDK which is provided by Blaxxun, Inc. For the natural language part, we used NLS API which was developed by the National Library of Medicine. This implementation has been tested on a Windows 98/2000 platform using Internet Explorer 5.0 and 6.0; Netscape 4.75, 6.0.

We present here a simple example to illustrate the way of how the system works. Currently, the system enables generation of the 3D scenes and manipulation of the properties of 3D virtual objects by using natural language input. Consider a storyline that includes the following sentences to describe the environment:

There is a yellow room.
A table is in the room.
There is a green vase on the table.
A book is next to the vase.
There is a picture on the left wall.

Theses are interpreted and presented one by one. The listing shown below presents the XML formatted semantic presentation, and Figure2 shows an output 3D scene from the scene descriptions.

```
<?xml version='1.0' encoding='UTF-8'?>
<sentence-encoding>

<!-- =====>
<np reference="room_1" adjective="yellow"/>
<verb-frame verb="is" object="room_1"/>

<!-- =====>
<np reference="table_1"/>
<verb-frame verb="is" subject="table_1" object="room_1"
preposition="in"/>

<!-- =====>
<np reference="vase_1" adjective="green"/>
<verb-frame verb="is" object="vase_1,table_1"
preposition="on"/>

<!-- =====>
<np reference="book_1"/>
<verb-frame verb="is" subject="book_1" object="vase_1"
preposition="next to"/>

<!-- =====>
<np reference="picture_1"/>
<verb-frame verb="is" object="picture_1,"
preposition="on"/>
</sentence-encoding>
```

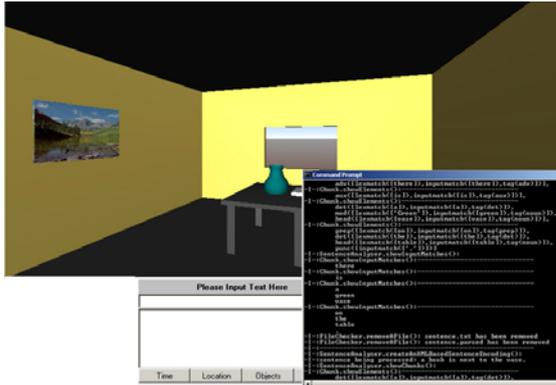


Figure 2. 3D scene generated from sentences

CONCLUSION

The goal of the research described here was to generate a 3DVE by using a simplified story-based natural language input. In this paper, we explained and discussed the different approach of using VRML and Java technologies to generate an interactive 3DVE. The EAI fills in the gaps between the built-in functionality of a VRML world and the programmability of Java through the use of an embedded Applet on an HTML Web page. We then presented the architecture of our system which overcomes the limitations of the EAI by integration of Java, XML, *etc* technologies. The advantage of choosing XML as the primary data structure makes the system easy to extend. Our approach also provides solutions for data accessibility, consistency and avoids redundancy. The example illustrates that the methodology works satisfactorily on generation 3D scene by manipulating the visual features of the VEs. However, the construction of the Story Visualiser system is still work in progress. It can be further improved by developing an instruction interface that will allow users to interact with the 3DVE in real time. It will also be necessary to expand the Descriptive and VRML objects database for more complex tasks.

REFERENCES

- Ames, A., Nadeau, D and Moreland, J. (1997) *The VRML Source Book*. 2nd ed. John Wiley & Sons, Inc. Canada.
- Brutzman, D (1998) The Virtual Reality Modeling Language and Java. *Comm. ACM*, vol. 41 no. 6, June 1998.
- Campbell, B. (1998) Extending VRML2 with Java. *Java Developer's Journal*. Vol. 3, Issue 6. SYS-CON Publications, Inc. VRMLJournal.com.
- Larsson, D (1999) Linking Java with VRML97. *Technique Report*. PELAB, Department of Computer and Information science. Linköping University, Sweden.

EAIFAQ

- <http://www.frontiernet.net/~imaging/eaifaq.html>
- Kimen, S. (1997) VRML Is, Java Does, And the EAI Can Help. <http://vrml.sgi.com/features/java/java.html>. Last access September 2001.
- Lea, R., Matsude, K. and Miyashita, K. (1996) *Java for 3D and VRML Worlds*. New Riders Publishing. Indianapolis. USA.
- Marrin, C., Kent, J., Immel, D. and Aktihanoglu, M. (1997a) Using VRML Prototypes to Encapsulate Functionality for an External Java Applet. http://www.marrin.com/vrml/papers/InternalExternal/Chris_Marrin_1.html. Last access November 2001.
- Marrin, C. (1997) Proposal for a VRML 2.0 Information Annex. External Authoring Interface Reference. Silicon Graphics. Inc. http://www.web3d.org/WorkingGroups/vrml-hisotry/eai_draft.html. Last access August, 2002.
- Marrin, C., McCloskey, B., Sandvil, K and Chin, D. (1997b) Creating Interactive Java Applications with 3D and VRML. A Silicon Graphics, Inc. White Paper. <http://cosmo.sgi.com/developer.html>. Last access November 2001.
- Marrin, C. (1998) EAI Specification. <http://www.web3d.org/WorkingGroups/vrml-eai/ExternalInterface.html>. Last access November 2001.
- McCarthy, M and Descartes, A. (1998) *Reality Architecture. Building 3D worlds with Java and VRML*. Prentice Hall Europe. Hertfordshire.
- Mehdi, Q., Zeng, X., and Gough, N.E. (2003) Story Visualization for Interactive Virtual Environment. *ISCA 12th International Conference on Intelligent and Adaptive Systems and Software Engineering*. California.
- Pesce, M (1997) VRML and Java, A Marriage Made in Heaven. http://developer.netscape.com/viewsource/pesce_vrml2/pesce_vrml2.html. Last access July 2003.
- Web3D Consortium. <http://www.web3d.org>.
- Wetering, H. (2001) Java: A Simple, Extensible, Java Package for VRML. *Proceedings Computer Graphics International 2001*, IEEE Computer Society Press, 2001
- Zeng, X., Mehdi, Q and Gough, N.E. (2002) Generation of a 3D Virtual Story Environment Based on Story Description. *Proc. of 3rd SCS Int. Conf. GAME-ON 2002*, London, 2002.
- Zeng, X., Mehdi, Q and Gough, N.E. (2003) Natural Language Inference Technology for Reasoning Visual Information of Virtual Environment. *Proc. of 4th SCS Int. Conf. On Intelligent Games and simulation, GAME-ON 2003*, London.

Mobile and Multiuser Games

| | |
|---|------------|
| Bartle, R. A. Massively multihero: Why people play virtual worlds | 128 |
| Simatic, M., Craipeau, S., Beugnard, A., Chabridon, S., Legout, M-C. and Gressier, E. Technical and usage issues for mobile multiplayer games | 134 |
| Kim, S. Y., Chu, C.-W. and Lee, E.-H. Development of a 3D game engine and a pilot game for PDA | 139 |
| McCoy, A., Delaney, D., McLoone, S. and Ward, T. Towards statistical client prediction-analysis of user behaviour in distributed interactive media | 144 |
| Thorn, D. and Slater, S. I. MMORPG on Mobile Devices? Considerations when designing distributed adventure games | 150 |
| Spyridou, E., Palmer, I. and Williams, E. Investigating team speech communication in FPS video games | 155 |

MASSIVELY MULTIHERO: WHY PEOPLE PLAY VIRTUAL WORLDS

Richard A. Bartle
Department of Electronic Systems Engineering
University of Essex
Colchester, UK
E-mail: rabartle@essex.ac.uk

KEYWORDS

Virtual worlds, MMORPGs, design, hero's journey.

ABSTRACT

Designers of virtual worlds have known for some time that different people want different things from these creations. Modern designs are therefore broadened to account for playing styles beyond those that the designers themselves prefer. The results are patchy, however: although designers know intellectually what players want, they don't always know emotionally why they want them, and have over- and under-emphasised features as a result. More dangerously, they have deliberately denied some critical player needs in the false belief that to allow them would cause their virtual world to fail.

This paper examines why players play virtual worlds, and identifies a key area where designers are getting it wrong.

INTRODUCTION

Virtual worlds (a catch-all term encompassing MMORPGs, MMOGs, MUDs and a dozen or more other acronyms) are persistent, computer-mediated environments through and with which a number of players may interact simultaneously.

It is widely accepted among players and designers that different players exhibit different behaviours in virtual worlds – that they find different things “fun” (Bartle 1996). Further investigation (Bartle 2003) has demonstrated that:

- Players exhibit dissimilar, but related and enumerable, playing styles.

- Players follow predictable paths through these playing styles over time as they play.
- Progression along these paths amounts to a quest for self-actualisation.
- This is what makes virtual worlds fun to an extent beyond that which can be derived from (other) computer games.

This later work also showed how following a development path through the playing styles was equivalent to the “hero's journey” of myth.

MYTH

In a famous analysis of myths ancient and modern from cultures across the world, Joseph Campbell identified a single template to which they all conformed: the monomyth, or *hero's journey* (Campbell 1949). In this, a would-be hero undertakes a journey to an “other world” of danger and adventure, where normal rules don't apply. Irrespective of the originating culture, the hero's journey follows the same, set pattern through a series of key events that results in the positive transformation of the individual undertaking it.

A hero's journey can thus be regarded as a prescription for self-discovery. If you complete the journey, you're a hero: you have self-actualised the “real” you. Unfortunately, you have to go to an *unreal* place to do so. Thus, rarely can an individual embark on their own hero's journey; they can only reflect on what it may be like, by (through story) identifying with those who have completed it. The movie *Star Wars* (Lucas 1977) follows Campbell's monomyth very rigorously, but the viewer doesn't get to be a hero – Luke Skywalker does.

Virtual worlds are almost unique in their ability to deliver a hero's journey to ordinary, everyday people. They do not do this by putting a character through the hero's journey formula, however: the journey is for the *player*. The journey is real; the virtual world is merely the adventure-filled "other world" in which most of the important events take place.

When a player signs up for a virtual world, what follows goes something like this. They create a character of a different disposition to their own, and role-play it. As they play, they come increasingly to identify with their character, changing their own disposition and that of the character in response to the various challenges that are presented to them. Through a continual process of adjustment and reflection, they are able to make incremental changes to their sense of identity – their feeling of who they are. The end result of this increasing *immersion* is that they and their character align and become one: no longer does the player feel that they are playing a character in a virtual world; instead, they feel that they, personally, are *in* that world.

Note that this is a psychological point of view, not one of narrative theory. What players consciously see as their main goal is actually driven by subconscious goals of which few individuals are fully aware: it's rare that players consider themselves to be on some grand journey of the self – they simply want to have fun. What "having fun" means to each player, however, changes as they progress; the way it changes is precisely in tune with the monomyth. Within a virtual world, the challenges presented are arranged such that each player can always find whatever is right for them at their current stage of development; this leads them naturally to the next stage.

It is important that virtual world designers understand this process, because otherwise they may inadvertently (or otherwise) derail it. As we shall shortly see, this can lead to problems in the long term.

The hero's journey comes in three phases: departure, initiation and return. Departure takes place in the "mundane world" (*i.e.* the real world). Initiation is exclusively in the "other world" (*i.e.* the virtual world). Return concerns the

homecoming to the mundane world from the other world, but takes place in both; for the hero, the other world loses its mythical significance and becomes just another part of the real world. Again as we shall shortly see, this has critical but usually ignored consequences for virtual world design.

It should be noted that although the hero's journey takes a male point of view, it works for women too. There is proposed a related heroine's journey (Murdoch 1990), but this primarily concerns how women (and possibly some men) develop as individuals in the real world; it doesn't involve a journey to an "other world".

THE WORLD OF ADVENTURE

For virtual world designers, the only part of the player's quest for self-understanding that they can influence is that which takes place in the virtual world itself. For the purposes of this paper, we don't have to examine all 17 steps of the hero's journey; rather, we need only examine the sequence of six that are under the virtual world designer's control up to the point where the problems start. These begin with the final step of the departure phase.

The Belly of the Whale

To enter the world of adventure is, for the (would-be) hero, akin to an act of rebirth. This is often symbolised in myth by the hero's disappearance into a womb-like object (a cave, a temple, a belly of a whale) from which they are expelled into the world of adventure.

In virtual world terms, this is character creation. The player gets the chance to annihilate the self and create a new self as whom they will journey into the unknown. If players don't create a character – if they play as themselves – then there can be no hero's journey for them.

The Road of Trials

New-born into the world of adventure, the hero is faced with a series of trials – obstacles that must be overcome if they are to progress. These are usually not too difficult at first (we've all killed the rats), and some can even be failed.

The purpose of the challenges is to teach the hero the rules of the special world in which they now find themselves. Later challenges don't have this function: they get progressively harder, forcing the hero to change to overcome them.

At the end of the road of trials, the player has demonstrated sufficient skill, confidence and maturity to be prepared for what is to come.

In virtual world development terms (Bartle 2003), this matches almost exactly the griefer/opportunist stage. The player needs to find out what the limits of action are. Those who test the physical limits (opportunist) act on the world to see what it allows – the natural laws. Those who test the social limits (griefers) act on the players to see what they allow – the social norms.

The Meeting with the Goddess

In the monomyth, the hero next experiences an unconditional love, of the same power and nature as that which young children have for their mothers. The goddess to whom they give this devotion represents the totality of knowledge: the perfection that once was, that awaits rediscovery. If the hero is not consumed by the knowledge, they are liberated by it.

This step is also known as the *marriage of opposites*. The hero is imperfect, and needs to learn to stop regarding their self in a dualistic way. The goddess is life, but also death; in marriage, the hero is shown to be capable of enduring both.

Although this sounds like just so much psychological flannel, stripped of its symbolism it makes eminent sense: in the light of the new knowledge that the hero is gathering, their self-image begins to coalesce about new points.

In virtual world progression terms, this maps onto the scientist/networker phase. The player has a good enough model of the world and/or its society to be able to interact, rather than merely act. The player actively seeks further knowledge, in order to realise their potential (*i.e.* become complete). Players with a physical bent (scientists) will interact with the world to discover what it reveals – they'll explore. Players with a social bent

(networkers) will interact with other players to discover what they reveal – they'll enquire.

Woman as the Temptress

“Woman” here is a metaphor for the temptations of the hero's mundane life. It can be lust, fear, uncertainty – anything that might distract the hero from the journey. The hero must resist the temptation to return to their old ways; they must decide whether they are pure enough to continue.

This is a point of change. Knowing what lies ahead, the hero rejects (or is repulsed by) their old self, and commits to becoming their new self. In virtual world terms, it marks the difference between gaining knowledge and putting it into practice. Do they want to apply what they have learned, or were things better before they started playing?

Atonement with the Father

This is the most important step of the hero's journey. All previous steps lead to this; all subsequent steps lead away from it. In virtual worlds, it's the “game” period – it's what the virtual world is ostensibly about.

The “father” is the most powerful entity in the hero's existence, personified in virtual worlds as the (lead) designer. The hero wants the father's acknowledgement that they are worthy, but the father only accepts those who have passed all the tests. Because the father's ogre aspect cannot be defeated by those who have not passed the tests, the approaching hero must have faith that the father is merciful, then rely on that mercy.

This is the most transformative of steps, indicating the correction of whatever imbalance of the self drove the hero to the world of adventure in the first place. External validation by the father is symbolic of internal validation by the hero. You make *yourself* the father, by finally abandoning who you *were* and becoming who you *are*. You have the ability to control your destiny: all you need is the recognition that your faith in yourself is justified.

In player development terms, this corresponds to the politician/planner step. Players attempt to meet

the criteria that the designer has set down as the “aim” of the “game”. Players taking the physical approach (planners) act on the world to shape it so as to achieve their goal – they effect change. Players taking the social approach (politicians) act on other players to shape them so as to achieve their goal – they affect change.

There is a huge problem here, however, in that most virtual worlds don’t have a recognised end (which is what an atonement with the designer amounts to). This is especially true of the large-scale commercial worlds. Players progress to the next step only reluctantly, having kept their part of the bargain but feeling frustrated because their achievement has not been formally recognised.

Apotheosis

“Apotheosis” means to become (as) a god. The hero feels peace and fulfilment, their life in harmony with the “other world” and its people. They have an implicit understanding of it; old challenges no longer seem important.

This corresponds to the hacker/friend stage of player development. The player no longer feels the need to compete, control and achieve; they no longer play a game – they play to be who they are. Hackers interact with the world for the sheer joy of knowing it; friends interact with players for the sheer joy of knowing them.

IMPLICATIONS FOR DESIGN

Most game-like virtual worlds facilitate the hero’s journey reasonably well until they reach the end of the atonement step. It’s possible that by paying attention to the symbolism some of the earlier steps could be made more effective (e.g. by siting character creation in some dark, cavernous setting), but on the whole they work just fine.

With the atonement step, however, things fall apart all too easily. This is not through any fault of the players; rather, it’s the designers’ lack of appreciation of what they are designing that is to blame. There are a number of common difficulties, the most significant of which merit explanation.

Lack of Atonement

Players can rarely “win” virtual worlds. This is not because there’s no obvious end-point – it’s usually almost trivially easy to define one, in fact. The problem is that virtual world designers (or those who pay their salaries) are afraid to tell players that they have won for fear that this will cause them to stop playing.

This is ultimately self-defeating. Players are profoundly frustrated by a lack of atonement – they need closure. They go from virtual world to virtual world seeking it but never finding it. They judge all virtual worlds by the standards of the first, even when by all impartial measures the later worlds they visit are superior. The “father” rejects them every time, therefore they reject the father every time, and their disenchantment deepens.

Ironically, the basic design stance that leads to this disenchantment is flawed. It’s *OK to give players atonement* – to let them “win” the “game” – because:

- People actually keep on playing after they’ve “won”. All the evidence from long-term textual virtual worlds points to this. Some people have played the same virtual world for 15 years or more. They don’t leave the virtual world when they win atonement: they take it into their reality.
- People who want to leave a virtual world will stay with it if they can sense a definite point at which it ends for them. If they can’t, they’ll just quit there and then. Seeing light at the end of the tunnel gives them the endurance to carry on until they reach it. They’ll still quit, but they’ll play longer before doing so.
- Even if you don’t buy any of this monomyth argument, is it better for players to leave with positive feelings of success or with negative feelings of frustration? Which kind of ex-player is going to tell their friends to try out your virtual world?

People play virtual worlds to become heroes. However, they can only be heroes in the real

world. To complete their hero's journey, they must be allowed to leave the virtual world. Only this way will they ever wish to stay.

Undeserved Atonement

Praises of the unworthy are felt by ardent minds as robberies of the deserving. (Coleridge 1817)

For atonement to mean anything, it must only be available to those who have passed the tests. If the father can be fooled into granting atonement to those who haven't passed the tests, this makes atonement worthless.

Put another way, players don't like cheating. They don't like anything that undermines their own sense of achievement – and that includes the buying of atonement (or the means to obtain it) using real-world money. Purchasing a high-level character has the same effect on high-level players as purchasing a qualification would have on people who earned their qualification legitimately – it eats at the trust that is necessary for the system to work.

It's not impossible for virtual worlds with commodification to offer their players a hero's journey, but it's rather more difficult.

Premature Atonement

A virtual world that is too easy enables atonement to be obtained before the player is ready for it. Players normally take several months at normal rates of play (which can be quite intense – 2 to 4 hours a night) to reach the necessary level of immersion. If atonement comes too soon (in the scientist/networker phase, for example), then it will feel all wrong.

This implies that virtual worlds need a critical mass of content if they are to be valid. Breadth of content is important to begin with, but it cedes to depth as players progress. With insufficient content, or content of an insufficient level, atonement will be reached before the player can draw any benefit from it.

This also implies that there is such a thing as too much content. Indeed there is, as treadmills demonstrate, but too much is better than too little.

So long as some end is realistically attainable after the player has passed the scientist/networker step but before they've reached the hacker/friend one, atonement will feel acceptable to them.

Lack of Journey

Some virtual worlds have no hero's journey, offering as they do no metric by which to measure success. For many of these, this is entirely appropriate and expected: educational virtual worlds are rarely created to teach the realization of the self, for example. Yet for other non-game worlds, the hero's journey can still apply.

It's an observed phenomenon that if players feel they are progressing along some dimension that the virtual world itself doesn't recognize, they'll come up with their own pecking order for it (Raybourn 1998). This is true even of virtual worlds that are formally social- rather than adventure-oriented.

In other words, if a virtual world is sufficiently separated from real life to qualify in players' minds as an "other world", they will make up their own "game" to drive their activities. Unfortunately, these "games" don't always have an atonement mechanism (although some do, for example in awarding unrestricted build privileges to suitably skilled individuals).

Designers of officially non-game virtual worlds should be aware that some if not all their players may thus nevertheless embark on a hero's journey. To that end, they should have the apparatus in place to grant atonement when necessary – to allow players to feel they've "won" something that was never officially intended to be a "game".

Meaningless Atonement

This is perhaps the trickiest problem facing designers wishing to give their players the complete hero's journey experience. For atonement to mean anything, it must only be given to those who have passed the tests. The sad fact is, however, that not every player is able to pass the tests – not everyone can be a hero.

Sooner or later, it occurs to players even of virtual worlds boasting a winning condition that failure

was never an option. All it takes to keep going is time. Tests may get harder, but they're never so hard that you can't pass them. Anyone with half a brain can plod, plod, plod to "the end", whatever that is. What, then, is the point of trying? The only way you're not going to finish the journey is if it becomes so boring that you lose interest. Atonement is guaranteed for all, it's just a matter of time.

There is a solution to this, but it's controversial: it makes the price of atonement too high for most people to bear. This is the introduction of *permanent death* – the possibility that a character can be obliterated forever as a result of failing a test. Players must create a new character, start again, and attempt to recover their lost self (creating in the process a stronger, new self).

Permanent death as a concept offers many design advantages even without consideration of its monomythical elements. Unfortunately, it offers one disadvantage that completely trumps all these: players *hate* it when it happens to them. They hate it so much that newbies won't even contemplate *playing* a virtual world where they could "lose two months' play" in a single moment (even if by losing it they would then gain twice as much play of a kind more suitable to who they are).

Players say they want to be heroes. What they often mean is that they want to be treated how a hero would be treated. It is only with experience that they realize that the only way this can happen is for them to become actual heroes themselves. If newbies were more up for it, the mere possibility of the permanent death of characters would not be regarded by so many as a barrier to fun.

CONCLUSION

Players play virtual worlds as a means for self-discovery. They do this by subconsciously following a predetermined path – the hero's journey – that the architecture of virtual worlds

opens up for them. Unfortunately, this path is often blocked by understandable but ultimately misguided design decisions. A fuller appreciation by designers of the meaning and purpose of the path's various steps would ultimately benefit both parties: players would be able to finish their journey, and virtual world developers would yet keep their custom.

My advice to virtual world designers is this: give players a meaningful, deserved "win" condition that arrives at the right time, is triggered by a valid measure of mastery and is plod-proof; in return, they'll give you your virtual world.

REFERENCES

- Bartle, R.A. 1996. "Hearts, Clubs, Diamonds, Spades: Players who Suit MUDs" *Journal of MUD Research 1 (1)*.
<http://www.mud.co.uk/richard/hclds.htm>
- Bartle, R.A. 2003. *Designing Virtual Worlds*. New Riders, Indianapolis, IN.
- Campbell, J. 1949. *The Hero with a Thousand Faces*. Bollinger Series 17, Princeton University Press.
- Coleridge, S. Taylor. 1817. *Biographia Literaria*.
<http://www.gutenberg.net/etext04/bioli10.txt>
- Lucas, G. 1977. *Star Wars*. 20th Century Fox.
- Murdoch, M. 1990. *The Heroine's Journey*. Shambhala Publications, Boston MA.
- Raybourn, E.M. 1998. "The Quest for Power, Popularity and Privilege in Cyberspace: Identity Construction in a Text-Based Multi-User Virtual Reality." In *Proceedings of the Western Speech Communication Association Conference* (Denver)
<http://www.cs.unm.edu/~raybourn/moo5d~1.htm>

TECHNICAL AND USAGE ISSUES FOR MOBILE MULTIPLAYER GAMES

Michel Simatic*, Sylvie Craipeau*, Antoine Beugnard*, Sophie Chabridon*, Marie-Christine Legout*, Eric Gressier**

* Groupe des Écoles des Télécommunications; 46 rue Barrault; 75634 Paris Cedex 13; France

E-mail: Antoine.Beugnard@enst-bretagne.fr; {Sophie.Chabridon, Sylvie.Craipeau, Marie-Christine.Legout, Michel.Simatic}@int-evry.fr

** CNAM-CEDRIC; 292 rue Saint Martin; FR-75141 Paris Cedex 03; France

E-mail: gressier@cnam.fr

KEYWORDS

Multiplayer mobile games, technical issues, sociological issues, psychological issues

ABSTRACT¹

This paper presents the results of an undergoing project dealing with issues for mobile multiplayer games. First it takes a technical point of view. It presents our work-in-progress on the following issues: Communication middleware (through a prototype compliant to the *Open Mobile Alliance* specifications), high level communication abstractions (which can be provided to multiplayer games), latency awareness (through a prototype mixing high latency GPRS communications with low latency Bluetooth communications), consistency (taking into account mobile phone limited resources), and databases (investigating three solutions based on grid DBMS). Our work also studies sociological and psychological aspects. After presenting the methodology used for the study, it shows how mobile gaming provides a second skin to its player (giving him the ability to withdraw from others, or to stay present to them), and is a tool for socialisation and appropriating time (by favouring a feeling of mastery). This last point suggests that mobile multiplayer games can only be appreciated if they take into account player's time constraints.

INTRODUCTION

Computer games, whether played on a PC or a console, are among the most commercially successful applications. Now many companies are convinced there will be such a success with mobile games. For instance, *Datamonitor* projects that by 2005, 80 percent of all mobile users in the US and Western Europe will play mobile games at least occasionally (Leavitt 2003). Indeed customers base more and more their decision for buying a mobile phone on the quality of its embedded games. Nevertheless when considering mobile multiplayer games, the success remains confidential. For instance *TibiaME* (mobile version of PC's *Tibia* game) experiences 50 daily players (Nokia 2003b) to be compared to the thousands of MMOG regular players (Woodcock 2004). In order to better understand the

reasons of this lack of success, the *Groupe des Écoles des Télécommunications* has started an internally funded project called MEGA (Mobile MultiplayEr Games Architecture). Its goal is to analyse issues for mobile multiplayer games. This work-in-progress paper presents the first results of this project in the technical field and the usage field.

TECHNICAL ISSUES FOR MOBILE MULTIPLAYER GAMES

Today technical issues for mobile monoplayer games are well known: Limited size for applications (Nokia 2003a), scarce energy resources (Capra et al. 2001)... the main issue being the devices anarchy (Palm 2003). Indeed when PC market is rather homogeneous, mobile market experiences heterogeneity is the rule. For instance depending upon its firmware the same mobile phone can have a different behaviour.

To identify technical issues for mobile multiplayer games, we had a state-of-the-art activity. In addition we provided an inquiry towards game industry actors: developers, editors, technology providers, mobile operators... Until now we got back few answers (some actors consider the subject to be too strategic to answer to the inquiry). Nevertheless it confirms several issues identified thanks to our state-of-the-art: Communication middleware, communication abstractions, latency, consistency and databases.

Communication Middleware

A console/PC world multiplayer game can be run on a single machine as a split-screen application. With a mobile this alternative is not realistic: Developers of multiplayer mobile game have to deal with network communications. If they do not want to move out of their basic trade they may be interested in using a communication middleware.

There are two levels of functionalities in a communication middleware: On one hand, the "intermediation level" which takes care of the management of game communities, forums, high score storage, and on the other hand, the "middleware level" responsible for the communications between modules during game play. Both levels are standardized by the *Open Mobile Alliance* (OMA 2003, OMA 2004).

Some companies (e.g. Terraplay (Terraplay 2004)) propose products taking care of those levels. Nevertheless they are not OMA-compliant. Moreover their cost is not compatible with budgets of small developer studios (a mobile game budget is about 100 k\$ (Nokia 2003a)).

¹ The work presented in this paper takes place in the context of a *Groupe des Écoles des Télécommunications* internal project, done in cooperation with the following partners: CNAM-CEDRIC, France Telecom Research & Development, Pastagames, Université Bretagne Sud.

This is why, in cooperation with Pastagames (Pastagames 2004) and CNAM-CEDRIC, we are developing the prototype of an open source middleware: *GAMing Services Platform* (GASP). Based on OMA specifications, GASP shall offer both functionalities levels with GPRS communications between mobiles and a server for DoJa (iMalin 2004) or MIDP2 (Knudsen 2003) multiplayer games. A version allowing Bluetooth-only communications between mobiles is foreseen.

Communication Abstractions Dedicated to Games

As it happened in other software industry sectors, it is likely that in the mid-term, game developers will use software components to increase their productivity (by automatically taking care of the device anarchy, for instance). In this context communications will be managed by dedicated software components (Cariou 2003).

This is why we have undertaken an analysis of games and game middlewares in order to extract the communication abstractions present in games. Doing so we will be able to specify these communication abstractions and propose architectural variants of their implementations. Game developers would be able to select abstract communication patterns and choose the appropriate implementation depending on the underlying platform or the network architecture as it is proposed in (Cariou et al. 2002).

Latency

Latency is defined by the response time between an action and the materialization of its effect on all players' machines. With First-Person Shooter games it must be less than 100 milliseconds. For Real-Time Strategy games it can be as high as 500 milliseconds as long as the jitter (the variance of the latency over time) is low (Smed et al. 2001). Now with mobile phone networks such as GPRS, observed latency is around 1 second (Nokia 2004).

The first solution to deal with this gap is to develop games compatible with this 1 second latency, e.g. turn-based games (Nokia 2003a).

Another solution is to take advantage of all of the available communication infrastructures to make other game types available. For instance, *Real Tournament* project relies on a wireless MAN consisting of GPRS and IEEE 802.11 hotspots (Mitchell et al. 2003).

In MEGA project, we are prototyping the mixing of GPRS and Bluetooth, by extending zone-based architecture (Matas Riera et al. 2003). In our architecture, we use a tree of mobiles. The root of the tree is the server. It interacts with its children through GPRS connections. Each of them interacts with its own children through Bluetooth connection. If scatternets are provided in the Bluetooth implementation, these "level-2" children can be in contact with "level-3" children, which themselves can be in contact with "level-4" children...

Communications are based on *dynamic multicast* principles (Ramakrishna et al. 2003) with the following addition. A child can send two types of messages: "Global" messages to be sent to all of the machines participating to the protocol whereas "Local" messages to be sent to all mobiles having the same "level-1" mobile as the sender. To send a message, a mobile sends it to its father, which sends it to its own father... If the message is "global", when it reaches the root of the tree, the root is responsible for forwarding it to all of its children, which in turn forward it to all of their own children... If the message is "local", this process takes place as soon as the message reaches "level-1" child.

We believe this architecture is well adapted to games where several groups of players play together. A subtree of mobiles under a "level-1" mobile handles each group. Communications inside the group are "local" messages handled only with Bluetooth, thus guaranteeing low latencies. Communications between groups require GPRS. Their latency can be integrated in the game design so that it does not reduce game experience. We intend to evaluate the behaviour of this architecture regarding latency with experimentation.

Consistency

Two machines participating to a multiplayer game can have at the same time a different vision of the game state because message propagation time is not bounded and messages can be lost. Several algorithms have been proposed to keep game sessions consistent. For example in *Trailing State Synchronization* (TSS), each machine keeps several states of the whole game. If an inconsistency is detected TSS switched the game from the leading state to one of the trailing states (Cronin et al. 2002).

Our goal is to study the applicability of such algorithms in a context of limited resources as with a mobile phone. We are currently experimenting a simple mechanism as mentioned in (Bernier 2001). The main idea is to send messages like "I have moved from x1 to x2; Meanwhile I did this and that" and suppose that clients are trustworthy (which is ensured by GASP platform). One of the main drawbacks of this algorithm is that it can lead to strong inconsistencies of the type "I have been shot by a dead man" (Mauve 2000). Now if we use this algorithm in the context of a game where players cooperate, we avoid such problem. Thus we can take advantage of the qualities of this algorithm: Limited stream of data from mobiles to server and acceptable streams of data from server to mobiles.

Databases

In the context of mobile multiplayer games, requirements for management of persistent data can be important (important number of players, multimedia data...). In MEGA we study this subject according to two directions:

- Characterize the requirements: This direction should have received inputs from the answers to the inquiry. As these inputs are too scarce, we have decided to concentrate ourselves on the game architecture presented previously.

- Evaluation of solutions based on grid DBMS: MMOG games induce high constraints on DBMS (100.000 simultaneous players generate a load of 200 transactions per second on the DBMS (Butterfly.net and IDC 2003)). Clearly mobile multiplayer games are currently far from overloading that much the DBMS they use. Nevertheless we want to study several alternatives to mysql and Postgresql commonly used by small development studios. Thus we have made a comparative analysis of C-JDBC (ObjectWeb 2004), lega@net (Ganarski et al. 2003) and Postgres-R (Postgres-R 2003): By running on grids they allow better performances and fault-tolerance. We are currently making a deeper evaluation of C-JDBC by testing it against TPC-W benchmark (a web site oriented benchmark, but significant enough for interactive applications like games).

This section presented several technical issues for mobile multiplayer games studied in the context of MEGA project. One can notice that there is never a killer solution for an issue. There are only solutions more adapted to certain types of games than other solutions.

Next session takes an interest in usage issues.

SOCIOLOGICAL AND PSYCHOLOGICAL ISSUES FOR MOBILE MULTIPLAYER GAMES

In order to identify the potential usages of interactive games in mobility situations, it is necessary to joint mobility experiences, game experiences, and mobile phone usage.

Understanding usage logics means understanding dynamicity of mobile interactive games usages thanks to the knowledge of mobile usages on one hand and mobile players on the other hand. For instance, mobile phone usage plays a part in daily life looking more and more like an "occupational zapping" (Jauréguiberry 2003); how does it influence the way people play?

Moreover are on-line players the same as mobile players and, if it is the case, how to joint these different usages?

As sociologist and psychologist, our research on these questions studies: daily practices of mobility, how players using mobile phones occupy time and space, how they carry out their social commitments. First subsection describes our methodology. Second subsection joints game practises, time and socialisation. Final subsection presents our first results.

Methodology

We have led one hour-long qualitative interviews with players. We followed classical method to collect players' anecdotes in order to understand the impact of the situation on game practise. Interviews are oriented according to four themes: 1) Game practise (buying act, frequency, duration, type of games); 2) Game experience (motivations, game pleasure, requirements, experiential dimension); 3) Mobility (circumstances, moments which trigger the desire to play) and multiactivity (usage of the mobile and its different

functionalities, mobile and daily life); 4) Sociability (communications with other players outside of the game, forums, communications during the game).

Our inquiry was split into two phases. First phase consisted in interviews inside a population of players, students of Telecommunication engineering schools. Our goal was to apprehend categories of most-used games, criteria pertinent in game practises, and explanatory phenomena in game practises. Second phase concerned a sample of players selected thanks to two game development companies (one develops *Clint* a multiplayer game (Clint 2003), the other one develops games to be downloaded). We selected this sample according to the type of games and the profile of the players. Our goal was to check the impact of game and technology on the practises: 1) Teenagers/adults/man/woman; 2) Large audience games (Tetris, arcade, sport)/ "gamer" games (action, adventure); 3) Embedded games/downloadable games/WAP; 4) Monoplayer/multiplayer.

Game Practices, Time and Socialisation

There are almost no multiplayer online games in France. We identified one (*Clint*) where game phases are sequential and do not group more than two players. This usage leads to a discontinuous time for the player. Only persons very motivated by this game play it.

Generally speaking it seems game practises on mobile take fully place in the time (as it is experienced by the players) and game vector (that is the mobile phone by itself) has a very important role.

In fact the usage of mobile for playing seems tightly linked not only to the actual offer (with its technical and ergonomical limits) but also (and perhaps above all) to the definition and social representation of time and its organisation.

Mobile phone as a game vector: A second skin

To summarize, it is the mobile phone by itself, as a device extending our body, which gives rise of the desire to play: Always accessible, within sight, it seems to invite the player. It appears as a console ever and everywhere available. Thus it is used in the time interstices, scraps of time made available to the player. At the same time, the mobile phone plays the role of a second skin: Its user plays a game to show his withdrawal from others (in bus or in subway), or on the contrary to stay present without fully engaging himself in the relation (the person plays in the presence of his partner in life who watches television). The mobile phone becomes a tool for socialisation.

A tool for socialisation

As seen above practising a game on a mobile becomes a tool for socialisation. One can use it to show how much he is in foreground or background with others. Used in a public life, it allows furtive withdrawals in private life, smoothing out boundaries between public and private lives, as between time at work and off work. It is a tool for socialisation as it allows to direct energies and to keep a level of presence with the

possibility of withdrawing, for instance during work meetings.

Reappropriating time

The activity of playing on a mobile is directly linked with the boredom, boredom during the constrained time, in particular in transport. In an epoch where time is more and more constrained, mobile gaming opens a space of freedom and gives a feeling of mastery. The game takes place in these moments, which last ten to forty minutes. Thus it is obvious that the games can only be multiplayer or without time constraints, which is not the case of MMORPGs that require long durations.

If the game allows an action, this reappropriation will be even more important. Undoubtedly players are looking for that: An action, which favours feeling of mastery that opposes to the permanent uncertainty of our society (Balandier 1988). Mobile phone accompanies unceasing movement of its users. At the same time playing with it gives the ability to build permanence, an individual action, which make sense.

First Results

Until now we have made fifteen interviews with players. Concerning our first phase of interviews (the one with students of telecommunication schools), we can make the following statements:

- They all have a player past: They started to play very young with a console.
- They all play on other kinds of game support (consoles, PC).
- They do not really appreciate multiplayer games and WAP games, because either they are too costly in terms of connection time or they are not practical to use (“One cannot play in the subway”). Some do not know they exist.
- They mostly play to downloadable games of any kind: action games, adventure, arcade, and strategy. They download 2 to 3 games per month. They like easy to use and easy to understand games. They play during short period of times: 10 to 30 minutes.
- As a familiar object, the object “mobile phone” is an “inciter” to play. In daily life situations, it appeals to the player either because it is in the sight of the player or because it is used for another usage.
- Players play during idle time, waiting period, but also in constraint situations: at work, during a meeting or a course.
- Players play also at home on their sofa, in bed... Mobile gaming is also a relaxation moment.
- Two kinds of practises can already be distinguished: Some players may interrupt their activity in order to play because the desire to play takes over current activity; On the contrary, some other players planify their playing activity during the day.

After presenting the methodology used for the sociological and psychological study, this section presented how mobile gaming provides a second skin to its player (giving him the ability to withdraw from others, or to stay present to them), and is a tool for socialisation and appropriating time (by

favouring a feeling of mastery). This last point suggests that mobile multiplayer games can only be appreciated if they take into account player’s time constraints.

CONCLUSION

This paper presents the first results of an undergoing analysis of issues for mobile multiplayer games.

First it takes a technical point of view. It presents our work-in-progress on the following issues: Communication middleware, communication abstractions, latency, consistency, and databases. For each of these issues there is no killer solution, only solutions suitable to given types of games.

Then the paper takes a sociological and psychological point of view. After presenting the methodology used for the study, it shows how mobile gaming provides a second skin to its player, and is a tool for socialisation and appropriating time. This last point suggests that mobile multiplayer games can only be appreciated if they take into account player’s time constraints.

ACKNOWLEDGEMENTS

The authors want to thank the following persons who contribute to the project (and without whom the project would not exist): Raphaël Goumot, Jean-François Haize, Nicolas Pajot (*France Telecom Research & Development*), Assia Ait-Ali-Slimane, Nicolas Auray, Elise Bathany, Bruno Defude, Sven Falempin, Eugeniusz Hetmanski, Van-Tuan Le, Marie-Christine Legout, Romain Pellerin, Bertrand Seys (*Groupe des Écoles des Télécommunications*), Fabien Delpiano (Pastagames), Laurence Tobin (*Université Bretagne Sud*),.

BIBLIOGRAPHY

- Balandier G. 1988. (In french) *Le désordre*. Fayard.
- Bernier Y. W. 2001. “Latency Compensating methods in Client/Server In-game Protocol Design and Optimization.” In *Proceedings of the 15th Game Developers Conference*. (San Jose, CA, Mar. 20-24, 2001).
- Butterfly.net and IDC. 2003. “Butterfly.net: Powering Next-Generation Gaming with Computing On-Demand.” <http://www.butterfly.net/platform/technology/idc.pdf>
- Capra L., W. Emmerich and C. Mascolo. 2001. “Middleware for Mobile Computing”. ACM.
- Cariou E. 2003. (In french) “Contribution à un Processus de Réification d’Abstractions de Communication.” PhD thesis. Université de Rennes 1, école doctorale Matisse (Jun.).
- Cariou E., A. Beugnard and J-M Jézéquel. 2002. “An Architecture and a Process for Implementing Distributed Collaborations.” In *Proceedings of 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC2002)* (Lausanne, Switzerland, Sep. 17 - 20, 2002).
- Clint. 2003. “Clintfighting” <http://www.clintfighting.com>

- Cronin E., B. Filstrup, A. R. Kurc and S. Jamin. 2002. "An Efficient Synchronization Mechanism for Mirrored Game Architectures." In *Proceedings of Network and System Support for Games 2002 (NetGames)* (Braunschweig, Germany, Apr. 16-17). ACM.
- Ganarski S., H. Naacke, E. Pacitti and P. Valduriez. 2003. "Parallel Processing with Autonomous Databases in a Cluster System." Technical report. LIP6, University Paris 6 and Institut de Recherche en Informatique de Nantes.
- iMalin. 2004. (In french) "Introduction au DoJa." http://www.imalin.com/web2/article_affich.php?id_art=137
- Jauréguiberry F. 2003. (In french) *Les branchés du portable*. Puf (May)
- Knudsen J. 2003. *Wireless Java : Developing with J2ME, second edition*. The author's press.
- Leavitt N. 2003. "Will Wireless Gaming be a Winner?" *IEEE Computer Magazine*, Jan.: 25-27.
- Matas Riera S., O. Wellnitz and L. Wolf. 2003. "A Zone-based Gaming Architecture for Ad-Hoc Networks." In *Proceedings of Network and System Support for Games 2003 (NetGames)* (Redwood City, Ca, May 22-23). ACM.
- Mauve M. 2000. "How to Keep a Dead Man from Shooting." In *Proceedings of the 7th International Workshop on Intercative Distributed Multimedia Systems and Telecommunication Services (IDMS)*. 199-204.
- Mitchell K., D. McCaffery, G. Metaxas, J. Finney, S. Schmid and A. Scott. 2003. "Six in the City: Introducing Real Tournament - A Mobile IPv6 Based Context - Aware Multiplayer Game." In *Proceedings of Network and System Support for Games 2003 (NetGames)* (Redwood City, Ca, May 22-23). ACM.
- Nokia. 2003. "Introduction to Mobile Game Development (Version 1.1)." *Nokia forum*, Jan.: <http://www.forum.nokia.com/main.html>
- Nokia. 2003. "TibiaME Case Study (Version 1.0)." *Nokia forum*, Jun.: <http://www.forum.nokia.com/main.html>
- Nokia. 2004. "Multiplayer Game Performance over Cellular Networks." *Nokia forum*, Jan.: <http://www.forum.nokia.com/main.html>
- ObjectWeb. 2004. "C-JDBC - Home Page." <http://c-jdbc.objectweb.org/>
- OMA. 2003. "Gaming Platform Version 1.0 (Draft version)." *Open Mobile Alliance*, May: <http://www.openmobilealliance.org/>
- OMA. 2004. "Gaming Platform Version 2.0: Client/Server Protocol (Draft version 0.8.1)." *Open Mobile Alliance*, Mar.: <http://www.openmobilealliance.org/>
- Palm T. 2003. "The Birth of the Mobile MMOG." *Gamasutra web site*, Sep.: http://www.gamasutra.com/resource_guide/20030916/palm_pfv.html
- Pastagames. 2004. (In french) "Bienvenue sur le site de Pastaga Point Net." <http://www.pastagames.com/>
- Postgres-R. 2003. "The pgreplication Project." <http://gborg.postgresql.org/project/pgreplication/projdisplay.php>
- Ramakrishna V., M. Robinson, K. Eustice and P. Reiher. 2003. "An Active Self-Optimizing Multiplayer Gaming Architecture." In *Proceedings of the Adaptive Middleware Services Workshop (AMSW)*.
- Smed J., T. Kaukoranta and H. Hakonen. 2001. "Aspects of Networking in Multiplayer Computer Games." In *Proceedings of The International Conference on Application and Development of Computer Games in the 21st Century* (Nov.).
- Terraplay. 2004. "The real-time multiplayer network technology." <http://www.terraplay.com>
- Woodcock B.S. 2004. "An Analysis of MMOG Subscription Growth - Version 9.0." <http://pw1.netcom.com/~sirbruce/Subscriptions.html>

BIOGRAPHY

Michel Simatic graduated from *Institut d'Informatique d'Entreprise* and received a DEA in Computer systems from the *University of Paris 6-Pierre et Marie Curie* in 1990. Then he joined telecom manufacturer *Alcatel*. There he first worked as a research engineer. His main subject was to provide high availability to an object-oriented process control supervision system. Afterwards he became a system architect mainly responsible for specifying architectures designed to interface *Alcatel's* products with customers' applications. He finally worked as a team leader. In 2003, Michel Simatic joined *Institut National des Télécommunications (Groupe des Écoles des Télécommunications)* as lecturer/researcher. He gives courses on operating systems, middleware and algorithmic. His research field concerns massively multiplayer games in mobile environment.

Doctor Sylvie Craipeau, professor in sociology, at *Institut National des Télécommunications (Groupe des Écoles des Télécommunications)* is member of *International Sociological Association*, and *Société Française des Sciences de l'Information et de la Communication*. She studies computer games since 2002 (she organized workshop "Internet, jeu et socialisation" in *Groupe des Écoles de Télécommunications* in December 2002). Her other research fields concern the social dimensions about new technologies especially in the space of work, and biometrics. Her main publications are:

- Auray N. and S. Craipeau. 2003. *Les jeux en ligne*, Lavoisier Hermes.
- Craipeau S. and M-C Legout. 2003. "La sociabilité mise en scène, entre réel et imaginaire." In *La pratique du jeu vidéo : réalité ou virtualité ?*, M. Roustan, L'Harmattan.
- Craipeau S., N. Auray and B. Seys. 2003. "Technologies de l'information et de la communication et sociabilité dans les jeux." Report. Groupe des Écoles des Télécommunications.
- Craipeau S. 2001. *L'entreprise commutante, travailler ensemble séparément*, Hermes Lavoisier.
- Bousard V., S. Craipeau, E. Drais, O. Guillaume, J-L Metzger. 2002. *Le socio manager*. Dunod (Manpower price in 2004).
- Craipeau S., G. Dubey and X. Guchet. 2004. "Les usages sociaux de la biométrie." Report. Groupe des Écoles des Télécommunications.

DEVELOPMENT OF A 3D GAME ENGINE AND A PILOT GAME FOR PDA

Sung-Ye Kim, Chang-Woo Chu, Eun-Hee Lee
Division of Digital Contents Research
Electronics and Telecommunications Research Institute
161, Gajeong-dong, Yuseong-gu, Daejeon, Korea
E-mail: {inside, cwchu, eunhee}@etri.re.kr

KEYWORDS

Mobile 3D game, 3D engine, 3D puzzle, PDA device

ABSTRACT

The advancement of computer graphics and hardware technology have nourished handheld or mobile devices to be equipped with fully implemented 3D graphics capabilities such as a 3D game. In this paper, we describe a development of 3D game engine for PDA device based on PocketPC and a pilot game, 3D Puzzle, implemented by our 3D game engine. The engine developed by us shows 8~9 fps speed rendering about 850 polygons with texture under landscape flush mode.

INTRODUCTION

Recent innovation in handheld devices such as PDAs and mobile phones made it possible to enjoy high-quality 3D games contents on them. The 3D games developed for those handheld devices generally fall into genres of simulation, shooting, puzzle and role-playing. The followings are the most recognized 3D engines.

Fat-Hammer's X-Forge(Fat-Hammer) is a 3D game engine for mobile phones, PDA and handheld game consoles. It delivers PC quality gaming environment to handheld devices by providing 2D, 3D graphics API, multi-channel audio capability and so on. Swerve's Superscape engine(Swerve) offers convenient environment such as well defined SDK and authorizing tools to game developers and manufacturer of mobile devices in order to develop various contents. Segundo3D(Ideaworks3D) of Ideaworks3D provides an outstanding 3D rendering performance especially as well as a compression of game data. Moreover, because of a high portability, it is used to porting contents for PC into those for PDA.

As the need of standard is claiming in mobile 3D industry, international standard organizations for mobile 3D such as Khronos Group(Khronos), JSR-184 etc. were founded and active. The Khronos Group, a member-funded industry consortium focused on the creation of open standard APIs, ratified the standard for embedded 3D graphics, OpenGL|ES Ver.1.0 and will announce OpenGL|ES Ver.2.0 for supporting full shading in 2005. JSR(Java Specification Request)-184(JSR-184) is attracting big attention with OpenGL|ES in mobile 3D fields. Moreover, because GSM phone occupying a communication market over the world is

adopting a Java environment mainly, the importance of JSR-184 mobile 3D graphics API for J2ME™, an optimized 3D graphic API in Java environment, is getting increase gradually(KIPA 2004).

In this paper, we describe a 3D game engine and a pilot game developed for PDA in point of implementation. We don't follow the standard for mobile 3D in our development. Our engine is composed of pure C-codes in WindowsCE environment according to constraint in the project and don't use other 3D graphic API. The pilot game was developed to check the performance of our 3D game engine out.

CONSIDERATIONS FOR PERFORMANCE

The core architecture of a 3D game engine for handheld devices is generally identical to that for PC. However, most handheld devices commonly are not getting supports by hardware for 3D acceleration. Therefore, due to this kind of limited hardware condition, it is important issue to improve the performance of 3D engine for handheld devices. Therefore, the 3D engine development for game for handheld devices requires an elaborate level of consideration in implementing algorithms and optimization.

Fixed Point

Although XScale CPU has a co-processor compared with ARM core, it has a weak capability for math operation. Therefore, we need to use a fixed point for floating point operations. In this paper, we used a 16:16 fixed-point real number of 32-bit and changed a decimal point to prevent an overflow or an underflow dynamically at each calculation.

Sorting

When rendering transparent objects, we should use a sorting. In case of rendering opaque objects, a depth buffer to speed up rendering rate is used by checking depth value, Z, for all faces in view frustum. However, for transparent objects, if we use the way like this, we cannot see the opaque object hidden by transparent object from camera. Therefore, in case of transparency, we have to draw objects being far from a camera first. Because the rendering of transparent objects is decreased an effect of depth buffer, it is good to minimize the number of transparent object in your game environment.

Optimization for Scanline

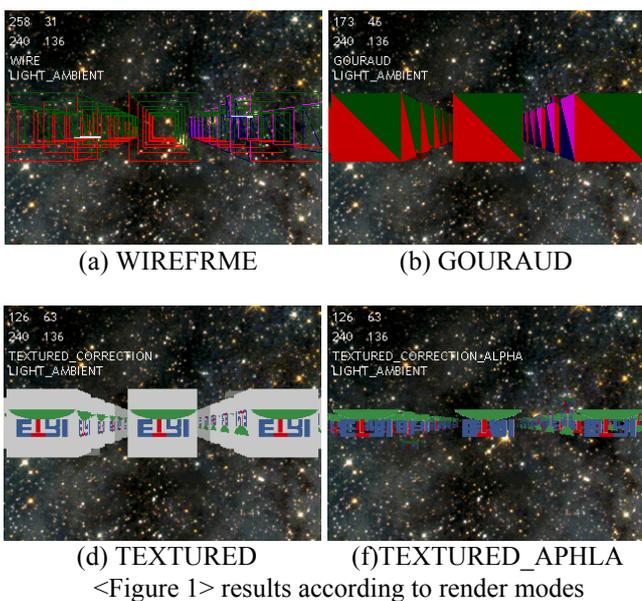
Because a Mits-M400 that is our target hardware for development a 3D game engine and a pilot game has consecutive addressing order in the direction of portrait, a rendering rate is faster on a portrait flush mode than a landscape. The initial setting for scanline is performed by changing function pointer with render mode and flush mode to minimize the loss produced by conditional jumping.

3D GAME ENGINE FOR PDA

Our 3D game engine for PDA has a frame buffer that is 16-bits unsigned short type and 240x320 dimensions and it provides two flush modes that are portrait and landscape. In this paper, we mainly describe the rendering engine in whole engine.

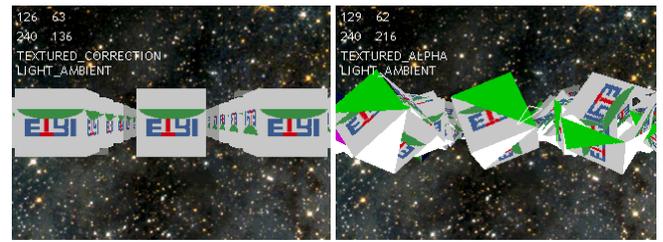
Render Modes

Our rendering engine provides total 11 render modes for raster. They consist of 4 simple render modes like wireframe, flat, gouraud, and texture and their 7 compound render modes. For example, gouraud_alpha, textured_alpha, gouraud_textured, gouraud_correction, textured_correction, textured_correction_alpha and gouraud_textured_corrections. Figure 1 shows that some of screen shots at each render mode.



(a) WIREFRME (b) GOURAUD
(d) TEXTURED (f)TEXTURED_APHLA
<Figure 1> results according to render modes

In addition, our rendering engine provides two render priority modes, engine priority mode and mesh priority mode. Therefore, we can organize game environment freely. An engine-priority mode force all objects in an environment to be drawn by render mode set in an engine and a mesh priority mode make each face to be displayed by its own render mode. Figure 2 shows the results at each render priority mode. Figure2(a) shows an engine priority mode. An engine use a TEXTURED_CORRECION render mode. A (b) shows a mesh priority mode. In (b), although an engine use TEXTURED_ALPHA render mode, each face has GOURAUD, TEXTURED_CORRECTION and ALPHA render modes respectively is draw with its own render mode.

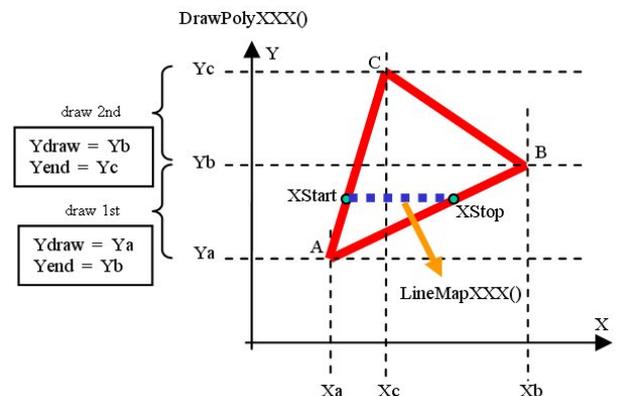


(a) Engine-priority mode (b) Mesh priority mode
<Figure 2> Results of render priority mode

We think these kinds of various setting methods for render mode allow developers to build game environment variously and conveniently.

Rasterizer

A scanline algorithm is the most commonly used rendering algorithm. In figure 3, a triangle ABC made up of vertex A, B and C is drawn by DrawPolyXXX() that is a drawing function for just one polygon according to each render mode. The name, XXX, of the function means render mode like GOURAUD or TEXTURED. In DrawPolyXXX(), LineMapXXX() is called for drawing one scan-line connected with a start point, XStart and an end point, XStop. Figure 3 shows the processes like these. Texture coordinates u and v, a depth z and an intensity of current scan-line are calculated by interpolating those of each vertex linearly. At this time, we use a frame buffer for color intensity and depth buffer for depth information.



<Figure 3> Rasterization of Polygon

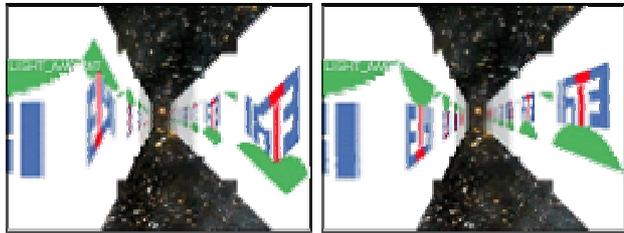
Texture Mapping

Our 3D engine use TGA formatted image data as a texture and we can set up an arbitrary alpha color in game contents. For texture mapping, scan-line functions like DrawPolyTextured() and LineMapTextured() are called. In scan-line functions, we implemented Z-correction to remove a perspective-incorrect produced by linear interpolation of a Z value.

Z-Correction

The visible differences between perspective correct and incorrect interpolation is most noticeable when looking at the texture mapped quads as figure 4(a). To interpolate

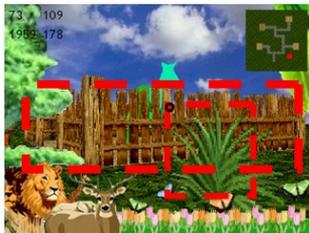
colors and texture coordinates correctly, we interpolate v/z and $1/z$ instead of interpolate u and v for texture mapping or compare z value for depth buffer in scan-line functions. Because our engine use a fixed-point real values, more elaborate validity check is needed. Figure 4 shows the result of the differences between perspective correction and perspective incorrection.



(a) No Z-correction (b) Z-correction
<Figure 4> Result of Z-Correction

Alpha

The order of rendering transparent objects is very important to an engine. A mesh data or a chromakey in a texture can include whether any object has an alpha property. Our engine provides 100% and 50% transparency. Figure 5 shows the result after sorting of transparent objects. There are two objects, which have alpha textures as you can see figure 5.



<Figure 5> Rendering after sorting transparent objects

Buffer Management

Our 3D game engine has a frame buffer, a depth buffer and ID buffer. A frame buffer and a depth buffer are all 16-bits unsigned short type and 240x320 dimensions. To minimize an access to memory, a frame buffer and a depth buffer share the same memory address. Therefore, in case of no sorting we can use two buffers by just one access in scan-line functions. Because PDA devices do not support hardware acceleration for 3D rendering, it is import to minimize the number of access to memory. An IDbuffer is used for user interactions by picking with stylus on the screen of PDA.

PILOT GAME: 3D PUZZLE

We developed a 3D Puzzle game as a pilot game using our game engine. 3D Puzzle is an extension of 2D puzzle game. The objective of 3D Puzzle is to complete the game environments, which have imperfect objects. When game user starts the 3D Puzzle, he selects a category. The category is a set of objects, which have common properties such as animals and vehicles. If a game user selects a category, he can navigate a 3D game world, which is composed of the

puzzle objects and background environments. For example, if user selects the animal category, the gamer goes into the zoo environments, which has elephant, monkey, giraffe etc. In 3D game world, the game user navigates, finds the imperfect puzzle objects and completes the object. If the gamer completes all puzzle objects, he can select another category.

The 3D Puzzle has 5 screen states as in Table 1. Each state has its own initial buffer values and window procedures. Table 1 presents the scene and the user's activities of each screen state.

<Table 1> Screen states of the 3D Puzzle

| Screen State | Functionality |
|------------------------------------|--|
| Start (P_INIT) | -Shows categories in small 3D view port -User selects a category |
| Moving 3D game world (P_INIT2NAVI) | -When the gamer selects a category, 3D view port is gradually expanded to full screen. |
| Navigation (P_NANI) | -Shows 3D game environments -The gamer navigates the game world and selects an imperfect model. |
| Solving puzzle (P_GAME) | -Shows the whole object and fragments. -The gamer examines the whole objects and pick the incomplete part of the model to fit the fragment. |
| Option (P_OPTION) | -The gamer setup options and read help. |

Figure 6 shows some screen shots of the 3D Puzzle. Figure 6(a) is the initial scene. If the gamer selects start button or up-centered 3D view port, the scene is converted into 3D game world such as Figure 6(b). Before completing a model, it is rendered without textures. If the gamer puts the fragments properly, they are textured as Figure 6(c). When all fragments are properly placed, the gamer can see the suitable scene as Figure 6(d).



(a) Initial screen



(b) Before completing a giraffe



(c) Solving puzzle



(d) after completing a giraffe

<Figure 6> Screen shots of 3D Puzzle

Game Data

The game data of 3D Puzzle are composed of geometric data (*.msh), game configuration data and puzzle data. Table 2 represents the game data used in 3D puzzle.

<Table 2> Game data of 3D Puzzle

| | |
|----------------------|---|
| -game.init | -Initialization data |
| -zoo.ctg | -Category configuration |
| -zoo.msh | -Geometric game world data of the category |
| -elephant.msh | -Puzzle model data |
| -elephant.grp | -Fragment data |
| -elephant_piece0.msh | -Geometric data of 0 th fragment |
| -elephant_pieceN.msh | -Geometric data of n th fragment |
| -elephant_sl.msh | -LOD data of the puzzle model |
| -zoo.ui | -User interfaces |

The initialization data of 3D Puzzle (*.init) include the number of the category, the category configuration file names and the 3D game world geometry file names. The category configuration file (*.ctg) has an initial position of the virtual camera for game world navigation, the puzzle models and their LOD mesh and puzzle fragment information. The mesh information for 3D game world, puzzle models and fragments are represented in the *msh* file supported by our game engine. The fragments are composed of some adjacent polygons assigned by a special editor. The correspondences between fragment meshes and puzzle model is defined in the fragment data (*.grp).

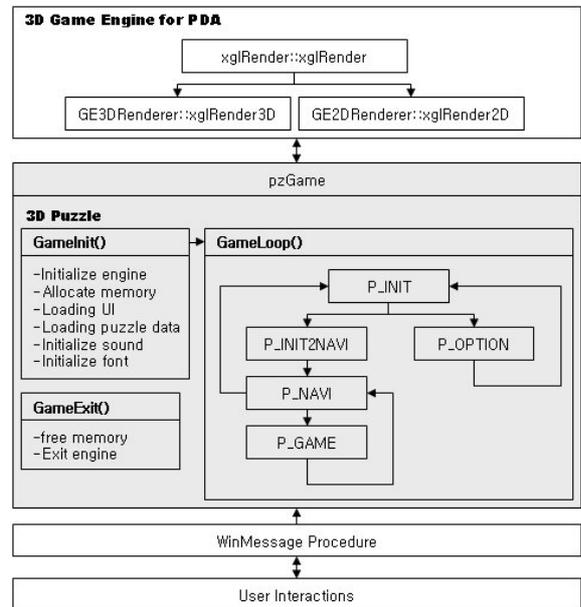
Execution Architecture

The execution architecture of 3D Puzzle is shown in Figure 7. It has three modules such as initialization, game loop and finalization. When the game is started, the game engine is initialized and the several game data are loaded. While the gamer plays 3D Puzzle, the game loop module processes events and changes screen states as in Table 1 and Figure 6. When the gamer exits from the game, the finalization module returns memories and stops the engine.

Functionality

The pilot game, 3D Puzzle, is based on our game engine. The functionalities of the game engine used in this game are summarized in Table 3. When the gamer navigates 3D game world as in Figure 6(b) and (d), the game engine must process the puzzle models as well as the environmental meshes.

The game engine has to process a small number of polygons to increase frame rate and produce a reasonable scene. As in common 3D PC games, 3D Puzzle uses LOD based on the distance between the position of the model and the virtual camera for navigation. The category configuration file (*.ctg) includes the number of LOD level and the distance for exchanging meshes.



<Figure 7> Runtime structures of the 3D Puzzle

<Table 3> the functionalities of 3D Puzzle

- Landscape flush mode (320X240)
- Camera Transform & View port Control
- Shading
- Texture Mapping (with Z-correction)
- Alpha Texture
- Billboard
- Depth Sorting
- Bitmap Font,
- 2D Primitives
- 2D Sound (Background and picking sound)
- LOD
- Stylus Picking
- User Interface
- PDA Button control

User Interface

The user interface of 3D Puzzle is made using a user interface editor in part of the game engine. This GUI editor creates the initial values of the frame buffer, depth buffer and ID buffer for picking. This value reduces time in clearing buffers each frame.

Figure 8 shows initial buffer values when the gamer navigates 3D game world as an example. The initial values of the frame buffer include a border frame, a mini-map and the sky as in Figure 8(a). Because the border frame and the mini-map must be shown every frame, the depth values for these parts are minimum values as in Figure 8(b), in red. In

this example, there are no GUI components, so the initial ID buffer values are all zero.



(a) frame buffer (b) depth buffer (c) ID buffer
 <Figure 8> Initial buffer values of the 3D game world



<Figure 9> the final screen of 3D Puzzle

RESULT & PERFORMANCE

According to the render modes, sorting or no sorting and static or rotation condition, full statistics rendered with a mesh data used in figure 1 are summarized in Table 4. The data in figure 1 is composed about 480 polygons. We examine these performances on Mits-M400 PDA of XScale GL3 Asm under horizontal flush mode. Although we developed some assembly codes to speed up scanline and math operations, they weren't used for this comparison. In the case of being used portrait flush mode, we can get an improvement of 2 fps rather than a landscape flush mode is used. In addition, when we add our assembly codes to our engine, the performance of our engine grow about 2~3fps. Finally, we get 12~17fps rendering rate under the textured alpha render mode with Z-correction that is regarded as the most general render mode used in 3D game.

<Table 4> Comparison of speed at each render mode

| Render mode | Sorting | | No Sorting | |
|----------------------|-----------|----------|------------|----------|
| | static | rotation | static | rotation |
| Wireframe | 30 | 18 | 30 | 21-22 |
| Flat | 20 | 13 | 17 | 11-12 |
| Gouraud(G) | 18 | 12 | 15 | 10-11 |
| G + Z-correction(Zc) | 18 | 12 | 14-15 | 10 |
| G + Alpha(A) | 17 | 11 | 12-13 | 9 |
| Textured(T) | 14 | 10 | 9-10 | 7 |
| T + Zc | 12 | 8 | 8-9 | 6 |
| T + A | 13 | 9 | 9 | 6 |
| T + Zc + A | 12 | 8 | 7-8 | 5 |
| G + T | 13 | 9 | 9 | 6 |
| G + T + Zc | 12 | 8 | 8 | 5-6 |

Figure 9 shows the final screen of our 3D Puzzle. User can play it with stylus and buttons of PDA. In figure 9, 3D Puzzle was composed a zoo category model of about 500 polygons and 4 animal models of about 350 polygons totally. After visibility check, we can see about 150 polygons on screen at once. This pilot game is performed with 8~9fps rendering rate at least under landscape flush mode without supporting by assembly code.

ACKNOWLEDGEMENT

This work was supported by grants from the Samsung electronics company (Development of 3D game engine for PDA).

REFERENCES

- (Fat-Hammer) X-forge, <http://www.fathammer.com>
- (Swerve) Superscape, <http://www.superscape.com>
- (Ideaworks3D) Segundo3D, <http://www.ideaworks3d.com>
- (KIPA 2004) An industrial white paper for Digital contents, 2003. *Digital Contents Technologies and Standards*, Korea IT Industry Promotion Agency.
- (Khronos) Khronos Group, <http://www.khronos.org>
- (JSR-184) Java Specification Request 184: Mobile 3D Graphics API for J2ME™, <http://jcp.org/en/jsr/detail?id=184>
- (MIDP) Mobile Information Device Profile(MIDP), <http://java.sun.com/products/midp/>

BIOGRAPHY

The first author is a researcher at the division of a digital contents research in Electronics and Telecommunications Research Institute in Korea. She was received a B.S. in computer science and engineering in 1998 and a M.S. in 2000 from Chung-Ang University in Korea. Her research interests include 3D rendering, global illumination, inverse rendering, image-based lighting and 3D game.

TOWARDS STATISTICAL CLIENT PREDICTION – ANALYSIS OF USER BEHAVIOUR IN DISTRIBUTED INTERACTIVE MEDIA

Aaron McCoy*, Declan Delaney^, Dr. Seamus McLoone* and Dr. Tomás Ward*

*Department of Electronic Engineering, ^Department of Computer Science

National University of Ireland Maynooth

Maynooth, Co. Kildare

Ireland

E-mail: amccoy@eeng.may.ie

KEYWORDS

Distributed interactive media, Torque game engine, Time-series data analysis, Behavioural modeling, Statistical client prediction, Networked multiplayer games, Dead-reckoning.

ABSTRACT

Distributed interactive media such as networked multiplayer computer games offer users the opportunity to interact and share experiences within a virtual environment. More often than not, these interactions are required to be performed in real-time, a constraint which poses problems given the underlying network capabilities used to transmit information. In these real-time distributed systems, the amount of information that needs to be shared between participants in order to maintain complete game-state fidelity is too large. As a result, trade-offs must be made over what information requirements are necessary to maintain a level of consistency that will provide adequate quality of interaction for the users. One possible solution to this problem is the use of statistical modeling techniques that attempt to capture the individual behaviour of system users. These models can then be used to predict the likely future behaviour for the users, thus reducing the shared information requirements. In this paper we present some preliminary analysis of the behaviour of users within a distributed interactive application, with a goal towards future work of attempting to develop and incorporate statistical models of user behaviour for the purpose described above.

INTRODUCTION

Distributed interactive media (DIM) such as networked multiplayer games are prone to quality-of-interaction and scalability problems as a consequence of non-ideal communication infrastructure characteristics such as network latency and bandwidth. This well-known problem is generally dealt with through careful entity state update procedures that filter the game-state based on criteria such as client relevancy or state changes. An example of the former is area-of-interest techniques (Singhal and Zyda, 1999) while examples of the latter are delta-compression (Van Hook et al. 1994) and dead-reckoning (IEEE 1993). In addition QoS techniques are sometimes used to ensure that all participating clients are given sufficient network resources to meet quality-of-interaction criteria (Internet2 2004). The communication of game state changes is the key issue in all of the above and the games industry-standard techniques are based chiefly on variants of dead-reckoning, which is an example of an entity state extrapolation mechanism. Rather than updating entities

over the network once per simulation loop (which we will refer to as the game loop) all clients in the DIM maintain a local model, usually a linear extrapolation, of entity dynamics. This model is only updated when the client responsible for the entity determines that the difference between the true entity state and that of the model as used by all other clients has deviated by some pre-defined threshold amount. Only this update then needs to be transmitted to all the participating clients hence reducing the number of packets required to maintain a tolerable fidelity across the DIM.

Such a technique obviously helps reduce bandwidth requirements and therefore aids in scalability. More subtly it also aids in the reduction of communication latency, one of the key factors in maintaining a high quality of interaction for the user. This is apparent if we look at the individual components which make up latency in distributed interactive media for any particular link between two participating clients i and j :

$$T_{ij} = \tau_c + \frac{K_{ij}}{B_{ij}} + O_{ij}$$

where K represents the generation rate of state information during the global gameloop in bits per second and B is the bandwidth of the link to the particular client in question. τ_c represents the physical propagation delay. O represents all other processing overheads. Obviously through an increase in link bandwidth or a reduction in K the latency can be reduced. It is through such information rate reduction techniques such as in dead-reckoning that latency problems can be dealt with in DIM.

Recently in an attempt to further improve the power of entity state update mechanisms using the concept of state extrapolation, a technique known as the hybrid model approach has been proposed (Delaney et al. 2003). This concept is very powerful and yet quite simple. In this paradigm entity state are extrapolated based on a combination of low order short term extrapolation and longer term statistical inference. Essentially the technique relies on extrapolating state changes based on previous examples of state behaviour in similar circumstances. In the absence of good information on typical state changes for an entity, the model is switched to simple low order extrapolation as in dead-reckoning. By switching between the two models, hence the term hybrid model, entity states can be extrapolated further than currently possible under dead-reckoning and other entity state extrapolation schemes. Further, in the absence of good statistical information, long-

term heuristic models can be used in lieu as provided by the DIM designer. Consequently the technique has superior performance to dead-reckoning alone as demonstrated in (Delaney et al. 2003). However for the technique to realize its potential in the field of DIM it is imperative that techniques and methods should be developed to first of all recognize, and second of all represent, such statistical models. All work in this area so far has concentrated on demonstrating the concept for fixed statistical spatial models that naturally arise out of static navigation tasks in typical DIAs (Marshall et al. 2004). We are currently studying the possibility of automatic recognition of statistically similar behaviour among entity dynamics in important classes of DIM.

In a previous paper we have shown that patterns of behaviour emerge for human-human interaction in such DIM (McCoy et al. 2004). However, in an attempt to bridge the more solid statistical categorization that arose out of fixed spatial models with such patterns, an intermediate study has been conducted in which the reactive coupling between human agents has been loosened through the investigation of interaction between human users and finite-state machine-driven agents (known as ‘bots’). This paper reports on work done on this area so far in exploring such human user behaviour in these multiplayer computer games. It is hoped that through the analysis of such behaviour, categorization can be determined with the goal in mind that statistically similar patterns of behaviour can be recognized and ultimately exploited to predict future behaviour. This would allow pre-emptive gamestate changes ahead of time and so reduce latency problems in such DIM through both update packet rate reduction and perhaps pre-emptive transmission.

TEST ENVIRONMENT AND TEST SCENARIOS

We use the Torque Game Engine from GarageGames (Marshall et al. 2004) to construct simple test environments and test scenarios. This allows us to perform experiments in a relatively controlled manner, and provides us with the ability of recording information. During a user’s interaction within the test environment, various data is time-stamped and collected in a log file for subsequent analysis. Firstly, 3-dimensional positional data for each user is recorded at regular sampling intervals (at the rate of at least 1 Hertz, but usually higher), allowing us to reconstruct a user’s positional state over time with respect to that of another user and any events that occur. This positional data is represented as time-series datasets for analysis, examples of which can be seen in the results and data analysis section. Secondly, direct interface control interactions (i.e. keyboard and mouse button presses) performed by the user to control their player onscreen are time-stamped and recorded whenever they occur (this consists of primarily movement and weapon firing instructions). This allows us to reconstruct a user’s control sequence of actions with respect to any events that occur. Given the limited complexity of our test environment described below, the only events which we are concerned with here are both weapon firing events and disabled events (where one user is disabled by weapons fire from another), and these are recorded and time-stamped when they occur.

The test environment that we used for the experiments reported in this paper is a simple enclosed environment consisting of several building and tower like structures along with sets of trees and rocks. These provide some visual stimulation for the users and also helps obscure their view in certain areas, forcing them to move about. The experiments are performed in the style of a First-Person Shooter (FPS) game, whereby users interact with the environment as though they were looking through the eyes of their player (see Figure 1). FPS games are one of the most popular genres of games currently in the market, and their networked multiplayer capabilities make them an obvious choice for research into distributed interactive systems in general.

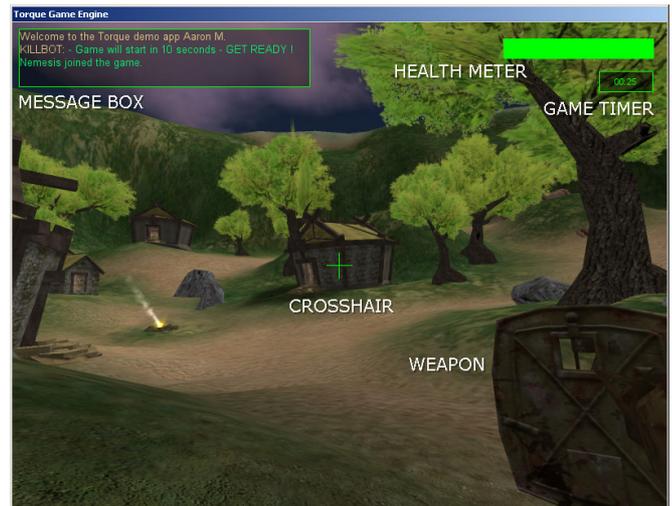


Figure 1: In-game screenshot of the test environment.

Test Scenario 1

This was a simple test scenario set up with the intention of analyzing the behaviour of a user towards a pseudo-dynamic goal. At the beginning of the game, the user is spawned randomly inside one of the buildings. In addition, a computer controlled opponent (i.e. the ‘bot’) is spawned randomly at one of the predefined pathnodes that were placed beforehand throughout the environment. These pathnodes join together to form a path network, and this path network is traversed by the bot pathnode-by-pathnode (a representation of this path network can be seen in Figure 2). The bot is set on a looping run so that all it does is constantly follow the path over and over again – no artificial ‘thinking’ or reacting occurs for the bot. The goal for the user in this case is to disable the bot a specified number of times using their weapon before the specified time-limit runs out. The user is encouraged to score as high as possible over a number of runs such that more typical focused behaviour is exhibited rather than carefree wandering through the environment. This is so as to replicate the typical conditions under which FPS games operate. Typical behavioural patterns we would expect to see should be target seeking, pursuit and firing action. Such classes of behaviour are useful labels (albeit not always clearly distinguishable) and have been applied with some success in our previous work (McCoy et al. 2004).



Figure 2: Path network used for navigation by the bot.

Test Scenario 2

This test scenario was set up with the intention of analyzing the user's behaviour towards a bot that had limited reactive capabilities towards their own actions, namely the ability to fire its own weapon and damage the user. It was set up in exactly the same manner as described for test scenario 1, with the exception that the bot is given a sensor field that can be used to detect the presence of the user. If the user comes within the bot's sensor field and within direct line-of-sight (LOS) of the bot, the bot is instructed to fire its weapon in the direction of the user. If the user goes out of LOS of the bot or out of its sensor field radius, then the bot is instructed to stop firing its weapon. The movement capability of the bot is fixed as in test scenario 1, meaning that it never deviates its position off of the path network on which it navigates through the environment, regardless of whether it is currently firing its weapon at the user or not. Again, the goal for the user in this case is to disable the bot a specified number of times before the time-limit is reached. Unlike test scenario 1 however, it was now possible for the bot to win the game by disabling the user the specified number of times. The motivation for this test scenario is to determine if any resultant defensive or evasive behaviour can be recognized in addition to those behaviours stated for the previous scenario.

RESULTS AND DATA ANALYSIS

In this section, we shall present a specific selection of results taken from data that was collected for users of varying 'expertise' with regard to not only this particular test environment, but also with computer games in general and FPS specific games. Each user was asked to play a number of successive games in each test scenario, and they were given a specific score that they had to reach to complete the game.

Test Scenario 1

Figures 3-5 present state data over time for several different users of varying ability. In each figure, the first 3 subplots present positional data for the user (Player 1) and the bot (Player 2), and are broken into x, y and z coordinates and

plotted separately. Overlaid on each of these are solid vertical lines representing each point in time where the user disabled the bot, and these lines allow us to segment the data into partitions. The final subplot within each figure uses solid vertical lines to represent each point in time where the user fired their weapon. In Figures 9-11, we have plotted the relative position of the user with respect to the bot (i.e. taking the bot's position as the origin for each time-step). Each of the data partitions are plotted individually, with a dark circle representing the starting position of the user, and a dark 'x' marking their end position (i.e. when they disabled the bot).

From visual inspection of Figure 3 (novice user), we can see several areas where the positional data of the user and the bot seem to correlate highly, with one essentially 'following' the other one. This corresponds to a pursuit strategy, where the user has found the bot and is now pursuing them with the intention of disabling them (from time 50 to 100 for instance). It is interesting to note that this is less pronounced in the data for the advanced and expert user. The primary reason for this is that these users tend to wait in a central area and let the bot come to them rather than engage in pursuit, due to the fact that they often quickly learn the fixed movement pattern of the bot along the path network. As a result, their movement patterns tend to display significantly less variation than that of the novice user.

Also, they tend to take less time disabling the bot than less experienced users. This is evident from the shooting events seen in each figure, where we see that the novice user tends to fire their weapon in large spreads, while the more advanced users fire shorter, accurate bursts that tend to disable the bot quickly, as evidenced by the correlation between the shooting events and disabled events. Obviously if the bot had learning behaviour or was controlled by another expert user such a fixed strategy would only provide short-term results, as Player 2 would soon adapt to deal with this. Indeed it is this constant interplay between various strategies in the human-human case that makes such online games both enjoyable for the user and difficult to predict for researchers (McCoy et al. 2004).

Another point of interest is the correlation of the distance between user and bot, the firing events of the user, and the bot being disabled. In most cases, we can see the positions converging close together right before the bot is disabled, indicating the user moving closer towards the bot. This is particularly evident from inspection of Figures 9-11, where we can see the convergence of the user's trajectories relative to the bot (where each trajectory starts as a dark circle and works its way towards a dark 'x' marker). This agrees with our intuitive notion that in order to disable the bot quicker, we should get closer to them (particularly in this case where the bot does not fire back). We also notice the variance in the trajectories that appear under different circumstances. For instance, long-winding trajectories typically represent some random or wandering type behaviour for a user, whereas shorter more convergent trajectories would often represent attack-type behaviour. Of interest are the trajectories that appear to both diverge and subsequently converge rather sharply. These often represent cases where a user has

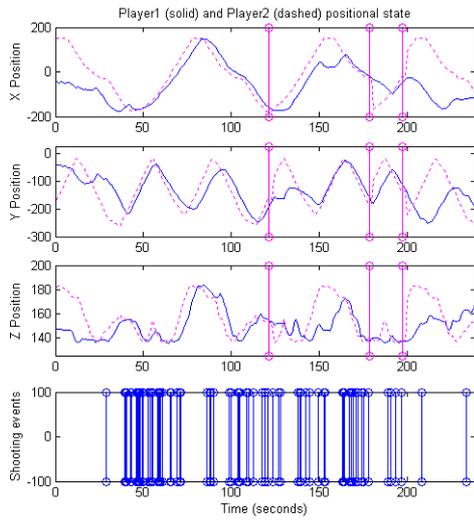


Figure 3: State data over time for a novice user.

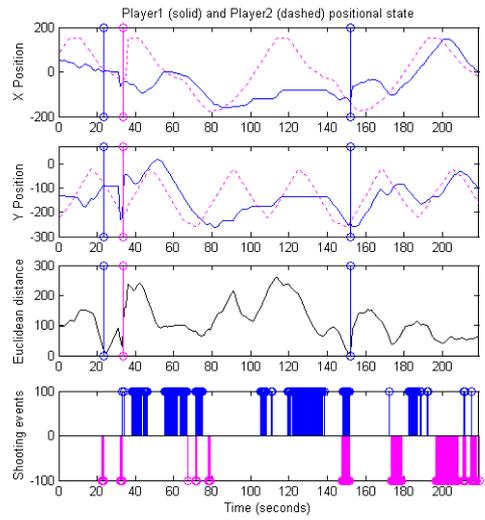


Figure 6: State data over time for a novice user.

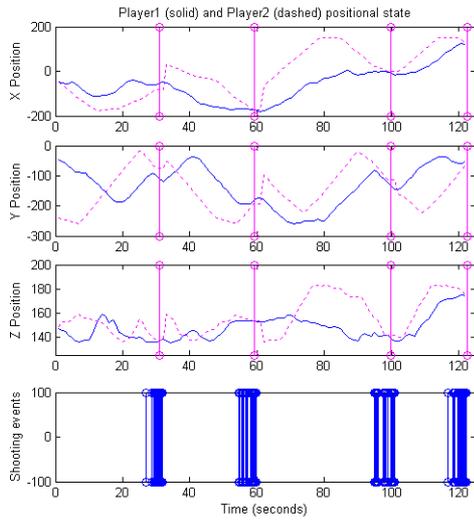


Figure 4: State data over time for an advanced user.

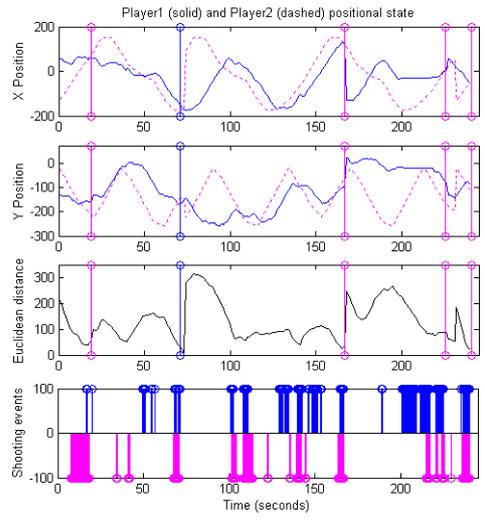


Figure 7: State data over time for an advanced user.

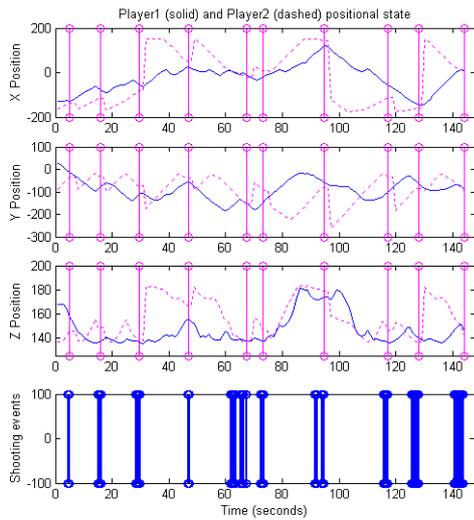


Figure 5: State data over time for an expert user.

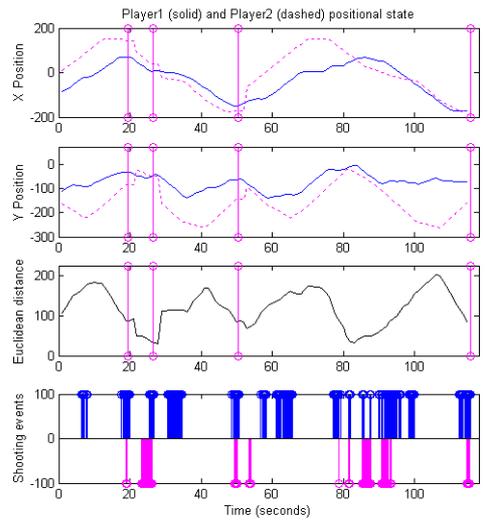


Figure 8: State data over time for an expert user.

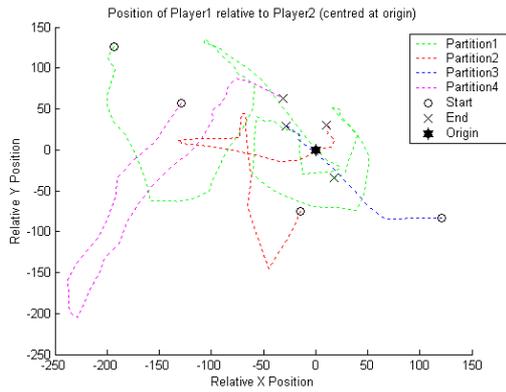


Figure 9: Trajectory for novice user relative to bot.

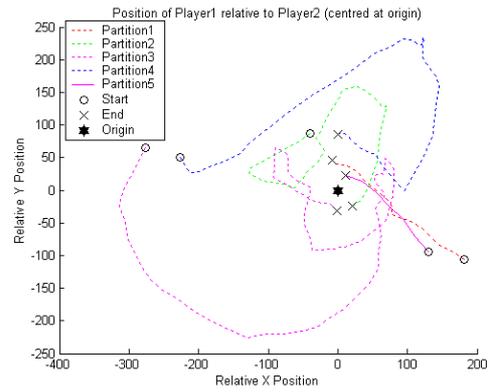


Figure 13: Trajectory for advanced user relative to bot.

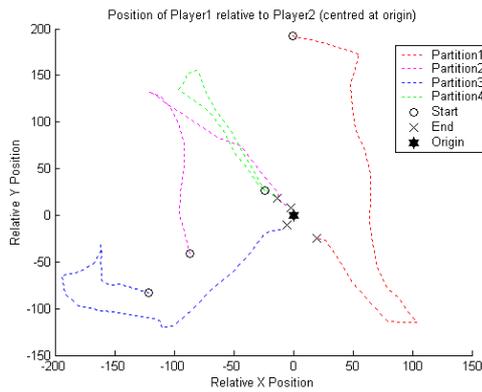


Figure 10: Trajectory for advanced user relative to bot.

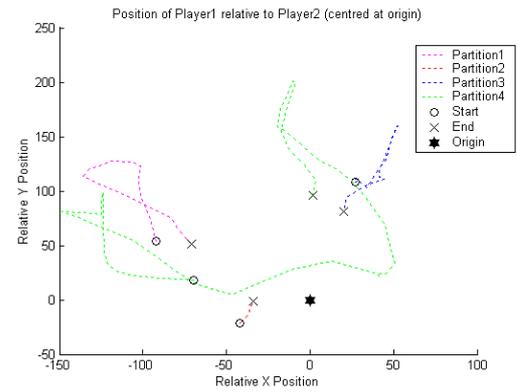


Figure 14: Trajectory for expert user relative to bot.

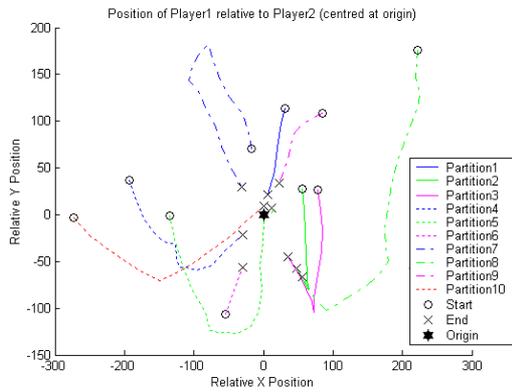


Figure 11: Trajectory for expert user relative to bot.

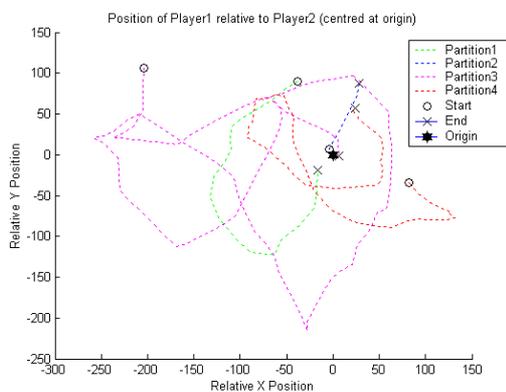


Figure 12: Trajectory for novice user relative to bot.

engaged the bot but not disabled it, and having learned the bots predictable movement strategy, chooses not to pursue it but instead chooses to wait in a suitable area for ambush.

Test Scenario 2

Figures 6-8 present state data over time for users of varying level of ability as in the previous section. However, here we have chosen to omit the z positional data subplot in favour of a subplot showing the 3-dimensional Euclidean distance between the user and the bot at each time step. As before, the first two subplots of each figure represent the x and y positional data for both the user and the bot, overlaid with disabled events for *both* (unlike the last section, which only showed disabled events for the bot). Finally, the last subplot shows shooting events for *both* user and bot (where the shooting events for the user are above the center line and the shooting events for the bot are below the center line). Figures 12-14 present plots of the user's position relative to the bot for each timestep, as detailed in the previous section.

From inspection of the plots, we can see that in general the distance between the user and the bot tends to converge before a disabled event occurs, implying the user moving closer to the bot in an attempt to increase their chances of disabling it. This is again evident from the plots of user trajectories relative to the bot (Figures 12-14), where we can see the convergence of the trajectories (although in the case of the expert user, it is less pronounced). These disabled events provide natural partitions of the data due to the

random respawning of a player within the environment after they are disabled, leading to jumps in the positional state (random respawning is a very common system used in FPS game in general).

In the case of the novice user (Figure 6), we tend to see less of the pursuit strategy that was so prominent for test scenario 1 (Figure 3). This is quite evident from the subplot showing shooting events, where we see bursts of firing spread apart, indicating more rapid engagements between user and bot rather than prolonged pursuing. Part of the reason for this is that because the bot now has the ability to fire back, users are much more cautious about how they approach the bot, and often like to hide behind cover and attempt to 'ambush' the bot as it navigates along the path network. It is interesting to note the variation in distance between user and bot for the case of the expert user, where we can see an extended period of time (from 50 seconds onwards) during which distance converged but subsequently diverged, coupled with a reasonably high degree of shooting events. This would indicate that the user engaged the bot and subsequently broke off their attack, but did not engage the bot in direct pursuit. Rather, having learned the bot's movement pattern, they most likely waited for the bot to return on its course and then re-engage it, at which point they finally disabled the bot. This is consistent with the results shown in Figure 14, where we can observe the divergence of the user's trajectories coupled with the subsequent convergence, indicating a section of time that most likely involved multiple periods of interaction between user and bot.

Also evident is the fact that the end points of the expert user's trajectories (marked by dark 'x' points) have a higher spread as opposed to both the novice and advanced users (Figures 12 and 13 respectively) – this would indicate greater accuracy on the part of the expert user at disabling the bot with his weapon.

CONCLUSION

From the results already shown for a number of users it is clear that certain patterns of behaviour do emerge. In both test scenarios correlated dynamical behaviour occurs, indicating that the respective entity states cannot be statistically independent. In intuitive terms this means for example that if the target player changes velocity dramatically it is often the case that the pursuing player will do likewise. This rather obvious observation could be exploited in entity state extrapolation through predicting such changes where the conventional paradigm of dead-reckoning would have had to transmit a packet indicating a velocity change. It may even be possible to preempt shooting events which conventional techniques have no possibility of predicting.

In another step towards modeling the human-human interaction that is so important in DIM another intermediate step will be taken in which the bot can exhibit more human-like behaviour. This can be as simple as a hunting and pursuit behaviour where the bot having spotted Player 1 will attempt to close the distance before attempting to disable the

opponent. Such behaviour should elicit more interesting defensive or evasive behaviour in the human user that will be important to analyze.

Further analysis along these lines should yield insight, tools and results that will allow better and more comprehensive analysis of human users interacting in the same environments. Consequently it will be possible to make better guesses about what such users may do next and hence achieve quality of interaction and scalability benefits for the distributed case.

ACKNOWLEDGEMENT

This material is based upon works supported by Enterprise Ireland under grant no. SC/2002/129/.

REFERENCES

- Delaney, D., Ward, T., and S. McLoone. 2003. "Reducing Update Packets in Distributed Interactive Applications using a Hybrid Approach". 16th International Conference on *Parallel and Distributed Computing Systems (PDCS 2003)*. August 13-15, Reno, USA, pp.417-422.
- IEEE. 1993. IEEE Standard for information technology – protocols for distributed simulation applications: Entity information and interaction. IEEE Standard 1278-1993. New York: *IEEE Computer Society*, 1993.
- Internet2. 2004. WebSite: <http://www.internet2.edu>
- Marshall, D., McCoy, A., Delaney, D., Ward, T., and S. McLoone. 2004. "A Realistic Distributed Interactive Testbed for Static and Dynamic Entity State Data Acquisition". In *Proceedings of The Irish Signals and Systems Conference 2004*. June 30 - July 2, Belfast, Northern Ireland, pp. 83 - 88.
- McCoy, A., Delaney, D., McLoone, S., and T. Ward. 2004. "Investigating Behavioural State Data-Partitioning for User-Modelling in Distributed Interactive Applications". To appear in *proceedings of The 8-th IEEE International Symposium on Distributed Simulation and Real Time Applications*. October 21-23, Budapest, Hungary.
- Singhal, S. and M. Zyda. 1999. *Networked Virtual Environments: Design and Implementation*. First Ed, Addison-Wesley, pp. 195 - 213.
- Van Hook, D.J., J.O. Calvin, and D.C. Miller. 1994. "A protocol independent compression algorithm (PICA)". *Advanced Distributed Simulation Memorandum 20PM-ADS-005*, MIT Lincoln Laboratories, Lexington, MA.
- ## BIOGRAPHY
- Aaron McCoy received his B.Sc. degree from the National University of Ireland, Maynooth in 2002, specializing in computer science and theoretical physics. His main areas of interest are distributed systems, networked virtual environments and the use of artificial intelligence in interactive games. He is currently studying for his PhD at NUI Maynooth in the area of user modeling in Distributed Interactive Applications.

MMORPG ON MOBILE DEVICES? CONSIDERATIONS WHEN DEVELOPING DISTRIBUTED ADVENTURE GAMES

David Thorn and Stuart Slater
Games Simulation and AI Centre, School of Computing and Information Technology
University of Wolverhampton, UK

d.c.thorn@wlv.ac.uk
s.i.slater@wlv.ac.uk

KEYWORDS

Adventure Gaming, Wide Area Mobile Gaming,
Games Design, Multiplayer Gaming, Gaming

ABSTRACT

This paper is primarily a review of the differing platforms & technologies available for the development of MMORPG (massively multiplayer on-line role playing games) by developers. Current and future technologies that may assist researchers and developers to create distributed mobile games are discussed in relation to PC migration to mobile platforms. The paper begins with an introduction to the genre, its history and the direction the genre is currently heading. This is followed by generic elements from PC based adventure games including game-play that could be easily migrated to the mobile platform. It continues by examining the attraction of the genre to the games players and how technology is changing the perception of the games, detailing some of the latest technologies available for mobile phones, PDA's and next generation handheld devices by Nintendo and Sony. These technologies are presented with their potential limitations and unique features to allow researchers & developers a more informed choice when deciding which platform would suit their proposed project. This takes into consideration meeting the visual, auditory and input challenges, game play and of course the target market. Particular attention has been given to the extensive multiplayer options that are implemented on a range of platforms with an evaluation for multiplayer gaming on a mobile device.

INTRODUCTION

As the market for PC and console games slows, publishers are looking to other mediums such as mobile phones and PDA's in order to attract traditionally non-games buyers to the market. This drive along with the rapid technological advancement in mobile computing and communications has led to ground breaking technologies linked to mobile devices such as the incorporation of dedicated graphic chipsets by NVIDIA on some mobile phones (NVIDIA 2004). For the consumer this has meant a very short move from 2D games that mirrored 80's classics, to the

development of 3D games such as FIFA 2004 (EA Games). The growth of mobile gaming is clearly reflected in many aspects of the industry including the jobs being advertised and the emergence of mobile phone games charts in industry weekly publications such as MCV (www.mcvuk.com). It is safe to say that the mobile games industry is seeing both a growth in sales & technology that in some respects mirrors the console growth of the late 1990's.

As the number of mobile gamers grows, so is their seeming desire for easy to use, fun and non-intensive games, clearly shown from the high growth sales of arcade classics such as Pacman (Namco). This is probably based on the mobility and short usage patterns of such devices e.g. whilst on train journeys or waiting for appointments. In parallel with a growth in mobile gaming is a growing market place for distributed gaming across broadband and other mediums. Unfortunately at present most games downloadable onto phones and PDA's (Personal Digital Assistants) are single player games with localised content, but this is changing as gamers seem to be looking for a greater challenge by interacting with other gamers across the Internet and other mobile transmission mediums, one such area is in the PC adventure game genre which has one of the largest pay to play markets.

This primary focus of this paper is to discuss the area of mobile adventure gaming including its history and unique selling points, current and future mobile technologies, demographics of the market and the differences in usability from more traditional solo play gaming to offer a compelling argument for more research in this area.

THE PAST & PRESENT OF ADVENTURE GAMES

Adventure games are one of the oldest genres within the games industry with roots traceable back to 1976 and "Colossal Cave Adventure" written by William Crowther and Donald Woods. This and other early examples of the genre such as the Zork series were completely text based and involved user interaction via

the typing of predetermined verbs or actions, the game play was based on the premise of the gamer progressing through the game and solving puzzles.

The first adventure game to feature any kind of graphics was Ultima in 1980 (California Pacific), game play was still consistent with previously released adventure games with the added bonus of limited images of landscapes and characters. As graphic technology on computers improved through the early 1980's games started to appear with greater use of graphics such as the Hobbit. By the mid 1980's adventure games featured both greater interaction for the gamer with the use of mice and improved graphics. Games such as Bards Tale (EA) allowed both game character advancement and control of multiple game characters, this was in parallel with a growth in non-computer role playing games such as WARHAMMER (Games Workshop) and BattleTech (FASA). By the early 1990's games such as the Monkey Island series (LucasArts) and the Simon the Sorcerer series (AdventureSoft) still relied on the main interaction being performed by using the mouse to select the allowed verbs from a panel on the screen and then using the point and click interface to interact with the environment. By the late 1990's the market for console and PC games had grown to an extent that gamers now primarily bought games based on the graphics on the back of the box and so a reliance on better and more interactive graphics drove the market forward with games such as Baldurs Gate and Dungeon siege (Microsoft) pioneering interactive adventure games allowing improved game play through greater visual interaction. Moving forward to 2004 and a drive for easier networking and fast internet connections through broadband is predominant in the marketplace, an increasing number of gamers are looking to the interactivity, community and challenge from other human opponents as opposed to scripted AI opponents.

MMORPG

Adventure games for the PC market follow set formulas governed by the genre and in many ways reflect adventure board games such as Dungeons and Dragons (TSR). Games such as Neverwinter Nights (Bioware) and Baldurs Gate (Bioware) utilise the rules of Dungeons & Dragons (the non-computer role playing game), which drastically reduces the need for design into rule systems, worlds, stories and characters as they rely on over 20 years of tried and tested documentation and interactive commercially successful gaming.

Adventure games have grown in recent years into both more involved single player games and online multiplayer experiences. Some games are built purely to be played on-line against other gamers often based

on either fantasy worlds such as the Lord of the Rings style games with mythical creatures or futuristic science fiction settings such as Planetarion, and Hyperiums. These games generally have little or no actual graphics, other than a basic GUI (Graphic User Interface) and instead rely on tactics and planning for gameplay rather than the 'hack and slash' style games found on consoles. These games do not generally follow a set plot, the general idea is to provide the gamers with a back-story, the opportunity to create or join a community and very simple goals to achieve.

Through this blend of interactivity and a varied number of players, games often become fuller experiences lasting months (Planetarion) or even years (Hyperiums, Warriors 2). These games are often paid monthly to the games provider and are known as 'Massive Multiplayer Online Role Playing Games' (MMORPG for short). A growing market in MMORPG is a much more technologically advanced gaming system utilising chat windows, 3D graphics and immersive environments games such as Eve Online which use the main functionality of standard Internet Relay Chat client allowing multiple discussions to be tracked by the players simultaneously. This is coupled with virtual worlds that can be huge in scale allowing for a greater number of players to be online in the game simultaneously without the game becoming crowded. Remembering that each player is paying a monthly fee, the greater number of online gamers the greater the market share, therefore the support for greater interaction and the creation of more immersive environments is of paramount importance to developers.

IMMERSIVE MOBILE GAMES

There are 5 generic elements to computer games design (Howland 1998), they are the graphics, interface, game play, story and sound.

When these elements are combined successfully the experience starts to become immersive for the gamer (Slater 2002). Whether the *graphics* are 2D or 3D is unimportant depending on the genre, but the graphics traditionally are the main selling point of many games on consoles and PC. The *interface* between the player and the game is important because if the graphics look realistic then the players will expect the controls and events to be well thought out and user friendly. *Gameplay* is often overlooked in some games but is considered to be the fun factor, playability and difficulty of the game. The *story* is important in so much that it supports the gameplay and adds to the immersion. Developers often use cut scenes to support the story line, but this is much more difficult with bandwidth limitations of MMORPG and is often partially solved by the use of Flash(Macromedia)

movies or introductory movies on disk that are viewed prior to interaction with the game. *Sound* is the final part of the immersion that allows the user to audibly hear character speech or hear music playing to help build an immersive environment.

When developers consider portable devices such as phones and PDA's for games development, the design elements listed above that combine to create immersive experiences become more difficult to implement fully due to both limitations with technologies such as GPRS, Wi-Fi and Bluetooth (bandwidth limitations) and lower target hardware resources such as memory, screen size and processing power. The interface is also limited on mobile devices many having little in the way of support for games. Hardware interface devices such as keyboards are a luxury on PDA's such as the IPAQ 4350. This PDA provides all of the required hardware for mobile gaming with the inclusion of Wi-Fi, Bluetooth and the Microsoft Windows PocketPC 2003 operating system but the high cost and low availability rule it out as a commercial games platform. Newer dedicated mobile gaming devices such as the PlayStation Portable and the Nintendo DualScreen look set to provide Wi-Fi gaming capabilities, but as with mobile phones and PDA's the memory capacity and clock speed of the integral processor are limited. The PlayStation portable has identical controls to the normal joy pads provided with PlayStation 1 and 2 which improves user interaction through common control associations (Sony 2004). The Nintendo DualScreen again boasts a directional pad and four control buttons, a common implementation for the increasingly popular Game Boy hand helds.

Online games designed for mobile devices should have similar gameplay elements to traditional PC games which include preventing repetition and allowing progression through some type of advancement system which starts easy for the gamer, moving onto progressively harder challenges as the gamers skills and confidence grow. The artificial intelligence system utilised for believable NPC's (non player characters) must also be considered carefully because of the limited processing power of mobile devices. AI in online server based games do not suffer quite as badly with hardware limitations but the need to send this information to the portable devices has to be considered. The story element is not particularly an issue as this can be enhanced with the inclusion of screens to at least show a textual description of the back plot and what the character is involved in. Sound poses more of a problem with portable devices but the technology along with graphics is improving dramatically and many phones can utilise MP3's easily.

The design should incorporate these factors whilst still providing atmosphere for the game in terms of background music or sound and the careful use of sound effects whilst in the game must be linked back to the hardware limitations and immersive factors for the gamer.

MOBILE GAME TECHNOLOGY

Pay to Play adventure gaming is done exclusively away from the mobile platforms via PC's and using the world wide web many using bespoke client systems which utilise the internet protocol suite to allow players to compete on a global scale. The limited penetration of mobile gaming is due to a number of factors. Firstly, only 8% of the mobile games revenue for the year 2000 was from wireless play of all genres with a predicted growth of 32% by 2005 (Game-Research 2002). Secondly many mobile games are re-releases of 80's classics offering very little in the way of new game play or on line experiences, though it has to be said that the games charts being seen in many charts such as MCV clearly show the demand for classics such as Pacman being very high. If the 80's games such as Pacman can be ported easily to mobile platforms then why not adventure games such as Zork? Using SMS technology, it is possible for players to interact with each other whilst following a plot, automated text messaging service can be set up to send players the details about their progress and whether there is another gamer in the virtual world they can interact with. The gamer could then be given the option to either move from the location, interact with the other players at that location (again via the SMS 'server') or interact with the location itself – searching for treasure, opening doors, fighting virtual enemies or indeed other players. All of this can be done by utilising verbs as seen in early text based adventures such as Zork. If more immersive or graphical adventure games were required then the use of services such as picture and multimedia messaging (MMS) could be incorporated. These technologies could improve playability for many gamers allowing them to 'see' their surroundings and other player's avatars. Although animation is possible it's use in real-time events such as battles or fully animated conversations is almost impossible with the current transmission mediums.

A recent advance in mobile gaming is the creation of location-based services, which are available on platforms such as the Nokia N-Gage which allow players to utilise mobile networks to create a multiplayer gathering without the need of personal computers or the related networking hardware. Location based services (LBS) are an example of this, although the technology was not developed with the gaming industry in mind, it provides a framework that

can be used by games developers. LBS technologies utilise various means to pinpoint the longitude, latitude and altitude of the cell. The most basic LBS denote the use of GSM cells, which vary in size and do not cover the entire population. These cells can be used to pinpoint other players in the same geographical location as the gamer attempting to find opponents. GSM cells are not a standard size and thus cannot give a constant sample size to generate games with. A far more accurate method utilises GPS satellites that have a far greater accuracy but need specialised receivers that are not currently economically viable to include in most mobile phone handsets. Another LBS that can be used by games developers to enrich their projects is short-range positioning beacons. Unlike both GSM and GPS, these are private circuits that utilise Wi-Fi connections and particularly Bluetooth technology. The interoperability of Bluetooth services with other platforms such as Personal Computers and PDA's gives a distinct advantage over the other mediums as discussed above mainly because of the reduced cost to the consumers who would merely have to pay monthly fees rather than expensive phone charges. Microsoft Windows CE (Pocket PC) and the J2ME from Sun Microsystems have the capability to utilise GPS.

CONCLUSIONS

Online adventure gaming in comparison to other areas of the games industry such as console games sales is still in its infancy though the uptake of high speed services such as broadband is helping to improve market share. Some games such as *Neverwinter Nights* and *Diablo II* are purchased in a retail store and there is no additional charge to play online. Other games such as *Ultima Online* and *Everquest* require little or no bespoke software but there is a charge made at regular intervals to use the servers that are provided by the developer or publishing house. Both styles of games can be written in a multitude of programming languages such as browser based games using PHP, JavaScript and ASP. In relation to technology, interactive adventure games such as *Eve Online* provide a bespoke client system that connects to servers based on the geographic location of the player and has topped ten thousand players in the same arena at the same time in April of 2004. With the client not needing any other software to run, the level of immersion and presence of other players is greatly increased, as the player can feel more in tune with the environment that they can see on their systems. The inclusion of a bespoke client in some games does not wholly detract from those that use browsers or other third party software in which their games run. Although the immersion from a visual standpoint is not as impressive, these games tend to rely more on imagination, gameplay and plot and less on graphical

interfaces and impressive effects. As the adventure game genre tends to be more based around the story than the visual detail this leads to the conclusion that utilising the browser style of online gaming would be more productive as an initial prototype on which to test the ideas of the developers. It is these client systems which may well be suited to portable devices such as PDA's and phones as much of the processing can be done server side and only updates sent to the devices.

The advantages of the mobile platform include its versatility in terms of allowing gamers to play wherever they are when they feel the desire to play. It also includes the mass market of mobile phones that has yet to reach a state of saturation and the multiple different techniques that can be used to provide the interactivity. The disadvantages can be seen as its limited scope in terms of serving many gamers simultaneously, the difficulty in creating interactive methods for the devices and the need with MMORPG for server based resources to provide the full experience to the gamer.

AUTHOR BIOGRAPHY

David Thorn is currently a student at Wolverhampton University. He is also a Microsoft Student Partner and working on a Microsoft supported project into developing a games framework to allow multiplayer games across mobile devices.

Stuart Slater is currently working as a Senior Lecturer in IT and Computing at the University of Wolverhampton, and a member of the "Games Simulation and Artificial Intelligence" research group (GSAI). His research currently involves both developing emotion systems for NPC's in games and the development a cross platform (PC & Mobile) game frameworks for teaching games development.

References

- Bioware. www.bioware.com. Last Accessed June 2004
- Blizzard Entertainment. *Diablo II Homepage*. <http://www.blizzard.com/diablo2/>. Last Accessed July 2004.
- CCP. *Eve Online Homepage*. <http://www.eve-online.com/>. Last Accessed July 2004.
- Crawford, C. (1982) *The Art of Games Design*.
- Eladhari, M. (April 2003) 'Trends in MMOG Development' Game-research.com, http://www.game-research.com/art_trends_in_mmog.asp. Last Accessed June 2004.

Epic Games. *Unreal Tournament Series Homepage* <http://www.unrealtournament.com/>. Last Accessed June 2004.

Evol Interactive, AB *Warriors 2 Features Page*, <http://www.warriors2.com/default.asp?iframe=3>, Last Accessed June 2004

Game-Research.COM (March 2002) *Wireless Gaming*, <http://www.game-research.com/wireless.asp>, Last Accessed June 2004.

Howland, G. (August 1998) “*Game Design: The Essence of Computer Games*”. <http://www.lupinegames.com/articles/essgames.htm>. Last Accessed August 2004.

Hyperium. <http://www.hyperiums.com>. Last Accessed June 2004.

I4U Future Technology News (March 2003) “*Your Personal GPS in a Mobile Phone*”. <http://www.i4u.com/modules.php?name=News&file=article&sid=263>. Last Accessed June 2004.

Jones, R.M. (1999) “A Time Line of Events Relevant to Computer and Video Games”. Colby College, <http://www.cs.colby.edu/~rjones/courses/cs398/history.html> Last Accessed June 2004.

Mahmoud, Q.H. (March 2004) “*J2ME and Location-Based Services*”. <http://developers.sun.com/techttopics/mobility/apis/articles/location/> Last accessed June 2004.

Manninen, T. (2001) “*Interaction Forms and Communicative Actions in Multiplayer Games*”. Gamestudies.org. <http://www.gamesstudies.org/0301/manninen> Last Accessed June 2004.

Monson, H. (December 1999) “*Bluetooth Technology and Implications*” <http://www.sysopt.com/articles/bluetooth/index2.html> Last Accessed June 2004.

Morrison, M. (1996) “Teach Yourself Internet Game Programming with Java in 21 Days”. Sams.net Publishing. <http://bookshelf.sleepnet.net/files/Internet%20Game%200Programming%20with%20Java/ch17.htm> Last Accessed June 2004.

Nvidia, (2004) “NVIDIA Selected by Mitsubishi for New i-mode™ Mobile Phone” http://www.nvidia.com/object/IO_10598.html Last Accessed August 2004.

Partanen, J.P. (June 2001) “*Mobile Gaming: A Framework for Evaluating the Industry 2000-2005*”, Gaptime Century Ltd., <http://www.gaptime.com/mobilegaming.pdf>, Last accessed June 2004.

Peabody. (1997). Washington State University, <http://www.vancouver.wsu.edu/fac/peabody/game-book/Chapter2.html>, Last Accessed June 2004.

Sim-tech “*Planetarion Manual*”. <http://jpaweb01.planetarion.com/manual.pl>. Last Accessed 06/03.

Slater, S. (2002). "Enhancing The Immersive Experience" GAME ON 2002 . 3rd International Conference on Intelligent Games and Simulation. University of Westminster

Sony Computer Entertainment. (2004) “*Sony Computer Entertainment Inc Announces Product Specifications of Handheld Video Game System, PlayStation Portable (PSP)*”. <http://www.us.playstations.com/pressreleases.asp?id=207>. Last Accessed 24/08/04.

Vogiazou, Y (April 2002) “*Presence Based Massively Multiplayer Games – Exploration of a new concept*”, Knowledge Media Institute / Open University. <http://kmi.open.ac.uk/publications/papers/kmi-tr-123.pdf> Last accessed June 2004.

INVESTIGATING TEAM SPEECH COMMUNICATION IN FPS VIDEO GAMES

Eleni Spyridou[†], Ian Palmer
School of Informatics
University of Bradford
Bradford BD7 1DP, United Kingdom
Email: e.spyridou@btopenworld.com
i.j.palmer@Bradford.ac.uk

Elric Williams
Faculty of Media
Trinity College
Leeds, United Kingdom
Email: elric.williams@btopenworld.com

[†]Eleni Spyridou is a scholar of the Public Benefit Foundation Alexander S. Onassis.

KEYWORDS

Speech communication, human-agent teams, multimodal interface, video games

ABSTRACT

Speech is natural. It evolved in response to human need for communication (Spyridou 2004). Participants tend to take turns in speaking, whilst giving constant non verbal feedback in the form of body language (Spyridou 2004; Shmandt 1994). A conversation involves at least two or more participants, who share knowledge in turns providing a mutual feedback. An advanced conversational system should not only speak and listen, but also understand, pose questions and take turns in a conversation. In this paper we investigate how users communicate with their teammates in a team multiplayer FPS video game. How they use speech to seek information, ask for help, describe situations, give explanations, instructions and commands. If the user is to feel comfortable playing alongside an agent, trust it and be persuaded by it, it is then vital to understand first how human users perform these functions and then provide the agent with similar behaviour. The experimental framework has features that have been emulated from the Unreal Tournament™ and Call of Duty™ video games.

INTRODUCTION

Speech is a challenging means of interaction and communication (Spyridou 2004). In order to employ the interaction and communication with the computer system effectively, human computer interaction techniques and an understanding of natural language, selected with immense care, are required to bridge the gap between human conversation and computer interfaces.

What makes a team in a team multiplayer FPS video game is the presence of a shared goal. Within a healthy team, group roles develop and change dynamically to meet new and

unanticipated challenges (Bruemmer, Marble & Dudenhoefter 2002). We are interested to investigate how teammates use speech to seek information, ask for help, describe situations, give explanations, instructions and commands when they play alongside other human players and how this natural communication can be applied to human-agent teams retaining ‘natural communication’.

In this paper we describe our experiments using natural language in a multimodal interface in a multi-user and single user – single agent environment and we then compare our findings. We conclude with some discussion points on the use of speech control in FPS video games.

EXPERIMENTS ON NATURAL LANGUAGE IN A MULTIMODAL INTERFACE

We have conducted experiments using team multiplayer FPS video games using Natural Speech (NS), to investigate how users communicate with each other when they are in a team sharing a common goal. The purpose of the experiments is to prepare the foundations for the construction of an embodied conversational computer agent with the ability to establish and maintain a social relationship with a user. If the user is to feel comfortable with the agents, trust them or be persuaded by them, then it is vital to understand how humans perform these functions and provide the agents with similar behaviour. Since the subjects used NS the commands issued were expected to be lengthy and of high complexity (Spyridou, Palmer & Williams 2003a; 2003b; Spyridou 2004). The experiments are divided into two categories: Multi-user environment, and single user – single agent environment.

Multi-User Environment

The multi-user environment is divided to: Public and Clan players. Public is the term used for a body of people who share some common interest; playing FPS video games, for example. Clan, however, is an alliance of network shooter players, who often fight united against other players or clans and compete in video game leagues (Spyridou 2004). We can liken public players to amateurs and clan players to professionals.

Public Players

The twenty-one participants (19 male, 2 female) were undergraduate and postgraduate student in the department of EIMC, of the School of Informatics at the University of Bradford. They were not paid to attend an hour session on successive days and play Unreal Tournament. All participants had experience with PC's and familiarisation with video games.



Figure 1: Public players in action

We used Intel Pentium III processors with 128 RAM, Microsoft Windows 2000 operational system, DirectX 8.1 compatible sound card and s 3D accelerator video card with 16MB VRAM. The multiplayer mode was supported with Internet (TCP/IP) and LAN (TCP/IP and IPX). For speech communication we used Microsoft's Game Voice SideWinder. SONY minidisk recorders captured players' speech commands while playing the video game.

Participants were given a booklet with instructions describing the mode 'Capture the Flag' and examples of the verbal commands they could use to communicate with their teammates. The users were allowed to change the control keys according to their preferences for a more natural game play. They then had to adapt those commands, create their own and speak to each other as naturally as possible. Each individual was in a separate room and given full instructions of the use of the game and hardware. The players were given 15 minutes unrecorded playing time to familiarise themselves with the control systems.

Results from Public Players

As it was expected in the multi-user environment, all the users issued commands using natural

language, with long and highly complex sentences. Through the use of dialogue, the players constructed plans for the annihilation of the enemy team and ultimately the victory of their own team.

From the plethora of the speech commands that were gathered from the experiments, a lexicon was compiled and broken down by different types of phrase issued by the subjects.

It was found that 80% of the issued instructions were statements, 10% were questions and another 10% was exclamations. Further analysis showed that 48% of comments initialised a vocal stream while 52% were replies.

Comparing the 52% replies with 10% questions it was concluded that:

Users reply to statements, which do not require a reply (informational statements). For example:

[Player 1]: *I'm heading for the shaft*

[Player 2]: *Right. I'm going for the lift then. There is one of them coming down this.*

Users answer exclamations and replies with vocal feedback. For example:

[Player 1]: *Do you have the flag?*

[Player 2]: *Yeah*

[Player 1]: *Ok*

Users require a stream of vocal contact to aid communication.

Any multimodal immersive environment, which utilises speech, must supply a large quantity of vocal feedback to the user. This feedback should be constant and be able to extend the vocal stream into a conversation.

Clan Players

The fifteen participants (all male) were all members of clan X¹. They were not paid to attend a ninety-minute session and play Call of Duty. All participants had experience with PC's and were experts on this particular FPS video game.

For this experiment a minimum specification of Intel Pentium III processors were used with memory of 128 MB of RAM. The operating system was Windows XP. The audio system was a DirectX 9.0b compatible 16bit sound card and latest drivers. The video system was a 3D Hardware accelerator card; DirectX 9.0b compatible 32MB Hardware accelerator with full

¹ The clan asked to remain anonymous. Since this work will be published, they did not want their tactics to be revealed to rival clans. The Author named them, the generic name clan X

T&L capabilities video card and latest drivers. The multiplayer mode was supported with Internet (TCP/IP) and LAN (TCP/IP). For speech communication they used TeamSpeak, which is designed to run in the background of online games. The team used the 'Start/Stop Recording' function to record the session in TeamSpeak. The recording is done in .wav format, PCM 22Khz mono in 16 bits.

The participants were not instructed to play the game in any particular way and were not consulted as to how they could communicate with each other. The observer was not present during the match; the recording was later made available to the Author. All clan members knew they were being recorded. The recordings are taken from a league match clan X fought against another clan in the U.K. The players were completely familiar with the control system of both the game and TeamSpeak.

Results from Clan Players

In clan X each player knew his place in the environment and when in need of help he would ask for back-up making known the relevant location. For example: *"Can you float Nick? So you can hear and come and help down by the ramp wall?"*

Each clan member had a task. Both during and after the completion of the task the players would give feedback. For example: *"The ramp is clear"* or *"Got it covered"*.

Although clan X said they had no leader in the team, the recording showed the clear presence of an Alpha male. This player would congratulate them on their efforts, reshape them, track time and indicate tactics and formations for attack and defence. This person made the majority of long and highly complex commands. For example: *"There is still plenty of time lads, so don't rush it"* or *"played lads"*.

The team had slang for some locations on the map in order to be shorter when spoken. For example: instead of *"police station"* they were saying *"playstation"*.

The commands used by the clan were highly precise at times, which was expected as generally clan members are better and more organised players than public players. Clan players know the maps intimately; they practice together so they have a recognised plan and generally are more experienced FPS players. For example: *"Right guys, let's do a measured approach though the T junction"* or *"Right guys, the assault squad will end rush. No nades. The three support guys will nade"*.

Locations were identified both relatively and absolutely, sometimes both in the same sentence. For example: *"One on the ramp to your right"*.

It can be surmised that relative commands show that clan members are immersed to some extent in the environment as if they were playing in a real-life terrain; Similar to a paintball game. For example: *"There is still one left, just to your left"*.

Comparison of Results in Public and Clan Players

Public players rarely stick to the plan, if there is one. They want a piece of individual adventure, to kill the enemy rather than waiting in a defensive position. Clan players call this 'hunting' and is frowned upon as it destroys team-play. Clan players stick to their plan. For example: *"3PPHSs to the garage, 2 with me, the rest run up the ramp"*.

Clan members knew that each member has a certain position on the map and a task to perform. Therefore, questions like "where are you?" were superfluous and were not used. Public players however, asked this question frequently; 33% of all questions. Possibly because they were lost in an environment that they did not know intimately, although this uncertainty is expected as they have no set plan.

Clan members' phrase patterns were very difficult to identify as none of their communication streams were similar. Public members' phrase patterns were similar, in that similar phrases were used repeatedly. This makes their communication more pattern-identifiable.

When clan players are in position they frequently gave location status updates without being asked. For example: *"I am at the playstation"*, *"I am downstairs office"*. Whereas public players only gave their location when asked.

Single User – Single Agent Environment

For the second category a new set of experiments were conducted. The aim was to find whether the complexity of the language used from the user in a single user – single agent environment is lower than in the natural communication previously recorded in the multi-user environment. For this experiment the Wizard of Oz paradigm was used.

The Wizard of Oz allowed the observation of a user operating an apparently fully functioning system, whose missing services were supplemented by a hidden wizard. The subjects were not aware of the presence of the wizard and have led them to believe that the computer system was fully operational. The wizard may observe the subject by any means such as a dedicated computer system connected to the observed system over a network. When the subject invoked a function that was not available in the observed system, the wizard simulated the effect of the function.

The twenty-one participants (19 male, 2 female) were undergraduate and postgraduate students in

the department of EIMC, of the School of Informatics at the University of Bradford. They were not paid to attend a half an hour session on successive days and play Unreal Tournament 2003. All participants had experience with PC's and familiarisation with video games. In this experiment the video game, system requirements, speech recognition, and method of collecting the data are exactly the same with the multi-user environment.



Figure 2: Subjects in WoZ

In this experiment a single user was told that his partner was a computer-controlled character, which was able to recognise vocal commands. The “wizard” controlled this character. The user was then asked to play a multiplayer Capture the Flag game.

If the user's commands were garbled, distorted or otherwise unintelligible the wizard would not perform any action. Otherwise the wizard acted precisely upon the commands. For example: “Go up the stairs and cover me”. The wizard will guide his avatar up the nearest stairs and attempt to protect the user. After the session, the subjects discussed their reactions with the Author.

Results from WoZ paradigm

Although the users were informed that the system would understand natural language, and therefore complex commands. 67% of the subjects issued simplistic commands similar to the examples given on the instruction sheet.

66% of the users, when they found an initial simplistic command that worked, would not deviate into a more complex string of instructions. For example, once the command “*go base*” was proven to work, the users would continue to use it rather than experimenting with more complex instructions; for example, “*go to our base and defend the flag*”.

24% of the subjects altered their natural speech habits in order to ‘help’ the computer understand their commands, i.e. some subjects would speak artificially loudly and slowly.

57% of the users garbled and issued unclear and unfinished commands.

Even though it was stated in the instructions that the purpose of the experiment was to communicate naturally with the A.I., 24% of the users played the game to win, ordering the agent to defend the base whilst they would attack and collect the flag.

The users were told to use location names for the map supplied, should they need covering fire from the agent so that it would understand where they were and be able to comply. However, only 24% did as instructed. The rest would ask for cover fire without specifying their location.

While the users were told that the agent would not give any verbal or text feedback, 76% of the users would still ask questions to the agent requesting such things as the agent's location and progress, and 57% would compliment and praise the agent's actions. The most frequent question was “where are you?”.

Some 24% of the users tended to assume that the system was more simplistic than they had been told and would commonly miss out prepositions and definite articles. For example, “*go base*” instead of “*go [to] [the] base*”.

90% of the users found communicating with their teammate via speech precise, impressive, and an interesting experience. They commented that they found speech made playing this video game genre with agents as teammates much more enjoyable and fun.

They also found that speech enhanced the game play a lot since it made a big difference talking to their teammate rather than having to type.

A predictable comment was the great need for feedback; both negative and positive evidence (Brennan & Hulteen, 1995) of commands and communication that has been understood as well as confirmation of actions taken.

There was an understandable disbelief of some users that speech recognition software could work that well. When informed that in the ultimate human-agent team game, the agent would be able to give to the human orders, users replied that they would not trust the agent's commands.

It was concluded that in FPS video games, speech should be the only modality to communication, giving instructions and queries to the teammates in the form of dialogues with appropriate feedback.

DISCUSSION AND FURTHER WORK

Brennan and Hulteen (1995) found that people have many other demands on them while they are speaking; such as performing tasks, planning what to say next, and they frequently do not produce the kind of fluent speech a speech recogniser has been trained to process. However, fluent speech can be found in formal written languages and therefore may apply to some speech recognition applications such as Speech-to-Text software. Our experiments showed that users do not speak fluently on syntactic, semantic, and pragmatic levels in their communication when playing a team video game.

Results also showed that clan players do not ask “*where are you?*” or “*how are you?*” questions, similarly present FPS games that utilise speech communication do not allow for users to ask these questions. This makes us wonder whether game developers design FPS video games utilising speech around clan gameplay. Although this is not documented, it would be unfair to offer a clan-oriented product to the public market when the latter type of users ask these questions frequently.

Furthermore, it seems that game developers do not design games that utilise speech as a fully functioning part of the gameplay interface, but rather as an optional extra. The speech commands in these games can also be triggered through use of the normal interface device (gamepad, keyboard, etc.). It is the authors’ opinion that speech recognition in these games is being used solely to make the game stand out from the others and sell more copies. That is why the FPS video games that utilise speech in the market today (SOCOM: U.S. Navy Seals 2002; SOCOM II: U.S. Navy Seals 2003; Rainbow Six 3 2003; S.W.A.T.: Global Strike Team 2003) specify the state-of-the-art speech recognition along with better graphics and game levels. The common denominator is that the human player is always put in the Alpha male position; the leader who will command the team and plan strategies.

It is not easy to compare the lexicon of the aforementioned FPS video games with the lexicon acquired in our experiments. The difference is that these FPS games are all based on military simulations. Therefore, the commands available are of a certain style. A military command should be brief, concise, and to the point (Spyridou 2004). The less the commander says the less chance for any misunderstanding. In Unreal Tournament™’s Capture the Flag (CTF) game, there are no strict military commands. The players may have a base and flag to defend but there are no restrictions in their vocal communication. There is more freedom to the users’ speech when playing for attack and defence. Nevertheless, when the users in the Wizard of Oz experiment were told that they could speak as much as they wanted to the agent, give it various commands and test how accurately or not it could understand the human voice, none did so. Their commands were similar to the military ones; short, concise, and straight to the point. This can possibly be attributed to the subjects being ill at ease with the system and experimental environment, their inexperience with speech recognition, and perhaps they did not realise the full potential of what they could say.

Throughout the experiments the most important finding was the importance of feedback. Feedback provides an essential role in video game interaction. It gives the player evidence of

closure, thus satisfying the communication expectations users have when engaging in a dialogue. The more difficult the communication is the more important the partner feedback is.

A team is about communication (Spyridou 2004). Speech is used to seek information, ask for help, express feelings, and describe situations, giving explanations, instructions, and commands. The context and function of speech will vary enormously and will have significant effect on all aspects of the dialogue, from lexical choice to length of utterance (Cockcroft 2000). People constantly shift or accommodate their language choices, including accent in conversation to fit the context and purpose, often without realising it.

For future work, we are proposing a prototype of an embodied conversational computer agent, which will be constructed with the ability to establish and maintain a social relationship with the user in an FPS video game environment. In this type of video game, the central principle is not only the ability of the human to understand the agent’s performance, but also the agent’s ability to identify human needs and interfere if needed. The objective is to create human-agent teams, where each member whether human or agent, will take initiatives and through communication and team effort, achieve and complete a common goal.

Technologists developing speech-oriented systems have a tendency to think that users can adjust their speech to whatever they manufacture. In general, they have relied on instruction, training, and practice with the system challenging users to speak in a style that matches the system’s processing capabilities (Oviatt and Van Gent 1996).

According to Rouse (2001) one of the main reasons players play video games is to be immersed (Newman 2004).

Virtual Reality will become the ultimate gaming platform. Users will be immersed in a fully 3D environment, where they will speak and interact with other people, agents, and objects setting off realistic games. As VR research will provide the technology and implementation to facilitate games becoming more immersive and real, Natural Speech research should focus on issues of dialogue and feedback, rather than the accuracy of the underlying speech recognition technology. Ultimately, the most successful speech interface in a video game should not be the one which accurately follows a long list of verbal commands, but one that fits gracefully into the situation and makes the user feel socially engaged and comforted.

REFERENCES

Brennan, S. E. & Hutleen, E. A. 1995, 'Interaction and Feedback in a Spoken Language System: A theoretical framework', *Proceedings of Knowledge-Based Research*, Stockholm.

Bruemmer, D. J., Marble, J. L., & Dudenhoefter, D. D. 2002, *IEEE Conference on Human Factors and Power Plants*, Scottsdale, AZ.

Cockcroft, S. 2000, *Investigating Talk*, Hodder & Stoughton, London.

Newman, J. 2004, *Videogames*, Routledge, London.

Oviatt, S. L. & Van Gent, R. 1996, 'Error resolution during multimodal human-computer interaction', *Proceedings of the International Conference on Spoken Language Processing*, vol. 2, pp. 204-207, University of Delaware Press.

Rouse, R. 2001, *Game Design Theory and Practice*, Plano, TX: Wordware Publishing.

Schmandt, C. 1994, *Voice Communication with Computers*, Van Nostrand Reinhold, New York.

SOCOM: U.S. Navy Seals, [Online], Available: <http://www.us.playstation.com>

SOCOM II: U.S. Navy Seals, [Online], Available: <http://www.us.playstation.com>

Spyridou, E., Palmer, I. J. & Williams, E. J. 2003a, 'The use of Speech in Multimodal Interfaces in Computer Games', *Proceedings of the Eurographics Ireland Workshop Series*, Vol. 2, pp. 17-22.

Spyridou, E., Palmer, I. J. & Williams, E. J. 2003b, 'Speech Interaction for Networked Video Games', *Proceedings of the HCI International Conference*, Vol. 1, PP. 1046-1050.

SWAT: Global Strike Team, [Online], Available: <http://swatgst.vu-games.com/uk/>

Tom Clancy's Rainbow Six 3, [Online], Available: <http://www.rainbowsix3.com/us>



Eleni Spyridou received her PhD and BSc (Hons) in 2004 and 2000 respectively at the University of Bradford, and has a degree in DeskTop Publishing from the Vocational Training Institute "DOMH", in Athens, Greece. Her current research focuses on conversational human-agent team video games using natural language, statistical Natural Language Processing and multimodal interfaces. She is a member of IEEE and an Expert Evaluator/Reviewer for the European Commission. She enjoys playing FPS and online fantasy RPG video games, her favourites are UT2003, Team Fortress and King of Chaos. In her leisure time she is interested in calligraphy and collects stationary. Contact her at: e.spyridou@bopenworld.com

Neural Networks in Games

| | |
|--|------------|
| Charles, D. and McGlinchey, S. The past, present and future of artificial neural networks in digital games | 163 |
| Bateman, R. M. and Doholi, S. Reinforcement learning algorithm using neural networks for playing Othello | 170 |
| Liang , Q., Ehlert, P. and Rothkrantz, L. Towards a neural control artificial pilot | 175 |
| Mańdziuk, J. and Mossakowski, K. Looking inside neural networks trained to solve double dummy bridge problems | 182 |
| Churchill, J., Cant, R. and Al-Dabass, D. Using neural network techniques within a Go playing program | 187 |
| Graepel, T., Herbrich, R. and Gold, J. Learning to fight | 193 |

THE PAST, PRESENT AND FUTURE OF ARTIFICIAL NEURAL NETWORKS IN DIGITAL GAMES

Darryl Charles¹ and Stephen McGlinchey²

¹School of Computing & Information Engineering, University of Ulster, N. Ireland.
dk.charles@ulster.ac.uk

²School of Computing, University of Paisley, High Street, Paisley, Scotland.
stephen.mcglinchey@paisley.ac.uk

KEYWORDS

Neural Networks, learning, adaptive.

ABSTRACT

Over the past 20 years, the topic of artificial neural networks has been a vibrant area of AI research, leading to new algorithms that have been used in a variety of disciplines including engineering, finance, artificial perception and control & simulation. Despite this, there has been a limited impact on the commercial games industry. This paper reviews some of the successful uses of neural networks in games and identifies the positive elements of their use, and discusses some of the factors that have deterred their use amongst game developers. Addressing these weaknesses, we outline ideas for future research that may aid game developers in producing more convincing AI, and may supplement or replace more traditional techniques.

INTRODUCTION

There have been attempts to use artificial neural networks in digital games for quite a number of years now and the reason for this is quite straightforward; artificial neural networks are about learning, and the effective use of learning technology in games has been something that many in the game design development industry have desired for a number of years now. It also helps that neural networks are relatively well known and understood – particularly by computer science graduates – and their use is also popular because they (loosely) model biological neural networks such as those in our own brains, and so the link to learning and human-level intelligence is therefore very tangible.

Learning mechanisms in digital games may be offline or online. With offline learning we train the AI during the development process only. Once the product is released, the AI is unable to continue learning as a game is played. For example, the AI could observe and model player behaviour using learning algorithms such as artificial

neural networks (McGlinchey, 2003). This may be used to create believable characters by imitation of a typical (or perhaps expert) player or a combination of features from a variety of players, or perhaps to model players or groups of players in order to respond appropriately to a player in-game. Online learning means that the AI learns (or continues to learn) whilst the end product is being used, and the AI in games is able to adapt to the style of play of the user. Online learning is a much more difficult prospect because it is a real-time process and many of the commonly used algorithms for learning are therefore not suitable. Instead these algorithms must be adapted for real-time dynamic processes. Real-time strategy (RTS) games are a particular candidate for online learning algorithms and some interesting approaches are being developed (Fyfe, 2004). In some situations a combination of both offline learning and online adaptation is the most appropriate approach (Livingstone & McDowell, 2003). These aspects of the implementation of learning technologies into games are inherent to the use of neural networks in games and we will revisit them often throughout the paper.

NEURAL NETWORK LEARNING

Most attendees of this conference will be familiar with the different categories of learning for neural networks: supervised, unsupervised and reinforcement learning. So we only provide a brief overview. With supervised learning, we provide the network with input data and the correct answer i.e. what output we wish to receive given that input data. The input data is typically propagated forward through the network until activation reaches the output neurons. We can then compare the answer, which the network has calculated with that which we wished to get. If the answers agree, we need make no change to the network; if, however, the answer which the network is giving is different from that which we wished then we adjust the weights to ensure that the network is more likely to give the correct answer in future if it is again presented with the same (or similar) input data. This weight adjustment scheme is known as supervised learning or learning with a teacher.

With unsupervised learning there is no external teacher and learning is generally based only on information that is local to each neuron. This is also often referred to as self-organisation, in the sense that the network self-organises in response to data presented to the network and detects the emergent collective properties within the data. Unsupervised neural methods are often used in an exploratory manner; we use statistical relationships between data variables in order to establish an understanding of the nature of the data. Unlike supervised learning, we do not know the answers before we begin training.

A third less commonly used form of neural learning is reinforcement learning. This learning relates to maximizing a numerical reward signal through a sort of trial-and-error search. In order to learn the network is not told which actions to take but instead must discover which actions yield the most reward by trying them – if an action has been successful then the weights are altered to reinforce that behaviour otherwise that action is discouraged in the modification of the weights. Reinforcement learning is different from supervised learning in that with supervised methods, learning is from examples provided by some knowledgeable external supervisor. With interactive sorts of problems it is quite often unrealistic to expect to be able to provide examples of desired behaviour that are both correct and representative for all scenarios which an agent may encounter. Yet this is perhaps where we would expect learning to be most beneficial, particularly with agent technology where an agent can learn from experience.

For those just starting to work with neural networks in digital games there are now many introductions to the topic available (for example (Champanand, 2002, Sweetser, 2004) and gameai.com is also well worth a visit. To delve more deeply you may wish to refer to one of the books in the area such as the introductory text by Gurney (Gurney, 1996) or the more advanced book by Haykin (Haykin, 1998).

CURRENT APPROACHES

It may be observed even from a brief literature review that the use of neural networks is still quite rare in mainstream commercial games and that the range of neural networks used is very limited – the error back-propagation algorithm is the most widely used neural network because it is the most well known. However, the use of neural networks in digital versions of classic games such as Mastermind, Othello, Checkers (Draughts), and Backgammon is not unusual and has been successful in many situations as with Big Blue (see gameai.com). However, the use of neural networks in this type of game mostly focuses on strategy and the games are often more slowly paced. Modern digital games generally have more dynamic environments and the CPU

has to deal with much more than just the AI. Current commercial digital games are varied and strategy is only one aspect of these games that we may apply neural networks to. Having said that, there are surprisingly few examples of the use of neural networks in commercial games, a couple of the best examples including “Colin McRae Rally 2” which uses neural networks to train the non-player vehicles to drive realistically on the track, and “Creatures” which uses neural networks along with evolutionary algorithms to dynamically evolve unique behaviours for game creatures. Black & White is the most high profile example of a recent game that utilises in-game learning – neurons are incorporated into an AI module for the game avatar, and these neurons are iteratively re-trained based on game feedback. The game uses a form of Perceptron learning within modules, for example, to model an avatar’s desire (Evans, 2002). The output of the neuron providing a measure of desire based on inputs which represent levels of “desire sources” for avatar attributes, such as: hunger, tastiness (of food), and unhappiness. The agent architecture is loosely modelled in the first place from psychological/philosophical ideas.

Social simulation games such as The Sims (Electronic Arts, 2001) naturally lend themselves to dynamic learning; these games are based on interaction between characters and objects due to environmental and social input. A character makes decisions within the game based on their current state and the state of the environment, for example if a character is hungry and they are close to a fridge containing food then they will prepare some food and eat it. A character may change their preferences or reactions over the period of the game based on “experience”. Recent academic research has demonstrated the use of neural networks (MacNamee & Cunningham, 2003) to create intelligent social controllers for agents that represent non-player characters. Other interesting recent examples of the use of neural networks within games include an approach for strategic decision making (Sweetser, 2004a), use of a self-organising map for modelling player behaviour (McGlinchey, 2003), and modelling player behaviour in first person shooters (FPS) using a method involving a multi-layer perceptron network (Geisler, 2004).

POTENTIAL FUTURE APPLICATIONS FOR NEURAL NETWORKS IN DIGITAL GAMES

There are a wide range of neural networks that have not even been attempted to be used in games applications, particularly unsupervised and reinforcement learning methods. Here we discuss a few potential techniques within the context of a number of key application areas for neural networks in games that have either not been addressed yet or have only been tackled recently.

Online Learning

Learning technologies for digital games have become increasingly important (Rabin, 2002). Yet, while there are a number of examples of games that use “off-line” learning – for example, Quake III Bots may be trained using artificial neural networks or genetic algorithms – there are only a few examples of games that explicitly use “on-line” dynamic learning within a game, e.g. Black & White as discussed earlier.

The most significant issue with on-line learning is that it may produce unpredictable results; sometimes these effects serve to enhance but more often it leads to erratic game behaviour that reduces the quality of gameplay, and in worse scenarios will introduce dynamic game bugs. Testing, debugging and balancing games that incorporate learning is a challenging task (Barnes Hutchens, 2002). With the use of neural networks we have the added problem that, although they are very good for learning purposes, most neural algorithms can not easily be adapted incrementally but would generally require complete retraining online. Retraining is often very slow, and in many cases a small quantity of new data examples will not be enough to significantly impact the training of the algorithm, and of course, retraining the network completely online may lead to unsatisfactory results. An element of control is lost because tuning of the neural network by the developer will not be possible, as it is with the offline training of the network. These factors are presented not to discourage the reader from using neural networks for online learning but to encourage the development of new techniques and the use of suitable existing methods to approach this problem. For example, it is likely that dynamic online neural networks may need to be constrained to operate within predefined boundaries – i.e. the outputs of the networks are restricted to pre-tested values.

There are significant obstacles in the way of developing generic, robust and effective dynamic learning algorithms and architectures for digital games but the potential rewards are great (Charles, 2003). Perhaps the greatest potential gain with on-line learning is with the dynamic adaptation to player behaviour, play patterns and skill levels. In particular, a worthy pursuit is to develop technologies that may learn where a human player is being challenged too much or too little and modify the player’s character attributes, AI opponent behaviour or game environment accordingly. These alterations may be temporary, just to finish a particularly challenging section or the changes may be implemented for a longer time and player’s progress monitored. The flexibility afforded by dynamic learning mechanisms may also be used to counter a player benefiting unduly from – or being hindered by – unforeseen player behaviour or minor bugs in the game design. The capability of a game to self-adapt in these situations to prevent a significant deterioration in

gameplay due to minor design oversights and player behaviour is certainly a laudable goal.

Player Centred Approaches: Player Modelling and Learning about the Player

It is perhaps not an obvious or much discussed issue relating to digital game AI but an important one nonetheless – that of attaining a more wide-spread appeal to entertainment of playing digital games. We need to keep the state of the games industry in perspective, the games industry continues to grow rapidly but it still represents only a small proportion of the entire entertainment and media industry. Even though there are a wide range of age groups playing games now, thanks in part to the release and marketing of the PlayStation and the more mature content of PC games, there is still a wide range of people who never even try to play a game, or simply give up after a short attempt.

All game players are different; each has a different preference for the pace and style of gameplay within a game, and the range of game playing capabilities between players can vary widely. Even players with a similar level of game playing ability will often find separate aspects of a game to be more difficult to them individually and the techniques that each player focuses on to complete separate challenges can also be very different. For these reasons and others it can be very difficult to design a game that caters for a wide range of player capability and preference. Game developers have traditionally dealt with the range of player abilities in a very straightforward manner, for example, by allowing the player to select a difficulty level at the beginning of the game, as with the classic first person shooter “Doom”. Once a player selects their level of difficulty for a game designed in this way, then there is usually no attempt within the game to monitor how a player is performing in order to adjust the level of challenge or gameplay experience. While the concept of an adaptive game is a controversial topic among some gamers and developers, there are clear benefits to tailoring the game experience to particular player types – especially for educational games (Beal et al, 2002). Catering for the individual more effectively could help attract a wider participation, if for no other reason than making it easier for players to get started, progress and complete a game, and therefore widening the accessibility of games.

The use of neural networks for the player modelling process is quite an obvious approach but the authors are not aware of them having been used for this purpose in games yet and so we provide an overview to a few possible supervised and unsupervised approaches below. Neural networks are good at detecting patterns and clustering data (depending on the method) and so we can use a variety of neural network techniques in different ways to identify or understand different players. For example, we can use player reaction times, choices made, styles of play, accuracy of shots/hits, how often a stage needs to be repeated before completing, average health, number of deaths per level, kills per level per

possible kills to build models of player's offline through the training of a neural network such as the multilayer perceptron network trained with the error back-propagation algorithm. These variables may be observed online used directly to decide how to change the parameters of the game environment, attributes of the player character or non-player character behaviour dynamically.

Another neural network approach to player modelling is to use a clustering algorithm. In this way we use the neural networks to cluster player types according to out-of-game and in-game data, grouping player with a similar profile into the same group type. There is a wide range of ways in which this may be done, for example we could use a radial basis network with fixed cluster centres to classify the players, with the centres fixed on different areas of the data space that we believe to provide a good "centre" for our player classification. By monitoring and adapting the player profile throughout the game then the player may achieve a new classification, and thus the game would respond differently. Radial basis networks may also have moving cluster centres and so the centres can be moved automatically during training to fit the data more appropriately.

We may use unsupervised neural networks in particular so as to form a statistical understanding of player data, to explore or investigate structure or patterns in data on the basis of statistics or information theory (or similar). Using projection methods such as Principal Components Analysis and Factor Analysis we typically want to explore the relationship between the input variables which is a different approach to clustering methods. Factor Analysis can identify relationships between sub-sets of the data variables that may be used to identify more refined aspects of player behaviour, e.g. output one could identify the overall capability of the player and output two may identify whether the player is cautious or just dashes in etc. Being able to identify more subtle or complex aspects of player behaviour could be very valuable in tailoring the game experience to the player, and it also potentially opens up new possibilities for dynamic gameplay. For example, if we are able to discover patterns that relate more to player emotion or motivation then this may be used with other sensory devices to discern the needs or desires of the player and the game can be adapted to account for this.

Intelligent Character Animation

The Artificial Intelligence in a game is perhaps one of the most influential ingredients for enabling a game player to suspend disbelief long enough to become properly immersed into the gameplay. If characters or objects behave in an obviously unexpected – or unintelligent – way, then the game experience is very much diminished. The quality of graphics in digital games has reached an incredible degree of realism, as witnessed by games like

Doom III (ID Software, 2004), and realism of visuals is important, of course, because many of us enjoy the "wow" factor afforded by the visual impact of the newest and most graphically advanced game – this facet clearly sells games. Visual realism is only a part of what makes a game world and the characters in it believable, if any aspect of the game shatters our immersive gameplay experience and we are less able to suspend disbelief within the game world. In other words we may have a beautifully created wall using the latest vertex and pixel shader programs to enhance the illusion of the game world existence, but the illusion is shattered when our supposedly intelligent character continually bangs his head off the wall in an attempt to get round it or walks in mid-air!

Intelligent character animation is one approach to improve this aspect of player immersion. For example, a Neural Network may be used as the "decision maker" for an animating character and when paired to a fuzzy controller system this particular agent architecture can be useful (Wen et al, 2002). Neural networks may also have broader uses in character animation; for example, it should be possible to train a neural network to act as a transformation matrix in order to interpolate in the mesh blending technique described above. This will provide benefits during game development, but also opens new possibilities for run-time generation of animation data, allowing game characters to be truly responsive to game events and user interactions.

Let us first consider savings that can be made during development. Motion captured data can be time consuming and expensive to post-process due to several factors. Firstly, data from optical motion capture systems is normally incomplete due to optical occlusions, and this data must be completed by artists. Magnetic motion capture systems also have problems since the sensors produce noisy data. Mechanical systems have neither of these problems, however, actors and artists tend to dislike using these systems since they constrain the actor, and they can only be applied to simple bone structures. Neural networks have been used for dealing with noisy and incomplete data in other disciplines, and if new research in intelligent character animation can tackle these problems, the cost of using motion capture systems would be significantly reduced. Neural networks have also been trained on motion data and later used to synthesise key-framed motion data and this is the basis of at least one commercial tool for automatic generation of animation.

Current methods of animating characters can produce very impressive results; however, this comes at a significant cost, requiring skilled animators to work at a low level, specifying limb and joint positions and orientations, and restricting games to replaying fixed animation sequences. The idea of a "virtual actor" is to allow a director (or game developer) use a high-level set of instructions (e.g. creep, walk, run, read, say etc.) to direct the actions of avatars, and this may include adverbs describing style and emotional state (e.g., fearfully, excitedly). Convincing virtual actors will allow game

developers to be less concerned with low-level details, and focus their efforts on drama and emotion, which can add significantly to the immersive qualities of games. Real-time generation of animation for virtual actors is an area where neural networks may be useful.

Prediction

Neural networks have been successfully used for prediction in several application areas including finance, weather forecasting, power consumption, sales forecasting etc. It is not uncommon for a variety of different types of neural networks to be used for prediction generally in the computing and engineering world but they have not been used much (or at all) for this purpose in digital games. Prediction can be useful in quite a few ways in games, especially in strategic aspects of the games. For example in a real-time strategy game it is interesting to explore predictive approaches for a computer opponent in building its strategy, and in a 1st person action/adventure game non-player character prediction of player movement or strategy would be interesting in countering their movement. An added bonus of the use of a predictive approach is that behaviour can be non-deterministic and thus potentially more believable and can provide a more varied and interesting computer opponent.

Human-level Intelligence Studies

In one of the well known early papers (Laird, 2001) of the recent surge of interest into digital games research it was suggested that digital games provide an excellent platform to explore human level intelligence – which is after all one of the key original reasons that researchers began to work on Artificial Intelligence. Part of the reasoning behind this argument being that the virtual worlds and characters of commercial games are so rich in detail and that they provide an opportunity for a player to become immersed in realistic environments and interact with believable characters. An individual computer game or videogame may be played by millions of people and so this offers significant opportunity for study, and many games also offer high quality and easy to use game content development kits, e.g. “Neverwinter Nights”, that provide an opportunity for the creation of suitable tailored experimental test-beds.

However, we must be careful in acknowledging the difference between human level intelligence imitation and the form of AI common in games which has more of a relationship to the Turing Test (Turing, 1950) and creating believable behaviour (or fooling the player). The inspiration for developing AI opponents for games may be traced back to the Turing Test, since the original Turing Test may be thought of as a kind of game in which a computer must be programmed to fool an interrogator into believing that it is real woman as often as a man can

fool the interrogator that he is a woman. The original question posed by Turing Test has evolved over the years to, “can a machine play a game of skill as well as a human being?” or “can the program compete with people?” (Fogel, 2002). In a way this is a distortion of the original goal of early AI research in that a central objective has been to understand human-level intelligence and replicate this functionality holistically. Writing a program that competes with a human opponent in a computer game is often as much to do with having enough raw computing power to process a large set of rules and creating an illusion of intelligence than it is about developing convincing human-level intelligence models.

Neural networks and models of the brain that include neural networks can prove very useful when exploring more human aspects of AI in games – because of their learning capabilities and resemblance of brain function. An interesting potential future technology related to human-level intelligence research involves the development of character AI architectures which allow us to “grow” or evolve game characters off-line – outside the game – and then insert this character into the game so that it will continue to learn. Could such a character be retrained and used in future games – a bit like a game actor? Would a player be able to extract an intelligent character from one game for use, with retraining, in a future game release? – sort of like an extended, intelligent, version of the character game save. Some work has been performed in this area already, where one well-known AI researcher believes that he can grow a conscious character on his computer (Cohen, 2002) and that characters such as these may be sold to game development companies.

With more lifelike characters and realistic emotional representation in our games we may have to consider the moral and ethical implications of decisions made by gamers even more than we do now and deliberately design-in effective consequences for actions. These issues become more significant as game characters approach some form of realistic consciousness, however, utilising AI to construct well-designed moral dilemmas and emotionally effective set pieces with games opens a range of new and interesting gameplay scenarios.

DISCUSSION

Most games allow only limited processing resources for AI, and this can often prohibit many advanced AI techniques. It is reported that 50% of the processing resources were allocated to AI in the game “Creatures” (see <http://www.gameai.com/cgdc97notes.html>). However, few commercial games have this rich allocation of processing resources to AI; 1% - 5% is a more typical allocation. With such limited resources available, it is often perceived that neural networks are too computationally expensive to be used in the majority of commercial games – particularly when the AI is to be trained online. This is a fair criticism of some of

the well-known neural network methods such as the error back-propagation algorithm. However, there are many other neural algorithms that have a comparatively low computational cost, such as some Hebbian learning methods, topology-preserving maps, radial basis networks and Learning Vector Quantisation (LVQ). Moreover, some older neural network training methods can be implemented using optimisations that vastly reduce their computational cost. (Kohonen, 1996) For offline applications, the argument of prohibitive computational cost is rarely valid, since it is the weight update (training) procedures that normally require the majority of the computation – not the feed-forward, or sensory phase, which would be used at runtime.

Inexperienced users of neural networks often encounter problems with parameter selection. Let us consider the radial basis function network as a typical example. For this network, the user must choose a suitable number of centres, and each of these must be initialised to a point in the data space. There are several other parameters that need to be “tweaked” to ensure that the network converges to a stable state without over-fitting the data, including the centre variances, the initial weight values, the learning rate, and any learning rate annealing strategy. The set of parameters that works best will vary between different data sets and applications, and it can be time-consuming for a developer to find an acceptable set. The problem is further exemplified in online training, where there is no expert to hand-pick parameters. The problem of parameter selection has been tackled in recent years by several probabilistic methods, which have created a great deal of interest amongst the computational intelligence community. There are now many probabilistic neural algorithms (e.g. (Bishop, 1998, Hinton et al, 1995, and Yin & Allinson, 2002) that work using objective functions to train the network. These methods tend to have fewer user-selected parameters, and where parameters must be chosen they tend to be less sensitive to picking critical values. To our knowledge, this exciting area of neural network research has yet to be applied to games-specific applications, and this promises to be a worthwhile area for future research.

As implied in this discussion section, progress has been limited in the use of neural networks within digital games largely due to a lack of knowledge or understanding among researchers and game developers of the wide range of methods that may be applied to game AI. This situation can be improved by those of us with a wider and more detailed knowledge of neural methods providing a range of successful, persuasive and meaningful neural network enhanced game AI examples.

REFERENCES

- Barnes J and Hutchens J. 2002. “Testing Undefined Behaviour as a Result of Learning”, *AI Game Programming Wisdom*, pp 615-623, Charles River Media.
- Beal C, Beck J, Westbrook D, Atkin M & Cohen P, “Intelligent Modeling of the User in Interactive Entertainment”, pp 8 – 12, *AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Stanford, 2002.
- Bishop, C.K., Svensen, M., Williams, C.K.I., “GTM: The Generative Topographic Mapping” *neural Computation* 10(1) pp215-234, 1998.
- Buckland M, “Advanced AI Techniques for Game Programming”, Chapter 10, *Real Time Evolution*, Premier Press, 2002.
- Cander, L. R., “Ionospheric forecasting technique by artificial neural networks” *Electronic Letters* 34(16) pp1573-1574, 1998.
- Chamandard A, “The Dark Art of Neural Networks”, *AI Game Programming Wisdom*, pp 640-651, Charles River Media, 2002.
- Charles D, "Enhancing Gameplay: Challenges for Artificial Intelligence in Digital Games", *Proceedings of the 1st World Conference on Digital Games 2003*, University of Utrecht, The Netherlands, 4-6th November 2003.
- Cohen D, “Game Over”, *New Scientist*, pp 47-49, 29th June 2002.
- Evans R, “Varieties of Learning”, pp 567-578, *AI Game Programming Wisdom*, Charles River Media, 2002.
- Fogel, D, "BLONDIE24: Playing at the edge of AI", Chapter 1, *Morgan Kaufman*, 2002.
- Fyfe C, “Dynamic Strategy Creation and Selection Using Artificial Immune Systems”, *International Journal of Intelligent Games & Simulation*, Vol. 3 No. 1, pp 1 – 6, 2004.
- Geisler B, “Integrated Machine Learning for Behavior Modeling in Video Games”, *Challenges in Game Artificial Intelligence*, AIII Press, 2004
- Gurney K, “Introduction to Neural Networks” *Routledge*, 1997.
- Haykin S, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1998.
- Hinton, G.E., Dayan, P., Frey, B.J., Neal, R.M. “The "wake-sleep" algorithm for unsupervised neural networks” *Science* 268(5214) pp 1158-61, May 1995.

Kohonen, "T. The speedy SOM." Technical Report A33, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland, 1996

Laird J.E. & Van Lent, M. Interactive Computer Games: Human-Level AI's Killer Application, *AI Magazine*, 22 (2), 15-25, 2001.

Livingstone D. & McDowell R, "Fast-Marching and Fast Driving: Combining Off-Line Search and Reactive A.I.", International Conference on Intelligent Games and Simulation (Game-On), London, 19th-21st November, 2003.

MacNamee B and Cunningham P, "Creating Socially Interactive Non Player Characters: The μ -SIC System", International Journal of Intelligent Games and Simulation, Vol. 2 No. 1 Feb. 2003.

McGlinchey S, "Learning of AI Players from Game Observation Data", GAME-ON 2003, 4th International Conference on Intelligent Games and Simulation, pp. 106-110, Nov. 2003.

Rabin S (Ed.), *AI Game Programming Wisdom*, preface, Charles River Media, 2002.

Sweetser P, "How to Build Neural Networks for Games", *AI Game Programming Wisdom II*, pp 615-625, Charles River Media, 2004.

Sweetser P, "Strategic Decision-Making with Neural Networks and Influence Maps", pp 439-446, *AI Game Programming Wisdom II*, Charles River Media, 2004.

Turing A M, "Computing Machinery and Intelligence", *Mind* 59 (1950): 433-60.

Wen Z, Mehdi Q & Gough N, "Neural Networks for Animating Variations in Character Behaviours" "GAME-ON 2002, 3rd International Conference on Intelligent Games and Simulation, pp. 189-196.

Yin, H. and Allinson, N.M., "Bayesian learning for self-organising maps," *Electronics Letters*, Vol. 33, No. 4, pp. 304-305, 1997.

REINFORCEMENT LEARNING ALGORITHM USING NEURAL NETWORKS FOR PLAYING OTHELLO

Richard M. Bateman and Simona Doboli

Department of Computer Science

Hofstra University

Hempstead, NY, USA 11549-1000

E-mails: RichardMBateman@aol.com, Simona.Doboli@hofstra.edu

KEYWORDS

Reinforcement Learning, TD learning, Neural Networks, Othello.

ABSTRACT

The paper presents a reinforcement learning algorithm using neural networks for learning an agent to play the Othello game. The system has no initial knowledge of the game, except of its rules. Reinforcement learning techniques allow an agent to learn successful game strategies by repeatedly playing the game. The only information received by the agent during learning is a reinforcement signal at the end of each game. Neural networks are known for their good generalization for untrained inputs. Since the state space for a game like Othello is very large, we use neural networks to represent the value function. The approach is similar to that of Tesauro for the game of Backgammon. The learned network can play Othello at a level close to that of expert programs as shown by its performance in world-championship games.

INTRODUCTION

Reinforcement learning is an *on-line* method of learning from experience (Sutton and Barto, 1998). The aim is to learn an optimal value function that assigns to each state of the environment a value representing the estimated reward from that state until the end of an episode. All learning is done while the agent interacts with the environment: the agent repeatedly observes states, takes actions, receives rewards and updates its value function. The only information the agent receives about its actions is a reward signal, which is known only at the end of the game. Reinforcement learning has been used successfully in spatial navigation problems (Arleo and Gerstner, 2000), game playing (Backgammon) (Tesauro, 1995), trading agents and markets (Moody and Saffell, 2001; Tesauro and Kephart, 1999) and job-shop scheduling (Zhang and Dietterich, 1995).

Reinforcement learning works well for problems with a small number of states, where each state can be visited a large number of times - a requirement for convergence to the optimal value function (Sutton and Barto, 1998). Situations with a large number of states require a different approach, otherwise an unfeasibly long training time is needed. The value function for the visited states is updated during learning, while a function approximation method is used to evaluate the value function of the rest of the states (Sutton and Barto, 1998). We chose neural networks to represent the value function (Sutton and Barto, 1998; Arleo and Gerstner,

2000; Foster et al. 2000). Neural networks can learn any nonlinear mapping based on their well known property of universal approximators. Moreover, neural networks have very good generalization capabilities for inputs unseen during learning.

In this paper, we use a reinforcement learning algorithm implemented with neural networks for training an agent to play the Othello game. The value function is represented by a multi-layer feed-forward neural network trained with the back-propagation algorithm (Rumelhart and McClelland, 1986; Rumelhart et al. 1986). Similar to how Gerald Tesauro created an agent to play the game of Backgammon (Sutton and Barto, 1998), an agent is created for the purpose of learning how to play the game of Othello, which is a game with a massive state space and a small set of rules. It is a game played between two players with a simple objective: to end the game with more pieces of your color than your opponent's on the board. Other approaches to playing Othello include using min-max trees (that uses a heuristic based on the number of pieces of your color) and by directly assigning values to specific tiles on the board (and the agent would then try to always place a piece on the most highly-valued tile) (Sutton and Barto, 1998). The problem with these heuristic methods is that it is unknown how useful these strategies might be, the agent's play may be predictable, and in the case of game trees it might be impossible to build a tree beyond a small number of moves. By using reinforcement learning to allow the agent to develop its own strategy, human learning can be simulated.

The results of the best trained agent were compared to the performance of world champions. Several world championship games were examined to see how often the agent would choose the same move as the expert and how often the agent would believe that the expert's move was a good one. This comparison is used to demonstrate that learning did indeed occur.

PROBLEM DESCRIPTION

Othello is a game played between two players, Black and White, on an eight by eight grid. Each player has pieces of his color on the board; the goal for the player is to have more pieces of his color than his opponent's on the board at the end of the game. To do this, a player must outflank his opponent's pieces, which then causes them to be flipped over. The following example diagram shows how White is able to outflank her opponent in three different ways, causing exactly five of Black's pieces to become white.

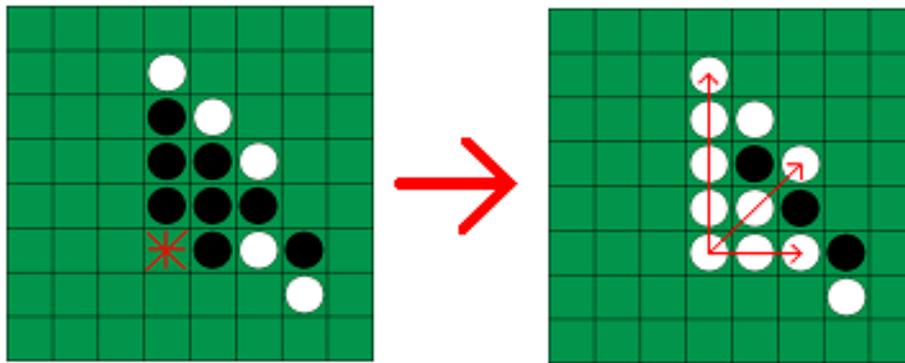


Figure 1: In this example, it is White's turn to move. White places her disc in such a way as to outflank five of Black's pieces. All outflanked pieces are flipped over, thereby taking five pieces from Black and giving them to White.

Every turn, a player must make a move that outflanks at least one of his opponent's pieces; if such a move cannot be made, the player loses his turn. If the other player also cannot move, the game ends (this most commonly occurs when the board is full). When the game is over, the number of pieces of each color is counted; whichever player has more pieces is the winner.

A game state for Othello is defined by the arrangement of black and white discs on the board, and the name of the current player. The next state would occur after the current player has placed a disc on the board, outflanked at least one piece, and flipped all outflanked pieces over to her color. (The upper bound of Othello's state space is 3^{64} : there are 64 tiles, each of which can bear a white or black tile, or be empty).

METHODS

The purpose of reinforcement learning is to find an optimal value function, where the value of it in each state ($V(s)$) represents the estimated discounted reward from that state until the end of the game (Sutton and Barto, 1998; Tesauro, 1995). The value function guides the action taking process: at every move in the game, the action that is taken is the one that leads to the best possible next state. After each action, the agent receives a reward signal based on which the value function is updated. In $TD(\lambda)$ - a variant of reinforcement learning called temporal difference - the observed reward at time t is used to update the value of all states, not only of the current state ($TD(0)$). Previously visited states and states visited closer in time to t - chosen by an eligibility trace - are affected more than the rest. In this way, learning updates not only the present state - either good or bad - but also the states on the path to it (Sutton and Barto, 1998). Learning is hard because the reward signal is received only at the end of the game indicating a win or a loss. Whether an intermediary move is good or bad is known only at the end of the game. By repeatedly playing the game, learning proceeds from the end of the game to the beginning, as reward slowly propagates back from later moves to earlier moves.

Since Othello has a very large state space, the value function is difficult to represent with a table, and more importantly, it is impossible to ensure that each state will be visited a very large number of times - a requirement for convergence to optimal value function. That is the reason why, for problems like the Othello game, a function approximation method is used to interpolate the value function for unvisited states. We use feed-forward neural networks to represent the value function.

The neural network has three layers as shown in Figure 2. There are N nodes in the input layer, H neurons in the hidden layer and a single neuron in the output layer. The input and hidden layers are both augmented with a bias neuron. (Nodes in the input layer are not considered neurons because no processing occurs within them).

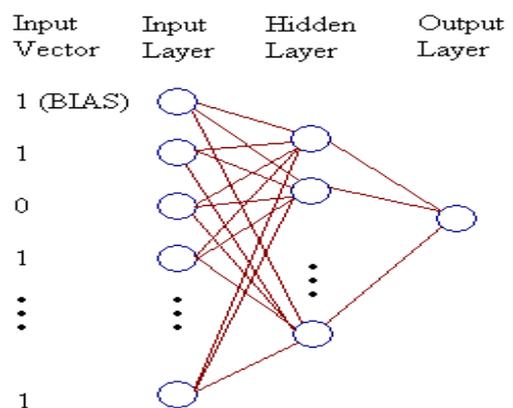


Figure 2: The structure of the neural network.

The activity of the N sensory nodes in the input layer represents the current state of the Othello board. There are 194 input nodes: 192 neurons represent White's and Black's pieces arranged on the 64 tiles, as well as which of the 64 tiles are empty or not; 2 neurons indicate whose turn it is. The hidden neurons receive projections from all input nodes, plus the bias neurons. The activation function of a hidden neuron is the sigmoidal function:

$$(1) \quad x_k = f(h_k) = 1/(1 + e^{-h_k})$$

where $h_k = \Sigma(w_{ki} * x_i)$ is the weighted sum into a hidden neuron, w_{ki} , the weights between input nodes and hidden neurons, and x_i , the values of the input nodes, including the bias neuron. The activation function of the output neuron is also the sigmoidal function. The output of the network represents the value function of the input state at time t : $V(s_t)$.

All weights are updated using the back-propagation algorithm (Rumelhart et al. 1986). The error (δ_t) that is back-propagated from the output to the input is the difference between the estimated reward from state s_t ($r_t + \gamma V(s_{t+1})$) and the current value of that state ($V(s_t)$):

$$(2) \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

where r_t is the observed reward in state s_t , and γ represents the rate at which reward from future states is discounted (Sutton and Barto, 1998). The value of r_t is 0 all through the game, except for the last state in each game, where a reward signal of +1 is produced if Black has won, and a reward signal of 0 is produced if White has won. The update equation for all weights in the network is:

$$(3) \quad W_t = W_{t-1} + \alpha \delta_t e_t$$

with W_t the vector of all weights in the network, α the learning rate, and e_t the eligibility trace, a vector of the same size as W_t (Sutton and Barto, 1998):

$$(4) \quad e_t = \gamma \lambda e_{t-1} + \nabla_{W_t} V(s_t)$$

with λ , representing the eligibility of a weight to being updated. At each time step, the eligibility trace goes down proportional with λ . The increment of the eligibility trace is proportional with the gradient of the network function with respect to the weights: $\nabla_{W_t} V(s_t)$.

Equations (1-4) represent the TD(λ) algorithm implemented on neural networks. The complete learning algorithm is described below:

Initialize the weights of the network to small, random values.
 π : the policy for choosing moves.
 α : the learning rate
 ε : the probability of choosing a random move (used in π)

Repeat for each episode:

$s_t \leftarrow$ The initial state of the episode

Repeat for each step of the episode :

$a \leftarrow$ action given by π for s_t

Take action a , observe reward r , and next state, s_{t+1} .

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

$$e_t = \gamma \lambda e_{t-1} + \nabla_{W_t} V(s_t)$$

$$W_t = W_{t-1} + \alpha \delta_t e_t$$

$$s_t \leftarrow s_{t+1}$$

Until s_t is terminal

Update α , ε

The above algorithm is implemented as follows: Training involves two players (Black and White) sharing a single neural network which is updated after every move is made. The policy, π , for choosing moves is as follows: choose a random move with ε probability; otherwise, choose from among the set of moves that differ in value from the best move by a small amount (usually 0.02). After each game, α and ε are updated by the following equations:

$$(5) \quad \alpha = (\alpha_0) * (0.99^{N_g/200}) + 0.0025$$

$$(6) \quad \varepsilon = (\varepsilon_0) / (e^{N_g/10000}) + .002$$

where N_g is the number of games played.

RESULTS

Fifteen neural networks were trained differing in regard to four key parameters: the number of hidden neurons, the way the learning rate (α) decayed, the way epsilon (ε) decayed, and the number of training epochs. The networks were then pitted against each other in a round-robin style tournament. In order to increase the variety of games played between any pair of networks, ε was reduced to 0 in all cases, but the tolerance for determining the set of best moves was increased to 0.05. After the tournament, the network that achieved the best performance was then chosen for further analysis.

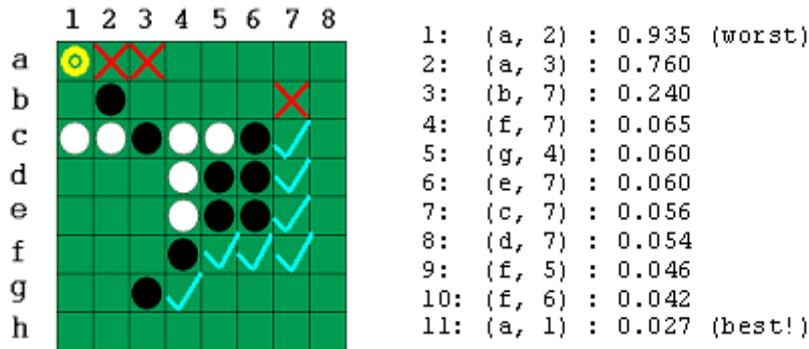


Figure 3 : A sample board evaluation.

The one selected had ninety-six hidden neurons, a learning rate of 0.5 that decayed the most slowly, an epsilon value of 0.9 that also decayed very slowly, and had played 100,000 training games with itself.

Figure 3 gives a sample board evaluation. The network, or agent, looks at each of the next states that can result from each of its possible legal moves. It then evaluates the worth of each of those next states, and then chooses the move that leads to the most valuable state. In this scenario, it is White's turn and White can make 11 moves. Values close to 0 indicate states good for White, and values close to 1 indicate states good for Black. The network considers 3 moves particularly bad (marked by red X's), 7 moves good (blue checkmarks), and 1 move excellent (the yellow circle). The network has learned that the corner can be a valuable place to put a piece. Moreover, it notes that tiles near the corner are very poor places to put pieces. (Placing a piece near the corner would allow the enemy to grab the corner the next turn).

The seemingly best network was further studied to see how its performance compared to the performance of world championship games (Mandt, 2003). The games were presented to the network state by state; what was analyzed was the relationship between the values of what the network thought was the best move and the move the world champion selected. The point of the analysis is to illustrate that genuine learning must have occurred if the network tends to agree with experts. Figure 4 shows the results of this analysis.

Around 30% of the time the network chose the exact same move the expert did, and around 64% of the time the network believed that the expert's move was close in value to the move it chose. Considering that there are about an average of 8 moves a turn, the 30% exact match is much better than the 12.5% chance of choosing the move randomly. It is also important to note how these results would be different for an untrained network. An untrained network would not choose the exact same move as the expert with any great probability; however, since to an untrained network all states are equally valuable, the untrained network would believe that the expert's move was close to the best move in value 100% of the time. The reason that the results are important for a trained network is that, for a given state, the trained network assigns widely different values to the possible next states (as can be seen from the sample

evaluation). The high percentage of closeness of value of the expert's move to the network's move is only relevant to a trained agent.

After this comparison of the network's choices to an expert's, the network played several games against human players experienced with Othello (and thus probably playing at an intermediate level). The players consisted of a senior university student majoring in mathematics, a professor of computer science, and three on-line players who were playing in Yahoo!'s intermediate area for Othello (Reversi) players. The program successfully beat all but the professor, who conceded that the program offered a challenging level of play.

CONCLUSIONS

We have shown that a reinforcement learning algorithm using neural networks was able to learn to play Othello without an external teacher. Considering that the network started with only the rules of the game, it is significant that after training, the network tended to agree with world champions. There is a caveat, however: training an agent to play Othello using a neural network does not result in optimal game play. One problem with this method of learning is that the network only learns to play against one style of strategy. No matter whom the network's opponent, it will always choose the same moves. In other words, the network never considers the opponent's playing strategy. Moreover, against expert computer programs that might employ preprogrammed heuristics, the network is at a disadvantage. One possible improvement that could be made to the neural network program would be to augment its decision-making with a game tree. For example, perhaps during the last five moves of the game, the network could be abandoned and a game tree could be used to analyze those critical last few moves. Although the network only reaches a good level of play, it was able to reach this level on its own without any help from any strategist guiding it. The combination of neural networks and reinforcement learning is thus useful in applications in which no strategies are currently known; these strategies can then be learned by the network.

| | | | |
|------------------------------------|-------|--------------------------------------|-------|
| Exact Match: | | Close (Within 0.02): | |
| Opening : | 21.8% | Opening : | 52.5% |
| Midgame : | 25.1% | Midgame : | 41.3% |
| Endgame : | 41.4% | Endgame : | 50.7% |
| Average : | 29.4% | Average : | 48.2% |
| Fairly Close (Within 0.05): | | Somewhat Close (Within 0.10): | |
| Opening : | 71.8% | Opening : | 93.5% |
| Midgame : | 59.3% | Midgame : | 78.2% |
| Endgame : | 61.0% | Endgame : | 72.6% |
| Average : | 64.0% | Average : | 81.4% |

Figure 4 : An analysis of world championship games.
 Note: The Opening, Midgame, and Endgame consist of 20 moves.
 Not all games last 60 moves.

REFERENCES

Arleo, A. and W. Gerstner. 2000. Spatial cognition and neuro-mimetic navigation: A model of hippocampal place cell activity. *Biological Cybernetics* vol. 83: 287-299.

Foster, D.J., R.G.M. Morris, and P. Dayan. 2000. A Model of Hippocampally Dependent Navigation, Using the Temporal Difference Learning Rule, *Hippocampus* 10:1-16.

Jain, A.K., M. Jianchang, and K. Mohiuddin. 1996. Artificial Neural Networks: A Tutorial. IEEE Computer, Special Issue on Neural Computing, vol. 29(3):31-44.

Mandt, M. Basic Othello Strategy. 2003. <<http://www.generation5.org/content/2002/game02.asp>> (23 Dec 2003).

Moody, J. and M. Saffell. 2001. Learning to Trade via Direct Reinforcement. *IEEE Trans. on Neural Networks*, vol. 12(4).

Rumelhart, D.E. and J. L. McClelland. 1986. Parallel Distributed Processing: Exploration in the microstructure of cognition. MIT Press, Cambridge MA.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams. 1986. Learning representations of back-propagation errors, *Nature(London)*, vol. 323, pp. 533-536.

Sutton, R.S. and A.G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, A Bradford Book.

Tesauro, G. 1995. Temporal Difference Learning and TD-Gammon *Communications of the ACM*, Vol. 38(3):56-68.

Tesauro, G. and J.O. Kephart. 1999. Pricing in agent economies using multi-agent Q-learning. *Proceedings of the Workshop on Decision Theoretic and Game Theoretic Agents*, London, England, 5-6 July(b).

Zhang W., T.G. Dietterich. 1995. "A reinforcement learning approach to job-shop scheduling." *Proceedings of IJCAI95*

"The Cloning of Joella Feinstein." 1992. British Othello Federation Newsletters, February. <<http://www.britishothello.org.uk/cojfall.pdf>> (31 Dec 2003).

TOWARDS A NEURAL CONTROL ARTIFICIAL PILOT

Qiuxia Liang, Patrick Ehlert, and Leon Rothkrantz

Department of Media and Knowledge Engineering

Faculty of Electrical Engineering, Mathematics, and Computer Science

Delft University of Technology

Mekelweg 4, 2628 CD Delft, the Netherlands

E-mail: {Q.Liang, P.A.M.Ehlert, L.J.M.Rothkrantz} @ ewi.tudelft.nl

ABSTRACT

In this paper we will address our efforts to design a neural control system that can control a simulated aircraft, which ultimately should lead to a realistic artificial pilot. The system we designed consists of a flight plan module, the actual neural controller module, and a graphic user interface. The goal of the flight plan module is to manage the global control of the whole system. For the neural controller we chose to use a Forward Modeling and Inverse Controller. The Jordan Network was used to construct the pre-trained identifier and the online learning controller. Our first experiments showed that improvements were necessary to make the aircraft fly more smoothly. Although the aircraft “wobbles” a bit at the start of a new flight procedure, the controller is able to adapt to changing circumstances during flight.

1. INTRODUCTION

The Intelligent Cockpit Environment (ICE) project is a project of the Department of Media and Knowledge Engineering of Delft University of Technology. Originally, the main purpose of this project was to investigate techniques that can be used to create a situation-aware crew assistance system [Ehlert and Rothkrantz 2003]¹. Basically, a crew assistance system functions as an electronic co-pilot looking over the shoulder of the crew of an aircraft and helping out when necessary.

A secondary objective of the ICE project has been to create a realistic artificial pilot, also called flightbot, that can be used for simulations. Such a pilot can increase the realism of flight simulators and enhance the training of real pilots as well as study the different ways different pilots are flying.

In this paper we will address our efforts to design a neural control system that can control a simulated aircraft. The control structure we used for this application is called

Feed Forward and Inverse Control and consists of two neural networks. One is a pre-trained network and the other is an online learning network for inverse control.

2. RELATED WORK

2.1. Simulating pilots

There are several projects that deal with the construction of artificial pilots. Here we will shortly mention two examples found in literature; the TacAir-Soar and TAC BRAWLER projects.

TacAir-Soar is a rule-based system that generates believable human-like pilot behaviour for fixed-wing aircraft in large-scale distributed military simulations [Jones et al 1999]. Each instance of TacAir-Soar is responsible for controlling one aircraft and consists of a Soar architecture [Laird, Newel and Rosenbloom 1987] linked to the ModSAF simulator [Ceranowicz 1994]. The advantage of using Soar is that the reasoning and decision-making of the system is similar to the way humans are generally believed to reason.

TAC BRAWLER is a simulation tool for air-to-air combat developed by the Linköping University in collaboration with Saab Military Aircraft AB in Sweden [Coradeschi, Karlsson, and Törne 1996]. The system is designed specifically for air-to-air combat experts and allows the experts to specify the behaviour and decision-making of the intelligent pilot agents, without the help of a system expert. The agents in TAC BRAWLER are modelled by decision trees. These trees contain production rules that describe the agent’s dynamic task priorities. During one decision cycle, several branches of the tree can be processed in parallel after which all selected actions are evaluated for priority and compatibility.

Both TacAir-Soar and TACBRAWLER try to simulate realistic pilot flight behaviour and both focus primarily on decision-making during flight. However, both systems use a rule-based approach for aircraft control. No project was found that uses a neural control approach.

¹ More information on the ICE project can also be found via <http://www.kbs.twi.tudelft.nl/Research/Projects/ICE/>

The possible advantage of neural control over a rule-based approach is that there is no need to specify rules for pilot behaviour. Although rule-based approaches are very suitable to model normative pilot behaviour, it is much more difficult to model the different styles that different pilots use for flying. Secondly, using neural networks automated learning can be used to avoid the difficult process of explicating flight rules and finetuning rules and parameters. Thirdly, neural networks allow automatic adjustment to changing circumstances, such as different weather conditions, different aircraft, and malfunctioning controls.

2.2. Neural networks and flight control

The first Neural Network (NN) controller was developed by Widrow and Smith in 1963. Since then many applications have shown that NNs can be applied successfully to control unknown nonlinear systems. There have been a number of studies that investigated neural networks for flight control, for example [Calise 1996],[Wyeth et al 2000],[Pesonen et al 2004]. However, all neural control studies try to improve (a particular part of) automated flight control and focus mainly on the control of Unmanned Aerial Vehicles (UAVs), helicopters, and missiles. Their goal is simply to create a controller that functions as good as possible. We did not find any studies that investigated neural networks for simulating realistic flight behaviour of real pilots.

3. SYSTEM DESIGN

3.1. General system scheme

The essential element of a powerful and flexible neural control system is of course the controller itself. However, to create a flightbot we also need some assistant parts, for example a flight-planning system.

Our system has been divided into three parts according to the different tasks and functions which are; the graphic user interface, the flight plan module, and the neural controller module. Figure 1 shows the general system scheme. The user interface sends orders from the user to the flight plan system. The flight plan system will analyse the order to determine whether it is reasonable or not. If it is reasonable, the planning system will create a flight plan, which consists of at least one flight procedure. Then, the planning system will send different data (the desired plant output) to the controller module corresponding to each flight procedure. The controller will produce the necessary control data that will finally be applied to the plant (aircraft).

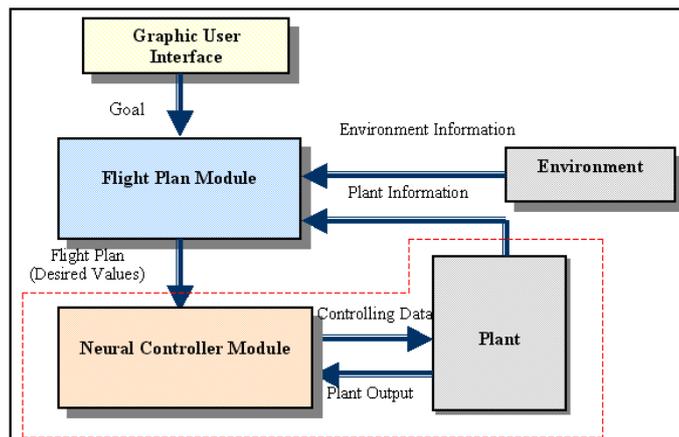


Figure 1: general scheme of the neural flight control autopilot system

3.2. Flight plan module

The goal of the flight plan module is to manage the global control of the whole system. The flight plan module will keep an eye on the flight process and update its flight records to provide the proper plant output data. To be precise the duties of the flight plan module are:

- Analyzing the reasonability of the current goal;
- Deciding on the flight plan;
- Providing the controller with the necessary data corresponding to each part of the flight plan;
- Checking the current flight situation.

Figure 2 shows a scheme of the flight plan system module.

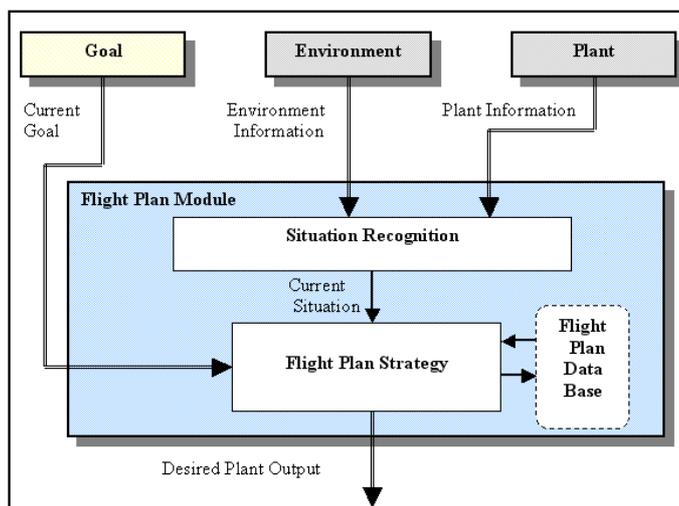


Figure 2: the flight plan system model

The flight plan module chooses one of more procedures based on the goal that is currently set by the user as well as on the state of the aircraft and the environment. Figure 3 shows the (simplified) relationship between the goal of the flight plan module and the chosen flight procedures. With “default flying” we mean straight and level flight.

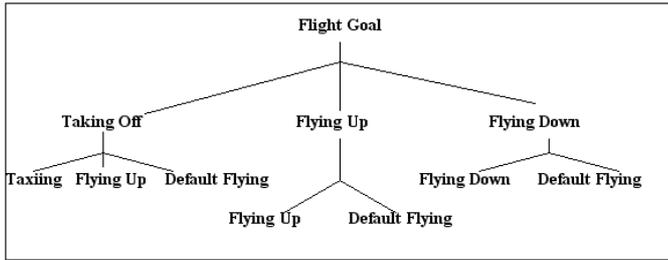


Figure 3: flight strategies

3.3. Neural control module

For the neural control module we have looked into several neural network control topologies. For our purposes we investigated three often-used neural network control topologies: Direct Inverse Control, Neural Predictive Control and Forward Modeling and Inverse Control.

Topology comparisons

In the Direct Inverse Controller the structure will force the network to represent the inverse of the plant. However, there are drawbacks to this approach. First, if the nonlinear system mapping is not one-to-one then an incorrect inverse can be obtained. Second, inverse plant models are often instable, which may lead to the instability of the whole control-system.

The Neural Predictive Controller consists of four components, a plant to be controlled, a reference model that specifies the desired performance of the plant, a neural network modelling the plant, and an optimisation model used to produce the plant input vector. The object is to have an input vector for which the value of the cost function is lower than a defined value. Then the first element of the plant input for current time will be applied to the plant. The Newton-Rhapon algorithm has been widely used for the optimisation model to determine the best-input vector. The main disadvantages of Neural Predictive Control are that numerical minimization algorithms are usually very time consuming (especially if a minimum of a multivariable function has to be found), what may make them unsuitable for real-time applications. When sampling intervals are small, there may be no time to perform minimum searching between sampling. Additionally, the prediction controller requires an accurately trained neural network model to simulate the plant, since the result of the whole controller system depends on the correct prediction value.

The Forward Modeling and Inverse Control (see Figure 4) has an additional NN plant model, compared to the Direct Inverse Control, which is used in the inverse neural network training process. The error signal is propagated back through the forward model and to the inverse model. However, only the inverse network model is adapted during this procedure.

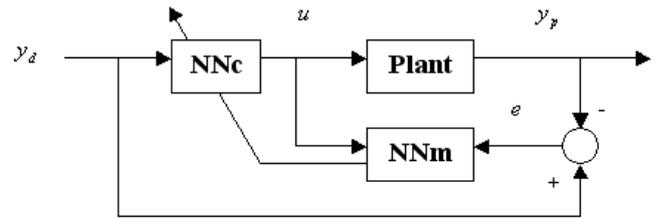


Figure 4: basic Forward Modeling and Inverse Control

The error signal for the training algorithm in this case is the difference between the training signal and the system output (it may also be the difference between the training signal and the forward model output in the case of noisy systems, which is adopted when the real system is not viable). Jordan and Rumelhart [1992] have showed that using the real system output it can produce an exact inverse controller even when the forward model is inexact, which will not happen when the forward model output is used. Another plus is that since the controller neural network gets trained assuming the correct plant input is equal to the backpropagated error from the forward model plus controller output, the training process will be stable.

All things considered, we have chosen the Forward Modeling and Inverse Control, mainly because we want our pilot controller to run in real-time alongside flight simulator software, so we do not have much CPU time available. In addition, as mentioned above Forward Modeling and Inverse Control is better in producing an inverse controller.

Airplane system modeling

Figure 5 shows the representation of the basic airplane model used for our application, which has four inputs (elevator, throttle, rudder and aileron control) and four outputs (airspeed, pitch, heading and bank).

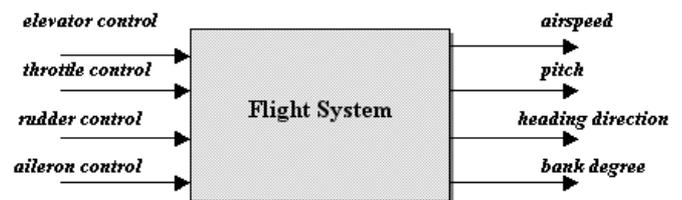


Figure 5: the basic airplane model

The elevator and throttle directly influence the airspeed and pitch of an aircraft, whereas the rudder and aileron directly influence heading and bank. The ailerons control is used to bank the airplane in the direction the pilot wants to turn, and the rudder control is used to keep the nose of the airplane pointing to the direction of turn.

If we only look at the relation between elevator/throttle and airspeed/pitch, we can represent this dynamical system as in Figure 6, which is used for the input and output analysis.

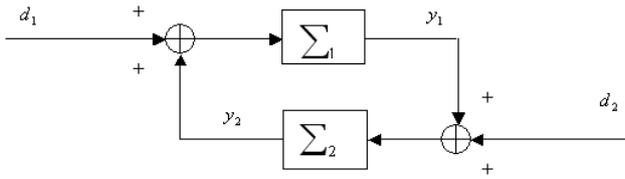


Figure 6: input-output relationship for the elevator/throttle and airspeed/pitch model

This interconnected dynamic system has (d_1, d_2) as input and (y_1, y_2) as output, in which d_1 denotes the elevator input, d_2 denotes the throttle input, y_1 denotes the pitch output and y_2 denotes the airspeed output. In the dynamic sub-system Σ_1 , the input $d_1 + y_2$ produces the output y_1 , which means the current elevator input and current airspeed value determine the pitch value of the next cycle. In the dynamic sub-system Σ_2 , the input $d_2 + y_1$ produces the output y_2 , which means the current throttle input and current pitch value determine the airspeed value of the next cycle.

Besides airspeed and pitch, there are also some other parameters influenced by the throttle and the elevator controls, like altitude and vertical speed. Compared with airspeed and pitch, those parameters are the indirect results of the throttle and the elevator controls. For example, if the airplane is in the air and pitches up, then the altitude will increase and the vertical speed will be a positive value. Therefore, we regard the altitude value and the vertical speed value as the references, instead of the parameters that should be used in the system modelling. For example, when the user sets a flight order for the airplane, besides the flight action he (she) will also be asked to set the altitude the airplane should fly to and this value is checked by the system during flight to analyse the situation.

For more details on how throttle and elevator influences airspeed and pitch, or on flying the Cessna aircraft in general, the interested reader is referred to the Microsoft Flight Simulator manual [2002].

4. IMPLEMENTATION

All software was written using Visual C++ 6.0 environment and in C language. For each module we tested the functions separately before I did a full system's test. The neural networks were implemented using a program that is called the Stuttgart Neural Network Simulator (SNNS). SNNS is an open source program, which not only provides the interface to construct the neural network and simulate its running, it also offers a variety of kernel functions for the creation and manipulation of networks that can be combined in the user's own program [Zell et al 1995]. The simulator we used to test our controller is the Microsoft Flight Simulator 2000. The default Cessna 172 aircraft was chosen for all experiments described in this paper.

For simplicity reasons, in our first implementation we only looked at elevator/throttle and airspeed/pitch. We did not implement turning.

We first trained a neural network to model the airplane plant using SNNS. The topology we used to construct the identifier is the Jordan Network. From experiments we found that, due to its simplicity, the Jordan Network (see Figure 7) is better for on-line training than the other network we tried, a Non-linear AutoRegressive Moving Average (NARMA) network (see also [Liang 2004] for more details).

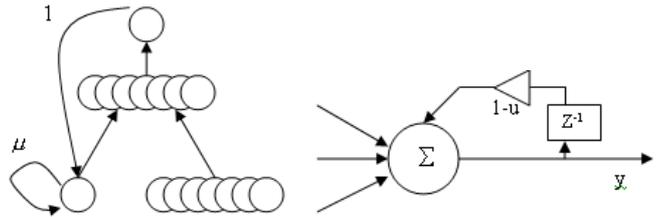


Figure 7: topology of the Jordan Network and the structure of its context PE

The difference between a common neural network and a recurrent neural network, such as the Jordan Network, is that in a recurrent neural network there is a context Processing Element (PE). In the right part of Figure 7 you can see, the one-step delay in the context PE. After this one-step delay, the output of the neuron is returned as an input. For the Jordan network, the context PEs only exists in the input layer, and there is no recurrency in the input-output path.

Based on early experiments, we came to the conclusion that for an identifier whose input-output relationship is not so complex, one hidden layer with around 20 neurons is enough. Of course, one can construct a multi hidden layer neural network with each hidden layer having around 12 neurons. However, it will not improve results much and only waste time in training. Therefore, in our application we used only one hidden layer.

After training the identifier was fixed. The controller was also constructed using the Jordan Network and trained in real-time.

5. TESTING

During the implementation phase of each module, we already tested each function separately. When the implementation was finished we performed a full system's test. During the full system's test we encountered several problems in the current control system:

- The airplane wobbled a lot at the start of each flight procedure (visible in Figure 8);
- The airplane kept descending during the default flying procedure (see the lower picture in Figure 9);

- The airplane changed its behaviour dramatically when going from one flight procedure to another (see the upper picture in Figure 9);

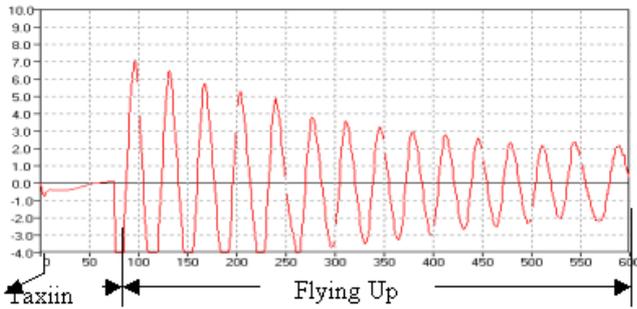


Figure 8: pitch error during the taxiing and flying up procedure

comparison, you may see the pitching magnitude has decreased considerably.

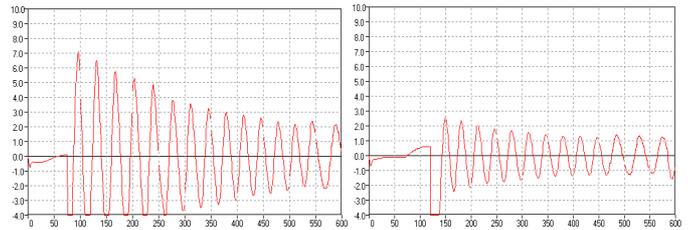


Figure 10: pitch error comparison between original (left) and improved controller (right)

From the pitch signal, which is the lower plot in Figure 11, you can see that at the start of the default flying procedure, the pitch is levelling off gradually, resulting in a slow increase of the altitude until it settles on a certain value. Compared to the altitude plot in Figure 9, it is clear that the improved reference model of the desired plant output makes the airplane fly much better.

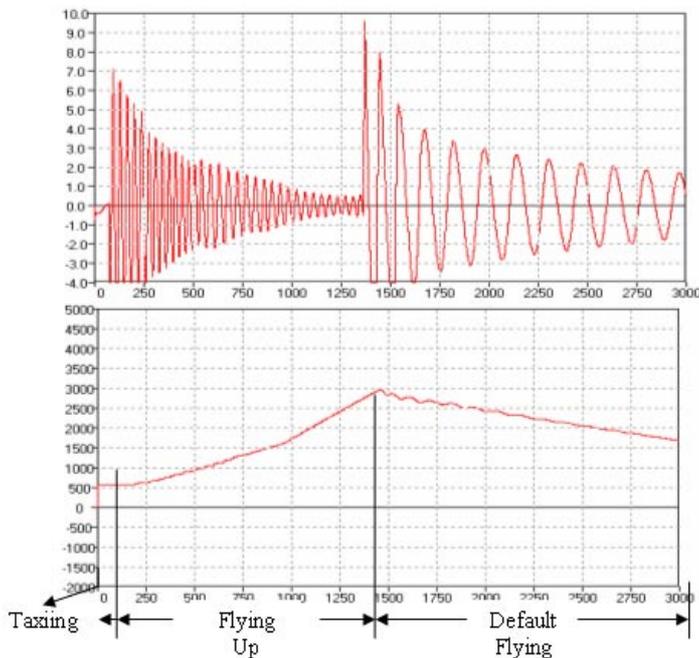


Figure 9: pitch error (top) and altitude (bottom) during take-off

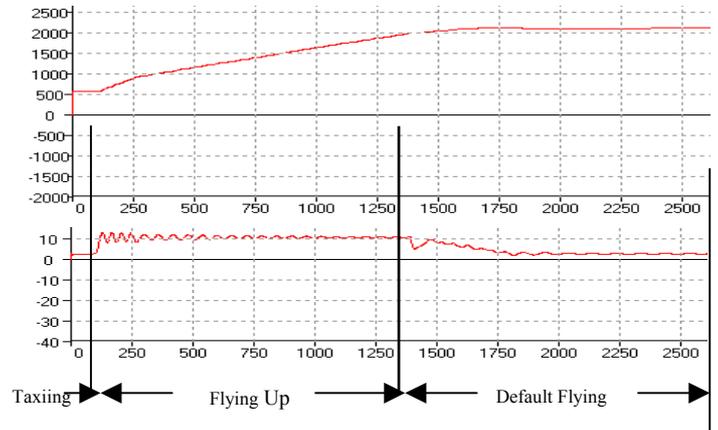


Figure 11: altitude (top) and pitch error (bottom) during the take-off and fly to 2000 feet procedures

6. IMPROVEMENTS AND EVALUATION

Trying to solve the above-mentioned problems, we came to the following solutions;

- Limit the controller's output range of the elevator;
- Change the desired pitch value for the default flying procedure, slightly above 0;
- Modify the reference table used by the controller to make the desired pitch output change gradually.

6.1. Results

Figure 10 shows the pitch error during the taxiing and the flying up procedure. The pitch error shown in the right plot is taken from the airplane controlled by the improved controller and the data shown in the left plot is from the airplane controlled by the previous controller. From the

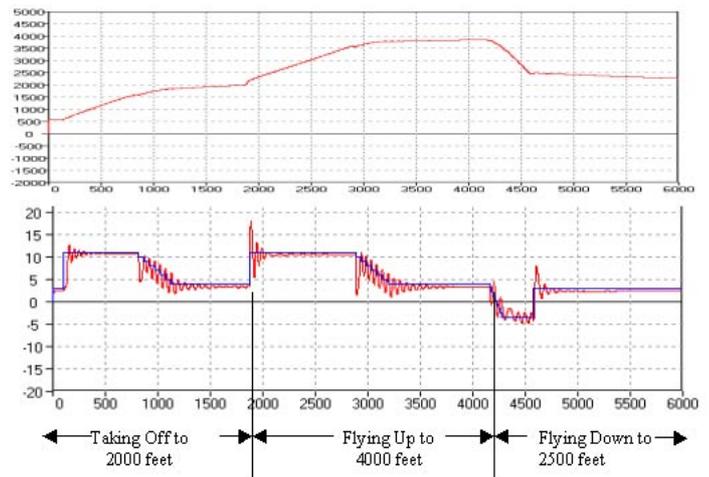


Figure 12: altitude (top) and pitch error (bottom) during take-off, flying up and flying down procedures

Figure 12 shows the altitude and pitch error during a flight were the aircraft flew up to 2000 feet, then up to 4000 feet, and again down to 2500 feet. The red line in the lower part of Figure 12 shows the actual pitch output value while the blue line is used to visualize the desired pitch output value. From the comparison we can see that the neural controller can respond to changes of the desired pitch value immediately. In another words, the controller has fast reaction ability.

6.2. Controller stability analysis

There are several ways to analyse the stability of the controller. For example, we may characterize stability from an input-output viewpoint, or we can characterize stability by studying the asymptotic behaviour of the state of the system near steady-state solutions, like equilibrium points. We prefer to use the steady-state stability analysis, so we studied if the current system is asymptotic stable and characterized the attraction region.

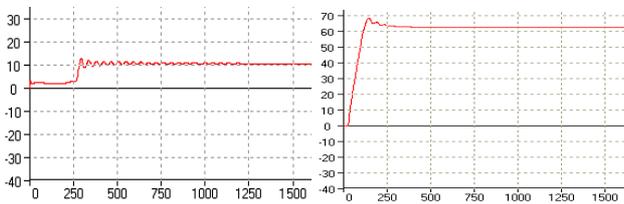


Figure 13: elevator (left) and throttle input (right) during the flying up procedure

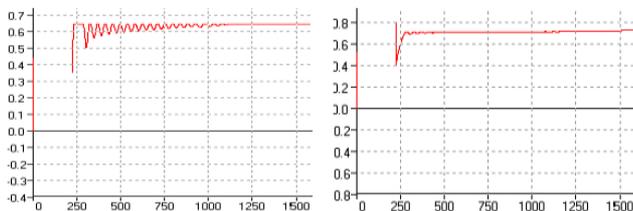


Figure 14: pitch (left) and the airspeed output (right) corresponding to Figure 13

Figure 13 shows that at the start of the control phase both the elevator input and the throttle input start with an arbitrary value. As the process goes on, both slowly settle on a certain value to achieve the desired pitch and airspeed output. Figure 14 includes the corresponding pitch value and the airspeed value taken from the elevator control input and the throttle control input.

With a smaller input the output will be smaller also, and the control inputs will finally settle on a certain value to achieve a certain desired output. These two characteristics indicate that this control system is asymptotic stable.

7. CONCLUSIONS AND DISCUSSION

7.1. Conclusions

The results presented in the previous chapter demonstrate that the current neural flight controller system can:

- Control the airplane to take off, fly up and fly down;
- Run alongside the Microsoft Flight Simulator, which is a large CPU time consuming application;
- Control the airplane so that it achieves a stable flight;
- Respond to the changes of the desired plant output immediately;
- Provide the current flight situation to the user and visualize the evaluation data in 2D coordinates in real time.

The test results also show that the training of the controller neural network is affected by the pre-defined desired plant output. Therefore, setting the proper desired plant output for each flight procedure is very crucial for a good controller system.

As mentioned before, one of our improvements was to limit the output range of the elevator control to decrease the “wobbling” pitching magnitude, but this phenomenon still occurs (to a much smaller extent) at the beginning of each flight procedure. It does disappear as training progresses, as can be seen in Figure 15.

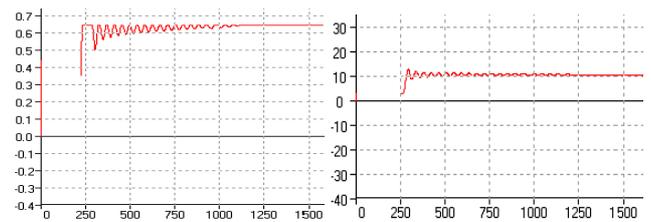


Figure 15: normalized elevator input (left) and pitch value (right) during flying up procedure

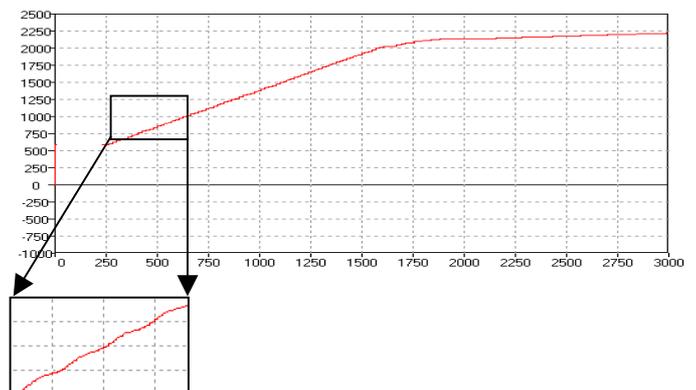


Figure 16: altitude corresponding to Figure 15

7.2. Discussion

Unfortunately, the “wobbling” pitch phenomena cannot be avoided if we stick to the continuous online training of the single neural controller system that we use. However, we feel that this is not a big problem, since this phenomenon can also be found in normal piloting behaviour, which we ultimately intend to simulate. Figure 16 shows that the overall change in altitude progresses as normal and small variations are only visible on closer inspection.

Because of its online training ability, the developed neural controller system can adapt itself to new situations as they arise. This makes our neural network controller more flexible than rule-based control technique, like fuzzy control. With fuzzy control one would define the control rules beforehand, based on the experience of the expert. The advantage of fuzzy control is that for some problems one may have an intuitive idea about how to achieve high performance control, but the consequent problem is that a human expert cannot predict all situations that can occur. Even if the expert is able to predict everything and write it into rules, the rule base will become large and complex, and might not balance the stability criteria and the performance objectives.

Our neural flight control system is flexible and can be applied to different aircraft applications. The architecture will remain the same. Adapting to other aircraft only requires replacing the pre-trained neural network identifier by another suitable one and indicating the desired output of the airplane.

8. FUTURE WORK

As discussed in the last section, the neural control system will make the airplane fly not smoothly at the start of a flying procedure. Although it is not very troublesome we would like to be able to simulate pilots that do not try to correct the pitch. One possible solution we are thinking of is the following. The current neural network controller is trained at each time there is a training pattern available, which will make the controller quite “sensitive”. Instead of training the controller every time a training pattern is available, we could train it when, for example, 10 training patterns are available. Then the airplane could keep the current pitch for a while.

Of course our current neural control autopilot system is still quite limited. It can only control the airplane to fly up and down in a straight line. For future’s work, we plan to incorporate more functions such as turning and landing.

Once we have a fully functional controller we plan to compare the behaviour of the resulting controller with the flight behaviour of the pilot who delivered the training data. Additionally we plan to record flight data from different types of pilots, which would allow us to train different controllers with different flying behaviour.

REFERENCES

Calise, A.J (1996) “Neural networks in nonlinear aircraft flight control”, *IEEE Aerospace and Electronic Systems Magazine*, Vol. 11, Issue 7 (July), pp. 5-10

Ceranowicz, A (1994) “Modular semi-automated forces”, in *Proceedings of the 26th conference on Winter simulation*, Orlando, Florida, US, pp. 755-761

Coradeschi, S., Karlsson, L. and Törne, A. (1996) “Intelligent agents for aircraft combat simulation”, in *Proceedings of the 6th Computer Generated Forces and Behavioral Representation Conference*, pp. 23-25, July 1996, Orlando, Florida, US.

Ehlert, P.A.M. and Rothkrantz, L.J.M. (2003) “*The Intelligent Cockpit Environment Project*”, Research Report DKS03-04/ICE 04, Knowledge Based Systems Group, Delft University of Technology, The Netherlands.

Jones, R.M., Laird, J.E., Nielsen, P.E., Coulter, K.J., Kenny, P. and Koss, F.V. (1999) “Automated intelligent pilots for combat flight simulation”, in *AI Magazine*, Vol. 30, No.1, pp. 27-41

Jordan, M.I. and Rumelhart, D.E. (1992) “Forward models: Supervised learning with a distal teacher”, in *Cognitive Science*, Vol. 16, No.3, pp. 307-354

Microsoft (2002) “*Rod Machado’s Ground School*”, Microsoft Flight Simulator 2002 manual.

Laird, J.E., Newell, A. and Rosenbloom, P.S. (1987) “Soar: and architecture for general intelligence”, in *Artificial Intelligence*, Vol. 33, No.1 pp.1-64

Liang, Q. (2004) “*Neural flight control autopilot system*”, MSc thesis, Research Report DKS04-04 / ICE 09, Delft University of Technology, The Netherlands.

Pesonen, U.J., Steck, J.E. and Rokhsaz, K. (2004) “Adaptive neural network inverse controller for general aviation safety”, in *AIAA Journal of Guidance, Control, and Dynamics*, Vol.27, No.3 (May–June), pp. 434-443

Wyeth G.F., Buskey G. and Roberts J. (2000) “Flight control using an artificial neural network”, in *Proceedings of the Australian Conference on Robotics and Automation (ACRA 2000)*, August 30 - September 1, Melbourne, pp. 65 -70.

Zell, A., Mamier, G., Vogt, M., Mach, N., Huebner, R., Herrmann, K.U., Soyez, T., Schmalzl, M., Sommer, T., Hatzigeorgiou, A., Doering, S., Posselt, D (1995) “*SNNS Stuttgart Neural Network Simulator*”, User Manual, version 4.1, Report No. 6/95, University of Stuttgart. See also <http://www-ra.informatik.uni-tuebingen.de/SNNS/>

LOOKING INSIDE NEURAL NETWORKS TRAINED TO SOLVE DOUBLE-DUMMY BRIDGE PROBLEMS

Jacek Mańdziuk and Krzysztof Mossakowski
Faculty of Mathematics and Information Science
Warsaw University of Technology
Plac Politechniki 1, 00-661 Warsaw,
Poland
E-mail: {mandziuk, mossakow}@mini.pw.edu.pl

KEYWORDS

Bridge, Double Dummy Problem, Artificial Neural Networks, Connection Weights, Knowledge Representation

ABSTRACT

Several feed-forward neural networks were trained to solve double dummy bridge problems. The training was solely based on presentation of sample deals and target number of tricks to be taken by a pair of players. No human knowledge of the game of bridge and even no rules of the game were presented. Analysis of connection weights of trained networks reveals some phenomena, which nevertheless can be explained using human knowledge and experience in the game of bridge.

INTRODUCTION

This paper presents results of training artificial neural networks to solve double dummy bridge problems. In these problems all four hands are fully revealed and the question is how many tricks can be taken by one pair of players under the assumption of optimal play of all sides. This research is the first step on the way to create a program able to efficiently play bridge without explicitly build-in human knowledge. Hence, the training data contained only deals with target information about the number of tricks to be taken by one pair of players. It must be emphasized that the rules of the game were not presented as well.

THE DATA USED IN EXPERIMENTS

All deals used in experiments were taken from GIB Library, created by Ginsberg's Intelligent Bridgeplayer (Ginsberg 2001).

The total number of deals included in this library is equal to 717,102. This set of deals was divided into three parts. The first part containing 500,000 deals was used during training, the next one with 100,000 deals during validation and the remaining deals were used for testing. In most cases each of these three data sets (training, validation and testing) was composed of 100,000 deals.

Each deal included in the library consisted of all hands fully revealed and the number of tricks taken by the pair of players NS under the assumption of perfect play of both parties. In our experiment the attention was fixed on the no trump play with player W making defender's lead.

NEURAL NETWORK ARCHITECTURES

In all experiments feed-forward neural networks with logistic (unipolar sigmoid) activation functions and randomly set initial weights, trained using Rprop algorithm (Riedmiller and Braun 1992), were used. Training and testing were performed using Java Neural Network Simulator (JNNS).

Input neurons

Coding a deal was based on 52 real numbers, one for each card. The cards were ordered in a predefined way, i.e. first Spades, then Hearts, Diamonds and Clubs. In each color cards were ordered from two to ace (see Figure 1). The coded value of each card denoted the player having the card, i.e. 1.0 for N, 0.8 for S, -1.0 for W and -0.8 for E.

Some experiments with other codings were also performed, but accomplished results were worse. The above described way of coding a deal leads to a very fast training. Networks using this coding needed only several hundreds of iterations, compared to a few tens of thousands iterations required for alternative codings.

Hidden neurons

The number of hidden layers and hidden neurons varied. The best network had 25 hidden neurons (52-25-1). The networks with bigger number of hidden neurons accomplished better results for training set, but worse for testing deals. On the other hand networks with fewer hidden neurons yielded worse results, but this degradation was relatively small, including the case of a network without hidden layer (52-1).

Output neuron

In all experiments presented in this paper, a single output neuron represented the number of tricks to be taken by one pair of players. The number of tricks was computed as a linear transformation from integer values: 0, ..., 13 to the range of real values [0.1, 0.9].

Some experiments with different mappings from integer number of tricks to real output value of the network were also performed, however with no significant difference in results. Also some other approaches of representing target number of tricks were tested, but the results were inferior.

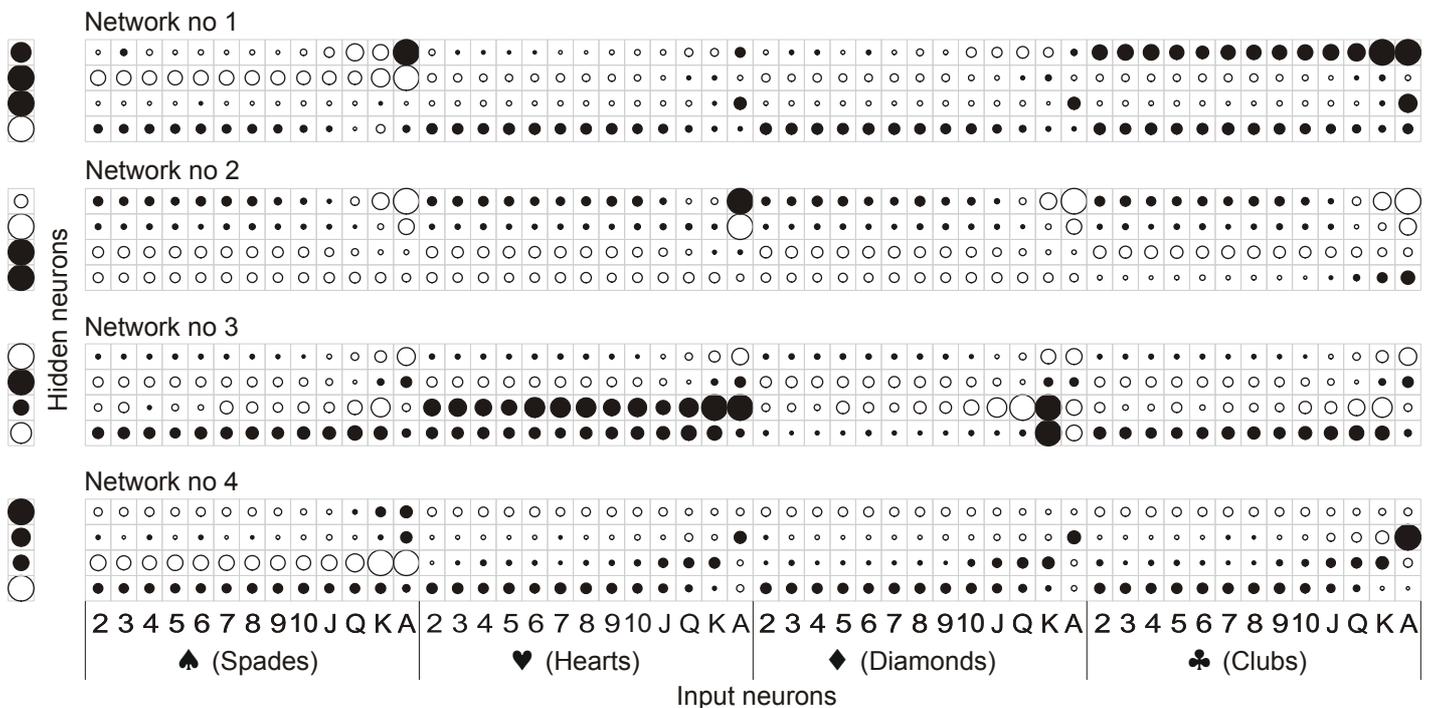


Figure 2. Weights of connections of 4 independently trained neural networks with 4 hidden neurons (52 - 4 - 1).

The simplest trained network had no hidden neurons, so it contained 52 connections, which weights are presented in Table 2.

These weights of connections are very similar to Work point count - the human way of estimating the strength of cards (ace - 4 points, king - 3, queen - 2, jack - 1). In order to have a point of reference this human scoring of cards was used to create a naive estimator. The number of tricks to be taken by NS was estimated by $(13/40) * \text{points_of_NS}$. This estimator for the testing set achieved the result of (86.19 | 61.32 | 22.52).

Networks with 4 hidden neurons (52-4-1)

Fig. 2 presents weights of connections of 4 neural networks, each having 4 hidden neurons (52-4-1). All these networks were trained independently using the same data. The results achieved by them were of similar quality.

It can be noticed that most of weights with the biggest absolute values are assigned to connections from input neurons representing aces and kings. This feature is quite natural - these cards are the most important in the game of bridge, especially in no trump play.

Some "special" hidden neurons, which fix their attention only on one suit, can also be pointed (e.g. the third one in the first network). More such neurons will appear in the networks with bigger number of hidden neurons.

Another interesting phenomenon concerns big absolute values of weights of all connections from hidden neurons to the output one. The absolute value of connection weight determines the relevance of the source neuron, hence a

conclusion that all hidden neurons are relevant in these networks can be drawn from this feature.

Networks with 8 hidden neurons (52-8-1)

Weights of connections of 2 neural networks each having 8 hidden neurons (52-8-1), trained using the same data and achieving similar results, are presented in Fig. 3

The first conclusion, which can be drawn from the figure, is the presence of many hidden neurons focused on one particular suit. Another observable feature is gradually increasing importance of inputs from two to ace.

There exists one hidden neuron which weights of input connections are quite surprising (the first hidden neuron of the second network). This neuron seems to be irrelevant for the network since the weight of its connection to the output neuron equals -0.068, whereas all other connections from hidden neurons to the output one have absolute values greater than 0.499. The number of such neurons increases for more complicated networks. In additional experiments we have checked the effect of pruning such "suspicious" nodes, but doing so resulted in degradation of network's performance.

Network with 25 hidden neurons (52-25-1)

Fig. 4 presents the network with 25 hidden neurons (52-25-1) which achieved the best results (in case of more complicated networks, having more hidden neurons or more hidden layers, the effect of overfitting was observed).

Rectangles drawn using long-chain lines mark parts of input connections of hidden neurons which are specialized in one

suit. This kind of weight pattern is repeatable, i.e. such four

hidden neurons, focused on one suit only, could be found in

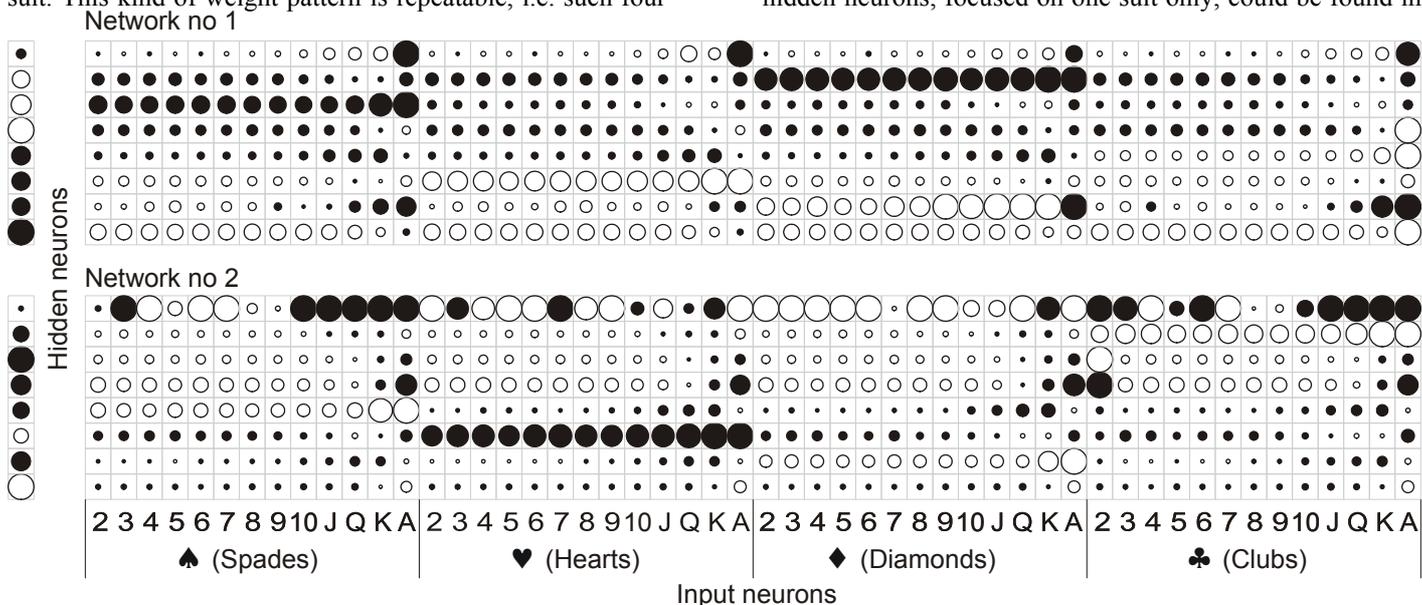


Figure 3. Weights of connections of 2 independently trained neural networks with 8 hidden neurons (52 - 8 - 1).

all neural networks with 25 hidden neurons, trained using the same data.

Comparison of absolute values of connections from input to hidden neurons in any trained network with 25 hidden neurons distinguished exactly 4 leading connections. These connections had absolute values much greater than the others (in the network presented in Fig. 4 values of these connections were equal to: -26.89, 26.71, 26.69 and 24.19, resp. while the biggest absolute value of the remaining connections was equal to 6.47). It is not surprising that all these distinguished connections had input neurons assigned to aces.

Another interesting feature which appeared in all trained neural networks with 25 hidden neurons, was the presence of 4 hidden neurons specialized in five cards from one suit: ten, jack, queen, king and ace (in Fig. 4 marked using the dotted line). In all these groups the most important were queens and kings, jacks were less important, but still much more relevant than aces and tens. The hypothesis is that these hidden neurons are responsible for very important aspect of the game - the finesse.

CONCLUSIONS

The most important conclusion, which can be drawn from analysis of connections of trained neural networks, is the possibility to explain some patterns using human knowledge of the game of bridge.

In the trained networks estimating strengths of suits, which is fundamental in human analysis of a deal, is performed by assigning one hidden neuron for each suit (in Fig. 4 weights of these neurons are marked with dotted line). Such neurons consider values of cards - the connection from input neuron representing an ace has weight of biggest absolute value and the connection representing two - the smallest one.

Another four hidden neurons are specialized in a group of cards from one suit - king, queen and jack. This is also a part of human analysis of a deal, which allows to take into account a possibility of finesse - very important aspect of the play phase.

It must be emphasized that all networks were trained only by presenting deals and target numbers of tricks. There was no human knowledge of the game involved in training and actually even the rules of the game were not implemented. In this context results achieved by networks and the existence of particular weight patterns look interesting and give the promise for future research.

FUTURE PLANS

The goal of experiments presented in this paper was to verify neural networks' ability to solve double dummy bridge problems. The best networks were able to perfectly point the number of tricks in more than one third of deals and gained about 80% accuracy when one trick error was permitted. Only in less than 5% of deals the error exceeded 2 tricks. In our opinion these results are satisfactory, especially when the fact of avoiding presentation of human knowledge and the rules of the game is emphasized.

On the other hand small difference in results between the best network and the simplest one (the one without hidden neurons) is quite surprising. The training set included 100,000 deals with no pre-selection. The hypothesis is that in such a big set of input data each network fixed its attention on the most important aspect (from statistical point of view), i.e. scoring hands' values.

The ultimate goal of this research is to create a bridge playing program under the assumption of avoiding explicit presentation of human knowledge and experience in any form. Due to this assumption we consider using

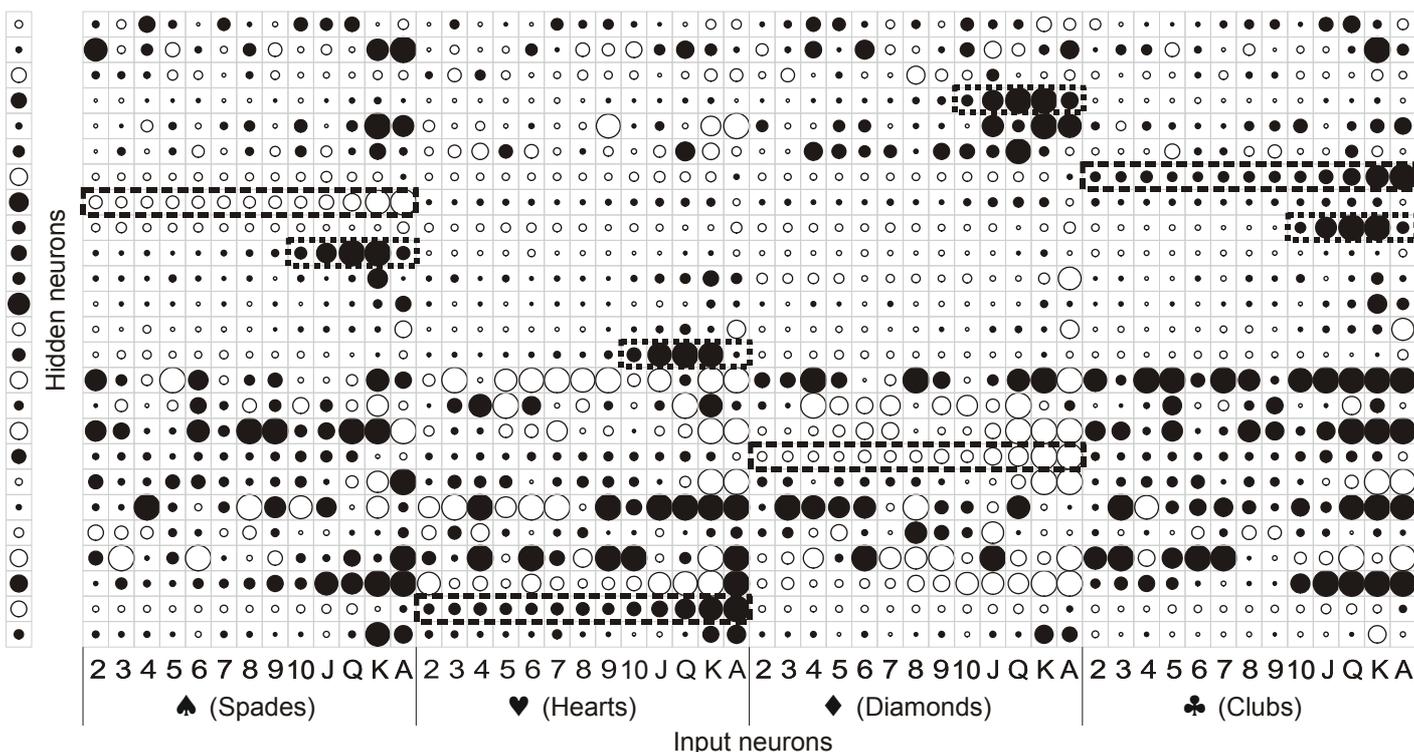


Figure 4. Weights of connections of trained neural network with 25 hidden neurons (52 - 25 - 1).

reinforcement learning methods in further development of this work. Results of experiments described in this paper give a hope of successful application of neural networks as supporting tools for these methods. The very next step of this research is to apply an ensemble of networks each of which is specialized in particular types of deals, in order to enhance the overall test score and to learn more about networks' internal representation of the game.

REFERENCES

- Ginsberg M.L. (2001), GIB: Imperfect Information in a Computationally Challenging Game, *Journal of Artificial Intelligence Research* 14, 303–358.
- Mossakowski K, Mańdziuk J. (2004), Artificial Neural Networks for Solving Double Dummy Bridge Problems, In: L. Rutkowski, J.H. Siekmann, R. Tadeusiewicz and L.A. Zadeh (Eds.), *Artificial Intelligence and Soft Computing - ICAISC 2004, 7th International Conference, Zakopane, Poland, Lecture Notes in Computer Science 3070, Springer-Verlag, 915–921.*
- Riedmiller, M., Braun, H (1992).: A fast adaptive learning algorithm. Technical report, University Karlsruhe, Germany

BIOGRAPHY

Jacek Mańdziuk, Ph.D., D.Sc., received his M.Sc. (Honors) and Ph.D. in Applied Mathematics from the Department of Applied Physics and Mathematics, Warsaw University of Technology, Poland in 1989 and 1993, resp.

and the D.Sc. degree in Computer Science from the Polish Academy of Sciences in 2000.

He is currently Associate Professor at the Warsaw University of Technology and Head of Computer Department in the Warsaw branch of the Mazovian Gas Company.

His research interest include game playing, financial modeling, time series prediction, pattern recognition, optimization, constructive learning/training methods for neural networks.

Dr Mańdziuk is the author or co-author of one book and about 40 scientific papers published in international journals and conference proceedings.

He was a Program Committee member of various international conferences, Program Co-Chair of International Workshop on Adaptive Systems in Soft Computing and Life Sciences and served as the referee for several international journals related to computational intelligence. He is the member of International Neural Networks Society.

In 1996/97 he has been awarded the Senior Fulbright Research Grant and visited the International Computer Science Institute in Berkeley and the EECS Department University of California at Berkeley.

In the artificial game playing domain he is currently pursuing research in the area of self-learning and self-improving systems, able to learn without providing any human knowledge or experience. His main interests are devoted to the game of bridge (where he is conducting research together with Krzysztof Mossakowski), the game of give-away-checkers (together with Daniel Osman) and the game of Reversi/Othello.

For further information please consult the web page: <http://www.mini.pw.edu.pl/~mandziuk>

USING NEURAL NETWORK TECHNIQUES WITHIN A GO PLAYING PROGRAM

JULIAN CHURCHILL, RICHARD CANT, DAVID AL-DABASS

School of Computing & Informatics
Nottingham Trent University
Nottingham NG1 4BU
E-mail: richard.cant@ntu.ac.uk

KEYWORDS

Go, Neural Networks, Artificial Intelligence, Games.

ABSTRACT

Artificial neural network algorithms represent a valid approach to the exploration of search strategies for computer games systems. The oriental game Go presents one of the most challenging search problems in the games domain. A set of experiments using artificial neural networks are presented in this paper to investigate the application of these techniques to the game of Go in particular and other computer games in general.

INTRODUCTION

This paper presents experiments with neural network techniques, designs and methods of use, within a Go playing framework. The idea behind this was to develop a neural network that could be used with a game tree search algorithm to provide a selective full board search for the best move available. The network would be used to assign values to given moves allowing efficient node ordering within a game tree and a method to select branches to prune if required.

What is Go?

Go is a board game with its origins in the Far East. It is a relatively simple game the complexity of which emerges as you become familiar with the ideas presented. A comparison with Chess is often made, as these are both board-based games of zero-chance (Burmeister and Wiles 1995). The rules are simpler in Go, however the board is larger and due to the unrestrictive nature of the rules there are many more moves available for the Go player to consider.

A board of 19x19 intersections is used onto which the two players, one placing black stones the other white stones, take turns to play a single stone of their colour onto an empty intersection. Black plays the first move in a non-handicap game. The main aim of the game is to surround as much territory as possible whilst confining your opponent to a minimum amount of space on the board. Territory is defined as empty intersections that one side or the other is clearly in uncontested control of.

A player can pass at any turn instead of placing a stone. Stones may be captured anytime after they have been placed on the board and are removed if they are captured. The captured stones are added to the players score at the end of the game. The concept of liberties play an important part in the process of keeping stones alive or killing them. A liberty is any empty intersection directly adjacent to a stone and can be thought of as breathing space for the stone or stones it is adjacent to. A stone is captured when the last of its liberties are removed. Connected stones of the same colour share liberties and will live or die together. Suicide counts as an illegal move unless it captures some opponent's stones in the process.

The end of the game is reached when both players pass consecutively signalling a mutual agreement that the game has gone as far as it can. Stones that are effectively dead, captured stones and territory points are then totalled up and the winner declared. A small amount of points is usually added to whoever is playing white, since as black plays first in an even game a small advantage is assumed.

The board edges and corners have an important role in the game as they are treated as an imaginary wall of opposing colour stones. This means a lone stone on the edge will have only 3 liberties, as opposed to 4 away from the edges, and in the corner will have only 2.

NEURAL NETWORKS

The inspiration behind the neural network idea is the simulation of neurons in a biological brain. Biological neurons receive stimulus signals from other neurons and when a certain activation level is reached the neuron fires signals to all the other connecting neurons. The change in the strength of the connections is dynamic and it is this particular feature that allows networks of neurons simulated on a computer to be trained to respond to patterns of input, giving appropriate patterns of output. More information can be found in (Heitkoetter and Beasley 1994; Fausett 1994).

Artificial neural networks can be constructed from simple processing units, artificial neurons, and can be trained to learn complex functions. This seems an

appropriate method to use in a Go playing program, since these techniques offer a possible way to capture some of the intuitive move recognition required to play at more advanced levels.

OUR IDEAS

Many uses for neural network methods can be imagined for application to the problem of playing Go. The authors original intended use for a neural network was to score proposed moves given a local board situation, for instance a square area of 9x9 intersections centred on the move to be score. For a complex task such as this there is also the matter of interpreting the output of the neural network. Ideally it would be a score relative to all other possible moves at that point in the game but if it was then it would be easy to write an excellent if not perfect Go playing program.

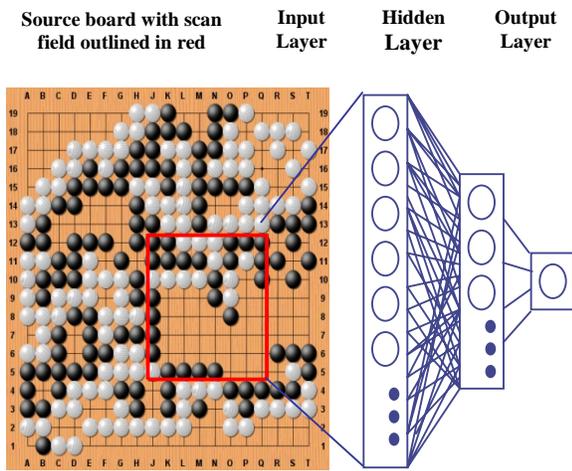


Diagram 1 – Using the neural network

The neural network output must represent some measure of the usefulness of the move at this point in the game or the training would not improve, so we choose to call this output a plausibility score, to distinguish it from the idea of a global absolute ‘one move is better than another’ score. Diagram 1 illustrates how the network would be used in practice.

A Neural Network as a Move Ordering Mechanism

Game tree search algorithms allow the construction of a tree of possible moves and responses. When coupled with an evaluation function to assess tree nodes the best available move can be deduced within a given depth. This is a good solution for problems that have a very shallow depth and narrow branching factor but as soon as we get into games any more complicated than Tic-Tac-Toe, the limits of game tree search become apparent. To know the very best move at any position in a game of Go would require the search to cover an enormous search space, estimated at around 10^{170}

positions (Allis et al. 1991). The number of moves available at each position in a game of Go averages at around 180, combined with a game length of around 200 moves makes a straightforward game tree search impractical.

The order in which moves are generated and processed by the search algorithm can greatly affect the efficiency of the search when the common Minimax algorithm with Alpha-Beta pruning is used. Minimax is a game tree search algorithm for zero-chance two player games. (Russell and Norvig 2003) The efficiency of the Alpha-Beta pruning depends on a good ordering of nodes; the best available moves should be examined as early as possible. One use for the developed neural network would be to enhance a game tree search program by using the network to provide a good estimated ordering.

Input Schemes

Much time was spent on experimenting with different algorithm parameters, input field shapes and sizes, how to use the training data and various network input schemes. The methods for encoding the board area around the move to be scored incorporated and contrasted quite a few different features. Varying input schemes for the neural network were created, to allow comparison of the contribution each feature made towards evaluating a move.

A lot of information can be extracted from a raw board position. One of the challenges of designing the networks was deciding which features would contribute positively to the network performance and learning capability. Some features impacted negatively on the network, they overbalanced or mislead the network during learning due to the feature being overly strong compared to other features. One such feature found to produce a derogatory effect on the performance of a network trained by other researchers, was the distance to the last move played (Enderton 1991).

It seems an intuitively important feature since play in Go tends to be near the last move played, until one player has built up enough confidence in the local position to play elsewhere on the board. The frequent occurrence of consecutive moves played close to each other in the training data resulted in this feature dominating the network’s play, disproportionately scoring moves that were closest to the last move played. Part of the reason for this feature causing such a problem was the fact that these networks had no strategic input (nor indeed was this intended in their design). The networks trained here and by some other researchers were designed to look at a static local position without temporal knowledge, information about the global board situation or therefore any form of tactical or strategic planning. These concepts were to be dealt with by more appropriate mechanisms such as game tree search.

Redundancy within feature sets can easily occur, where one feature implicitly provides the information of one or more other features, so experiments were conducted with networks with varying input architectures to assess the impact of the features. The full set of features available and the actual input schemas used are detailed in Table 1. The experiments mostly focussed on the use of multiple state neurons and the inclusion of distance to the board edge information.

The initial network design had only raw input from the board with a very simple encoding scheme: 1 neuron per point, with the value indicating the board point content as follows: *black*(+1), *white*(-1), *empty*(+2), *offboard*(-2). After this design multiple state neurons per point were used to represent the contents of the board point by selecting only one of these state neurons to be activated at a time. For the input schemes with 3 neurons per board point the states were *ourcolour*, *theircolour* and *empty*. Those with 4 neurons per board point had the addition of an *offboard* state neuron.

| Input Scheme | State Neurons | Distance Neurons | Other Features |
|---------------------------|---------------|------------------|---|
| 0 - Raw | 1 | 0 | - |
| 1 - 3 states | 3 | 2 | - |
| 2 - Binary distance | 3 | 18 | - |
| 3 - Offboard state | 4 | 18 | - |
| 4 - Liberties | 4 | 18 | Liberties of surrounding strings. |
| 5 - Distance to last move | 4 | 18 | Distance to last move. |
| 6 - Stress last move | 4 | 18 | Last move emphasised in input. |
| 7 - Binary states | 2 | 0 | Binary encoding of states using <i>ourColour</i> and <i>theirColour</i> |
| 8 - Binary states | 2 | 0 | Binary encoding of states using <i>black</i> and <i>white</i> |
| 9 | 3 | 0 | - |
| 10 | 4 | 0 | - |
| 11 | 4 | 2 | - |
| 12 | 3 | 18 | - |

Table-1 – Input Scheme Details

The distinction between *ourcolour* and *theircolour* rather than *black* and *white* was made to allow colour reversal invariance to be handled. Just as with rotation and reflection symmetries, colour reversal results in an equivalent position, but from the other colours point of view unlike rotation and reflection. To train the network to deal with this would require presenting the network with a colour reversed pattern for each training pattern, effectively doubling the training time. However solutions are possible to solve this issue during network input pre-processing. One method is to always look at training pairs from blacks point of view, then whenever the network is actually used, training included, the stone colours must be reversed if it is being used to score potential moves for white. The alternative method used here is to always score moves from the point of view of

the colour to play – represented by *ourColour*. This is a more appropriate method to use when the input to the network is separated into the possible board point states.

To limit the number of weights and so also the amount of computation time required to use and train the networks, designs 7 and 8 looked at combining the states and encoding them through only 2 neurons. In scheme 7 an encoding used successfully by other researchers is included (Schraudolph et al. 1994). Using 2 neurons the states are encoded as *ourcolour*(+1,0), *theircolour*(-1,0), *empty*(0,+1), *offboard*(0,-1). For scheme 8 the states *black* and *white* replace *ourcolour* and *theircolour*.

Input schemes 4, 5 and 6 introduce some simple extra features to the input. Scheme 4 adds the liberties of the surrounding strings, providing 4 neurons per direction, north, south, east and west. One of the set of 4 neurons per direction is activated to indicate 1, 2, 3 or >3 liberties for the adjacent string if present. Scheme 5 adds an extra neuron to encode the Euclidean distance to the last move played. Scheme 6 handles this slightly differently by emphasising the input signal of the point that was last played.

ALTERNATIVE IDEAS

The main aim for developing the network was to aid the ordering of moves but other options were also considered, amongst them an area finder network that could identify the most urgent part of the board to play in. Also the problem of moves being scored according to the local situation without considering the relative value of alternative moves was looked at.

The urgency network idea was developed to cope with the problem of having move plausibility scores that related only to the board state with no sense of whether one move should be played before another. A network architecture that made the move scores relative to each other would be much more useful and accurate for the task of ordering the moves. To be able to adjust the scores so the network could output a relative urgency value required it to consider at least two moves at a time. A design was drawn up to train a network that could judge whether one move was more urgent than another. In actual play an appropriate sorting algorithm could be used to order the full set of available legal moves at a given position after using the network.

Two alternative designs were created. The first used the entire 19x19 board as input; introducing two additional neurons per board point to the four board point state input schema and dropping the redundant *offboard* state neurons. The additional neurons were labelled MOVE A and MOVE B, each to be activated for only a single, exclusive board point, the points corresponding to the first and second move to be compared. Two output neurons were specified to allow the network to output a relative score for each MOVE A and MOVE B. This

would let the program using the network compare the moves relative to each other. If the scores were not different enough from one another, i.e. both were less than 0.5 or greater than or equal to 0.5, then no confidence in the quality of the relative comparison was assumed. For training, the example games were stepped through taking two consecutive moves of the same colour at each board position and randomising between presenting the first move as MOVE A or MOVE B. This was done so that neither MOVE A nor MOVE B would learn a bias as to which neuron represented the more urgent move. The more recent move was given an output value of 0.9, the later move, 0.1.

The second design took advantage of the previous move scorer network experiments and attempted to use some of the trained weights from those networks that had trained successfully. It used the four state input pattern and so could use a ready trained set of weights from any network that used the same schema. An additional 2 hidden layers of neurons were added to create the urgency network and the first two layers taken from the ready trained networks were frozen, hopefully allowing the network to assimilate information about relative scoring in the last two layers.

The training for this design used a reinforcement scheme. Again two consecutive same colour moves were scored and compared at each ply in the training games. This time if the earlier move didn't score at least 0.2 better than the later move a reinforcement factor of +0.2 was added to the output score and used as the new output training vector for the earlier move. The output score for the later move had a reinforcement of -0.2 added to its score and this was used as the later moves output training vector. Some initial results were gathered from these designs. The first design stabilised quickly with a poor performance score, suggesting that perhaps the network had only learned that the more urgent move was MOVE A 50% of the time and MOVE B the rest of the time. The second design also stabilised very quickly, the extra hidden layers adapting to the ready trained weights fast, however the performance never bettered the performance of the original source network indicating that nothing new was learnt and that the extra hidden layers had merely adapted around the frozen source weights.

EXPERIMENTS AND RESULTS

Unless specified the training data that was used for training all the neural networks was taken from the No Name Go Server (NNGS), a games server where Go players can conduct games in real-time over the Internet. All games played from the year 1995 to 2003 were collected and in all the experiments where the playing rank of the training games was not being varied, games with confirmed player ranks of between 25 Kyu and 5 Dan were used. The ranks were calculated automatically by the server based on player performance.

Unfiltered this gave a total of 395,972 game records in Smart Game File Format (SGF 1997), giving approximately 80 million training pairs. The results presented here represent some highlights from a larger more comprehensive set of experiments.

Where it is usual to use a fixed size training set, we took the approach of using a non-repeating set of data. Since we had a large number of example games from the NNGS server we felt it was unnecessary to impose this training set size limitation. For comparison, experiments were also conducted with limited sized training sets and the results are compared later in this paper. The validation data set consisted of 10 games in SGF format selected from the Nihon Ki-in Championship, the Tengen Tournament and a few professional level games from NNGS. The networks were tested every epoch, for the purposes of non-repeating training sets an epoch was set to 20,000 training patterns. The value used to monitor the progress of the neural network training was the average rank that the network placed the move actually played, in each game from the validation set, compared to all other legal moves globally available in that position.

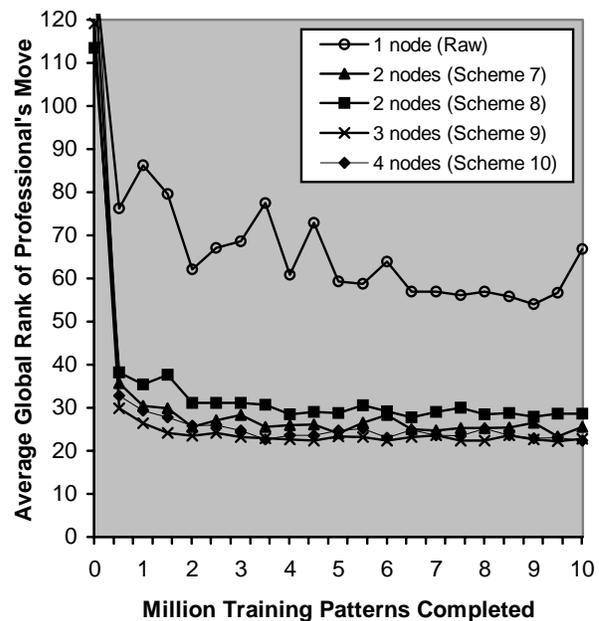


Figure 1 - Comparing the Number of Input Neurons Per Board Point

Figure 1 shows the training progress for network designs with differing state separation policies. The first obvious item to note from these results is that separating the possible states a board point can be allows the neural networks to be trained to a higher quality and a much more stable weight set than by not separating the states. This means that the network saves training time and effort from having to classify the input itself. This is a simple form of input pre-processing which would otherwise require a number of weights or possibly an

entire layer of free parameters to be dedicated to this straightforward classification task. Further improvements are apparent as the various states are allocated individual neurons. Of course the extra neurons will cause the network to use more processor time; a balance between accuracy and speed must be found to suit the networks intended use.

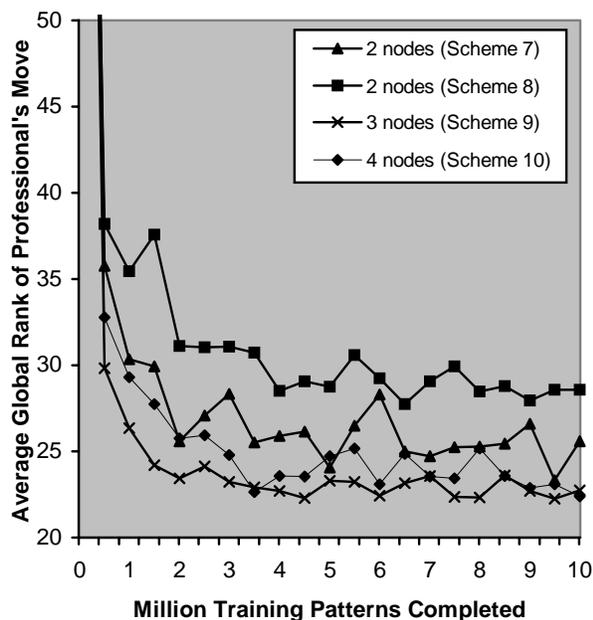


Figure 2 - Detail of Figure 1

Figure 2 shows a detailed view of the more complex input schemes. Scheme 8 with the encoding of the board point contents using 2 neurons, *black*(+1,0), *white*(-1,0), *empty*(0,+1) and *offboard*(0,-1), makes efficient use of the available input neurons and allows properties that some states share to be included, such as both black and white assigning a value to only the first neuron, encouraging the stone present/absent distinction to be learnt for both the black and white states. Surprisingly further improvement is gained from the addition of our method of colour symmetry handling in scheme 7, although it is unclear why this should be so.

The difference between using 3 state neurons or 4 seems to be uncertain, but the similar performance certainly suggests that the extra *offboard* state neuron that scheme 10 provides may be redundant. Without an explicit state neuron the *offboard* state would be implicitly encoded by not activating any of the other state neurons. From figure 2 it appears that this is enough for the network to take advantage of the properties of the board edges.

The graph in figure 3 displays a comparison between training over fixed size training sets and a non-repeating set. From this it can be observed that the non-repeating

set clearly allowed the neural network to train to a higher standard than the fixed size sets. It can also be noted that the size of the fixed size training sets directly correlates with the training performance. A larger training set produces better results. The actual sizes of the sets used were 20,000 patterns for the small set, 100,000 for the large set and 140,000 patterns for the very large set. It also appears that a limit to the performance of a repeated set, given a large enough size, could be reached and that the non-repeating set either exceeded or at least equalled that limit.

Of further note is the appearance of overfitting, a common phenomenon when training neural networks. This may be observed in the gradual performance degradation shown by the small set after about 1 million training patterns have been processed. It might be conjectured that the use of a non-repeating training set prohibits the occurrence of over fitting since the training is not being restricted to an artificial subset of the problem.

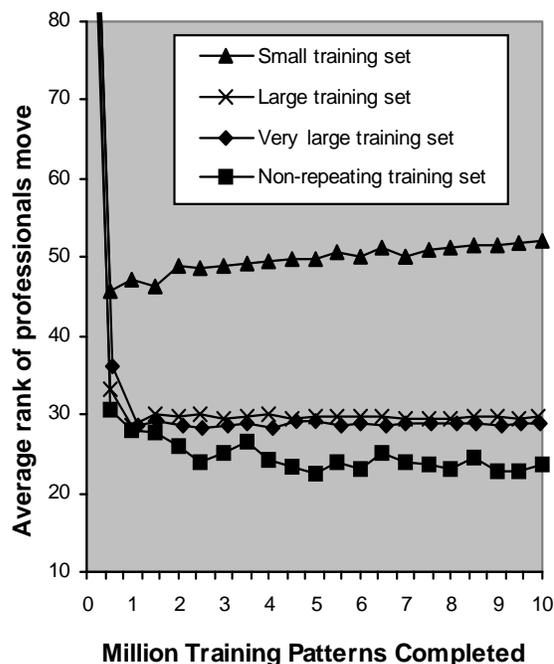


Figure 3 - Using Repeated Training Sets

The sheer mass of possible positions in Go lends itself towards a non-repeating set where it will be large enough to encompass as many possible situations as possible and as with any learning-by-example problem a well chosen training set should improve performance further.

CONCLUSIONS AND FUTURE WORK

The results presented in this paper compare some different network architectures and training methods relevant to training a neural network to evaluate a

potential Go move given some information about the local board area. Several networks were trained that could be used in a Go playing program to increase the efficiency of game tree search mechanisms already in place.

The neural network experiments revealed that under the conditions described in this paper, separating the possible board point states and using individual neurons to indicate the condition of each point could increase the training performance. A performance increase was also noted when using a non-repeating training set, as opposed to a traditional finite sized set. This may be a useful alternative to creating a well-chosen set of training patterns, when a large amount of well spread domain data is available.

The results of further experiments currently being conducted to determine the value of combining this hard, algorithmic, game tree search approach with more flexible machine learning methods represented by neural networks will be presented in a future publication.

REFERENCES

Allis, L. V.; H. J. Van Der Herik and I. S. Herschberg. 1991. "Which Games Will Survive?". In *Proceedings of Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*, pages 232 – 243, Ellis Horwood.

Burmeister, J. and J. Wiles. 1995. "An Introduction to the Computer Go Field and Associated Internet Resources" Available on the Internet at <http://www2.psy.uq.edu.au/~jay/go/CS-TR-339.html>

Churchill, J.; R. Cant and D. Al-Dabass. 2004. "A Hybrid Genetic Alternative To Game Tree Search In Go", *UKSIM2004, Conf. Proc. of the UK Simulation Society*, St Catherine's College, Oxford, 29 - 31 March 2004, pp156-161, ISBN 1-84233-099-3.

Churchill, J.; R. Cant and D. Al-Dabass. 2003. "Genetic Search Techniques for Line of Play Generation in the Game of Go", *Game-on 2003, 4th International Conference on Intelligent Games and Simulation*, IEE Savoy Place, London, November 19-21, 2003, pp233-237, ISBN 90-77381-05-8.

Enderton, H. 1991. "The Golem Go program" *Technical Report CMU-CS-92-101*, School of Computer Science, Carnegie-Mellon University. Available on the Internet at <ftp://www.joy.ne.jp/welcome/igs/Go/computer/golem.s.h.Z>

Fausett, L. 1994. "Fundamentals of Neural Networks: Architectures, Algorithms and Applications" *Chapter 6*, Prentice-Hall, ISBN 0-13-334186-0.

Heitkoetter, J. and D. Beasley, eds. 1994. "The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions (FAQ)", USENET : comp.ai.genetic. Available via anonymous FTP from <rtfm.mit.edu:/pub/usenet/news.answers/ai-faq/genetic/> About 90 pages.

NNGS, No Name Go Server. Additional information can be found at <http://nngs.cosmic.org/>

Russell, S. and P. Norvig, 2003. "Artificial Intelligence – A Modern Approach", 2nd edition, pp165-171, Prentice Hall.

Schraudolph, N.; P. Dayan and T. Sejnowski. 1994. "Temporal Difference Learning of Position Evaluation in the Game of Go". In *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann. Available on the Internet at <http://www.gatsby.ucl.ac.uk/~dayan/papers/sds94.pdf>

SGF, Smart Game File Format, FF[4]. 1997. Specification by A. Hollosi. Additional information can be found at <http://www.red-bean.com/sgf/>

LEARNING TO FIGHT

Thore Graepel, Ralf Herbrich, Julian Gold
Microsoft Research Ltd.
7 J J Thomson Avenue, CB3 0FB
Cambridge, UK
Email: {thoreg|rherb|jgold}@microsoft.com

KEYWORDS

Reinforcement Learning, Fighting Games, SARSA, Q-Learning, Markov Decision Process

ABSTRACT

We apply reinforcement learning to the problem of finding good policies for a fighting agent in a commercial computer game. The learning agent is trained using the SARSA algorithm for on-policy learning of an action-value function represented by linear and neural network function approximators. We discuss the selection and construction of features, actions, and rewards as well as other design choices necessary to integrate the learning process into the game. The learning agent is trained against the built-in AI of the game with different rewards encouraging aggressive or defensive behaviour. We show that the learning agent finds interesting (and partly near optimal) policies in accordance with the reward functions provided. We also discuss the particular challenges arising in the application of reinforcement learning to the domain of computer games.

INTRODUCTION

Computer games constitute a very interesting domain for the application of machine learning techniques. Games can often be considered simula-

tions of (aspects of) reality (Salen and Zimmerman, 2004). As a consequence, modelling the behaviour of agents in computer games may capture aspects of behaviour in the real world. Also, the competitive and interactive nature of games allows the exploration of policies in a rich dynamic environment. In contrast to modelling behaviour in the real world, there are (at least theoretically) two great advantages enjoyed by a simulation/game approach: i.) full control of the game universe including full observability of the state ii.) reproducibility of experimental settings and results.

Computer games provide one of the rather few domains in which artificial intelligence (game AI) is currently applied in practice. That said, it is a common complaint of *gamers* that the game AI behaves either in boring ways or is too strong or too weak to provide interesting and entertaining game play. Hence, adaptive game AI has the potential of making games more interesting and ultimately more fun to play. This is particularly true since the sophistication of other areas of computer games such as sound, graphics, and physics have leapt ahead of AI in recent years and it is anticipated that advances in game AI will be a considerable driving force for the games market in the future. In fact, games such as *Creatures*, *Black and White* and *Virtua Fighter 4* were designed around the notion of adaptation and learning.

Machine learning can be applied in different computer game related scenarios (Rabin, 2002). *Supervised learning* can be used to perform *be-*

havioural cloning of the player, e.g., to represent him as an avatar at times when he is not available in person for a multi-player game (this includes mitigating latency in network-based games). However, the most appealing application of learning may be a non-player character (NPC) that adapts by *reinforcement learning* (Sutton and Barto, 1998) in response to the opponent’s behaviour and the environment during game play (see Stone and Sutton, 2001 for a RoboCup application), or even an agent who learns by playing against a clone of himself (see Tesauro, 1995 for a Backgammon application). Alternatively, such an adapting NPC may be useful at development time to create built-in AI adapted to varying conditions, or even to systematically test built-in AI for exploitable weaknesses.

In this paper we apply the SARSA reinforcement learning algorithm (Rummery and Niranjan, 1994; Sutton and Barto, 1998) together with a linear action-value function approximator to *Tao Feng*, a state-of-the-art commercial fighting game released for the Xbox game platform in 2003. Fighting games constitute a classical genre of computer games in which two opponents carry out a martial arts fight. *Tao Feng* provides about 12 different fighting characters with varying styles, combat taking place in 10 different arenas. There are over 100 different actions (moves and combo attacks) available to the player. The game comprises 350 000 lines of code of which 64 000 constitute the built-in AI, which is implemented as a non-deterministic finite-state machine.

A particular focus of the paper will be a discussion of the challenges that had to be met in order to adapt standard reinforcement learning to a real-time computer game and integrate the learning process into such a complex code base. The paper is structured as follows: We give a brief introduction to reinforcement learning with an emphasis on learning the action-value function. Then, we describe and discuss the specific choices made for applying reinforcement learning to *Tao Feng*. Finally, we present and discuss experimental results.

REINFORCEMENT LEARNING AND THE ACTION-VALUE FUNCTION

Markov Decision Processes

We model the agent’s decision and learning process in the framework of reinforcement learning (Sutton and Barto, 1998) which aims at finding an (near) optimal policy for an agent acting in a Markov decision process (MDP). An MDP is characterised by a tuple $(\mathcal{S}, \mathcal{A}, T, \mathcal{R})$ with

1. A *state space* \mathcal{S} with states $s \in \mathcal{S}$. In the most straight-forward case \mathcal{S} is a finite set. In *Tao Feng* the state can be represented by nominal features such as physical situation of a player (*on the ground, in the air, knocked*) or spatial features (wall behind, wall to the right etc.) However, depending on the representation chosen, real-valued state features such as distance between players or state of the health bar are conceivable as well.
2. An *action space* \mathcal{A} with actions $a \in \mathcal{A}$. We will only consider the case of \mathcal{A} being a finite set. More precisely, we are dealing with action spaces $\mathcal{A}(s)$ that depend on the current state s . Typical actions in *Tao Feng* include punches, kicks, throws, blocks and combo moves.
3. An unknown stochastic *transition dynamics* $\mathcal{T}_{s,s'}^a : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ which gives the probability of a transition from state s to state s' if action $a \in \mathcal{A}(s)$ is taken,

$$\mathcal{T}_{s,s'}^a := \mathbf{P}_{s_{t+1}|s_t=s,a_t=a}(s') . \quad (1)$$

In *Tao Feng*, the dynamics is given by the (partially stochastic) state machine that drives the game. In particular, the dynamics derives from the combination of the game engine and the built-in AI of the game.

4. An *average reward function* $\mathcal{R}_{s,s'}^a : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ which assigns a reward to the agent if it carries out action $a \in \mathcal{A}(s)$ in state s and ends up in state s' ,

$$\mathcal{R}_{s,s'}^a := \mathbf{E}_{r_{t+1}|s_t=s, a_t=a, s_{t+1}=s'} [r]. \quad (2)$$

The reward is a key element for learning, because it provides the feedback signal on which the learning agent can improve. In Tao Feng the reward is typically tied to the health-bar, the traditional goal of the game being to decrease the opponent's health-bar while maintaining one's own.

As seen above, a Markov decision process models some relevant aspects of the fighting agent's situation. However, one must keep in mind that two important aspects are neglected in this model:

1. The state space used in practice provides only an approximation to the true and complete state, parts of which remain unobservable (see, e.g., Kaelbling et al., 1998 for a discussion of partially observable MDPs). The missing state information includes details of the environment as well as hidden states in the opponent's built-in AI. However, the problem is partly overcome by the assumption of a stochastic transition dynamics which allows us to model the resulting uncertainty as a noise process.
2. The MDP model is ignorant of the adversarial aspects of fighting in that the behaviour of the opponent is simply captured by the transition dynamics \mathcal{T} given in (1). Depending on the nature of the game AI it might be more appropriate to consider game-theoretic models that take into account that there is more than one decision-making agent involved (see, e.g., Filar and Vrieze, 1996 on competitive MDPs)

Learning in Markov Decision Processes

The goal of the agent is to devise a sequence of actions $(a_t)_{t=0}^{\infty}$ so as to maximise his aver-

age expected reward. In order to make the agent autonomous in a given environment it must be equipped with a (stochastic) *policy* $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ which prescribes the probability of taking action $a \in \mathcal{A}(s)$ in state $s \in \mathcal{S}$, satisfying $\sum_{a \in \mathcal{A}(s)} \pi(s, a) = 1$ for all states $s \in \mathcal{S}$. A typical goal of reinforcement learning is to find a policy π that maximises the *discounted return*

$$R_t := r_{t+1} + \gamma r_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \quad (3)$$

where $0 \leq \gamma < 1$ is called the discount rate. The infinite sum R_t takes finite values for $\gamma < 1$ (convergence of geometric series), $\gamma = 0$ corresponding to a "myopic" agent and larger values of γ increasing the planning horizon.

We will focus on methods involving the state-action value function $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ for a given policy π defined as

$$\begin{aligned} Q^\pi(s, a) &:= \mathbf{E}_{\pi|s_t=s, a_t=a} [R_t] \\ &= \mathbf{E}_{\pi|s_t=s, a_t=a} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \right]. \end{aligned} \quad (4)$$

Its value indicates how beneficial (in terms of future expected discounted reward) it is for the agent to take action $a \in \mathcal{A}(s)$ when in state s . We prefer the state-action value function $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ to the state-value function $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$, because Q^π immediately provides a policy without requiring a model of the dynamics \mathcal{T} .

The "optimal" way of using Q^π for generating a policy is by choosing the action a^* with the highest associated Q^π value in a given state s . However, this choice is optimal only with respect to exploiting current knowledge. In order to be successful in the long run we need to balance *exploration* and *exploitation*. We use the *soft-max* or *Gibbs policy*

$$\pi(s, a) := \frac{\exp(\beta Q(s, a))}{\sum_{a' \in \mathcal{A}(s)} \exp(\beta Q(s, a'))}, \quad (5)$$

where the "temperature" parameter $\beta \geq 0$ determines how peaked the probability distribution is around a^* .

SARSA and Q-Learning

The SARSA algorithm (Rummery and Niranjan, 1994) is an on-policy temporal difference learning algorithm for Markov decision processes (Sutton and Barto, 1998). It is based on the following update equation

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha [r + \gamma Q(s', a')],$$

where $0 \leq \alpha \leq 1$ is a learning rate parameter and $0 \leq \gamma < 1$ is the discount factor for future rewards. The update equation states that for a given state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ the new state-action value is obtained by adding a small (depending on α) correction to the old value. The correction is the difference between the immediate reward r increased by the discounted future state-action value $\gamma Q(s', a')$ and the old state-action value $Q(s, a)$. SARSA (s, a, r, s', a') is an on-policy learning algorithm in the sense that it estimates the value of the same policy that it is using for control. Q-Learning (Watkins and Dayan, 1992) constitutes an off-policy alternative to SARSA and replaces the term $\gamma Q(s', a')$ by $\gamma \max_{a' \in \mathcal{A}(s')} Q(s', a')$ in the above equation. This allows for separating the policy being evaluated from the policy used for control.

The update equation as given assumes a tabular representation of the Q function. In practice, even for small problems such a representation is unsuitable for learning because an unrealistic amount of data would be necessary to estimate all the independent table entries reliably. As a consequence we decided to represent the state-action value function $Q(s, a)$ with different function approximators (see Sutton and Barto, 1998).

REINFORCEMENT LEARNING IN TAO FENG

The goal of our project was to develop a learning fighter that starts at a level of ignorance comparable to a human beginner, plays Tao Feng either

Algorithm 1 SARSA with linear function approximator and game embedding

Require: Learning rate $0 < \alpha$, SoftMax parameter $\beta > 0$ and Discount rate $\gamma < 1$

Require: functions `getValidActions()`, `getStateVector()`, `getReward()`

Require: functions `submitAction(a)`, `getExecutedAction(a')`

Require: function `selectSoftMaxAction(s', A, {wa}, β)` (see Equation (5))

Initialise $\forall \tilde{a} : \mathbf{w}_{\tilde{a}} \leftarrow \mathbf{0}$ and set $a \leftarrow$ undefined,

for every frame in turn **do**

$s' \leftarrow$ getStateVector()

$A \leftarrow$ getValidActions()

$a' \leftarrow$ selectSoftMaxAction($s', A, \{\mathbf{w}_a\}, \beta$)

 submitAction(a')

if getExecutedAction(a') $\neq a'$ **then**

Continue

end if

if $a \neq$ undefined **then**

$\mathbf{w}_a \leftarrow \mathbf{w}_a + \alpha (r + \gamma \mathbf{w}_{a'}^T s' - \mathbf{w}_a^T \mathbf{s}) \mathbf{s}$

$r \leftarrow$ getReward(\mathbf{s}, s')

$\mathbf{s} \leftarrow s', a \leftarrow a'$

end if

end for

against the built-in AI or against a human player, and develops fighting skills corresponding to the reward function provided.

Choice of Learning Algorithm

The choice of the learning algorithm was mostly determined by the design of Tao Feng. Although we were in possession of the full code base of the game, in practice, our ability to control the player as well as to observe the environment was severely limited by the structure of the code and the concurrency of processes. Hence, at any point in time we know neither the exact state nor the exact transition dynamics (1) and reward function (2) for Tao Feng. The application of methods based on the

state value function would require us to learn a separate model of the Tao Feng dynamics, thus introducing a layer of complication.

Although we originally intended to apply Q-Learning (Watkins and Dayan, 1992) it turned out to be very difficult to reliably determine the set $A(s)$ of available actions for the evaluation of $\gamma \max_{a' \in A(s')} Q(s', a')$. The reason for this complication lies in the graphical animation system of Tao Feng, which rejects certain actions depending on the animation state. As a consequence, it is only possible to submit a given action a (using `submitAction(a)`) and to check (using `getExecutedAction(a)`) which action has actually been performed. We chose the SARSA algorithm (Rummery and Niranjan, 1994) because it does not require knowledge of $A(s)$.

Choice of Features, Actions, and Rewards

Features There are essentially three groups of useful features, environment-related, opponent-related and agent-related features. Environment-related features such as “blocked behind” are designed to facilitate navigation of the arena. Opponent-related features such as “opponent’s physical state” and “distance to opponent” are designed to make the agent react to the opponent. Finally, features related to the learning fighters themselves such as “my previous action” and “my physical state” may serve to give the agent’s actions continuity and consistency.

Actions While the game provides over 100 different actions, we focused on a number of atomic actions such as simple punches, kicks and throws. In addition, we constructed meta-actions that are composed of repeated atomic actions such as `block`, `stepleft`, `stepright` and `lungeback`. This was necessary because some of these actions have a very short duration and their execution in isolation has almost no effect. As an example, we constructed a meta action

`block25` that holds up the block for 25 frames corresponding to half a second.

Rewards In order to assess rewards it is necessary to define the end of a round. Since in Tao Feng the two opponents do not act in sync (multi-threading) there is no clear definition of a round. We assign reward to the agent only when the subsequent action has been successfully selected thus indicating completion of the previous one. As a consequence, rounds have different durations Δt (measured in seconds) over which the reward-per-action r is spread out. We take this into account by considering the rate of reward $r/\Delta t$ in the learning.

Implementation Issues

The integration of the learning algorithms into the code base was hindered by the complex multi-threaded and animation centred architecture of the system. Systematic monitoring of the learning process was only possible because we devised an on-line monitoring tool that continuously sent data such as rewards, actions and parameters from the Xbox via the network to a PC, where the data was analysed and visualised in Matlab. Although the implementation as such is not planned to be productised, it served as a test-bed for a library of reusable modules including function approximators (look-up tables, linear function approximation, and neural network) and learning algorithms (Q-Learning and SARSA), which are suitable for use in future Xbox games.

EXPERIMENTS

We performed experiments in order to see if we could learn a good policy for fighting against the built-in AI. We employed the SARSA learning algorithm with function approximation as detailed in Algorithm 1. Throughout we used the parameter settings $\alpha = 0.01$ and $\gamma = 0.8$. We used two types of reward functions depending on the change

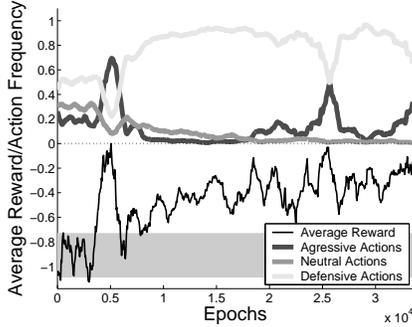


Figure 1: Average rewards and frequency of selected action classes for a Gibbs policy with $\beta = 2$ in the aggressive setting using *linear* function approximators. The gray band indicates performance of a uniformly random policy (mean ± 2 standard deviations)

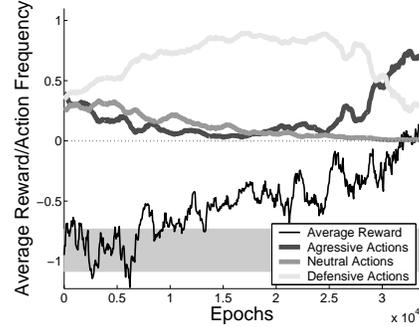


Figure 2: Average rewards and frequency of selected action classes for a Gibbs policy with $\beta = 1$ in the aggressive setting using *linear* function approximators. The gray band indicates performance of a uniformly random policy (mean ± 2 standard deviations)

in health of both combatants, ΔH_{self} and ΔH_{opp} . In order to encourage an aggressive fighting style we defined $r_{\text{aggressive}} := 0.7 \times \Delta H_{\text{opp}} - 0.3 \times \Delta H_{\text{self}}$. In contrast, we also defined a peace-encouraging reward function $r_{\text{aikido}} := -0.5 \times (\Delta H_{\text{opp}} + \Delta H_{\text{self}})$.

In order to represent the game state s the learning agent used the following 15 features, which were grouped into a feature vector $\mathbf{s} \in \mathbb{R}^3 \times \{0, 1\}^{12}$ suitable for input to the function approximators used:

- Distance to opponent coded in terms of three real-valued features based on unit-variance Gaussians placed at 1, 3 and 5 meters,
- 4 binary features coding which of the four sides of the agents are blocked,
- 6 binary features coding the previous action of the opponent,
- 2 binary features coding the physical situation of the opponent player (in air and knocked).

The learning agent was equipped with three classes of actions:

- Aggressive: throw, kicktrail, kicklead, punchlead, punchtrail.
- Defensive: block10, block25, block50, stepleft, stepright, lungeback.
- Neutral: getup, run10, run25, run50, crouch10, crouch25, crouch50.

In a first set of experiments we consider the reward function $r_{\text{aggressive}}$ and use linear function approximators for the Q -function. The results for $\beta = 2$ and $\beta = 1$ are shown in Figure 1 and 2. In both cases, the reward rate increases with considerable fluctuations by 0.8 and 1.1, respectively. Starting from the random policy, which loses approximately one reward unit per second, the learning agent achieves a reduction of the loss to -0.2 sec^{-1} for $\beta = 2$ and even a net gain of reward of 0.1 sec^{-1} for $\beta = 1$. In the case $\beta = 2$ the average reward stagnates at a sub-optimal level presumably being stuck in a local optimum. From the action frequencies it can be seen that despite the aggressive reward function, the agent prefers defensive actions and lacks the

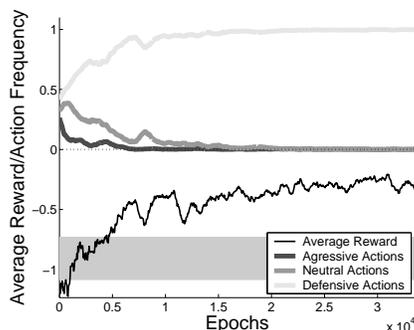


Figure 3: Average rewards and frequency of selected action classes for a Gibbs policy with $\beta = 2$ in the aggressive setting using a *neural network* function approximator with 3 hidden units

degree of exploration necessary for discovering the aggressive moves required for winning. In the more explorative setting of $\beta = 1$, the agent learns more slowly due to increased exploration but eventually discovers the usefulness of aggressive actions around episode 30 000 which results in a winning policy.

In a second set of experiments we replaced the linear function approximator with a feed-forward neural network with 3 hidden units. The results are shown in Figure 3. The learning agent finds a policy similar in performance to the linear function approximator at the same SoftMax parameter $\beta = 2$. Interestingly, the learning curve is smoother and the action selection appears to be less explorative. However, we did not fully explore the parameter space in order to avoid such local optimal.

In a third set of experiments, the reward function $r_{\text{aggressive}}$ was replaced with r_{aikido} and we returned to linear function approximation. As can be seen from Figure 4, the learning eventually results in an optimal policy (zero reward, i.e., no punishment). Note that we only depicted the *class* of actions selected by the learning agent. For example, in this particular case, a successful behaviour was achieved by an ingenious combination of side stepping and blocking.

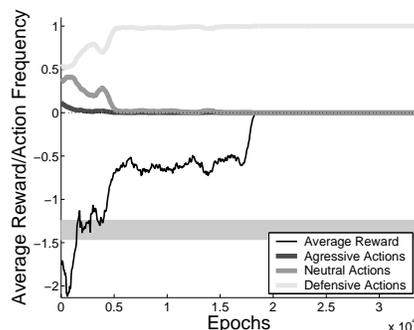


Figure 4: Average rewards and frequency of selected action classes for a Gibbs policy with $\beta = 2$ in the Aikido setting using a *linear* function approximator.

In summary, our experimental results show that good policies can be learnt for varying reward functions. Some of the resulting fighting agents displayed interesting behaviour, others exposed weaknesses of game engine and the built-in AI: They found simple repetitive patterns of actions that exploit gaps in the rule-based built-in AI, such as the optimal policy found in Aikido mode (see Figure 4). More exploration of the parameter space and the feature set as well as the incorporation of an eligibility trace in the SARSA update (Sutton and Barto, 1998) may further improve the policies found.

CONCLUSIONS

This work demonstrates that reinforcement learning can be applied successfully to the task of learning behaviour of agents in fighting games with the caveat that the implementation requires considerable insight into the mechanics of the game engine.

As mentioned earlier, our current approach neglects hidden state information and adversarial aspects of the game. One idea to tackle this problem is to separately model the game engine and the opponent. Based on these two models, standard planning approaches (e.g., min-max search,

beam search) can be employed that take into account that there is more than one decision maker in the game. Also an important aspect of human fighting game play involves timing which is hardly captured by the MDP model and requires explicit representation of time.

Watkins, C. J. C. H. and P. Dayan (1992). Q-learning. *Machine Learning* 8, 279–292.

ACKNOWLEDGEMENTS

We would like to thank JCAB, Glen Doren and Shannon Loftis for providing us with the code base and Mark Hatton for his initial involvement in the project.

References

- Filar, J. and K. Vrieze (1996). *Competitive Markov decision processes*. Berlin: Springer.
- Kaelbling, L. P., M. L. Littman, and A. R. Cassandra (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 99–134.
- Rabin, S. (2002). *AI Game Programming Wisdom*. Hingham, Massachusetts: Charles River Media, Inc.
- Rummery, G. and M. Niranjan (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University, Engineering Department.
- Salen, K. and E. Zimmerman (2004). *Rules of Play: Game Design Fundamentals*. Cambridge, Massachusetts: MIT Press.
- Stone, P. and R. S. Sutton (2001). Scaling reinforcement learning toward RoboCup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 537–544. Morgan Kaufmann, San Francisco, CA.
- Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press.
- Tesauro, G. J. (1995). Temporal difference learning and td-gammon. *Communications of the ACM* 38(3), 58–68.

Intelligent Agents

| | |
|---|-----|
| Anderson, E. F. A NPC behaviour definition system for use by programmers and designers | 203 |
| Broekens, J. and DeGroot, D. Scalable and flexible appraisal models for virtual agents | 208 |
| Hirokazu Notsu, Yoshihiro Okada and Koichi Nijima Component based approach for emotional behavior of 3D CG characters | 216 |
| Jacobi, D., Anderson, D., von Borries, V., Elmaghraby, A., Kantardzic, M., Ragade, R., Mehdi, Q. H. and Gough, N. E. Building intelligence in gaming and training simulations | 221 |
| Szarowicz, A. and Francik, J. Human motion for virtual people | 228 |
| Wen, Z., Mehdi, Q. H. and Gough, N. E. IAgent: A real time intelligent agent animation toolkit | 236 |
| Yannakakis, G. N and Hallam, J. Interactive opponents generate interesting games | 240 |
| Davies, N. P., Mehdi, Q. H., Gough, N. E. Anderson, D., Jacobi, D. and Bornes, V.V . A review of potential techniques for the creation of intelligent agents in virtual environments | 248 |

A NPC BEHAVIOUR DEFINITION SYSTEM FOR USE BY PROGRAMMERS AND DESIGNERS

Eike Falk Anderson
The National Centre For Computer Animation
Bournemouth University, Talbot Campus
Fern Barrow, Poole, Dorset BH12 5BB, UK
E-mail: eanderson@bournemouth.ac.uk

KEYWORDS

game-bots, behaviour definition, scripting language, virtual machine, mini-language.

ABSTRACT

In this paper we describe ZBL/0, a scripting system for defining NPC (Non Player Character) behaviour in FPS (First Person Shooter) games. ZBL/0 has been used to illustrate the use of scripting systems in computer games in general and the scripting of NPC behaviour in particular in the context of a book on game development [Zerbst et al 2003]. Many novice game designers have clear ideas about how the computer game they imagine should work but have little knowledge – if any – about how their ideas can be implemented. This is why books on game creation (design, programming etc.), as well as all-in-one game creation systems – especially designed for ease of use and intended for an amateur audience – enjoy great popularity. A large proportion of these books however merely present solutions in the form of descriptions and explanations of specific implementations with inadequate explanations of principles. While this may benefit rapid application development it often does not lead to a deeper understanding of the underlying concepts. The understanding of rule-based behaviour definition through simple scripting in computer games and the development of such scripts by programmers and designers is what we aim to address with the ZBL/0 system.

INTRODUCTION

Until very recently the major part of the artificially intelligent behaviour displayed by game characters in computer games was hard-coded into the game program itself. Any changes requested by the game's designers needed to be communicated to the game programmers who would spend a large amount of development time implementing these small changes to the game. A much more efficient approach which is now used more and more frequently is to empower designers to implement those changes themselves by making games more extensible and easily modifiable and by providing designers with the tools to extend and modify the games. As a side effect, developers have realized that this also enables players to modify a game themselves which adds value to a game and dramatically adds to its shelf-life (see Table 1). The question that now arises is how this extensibility can be achieved. This is especially important when it comes to the modification of the NPC (Non Player Character) behaviour in those extensible games. In some cases where no hard-coded solutions are used the NPC behaviour is generated by project-specific proprietary software tools, other games use commercially available middleware systems and some games use a scripting language of some sort. Scripting removes a large part of the – previously hard-coded – internal game logic from the game engine and transforms it into a game asset.

| Are there any truly <i>good</i> reasons to build an Extensible AI into your game? | |
|---|-----|
| Absolutely! | 40% |
| Sure! | 23% |
| Maybe. | 29% |
| No way! | 4% |
| Never! | 2% |
| Other. | 1% |

Table 1 – computer game extensibility reasons poll (source: <http://www.gameai.com>)

This allows the game to be modified without the need for the game code to be recompiled, a task that can be accomplished by a game designer alone. “Parallel development” becomes possible, which means that the programmers’ time is freed up as they no longer need to concern themselves with design elements which designers can now manipulate themselves with scripts [Huebner 1997]. However, a scripting language that is supposed to be used by non-programmers as well as by programmers needs to be designed accordingly. It is likely that for some game designers this will be the first programming language that they encounter so it is only logical that it should embrace some of the methods used in introductory programming languages.

THE RATIONALE BEHIND THE ZBL/0 SCRIPTING SYSTEM

The command syntax of the ZBL/0 language is similar to that of related procedural languages like C [Kernighan and Ritchie 1988], Pascal [Wirth and Jensen 1974] and especially PL/0 [Wirth 1977]. The ZBL/0 language only supports a limited set of control structures (simple iteration, condition/alternative and sequence) and the definition of simple procedures. In that respect, ZBL/0 can be counted in the family of mini-languages (toy languages – see Figure 1) used in teaching [Brusilovsky et al 1997] and in this role it has been used as a reference system to illustrate the development of NPCs [Zerbst et al 2003] for FPS (First Person Shooter) games (see Figure 2). Like other mini-languages ZBL/0 provides a task specific set of instructions and queries which allow users to take control of virtual entities acting within a micro world. In the case of ZBL/0 the scripting system and programming language were designed specifically with the definition of NPC behaviour in FPS games in mind which is reflected in the functions and procedures of the language. The use of computer games as the environment for a mini-language programmed NPC is not a new idea. There are several examples of games – most of which are available on-line

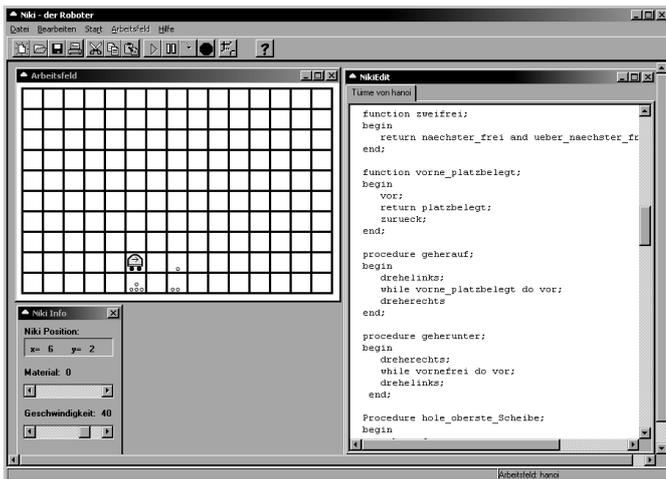


Figure 1 – Mini-language system “Niki the robot” which partially inspired the development of the ZBL/0 language (<http://www.hupfeld-software.de/niki.php>)

like Robocode [Li 2002], Crobots, Jrobots or GUN-TACTYX [Boselli 2004]. In such games the player interaction is limited to the programming of virtual entities that play the games. In addition to the use of ZBL/0 as an educational tool, the development of the ZBL/0 system is our first step towards the development of a generic behaviour definition system for artificially intelligent entities in computer games. We are using it to explore various system architectures for integrating virtual machines into applications – simple games and more complex game engines – that allow scripts to be executed and interpreted in real-time. We have also used the system as a test-bed for interfacing behaviour-definition systems with computer games.

SCRIPTING LANGUAGES FOR BEHAVIOUR DEFINITION IN GAMES

Many developers use well established existing generic scripting systems or permutations of these systems (modified according to the game’s requirements) to add scripting facilities to their games. A popular choice for building the scripting solutions in games is the scripting language Lua. Lua is a generic programming language which was originally designed to be used to extend programs by adding various scriptable features which is why the creators of Lua have dubbed it an “extensible extension language” [Jerusalemshy et al 1996]. Most of the other mature scripting languages which can be embedded in computer game engines are generic, i.e. not specialised for specific tasks [Varanese 2003]. A different approach which is also frequently used is to have proprietary purpose-built scripting languages that are dedicated to a single game, like the scripting languages QuakeC in Quake, UnrealScript in Unreal or Scrit in “Dungeon Siege”. When used to define game character behaviours, in simple cases the sole use of scripts is the initial configuration of the NPC behaviours. These initialization scripts [Tapper 2003] are the simplest form of scripts. During program runtime they are usually only executed once, at program start-up, while the application is initialising, setting internal program parameters to the values in the script. These scripts are often nothing more than lists of values, sometimes using additional syntactic elements to make them easier to read and edit. In more complex event based scripting systems, the occurrence of an event within the game triggers the execution of a script or part of a script. This means that scripts do not run in a pre-defined order but rather when a specific situation in the game-world has occurred. Some of these scripting systems use events that are built into the game engine as predefined events and scripts only define the event handlers and possibly additional conditions that may influence the

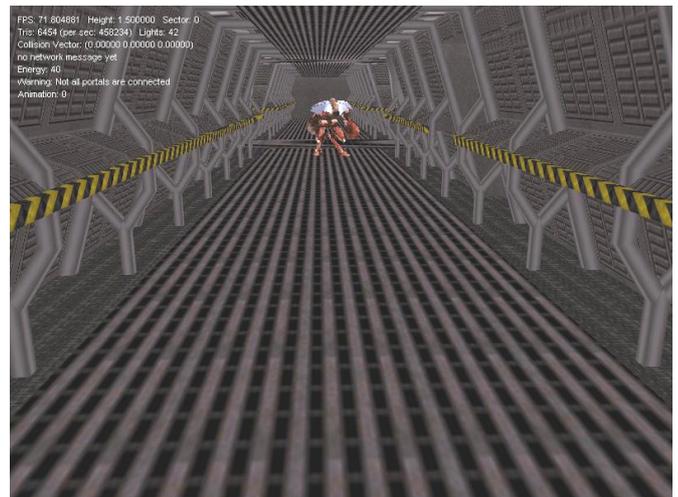


Figure 2 – sample game implementation “Pandora’s Legacy”

trigger mechanism. More sophisticated scripting systems first define the triggers and the situations in which they should act on events in addition to the event handlers themselves. These also include rule-based scripting systems which can be used for the definition of domain knowledge in expert systems, an example of which are intelligent NPCs in many computer games. The most complex scripting solutions are programs that use high-level abstract descriptions to define complex behaviours. A scripting system that controls the behaviour of autonomous agents in a virtual game world usually exists on two levels. The higher level is a scripting language that is often modelled on “traditional” procedural, functional or object oriented programming languages. The lower level is the corresponding scripting engine which interfaces with the game. Some of these systems will execute scripts in a continuous loop, constantly (re-)evaluating the current situation within the game. Other systems will execute a script only once and any kind of repeating operation, to be executed by the scripting system, will have to be implemented as a looping operation within the script itself. An example of the latter is our ZBL/0 scripting system. Scripting engines of this kind can take the form of an interpreter which translates and executes scripts at runtime. Alternatively it could be a virtual machine, executing scripts that have previously been translated into an intermediate code by a compiler. Both forms of scripting system provide the same benefits to games, as both allow the alteration of NPC behaviour by modifying a script program. This means that the game application itself does not have to be recompiled for the changes to the game’s NPCs to take effect.

Design Issues

While this is clearly advantageous for game development, for computer game developers to truly benefit from any kind of scripting system it has to be designed to be intuitive, i.e. the scripting language must be easy to learn and possibly easier to use than traditional programming languages. One way this could be achieved would be by making it as similar to a natural language as possible. It is our belief that a close resemblance of a behaviour definition programming language to natural language as suggested by Funge [Funge 1998] may easily prove counterproductive. This is because natural languages are context sensitive and contain too many ambiguities which require additional specification to clarify problems and to resolve these ambiguities. We think that the additional effort required to do this would negate all the benefits gained from the use of a natural language structure in the first place. Moreover, linking a programming language’s structure intrinsically to a specific natural language would make it much

more difficult for non-native speakers of the natural language to write meaningful computer programs, while it would become practically impossible for programmers who do not know the natural language to write programs at all. Providing multi-language versions of a programming language is unrealistic, as the language would have to be modified according to the structure of each of the supported natural languages. We are also convinced that the notion that a traditional programming language may be too complex for non-programmers to use is incorrect. Robert Huebner's [Huebner 1997] case study of how scripting support was implemented in the game "Jedi Knight: Dark Forces" describes the C based proprietary scripting language COG. The similarity of COG to the programming language C not only simplified the development of the language but it also made it easier to learn and understand for the designers – non-programmers – who used COG for the creation of the game. He concludes that the design was so successful that designers managed to generate scenarios which would have appeared inconceivable and very hard to realize if it had not been for the COG scripting system. Further evidence for this can be found in the film effects industry where many artists have been using complex scripting systems for many years. For many designers the use of a scripting system will be the first time they are exposed to a programming language. An important consideration in the design of a programming language for novice programmers therefore is the analysis of how many of these non-programmers will go about using this language. Poiker [Poiker 2002] explains how novice programmers write programs employing a mixture of "copy and paste" with "trial and error". For this reason, programming languages that are supposed to be used by novice programmers need to have a WYSIWYG (What You See Is What You Get) character with program source code being able to deliver predictable results. In the context of the novice programmer's introductory programming language, McIver and Conway [McIver and Conway 1996] have identified seven "deadly sins" and design principles and their potential problems and benefits. They argue that a language which has too many different features ("more is more") or too few features ("less is more") or which contains too many syntactical "false friends" ("grammatical traps", "violation of expectation", "excessive cleverness") would make it very hard for users with little programming experience to comprehend the language and to understand what a program does. However, McIver and Conway conclude that the ideas they present can only be taken as a guide – not a general solution – and that ultimately the success of the language design can only be measured through user feedback.

THE ZBL/0 SYSTEM

The requirements for the ZBL/0 scripting system were straightforward:

- The system was to be used to define NPC behaviour as an extension to computer games of the FPS genre.
- The NPCs defined by the language only needed to support deterministic behaviour.
- No complex datatypes or control structures needed to be implemented as the system was supposed to be used to demonstrate general concepts of NPC behaviour scripting in the context of a book on computer game development [Zerbst et al 2003].

Consequently the development of the system from conception to first use was achieved in a very short period of time. The first fully working prototype for the ZBL/0 system for example was completed over a period of little more than a fortnight. ZBL/0 [Anderson 2003] is a very simple scripting language for the definition of game-bots. The ZBL/0 system consists of a compiler for game-bot programs (NPCs) written in the ZBL/0 language and a robust virtual machine that can be integrated into any game engine.

| | | |
|---------------|---------------|-------------|
| const | do | else |
| function | if | return |
| then | var | while |
| alive | armour | back |
| backstep | blocked | crawl |
| danger | die | duck |
| face | find | fire |
| front | health | idle |
| initialize | jump | jump_back |
| jump_left | jump_right | |
| jump_up | left | memorize |
| object | object_ahead | |
| obstacle | owns | respawn |
| right | rnd | spawn |
| spawned | step | strafe_left |
| strafe_right | target | |
| target_ahead | target_alive | |
| target_armour | target_health | |
| turn | turn_left | turn_right |
| use | using | |

Table 2 – ZBL/0 keywords (instructions & intrinsic functions)

ZBL/0 is based on the PL/0 model programming language [Wirth 1977] and therefore belongs to the PASCAL family of programming languages. There is only one variable datatype in ZBL/0 which can be used to store numerical values (integer as well as floating point). The function set for controlling bots is intrinsic to the ZBL/0 scripting language, i.e. built into the language (see Table 2). As a result they do not have to be enabled by means of inclusion of a standard library of functions. This intrinsic function set consists of 45 functions representing actions and sensor queries that can be performed by an NPC in FPS games like turning towards an opponent, moving in a specified direction or firing a weapon. The function identifiers are self explanatory for easy understanding. The current version of the language allows functions to be user-defined, but function parameters in user-defined functions are not supported. Instead they have to be emulated through the use of global variables. The ZBL/0 system uses a parallel stack-based virtual machine – the system is multi-tasking and allows more than one ZBL/0 program to run simultaneously. Run-time errors in ZBL/0 programs result in the termination of the game-bot program but do not affect the execution of the virtual machine within its host application. The ZBL/0 virtual machine is self-contained and accessible from the host application solely through a fixed interface, the ZBL-API (Application Programmer Interface). The interface to the ZBL/0 virtual machine provides games with the ability to associate NPC functionality with in-game functions for actions which would be expected to be performed by a player of these games, therefore allowing NPCs to compete with human players on a level playing field. Once a ZBL/0 program has been loaded into the virtual machine only a single function-call to the API is required for each main program loop to execute the game-bot programs. The simplicity of the system lies in the fact that none of the game-bot functions are provided by the language as such. Instead they need to be implemented within the game engine – the host application – and mapped to the corresponding intrinsic function identifier in ZBL/0. The game engine itself does all the work while the script only ties together the different game engine components that provide the NPCs with functionality. A side effect to this is the ability of the system to be adapted to games of different genres (see Figure 3). The function bindings between the host application and the ZBL/0 virtual machine are realised using the multiple-inheritance functionality of the C++ programming language [Stroustrup 1997]. Objects of a game-bot class can be registered as NPCs with the virtual machine. This game-bot class is created by inheriting player functionality from a player-class in the application and the game-bot interface from an abstract class which is part of the ZBL-API. This abstract class provides a number of methods

```

# a moderately sophisticated CycleBot
var direction;
function random;
var r;
{
  r = rnd 20;
  if r > 5 & r < 15 then
    return 1;
  else
    return 0;
};
{
  spawn;
  direction = random;
  while alive = 1 do
  {
    if blocked front = 0 then
    {
      step;
    };
    else
    {
      if blocked left = 0 then
      {
        if blocked right = 0 then
        {
          if direction = 1 then
          {
            turn_left;
          };
          else
          {
            turn_right;
          };
          direction = random;
        };
        else
        {
          turn_left;
        };
      };
      else
      {
        if blocked right = 0 then
        {
          turn_right;
        };
      };
      step;
    };
  };
};
}

```

Table 3 – a simple ZBL/0 game-bot script for a Tron-like “lightcycle” game (see Figure 3)

that are equivalent to the intrinsic functions of the ZBL/0 language. An implementation of these abstract methods in the inherited class then allows ZBL/0 programs in the virtual machine to control a game-bot character in the application. However therein also lies the main weakness of the ZBL/0 system, as any NPC script – no matter how well designed – cannot perform well if the NPC related functions of the game engine do not work well. Also, while this method makes it very easy for the virtual machine to execute functions within the host application it also limits the extensibility of the ZBL/0 system, as the type and number of the functions that can be registered with the virtual machine is fixed by the ZBL-API. On the other hand, this system allows the designer to create effective NPCs through the combination of a small number of simple functions (see Table 3).

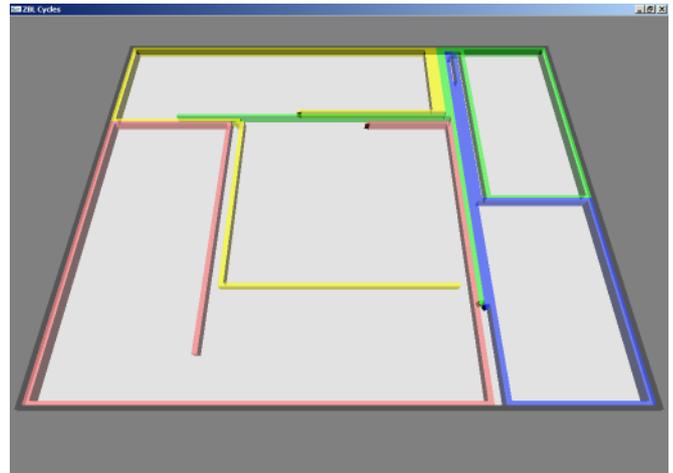


Figure 3 – four ZBL/0 “lightcycles” competing for survival in the demonstration application

A ZBL/0 Example Program

As an example for the capabilities of the ZBL/0 system as well as a sample to demonstrate the integration of the ZBL/0 virtual machine into a computer game we have created a version of the “lightcycle” racing game (see Figure 3) that featured in the 1982 film Tron (<http://www.imdb.com/title/tt0084827>). Players in the game move their “lightcycles” across the playing field, dragging growing walls of light energy behind them. The aim of the game is for players to survive as long as possible by avoiding collision with walls while at the same time trying to force other players to collide with walls by reducing their freedom to manoeuvre. In our version of the game all players are controlled by scripts written in the ZBL/0 language (see Table 3). The sensors of the scripted players allow them to test the path immediately in front of them for two steps ahead and one step to each side. The strategy they employ to play the game does not involve any planning but is only a small set of simple rules:

1. If the path in front of the “lightcycle” is not blocked, it moves forward (intrinsic function “step”).
2. Otherwise if there is no obstacle to the left and no obstacle to the right, the player chooses a random direction (determined by function random) or if there is an obstacle to the right, the player turns left (intrinsic function “turn_left”) and moves forward.
3. In the case of an obstacle to the player’s left but no obstacle to the right, the player turns right (intrinsic function “turn_right”) and moves forward.

The result of this script is an effective player that is perceived to be far more intelligent by onlookers than it actually is.

FUTURE WORK

The current version of ZBL/0 only provides the basic tools necessary for using the system with game engines that are programmed using the C++ programming language. As experience has shown that the addition of tools is beneficial for the process of program development, future work on the ZBL/0 system will mainly focus on the expansion of the toolkit. The creation of a graphical user interface – in addition to the command-line tools – to complement the language interface of the system by providing a text-editor incorporating a number of intuitive programming aids found in modern program development tools (syntax highlighting, code completion etc.) will be the first goal. Further goals will be optimizations of the compiler, the addition of support for run-time debugging as well as source-level debugging of ZBL/0 programs and possibly the provision of language bindings for other

programming languages than C++. We have used ZBL/0 to test possible architectures and interfaces for the virtual machine of a more generic behaviour definition system for artificially intelligent entities in computer games. This system that we propose is a modular and easily extendable system that will provide game developers with an intuitive method for the creation of game character AI, as well as the tools for doing so. A newer, experimental version of the ZBL/0 system which is still undergoing testing can be dynamically extended through a plug-in architecture which allows external libraries to be integrated with the system's virtual machine. This is an important feature which will also be implemented in our more generic behaviour definition system.

CONCLUSION

We have presented ZBL/0, a simple behaviour definition system for game characters in FPS games, designed to be used by game programmers as well as by game designers. ZBL/0 is much smaller, more restrictive and far less extensible than many other scripting systems – the language is dedicated to only one genre of computer games and the AI entities that populate them. Following the example of mini-languages the ZBL/0 language is based on a traditional programming language which has been reduced to the simplest features to make the system easily accessible for programmers and non-programmers alike. We strongly believe that ZBL/0 is easy to learn and master. For the past fifteen years, artists at the National Centre for Computer Animation have learnt to use scripting languages and have successfully used that knowledge for scripting procedural animation. This has convinced us beyond doubt that the use of scripting systems can be picked up by users with no prior knowledge of computer programming. The functionality of ZBL/0 is entirely dependent on the implementation of the host application, yet it shows how relatively simple methods can be used effectively for NPC creation in computer games (see Table 3). As an additional benefit this also allows the system to transcend its limitations by allowing it to be adapted to other game genres than only FPS games (see Figure 3). Some parts of the ZBL/0 system have shown weaknesses in the original design concept which we intend to address with our future work. For instance the lack of extensibility provided by the method in which function bindings are implemented in ZBL/0 has convinced us that a different approach will have to be used for our more generic behaviour definition system. For similar reasons we believe that the use of mainly intrinsic functions results in the main cause of inflexibility of the ZBL/0 system. An implementation using external libraries to provide the core language with functionality would have made the system much more extensible and flexible. The plug-in architecture that was implemented in the latest version of the ZBL/0 system will be able to deliver a partial solution to this problem. This feature of the ZBL/0 virtual machine will be used in a similar fashion in the creation of our more generic behaviour definition system.

ACKNOWLEDGEMENTS

I would like to thank the members of the ZFX team (at www.zfx.info) – especially Milo Spirig, Sebastian Pech and Oliver Düvel – for their valuable feedback and for program testing. Their suggestions and comments have been encouraging and very useful for the design of the ZBL/0 system. I also need to thank Stefan Zerbst for “mentioning” that a scripting extension to the game “Pandora’s Legacy” might be beneficial, prompting me to design ZBL/0 in the first place. Furthermore I need to extend my gratitude to everyone whose comments and suggestions contributed to this paper, especially Prof. Peter Comminos for inspiration, support and help in the preparation of this paper and Steffen Engel for encouragement.

REFERENCES

- Anderson, E.F. (2003). “ZBL/0 - the ZFX Bot Language”. ZFX - 3D Entertainment – <http://zbl0.zfx.info>
- Boselli, L. (2004). “GUN-TACTYX - Historical Background” – <http://gameprog.it/hosted/guntactyx/info.php#intro0>
- Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., and Miller, P. (1997). “Mini-languages: A Way to Learn Programming Principles”. *Education and Information Technologies* 2 (1), pp. 65-83.
- Funge, J.D. (1998). “Making Them Behave: Cognitive Models for Computer Animation”. PhD Thesis, University of Toronto
- Huebner, R. (1997). “Adding Languages to Game Engines”. *Game Developer*, Vol. 4(1997): nr 9
- Ierusalemshy, R., de Figueiredo, L. H. and Celes, W. (1996). “Lua—an Extensible Extension Language”. *Software : Practice & Experience*, Vol. 26(1996): nr 6, pages 635-652
- Kernighan, B.W. and Ritchie, D.M. (1988). “The C Programming Language”, Prentice-Hall
- Li, S. (2002). “Rock ’em, sock ’em Robocode!” IBM developerWorks: Java technology – <http://www-106.ibm.com/developerworks/library/j-robocode/>
- McIver, L. and Conway, D. (1996). “Seven Deadly Sins of Introductory Programming Language Design”. *Proceedings of Software Engineering: Education and Practice (SE:E&P’96)*, pages 309-316
- Poiker, F. (2002). “Creating Scripting Languages for Nonprogrammers”. *AI Game Programming Wisdom*, Charles River Media, pages 520-529
- Stroustrup, B. (1997). “The C++ Programming Language”, 3rd Edition. Addison Wesley
- Tapper, P. (2003). “Personality Parameters: Flexibly and Extensibly Providing a Variety of AI Opponents’ Behaviors”. *Gamasutra* – <http://www.gamasutra.com>
- Varanese, A. (2003). “Game Scripting Mastery”. Premier Press
- Wirth, N. and Jensen, K. (1974). “PASCAL - User Manual and Report”, Springer-Verlag
- Wirth, N. (1977). “Compilerbau”, Teubner
- Zerbst, S., Düvel, O. and Anderson, E. (2003). “3D-Spieleprogrammierung”. Markt + Technik

SCALABLE AND FLEXIBLE APPRAISAL MODELS FOR VIRTUAL AGENTS

Joost Broekens and Doug DeGroot
Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
2333 CA Leiden
Netherlands
E-mail: {broekens; degroot}@liacs.nl

KEYWORDS

Computational Models of Emotion, Appraisal Theory, Scalability, Dynamic Models, Games and Virtual Agents

ABSTRACT

Computational models of emotion are useful in a variety of domains, including games, virtual reality training and HCI to name a few. Many of these models are inspired by appraisal theory. Most appraisal theories share with virtual agents the assumption that beliefs, desires and intentions are the basis of reasoning and thus of the emotional evaluation of the agent's situation. Consequently most computational models of emotion are deeply embedded into the agent model. In this paper we address the problem of how to emotionally instrument a system in a modular and extensible way, so that emotional sophistication can be added incrementally to a system. We propose a solution based on a modular, signal-based approach to computational emotions that allows us to develop scalable appraisal models that are easily added to non-emotional systems. Our approach allows runtime trade-off between emotional quality and performance, which makes it particularly useful in domains in which available computation time is unknown, like the gaming domain. We present experimental results that back-up our approach.

INTRODUCTION

In psychology emotion is often defined as a psychological state or process that functions in the management of goals and needs of an individual. This state consists of physiological changes, feelings, expressive behaviour and inclinations to act. Emotion is elicited by the evaluation of an event as positive or negative for the accomplishment of the agent's goals. Thus, according to this view an emotion is a heuristic that relates the events from the environment to the agent's goals and needs (Oatley 1999). Additionally, emotion is a communication medium.

Computational models of emotion are embedded in agents in a variety of domains including HCI and electronic tutors, non-player characters (NPCs) in games, virtual-reality safety training environments and decision-making and planning. Emotions are embedded in virtual agents primarily to create an enhanced sense of realism, using emotional expression and the interplay between emotions and plans (Marsella and Gratch 2001). It has been argued that emotions and emotion-like phenomena are a good way of enhancing realism and thereby entertainment value of NPCs in games (Baillie-de Byl 2003, Mac Namee and Cunningham 2003).

The majority of computational models of emotion embedded in virtual agents is inspired by appraisal theories,

cognitive theories of emotion that attempt to explain why a certain event results in one emotional response rather than another and why a certain emotion can be elicited by different events. The key concept of most appraisal theories is that the subjective cognitive evaluation of events in relation to the agent's goals and needs is responsible for emotion (Roseman and Smith 2001). More generically one can say that events have to be evaluated as having personal meaning (van Reekum 2000). This evaluation is called appraisal. Most appraisal theories assume that appraisal is a necessary and sufficient condition for emotion (Roseman and Smith 2001).

Agents often use a belief-desire-intention (BDI) based architecture (Jennings et al. 1998). If cognitive evaluation of events in relation to the agent's goals and needs is sufficient for emotion then the addition of a subjective evaluation of events related to the beliefs, desires and intentions of an agent is sufficient for computational emotions. This explains the current popularity of appraisal theories in emotional agents.

Computational models of emotion must often be deeply integrated with the agent's non-emotional components because they depend on the BDI architecture of the virtual agent, as mentioned above. This deep integration has two problems. First, it takes quite some effort to add emotions to a non-emotional agent, because the computational model of emotion needs to be embedded into the agent's architecture. Second, computational models of emotion are difficult to adapt and upgrade in an incremental fashion for the same reason. Both problems are important for game development. From a marketing and sales point of view one wants to be able to incrementally add emotional sophistication to NPCs to sell upgrades of a game. From a technical point of view one wants to be able to evaluate which version of a computational emotional model to use based on e.g. performance, quality and stability.

We have investigated these problems and propose the FeelMe framework for computational emotions that is inspired by appraisal theory *and* allows the development of scalable appraisal models. The ability of the FeelMe framework to dynamically integrate the results of different emotional instrumentations that run simultaneously enables the development of scalable appraisal models. A scalable appraisal model can be used to emotionally instrument an agent (virtual agents, NPCs) in an incremental manner. A scalable appraisal model also allows runtime trade-off between emotional quality and performance. This trade-off ability makes these models particularly useful in domains in which computation power is an unknown factor, like the gaming domain.

In the next section we explain scalability of computational models of emotion and incremental instrumentation in more detail. Then we describe the FeelMe framework. We continue with a detailed description of how the FeelMe framework can be used to integrate different emotional instrumentations and what constraints exist for these instrumentations. Finally we present a proof-of-concept experiment with a game agent that uses our approach. The experiment shows that scalable appraisal models are possible, pointing out that the FeelMe framework enables game-character and virtual agents designers and developers to incrementally add more and more sophisticated emotions to their virtual agents.

SCALABILITY AND GAMES

Any system for use in games must be efficient in terms of computation required (Mac Namee and Cunningham 2003). Additionally, games must run on different platforms so computation power is not a known factor. Users with a high-end PC want to have high-end effects so the minimum system requirements to run a game cannot be taken as development standard. A good solution to this dilemma is to use scalable systems that are able to trade-off quality versus performance. This trade-off is often seen in 3D-graphic engines and chess-engines, in which level of respectively graphical detail and intelligence can be dynamically traded-off for respectively frame-rate and total game-time.

The systems that make up a game engine are usually triggered at regular intervals. Flexibility regarding the frequency of this triggering enables a different form of scaling, namely scaling based on a trade-off between temporal quality and performance. For example a 3D-graphic-engine can be triggered 10 times per second, in which case the frame-rate is low and the frames are staggering. However, individual rendered frames are still consistent and the sequence of frames still consistently shows the motion of objects and agents in the game, although less detailed. A computational emotional model able to do the same, that is, scalable quality and flexible triggering, thus has a practical advantage compared to one that can't. We refer to these two kinds of scalability as *runtime-scalability*.

INCREMENTAL INSTRUMENTATION

We define incremental emotional instrumentation of systems as a development process based on the step-by-step addition of complexity to a computational model of emotion resulting in meaningful and more sophisticated emotions of the agent that is consistent with the emotions of the simpler versions of the model. Consistent in our case means that the more complex version behaves equally meaningful as or more meaningful than the simpler version. By meaningful we mean that a human observer (e.g. the gamer) can - potentially in retrospect - understand why the agent exhibits a certain emotion or chooses to act in a certain way. In other words, more complex models should add sophistication to the emotions of the agent as well as add human understanding of the agent's emotion and related actions.

A possible first step to instrument a non-emotional system is by using event encoding. Based on common-sense, an

event is given a specific emotional property, just like emotionally laden words in language already have. When an event is encountered by an agent, the agent's emotion is changed accordingly. For example, a Quake bot seeing his team-mate die could be configured to experience sadness by defining a high "sadness" property for the 'team-mate-died' event. It could also be configured to experience anger. Actually, many different emotions would make sense and are not depending on the event but more so on the evaluation of that event. Although efficient and sufficient in some situations, this way of directly encoding emotions into properties of events does not work in general and is not psychologically plausible¹. According to appraisal theory, events are interpreted by the agent, after which the emotion is influenced. This interpretation includes reasoning about what the event means to the goals and needs of the agent, which is depending not only on the event but also on the current BDI state of the agent. So, an event does not *directly* influence the emotion - at least not in general - but the interpretation of the event does.

A BDI based approach to computational emotion evaluates events in the context of the current goal hierarchy of the agent, and determines the resulting emotion based on this evaluation. Switching from an event-encoding approach to a BDI-based approach is necessary for more meaningful computational emotions. However, some situations might be much easier to give emotional meaning using event encoding instead of using BDI-based appraisal. Also, when computing time becomes a bottleneck, an agent might need to switch to a simpler emotional instrumentation to save computation time for other sub-systems of the game.

Event-encoding and BDI based appraisal are two possible ways to emotionally instrument - or extend an emotional instrumentation of - a system. The relevant question for incremental emotional instrumentation of virtual agents is thus how to integrate the results of different concurrent emotional instrumentations? The ability to emotionally instrument a system in an incremental manner is referred to as *model-scalability*.

FEELME: A DYNAMIC APPROACH TO COMPUTATIONAL EMOTIONS

The FeelMe framework (DeGroot 2004) is a modular approach to computational emotions and is based on a strict separation of the computational emotional process in five main steps (see Figure 1). These steps are described in more detail in this section. This framework for computational emotions has been developed to study the effects of emotion on decision-making by using emotions as first-order objects in reasoning (DeGroot and Broekens 2003).

- The Decision Support System (DSS) provides mediated access to the existing system (e.g. an existing arcade game). Since some information in the environment of an agent or in its own internal state is not directly suitable for appraisal, the DSS translates this information before sending it to the Appraisal System. The DSS constructs to-be-appraised objects, based on the events occurring in

¹ In some domains psychological plausibility of emotions is of high importance, like virtual reality safety training.

environment of the agent and sends these objects to the Appraisal System.

- The Appraisal System (AS) continuously emotionally evaluates the constructed objects and interprets these in terms of values on a set of subjective measures, called appraisal dimensions. An appraisal dimension is a variable - e.g. arousal or valence - used to express the result of the emotional evaluation of a perceived object, for example a friend. The evaluation of the AS results in a continuous stream of n -dimensional vectors representing the *appraisal-results*, with n equal to the number of appraisal-dimensions. These vectors are sent to the Emotion Maintenance System. The number and type of appraisal-dimensions is configurable and need not be defined here. Just for the purpose of consistent terminology, in this paper we call any mechanism that produces appraisal-results an *appraisal mechanism*. Event-encoding can thus be called an appraisal mechanism, provided that it produces appraisal-results as defined above.
- The Appraisal Signal Modulator (ASM) can perform signal pre-processing on the incoming appraisal-results - like amplification of, dampening of and correlating certain appraisal dimension values - before these are sent to the EMS.
- The Emotion Maintenance System (EMS) continuously integrates the appraisal-results and maintains the agent's emotional-state. The emotional-state is also an n -dimensional vector. Appraisal-results induce changes to the emotional-state, thus for the EMS an appraisal-result is an n -dimensional vector of deltas of appraisal dimensions. This integration of deltas is what we refer to as a *signal-based* approach. The emotional-state of an agent can thus be understood as a continuously moving point in an n -dimensional space of appraisal dimensions. In this paper we use *emotional-state* when we refer to the vector that is maintained by the EMS. An emotional-state actually is not just a computer science approach to emotions. Many emotion theorists use this concept of a state to define emotion (Mehrabian 1980, Russell 2003, Reisenzein 2001, Scherer 2001).
- The Behaviour Modification System (BMS) selects, controls, and expresses the agent's emotional behaviour. The behavioural choices are based on the agent's emotional-state and additional knowledge the agent has.

mechanism that produces these vectors can be used. The Appraisal System can thus consist of multiple subsystems, provided that these produce the same kind of vectors. This opens up the possibility of adding more and more subsystems to add more and more detail to the virtual agent's emotions. The EMS integrates the appraisal-results. The most simple version of the EMS continuously adds-up all vectors sent to it by the AS (see formula 1 on page 6). Even such a simple paradigm allows integration of the results of different appraisal mechanisms into a meaningful representation of the emotional-state, provided that several guidelines for modular appraisal are used, as we will show. Also, this signal-based integration of appraisal-results from different concurrent mechanisms is highly compatible with the concept of appraisal integration by appraisal detectors as proposed by the appraisal theorists Smith and Kirby (2000).

MODULAR AND SCALABLE APPRAISAL

In order to build scalable appraisal models we have defined *appraisal banks*. In this section we explain what an appraisal bank is, why it facilitates the development of scalable appraisal models and what kind of appraisal-results are needed for effective integration of these result.

Context Sensitive Appraisal Banks

The Appraisal System (AS) is the complete appraisal system of an agent and an appraisal bank is a sub-systems of the AS (see Figure 1). An appraisal bank is an object (in the OO sense) that contains a set of functions that emotionally evaluate specific aspects of the agent's environment and internal state, for example all events related to survival. An appraisal bank is context sensitive, that is, the contribution of the bank's appraisal-result - as determined by the bank's evaluation functions - to the emotional-state of the agent depends on the situation of the agent. An appraisal bank can influence the contribution of another appraisal bank's appraisal-results through dependencies. Such dependencies allow the definition of causal connections which enable modelling of levels of appraisal and evaluation sequence (Scherer 2001, van Reekum 2000). To facilitate development of scalable appraisal models we enforce strict modularity of the AS by assuming that appraisal banks evaluate independent of each other.

We now explain why context sensitivity and evaluation sequence of appraisal banks facilitate the development of both model- and runtime-scalable appraisal models. First, context sensitivity facilitates the development of new - more elaborate - banks on top of older - more generic - banks. These new banks - for example based on BDI-based appraisal - can be sensitive to contexts where more meaningful emotions are needed but not achieved with the older banks - based on for example event encoding. The older banks can be sensitive to those contexts in which they work well. Context sensitivity of a bank can be configured by a game-character designer. Context sensitivity can also be implied based on the activity of appraisal banks (e.g. larger appraisal-results are more important than smaller ones). In this case the contribution to the emotional-state of one bank inhibits the contribution of another bank. Dependencies

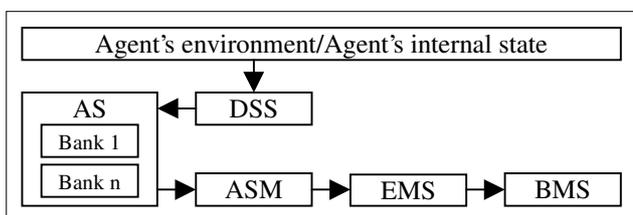


Figure 1. Overview of the components of the FeelMe framework relevant to this paper. Banks are explained later

Now, how does this help us integrate the appraisal-results of different concurrent appraisal mechanisms of an agent? The introduction of the emotional-state and its interaction with the Appraisal System are key. The AS outputs vectors that represent changes to the emotional-state. Any

determine which bank influences which. These dependencies can be used to build an interconnected set of appraisal banks that influence each other. Context sensitivity facilitates model-scalability because banks evaluate situations they recognise while other banks are silent.

Second, simple - computationally cheap - appraisal banks, and elaborate - computationally expensive - appraisal banks can be active simultaneously. An agent can dynamically adapt its appraisal effort (and thus computation time needed) by switching between simple and elaborate banks, depending on the context and the maximum amount of computing time available for its computational emotion system. Alternatively the user can adapt the emotional detail of the NPCs by configuring his game. A third way of adapting appraisal effort can be based on the distance between the NPC and the user's viewpoint. All three adaptation examples show the runtime-scalability potential of appraisal banks. Interestingly, the dynamic adaptation of appraisal effort depending on the situation and available resources is consistent with certain appraisal theoretic approaches towards emotion (Scherer 2001).

The set of functions in an appraisal bank can be designed to produce meaningful appraisal-results. A second bank can be completely separated from the first and also produce meaningful appraisal-results. If both banks work well together at the same time, they can be active at the same time. If they don't then one bank has to inhibit the other, or both banks have to be configured to be sensitive to mutually exclusive contexts (see below). Grouping appraisal in context-sensitive banks facilitates debugging of the appraisal model, since designers can focus on specific banks and assume other banks are deactivated. For example, in a typical RPG scenario this allows the development and debugging of an NPC's appraisal banks for battle, travel/quest and village/city situations.

In this paper we analyse the results of an experiment testing the difference between one bank, and two dependent banks in which the first inhibits the second.

Constraints for appraisal banks

If we assume that the Appraisal Signal Modulator (ASM) does not pre-process appraisal-results and the Emotion Maintenance System (EMS) only integrates appraisal-results by addition, what kind of appraisal-results do appraisal banks need to output for successful integration? Multiple constraints exist, of which we describe three.

First, appraisal-results need to be defined on the interval scale, addition of appraisal-results must be meaningful. To be more concrete, a 0.5 on the Pleasure dimension produced by bank 1 must emotionally mean the same as a 0.5 on the Pleasure dimension produced by bank 2. Furthermore, in order for the EMS to meaningfully integrate the values by adding up, a 0.5 increase or decrease on a certain dimension must always mean the same for that dimension. These two criteria do *not* have to hold *between* appraisal dimensions, a 0.5 Pleasure increase does not have to mean the same in terms of intensity-change as a 0.5 Arousal increase, but we will not go into this issue here.

Second, the set of appraisal banks (the Appraisal System) together must be able to produce non-zero appraisal-values

for all appraisal dimensions. These values must be both positive and negative. This allows the emotional-state to potentially be driven in all directions. Note that this does not need to hold for one bank in particular, since it is perfectly fine if one bank produces mostly positive values for a certain appraisal dimension while another produces mostly negative ones. This would still potentially drive the emotional-state in both directions. Being able to drive the emotional-state in all directions is needed to maximise emotional coverage. Emotional coverage is the ability of the computational model of emotion to attain all possible emotional-states, as defined by the appraisal-dimensions used in the computational model of emotion. Emotional coverage is important for several reasons, of which we mention only one. An agent that is designed to express a set of emotions must also be able to attain these emotions. Not being able to do so presents a huge loss of development effort (i.e. facial expression rendering, emotional behaviours, etc).

Third, appraisal banks need to respond to mutually exclusive contexts. This can be explained by the following. If we assume that r_1 and r_2 are the absolute values of two appraisal-results produced by respectively bank B_1 and B_2 at a certain time, and $r_2 \neq 0$, and the newer version of an appraisal model contains both B_1 and B_2 while the simpler version contains only B_1 , then the simpler model produces r_1 while the newer version produces $r_1 + r_2 \neq r_1$. Nothing can be said about how meaningful $r_1 + r_2$ is, even though r_1 and r_2 may be meaningful by themselves. At least two ways to ensure model-scalability - i.e. incremental emotional instrumentation - exist: first, appraisal banks are never active together in which case $r_1 + r_2$ never happens; second, B_1 knows about B_2 or vice versa so that they can adapt r_1 and r_2 . This introduces a dependency between two versions of the appraisal-model, and such a dependency limits model-scalability. There are several other issues that relate model-scalability, choice of appraisal dimensions and emotional coverage to each other, but these would diverge us too much from the main point.

Mutual exclusiveness is a rather restricting constraint. Fortunately another option is available. If $r_2 \approx 0$ then $r_1 + r_2 \approx r_1$. This means that, if B_1 and B_2 are active at the same time and B_2 is an appraisal bank that "fine-tunes appraisal" while B_1 "looks at the big-picture", then both banks can be active at the same time. Now B_2 incrementally adds more meaning to the appraisal model while staying consistent with the model only containing B_1 .

If we assume the Appraisal Signal Modulator (ASM) is pre-processing appraisal-results before the EMS integrates these, do the guidelines stay the same? Yes and no, appraisal-results still need to be defined on the interval scale, because the EMS still has to integrate them and the set of banks still must be able to produce non-zero positive and negative values for all appraisal dimensions in order to maximise emotional coverage. However, other scenarios are possible for the interplay between two or more banks. We explain one of these scenarios. If the ASM constructs a weighted average² of r_1 and r_2 where the weighing function is based on the intensity of the appraisal-result - intensity can be

² A similar weighted influence of appraisal-results - using attention as weighing function - has been proposed to explain the effects of concurrent appraisals on human emotion (Schimmack et al 2001).

calculate using for example the length of the appraisal-result vector, as shown in equation 2 page 6 - then $\min(r_1, r_2) < r_1 + r_2 < \max(r_1, r_2)$. This means that if both r_1 and r_2 are meaningful when used separately, the appraisal-result as integrated by the EMS is between r_1 and r_2 and has a high chance of also being meaningful. The problem with this approach is that appraisal-results from appraisal banks that should "fine-tune" the appraisal model - like the above example of B_2 - should be added to the appraisal-results of "big-picture" appraisal banks instead of integrated with these results in an average. In equations this means: if $r_2 \approx 0$ then $(r_1 + r_2)/2 \approx r_1/2$, while B_2 was designed to achieve $r_1 + r_2 \approx r_1$. To conclude, without further assumptions the ASM cannot solve the mutual exclusive contexts constraint, but it can soften it. Appraisal banks that "fine-tune" appraisal can be configured to be left untouched by the ASM, and all other appraisal banks can be either averaged by the ASM or mutually exclusive. When needed, the ASM opens up a wide range of different mechanisms to pre-process the appraisal-results of appraisal banks to makes these suitable for integration by the EMS.

EXPERIMENTAL ASSUMPTIONS

We have instrumented a Java version of the arcade game of PacMan (Chow 2003). Since we want to test if our signal-based, modular approach facilitates incremental emotional-instrumentation, and that this incremental instrumentation is feasible even for existing non emotional systems, programming a game ourselves would have seriously diminished the convincing power of our results.

The game of PacMan consists of an "eater" in a rectangular maze, filled with dots, power-pills, fruit and several ghosts. A human player controls the "eater". The goal it is to collect as many points as possible by eating the objects in the maze. When a ghost touches the "eater", it loses a life. When no lives are left, the game is over. However, if the "eater" eats a power-pill, it is temporarily able to eat the ghosts, thus reversing roles. When all dots are eaten, the game advances to the next - more difficult - level.

PacMan as Experimental Platform

We have chosen PacMan for the following reasons. First, PacMan has easy to define goals, like survival and collecting points. This facilitated development of an appraisal model with one bank related to survival (e.g. avoiding ghosts) and then extend this model with a bank related to the goal of collecting points (e.g. eating dots). Second, the "eater" in PacMan potentially has many different emotions that make sense. Eating ghosts, eating dots, being chased, chasing, etc. are all different situations relating to different emotions. This allows us to test to what extend emotional coverage changed depending on the appraisal-model. Third, PacMan is an 'action-packed' environment, which allows us to test our signal-based approach, under continuous-time constraints.

Pleasure Arousal Dominance Dimensions

Our approach does not prescribe a specific set of appraisal dimensions. We have chosen the Pleasure, Arousal,

Dominance (PAD) personality-trait and emotional-state scales by Albert Mehrabian (1980) for the following reasons. First, even though Mehrabian is not an appraisal theorist and his dimensions are generally not considered to be appraisal dimensions, he argues that any emotion can be expressed in terms of values on these three dimensions, and provides extensive evidence for this claim (Mehrabian 1980). This makes his three dimensions suitable for a computational approach³. Second, since the PAD scales are validated for both emotional-states and traits, they provide a useful basis for a computational framework that consistently integrates states and traits (even though we don't use traits in the experiments, this is very valuable for further instrumentation experiments). Last, Mehrabian (1980) provides an extensive list of emotional labels for points in the PAD space. Figure 2 gives an impression of the emotional meaning of combinations of Pleasure, Arousal and Dominance.

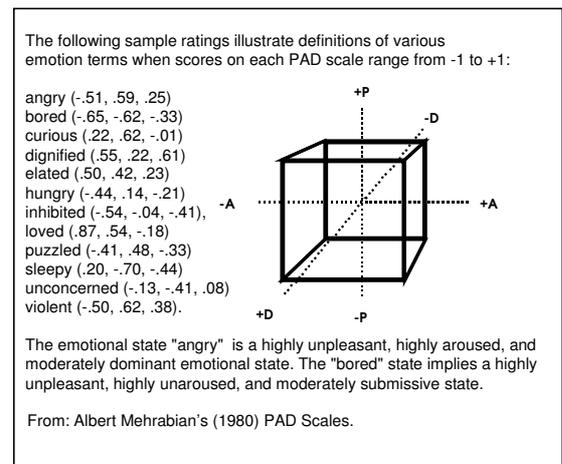


Figure 2: The Mehrabian P-A-D Temperament Scale

In an attempt to instrument PacMan in such a manner that the appraisal-results are defined on the interval scale - i.e. adding results from different banks means something -, we need guidelines to think about events in terms of appraisal-dimensions instead of emotions. What does a certain event mean in terms of P, A and D?

The Pleasure dimension is highly related to the impact an event has on the probability of fulfilment of an agent's desires (e.g. Mehrabian 1980, Reizenzein 2001, Scherer 1993). If the event increases the probability of the outcome, Pleasure is positive, else it's negative. The "eater" in the PacMan game has two desires: survival and collecting points. According to Mehrabian (1980), arousal is highly correlated with activity and alertness. This relates to the expectancy and novelty of an event. The appraisal dimensions expectancy and novelty are related to the amount of attention a certain environmental change gets (van Reekum 2000). Based on these observations we assume that Arousal is the amount of attention needed to address a certain event. A ghost needs a lot of attention, while eating a dot needs only a little.

Dominance is a measure for the influence the situation has on the agent's freedom of choice to act in different ways (Mehrabian 1980). High Dominance implies a large freedom of choice, while low Dominance implies little choice. This

³ It might even be very interesting to study in detail how these non-appraisal dimensions behave in a signal-based appraisal setting.

maps well to the PacMan game. For example, seeing a ghost decreases Dominance, while seeing an edible ghost increases Dominance. We have used these guidelines to think in terms of P, A and D.

Instrumentation of Appraisal Banks: Survival and Collecting Points

To test if context sensitive appraisal banks facilitate the development of scalable appraisal models, PacMan is instrumented in two ways. First, a simple instrumentation based one appraisal bank that emotionally evaluates events related to survival. Second, a more complex instrumentation based on two appraisal banks, one related to survival the other related to collecting points. In both banks we have used event-encoding to simulate emotional meaning of events. The DSS constructs the actual events. We now describe how events are interpreted by the two appraisal banks..

"Survival" bank

This bank appraises only survival related events (Table 1). The rationale for the P, A and D values is based on the guidelines described above. Pleasure depends on the level of obstruction versus conductance of an event related to a goal. For example, seeing a ghost is moderately obstructing for survival, while being eaten is highly obstructing. Arousal is related to the amount of attention an event needs. For example, seeing a ghost needs a moderate amount of attention while losing a ghost needs no attention (because the ghost poses no thread anymore). Dominance is related to the amount of freedom the "eater" has. For example seeing a ghost decreases the amount of freedom, while losing a ghost increases the amount of freedom.

Table 1: "Survival" bank

| Event | Pleasure | Attention | Dominance |
|----------------|----------|-----------|-----------|
| See_ghost | -.5 | 0.5 | -.5 |
| Lost_ghost | 1.0 | 0.0 | 0.5 |
| Eaten_by_ghost | -1.0 | 1.0 | -1.0 |

"Points" bank

This bank appraises only events related to the goal of collecting points. Table 2 shows the events for the "points" bank. Again, the rationale for the P, A and D values is based on the guidelines described above.

Table 2: "Points" bank

| Event | Pleasure | Attention | Dominance |
|------------------|----------|-----------|-----------|
| eaten_ghost, , | 1.0 | 1.0 | 0.0 |
| see_edible_ghost | 0.5 | 0.5 | 1.0 |
| eaten_fruit, | 0.5 | 0.2 | 0.0 |
| eaten_dot, | 0.2 | 0.2 | 0.0 |
| eaten_power | 0.2 | 0.2 | 0.0 |

Appraisal-results

Appraisal-results are produced by both appraisal banks and are based on situational change. This means that whenever an event is interpreted by an appraisal bank at time t , it

compares if this event has already been encountered at time $t-1$. If this is not the case, the appraisal dimension values associated with the event are sent as appraisal-result. If it is the case, nothing is sent. If an event is no longer encountered at time t while it was at time $t-1$, a relaxation function kicks in. This function is responsible for sending enough small values over a short time period - say until $t+x$ - so that these values - when summed - are the exact opposite of the appraisal dimension values associated with the event encountered at time $t-1$. The mechanism has been adapted to work for multiple events, but we will not go into this here.

One of the reasons for implementing appraisal banks in this way is that we are now sure that an appraisal bank outputs both positive as well as negative appraisal values for all appraisal dimensions that are used by the events the bank interprets. As mentioned above, this is an important criterion for emotional coverage. Another reason is that continuous exposure to, for example, eating dots would permanently drive the emotional-state to an extreme value (remember that an appraisal-result is a delta - a change - and that these are just added up by the EMS). Not going into the discussion of whether this is or isn't plausible, it is a problem we would have had to solve in one way or the other for the current experiment. We have chosen for a simple but effective appraisal mechanisms using both situational habituation -i.e. measuring situational change - and subsequent relaxation. We would like to stress, however, that this is just one of many ways an appraisal mechanism could be implemented in order to produce appraisal-results that maximise emotional coverage as well as protect the emotional-state from "walking to extremes".

Context sensitivity

In the simple instrumentation - using only the "survival" bank - context sensitivity is irrelevant. There is just one bank active at all times. In the complex instrumentation context sensitivity is of importance and implemented in the following way. Since survival is more important than points, the "points" bank is inhibited by the "survival" bank. This is implemented by weighing the contribution to the emotional-state of the appraisal-result of the "points" bank relative to the amount of emotional activation (appraisal-intensity) in the "survival" bank. Formula (1) implements the weighing function, where Δ_{goal} ' is the weighted appraisal-result vector as to send to the EMS by the "points" bank, Δ_{goal} is the non-weighted vector, $|\Delta_{survival}|$ is the length of the appraisal-result vector of the "survival" bank and the cubic root of 3 is the maximum length of an appraisal-result vector⁴.

$$\Delta_{goal}' = \Delta_{goal} * \left(1 - \frac{|\Delta_{survival}|}{\sqrt[3]{3}} \right) \quad (1)$$

If the "survival" bank is highly active, the appraisal-results from the "points" bank have almost no influence on the final appraisal-result sent to the EMS and vice-versa. This mechanism should result - and the experiment shows it does - in emotions produced by the complex model that are

⁴ Calculating intensity in a Pleasure-Arousal theory of emotion based on the length of the Pleasure-Arousal vector is psychologically plausible (Reisenzein 1994).

consistent with the emotions produced by the simple model in survival-related situations. This mechanism exemplifies context-sensitivity of appraisal banks. Note that appraisal banks allow abstraction from the actual event interpretation, facilitating a modular approach to the appraisal model.

Integration of Appraisal-results

Appraisal-results are integrated by the EMS using equation (2), where E_t is the emotional-state at time t , E_{t+1} is the new emotional-state, n is the number of appraisal banks and ΔPAD_i the appraisal-result vector of bank i at time t .

$$E_{t+1} = E_t + \sum_{i=0}^n \Delta PAD_i \quad (2)$$

The EMS adds up appraisal-results produced by the banks.

EXPERIMENTAL RESULTS

The experiment itself consists of playing the first level of the PacMan game (by eating all dots), losing a life two times during the process, and eating at least one Ghost. To be able to compare the two different instrumentations and the effect of triggering the appraisal banks of the instrumentations at different frequencies - i.e. at 5 times per second and 10 times per second - , we have configured PacMan in such a way that we were able to test all four instrumentations -i.e. both instrumentations at 5 and 10 times per second - in just one test-run. We instantiated four different versions of the emotion system and events were delivered to the appraisal banks of all four instantiations. Plots of the emotional-state changing over time have been generated.

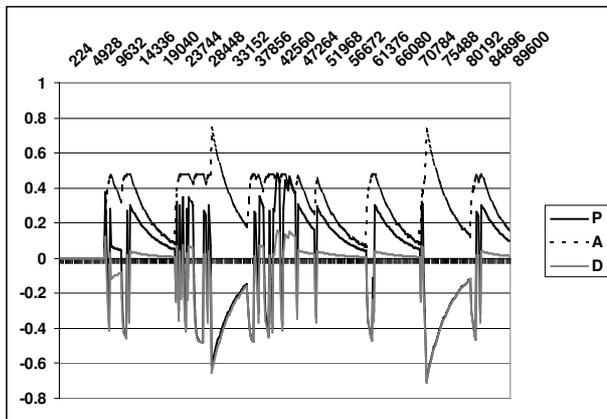


Figure 3: "Survival" PacMan, 200ms instrumentation

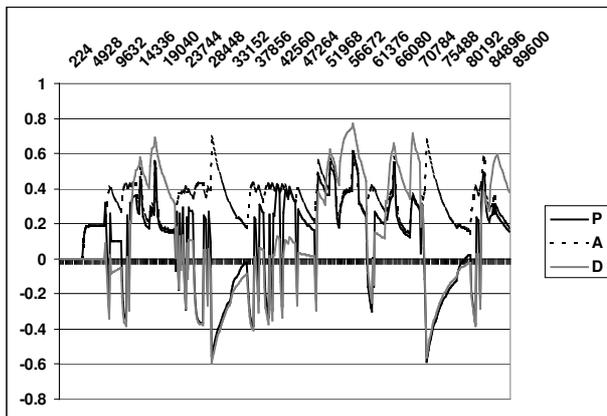


Figure 4: PacMan using both banks, 200ms instrumentation

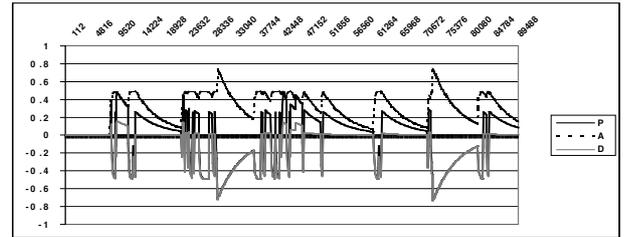


Figure 5: "Survival" PacMan, 100ms instrumentation

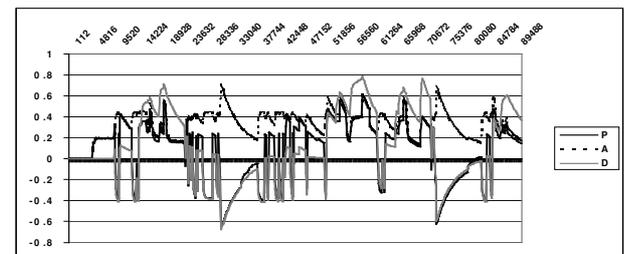


Figure 6: PacMan using both banks, 100ms instrumentation

Model scalability

All graphs clearly show broad emotional coverage (the 2-bank instrumentation shows broader coverage, however), irrespective of appraisal rate. Even with a minimal instrumentation based on 3 events, the emotional-state varies substantially from $(-P, +A, -D)$ to $(-P, -A, -D)$ to $(+P, -A, 0D)$. Furthermore, the ability to define context-sensitive appraisal-banks clearly allows us to first define this minimal instrumentation and subsequently scale-up the model by adding a new bank. The context sensitivity of the banks - using inhibition of the "points" bank - results in consistent behaviour of the emotional-state. In the "both banks" instrumentation the emotional-state is more meaningful due to the second bank, shown by the overall difference between Figure 3 and 4 and in particular the effect the "eaten_ghost" event has on P and D around $t=56672$. The emotional-state is at least as meaningful in those situations where the "survival" instrumentation already produced a meaningful emotional-state, shown by the effect the "eaten_by_ghost" event has on P, A and D around $t=28448$ and $t=70784$ in Figure 3 and 4. This shows that context-sensitive appraisal banks - enabled by our signal-based approach - facilitate model-scalability.

Runtime-scalability

Comparison of the results between the 100ms and 200ms instrumentations shows that our signal-based, context-sensitive appraisal banks are insensitive to a 100ms difference in triggering frequency. We can see that Figure 3 and Figure 5 as well as Figure 4 and 6 are pair-wise identical. This insensitivity is mainly the result of appraising situational *change* instead of the situation itself. Appraisal-results are identical assumed that the difference between the frequency of both instrumentations is not so large that the slower-frequency-instrumentation completely skips both the delivery and the retraction of an event from its current "blackboard". A large difference is thus a risk for appraising situational change, but many ways exist to solve this problem using

more sophisticated "event delivery". This shows that our approach supports flexible triggering.

The potential of our approach for runtime-scalability related to quality/performance trade-off is indicated by the fact that the "points" bank actually fills-in the non-emotional episodes of the "survival" bank. During run-time the "points" bank can be switched off, still resulting in meaningful but less detailed emotions, as shown by Figure 3. Of course in our case both banks consume virtually no resources, but in a situation where two different appraisal mechanisms are used, this runtime-scalability becomes useful.

CONCLUSION

In this paper we have addressed the problem of incremental emotional instrumentation of systems. That is, how to develop computational models of emotion based on a step-by-step addition of sophistication to a such a model resulting in meaningful and more sophisticated emotions of the agent that is consistent with the emotions resulting from the simpler models. We have proposed the FeelMe framework (DeGroot 2004) as a solution to this problem. The FeelMe framework is a modular, signal-based approach to computational emotions. In this paper we have focussed on the Appraisal System in the FeelMe framework. Context-sensitive appraisal banks are introduced to facilitate the development of scalable appraisal models. The results of an experiment we have conducted with a game agent show the following. An appraisal model using two appraisal banks - the first being sensitive to all events related to survival and the second being sensitive to all events related to collecting points - results in more sophisticated emotions than an appraisal model with just the "survival" bank. The "survival" appraisal bank in our experiment inhibits the "points" appraisal bank. This inhibition provides consistency between the two instrumentations. Consistency between appraisal models and incremental emotional sophistication are two of the requirements for model-scalability and runtime-scalability, indicating that context sensitive appraisal banks - enabled by our signal based approach - facilitate the development of scalable appraisal models.

Runtime-scaling of appraisal models is useful in domains in which computation time is an unknown factor, because it enables trading-off emotional quality with available computation time, just like 3D-graphic-engines and chess engines. The experiment also indicates that our signal-based approach is flexible regarding the frequency of appraisal. This flexibility enables a different form of runtime-scaling, namely scaling based on a trade-off between temporal quality and performance.

Even though the number of appraisal banks was small in order to test scalability, we think that these results show that our modular, signal-based approach to computational models of emotion has many benefits for the gaming and virtual agent arena.

FURTHER WORK

Possible extensions of our dynamic approach to computational emotions include modelling the mood of an NPC, modelling the effect mood can have on the emotional state, and the use of our approach in multi-agent environments.

REFERENCES

- Baillie-de Byl, Penny. 2003 "A Six-Dimensional Paradigm For Generating Emotions in Virtual Characters." *IJIGS 2(2)*: 72-79.
- Chow, Benny. PacMan. 2003
http://www.bennychow.com/pacman_redirect.shtml.
- DeGroot, Doug and Broekens, Joost. 2003. "Using Negative Emotions to Impair Gameplay." *Proc. of the BNAIC*: 99-106.
- DeGroot, Doug. 2004 "Integrating Emotional Traits and States in Virtual Humans", in preparation.
- Jennings, N. and Sycara, K. and Wooldridge, M. 1998. "A Roadmap of Agent Research and Development," *Autonomous Agents and Multi-Agent Systems 1(1)*: 7-38.
- Mac Namee, Brian and Cunningham, Pdraig. 2003. "Creating Socially Interactive Non Player Characters: The m-SIC System." *IJIGS 2(1)*: 28-35.
- Marsella S. and Gratch J..2001 "Modeling the Interplay of Emotions and Plans in Multi-Agent Simulations." *Proc. of the 23rd Annual Conference of the Cognitive Science Society*.
- Mehrabian, Albert. 1980. *Basic Dimensions for a General Psychological Theory*. OG&H Publishers. Cambridge Massachusetts.
- Ortony, A. Clore G.L., Collins A. 1998. *The Cognitive Structure of Emotions*. Cambridge University Press.
- Reekum, van C. 2000. *Levels of Processing in Appraisal: Evidence from Computer Game Generated Emotions*. Phd Thesis nr. 289, University de Geneve, Section de Psychology
- Reisenzein, R. 1994. "Pleasure-Arousal Theory and the Intensity of Emotions." *Journal of Personality and Social Psychology 67(3)*: 525-539.
- Reisenzein, R. 2001. "Appraisal Processes Conceptualized from a Schema-Theoretic Perspective: Contributions to a Process Analysis of Emotions." In *K.R. Scherer, A. Schorr and T. Johnstone (eds), Appraisal processes in emotion: Theory, Methods, Research*. Oxford University Press.
- Russell, J.A. 2003. "Core Affect and the Psychological Construction of Emotion." *Psychological Review 110(1)*: 145-172.
- Schimmack, U., Colcombe, S., & Crites, S. Pleasure and displeasure in reaction to conflicting picture pairs: Examining appealingness and appallingness appraisals. *Unpublished Manuscript*. 2001.
- Smith, Craig A., Kirby, Leslie D. 2000. "Consequences require antecedents: Toward a process model of emotion elicitation." In *Forgas, Joseph P. (Ed), Feeling and Thinking: The role of Affect in Social Cognition*. Cambridge University Press.
- Scherer, K. R. 1993. "Studying the Emotion-Antecedent Appraisal Process: An Expert System Approach." *Cognition and Emotion 7(3/4)*: 325-355.
- Scherer, Klaus R. 2001. "Appraisal Considered as a Process of Multilevel Sequential Checking." In: *K.R. Scherer, A. Schorr and T. Johnstone (eds), Appraisal processes in emotion: Theory, Methods, Research*. Oxford University Press

COMPONENT BASED APPROACH FOR EMOTIONAL BEHAVIOR OF 3D CG CHARACTERS

Hirokazu Notsu¹, Yoshihiro Okada^{1,2} and Koichi Nijima¹

¹Graduate School of Information Science and Electrical Engineering, Kyushu University
6-1 Kasuga-koen, Kasuga, Fukuoka, 816-8580 JAPAN
{h-notsu, okada, nijima}@i.kyushu-u.ac.jp

²*Intelligent Cooperation and Control, PRESTO, JST*

KEYWORDS

3D Games, CG Animation, CG Character, Neural Networks, Emotional Behavior, *IntelligentBox*

ABSTRACT

At present, behaviors of Non Player Characters (NPCs) in computer games are not satisfactory because they are pre-defined and then very simple. To overcome this problem, Namee, et al. proposed an agent system with emotional model units to create more natural behaviors of NPCs by controlling their actions according to their emotion (Namee and Cunningham 2001). The research purpose of the authors is to propose the software architecture that provides NPCs those have emotional behaviors by combining required primitive functionalities represented as software components. As the first trial to achieve this research purpose, the authors designed the agent system proposed by Namee, et al. using *IntelligentBox*, which is a component based 3D software development system. This paper explains the design of the agent system using *IntelligentBox* and the implementation of its emotional model, i.e., a simple stimulus response model. The authors also describe the validity of the implementation by showing the experiment.

INTRODUCTION

Recently 3D computer graphics has become frequently used in movies and video games. In this situation, the role of Non Player Characters (NPCs), which are 3D CG characters used in a video game, has become important and personalities of NPCs have become necessary in order to make them have various believable behaviors. Actually there are *The Sims* (thesims.ea.com) and *Black & White* (www.bwgame.com) as an example of the successful game, those have shown that the personalities, moods, and relationships of NPCs are important factors to make the game play entertaining. However, game designers have to write a lot of scripts for all NPCs to define their behaviors, and this requires much computation and sophisticated data structures if there are too many NPCs in the game. For these reasons, behaviors of NPCs in conventional games are very simple and then game players can easily predict behaviors of NPCs of a game and loose interest in playing it after they play several times. To overcome these limitations, an agent system is required to create more natural behaviors of NPCs by controlling their actions according to their emotion. Our research purpose is to propose the software architecture that allows us to develop

various agent systems only by combining required primitive functionalities represented as software components.

This paper explains the design of the agent system using *IntelligentBox* (Okada et al. 1995, 1998, 2000), which is a component based development system for 3D graphics applications. The paper also describes emotional model units implemented as composite components of *IntelligentBox*. Furthermore, we describe the validity of the implementation by showing the experiment.

PROACTIVE PERSISTENT AGENT ARCHITECTURE (PPA)

First of all, we refer to a proactive persistent agent architecture (PPA) proposed in the paper (Namee and Cunningham 2001). The PPA architecture has two properties as an agent system. One is that agents based on the PPA architecture are *proactive* in the sense that they can take the initiative and follow their own goals, irrespective of the actions of the player. The other is *persistence* refers to the fact that at all times, all NPCs in a virtual world are modeled, regardless of their location relative to that of the player. A schematic illustration of the PPA architecture is shown in figure 1.

The architecture is a modular system made up of four key components which determine their interaction by referring to *KnowledgeBase* and other components. Finally *Selection* determines next interaction of an NPC by referring interactions determined by each component units. These key components are as follows:

The *Schedule* manages schedule of a character. For example, a character might get up in the morning at home and go home after their work. An NPC moves autonomously without any intense computation or sophisticated data structures by instilling characters with a schedule.

The *Role Passing System* gives a role to a character and the adoption of each of roles would significantly changes the behavior of the character (Namee et al. 2002a). Depending on the situation in which the agent is found, different roles are layered on top of this simple agent to drive its behavior. The adoption of particular roles is driven by the agent's schedule.

The *Social Unit* determines behaviors of NPCs based on their emotional model. Their emotion consists of personality, mood and their relationship. This system is called the μ -SIC System (Namee and Cunningham 2002b).

Although the scenario of the game which we are going to create has not decided yet, these units mentioned above are not enough to allow NPCs achieve their goals of a game, so a traditional planning unit should be used. As the first trial of our research, we implemented the main functionality of the μ -SIC System as the composition of software components of *IntelligentBox*. In the following sections, we explain its details.

EMOTIONAL MODEL UNITS (IMPLEMENTATION OF THE μ -SIC SYSTEM)

In the u-SIC system, behaviors of NPCs are determined by Artificial Neural Network (ANN) that already learned their emotional model. Every NPC does not need to have its own ANN and there is an only one copy of the ANN in the system. Whenever NPCs take a next action, they request one copy of the ANN. The reason why the system was implemented in this way is to minimize the storage requirements.

In the followings, we describe our emotional model for the u-SIC system. Among them are Personality, Mood, Relationship and Interaction.

Personality

To define the personality of an NPC, we considered five factors model called Big five (Tanno 2003). Big five is based on the idea that our human personality can be specified by the combination of the fundamental characteristics. Five factors and their characteristics are as follows:

1. Extroversion
Activity and energy level traits, sociability and emotional expressiveness.
2. Agreeableness
Altruism, trust, modesty, pro-social attitudes.
3. Conscientiousness
Impulse control, goal directed behavior.
4. Neuroticism
Emotional stability, anxiety, sadness, and irritability.
5. Openness
Breadth, complexity, and depth of an individual life.

According to Byron (Reeves and Nass 1998), extroversion and agreeableness are more significant rather than other factors, especially when we classify our personalities based

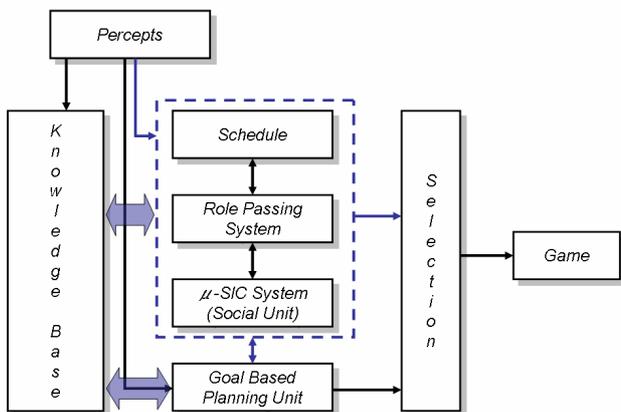


Figure 1: A schematic of the PPA architecture.

on the influence of external stimulus. Then we decided to employ extroversion and agreeableness as two main factors to express the personality of an NPC because we assume these factors are satisfactory for the emotional expression of a game character.

Mood

A character's mood is specified by two measures, valance and arousal. Valance means whether the mood is positive or negative, and arousal means the intensity of that mood. These two measures are fundamental elements to express the mood and independent on each other so the combination of these measures determines primary emotions. For example, we will feel angry when valance is low and arousal is high.

Relationship

The relationship among characters is an important factor to determine their action/interaction. As well as 3D CG characters, this agent system is also applicable to robots. We are supposed to apply our agent system to AIBO, which is an entertainment robot produced by Sony Corp. as well as a 3D CG character. Then, we uses an AIBO typed CG character in the experiment described in the later section. Our AIBO CG character has only friendship as the relationship parameter due to the simplicity.

Interaction

This subsection describes character's actions that an ANN determines by referring to emotion parameters. We specify seven actions as described below because these actions express character's emotions clearly and strongly correspond to primary emotions. Moreover these actions are considered to have the great influence on emotion parameters of other characters.

1. KISS
Expression of "love" to a partner
Action showing "joy"
2. FLIRT
Good impression against a partner
Action showing "welcome"
3. CHAT
Chat with a partner
No hostility and no feeling of "dislike" a partner.
4. NONE
No action against a partner.
5. SLEEP
Feeling of "dislike" and insult a partner.
6. RUNAWAY
Feeling of "fear" a partner.
7. ANGRY
Feeling of "anger" with a partner.

ARTIFICIAL NEURAL NETWORK

By referring to the current values of the emotion parameters, an ANN determines which interaction should be invoked. The back propagation of error was used to train the network. The network is a three layer perceptron network, which has a single middle layer. The middle layer consists of six nodes

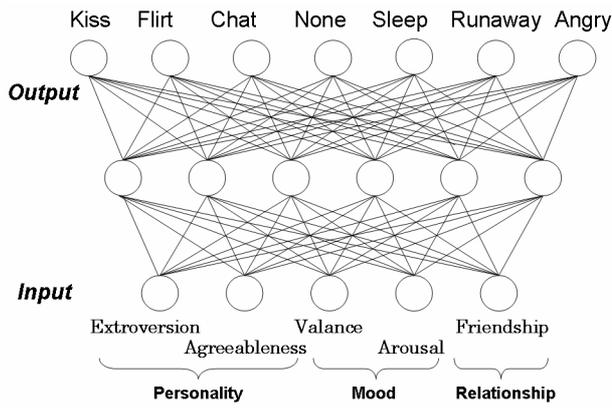


Figure 2: Structure of ANN.

as shown in Figure 2. The input layer consists of five nodes for the personality, i.e., extroversion and agreeableness, for the mood, i.e., valance and arousal, and for the relationship, i.e., friendship. Those have a value in 0.0 to 1.0. The output layer consists of seven nodes each for the possibility of each action/interaction which the current character should take. The output layer nodes also output a value in 0.0 to 1.0.

The ANN requires a training data set for the training. However, there are no available databases which contain the believable information on how people or animals interact with each other. An artificial data set was required for this reason and so we manually created it as follows. As the first step, we assumed the relationship as the distance between each of the primary emotions and each of the seven actions. For example, Angry is semantically far from Kiss but close to Sleep. As the second step, we defined the probability of

Table 1: Sample of a training data set.

| Sample input data | | | | | Sample output data | | | | | | | |
|-------------------|-----------|----------|---------|------|--------------------|-------|------|-------|------|---------|-------|--|
| Extro | Agreeable | Positive | Arousal | Like | Kiss | Flirt | None | Sleep | Chat | Runaway | Angry | |
| 0.7 | 0.2 | 0.8 | 0.9 | 0.9 | 0.9 | 0.7 | 0.5 | 0.1 | 0.8 | 0.7 | 0.3 | |
| 0.5 | 0.5 | 0.8 | 0.9 | 0.9 | 0.9 | 0.7 | 0.5 | 0.1 | 0.8 | 0.7 | 0.3 | |
| 0.8 | 0.2 | 0.7 | 0.2 | 0.7 | 0.7 | 0.9 | 0.1 | 0.3 | 0.8 | 0.5 | 0.5 | |
| 0.7 | 0.5 | 0.7 | 0.5 | 0.7 | 0.7 | 0.9 | 0.1 | 0.3 | 0.8 | 0.5 | 0.5 | |
| 0.5 | 0.5 | 0.2 | 0.2 | 0.5 | 0.3 | 0.1 | 0.9 | 0.7 | 0.7 | 0.5 | 0.5 | |
| 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.3 | 0.1 | 0.9 | 0.7 | 0.7 | 0.5 | 0.5 | |
| 0.2 | 0.8 | 0.5 | 0.2 | 0.2 | 0.1 | 0.3 | 0.8 | 0.9 | 0.3 | 0.3 | 0.7 | |
| 0.5 | 0.8 | 0 | 0.4 | 0.4 | 0.1 | 0.3 | 0.8 | 0.9 | 0.6 | 0.3 | 0.7 | |
| 0.4 | 0.5 | 0.8 | 0.8 | 0.5 | 0.6 | 0.7 | 0.5 | 0.1 | 0.9 | 0.7 | 0.3 | |
| 0.8 | 0.5 | 0.8 | 0.5 | 0.9 | 0.8 | 0.7 | 0.3 | 0.1 | 0.9 | 0.7 | 0.3 | |
| 0.2 | 0.2 | 0.5 | 0.7 | 0.2 | 0.2 | 0.5 | 0.4 | 0.3 | 0.6 | 0.9 | 0.1 | |
| 0.2 | 0.5 | 0.2 | 0.5 | 0.5 | 0.7 | 0.5 | 0.4 | 0.4 | 0.6 | 0.9 | 0.1 | |
| 0.5 | 0.2 | 0 | 0.8 | 0.2 | 0.3 | 0.5 | 0.7 | 0.8 | 0.4 | 0.5 | 0.9 | |
| 0.5 | 0.8 | 0.5 | 0.8 | 0.2 | 0.3 | 0.5 | 0.6 | 0.8 | 0.4 | 0.1 | 0.9 | |

each action according to its relationship with the five emotion parameters. For example, a character will select Angry when the values of extroversion and arousal are high and agreeableness, valance and friendship are low. Table 1 shows the partial data of 140 hand crafted data.

It is very likely that the network would over-fit to a training data set and not be applicable to new input data because the network could not be generalized well if the training data set is too small. To overcome this problem, we created a larger data set, 1400 training elements, by adding Gaussian noise to an original data set. A five-fold cross validation was

performed to check the validity of the ANN trained using the set of 1400 training elements. As a result, the ANN achieved the accuracy of 82%. This accuracy means that 82% of output data from the ANN are correct against the corresponding input data. Since this validation check is not for raising the accuracy, we consider this accuracy is enough to obtain satisfactory behaviors of NPCs.

INTELLIGETNBOX AND ITS COMPONENTS FOR EMOTIONAL BEHAVIOR

IntelligentBox is a constructive visual 3D software development system. *IntelligentBox* provides 3D reactive objects as its primitives called *boxes*. Each *box* has a unique functionality and a 3D visible shape. Its functionality is related to variables called *slots*. By transmitting *slot* values among several *boxes*, their functionalities are combined. *IntelligentBox* also provides a dynamic data lineage mechanism called 'slot connection' that allows users to construct 3D graphics applications by only combining existing *boxes* through direct manipulations on a computer screen.

New boxes for ANN

To implement the agent system using *IntelligentBox*, we developed the following four *boxes*, i.e., *NeuralNetBox*, *CharacterBox*, *CharacterControlBox* and *InteractionControlBox*

The functionality of *NeuralNetBox* is to provide an ANN. It calculates the output results of the network from the input

values. *CharacterBox* holds five parameter values, i.e., two personality parameters, two mood parameters and one relationship parameter as its five *slot* values. The functionality of *CharacterBox* is to change the emotion parameters of its corresponding character (NPC) so one *CharacterBox* is attached to each character. The individuality of a character is determined by the difference of emotion parameters. *CharcterControlBox* keeps the name of a character which is attempting to instigate an interaction and refers to *CharacterBox* attached to the character. *NeuralNetBox* gets input parameters from the character,

strictly speaking from the *CharacterBox*, specified by *CharacterControlBox*. *InteractionControlBox* plays the same role of *Selection* unit in the PPA shown in Figure 1. This *box* determines next action/interaction for the corresponding character according to output results gathered from each other unit. Although this paper describes only the implementation of the emotional model units, this *box* is extensible in order to accept output results from *Schedule*, *Goal Planning Unit* and so on.

Composite box and slot connection

The new composite *box* which has the functionality of the agent system is created by composing the above new developed *boxes* and by connecting their *slots*. Figure 3 illustrates parent-child relationship hierarchy of the component *boxes*.

Figure 4 shows messages to transfer *slot* data between each two *boxes*. *CollisionControlBox* detects the collision of characters with each other and then *CharacterControlBox* sets the emotion parameters of the current character to input *slots* of the *NeuralNetBox*. *NeuralNetBox* calculates output values of the network and hence sets those to the *slot* value of *InteractionControlBox*. Finally, *InteractionControlBox* determines next action and inform it to the corresponding character, strictly speaking the corresponding *CharacterBox*. Simultaneously *CharacterBox* changes its emotion

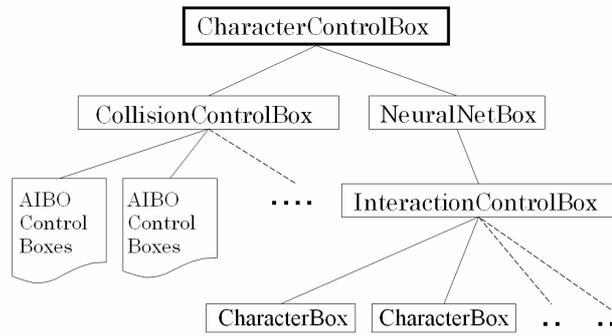


Figure 3: Parent-child relationship hierarchy of *boxes*.

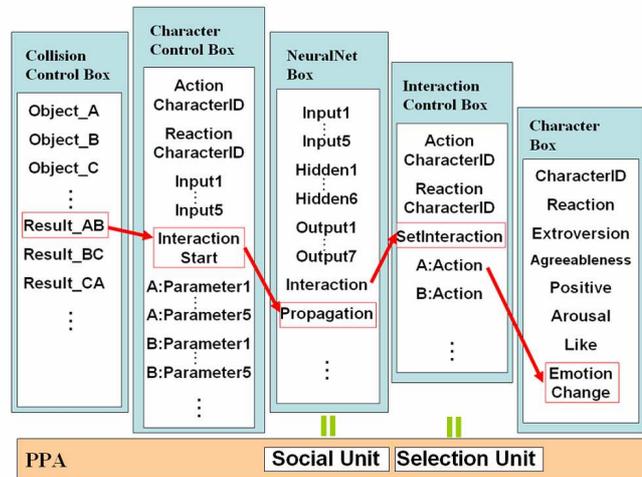


Figure 4: Data flow between each two *boxes* by slot connection.

parameters.

EXPERIMENT

Figure 5 shows a screen snapshot of the agent system developed using *IntelligentBox*. The colorful *boxes* located at the upper part of Figure 5 are the components of the agent system. Each *box* shows the *slot* values on its right-hand side. There are five AIBOs, 3D CG characters, in the field. On the upper part of each AIBO, its character name and its current action are displayed. AIBO moves in the field with randomly changing its direction and, they take an action related to the interaction determined by *InteractionControlBox* whenever they collide with other AIBO. The sphere surrounding an AIBO expresses the state of its emotion using RGB colors. Red expresses the intensity of arousal, Green expresses that of friendship and Blue is valance. If emotion parameters of a character changed, the color of the sphere also changes. Messages flow as explained below is performed when new interaction between two AIBOs occurs. In the figure 6, *CollisionControlBox* sends the message of the collision between two AIBOs to *CharacterControlBox* (1 in Figure 6). *CharacterControlBox* gets the emotion parameters from the

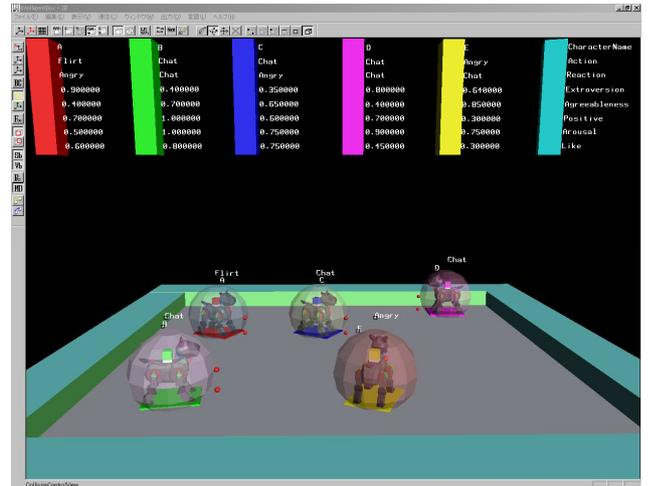


Figure 5: Screen snapshot of the agent system using *IntelligentBox*.

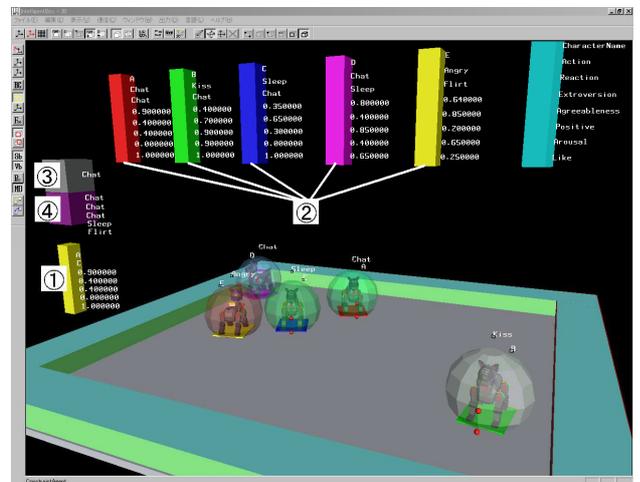


Figure 6: Screen snapshot of one scene during the simulation.

CharacterBox of the corresponding character which is attempting to start the interaction (2 in Figure 6) and sets the parameters to the input slots of *NeuralNetBox* (3 in Figure 6). *NeuralNetBox* calculates outputs of the network and sets them to the slot values of *InteractionControlBox* (4 in Figure 6). Consequently, *InteractionControlBox* determines a next action/interaction of the character and sets it to the *slot* of the corresponding *CharacterBox* with reference to the character name specified by *CharacterControlBox*. *CharacterBox* changes the emotion parameters according to the interaction. A character name, an interaction, and emotion parameters of each character are displayed on the right of the corresponding *CharacterBox*.

AIBOs select their new action according to their current emotion and the AIBO which receives the action changes its emotion sequentially. Moreover changing the emotion parameters affects to the next action of the character. In this way, it is possible to avoid simple and monotonous behaviors of NPCs.

CONCLUDING REMARKS

This paper explained the implementation of the agent system with emotional model units using *IntelligentBox* referring to the PPA. *IntelligentBox* is a component based 3D graphics software development system. We developed several components for the agent system to implement the main part of its functionality. We also checked behaviors of 3D CG characters, AIBO type CG characters, by the experiment. In this experiment, NPCs' interactions are determined based on just their emotion parameters, those interactions also affects other NPCs' emotion and then their next interactions are determined by the last interaction of other NPCs. We can confirm visually emotion changes of characters according to the colors of spheres surrounding each of them. In this way, game players will enjoy a game since it is difficult for them to predict next actions of NPCs even if they play the game repeatedly.

The composite component we implemented referring to the μ -SIC System in the PPA is not enough to represent real, natural human behaviors. When we need such real, natural human behaviors in 3D games, we should employ another sophisticated, complicated emotional model that enables to strictly simulate the emotion process of real human.

As future work, we will develop the functionality for applying motion data to NPCs in order to make NPCs walk naturally or take natural actions. We will also modify the component structure of the agent system to allow the user to interact the system. Moreover, we are supposed to create a game scenario and to develop other remaining components of the PPA, i.e., schedule units, goal based planning units and so on as well as the emotional model units in order to make behaviors of NPCs more natural.

REFERENCES

- Namee, B.M., Cunningham, P. 2001 "Proposal for an Agent Architecture for Proactive Persistent Non Player Characters," Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science pp. 221-232, .
- Namee, B.M. Dobbyn, S., Cunningham, P. and O'Sullivan, C. 2002 "Men Behaving Appropriately - Integrating the Role Passing

Technique into the ALOHA System" In (R. Aylett, L.Canamero Eds.), Proceedings of Animating Expressive Characters for Social Interactions, Symposium of the AISB'02 Convention, pp 59-62, Imperial College, London.

Namee, B.M and Cunningham, P. 2002 "The μ -SIC System: A Connectionist Driven Simulation of Socially Interactive Agents," University of Dublin, Department of Computer Science, Technical Report TCD-CS-2002-43.

Okada, Y. and Tanaka, Y. 1995 "IntelligentBox: A Constructive Visual Software Development System for Interactive 3D Graphic Applications." Proc. of Computer Animation '95, IEEE Computer Society Press, pp.114-125.

Okada, Y. and Tanaka, Y. 1998 "Collaborative Environments of IntelligentBox for Distributed 3D Graphics Applications." The Visual Computer, Vol. 14, No. 4, pp.140-152.

Okada, Y. and Itoh, E. 2000 "IntelligentBox: Its Aspects as a Rapid Construction System for Interactive 3D Games." Proc. of First International Conference on Intelligent Games and Simulation. SCS Publication, pp.114-125.

Reeves, B. and Nass. C. 1998 "The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places", Univ of Chicago Pr.

Tanno, Y. 2003 "Psychology of Personality: Personality considered from Big five and Clinical," SAIENSU-SHA, Tokyo (in Japanese)

BUILDING INTELLIGENCE IN GAMING AND TRAINING SIMULATIONS

Dennis Jacobi^a, Don Anderson^a, Vance von Borries^a,
Adel Elmaghraby^b, Mehmed Kantardzic^b, Rammohan Ragade^b,
Quasim Mehdi^c, Norman Gough^c

^aIntellas Modeling and Simulation, LLC 15424 Beckley Hills Drive, Louisville, KY, USA;

^bJ. B. Speed Scientific School, University of Louisville, Louisville, KY, USA

^cUniversity of Wolverhampton, School of Computing and Information Technology,
35/49 Lichfield S Wolverhampton, WV1 1SB United Kingdom

djacobi@intellas.com

KEYWORDS

Simulation, Agents, Artificial Intelligence, Game, Particle swarm

ABSTRACT

Current war games and simulations are primarily attrition based, and are centered on the concept of “force on force.” They constitute what can be defined as “second generation” war games. So-called “first generation” war games were focused on strategy with the primary concept of “mind on mind.” We envision “third generation” war games and battle simulations as concentrating on effects with the primary concept being “system on system.” Thus, the third generation systems will incorporate each successive generation and take into account strategy, attrition and effects.

This paper will describe the principal advantages and features that need to be implemented to create a true “third generation” battle simulation and the architectural issues faced when designing and building such a system. Areas of primary concern are doctrine, command and control, allied and coalition warfare, and cascading effects. Effectively addressing the interactive effects of these issues is of critical importance. In order to provide an adaptable and modular system that will accept future modifications and additions with relative ease, we are researching the use of a distributed Multi-Agent System (MAS) that incorporates various artificial intelligence methods. (Anderson 2002a, Anderson 2002b)

INTRODUCTION

The act of wargaming has been practiced for centuries as more of an art than a science. The first generation of wargames can best be described as “mind on mind” with such strategic abstractions as the game of chess. The second

generation of wargames can best be described “force on force”, with mathematically based attrition models as the engine for resolving an engagement. This paper formally introduces what has been called a Third Generation Wargame which uses the concept of “system on system” and that incorporates the interactive and cascading effects within each and between systems. The goal of a Third Generation system is to simulate as realistically as possible the variety of effects that military operations will have on a scenario. We use the term “scenario” to mean the strategic level of operations, including not only battle effects, but also other factors in the theater of operation such as morale, logistics, civilian support/unrest and refugees. The granularity in which to program such a complex system is one of the focuses of our research. The level of granularity that is sufficient to produce an effective and useable training and planning tool are one of the focuses of our research.

The use of a multi-agent (Sycara 1998; Weiss et al. 2000) architecture (MAS), with added intelligence was chosen as our platform for research for a number of reasons. Independent agents mirror the reality of the players in a theater of operations and allow for the interactive and cascading effects that occur. Agents can also be created at any level of granularity desired from an individual person to a large unit. Our research thus far has indicated that, even when the overall granularity level is at regimental level, certain key individuals do play important roles. Certain select key commanders, aces, and heroes can be made a part of the agent structure to influence the unit as a whole.

The purpose of our research is to design and build an architecture that will serve as a strategic training tool. The emphasis is on creating a tool. This is not a predictor of future action, but a tool in which strategic moves can be simulated to avoid major mistakes and provide, over several iterations, a set of boundaries in which the campaign is

likely to be conducted. Planners can conduct various “what if” exercises to measure reaction to their actions.

APPROACH AND METHODOLOGY

The approach that has been pursued is to take a portion of the North African Campaign during World War II from 8 November 1942 to 13 May 1943 and provide a well-researched historical campaign to test the accuracy of a computerized model. The initial version will have its agents “built out” with an increasing number of rules and attributes that will help determine actions it would take given a situation. Artificial intelligence and learning components will be added so that as a player has battle experiences it can learn to proceed differently the next time. The initial computerized version will be created by the development team to prove the methodology of the approach. A toolkit will later be created to provide trainers and planners the ability to place specific characteristics into generic agents to create their own scenarios. Key to the development is defining methods in which relationships between agents are coordinated in a multi-agent system, and who/how the relationships are created.

Agent Development

A set of generic agents will first be developed from which individualized characteristics can be added. Using object oriented technology concepts, an abstract generic agent will be defined, which has these characteristics (called attributes) and functions (called methods or member functions). Further, each agent has a knowledge base consisting of rules to act in given situations. This will be appropriate to handling the “what-if” situations. In addition, each agent has a learning component to handle learning specific tasks and having a learned response as a result. This component parallels training issues and subsequent performance.

The initial design of the agent-based system allows the integration of agent learning. The learning mechanisms will be fully designed and then adjusted based on our experimentation result. An intelligent agent will be created that will be given a minimum of background knowledge at the beginning (doctrine in the form of rules), and then learn appropriate “behavior” as it becomes more experienced. Initially, an agent will have in it attributes, methods, rules and a “blank” learning component. Attributes include the various assets in terms of personnel, equipment and capability. Methods are doctrinal based ways of operation. Rules provide the boundaries and limitations for the unit. Initial setting for the learning component may come from historical precedent, as in the case of a training system, or intelligence data, as in the case of a planning system. This would essentially bring the agents or “military unit” up to pre-battle readiness. Gradually, as the agent gains

experience, more knowledge would be stored for decision-making. This learning approach presents a satisfactory solution to the *trust* and *competence problems* of intelligent agents. While the agent gradually develops its ability, the users of the system obtain more trust in the agent’s decisions and actions. The generic design of the agent layers is show below in Figure 1.

| |
|--------------------|
| ID |
| Attributes |
| Methods |
| Rules |
| Learning Component |

Figure 1. Generic Agent Structure

Given this abstract agent concept which provides a template, specific agent types can be defined as a subclass (extension). This process may have further subclasses to desired levels, including one to one, many to one, one to many or many to many relations among agents as appropriate. Clearly, there may be an exponential growth in the complexity of these agents and their relationships. Hence agents will be aggregated at appropriate levels of granularity and the simulation will examine the relevant scenarios. (Davis 1995; Herz and Macedonia 2002; Smith 1998) Scenarios (theaters or circumstances) and transitions amongst the scenarios will determine which agents will be under consideration in the theater.

Mechanisms for learning by agents are discussed in Section 2.3. By acquiring knowledge from different sources, the agent gradually learns how to better execute the desired objective. Through incremental learning agents become more competent. As they accumulate knowledge about different situations they can handle them more successfully. Also, the agents can be trusted. (Palmer, Stone 2000)

Advantages of a Multi-agent Architecture

Four main characteristics describe the development of a military Multi-Agent System (MAS):

- *Real-time* domains are those in which success depends on acting in response to a dynamically changing environment.
- *Noisy* domains are those in which agents cannot accurately perceive the world, nor can they accurately affect it.
- *Collaborative* domains are those in which a group of agents share a common goal.

- *Adversarial* domains are those in which there are agents with competing goals.

Multi-agent systems in complex, real-time domains require agents to act effectively both autonomously and as part of a team. Focus should be placed on the problem of designing a collective of autonomous agents that individually perform sequences of actions such that the resultant sequence of *joint* actions achieves a predetermined global objective. The crucial design step in multi-agent systems centers on determining the private objectives of each agent so that as the agents strive for those objectives, the system reaches a desired global solution. Because of the inherent complexity of this type of multi-agent system, we will investigate the use of machine learning within multi-agent systems.

MAS takes care of the mechanics of executing actions controlled by an agent, passing messages between actions, coordinating multiple agents, arbitrating resource conflicts between agents, updating sensor values, and interleaving higher-level processes such as planning. When a group of agents in a multi-agent system share a common long-term goal, they can be said to form a team. Team members (or teammates) coordinate their behaviors by adopting compatible cognitive processes and by directly affecting each other's inputs via communicative actions. Other agents in the environment that have goals opposed to the team's long-term goal are the team's adversaries. The team member agent architecture within a flexible structure proposed, allows agents to decompose the task space into flexible roles and allows agents to smoothly switch roles while acting. The agents are assumed to have at their disposal the following resources:

- Inputs from the environment that give partial, noisy information;
- The ability to process the input information and use it to update a world model;
- Learning mechanisms that dynamically affect the model;
- Communication capabilities.

A MAS will also allow or even require distribution of the agents among either remote or co-located computer systems. The distributed system will allow for great flexibility in the processing power of the simulation, since computational tasks can be spread across multiple PC's. In an instructional setting, each command center could be run from a separate PC in geographically diverse locations.

An advantage of a distributed MAS is that each agent or groups of agents may be executed on different computers and in different operating systems (OS). Additional agents can also be added into the system without modifying existing agents. This creates a system that is more robust and less prone to failure. Agent classification is object

oriented which allows the sub-classing of agents to match the hierarchical classifications of the agent families. The methodology of our design is to divide a large task into a lot of sub-tasks to allow each sub-task to be solved by different agents. This creates an easier programming task by allowing a greater number of less complex programs to be written.

One of the major issues in defining an action set for an agent, and, arguably, one of the major issues in defining any kind of intelligent behavior is the problem of forming abstractions. No agent designer will want to specify the solution to a given problem in terms of primitive low-level actions and sensations. Instead, the designer will first build more powerful abstract actions, which encode solutions to a range of problems, and use these actions when faced with a new problem. Our MAS should support abstraction by providing the mechanisms to construct a hierarchy of actions. In the hierarchy, abstract actions are defined in terms of simpler ones, ultimately grounding out in the agent's effectors. The very lowest level of the hierarchy consists of very primitive actions, like move or apply-force. Although actions are abstract at higher levels of the hierarchy, they are nonetheless executable. At the same time, the hierarchy implements a multi-level computational architecture, allowing us, for example, to have both cognitive and reactive actions within the same framework. This means that higher levels should provide goals and context for the lower levels, and lower levels provide reports and messages to the higher levels (goals down, knowledge up). A higher level cannot overrule the information provided by a lower level, nor can a lower level interfere with the control of a higher level.

The term "plan" is used to denote an action that satisfies a goal. More specifically, an activity plan usually begins with a system at some initial state, specifies some desired final or goal state, and identifies constraints on the allowable sequence of actions. Usually, military planning is a part of a five-stage process:

- Mission analysis
- Intelligence preparation of the battlefield
- Development of courses of action
- Analysis of courses of action
- Decision and execution

These steps rely on a detailed and extensive knowledge base of the domain, environment, enemy and friendly capabilities. Planning is necessary when the goal is satisfied by several actions and we have to decide between them. A planner's effectiveness is determined by the ability to cope with the complexities of a continuous, dynamic, real-time domain. The planner's distinguishing feature is that it evaluates plans by efficiently simulating ahead in a more abstract space.

Learning Approaches

There are three main learning approaches that are being used in our research: Machine Learning (ML), Reinforcement Learning (RL), and Artificial Neural Networks (ANN). Implementation of Machine Learning mechanisms is a promising area to merge with the inherent complexity of multi-agent systems. Central to the process of learning, is the adaptation of behavior in order to improve performance. ML has the potential to provide robust mechanisms that leverage upon experience to equip agents with a large spectrum of behaviors, ranging from individual performance in a team, to collaborative achievement of independently and jointly set high level goals. Using hierarchical task decomposition, multiple ML modules can be combined to produce more effective behaviors than a monolithic ML module that learns straight from inputs and outputs.

The approach will break the problem down into several behavioral layers and use ML techniques when appropriate. Given hierarchical task decomposition, layered learning allows updates at each level of the hierarchy, with learning at each level directly affecting learning at the next higher level. Starting with low-level behaviors, the process of creating new behavior levels and new ML subtasks continues towards high level strategic behaviors that take into account both teammate and opponent strategies.

A learning component acquires its competence from different sources and in different ways:

- *Observing and imitating the successful actions and decisions of other agents*
- *Receiving positive and negative feedback from the user and higher command*
- *Receiving explicit instructions from the user*
- *Communicating and obtaining advice from other agents in the*

Reinforcement Learning (RL) represents the second alternative for learning in our MAS. This is the branch of machine learning that is concerned with an agent who periodically receives “reward” signals from the environment that partially reflect the value of that agent's private utility function. The goal of an RL algorithm is to determine how, using those reward signals, the agent should update its action policy to maximize its utility.

The maturing field of Reinforcement Learning provides much-needed mechanisms for model free and “online” learning features. It is ideally suited for the distributed environment where a “teacher” is not available and the agents need to learn successful strategies based on “rewards” and “penalties” they receive from the overall system at various intervals. As the number of agents

increases, the effects of any agent's actions will be swamped by the effects of other agents (noise), making the agent unable to learn well, if at all. In addition, agents cannot be used in situations lacking centralized calculation and broadcast of the single global reward signal. Complexity of synchronization and coordination increases exponentially. The problem is that the space of possible action policies for such systems is too big to be searched. (Chen et al. 2000, Mehdi, et al. 2002)

Artificial neural networks are a third promising approach in MAS learning. They provide a robust statistical learning process in noisy, uncertain, and dynamically changing environments, and therefore a possible solution for learning in war games. Many applications have shown that these networks have sufficient computational power to approximate a very large class of nonlinear functions; non-linearity is one of the main characteristics of complex military systems. Therefore, artificial neural networks offer great potential and power for developing intelligent agents with the inductive learning component based on previous experience. At the same time, these models are also very difficult to integrate into existing military applications. One of the important reasons and disadvantages of this approach is the requirement for a large number of cases (massive experience) to support significant improvements in the learning process. One method that can be used to compensate for these disadvantages is the use of Case Based Reasoning. (Pal, et al. 2001; Kolonder 1993)

Case Based Reasoning

Military operations have a very strong theory and historical record. In domains with strong experience another advantage of case-oriented techniques is their ability to learn from historical cases. Gathering these cases may improve the systems ability to find suitable similar cases for current problems. Therefore, the knowledge of experts does not only consist of formalized rules and procedures, but of a mixture of doctrinal knowledge and experience. The arguments for case-oriented methods of learning in are as follows:

- Reasoning with cases corresponds with the training process of military commanders.
- Incorporating new cases means automatically updating parts of the changeable knowledge.
- Textbook knowledge and experience can be clearly separated in a knowledge base, but used together in solving new cases.

The essential benefit from the CBR approach for our system is that the methodology can be applied with a small, or limited amount of experience and incrementally develop the

performance adding more cases to the case base as they become available. The main argument is that users of our system, even the experts, may not have enough or correct historical knowledge/experience in every situation.

Most of the previous inductive learning methods require a significant amount of cases and situations to build the agent's knowledge. Therefore, it was decided to design learning agents in our system using Case Based Reasoning (CBR) methodology. Some of the characteristics of a domain that indicate that a CBR approach might be suitable include:

- Records of previously solved problems exist or they will be acquired.
- Historical cases are viewed as an asset that ought to be preserved.
- Remembering previous experiences is useful especially in a new non-established domain.
- Experience is at least as valuable as textbook knowledge.

Case based learning models are simple yet surprisingly successful in providing extremely good prediction for human behavior in a variety of military applications. Furthermore, the case-based approach provides a more complete account of learning phenomena than rule-based, neural network or reinforcement learning. Essentially, learning occurs in case-based models through storage of a multitude of experience with past problems. New problems are solved through the retrieval of similar past problems. However, it is very important to select the right assumptions about the retrieval of past examples if we want learning with appropriate quality. Each historical case is represented as a point in n-dimensional space, and multidimensional scaling is necessary to treat all dimensions with the same weights in the comparison process. On the other side, based on experience or formalized previous knowledge, some features should receive more weight or attention in the learning process. The similarity (S_i) between new case C and old one x_i (stored in the knowledge base of cases) is assumed to be inversely related to the distance d:

$$S_i = \exp(-d(x_i, C)^n)$$

Evidence for one hypothesis, such as “the presence of a military threat”, is computed by summing of all of the activated samples that share the hypothesis:

$$E_1 = \text{sum}(S_i)$$

Evidence for alternative hypothesis, for example “the absence of a threat”, is computed for alternative samples:

$$E_2 = \text{sum}(S_j)$$

The final decision is based on a comparison of the evidence for each hypothesis, and for example the possible parameter is a probability of choosing the first hypothesis E_1 over the second E_2 :

$$P = E_1 / (E_1 + E_2)$$

While a flat case base is a common structure in most of the CBR applications, a hierarchical structure that stores the cases by grouping them can reduce the search process and increase its performance. There are no universal CBR methods suitable for every domain of application. The challenge in CBR for military wargames is to create methods that are suited for problem solving and learning in particular subject domains and for particular application environments.

Although case-based models have proven to be highly successful, there are some mainly theoretical problems we have to be aware of. These problems will be analyzed and solutions will be proposed in the implementation phase. Also, despite the obvious potential to the gaming world, it must be used carefully to avoid certain pitfalls such as:

- *Mimicking Stupidity* - copying a human strategy that is taught badly
- *Overfitting* - taught a certain section of a problem with a lot of details, and then expected to display intelligent behavior based on its local experience for the entire problem
- *Local Optimality* - a non-optimal solution is reached, in which any small change cannot improve performance of the system
- *Past Behavior* - the behavior that has been successful for the learning process in the past, is not useful any more

The practical approach for the learning process is to combine CBR mechanisms with methods of rule-based agent design. CBR retrieval is used to search for similar cases to support evidences for rule-based decisions obtained by other agents in a distributed system. A CBR part and a rule-base are applied in parallel, the results and the coordination of further steps is handled by meta-rules. Further investigation should be conducted with the implementation of a prototype of a Distributed Case-Based Learning System for multi-agent systems. In such a system, each of the agents has a partial and imperfect view of the problem-solving situation. This gives rise to a need for the agents to cooperatively access their case-bases to retrieve the “best sub-cases,” and to support or to revise their decisions and actions. A specialized learning agent will have a task to

coordinate all these activities with a case base in a consistent manner, and to provide cases for other agents that are most useful for the present problem-solving situation.

Particle Swarm Technology

Strategic level games have traditionally used a variety of methods from a simple roll of the dice to complex game engines to resolve conflicts of opposing game pieces. This often results in players trying to “game” the system instead of trying to anticipate enemy actions as in a real conflict. Our architecture will use agents as major commands, but will explore the use of particle swarm technology for conflict resolution. Particle swarm allows a large number of individual pieces to be controlled by their commanding intelligent agent and use relatively simple rules for moving around terrain and enemy engagement. Resolution of competing goals within an adversarial domain occurs through a series of individual and group actions. An agent structure can model the command levels and carry out the planning functions of those headquarters, but the management of agents is not optimal for resolution of the conflicting goals. Resolution will be done at the smallest level possible because that is a reflection of the real world that we are attempting to model.

Particle swarm algorithms are goal oriented in which the swarm is given a goal and each particle finds its path toward that goal. These goals are often thought of as static, but they can also be dynamic. The authors have experimented with particles detecting the changing edge of a cloud as it is moved by the wind. This experimentation provides a basis for the use of particles for conflict resolution in a dynamic environment. Under the guidance of its controlling agent, particles can have their immediate objective change based on what they encounter in their environment. Each particle will initially have simple rules to follow upon encountering a particle or group of particles from an opposing swarm. Each particle will have a value that represents its combat power and will be compared against the values of opposing particles. Within a defined engagement area, particles with the higher local combat power will proceed toward their main objective while those with the lower local value will be removed from the game board. The complexity of the rules and values of the particles can increase to take into account such factors as the level of supply, moral, experience, armament as well as a variety of other human and logistical factors.

Particle Swarm Optimization (PSO) defines each particle as a potential solution to a problem in D- dimensional space. Thus:

- Particle “i” is represented by: $X_i = (x_{i1}, x_{i2}, \dots, x_{iD})$

- Memory of “i”s” previous best position: $P_i = (p_{i1}, p_{i2}, \dots, p_{iD})$
- Velocity of “i” along each dimension: $V_i = (v_{i1}, v_{i2}, \dots, v_{iD})$

A new position for the particle is computed for each iteration by combining the P vector of the particle with the best fit for the local area designated “g” and the P vector of the current particle to adjust the velocity. The cognition component is the portion of the velocity influenced by the individual particle’s previous best position.

Our experience with the variety of swarm models indicates that the Full Model with variations may be of greatest use. Modification to vary each unit along with the different classes of particle groups will provide a unique conflict resolution tool. Minar et. al. has conducted research regarding swarm and agent technologies similar to what we are proposing. We have also experimented successfully with the applications of particle swarm algorithms in imaging and more recently in robot mapping of hazardous environments. (Hardin, et. al, 2004)

CURRENT STATE OF AUTHORS’ RESEARCH

Research and development activities that pertain to the development of this research has been completed to two related key areas:

- A MAS used for medical decision making
- A historically based strategic board game

A demonstration system for medical decision-making has been completed and tested. This provides the foundation to the agent architecture detailed above.

Research has also been completed on a historically based strategic board game. This research provides the historical validation foundation for a computer simulation. It also provides a game template that can provide a basis for the first round of computerization. Historical decisions, conditions and constraints can be used to “replay” history to test the accuracy of the system. The research team, with direction from the United States Air Force, has selected the Tunisian campaign of 8 November 1942 through 13 May 1943 as the template. This topic was chosen because:

With these two initial steps completed, research can be conducted in adding a learning component into the MAS, and computerizing the historical scenario with the intelligent MAS architecture.

CONCLUSIONS

A concept of Third Generation Wargames is advocated using intelligent multi agent systems (MAS). The dynamic characteristics of military operations lend themselves well to a multi-agent system. The value of Case Based Reasoning along with learning mechanisms for (artificial) agents (which can be embedded in modern military hardware) is tremendous. The exploration of the use of swarm technology for the resolution of competing goals is an innovative and efficient method of modeling the large numbers of individuals and weapon systems on the battlefield. The benefits of a completed system is the ability to train leaders more effectively to meet the demands of an increasingly smaller military decision cycle. Providing realistic wargaming tools for military planners further aids shortening the decision cycle and creating more accurate plans and contingencies for military operations. (Miller 1997)

ACKNOWLEDGEMENTS

Support for the research that resulted in this paper has been provided from the National Aeronautics and Space Administration (NASA) with a Phase I STTR grant to Intellas and the University of Louisville, and from the US Air Force with a Phase I SBIR grant.

REFERENCES

Anderson, D., Belknap, M., Cui, X., Elmaghraby, A., Jacobi, D., Kantardzic, M., Ragade, R. (2002), "A Distributed Agent Architecture for Human Operations on Space," for the International Society for Computers and their Application 11th International Conference on Intelligent Systems on Emerging Technologies, Boston, 2002.

Anderson, D., Belknap, M., Cui, X., Elmaghraby, A., Jacobi, D., Kantardzic, M., Ragade, R. (2002), "Computer Assisted Space Medic," World Space Congress, Space Ops 2002, Houston, Texas, 2002.

Pew, Richard and Mavor, Anne, editors, *Modeling Human and Organizational Behavior: Application to Military Simulations*, National Academy Press, 1998.

Sycara, K., "Multiagent Systems," *AAAI AI Magazine*, Vol. 19, No. 2, 1998.

Chen, J.R., Wolfe, S.R., and Wragg, S.D. "A Distributed Multi-Agent, System for Collaborative Information Management and Sharing," *Proceedings of the 9th ACM International Conference on Information and Knowledge Management*, 2000, 382-388.

Davis, P. K. "Distributed Interactive Simulation in the Evolution of DoD Warfare," *Modeling and Simulation. Proceedings of the IEEE*. Vol 83, No 8, 1995.

Herz J.C. and Michael R. Macedonia, "Computer Games and the Military: Two Views," *Defense Horizons*, April, 2002.

Kolodner J., *Case-based Reasoning.*, Morgan Kaufmann, San Mateo, 1993.

Lenz M. et al., *Foundations to Applications, Case-Based Reasoning Technology*, Springer, Berlin, 1998, pp. 273-297.

Mehdi, Q., Gough, N., Sulliman, H., "Virtual Agent Using a Combined Cognitive Map and Knowledge Base System," for the International Society for Computers and their Application 11th International Conference on Intelligent Systems on Emerging Technologies, Boston, 2002.

Mehdi, Q., Gough, N., Wen, Z., "A New Approach for Animating Intelligent Agents in Complex 3D Virtual Environment Based on Spatial Perception and Memory," for the International Society for Computers and Their Application 11th International Conference on Intelligent Systems on Emerging Technologies, Boston, 2002.

Miller, C.B., "USAF TACS Battle Management: Preparing for High Tempo Future Operations," <http://www.fas.org/irp/eprint/milis.htm>, 1997

Pal S. K., Dillon T. S., Yueng S. Y., *Soft Computing in Case Based Reasoning*, Springer-Verlag, London, 2001.

Palmer N., *Machine Learning in Games Development*, <http://ai-depot.com/GameAI/Learning.html>

Smith, R. D., *Military Simulation Techniques & Technologies*. Distributed Simulation Technology, 1998

Stone P., *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer* Chapter Layered Learning in Multi-Agent Systems , MIT Press , 2000.

Weiss, Gerhard, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, 2000.

HUMAN MOTION FOR VIRTUAL PEOPLE

Adam Szarowicz¹ and Jaroslaw Francik^{2,*}
Kingston University

¹School of Maths, ²School of Computing and Information Systems
Penrhyn Road, KT1 2EE
Kingston, United Kingdom

E-mail: {a.szarowicz, jarek}@kingston.ac.uk

KEYWORDS

Animated avatars, Q-learning, motion prototyping

ABSTRACT

While computer animation is currently widely used to create characters in games, films, and various other applications, techniques such as motion capture and keyframing are still relatively expensive. Automatic acquisition of secondary motion and/or motion prototyping using machine learning might be a solution to this problem. Our paper presents an application of the Q-learning algorithms to generate action sequences for animated characters. The techniques can be used in both deterministic and non-deterministic environments to generate actions which can later be incorporated into more complex animation sequences. The paper presents an application of both deterministic and non-deterministic updates of the Q-learning algorithm to automatic acquisition of motion. Results obtained from the learning system are also compared to human motion and conclusions are drawn.

INTRODUCTION

This paper presents learning techniques for animated characters that mimic human behaviour, especially in the context of interaction with physical objects. Anthropomorphic characters have been used before to simulate object manipulations, including interactions between individuals (Tomlinson et al, 2000, Russel and Blumberg, 1998) but this did not include action learning. A good example of an architecture addressing the problem of action learning is C4 (Isla et al, 2001). C4 tackled the problem of learning on the cognitive level – characters (usually four-legged creatures) learn how to respond to new commands and events. An extension of C4, which includes much greater

learning capabilities, is described in Blumberg et al (2002). The applied learning algorithm is a modification of the reinforcement learning technique (Sutton and Barto, 1998). However the task of the learning engine is not to learn the necessary motor skills but rather is defined on a higher level and happens in real-time during the interaction with the system. The system uses a so-called pose-graph to generate motion, the nodes of which are derived from source animation amended by an interpolation technique. Thus the animation is realistic and transitions can be generated in real-time but the actions must be prepared by an animator and pre-programmed into the system. Another example of applying reinforcement learning (RL) to animation includes Yoon et al (2000), where RL techniques were used to create motivational and emotional states for a human character. This system incorporates such concepts as motivation driven learning, organisational and concept learning but not motor learning. Similarly as before the learning occurs on a higher level and only affects the character's behaviour in an indirect way.

These systems visualise motion of the characters using a blend of motion-capture, keyframing and kinematics-based techniques. Quite often, however, motion is generated using dynamic simulation. This allows to create characters with very complex motor skills. Terzopoluos and his colleagues (Terzopoluos *et al* 1996) present a system for animating dynamically simulated fish and snakes. They employed machine learning to acquire complex motor skills for the simulated fish. The virtual characters are able to learn low-level motions and also high-level behaviours. In this approach physics-based simulation was used, based on a dynamic model of the fish with muscles and springs. However a similar approach applied to dynamic simulation of human figures requires that the characters have many

* Work supported by European Commission as a part of Marie-Curie fellowship.

degrees of freedom thus making it computationally expensive. Despite that complication a lot of research is being conducted in this field. Hodgins et al (1995) propose controllers for three different athletic behaviours. Apart from dynamic simulation they also use state machines, techniques for reducing disturbances to the system introduced by idle limbs, and inverse kinematics. Van de Panne and others (van de Panne et al, 2000) propose a limit cycle algorithm for the animation of a walking biped and a dynamic motion planner for simplified characters (Acrobot, Luxo). Anderson and Pandy (1999) investigated realistic simulation of human gait using a 23 degree-of-freedom model.

There have been few attempts to build dynamic controllers, which could control more than one specific motion. Examples of these are the ones proposed by Pandy and Anderson (1999) who tried to create a controller applicable to both jumping and walking behaviours and also the work presented by Faloutsos and his colleagues (Faloutsos, 2002), who combined several different controllers and additionally applied Support Vector Machines (see Christianini and Shawe-Taylor, 2000) to automatically learn preconditions of different dynamic actions as an off-line process. An alternative to physically-based simulation was proposed by Lee et al (2000) by implementing a system in which constraints imposed on motion of a character are calculated in a procedural way. Thus the calculations are faster and more stable and can easily be used in real-time applications. Metoyer and Hodgins (2000) presented a framework for rapid crowd motion prototyping, where simplified bipeds are playing American football. Additionally their agents can learn high level behaviours from real data using a memory-based learning algorithm.

Classic reinforcement learning has been applied to create successful board games implementations (Thrun, 1995), with unmanageable state spaces. Backgammon is the most successful example (Tesauro, 1994). Reinforcement learning has also been used in robotics to control one or more robotic arms (Davison and Bortoff 1994, Schaal and Atkeson, 1994), Sutton (1996) successfully applied RL to various optimisation tasks. Recently Tedrake and Seung presented a reinforcement learning technique for expanding a controller for the planar one-legged hopping robot (Tedrake and Seung, 2002). Solutions based on the Q-learning algorithm (Watkins,

1989) have also been modified and adapted. Examples include ant systems (Monekosso et al, 2002) or reward shaping (Ng et al, 1999) a technique in which additional rewards are used to guide the learning. A survey of reinforcement learning techniques can be found in Kaelbling et al (1996) and Touzet (1999).

Animation prototyping is a topic which has recently gained much popularity in the animation research community. Rapid prototyping techniques offer an opportunity to quickly sketch an animation sequence without need for a fully simulated motion. Fang and Pollard (2003) proposed a system for fast generation of motions for characters having from 7 to 22 degrees of freedom using physical simulation. Another recent system for creating and editing of character animation based on motion capture were presented, among others, by Dontcheva et al (2003) and Lee et al, 2002. The generated results are comparable to recorded human motion. Li with his colleagues (Li et al, 2002) described a system for synthesis of complex human motion (dancing) from motion captured data. The system learns so called motion textons (repetitive patterns in complex motion) and their distributions and can synthesise new motion. A similar concept was introduced by Liu and Popovic (2002).

In order to create believable characters, both the physical and cognitive aspects of an avatar must be implemented - or some variants thereof (Funge, 1999, Isla et al, 2001, Szarowicz and Forte, 2003). Modeling learning agents also includes a more or less complex structured environment where the characters thrive (Monzani, 2002). Realistic environments are usually implemented by imposing internal and external physical constraints, such as gravity, obstacles and body limitations (for example the limited movements of body limbs).

THE AVATAR MODEL

The used avatar model borrows its biomechanical characteristics from robotics. An avatar has a set of joints whose movements can be either prismatic (movements constrained on a 3D plane) or revolute (movements involving a rotation about an axis in 3D space). Then the kinematics of manipulators (Craig, 1989) rules all possible movements of joints as combinations of prismatic and revolute elemental movements. A standard goal usually includes more

or less complex object manipulations. In fact, using forward and inverse kinematics for a simple but articulated avatar the optimal sequence of simple actions fulfilling a goal can indeed be learnt (Szarowicz and Remagnino, 2004).

Learning is implemented by creating a suitable state space and applying reinforcement learning techniques to learn the optimal movements to reach an object of interest. Figure 1 illustrates the concepts of forward and inverse kinematics and the current model of the avatar.

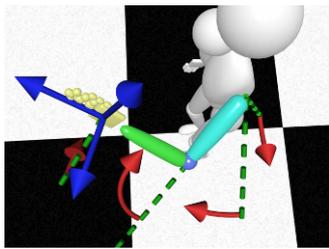


Figure 1 Position of the end effector can easily be calculated when all joint rotations are given (forward kinematics), the opposite task is the problem of inverse kinematics.

REINFORCEMENT LEARNING FOR AUTONOMOUS AVATARS

The avatars can perform a number of actions. The standard way of adding new actions and behaviors (seen as compositions of actions) to an avatar repertoire is to manually script them. Ideally, an avatar should allow for new actions but should also have a form of automatic generation of new actions. The reinforcement learning technique lends itself very well to the automatic acquisition of actions and behaviors. The implemented avatars use the Q-learning technique. All standard reinforcement learning techniques, and Q-learning in particular, do assume a scene evolving along a discrete time line, indicated by the t variable. A suitable state space is defined as well as all available actions for each defined state. The reader should refer to (Watkins, 1989, Sutton and Barto, 1998) for a detailed discussion on reinforcement learning techniques.

The quality of an action is kept up to date either using a table of quality values $Q(s_t, a_i)$ or a neural network (Bertsekas and Tsitsiklis, 1996) or more stable alternatives (for instance Baird and Moore, 1999). Results on both deterministic and non-deterministic approaches are described in the next sections. In all experiments the state space and the goals of the agent are explicitly defined. The Q-

learning was implemented by discretising the space into states and using a Q-table. The following list describes more details of the current implementation (see also Szarowicz et al, 2005, Szarowicz et al, 2003):

- An avatar can perform a number of simple actions including arm, forearm and hand motion illustrated in Figure 2 and textually described in Table 1.

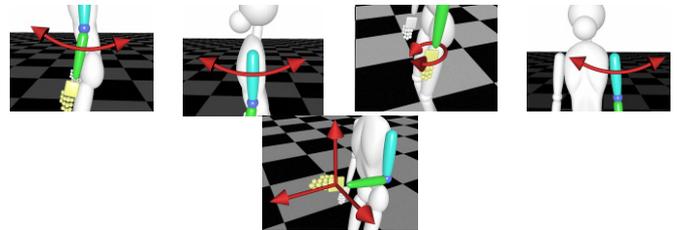


Figure 2 Avatar degrees of freedom for the teapot task: FK (first four) and IK (last) control

- The state space is different for each mode of control but in both cases it is discretised defining a number of degrees of freedom for the used joints. In the case of forward kinematics the degrees of freedom of the arm were defined as rotations around spatial axes (see first four illustrations of Figure 2), all rotations were discretised and constrained to realistic physical movements. In the case of inverse kinematics the discretisation is performed on the 3D space location of the end effector of the avatar, that is its hand (see last illustration of Figure 2).

- In both forward and inverse kinematics, walking along one dimension is considered as an additional action. Discretisation here is implemented along one axis in the two main directions of the ground plane, where the avatar moves. Similarly grabbing an object is considered to be an additional action.

- Other external objects (such as the door shown in the experiments) were represented as additional variables.

- For each possible state space dimension there are always two possible actions, indicating a movement of a body part (i.e. an arm, a forearm, a hand etc.) along such dimension in the two opposite directions. Examples include the avatar walking forward and backwards or moving its hand along the vertical axis resulting in lowering or raising the hand.

- Successful fulfilment of a goal is rewarded, collisions with the environment and violence of the biomechanical constraints are punished.

Although Q-learning convergence is not affected by the initial state, for optimisation reasons all animation experiments were biased towards a realistic starting state (ie with the avatar upright and both arms aligned with the body).

Table 1 Low-level actions used to train the avatar:

| Forward kinematics | Inverse kinematics |
|--|-------------------------------------|
| 1. Rotate arm up/down by $\Delta\alpha$ | Move palm by Δx |
| 2. Rotate arm forward/backward by $\Delta\alpha$ | Move palm by Δy |
| 3. Rotate forearm by $\Delta\alpha$ | Move palm by Δz |
| 4. Rotate hand along Z axis by $\Delta\alpha$ | |
| 5. Rotate shoulder along Z by $\Delta\alpha$ | |
| 6. Perform a grabbing action | Perform a grabbing action |
| 7. Move forward/backward by Δx | Move forward/backward by Δx |

LEARNING TASKS

Two example learning tasks are presented below, for more details the reader is referred to Szarowicz and Remagnino (2004).

The door opening task

For this task the goal of the agent was to get through a locked door. The door would be unlocked upon touching the door handle. The avatar would then have to push the door and pass through it. The agent was rewarded whenever its position was behind the door. The simple actions available to the agent were selected from Table 1 (actions 1,2,3,4,5,7), for the FK, α was set to 20 degrees, step size (Δx in action 7) was 35 cm, in all experiments $\gamma = 0.95$. Eight simple actions were available to the agent at each time step. These were three rotations – two for the arm and one for the forearm – in two opposite directions and walk along one ($2*3+2$). The task of the IK-controlled experiment was the same as for the FK previous one but the mode of control and the state and state-action spaces were changed. The simple actions available to the agent were 1,2,3 and 7 (Table 1, inverse kinematics column), $x = 35$ cm for walk (the size of a single step) and $\Delta x = \Delta y = \Delta z = 5$ cm for the motion of a hand, $\gamma = 0.95$. Therefore the agent could choose from 8 simple actions - hand motion along 3 spatial axes in two opposite directions for each axis plus walk ($2*3+2$).

The teapot lifting task

The goal here was to lift a teapot (z co-ordinate of the teapot position had to increase). Therefore the agent was rewarded whenever the end position of the teapot was higher than the start position. The simple actions available to the agent were selected from Table 1, for the FK these were actions 1,2,3,4,6,7. The learning parameters were set as follows: α was set to 10 degrees and $\gamma = 0.95$. The dimensionality of the task was 5 - 2 degrees of freedom for the left arm, 1 for the left forearm, 1 for

hand rotation and 1 for the state of the teapot. Ten simple actions were available to the agent at each time step (2 for each state-space dimension, as described earlier). An experiment with biped control using inverse kinematics was also conducted. The simple actions available to the agent in this case were actions from Table 1 (actions 1,2,3,7 of the inverse kinematics column), and $\Delta x = \Delta y = \Delta z = 8$ cm for the motion of a hand, $\gamma = 0.95$. Therefore the state-space was 4-dimensional and the agent could choose from 8 simple actions - hand motion along 3 spatial axes in two opposite directions for each axis plus the grabbing action. In all experiments the Q-table was represented as a lookup table and the values were initialized to 0 before the simulation.

LEARNING USING THE NON-DETERMINISTIC ALGORITHM

This section presents results obtained when applying the non-deterministic update of the Q-learning algorithm to the task of action acquisition. The task implemented using this technique is the IK-controlled teapot problem. The state space is same as in the deterministic implementation, and the length of the shortest solution obtained is also the same (10 simple actions). The convergence is reached faster – in approximately 800 interactions as opposed to about 3000 in the deterministic case (Figures 3 and 4) and the time necessary to reach the optimum solution is shorter as well – about 90 minutes on average (550 iterations). The convergence is also more stable (Figure 3). This suggests that the non-deterministic version of the algorithm generates comparable results in a shorter amount of time.

LEARNING WITH NON-DETERMINISTIC ACTION SELECTION

An additional simulation with the non-deterministic update has also been executed, in which the outcome of the action selection was randomised in some percentage of cases. The action selected by the agent according to its Q-table was replaced with a random action with some probability. The results of that simulation for different levels of action randomisation are presented in Figure 5.

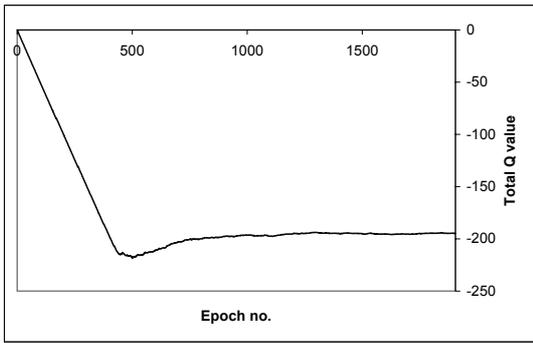


Figure 3 Convergence graph for the IK teapot non-deterministic problem

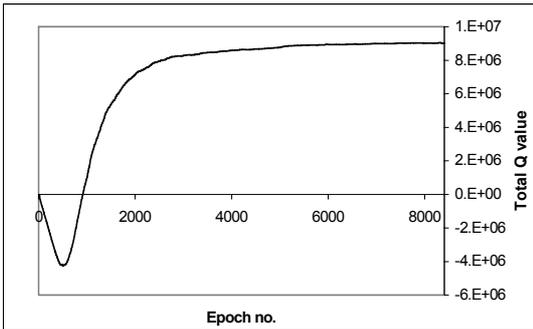


Figure 4 Convergence graph for the IK teapot deterministic problem

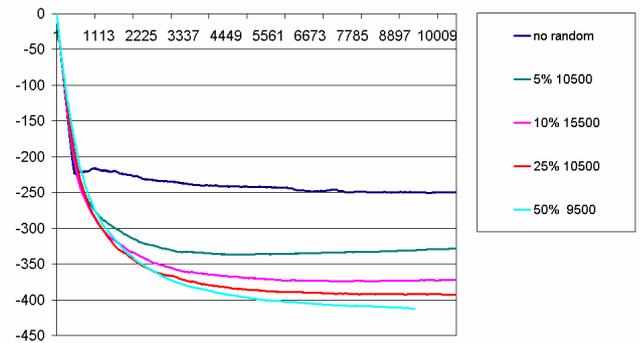


Figure 5 Convergence for randomised action selection updates

and in this case the motion can also be compared to human performance. Lower values of α (20 degrees) for the FK teapot task generate motion which is too jerky and inaccurate. Some artefacts are still present however, even in the quality motion for the FK problem, this mainly concerns unnecessary motions and especially a zigzag-like way of approaching the teapot which is present in some animations, but the result is resembling human motion with a sufficient detail as demonstrated in Figures 7 and 8. Indeed the way of executing the action achieved using the FK mode of control matches the way of executing the same action by a human actor without giving her additional guidelines prior to performing the task (Figures 7 and 8).

As presented in Figure 5 the speed of convergence is decreased with the growth of the uncertainty of the action selection mechanism. However, convergence is still reached, even for relatively high uncertainty levels. Although the results of this experiment do not have much significance in a fully predictable animated landscape, they suggest possible utilisation of the action acquisition scheme for robotic environments.

EVALUATION OF THE LEARNING RESULTS

The results obtained from applying the learning mechanism indicate that the IK learning mode is faster and easier to implement. The convergence is reached in a smaller number of iterations, compared to the FK case, and is more pronounced. However the ultimate assessment can only be made upon analysing the resulting animations. For the simpler door problem the generated motion resembles human actions to a large extent (Figure 6). Experiments with the teapot task have shown that a relatively low resolution of the state space discretisation is sufficient to generate believable result ($\alpha=10$ degrees)

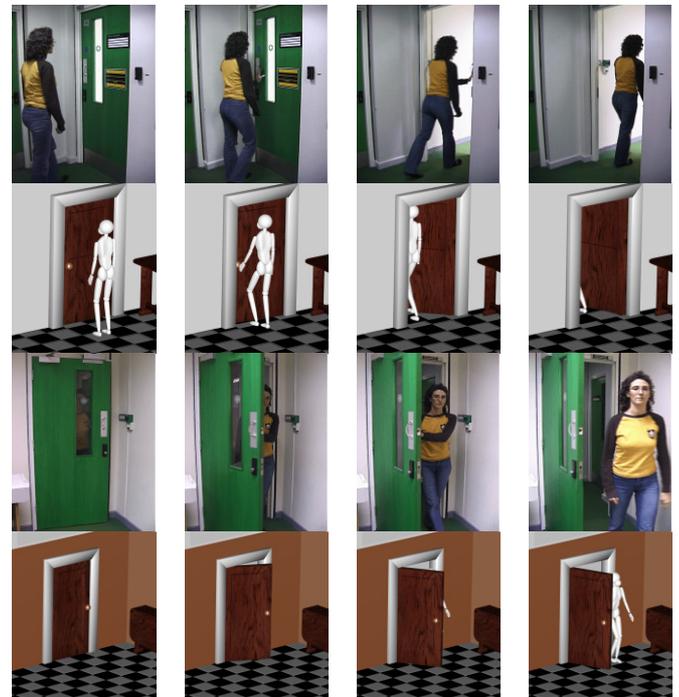


Figure 6 Motion generated in the Door experiment compared to human motion

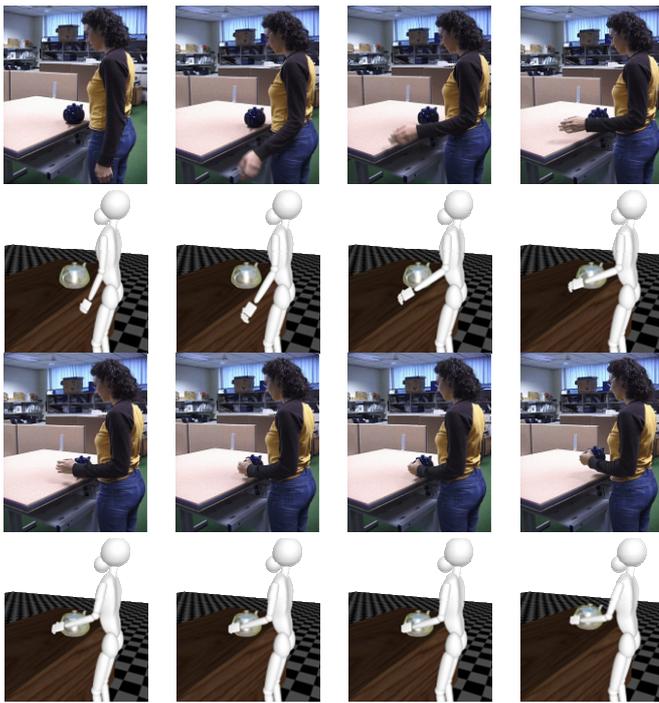


Figure 7 FK controlled motion compared to motion of a human actor

substantially increase the number of iterations required to find a solution. Therefore tasks for which more than 6-7 degrees of freedom is necessary may have to be simulated using the more compact IK control.

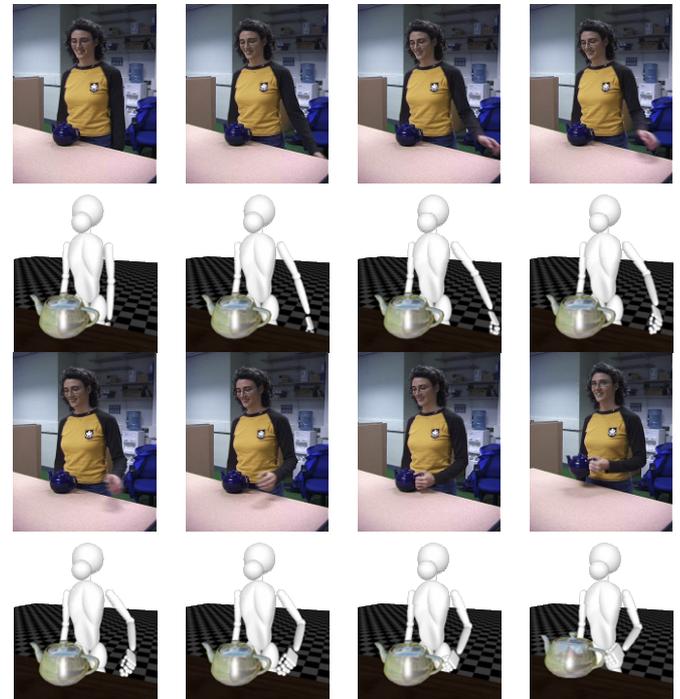


Figure 8 FK controlled motion compared to motion of a human actor

Results yielded by the IK controlled experiments (both deterministic and non-deterministic) are also interesting. First of all the state space is substantially smaller than for the FK experiments and therefore the solutions are found in fewer iterations. The resulting motion looks realistic, despite the fact that the human actor did not initially perform the action the way suggested by the IK solution. This does not mean humans can not perform the lifting task in this way as demonstrated in Figures 9 and 10, and the reason for this way being less natural is only in the fact that the table was relatively high. Reducing the height of the table changes the way of performing the task by humans (the hand does not have to be moved around the table). Moreover, the generated motion still looks natural, and contains fewer unnecessary artefacts compared to the FK solution, because IK control implicitly rejects some of the unnecessary moves. The IK state space can be represented in a more compact way (only three values need to be stored regardless of the hand position). This however causes problems when more expressive motion or combination of different modes of control are required (for the door opening task it was necessary to combine IK hand control and walking), as the representation of the state space for such extensions is more uniform when using the FK approach. The main problem with FK approach is its extensibility – additional degrees of freedom very quickly expand the state space and

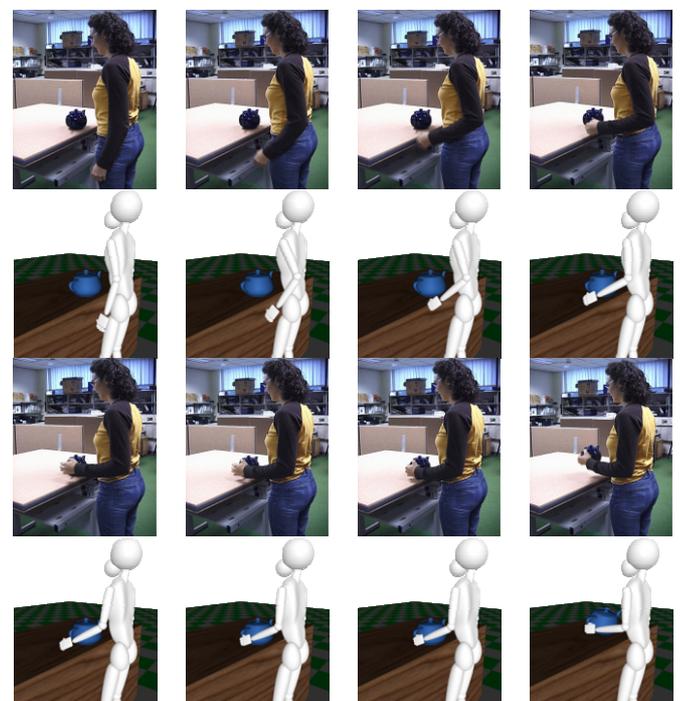


Figure 9 IK controlled motion compared to motion of a human actor

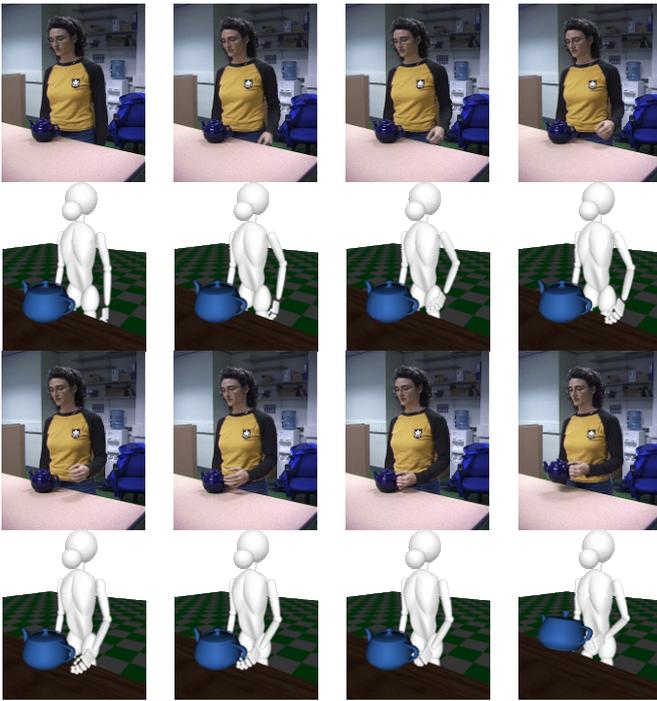


Figure 10 IK controlled motion compared to motion of a human actor

It also appears that the non-deterministic algorithm generates the solution faster than the deterministic one, maintaining the same quality of the results. Future implementations therefore should rely on this version of the Q-learning technique.

CONCLUSIONS

In summary, the learning technique presented here generated satisfactory results when applied to a non-trivial task. Comparison of the generated motion to the motion of a human actor indicates that the sequence is sufficiently realistic to be applied in an animation system mimicking human behaviour. Although the technique appears to be averagely scalable, some extensions, especially using the IK control and neural networks for state space approximation, will be possible to it, allowing to add a few additional degrees of freedom to simulate a task requiring the use of both hands or the head motion by the simulated biped. Additionally results obtained with a better hardware configuration suggest that a modern computer will improve the learning times by at least one order of magnitude.

REFERENCES

Anderson F. C. and Pandy M. G., Three-Dimensional Computer Simulation Of Gait, Bioengineering Conference Big Sky, Montana, June 16-20, 1999

Baird L. C. and Moore A. W., Gradient descent for general reinforcement learning, in Advances in Neural Information Processing Systems 11, eds., M. S. Kearns, S. A. Solla, and D. A. Cohn, Cambridge, MA, MIT Press, 1999

Bertsekas D.P. and Tsitsiklis J.N., Neuro Dynamic Programming, Athena Scientific, 1996 Craig J.J., Introduction to Robotics: Mechanics and Control, Addison Wesley, 1989

Blumberg B., Downie M., Ivanov Y., Berlin M., Johnson M. P., Tomlinson B., Integrated learning for interactive synthetic characters, ACM Transactions on Graphics, Vol. 21, Iss. 3, pp417-426, July 2002

Christianini N. and Shawe-Taylor J., Support vector machines and other kernel-based learning methods. Cambridge University Press, 2000

Davidson D. E. and Bortoff S. A., Acrobot software and hardware guide, Technical Report Number 9406, Systems Control Group, University of Toronto, Toronto, Ontario M5S 1A4, Canada, June 1994

Dontcheva M., Yngve G., Popovic Z., Layered Acting for Character Animation, Proceedings of the ACM SIGGRAPH San Diego, California, USA, 2003

Faloutsos P., Composable Controllers for Physics-Based Character Animation, Ph.D. Thesis, Department of Computer Science, University of Toronto, 2002

Fang A. C. and Pollard N. S., Efficient Synthesis of Physically Valid Human Motion, ACM Transactions on Graphics 22(3) pp417-426, SIGGRAPH 2003 Proceedings, 2003

Funge J. D., AI for Games and Animation. A Cognitive Modeling Approach, A K Peters Natick, Massachusetts, 1999

Hodgins, J. K., Wooten, W. L., Brogan, D. C., O'Brien, J. F., Animating Human Athletics, Proceedings of Siggraph '95, In Computer Graphics, pp 71-78, 1995

Isla D., Burke R., Downie M., Blumberg B., A layered brain architecture for synthetic creatures, in Proceedings of Seventeenth Joint Conference on Artificial Conference IJCAI-01, pp. 1051-1058, Seattle, USA, 2001

Kaelbling L.P., Littman M.L., Moore A.W., Reinforcement learning: A survey, Journal of Artificial Intelligence Research, vol.4 pp.237-285, 1996

Lee J. W., Baek N., Kim D., Hahn J. K., A Procedural Approach to Solving Constraints of Articulated Bodies, Eurographics 2000 Short Presentations Programme, Interlaken, Switzerland, August 20-25, 2000

Lee J., Chai J., Reitsma P. S. A., Hodgins J.K., Pollard N.S., Interactive Control of Avatars Animated With Human Motion Data, Proceedings of the 2002 ACM SIGGRAPH, San Antonio, Texas, USA, 21-26 July 2002

- Li Y., Wang T., Shum H.-Y., Motion Textures: A Two-Level Statistical Model for Character Motion Synthesis, Proceedings of the 2002 ACM SIGGRAPH, San Antonio, Texas, USA, 21-26 July 2002
- Liu K.C. and Popovic Z., Synthesis of Complex Dynamic Character Motion From Simple Animations, Proceedings of the 2002 ACM SIGGRAPH, San Antonio, Texas, USA, 21-26 July 2002
- Metoyer, R. A., Hodgins, J. K., Animating Athletic Motion Planning By Example. Proceedings of Graphics Interface 2000, pp. 61-68, Montreal, Quebec, Canada, May 15-17, 2000
- Monekosso N.D., Remagnino P., Szarowicz A., An Improved Q-Learning Algorithm Using Synthetic Pheromones in From Theory to Practice in Multi-Agent Systems, Lecture Notes in Computer Science, vol. 2296 Edited by Dunin-Keplicz, B. and Nawarecki, E., Springer-Verlag, pp. 197, March, 2002
- Monzani J.-S., An architecture for the Behavioural Animation of Virtual Humans, Ph.D. dissertation, Ecole Polytechnique Fdrale de Lausanne, 2002
- Ng A. Y., Harada D., Russell S., Policy invariance under reward transformations: Theory and application to reward shaping, Proceedings ICML-99, Bled, Slovenia, 1999
- Pandy M. G. and Anderson F. C., Three-Dimensional Computer Simulation Of Jumping and Walking Using the Same Model, in Proceedings of the VIIth International Symposium on Computer Simulation in Biomechanics, August 1999
- Russell K. B. and Blumberg B., Behavior-friendly graphics, in Computer Graphics International, pp. 44-, 1999
- Schaal S. and Atkeson C., Robot juggling: An implementation of memory-based learning. Control Systems Magazine, 14, 1994
- Sutton, R. S., Generalization in reinforcement learning: Successful examples using sparse coarse coding, In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference, pp.1038-1044, Cambridge, MA, MIT Press, 1996
- Sutton R.S. and Barto A. G., Reinforcement Learning: an Introduction, MIT Press, 1998
- Szarowicz A. and Forte P., Combining intelligent agents and animation, in AIxIA 2003 - Eighth National Congress on AI, Lecture Notes in Artificial Intelligence, vol 2829, Pisa, Italy, Springer-Verlag, 2003
- Szarowicz A., Mittmann M., Remagnino P., Francik J., *Automatic Acquisition of Actions for Animated Agents*, 4th Annual European GAME-ON Conference, November 19-21, London, United Kingdom, 2003
- Szarowicz A., Remagnino P., *Avatars That Learn How to behave*, European Conference on Artificial Intelligence ECAI 2004, Springer, Valencia, Spain, 2004
- Szarowicz A., Francik J., Mittmann M., Remagnino P., *Layering and Heterogeneity as Design Principles for Animated Embedded Agents* in International Journal of Information Sciences, to appear, Elsevier, 2005
- Tedrake R. and Seung H. S., Improved Dynamic Stability using Reinforcement Learning, Proceedings of the International Conference on Climbing and Walking Robots (CLAWAR'02), 2002
- Terzopoulos D., Rabie T., Grzeszczuk R., Perception and Learning in Artificial Animals, Artificial Life V: Proc. 5th Inter. Conf. on the Synthesis and Simulation of Living Systems, Nara, Japan, 1996
- Tesauro G., TD-Gammon, a self-teaching backgammon program achieves master-level play. Neural Computation, 6(2) pp.215-219, 1994
- Thrun S., Learning to play the game of chess. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, Advances in Neural Information Processing Systems 7, MIT Press Cambridge, MA, 1995
- Tomlinson B., Blumberg B., Nain D., Expressive autonomous cinematography for interactive virtual environments, in Proceedings of the Fourth International Conference on Autonomous Agents, Barcelona, Spain, 2000
- Touzet C. F., Neural Networks and Q-Learning for Robotics, IJCNN '99 Tutorial, 1999 International Joint Conference on Neural Networks, Washington, DC - July 10-16, 1999
- van de Panne M., Laszlo J., Huang P., Faloutsos P., Dynamic Human Simulation: Towards Agile Animated Characters, Proceedings of the IEEE International Conference on Robotics and Automation 2000, pp. 682-687, San Francisco, CA, 2000
- Watkins C. J. C. H., Learning from Delayed Rewards, Ph.D. dissertation, Cambridge, Psychology Department, 1989
- Yoon S.Y., Blumberg B. M., Schneider G. E., Motivation driven learning for interactive synthetic characters. In Proceedings of Autonomous Agents 2000

IAGENT: A REAL TIME INTELLIGENT AGENT ANIMATION TOOLKIT

Zhigang Wen, Quasim Mehdi, Norman Gough
School of Computing and IT
University of Wolverhampton
35/49 Lichfield Street, Wolverhampton
UK
Email: Z.Wen@cs.bham.ac.uk

KEYWORDS

Behavioural animation, Motion control, Emotion, DirectX.

ABSTRACT

This paper describes the design and implementation of IAgent: a real time intelligent agent animation toolkit on a PC platform. The animation system consists of 5 main components, namely environment, perception, behaviours, motion generator and rendering. The intelligent agent in the system is represented as a 3D human-like avatar that has a complex underlying structure with multiple degrees of freedom (DOFs). The agent relies on a fast virtual perception system to capture information from its environment and a behaviours system to determine what actions should be taken. A novel motion generation architecture and animation blending system have been developed to produce non-repetitive behaviours for the intelligent agent based on its momentary goal, internal and emotional states. The proposed system has been implemented in DirectX. Experiments have been carried out using the toolkit and the results have clearly demonstrated that the method produces convincing real time behaviours for a 3D virtual human agent.

INTRODUCTION

Today software titles demand ‘smarter’ participants in the simulated virtual world. For instance, in the entertainment industry like PC games, the Non-Player-Characters (NPCs) controlled by the computer are expected to exhibit convincing behaviours in respond to dynamic change of the environment and human player’s activities. The major problem for characters in those real time applications is that the character often performs its motion in the same way resulting in repetitive and unrealistic behaviours. Furthermore, the creation of human like virtual intelligent agent for real time applications presents even more challenges due to the complex underlying structure of the character. The main objective of this paper is therefore to design and implement an innovative intelligent agent animation system that integrates research efforts from several fields, notably computer graphics and animation and artificial intelligence, to animate a complex and realistic human like virtual agent in a 3D environment on a PC platform.

The virtual agent has a complicated underlying hierarchical skeleton for producing real time motion according to its perception from environment and behaviours module. The final rendering of the agent is implemented by using the skinned mesh algorithm to achieve smooth skin deformation

effect. The proposed animation system has the mechanism to dynamically generate realistic agent’s motions based on a hybrid method that combines parameterized motions, kinematics and animation blending.

Section 2 describes the design of the animation system and its major functional components. Section 3 briefly illustrates the implementation and a simple animation example. Section 4 finally draws conclusions.

THE ANIMATION SYSTEM

The animation system is divided into 5 main components, namely environments, perception, behaviours, motion generator and rendering module. Figure 1 shows high-level functional architecture of the framework.

The *virtual environment* normally contains geometrical objects, sound objects, and events. Geometrical objects refer to 3D objects with vertices and texture maps, which are detectable by the agent’s virtual vision. In a complex virtual environment containing large number of 3D objects, spatial partition techniques are useful to arrange these objects in some kinds of hierarchy that accelerate the agent’s objects detecting process (Mehdi et al. 2002). Geometrical objects are normally “seen” by the agent with additional properties being memorized such as the location of the object, time of being detected, ID of the object etc. Sound objects in the environment can be detected by the agent’s virtual audition sensor, subsequently affecting the agent’s behaviours.

The environment contains another special object called events. Events can generally be divided into two categories, namely functional events and dynamics events. Functional events can be considered as the “built-in” behaviours of the objects and they can be triggered by under some conditions. For instance, a door in a scene can be opened or closed by the agent. Dynamics event is generally generated in related to agent’s activities. This type of event will have significant impact on the agent’s emotional state, which will subsequently affect the agent’s behaviours or the way the agent execute its behaviours. For instance, the result of an agent’s attempt to capture some objects will cause agent become happier or angrier. One of the novelties of the proposed animation system is that it has the mechanism to visualize the subtle change of agent’s emotional states via its animated behaviours.

The *Behaviours module* contains several key functional components to perform action selection for the agent, namely decision-making, emotion, personality, memory, and internal states. The decision-making component relies on a hierarchical

action network to select actions for agent according to its perceptual information, internal states, memory and goals. Emotion plays an important role in creating believable agent behaviours (Bates 1994; Blumberg 1994). Psychological and neuroscience research indicates that emotions have a significant impact on human behaviours, both through their use as a non-verbal communication channel such as gesture, posture, facial expression and so on (Oatley and Johnson-Laird 1987). It is therefore important to incorporate emotion into our

$\beta(p, \omega_t)$ is calculate the decay of emotional sates based on personality and emotional state history.

Emotion in the developed animation system is primarily used to decide how the selected behaviours will be executed depending on agent's momentary emotional states, which is realized by the *Motion Generation module* described in the following contexts (Figure 2).

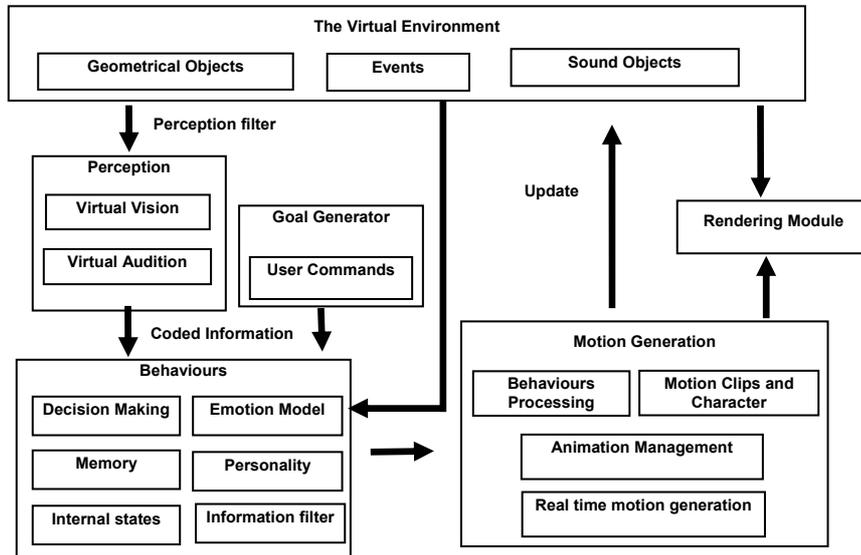


Figure 1 Architecture of animation system

animation system. Emotion model in the system is based on the OCC model (Ortony et al.1998). The OCC model specifies how events, agents and objects are appraised according to respectively their desirability, praiseworthiness and appealingness, which are defined by a set of parameters such as goals and attitudes. The process of integrating OCC model into agent behaviours can be divided into 4 steps, namely classification, quantification, interaction, and mapping (Bartneck 2002).

It is believed that personality and emotion have significant influences on behaviours and how behaviours are expressed (Marsella and Gratch 2002; Ball and Breese 1998). Different personality model has been studied in the psychology community such as the OCEAN model (Costa and McCrae 1992). This model has five dimensions, namely openness, conscientiousness, extraversion, agreeableness, and neuroticism. The link between personality and OCC emotion model is described in (Egges et al. 2004). Their idea is essentially to construct a *Personality-Emotion Influence Matrix*, in which indicates how each personality factor influences each emotion. A simple emotion update equation as proposed in (Egges et al. 2004) is therefore used in the animation system.

$$e_{t+1} = e_t + \alpha(p, \omega_t, a) + \beta(p, \omega_t)$$

Function $\alpha(p, \omega_t, a)$ is to calculate the changes of the emotional state based on personality p , emotional state history ω_t , and emotion influence a from OCC model.

The *Motion Generation module* decomposes those behaviours into low-level motions with control parameters that can be realized by the motion clips library. The *Motion Coordinator*, upon receiving motions with parameters, retrieves base motions from the motion clips library and joints from the character skeleton that are required to perform the motions. Base motions are primarily produced by motion interpolation. On top of these base motions, parameterized motions are generated based on outputs from the *Emotion to Motion*

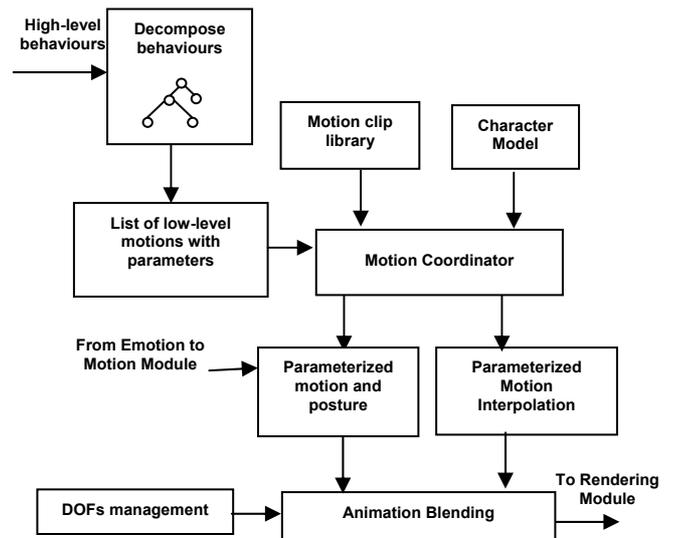


Figure 2 Architecture and functionalities of the motion generation module

Module. Such outputs are normally motion control parameters that are used to alter the way the base motions are animated or parameters to change the posture of the agent. All these

generated motions are finally blended together to produce the final motions for the agent. The DOFs management assure that motion blending does not violate the limits of character's joints therefore avoiding un-natural motion. The functionalities of this *Motion Generation* module can be extended by incorporating new developed components. For instance, a physically-based modelling based motion module can be incorporated to increase the motion realism. In such cases, the adjustments (e.g. the update to involved DOFs) from this module can be blended into the existing motions via animation blending and DOF management.

Rendering module is responsible for displaying both of character animation and the virtual world onto the screen. It receives animation requests from the *Motion Generation module* and activates corresponding animation procedures with control parameters. The major challenge of designing such module is to achieve a balance between visual realism and the controllability of the animated 3D agent. The skinned mesh animation algorithm is used in the system (Wen et al. 2002).

IMPLEMENTATION AND RESULTS

The system was implemented in DirectX with MFC on a PC platform. The initial 3D human model contains around 6000 polygons and has around 23 degree of freedoms. The agent, for simplicity and experimental purpose, has 1 internal state (*Energy*) and two emotional states (*Happy*, *Angry*). The agent has base motions, namely *Walking*, *Running*, and *Resting*. It

As shown in Figure 4, the events of failure or success to capture the object is evaluated and used to update the agent's emotional states, which subsequently resulting in different way of exhibiting its motions. The *Motion Generation module* produces parameterized motions and subsequently blends into the existing base motion animation sequences to achieve non-repetitive behaviours without the need to explicitly model such different style of animations in advance in modelling packages. The user is able to alter the agent's internal and emotional states through the user interface, observing the instant change of agent behaviours. Frame rate of the above simulation achieves an average of 100 based on a machine with *Pentium IV 2.8 GHz* CPU and a *Geforce 2 MX 400* graphics card. The graphics card used for the simulation is obsolete and current graphics hardware can deliver much better rendering performance.

CONCLUSIONS AND FUTURE WORK

This paper has presented an integrated system to animate human-like virtual intelligent agent. The agent has the ability to capture information from its environment and determine what actions should be taken based on its *behaviours module*. An innovative *Motion Generation module* is developed to realize the selected behaviours and produce parameterized motion along with pre-generated animation sequence depending on agent's momentary emotional states. Furthermore, as the *Motion Generation module* has the control of the character model to a degree of freedom level and a

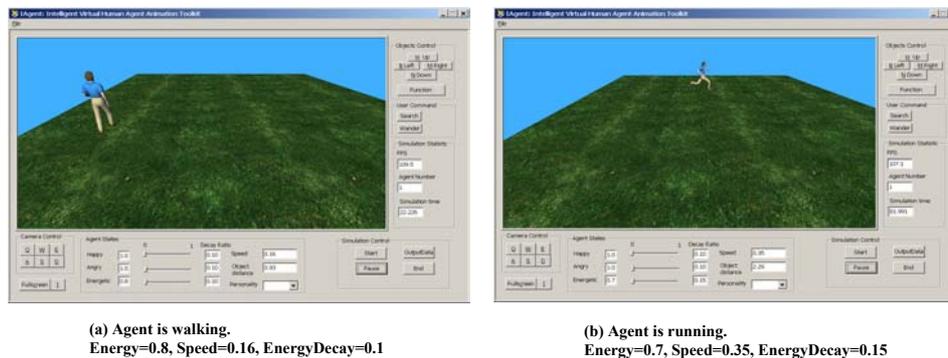


Figure 3 Agent in wandering state (emotional states is not taking effect)

also has three parameterized motions, namely *HeadMovement()*, *PostureChange()*, and *ArmMovement()*. Its motions are arranged into a hierarchical actions network. Agent changes its base motions based on the fuzzy internal state *Energy*. Parameterized motions are generated based on the two emotional states, *Happy* and *Angry*. Figure 3 shows that the agent is in wandering state. In this state, the agent only exhibits its base motions and emotional states are not being updated.

Figure 4 shows that the agent is in searching state. In this state, the agent is given the task to capture a object controlled by the user. Its emotional states are updated based on dynamic events in the environment. In this case, the failure or success to capture the object has impact on his two emotional states. Emotion states are also updated according to time passing. This simulation used a default personality profile.

flexible animation blending component, it is possible to integrate various existing motion planning, kinematics, and physically based motion into the framework in order to increase the realism of the human agent's motions. Future work can be enhanced by investigating the relationship between agent's mental states and body motions (e.g. posture, gesture) and how emotions can be expressed by such body motion.

REFERENCES

Ball, G. and Breese, J. 1998. "Emotion and Personality in a Conversational Character". In *Proceedings of the Workshop on Embodied Conversational Character* (California, USA, Oct.). 83-84.

Bartneck, C. 2002. "Integrating the OCC Model of Emotions in Embodied Characters." In *Proceedings of the on Virtual*

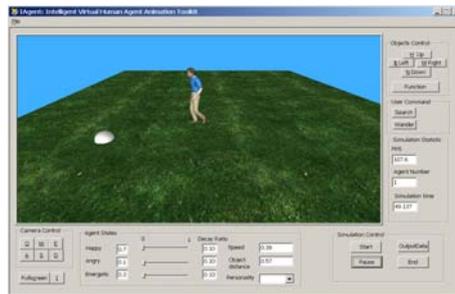
Conversational Characters: Applications, Methods, and Research Challenges (Melbourne, Australia).

Bates, J. 1994. "The Role of Emotion in Believable Agents." *Communications of the ACM* 37, no.7: 122-125.

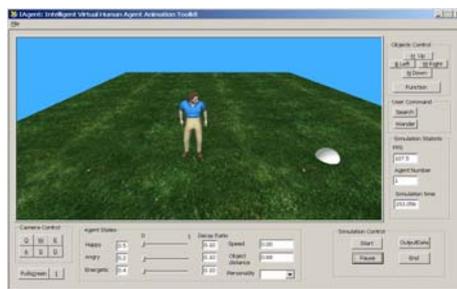
Mehdi, Q., Wen, Z. and Gough, N. 2002 "A new approach for animating intelligent agents in complex 3D virtual environments based on spatial perception and memory", In *Proceedings of 11th International Conference in Intelligent Systems* (Arlington, USA, Jun. 13-15).47-50.



(a) Agent is trying to capture the object
Happy=0.9, Angry=0.1, Energy=0.9



(b) Agent changed from running to walking due to insufficient energy. Agent's happiness decreases when time passes but has not captured the object. PM *PostureChange()* starts to be active but is still not obvious. This function has the control parameters "body lean angle" that is affected by emotional state *Happy*. Happy=0.7, Angry=0.1, Energy=0.3



(c) Agent is in resting mode. The agent has to stop to rest once the energy level drops to a low level. As the agent is in searching mode, he is still trying to locate the object by moving his head and body. Parameterized motions *HeadMovement()* and *BodyRotation()* are generated and blended into the base motion resting. The *HeadMovement()* has a number of control parameters such as rotation speed and rotation amplitude that are affected by emotional state *Angry*. The level of *Angry* increases when the agent expects to capture the object but subsequently fails to do so. The agent will rotate more rapidly with higher anger level. Happy=0.5, Angry=0.2, Energy=0.4



(d) Agent's happiness is at a very low level. The value of the "lean angle" parameter for the *PostureChange()* is significant and the agent looks "really sad and disappointed". Happy=0.2, Angry=0.5, Energy=0.4

Figure 4 Agent in searching mode (emotional states influence action)

Becheiraz, P. and Thalmann, D. 1998. "A Behavioral Animation System for Autonomous Actors personified by Emotions." In *Proceedings of First Workshop on Embodied Conversational Characters* (Lake Tahoe, California, Oct. 12-15). 57-65.

Blumberg, B. 1994. "Action-selection in Hamsterdam: Lessons from Ethnology." In *Proceedings of SAB'94* (Brighton, UK, Aug.8-12).108-117.

Costa, P. T. and McCrae, R. R 1992. "Normal Personality Assessment in Clinical Practice: The NEO Personality Inventory." *Psychological Assessment* 4:5-13.

Egges, A., Kshirsagar, S. and Magnenat-Thalmann, N. 2004. "Generic Personality and Emotion Simulation for Conversational Agents." *Computer Animation and Virtual Worlds* 15, no.1 (Jan.): 1-13.

Marsella, S. and Gratch, J. 2002. "A Step Toward Irrationality: Using Emotion to Change Belief." In *Proceeding of First International Joint Conference on Autonomous Agents and Multi-Agent Systems* (Bologna, Italy, Jul. 15-19). 334-341.

Oatley, K. and Johnson-Laird, P.N. 1987. "Towards a Cognitive Theory of Emotions." *Cognition and Emotion* 1, no.1:29-50.

Ortony, A., Clore, G.L. and Collins. 1988. *The Cognitive Structure of Emotions*. Cambridge University Press, UK.

Wen, Z., Mehdi, Q. and Gough, N. 2002. "A New Animation Approach for Visualizing Intelligent Agent Behaviours in a Virtual Environment." In *Proceedings of the Information Visualization* (London, UK, Jul.). IEEE.93-98.

BIOGRAPHY

Zhigang Wen currently works as a research fellow in the *eDrama* project at the School of Computer Science, University of Birmingham. His main research interests include real time virtual human animation, gesture generation system and natural language processing for 3D embodied conversational agent.

INTERACTIVE OPPONENTS GENERATE INTERESTING GAMES

Georgios N. Yannakakis and John Hallam
The Maersk Institute for Production Technology
University of Southern Denmark
Campusvej 55, 5230 Odense M, Denmark
E-mail: {georgios;john}@mip.sdu.dk

KEYWORDS

On-line Learning in Computer Games, Prey/Predator, Interactive Opponents, Interest Emergence.

ABSTRACT

In this paper we present experiments on neuro-evolution mechanisms applied to predator/prey multi-character computer games. Our test-bed is a computer game where the prey (i.e. player) has to avoid its predators by escaping through an exit without getting killed. By viewing the game from the predators' (i.e. opponents') perspective, we attempt off-line to evolve neural-controlled opponents, whose communication is based on partial implicit information, capable of playing effectively against computer-guided fixed strategy players. However, emergent near-optimal behaviors make the game less interesting to play. We therefore discuss the criteria that make a game interesting and, furthermore, we introduce a generic measure of this category of (i.e. predator/prey) computer games' interest (i.e. player's satisfaction from the game). Given this measure, we present an evolutionary mechanism for opponents that keep learning from a player while playing against it (i.e. on-line) and we demonstrate its efficiency and robustness in increasing and maintaining the game's interest. Computer game opponents following this on-line learning approach show high adaptability to changing player strategies which provides evidence for the approach's effectiveness against human players.

INTRODUCTION

In (Yannakakis et al. 2004), we introduced a predator/prey computer game named 'Dead End' for emerging complex and cooperative behaviors among agents through evolutionary procedures. In

this game the prey (i.e. player) has to avoid its eight predators (i.e. *Dogs*) by escaping through an exit without getting killed. Since there are eight *Dogs* on the game field, they are designed to be slower than the Player so that the game is fairer to play. This game's fundamental concepts are inspired from previous work of Yannakakis et al. (2003) where efficient cooperative behaviors, supported only by partial implicit communication, emerge amongst the agents of a complex multi-agent environment.

Similar to Luke's and Spector's (1996) work on the *Serengeti* world, we view Dead End from the predators' perspective. Our first aim is to emerge effective complex teamwork behaviors by the use of an off-line training approach, based on evolutionary computation techniques, applied to homogeneous neural controlled agents (Yao 1999). *Dogs* have to demonstrate good cooperative strategies in order to kill the Player and/or to defend the Exit. Such behaviors can be aggressive, defensive, or a hybrid of the two. Given the specific game, we believe that 8 predators are enough for cooperative behaviors to emerge.

However, playing a computer game like Dead End against well-playing opponents with fixed hunting behaviors cannot be regarded as interesting. The first stage of experiments on this test-bed, given an implicitly defined notion of interest, is presented in (Yannakakis et al. 2004). We believe that the interest of any computer game is directly related to the interest generated by the opponents' behavior rather than to the graphics or even the player's behavior. Thus, when 'interesting game' is mentioned we mainly refer to interesting opponents to play against. In (Yannakakis and Hallam 2004), we argue that the interest measure proposed (for the well-known Pac-Man game) defines a generic measure of any predator/prey game. Results obtained from Dead End and

presented here give evidence for this interest measure's generality, which defines one of the goals of this work.

We present a robust on-line neuro-evolution learning mechanism capable of increasing the game's interest (starting from well performing behaviors trained off-line) as well as maintaining that interest at high levels as long as the game is being played. In our Dead End predator/prey computer game we require *Dogs* to keep learning and constantly adapting to the player's strategy instead of being opponents with fixed strategies. In addition, we explore learning procedures that achieve good real-time performance (i.e. low computational effort while playing).

Recently, there have been attempts to mimic human behavior off-line, from samples of human playing, in specific virtual environments. In (Thurau et al. 2004) among others, human-like opponent behaviors are emerged through supervised learning techniques in a first person shooter console game. Even though complex opponent behaviors are emerged, there is no further analysis on whether these behaviors contribute to the satisfaction of the player (i.e. interest of game). In other words, researchers hypothesize --- by looking at the vast number of multi-player on-line games played daily on the web --- that by generating human-like opponents the player gains more satisfaction from the game. This hypothesis might be true up to a point; however, since there is no explicit notion of interest defined, there is no evidence that a specific opponent behavior generates more or less interesting games.

DEAD-END GAME

Dead End is a two-dimensional, multi-agent, grid-motion, predator/prey game. The game field (i.e. stage) is a two-dimensional square world that contains a white rectangular area named "Exit" (see Fig. 1) at the top. For the experiments presented in this paper we use the 16 X 16 cm stage presented in Fig. 1, which is divided into grid squares (of length 0.5 mm). The characters visualized in the Dead End game (as illustrated in Fig. 1) are a dark grey circle of radius 0.75 cm representing the Player and 8 light grey square (of

dimension 1.5 cm) characters representing the *Dogs*.

The relationship between the *Dogs* and the Player is mutually highly competitive. The aim of a Player is to reach the Exit, avoiding the *Dogs*. On the other hand, the aims of the *Dogs* are to defend the Exit and/or to catch the Player. In Dead End, if a Player succeeds in arriving at the Exit, this event is described as a *win*. Additionally, if a *Dog* manages to catch a Player, this event defines a *kill*. If there is neither a Player win nor a kill for a predetermined large period of time, then the outcome of the game is a *win* again. After either a win or a kill, a new game starts.

The Player moves at four thirds the *Dogs*' maximum speed and since there are no dead ends, it is impossible for a single *Dog* to complete the task of killing it. Since the Player moves faster than a *Dog*, the only effective way to kill the Player is for a group of *Dogs* to hunt cooperatively.

The simulation procedure of the Dead End game is as follows. Player and *Dogs* are placed in the game field (initial positions) so that there is a suitably

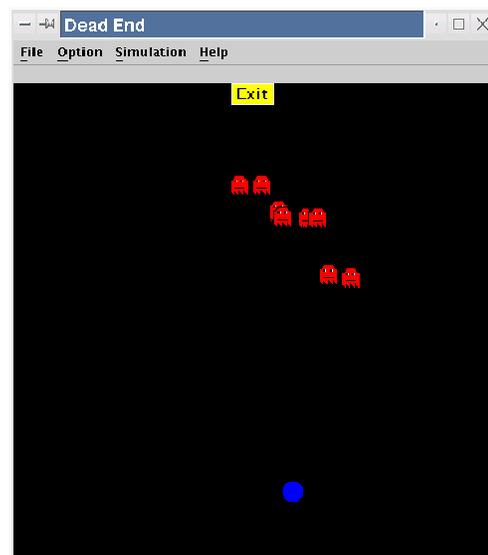


Fig. 1. Snapshot of the Dead End game

large distance between them. Then, the following occur at each simulation step. **(a)** Both *Dogs* and the Player gather information from their environment and take an individual movement decision, up, down, left or right. **(b)** If the game is

over (i.e. Player escapes through the Exit, Player is killed, or the simulation step is greater than a predetermined large number), then a new game starts from the same initial positions for the *Dogs* but from a different, randomly chosen, position at the bottom of the stage for the Player.

The Player

The difficulty of the Dead End game is directly affected by the intelligence of the Player. Its nature is significant because *Dogs*' emergent behavior is strongly related to their competitive relationship against it. To develop more diverse agents' behaviors, different playing strategies are required. We therefore chose three fixed *Dog*-avoidance and/or Exit-achieving strategies for the Player, differing in complexity and effectiveness. The non-deterministic initial position of the player is devised to provide *Dogs* with diverse examples of playing behaviors to learn from.

Randomly-moving (RM) Player

A Randomly-moving Player takes a movement decision by selecting a uniformly distributed random picked direction at each simulation step of the game.

Exit-achieving (EA) Player

An Exit-achieving Player moves directly towards the Exit. Its strategy is based on moving so as to reduce the greatest of its relative distances from the Exit.

Cost-based path planning (CB) Player

A cost-based path planning Player constitutes the most efficient *Dog*-avoiding and Exit-achieving strategy of the three different fixed-strategy types of Player. A discrete Artificial Potential Field (APF) (Khatib 1986), specially designed for the Dead End game, controls the CB Player's motion. The overall APF causes a force to act on the Player which guides it along a *Dog*-avoidance Exit-achievement path. For a more detailed presentation of the CB player, see (Yannakakis et al. 2004).

Any motion strategy that guides a Player to arrive quickly at the Exit, avoiding any *Dogs* and keeping to the straightest and fastest possible trajectory, is definitely a "good" strategy in terms of the Dead End game. Hence, the CB Player presents a "good"

behavior in this game and furthermore a reference case to compare to human playing behavior.

Neural Controlled Dogs

Artificial neural networks (ANNs) are a suitable host for emergent adaptive behaviors in complex multi-agent environments (Ackley and Littman 1992). A feedforward neural controller is employed to manage the *Dogs*' motion and is described in this subsection.

Using their sensors, *Dogs* inspect the environment from their own point of view and decide their next action. Each *Dog* receives input information from its environment expressed in the ANN's input array of dimension 6. The input array consists of the relative coordinates of (a) the Player, (b) the closest *Dog* and (c) the Exit. A *Dog*'s input includes information for only one neighbor *Dog* as this constitutes the minimal information for emerging teamwork cooperative behaviors. We deliberately exclude from consideration any global sensing, e.g. information about the dispersion of the *Dogs* as a whole, because we are interested specifically in the minimal sensing scenario.

As previously mentioned, a multi-layered fully connected feedforward ANN has been used for the experiments presented here. The hyperbolic tangent sigmoid function is employed at each neuron. The ANN's output is a two dimensional vector which represents the *Dog*'s chosen motion in X, Y coordinates.

Fixed strategy Dogs

Apart from the neural controlled *Dogs*, an additional fixed non-evolving strategy has been tested for controlling the *Dogs*' motion. *Dogs* of this strategy are called 'Followers' and they are designed to follow the Player constantly by moving at half the Player's speed (i.e. 1.0 cm/simulation step). This strategy is used as a baseline behavior for comparison with any emergent neural controller behavior.

INTERESTING OPPONENTS

In order to find, as objective as possible, a measure of interest in the Dead End computer game we first

need to define the criteria that make a game interesting. Then, second, we need to quantify and combine all these criteria in a mathematical formula. The game should then be tested by human players and have this formulation of interest cross validated against the interest the game produces in real conditions. This last part of our investigation constitutes a crucial phase of future work.

In order to simplify this procedure we will ignore the graphics' as well as the player's contribution to the interest of the game and we will concentrate on the *Dogs*' behavior that effects the game's interest. That is because, we believe, the computer-guided opponent character contributes the vast majority of features that make a computer game interesting.

By being as objective and generic as possible, we believe that the criteria that collectively define the interest of the Dead End game are as follows (see also (Yannakakis and Hallam 2004) for interest criteria definitions for the Pac-Man game).

- *When the game is neither too hard nor too easy.* In other words, the game is interesting when *Dogs* manage to kill the player sometimes but not always. In that sense, optimal behaviors are not interesting behaviors and *vice versa*.
- *When there is diversity in Dogs' behavior over the games.* That is, when *Dogs* are able to find different ways of hunting and killing the player in each game so that their strategy is less predictable.
- *When Dogs' behavior is aggressive rather than static.* That is, *Dogs* that move towards killing the player but meanwhile, move constantly all over the game field instead of simply following it. This behavior gives player the impression of an intelligent strategic *Dogs*' plan which increases the game interest.

In order to estimate and quantify each of the aforementioned criteria of the game's interest, we follow the same procedure introduced in (Yannakakis and Hallam 2004). Thus, the metrics for the three criteria are given by T (difference between maximum and average player's lifetime

over N games --- N is 50 in this paper), S (standard deviation of player's lifetime over N games) and $E\{H_n\}$ (stage grid-cell visit average entropy of the *Dogs* over N games) respectively. All three metrics are combined linearly (1)

$$I = \frac{\gamma T + \delta S + \varepsilon E\{H_n\}}{\gamma + \delta + \varepsilon} \quad (1)$$

where I is the interest value of the Dead End game; γ, δ and ε are criterion weight parameters (for the experiments presented here $\gamma = 1, \delta = 2, \varepsilon = 1$).

The measure of the Dead End game's interest introduced in (1) can be effectively applied to any predator/prey computer game (e.g. see (Yannakakis and Hallam 2004)) for a successful application on the Pac-Man game) because it is based on generic quantitative features of this category of games. These features include the time required to kill the prey as well as the predators' entropy throughout the game field. We therefore believe that (1) --- or a similar measure of the same concepts --- constitutes a generic interest approximation of predator/prey computer games. In fact, the two first criteria correspond to any computer game whereas the third criterion corresponds only to predator/prey games.

OFF-LINE LEARNING

We use an off-line evolutionary learning approach in order to produce some 'good' (i.e. in terms of performance) initial behaviors for the on-line learning mechanism. The ANNs that determine the behavior of the *Dogs* are themselves evolved (evolutionary process is limited to the connection weights of the ANN).

The evolutionary procedure is as follows. Each *Dog* has a genome that encodes the connection weights of its ANN. A population of 40 (we keep this number low because of the computational cost) ANNs (*Dogs*) is initialized randomly with initial uniformly distributed random connection weights that lie within $[-5, 5]$. Then, at each generation: **(a)** Each *Dog* in the population is cloned 8 times. These 8 clones are placed in the Dead End game field and play the game against a selected Player type for an evaluation period T

(e.g. 125 simulation steps). The outcome of this game is to ascertain the total number of wins (W) and kills (K). **(b)** Each *Dog* is evaluated via (2)

$$f = \alpha K - \beta W \quad (2)$$

where K and W are the total numbers of kills and wins respectively; α is the reward rate of a kill; β is the penalty rate of a win. **(c)** A pure elitism selection method is used where only the 20% fittest solutions are able to breed and, therefore, determine the members of the intermediate population. **(d)** Each parent clones an equal number of offspring in order to replace the non-picked solutions from elitism. **(e)** Mutation occurs in each gene (connection weight) of each offspring's genome with a small probability p_m (e.g. 0.01). A uniform random distribution is used again to define the mutated value of the connection weight.

The algorithm is terminated when a predetermined number of generations g is completed (e.g. $g=300$) and the fittest *Dog*'s connection weights are saved.

ON-LINE LEARNING

This evolutionary learning approach is based on the idea of *Dogs* that learn while they are playing against the Player. In other words, *Dogs* that are reactive to any player's behavior and learn from its strategy instead of being predictable and, therefore, uninteresting characters for game playing. Furthermore, this approach's additional objective is to keep the game's interest at high levels as long as it is being played.

Beginning from any initial off-line trained (OLT) group of homogeneous *Dogs*, the on-line learning (OLL) mechanism attempts to transform them into a group of heterogeneous *Dogs* that are interesting to play against. The OLL procedure is as follows. An OLT *Dog* is cloned 8 times and its clones are placed in the Dead End game field to play against a selected Player type. Then, at each generation:

(a) Each *Dog* is evaluated every T (T is 25 here) simulation steps via (3), while the game is played (where (x_d^k, y_d^k) and (x_p^k, y_p^k) are the cartesian

coordinates of the Player's and the *Dog*'s center respectively at simulation step k).

$$f' = \frac{1}{1 + \sum_{k=1}^T \{|x_d^k - x_p^k| + |y_d^k - y_p^k|\}} \quad (3)$$

By using (3), we individually promote each *Dog* that attempts to stay as close as possible to the Player during an evaluation period. **(b)** If the average fitness of the population is greater than a fixed threshold value then, go to **(a)** else, continue. **(c)** A pure elitism selection method is used where only the fittest solution is able to breed. The fittest parent clones an offspring that replaces the worst-fit member of the population. This offspring takes the worst-fit member's position in the game field. **(d)** Mutation occurs in each gene (connection weight) of the offspring's genome exactly as in the off-line learning algorithm.

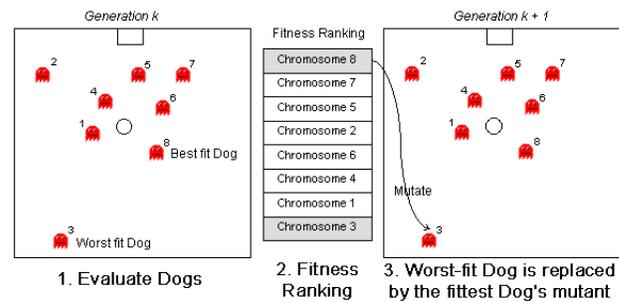


Fig. 2. The on-line learning mechanism

The algorithm is terminated when a predetermined number of generations g is completed (e.g. $g=5000$) and all 8 *Dogs*' connection weights are saved. Fig. (2) illustrates the main steps of the OLL algorithm.

We mainly use small simulation periods (i.e. $T=25$) to evaluate *Dogs* during OLL. The aim of this high frequency of evaluations is to accelerate the on-line evolutionary process. However, the evaluation function (3) constitutes an approximation of the examined *Dog*'s overall performance for large simulation periods. Keeping the right balance between computational effort and performance approximation is one of the key features of this approach. We therefore use

minimal evaluation periods capable of achieving good estimation of the *Dogs*' performance.

RESULTS

Results obtained from experiments applied on the Dead End game are presented in this section. These include, off-line and on-line learning emergent behavior analysis as well as experiments for testing robustness and adaptability of the OLL mechanism proposed.

Performance Measurement

In order to evaluate the performance of a team of *Dogs*, we record the total number of both kills K and wins W of the examined team, against a specific Player, by placing these agents in Dead End and letting them play the game for $12.5 \cdot 10^3$ simulation steps. We believe that this is a long enough period for testing a playing-behavior of a team of *Dogs* in an efficient way. This evaluation is called a *trial*. We then calculate the value $P = 100[K/(K+W)]$. This performance measurement (P) quantifies the Player-killing (K) percentage over the total number of games played ($K+W$).

Off-line Learning Experiments

The experiment presented in this subsection is focused on producing well-behaved *Dogs* in terms of the performance measure previously described. We train *Dogs* against all three fixed-strategy types of Player through the off-line learning mechanism. In this experiment we select $\alpha = \beta = 1$ in fitness function (2) --- providing equal opportunities for promoting both Player-hunting and Exit-defensive behaviors. The off-line learning experiment is described as follows.

(a) Apply the off-line learning mechanism by playing against each type of Player separately. Repeat the learning attempt (run) 10 times --- we believe that this number is adequate to illustrate a clear picture of the emergent behavior --- with different initial conditions. (b) Evaluate each one of the 10 teams of OLT *Dogs* against all three types of Player. Their performance and interest measurement are given by the average values obtained over the 10 trials. (c) Evaluate non-evolving randomly generated (i.e. untrained) as

well as Player-follower *Dogs* (i.e. Followers) against every Player type (run 10 trials and calculate their average performance and interest). The outcome of this experiment is presented in Table I.

Table I. The effect of off-line training on the *Dogs*' average performance ($E\{P\}$) and interest ($E\{I\}$) over 10 learning attempts

| | Playing against | | | | | |
|-----------|-----------------|----------|----------|----------|----------|----------|
| | RM | | EA | | CB | |
| | $E\{P\}$ | $E\{I\}$ | $E\{P\}$ | $E\{I\}$ | $E\{P\}$ | $E\{I\}$ |
| OLT/RM | 91.27 | 0.728 | 24.36 | 0.682 | 3.82 | 0.243 |
| OLT/EA | 62.55 | 0.555 | 96.01 | 0.661 | 51.27 | 0.486 |
| OLT/CB | 93.09 | 0.628 | 55.09 | 0.681 | 72.98 | 0.425 |
| Followers | 98.54 | 0.466 | 78.94 | 0.763 | 71.51 | 0.709 |
| Untrained | 75.58 | 0.401 | 62.46 | 0.498 | 17.77 | 0.425 |

As can be seen from Table I, there is a large performance improvement of the OLT *Dogs* in comparison to the untrained or even the Follower *Dogs* against all three types of Player. However, in most cases, OLT *Dogs* against a specific Player seem to get lower average performance values when playing against a Player other than the Player they have been off-line trained against. *Dogs* trained off-line against CB Players showed good overall performance against all types of Players. Therefore, among the three fixed-strategy Players, the CB Player provides the best off-line training for the opponent agents. This suggests that when *Dogs* learn from more complex and effective types of Players, they tend to generalize better.

An increased interest value when *Dogs* are trained off-line is also noticeable in all cases (see Table I). However, these emergent behaviors fail to compete the interest generated by the Followers in the majority of cases (mainly against the EA and CB Players).

The most typical emergent behaviors are pure Exit-defensive or pure Player-hunting behaviors but hybrids also occur frequently. The off-line learning mechanism, in the majority of cases, produces *Dogs* that defend the Exit and/or hunt the Player in a cooperative fashion. As stressed before, opponents in this game have to learn to cooperate in order to be successful (achieve a high performance value) against any playing strategy.

On-line Learning Experiments

The off-line learning procedure is a mechanism that attempts to produce near-optimal solutions to the problem of killing the Player and defending the Exit. These solutions will be the OLL mechanisms' initial points in the search for more interesting games. The OLL experiment is described as follows.

(a) Apply the OLL mechanism to all teams of OLT *Dogs* (see Off-line Learning Experiments section) playing against each type of Player separately. (b) Evaluate performance and interest values of each OLL attempt against each Player type. The outcome of this experiment is presented in Table II and Fig. 3.

As seen from Table I and Table II, the OLL mechanism manages to find ways of increasing the interest of the game regardless of the initial OLT behavior or the player. Due to space considerations we present only 3 out of the 9 OLL experiments in detail here. Fig. 3 demonstrates the learning mechanism's ability of producing games of higher than the initial interest as well as keeping that high interest for a long period. The mechanism demonstrated a similar adaptive behavior for all 9 different OLL experiments. This suggests that the evolutionary approach proposed shows a behavior of high robustness which furthermore manages to generate opponents' behaviors of much higher interest values.

The OLL mechanism tends to be a highly disruptive procedure (via the mutation operation) for high-interest group behaviors towards individual rewards. Such disruptive mutations can cause undesired drops in the game's interest generated by a team of *Dogs*. However, experiments show that *Dogs* trained by individual rewards (while playing) manage to maintain and even increase the game's interest.

Another important feature of the mechanism is its ability to quickly emerge interesting opponents to play against. It takes, in the worst case experienced, fewer than 500 OLL games for the mechanism to generate games of higher interest.

Table II. Best average interest values achieved by applying on-line learning on *Dogs* trained off-line. The respective average performance values are also presented

| | Playing against – On-line learning | | | | | |
|--------|------------------------------------|----------|----------|----------|----------|----------|
| | RM | | EA | | CB | |
| | $E\{P\}$ | $E\{I\}$ | $E\{P\}$ | $E\{I\}$ | $E\{P\}$ | $E\{I\}$ |
| OLT/RM | 86.73 | 0.758 | 36.45 | 0.762 | 43.09 | 0.721 |
| OLT/EA | 95.64 | 0.707 | 84.18 | 0.701 | 20.91 | 0.617 |
| OLT/CB | 97.09 | 0.685 | 53.64 | 0.745 | 60.92 | 0.610 |

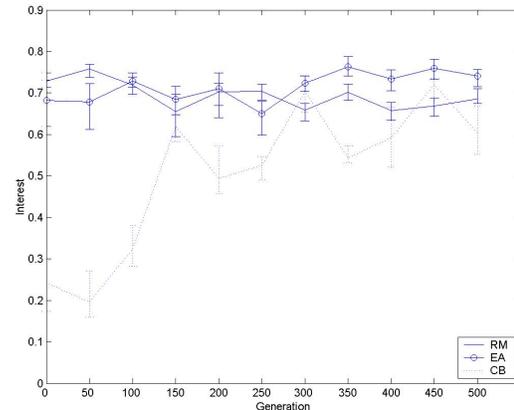


Fig. 3. Interest (averaging over 10 trials) evolution over the number of games played. Initial behavior: OLT/RM (initial and best interest values are presented in the first row of Table I and Table II respectively).

On the other hand (see Table I and Table II), in almost half cases, there is a decrease of the *Dogs*' average performance values. In general, *Dogs* that achieve high-performance values do not generate interesting games. This illustrates the tradeoff between optimality and interest in any computer game. In Dead End, optimal killing behaviors cannot produce interesting games.

CONCLUSIONS

The Dead End predator/prey computer game is devised as an interesting test-bed for studying the emergence of multi-agent cooperative behaviors supported by partial implicit communication through evolutionary learning mechanisms. We introduced an off-line learning mechanism, from which effective cooperative predator behaviors have rapidly emerged.

Predator strategies in predator/prey computer games are still nowadays based on simple rules which make the game quite predictable and, therefore, uninteresting --- by the time the player

gains more experience and playing skills. A computer game becomes interesting primarily when there is an on-line interaction between the player and his opponents who demonstrate interesting behaviors.

Given some objective criteria for defining interest in predator/prey games presented by Yannakakis and Hallam (2004) we introduced a method for explicitly measuring interest in the Dead End game. We saw that by using the proposed on-line learning mechanism (see also (Yannakakis et al. 2004)), maximization of the individual simple distance measure (see (3)) coincides with maximization of the game's interest. Apart from being robust, the proposed mechanism demonstrates fast adaptability to new types of player (i.e. playing strategies). Therefore, we believe that such a mechanism will be able to produce interesting interactive opponents (i.e. games) against even the most complex human playing strategy.

We believe that the methods used need to be tested on more complex Dead-End stages (i.e less *Dogs*) in order to provide more evidence for their generality, and the interest measure proposed needs to be cross-validated against human players. In addition, investigation of the heterogeneity's contribution on these results constitutes an important step for future work.

REFERENCES

Ackley D.H. and M.L. Littman. 1992. "Interactions between learning and Evolution." In *Artificial Life II*, C.G. Langton, C.Taylor, J.D. Farmer, and S. Rasmussen, eds. Sante Fe Institute Studies in the Sciences and Complexity. Reading, MA: Addison-Wesley, 478-507.

Khatib O. 1986. "Real-time obstacle avoidance for manipulators and mobile robots." *International Journal for Robotics Research*, vol. 5, no. 1, 90-99.

Luke S. and L. Spector. 1996. "Evolving teamwork and coordination with genetic programming." In *Genetic Programming 1996: Proceedings of the First Annual Conference*, J.R. Koza, D.E. Goldberg, D.B. Fogel, and R.L. Riolo, eds. Stanford University, CA, USA: MIT Press, 150-156.

Thureau C., C. Bauckhage and G. Sagerer. 2004. "Learning human-like Movement Behavior for Computer Games" In *From Animals to Animats 8: Proceedings of the 8th International Conference on Simulation of Adaptive Behavior (SAB-04)*, Los Angeles, USA, July 13-16, 315-323.

Yannakakis G.N. and J. Hallam. 2004. "Evolving Opponents for Interesting Interactive Computer Games." In *From Animals to Animats 8: Proceedings of the 8th International Conference on Simulation of Adaptive Behavior (SAB-04)*, Los Angeles, USA, July 13-16, 499-508.

Yannakakis G.N., J. Levine, and J. Hallam. 2004. "An Evolutionary Approach for Interactive Computer Games." In *Proceedings of the Congress on Evolutionary Computation (CEC-04)*, June, 986-993.

Yannakakis G.N., J. Levine, J. Hallam, and M. Papageorgiou. 2003. "Performance, Robustness and Effort Cost Comparison of Machine Learning Mechanisms in *FlatLand*." In *Proceedings of the 11th Mediterranean Conference on Control and Automation MED'03*. IEEE, June.

Yao X. 1999. "Evolving artificial neural networks." In *Proceedings of the IEEE*, vol. 87, no. 9, 1423-1447.

BIOGRAPHY

Georgios N. Yannakakis (Student Member, IEEE) received both the 5-year Diploma (1999) and the M.Sc. (2001) degree from the Technical University of Crete, Chania, Greece. He is a Ph.D. candidate at the School of Informatics, Centre of Intelligent Systems and their Applications, University of Edinburgh. G. Yannakakis is currently a visiting researcher at the Maersk Institute for Production Technology, University of Southern Denmark, Odense. He has published several research papers in the areas of cooperative multi-agent systems, evolutionary computation and AI in computer games.

A REVIEW OF POTENTIAL TECHNIQUES FOR THE CREATION OF INTELLIGENT AGENTS IN VIRTUAL ENVIRONMENTS

N.P. Davies¹, Dr Q.H.Mehdi¹, Prof N.E. Gough¹, D Anderson², D Jacobi², V.V Bornes²

¹RIATec
University of Wolverhampton
Wolverhampton, UK
E-mail: N.P.Davies2@wlv.ac.uk

²Intellas Modelling Simulation
LLC 1542
Beckley Hills Drive
Louisville, KY USA

ABSTRACT

This paper presents a review of potential architectures and tools for the production of Virtual Environments with Integrated Intelligent Characters. Initial research was carried out into the production of a real-time system for the creation of graphically realistic scenes for crime scene reconstruction in Davies et al (2004). The system was capable of rendering scenes produced via a graphical interface, and characters with pre-generated animation sequences could be placed and oriented in the scenes to act out crime events. It is proposed that these characters would be more beneficial if they were endowed with intelligent qualities so they could act in an autonomous manner when presented with a scenario. It is anticipated that this would produce a diverse set of actions and resulting scene disturbance which would be of benefit to forensic crime investigation students, who could theories about the events performed in the scene, and evaluate their responses against the actual events.

KEY WORDS

3D Scene Construction, Computer Graphics, Artificial Intelligence

INTRODUCTION

This paper builds on the work outlined in Davies et al (2004) which describes a graphical application for the efficient and cost effective creation of quasi-accurate three dimensional environments, which can be viewed from any position or angle in real time (Fig 1). A set of realistic objects including wall, floor and ceiling panels, and a set of common household objects including chairs and desks etc, and animated characters based on skinned mesh architectures were developed which can be placed and oriented

within the scene. Crime scenes can then be constructed from forensic information, and animated characters can populate the environments to recreate the crimes. It was anticipated that the simulations could be used by crime investigators to test and eliminate hypotheses, training purposes, and also be used in court reconstructions. The system produced some very promising results, although some limitations were identified including the implementation of animated characters where behaviour was restricted to a small set of pre-created animation sequence. No attempt was made to implement any level of intelligence; however, initial ideas were outlined for potential future development including the use of Belief Desire Intention architectures (BDI) (Bratman 1987) architectures for character intelligence. This paper outlines these ideas in more depth, and addresses the issue of character intelligence in order to develop a system that can simulate human-like behaviour in animated characters. Rather than an extended set of animation sequences, agents will be given the ability to assess and react to a set of high level tasks, and be able to relate to other agents and humans by creating, regulating and expressing emotions, while conforming to social norms and constrains. This will create a more versatile system which will be applicable to a broader set of applications in addition to crimes scene reconstruction, including entertainment e.g. computer games, scenario training and education.



Figure 1: Crime Scene Creator

AGENT THEORY

To facilitate the appearance of intelligence in a virtual environment an intelligent character (agent) is required exhibit a set of human like characteristics. This includes the capability to reason about the environment in which it is situated, combined with the ability to formulate and execute plans based on this knowledge to achieve a specified goal from a specified initial state, given definitions of the available actions. In the crime scene generator a typical scenario may as follows:

An agent leaves its house with the intention of stealing something valuable to sell at some point in the future. It evaluates its current location and reasons that a nearby neighbourhood has a good potential of having a suitable property to burgle as it is a wealthy area, and sets out in the appropriate direction. On arriving at the neighbourhood the agent begins to look for a specific property which satisfies some criteria e.g. not overlooked, appears empty, no alarm system etc. and will note several potential targets. When it has accumulated a list it will evaluate each property and conclude the most suitable target and plan a route to the house. However, on arrival it notes that a car is now in the driveway which was not there before and reasons that the house may now be occupied. With this new information it will reformulate its plans for the second most suitable target.

The above scenario suggests some of the primary capabilities required by an agent for this type of application. First: the ability to generate plans and the use of partial plans. The agent has an initial location i.e. home, and a specified goal i.e. raise money. Using these two factors it will generate a plan to make money, that is, steal something. Once this plan is conceived the agent will commit to it and will rule out other contradictory factors e.g. a friend may call asking the agent out for a drink which the agent will decline in favour of executing its plan. However, commitment may not be fixed. If a friend calls with new information such as a heavy police presence on that day the agent may drop the commitment to steal, and adopt a new commitment to go for a drink instead. The agent needs to reflect on the strength of its commitment to reason which is more appropriate in differing circumstances. The agent will also need to monitor

is environment. In the scenario above the agent commits to the task of robbing the most appropriate building, but takes note of the presence of a new car in the target property. If the agent did not monitor this change in the environment it could find itself in a negative situation e.g. a confrontation or even jail. However, the changes occurring in the environment must be filtered to outline the most relevant changes. There may be many changes happening in the environment that will have no impact on the current goal, e.g. a traffic warden is giving out parking tickets. It would be inefficient to factor in this information as it will have no effect on the overall plan and goal.

New alternative opportunities may appear that may need analysing. For instance, on the way to the burglary the agent may come across an unlocked car with an expensive stereo. The agent needs to realise that the initial goal of gaining money can be achieved much more simply by stealing the stereo, and could result in all house robbing plans to be dropped. Because the environment is dynamic, the agent will have to commit to, and begin executing, partial plans that do not contain all the information required. As the agent progresses through the plan, specific information can be elaborated upon. In the example, the agent will defer its decision about which house to burgle until it has gained enough information about the status of each property. This will not prevent the agent having an overall goal, or beginning the plan execution to reach the desired neighbourhood. The final area of interest for the crime scene application is the issue of multi-agent coordination and collaboration. For example, an agent may come across a house owner, guard dog, policeman etc. when burgling the house. This will have an impact on the characters behaviour. One agent's actions will have a direct impact on the other agent. If a guard dog attacks, they will start running, if they are confronted by the house owner, they may become violent etc.

ARCHITECTURES IMPLEMENTATION

The above outlines the theoretical requirements an agent should be endowed with in order to act intelligently. However, to achieve these behavioural characteristics on a computational

system would require a mass of power which is unfeasible based on the processing power of a typical high spec computer available today. The theory needs to be formalised into a coherent architecture under which an agent can be designed, with consideration made for optimising the architectures for a resource bound system. Traditionally, researchers have used the BDI, where behaviours are triggered by conceptually modelled intentions rather than scripted behaviour or complete animation sets (Fig 2.). The principle of this type of architecture is based on studies into human cogitative reasoning. In his publication Bratman proposed that human behaviour can be decomposed into three distinct modules. An agent will have a set of Beliefs considered to be its World View, it will also have a set of Desires it wishes to fulfil. By relating one to the other it will be able to formulate a sequence of Intentions of partial plans which it commits itself to performing in order to achieve some goal in a particular environment. Goals which are unachievable or conflict with current commitments are stripped out unless they can be recognised as potentially important to the task. To demonstrate the theories Bratman developed Intelligent Resource-bound Machine Architecture (IRMA) which utilised a means end planner and an opportunity analyser to identify new choices available and filters these choices to decide whether they are significant enough to be deliberated upon. Unfortunately, IRMA was only tested in abstract environments, which were artificially constrained test complexity. Its feasibility for real world complex, dynamic environments remains unproved.

The problems identified with this form of BDI are that the agent has to be omniscient; i.e. know all the relevant facts about its environment. This is unfeasible in real-world applications where factors may remain unknown. The actions the agent can performed have definite outcomes which again may not be the case in complex dynamic environment where there are many factors other than the agent. There is no notion of partial success. The goals are categorically either achieved or not. In real world scenarios there is a level of granularity to the definition of success. Finally, the actions performed are instantaneous state transformations. They have no temporal extent or fixed times of occurrence.

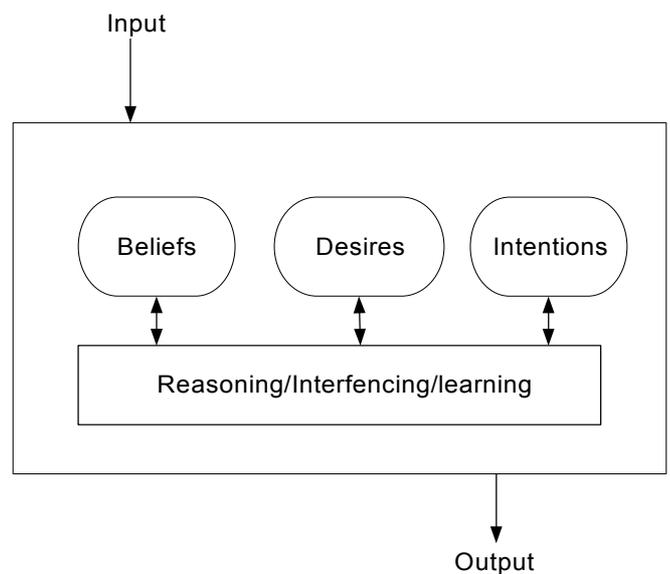


Figure 2: Simple BDI Architecture

The problems identified with this form of BDI are that the agent has to be omniscient; i.e. know all the relevant facts about its environment. This is unfeasible in real-world applications where factors may remain unknown. The actions the agent can performed have definite outcomes which again may not be the case in complex dynamic environment where there are many factors other than the agent. There is no notion of partial success. The goals are categorically either achieved or not. In real world scenarios there is a level of granularity to the definition of success. Finally, the actions performed are instantaneous state transformations. They have no temporal extent or fixed times of occurrence.

Rao and Georgeff (1995) expanded on BDI especially in the area of intentions which they base on a 'possible world' formalism to address some of these problems, and to base the architecture within a strong mathematical framework. They use a *time tree* to map an agent through a temporal simulation, with a single path leading up to node representing the current situation, and a branching future representing potential actions an agent can perform at a particular point in time. However, the resulting actions (events) are not guaranteed to have a successful outcome. The agent makes the choice, but the environment determines the result via events which transform one situation into another. Non-primitive events map to non-adjacent situations to create partial plans which can be

decomposed into primitive events to model hierarchical plan development. It is possible that an agent may attempt to execute an event, but fail to do so due to it being either impossible or due to some other event changing the environment in such a way as to make the goal invalid. This needs to be noted on the tree so it will be possible for an agent to attempt to execute events which may be unsuccessful. Rao and Georgeff base their system on logic similar to Computational Tree Logic, CTL (Emerson and Srinivasan, 1989), with state formulas, which are evaluated at a particular point in time, and path formulas which operate over a complete plan. These are treated distinctly. In a path through a time tree, an outcome can either be inevitable e.g. the result of the action will produce the same result regardless of the options taken, or optional e.g. there may be several different outcomes dependant on the course of action taken.

In the formalisation beliefs are modelled as a set of accessible worlds which the agent believes are possible to reach. This distinguishes between reachable desires, and desires that are unachievable by associating a narrow selection of applicable choices at a particular point in time. Intentions and are similarly modelled as accessible worlds which can be reachable by some course of action. This also prevents an agent committing to a goal which it unable to reach. However, it is not necessary for each state to have a corresponding intention. For example, a course of action may result in an unwanted negative side effect, however, the agent can still commit to that course of action without having to also adopt the side effect as an intention. D’Inverno (1998) tells us that Georgeff went on to develop the agent architecture Procedural Reasoning System (PRS) based on his version of BDI logic (Fig. 3), which in turn formed the foundation for distributed Multi-Agent Reasoning System (dMARS) that has to date proved to be the most successful agents application for real world applications, for example, problem solving on NASA’s Space Shuttle (Wired News 1997). In dMARS, BDI is modelled as a set operations performed on plans, and works on a cycle that first observes the world and the agents’ internal state for changes. Using this information it will update an event queue. New possible desires are generated by finding plans whose trigger event matches an event in the event queue, and a plan is selected from a set for

execution. This plan is pushed on to an intention stack dependant on whether it is a sub-goal or new task. The next step of the plan in the intention stack is then either executed, or, if the next step is a sub goal, the sub goal is expanded and placed in the event queue.

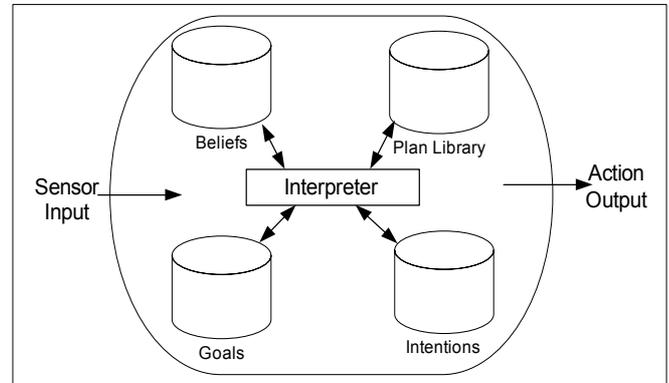


Figure 3: PRS BDI

HYBRID AGENTS

The BDI architecture has been demonstrated to be a feasible architecture for the development of intelligent agents as proved by its adoption and application. However, Roa and Georgeff (1995) claim that this type of architecture may already be dated and claim that the architecture is flawed for systems that learn and adjust their behaviour, and need to respond quickly to events in an environment. They state that better architectures such as InteRRAP (Müller and Pischel 1993) which is a layered BDI architecture that contains three hierarchical layers and a control mechanism would be better for real world applications where reaction times are paramount. In BDI systems, reactions are bounded by the reasoning and acting taking place in one execution cycle, which does not implicitly support functionality to deal with emergency situations. As an alternative they proposed a layered architecture incorporating the strong foundation of BDI logics for reasoning capabilities, and reactive layers to deal with specific situations such as obstacle avoidance. Reactions are triggered when a situation is recognised from the world model and acted on as a priority, however, longer term deliberation is carried out at a higher level based on the agents’ beliefs and goals. There is a final layer which introduces the concept of Multi Agent Systems

(MAS) where cooperation with other agents is achieved by the formulation of joint plans. As hybrid agent similar to InteRRAP has been implemented successfully by Yang and Chen (2003) for competing in a competition called RoboCup which has the ultimate aim of producing a team of intelligent robots capable of beating the World Football Champions by 2050. Actions such as kicking and dribbling are dealt with by a reactive layer, and individual and group tactics are implemented at higher levels.

Schmidt (2000) also outlines an alternative system to BDI for the modelling of human behaviour. He argues that behaviour restricted to simply belief, desires and intention is not appropriate to complex social environments. Schmidt claims human behaviour can be split into **Physical Conditions**, **Emotional States**, **Cognitive Capabilities** and **Social Status** which are stored as a set of variables in his PECS system. These are used to generate two types of behaviour, Reactive, where behaviour follows fixed rules e.g. instinctive behaviour where agents become hungry and therefore seek food, and Deliberative behaviour where agents reflect on given tasks e.g. how to find food. The system has been applied to a number of sample applications where agents have the ability to explore an environment to build up a cognitive view. Once this model has been built, the agent can navigate around it. It also incorporates the ability to learn, and forget, and make decisions based on an emotional state. The system is based on a framework to include reinforcement learning where an agent can adapt its behaviour based on previous successes or failures.

AI IMPLEMENTATION AND TOOLS

While the above outlines potential architectures which act as a good reference point for the conceptual and formal modelling of intelligent agents, the implementation can be less straightforward, and may take a considerable effort to create. As an alternative, the inclusion of third party tools which exhibit the fundamentals of BDI may be a solution to rapid development of intelligence within the crime scene generator. Examples of this type of application are; SOAR (2004), which is a common development platform used for the developed for multiple platforms including JAVA. JACK (2004); a commercially

available development environment which purports to be a fully agent-oriented. JADE (2004), which according to Braubach et al (2003) is a widely adopted multi-agent development platform working on top of the JAVA programming language which has an available add-on called Jadex which attempts to implement a BDI Infrastructure, and Sim_Agent (2004) which is capable of modelling individual and multi agent behaviour on multiple platforms.

While the use of third party tools is a viable solution to agent modelling, other problems may arise from their usage. The crime scene generator needs to be as graphically accurate as possible, which was accomplished in the original version via a graphics engine written in C++ utilising Microsoft's multimedia library DirectX 8.1. Methods will need to be investigated to examine how third party tools could be incorporated within this development environment, or whether alternatives can be found e.g. OpenGL or Java3D, and also to evaluate the various pros and cons of each system.

OTHER CONSIDERATIONS

The architectures identified so far have been used to model an environment, specify a list of actions that can be performed in that environment, deliberate about which actions to perform, and view the results those actions. However, there is a lot more to producing an agent that appears intelligent and acts in human like manner. For example, Burke and Blumberg (2001) are looking at reinforcement learning so entities can be taught what the right way to act is, and what is wrong, with the use of a feed back mechanism using praise and punishment techniques. Prendinger *et al* (2002) has developed systems for emotion modelling enforced by social constrains for language training. Takács and Kiss (2003) are focusing their research on producing highly accurate graphical entities that are capable of expressing a wide array of emotions based on the temporal transformation of facial displays between states to create smooth transitions. According to Schmidt (2000) modelling human behaviour is highly complex; there are many complementary elements that make up individuals. People have desires which are fulfilled by physically and verbally interacting with the world around them.

We have personalities, emotions and moods which are communicated via non-verbal means such as gestures and facial displays, and have evolved to live in social groups with an accompanying hierarchy of social status. All of these factors are governed by a set of social norms and rules which are used when interacting with other people for the benefit of the individual as well as survival of the species.

EMOTIONS

A feature that will be incorporated into the application will be the use of emotion to generate behaviour, and be used as a means for students to gain an understanding of the motivation of different personality types. Researchers are focussing on simulating these states, and develop of solid internal representations of emotion. According to Allbeck and Badler (2001) emotion can be modelled using the following criteria based using standard psychological models. An agent will have an overriding personality that will remain stable in a similar way to humans. Some people have a tendency to have a brighter outlook on life, and are slow to anger, while others of a more pessimistic attitude will be more likely to get depressed. It can be referred to as the characters temperament and can be split into five major areas: Openness: The ability to take on and accept new ideas, Conscientiousness: How dependable and organised a person is, Extraversion: Refers to sociability and friendliness, or reserved and shyness, Agreeableness: Whether a person is good natured and sympathetic, or rude and critical, Neuroticism: How nervous or secure a person feels.

Prendinger *et al.* (2002) uses these basic personality types in conjunction with a temporary mood state which influences the external expression of the emotional state. People can feel different emotions about many different things simultaneously. For example, they could be happy at someone who has given them a gift, whilst also being angry at someone for forgetting their birthday. In this system the internal emotional state of synthetic entities is modelled, which it is hoped will make the characters appear more believable in the context of language training. They describe two tools they developed. 'SCREAM' (SCRipting Emotion-based Agent

Minds) which contains an emotional model, emotion generation, emotion regulation, and emotion expression and is used to give a character goals and attitudes. The second tool MPML (Multimodal Presentation Markup Language) is an XML style markup language that allows characters movements to be scripted using a set of tags. Combining the two systems allows content authors to design the internal emotions of artificial characters, and combine them with the actions to embody the emotional response.

The agent mind generates and manages the emotional state via a maintenance module consisting of three sub-modules. The appraisal module reasons the emotional significance in set situations. Is angry at, is happy for etc and applies them to goals – would like to be happy, would like to cause discomfort. However, more than one emotion is triggered when interacting with multiple characters so the emotion resolution module makes sense of the many different emotional states a character is in and defines a dominate emotion e.g. the one with the highest intensity value. The agent can be happy for and angry at in different intensities. The emotions are logarithmically combined to produce a winning emotion. Finally, the emotion maintenance module handles the decaying process of different emotions with high intensity emotions dissipating more quickly than milder ones.

While the modelling of these emotions should produce characters that appear to have more human like properties, it is also important, as Giles et al (2003) explain, to express the emotional state. Failure to do so can make complicated processes seem like a sequence of arbitrary actions which fails to produce a sense of empathy from the viewer. Prendinger et al acknowledge this to be a drawback in their system as they use a limited package called Microsoft Agent as the graphical front end which cannot express none-verbal cues. It can be argued that Takács and Kiss have made the greatest advancement in emotion expression with their Temporal Disc Controller which takes facial expressions generated from the dominant emotions (neutral, happy, sad, fearful, disgust, anger) and places them in a disc. To get from one emotion to another, several intermediary facial expressions are passed through providing a smother transition from one state to another. The

faces are made from high definition models and produce some very life-like results. However, the system does not rely on a ridged internal representation of emotions. Instead it is reactive. If a users emotional state is 'Happy' the system will change its facial expression to 'Happy' to reflect back emotion and appear sympathetic. Gilles et al propose a more accurate representation of emotion by using a similar modelling technique to Prendinger's, but link it to an emotion expression system like Takács and Kiss. However, implementation is not complete. So far they have focussed on gaze production where the characters mover their eyes to look at areas of activity within the environment. In addition to facial animation, body language also plays an important function in portraying emotional states. According to Fromkin (1993) when humans communicate large amount of information is transmitted via none-verbal means such as gestures and facial displays. While a human can instinctively understand this none verbal information, it is a difficult problem to convert it into a meaningful algorithm in a computer program. This forms the basis for much ongoing research and will be further investigate in the future.

PATH PLANNING

An area that could be potentially incorporated into a reactive layer of hybrid agent architectures could be the ability to navigate an environment based on an agents' belief of the environment in which it is situated. One way to achieve this could be through the use of common path planning algorithms used in today's computer games. In these applications the environment needs to be split into a discreet set of regular sized cells in a grid formation, or way points. Using this conceptual space, path planning can be conducted via such algorithms as A*. Kuffner (1999) details methods regarding path planning for the natural and realistic movement of 3D animated characters in interactive applications. The research stems from research into motion planning, control, and sensing for autonomous mobile robots. The first technique combines a 2D path planner based on A*, a path-following controller, and cyclic motion capture data to generate the underlying animation. The second technique automatically generates collision-free human arm motions to complete high-level object grasping and manipulation via a Kinematic

approach. This uses a given target position and orientation in a workspace, and a goal configuration for the arm to feed an inverse kinematics algorithm that attempts to select a collision-free, natural posture. If successful, a randomized path planner is invoked to search the configuration space (C-space) of the arm, modelled as a kinematic chain with seven degrees of freedom (DOF). Another potential navigation method has been proposed by Suliman et al (2001) who use Spatial Cognitive Maps to generate a view of an environment from an agent's point of view based on exploration data. To make the models held manageable in size, a memory degradation technique is used. An area recently visited will be recorded in memory in high detail – i.e. it will remember a chair is located in a certain room in a certain part of the house. If a user requests the agent to fetch the chair, the character will know exactly where it is and go directly there. As time progresses and other areas are explored, the memory space will be over-written with new information and the characters knowledge of where a chair is located will gradually deteriorate. After a certain amount of time the character may know a chair is located in a certain area of a house, but not exactly which room it is in, and so a degree of exploitation will be required. This type of system may be more applicable to computer game systems for the production of human like behaviour, although the technique may be applicable in this case due to the optimisation of machine resources.

OBJECT INTERACTION

Once a character has reached its desired location and object, methods are required to interact with it. It could be possible to model all interactions as a set of intentions in a BDI architecture, however, it is possible to abstract these actions using the Smart Object approach. This technique allows objects to encapsulate information about themselves such as what task can be performed with them, and how they are to be interacted with. Using these objects, characters will be able to sense what is available in the environment, and form plans to achieve goal at a comparatively high level, negating the need for a scenario to be over programmed. Goncalves (2001) outline an integrated framework approach in which local perception and close manipulation skills are used

in conjunction with a high-level behavioural interface based on the smart object approach as support for a virtual agent to perform autonomous tasks. In their model sub-tasks are defined by scripts that the agent can perform. Information provided by low-level sensing mechanisms is used to construct a set of local perceptual features at run-time to target potential objects. Once objects are activated, based on their interactivity information and on the current task script, the agent can change its behaviour according to its mission goal defined in a global plan script. This allows the abstraction of the mechanism to link individual perceptions to actions. As a practical result virtual agents are capable of acting with more autonomy, enhancing their performance.

Each object contains information such as geometry, interactions that can be performed, parts, movements, grasp points, functionality etc. Interaction is defined by simple scripts which create interaction plans for each possible agent-object interaction. Primitive actions required by both the object and the agent are defined in a synchronized way. Agents are given high level tasks and can perceive the environment around them, and can then decide if the objects are useful to achieve these tasks. Agents do not hold a world view. Object information is gathered when agent approaches an object. Information is abstracted so there is a lower computational cost which combats the problems that arise when trying to coordinate high level goals with low-level actions.

CONCLUSIONS AND FUTURE WORK

We have now identified the basic tools and architectures that are required to produce a system for the production of life-like behaviour. These need to be further evaluated so a decision can be reached as to the most appropriate for the crime scene reconstruction application. Work will then be initiated for the implementation of the proposed architecture. Work has already been completed on the implemented of a path planning algorithm to allow the discovery of optimal paths in known environments. The next stage will be to implement a method for an agent to build knowledge of unseen environments by exploration to create a cognitive map. We will then integrate these tools into a framework to allow for deliberative and reactive behaviour to be incorporated using

emotion, reinforcement learning and memory degradation. Future work will include the investigation of the possibility of incorporating Natural Language Understanding techniques to allow for human computer interaction to build up crime events based on description.

REFERENCES

- Allbeck, J. Badler, N. (2001) *"Toward representing agent behaviors modified by personality and emotion."* Workshop on Embodied Conversational Agents -Let's specify and evaluate them! at AAMAS 2002, Bologna, Italy.
- Anastassakis, G. Panayiotopoulos, T. Ritchings, T. (2002). *Virtual Agent Societies with the mVital Intelligent Agent System* in Intelligent Virtual Agents, Third International Workshop, IVA 2001, Madrid, Spain, September 10-11, 2001, Proceedings. Lecture Notes in Computer Science 2190 Springer 2001, ISBN 3-540-42570-5
- Bratman, M. (1987) *"Intention, Plans, and Practical Reason"* Harvard University Press: Cambridge, USA
- Braubach, L., Pokahr, A., Lamersdorf, W., Krempels, K., Woelk, P. (2004) A Generic Simulation Service for Distributed Multi-Agent Systems, in: From Agent Theory to Agent Implementation (AT2A1-4) 2004
- Burke, R. Blumberg, B. (2001) *Using an Ethologically-Inspired Model to Learn Apparent Temporal Causality for Planning in Synthetic Creatures* Synthetic Characters Group, The Media Lab, MIT, Cambridge USA
- D'Inverno, M., Kinny, D., Luck, M. and Wooldridge, M. (1998) A Formal Specification of dMARS. In Proceedings of Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages 1365 , pages 155-176.
- Davies, N., Mehdi, Q., Gough, N. E. (2004), 'Crime Scene Reconstruction With Integrated Animated Characters', *Proceeding of*

- European Simulation Multiconference. Networked Simulation and Simulated Networks, Magdeburg, Germany, 388-393.*
- Emerson, E., Srinivasan, J. (1989) Branching time temporal logic. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 123--172. Springer-Verlag, Berlin, 1989.
- Fromkin, V., Rodman, R. (1993) *An Introduction to Language* 5th ed Harcourt Brace College Publishers, New York, USA
- Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: (1999.) The Belief-Desire-Intention model of agency. In: *Proceedings of Agents, Theories, Architectures, and Languages*
- Georgeff, M., Lansky, A. (1987). "Reactive Reasoning and Planning". *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, 1987
- Giles, M., Ballin, D., Dodgson, N., (2003) *Integrating internal behavioural models with external expression* University College London in Partnership with BText Technologies, Ipswich
- Goncalves, L Kallmann, M., Thalmann, D. (2001) *Programming Behaviors with Local Perception and Smart Objects: An Approach to Solve Autonomous Agent Tasks*, Proc. SIGGRAPI 2001, Florianopolis, Brazil
- JACK, The Agent Oriented Software Group (2001) [cited 12th October 2004]. <<http://www.agent-software.com/shared/home/>>
- JADE, Java Agent Development Framework (no date) [cited 12th October 2004]. <<http://jade.tilab.com/>>.
- Kuffner, J J. (1999) *Autonomous Agents for Real-Time Animation*. PHD Thesis, Stanford University, Stanford, CA, USA
- Müller, J.P., Pischel, M.(1993) *The Agent Architecture InterRaP: Concept and Application*. Technical Report RR-93-26, DFKI Saarbrücken (1993)
- Prendinger, H., Ishizuka, M. (2002). Scripting the bodies and minds of life-like characters. In *Proceedings Seventh Pacific Rim International Conference on Artificial Intelligence (PRICAI-02)*, 2002.
- Roa, A. & Georgeff, M. [1995], Bdi agents: From theory to practice, in 'Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, CA, June'.
- Schmidt, B. (2000) *The Modelling of Human Behaviour*. SCS-Europe BVA, Ghent, Belgium
- SIM_AGENT, The University of Birmingham School of Computer Science The Cognition and Affect Project (2004) [Cited 12th October 2004] <http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html>
- Suliman, H., Mehdi, Q., Gough, N. E. (2001), 'Spatial cognitive maps in agent navigation and path planning', *Proceeding of ISCA 10th Int. Conf.*, Arlington, Virginia USA, 27-31.
- Takács, B., Kiss, B. (2003), *The Virtual Human Interface: A Photo-realistic Digital Human*, in *IEEE Computer Graphics and Applications Special Issue on Perceptual Multimodal Interfaces*.
- Wired New, NASA Gets Its HAL (1997) [Cited 12th October 2004] <<http://www.wired.com/news/technology/0%2C1282%2C6284%2C00.html>>
- Yang, Y., Chen, X. (2003) *A Hybrid Agent Architecture for Dynamic and Unpredictable Environments* Workshop on Adaptability in Multi-Agent Systems The First RoboCup Australian Open 2003 (AORC-2003) Sydney Australia 2003

AI Tools for Games

| | |
|--|------------|
| Fyfe, C. Independent component analysis against camouflage | 259 |
| Kumar, P., Bottaci, L., Mehdi, Q. H., Gough, N. E. and Natkin, S. Efficient path finding for 2D games | 263 |
| MacLeod, A. Perudo as a development platform for artificial intelligence | 268 |
| Livingstone, D. and McGlinchey, S. J. What believability testing can tell us | 273 |
| Leen, G. and Fyfe, C. An investigation of alternative path planning algorithms: Genetic algorithms, artificial immune systems and ant colony optimisation | 278 |
| Nieuwenhuisen, D., Kamphuis, A., Mooijekind, M. and Overmars, M. H. Automatic construction of roadmaps for path planning in games | 285 |
| Hartley, T., Mehdi, Q. H. & Gough, N. E. Using value iteration to solve sequential decision making problems in games | 293 |
| Poster | |
| Cazenave, T. Monte Carlo real time strategy | 298 |

INDEPENDENT COMPONENT ANALYSIS AGAINST CAMOUFLAGE

Colin Fyfe
Applied Computational Intelligence Research Unit,
The University of Paisley,
Scotland.
Colin.fyfe@paisley.ac.uk

KEYWORDS: Independent component analysis, intelligence.

ABSTRACT We create an artificial game environment in which the signal from a human player can be sensed by an AI in one of 10 sensory channels. However the human can disguise his signal, effectively by splitting the signal between the 10 sensory channels and then corrupting it with high amplitude noise. We show that the technique of Fast Independent Component Analysis can overcome this signal and illustrate a small game in which the human can adapt his camouflage while the AI searches for the hidden signal.

INTRODUCTION

There has been increasing interest in using modern artificial intelligence techniques to make computer opponents (the AIs) truly intelligent. Modern artificial intelligence is based on distributed techniques which tend to be data driven such as artificial neural networks (ANNs) or genetic algorithms. These are often characterized as being adaptive: the parameters in the methods change often on-line in a manner dictated by the data. Most attempts to use ANNs have concentrated on supervised learning [Champanand, 2004]. However, true human intelligence is unsupervised or perhaps we should describe it as self-supervised: we self-organise our knowledge adaptively without the aid of an external prompt. The few attempts to use unsupervised artificial neural networks have tended to concentrate on varieties of competitive learning, particularly Kohonen's Self-organising Map (e.g. [Kohonen, 1995, McGlinchey, 2003]).

There is a second strand of unsupervised learning based on projections of data sets. Many of these are based on extensions of Principal Component Analysis [Fyfe, 1995] such as Exploratory Projection Pursuit [Fyfe and Baddeley, 1995] and Factor Analysis [Charles and Fyfe, 1998]. To the author's knowledge, the current paper is the first attempt to apply a neural network projection technique to game playing by an AI. We set up a situation in which a human player can be sensed by the AI through one of 10 sensory channels. However the human player is allowed to camouflage his signal to attempt to beat the AI. The AI uses the new technique of Independent Component Analysis [Girolami and Fyfe, 1997] to overcome the camouflage.

INDEPENDENT COMPONENT ANALYSIS

Independent Component Analysis is often thought of as an extension of Principal Component Analysis: it is also a projective method which attempts to identify the independent components of a signal. A typical problem is the "blind separation of sources" such as in the "cocktail party problem" in which there are a number of simultaneous speakers each of whose conversations we wish to identify separately. The linear problem has been largely solved: let there be n different sources, s_1, \dots, s_n mixed with a square mixing matrix, A . Then the received signal is an n -dimensional vector $\mathbf{x}=(x_1, \dots, x_n)$ so that $\mathbf{x}=\mathbf{A}\mathbf{s}$. The ICA method finds solutions y_1, \dots, y_n so that the y 's are equal to the original sources, s_1, \dots, s_n in some order. There is a further ambiguity in that the magnitude of the original signals cannot be determined unless we know something about the mixing matrix A .

In games in which speed is of the essence, we need a fast method for solving this and so we use Hyvarinen's FastICA method to find a separating matrix B . This method is defined as [Hyvarinen et al, 2001, page 210]

1. Center the data to make it zero mean.
2. Choose an initial (e.g. random) separating matrix B . Choose initial values of γ_i , $i=1, \dots, n$ either randomly or using prior information. Choose the learning rates μ and μ_γ .
3. Compute $\mathbf{y}=\mathbf{B}\mathbf{x}$
4. If the nonlinearities are not fixed a priori:
 - a. Update $\gamma_i=(1-\mu_\gamma)\gamma_i+\mu_\gamma E\{-\tanh(y_i)y_i + (1-\tanh(y_i)^2)\}$ where $E()$ is the expectation operator.
 - b. If $\gamma_i>0$, $g_i=-2\tanh(y_i)$ else $g_i=\tanh(y_i)-y_i$
5. Update the separating matrix by
$$\mathbf{B} \leftarrow \mathbf{B} + \mu[\mathbf{I}+\mathbf{g}(\mathbf{y})\mathbf{y}^T]\mathbf{B} \quad \text{where}$$
$$\mathbf{g}(\mathbf{y})=\{g(y_1), g(y_2), \dots, g(y_n)\}$$
6. if not converged, go back to step 3.

PROOF OF CONCEPT

We may apply this method to identification of a signal in a very noisy environment. We have modelled the situation in which an attacker is approaching within an extremely noisy environment. The method of transport of the attacker is not known *a priori* and the signal which is received by the AI may be disguised by the attacker. The

AI must identify the approach of the attacker within this noisy environment. We create a system in which the AI has 10 sensors corresponding to 10 channels with which to identify the attacker but

1. The environment is extremely noisy. We add noise from a zero mean, unit variance Gaussian distribution to each sensor independently at each time instant.
2. The attacker can choose to disguise the channel which he is using by employing any linear mixing matrix to his signal. So, for example if he is using channel 1, his signal should be $(1,0,\dots,0)$ but in practice can appear to the AI as e.g. $(0.37,0.41,0.21,-0.64,0.88,-0.13,0.55,-0.45,-0.76,0.43)$.

The AI must identify the signal quickly and accurately recovering the sensory inputs.

The sensors and signals can be of any type, however for the purposes of this paper we have created a visual signal. Each sensor operates on a 10X10 grid; the signal should be seen on a single channel: Figure 1 is deemed to be the signal representing a spaceship. Figure 2 shows an example of one of the other nine channels; all are merely noise. However the opponent is allowed to disguise the signal. For the purposes of this demonstration, we randomly mixed the signal and the 9 noise inputs to get 10 mixtures which are shown in Figure 3. It is difficult (we believe impossible) to identify the spaceship in any of these.

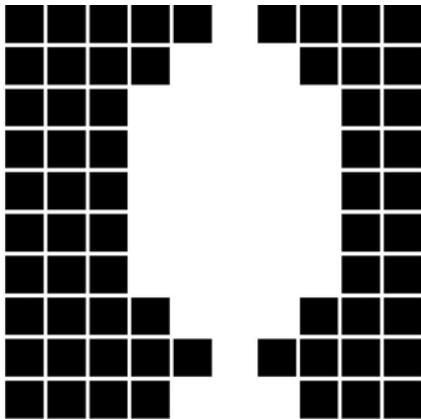


Figure 1 The spaceship to be identified.

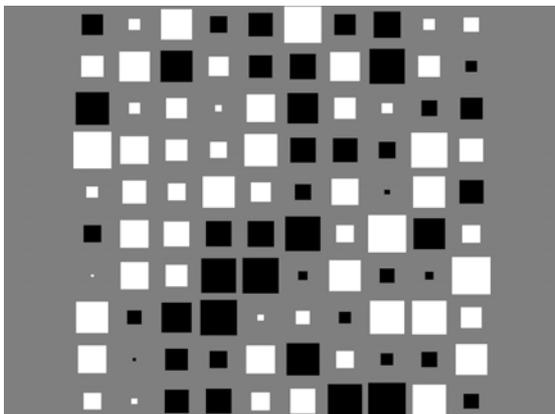


Figure 2. One of the noise signals which were mixed with the spaceship.

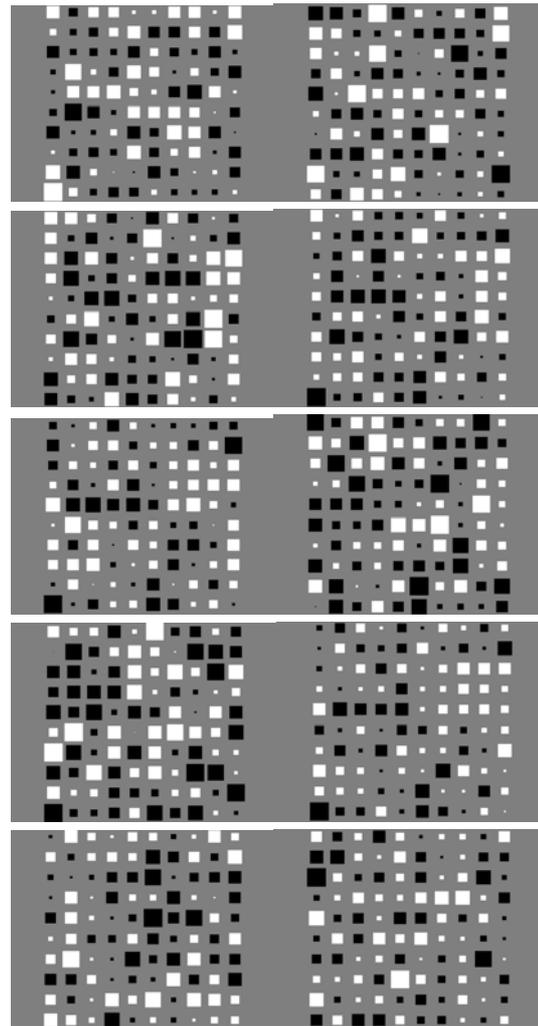


Figure 3. The 10 mixtures containing the spaceship and the 9 noise mixtures

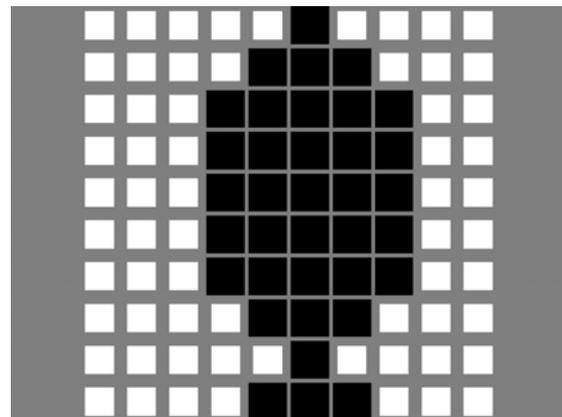


Figure 4. The recovered spaceship

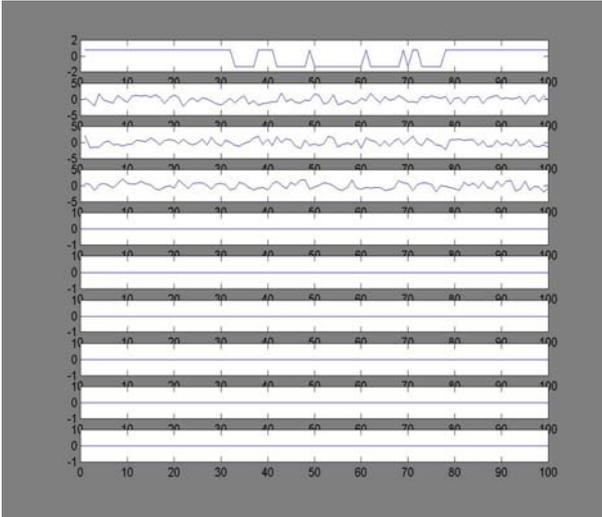


Figure 5. The method has attempted to identify 4 independent components (the top one is the spaceship) before acknowledging that there is no other signal in the mixture.

In Figure 4 we show the recovered spaceship. i.e. the method has identified the correct demixing matrix which will reveal the spaceship exactly. We note that it actually appears as a negative. This is an innate ambiguity in the problem: if we multiply one signal by -1 and the corresponding column of the mixing matrix A by -1, we get exactly the same mixture: we cannot tell whether we are recovering s_i or $-s_i$.

The method can be extended to deal with up to the same number of opponents as there are sensors; in the above scenario, we could have up to 10 opponents being identified at any one time. In Figure 5, we show the results of the same experiment (with a single signal) when we allow FastICA to search for 10 signals. The elements of the recovered y 's are now spread out as 100 dimensional vector. The top vector corresponds to the signal. The next three are due to FastICA finding some structure in the data which exists purely because of the limited number of samples. We can see that the remaining 6 attempts remain close to 0 and no structure of any type has been found.

The method is best introduced in a game in which the opponent has the option of changing his camouflage dynamically while the AI (using FastICA) must track the changes. It is possible to seed the FastICA method to use the previous search results as the starting position for the new search, however this gives too much advantage to the AI with the FastICA method. A more entertaining alternative is to start from scratch each time so that the FastICA method has to work harder to find the signal: this results in occasional failures so that its opponent's camouflage gains the upper hand for a short spell which means that the AI is blind for a spell. Occasionally too a

less than 100% accurate demixing matrix B is found giving an estimate of the signal such as shown in Figure 6.

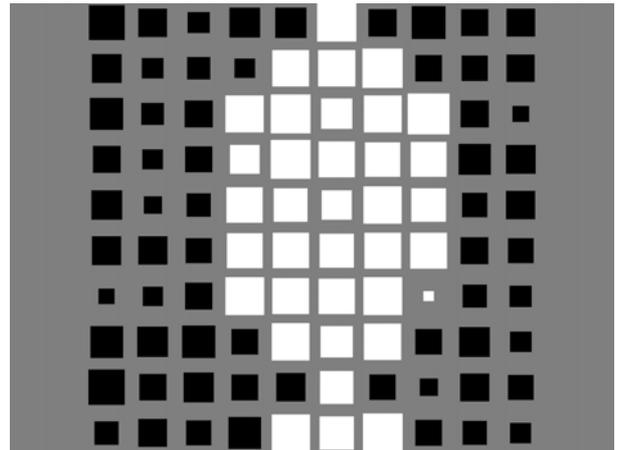


Figure 6. The camouflage is overcome but the de-mixing is not done perfectly.

THE GAME

The game is essentially 10 dimensional hide and seek.

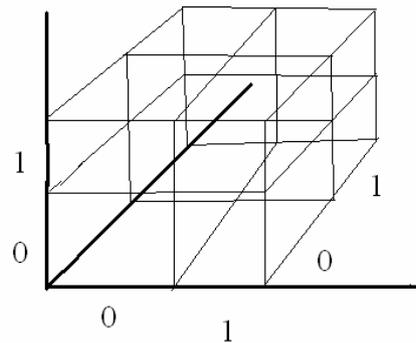


Figure 7. A 3 dimensional illustration.

Since 10 dimensions are rather difficult to visualise, we illustrate the basic idea in three dimensions in Figure 7. We split each dimension into 2 and label the halves 0 and 1. Thus the box in the face closest to the viewer in the top right corner is labelled (across, up, in)=(1,1,0). We are going to allow only movement into adjacent boxes and so from (1,1,0) we can only move to (1,0,0), (1,1,1) or (0,1,0). In our 10 dimensional hide and seek, we may correspondingly move directly from any 10 dimensional box to only 10 of the possible $2^{10}=1024$ boxes.

Various treasures are hidden in 8 of the boxes in this 10 dimensional space. This space is guarded by a fierce AI which searches the boxes continually for intruders. However the AI itself is limited by the movement restriction – it too can only move from the box it currently occupies into one of the 10 adjacent boxes. However if it gets to within a Hamming distance of 3 of the intruder, it

can smell the intruder and knows that an intruder is near. It will then search the nearby area getting positive and negative feedback as it gets closer or further away from the intruder. The human intruder also has an invisibility cloak – a linear mixing matrix as before – which he can use to try to escape detection but both this and the AI's FastICA method come with costs. Since the FastICA is not perfect, there is a possibility that even if the AI gets to the human's box, it will not be able to detect the human. The human is given information as to which box the AI is in and which he is in (both coded in 1s and 0s) at each instant in time.

The game comes with various levels of difficulty – we can increase the dimensionality of the space or we can increase the granularity of the boxes e.g. divide each dimension into three boxes but note that $3^{10} = 59049$ and that boxes may be adjacent to up to 20 others. This makes the game more interesting but much more difficult.

CONCLUSION

We began with the observation that true intelligence requires adaptation: it is impossible in life to determine each situation in advance and so living creatures have to be adaptable if they are to survive. Thus we wish to build this into our AIs with a view to making our computer games more enjoyable. We have illustrated how an unsupervised artificial neural network technique, Independent Component Analysis, can be used in a simple artificial game in which the human player is allowed to disguise his signal while the AI attempts to overcome the camouflage.

We can also state that, with the FastICA method, the AI's responses are real-time responses: the human is aware of very little pausing in the game itself. We note that this is not the case if we use neural network (e.g. [Girolami and Fyfe, 1997]) methods.

This paper is very much a proof-of-concept paper - we have shown how the technique can be used - but the next stage is more important: we now require to show that the technique can be incorporated into an existing computer game so that it may be used in the armoury of techniques which will be necessary to endow our AIs with true intelligence.

REFERENCES

- Champandard, A. J., 2004, AI Game Development, New Riders Publishing.
- D. Charles and C. Fyfe, 1998. Modelling Multiple Cause Structure Using Rectification Constraints. *Network: Computation in Neural Systems*, 9:167-182.
- C. Fyfe, 1995. Introducing asymmetry into interneuron learning, *Neural Computation*, 7(6): 1167-1181.

- C. Fyfe and R. Baddeley, 1995. Non-linear data structure extraction using simple hebbian networks. *Biological Cybernetics*, 72(6): 533-541.
- M. Girolami and C. Fyfe, 1997. Extraction of Independent Signal Sources using a Deflationary Exploratory Projection Pursuit Network with Lateral Inhibition. *IEEE Proceedings on Vision, Image and Signal Processing*.
- A. Hyvarinen, J. Karhunen and E. Oja., 2001, Independent Component Analysis, John Wiley and sons.
- T. Kohonen, 1995, Self organising Maps, Springer-Verlag.
- S. McGlinchey, 2003 Learning of AI Players from Game Observation Data, GAME-ON 2003, 106-110.

EFFICIENT PATH FINDING FOR 2D GAMES

Pawan Kumar⁽¹⁾ Len Bottaci⁽¹⁾ Quasim Mehdi⁽²⁾ Norman Gough⁽²⁾ Stephane Natkin⁽³⁾

pawan_skumar@hotmail.com l.bottaci@hull.ac.uk q.h.mehdi@wlv.ac.uk n.gough@wlv.ac.uk natkin@cnman.fr

¹Department of Computer Science
University of Hull
Hull HU6 7RX

²School of Computing and IT
University of Wolverhampton
Wolverhampton WV1 1EQ

³Computer Research Laboratory (CEDRIC/ CNAM)
292 Rue St Martin
75141 Paris Cedex 03 France

KEYWORDS

Path finding, A*, A Star, Path searching, Heuristics

ABSTRACT

In this paper we investigate different methods and algorithms from artificial intelligence that can be used for achieving efficient path finding within games and virtual environments. Path finding is a computationally expensive problem that is solved by searching. We investigate different optimization techniques and further develop techniques which can be incorporated within the existing algorithms to make path finding for 2D static environments faster, computationally less expensive and requiring minimum use of resources.

INTRODUCTION

Often games have characters that are controlled by players. Whenever a player issues a command, it is intended they behave intelligently in a manner consistent with their roles. This may include carrying a box, painting a wall, etc. But to do these tasks, they have to move from one place to another. This requires a realistic looking path between the two locations. As the number of characters increases, it may require multiple paths simultaneously. In general, human movement is an artificial intelligence (AI) or robotics problem for which there exists no general solution and therefore the aim of this study is to investigate different methods and algorithms from the artificial intelligence which can be used for efficient path finding and to develop a tool to find an optimal solution to the path finding problem.

In modern games, most of the resources are used in the enhancement of graphics and physics and very few are available for AI. Therefore, it is assumed that very limited resources are available (with respect to memory and processing) for finding the paths in real time. Further, we assume a 2D environment so that much of the work can be focused on development of efficient algorithm rather than dealing with the complexities associated with the 3D environments.

The efficiency of path finding within an environment mainly depends upon the complexity of the environment. By complexity, we mean how big the environment is, whether it is static or dynamic and how many and how large are the obstacles within the environment. Further, these obstacles can also be static or dynamic. Therefore to make things simple, we restrict our study to static environments that has large obstacles, small obstacles

and no obstacles. Dynamic environments would be considered in future as an extension to this study.

BACKGROUND

Path finding is an AI robotics problem that cannot be solved without searching. The main problem in path finding is the obstacle avoidance. One of the ways to approach this problem is by ignoring the obstacles until one encounters it (Stout, 1996). This is a simple step-taking algorithm that requires units' current position and its destination position to evaluate a direction vector and information as to whether the units neighbouring region is clear or blocked. This algorithm finds the path along with the movement but the paths generated by this are not realistic, computationally expensive, requires lot of memory. Therefore it becomes necessary to have entire knowledge of path before the movement is applied. This is also necessary in the case where there are weighted regions and finding the cheapest path is important.

Various algorithms exist from the conventional AI that can be used for path searching before its execution. These algorithms are presented in terms of changes in the state or traversal of nodes in a graph or a tree. Russell et al (1995) suggested these algorithms and broadly classified them in two genres. One genre is of uninformed search algorithms such as Breadth First Search (BFS), Bidirectional BFS, Depth First Search (DFS), Iterative Deepening DFS, etc. These algorithms have no additional information beyond the problem definition and they keep on generating neighbouring states or nodes blindly unless they find the goal. These algorithms do not consider weighted regions, are computationally expensive, requires more memory and may not yield paths in real time. However, they are simple to implement.

The other genre of algorithms uses problem specific knowledge or heuristics to find efficient solution. These include algorithms such as Dijkstra's algorithm, Best First Search (BeFS), A-Star (A*). Both the Dijkstra's and the BeFS finds an efficient and optimal path when there are no obstacles within the environment but in an environment with obstacles, the former yields a shortest path but is computationally expensive whereas the later works less but generates non-optimal paths. The A* on the other hand, combines the best of both the algorithms and guarantees to yield efficient and shortest path. It is probably the best choice for path finding since it can be significantly faster, flexible and can be used in wide range of contexts.

Typically, the A* algorithm traverses within an environment by creating nodes that corresponds to various positions it explores. These nodes not only hold a location but also have three attributes associated, as suggested by Matthews (2002), which are as follows:

1. Goal Value (g): This represents cost to get from starting node to this node. This is the exact cost that depends on the environment.
2. Heuristic Value (h): This represents estimated cost from this node to the goal node.
3. Fitness Value (f): This is the sum of g and h values. This represents the best guess for the cost of this path going through this node. The lower the value of f , the better is the path.

The g value represents the path from start that is supposed to minimise any cost related factor such as distance travelled, time of traversal, fuel consumed, etc. Other factors can also be added such as penalties for passing through undesirable areas, bonuses for passing through desirable areas, aesthetic considerations such as diagonal moves are more expensive than orthogonal moves, etc.

On the other hand, the h value gives an estimate of cost to the goal. It is the most important factor for efficiently working of the A*. A bad heuristic can slow down A* and/ or produce bad looking paths. Generally, a heuristic is an under-estimate of the actual cost to goal so that A* always generates shortest paths but under-estimating the heuristic too much is also not beneficial to A* as it will allow the A* to look for more and more better paths and would take longer time to return the path.

The A* extracts the node which has minimum f value and uses two lists, namely an Open and a Closed, for unexamined and examined nodes respectively. These lists forms the basis for the A* and their associated data structures forms how efficient the A* works.

The implementation of A* in games depends on the nature of the game, the representation of the world, information about the neighbours of each node, the cost functions (including heuristics) and speed and memory issues associated with path finding. No matter how the world looks like, its background has to be quantized so that A* gets the search space to search. Stout (2000) suggested various ways to quantize the world such as Rectangular Grids, Quad Trees, Convex Polygons, Navigation Meshes, etc. Most of these representations require great deal of interaction with artists and modellers of the world. For 2D environments, rectangular grids offer an easy way of representing the search space by partitioning into regular grid of squares. This also allows an efficient access of neighbouring nodes to speed up searching. For a typical node at location (x,y) , a neighbour can simply be generated at location $(x+1,y)$, $(x,y+1)$, $(x+1,y+1)$, $(x-1,y)$, etc. For other techniques, a lookup table is created consisting of information about the neighbours for the fast access of the locations neighbour.

So far it's been discussed above that heuristics forms a major part in working of an A* but what kind of heuristic to be used is another issue. The type of

heuristics used mostly depends on the search space representations and speed and accuracy issues associated with the path finding. Patel (2001) has suggested some heuristics such as Manhattan Distance, Diagonal Distance, Straight Line Distance as possible heuristic choices that can be used and tweaked on rectangular grids to the needs of ones game. So far, Manhattan Distance heuristic is the best underestimate of all.

Although A* is the best search algorithm, it should be used wisely within a game as it may lead to wasting of resources. This is typically the case when there are large environments within a game that leads to generation of hundreds and thousands of nodes in Open and Closed lists. This not only requires excessive amount of memory but also requires too much of processing time which a game cannot afford. Apart from that, there could be a situation when no possible path exists, thus resulting the A* to be most inefficient as it examines every possible location from the start before determining that it is impossible to get to the goal. Moreover, paths generated by the A*, although shortest, may not be aesthetically good and would possibly need to be straightened up, even making them smoother and direct. Thus to overcome the weaknesses of A* and to have the optimal use of resources, it requires optimizations on the A* and the path finding. These are discussed in detail as they are dealt in this study.

THE PATH FINDER TOOL

The Initial Framework

The above research has brought solid understanding of what is required to develop the tool. We first develop an application framework using Microsoft Foundation Classes (MFC) and OpenGL graphics library using C++. MFC is the obvious choice as it provides a good interface to build graphical user interfaces (GUI) under Windows environment while OpenGL provides a clean and user-friendly graphics application programmer interface (API). The development took place within Microsoft Visual Studio .net environment that provide ample tools for debugging and object-oriented programming.

We base our initial design on a design pattern of Model-View-Controller (MVC). The MVC has been proven to be most powerful architecture for GUI. It separates the modelling of the domain, the presentation, and the actions based on user input into three classes (Burbeck, 1992). The figure 1 below represents the relationship between these three classes namely the model, the view and the controller.

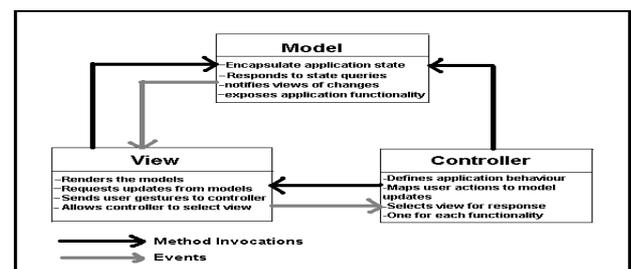


Figure 1: Model-View-Controller pattern

We decide to use MVC pattern as it will allow adding new functionality in the future without making any drastic changes. These additions may include creation of multiple views and controllers and maintaining synchronization of the views whenever a model changes, addition of models of different types with separate views and/ or controllers for these models, porting of existing work to another platform, etc.

The A* Algorithm

The A* forms the core of the study and is much like a custom building i.e. along with the algorithm, it needs information regarding memory (storage), environment (search space) and start and end locations. As discussed, we partitioned the space into rectangular grid with same height and width as the size of the environment. This partitioning is carried out in two levels of inheritance as suggested by Higgins (2002). This has two significant advantages. Firstly, a generic nature of path finding engine can be build to support different environments with same basic functionality. Secondly, this technique emphasise the use of templates instead of base classes and virtual functions, which significantly reduces the assembly overhead associated with the virtual functions.

Further, A* requires some information from the grid that whether a particular grid square is passable or obstructed? In addition, it needs information as to whether a particular node is in Open or Closed list? This information needs to be passed to A* as quickly as possible and at the same time it should be stored efficiently. We approached this by using an unsigned char data structure that stores these different states as status flags. Figure 2 shows C/ C++ representation of the status flags.

```

typedef unsigned char ASFlags;
enum
{
    asfClear           = 0x00,
    asfPassable      = 0x01,
    asfBlocked       = 0x02,
    asfnOpen         = 0x04,
    asfnClosed      = 0x08,
    asfObstructed   = 0x16,
};

```

Figure 2: A* states as status flags

By using single variable, it requires 1 byte per A* node to store its state information which can be retrieved by simple array as a lookup. The size of the array is made to the maximum size of the search space and storage and retrieval of information is made by efficient use of bitwise operators. With this, a node can be in more than one state at one time. This not only reduces memory requirements but also allows path-finding data to be made independent of the search space. This allows path finding for multiple characters to be done simultaneously.

The A* uses this node information in order to keep track of nodes presence in either Open or Closed list. For this, efficient data structures are need for both the lists. With above status flags, no additional data structure is used for Closed list as its functionality is achieved by simply updating the status flags. However, main task of A* lies in the working of Open list. Typically, Open list operation is extraction from a sorted list, insertion into a sorted list, updating the cost of a node in the list and resorting the list, and determining whether it is empty or not. Patel (2001) suggested different data structures that can be used for Open list and recommended the use of priority queues as the efficient data structure. Although, priority queues can be implemented by standard template library (STL) as suggested by Nelson (1996), its STL implementation is limited and does not perform all Open list operations. Instead, we approached to implement priority queues as binary heaps and used STL heap operations on STL vector container. A binary heap is a sorted tree in which a parent always has a value lower than its children. However there is no ordering among the siblings and so it is not a completely ordered tree but is sufficient for A* to perform the insertions and extractions in only $O(\log n)$ (Lester, 2003). Figure 3 and 4 shows a typical case of binary heap in a tree and array (STL vector) representation, respectively.



Figure 3: Binary heaps tree representation

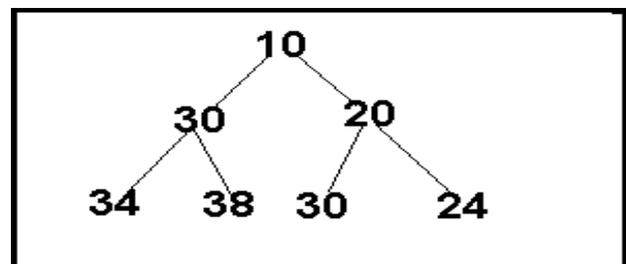


Figure 4: Array representation of binary heap of figure 3

Memory Management

A* requires memory for extraction of nodes and so it is important to have a memory manager which provides an efficient way of dynamic memory allocation for A* nodes. We implement this by using the buffering technique (Figure 5). In buffering, a piece of memory is kept aside by the system to be used for dedicated task. Here we reserve this for the storage of A* nodes.

For A*, it is a good way to manage nodes because A* requires lot of nodes to progress its search. Initially, when a request is made, a piece of memory is dedicated before A* starts execution. During its course of execution, if all the memory gets exhausted, a new buffer is created to progress its search. The size of this buffer is allowed to change so that less memory is wasted. This size mainly depends on the complexity of the environment and therefore requires tuning before it is used in ones application.

This design has significant advantage even though sometimes-extra memory is allocated which increases the memory requirement. Firstly, this results in better use of memory with respect to fragmentation. If smaller nodes are created and deleted on the fly, it leads to fragments in the memory that would make this piece of memory unsuitable for other purposes. Secondly, creation and deletion of new nodes at run time requires same time as creating one large chunk of memory. If smaller nodes were created at run time then this would hit the performance.

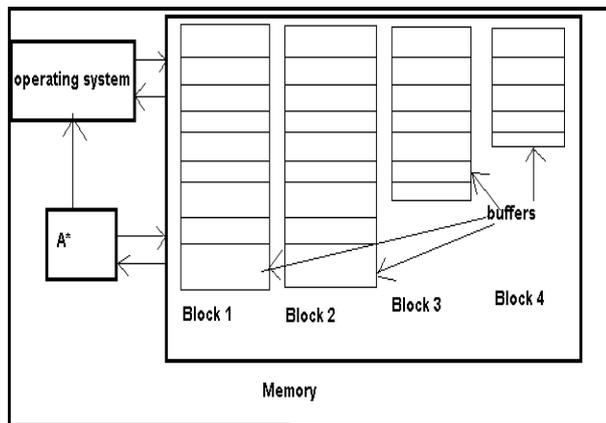


Figure 5: Buffering for memory management

Costs and Heuristics

This forms the main part of research within this study. A* requires two cost functions to proceed its search. These are the actual cost (g) and the heuristic cost (h), which depends on the environment and search space representation.

For rectangular grids, we assume movement in all possible directions and therefore each A* node has a maximum of eight neighbours (four diagonal and four orthogonal) (figure 6). For A* to generate straight paths, a penalty is added for a movement towards the diagonal neighbour as shown in figure 6. However, this cost is scaled by a factor of 10 in order to avoid any floating point calculations to speed up searching within the A*.

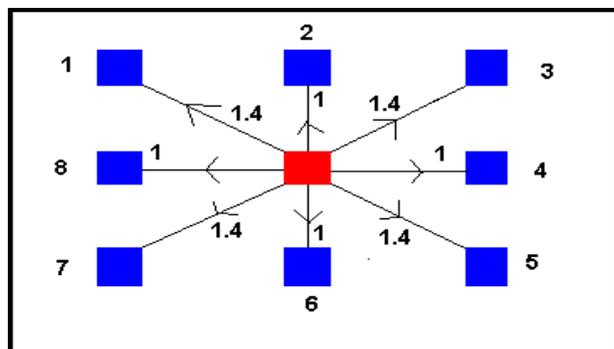


Figure 6: A* node with its neighbours and their respective costs of movement.

Initially the Manhattan Distance heuristic is used as it is supposedly the best underestimate heuristic for rectangular grids (Patel, 2001). The underestimated Manhattan distance simply adds the absolute values of

the difference of their respective X and Y coordinates (figure 7). This is further scaled by factor of 10 in order to avoid floating point calculations and to make it consistent with the scale of actual cost.

$$H = (\text{abs}(X \text{ current} - X \text{ goal}) + \text{abs}(Y \text{ current} - Y \text{ goal})) * \text{Factor}$$

Figure 7: Manhattan distance heuristic

The Manhattan distance heuristic generates optimal path in real time. However, this is true in the case where there are no or very few static obstacles. As the size and the number of obstacles increases, A* not only spends more time on searching but also requires more memory for the nodes as it needs more nodes to find a path. Thus in order to reduce the time and memory requirements when finding paths with obstacles, Rabin (2000) suggested an overestimation in heuristics such that sub optimal realistic looking path are generated in a speed in consistent with a regular Manhattan distance heuristic function with no obstacles. This requires combining of an underestimated Manhattan distance heuristic along with an overestimated heuristic. However overestimation is a research issue and no general solution exists at present. We approached this problem by perceiving ideas from Patel (2001) diagonal movement cost along with lot of experimentation and have come up with an overestimate as shown in figure 8.

$$\text{Overestimate} = \max(\text{abs}(X \text{ current} - X \text{ goal}), \text{abs}(Y \text{ current} - Y \text{ goal})) * 15$$

Figure 8: Overestimate heuristic cost.

The value of 15 as a scale factor is determined by constantly tuning the heuristic on a series of data set. Initially A* algorithm runs on the Manhattan distance heuristic till it encounters an obstacle and then it runs on the overestimated heuristic. This not only has significant performance improvement both in terms of memory and the speed of path finding but also results in realistic and optimal looking paths as generated with Manhattan distance heuristic only. A sample test of this is shown in the following section.

A Sample Test

We checked the developed heuristic on predefined set of start and end locations in an environment which has a large static obstacle. The following figures (9 and 10) show and compare the type of path generated by using different heuristic functions for same start and end location.

Clearly from figures 9 and 10, the paths generated are nearly the same. The Manhattan distance heuristic makes A* to search more number of nodes in order to generate the shortest path. This is evident from figure 9 which shows the nodes searched in different colour from the original grid colour. Also, this requires 120 update cycles of the A* algorithm.

PERUDO AS A DEVELOPMENT PLATFORM FOR ARTIFICIAL INTELLIGENCE

Alasdair Macleod
Department of Computing
University of the Highlands and Islands
Lews Castle College
Stornoway
Isle of Lewis, UK
Email: Alasdair.Macleod@lews.uhi.ac.uk

KEYWORDS

Perudo, Artificial Intelligence, Opponent Modelling, Games.

ABSTRACT

It is proposed that the dice game of Perudo is a more suitable test platform for the development of Artificial Intelligence techniques than traditional perfect information games such as chess. The game and the fundamental principles will be outlined and it is suggested that the unusually high importance of deception makes it impossible for a machine to play expertly without adopting a sophisticated opponent model. A web-based playing system and the development of artificial players with a DLL is described.

INTRODUCTION

Games have traditionally been used as a test bed for the principles of artificial intelligence (AI) because of the belief that lessons learnt from the exploration of a micro-world are then transferable to the real world. Chess was the original choice of game platform when the discipline was at a foundational stage (Shannon 1950, Turing 1953). However chess is a perfect information game and solving the game is essentially a formal task. Whilst the search space is large, the game may be played effectively by applying brute-force linear computing methods alone. In recent years the objective of game-playing programs has been to beat the best human player by whatever means possible and, largely because of the dramatic advances in computing power, was achieved in 1997 by the program 'Deep Blue' beating the World Champion Gary Kasparov (Seirawan *et al.* 1997).

The power of effective search methods coupled with immense hardware clout is evidenced by this success - little or no use was made of AI techniques: in fact, seven years on, the current top program, Shredder 7.04 with an Elo rating of 2810 (comparable to that of Kasparov), still makes very limited use of AI.

The legacy of the association between AI and chess, though not what was originally envisaged, is nonetheless significant. Highly sophisticated and effective search algorithms have been developed that are now universally applicable. There is also a valuable lesson to be gained from the entire exercise by AI practitioners: once economic factors become significant, AI will not be adopted for its own sake – it must confer a performance advantage.

What went wrong with the notion of chess as a development platform? Do games now have any role to play in AI development? We now understand that 'Intelligence' is a combination of many separate elements that have still to be fully understood, far less artificially reproduced and brought together to form coherent systems that can perform mundane tasks to a reliable level. It is this ability to perform mundane tasks that is the real measure of AI and the problem with chess is that it does not demand a simulation of mundane skills by the machine (although the human uses such concepts to constrain the search space as an alternative to a complete tree exploration).

Figure 1 shows a problem that clearly illustrates the huge gulf between the human and the

machine in the context of intelligence in spite of the chess-playing success of the machine.

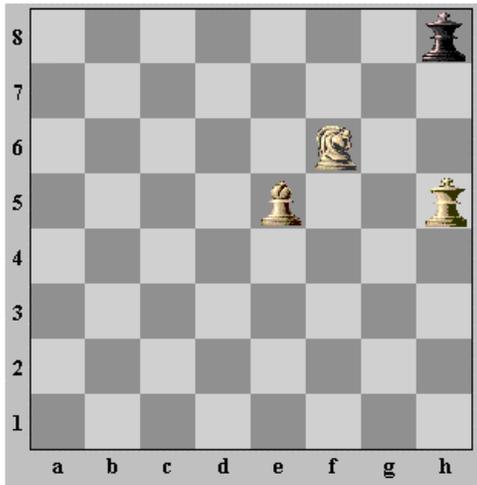


Figure 1 White to play and checkmate in half a move (WB on e5, WN on f6, WK on h5 and BK on h8).

The objective is for White to checkmate the black king in half a move. No existing program can solve this puzzle because the normal constraints are violated. However, the human is not put off and may reason as follows: A move consists of two parts, the lifting of a piece and then its placement on a new square; slightly lifting the Knight off its current square therefore constitutes half a move. The Knight still covers the squares g8 and h7 but now the Bishop can ‘see’ through to the black King and deliver checkmate along the exposed diagonal.

Games can still have an important role in the development of AI, but the game must be carefully selected. Nareyek (2004a and 2004b) has suggested that Internet multiplayer games are suitable for AI development whilst questioning the benefits of some traditional game platforms. The game of Go has also been proposed because the size of the search space is vastly greater than current computers can manage (Cant *et al.* 2001).

Bridge and poker have been exploited with significant success because they are incomplete knowledge games that involve risk management, require multiple competing agents and use deception (unreliable information). However both bridge and poker have relatively complex rules and involve both mundane abilities and expert skills (Schaeffer and Japp van den Kerik 2002, Billings *et al.* 1998). Although many games are now being considered as test beds for AI, it is desirable to conduct a formal analysis of games to extract general and specific characteristics appropriate

to AI research. One such analysis (Macleod 2004a) presented a taxonomy based on the suitability of game types for the testing and development of artificial entities to perform mundane tasks. Games categories were readily identified where the benefits of deep search and the application of complex pre-defined rules are limited.

The dice game of Perudo was identified as potentially suitable. The rules can be learnt by anyone after witnessing a single game but the game is notoriously difficult to play well because of the unusual importance of deception. Intellectual skills are not essential – normal ‘human’ behaviour can ensure victory. We will consider how Perudo may be used to explore AI concepts both at a research and pedagogical level. The game structure is so simple that the user interface programming overhead is low, and an experimenter with limited programming time can quickly develop game-playing agents.

RULES AND BASIC PLAY

Perudo is a member of a family of related dice games with the generic name Bluff of which Liar’s Dice is best known. Each variant has its own nuances but we actually select a simplified rule set and make slight changes to the game nomenclature to make the concepts more familiar to the non-player. Game play using the formal rules is demonstrated with Flash animation at <http://www.perudo-lejeu.com> (in French).

Perudo is a dice game for between 2 and 6 players. Each player starts with a cup and 5 dice. The dice are shaken in the cup then dropped onto the table and hidden from view under the now upturned cup. Each player may peek at his or her own dice, and then a bidding cycle begins. Players estimate or guess the total count of a particular dice value when the cups are eventually raised and all the dice are revealed, then bid accordingly. Each bid must exceed the previous, either by a higher count or a bigger dice value, or both. Instead of raising, a player may instead choose to call. All the dice are then revealed, player-by-player. If the number of the previous bid was not at least reached, the last bidder loses a die; otherwise the caller loses a die. The last person with a die or dice is the winner.

There are some minor complications. 1’s are wild and will represent the value bid when the dice are revealed; when any player goes from two to one dice, a palifico round begins – the

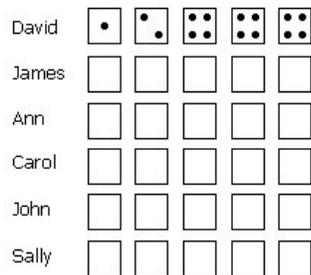
starting dice value cannot be altered by subsequent bids, only the count. This gives the player with the single remaining die an important advantage for that round only.

The game strategy is relatively straightforward. Probability is an important constraint. For example, with 6 players there are initially 30 dice. The most likely distribution of any dice value is a binomial centred on 10. However, in most games there will be 13 of some particular number. Bids should give a clue about unbalanced distributions ... but players can lie. Deception (bluff) and aggression turn out to be the most important elements in the game. Over-reliance on probability will almost certainly result in loss.

Players can form coalitions through friendly co-operative bidding to ‘gang-up’ on a stronger player. Players often play intuitively and rarely perform a search that is greater than 3 ply. The tactics alter when player strategies become apparent as the rounds progress. It is important to respond to the other players’ strategy – in formal terms, accurate opponent modelling is essential.

To illustrate the complexity of the game and its relevance to AI, we can consider the thought process as we go through a simulated game.

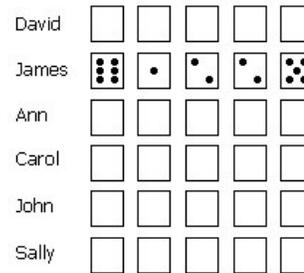
Consider a scenario with six players (reconstructed from the post-mortem analysis of an actual game). David is to go first (Figure 2):



“I could go ten fours, but the bidding may come back round to me as twelve fours, which would be tricky. I really want to keep my fours as a saving bid in case the bid comes back round to me again. In any case, I want to squeeze James because he annoyed me last game. I could try ten sixes. That will force eleven something (he is unlikely to call me with less than a 50% probability of success). Ann is cautious and may call 11 something. Yes, I bid ten sixes.”

Figure 2 What David, to bid, can see and his thoughts.

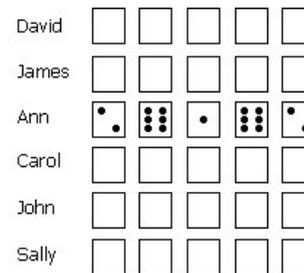
Figure 3 illustrates the James’s response to the first bid.



“Ten sixes. Annoying. I have two sixes in my hand, but he’s probably bluffing – I cannot go eleven sixes because Ann will almost certainly call me. She is cautious and will be reluctant to go twelve anything. If I call, David will gloat if there really are ten sixes. However, I have three twos. If David is bluffing, he is more likely to have twos. Eleven twos is also reasonable because it does not force Ann up to twelve. OK, eleven twos.”

Figure 3 James’s response to David’s opening bid of ten sixes.

Ann’s response is shown in Figure 4.



“Eleven twos. I think I see what’s going on. David is annoyed and probably has no sixes. James doesn’t have sixes either. Although I have three sixes, a bid of eleven sixes is far too risky. James probably has four twos – he wouldn’t bluff me because I will find out when the round is called. I am then more likely to call him in future. I can’t really call eleven two’s or go eleven of anything else, but twelve two’s is probably safe. Carol thinks I am cautious and will reason that James and I have seven two’s between us and will probably raise the bid if she has any twos at all. Twelve twos.”

Figure 4 Ann’s response to James’s eleven twos.

The scenario clearly illustrates the mixture of accurate and faulty lines of thought that makes the game so difficult to play. The game is certainly skilful (as initial tests with artificial

players has demonstrated), and would be enormously difficult for a machine to play well without a battery of AI features. The overriding importance of player modelling suggests that the optimisation of deception cannot be achieved by the simple application of Game Theory with its assumption of rationality. For this reasons it is proposed that the game is worthy of detailed consideration for potential use in AI research.

METHOD OF IMPLEMENTATION

For research purposes, the primary requirement is a web-based mechanism that will permit a large number of simultaneous games to be played. A development environment should be made available to students, hobbyists or even serious practitioners to enable them to build artificial players and automatically join them into games. The web site administration must utilise a good rating system so that the performance of a constructed agent can be accurately and reproducibly quantified for research and learning purposes. The main pedagogical benefit is the ability to learn about and experiment with AI methods. To help evaluate and understand existing and new concepts, the experimenter may typically tweak the machine algorithm, let the unit play continuously for a week (equivalent to about 500 games) then note the rating level to which its performance converges. Further iterations may follow.

It is however desirable that the majority of players in a game are human and for this reason it is important the site attracts as many people as possible for casual games. The design and structure of the game engine is therefore of critical importance.

The system has already undergone two years of development. The initial implementation was a socket level p2p system with a coordination server (built with C#.NET) with XML data packets and running on the University intranet. Though extremely effective, data security and the firewall meant the general public could not access the server. An additional problem was that a program download and installation was required – this tends to discourage casual users.

The second implementation was as an ASP website with the server managing all the game processing. This suffered from the known problems with all such systems: the server was quickly saturated by requests and failed to respond above a limit of one hundred simultaneous games; a client-pull mechanism is

needed to enable the client to check the progress whilst waiting for a bid - this resulted in a rather annoying flicker; cookies needed to be enabled – may users disable cookies.

The current, and preferred implementation is a modification of the ASP system. The server maintains the global variables that represent active games. By calling one of a set of ASP pages (effectively functions) the client may modify the variables, for example by posting a bid, or reading the game status. Server processing is therefore much reduced. Data is conveyed back to the client as plain text in the variable pair format used by Flash e.g. “?Variable1=5?Variable2=3”. The client program is organised as a finite state machine, with progression through the states achieved by accessing the global array and the central player database using thirty-three ASP functions. Data is managed on the client side by a Flash program that interprets the data and displays the game in an attractive way. Most browsers include Flash, hence the game can be played simply by accessing the server URL. Cookies do not need to be enabled, and there is no flicker. To play a game, go to www.playperudo.com. The site will be active from 1st December 2004 – the site is currently undergoing a very time-consuming system test.

The same functional effects could also have been achieved through a Java implementation, however the superior graphical capabilities of Flash make it preferable.

Because access to the game functions requires a password to maintain game security and integrity, developers are encouraged to query the server through the protective layer of a DLL. The purpose is to protect the scoring system, not to hide the operation. The DLL and sample programs can be requested once the site is operational.

The development of a web-based game manager is a challenge even for a game of this simplicity and there are lessons of general educational value. The system development history and the operation of the server are described in detail elsewhere (Macleod 2004b). The structure is applicable to the Internet implementation of a number of turn-based games.

CURRENT ACTIVITIES AND FUTURE DEVELOPMENTS

Using the DLL, a student will typically develop a user interface to play the game prior to creating artificial players. Figure 5 shows a

typical student interface created with Visual Basic 6.

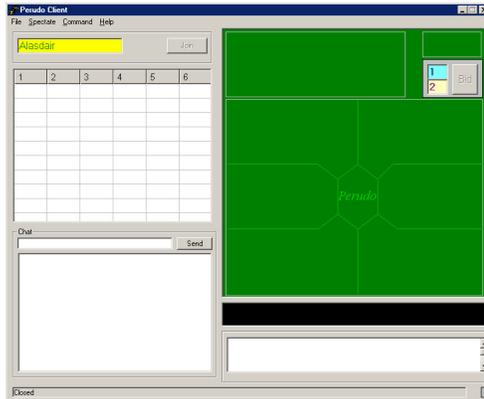


Figure 5 Typical client Perudo interface produced with the DLL.

The importance of the rating system makes it subject to extensive investigation. Rating systems based on a modification of the chess Elo statistical rating system are being evaluated but the difficulty is that the playing order cannot be ignored. Self-play experiments were conducted using artificial players with fixed handicaps in each permutation of position. A complex empirical relation between player separation and influence was revealed, suggesting a distance weighting is appropriate.

Experiments on artificial players with stereotypical characteristics (probability, caution, aggression, truthfulness, random and bluff) are continuing but have already demonstrated convergence with clear-cut performance distinctions. Significant rating noise is present and reflects the randomisation of the move selection process but has not yet revealed the advantage, if any, associated with the random dice configuration. It is believed that for good human players the dice distribution is not particularly significant (if players have the same number of dice).

Several AI concepts are being tested using this platform: an evolutionary mechanism to optimise the probabilistic decision parameters is being developed; simple back-propagation neural networks are learning the game characteristic function; experiments with temporal-difference learning have begun; fuzzy algorithms are being considered; a game theoretic analysis of the game is under way; expert systems are being assembled; opponent models are being constructed.

In general, the environment makes such developments very easy and quick. Self-play

experiments (which do not require the server) can complete a six-player game in 50 ms using a high-spec PC. Over 1,000,000 games a day can therefore be played, more than enough to test a learning algorithm.

A major tool is being produced for AI development. Within a graphical environment, developers may drop and connect AI units (e.g. Expert System Module, TD Learning Module, SOFM, Opponent Model etc), set properties, and connect the composite system to the DLL to rapidly build bespoke players.

REFERENCES

- D. Billings, D. Papp, J. Schaeffer, D. Szafron, "Poker as a Testbed for Machine Intelligence Research", Lecture Notes in Artificial Intelligence volume 1418, Springer Verlag, Robert Mercer and Eric Neufeld (editors), (Proc. 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI'98), Vancouver Canada (1998)
- R. Cant, J. Churchill, D. Al-Dabass, "Using Hard And Soft Artificial Intelligence Algorithms To Simulate Human Go Playing Techniques", *Int. J. of Simulation*, 2(1), 31-49 (2001)
- A. Macleod, "Games for AI research", *To be published* (2004a)
- A. Macleod, "The development of a simple turn-based internet server", *To be published* (2004b)
- A. Nareyek, "Computer Games - Boon or Bane for AI Research?", *KI* 18(1), 43 (2004)
- A. Nareyek, "AI in Computer Games", *ACM Queue* 1, 10 58-65 (2004)
- J. Schaeffer, K. Japp van den Kerik, "Games, computers, and artificial intelligence", *Artificial Intelligence* 134, 1-7 (2002)
- Y. Seirawan, H.A. Simon, T. Munakata, "The implications of Kasparov vs. Deep Blue", *Communications of the ACM*, 40, Issue 8, 21-25 (1997)
- C.E. Shannon. "Programming a computer for playing chess", *Philosophical Magazine*, 41, 265-275 (1950)
- A. Turing, "Chess", in "Digital Computers Applied to Games, of Faster than Thought", Chapter 25, ed. B. V. Bowden, Pitman, London (1953)

ACKNOWLEDGEMENTS

The theoretical aspect of this work was supported by the University of the Highlands and Islands Sabbatical Scheme and Lews Castle College. Western Isles Enterprise provided assistance with the practical development of the system.



WHAT BELIEVABILITY TESTING CAN TELL US

Daniel Livingstone and Stephen J. McGlinchey

School of Computing

University of Paisley

High Street

Paisley

PA1 2BE

Email: {daniel.livingstone, stephen.mcglinchey }@paisley.ac.uk

KEYWORDS

Believability, AI Evaluation

ABSTRACT

A number of recent works on AI for computer games have promoted the idea of ‘believable’ AI as being desirable. That is, AI that fools observers or players into thinking that a human, not a machine, is playing the game or controlling the characters that a player interacts with. This process of testing whether or not game players and/or observers believe that they are interacting with (or watching interactions with) humans not machines we term *believability testing*. We review this idea of believability and show that, despite some flaws, believability testing can be useful in evaluating and improving AI for computer games.

INTRODUCTION

In general, game AI is less interested than traditional academic AI with recreating intelligence, than with creating the appearance of intelligence (see, e.g., Rabin 2002). As such, when creating machine opponents for computer and video games it is not simply their ability to challenge players, but to leave players with the impression that they have been competing against an opponent that thinks and acts like another human opponent would – in its use of strategy and tactics, in its reactions and actions.

In this regard, game AI is much like a limited version of a Turing Test, in that success is determined by the ability to deceive people into thinking that the machine is human. Or at least as a limited version of the conventional conception the Turing Test, if not of the Imitation Game as specifically posited by Turing (Hayes and Ford 1995; Harnad 2000; Turing 1950).

If this is the case, then game AI is perhaps best tested by having players and observers somehow rate the AI in terms of humanness, or to distinguish AI and human players. We present a brief review of past work on testing and evaluating game AI using these approaches, and by conducting our own tests on a system developed to automatically create human like AI opponents (McGlinchey 2003) we report on the value of such approaches. In this we consider how such evaluation might best be conducted, how the results should be interpreted, and what potential the extended use of believability testing has for improving the field of game AI. In this work we concentrate on the use of AI for controlling opponents which might conceivably be under human control, and deliberately leave aside issues relating to non-player character AI and believability in games – although here too the ability to evaluate believability is important (Mac Namee 2004, p123).

EVALUATING BELIEVABILITY

The process of judging how believable the AI opponents and characters in computer games are is by nature a subjective process – being dependant on what observers think of the behaviours they perceive. With no objective measure to rely on, believability can only be determined by asking observers whether or not they think a character or opponent is under human or machine control – such as by using questionnaires.

Such believability testing was carried out on the Soar Quakebot, an AI player for Quake II (Laird and van Lent 1999). Laird and Duchi (2000) recorded games in which a human played against either one of several other human players or against the Soar Quakebot in one of several parameter configurations. Judges were shown videos of games, and asked to rank the opponent player in each game for skill and degree of “humanness”.

Although the amount of testing conducted was small, some interesting results were apparent. For example, bots with human-like decision times were rated as more human than bots with significantly faster or slower decision times, and bots with more tactical reasoning were rated as being more human-like than less tactical ones. Super-human performance was a give away not only in decision times, but in aiming – some bots being too good at aiming to be considered human by the judges. From this preliminary work, three guidelines for AI for action game characters emerge:

- Give the AI a human like reaction and decision time (one twentieth to one tenth of a second).
- Avoid giving the AI superhuman abilities (e.g. overly precise aiming).
- Implement some tactical or strategic reasoning, so that the AI is not a purely reactive agent.

While questionnaires and large groups of test subjects can limit unwanted effects due to observer subjectivity, (Mac Namee 2004) notes that careful questionnaire construction can help further reduce the influence of subjective judgements. Rather than ask how believable an AI is, Mac Namee evaluates different AI's by presenting observers with two at a time, and asking them to decide which is more believable. Additional questions ask judges to comment on any differences they notice between two versions of an AI implementation.

Mac Namee shows that not only is believability a subjective measure, but one that may be subject to cultural effects. One test in particular, where subjects are shown two versions of a virtual bar populated by AI agents, demonstrates this. In one version agent behaviour is modelled according to rules where agents attempt to satisfy competing goals – buy beer from the bar, drink it while sitting with friends and go to the toilet as required. In the other version new short term goals were picked randomly every time agents completed an action. The one Italian test subject felt that having agents return to sit at the same table time after time was unrealistic, whereas the other (Irish) subjects mostly felt that this behaviour was more believable.

While further tests to determine whether this, or other, results are indeed culturally significant differences have yet to be carried

out, the possibility of such differences existing does appear quite real.

An additional, important difference noted by Mac Namee is that game playing novices may find it difficult to notice any differences between different AI implementations. The experience of playing a game is so new to them that they may fail to notice the significant differences between two versions of an AI – even where these differences are readily apparent to individuals with greater game-playing experience.

Related work also exists in Software Agent research, where conversational agents are used to front a range of applications, and it is important to evaluate how users react to such agents. For example, (Bickmore and Cassell 2001) look at how users interact with a conversational estate agent. Again, questionnaires are the prime means of gathering data on user reactions and perceptions and statistical measures are presented to demonstrate how different types of users vary in their response to different versions of the software agent.

Commercially, player perception of AI agents in games was tested during the development of Halo (Butcher and Griesemer 2002). Amongst the findings here, it was noted that AI behaviours needed highly exaggerated animations and visible effects in order to be noticeable to players. The counter-intuitive implication here is that unrealistically over-emotive reactions and actions may appear to players to be more realistic than more subtle reactions. This might not be the case when observers rather than players rate the game, as observers may have more time to study the agents than players involved in the game would (Laird and Duchi 2000), but the aim of a game is to satisfy the player of a game – not an observer.

From this work, it appears that conducting believability testing can be a useful exercise – from which general design guidelines, or very specific changes to improve believability of the AI in a specific game, may be derived. In the next section we report on our own believability tests, and demonstrate how in passing a believability test we might yet deem that an AI has failed to reproduce human behaviour.

TESTING AN AI TRAINED ON PLAYER DATA

In (McGlinchy 2003) player data from games of Pong was captured and subsequently used to train Artificial Neural Network AI Pong players. In this implementation of “Pong”, Figure 1, when the ball collides with a bat, the ball’s velocity vector is reflected and then rotated, allowing the player to exert some control over the direction of rebound from their bat.

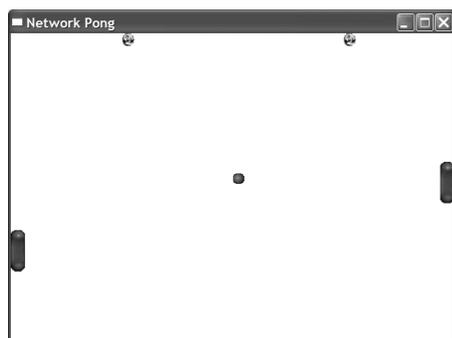


Figure 1: *The game of “Pong”*

It was observed that not only was the AI able to play a reasonably good game of pong, but that it actually imitated elements of the play style of the players it had been trained on. For example, one player in particular would ‘flick’ the bat upon hitting the ball (as if to put spin on the ball, although not actually possible in the implementation of Pong), and when trained on data gathered from this player the AI would do likewise.

Although the AI imitated the distinct play style of whatever player it had been trained on, would it be able to fool judges in believability tests? To determine this, we needed to conduct our own believability tests.

A number of games were recorded and played back to test subjects, who were asked to judge whether the controllers of each bat were human or machine. For each game demonstrated, judges were able to state whether they thought the left bat, right bat, both or neither were controlled by a human. An additional question asked judges to explain how they thought they were able to tell the difference between computer and human players. The games shown to the judges consisted of a mix of these possible alternatives. The game players consisted of three humans, three AI trained on player data and an AI player programmed using simple logic. In total, eight games were shown to each subject: one with two human players, two with

two AI players and the remaining each with one human and one AI player. The programmed AI was only used in one of the games, and was played against a human player.

The programmed AI works as follows: whenever the opposing player hits a shot towards the AI player, the AI projects the point where the ball will intersect the bat’s axis. After a short delay to simulate a human reaction time, the bat is moved gradually towards the point of intersection. In each iteration of the game’s main loop, the bat’s new position, p' , is given by the equation below, where p is the bat’s current position, d is the projected point of impact, and w is a weight parameter that affects the speed at which the bat is moved to its target position.

$$p' = wp + (1 - w)d$$

In our experiments, we used a value of 0.97 for w . To make the AI player fallible, the target position for the bat was moved by a random amount between -20 and +20 pixels from a uniform distribution. (the bat’s height was 64 pixels).

A Self-Organising Map (SOM) trained on player data was used to represent each of the other AI players. To find a target position to move the bat to, the ball’s speed and position are fed into the network, and the first and second placed winning SOM nodes are selected. Both winners suggest a position for the AI player’s bat to be moved to, and an interpolation of these values is used. In order to help give the bat a realistically smooth style of motion, the bat position is updated each frame, moving it towards the SOM’s targeted position. The winner search may be performed in every iteration of the main loop, although this can be wasteful of processing resources, and it is acceptable to do a winner search less often, e.g. every 300 milliseconds with little or no noticeable effect on the game.

Once these adjustments were made, the SOMs, the programmed AI and the human players were recorded to build up the set of eight game recordings noted above. With this in place, a number of subjects were shown the recorded games and their completed questionnaires were reviewed.

WHAT DOES TESTING THE HUMAN-PLAYER TRAINED AI TELL US?

The compiled results from initial tests seem quite promising. Overall the chance of an AI being correctly identified as such appears to be at roughly the level of chance. Slightly more responses indicate that judges have mistaken AI players for human than vice-versa, and the AI's were mistaken for human players more often than they were correctly identified.

Looking at individual returns gives a slightly different picture. While one respondent correctly judged 14 out of 16 players, another misclassified 14 out of 16. Many of the subjects varied from chance significantly – either getting most responses correct, or most responses incorrect. This indicates that some of the observers were in fact able to distinguish between human and machine controllers – even though they made incorrect inferences about which was which.

By providing an additional free-text question that asked respondents to explain how they were able to distinguish the two types of player, we were able to check responses to see what aspect of the AI behaviour might have led to this result. The answers here showed that some of the judges – in particular the judges who got most identifications correct *and* the judges who got most wrong – noticed that some bats moved with more jerky and sudden movements than others. In most cases, these were the AI bats – although some observers thought this behaviour was a marker of human control.

Believability testing has allowed us to test the current implementation of the player-trained SOM AI, and as changes are made will provide a means for testing and comparing successive versions of the AI. Aside from telling us what must be done to improve our current AI (additional smoothing and/or adding momentum to the AI bat being the most immediate changes), these results have also demonstrated the problem inherent in having observers score game AI in terms of its humanness – what appears human-like to one judge might not to another.

BELIEVABILITY OF AI IN COMMERCIAL REAL-TIME STRATEGY GAMES

We have also embarked on a study of believability in commercial Real-Time

Strategy (RTS) games. The slower pace and extended duration of RTS games means that players have a good amount of time in which to observe the behaviour of their opponents. So rather than capture games and show them to observers, we are asking players to select a single game to play against an AI player in their own time. The players are then asked to complete a questionnaire on the behaviour of their AI opponent. Here, the respondents clearly *know* that they are facing machine opposition, but believability – the extent to which the AI acts like a human opponent might – remains both important to the player and testable.

(Wetzel 2004) presents a review of persistent AI failures across genres of computer games, and suggests that the first step in fixing the problems is to find out what they are. By conducting believability testing developers can find failures before their customers do, potentially fixing them so that their customers never find the mistakes.

Although our RTS survey is in very early stages, the returns do indicate that players are aware of failings that exist in some of the available games. We suspect not only that weaknesses exist in all games, but that classes of AI failure are common to large ranges of RTS games. As indicated by our preliminary finding, these are likely to allow players to adapt to and *exploit* the strategies employed by their AI opponents in a way that is not possible against other human opponents.

CONCLUSIONS

Believability testing has demonstrated its worth as a means for evaluating game AI. As well as indicating when a game AI succeeds in recreating human like behaviour, testing can highlight why game AI fails to do so – and this can allow developers to focus their efforts on the areas where their AI systems perform weakest.

Some care must be taken in conducting such tests, however. Where results from a large set of tests might indicate that an AI performs admirably in fooling observers, close inspection of the results can show otherwise – as has been the case with the tests conducted on player-trained pong-playing SOMs. 'Humanness' is clearly a highly subjective measure. In computer games where only a tiny aspect of human behaviour is actually observable, and where different observers can have quite different expectations, attempting to measure humanness may be fatally flawed.

Finally, a crucial question about believability remains. For any given computer game AI, what is it supposed to believably *be*? Should an AI act believably like another human playing the game – or like a characters in the game setting conceivably would act? Whether an AI soldier in some historical war setting should act like a modern player of the game, or like its historical model, will lead to quite different expectations about its behaviour. And accordingly, we have not presented criterion by which believability can be measured. In conducting believability tests it is most important to be clear about what we want believability to mean – and that remains a game designer's choice.

ACKNOWLEDGEMENTS

Daniel Livingstone would like to thank The Carnegie Trust for the Universities of Scotland for supporting his current research.

REFERENCES

- Bickmore, Timothy, and Justine Cassell. 2001. Relational Agents: A Model and Implementation of Building User Trust. Paper read at SIGCHI Conference on Human Factors in Computing Systems, at Seattle, WA.
- Butcher, Chris, and Jaime Griesemer. 2002. The Illusion of Intelligence: The integration of AI and level design in Halo. Paper read at Game Developers Conference, 21st-23rd March, at San Jose, C.A.
- Harnad, S. 2000. Minds, Machines and Turing. *Journal of Logic, Language and Information* 9:425-445.
- Hayes, P.J., and K.M. Ford. 1995. Turing Test Considered Harmful. Paper read at Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-95), at Montreal.
- Laird, John E., and John C. Duchi. 2000. Creating Human-like Synthetic Characters with Multiple Skill Levels: A Case Study using the Soar Quakebot. Paper read at AAAI 2000 Fall Symposium: Simulating Human Agents, November 3rd–5th, at North Falmouth, MA.
- Laird, John E., and Michael van Lent. 1999. Developing an Artificial Intelligence Engine. Paper read at Proceedings of the Game Developers Conference, November 3rd–5th, at San Jose, CA.
- Mac Namee, Brian. 2004. Proactive Persistent Agents: Using Situational Intelligence to Create Support Characters in Character-Centric Computer Games. PhD Thesis, Department of Computer Science, University of Dublin, Dublin.
- McGlinchey, Stephen. 2003. Learning of AI Players from Game Observation Data. Paper read at Game-On 2003, 4th International Conference on Intelligent Games and Simulation, at London.
- Rabin, Steve. 2002. *AI Game Programming Wisdom*. Hingham, MA: Charles River Media, Inc.
- Turing, A.M. 1950. Computing machinery and intelligence. *Mind* LIX (236):433-460.
- Wetzel, Baylor. 2004. Step One: Document the Problem. Paper read at AAAI workshop on Challenges in Game AI, July 25th-26th 2004, at San Jose.

AN INVESTIGATION OF ALTERNATIVE PATH PLANNING ALGORITHMS: GENETIC ALGORITHMS, ARTIFICIAL IMMUNE SYSTEMS AND ANT COLONY OPTIMISATION

Gayle Leen and Colin Fyfe
 Applied Computational Intelligence Research Unit,
 The University of Paisley,
 Paisley.
 Gayle.leen,colin.fyfe@paisley.ac.uk

KEYWORDS: genetic algorithms, artificial immune systems, ant colony optimisation.

ABSTRACT

We investigate the use of three computational intelligence techniques for path planning in computer games. We show that while genetic algorithms and artificial immune systems are effective at finding short useful paths, ant colony methods are much less useful in this task. The Artificial Immune Systems were best in terms of speed and shortness of path found.

PATH PLANNING

Path planning is a common problem in the field of game AI because we desire our characters to be able to navigate around game worlds. Navigation is most often performed in computer games with symbolic AI algorithms such as A*. Such algorithms are now standard fare and somewhat distant from leading edge research in artificial intelligence. We investigate in this paper some of the more recent algorithms applied to this task.

In this paper, we assume a graph based navigation system and investigate the problem of static obstacle avoidance using the computational intelligence techniques: genetic algorithms (GAs), artificial immune systems (AIS) and ant colony optimisation (ACO). These techniques have individually been applied to path planning in the past but the time is now apposite for a comparative study of their performances.

FORMULATING THE ENVIRONMENT

For this experiment, we will set up the terrain / search space as follows: We set up an m by n grid with each possible location labelled as:

| | | | | | | |
|-------|--------|--------|---|---|---|----------|
| $n-1$ | $2n-1$ | $3n-1$ | . | . | . | $m*n-1$ |
| $n-2$ | $2n-2$ | $3n-2$ | . | . | . | . |
| . | . | . | . | . | . | . |
| . | . | . | . | . | . | . |
| 0 | n | $2n$ | . | . | . | $(m-1)n$ |

We use a grid where $m = n = 10$, and place obstacles on the terrain at the following locations: 30 – 36, 39, 70 – 71, 74 – 79

so that the terrain is as shown in Figure 1. Note that the game is played on a cylinder: we have wraparound so that e.g. location 29 is adjacent to location 30.

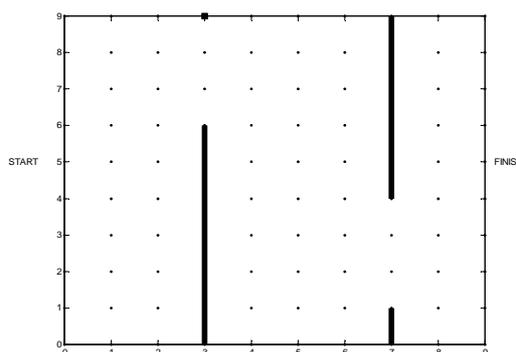


Figure 1: The artificial terrain: a path must be found from the start side to the finish side avoiding the two walls.

We wish to find a path from the ‘START’ side of the grid to the ‘FINISH’ side. We restrict the movement of the path to forward (\rightarrow), left (\uparrow), and right (\downarrow). However, we allow the path to wrap around the grid; i.e. moving left from location $n - 1$ will lead to location n , and moving right from location n will lead to location $n-1$. The paths may not traverse the obstacles.

GENETIC ALGORITHMS

Genetic algorithms were invented by Holland in the 1960s and is a method of moving one population of chromosomes to a new population by using a kind of “natural selection” together with genetics-inspired operators of crossover, mutation and inversion (Mitchell, 1996)

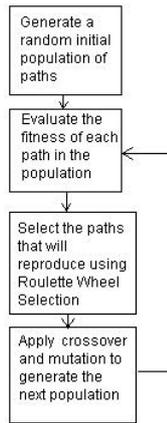


Figure 2: The procedure of a path planning genetic algorithm

The procedure that the path planning GA follows is illustrated in Figure 2.

Each chromosome in the population represents a path that starts from the START side of the grid and complies with the constraints defined above, and consists of an array of grid points which constitute the path. Figure 3 illustrates the two paths:

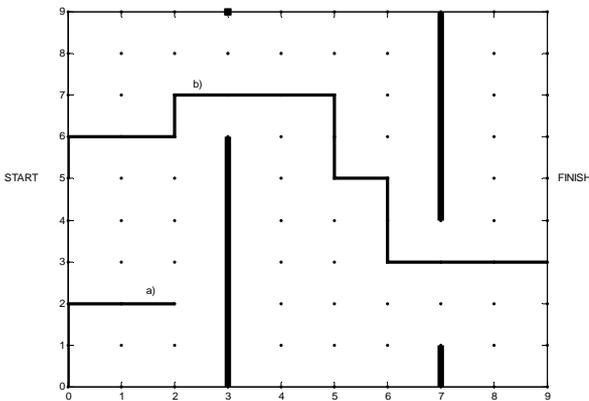
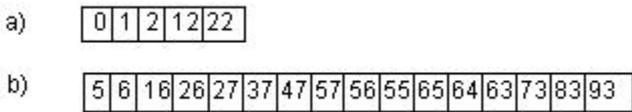


Figure 3: Two examples of chromosomes representing possible paths through the terrain. a) shows a path which goes from START and stops at an obstacle whilst b) represents a path between START and FINISH

1. Initialising the population

The first element of the chromosome is initialised with a random square from the ‘START’ side of the grid. The next element of the chromosome is picked at random and can either be left, right or forward. We keep lengthening the chromosome in this manner until it reaches the other side of

the grid or it encounters an obstacle. Note that the algorithm does not have any knowledge about the ‘correct’ solution.

The population is evolved for a number of generations by repeating 2-5 till convergence:

2. Fitness evaluation

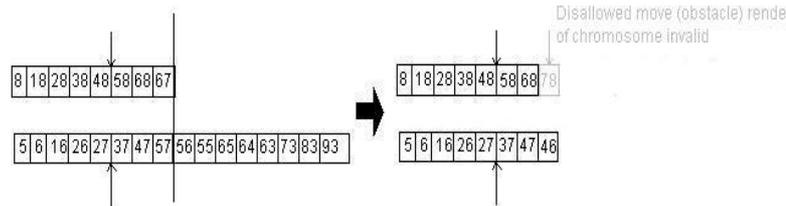
The fitness of each chromosome is evaluated using an objective function. Intuitively, the further a path defined by a chromosome reaches towards the goal, the fitter the chromosome is.

3. Selection of paths for reproduction

We select chromosomes for reproduction using the Roulette Wheel selection method: the higher the fitness of a chromosome, the greater the probability that it has of being selected.

4. Crossover

The next population is then generated by reproducing using the crossover operator: with probability equal to the crossover rate, two chromosomes are selected at random and single point crossover is performed. We define the crossover operator as follows:



If the two chromosomes are not the same length then we crop the longer chromosome to the length of the shorter one. A crossover point is then randomly selected, and from this point for each chromosome, we change the genetic material so that it follows the same path shape as the other. If at any point, the path moves to a location that is disallowed, we discard the rest of the chromosome.

5. Mutation

Genetic variation is created by randomly varying the offspring via the mutation operator: with probability equal to some mutation rate, a point in the chromosome is randomly selected in the region [0, length-2]. The chromosome is then extended from this point (grid location) as in 1.

ARTIFICIAL IMMUNE SYSTEMS

We use the clonal selection theory as formalised by (Burnet, 1959) as a method of path finding. The clonal selection theory is used to describe the basic properties of an adaptive immune response to an antigenic stimulus. Those cells that recognise an antigen are cloned, with random mutation being used to enhance affinity to the stimulus. (de Castro and Timmis, 2002). We have previously used artificial immune systems to play competitive computer games (Leen and Fyfe, 2004).

Our path finding algorithm is as follows:

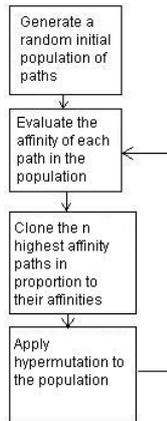


Figure 4: AIS path planning algorithm

1. Initialising the population

We first generate a random population of paths as with the path planning GA. The antibodies represent paths in the same way that the chromosomes represent paths in the GA.

2. Fitness evaluation

We then evaluate the affinity of each path in the population using a fitness function.

3. Cloning

The paths with the highest affinity are then cloned with probability in proportion to their affinities, so that the number of clones made is proportional to the affinity. This is similar to the Roulette Wheel Selection used in the path planning GA.

4. Mutation

We then apply mutation to the population at a rate inversely proportional to the affinity such that the best adapted paths to the solution are mutated with a small probability and vice versa. This fitness dependent mutation is represented in the algorithm as:

$$\text{mutation_rate} * (1 - \text{fitness}) + \beta$$

We use a fitness function where the maximum fitness is 1, and $\beta < 1$ is the probability that a solution with fitness 1 will be mutated with the mutation rate.

ANT COLONY OPTIMISATION

It is found that even though every single ant in an ant colony has its own agenda, the whole ant colony is organised. A global order emerges from processes and interaction at a local level, without any supervision; in effect, an ant colony can be seen as a decentralised problem solving system. This self organisation can be seen in the case of foraging in ants. Individual ants deposit a chemical substance called pheromone

as they move from a food source to their nest, leaving a pheromone trail for other ants to follow.

The method used relies on:

Positive feedback through trail laying and trail following. Each ant follows pheromone trails to the food source which have the highest concentration of pheromone. Since it takes less time to traverse the shortest paths, there will be a greater amount of pheromone deposited on those trails. More ants will then be recruited to these trails, so the whole ant colony will eventually use shortest paths to the food source.

Negative feedback through pheromone evaporation. This counterbalances the positive feedback.

Randomness. We let the ants follow trails with some level of error so that it is possible to find undiscovered better paths – i.e. so that the system does not get trapped in sub-optima.

Ant colony path planning

Each ant is placed on a randomly chosen square at the START side of the grid and can act according to the following behaviours:

Wandering:

The ant tries to find the food source. It can move left, right, or forward with equal probability, providing that there is not an obstacle in its path. If it reaches the food source, it then starts to return to the nest (homing behaviour).

Homing:

The ant tries to get back to the nest. It can move left, right or back with equal probability, providing that there is not an obstacle in its path. It deposits a decreasing amount of pheromone along its path. If it reaches the nest, it then starts to return to the food source.

Following:

If the ant is on a square of the grid that contains a concentration of pheromone larger than some predefined amount PHERORATE then:

If the square forward to it contains pheromone greater than PHERORATE, the ant moves to this square with probability 0.5

OR

If the square to the left of it contains pheromone greater than PHERORATE, the ant moves to this square with probability 0.5

ELSE

If possible, the ant moves to the square on its right

If the ant reaches the food source, it then starts to return to the nest (homing behaviour)

For each time step, the pheromone concentration on each square decreases by some factor.

EXPERIMENTS

Finding any possible path (GA and AIS)

We ran each simulation for 10000 generations/iterations, using a fitness function of

$$f(d) = \frac{1}{d + 1}$$

where d is the distance of the end point of the path from the FINISH side.

GA parameters: Population size: 100, Crossover rate: 0.01, Mutation rate: 0.01

AIS parameters: Population size: 100, Mutation rate: 0.1, β : 0.01

Table 1 shows average values and standard deviation of the time taken to find a path to the finish. Both algorithms find a possible path; the artificial immune system is somewhat faster.

| | Mean generation which first found path to finish | σ |
|-----|--|----------|
| GA | 37.0 | 47.8 |
| AIS | 19.2 | 8.2 |

Table 1. Mean and standard deviation of time taken to find path to finish

It was found that the both the GA and AIS converge to a solution that finds a path to the FINISH side. Figure 5 shows the typical evolution of the mean and best fitness with each generation / iteration, and shows that both path planning algorithms quickly converge to valid possible paths across the terrain (i.e. have fitness of 1).

The crossover and mutation operators act as a way of exploring the search space for the GA. The exploration is done by the mutation operator in the AIS. The Roulette wheel selection enables exploitation of good solutions for the GA while the clone operator performs the same function in the AIS.

The AIS outperformed the GA. However it may be possible that the GA solutions are in some way better than those found by the AIS. We thus, in the next section, penalise poor quality solutions i.e. longer meandering solutions.

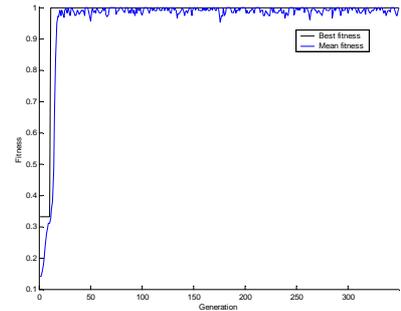
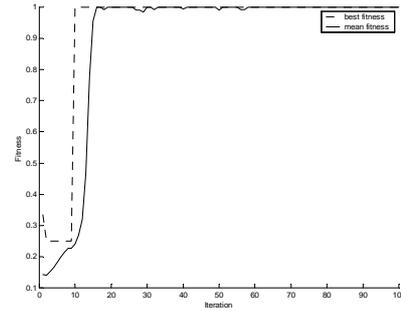


Figure 5: The evolution of mean and best fitness of the population for the AIS (top) and GA (bottom) path planning algorithms with each iteration and generation respectively. Those solutions with fitness 1 represent possible valid paths that connect the START to the FINISH side of the terrain.

Finding the shortest possible path(GA and AIS)

To find the shortest possible path from the START side of the terrain to the FINISH side, we modified the fitness function used in the previous section to include a penalty for longer paths:

$$f(d,l) = \frac{1}{d + 1} - \alpha l$$

where d is the distance of the end point of the path from the FINISH side, and l is the length of the path.

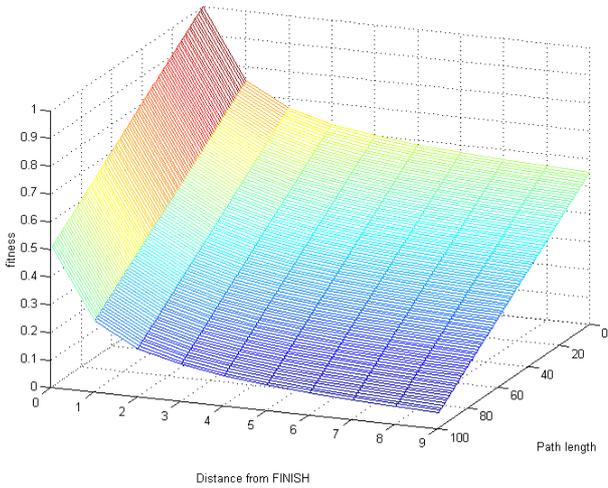


Figure 6: Fitness function used to find the shortest possible path across the terrain

GA parameters: Population size: 100, Crossover rate: 0.01, Mutation rate 0.01

AIS parameters: Population size: 100, Mutation rate 0.1, β : 0.01

We ran each simulation for 100 iterations, using $\alpha = 0.01$ giving a fitness function of

$$f(d, l) = 0.5 \left(1 + \frac{1}{d+1} - 0.01l \right)$$

which has been scaled to give fitness values between 0 and 1, as shown in Figure 6.

| algorithm | Average length of best shortest path |
|-----------|--------------------------------------|
| GA | 23.5 |
| AIS | 21.8 |

Table 2. Both algorithms find similar lengths of paths with the AIS again somewhat better.

Figure 7 shows the best paths found by the GA at generations 500, 3000, 8000. Note that while we have shown the path as crossing the terrain from location 49 to location 50 in the top two diagrams, the actual route uses the wrap around effect due the map lying on a cylinder.

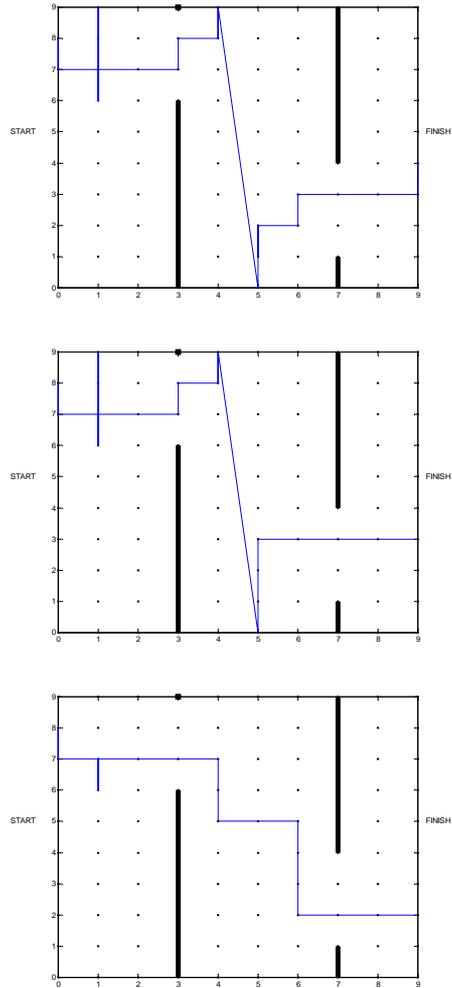


Figure 7 The best path found by the GA at generations 500(top), 3000 (middle) and 8000 (bottom).

Figure 8 shows the best paths found by the AIS at iterations 10, 200, 600 and 800. Note that both the final best GA path and the final best AIS path both contain an irrelevant lateral movement. Such movements contribute nothing to the fitness of the chromosome or antibody (indeed are penalised under the new fitness function) but are difficult to mutate out of the path with the current definition of the operators.

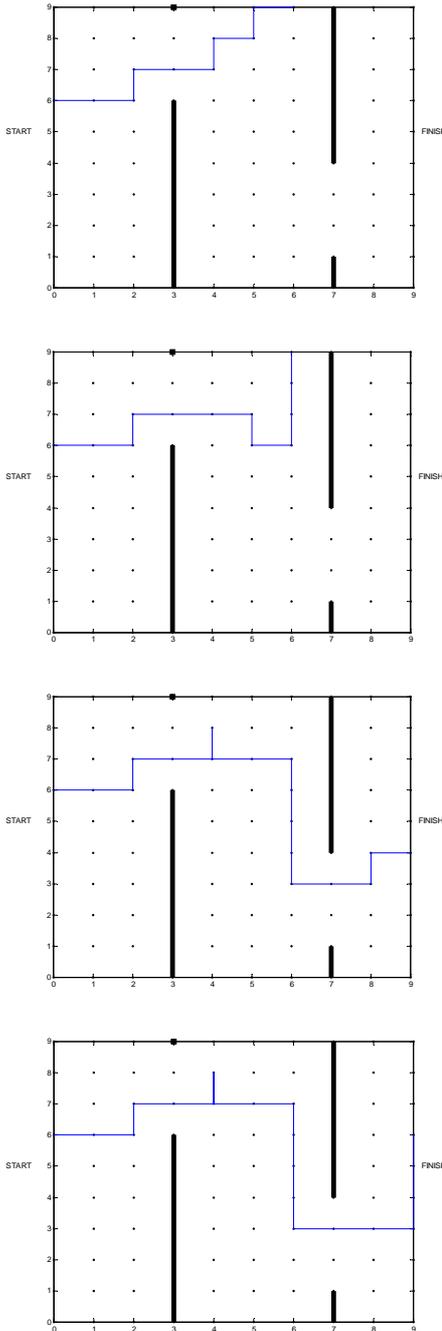


Figure 8. The best paths found by the AIS at iterations 10, 200, 600 and 800 respectively from top to bottom.

Both algorithms do not find the optimal shortest path for the terrain. We also find that due to the way that the mutation operator is defined in both cases, that the algorithms cannot correct sub-optimal (i.e. not the shortest) sections of path near to the START side of the terrain.

Using Ant colony optimisation

The simulation was run for 1000 iterations with the following parameters:

Number of ants: 100, pheromone threshold above which ants will begin following a trail = 2, Initial amount of pheromone gained at food source = 1,

Multiplicative decrease in pheromone when leaving trail = 0.95, Multiplicative pheromone evaporation rate = 0.9

It was found that even though the ants managed to find paths between the food source and the nest, they did not recognise any optimal (shortest) path. This could be due to the high concentration of ants on the grid.

Figure 9 shows amount of pheromone at each square of the grid (the zeroes here correspond to obstacles). Note the high concentrations of pheromone on the finish side of each obstacle. This is due to the returning ants hitting the wall until by chance they find the gap. This aimless wandering, though, does leave a trail of pheromone which subsequent ants prefer to follow, thus reinforcing the aimless wandering and so on.

CONCLUSION

We have examined three path planning algorithms for static obstacle avoidance based on genetic algorithms, artificial immune systems, and ant colony optimisation. It has been found that the ACO path planning algorithm that we used proved ineffective in finding a path through our simple terrain. This is a somewhat surprising finding since it would seem a priori that such algorithms were specifically designed for just such a task but the message to the Games programming community is clear: don't use this method.

The path planning algorithms based on genetic algorithms and the clonal selection theory from the artificial immune systems field had similar performance in our simulations in that they managed to plan the desired path, but this is not surprising since they are both population based searches, and the mutation and crossover operators of the GA that we defined play a similar role to the mutation operator of the AIS. It would be of interest to refine the crossover operator for our GA path planner so that it can combine 'good' sections of suboptimal paths to create a better path, since at present, our crossover operator acts in a similar way to the mutation operator – i.e. it explores rather than exploits the search space. Whether evaluation is carried out in terms of speed of convergence or length of path, the artificial immune system slightly outperformed the genetic algorithm.

Future work will continue the comparison by employing Kohonen's Self Organizing Map (Kohonen, 1995) on this task within the same environment. We will also create new artificial environments to investigate whether our findings are repeated irrespective of the terrain or if they are terrain-dependent.

Nest (START)

| | | | | | | | | | |
|----------|----------|----------|---------|---------|---------|----------|---------|---------|---------|
| 1.40997 | 3.66252 | 2.06639 | 0 | 6.96789 | 2.49303 | 1.96E-44 | 0 | 24.4803 | 1.60084 |
| 0.908678 | 4.97126 | 5.95227 | 6.82075 | 10.0627 | 1.60749 | 0.030567 | 0 | 26.8926 | 3.53956 |
| 1.33563 | 4.40103 | 5.97537 | 8.13243 | 18.778 | 1.14525 | 0.036784 | 0 | 34.9556 | 4.57057 |
| 0.72479 | 2.07672 | 1.67946 | 0 | 29.7371 | 5.20419 | 1.29479 | 0 | 44.757 | 7.35604 |
| 0.032557 | 0.756081 | 0.497769 | 0 | 44.2334 | 8.11942 | 3.93274 | 0 | 44.6979 | 13.7203 |
| 0.170638 | 0.670383 | 0.370848 | 0 | 59.4028 | 9.65998 | 11.1574 | 0 | 43.442 | 24.5968 |
| 0.164501 | 0.214602 | 0.135284 | 0 | 75.1899 | 24.8091 | 40.7821 | 43.2066 | 67.1174 | 51.4776 |
| 0.002246 | 0.002761 | 0.142415 | 0 | 75.0543 | 22.9563 | 37.5153 | 44.459 | 78.9772 | 59.0538 |
| 1.67E-06 | 0.332616 | 0.149909 | 0 | 69.4062 | 15.5588 | 11.3653 | 0 | 87.9552 | 37.7988 |
| 0.325646 | 0.963015 | 0.857166 | 0 | 36.1294 | 14.0306 | 4.54102 | 0 | 58.0405 | 30.4376 |

Food source (FINISH)

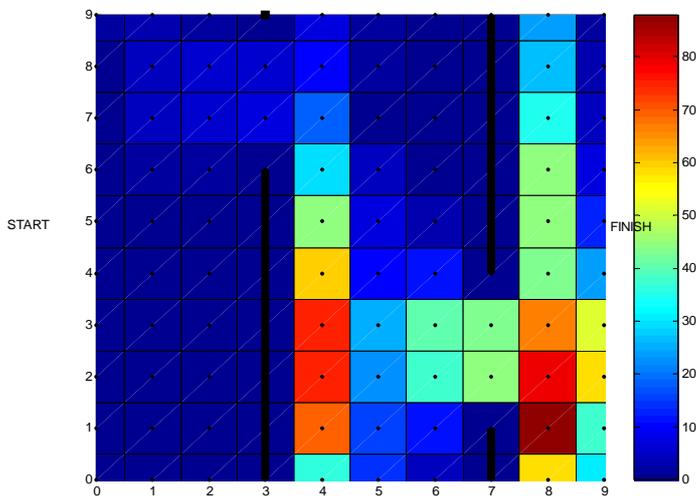


Figure 9. The pheromone deposited after the ant colony optimisation has found a route from start to finish. Note the concentrations of pheromone on the finish side of each wall.

BIBLIOGRAPHY

Burnet, F.M., (1959), *The Clonal Selection Theory of Acquired Immunity*, Cambridge University Press.

De Castro, L.N. and Timmis, J., (2002), *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer-Verlag.

Kohonen, T. 1995, *Self Organizing Maps*, Springer-Verlag.

Leen, G. and Fyfe, C (2004). Agent wars with artificial immune systems, *3rd International Conference on Entertainment Computing, ICEC2004*.

Mitchell, M., (1996). *An Introduction to Genetic Algorithms*. MIT Press.

AUTOMATIC CONSTRUCTION OF ROADMAPS FOR PATH PLANNING IN GAMES

D. Nieuwenhuisen A. Kamphuis M. Mooijekind M.H. Overmars
Institute of Information and Computing Sciences
Utrecht University
P.O. box 80089, 3508TB Utrecht, The Netherlands
Email: {dennis,arnok,mmooijek,markov}@cs.uu.nl

KEYWORDS

Path planning; Roadmap; Probabilistic Roadmap Method; Games; Camera movement; Groups

ABSTRACT

Path planning plays an important role in many computer games. Currently the motion of entities is often planned using a combination of scripting, grid-search methods, and reactive approaches. In this paper we describe a new approach, based on a technique from robotics, that computes a roadmap of smooth, collision-free, high-quality paths. This roadmap can be used to obtain instantly good paths for entities. We also describe applications of the technique for planning the motion of groups of entities and for creating smooth camera movements through an environment.

INTRODUCTION

In many computer games, entities like enemies, non playing characters (NPCs), and vehicles, must plan their motions between locations in the virtual world. Currently this is typically achieved using a combination of scripting, A^* -like grid search, and local reactive methods.

In scripting, the designer explicitly described the paths that can/must be followed by the entities. This is normally part of level design. Scripting is a time consuming process for the designer. In addition, it can lead to repetitive behavior that is easily observed by the player. Scripting gets increasingly complicated when many entities move in the same space.

Grid based methods divide the world in a grid of cells and plan motion using an A^* -like search on the free cells (see e.g. (DeLoura 2000; Russell and Norvig 1994)). It is often used in real-time strategy games where there is a natural division of the world in cells. When the world becomes complicated and large and many entities must move around, grid based methods take a large amount of computer time. Pruning the search reduces the time but might lead to wrong paths. Also, motions created by grid search tend to be unnatural, as can be observed in many RTS games.

Reactive methods adapt a previously computed motion to ob-

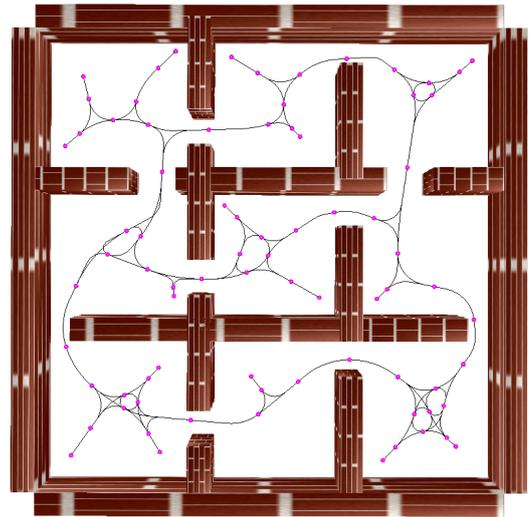


Figure 1: An example of a smooth roadmap computed by our technique.

stacles found near the path that were not taken into account during initial planning; for example other entities or small, movable objects (see e.g. (Lamiroux and D. Bonnafous 2004; Stout 1996; Baert 2000; Pinter 2001)). Even though theoretically reactive methods can be used to compute full paths, this normally leads to dead-lock situations in which the entities no longer know where to move (e.g. they get stuck in a corner of the room). Another problem with reactive methods is that it is often difficult to adapt the internal animation of the character to the motions produced.

In robotics, many other path planning approaches have been developed that might be applicable to path planning in games. In robotics though, the emphasis is often on the motion of a complicated robotic system in a relatively simple environment. In games the opposite is true. From a path planning perspective, the entity can often be modeled as a simple vertical cylinder, while the environment can be very complicated with tens of thousands of obstacles.

One popular path planning technique in robotics is the Probabilistic Roadmap Method (PRM). It has been studied by many authors, see e.g. (Kavraki et al. 1996; Švestka and Overmars 1998; Kavraki and Latombe 1994; Holleman and Kavraki 2000). In a preprocessing phase this method builds a roadmap of possible motions of the robot through the environment. When a particular path planning query must be solved, a path is retrieved from this roadmap using a simple



(a) A group of five characters should attack the site pointed to by the arrow.



(b) The group inappropriately splits up, losing some of its troops.

Figure 2: One of the problems with the current techniques for motion planning for multiple units is that the group splits up to reach the goal. This scene was taken from *Command and Conquer: Generals* from EA Games.

and fast graph search. The PRM approach is suited for very complicated environments. Unfortunately, the roadmap produced by the method can be rather wild, leading to ugly paths that require a lot of time-consuming smoothing to be useful for gaming applications.

In this paper we will describe a new path planning approach, building on the PRM method, that can be effectively applied in gaming applications. The approach also constructs a roadmap of possible motions but guarantees that the paths are short, have enough clearance from the obstacles, and are C^1 continuous, leading to natural looking motions. See Figure 1 for an example of a roadmap computed with our approach. After building the roadmap, which can be done as a preprocessing phase, paths can be retrieved almost instantaneously, and do not require any post processing.

Besides the standard application, in which the roadmap is used for planning the paths for individual entities, we describe two additional applications. First, we consider the motion of groups of entities. In games, this problem is often solved using a combination of grid-based planning and flocking (Reynolds 1987; Reynolds 1999). Unfortunately, this can lead to unwanted behavior where the group of entities splits up (see Figure 2 for an example). We will use the smooth paths computed by the new planning approach as a backbone path and then use a social potential field approach to guide the flock through a corridor around this path (extending our earlier work in (Kamphuis and Overmars 2004)). This results in a naturally looking motion in which the group is guaranteed to stay together.

In games also the virtual camera, through which we observe the world, moves through the environment. Currently the camera is often under direct control of the user (in first person games) or under indirect control of the user (in third person games). Direct camera control is difficult, easily leads to motion sickness due to redundant motions, and is often not required. Building on our earlier work in (Nieuwenhuisen and Overmars 2004a) we describe a method in which the user only

specifies positions of interest and the camera automatically moves to such positions using a smooth, collision free motion. For computing the camera path we will use the new planning approach described. This is then combined with techniques to control the view direction and the speed of the camera to obtain a camera motion that is pleasant to watch.

ROADMAP GENERATION

In this section we will describe how, in a preprocessing phase, a roadmap of possible motions for the entities can be computed. A roadmap is normally represented as a graph in which the nodes correspond to placements of the entity and the edges represent collision-free paths between these placements. A standard technique for automatic roadmap creation is the probabilistic roadmap method (PRM).

Unfortunately, the PRM method leads to low quality roadmaps that can take long detours. This is due to the random nature of the PRM method. Techniques exist to improve paths in a post-processing stage but this is time-consuming and can still lead to long detours. Here we present a variant of the PRM method that leads to short, smooth and high quality roadmaps. These roadmaps can then during the game be used to solve path planning queries almost instantaneous using a simple shortest path graph search algorithm (for example Dijkstra's shortest path algorithm).

In the preprocessing phase we create the roadmap graph, consisting of vertices (V) and edges (E). For the placement of the entity we only take its position (x, y) into account since these are the only parameters that are important for planning the path. Later, the other parameters (such as orientation) can be added depending on the application. The edges of the graph will represent straight line and circular paths between the vertices. Only vertices and edges that are collision free are allowed in the graph.

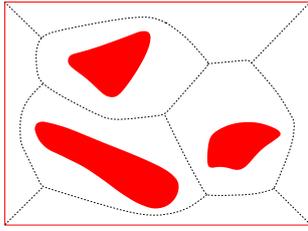


Figure 3: An example of a Voronoi diagram. We treat all four boundaries of the workspace as separate obstacles.

In order to be able to use the graph for as many different queries as possible, we need a good coverage of the space. Many improvements for the PRM method have been proposed in order to achieve this (see e.g. (Bohlin and Kavraki 2000; Boor et al. 1999; Nissoux et al. 1999; Wilmarth et al. 1999; Hsu et al. 2003; Branicky et al. 2001; Isto 2002)), but all are based on the same underlying concept and lead to roadmaps consisting of straight line segments only that result in low path quality.

For a roadmap that is used to steer entities in games, we can formulate the following criteria:

- The paths generated by the roadmap should always keep some minimum amount of clearance from the obstacles in the scene.
- The paths should be smooth i.e. it should be C^1 continuous.
- A path needs to be created very fast (not delaying the motion) and should be short, not taking any detours.

Creating Samples on the Voronoi Diagram

The Voronoi diagram of a scene defines for every obstacle a set of points in the free space that are closer to this obstacle than to any other obstacle in the scene; together these points form the Voronoi diagram (see Figure 3 for an example). An important property of the voronoi diagram is that it maximizes the clearance with the obstacles and thus, given the criteria of the previous section, it can very well be used to improve roadmap quality.

Here, we propose a new variant of the PRM method that uses the Voronoi diagram as a guide. The method works as follows. In every iteration of the algorithm we randomly pick a sample (placement) of the entity, we call it c . Then we check whether c is collision free for the entity. If this is the case, we continue by retracting it to the Voronoi diagram using the following procedure. We calculate the closest point on an obstacle from c , we call this point c_c (Figure 4(a)). Next, another sample c' is moved from c in the opposite direction of c_c using as a step size the distance between c and c_c (Figure 4(b)). We proceed until the closest obstacle to c' changes. We now have two samples c and c' , both having another closest obstacle. The above procedure guarantees that the Voronoi diagram passes through a point between c and c' .

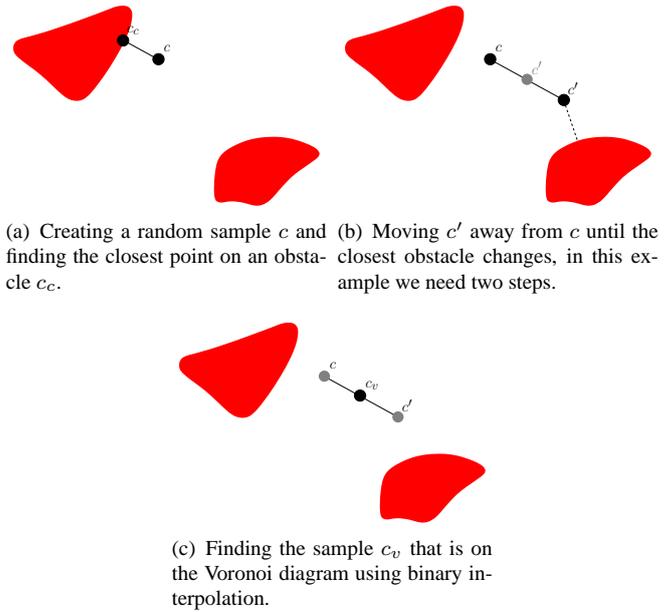


Figure 4: Retracting samples to the Voronoi diagram.

We continue by using binary search between c and c' , with precision ϵ until we have found a sample that has two obstacles at the same distance. This sample, called c_v , is at most a distance ϵ away from the Voronoi diagram (Figure 4(c)). Now, we add c_v to the list of vertices V in the roadmap graph.

After adding a sample as a vertex to the graph, we determine its neighbor vertices. The set of neighbor vertices of vertex v , called N_v is defined as all vertices V that are closer to v than some chosen maximum neighbor distance. For each vertex v_n in N_v we test whether the straight line connection between v and v_n is collision free. If this is the case, then we add the connection (v, v_n) as an edge to the set of edges E of the graph. If two vertices are already connected in the graph (via other vertices), then we only add the new edge if the path between v and v_n is shortened considerably by at least some constant K (for more details see (Nieuwenhuisen and Overmars 2004b)).

Retracting Edges

We have retracted the nodes of the graph to the Voronoi diagram (within a certain boundary ϵ) but when connecting the samples with edges, these edges are usually not on the Voronoi diagram and can get very close to obstacles (Figure 5(a)). In order to solve this problem, edges are retracted to the Voronoi diagram until every part of the edge is at least some pre-specified distance away from the obstacles. We achieve this by proceeding in the following manner: if (a part of) an edge is too close to an obstacle, this edge is split in two equal length parts and the middle point is retracted to the Voronoi diagram using the same procedure as described in the previous section. This procedure is recursively repeated for the two new edges until every edge has enough clearance with the obstacles. An example of this procedure is shown in Figure 5. In some cases (when the edge passes through a very narrow corridor), the clearance threshold will never be reached and the edge will

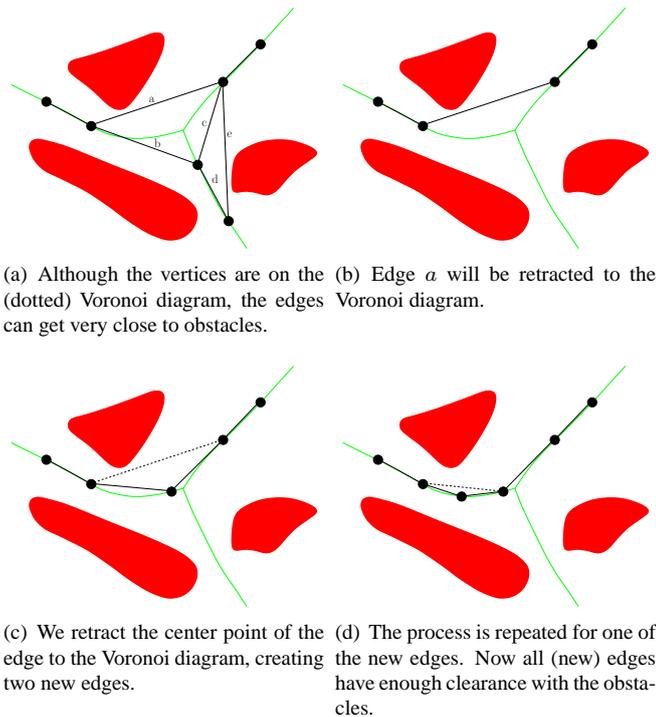


Figure 5: Retracting an edge to the Voronoi diagram.

be split an infinite number of times. In order to prevent this, we stop retracting the edge if its length is shorter than some predefined value.

Retracting edges to the Voronoi diagram may result in some edges overlapping each other. For example in Figure 5(a) if edges c and d are already retracted, then retracting e will result in overlap. Fortunately, detecting overlapping edges is easy. For every pair of edges e_i and e_j , we check how far their endpoints are away from the other edge. If this distance is smaller than some predefined distance, then we try to project the vertices of e_i on e_j and vice versa. If at least one of these projections is successful, we call the two edges overlapping and we can join them. We can distinguish four different kinds of overlapping edges. In Figure 6 these are shown together with the situation after removing the overlap.

Improving the Roadmap

In the previous sections, we optimized the clearance in the roadmap. It can be improved further though. In particular we would like to make sure that a path runs through every corridor between the obstacles. This can be achieved by applying some of the known techniques for finding paths in narrow corridors (see e.g. (Geraerts and Overmars 2004; Boor et al. 1999; Hsu et al. 2003)). On the other hand, allowing cycles in the graph in a controlled manner can lead to the same result (Nieuwenhuisen and Overmars 2004b).

If a vertex v has only two neighbor vertices, it may be possible to directly connect those two neighbors, bypassing v . This would lead to longer edges and, thus, less (unnecessary) rotations. We remove these vertices if the merged edge is collision

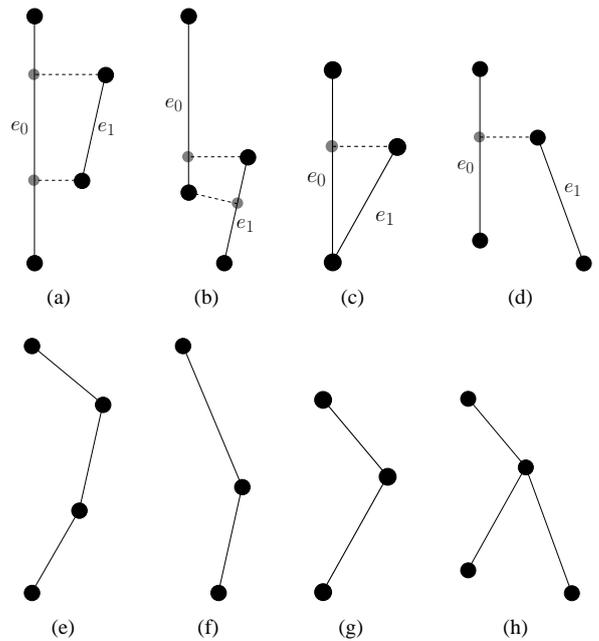


Figure 6: Merging two overlapping edges e_0 and e_1 . Four different cases can be distinguished (a..d). The results after merging are shown in (e..h).

free and has enough clearance.

Circular Blends

After retracting the vertices of the graph to the Voronoi diagram and after adding some minimal amount of clearance to the edges we still end up with a graph that consists of straight line segments. Following such a path will have C^1 discontinuities at the vertices that cause sudden directional changes to an object that follows the path. In order to solve this problem we will replace parts of the straight line edges by circular blends.

The degree of a vertex is defined as the number of edges that is connected to this vertex. If a vertex has degree 1, it is an endpoint of a path segment, and no circular blend needs to be added. If a vertex has degree 2, the addition of the circular blend is straightforward. We find the centers of the two edges, and use these to create a circle arc that touches both edges. Now, we use this to replace a part of the path (see Figure 7(a)). If the degree of a vertex v is higher than 2, we find the centers of all incoming edges. We now add a blend for every pair of these centers (see Figure 7(b) for an example of a vertex with degree 3).

In the previous section, we retracted the edges of the graph to the Voronoi until they had at least some predefined clearance. Adding circular blends may decrease this clearance. Since we do not want the clearance to be lower than some predefined value, we check the minimum clearance of each circular blend. If it is too low, then we replace the blend by another blend that has a smaller radius. We repeat this until the blend has enough clearance. This procedure is shown in Figure 7(c).

Figure 1 shows an example of a typical roadmap graph created

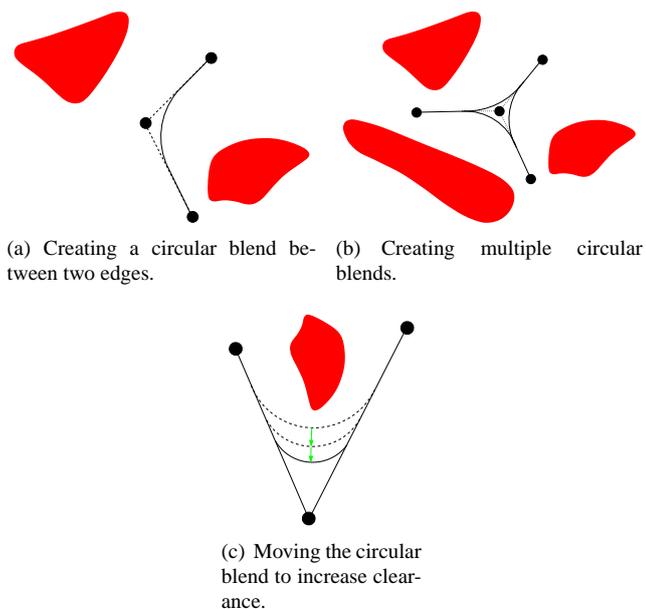


Figure 7: Creating the circular blends.

with this method. Computing this roadmap took about 1 second on a Pentium IV 2.4 GHz. Realize though that roadmaps can be computed during the creation of the scene and can easily be stored with it.

Answering Path Queries

During the game, whenever an entity has to move to a new location, it can search the graph to plan its route through the environment. Because of the properties of the roadmap, paths will be short, have enough clearance from the obstacles, and are smooth. To obtain a path we first need to connect the current position of the entity to the graph. This can easily be done by finding the closest vertex in the graph and connecting the current placement to this vertex using circular blends. We proceed in the same way for the goal placement of the entity. Now we can use a shortest path algorithm in the graph to find the path between the current and goal positions. See Figure 8 for an example of such a path.

Computing paths during game play is extremely fast. Even for a large roadmap graph consisting of 1000 vertices and 3000 edges, the calculation of the shortest path takes less than 10ms on a Pentium IV 2.4GHz.

PATH PLANNING FOR GROUPS

Games are often populated with a large number of moving entities. The entities should often behave as a coherent group rather than as individuals. For example, one needs to simulate the behavior of whole army divisions. Current games solve the problem of path finding on the entity level, i.e. they plan the motion of individual entities, using techniques like flocking to keep the entities together. However, in cluttered environments this often leads to non-coherent groups. There is no guarantee

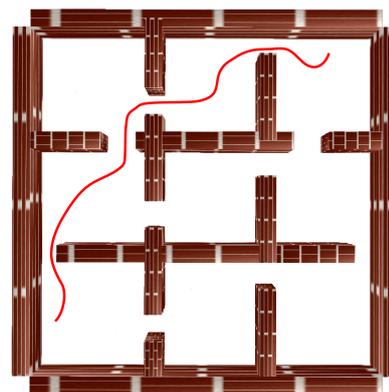


Figure 8: An example of a resulting path.

that the entities will stay together, albeit that 'staying together' is not well defined. Even though the entities all have a similar goal, they try to reach this goal without real coherence. This results in groups splitting up and taking different paths to the goal, for example as in Figure 2.

We will briefly describe a novel technique in which groups do stay together. More details can be found in (Kamphuis and Overmars 2004). We are given a game level in which a group of entities must move from a given start to a given goal position. The entities must avoid collisions with the environment and with each other, and should stay together as one group. The entities are modeled as discs (or cylinders) and are assumed to move on a plane or terrain. Later, the resulting paths for the cylinders can be used to animate avatars, e.g. sprites or motion captured human-like avatars.

The method works as follows: First, a so-called *backbone path* for a single entity is computed. This path defines the homotopic class used by all entities. Two paths P_0 and P_1 are said to be in the same homotopic class only if P_0 can be continuously deformed into P_1 without intersecting the obstacles. Next, a *corridor* is defined around the backbone path in which all entities must stay. Finally, the movement of the entities is generated using force fields with attraction points on the backbone path. By limiting the distance between the attraction points for the different entities, coherence of the group is guaranteed.

Backbone Path Planning and the Corridor

The first phase of the approach consists of finding the backbone path. Since every entity should be able to traverse the path, the clearance on the path should be bounded by a minimum value, namely the radius of the enclosing circle/cylinder of the largest entity. The backbone path can thus be defined as follows: A backbone path is a path in the 2D workspace, where the clearance at every point on the path is at least the radius of the enclosing circle/cylinder.

Although finding a path with a minimum clearance of the radius of the enclosing circle is required, we prefer a larger clearance, since a larger clearance leads to behavior that is more coherent. Also we prefer the paths to be smooth and short. Hence, the paths in the roadmap created with the method de-

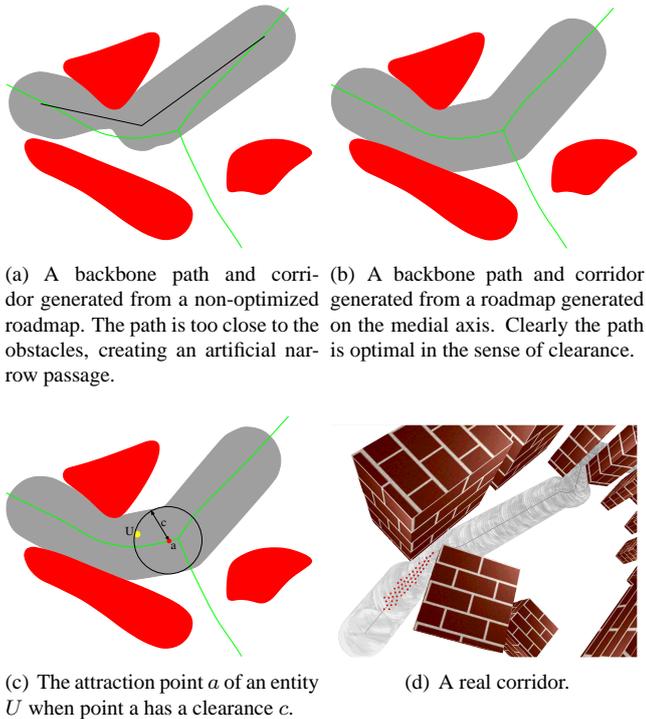


Figure 9: The corridor

scribed above are very well suited for this application.

From the backbone path, a corridor is created. For this we use the clearance along the path. On every point along the path, the clearance is defined as the radius of the largest circle around the point that does not intersect with the environment. The value of the clearance is upper bound by the maximum group width, i.e. the clearance used can never exceed the maximum group width. The union of all the upper-bounded clearance circles forms a corridor around the backbone path. Figure 9(a) shows a path generated with a technique that produces paths too close to obstacles, resulting in artificial narrow passages. In contrast, the path in Figure 9(b) was generated with the roadmap approach described above, and lies far from obstacles. The resulting corridor is much more natural.

Generating the Motion inside the Corridor

Once the corridor is created, we need to use it to generate the motion of the individual entities. The approach used is an artificial force field technique. Forces are defined that act on the entities and influence their movement. Every entity in the group has a corresponding attraction point on the backbone path. This attraction point is selected as the maximum advanced point p along the backbone such that the entity is still inside the circle centered at that point p with radius equal to the clearance at p (see Figure 9(c)). The attraction points make the entities move forward and keep the entities inside the corridor. The entities also repulse each other to avoid collisions between them. Additional forces could be incorporated, for example to accomplish formations.

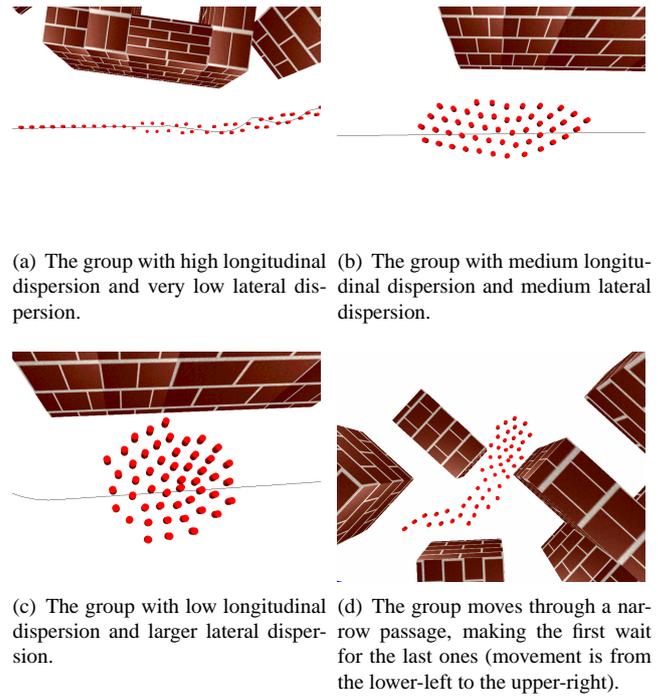


Figure 10: A group of 50 entities moving in a virtual world. These paths and behaviors are created with the same approach, only by varying the parameters.

Keeping Coherence in the Group

In order to keep the group coherent, the dispersion should be upper bounded. Due to the manner in which the corridor is constructed, the lateral dispersion (dispersion perpendicular to the backbone path) is automatically upper bounded by the group width. However, the longitudinal dispersion (in the direction of the backbone path) is not yet bounded in this approach. To achieve this, the distance along the path from the least advanced attraction point to the most advanced attraction point is limited. This results in the entities in front waiting for the entities at the back.

Results

The behavior of the group can be controlled by adjusting the coherence parameters, *lateral dispersion* and *longitudinal dispersion*. Figures 10(a) to 10(c) show a group of 50 entities moving through an environment. In these pictures the lateral and longitudinal dispersion is varied, resulting in a longer, more stretched group (10(a)) or more compact group (10(c)).

Figure 10(d) shows the same group moving through the environment from the left lower corner to the right upper corner. The most advanced entities, i.e. the entities that passed the narrow passage earliest, wait for the last entity to pass the passage.

We tested the performance of the approach to show that the technique is usable in real-time applications as computer games. For this, we developed a typical implementation. In this implementation we created numerous paths. The proces-

processor usage during the creation of the paths was minimal. For groups of 50 to 100 entities the processor usage did not exceed 1 percent. More efficient implementations could further decrease the processor usage.

PLANNING CAMERA MOTIONS

Every game has a camera through which the player views the world. Usually this camera moves depending on user input and place of action. A camera motion directs the camera from one position to another while controlling camera speed and view direction. There are many situations in which an automatic camera motion in a game could be of great use. Think for example about an RPG where the user has almost completed a level, but wants to get back to the start of that level to pick something up she forgot. Without automatic camera motions, the attention of the user is needed to walk the whole way back just to get one item. Also in many adventures it would be nice to be able to specify a location in the interface and have the game create a fast and efficient motion to that location without warping directly to it (causing the user to get disoriented).

The roadmap from the previous section can easily be used to steer a camera. Using this roadmap, the camera is guaranteed to keep a certain amount of clearance from the obstacles and the circular arcs make sure that the camera motion is gentle. The roadmap alone however is not enough to create a smooth camera motion. Camera theory (Millerson 1973; Wayne 1997) shows that we need to take care of two more variables. First, the speed of the camera should be adapted according to the curvature of the path. Otherwise, objects will move too fast through the view. Second, the user should get cues about where the camera is going. In particular the viewer should be able to anticipate a camera rotation. We will resolve these issues in the next two sections.

Adapting the Camera Speed

Smoothness of the path is not enough for a smooth camera motion. The speed of the camera along the path should be adapted according to the curvature of the path. Also there should be a maximum acceleration and deceleration for the camera in order to prevent too abrupt speed changes.

Since our path consists of straight lines and circle arcs, we can adapt the speed of our camera by making use of the radius of the arcs. The smaller the radius, the lower the camera speed. When the camera leaves an arc with a small radius, we accelerate until we have reached the maximum speed of the current arc or straight line. If, on the other hand, the next arc requires a lower speed than the current camera speed, we must start decelerating on time, such that when we reach the next arc, our speed is sufficiently low. A speed diagram can be computed efficiently that satisfies both the constraints on the maximal speed for each arc and the bounds on acceleration and deceleration. See Figure 11 for an example of such a speed diagram for a simple path.

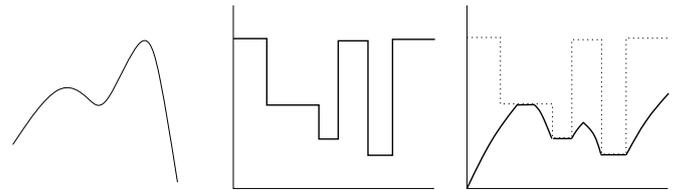


Figure 11: A speed diagram. The left image shows the path, the middle image shows the maximal speed allowed at each position. The right image shows the actual speed, taking acceleration and deceleration bounds into account.



Figure 12: An implementation of the techniques. The user can click on a location in the map at the left top and the program creates a smooth camera motion to that location.

Smoothing the Viewing Direction

Intuitively one might think that the viewing direction should be equal to the direction of the camera motion. As stated before however, the user should be given cues about where the camera is heading. We can achieve this by always looking at the position the camera will be in a short time. Experiments show that about 1 second is the right amount. Note that, as we fix the time we look ahead, the distance we look ahead changes depending on the speed of the camera. This is exactly what we want to achieve as in sharp turns we want to look at a nearer point than in wide turns. Looking ahead has another important effect. If we would look in the direction of motion and the camera reaches a circular arc, then it suddenly starts rotating at the start of the arc. Stated more formally, the rotation of the camera is only C^0 continuous. It can be proved that looking ahead solves this issue by making the camera rotation C^1 continuous.

Results

We implemented this approach in a walk-through system for virtual worlds. See Figure 12 for a screenshot. Rather than letting the user steer the camera directly, we display a map in the top left corner. By clicking on the map the user indicates the position she wants to move to. A smooth camera motion is then calculated in the way described above. The processor time required for this is minimal. Experiments indicate that this is a pleasant way to inspect the environment.

CONCLUSIONS

In this paper we have described a new technique for automatic construction of high-quality roadmaps in virtual environments and we have shown how these can be used to plan the motion for individual entities, groups of entities, and the camera through which we observe the world.

We described our method as a 2-dimensional approach in which entities move on a ground plane. It is though easy to extend it to e.g. terrains and even movement in buildings in which the roadmap would automatically follow the corridors and stairs.

Roadmap construction is best seen as being part of the construction of the virtual world. It is easy to incorporate special requirements from the level designer. For example, the designer can add fake obstacles to force the path to e.g. stay on the sidewalks of a road. Also the designer can easily manipulate the roadmap graph by manually adding, changing, or removing nodes. Moreover, weights can be added to the graph to e.g. indicate preferred routes.

ACKNOWLEDGEMENTS

This research was supported by the Dutch Organization for Scientific Research (N.W.O.). This research was also supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments).

REFERENCES

- Baert, S. (2000). Motion planning using potential fields. *gamedev.net*. url: <http://www.gamedev.net>.
- Bohlin, R. and L. Kavraki (2000). Path planning using lazy prm. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 521–528.
- Boor, V., M. Overmars, and A. van der Stappen (1999). The gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1018–1023.
- Branicky, M., S. Lavalley, K. Olson, and L. Yang (2001). Quasi randomized path planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*.
- DeLoura, M. (Ed.) (2000). *Game Programming Gems 1*. Charles River Media.
- Geraerts, R. and M. Overmars (2004). A comparative study of probabilistic roadmap planners. In *Algorithmic Foundations of Robotics V, Springer Tracts in Advanced Robotics 7*, pp. 43–57. Springer-Verlag Berlin Heidelberg.
- Holleman, C. and L. Kavraki (2000). A framework for using the workspace medial axis in prm planners. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Volume 2, pp. 1408–1413.
- Hsu, D., T. Jiang, J. Reif, and Z. Sun (2003). The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Robotics and Automation*.
- Isto, P. (2002). Constructing probabilistic roadmaps with powerful local planning and path optimization. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2323–2328.
- Kamphuis, A. and M. H. Overmars (2004, August). Finding paths for coherent groups using clearance. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)*, pp. to appear.
- Kavraki, L. and J.-C. Latombe (1994). Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2138–2139. IEEE Press, San Diego, CA.
- Kavraki, L., P. Švestka, J.-C. Latombe, and M. Overmars (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, 556–580.
- Lamiroux, F. and O. L. D. Bonnafofus (2004). Reactive path deformation for nonholonomic mobile robots. In *IEEE Transactions on Robotics*, pp. to appear.
- Millerson, G. (1973). *TV Camera Operation*. Focal Press, London. ISBN: 0-24050-850-5.
- Nieuwenhuisen, D. and M. Overmars (2004a). Motion planning for camera movements. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 3870–3876. IEEE Press, San Diego, CA.
- Nieuwenhuisen, D. and M. Overmars (2004b). Useful cycles in probabilistic roadmap graphs. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 446–452. IEEE Press, San Diego, CA.
- Nissoux, C., T. Siméon, and J.-P. Laumond (1999). Visibility based probabilistic roadmaps. In *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, pp. 1316–1321.
- Pinter, M. (2001, March). Toward more realistic pathfinding. *gamasutra.com*. url: <http://www.gamasutra.com>.
- Reynolds, C. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics 21(4)*, 25–34.
- Reynolds, C. (1999). Steering behaviors for autonomous characters. In *Game Developers Conference*.
- Russell, S. and P. Norvig (1994). *Artificial Intelligence: A Modern Approach*. Prentice Hall.
- Stout, W. (1996, October). Smart moves: Intelligent pathfinding. *Game Developer*.
- Švestka, P. and M. Overmars (1998). Coordinated path planning for multiple robots. *Robotics and Autonomous Systems 23*, 125–152.
- Wayne, M. (1997). *Theorising Video Practice*. Lawrence and Wishart, London. ISBN: 0-85315-827-4.
- Wilmarth, S., N. Amato, and P. Stiller (1999). Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1024–1031.

USING VALUE ITERATION TO SOLVE SEQUENTIAL DECISION PROBLEMS IN GAMES

Thomas Hartley, Quasim Mehdi, Norman Gough
The Research Institute in Advanced Technologies (RIATec)
School of Computing and Information Technology
University Of Wolverhampton, UK, WV1 1EL
E-mail: T.Hartley2@wlv.ac.uk

KEYWORDS

Value iteration, artificial intelligence (AI), AI in computer games.

ABSTRACT

Solving sequential decision problems in computer games, such as non-player character (NPC) navigation, can be quite a complex task. Current games tend to rely on scripts and finite state machines (FSM) to control AI opponents. These approaches however have shortcomings; as a result academic AI techniques may be a more desirable solution to solve these types of problems. This paper describes the process of applying the value iteration algorithm to an AI engine, which can be applied to a computer game. We also introduce a new stopping criterion called game value iteration, which has been designed for use in 2D real time computer games and we discuss results from experiments conducted on the AI engine. We also outline our conclusions which state that the value iteration and the newly introduced game value iteration algorithms can be successfully applied to intelligent NPC behaviour in computer games; however there are certain problems, such as execution speed, which need to be addressed when dealing with real time games.

INTRODUCTION

Whilst playing computer games online against human opponents, it became apparent that it was a more interesting playing experience, than that of playing against non-player characters (NPCs). The human opponents were more difficult to anticipate and were more challenging, in comparison to their NPC counterparts. As a result, we tend only to play the single player aspect of a computer game a handful of times before we feel the game's gameplay becomes predictable and easy to beat.

This is backed up by Jonathan Schaeffer (2001 in Spronck *et al.*, 2003) who states that the general dissatisfaction of game players with the current levels of AI for computer controlled opponents makes them prefer human controlled opponents. Currently commercial computer game AI is almost exclusively controlled by complex "manually-designed scripts" (Spronck *et al.*, 2002). This can result in poor AI or "Artificial Stupidity" (Schaeffer, 2001 in Spronck *et al.*, 2002).

The predictability and any "holes" within a scripted computer game can then be exploited by the human player (Spronck *et al.*, 2002). The game industry is however constantly involved in employing more sophisticated techniques for NPCs (Kellis, 2002), especially in light of the increase in personal PC power, which enables more time to be spent processing AI. Recent games, such as Black &

White (Lionhead, 2001) use learning techniques to create unpredictable and unscripted actions. However most games still do rely on scripts and would benefit from an improvement in their AI.

These observations formed the basis of a research project into the field of AI and computer game AI. The aims of this project were to research computer games in order to shed light on where computer game AI can be poor and to research AI techniques to see if they might be able to be used to improve a computer game's AI. The objectives of the project were the delivery of a computer game AI tool that demonstrated how an AI technique could be implemented as an AI engine and a computer game that demonstrated the engine. This paper demonstrates how Markov decision processes can be applied to a computer game AI engine, with the intention of showing that this technique will be a useful alternative to scripted approaches. This paper covers the implementation of the AI engine; the implementation of the computer game will be covered in our next paper.

MARKOV DECISION PROCESSES

Markov decision processes (MDPs) are a mathematical framework for modelling sequential decision tasks / problems (Bonet, 2002) under uncertainty. According to Russell and Norvig, (1995), Kristensen (1996) and Pashenkova and Rish (1996) early work conducted on the subject was by R. Bellman (1957) and R. A. Howard (1960).

The technique works by splitting an environment into a set of states. An NPC moves from one state to another until a terminal state is reached. All information about each state in the environment is fully accessible to the NPC. Each state transition is independent of the previous environment states or agent actions (Kaelbling and Littman, 1996). An NPC observes the current state of the environment and chooses an action. Nondeterministic effects of actions are described by the set of transition probabilities (Pashenkova and Rish, 1996). These transition probabilities or a transition model (Russell and Norvig, 1995) are a set of probabilities associated with the possible transitions between states after any given action (Russell and Norvig, 1995). For example the probability of moving in one direction could be 0.8, but there is a chance of moving right or left, each at a probability of 0.1. There is a reward value for each state (or cell) in the environment. This value gives an immediate reward for being in a specific state.

A policy is a complete mapping from states to actions (Russell and Norvig, 1995). A policy is like a plan, because it is generated ahead of time, but unlike a plan it's not a sequence of actions the NPC must take, it is an action that an NPC can take in all states (Yousof, 2002). The goal of MDPs is to find an optimal policy, which maximises the expected

utility of each state (Pashenkova and Rish, 1996). The utility is the value or usefulness of each state. Movement between states can be made by moving to the state with the maximum expected utility (MEU).

In order to determine an optimal policy, algorithms for learning to behave in MDP environments have to be used (Kaelbling and Littman, 1996). There are two algorithms that are most commonly used to determine an optimal policy, however other algorithms have been developed, such as the Modified Policy Iteration (MPI) algorithm (Puterman and Shin, 1978) and the Combined Value-Policy Iteration (CVPI) algorithm (Pashenkova and Rish, 1996).

The two most commonly used algorithms for determining an optimal policy have a foundation and take inspiration from Dynamic Programming (Kaelbling and Littman, 1996) which is also a technique for solving sequential decision problems. In addition problems with delayed reinforcement are well modelled as MDPs (Kaelbling and Littman, 1996). There are many algorithms in the area of reinforcement learning (For example: Q learning) that address MDP problems (Mitchell, 1997), in fact understanding Finite MDPs are all you need to understand 90% of modern reinforcement learning (Sutton and Barto, 2000).

The two most commonly used algorithms are value iteration (Bellman, 1957) and policy iteration (Howard, 1960). The value iteration (VI) algorithm is an iterative process, which calculates the utility of each state, which is then used to select an optimal action (Russell and Norvig, 1995). The iteration process stops when the utility values converge. Convergence occurs when utilities in two successive iterations are close enough (Pashenkova and Rish, 1996). The degree of closeness can be defined by a threshold value. This process was however, observed to be inefficient, because the policy often becomes optimal long before the utility estimates reach convergence (Russell and Norvig, 1995). Because of this another way of finding an optimal policy was suggested. It is called policy iteration.

The policy iteration (PI) algorithm generates an initial policy, which usually involves taking the rewards of states as their utilities (Pashenkova and Rish, 1996). It then calculates the utilities of each state, given that policy (Russell and Norvig, 1995). This is called value determination (Pashenkova and Rish, 1996; Russell and Norvig, 1995). It then updates the policy at each state using the new utilities. This is called policy improvement (Pashenkova and Rish, 1996). This process is repeated until the policy stabilises. The process of value determination in policy iteration is achieved by a system of linear equations (Pashenkova and Rish, 1996).

This works well in small state spaces, but in larger state spaces this system is not efficient. However arguments have been made that promote each approach as being better for large problems (Kaelbling and Littman, 1996). This is where other algorithms such as modified policy iteration (MPI) can be used to improve the process. Modified policy iteration was introduced by Puterman and Shin (1978). In modified policy iteration, value determination is similar to value iteration, with the difference being that utilities are

determined for a fixed policy, not for all possible actions in each state (Pashenkova and Rish, 1996). The problem with this process is that the number of iterations of the value determination process is not determined. Pashenkova and Rish (1996) state that Puterman (1994) proposed the following options that could be used to solve this problem. Firstly, simply use a fixed number of iterations, secondly choose the number of iterations according to a predefined pattern and thirdly use the same process as value iteration.

COMPUTER GAMES & APPLICATIONS

There are many different types of commercial computer games available today; these include Real Time Strategy (RTS) games, sims games, God games and First person shooters (FPS) (Tozour, 2002). The AI in these and other type of games could possibly benefit from MDPs.

The most obvious computer game application for MDPs is a grid world navigation example, where the game world is split into a grid, which an NPC uses to navigate from one location to another. This example can be found in most literature on the subject including Russell and Norvig (1995) and Mitchell (1997). The task of moving NPCs in these types of game is in essence a sequential decision problem. This is exactly what the MDPs framework solves. This use of MDPs could be applied to RTS, FPS or 2D platform games. Other applications of MDPs include decision-making and planning. For this work we propose to apply MDPs to NPC movement in a 2D style game, such as Pac-man (Namco, 1980). We have chosen this type of game because it operates in real time and offers plenty of scope to explore the different features of MDPs.

DEVELOPMENT

In this section we present the development of the VI algorithm as an AI engine for use in real time 2D style computer games. The VI algorithm was implemented with a convergence threshold as the stopping criterion. However we also looked into creating our own stopping criterion, which was based around VI and designed for speed and use in real time computer games.

Value iteration using convergence as a stopping criterion is designed to find the optimal policy. However a less than optimal policy is acceptable in computer games if it speeds up processing time and still allows the NPC to reach its goal in an appropriate and acceptable manner. We have developed a new stopping criterion, which is as simple and quick as possible, but which still should achieve a workable policy. We call the new stopping criterion "Game Value Iteration" (GVI) and it works as follows: we simply wait for each state to have been affected by the home state at least once. This is achieved by checking if the number of states, with utilities that are equal to or less than 0 (zero) are the same after 2 successive iterations. All non-goal states have a reward (cost), which is slightly negative depending on their environment property (i.e. land, water etc.). Since utilities initially equal rewards, a state's utility will be negative until it has been affected by the positive influence of the home state.

As a result the number of cells with negative utilities will decrease after each iteration. However some states may always retain a negative utility, because they have larger negative rewards due to their environment property and they may be surrounded by states with similar environment properties. Consequently when the number of states with negative utilities stays the same for 2 successive iterations we can say that the states are not optimal, but they should be good enough for a workable policy to exist, which the NPC can use to navigate the map. Before this point it is likely that no workable policy for the entire environment would exist. This stopping criterion assumes rewards can only be negative and there is a positive terminal state which is equal to 1. Also note that, checking if a state's utility is greater than 0 is not required for the terminal states, because their utilities never change. When each state has been affected by the home state at least once we can say that the states are not optimal, but they should be good enough for a workable policy to exist, which the NPC can use to navigate the map.

An AI engine program was developed in Microsoft Visual Basic in conjunction with the AI engine. This program contained the AI engine itself and an environment to test the engine. The environment consisted of a top down view, just like a 2D style game and was made up of a 10x10 grid of cells, each cell in the grid having different properties associated with it. For example a cell could have a land, wall or water property. Figure 2 shows an example of how the grid based environment would look. Figure 2 is based on an example of this type of environment found in Russell and Norvig (1995).

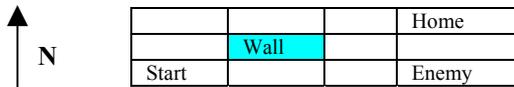


Figure 1: Example of the grid based environment.

The properties of an environment are used by the NPC (i.e. AI engine) to affect the reward value for each cell. For example water could mean slower movement for the NPC, so by giving cells with the water property an additional negative reward value (i.e. -0.02) it will mean that the reward for being in that cell is slightly less than cells with no water property. When the utility value of each cell is created the utility values of cells with the water property will be less than those with no water property. So when an NPC makes a choice of which cell to move to it will be less likely to move to the cell that has the water property.

The NPC will be able to move in one of four directions North, East, South, or West, which will supposedly move the NPC one cell in the intended direction, but only with a certain amount of probability (Pashenkova and Rish, 1996), such as 0.8. However this will depend on the obstacles in the grid such as a wall or the edge of the grid.

The NPC will begin in a start state, which can be any cell in the grid, except the enemy cell or home cell. The terminal states, where the simulation ends, are the home and the enemy states. In 2D style game the home state for the NPCs will be the human player. The home terminal state is the positive terminal state for the NPC and the enemy terminal

state is the negative terminal state, which the NPC will avoid.

IMPLEMENTATION

This section covers how MDPs were implemented as an AI engine. As stated above the utility value of each cell in the grid (game environment) was determined by using the value iteration algorithm. We used two different stopping criteria: utility convergence and our new stopping criterion, called game value iteration, to ensure that each cell in the grid creates a usable policy for the NPC.

When the utility values for each cell are initialised they are initialised to the reward value of each cell. Each non-goal state always has a slightly negative reward on top of any cell property rewards. The cell(s) containing the enemy will have a reward value of -1 and the cell containing the home (or goal) will have a reward value of +1, regardless of the cell's other properties.

A schematic description of the GVI algorithm is given below. The value iteration algorithm is implemented exactly as it is in Russell and Norvig (1995). The GVI algorithm is based on this algorithm.

```

function GAME VALUE-ITERATION(M, R) returns a utility function
  inputs:  M, a transition model (or transition probabilities)
           R, a reward function on states
  local variables:
    U, a utility function, initially identical to R
    U1, a utility function, initially identical to R
    AllStatesChanged, an all states changed flag initially equal to false
    NumStatesBelowZero, stores the number of states below zero
    LastNumStatesBelowZero, stores the last number of states below zero

  repeat
    for each state i do
      U1[i] ← R[i] + maxa ∑ Mija U[j]
    end
    U ← U1
    LastNumStatesBelowZero = NumStatesBelowZero

    for each state i do
      if U(i) ≤ 0 then
        if U(i) ≠ AnyTerminalStates then
          NumStatesBelowZero = NumStatesBelowZero + 1
        end if
      end if
    end
  until AllStatesChanged = true
  return U

```

The step in the schematic description above, where the utility values are determined is the first *for* loop just after the *repeat* statement. The equation in that loop can also be seen below.

$$U_1 [i] \leftarrow R[i] + \max_a \sum_j M_{ij}^a U[j]$$

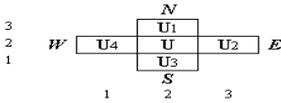
Where $U_1[i]$ is the new utility value estimate for a cell in the grid and $R[i]$ is the reward value. \max_a is select the utility that returns the maximum value. i is the index of all cells in the grid and j is the index of the number of cells surrounding i (i.e. possible moves, north, south, east, west). M is the transition model (the probability of moving in a certain direction) and U is the current utilities.

Given the value iteration equation above, the utilities for each state can be determined, and given the fixed policy of maximising expected utilities, an NPC will be able to make a move in any state. No matter what the outcome of any action

is, the NPC will always know where to move next, by selecting the cell that has the highest expected utility. Next we are going to show an example of how the equation will work in practice. It demonstrates for one iteration how the utility value for one cell in the grid will be determined.

Key

U = Utility.
P = Probability.



PA = 0.8.
PB = 0.1.
PC = 0.1.
PD = 0.0.

To work out the utility of cell 2,2 the following will be conducted:

$$\text{Action N} = PA * U1 + PB * U2 + PC * U4 + PD * U3.$$

$$\text{Action E} = PA * U2 + PB * U3 + PC * U1 + PD * U4.$$

$$\text{Action S} = PA * U3 + PB * U4 + PC * U2 + PD * U1.$$

$$\text{Action W} = PA * U4 + PB * U1 + PC * U3 + PD * U2.$$

U = Reward + The action that returns the maximum value.

This process is repeated for every cell in the grid, except for the enemy's cell(s), the home cell and any wall cells. If the utility is being calculated for the cell next to a wall or a cell on the edge of the grid, there will be no possible move in those directions. If this occurs, then the utility value of the cell whose utility is being calculated will be used. One iteration is complete when every cell has been visited once. The process is repeated until the stopping criterion is met.

EXPERIMENTAL RESULTS

Many different experiments were conducted on the AI engine through the AI engine program. The results of these experiments were used to help implement a computer game and to validate our work. The parameters that were varied in the experiments included the configuration of the maps (i.e. locations of obstacles and goal states) and the reward values associated with cell properties.

However the results discussed here mainly look at determining the appropriate threshold value for VI, determining whether the GVI algorithm works in practice and comparing each algorithm's performance. In our experiments an NPC was setup to learn what action to take in each cell by using the VI algorithm. Tables 1 and 2 show some of the results of this work and screenshots of the test maps used to produce the results in those tables.

For all experiments the following things were kept the same: there were two goal states, +1 (home) and -1 (enemy), and there was a cost of -0.0000001 for all non-goal states. The probability of moving in the intended direction was 0.8 and the size of the game world was 10x10.

The HD column in tables 1 and 2 stands for hamming distance between the generated policy and the optimal

policy. The optimal policy is the policy obtained by running the algorithm with the same initial data and maximum precision (Pashenkova and Rish, 1996). The use of hamming to determine the difference between a policy and an optimal policy is based on that used in Pashenkova and Rish (1996).

| Convergence Threshold | No. of Iterations to Convergence | Agent Successfully Navigated Map | No. of steps Agent took To Navigate Map | HD |
|-----------------------|----------------------------------|----------------------------------|---|----------|
| 1.00 | 1 | No | N/A | 70 |
| 0.5 | 4 | No | N/A | 61 |
| 0.25 | 8 | No | N/A | 39 |
| 0.125 | 12 | No | N/A | 11 |
| 0.0625 | 15 | No | N/A | 2 |
| 0.031250 | 19 | Yes | 18 | 0 |
| 0.015625 | 21 | Yes | 18 | 0 |
| 0.007812 | 23 | Yes | 18 | 0 |
| 0.003906 | 24 | Yes | 18 | 0 |
| 0.001953 | 25 | Yes | 18 | 0 |
| 0.000977 | 26 | Yes | 18 | 0 |
| 0.000488 | 27 | Yes | 18 | 0 |
| 0.000244 | 28 | Yes | 18 | 0 |
| 0.000122 | 29 | Yes | 18 | 0 |
| 0.000061 | 30 | Yes | 18 | 0 |
| 0.000031 | 30 | Yes | 18 | 0 |
| 0.000015 | 31 | Yes | 18 | 0 |
| 0.000008 | 32 | Yes | 18 | 0 |
| 0.000000 | 59 | Yes | 18 | - |
| GVI | 18 | Yes | 18 | 1 |

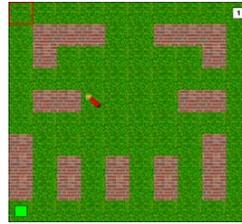


Table 1: The environment map and the results produced from experiments conducted on the map.

| Convergence Threshold | No. of Iterations to Convergence | Agent Successfully Navigated Map | No. of steps Agent took To Navigate Map | HD |
|-----------------------|----------------------------------|----------------------------------|---|----------|
| 1.00 | 1 | No | N/A | 27 |
| 0.5 | 4 | No | N/A | 27 |
| 0.25 | 7 | No | N/A | 27 |
| 0.125 | 10 | No | N/A | 26 |
| 0.0625 | 14 | No | N/A | 22 |
| 0.031250 | 19 | No | N/A | 15 |
| 0.015625 | 20 | No | N/A | 12 |
| 0.007812 | 22 | No | N/A | 14 |
| 0.003906 | 27 | Yes | 32 | 9 |
| 0.001953 | 34 | Yes | 36 | 3 |
| 0.000977 | 40 | Yes | 36 | 0 |
| 0.000488 | 46 | Yes | 36 | 0 |
| 0.000244 | 50 | Yes | 36 | 0 |
| 0.000122 | 53 | Yes | 36 | 0 |
| 0.000061 | 56 | Yes | 36 | 0 |
| 0.000031 | 60 | Yes | 36 | 0 |
| 0.000015 | 63 | Yes | 36 | 0 |
| 0.000008 | 66 | Yes | 36 | 0 |
| 0.000000 | 210 | Yes | 36 | - |
| GVI | 32 | Yes | 36 | 4 |



Table 2: The environment map and the results produced from experiments conducted on the map.

The maps used for the experiments above attempt to represent a maze like world that you would expect to see in 2D style games. However we also experimented with simpler and more complex maps. Tables 1 and 2 show that the largest threshold, which produces an optimal policy, is 0.031250 (Table 1). However this threshold does not produce an optimal policy in Table 2. This shows that the utility thresholds, which produce an optimal policy, vary from map to map. In general we observed that as map complexity increased, they required more iterations and smaller thresholds to achieve workable and optimal policies. This could cause problems in computer games because maps are constantly changing and vary from level to level. As a result it's reasonable to say that a conservative threshold would have to be used to ensure that a map always converged to an optimal or near optimal policy. Tables 1 and 2 also show that the utility values for the VI algorithm converge after the policy has converged. This result is consistent with previous work in the area, such as Pashenkova and Rish (1996) and Russell and Norvig, (1995) and is a recognised issue with this algorithm.

From tables 1 and 2 we can see that the GVI algorithm produces a workable policy that is less than optimal, but

converges at a low number of iterations. This means the algorithm should on average be quicker to run than VI because larger numbers of iterations require more processing time. Also a benefit of this algorithm is that it automatically adapts to the complexity of the game world, so it should always produce the best policy it can, without running any unnecessary iterations. The algorithm should always produce a workable policy, but it will not necessarily be the optimal policy. In our experiments above this seems to matter very little, because the hamming distance is very small, but on a large map (E.g. 20x20 or 40x40) this difference might become significant.

We also conducted experiments on the reward values of cells to see how they affect an agent's movement. These experiments showed that the affect a negative reward would have on an NPC depended on how optimal the policy was. If a zero threshold was used with VI, a small negative value (i.e. -0.02) for the cell property water would be enough to affect the NPCs behaviour so it would be likely to avoid water until it was necessary so go through it. However for less optimal policies this value would need to be slightly bigger to have a similar affect (i.e. -0.06). This affect is just like the one discussed in the paragraph above. Because the policy is not optimal the water (or enemy's) effect on the game environment is lessened. Also it is worth noting that if the negative rewards are increased by too much this can also cause problems, because they can have too great an affect on the cell's utility which can prevent the GVI algorithm from converging to a workable set of utilities.

DISCUSSION

The experiments conducted on the AI engine program have shown that MDPs using both VI and the newly introduced GVI algorithms can be used to create intelligent NPC behaviour. The movement produced by the AI engine appeared to the authors to be less scripted and deterministic than that in researched 2D style computer games. The AI engine also offers interesting environment features through creative use of reward values. This could make the MDPs AI engine interesting to computer game players and the computer game industry, because it offers a different approach to solving the problem of AI in 2D style games.

The MDP AI engine with VI and GVI as an AI tool for NPC navigation offers game developers a different approach to applying AI to 2D style games. However from the results of this work and our observations we can see that there are limitations with this technique that need to be researched further. Firstly, even though the VI algorithm works in our AI engine (which has just 1 NPC), it is very processor intensive. The GVI algorithm does overcome this problem; however this algorithm would need to be tested further to prove its usefulness. Secondly, the experiments conducted here were only on 10x10 grids. This size grid is quite small for a game environment, so experiments would need to be conducted on larger grids to determine if the VI and GVI algorithms can execute quickly enough and the less than optimal policy for GVI is still viable. The hamming distance between the GVI algorithm policies and the optimal policies was quite small in our experiments; however it could be a lot larger in bigger game environments.

CONCLUSIONS AND FUTURE WORK

This paper has shown that Markov decision processes using Value iteration and the newly introduced GVI algorithm can be successfully applied to an AI computer game engine. The development of the AI engine and the experiments conducted on the AI engine allowed a greater understanding of this approach and the problems involved, in relation to computer games.

There is plenty of scope for further work in this area. Firstly we intend to apply the AI engine to a 2D real time computer game to determine if the technique can operate successfully in this domain. Secondly we plan to extend the size of the game environments and confirm that the use of a less than optimal policy still produces a viable solution in larger environments.

REFERENCES

- Bellman R. (1957) *Dynamic Programming* Princeton University Press, Princeton, New Jersey.
- Bonet B. (2002) An ϵ -Optimal Grid-Based Algorithm for Partially Observable Markov Decision Processes. *in Proc. 19th Int. Conf. On Machine Learning*. Sydney, Australia, 2002. Morgan Kaufmann. Pages 51-58.
- Howard R. (1960). *Dynamic Programming and Markov Processes*. Cambridge, MA: The MIT Press.
- Kaelbling L. and Littman, M. (1996) Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-285.
- Kellis E. (2002). An Evaluation of the Scientific Potential of Evolutionary Artificial Life God-Games: Considering an Example Model for Experiments and Justification. MSc. Thesis, University of Sussex.
- Kristensen A. (1996), Textbook notes of herd management: Dynamic programming and Markov decision processes <<http://www.prodstyr.ihh.kvl.dk/pdf/notat49.pdf>> (accessed 24 April 2003).
- Lionhead Studios / Electronic Arts (2001) Black & White. <<http://www.eagames.com/>>.
- Mitchell T. (1997). *Machine Learning*, McGraw Hill: New York.
- Namco (1980), Pac-man. <<http://www.namco.co.uk/>>.
- Pashenkova E. and Rish I. (1996) Value iteration and Policy iteration algorithms for Markov decision problem. <http://citeseer.nj.nec.com/cache/papers/cs/12181/ftp:zSzzSzftp.ics.uc i.edu:zSzpubzSzCSP-repositoryzSzpaperszSzmdp_report.pdf/value-iteration-and-policy.pdf> (accessed 23 April 2003).
- Puterman M. (1994) Markov decision processes: discrete stochastic dynamic programming. New York: John Wiley & Sons.
- Puterman M. and Shin M. (1978) Modified policy iteration algorithms for discounted Markov decision processes. *Management Science*, 24:1127-1137.
- Russell S. and Norvig P. (1995). *Artificial Intelligence A modern Approach*, Prentice-Hall: New York.
- Spronck P., Sprinkhuizen-Kuyper I. and Postma E. (2002). Evolving Improved Opponent Intelligence. *GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation* (eds. Quasim Medhi, Norman Gough and Marc Cavazza), pp. 94-98.
- Spronck P., Sprinkhuizen-Kuyper I. and Postma E. (2003). Online Adaptation of Game Opponent AI in Simulation and in Practice. *GAME-ON 2003 4th International Conference on Intelligent Games and Simulation* (eds. Quasim Medhi, Norman Gough and Stephane Natkin), pp. 93-100.
- Sutton R. and Baro A. (2000) *Reinforcement Learning An Introduction*. London: The MIT Press.
- Tozour P. (2002) The Evolution of Game AI in Steve Rabin (ed) *AI Game Programming Wisdom*, Charles River Media, pp. 3-15.
- Yousof S. (2002) MDP Presentation CS594 Automated Optimal Decision Making, <<http://www.cs.uic.edu/~piotr/cs594/Sohail.ppt>> (accessed 27 April 2003).

MONTE CARLO REALTIME STRATEGY

Tristan Cazenave

Laboratoire d'Intelligence Artificielle
 Université Paris 8, Saint Denis, France
 cazenave@ai.univ-paris8.fr

1 Introduction

We present a simple real time strategy game and five algorithms that play this game. The algorithms are compared with head to head confrontations. The best one performs Monte-Carlo simulations associated to some goals of the game.

Monte carlo simulations have been used to select moves in strategic games such as Go [Bruegmann, 1993; Bouzy and Helmstetter, 2003].

2 A Simple Real Time Strategy Game

We have designed a simple real time strategy game for our experiments. It is played on a 50 by 20 map. The two opponents, Blue and Red, start each with a 5 by 3 base in the upper and lower middle part of the map. They also own 10 units each, aligned below the upper base for Blue, and aligned above the lower base for Red. The bases have a health of 100, and each unit has a health of 30 at the beginning of the game. Each unit occupies a one by one square on the map, and it can move in any of the eight directions provided the goal neighbor square is empty. Units can also shoot enemies that are located in the eight neighboring squares.

3 Move Selection Algorithms

The random move algorithm picks randomly one move out of the nine possible moves (the eight directions and the stay in the same place move). This strategy mostly serves as an etalon for the other strategies. All strategies should perform better than the random move strategy.

The nearest enemy unit strategy consists in moving toward the nearest enemy unit and shooting at it as soon as it is in its neighborhood.

The evaluation function adds the health of the friend base and of all the friend units, and subtracts the health of the enemy base and of all the enemy units.

The Monte-Carlo moves strategy consists in performing a given number of simulations, and to choose moves according to the statistics collected on the final results of the simulations. In each simulation, the first move of each unit is played randomly. The subsequent moves are played with the nearest enemy unit strategy for both sides.

The Monte-Carlo double moves strategy starts each simulation as the Monte-Carlo moves strategy by playing random moves for the units, and also continues the simulation using

Table 1: Sum of results for each algorithm.

| <i>Algorithm</i> | <i>Sum of results</i> |
|-----------------------|-----------------------|
| <i>random</i> | -960 |
| <i>mc_double(100)</i> | -530 |
| <i>mc_moves(100)</i> | -185 |
| <i>nearest</i> | 530 |
| <i>mc_goals(100)</i> | 1145 |

the nearest enemy unit strategy. But instead of recording the scores of each individual move, it records the scores of pairs of moves for pairs of units. A score is maintained for each possible pair of moves of each possible pair of units.

The Monte-Carlo goals strategy does for the goals of the game what the Monte-Carlo moves strategy does for the moves. The only kind of goal we have currently tested is to attack an enemy unit or base. At the beginning of each simulation, a target enemy unit is set for each friend unit. During the simulation, the friend units will hunt for their target enemy unit. Once the target enemy unit is dead, the friend unit reverts to a nearest enemy unit strategy. The enemy units use the nearest enemy unit strategy during all the simulation.

A score is maintained for each possible goal of each possible unit. After a fixed number of simulations, the goal with the highest score is chosen for each unit.

Table 1 is created summing for each move selection algorithm its score against all the other algorithms. They have been tested with 100 simulations before each move, enabling a very fast move decision process.

One promising area for future work is to add higher level tactical goals such as protecting the base, protecting an area or helping another unit. Testing the strategies in a more complex real time strategy game is also appealing.

References

- [Bouzy and Helmstetter, 2003] B. Bouzy and B. Helmstetter. Monte Carlo Go developments. In *Advances in computer games 10*, pages 159–174. Kluwer, 2003.
- [Bruegmann, 1993] B. Bruegmann. Monte Carlo Go. <ftp://ftp-igs.joyjoy.net/go/computer/mcgo.tex.z>, 1993.

Social/Humanities Aspects of Games

- Anderson, D., Arnold, S. E., Jacobi, D., Mehdi, Q. H. and Gough, N. E.**
Turning a corner: Games and the social content 301
- Krzywinski, A., Helgesen, A. S. and Chen, W.**
Caeneus architecture – an agent for social games 306
- Thawonmas, R., Zhai, Y. and Konno, Y.**
Evaluating reputation of online-game players based on incoming chat messages
311
- Thornton, J. S. and Purdy, J. H.**
**Profiling the physiological and psychological effects of playing a networked ‘first
person shoot-em-up’ computer game: Quantitative descriptive pilot study**
317
- Wiklund, M.**
**Games and peer-to-peer file sharing: Attitudes towards illegal distribution of
computer games** 325
- da Fonseca, J. B.**
**Mastermind with an unlimited number of lies as a modeling tool of cognitive
processes involved in human justice** 330

TURNING A CORNER: GAMES AND THE SOCIAL CONTENT

Don Anderson^a, Stephen E. Arnold^a, Dennis Jacobi^a
Quasim Mehdi^b, Norman Gough^b

^aIntellas Group, LLC, 15424 Beckley Hills Drive, Louisville, KY 40245

^bUniversity of Wolverhampton, School of Computing and Information Technology,
35/49 Lichfield S Wolverhampton, WV1 1SB United Kingdom
danderson@intellas.com

KEYWORDS

Games, Social Content, Simulation, Modeling, Artificial Intelligence

ABSTRACT

Electronic games have moved to the mainstream. With this change has come a new set of challenges for engineers, developers, and funding entities. Third-generation game warfare goes beyond the reflex-reaction of the first and second-generation games, particularly with regard to warfare. The task today is to blend a data rich environment, near real time updates, and cognitive change within a context. Funding agencies, particularly for government-sponsored projects, require two different approaches to engineering design. The first is the need for architecting so that one or more elements can be easily repurposed. Repurposing means that the cost of developing a function or feature can be spread across multiple event delivery platforms. The second is the need for cost reduction and even cost recovery. The outcome of these two different engineering boundaries is a change in the way second-generation games and third-generation games are planned, constructed, implemented, and repurposed. The outlook for games is more robust than for some other types of applications but the opportunities come with greater costs. Changes include the need for online support, use of game-like

functions outside of a game environment on analysts' desktops, and an increased discipline with regard to code, team composition, and engineering tactics.

INTRODUCTION

The challenges of modern game development were encountered in a recent project in which Intellas Modeling and Simulation worked with a noted board game designer, Vance von Borries, to create a concept for a third-generation training and battle simulation for the United States Air Force. Until the present, war games and simulations have been primarily attrition based and are centered on the concept of "force on force," and have been designated as "second-generation" war games. So-called "first generation" war games were focused on strategy with the primary concept of "mind on mind." This effort views "third generation" war games and battle simulations as concentrating on effects based operations with the primary focus being "system on system."

The new system will take into account all the factors of the previous generations such as strategy, tactics, and attrition, but will also include logistics, cascading effects, doctrine (both military and social), command and control, and differentiation

between allied, enemy and coalition forces. (Pew et al. 1998) The revised system will be designed to be flexible and scalable with the capability of modeling a variety of scenario types that will include peacekeeping operations, homeland security and police actions in addition to the typical military combat scenarios. Details of the framework for the project can be found in Jacobi et al (2003). A macro view of a possible framework is illustrated in Figure 1.

ENGINEERING ISSUES

The primary engineering issues for a project of this scope and magnitude can be categorized in three main areas: Scalability, modularity, and system intelligence.

Scalability has generally been absent in the development of most game and

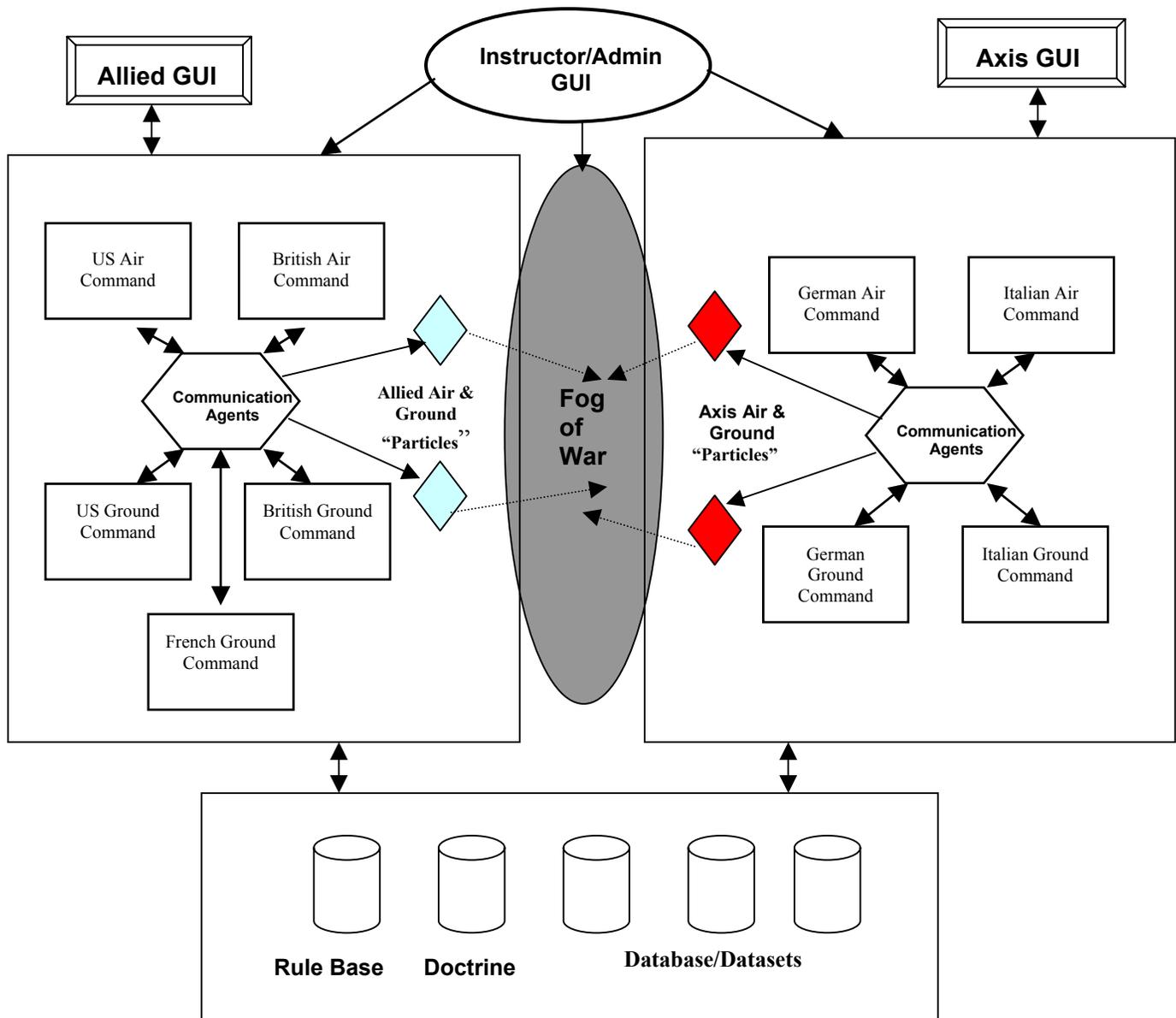


Figure 1: Simulation Macro View

simulation software. There is a vast gap between systems designed to run on standard personal computers (PCs) and those designed to run on large scale and super computers. The middle ground is barely touched at present. The on-line gaming paradigm has gone a long way towards making users aware of the need for a system that can scale to many users, but in government applications in particular, this creates security issues. For training purposes the scalability problem can be addressed by creating a distributed system (Anderson et al. 2002) that can run in a closed loop network or across secure lines between installations. Also a properly designed scalable system should lend itself to the ability to run on a single PC, multiple PCs or other suitable platforms, and also the ability to have higher level features that can be implemented and connected to on larger scale systems, clusters, and/or supercomputers.

Modularity, typically in the use of object-oriented design, has become the standard in the software industry as a whole. Developers have in general moved from the older methods of top down programming to the use of objects and modules of code. However the challenge is to take this concept to the next level through the creation of distributed systems in which the objects or modules can reside on a variety of PCs, platforms and other devices. Also the use of agents (Weiss 2000; Chen and Wragg 2000, Mehdi 2002) as well as newer methodologies such as particle swarm technology are needed to move to the next level of development and deployment of these systems to bridge the gap between games, training tools and higher level analytical simulations.

This new paradigm also requires new methods of system intelligence to be devised. Most games and simulations have used scripted intelligence, which is only as good as the scripts that are referenced. This need is the most difficult challenge in this arena, to develop systems that are capable of learning, and beyond that to be able to mimic the imperfect decision making of human intelligence to add reality to the game or simulation.

PROGRAMMING COSTS

The cost of software development has not benefited from Moore's Law. For example, the cost of developing a game for the Sony PS1 was in the mid-six figures. The cost for developing a game for Sony's PS2 rose to about \$3 million per title. The cost of an original game for the Sony PS3 is likely to hit \$10 million. The Xbox game cost parallels the costs of developing for the PlayStation platform. The likely savings for the Xbox2 will come from Microsoft's use of a modified PowerPC chip, also used by Nintendo, and development tools that use the DirectX technology. The benefit for developers is that costs of developing for the Xbox2 versus the Sony PS3 is that overall development costs are likely to be somewhat lower. Sony has invented a new chip, a new graphics subsystem, and, therefore, new software development tools.

The same situation exists in virtually every sector of the commercial or enterprise software arena. Costs for license fees are flat or drifting downwards. The costs for programming, maintenance, and support are rising faster than any other cost associated with modern systems.

There are notable exceptions, and these warrant mentioning:

1. An individual not charging his time to a project with the requisite technical skills can program a winner. Whether one points to the success of Tetris or the handiwork of Shawn Fanning, it is possible to make millions at very low cost.
2. Open source programming provides a viable alternative to branded network operating systems. The open source revolution is likely to persist; however, for certain enterprise applications the fear of rogue code or security vulnerabilities effectively keeps certain open source software out of some organizations.
3. Recycling “old code” with today’s programming tools can reduce the cost of migrating certain applications from one platform to another. Microsoft’s new approach to Xbox development is that the SDK allows the programmer to compile for specific devices, including wireless platforms. By making repurposing

faster and easier, programming costs are comparatively lower than approaches that don’t use the most modern tools.

4. Modular structure, the use of ANSI standard C, and Extensible Markup Language can shave time from a programming project, thus reducing costs.

Nevertheless, overall game development costs are rising and there is little evidence that development costs will trend down in a significant way in the near to mid term.

The market has driven a change in development methodologies as illustrated in Figure 2. Efficiency and modularity are required to reduce development costs and provide a platform for future efforts. The organization of the project has become as important as the product to be developed. Since time has become increasingly an important factor, parallel development is essential to expedite the testing and release of the product both for the government and private sector markets. Investors are looking to see a quick return on their investments.

| Old Development Methodology | | New Development Methodology |
|-----------------------------|----------------------------------|--|
| 1 | Unstructured code | Structured and modular code |
| 2 | Assembler | C, C++ |
| 3 | 1 to 3 developers | 2 to 4 teams, each with two to four developers |
| 4 | Graphics done ad hoc | Graphics specialists working in a way similar to the design team on a motion picture |
| 5 | No antecedent | Based on antecedents or a motion picture parallel shoot |
| 6 | Serial development | Parallelized development; teams may be dispersed |
| 7 | No documentation | Automatic documentation plus special notations for proprietary elements |
| 8 | No or casual source code control | Configuration management |
| 9 | Ad hoc compiles and tests | Engineering best practices |

Figure 2: Comparison of Old and New Development Methodologies

CONCLUSIONS

For government game development projects, the goal is to create a code base that can meet the needs of the government client and cross over to generate commercial revenue. America's Army has become the model for that type of development. There are, then, some general guidelines that game developers will want to keep in mind.

OUTLOOK

What's ahead for government-funded game development?

1. Increasing pressure for commercializing certain games or components in order to generate cost recovery
2. Games will be engineered in the same way that other high performance government systems are designed and built
3. Reuse of graphics and code will expand beyond the "game application"; for example, recruiting commercials
4. Online is no longer an option. Games must run locally and support Web services.
5. Costs will continue to increase.

REFERENCES

Anderson, D., Belknap, M., Cui, X., Elmaghraby, A., Jacobi, D., Kantardzic, M., Ragade, R. (2002), "A Distributed Agent Architecture for Human Operations in Space," for the International Society for

Computers and their Application 11th International Conference on Intelligent Systems on Emerging Technologies, Boston, 2002.

Chen, J.R., Wolfe, S.R., and Wragg, S.D., "A Distributed Multi-Agent, System for Collaborative Information Management and Sharing," Proceedings of the 9th ACM International Conference on Information and Knowledge Management, 2000, 383-388.

Jacobi, D; Anderson, D.; Borries, V; Elmaghraby, A; Kantardzic, M; Ragade, R; "Building Intelligence in Third Generation Training and Battle Simulations," for the International Society for Optical Engineering AeroSense Conference, April 21-25, 2003

Mehdi, Q., Gough, N., Sulliam, H., "Virtual Agent Using a Combined Cognitive Map and Knowledge Base System," for the International Society for Computers and their Application 11th International Conference on Intelligent Systems on Emerging Technologies, Boston, 2002.

Pew, Richard and Mavor, Anne, editors, "Modeling Human and Organizational Behavior: Application to Military Simulations," National Academy Press, 1998.

Weiss, Gerhard, editor, "Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence," The MIT Press, 2000.

CAENEUS ARCHITECTURE: AN AGENT ARCHITECTURE FOR SOCIAL GAMES

Aleksander Krzywinski, Arne S. Helgesen and Weiqin Chen
Department of Information Science and Media Studies
University of Bergen
Arne.Helgesen, Aleksander.Krzywinski@student.uib.no
Weiqin.Chen@infomedia.uib.no
11 October 2004

KEYWORDS

Agent, Agent Architecture, Social Games, Diplomacy, Explicit Knowledge, Tacit Knowledge, Caeneus Architecture

ABSTRACT

The authors present an architecture of an agent capable of playing Diplomacy, the Caeneus Architecture. The architecture is built up as a three layer, one pass, vertical layered agent. Diplomacy is a strategic board game with great focus on diplomatic negotiations between the players. They argue that in social games the participants relates to two kinds of input; the game system (rules and resources) and the social system (communication among the actors). The lower layers of the Caeneus Architecture deal with the former of these two, while the top layer uses the refined information about the game system to carry out diplomatic negotiations with fellow players. The authors outline their prototype implementation of Caeneus, with particular emphasis on the modules in the three layers.

INTRODUCTION

According to Polanyi (1966), knowledge can be divided into explicit and tacit knowledge. Polanyi (1962) argues that a large part of human knowledge is tacit. This is particularly true of operational skills and know-how acquired through practical experience. Knowledge of this type is action-oriented and has a personal quality that makes it difficult to formalize or communicate. Explicit knowledge is codified knowledge that can be transmitted in a formal, systematic language, and can be found in books, newspapers, maps, and other written resources.

Games like Chess, Go and Checkers have been widely used in early AI-research. According to Luger "most games are played using a well-defined set of rules[...] The board configurations used in playing these games is easily represented on a computer, requiring none of the complex formalisms needed to capture the semantics subtleties of more complex problem domains" (Luger, 2002, p. 18). In other words, these games can easily be represented and formalized. It is possible to argue that tacit knowledge is not needed to play such games; one can simply rely on the explicit knowledge about the concepts of the game. We are not arguing that a human player does not use any tacit

knowledge, we are arguing that this knowledge is not needed.

Not all games are purely based on the ability to calculate the next movement of a single piece. Board games like Monopoly, Settlers, Diplomacy and modern computer games like Half-Life, Sims and Thief require more than just brute force algorithms to be played like a human. The social aspect in such games is more important than in Chess. The ability to communicate through different kinds of deals between the participants is crucial.

Games like the ones described above, are what we would argue to be social games. The players have to interact with the social system in addition to the game system - as illustrated in Figure 1. In non-social games, like chess, there is no social interaction of importance to the outcome of the game. By game system we mean the game elements Klabbers (1999) refer to as rules and resources, while the social system is comprised of the actors and the interaction between them.

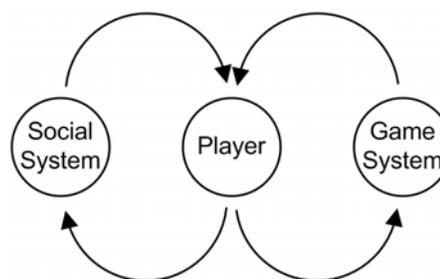


Figure 1: Interaction in social games

Laird stresses that if we seek to build human-level AI system, we must focus on games such as this. "They can provide the environments for research on the right kinds of problem that lead to the type of incremental and integrative research needed to achieve human-level AI" (Laird, 2001). Wooldridge points out that; "classical AI has largely ignored the social aspects of agency. [...] part of what makes us unique as species on Earth is not simply our undoubted ability to learn and solve problem, but our ability to communicate, cooperate, and reach agreements with our peers" (Wooldridge, 2002, p. 10).

The social aspect is assertive in Diplomacy - it is all about negotiation and persuasion. Alliances shift and change throughout the game, often every turn, and it is sometimes impossible to know who is truthful. Each player's social and

interpersonal skills are as important to the game as their strategic abilities. When designing the game mechanics, reference was made to the Napoleonic principle “Unite to fight - separate to live” (Calhamer, 1971). Diplomacy is in all respects a highly social game, and our multiagent system is designed to operate within the boundaries of this game.

In the remainder of this paper we will outline the Caeneus Architecture and describe a prototype implementation. First we want to give the reader a brief introduction to Diplomacy.

A SHORT EXPLANATION OF DIPLOMACY

Diplomacy was created by Allan B. Calhamer in 1954 and released commercially in 1959. It has later been published by Games Research, Avalon Hill and Hasbro.

The game is simulating the First World War. Seven nations fight for domination over Europe. The board is a map of Europe (showing political boundaries as they existed in 1914) divided into 75 regions - 34 of the land regions contain supply centres. For each supply centre they control, a player can build and maintain an army or fleet on the board. Victory is achieved by controlling 18 supply centres.

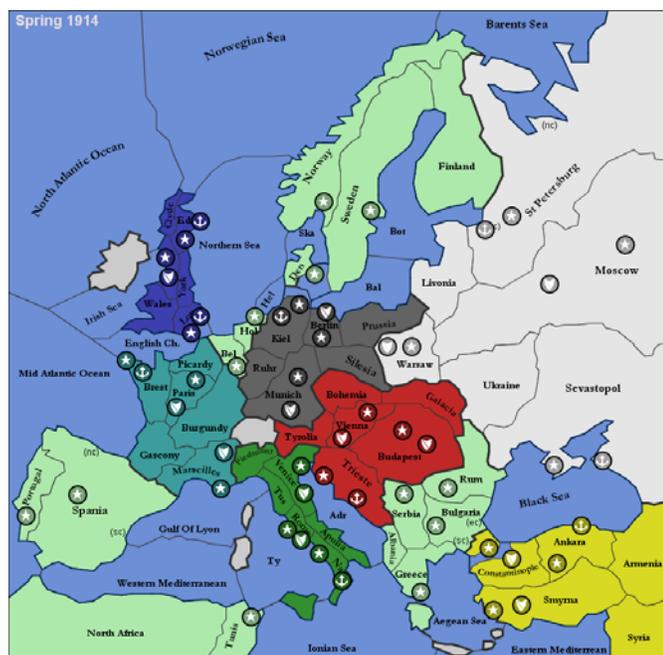


Figure 2: The Diplomacy Board in its initial state

The game mechanics are relatively simple. Only one unit may occupy a region at any time. The greatest concentration of force is always victorious. There is no chance involved. If the forces are equal in strength, standoff results and the units remain in their original positions.

Initially each country is roughly equal in strength, thus it is very difficult to gain territory - except by attacking as part of an alliance. Negotiations for forming alliances are an important part of the game, because numerical superiority is crucial. Secret negotiations and secret agreements are

explicitly encouraged, but no agreements of any kind are enforced.

Each game turn begins with a negotiation period, and after this period players secretly write orders for each unit they control. The orders are then revealed simultaneously, possible conflicts are resolved and the next turn can commence.

AN OUTLINE OF THE CAENEUS ARCHITECTURE

It has become widely accepted that purely reactive control techniques are not capable of handling dynamic, unpredictable, multiagent worlds (Jennings, 1998, p.13). We therefore conclude that a totally reactive approach would not be successful in a social game like Diplomacy. Thus the system must be able to perform some proactive behaviour and to communicate and plan further action with its opponents.

An agent represents a player in the game. Even with potentially seven agents playing, the system is not a multi agent system, at least not according to Sycara (1998, p.2). One key aspect of this environment is that the agents compete in the longer term but must cooperate temporarily. The architecture is similar for every agent, they are clones, but they do not use this fact to accurately predict the actions of opposing agents.

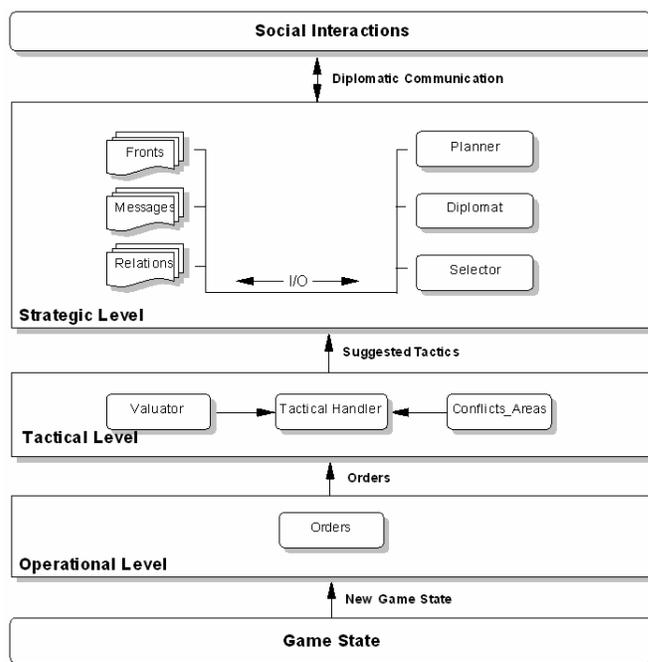


Figure 3: The Caeneus Architecture for Diplomacy

The operational layer focuses on the actions single units can perform, while the tactical layer combines these actions into complete tactics for all of these player’s units. The strategic layer interacts with other players and is responsible for the social part of the game.

The reason for choosing three layers is twofold. Firstly, we have observed that in games like Diplomacy the participant

engage in three separate activities when playing: Observing the game state; considering the next move; and engaging in negotiation with fellow players. This action triplet is directly related to the three layers of our model, operational, tactical and strategic layer, respectively. Secondly the three-layered approach is well known in agent architecture research. For example Jennings et al. 1998, Sycara 1998, Ferguson 1992 all present agent architectures that use this approach. Our initial idea about the Caeneus Architecture falls naturally into this tradition.

When a new game state is presented to the agent, the lowest layer reports the possible orders. This triggers the next layer to compute a list of aggregate orders for many possible scenarios. These lists, also referred to as tactics, form the basis for the evaluations done in the strategic layer. This implies that appearance of new information from the layer below triggers each layer. Thus the flow of control is inherent in the architecture itself in an elegant manner.

A CAENEUS PROTOTYPE

It was apparent that a prototype implementation of the Caeneus Architecture was needed in order to establish if it is usable in its intended context. The prototype development focused on three separate parts: The game system, the agent prototype and a test environment. What follows is an outline of the considerations, algorithms and data structures used in the agent prototype.

The prototype is developed in Java using the Eclipse IDE. We will now describe the prototype layer by layer, beginning at the bottom with the operational layer.

Operational Layer

This level concerns itself with the possible orders an individual unit can be given. In Diplomacy, close cooperation with other players are a key to success, and therefore all units on the board are taken into account.

Orders Module

This module is responsible for creating a collection of all the orders that all the units in a game state can be given. This is done by iterating through all the units and running separate algorithms for each of the order-types, traversing the map graph in various ways:

Hold: Every unit can hold

Move: Iterate through all adjacent provinces and create a move order for each if a path for the unit exists, in addition coastal armies may move by convoy to other coastal provinces if a chain of fleets exist between them.

Support: Create a support order for all holds and all movements into all provinces the unit can move to (without convoys).

Convoy: Create a list of the fleets connected to the fleet in question. Create a convoy order for each army residing in

coastal provinces on the fringes of the fleet group to all other coastal provinces bordering the group.

Tactical Layer

According to the architecture, this level knows nothing of the diplomatic relations of the strategic layer, hence all other countries are considered enemies. It is also important to note that this level is completely recalculated after each update in the game state. It does not keep any records of results of previous tactics, and as a consequence it always calculates all possible tactics. This solution needs more computation time than one only delivering tactics to solve specific game scenarios, but given the turn based context, there is no real need for optimization. More importantly, this level of the architecture is supposed to supply the topmost layer with all possible scenarios by design.

Conflict Areas

The purpose of the conflict areas module is to identify the provinces where a given player can risk facing opposition in the current round. This information is important as it enables the agent to focus on only those areas that may spawn conflict, and ignore areas that won't.

The end product of this module is henceforth a list of ConflictArea objects that in turn consists of one or more Province objects. These ConflictAreas later form the basis for the construction of Front objects.

The four-step algorithm for constructing these Conflict Area objects can be described as follows:

Step 1: For each unit; add all neighbours that are either occupied by an enemy unit, or border such a province, to a list.

Step 2: Split this list up if it consists of several isolated groups of provinces.

Step 3: Split the new list up if there are gaps in the enemy lines.

Step 4: Merge lists where one list is a subset of another list.

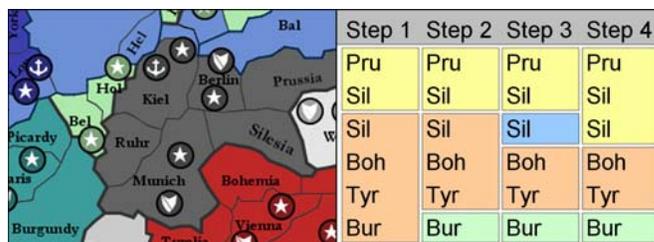


Figure 4: A visualization of the Conflict Area algorithm from the perspective of Germany in the game's opening phase. As the figure shows, Germany has three conflict areas: Pru and Sil; Boh and Tyr; and Bur.

Valuator

The task of this module is to give a value to each province on the map, either on a general basis (with little influence from

the game state) or from the perspective of a given country. It considers these four conditions:

- The number of adjacent provinces
- The number of adjacent supply centres
- Existence of a supply centre
- The level of hostile defence present

Front

The end product of this module is a list of Front objects detailing the areas where the agent might face opposition in the current round.

Algorithm for creating a Front object

Step 1: Use a ConflictArea object to tell what provinces the Front includes

Step 2: Identify units involved in the ConflictArea as primary (directly involved) or secondary (indirectly involved).

Step 3: Based on three different algorithms, offensive, defensive and withdrawal orders are created for all the involved units, and each set of orders is stored as an Operation object

Step 4: All the Operations are evaluated and given a maximum and minimum utility value based on adjudication of these orders versus a wide range of probable hostile orders

After the completion of the above algorithm the current game state has been refined into a set of Fronts for the agent's country. Each Front includes a sorted list of Operations which together constitutes the entire scope of orders the strategic layer needs to acknowledge.

Strategic Layer

The Caeneus Architecture is based on the principle that the two lowest layers refine the new game state to complex data structures that forms the basis for the decisions of the strategic layer. They are not invoked again until a new game state is received from the game engine. This implies that the remaining time is solely utilized by the strategic layer for social communication and internal computation.

This layer can contain many more modules than we have suggested in the design, and in our prototype we have only implemented three modules; selector, diplomat and planner. We believe that the implementation of these three modules will enable us to see trends during evaluation that say something about how well suited the Caeneus Architecture is for social games.

In addition to these three modules, three data structures are also included in this layer; Front list, relational variables and communication history. We consider these as part of the I/O for the three implemented modules.

Relational Variables

If the Caeneus Agent is to manage the social aspect of playing Diplomacy it is important to have sustaining social

relations to the other players. For instance if someone have deceived the agent before it must remember and consider this treachery at later communication with that player. Otherwise the agent would be totally inept at social interaction as it could be tricked over and over again. We have four different relationship variables; trust, debt, power balance and friendship - ranging from -100 (bad) to 100 (good).

It is also important to have a representation of the characteristics of each opponent. This is an aggregation of what the agent sees the other players doing. If one player appears overly aggressive and attacks at any opportunity, it is important for the agent to remember this and take it into consideration. We use four different characteristic variables; aggressiveness, independence, loyalty and closeness to winning – also ranging from -100 to 100.

Modules

The modules are structured in a way inspired by Brooks' Subsumption Architecture (Brooks, 1985). The reason for this is primarily that new diplomatic messages may arrive at any given moment, and they may entirely change the premises for the outcome of this round. Secondly the addition of more complex modules can be done without changing existing ones significantly. The following aspects of Subsumption are retained:

- The lower modules operate without the knowledge of the existence of higher layers.
- The higher modules subsumes the input or output from lower modules

Selector

As the Selector module is the lowest layer in our subsumption inspired structure, this module had to be implemented, but in our prototype its implementation is quite mundane. When it is time to post orders to the server, this module simply selects the Operations that has the highest total value and posts the orders they consist of.

Diplomat

In a game like Diplomacy it is evident that the Diplomat module does crucial work. We have stressed that the game is all about negotiation and making favourable deals. The module is responsible for communicating with both human and agent players.

To enable communication, human and computer players alike must use a set of standard messages with well defined meanings. The main categories of messages are:

- Intelligence (rumours, relations, warnings)
- Suggestions (specific orders)
- Cooperation (short-term)
- Alliances (long-term)

The impact of a received message depends on the senders relation variables. For instance rumours from a sender with a

trust value below a certain threshold, will not have any impact on the Diplomat.

The Diplomat surveys the Fronts and selects the players, with whom cooperation should be sought. This selection is also dependant upon the relationship variables.

Planner

This module has a quite pretentious name compared to what the implementation actually does. Since the player with 18 supply centres wins the game, this module simply increase the value of the 18 centres that at any given time are easiest for the agent to conquer. In this way there is an increased probability that tactics concerning those centres are chosen, and thereby decreasing the effort of winning.

This is what our algorithm considers as easy centres:

- Neutral centres
- undefended centres
- Centres owned by weak players
- Peripheral centres

In addition to this, centres close to the player's power base have their value increased even further, while centres too far away have their values decreased. More complex behaviours could be imagined implemented into this module, for instance to make the agent capable of selecting appropriate short term goals.

PROBLEMS WITH THE PROTOTYPE

During development, test cases for the different algorithms were predominately taken from scenarios happening early in a game of Diplomacy. At most the algorithms were tested on four consecutive game states, but often we found one to be enough. This leads us to believe that the algorithms we have implemented may not be optimal for every game situation the agent can face. This is especially the case with the modules on the tactical level.

We do not, however, feel this is a serious shortcoming of the prototype as we are confident that results of later evaluation still will give us significant information regarding the prototype's performance in the social aspects of the game.

Only rigorous testing of the prototype throughout many complete Diplomacy games can tell us if the algorithms are appropriate for all the phases of a Diplomacy session, but the development of tactically good game AI is not the main focus of this project. It would, however, be a bonus if the tactical decisions made by the agent were

CONCLUDING REMARKS AND FUTURE WORK

In this paper we have introduced the social board game Diplomacy, outlined an architecture for a computer agent playing this game and we have described a prototype implementation of this architecture.

During the implementation of the prototype, which is just recently completed, we discovered that the Caeneus

Architecture was suitable for the socially challenging Diplomacy environment. Our challenge now is to gather and analyze data to say for certain if this feeling has any scientific ground.

To conclude the project we are now evaluating the performance of the prototype. The result of this evaluation is twofold. Firstly it says something about how well the architecture suits the Diplomacy game. Secondly it may also say something about the fruitfulness of the Caeneus Architecture regarding social games in general. However we do not expect the results to give any definite indication with respect to the latter.

REFERENCES

Brooks, R. A. (1985) "A Robust Layered Control System for a Mobile Robot", IEEE J. Rob. Autom. 2. 14-23; also MIT AI Memo 864, September 1985

Calhamer, A (1971) "The Invention of Diplomacy", <http://www.diplomacy-archive.com/resources/calhamer/invention.htm>, Reprinted from Games & Puzzles No.21 (January 1974)

Ferguson, I. A. (1992) "Touring Machines - Autonomous Agents with Attitudes", IEEE Computer 25 (5), May 1992

Greenwals, A., (2003) "The 2002 Trading Agent Competition: an overview of agent strategies", AI Magazine, Spring, 2003.

Jennings et al. (1998) "A Roadmap of Agent Research and Development", International Journal on Autonomous Agents and Multiagent Systems, 1 (1), p. 7-38

Klabbers, J.H.G. (1999) "Three easy pieces: a taxonomy on gaming", Simulations and Games for Strategy and policy Planning. D. Saunders & J. Severn (Eds.). London: Kogan Page, 1999

Laird, J. E. (2001), "Human-Level AI's Killer Application Interactive Computer Games", AI Magazine, Summer, 2001

Luger, G. F. (2002) "Artificial Intelligence – Structures and Strategies for Complex Problem Solving", forth edition, first published 1989. ISBN 0-201-64866-0.

Polanyi, M. (1966), "The Tacit Dimension", London Routledge & Kegan Paul. Publisher: Peter Smith Publisher Inc; (June 1, 1983) ISBN: 0844659991

Polanyi, Michael (1962). "Personal knowledge: towards a post-critical philosophy". New York: Harper Torchbooks.

Sycara, C.P. (1998) "Multiagent Systems" AI Magazine, Summer 1998

Wooldridge, M. (2002), "An introduction to MultiAgent Systems", Published in February 2002 by John Wiley & Sons (Chichester, England). ISBN 0 47149691X.

Wooldridge, M. (1999), "Intelligent Agents", in G. Weiss (Ed) Multiagent Systems, MIT Press, April 1999.

EVALUATING REPUTATION OF ONLINE-GAME PLAYERS BASED ON INCOMING CHAT MESSAGES

Ruck Thawonmas, Yao Zhai, and Yuki Konno
Intelligent Computer Entertainment Laboratory
Department of Human and Computer Intelligence, Ritsumeikan University
Kusatsu, Shiga 525-8577, Japan
URL: www.ice.ci.ritsumei.ac.jp
E-mail: ruck@ci.ritsumei.ac.jp

KEYWORDS

Reputation Systems, Chat Messages, Massively Multi-player Online Games, Online Communities

ABSTRACT

In this paper, we propose an online-game player reputation system that is based on incoming chat messages to each player. The key concepts for implementation of the proposed system are that good players are those who received many chat messages and that good players are those who received chat messages from other good players. The proposed system is tested using computer simulations, in which issues on balance, reliability, and characteristics of different implementation recipes are examined.

INTRODUCTION

The online game industry is one of the fastest growing industries. Among various types of online games, MMOGs (Massively Multiplayer Online Games) have the most important role. According to The Themis Group (Alexander et al. 2004), estimated worldwide revenues of MMOGs will rise from 1.30 Billion USD in 2004 to 4.10 Billion USD in 2008, and to 9 Billion USD in 2014.

In MMOGs, social connections among players are naturally formed. Being in a good community usually makes a player addicted to the game. For game companies, this means *more revenues*. To maintain a good community, many types of social systems have been developed. Typical existing social systems are as follows (Pizer 2003):

Chat systems for conveying messages among players,

Guild systems for letting players form their own communities,

Reputation systems for helping players determine whom they can trust.

Among these social systems, reputation systems are arguably most sophisticated, and are gaining a lot of interests among developers and researchers on online communities (see for example <http://depts.washington.edu/ccce/digitalMedia/rep.html>). For MMOGs, existing reputation systems (Brockington 2003), such as those in *Ultima Online*, *EverQuest*, *Neverwinter Nights*, can automatically detect nasty players by checking, for example, whether or not a player kills other player characters. The reputation systems then reduce the reputation values of those nasty players accordingly. Detection of good players is technically more challenging. For this task, some MMOGs, such as *Lineage II* (<http://www.lineage2.com/>), have a reputation system based directly on user feedback. User feedback, unfortunately, is not always reliable.

In this paper, we propose a new reputation system that evaluates the reputation of each player based on incoming chat messages. Our system is partly inspired by PageRank (Brin and Page 1998) used in Google (<http://www.google.com>) as well as Activity Rating proposed by Kobayashi in his M.Eng. Thesis (Kobayashi 2002). In PageRank, the importance of a page relies on whether or not that page is cited by many other pages and/or by important pages. The basic concepts of the proposed system, their implementations, and experimental results are given in the following sections.

PLAYER REPUTATION

As in Activity Rating, four basic concepts that we employ for evaluating the reputation of each player in the game are as follows:

Concept I Players who receive many chat messages are good.

Concept II Players who receive chat messages from good players are good.

Concept III Players who receive chat messages from many different players are good.

Concept IV Players who have recently received chat messages are good.

Concepts I and II are similar with the key concepts of PageRank. Following the recipe of PageRank, the reputation of the sender of a chat message is evenly distributed to that of the receivers. However, PageRank iteratively computes the importance of a given page by summing the evenly distributed importance of the citing pages until the resulting value converges. It thus requires high computational costs. In our reputation system, upon receiving a chat message, the evenly distributed reputation from the sender is added to the current reputation of the receiver of the chat message.

Concept III indicates that a player who received, say, one hundred chat messages, each from a different sender, has higher reputation than a player who received one hundred chat messages from the same sender. For implementing this concept, we introduce a sender list of player i that keeps track of the senders of the chat messages received by player i in FIFO (first-in, first-out) discipline. Our use of the sender list is straightforward. Namely, a chat message from a sender who is not in the sender list of player i has the highest weight in calculation of the reputation of player i . For a chat message from a sender who is in the sender list, the nearer to the front of the list the sender is, the higher weight has the chat message in calculation of the reputation of player i .

Concept IV is for better reflecting the current reputation of a player. Here, it is simply implemented by regularly decreasing the reputation of each player. This kind of implementation increases the gap between the reputation of the players who have recently received chat messages and that of the players who have not.

Now we are ready to give an implementation recipe of the proposed reputation system. Let $rep(x)$ denote the reputation of player x . We assume player S sends a message to M other players, R_1, R_2, \dots, R_M . Upon receiving the message, the reputation of R_i , where $i \in \{1, 2, \dots, M\}$, is given as follows:

$$rep(R_i) = \min(rep(R_i) + \alpha \frac{rep(S)}{M} + \beta(R_i, S), rep_{MAX}). \quad (1)$$

In (1), the maximum reputation of each player is limited to the parameter rep_{MAX} . The term $\alpha \frac{rep(S)}{M}$ implements Concepts I and II and uses α , a small positive constant, for controlling the evenly distributed (among

M receivers of the message) reputation from player S . The term $\beta(R_i, S)$ implements Concept III and its value is evaluated before S is added into the sender list of R_i ; this term is defined as follows:

$$\beta(R_i, S) = \begin{cases} \gamma_0, & \text{if } send(R_i, S) = 0 \\ \frac{\gamma_1}{2^{send(R_i, S)-1}}, & \text{otherwise,} \end{cases} \quad (2)$$

where γ_0 and γ_1 are small positive constants, $\gamma_0 \geq \gamma_1$, and $send(x, y)$ is the function that returns the position of player y from the front of the sender list of player x . For the sender list of size L , if player y is present in the list, $send(x, y) \in \{1, 2, \dots, L\}$, where the value of 1 indicates the front and L the back of the list; otherwise, $send(x, y) = 0$.

A typical implementation of Concept IV is given as follows: for every N chat messages sent by player i ,

$$rep(i) = \max((1 - \tau)rep(i), rep_{MIN}), \quad (3)$$

where τ is a small positive constant, and rep_{MIN} is the parameter defining the minimum reputation of each player.

EXPERIMENTS

To examine the characteristics of the proposed reputation system, we conducted computer simulations in which, unless stated otherwise, $\alpha, \gamma_0, \gamma_1, L, \tau, rep_{MAX}, rep_{MIN}$, and N were set to 0.0001, 0.002, 0.0016, 5, 0.035, 1.0, 0.1, and 20, respectively. For each player, their initial reputation was set to 0.1.

Balance of Reputation

Here, we report our results where there are two types of players in the game, i.e., socializers and standard players. Following the definition in (Bartle 2004), socializers are players whose main emphasis is to interact with other players. In addition, socializers tend to give useful information to other players. In this respect, they are good players.

In our simulations, 50 players out of 500 players are socializers who send/and receive chat messages twice more than the other 450 players, simply called standard players. In particular, the following experimental procedure was performed.

Step 1 Randomly divide 500 players into 100 groups, each having at most 10 players.

Step 2 Have all players in each group randomly send chat messages to other players in the same group.

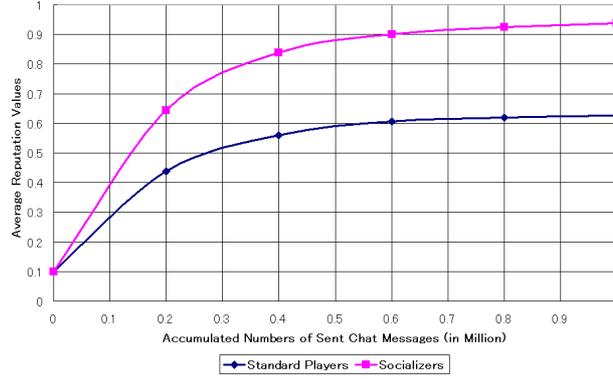


Figure 2: Average Reputation Values over Accumulated Numbers of Sent Chat Messages for the Standard Players and the Socializers

Step 3 Repeat Step 2 until the average number of messages sent by each player reaches 20.

Step 4 Have all 50 socializers randomly send chat messages to other socializers.

Step 5 Repeat Step 4 until the average number of chat messages sent by each socializer reaches 20.

Step 6 Repeat the procedure from Step 1 until the total number of sent chat messages reaches one million.

Figure 1 shows the histograms of the player reputation at different accumulated numbers of sent chat messages. At 0.2 million messages, almost all reputation values are clustered around the middle. These values are shifted rightward as the number of sent messages increases. At 1.0 million messages, the reputation values form two clusters, the bigger one around the range [0.6, 0.7] and the smaller one around [0.9, 1.0]. From Fig. 2 discussed below, the former represents the cluster of the standard players, and the latter that of the socializers.

Figure 2 shows the average reputation values of the standard players and those of the socializers over different accumulated numbers of sent chat messages. As can be seen, both series sharply rise in the beginning and then saturate, to 0.63 for the standard players and 0.94 for the socializers.

From the above results, reputation is well balanced for each player type. In addition, the socializers eventually have higher reputation than the standard players, as we expect they should.

System Reliability

Here we want to know whether or not our reputation system can cope with cheating by players. For example,

a situation may arise where some players intentionally continue sending chat messages to particular friends just for the only purpose of increasing their friends' reputation. We tested the reliability of the proposed system by simulating the following two scenarios.

Scenario 1 Player *A* continues receiving chat messages from a friend whose reputation value is 0.1.

Scenario 2 Player *B* continues receiving chat messages from a friend whose reputation value is 0.9.

Figure 3 shows the reputation values of player *A* and those of player *B* over different accumulated numbers of received chat messages. From this figure, though the reputation values of both players increase as the accumulated number of received chat messages increases, a relatively high number of chat messages are needed to double the reputation value of players *A* and *B*, i.e., more than 1,000 chat messages and 600 chat messages, respectively.

Comparisons among Different Recipes

Here, we compare three different recipes for implementing Concepts I-IV. Each recipe consists of two parts, one for Concepts I-III and the other for Concept IV. By assuming that player *S* sends a message to *M* other players, R_1, R_2, \dots, R_M , each recipe is given in the following.

Recipe 1 This is the one we have discussed so far, i.e., (1) for Concepts I-III and (3) for Concept IV.

Recipe 2 For Concepts I-III, the reputation of R_i is given as follows:

$$rep(R_i) = \min(rep(R_i) + \beta(R_i, S) \frac{rep(S)}{M}, rep_{MAX}), \quad (4)$$

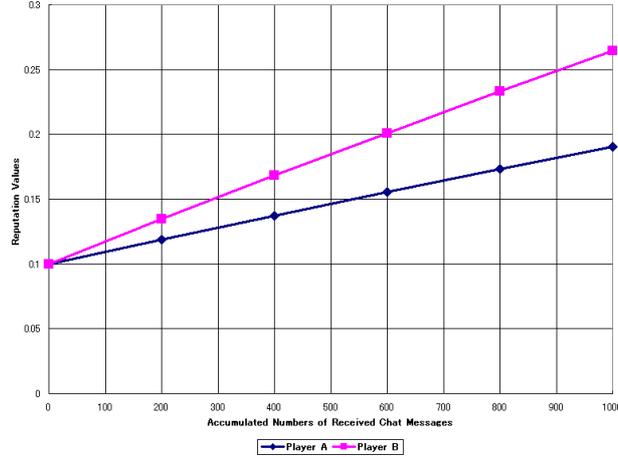


Figure 3: Average Reputation Values over Accumulated Numbers of Received Chat Messages for Player A and Player B

where γ_0 and γ_1 were set to 0.004 and 0.003 in the experiments, respectively. For Concept IV, (3) is used.

Recipe 3 For Concepts I-III, the reputation of R_i is given as follows:

$$rep(R_i) = \min(rep(R_i) + \beta(R_i, S), rep_{MAX}), \quad (5)$$

where γ_0 and γ_1 were set to 0.004 and 0.003 in the experiments, respectively. Concept IV is implemented as follows: for every N chat messages sent by player i ,

$$rep(i) = \max(rep(i) - \delta, rep_{MIN}), \quad (6)$$

where δ is a small positive constant; in the experiments, it was set to 0.035.

Figure 4 shows for each recipe the average reputation values of the standard players and those of the socializers over different accumulated numbers of sent chat messages. As it is clear from this figure, each recipe has its own characteristic curve. In practice, they should be properly selected by game designers. For example, Recipe 1 seems to be a good candidate if the game designer wants the player reputation to rise more easily in the beginning and then gradually make it more difficult.

DISSCUSSION ON PRACTICAL ISSUES

In most MMOGs the concept of a reputation system is informed in advance to all players. So, for successful use in practice, the proposed reputation system definitely should be operated together with other related

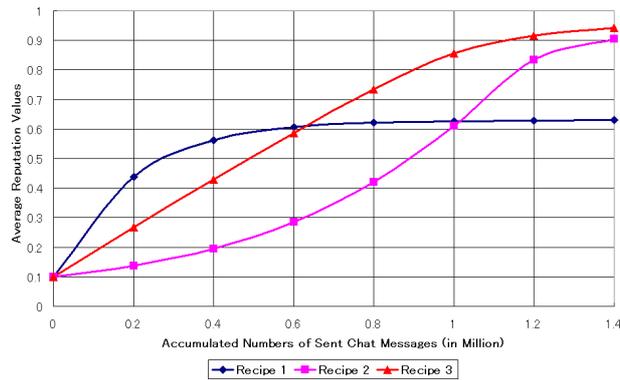
systems such as a system that detects cheating by a guild of dedicated players or bots. Such cheating can be easily noticed by, for example, an unnatural rise in the reputation value. Players involved in detected cheating (of course this has to be confirmed manually by Game Masters) might be penalized by increasing the length of their sender lists or decreasing the values of α , γ_0 , and γ_1 . This would make such players more difficult to raise their reputation values.

We also expect that once a player is aware of the proposed reputation system, they would refrain from sending negative comments to bad players, such as killers as defined in (Bartle 2004). They don't want to unnecessarily raise the reputation of those bad players. Instead, players would send such complaints to Game Masters.

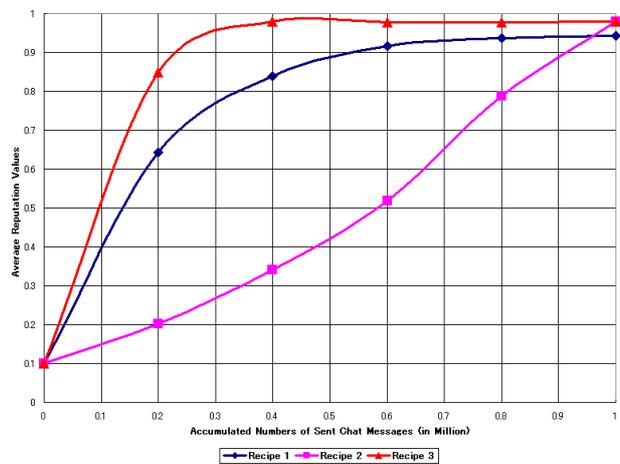
CONCLUSIONS AND FUTURE WORK

In this paper we have presented how a reputation system for online-game players can be constructed based on incoming chat messages to each player. The system has a good balance for players of the same type and well reflects players' chatting characteristics. It can also gracefully tolerate cheating by players. We have discussed three different recipes for implementation of the proposed system, each having its own characteristics that should fit different online-game design concepts.

As our future work, we plan to study how to take into account also the content of chat messages in calculation of the player reputation. We also plan to test the reputation system in an edutainment multiplayer online game called The ICE (Thawonmas and Yagome 2004), under development at our research laboratory.



(a) Standard Players



(b) Socializers

Figure 4: Average Reputation Values over Accumulated Numbers of Sent Chat Messages for Recipes 1, 2, and 3

ACKNOWLEDGEMENTS

This work has been supported in part by the Ritsumeikan University's **Kyoto Art and Entertainment Innovation Research**, a project of the 21st Century Center of Excellence Program funded by the Japanese Ministry of Education, Culture, Science and Technology; and by Grant-in-Aid for Scientific Research (C), Number 16500091, the Japan Society for Promotion of Science.

REFERENCES

Alexander, K.; Bartle, R.; Castronova, E.; Costikyan, G.; Hayter, J.; Kurz, T.; Manachi, D.; Smith, J. 2004. "The Themis Report on Online Gaming 2004." www.themis-group.com/reports.phtml.

Bartle, R. 2004. *Designing Virtual Worlds* News Riders Publishing,

Boston·Indianapolis·London·Munich·New York·San Francisco.

Brin, S. and L. Page. 1998. "The anatomy of a large-scale hypertextual Web search engine." *Computer Networks and ISDN Systems*, vol. 30, no. 1–7, pp. 107–117.

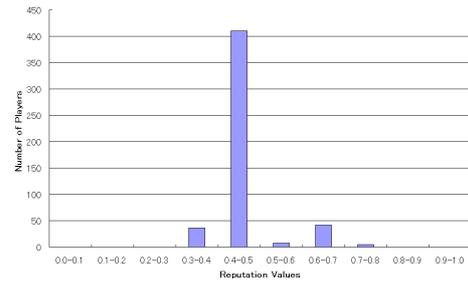
Brockington, M. 2003. "Building a Reputation System: Hatred, Forgiveness, and Surrender in Neverwinter Nights." In *Massively Multiplayer Game Development* Alexander, T., ed., Charles River Media, Inc., Massachusetts, pp. 454–463.

Kobayashi, T. 2002. "Users' Activity Rating Algorithm for Online Communication" M.Eng. Thesis, Department of Communications and Computer Engineering. <http://forest.kuee.kyoto-u.ac.jp/research/master/2001/pdf/toshi-mt.pdf> (in Japanese)

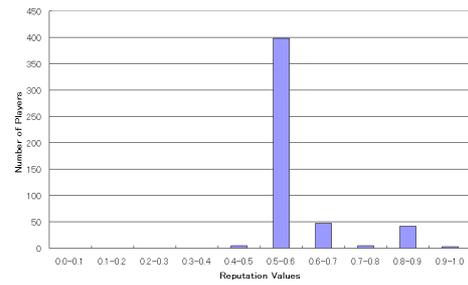
Pizer, P. 2003. "Social Game Systems: Cultivating Player Socialization and Providing Alternate Routes to Game Rewards." In *Massively Multiplayer Game Development*

Alexander, T., ed., Charles River Media, Inc., Massachusetts, pp. 427–441.

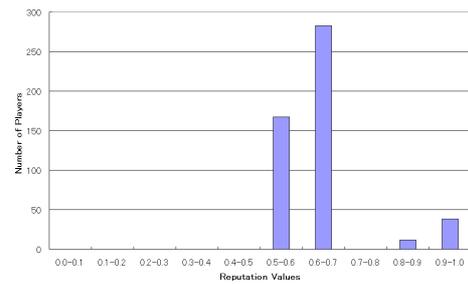
Thawonmas, R. and T. Yagome. 2004. "Application of the Artificial Society Approach to Multiplayer Online Games: A Case Study on Effects of a Robot Rental Mechanism." In *Proc. of the 3rd International Conference on Application and Development of Computer Games (ADCOG 2004)* (HKSAR, Apr.), pp. 12–17.



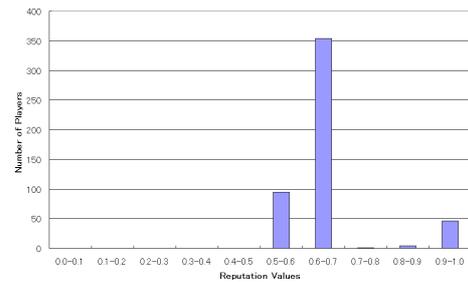
(a) 0.2 million messages



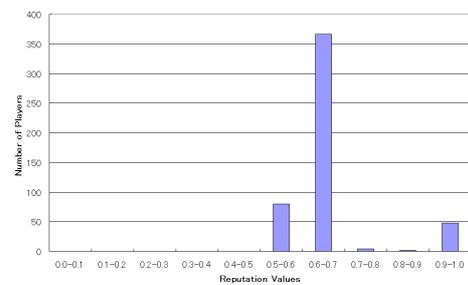
(b) 0.4 million messages



(c) 0.6 million messages



(d) 0.8 million messages



(e) 1.0 million messages

Figure 1: Histograms of the Player Reputation

PROFILING THE PHYSIOLOGICAL AND PSYCHOLOGICAL EFFECTS OF PLAYING A NETWORKED 'FIRST PERSON SHOOT-EM-UP' COMPUTER GAME: QUANTITATIVE DESCRIPTIVE PILOT STUDY

Dr. Jeremy S. Thornton and Dr. Jon H. Purdy
Video Games Research Team
Department of Computer Science
The University of Hull
HU6 7RX, Hull, United Kingdom
E-mail: j.s.thornton@dcs.hull.ac.uk

KEYWORDS

Research, computer games, physiological, psychological, visualization, statistical surfaces.

ABSTRACT

Objectives: To describe the phenomena as they exist by observing the physiological and psychological impact on subjects of playing a networked competition 3 dimensional first person combat computer video game. To develop, trial and refine research methods and measurement techniques. In particular, a client server model for delivering online tests, questionnaires and collecting results. To take raw data and summarize it in a useable form by exploiting 3-dimensional (3D) statistical surfaces.

Design & Setting: A 2 day, two session, participant observation study in the post-graduate computer laboratory at The University of Hull. Volunteers were invited to play a networked competition of the successful first person 3 dimensional combat game "Unreal Tournament". Prior to playing, subjects completed a personality questionnaire, a test of cognitive function and a questionnaire to measure moment affect. Immediately post game play subjects completed a questionnaire to measure moment affect and a test of cognitive function.

Conclusion: Playing a 3-dimensional 1st person combat game as part of networked competition has a measurable effect on cognitive function and mood, and that changes in mood may correlate with personality. A robust client-server model for delivering psychometric tests has been developed and is ready for larger scale trials. Immersive visualization of data is a very useful aid to interpretation.

INTRODUCTION

The purpose of this article is to describe the results of a novel approach to measuring and

describing the physiological and psychological effects of playing computer video games. In its annual report "Entertainment and Media Outlook" (June 2003) PricewaterhouseCoopers forecast double digit compound annual growth rate for videogames between 2003 - 2007. It goes on to predict that spending on the entertainment and media industry around the globe will surpass \$1.1 trillion in 2003, rising by 3.7 percent from its 2002 level. PricewaterhouseCoopers also singled out the Computer and Console Video Game (C&CVG) industry as the fastest growing entertainment/media segment. Along with complexity, the sheer variety of games has also multiplied. Everything from war to love, from history to career planning, has been turned into some form of game. And you can play them by yourself or with thousands of like-minded gamers over the Internet. Clearly videogames are becoming a mega-business, with industry forecasts predicting that the interactive entertainment will hit \$21.4 billion annually by 2005.

But the C&CVG industry is in trouble. The transition to 3rd generation game consoles and increasing home personal computing power will lead to a vast increase in development costs, anticipated at \$20 million. The computer games market is increasingly hits driven with the top 100 games generating more than 65% of the sales in 2002. Many publishers have adopted a portfolio approach, funding the initial development of 20 games with only one or two expected to be completed and make a profit. Such an increasingly difficult environment makes it hard for developers to survive. "Testing should empower designers." - Shamus Blackley (Game Developer's Conference Europe 2003). However, at an industry level the perception is that "current testing systems are broken". Either they are just no longer up to the task or offer poor feedback into core design. Consequently, the gap is widening between the developers and game designers, with job of creating and designing games, on one hand and the

publishers who take the burden of financial risk. On the other, developers and designers want to produce new and exciting games that stimulate emotions and responses in people that encompass the joy of play and publishers have to assess the game in terms of its ability to recoup investment and generate profit. Current testing serves neither purpose. It appears late in the development process and serves only to highlight bugs or offer feedback on potential playability problems or user interface issues. It neither informs core design issues nor measures the likely success.

There is a large body of psychological and psycho-social research into the effects of playing video games. Most of the research focuses on the effects of C&CVG playing on violence and aggression, particularly in children, and its potential effects on society. Much less work can be found on the physiological impact of C&CVG playing other than on simple changes in heart rate and blood pressure. A notable exception being some early work by Dr. Paul Lynch (University of Oklahoma) in 1999 measuring epinephrine, norepinephrine, cortisol and testosterone in subjects. Although Bill Fulton of Microsoft is pioneering an intensive approach to game testing, what seems to be missing completely is any attempt to characterize games, particularly successful, genre defining ones in terms of their effects on the cognitive function, mood, cardio-vascular system and hormone profile of players.

METHOD

Study Design and Subjects

This study was a pilot, quantitative, description of 20 male subjects' moment affect and cognitive function before and after playing a networked 'last man standing' 3-dimensional first-person combat computer game, including a during-game observation of heart rate in a subset of 4 subjects.

Subjects were recruited via e-mail flyer and web based registration site. Although not exclusively recruited, only male subjects were chosen for the study in order to increase the homogeneity of the study population, bearing in mind the small sample size, to increase the likelihood of finding significant results. Demographic and biometric data were self reported by the study subjects using web based forms at the start of each session. No ethnicity data was requested in the pilot study.

Two groups of subjects were studied in separate sessions. The subjects were tested at the same time of day on both occasions. All participants were exposed to the study game as part of a networked competition between midday and 3 pm. On arrival at the game testing laboratory subjects were given the opportunity to discuss the protocol and ask any questions. At the start of each session the subjects completed the online questionnaires for demographics and game play, 16PF Personality Profile and Annett's Handedness score. Short-term changes in moment affect were measured with the Positive and Negative Affect Schedule - Extended (PANAS-X copyright, 1994, D. Watson & L. A. Clark, reprinted with permission). To gauge any changes in cognitive performance, subjects were asked to complete a computerized version of the Stroop Test. These same tasks were then repeated immediately after the game playing session.

In addition a small subset of the study population underwent heart rate monitoring & recording during the game playing session using a chest strap transmitter and remote watch receiver.

Game Choice

The most potent game pattern for a successful computer or game-console game seems to be almost Pavlovian in its simplicity, stimulus and response is the essence. The Stimulus-Response type of game is very popular and generally found to be of a combative model and almost always contains the following features: highly competitive scenario and high level of stimulus response activity; high level of perceived violence and high level of excitement. Within the games industry and its press the term coined for this type of game and its subsequent genre is a "first person shoot-em-up". At the time of the study one of the most popular examples was a game by Epic called 'Unreal Tournament'.

One of the particular concerns when selecting a game for the study was comparing results from subjects that had played the game previously with results from those that had not. Anecdotal evidence suggests that there is an introductory or learning stage when a player becomes familiar with the game mechanics and its idioms. This stage seems to be characterized by a certain effort of will to overcome the initial low level of reward from the game in order to achieve some future, implicitly promised, yet nebulous greater

reward. Although it was clear that learning and skill development within the game was ongoing, it was also clear that in order to improve the quality of any results subjects should have played the study game at least once previously.

The choice of 'Unreal Tournament' (URT) was finally made following a high positive response rate from the recruited population when questioned as to whether they had played URT previously. Thus, based on availability, the study population was selected from male respondents who had previously played URT.

Measures

Self Completed Demographics

Along with basic demographic details an attempt was also made to collect data on self reported game playing activity. No validated tools are available in the public domain for assessing the frequency of subject game playing and the costs of using the services offered by market research companies were prohibitive.

16PF5 Personality Profile Questionnaire

In 1949, Raymond Cattell published the first edition of the 16PF Questionnaire (Maraist 2002; Russell 1994, 1995) the 16 Personality Factor Questionnaire. It was a revolutionary concept: measuring the whole of human personality using structure discovered through factor analysis. The 16PF Fifth Edition Questionnaire used in this study is stated as representing 'a controlled, natural evolution of the 16PF Questionnaire, enhanced and updated to reflect the changes in today's society.'

The 16PF Questionnaire is a self-report assessment instrument that measures the sixteen normal adult personality dimensions. From client responses to the questionnaire, standardized scores (stems) are derived for each of the sixteen personality factors and scores for five Global Factors (the original Five-Factor Model) are computed. These scores enable the formulation of personality models/hypotheses useful in research for predicting human behavior associated with C&CVG playing.

Using dimensions discovered through factor analysis, the 16PF Questionnaire assesses the whole domain of human personality. It measures levels of:

- Warmth
- Reasoning
- Emotional Stability
- Dominance
- Liveliness
- Rule-Consciousness
- Social Boldness
- Sensitivity
- Vigilance
- Abstractedness
- Privatness
- Apprehensiveness
- Openness to Change
- Self-Reliance
- Perfectionism
- Tension

The 16PF model is hierarchical. When the sixteen primary traits were themselves factor-analyzed, they revealed five Global Factors which describe personality at a broader level.

These Global Factors are:

- Self-Control
- Extraversion
- Anxiety
- Tough-Mindedness
- Independence

These five Global Factors help to show the degree of relationships among the sixteen primary scales. Validity studies presented in the 16PF Fifth Edition Technical Manual provide considerable evidence of the construct validity of the primary and global scales.

Annett's Handedness Scale

Handedness is a vague term, and can mean many things to many people. This problem extends into science and researchers define handedness based on different theoretical assumptions. Handedness can be considered as that which performs faster or more precisely on testing or the hand that one prefers to use (regardless of performance). Alternatively categories can be used containing either 2, 3 or 5 criteria. Because hand preference is considered a marker for cerebral hemispheric dominance for speech and language considering handedness as a continuum and using tools to measure it as such probably reflects the degree of brain lateralization of a subject. (Annett 1970; Broca 1865; Carlstedt 2001, Cerf 1998)

PANAS-X Mood Scale

To assess the emotional states of game players at the point in the time (moment affect) before

and after game play the validated and reliable 60 item Positive and Negative Affect Scale – Expanded (PANAS-X) was used. (Watson and Clark 1994) This easy to administer test can be completed by most subjects in 10 minutes or less giving scores for fear, sadness, guilt, hostility, shyness, fatigue, surprise, joviality, self assurance, attentiveness and serenity. With kind permission from Professor Lee Anna Clark and Professor David Watson.

Stroop Test

Discovered in the 1930's by J. Ridley Stroop (Stroop 1935) the clashing or cognitive dissonance effect of a colour word such as 'blue' appearing in another colour such as red requires a cognitive mechanism called inhibition to stop one response in order to perform the correct one. Most humans are so proficient at reading printed words that they cannot easily ignore their meaning, in fact it requires considerable effort to do so. Thus the Stroop Test uses this phenomenon as a measure of mental vitality and flexibility. This test was delivered using The Genov Modified Stroop test software (with kind permission) measuring speed of completion as well as accuracy.

Heart Rate Monitoring

Heart rate variability (HRV) refers to the beat-to-beat alterations in heart rate. Under resting conditions, the ECG of healthy individuals exhibits periodic variation in the spikes of electrical activity associated with the contraction of the main chambers of the heart. These spikes are termed R waves and the gaps between them the R-R intervals. This rhythmic phenomenon, known as respiratory sinus arrhythmia (RSA), fluctuates with the phase of respiration, being accelerated during inspiration revealing cardio-acceleration and decreasing during expiration, revealing cardio-deceleration. This phenomenon is mediated by the amount of activation the heart receives from the autonomic nervous of the body, being reduced in expiration.

CLIENT-SERVER MODEL OF DELIVERY

Remote testing was one of the key design aims to be piloted. A scenario of delivering the questionnaires and tests online with real-time remote data gathering and interpretation was envisaged. In the pilot study this was progressed as far as the questionnaires. The PHP5 server side object orientated scripting language provided the intelligence behind the secure web delivered questionnaires feeding a

MySQL database that could be scored and interrogated remotely by the testers. The ever reliable, Apache Server software served the pages to the high specification networked game machines.

RESULTS

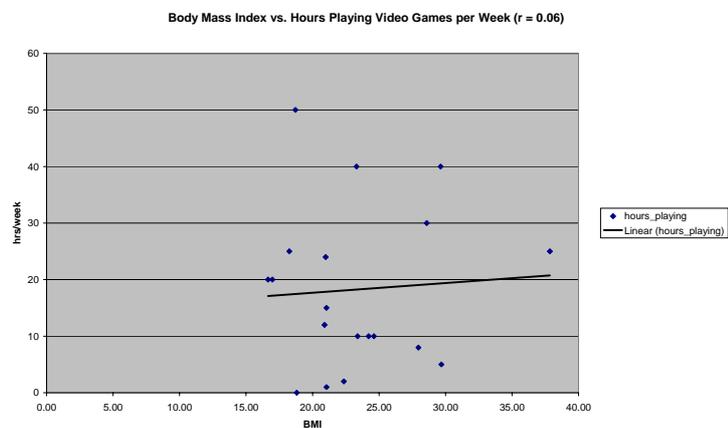
Subject Disposition and Baseline Characteristics

Subjects were aged 17 – 40 years (mean 21.7, 95% CI +/- 0.1) and all males, n = 20 - appendix 1. Subject height and weight (self reported) were used to calculate the body mass index (BMI) in order to have a crude measure of the study population's degree of body fat and hence fitness. BMI being calculated as: weight (kg) / height (m)² giving a population mean BMI of 23.4 (95% CI +/- 0.1). As per table 1. a body mass index between 20.7 and 26.4 is in the 'normal' range for men, according to a recent definition (NHANES II 1980) This definition is used by the World Health Organization (WHO) as its international standard.

| Adults BMI: | Women | Men |
|-------------------|-------------|-------------|
| Underweight | < 19.1 | < 20.7 |
| In normal range | 19.1 – 25.8 | 20.7 – 26.4 |
| Margin overweight | 25.8 – 27.3 | 26.4 – 27.8 |
| Overweight | 27.3 – 32.3 | 27.8 – 31.1 |
| Very overweight | > 32.3 | > 31.1 |

Table 1. Body Mass Index Definitions

Subjects self reported the number of hours spent playing computer or console video games during an 'average week' giving a mean of 18.26 hrs/wk (95% CI +/- 0.2). Comparing BMI with the time spent playing video games produced a product-moment correlation coefficient very close to zero, indicating no relationship between these 2 factors in the study population (r = 0.06).



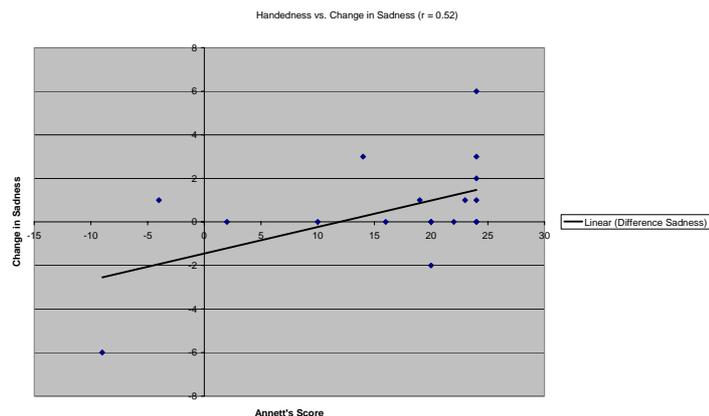
Graph 1 – BMI Correlation with Hours of Game Play

Observations

From the point of view of the observer there is very little activity taking place. As to the sensory activity of the game player, the only senses involved are sight and hearing, and they are only marginally active. The marginal activity is due to the minimal use of most of the eye functions: the lens is not activated (constant distance to the object), nor is the pupil (constant light levels) and the muscles which produce eye movements (fixed object). The sound coming from the loudspeaker only punctuates the visual activity of the game. Although the visual quality and imagery has advanced to a startling degree since the early 'space-invaders' type of C & CVG, there is still little requirement for discriminating vision or hearing sensory effort. C&CVG play, in the subjects observed seemed to be characterized by automatic and rapid motor movements punctuated by groans, grimaces and expletives, elimination of conscious thinking and self-consciousness, and feelings of challenge stimulated by exact objectives. There seemed to be no effort of will required to engage in game play but rather the opposite – forcible physical contact and loud verbal instruction were required to interrupt play. Similarly the urge to initiate play was so strong that clear written and verbal instructions were ignored by some subjects such that they began to play prematurely!

Handedness

The study group was strongly skewed towards right handedness. However a continuum is present within the study group and when compared with the difference in errors on the pre and post game Stroop tests the product-moment correlation co-efficient hints at a connection between right handedness and increased tendency to be error prone ($r = 0.33$). The relationship between right handedness and increase in speed of completion of the Stroop test in the study group was stronger ($r = 0.4$). Surprisingly the change in the moment affect modality of 'sadness' showed some correlation with handedness ($r = 0.52$). Within the study group the more right handed the subject the more inclined to show an increase in 'sadness' following exposure to game play.



Graph 2 – Handedness Correlation with Change in Sadness

Cognitive Function

A method of assessing significance in a clinical setting is the *percent improvement*. The convention is that a 25% change represents a significant difference. This technique has transferred well into this domain of research. For cognitive function 90.9% ($n = 10$) of the subjects showed an increase in speed of completion of the Stroop Test (Mean = 0.1s +/- 0.8s). With a population percent improvement of 10.02%; of which 33.3% showed an increase in error rate, 33.3% a decrease and 33.3% no change. The Product-Moment Correlation Coefficient ($r = 0.17$) showed no significant correlation between change in speed of completion and change in error rate.

Moment Affect

Percent improvement in the 8 moment affect modalities were: positive affect (-3.26%), hostility (+8.87%), fear (-5.26%), sadness (+6.76%), guilt (+8.90%), fatigue (-16.88%), surprise (+53.25%) and shyness (-10.48%). The variances and standard deviation (SD) were generally large: positive affect (SD = 7.76), hostility (SD = 5.43), fear (SD = 1.98), sadness (SD = 2.31), guilt (SD = 4.29), fatigue (SD = 1.89), surprise (SD = 3.19) and shyness (SD = 2.06). Thus, the counter-intuitive result is that the most significant change in the study population's mood was the increase in the surprise modality. Subjects scoring the words *amazed*, *surprised* and *astonished* much more highly after playing the game. The other key changes were reduction in fatigue, shyness and increase in hostility suggesting that the game play experience is an alerting and activating one. However, the fall in positive affect and increase in sadness and guilt imply that the

game play experience is not, nor need be necessarily a pleasurable one.

Personality and Affect

Correlating the 11 modalities of personality measured by the 16PF Questionnaire with the 8 of moment affect measured by the PANAS-X results in 120 scatter plots, line of best fit and calculation of the Product Moment Correlation Coefficient (PMCC) similar to the graphs presented above for body mass index and handedness. In order to aid the interpretation of such a large number of graphs software was developed with our visualisation colleague James Osbourne to produce a coloured 3D statistical surface which could be back projected and viewed with 3D flicker glasses in order to immerse the researchers in their data.

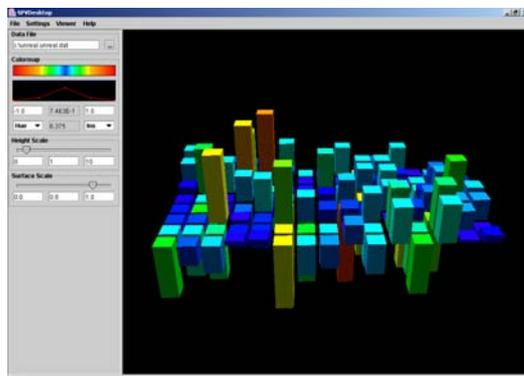


Figure 1 – Prototyping the Visualization



Figure 2 – A Collaborative Discussion

Comparing personality modalities with the change in moment affect modalities revealed weak positive correlations between: liveliness and change in shyness ($r = 0.503$); rule-consciousness and change in fear ($r = 0.537$) and weak negative correlations between: social-boldness and change in sadness ($r = -0.526$); abstractedness and change in fatigue ($r = -0.59$).

DISCUSSION

Body Mass Index

Attempts to assess the degree to which subjects fit the stereotype that video game players are overweight, unfit and at greater risk of suffering ill health produced no evidence to support this stereotype. The hypothesis that the more hours spent playing C&CVG the more unfit the person did not hold in the study population and rather with a product-moment correlation coefficient close to zero ($r = 0.06$) the null hypothesis is validated. However, larger sample sizes would be required to give enough power to conclusions and before being able to expose the stereotype as a shibboleth. Further, body mass index is a reliable indicator of total body fat, which is related to the risk of disease and death (National Heart, Lung and Blood Institute 1998). The score is valid for both men and women but it does have some limits. It may overestimate body fat in athletes and others who have a muscular build. It may also underestimate body fat in older persons and others who have lost muscle mass. According to the National Heart, Lung and Blood Institute guidelines (www.nhlbi.nih.gov), assessment of risk from being overweight involves using three key measures. Namely, body mass index (BMI) waist circumference, and risk factors for diseases and conditions associated with obesity. (WHO 1997) Future work ought to collect this data explicitly with a view to looking for correlations between health and time spent playing computer video games.

Handedness

Using a continuum model of handedness to reflect brain lateralization seems valid (Briggs & Nebes 1975), but does brain lateralization play any role in the gaming experience? The human brain is a paired organ; it is composed of two halves (cerebral hemispheres) that look pretty much alike. The term brain lateralization refers to the fact that the two halves of the human brain are not exactly alike. Each hemisphere has functional specializations whose neural mechanisms are localized primarily in one half of the brain or the other. In this manner the brain avoids duplication of function. The hemispheres always work together so that we will experience a combination of right and left hemisphere in everything we do. However, there is a

tendency for one hemisphere to be dominant. This dominance affects how we respond to new experiences and situations and may well play a role in the gaming experience. Although the study population was small and the statistical conclusions consequently weak there was enough evidence to question the null hypothesis.

Game Play Affects Cognitive Function

The study population showed a definite increase in cognitive activation characterised by the marked increase in speed of completion. The impact on error rates was not resolvable in the pilot study due to the small number of subjects. A reasonable hypothesis would be that any C&CVG game hoping for success must generate these kinds of changes in cognitive function in samples from the target population.

Personality Affects Mood Changes

Although only a small scale pilot many nuggets of information present themselves. The positive correlation suggested between lively personality types and reduction in the moment affect modality of shyness. Subjects of a more rule-conscious nature may show a positive correlation with change in fear following playing Unreal Tournament. The negative correlation with sadness and socially-bold personalities and the observation that those of a more abstract nature show a negative correlation with change in fatigue could inform decisions about game design. However, the $>.5$ correlations show a lack consistency in terms of any pattern. Therefore, as would be expected for a population size of only 20, the correlation profile fails to disprove the null hypothesis and speculation about which correlations happened to come out highest should be avoided.

“Personality is that which permits a prediction of what a person will do in a given situation.”
Raymond B. Cattell. The potential to predict how certain personality types will react to in certain game play situations has potential for informing design and marketing decisions. However such nuggets could turn out to be Fool’s Gold without larger scale studies to add power to such conclusions.

Physiological Measurement

Despite very noisy data from the in game recording of heart rate the results suggested heart rates in an exercise region equivalent to

running for the majority of the experiment. This phenomenon is in keeping with the general emotional and cognitive activation shown so far. Future work such as the spectral analysis of the RR interval during game play would be valuable to visualize for direct examination of the information encoded in the frequency, phase and amplitude about the impact of game play on the sympathetic-parasympathetic nervous system axis.

CONCLUSION AND APPLICATIONS

Is it because the players stop thinking about their problems or is it the excitement of the competition that fastens the player to the situation created by the game? What profile of psychological and physiological impact does a successful game have? Are the effects of different genres different? What role does age, sex or ethnicity have upon the result? Does the release of adrenalin or natural euphoriant chemicals give the player a feeling of wellbeing related to the stimulus received by the game playing?

We believe that exploring, measuring and profiling the effects of playing C&CVG on representative samples of subjects will inform hypotheses on their design and marketing. Empowering design with effective feedback is the key to continuing success and reliable investment.

AKNOWLEDGMENTS

David Watson and Lee Anna Clarke
University of Iowa
PANAS-X copyright, 1994, D. Watson & L. A. Clark, reprinted with permission
Web Link to the PANAS-X manual:
<http://www.psychology.uiowa.edu/Faculty/Clark/PANAS-X.pdf>

James Osborne
The University of Hull
<http://www.jamesosborne.net/>
J.A.Osborne @dcs.hull.ac.uk

REFERENCES

- Annett, M. 1970. “A classification of hand preference by association analysis.” *British Journal of Psychology* 61:303-321.
- Briggs G and Nebes R: 1975 “Patterns of hand preference in a student population.” *Cortex*, 11:230-238
- Broca, P. 1865. “Du siège de la faculté du langage articulé.” *Bulletins de la Société d'Anthropologie* 6:377-393.
- Carlstedt, R. 2001. “Line Bisecting Test Reveals Relative Left Brain Hemispheric Predomance in Highly Skilled

Athletes: Relationships Among Cerebral Laterality, Personality and Sport Performance.” *Dissertation*

Cerf B. et al 1998. “Functional lateralization of human gustatory cortex related to handedness disclosed by fMRI study.” *Ann. N. Y. Acad. Sci.* 30:855:575-8

National Heart, Lung and Blood Institute. June 17, 1998 *Clinical Guidelines on the Identification, Evaluation, and Treatment of Overweight and Obesity in Adults.*

National Heart, Lung and Blood Institute Definitions June 17, 1998 NHANES II Survey (USA 1976 – 1980)

Maraist C. and Russell M. 2002 “16PF Fifth Edition Norm Supplement, Release 2002, Institute for Personality and Ability Testing” Illinois USA

Russell M. and Darcie L. 1995 “The UK edition of the 16PF5: Administrator’s Manual.”

Russell M. and Karol D. 1994 “Administrator’s Manual for 16PF Fifth Edition” Institute for Personality and Ability Testing, Illinois USA.

Stroop J. R. 1935. “Studies of Interference in Serial Verbal Reactions.” *Journal of Experimental Psychology*, 18, 643-662

Yonkers 1990 “Measurement of Personality” New York: World Book Co.

Watson D. and Clark L. A. 1994. “The PANAS-X Manual for the Positive and Negative Affect Schedule – Expanded Form.”

WHO, Geneva, June 1997 *Preventing and Managing the Global Epidemic of Obesity.* Report of the World Health Organization Consultation of Obesity.

GAMES AND PEER-TO-PEER FILE SHARING: ATTITUDES TOWARDS ILLEGAL DISTRIBUTION OF COMPUTER GAMES

Mats Wiklund
Department of Computer and Systems Sciences
Stockholm University
Forum 100
164 40 Kista
Sweden
E-mail: matsw@dsv.su.se

KEYWORDS

Computer games, Peer-to-peer file sharing, Software piracy, Attitudes.

ABSTRACT

As peer-to-peer file sharing is a widespread and user friendly technique ideally suited to distribute illegally produced copies of computer games, the users attitudes towards acquiring games through this medium is of great interest.

To obtain information on the extent to which peer-to-peer file sharing is associated with computer games distribution, and the nature of these associations, an empirical study was conducted. Children from age 10 to age 15 were interviewed about their computer based communication habits and attitudes. To ensure unbiased results, games and games related issues were never brought up by the interviewer.

Results show that the distribution of computer games were *spontaneously* pinpointed by 15.58% of the interview subjects being asked about their peer-to-peer file sharing habits. Younger students showed a significantly more positive attitude towards this activity, while a majority of the older students pointed out negative aspects of acquiring computer games this way. Through the negative quotes given, the concept of empathy with game designers is identified as having potential as a possible counterfactor.

BACKGROUND

The computer games phenomenon is a highly diversified one, showing a wide variety of genres and functionality. While it may not be possible to select one single feature that with certainty has influenced the games development more than all others, it does seem clear that the area of communication is at least a strong candidate. With it has come popular genres like multiplayer versions of first-person action games and massively multiplayer online role-playing games. Such communication features embedded in computer games have both introduced new dimensions to the player experience, as well as taken over part of the responsibility for narration and game plots. This shift can be quite notable, as Klastrop puts it: "*Did anyone notice when the story left?*" (Klastrop 2001). Multiplayer games has, at least to a degree, replaced predefined story with the distributed efforts and interactions of all those participating in the game. Used this way, player-to-player interaction is

enhancing the player experience through in-game communication.

In-game player-to-player communication can be divided into in-character and out-of-character communication, referring to the way the players carry out the communication. Using in-character communication, performed during gameplay and mediated through the players avatar, players can add to the games atmosphere while preserving the style of the game. Such in-character communication can be a very important aspect of the game, and players may rely on it heavily. As shown in a survey by Heide Smith, a majority of the participants either mostly agrees or totally agrees to the statement "*Communication/chat with other players is an appealing part of online gaming*", and 81.4% of the participants (those replying "*Sometimes*" excluded) stated that they often or all the time judged other players on the basis of dialogue (Heide Smith 2003).

The in-game intraplayer communication can leave a very strong impression on the player, possibly the strongest. As Klastrop describes her impressions after a study involving active participation in the multiplayer online role-playing game EverQuest: "*I also take with me the experience of becoming part of a social network which goes beyond the individual character and also includes sharing a communal experience of EverQuest as a prop and tool...*" (Klastrop 2003). Also in multiplayer action games with a modest number of players and a high degree of fast combat situations, player-to-player communication may be a key feature, as observed by Wright et al.: "*The meaning of playing Counter-Strike is not merely embodied in the graphics or even the violent game play, but in the social mediations that go on between players through their talk with each other and by their performance within the game. Participants then, actively create the meaning of the game through their virtual talk and behavior borrowing heavily from popular and youth culture representations.*" (Wright et al. 2002).

Apart from in-character communication, also out-of-character communication may be appropriate, although it may be discouraged in games that rely heavily on atmosphere. This is noted by Pajares Tosca in her study of the online version of the role-playing game Vampire: The Masquerade – Redemption: "*In my experience, players will only go OOC*" [Out-Of-Character] "*when they experience some technical problem (or for example wants to tell the others that they are going to be AFK, Away From Keyboard,*

for a couple of minutes.” (Pajares Tosca 2001). Keeping the communication within the boundaries of game atmosphere may be even more important if it is implemented in such a way that allows not only text based messages but also sound. In such a case, crucial parts of the “*Rich Interaction*” outlined by Manninen can be implemented in multiplayer games (Manninen 2001, 383-398). Such interaction can (apart from visual keys) include paralanguage, defined as the non-verbal audio part of speech (Manninen and Kujanpää 2002, 383-401), and informative spatial sound effects that might add significantly to realism. As Furness points out: “*Humans like parallel input. People make use of a combination of sensory stimuli to help reduce ambiguity. The sound of a letter dropping into a mailbox tells us a lot about how full the mail box is. The echoes in a room tell us about the material in the fixtures and floor of a room.*” (Furness 2001, 80-98).

The observation that in-game communication is a powerful tool that can add significantly to gameplay experience is also supported by the extent to which communication-intensive games are played. A survey conducted by Egenfelt-Nielsen in 2002 showed that 70.91% of the participating players played online games 6 hours or more per week, and 46.94% played 12 hours or more per week. As many as 17.24% played 24 hours or more per week (Egenfelt-Nielsen 2002). A study by Castronova on the online massively multiplayer role-playing game EverQuest shows that 31.5% of the players over 18 years of age devoted more time in a typical week to playing EverQuest than they did to working (Castronova 2001). Regarding the number of players involved in multiplayer online gaming, Sony Online Entertainment Inc. reports having sold over 2 million copies of EverQuest, experiencing over 118,000 simultaneous players during peak hours. Sony Online Entertainment Inc. reported having more than 750,000 active player accounts (including other game titles from the same company) in May 2004 (Sony 2004).

The Other Side of The Communication Coin

Apart from the in-game player communication outlined above, also out-of-game communication between players may occur. Undoubtedly, this too can be of great value to gameplay, but in this case there are aspects of the communication that game designers might disapprove of. Such downsides of computer based communication (from the game developers point of view, and possibly also by quite a few players), includes the buying and selling of game world artefacts and game characters. As described by Castronova, game entities are sometimes being traded for real money in web based auction houses: “*Records at one web site show that on an ordinary weekday (Thursday, September 6, 2001), the total volume of successfully completed auctions (N=112) was about \$9,200.*” (Castronova 2001). In a later paper Castronova notes some countermeasures emerging in the design of games: “*Two games (Ultima Online and Dark Ages of Camelot) now offer methods to effectively start out ahead*” ... “*These strategies help companies discourage the buying and selling of avatars outside the game, perhaps at a cost to the atmosphere within the world.*” (Castronova 2003).

A special case of out-of-game communication that may have severe effects for game developers is the distribution of illegally produced copies of the computer games themselves, often referred to as software piracy. It is estimated by the commercial software industry’s interest organisation Business Software Alliance that 36% of all commercial

software units installed world-wide in 2003 were illegally obtained, with peer-to-peer file sharing being pointed out as a factor increasing this software piracy (Business Software Alliance 2004). The functionality provided by peer-to-peer file sharing technology is ideal for distributing and acquiring illegally copied games, while retaining a high degree of anonymity. As peer-to-peer file sharing technology is widespread, the extent to which it is being used to distribute illegally created copies of computer games is a key issue. Of special interest in this context, from the perspective of stopping software piracy of games, are the users attitudes towards games being copied and distributed illegally, and the possible association of this activity with peer-to-peer file sharing systems.

A survey performed on-line by Harris Interactive in 2004 has compared young computer users attitudes towards obtaining various types of material through the internet. Results show that 83% of the 8-18 year old participants were aware that computer games are copyrighted, slightly fewer than the 88% of them that were aware of movies and music being copyrighted. Although not specific about the internet technology being used (peer-to-peer file sharing, or other), the survey also shows that 32% of the respondents had actually downloaded computer games from the internet without paying for them (Harris Interactive 2004). The scale of the problem is increased by technological advances such as increasing transmission speeds, since faster internet connections enable users to download large files, such as computer games, more quickly. By the end of 2003, there were 70 million households with broadband internet connections, a figure that is estimated to reach 170 million broadband connections by the end of 2007 (Business Software Alliance 2004).

RESEARCH QUESTION

As peer-to-peer file sharing can be perceived as more or less closely associated with the illegal distribution of computer games, there is a possibility that some individuals see peer-to-peer file sharing as a natural way to acquire computer games. Since a high degree of user friendliness in combination with easily achieved anonymity may increase the likelihood that illegal activities are carried out, the users attitudes towards such activities are crucial. In the absence of other types of barriers, the users attitudes are all that stand between them and active software piracy of games. The question arises: Is the illegal nature of copying (most) games this way hidden, or perceived as less serious, obscured by the availability and ease of use provided by peer-to-peer systems? And if so, is there something in the users view of the situation that might serve as a barrier preventing such illegal copying of games?

The research issue addressed in this paper is to find out if peer-to-peer file sharing is primarily associated with illegal distribution of computer games by some individuals, and, if so, the nature of these associations. An attempt is made to identify user attitudes with a potential to serve as counterfactors against the illegal copying of computer games.

METHODOLOGY

The empirical contribution of this paper consists of a study that was conducted using in-depth interviews with students of various ages. The study was conducted in the two cities of Stockholm and Umeå, Sweden, where students in their 4:th to 9:th grades (normally corresponding to the years in which

the students reach the ages of 10 to 15, respectively) were interviewed about a wide range of activities related to communication through computers and mobile phones. In this paper the findings regarding copying of computer games in general, and through peer-to-peer file sharing technology in particular, are described and analysed.

In each class, all the students in the class were interviewed, to ensure that not just students interested in computer related issues participated, but rather the full variety of students present. The classes were selected at random, with the option for the teacher to decline if he/she felt the need to do so. However, all teachers welcomed their students to participate in the study. Students not present in school on the day when their classes were interviewed were excluded from the study.

A key aspect of the interviews was not letting the interview subjects know that computer games were of specific interest. To achieve unbiased results, the interviewer never mentioned computer games or game related issues in the questions. This method was chosen specifically, so that the interviewer did not influence the students to focus on game related issues more than they would otherwise have done spontaneously. Only in follow-up questions when the student already had brought up game related subjects on his or her own initiative, did the interviewer explicitly refer to game related issues.

The interviews were conducted individually in a separate room, away from the class room, with no possibilities of anyone else overhearing the conversations. The students retained full anonymity, only being identified by a sequential number untraceable to the specific individual. Each student was informed of this anonymity, and that his or her answers would not be disclosed to anyone else. By taking these measures, the risk of students not daring to reveal their computer based communication habits were eliminated as much as possible.

During the interviews, the interviewer followed a fixed form with questions to ensure equal coverage of topics with all students. Only follow-up questions may differ somewhat among the students, depending on the answers given. All interviews were recorded in their entirety on a portable tape recorder. Later, the information from the tapes were extracted and entered into a database for processing. Key quotes were translated to English for the purpose of appearing in this paper.

RESULTS

131 students were interviewed, out of which 84 reported using peer-to-peer file sharing. When questioned about their peer-to-peer file sharing habits, 15.58% of the peer-to-peer file sharers gave answers relating directly to computer games distribution. When asked the question *“What is good about peer-to-peer file sharing?”*, 9.52% of the peer-to-peer file sharers pointed out aspects of computer games distribution. When asked the question *“What is bad about peer-to-peer file sharing?”*, 5.95% of the peer-to-peer file sharers pointed out aspects of computer games distribution.

By Age, Grades 4-6

Dividing the interview subjects into age groups, 25.00% of the 32 peer-to-peer file sharers in grades 4-6 (normally corresponding to the year in which the students reach the ages 10, 11, and 12, respectively) gave answers relating

directly to computer games distribution. When asked the question *“What is good about peer-to-peer file sharing?”*, 18.75% of the grade 4-6 peer-to-peer file sharers pointed out aspects of computer games distribution. When asked the question *“What is bad about peer-to-peer file sharing?”*, 6.25% of the grade 4-6 peer-to-peer file sharers pointed out aspects of computer games distribution.

By Age, Grades 7-9

Dividing the interview subjects into age groups, 9.61% of the 52 peer-to-peer file sharers in grades 7-9 (normally corresponding to the year in which the students reach the ages 13, 14, and 15, respectively) gave answers relating directly to computer games distribution. When asked the question *“What is good about peer-to-peer file sharing?”*, 3.85% of the grade 7-9 computer chatters pointed out aspects of computer games distribution. When asked the question *“What is bad about peer-to-peer file sharing?”*, 5.77% of the grade 7-9 computer chatters pointed out aspects of computer games distribution.

Quotes pinpointing distribution of games

| <i>“What is good about peer-to-peer file sharing?”</i> | |
|---|---------------|
| <i>“All the games are there”</i> | Boy, grade 4 |
| <i>“The games that are there”</i> | Boy, grade 4 |
| <i>“The games you get”</i> | Boy, grade 4 |
| <i>“That all the games are free”</i> | Boy, grade 7 |
| <i>“If you can’t afford to buy a game”</i> | Boy, grade 5 |
| <i>“The games!”</i> | Boy, grade 5 |
| <i>“Its good for us that get the games”</i> | Boy, grade 6 |
| <i>“You don’t have to pay for the games there”</i> | Girl, grade 8 |

Table 1. Positive quotes about peer-to-peer file sharing, mentioning games distribution.

| <i>“What is bad about peer-to-peer file sharing?”</i> | |
|--|---------------|
| <i>“When it says it’s a game but it isn’t”</i> | Girl, grade 8 |
| <i>“When the game doesn’t work”</i> | Boy, grade 7 |
| <i>“There can be viruses in the games”</i> | Boy, grade 6 |
| <i>“Its bad for them that made the games”</i> | Boy, grade 6 |
| <i>“Its bad that there are viruses in some of the games”</i> | Girl, grade 8 |

Table 2. Negative quotes about peer-to-peer file sharing, mentioning games distribution.

DISCUSSION AND CONCLUSIONS

It can be concluded from the results described in this paper that peer-to-peer file sharing is being spontaneously associated with computer games distribution by 15.58% of all the interviewed peer-to-peer file sharers. While this figure might seem small at first sight, its still a significant indication that peer-to-peer file sharing is perceived by many as a natural method of distribution of computer games.

The interviewed students are not just those playing computer games, but all the students in the classes in question, thus including those who never play games at all. In the light of this fact, the 15.58% figure is quite impressive, and even more so since the subject of computer games were never mentioned by the interviewer, but associated to spontaneously by the students themselves.

Age Issues

The spontaneous association of peer-to-peer file sharing in general with the distribution of computer games is considerably more common in the younger age group in the study, 25.00% in the grades 4-6 group versus 9.61% in the grades 7-9 group.

It is interesting to note that the ratio of positive versus negative quotes about peer-to-peer file sharing mentioning distribution of games is very different in the two age groups: Among the younger grades 4-6 peer-to-peer file sharers, 75% of the quotes mentioning computer games distribution were given when discussing positive aspects of peer-to-peer file sharing, while only 25% of the quotes mentioning computer games distribution were given when discussing negative peer-to-peer file sharing aspects. In the group of older peer-to-peer file sharers, the situation is reversed: 40% of the quotes involving games distribution were given when discussing positive peer-to-peer file sharing aspects, while 60% of the quotes involving games distribution were given when discussing negative peer-to-peer file sharing aspects.

The underlying reasons behind the older students being more negative to the peer-to-peer game copying phenomenon, while the younger students are being more positive, may possibly be linked to increased insights about various downsides of peer-to-peer file sharing of illegally copied games, that comes with age and experience. This is not the only possible explanation though, and other factors not linked to age or experience may also contribute, as discussed in the section *Quotes From individual Answers* below.

Quotes From Individual Answers

The individual answers from the students during the interviews reveal various situations in which peer-to-peer file sharing is perceived as either having positive or negative sides in the context of distributing copied computer games. This is the result of the students being asked the questions *"What is good about peer-to-peer file sharing?"* and *"What is bad about peer-to-peer file sharing?"*. With the frequencies shown above, the replies to these questions contained references to computer games distribution.

Some of the positive quotes are not explicit about any cost or price factors being involved, although this seems implicitly likely. Instead, these quotes merely refer to the accessibility as such, as in: *"All the games are there"* (boy, grade 4) or *"The games you get"* (boy, grade 4). In other cases, though, the cost is explicitly referred to in the quotes, such as in: *"If you can't afford to buy a game"* (boy, grade 5). Interestingly, some quotes have an air of naivety about them, possibly hinting that the illegal nature of the activities are sometimes not even fully understood by the students: *"You don't have to pay for the games there"* (girl, grade 8) and *"That all the games are free"* (boy, grade 5) are examples of this.

The majority of the negative quotes are directly related to practical issues, such as the downloaded games not working, or being infected by computer viruses. *"When the game doesn't work"* (boy, grade 7) and *"Its bad that there are viruses in some of the games"* (girl, grade 8) are typical examples of this. One quote may relate to either countermeasures to reduce illegal distribution, or possibly practical joking: *"When it says it's a game but it isn't"* (girl, grade 8).

The observed practical nature of the negative concerns expressed in the quotes is consistent with findings from an on-line survey performed by Harris Interactive in 2004: To the question *"Which of the following things have worried you about downloading software, music, or games on the internet in the past without paying for it?"*, the most common answer was *"Accidentally downloading a virus onto the computer"*, given by 60% of the respondents. The more ethically based answer *"Feeling that this is just not something that is right to do"* was just the 5:th most common answer, given by 39% of the respondents (Harris Interactive 2004).

One of the quotes that are being negative to illegal distribution of computer games through peer-to-peer file sharing stands out from the rest conceptually as it displays empathy with game developers losing money through software piracy: *"Its bad for them that made the games"* (boy, grade 6). Different feelings seem to fight each other within this particular student, though, as this same individual also remarks *"Its good for us that get the games"*.

Not everyone is even aware of the fact that game developers suffer from losses through software piracy of games, though. The 2004 on-line study by Harris Interactive shows that 26% of the respondents agreed to the statement *"It doesn't hurt anybody when I do this"*, referring to the downloading of commercial games without paying for them (Harris Interactive 2004). Among those that are aware of the game developers losses, it may still be unknown how much hard work is actually involved in creating a computer game.

It is interesting to note that, although ambivalent, the concept of empathy with game developers made the above mentioned individual perceive peer-to-peer file sharing of copied games as an at least partly negative activity. None of the quotes expressed any worries of getting caught or being made liable for damages, suggesting that empathy with game developers might be a more effective preventive factor. Thus, making the effort and hard work involved in developing computer games visible, and publicly known, may prove to be a more important step than legislation, towards reducing the illegal distribution of computer games.

FUTURE RESEARCH

In this paper, a possible counterfactor against illegal distribution of computer games has been identified: empathy with game developers due to the hard work involved in creating games. To further investigate the potential of this possible counterfactor, it needs to be established if there is a correlation between the degree of perceived empathy with game developers, in relation to the degree with which individuals actually download illegally distributed computer games. If such a study confirms (an inverse) correlation between the two, it may be fruitful to evaluate various methods to increase this empathy, as a means to reduce software piracy of games.

REFERENCES

- Business Software Alliance. 2004. *BSA and IDC Global Software Piracy Study*.
<http://www.bsa.org/customcf/popuphitbox.cfm?ReturnURL=/globalstudy/loader.cfm?url=/commonspot/security/getfile.cfm&PageID=16947>
- Castronova, Edward. 2001. *Virtual worlds: A first-hand account of market and society on the cyberian frontier*. CESifo Working Paper No. 618.
http://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID294828_code020114590.pdf?abstractid=294828&mirid=1
- Castronova, Edward. 2003. On Virtual Economies. In *Game Studies – the international journal of computer game research*, volume 3, issue 2, December 2003.
<http://www.gamestudies.org/0302/castronova/>
- Egenfelt-Nielsen, Simon. 2002. Online gaming habits. In *Game Research – the art, business and science of computer games*.
http://www.game-research.com/art_online_gaming.asp
- Furness, Thomas A. 2001. Toward tightly coupled human interfaces. In *Frontiers of Human-Centered Computing, Online Communities and Virtual Environments*. Edited by Ershaw R, Guejd R, van Dam A and Vince J. London: Springer-Verlag.
- Harris Interactive. 2004. *Tweens' and Teens' Internet Behavior and Attitudes About Copyrighted Materials*.
<http://www.bsa.org/usa/research/loader.cfm?url=/commonspot/security/getfile.cfm&pageid=15786&hitboxdone=yes>
- Heide Smith, Jonas. 2003. Avatars you can trust – A survey on the issue of trust and communication in MMORPGs. In *Game Research – the art, business and science of computer games*, September 10, 2003.
http://www.game-research.com/art_avatars_trust.asp
- Klastrup, Lisbeth. 2001. The art of being there - multi-user performances as net art. In *Localmotives* nr 5, June 2001.
<http://www.it-c.dk/people/klastrup/localmotives/Localmotives.htm>
- Klastrup, Lisbeth. 2003. A poetics of virtual worlds. In *Proceedings of the fifth international digital arts and culture conference*, May 19-23, 2003, RMIT, Melbourne, Australia.
- Manninen, Tony. 2001. Rich interaction in the context of networked virtual environments – experiences gained from the multiplayer games domain. In *Joint Proceedings of HCI 2001 and IHM 2001 Conference*. Edited by Blanford A, Vanderdonck J and Gray P. Springer-Verlag.
- Manninen, Tony, and Kujanpää, Tomi. 2002. Non-verbal communication forms in multi-player game sessions. In *Proceedings of HCI 2002 Conference*. Edited by Faulkner X, Finlay J and Détienne, F. Springer-Verlag.
- Pajares Tosca, Susana. 2001. Role-playing in multiplayer environments, Vampire: The Masquerade – Redemption. Paper presented at the Computer Games and Digital Textualities conference, March 2001, at the IT-University, Copenhagen, Denmark.
- Sony. 2004. Square Enix to publish Sony Online Entertainment's EverQuest® II in Japan. Press release from Sony Online Entertainment, May 11, 2004.
http://sonyonline.com/corp/press_releases/051104_square_sony.html
- Wright, Talmadge, Eric Borgia and Paul Beridenbach. 2002. Creative player actions in FPS online video games – Playing Counter-Strike. In *Game Studies – the international journal of computer game research*, volume 2, issue 2, December 2002.
<http://www.gamestudies.org/0202/wright/>

MASTERMIND WITH AN UNLIMITED NUMBER OF LIES AS A MODELING TOOL OF COGNITIVE PROCESSES INVOLVED IN HUMAN JUSTICE

José Barahona da Fonseca (JBFO@Fct.Unl.Pt)

Department of Electrical Engineering, Faculty of Sciences and Technology, New University of Lisbon
Monte da Caparica, 2829-516 CAPARICA, PORTUGAL

ABSTRACT

In 1986 I made an experiment that consisted in the development of the *Anti-Mind* and *Master Mind with Feedback* programs written in the Basic language (Barahona da Fonseca, 1989). The *Anti-Mind* program simulates a *good* player of the Master Mind game, discovering the secret code defined by the human operator (a sequence of numbers in a pre-defined interval) very quickly. Then I used the algorithm of *Anti-Mind* to help and correct a human operator trying to discover the secret code defined by the computer resulting in the *Master Mind with Feedback*.

Let's take an example to show that in some cases it seems that the computer *thinks* better than the human, and has a higher IQ:

Anti-Mind Program

CPC=Number of Correct Digits in Correct Position

CPI=Number of Correct Digits in Incorrect Position

3 Digits

Interval [0,3]

1. 103 CPC,CPI=1,1
 2. 132 CPC,CPI=1,1
 3. 120 CPC,CPI=0,2
- **Enough Information!**
Secret code=?

The computer knows that the information is enough and the secret code. And you?

In this paper I will also present a recent development, the algorithm of Anti-Mind with an unlimited number of lies which detects and identifies all the moves that are lies or errors, which is based on the Anti-Mind algorithm (Barahona da Fonseca, 1989, 2003), and I will present an example with 6 lies and a secret code with 4 digits varying between 0 and 9. I will discuss, at the light of Cognitive Science, why no human is capable of doing this. Finally I will propose a neural network architecture and a learning algorithm based on a hybrid reinforcement algorithm to model the learning process of a human interacting with Mastermind with Feedback. The Anti-Mind with Lies algorithm seems to be a good departure point to the modeling of human justice and the development of computational aids to help in the process of finding the truth from lots of noisy data and data with lies and training software for lawyers and criminal investigators.

Keywords: Mastermind Game, AI, Cognitive Science, Simulation of Human Behaviour, Human Cognitive Limitations

INTRODUCTION

We all have the idea that the Brain is a very powerful logic processor. My results point to the contrary: it seems the Brain is not so that a powerful logic processor, and most of us have a great difficulty to combine (equivalent to logical conjunction AND) various incomplete informations like in the Mastermind game. The problem is that, if we don't repeat the digits in the first moves, the logical expressions that represent the possible hypotheses coherent with each move are more complex and much more their conjunction. For example, if the first move is 1333 cpc=1 cpe=0, the logical expression of the possible hypotheses coherent with this information is

$(1,1)\&(3,de) \oplus (1,de) \& [(3,2)\oplus(3,3)\oplus(3,4)] \& (3 \text{ doesn't exist in position 1})$

where \oplus represents the exclusive or (XOR) logical operation and (i,j) means *digit i exists in position j* and (i,de) means *digit i doesn't exist*.

But if the first move is

0254 cpc=1 cpe=2, the logical expression of the possible hypotheses coherent with this information is much more complex (assuming a maximum digit of 6):

$(0,1)\& \{ (2,3)\&(5,2)\&(4,de)\&[(3,4) \oplus(1,4) \oplus(2,4) \oplus(5,4) \oplus(6,4)] \oplus(2,3)\&(5,4)\&(4,de) \&[(3,2) \oplus(1,2) \oplus(5,4) \oplus(6,4)] \oplus(2,4)\&(5,2)\&(4,de)\&[(3,3) \oplus(1,3) \oplus(5,3) \oplus(6,3)] \oplus(2,3)\&(4,2)\&(5,de)\&[(1,4) \oplus(3,4) \oplus(4,4) \oplus(6,4)] \oplus(2,4)\&(4,2)\&(5,de)\&[(1,3)\oplus(2,3)\oplus(4,3)\oplus(6,3)]\oplus(2,4)\&(4,3)\&(5,de)\&[(1,2)\oplus(2,2)\oplus(3,2)\oplus(4,2)\oplus(6,2)]\oplus(5,2)\&(4,3)\&(2,de)\&[(1,4) \oplus(3,4) \oplus(4,4) \oplus(5,4) \oplus(6,4)] \oplus(5,4)\&(4,2)\&(2,de)\&[(1,3) \oplus(3,3) \oplus(4,3) \oplus(5,3) \oplus(6,3)] \oplus(5,4)\&(4,3)\&(2,de)\&[(1,2) \oplus(3,2) \oplus(4,2) \oplus(5,2) \oplus(6,2)] \oplus(2,2)\& \{...\} \oplus(5,3)\& \{...\} \oplus(4,4)\& \{...\}$

note that $A\oplus B = A\&\text{not}(B) + \text{not}(A)\&B$, where + means logical OR.

Now imagine the conjunction of various expressions like this...only a very powerful logic processor would be capable to make the conjunction (logical AND) of various expressions so complex without getting lost...as it happens with us when we try to understand how the anti-mind algorithm reaches the conclusion that the information is enough and finds the secret code.

I have rewritten the Anti-Mind and Mastermind with feedback in Matlab[®] with some minor modifications in the algorithm and some more profound modifications in the implementation. I made some simulations that point to a

worst-case performance of 7 moves for the classic Mastermind game (4 digits varying between 1 and 6) and Donald E Knuth (1976) claims, without proof, that his strategy guarantees a maximum of 5 moves! He showed that the best first guess is 1122. Knuth only solved the classic Mastermind game with 4 digits varying between 1 and 6 and didn't try to generalize or solve the same problem for a greater number of digits and/or maximum digit. It seems (Knuth, 1976) was the first serious published work about the solution of the classic Mastermind game. Since then many proposals were published (Chvátal, 1983; Flood, 1985, 1988a,b; Irving, 1978; Koyoma, and Lai, 1993; Neuwirth, 1982; Rada, 1984; Rao, Kazin, and O'Brien, 1986; Shapiro, 1983), but no one did beat the worst case performance of Knuth's strategy.

Inspired in the work of Andrzej Pelc (2002), I developed the Anti-Mind with lies that solves the generalized Mastermind game with an unlimited number of lies or errors. I did an exhaustive bibliographic research and I did not find any published work similar to Anti-Mind with Lies neither to Mastermind with Feedback (Barahona da Fonseca, 1989, 2003).

THE ANTI-MIND ALGORITHM

There are three main ideas behind my very simple anti-mind algorithm. The first one is to translate each move and its cpc and cpe not in a complex logical expression but in a set of *good* moves, that is, coherent with this information. The second one is that to select the subset from the actual good moves can be done simply considering the last move as the secret code and comparing it with the other good moves: the selected good moves must have $cpc_i=cpc$ and $cpe_i=cpe$; this later condition guarantees that the selected good moves have cpc digits coincident with the last move and cpe digits that exist in the last move in different positions, that is, are *coherent* with the new information. The third one is that applying successively this rule of selection is equivalent to the conjunction (logical AND) of the logical expressions that define the good moves associated with each trial.

Before enunciating the algorithm in formal terms let's see an example, in order to acquire the intuition of what is going on.

My anti_mind.m has the syntax `anti_mind(n_digits,max_digit, flag_trace, flag_n_h)` When we make `flag_trace=1`, the algorithm asks after each move if we want to see all the actual good moves, and if we make `flag_n_h=1`, the algorithm will display the number of actual good moves before each trial. So let's consider 4 digits varying between 0 and 5, the secret code being 0535:

```
>> anti_mind(4,5, 1, 1)
Number of Hypothesis=1296
Move 1
Move=4 2 0 4
```

```
cpc=0
cpe=1
Number of Good Hypothesis=276
>>trace?0
Move 2
Move=2 1 5 3
cpc=0
cpe=2
Number of Good Hypothesis=52
>>trace?0
Move 3
Move=3 3 4 5
cpc=1
cpe=1
Number of Good Hypothesis=11
Move 4
Move=5 0 3 5
cpc=2
cpe=2
Number of Good Hypothesis=1
>>trace?0
**ENOUGH INFORMATION**, Secret Code:
0535
```

If you compare the selected 11 *good moves* with move 3, you see that each *good move* has only one digit coincident with move 3, since $cpc=1$, and only one digit that exists in move 3 but in a different position, since $cpe=1$; the remaining two digits don't contribute to cpc and cpe. I regrouped the referred 11 *good moves* to make more clear how the algorithm translates the *logic condition* in a set of moves:

```
Move 3
Move=3 3 4 5
cpc=1
cpe=1
[considering (3,1)->cpc=1 &(3,3)->cpe=1 &(the remaining
positions 2,4 occupied by 0,1 ∉ {3345})&(4,de)&(5,de)]:
3 0 3 1
[considering (3,1)->cpc=1 &(4,2)->cpe=1 &(the remaining
positions 3,4 occupied by 1 ∉ {3345})&(5,de)]:
3 4 1 1
[considering (3,2)->cpc=1 &(3,3)->cpe=1 &(the remaining
positions 1,4 occupied by 0,1 ∉ {3345})&(4,de)&(5,de)]:
0 3 3 1
[considering (3,2)->cpc=1 &(3,3)->cpe=1 &(the remaining
positions 1,4 occupied by 0,1 ∉ {3345})&(5,de)]:
1 3 3 0
[considering (4,3)->cpc=1 &(5,2)->cpe=1 &(the remaining
positions 1,4 occupied by 5 and 1 ∉ {3345}) & (3,de)]:
5 5 4 1
[considering (4,3)->cpc=1 &(5,2)->cpe=1 &(the remaining
positions 1,4 occupied by 1 ∉ {3345}) & (3,de)]:
1 5 4 1
[considering (5,4)->cpc=1 &(3,3)->cpe=1 &(the remaining
positions 1,2 occupied by 0 ∉ {3345}) & (4,de)]:
```

```

0 0 3 5
[considering (5,4)->cpc=1 &(3,3)->cpe=1 &(the remaining
positions 1,2 occupied by 5 and 0 ∉ {3345}) & (4,de)]:
0 5 3 5
[considering (5,4)->cpc=1 &(4,2)->cpe=1 &(the remaining
positions 1,3 occupied by 1 ∉ {3345}) &(3,de)]:
1 4 1 5
[considering (5,4)->cpc=1 & (3,3)->cpe=1 &(the remaining
positions 1,2 occupied by 5 and 0 ∉ {3345}) & (4,de)]:
5 0 3 5
[considering (5,4)->cpc=1 & (4,2)->cpe=1 &(the remaining
positions 1,3 occupied by 5 and 1 ∉ {3345}) & (3,de)]:
5 4 1 5

```

Although the number of *good moves* varies depending on the selection (which is random) of each trial, we can say that for this code we have about $N_{\text{possible_games}}=1296*276*52*11=204.02.112$ manners to find this code. Due to the limitations of the PCs to which I had access, I did use only 200 *enough information games* for each possible secret code in the referred simulations with `anti_mind_auto.m` and `anti_mind_auto2.m`, which must be very small compared with the *possible enough information games*, that are certainly less than $N_{\text{possible_games}}$ but surely much greater than 200. It is in this sense that the results of my simulations are *not reliable*.

After this digression I think you are just guessing my Anti-Mind algorithm which I will present in pseudo-code:

```

1. input(n_digits,max_digit)
2. n_good_moves=(max_digit+1)n_digits
3. move_order=1
4. good_moves(1,1 to n_digits)=zeros(1,1 to n_digits)
5. while n_good_moves > 1
    if move_order = 1
        move=round(random('uniform',0,max_digit,
1 to n_digits))
    else
        move=
good_moves(index(round(random('uniform',1,n_good_
moves)),1 to n_digits)
    display(['Move ', move_order, ' ', move])
    input(cpc,cpe)
    if cpc = n_digits
        display(' I got it!')
        break
    if move_order=1
        [good_moves,
index]=generate_good_moves(n_digits,max_digit,move
,cpc,cpe)
    else
        [index,n_good_moves]=
select_good_moves(n_good_moves,good_moves,index,
move,cpc,cpe)

```

```

6. if n_good_moves=1
    display(['**Enough Information**
        Secret Code=', good_moves(index(1),1 to
n_digits)])
7. if n_good_moves=0
display('**You Didn't Respect the Rules!**')

```

MASTERMIND WITH LIES

Before presenting my Anti-Mind with lies algorithm, which finds any secret code with an unlimited number of lies, let's see an example with 6 lies, 4 digits varying between 0 and 9 and secret code 1559:

```

>> anti_mind_w_n_lies(4,9,0,1)
Number of Hypothesis=10000
Move 1=9 2 5 4, cpc=0, cpe=3, Number of
Hypothesis=312
Move 2=5 9 8 2, cpc=0, cpe=2, Number of Good
Hypothesis=93
Move 3=4 5 6 9, cpc=3, cpe=0, Number of Good
Hypothesis=6
Move 4=4 5 3 9, cpc=3, cpe=0, Number of Good
Hypothesis=5
Move 5=4 5 0 9, cpc=3, cpe=0, Number of Good
Hypothesis=4
Move 6=4 5 7 9, cpc=2, cpe=0, Number of Good
Hypothesis=0
**You Said one or more Lies...and Later On I will Tell you
Where!!**
Number of Hypothesis=4
Move 7=4 5 4 9, cpc=2, cpe=0, Number of Good
Hypothesis=0
**You Said More than 1 Lies...But I will Find Them!!**
Number of Hypothesis=4
Move 8=4 5 7 9, cpc=3, cpe=0, Number of Good
Hypothesis=3
Move 9=4 5 7 9, cpc=2, cpe=0, Number of Good
Hypothesis=0
**You Said More than 2 Lies...But I will Find Them!!**
Number of Hypothesis=3
Move 10=4 5 9 9, cpc=3, cpe=0, Number of Hypothesis=2
Move 11=4 5 4 9, cpc=2, cpe=0, Number of Good
Hypothesis=0
**You Said More than 3 Lies...But I will Find Them!!**
Number of Hypothesis=2
Move 12=4 5 1 9, cpc=2, cpe=1, Number of Good
Hypothesis=0
**You Said More than 4 Lies...But I will Find Them!!**
Number of Hypothesis=1
Move 13=1 5 9 9, cpc=3, cpe=0, Number of Good
Hypothesis=0
Number of Hypothesis=1
Move 14=1 5 0 9, cpc=3, cpe=0, Number of Good
Hypothesis=0
Number of Hypothesis=1
Move 15=1 5 3 9, cpc=3, cpe=0, Number of Good
Hypothesis=0
Number of Hypothesis=1

```

Move 16=1 5 6 9, cpc=3, cpe=0, Number of Good Hypothesis=0
****You Said More than 5 Lies...But I will Find Them!!****
 Number of Hypothesis=1
 Move 17=1 5 5 9, cpc=4, cpe=0
**** You Lied 6 Times in Moves: 1 3 4 5 8 10 ****

After this digression you must be guessing the algorithm of Anti-Mind with lies, which I present next in pseudo-code, first at a high level of abstraction and then at a more detailed level of description:

1. [flag_lies,p_moves]=anti_mind(n_digits, dig_max)
2. n_lies=1
3. while flag_lies
4. [lies,f_end]=generate_combination_of_lies(n_lies)
5. if f_end n_lies=n_lies+1
6. else moves=disregard(p_moves,lies)
7. [flag_lies,p_moves]=anti_mind2(n_digits, dig_max,p_moves,moves)
8. endif
9. endwhile
10. display('You Lied in Moves': lies)

```
function anti_mind_w_n_lies(n_digits,dig_max)
anterior_moves=zeros(1, n_digits +2)
% anterior_moves(1,1)=number of previous guesses
% anterior_moves(1,2)=flag_lies
% anterior_moves(i+1,1:n_digits)=guess_i
% anterior_moves(i+1, n_digits + 1)=cpc_i
% anterior_moves(i+1, n_digits + 2)=cpe_i

anterior_moves= anti_mind6(n_digits, dig_max,
anterior_moves)

flag_lies=anterior_moves(1,2)

if flag_lies
disp('**You Said one or more Lies...and Later On I
will Tell you Where!!**')
else
return % go back to Matlab line of command...
end % of 'if flag_lies'

n_lies=0 % total number of lies
n1=anterior_moves(1,1)
cf=(dig_max+1)^n_digits % number of possible secret
% codes

while flag_lies % main outer cycle

n2=anterior_moves(1,1)+1

n_lies=n_lies+1

if n_lies < (n1+1)
```

```
n_lies_n1_l=n_lies

else

n_lies_n1_l=n1

end % of 'if n_lies < (n1+1)'

% varying the number of lies in n1
for n_lies_n1=n_lies_n1_l: -1 :1

% Initialising lies(i)
for i=1:n_lies_n1
lies(i)=i+1;
end

for i=n_lies_n1+1:n_lies
lies(i)=n1+ i-n_lies_n1+1;
end

flag_limite=1; % this flag means that we didn't
% exhaust all the possible ways of
% 'making' n_lies

while flag_limite*flag_lies

% first regenerate good_moves neglecting moves lie-1 %
assumed as lies that are coherent with this new
% subset of anterior moves

new_combination=zeros(1,n_digits);
cardinal_g_m=0 % cardinal of good moves

for c=1:cf % generate all the possible
% combinations and see which of them
% are coherent with
% anterior_moves/lies
flag_coher=1; % coherence flag

% let's see if new_combination is coherent with all the
% previous guesses and cpc_i and cpe_i

for j_a=2:anterior_moves(1,1)+1
j_a=see_if_j_a_is_in_lies(j_a,lies,n_lies)

if j_a > anterior_moves(1,1)+1
break % if this condition is true that means that
% don't make sense to search for more
% anterior guesses since we reached the last one
end

% second calculating cpc_i and cpe_i resulting from % the
comparison between new_combination and
% anterior_moves(j_a,1:n_digits)
```

```

[cpc_i cpe_i]
=calculate_cpc_cpe(new_combination,anterior_moves(j_a,1
:n_digits))

flag_coher=(cpc_i==anterior_moves(j_a,n_digits+1))

if flag_coher
flag_coher=(cpc_i==anterior_moves(j_a,n_digits+2))

else

break

end % of 'if flag_coher'

if (1-flag_coher)
break % since we are making the logical AND, if
    % flag_coher=0, it don't makes sense to continue
end

end % of for j_a=..

if flag_coher
cardinal_g_m=cardinal_g_m + 1
good_moves(cardinal_g_m,1:n_digits)=
new_combination(1:n_digits)
end

new_combination=
generate_new_hypothesis(new_combination, dig_max)

end % of the for c=...

flag_lies=(cardinal_g_m==0)

if (1-flag_lies) % flag_lies=0 means we need more
    % information to reach a conclusion!
anterior_moves=anti_mind6(good_moves,
    cardinal_g_m,anterior_moves)
flag_lies=anterior_moves(1,2)

end % of 'if (1-flag_lies)'

% generating new combination of possible lies

if flag_lies
lies=generate_new_lies(n_lies,n_lies_n1,lies,n1,n2)
flag_limite=act_flag_limite(lies,n_lies_n1,n_lies,n1,n2)
end

end % of the 'while flag_limite*flag_lies'

if (1-flag_lies)
break % if flag_lies=0 that means we have found the
    % secret code and 'lies' is the set of guesses
    % where the human user lied: so we must go
    % out of the 'for n_lies_n1=... ' cycle

```

```

end

end % of the 'for n_lies_n1=...'

if flag_lies
disp(['**You Said More than ' integer2string(n_lies)
    ' Lies...But I will Find Them!**' ]);
end % of 'if flag_lies'

end % of the outer cycle 'while flag_lies'

disp(['**You Lied ' integer2string(n_lies) ' Times in
    Moves:']);

disp(['** ' integer2string(lies-1) ' **']);

% in Matlab, making an array-1, results in the
% subtraction of 1 in all elements of the array

return

```

To help you to understand this tricky algorithm I will present the following results, without demonstration:

Anti-Mind Theorem: If the codemaker, the human user, never makes errors neither say lies, then the Anti-Mind algorithm (Barahona da Fonseca, 1989, 2003) always finds the secret code in a finite and bounded number of moves.

Anti-Mind with Lies Theorem: If the codemaker, the human user, makes at least one error or say at least one lie, then the Anti-Mind algorithm will always reach a situation where the set of good moves (or good hypothesis) is an empty one.

Minimum Information to Identify One Lie Theorem: After reaching the conclusion that exist at least one Lie in the first NI moves, that is $\#(good\ moves)=0$, if I discard one of these NI moves, to find that this new set of moves contains at least one lie, it will be necessary to *generate* at least one more guess, for any NI first moves that contain at least one lie.

Minimum Information to Identify N Lies+1 Theorem: Generalizing this last result, after testing all the possible manners of making N Lies without having found the secret code, to test the first way of making N lies + 1 it will be necessary to *generate* at least one more guess, calling the *anti_mind6* function, a variation of my original *anti_mind* algorithm (Barahona da Fonseca, 1989, 2003) that puts in memory the guesses it generates and *don't* assumes that when $\#(good\ moves)=1$ it has found the secret code, as does my *anti_mind* algorithm (*enough information* situation), because this latter one assumes that there are no lies in the previous answers.

As a *corollary of this theorem* we have that $NI+N$ Lies is a lower bound of the number of guesses necessary to find the secret code and identify the lies. In the above example we have $NI=6$, N Lies=6 and we have 17 guesses.

Uniqueness of Lies Set Theorem: When Anti-Mind with Lies finds the secret code discarding the set of previous

guesses *lies*, there will be no doubt that *lies* are the guesses where the human user lied and there will not exist any other guesses where the user lied.

So, the main idea is that after being detected one or more lies, begin to make hypothesis about which of the previous moves were lies, neglect these moves and regenerate the set of good hypothesis and select one randomly until *cpc* is not equal to the number of digits of the secret code, that is until it don't find the secret code.

We begin to make the hypothesis of one lie, then test all possibilities of occurrence of one lie, if we didn't find the secret code then we make the hypothesis of two lies and so on, till we find the secret code.

CONCLUSIONS AND FUTURE WORK

Although a very simple game, Mastermind puts in evidence human cognitive limitations (Stillings *et al.*, 1995). My Anti-Mind algorithm seems to show a performance better than the performance of most human Mastermind players (Barahona da Fonseca, 2003).

To clarify this question I'm planning to make experiments with a sample of very good human Mastermind players with Mastermind with Feedback.

Since nobody till today did beat the worst case performance of Knuth's strategy, that is five guesses, it seems that it may be very near an optimal strategy as defined by Viaud(1979).

This is the context where I will obtain all the possible optimal strategies for the standard Mastermind game and for some variations with a greater maximum digit and/or greater number of digits.

I will also explore the Anti-Mind with Lies and Mastermind with Feedback as a training programs for criminal investigators and lawyers since they work with noisy data and data with lies, and they must detect and identify incongruences in factual and acquired data of a criminal or juridical process and must do good guesses coherent with all factual and acquired data. I'm now working on the *Offline Anti-Mind with Lies* where I consider a set of moves already defined and I consider that may exist various solutions, that is various secret codes; this is equivalent to some complex criminal processes where a subset of the data points to a homicide, another subset to a suicide, another subset to a natural death and another subset to the falsification of the autopsy.

It seems that Intelligence and Creativity result from not 'soft' and complex neural processes but, by the contrary, emerge from very simple brute force search algorithms that may run in parallel.

A more intelligent or creative person would be someone with better and more efficient search and hypothesis generation algorithms, which would not discard any hypothesis of problem solution and does a better *hypothesis test scheduling* and makes better *hierarchies* of them.

REFERENCES

- Barahona da Fonseca, J. (1989). *Curriculum Vitae*, (written in Portuguese), author's edition.
- Barahona da Fonseca, J. (2003). Anti-Mind and Mastermind with Feedback: an Example where the Computer 'Thinks' better than the Human. *Proceedings of Congresso em Neurociências Cognitivas*, November of 2003, in Press, University of Evora, Evora, Portugal.
- Bento, L. and Pereira, L. (1996). Mastermind por Algoritmos Genéticos. *Proceedings of Workshop on Genetic Algorithms and Artificial Life*, pp. 43-47, (written in Portuguese), DEEC, IST, Technical University of Lisbon, Portugal.
- Bento, L., Pereira, L. and Rosa, A. C. (1999). Mastermind by Evolutionary Algorithms. *Proceedings of ACM SAC 99*, pp. 307-311, San Antonio, Brasil.
- Chang, Cheng-Yaw (1993). *Application of a Neural Net to a Deductive Problem: Mastermind*. MSc Thesis, Computer Science Department, University of Texas at Arlington, TX, USA.
- Chvátal, V. (1983). Mastermind. *Combinatorica*, 3:325- 329.
- Flood, M. M. (1985). Mastermind Strategy. *Journal of Recreational Mathematics*, 18(3):194-202.
- Flood, M. M. (1988a). Sequential Search Sequences with Mastermind Variants- part 1. *Journal of Recreational Mathematics*, 20(2):105-126.
- Flood, M. M. (1988b). Sequential Search Sequences with Mastermind Variants- part 2. *Journal of Recreational Mathematics*, 20(3):168-181.
- Irving, R. W. (1978). Towards an Optimum Mastermind Strategy. *Journal of Recreational Mathematics*, 11(2):81-87.
- Koyoma, K. and Lai, T. W. (1993). An Optimal Mastermind Strategy. *Journal of Recreational Mathematics*, 25(4):251-256.
- Knuth, D. E. (1976). The Computer as Master Mind. *Journal of Recreational Mathematics*, Vol. 9(1), 1976, pp. 1-6.
- Lo, P. C. K. (1998). Genetically-Evolved Mastermind Strategy: A Self-Simplifying Symbolic Approach to Reinforcement Learning. in John R Koza (ed.) *Genetic Algorithms and Genetic Programming at Stanford*, Stanford University Bookstore, Stanford, CA.
- Merelo, J. (1996). *Genetic Mastermind, a case of Dynamic Constraint Optimisation*, GeNeura Technical Report G-96-1, University of Granada, Spain.
- Neuwirth, E. (1982). Some Strategies for Mastermind. *Zeitschrift für Operations Research*, 26:B257-B278.
- Pelc, A. (2002). Searching Games with Errors: Fifty Years of Coping with Liars. *Theoretical Computer Science*, 270, 71-109.
- Rada, R. (1984). Mastermind in SIGART. *SIGART Newsletter*, 89:24-25.
- Rao, T. M., Kazin, G. and O'Brien, D. (1986). Algorithms to Play Mastermind. *SIGART Newsletter*, 95:33-35.
- Shapiro, E. (1983). Playing Mastermind Logically. *SIGART Newsletter*, 85:28-29.
- Stillings, N. A., Weisler, S. E., Chase, C. H., Feinstein, M. H., Garfield, J. L. and Rissland, E. L. (1995). *Cognitive Science: An Introduction*, second edition, MIT Press.
- Tesauro, G. (1994). TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, 6(2), 215-219.
- Viaud, D. (1979). Une Formalisation du Jeu de Mastermind. *RAIRO- Recherche Operationelle*, 13(3), 307-321.

Education for Games Design and Development

| | |
|---|------------|
| Doughty, M. Computer games development education at university | 338 |
| Livingstone, D. and McMonnies, A. Is DarkBASIC to be considered harmful? | 342 |
| Masuch, M. and Nacke, L. Power and peril of teaching game programming | 347 |
| Connelly, T.M., McLellan, E., Stansfield, M., Ramsay, J. and Sutherland, J. Applying computer games concepts to teach database analysis and design | 352 |
| Cowling, P., Fennell, R., Hogg, R, King, G., Rhodes, P., Sephton, N. Using bugs and viruses to teach artificial intelligence | 360 |
| Fortuna, H. S. Teaching console games programming with the Sony Playstation 2 Linux kit | 365 |

COMPUTER GAME DEVELOPMENT EDUCATION AT UNIVERSITY

Mark Doughty
University of Lincoln,
Brayford Pool, Lincoln, UK
mdoughty@lincoln.ac.uk

KEYWORDS

Games computing, software development

ABSTRACT

This paper articulates some of the challenges for computer game development courses at university level. A typical course development of this type is described. The need to include creative methods alongside more formal software development methodologies as core elements of computer game education is proposed and placed within the context of an industry specific framework. The evolutionary nature of the computer game industry requires that computer game development programmes at university should be equally evolutionary and adaptable to change.

INTRODUCTION

The computer games industry has risen in global prominence and size during the last ten years (DFC Intelligence, 2004; SSC 2002). The contemporary appeal of interactive entertainment software has resulted in computer games being played by a much broader cross-section of society. This apparent 'legitimization' of gaming as an accepted leisure pastime has also resulted in the software development education sector responding. A wide variety of games development related degree awards have emerged into the university sector since 1997. Current UCAS (Universities and Colleges Admissions service) statistics list 25 universities or institutes offering 59 degree programmes (UCAS, 2004). The academic and skill focus of these awards varies with each institution. Programmes range from being arts and design based, to being purely focused on computer game programming and software development.

CREATIVITY AND SOFTWARE ENGINEERING IN THE GAME DEVELOPMENT INDUSTRY

These two seemingly opposing concepts are both essential components for the success of a computer game product. Creative design and innovation are requirements for compelling gameplay and longevity of game products. Yet, while these are critical to a games success, if the development process is over

budget or deadlines are missed, the benefits of successful design are lost. Rucker proposes two criteria for a successful game: is it beautiful? and does it make money? (Rucker, 2003). These criteria sum up the competing needs for computer game software; games need to be compelling and offer a fun and entertaining experience, and they need to also be a success in the business sense.

Educating potential game developers to embrace and recognize both of these competing criteria requires a curriculum balanced with creative opportunity and software engineering methodology. While conventional thought may see these concepts as being difficult to coexist within a curriculum, it is proposed that not only can they coexist within a games computing curriculum, but that they are both necessary components.

The curriculum developed by many of the institutions listed in the UCAS statistics give students exposure and opportunity to develop skills in all main areas of game development. Students are encouraged to develop their own creativity and essential computing skills, and to recognize that software engineering methods are as important as creative design in the success of a computer game product.

CASE STUDY: A TYPICAL COURSE DEVELOPMENT

During the mid nineties, the Faculty of Applied Computing Sciences at the University of Lincoln looked to consolidate its existing portfolio of computing degrees and also to expand into new and emerging computing disciplines such as computer game development, system security and media technologies. These developments were designed to bring the computing curriculum up to date, and also to respond to the changing demands of the undergraduate market. The first award, BSc(Hons) Computing (Games, Simulation & Virtual Reality), was validated for delivery in 1997. Refinements and refocusing of this undergraduate programme into BSc(Hons) Games Computing occurred in 2001.

The motivations for the development of games computing curricula at undergraduate level are numerous. The leisure software sector has experienced rapid growth, both globally and in the UK. Games software represents the majority of this market (SSC, 2002). The demand for undergraduate computing programmes in this field has increased in line with this increase in popularity of computer games as a leisure pastime. In developing a programme of study which addresses industry and student demand, a number of challenges need to be addressed. The challenges of acceptance of what is often seen as a 'non-academic' or 'rigor-less' subject are faced at both public and university management level. Students, and increasingly their parents, need to see that the course will deliver value for money and value in the job market, while university management need to ensure that standards and benchmarks are reached to ensure an effective educational experience for the student, as well as maintaining university quality responsibilities (THES, 2003). The games computing programme at Lincoln offers the basis for a career as a computing professional. It has undergone rigorous validation and approval to ensure that it meets the standards set by the Higher Education Funding Council for England (HEFCE).

Another critical challenge is the effective integration of creativity and free-form development, considered by many to be at the heart of innovative computer game development, with the structured and more formal nature of software engineering principles and practice.

The computer games industry thrives on creativity and innovation. New ideas, groundbreaking gameplay and interaction mechanisms drive the popularity – and ultimately sales - of computer game units. The responsibility of university games computing curricula is to encourage student activity and achievement in creation and innovation, while ensuring that more structured and formal software engineering principles are upheld as requisites for efficient and economic software development.

The rationale behind the development and operation of the BSc(Hons) Games Computing award is to produce graduates with critical and intellectual abilities within the games computing field, along with domain specific skills and experiences such as creative game design and game software development. In order to satisfy the demands of this rationale, the programme seeks to achieve a number of broad goals (University of Lincoln, 2001):

- To provide students with an education and learning experience that will equip them to operate on graduation as autonomous computing professionals;
- To develop professional and transferable skills in a wide range of methods, techniques and practices appropriate for the task domain of a professional computer game developer; and
- To develop a rich and up-to-date set of practices and techniques which students can deploy in state-of-the-art computer game software design and development contexts.

Curriculum design within the Faculty of Applied Computing Sciences follows a 'thematic' approach (Reeve, 2003). An undergraduate award is defined by a combination of two 'subject themes' and a single skills theme. A 'subject theme' is programme of study that addresses a particular cognate area or field of interest. (see Figure 1).

| | |
|-----------------------------|--|
| Subject of Computing | <ul style="list-style-type: none"> ▪ Computing theme ▪ Games Computing theme ▪ Software Development theme ▪ Internet Systems theme |
| Subject of Informatics | <ul style="list-style-type: none"> ▪ Informatics theme ▪ Applied Informatics theme |
| Subject of Media Technology | <ul style="list-style-type: none"> ▪ Media Technology theme ▪ Digital Media theme |

Figure 1: Curriculum themes within the Faculty of Applied Computing Sciences, University of Lincoln.

The BSc(Hons) Games Computing undergraduate award is comprised of the two themes of Games Computing and Software Development. This combination is one, which reflects the recommendations of the International Game Developers Association (IGDA) for game related educational programmes. The IGDA Curriculum Framework (IGDA, 2003) acts as a guideline for university awards in the computer game development area. It describes the knowledge

areas and practical skills required to make and study games in a flexible format, which allows it to be adapted by individual institutions. Some issues faced when developing a programme of study within this framework are explored in the next section.

The recruitment history for the BSc(Hons) Games Computing at Lincoln shows a steady development and strengthening of the student base during the last six years of recruitment. This steady rise in recruitment figures could be seen as a consequence of the contemporary popularity of computer gaming, and reflects an increasing interest by potential undergraduates in the games computing field. It has also provided positive feedback to the staff evolving the degree.

THE CHALLENGES OF MAPPING PROVISION TO INDUSTRIAL REQUIREMENTS

One of the aims of a provider of university undergraduate awards in any field, is to deliver relevant programmes of study which are responsive and reflective of the needs of the industries into which the graduates progress. The games computing industry is no different in this respect. It does, however, pose its own unique challenges to the developers and deliverers of undergraduate awards in this area.

Among the defining characteristics of a computer game product is the recognition that it is a synergistic combination of creative design effort and software development processes. This characteristic is reflected in the definition of the IGDA Curriculum Framework 'core topics' (see Figure 2). Practitioners and academics have defined these topic areas as a list of general areas relevant to the construction of a game-related curriculum. As acknowledged in the Framework document, no single curriculum can apply to all institutions delivering courses in this field; rather, each institution will interpret and apply its own resources to their course development within the guidance of the Framework.

| |
|--|
| <p>Critical game studies Games and society Game design Game programming Visual design Audio design Interactive storytelling Game production Business of gaming</p> |
|--|

Figure 2: IGDA Curriculum Framework core topics

The faculty or department that develops an

undergraduate programme in this field will naturally utilize the talent and expertise, which are contained within its academic body. Faculties of art and design, for example, may interpret and implement an undergraduate games development programme in a different way to more technology-oriented faculties of computing or computer science.

The application of a guiding framework to an undergraduate degree programme can be enabling in that it can provide an amount of legitimacy and focus to the content. It can give students clear and transparent direction to their studies and their approaches to potential employers upon graduation. When developing or revising a programme of study, multiple frameworks may be referred to in the process. Guidelines such as the accreditation framework of the British Computer Society (BCS, 2004) can be equally applied to games development related programmes, and can indeed be complementary to an industry sector specific set of guidelines, such as the IGDA Curriculum Framework. Recent work commissioned by the Entertainment and Leisure Software Publishers Association (Steele et al, 2004) seeks to further the development of curriculum guidelines and to introduce the possibility of industry led accreditation for programmes of study.

Inspection of the IGDA Curriculum Frameworks core topics reveals that not only are the creative and aesthetic topic areas of computer game development ideally addressed by a programme of study, but that the more disciplined and structured elements of software development and software engineering principles should also be considered. It is evident from the IGDA Curriculum Framework that the games industry requires effective software engineering principles to be a core skill of any games related graduate. The use of software engineering processes in the game development cycle often ensures that games developers are able to learn from mistakes in previous developments (Rollings and Morris, 2000). Rollings and Morris go on to offer a detailed argument and case study material to support this statement.

The evolution of Games Computing programme at Lincoln has been one that reflects this need for software engineering practice. With these factors in mind curriculum development is ongoing, and seeking to identify and apply effective pedagogic principles that incorporate creative practice along with traditional software engineering principles. This has led to new developments in interdisciplinary research by staff from the Faculty of Applied Computing Sciences and

the School of Architecture at the University of Lincoln (O’Coill and Doughty, 2004; Rank et al., 2004). This research is continuing to explore and take advantage of the application of computer game technologies to participatory design within community based architecture projects.

CONCLUSION

The key requisite of any university based computer game development programme is that it has relevance and responsiveness. The game industry has evolved rapidly during the last twelve years, and as a result demands for graduates with specialist knowledge and skills in the games computing field has raised. The development of university-based courses delivered by computing, media and technology based faculties has taken place in line with this industry expansion. Rather than relying purely on graduates from generic disciplines of, for example, computer science, physics and mathematics, the industry can now select from specialist graduates from games related programmes.

One key aspect of the games industry which needs to be reflected in the university based educational programmes is the combination of creative and innovative thinking along with the application of structured software development practices. This aspect is one, which is reflected by the IGDA Curriculum Framework guidelines, and one that is central to the development of the computer games programme at the University of Lincoln.

REFERENCES

British Computer Society, 2004 *Guidelines on Course Accreditation and Exemption* <http://www.bcs.org/BCS/Products/HEAccreditation/courseguidelines.htm>

DFC Intelligence. 2004. *The Business of Computer and Video Games 2004*. www.dfcint.com

Steele, B., Parry, P., Marshall, I., “Proposed Accreditation Scheme for University Courses Delivering Learning for the Entertainment Software Industry” ELSPA Working Paper, version 1, Feb, 2004.

Spectrum Strategy Consultants. 2002. *From Exuberant Youth to Sustainable Maturity: Competitive Analysis of the UK Games Software Sector* Department for Trade and Industry.

IGDA Education Committee. 2003. *IGDA Curriculum Framework: The Studies of Games and Game Development*, v2.3beta. www.igda.org/academia

O’Coill, C., Doughty, M. 2004. “Computer Game Technology as a Tool for Participatory Design”. In *Proceedings of eCAADe*, September 15 – 18, Copenhagen, 2004.

Rank, S., O’Coill, C., Boldyreff, C., Doughty, M. 2004. “Software, Architecture and Participatory Design.” In *Proceedings of SIGSOFT 2004/FSE-12 Workshop on Interdisciplinary Software Engineering Research (WISER2004)*, November 5, 2004. Newport Bch. Ca.

Reeve, P. 2003. *Curriculum Model*. University of Lincoln, Faculty of Applied Computing Sciences, UK.

Rucker, R. 2003 *Software Engineering and Computer Games* Addison Wesley.

Rollings, A., Morris, D. 2000 *Game Architecture and Design* Coriolis Technology Press.

THES, *Now Games Fans Can Console Themselves* Times Higher Education Supplement 21/11/03

IS DARKBASIC TO BE CONSIDERED HARMFUL?

Daniel Livingstone and Alistair McMonnies
School of Computing
University of Paisley
High Street
Paisley
PA1 2BE

Email: {daniel.livingstone,alistair.mcmonnies}@paisley.ac.uk

KEYWORDS

Programming languages, DarkBASIC, Education

ABSTRACT

The use of BASIC to teach programming has long been controversial, and remains so. Modern versions of BASIC allow students to develop skills in structured and even object-oriented programming. DarkBASIC is one such version, which includes procedures and very easy access to DirectX. As such, it can allow students the opportunity to create video game projects with a very low entry barrier. We review DarkBASIC, highlighting a number of particular strengths and weaknesses of the language, and examine whether we can justify its inclusion in a university syllabus.

INTRODUCTION

The programming language BASIC was originally developed for student use, with one of its original design principles being that it should “Be easy for beginners to use” (Wikipedia 2004). Although easy to use, BASIC has always had a problem gaining acceptance within computer science. The ‘go to’ statement, commonly used to jump between different sections of code in BASIC (although not unique to the language) was famously criticised by (Dijkstra 1968), who in a later article made a far more damning claim:

“It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.”

(Dijkstra 1975)

It should be said that this latter article makes a number of unsupported, and quite opinionated claims. FORTRAN is called “the infantile disorder” and a claim is made that COBOL cripples the mind.

Despite Dijkstra’s condemnation, BASIC has remained popular and forms the basis of Microsoft’s highly successful Visual Basic programming language and IDE – taught to many thousands of students at universities across the world. With Visual Basic .NET, BASIC has now evolved into a language possessing a full range of object-oriented programming features (McMonnies 2004). We have used, over the last three years, a recent version of BASIC to introduce first year undergraduate

students to games programming. The language used, DarkBASIC*, incorporates functions, does away with line numbering and allows easy access to many of the powerful features of DirectX. This makes it a strong training ground for games programming.

As part of the Computer Games Technology BSc (Hon) degree an introductory module was developed to give students an overview of games development. One of the goals of this module was to give students the opportunity to create simple graphical computer games in their first semester at university. The environment used would have to be simple to learn – some students enter with no previous programming experience – yet flexible enough to allow students to create a range of games according to their own designs. Teaching programming itself was not one of the aims of the module – students take another programming module (currently C++ based) in the same semester that has this aim.

Accordingly, the environment or language chosen would have to satisfy a few goals:

1. Easy to learn – even for programming neophytes
2. Allow integration of graphics and sound with little programming effort
3. Be flexible
4. Give students a good introductory experience in games creation – with understanding of code/graphics and sound separation and integration
5. Support structured and modular programming – to try to avoid ‘spaghetti code’ and the introduction of too many bad habits

A few competing games development languages and environments were compared. Some were rejected out of hand for quite obviously failing to meet the goals – either being too limited and inflexible (‘no-programming required’ games creation toolkits) or by being seemingly too complex for complete beginners. The final shortlist consisted of DarkBASIC and BlitzBasic (now known as BlitzPlus), and the former was selected for its apparent shallower learning curve and greater immediacy. Over time problems with DarkBASIC have become apparent, and it is now required to evaluate its use and misuse – and decide whether we can justify its continued use, even in the very limited role it has in the degree programme.

* Two versions of DarkBASIC are available from <http://www.thegamecreators.com/>. Both versions possess the many quirks described in this paper.

BOOTSTRAPPING AND BEYOND

The bootstrapping problem when learning to program is an important one to consider, and it is here that DarkBASIC performs particularly well compared to the likes of C++ and Java. Roughly put, in order to learn programming, students have to write programs. And to write even simple programs they need to learn something about the programming language they are using. Consider the following “Hello World” programs:

```
#include <iostream>
main()
{
    std::cout << "Hello World";
}
```

```
class HelloWorld {
    public static void main(
        String args[]) {
        System.out.print("Hello World");
    }
}
```

Figure 1. The C++ (top) and Java Hello World programs

In order to teach even this, we either have to explain a range of language features – likely to confuse and perplex the neophyte– or tell them not to worry about it, as it will all become clear later. Either way, the students’ first program is likely to leave them with the feeling that they don’t really understand what they have just done. Like other versions of BASIC, the single-line DarkBASIC version (PRINT “Hello World”) leaves nothing that requires further explanation. The downside of this ease of understanding is that student may gain unrealistic expectations about the ease of programming as an activity – and be given a false sense of security.

Introducing students to interactive graphics programming introduces far greater levels of complexity in both Java and C++. For both, there are feature rich libraries and APIs to be learned, and programs to perform very simple tasks, such as show a box move around the screen, may require many dozens of lines of code. Figure 2 shows how simple it is draw a picture (loaded from file), and have it move around according to user input in DarkBASIC. The program also uses double-buffering to eliminate flicker.

```
SYNC ON
LOAD IMAGE "tank.bmp", 1
x = 250
y = 200
DO
    CLS
    IF upkey() THEN y = y - 1
    IF downkey() THEN y = y + 1
    IF leftkey() THEN x = x - 1
    IF rightkey() THEN x = x + 1
    PASTE IMAGE 1, x, y
    SYNC
LOOP
```

Figure 2. Loading an image and drawing it on screen under user control.

STRUCTURED PROGRAMMING WITH DARKBASIC

DarkBASIC is certainly easy to learn, but what of its support for structured programming? Like most versions of BASIC, DarkBASIC features both the GOTO and GOSUB commands – which facilitate jumping around code, while keeping all variables globally visible. DarkBASIC also includes a range of other commands for more structured programming – allowing GOTO and GOSUB to be avoided altogether. Before we review those, we begin our look at the language by considering some issues regarding variables.

Variables and Scope

The three common BASIC variable types are all well supported – integer, real number and string. Arrays are declared as usual with the DIM statement, although unusually for BASIC these are zero indexed but are of declaration length+1 to maintain compatibility with other BASIC languages. However, other variables do not need to be declared before use. This can be problematic for learners, as spelling mistakes in variable names lead to programs which execute – sometimes with hard to trace errors. Unlike Visual Basic there is no option to force the declaration of all variables before their first use.

DarkBASIC also maintains a number of system vectors for storing sound and graphics data – the use of these global lists is quite non-standard and has been the source of confusion for some students.

The scope model in DarkBASIC is also quite limiting. Essentially, all variables are local – except for arrays declared in the main body of the program which are automatically global. The only way to create a global variable is to declare it as a single element array.

Further, as with most versions of BASIC, there is no support for “struct”s or user-defined abstract data types. Accordingly, there is no natural way of grouping the data elements for game objects other than by using sets of variables and/or arrays.

Loops

The example shown in Figure 3 demonstrates a repeating and never-ending DO...LOOP (DarkBASIC programs are escapable by hitting the ESCAPE key, however). DarkBASIC also supports FOR loops, as well as REPEAT... UNTIL and WHILE... ENDWHILE loops.

Procedures

DarkBASIC introduces FUNCTION, ENDFUNCTION and EXITFUNCTION keywords, which provide some support for procedural programming – although with some important limitations. Arguments passed into a function are passed by value – DarkBASIC does not support call by reference or pointer. As all parameters are passed by value the only ways to alter the value of variables within a function, and have the variables retain the altered values after the function terminates, are to either return the new value – obviously limiting functions to altering the value of a single variable only – or to make widespread use of variables with global scope. Neither solution is particularly satisfactory – especially when trying to instil good programming practice in young programmers.

Keyword Problems

For working with DirectX, DarkBASIC introduces a large number of new reserved words. Many of these are based quite closely on the API function names used in DirectX, thus exposing students to the API earlier than would otherwise be the case. What is very unusual, however, is that many of the reserved words are in fact multi-word commands. For example, instead of the expected PASTE_IMAGE reserved word to allow an image to be drawn on screen (or to a bitmap in memory), the reserved 'word' is in fact "PASTE IMAGE". Further, "IMAGE" itself is NOT a reserved word, and can be used as a variable name.

The use of multi-word keywords, of which the component parts are not necessarily reserved words, can result in confusing code – and makes confusing code very easily written.

DOCUMENTATION

A simple user manual ships with DarkBASIC, which also includes a number of inbuilt tutorial lessons. There is also a third-party book on learning to program with DarkBASIC (Harbour and Smith 2003). The user manual covers all of the DarkBASIC commands, and while it does include a small section on programming principles, it does not teach programming as a traditional textbook would, and has no coverage at all of algorithms or problem solving. 'Fast Track Learning' lessons jump from a three line 'Hello World' type program, with no variables, to a much more complex 3D graphical game of around a hundred lines. A further ten tutorial exercises are paced much better, but the examples do include the use of the partial keyword "image" as a variable name!

(Harbour and Smith 2003) is better suited for learning to program, but is priced such that it is hard to recommend to students who will only be working with DarkBASIC for three months of their academic career.

LEARNING ABOUT GAMES DEVELOPMENT

It is clear that there are some problems with DarkBASIC, but by using it in teaching it is possible to teach students at an early stage a range of basic and advanced issues and concepts relating to the development of interactive graphical applications.

Basic concepts such as screen resolutions, coordinates, colour depth and RGB values can all be covered simply in class and reinforced by practical work. Double buffering is supported, and easy to use, in DarkBASIC – and examples can be easily put together to demonstrate the difference between buffered and unbuffered graphical applications.

Support for sprites demonstrates the difference between game objects and images – a single object having multiple images, one for each possible pose. And the overall process of creating games in DarkBASIC demonstrates very clearly the separation between development of code, art and sound resources in games – experience has shown that many students don't clearly understand the distinction that exists between game programmers and artists when they start the course.

STUDENT PROJECTS

The main practical deliverable for students in the module is a 2D game project of their own design. Many, but by no means all, of these games are of very high standard given the limited time available, and the limited experience of their authors (see Figure 3 for a screenshot from one such game). These projects show that by using DarkBASIC we have been able to achieve the first four of our five goals. It should be noted that not all students manage complete games, though this is often due to the delayed start of projects.

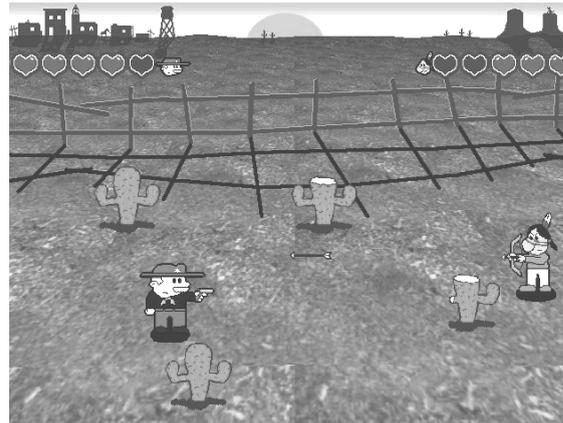


Figure 3. *Quickdraw*, a two player game developed by a student with no previous programming experience.

The fifth goal – of encouraging structured and modular programming has not been so clearly achieved, however. Inspecting the code of student projects is a variable experience. A few projects manage to keep to structured programming principles. A very few use principles of modular programming. But in many cases the principles of good structured and modular programming are most marked by their absence in student projects.

HAS ANY HARM BEEN DONE?

The question in the title of this paper asks whether we should consider DarkBASIC to be harmful. Does using DarkBASIC in a first year module irretrievably harm all the students who use it? The answer to this question would have to be no, it does not. Many students who have taken this module progress well in other programming modules, with no evidence of long term damage.

Looking at results from the most recent first year, we can compare the results gained in two C++ programming modules (Introduction to Programming and 2D Graphics Programming) for students who have taken Introduction to Game Development module (which uses DarkBASIC) against the results for students who have not been exposed to DarkBASIC. These results (Figure 4) appear to show that exposure to DarkBASIC does not impair the learning of C++ programming.

There are three possible explanations for these results

1. Learning DarkBASIC has no negative impact on learning other (structured, modular and/or object-oriented) programming languages/approaches
2. As the second semester module uses DirectX, exposure to elements of DirectX in DarkBASIC

may offset any negative impact from exposure to DarkBASIC's imperfections

- As the second semester module uses 2D graphics programming as a basis for teaching C++, non-games students may perform relatively poorly as they are less interested in graphics programming.

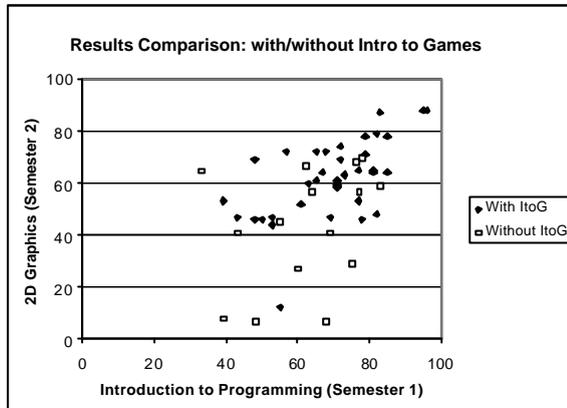


Figure 4. Student performance in the first semester C++ module is a good predictor of performance in the second semester C++ module, whether or not students have been exposed to DarkBASIC.

It is quite possible that all three explanations are true to some extent – although not necessarily for all students. Feedback from students indicates that (3) is true for *some* non-games students. (2) also seems quite likely to be true, and it is clear that for many students (1) is also true. However, student feedback over the years the module has been taught also shows that for some students the effort involved in having to learn two programming languages at the one time is great – and indeed, we know that for many students learning the one programming language is a challenge of significant difficulty.

A small number of students do also drop out of the degree programme – many of these transferring to a sister programme with less computer programming content (some of these students do not complete the second semester programming module, and accordingly their results do not appear on Figure 5). These students commonly cite difficulties with, or dislike of, computer programming as being a factor in their decision. It is hard to estimate the impact that learning two programming languages at the one time has had on such decisions, but the fear is that such exposure is a contributory factor leading to the loss of students from the programme.

From this study, clear evidence of harm from the use of DarkBASIC has not been found. However, there remain clues that there is most likely a negative impact on some students, particularly in their gaining a comfortable familiarity with control structures and the idea of modularity. We feel that we can improve on DarkBASIC, perhaps not as a raw games programming engine, but certainly by providing an environment that will have less likelihood of promoting poor programming habits, or of putting students off programming altogether.

ALTERNATIVES TO DARKBASIC

The most obvious alternative would be to use BlitzPlus, another version of BASIC written to give hobby programmers easy access to DirectX. BlitzPlus has a number of apparent advantages – for example, the unusual treatment of keywords and reserved words of DarkBASIC is avoided and structs are supported (although with an unusual notation). However, arrays still automatically have global visibility and functions do not support call by reference or pointer.

Using BlitzPlus students would still be required to learn two different programming languages simultaneously. Perhaps a better alternative would be to use a custom

```
void gameMain()
{
    db_image background;    // A double-buffered image object
    // Establish a backdrop for the game on the client area...
    background = new db_image("gameboard.bmp");
    // Make this the form's background...
    this->backgroundimage = background;

    // Create an object to move across the game board...
    sprite tank1 = new sprite("tank.bmp");
    int x = 250;
    int y = 200;

    do {
        input = controller.getMove()
        if (input == UP) y = y - 1;
        if (input == DOWN) y = y + 1;
        if (input == LEFT) x = x - 1;
        if (input == RIGHT) x = x + 1;
        if (input == ESCAPE) break;
        background.beginDraw(); // This starts a draw sequence to the
                                // 'background' bitmap
        background.drawSprite(tank1,x,y);
        background.endDraw(); // This swaps back- with foreground.
    } while (true);
}
```

Figure 5. In development: A framework to allow students to develop 2D games in C++ as simply as is possible in DarkBASIC.

games creation framework to allow beginning students to use C/C++ in creating their first games. A framework to simplify the development of DirectX applications was previously presented by (Slater 2003). While this removed the need for students to develop DirectX applications from scratch, by giving them an initial 'empty' project, it still required that they deal with many of the complexities of DirectX and C++.

We propose instead the creation of a framework which allows students to write simple 2D games in C/C++, using commands similar to those found in DarkBASIC and BlitzPlus. The program shown in Figure 2 could be re-written using this framework as shown in Figure 5. The code fragment, which reflects our current thinking on the framework, is notable for a number of reasons:

- Although the framework will be a windows .NET form-based application, it will provide access to user-input (from a 'controller' class object) that avoids the use of event-handlers, which are standard mechanisms for catching user input in this arena. We aim to simplify the handling of input by presenting the framework as a fully modal environment.
- Typically graphics are drawn in the paint event in a Windows application. This approach has many advantages, but can be confusing to new programmers since the graphical state must be made accessible to a number of event handlers. The simple 'draw-now' command structure gets over an initial hurdle when considering the control structure of the application.
- A number of additional operations are involved compared to the DarkBASIC version. Each additional statement has a purpose that is clear and easy to explain in a games programming context (establishing a background, creating a foreground character, initiating and completing a screen update), and the additions are likely to lead to a better understanding of the processes.
- gameMain() will be an entry point that encompasses the entire game processing – it will be entered after initialisation and clean-up will be performed after exit.
- Since gameMain() is built around a simple loop, this could provide opportunity to demonstrate (and give easy access to) the notion of threads in processing – although working with threads is unlikely to be required of students in their first semester.

Thus we hope to introduce a library designed to promote educational objectives to replace a language that works in spite of its educational failings.

CONCLUSIONS

Our experience shows that DarkBASIC can be used successfully to introduce many concepts related to the development of interactive graphical applications to first year undergraduate students, while allowing them the opportunity to design and create their own games – even for students with no prior experience of programming. For students to complete their first playable games within three months of starting university is something of an achievement.

Despite this, and without clear and unambiguous evidence, there remains some belief that using

DarkBASIC in the first semester is having a negative effect on progress for some students. Accordingly, a replacement is sought, although it is felt that existing alternative packages (such as BlitzPlus) are unlikely to resolve all of the problematic issues.

Accordingly, we propose to develop a C++ .NET library giving students DarkBASIC-like access to basic 2D graphics operations. The key advantage is that students will be learning fundamentally the same programming language as is used in other first year modules (and by most of the games industry) – C++. With only the one programming language syntax to be learned, all students will need to learn for this module will be the framework specific features. Follow on modules will abandon the framework, but the experience in programming in C++ (and core knowledge gained in interactive graphical programming) should be transferable.

ACKNOWLEDGEMENTS

Daniel Livingstone would like to thank The Carnegie Trust for the Universities of Scotland for supporting his current research.

REFERENCES

- Dijkstra, Edsger W. 1968. Go To Statement Considered Harmful. *Communications of the ACM* 11 (3):147-148.
- . 1975. How do we tell truths that might hurt? *Sigplan Notices* 17 (5).
- Harbour, Jonathan S., and Joshua R. Smith. 2003. *Beginner's guide to DarkBASIC game programming*. Edited by A. LaMothe, *Game Development*. Cincinnati: Premier Press.
- McMonnies, Alistair. 2004. *Object-Oriented Programming in Visual Basic .NET*. Harlow, England: Pearson Addison Wesley.
- Slater, Stuart. 2003. Rapid Application Development of Games for Undergraduate and Postgraduate Projects Using DirectX. Paper read at Game-On 2003, 4th International Conference on Intelligent Games and Simulation, at London.
- Wikipedia. 2004. *BASIC programming language* 2004 [cited 30th July 2004]. Available from http://en.wikipedia.org/wiki/BASIC_programming_language.

POWER AND PERIL OF TEACHING GAME PROGRAMMING

Maic Masuch and Lennart Nacke
Games Research Group
Department of Simulation and Graphics
University of Magdeburg
39106 Magdeburg, Germany
E-Mail: masuch@isg.cs.uni-magdeburg.de

ABSTRACT

This paper reviews the teaching of computer games as an academic subject for students at university level. The benefits, challenges and problems of teaching game design and game programming are investigated and discussed. We present an overview of experiences with two different approaches to game programming, one in Magdeburg, Germany, and one in Dunedin, New Zealand. A comparison and a list of practical advices for lecturers of similar courses concludes the paper.

KEYWORDS

Computer Games, Games Education, Game Design, Game Programming, Computer Science Curriculum, Video Games

1 INTRODUCTION

Computer games are on their way becoming established in academic research and teaching (Squire 2003). In past years games research was strongly concerned with possible negative effects of computer games on social behaviour of young gamers – as it has been discussed before for many other new media like theatre, movies, television, video and likewise. Recently, the academic focus shifted towards a broader and more balanced argument on games and education, taking also the positive effects of gaming into account.

Simultaneous to this discourse the number of art schools and multimedia colleges offering game design courses increased. These schools, however, concentrate more on practical issues, e.g. teaching primarily the use of tools for game development (Yu 2002). Recently universities have turned their attention to games and quite a number of researchers have focused on game related technologies. Teaching game development is becoming an increasingly popular subject at universities (Masuch and Freudenberg 2003, McCallum et al. 2004). Students and teaching staff are highly motivated to study a subject which many of them cultivate as a hobby anyway. Also casual games and even non-gamers find computer games fascinating as games receive increasing attention and reputation in public.

We present our experiences with two fundamentally different approaches to computer game education and discuss what worked out well and what did not. This text elucidates the benefits and challenges this subject offers for academia.

2 TEACHING GAMES

Teaching the skills necessary for the development of games is interesting and difficult at the same time. We found out that, somehow, games are difficult to approach from an academic point of view.

On the one hand this is due to the youth of this research area. Even with the large number of publications that address games very recently, there are no long research and teaching experiences to call on for reference. On the other hand, games research has not yet established itself as an accepted academic discipline among other sciences. Often officials and representatives of more traditional Universities and computer science departments look at this field with reservation and worry about their reputation since games arise directly from the entertainment industry. Furthermore, the deep immersion in games makes it hard to reflect on them at an academic level as most students will be gamers themselves (either casual or hardcore). Nevertheless, universities that teach and research on electronic games, are starting to get a benefit out of it.

Thus, the interest in methods and techniques that investigate game development as a teaching subject has grown, especially at computer science faculties. However, sometimes the term “computer games” is generalised when only talking about specific aspects of programming (e.g. a number of books on DirectX give the impression that they just added a game topic as sales pitch). Somehow similar, for many computer science courses games – as field of application – can provide a very good motivational source. However, in our approach to teaching computer games we aim at teaching the whole process, not just one specialized part or topic separate from the others. We claim that courses on game development should cover more than just real-time 3D-Graphics. This misuse of terms harms the reputation of computer game development as an

academic subject and makes it difficult to establish an academic critique, as there are no agreed terms yet.

2.1 Motivation

Game programming and game development are interdisciplinary fields that require educators to go beyond standard topics of computer science. Although we focus on teaching game development at computer science faculties, we think it is rewarding to reflect on a humanistic level what your software is capable of doing. Here, games give an excellent bridge for engineers towards reflections on the impacts of their software. Many game developers state that prior to the point of programming a game, relevant factors like game play, player reward and popular culture have to be considered to develop a software product that has some chance to compete on the mass market.

The integration of literally speaking “multimedia” makes games also a very interesting subject because you have to consider compression and media programming issues as you try to deliver the best looking results in the smallest package possible.

Before stepping further into the discussion it seems necessary to clarify the difference between game design and game programming. As these terms are often misused and mixed-up, even by people who implicitly know the very difference, there is some confusion about these two concepts.

2.2 Game Design vs. Game Programming

We understand *Game Design* as a creative process of developing a game concept, its core elements and structure. This includes art work and story as well as considerations of playability and game balance. The actual realization of the game, however, can be restricted to a paper document. Strictly speaking computer game design does not require any game programming.

The term *Game Programming* refers to the process of actually coding a game. It consists of making a project plan for the realization of a game idea and actually programming the game elements. This might include the programming of a game engine or “only” implementing the game assets and interaction in an existing game engine.

We use the term *Game Development* to cover Game Design, Game Programming and all other production related topics.

2.3 General Approach to Teaching Game Development

When teaching game development at an university level there are different approaches at different institutions around the world, depending on their curriculum.

Some are going for a very narrow approach in topics, while others are trying to cover all aspects relevant to game programming in one module instead of a degree program. Of course, many educational facilities that offer a degree in “Game Design” just respond to the demand of their customers – the students. However, from the point of view of the games industry a too general “Game Design” degree programs remain questionable, as those breed all-round-talents instead of specialized team players with deep expertise in their specific domain.

Often it is necessary to make a trade-off between a more technical or more artistic (if not humanistic) side of teaching. This depends on the type of curriculum of the university. Obviously, the main focus of a game course depends on the main teaching direction i.e. programming or design. A good way to address this disjunction is to built interdisciplinary teams. In general, game design does not necessarily require programming skills and can be taught at a beginner level (1st to 2nd year) and game programming as specialization later at an advanced level.

The IGDA suggested a helpful curriculum for game development education (IGDA 2003). It is broad enough for every educational facility to shape their individual “Games”-curriculum – may it be a liberal arts college or a computer science department of a technical university. In the next section we introduce our experiences with teaching games as course modules at two universities.

3 EXPERIENCES IN TEACHING GAMES

We will introduce two different approaches to game development for computer science students at two universities (University of Magdeburg, Germany, and University of Otago, New Zealand) in different countries and social contexts.

3.1 Otto-von-Guericke University of Magdeburg

At the Otto-von-Guericke University of Magdeburg the first game programming course was established in 1999 (Masuch 2004). Almost six years of teaching experience later it is interesting to take a look at the development from then to now.

The first try at teaching computer games was to give a very broad and “all inclusive” overview of topics that are used in the world of electronic gaming. So the start was one broad scope course on computer games that featured a programming assignment in parallel with the lectures. Of course most of the students were very eager to create complex games. Most of them took on a task that was far too large for them to finish within one semester given that they also had other courses running parallel to this one. Some of the projects were never

finished – lecturers and students came out of this with lots of experience but also some frustration.

The second run had a new concept. We split the computer games course into two courses Computer Games I and Computer Games II. The first course pursued a broader approach with the techniques on the one side and social/humanistic topics on the other. The second course had its focus on algorithms and tools. Parallel to this second course the students would do group-based project work on a game. We had a large team working on a single project. The most capable and motivated students finally drove it to a successful end. The projects were much more successful and the outcome motivated students as well as lecturers.

The third year included a third course besides CGI and CGII that concentrated solely on game design. Prior to the course, we conducted an informal evaluation of several game design tools and finally used an open source 3D scripting environment. Unfortunately, the team collaboration that worked fine in the evaluation turned out as a catastrophe in real project work. While in the evaluation we managed to integrate our test-scenes into one game-world, the system became unpredictable and unstable dealing with more complex scenes and interactions. It was hardly possible to integrate the work of team members in a single project file and students were more occupied (and frustrated) fighting the tool then to get on with their game. No team managed to conclude their work in a playable prototype. Now, in our sixth year we can look back on developing successful and working curriculum on computer games for computer science and computational visualistics students.

The process of breaking “computer games” down into its subtopics went very well, so it was used again to subdivide topics a little more and focus on specific aspects of computer games in different courses. The several parts of computer game development covered in separate lectures were: “Techniques and Reflections”, “Algorithms and Tools”, “Modelling and Animation”, “Game Design” and other game related lectures, respectively seminars like “educational gaming”, “games and simulation”, etc.

So instead of trying to put all game-relevant topics into one course, we now provide a range of diverse courses that cover many of the topics that are part of game development.

3.2 University of Otago, Dunedin

A different approach for a computer game course was co-developed at the University of Otago, New Zealand (McCallum 2004). The primary approach of this course was to introduce students to the concepts of game design in the format of an “all-day” summer school. This is an intense course block format running for six weeks

during the summer holidays (four days a week with lectures, labs and tutorials and one day just with guest lectures), the course – a completely different approach from the one in Magdeburg – is not mandatory, which assured beforehand that only interested and motivated students would participate.

Since this was the first and only course of this kind at the University of Otago, the intention of the lecturers was to give a broad overview of the topics covered in the game development process. Thus the course used the IGDA curriculum framework (IGDA 2003) as an outline of topics that were suitable for students at an university level of education. Although the framework is meant as a guideline for an entire degree program on game design, the ambitious plan was to cover all of the topics (on a very shallow basis). The course was co-created by the design and the computer science department.

As the staff at the university had never taught computer game design before it was difficult to evaluate the IGDA framework and tailoring it to the intended audience and timeframe (six weeks). Like the first course at the Otto-von-Guericke University this was an “all inclusive” approach, which tried to cover everything: broad and shallow. Nevertheless the course also included project-based group work within a small timeframe towards a predefined goal – a finished game. One of the unfortunate decisions in the planning was to let the students choose which tools to use for achieving this goal. Thus, the teaching staff had to consider different problems with different tools and even different languages since the students employed techniques from DirectX programming to 3D Game Studio with its embedded C-Script language.

The overall focus on a final product maintained high levels of motivation throughout the course for the students. Unfortunately the time that they needed for programming their game prototypes left little time to reflect enough on social issues and non-programming topics involved in the process of computer game design.

One of the outstanding things in this course was that the students had to give feedback at least every week but were encouraged to keep staff informed every day. Since most of the staff worked in the labs together with the students this concept worked out well at the price of the workload of the staff exceeding even that of the students and spare time shrunk to zero.

A bit surprising (but also rewarding) was the fact that all students finally managed to present some kind of game prototype at the end of the course. Not all of them were of equal quality but most of them were at least somewhat playable.

The range of topics were another problem that required expertise from different departments. Thus, even though the focus was on graphics and programming for games, a large number of lecturers from other departments were engaged to add value from their research areas, which among others included game business, media studies, character animation.

In the end, this overview format was a good way of showing to the computer science faculty how broad the scope of computer games is and how many areas of research can be involved. After the course was finished, the University of Otago also agreed to host New Zealand's first Game Developers Conference. This event has attracted more attention to game development in this small country and lead to the establishment of many interdisciplinary projects.

4 POWER AND PERIL

Based on our experiences we conclude that the teaching of game development can offer a wide variety of benefits. However, it also contains some inherent problems that have to be addressed. In the following sections 4.1 and 4.2 we summarize the common experiences from both approaches. Arguably, taking discussions with lecturers of similar game courses into respect, we think that – to a certain extend – these can be generalized. As the benefits of teaching game design and programming are more obvious than the drawbacks, we will discuss the perils in more detail.

4.1 Potential and Benefits

The use of games as an academic topic can have tremendous benefits. Among others, we discovered the following favourable aspects:

Motivation – Students motivation to learn is the key to successful teaching. Games turned out to be a motivational source second to none.

Versality and interdisciplinarity – Games are complex multimedia projects and thus ideally suited to drawing together various aspects of computer science. Further, game development inherently requires input from many disciplines: programming, design, sound, business, and many more. The ability of students to talk to and work with people from other disciplines is an important one in the game production process, both inside and outside of University.

Self-contained projects – The outcome of a course can be a finished product (if the student project is completed). Knowing that they are working towards a product is intensely motivating for students as they can possibly include it in their portfolio and use it for future job applications.

Practical experience – practical experience is given to the students by using industry-like tools and atmosphere, which makes the educational training highly relevant for industry training. After finishing the project work at university this experience can also be used by entrepreneurial students to start their own spin-off company (which has actually happened).

Teamwork – students learn how to work in teams by a group-project based teaching curriculum. In general teamwork is one of the best things to go along with classic teaching, not only because it is a necessary soft skill required by the market but also because it brings social aspects in an otherwise theoretic environment. Having students work together in groups and assigning them to different areas of computer game development allows a high level of social interaction and personal development. On the other hand, this demands, that the educator faces the challenge of grading students individually rather than as a team. This does not need to become a problem as meetings with students can be arranged on a weekly basis, or a feedback system can be established so that it is completely transparent who does which kind of work in the group.

4.2 Challenges And Problems

Teaching game courses also comprises some drawbacks that need to be considered:

Claim vs. capability – Students are influenced by existing media. For most students a game is simply a larger software project; while trying to compete with popular game titles they underestimate the amount of work and the capabilities needed for completion by an order of several magnitudes.

Interdisciplinary teamwork – Since games cover a broad range of topics, the audience attracted to a computer games course will eventually attract people from different subjects. This leads to students with very diverse interests and capabilities, which makes it difficult to form homogeneous teams. Therefore it is important that every member identifies himself with the task assigned and the team. Teams should choose a project lead and organize team events.

Missing focus and complexity of projects – In a single course the lecturer has to focus on either game design, game programming or reflections on games not both – or worse – all three. Trying to cover more than one of these topics in a single course, can lead to a watering down of the content of any one topic. Regarding practical exercises, it is futile trying to implement all aspects of a game (or even a game engine). Again, focus should be set on specific aspects of a game.

No experience and use of wrong tools – In comparison to other Computer Science subjects there is little experience in teaching and programming games to

build upon. Many course topics come from other areas of expertise (e.g. psychology, pedagogy, drama theory, law, design, fine arts etc.) so that lecturers have to get assistance from other departments (good for interdisciplinary cooperation, bad for the time table) or have to research all relevant information on their own. The workload for this is much higher than in classical sciences. Further, it is futile to build a game and a game engine from scratch. Finding adequate tools that are suitable for teaching the specific course aspects (even focusing on game design or game programming) can be difficult. Our recommendation is to begin software evaluation at least one semester early, preferably with a well documented prototype.

5 CONCLUSION

Summing all up, we think that computer games have an enormous potential in teaching and research. Our experiences in the last years with teaching game programming to computer science students showed to us that games are exceptionally well in

- *motivating* students,
- teach them *interdisciplinary* teamwork and
- give them hands-on experience in *multimedia integration*.

After having analysed these two different approaches to the teaching of computer games and the difficulties that they were facing, some final conclusions can be drawn. The approach of the University of Otago to teach a complex subject like this in summer school requires full-time availability of students and lecturers. It is a suitable course format to emulate the “crunch period” similar to the game industry. The amount of work necessary for preparing the lectures and the course was underestimated. More time would have been helpful. In addition, the University of Otago did only have limited resources to provide such a course.

The prerequisites in Magdeburg now are much better, but it all started very similar to the approach made at the University of Otago. The games research group is now an established part of the faculty and therefore can allocate more resources on teaching and research. A number of lectures have been developed, each with a focus on a different aspects of computer games. Both directions are unwearingly supported by students while undergoing a practical software lab or a lab internship. Our start-up phase is over now, but there is still much room for improvement.

Finally, we would like to share some practical hands-on advice for people who are planning similar courses:

- Focus on either game programming or game design, not both.

- Ensure equal levels of programming experience and capabilities.
- Students should not start to learn tools whilst in production – use crash courses in advance of the lecture for preparation.
- Do not allow different platforms, engines, languages or tools – students should help each other.

And finally – if you only take one advice from this paper – then the most important one: Have fun!

ACKNOWLEDGEMENTS

We would like to thank all our students who made teaching such a rewarding effort and our student advisors: Bert Vehmeier, Ralf Armin Böttcher and Jan Fietz for their help in Magdeburg. We would also like to thank Simon McCallum for his personal devotion and optimism of starting a game programming course from scratch in New Zealand.

REFERENCES

- IGDA Education Committee, 2003. IGDA Curriculum Framework - The Study of Games and Game Development, Version 2.3 beta. February 25, 2003.
- McCallum, S., 2004. COSC360 - Computer Game Design at the University of Otago, Dunedin, New Zealand: <http://cosc360.otago.ac.nz>
- McCallum, S.; Makie, J. and Nacke, L., 2004. Creating a Computer Game Design Course. In *Proceedings of the New Zealand Game Developers Conference, (NZGDC)*.
- Masuch, M., 2004. Computer Games Research Group at the University of Magdeburg: <http://www.isg.cs.uni-magdeburg.de/games>
- Masuch, M. and Freudenberg, B., 2002: Teaching 3D Computer Game Programming. In Ralf Dörner, Christian Geiger, Paul Grimm, and Michael Haller, editors, *Workshop Proceedings Production Process of 3D Computer Graphics Applications -- Structures, Roles, and Tools*, Aachen, Shaker.
- Squire, K., 2003: Video games in education. *International Journal of Intelligent Simulations and Gaming* (2) 1.
- Yu, C., 2002: Developing a Game Programming Course For Computer Science Majors In a Liberal Arts College. In *First International Conference on Information Technology & Applications, (ICITA)*.

APPLYING COMPUTER GAMES CONCEPTS TO TEACH DATABASE ANALYSIS AND DESIGN

Thomas M Connolly, Evelyn McLellan, Mark Stansfield, Judith Ramsay,

School of Computing
University of Paisley
High St, Paisley, PA1 2BE
Scotland
E-mail: thomas.connolly@paisley.ac.uk

John Sutherland

School of Social Sciences, Moray House School of
University of Paisley Education
University of Edinburgh
Edinburgh, EH8 8AQ
Scotland

KEYWORDS

Computer simulation games, database analysis and design, constructivist learning environments, problem-based learning, motivation, engagement, neuroplasticity.

ABSTRACT

The study of database systems is typically core in undergraduate and postgraduate programmes related to computer science and information systems. However, one part of this curriculum that many learners have difficulty with is database analysis and design, an area that is critical to the development of modern information systems. This paper reflects on these difficulties and explores a range of pedagogical issues surrounding the development of a problem-based learning environment based on interactive visualisation and computer games to help overcome these difficulties and help the learner develop the skills necessary to understand and perform database analysis and design effectively. The paper proposes a set of guiding principles upon which the proposed learning environment is being developed.

INTRODUCTION

The database is now the underlying framework of the information system and has fundamentally changed the way many organisations and individuals work. This is reflected within tertiary education where databases form a core area of study in undergraduate and postgraduate programmes related to computer science and information systems, and typically at least an elective on other data-intensive programmes (ACM/IEEE 2001; EUCIP 2003). The core studies are commonly based on the relational data model, SQL (the *de facto* language for relational DBMSs), data modelling and relational database design. This curriculum supports industry needs where the relational DBMS is the dominant data-processing software currently in use, with estimated new licence sales of between US\$6 billion and US\$10 billion per year (Connolly and Begg 2004).

The core relational theory is a mature and established area in relation to other parts of the computing curriculum. However, one part of this curriculum that many learners have difficulty with is database analysis and design. For the purposes of this paper we use the term 'database analysis and design' to encompass requirements analysis, conceptual database design (including ER modelling), logical database design (including mapping to the relational model and validating the model using normalisation) and physical database design.

In this paper, we explore a range of teaching techniques that supplement traditional teaching methods with more non-traditional methods based on interactive visualisation and computer games to help overcome these difficulties and help the learner develop the skills necessary to understand and perform database analysis and design effectively.

Problems with Teaching Database Analysis and Design

Mohtashami and Scher (2000) note that pedagogical strategies for teaching database analysis and design traditionally follow a similar modality to that of other technical programmes in computing science or information systems. A significant amount of technical knowledge must be imparted with the lecturer becoming a 'sage on stage' and the students passive listeners. While students tend to cope well with basic concepts and practical components of the core database curriculum, for example, understanding the properties of the relational data model, the basics of SQL and using a relational DBMS such as Microsoft Office Access or Oracle, one area that many students find difficult is the abstract and complex domain of database analysis and design. A comparable problem has been identified with object-oriented analysis and design, which is also highly abstract (for example, Yazici *et al.* 2001). This is borne out by a recent European survey that found that the primary skill that organisations considered to be lacking in both new IT graduate recruits and current IT staff was database design (database tuning and database administration were second and third, respectively) (Connolly and Laiho 2004).

While databases have become so essential to organisations, Kroenke (2003) states "unfortunately, increased popularity has not meant increased competency. Many students (as well as professionals) have been deceived by the simplicity of creating small databases using products such as Microsoft Access. With this background, they believe they know sufficient database technology to create databases that have more complicated structure and greater processing complexity. The result is often a mess: databases are hard to use, barely meet system requirements, and are difficult to redesign." To undertake database analysis and design effectively for an even moderately complex system, a student requires (among others) the skills to:

- work in a project team and apply appropriate fact-finding techniques to elicit requirements from the client (both 'soft', people-oriented skills);
- conceptualise a design from a set of requirements ('soft', analytical skills),
- map a conceptual design to a logical/physical design ('hard', technical skills);

- reflect and review intermediate designs, particularly where information complexity is present (a combination of 'soft' and 'hard' skills).

Students often have considerable difficulty comprehending implementation-independent issues and analysing problems where there is no single, simple, well-known or correct solution. They have difficulty handling the ambiguity and vagueness that can arise during database analysis. Students can also display an inability to translate classroom examples to other domains with analogous scenarios, betraying a lack of analytical problem-solving skills. For the students these problems can lead to confusion, a lack of self-confidence and a lack of motivation to continue. In teaching database methods we are trying, as Postman and Weingartner (1971) state, "to help students to become more efficient problem solvers", to avoid "the right answer that only serves to terminate further thought" and to reach a position where the student "must learn to depend on himself as a thinker". However, as noted by Eaglestone and Baptista Nunes (2004) there are a number of other factors that have impacted database teaching that are outwith the students' control, such as increasing student numbers and the development of online delivery with students who are geographical dispersed and have diverse backgrounds.

In this paper we explore the use of interactive visualisation and computer games to provide a learning environment to supplement traditional methods of teaching database analysis and design. We have chosen to examine such an environment for several reasons. First, the younger generation have grown up in a technologically sophisticated environment, which has led to changes in their experiences, attitudes and expectations. It therefore makes sense to investigate and exploit those aspects of the technologies the modern learner has been exposed to, such as computer games, with a view to identifying those aspects that might be transferable *in pedagogical terms*, into teaching. Second, there is empirical evidence that games can be an effective tool for enhancing learning and understanding of complex subject matter (Ricci *et al.* 1996; Cordova and Lepper 1996). Third, educationalists are interested in the intensity of involvement between instructional strategies, motivational processes and learning outcomes. It would be highly desirable to harness the appropriate properties of computer games that enhance learning and improve student performance.

This paper is structured into four further sections. The next section discusses the pedagogical basis for developing a problem-based learning environment based on visualisation and computer games to teach database analysis and design, leading in the section thereafter to a set of principles for the design of the proposed learning environment. The penultimate section discusses the on-going design of this environment. The final section provides some concluding remarks and directions for future research.

PREVIOUS RESEARCH

In this section we examine previous research related to the use of computer games in education, specifically the research suggesting that current learners (the 'digital natives') have different experiences, attitudes and expectations and therefore require a more appropriate pedagogical model of teaching; the importance of motivation and flow to learning; the use of simulation and games in education; constructivism as a pedagogical approach to learning and the appropriateness of problem-based learning for our purposes.

Are Current Learners Different?

Prensky (2001) advocates that learning should be engaging although usually it is not for the current younger generation going through Higher Education (whom he describes as *digital natives*). He argues that learning today is unengaging compared to all the alternatives like television, computer games and even work. Digital natives have grown up in a technologically sophisticated environment; an environment populated by home computers, the Internet, graphic-rich movies, multi-player Internet gaming, Nintendo GameBoys™, XBoxes™, DVD players, mobile phones, interactive television and iPods™, which has led to a change in their experiences, attitudes and expectations. Contrast this with the pre-digital generation (today's teachers) who grew up largely with the passive technologies of books, television and radio and who were "educated in the styles of the past" (*op cit*). Papert, quoted in Prensky (*op cit*) states that "The reason most kids don't like school is not that the work is too hard, but that it is utterly boring." Schank (1997) also believes that education has not changed and that teacher-centred models are still dominant in adult learning, despite research clearly demonstrating that "these methods no longer seem to be working" (Biggs 1999). Schank supports the use of simulations and games and presents the premise that "when learning isn't fun, it's not learning" (additionally suggesting that real learning does not occur until the learner fails).

It is important that educational techniques keep step with social developments such as widespread technology adoption, especially as it is now understood that the human brain exhibits *neuroplasticity* (Kolb *et al.* 2001; Whishaw *et al.* 2003). That is, it dynamically reorganises itself to adapt to novel experiences (neuroplasticity is also the mechanism that allows the brain to compensate for injury by enabling existing neural pathways to take over functions that were previously managed by the affected area(s)). It is likely that the exposure of today's learners to novel stimuli such as computer games early on in life has led to their developing different neural pathways to those of the children of thirty years ago. Connelly (2004) has observed that "physiological capacity, learning styles, and neuroplasticity all combine to provide a foundation on which educators can build more powerful teaching techniques". The development of more powerful, yet cost effective, teaching techniques represents one potential solution to the progression and retention challenges facing educational institutions. This problem is especially prevalent in respect of first and second year undergraduate students.

With respect to education, two particular types of physiological change take place in the brain during learning (Drubach 2000), namely changes in the behaviour of synaptic gaps between neurons and an increase in the number of synapses available for communication between neurons. Simply said, the brain of the modern learner is configured significantly differently to that of its 1970's counterpart. It therefore makes sense to investigate and exploit those aspects of the technologies the modern learner has been exposed to, such as computer games, with a view to identifying those aspects that might be transferable *in pedagogical terms*, into teaching applications.

Motivation

Motivation is a key concept in many theories of learning. Katzeff (2000) stresses motivation is a critical factor for instructional design and for learning to occur the learner must be motivated to learn. In self-determination theory, Deci and Ryan (1985) distinguish between different types of motivation based on the various reasons or goals that give rise to an action. The most basic distinction is between intrinsic and extrinsic motivation. Intrinsic motivation refers to doing something because it is inherently interesting or enjoyable while extrinsic motivation refers to doing something because it leads to a separable outcome (such as a verbal reward like praise or a tangible reward like money).

Malone and Lepper (1987) present a theoretical framework of intrinsic motivation in the design of educational computer games. They postulate that intrinsic motivation is created by four individual factors: challenge, fantasy, curiosity and control and three interpersonal factors: cooperation, competition and recognition:

- *Challenge*: A player must be able to vary the difficulty of the game, and there should be multiple goals for winning the game. There should also be sufficient randomness in the action and constant feedback about performance.
- *Fantasy*: The player should feel involved with the characters in the game and the gaming environment.
- *Curiosity*: The activity should offer sensory stimulation and enough novelty to want to continue playing the game. Senge (1990) believes that people are inherently curious, creative and seek challenges that relate to what they value.
- *Control*: The player should feel in control over the activity in the game, be able to make choices and to witness the effects of such choices. When the choices are genuinely unclear, the learner should be able to gather information to make an informed choice.
- *Cooperation*: The player should feel satisfaction by helping others achieve their goals.
- *Competition*: The player should feel satisfaction by comparing their performance favourably to that of others.
- *Recognition*: The player should feel satisfaction when others recognize and appreciate their accomplishments.

Interestingly these factors also describe what makes a good game, irrespective of its educational qualities. While intrinsic motivation is highly desirable, many of the activities in which learners engage in is directly influenced by extrinsic rather than intrinsic motivation (Csikszentmihalyi and Nakamura 1989). Unfortunately evidence suggests that extrinsic motivators may lead to merely short-range activity while actually reducing long-range interest in a topic while with intrinsic motivators learners tend to persist longer, work harder, actively apply strategies and retain key information more consistently (Guthrie *et al.* 1996). Thus, extrinsic motivators must be supported by intrinsic motivators, otherwise the result is likely to be a reduction in the very behaviour we want to promote. One of the most serious problems that research has pointed out during the past two decades is that extrinsic motivation when used alone is likely to have precisely the opposite impact that we want it to have on learner achievement (Lepper and Hodell 1989).

In determining what makes a particular situation or activity intrinsically motivating to an individual, the concept

of *flow* is often mentioned. Csikszentmihalyi (1990) used the term 'optimal experience' or 'flow' to refer to feelings that his subjects reported to have experienced while involved in leisure and work activities. He suggested that individuals achieve a sense of flow when the challenge level matches their skill level and to reach this state of optimal experience: "there must be a goal in a symbolic domain; there have to be rules, a goal, and a way of obtaining feedback. One must be able to concentrate and interact with the opportunities at a level commensurate with one's skills" (Csikszentmihalyi *op cit*). Tasks that are too difficult raise anxiety and tasks that are too easy contribute to boredom; both situations decrease motivation toward learning. Flow is characterized by a feeling of being in control, by highly focused attention, and by an adequate match between challenge and skill, resulting in an intense state of joy and emotional involvement in an activity for its own sake.

The conditions likely to induce the flow state are (adapted from Malone (1981)):

1. *Challenge*: The activity should be structured so that learners can increase or decrease the level of challenges being faced in order to match exactly their skills with the requirements for action. Additionally, the activity should have a broad range of challenges, and possibly several qualitatively different ranges of challenge, so that learners may obtain increasingly complex information about different aspects of themselves.
2. *Control*: It should be easy to isolate the activity, at least at the perceptual level, from other stimuli (external or internal) that might interfere with involvement in it.
3. *Performance criteria*: There should be clear performance criteria; learners should be able to evaluate how well or how poorly they are doing at any time.
4. *Feedback*: The activity should provide concrete feedback so that learners can tell how well they are meeting the performance criteria.

Simulation and Games

Crookall and Saunders (1989) view a *simulation* as a representation of some real-world system that can also take on some aspects of reality for participants and users. According to Garris *et al.* (2002) the key features of simulations are that they represent real-world systems, contain rules and strategies that allow flexible and variable simulation activity to evolve, and the cost of error for participants is low, protecting them from the more severe consequences of mistakes.

In contrast, Caillois (1961) defines a *game* as an activity that is voluntary and enjoyable, separate from the real world, uncertain, unproductive (in that the activity does not produce any goods of external value) and governed by rules. Crookall *et al.* (1987) believe that a game is not intended to represent any real-world system, rather it is a 'real' system in its own right. Like simulations, games also contain rules and strategies but the costs of losing are generally only consequential within the game world. Finally, Kriz (2003) defines a *simulation game* as: "the simulation of the effects of decisions made by actors assuming roles that are interrelated with a system of rules and with explicit references to resources that realistically symbolize the existing infrastructure and available resources".

Prensky (*op cit*) defines the key characteristics of (simulation) games as: rules, goals and objectives, outcomes and feedback, conflict (and/or competition, challenge, opposition), interaction, and representation of story.

Simulation games in education are a research area that may be conceptualised as the intersection of learning concepts, information technology and user interfaces (Chiou 1993). In this conceptualization, learning concepts serve as the foundation for simulation games to ensure that technology does not become the dominant factor. However, Squires *et al.* (2003) note that a key problem in the development of educational games is balancing how much of the game is a game and how much of the game is learning. If there is insufficient enticing game play included or if there is insufficient appropriate academic content the result will be a failure. This raises an important issue and one that we have addressed by involving many participants in the design, including previous database students, teachers and gamers.

Constructivism and Learning Environments

Constructivist and sociocultural theory

While traditional education has been guided by the paradigm of didactic instruction, which views the learner as passively receiving information, there is now an emphasis on *constructivism* as a philosophical, epistemological and pedagogical approach. Constructivism focuses on knowledge construction, not knowledge reproduction. Collins linked constructivism and technology suggesting that the new technology required adopting this approach to pedagogy (1991). Vygotsky's *sociocultural theory* of learning emphasises that human intelligence originates in our culture. Individual cognitive gain occurs first in interaction with other people and in the next phase within the individual (Forman and McPhail 1993). Thus, constructivism stresses active engagement for effective learning to take place while socioculturalism stresses the importance of interpersonal communication. These two models are not mutually exclusive but merely focus upon different aspects of the learning process. For most educationalists constructivism provides more scope for realising possible learning benefits of using computer technology.

According to Gance (2002) the main pedagogical components commonly associated with these models are:

- A cognitively engaged learner who actively seeks to explore her environment for new information.
- A pedagogy that often includes a hands-on, dialogic interaction with the learning environment. For example, actually designing a database is preferred to simply being told how to design a database.
- A pedagogy that often requires a learning context that creates a problem-solving situation that is realistic.
- An environment that typically includes a social component often interpreted as actual interaction with other learners and with mentors in the actual context of learning.

Constructivist learning environments and problem-based learning

Many researchers have expressed their hope that constructivism will lead to better educational software and better learning (for example, Brown *et al.* 1989; Jonassen 1994). They emphasise the need for open-ended exploratory authentic learning environments in which learners can develop personally meaningful and transferable knowledge and understanding. This has led to the development of

guidelines and criteria for the development of a *constructivist learning environment* (CLE) - "a place where learners may work together and support each other as they use a variety of tools and information resources in their guided pursuit of learning goals and problem-solving activities" (Wilson 1996). See, for example, Cunningham *et al.* 1993; Grabinger and Dunlap 1995; Savery and Duffy 1995; Ben-Ari 2001; Gance 2002. The principles proposed by Cunningham *et al.* (1993) and Savery and Duffy (1995) are frequently cited within the literature and are summarised in Table 1. defines a constructivist learning environment as. Simulation games are constructivist learning environments in which learners are invited to actively solve problems (Leemkuil *et al.* 2003).

| Cunningham <i>et al.</i> (1993) | Savery and Duffy (1995) |
|---|--|
| Provide experience of the knowledge construction process. | |
| a) Provide experience in and appreciation for multiple perspectives. b) Encourage the use of multiple modes of representation. | Encourage testing ideas against alternative views and alternate contexts. |
| Embed learning in realistic and relevant contexts. | a) Anchor all learning activities in a larger task. b) Design an authentic task. c) Design the task and the learning environment to reflect the complexity of the environment they should be able to function in at the end of learning. |
| Encourage ownership and voice in the learning process. | a) Give the learner ownership of the process used to develop the solution. b) Support the learner in developing ownership for the overall problem or task. |
| Embed learning in social experience. | |
| Encourage self-awareness of the knowledge construction process. | Provide opportunity for and support the reflection on both the content learned and the learning process. |
| | Design the learning environment to support and challenge the learner's thinking. |

Table 1 Comparison of principles of Cunningham *et al.* (1993) and Savery and Duffy (1995).

The *problem-based learning* (PBL) model encompasses these principles. PBL started out in the 1960s in medical education in the USA and Canada where groups of students were presented with a problem in the form of a patient with particular symptoms (Biggs *op cit*). The students' task is to diagnose the patient's condition and be able to justify the diagnosis and recommend treatment. In diagnosing the condition, the students have to discuss the symptoms, generate hypotheses based on whatever knowledge and experience they have and identify learning issues. At the end of each session, the students reflect verbally on their current hypotheses and each student assumes responsibility for investigating one of more of the identified learning issues through self-directed learning.

The teacher (facilitator) is available for consultation and plays a significant role in modelling the metacognitive thinking associated with the problem-solving process. This reflects a *cognitive apprenticeship* environment (Collins *et al.* 1990) with *coaching* and *scaffolding* (e.g. offering hints, reminders and feedback) provided to support the learner in developing metacognitive skills. As these skills develop, the scaffolding is gradually removed. The intention is to force learners to assume as much of the task on their own, as soon as possible. The cognitive apprenticeship model also advocates:

- *modelling*, which involves an expert (the teacher) performing a task so that the learner can observe and build a conceptual model of the processes required to accomplish it;
- *articulation* (either verbal as mentioned above or written);
- *reflection*, to enable learners “to compare their own problem-solving processes with those of an expert, another learner, and ultimately, an internal cognitive model of expertise” (Collins *et al. op cit*);
- *exploration*, to push learners into a mode of problem-solving on their own.

Savery and Duffy (*op cit*) comment that PBL should stimulate, and therefore engage the learner in, the problem-solving behaviour that the practicing professional would employ. The PBL approach is now used across a range of subject disciplines.

A similar concept to articulation that has been cited as an important element of simulation games is *debriefing* (Crookall 1995; Lederman and Kato 1995). Games and simulations differ in that simulations include elements of the real world whereas games are “separate from the real world”. Debriefing is an essential element of any simulation game because it links what has been experienced during the simulation with learning. Debriefing provides the opportunity for learners to consolidate their experience and assess the value of the knowledge they have obtained in terms of its theoretical and practical application to situations that exist in reality (Kriz *op cit*).

GUIDING PRINCIPLES FOR THE LEARNING ENVIRONMENT

We illustrate the influences for the problem-based learning environment that we are developing to teach database analysis and design based on the above research in Figure 1, depicting the relationships between the game, the teacher, learners and the environment. In addition, we put forward our own principles for the learning environment as follows:

1. Start with an authentic problem grounded in professional practice. This problem should be both realistic and sufficiently complex to develop analytical and problem-solving skills.
2. Encourage learners to take responsibility (ownership) for learning and to be aware of the knowledge construction process.
3. Allow learners to develop their own process to reach a solution.
4. Provide learners with the opportunity to experience and appreciate other perspectives (this may come about as part of the next principle).
5. Provide opportunities for interaction and collaboration, either learner-learner, learner-teacher or learner-system.

6. Ensure that the learning environment motivates, engages and challenges the learner.
7. Provide feedback mechanisms to enable learners to be fully aware of their progress.
8. Provide support mechanisms for learners using coaching and scaffolding.
9. Be flexible to support different learning styles.
10. Provide opportunities for reflection, self-evaluation, articulation and debriefing.
11. Provide an integrated assessment.

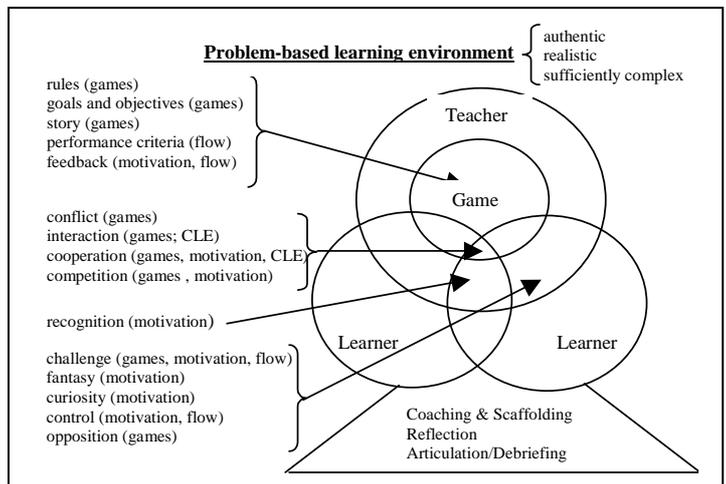


Figure 1 Influences for the problem-based learning environment.

DESIGNING THE LEARNING ENVIRONMENT

Video case studies have been used for several years within computing-related undergraduate and postgraduate modules in the School of Computing at the University of Paisley. The videos were developed by the University to provide students with real-world organisational problem scenarios such as organisational change within a library, a marina and a veterinary practice through which they could develop and apply a range of different skills and concepts. Although the use of the videos was found to be engaging, their main drawback was that students could not interact with the characters and scenarios presented to them, they could only view them in a sequential, linear and passive fashion. In addition, for several years the School has been developing online learning materials for various undergraduate and postgraduate modules/programmes as well as interactive visualisations that enhance these materials. In particular, material has been developed for the undergraduate ‘Introduction to Database Systems’ and the postgraduate ‘Fundamentals of Database Systems’ modules.

To develop the students’ learning experience further in these two modules, it was decided to develop an educational simulation game around the video case studies and use the interactive visualisations and online learning materials as a form of *digital scaffolding* in an attempt to increase student interactivity and engagement with the problem scenarios being presented. For example, students would be able to interact with the characters by asking different types of preset questions, which would influence the outcome of the problem situation. The simulation game provides the opportunity for students to learn and apply a range of relevant skills and

techniques relating to database analysis and design within a more interactive, engaging and stimulating environment more akin to the real-world setting that students may find themselves in industry.

The simulation game is part of a wider learning environment as shown in Figure 2. The following three main components form the learning environment:

- The online learning units/topics (entry level 1) introduce the concepts to be explored; these units are structured in a hierarchical manner allowing students to ‘drill down’ to obtain further details. Topics are hyperlinked to allow non-sequential browsing.
- The visualisations (entry level 2) enhance learning by providing animated walkthroughs of specific examples (e.g. construction of an ER diagram or the process of normalisation).
- The simulation game (entry level 3) provides a real-world simulated environment within which to apply skills and techniques.

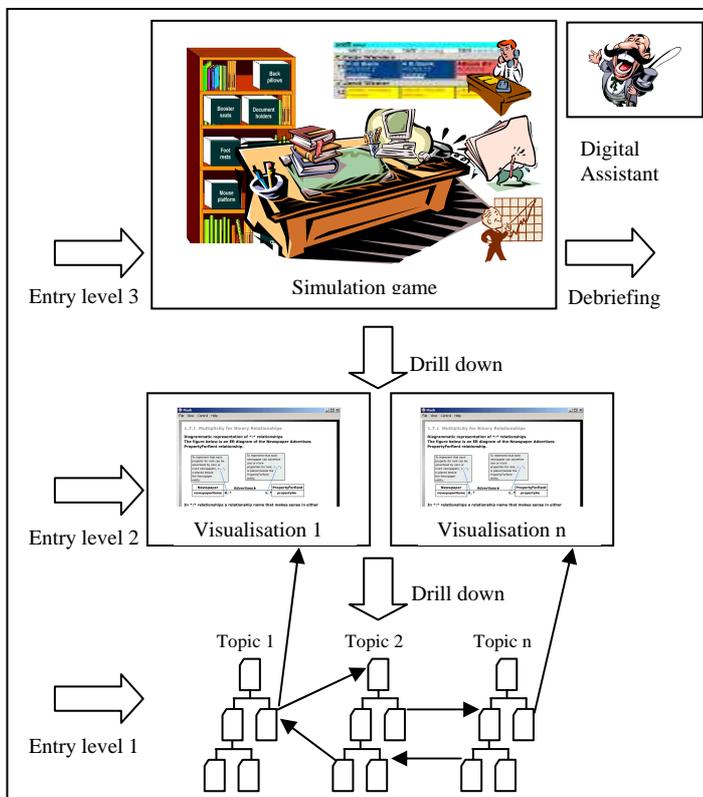


Figure 2 The learning environment.

Participatory design has been adopted for all three entry levels in which academics, students and practitioners have all been involved at various stages of the project. The simulation game is part of a natural evolution of the learning environment in which all three elements work together. This means that students working through the simulation game can pause at appropriate points and ‘drill down’ via the Digital Assistant to the interactive visualisations or to individual topics.

Using the theoretical framework offered by Malone and Lepper, the simulation game is based on the following principles:

Challenge

The students are provided with a real-world organisational problem scenario comprising a mixture of challenging soft/people-oriented issues as well as more structured/technical related issues. Conceptually the game is divided into a number of different levels each comprising a set of tasks. At each level, the students are given feedback on their performance. There is an element of time in the game so it is important that the students manage their tasks carefully in terms of how much time they can devote to each one. Once the time limit has been reached, students will not be able to continue on the task unless they allocate time away from other tasks yet to be completed. Spending too much time on one task might seriously hinder them in completing another task later on. The system provides feedback at various points as well as progress and life indicators.

Fantasy

Within the game, the organisational setting is viewed from the perspective of the different stakeholders within the situation (e.g. senior management, middle management, administrative staff etc.). This form of role-play helps the students imagine themselves in the real-world setting since they need to interact with all stakeholders in order to learn about all aspects of the problem situation. The students are constrained on how much time they can spend interviewing a certain character. To add realism, unexpected events may arise such as the character being unexpectedly called away or not turning up to the interview. This helps the students experience some of the frustration that often happens when working within a real-world organizational problem situation.

Curiosity

Being a real-world simulation means that the situation could change with events affecting the behaviour and views of the stakeholders within the problem situation, therefore, the simulation can stimulate the sensory curiosity of the students. In addition, it is only through the application of fact-finding techniques that students learn about the problem situation from the perspective of the stakeholders and gain the necessary data to develop the database. As shown in Figure 2 (entry level 3), this element of curiosity is shown by the use of relevant icons and representations in the simulation game that mirror real-world objects and situations. During the knowledge elicitation phase, students can click on various objects to find out more about the situation. For example, by clicking on a person the students see a video clip of that person giving their views on a certain matter or observe the person undertaking a relevant task. Clicking on a report yields part of its contents, which could be relevant to the problem situation. In the same way a chart might show important trends and figures. This gets the students into the habit of having to search for relevant information in the same way as they might if they were conducting the task in the real-world. Certain sources might prove fruitless or provide conflicting information, so the students have to be very careful and thorough in their knowledge elicitation.

Control

There is a clear cause-and-effect relationship between the students’ actions and the outcome of the simulation game. The success and soundness of the database design is based on how well the students have completed the previous levels, in particular areas such as knowledge elicitation. Therefore, it is

vital that students ensure that their analysis and design is robust. As shown in Figure 2, the students can pause the simulation game at certain points to 'drill down' to lower entry levels to access materials that might help them complete a task.

Cooperation

Typically the simulation game is played by students working in groups of between 2 and 4. Therefore, in order to adequately complete all of the tasks it is important that each group work effectively together. Each group must decide how the various tasks and levels are to be tackled, with different group members having responsibility for different tasks. As a result effective cooperation within the group is vital to complete the work to a satisfactory standard. Effective cooperation within a group is a vital real-world skill that the students will have to demonstrate when they graduate and seek employment in the field.

Competition

There is a strong element of competition between groups in undertaking the simulation game. Competition provides impetus for the groups to complete the tasks and levels to a high standard and within the time limits. In addition, since the simulation game forms part of a formal assessment, there is pressure on the students to perform well both within each group and between groups to achieve a good mark that contributes to their overall module grade. This element of competition and pressure on the students does add an element of realism, reflecting the industrial environment where there is a high degree of pressure to undertake tasks within time, cost and quality targets.

To add more competitive play, various activities will be scored using both qualitative and quantitative criteria. For example, attempting to validate a conceptual model without having identified all appropriate data will result in marks being deducted.

Recognition

Recognition is achieved through the scoring system, which maintains a highest running score, and the feedback mechanism, which allows students to compare their performance against others.

SUMMARY AND FUTURE DIRECTIONS

This paper has discussed some of the pedagogical issues underpinning the development of a constructivist learning environment using problem-based learning and a simulation game and interactive visualisations to help teach database analysis and design. Levels 1 and 2 of the environment have been developed and initial findings are very positive with performance improvements over traditional teaching methods of between 10%-12% (Connolly *et al.* 2003). Levels 3, the simulation game, is in the development stage but we hope to have this ready for initial use in semester 2 of session 2004/5. Future work will include evaluation of the appropriateness of the environment for different groups of students (undergraduate and postgraduate, full-time and part-time). Additional work will be necessary to consider the applicability of this environment to online students, particularly the collaboration element of the activities.

REFERENCES

- ACM/IEEE. 2001. "ACM/IEEE Computing Curricula". Dec. 15 2001. <http://www.computer.org/education/cc2001/> (23 July 2004).
- Becta. 2004. "Computer Games in Education Project." <http://www.becta.org.uk/research/research.cfm?section=1&id=2846> (28 June 2004).
- Ben-Ari, M. 2001. "Constructivism in Computer Science Education." *Journal of Computers in Mathematics and Science Teaching*, 20(1): 45-73.
- Biggs, J. 1999. "Teaching for quality learning at university". *Society for Research into Higher Education*.
- Brown, J.S., A. Collins and P. Duguid. 1989. "Situated cognition and the culture of learning." *Educational Researcher*, Jan-Feb: 32-42.
- Caillois, R. 1961. *Man, play, and games*. Free Press, New York.
- Chiou, G. 1993. "Some potential areas of research and development in the space of computer-based learning." *Educational Technology*, 33(8): 19-23.
- Collins, A. 1991. "The role of computer technology in restructuring schools." *Phi Delta Kappan*, 73: 28-36.
- Collins, A., J.S. Brown and S.E. Newman. 1990. "Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics." In *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, L.B. Resnick ed. Lawrence Erlbaum, Hillsdale, N.J.
- Connelly, J.O. 2004. "Beyond the technology." <http://www.sosu.edu/cidt/briefs/beyond.htm> (23 July 2004)
- Connolly, T.M. and M. Laiho. 2004. "Database Technology Professional European Survey Analysis." <http://www.dbtechnet.org> (23 July 2004).
- Connolly, T.M. and C.E. Begg. 2004. *Database Systems: A Practical Approach to Design, Implementation, and Management*, 4th ed. Addison Wesley, London.
- Connolly, T.M., E. McLellan and M.H. Stansfield. 2003. "Online Learning: Better Results, Better Learning". *9th Annual Conference on Asynchronous Learning Networks*, Orlando, Florida, November 2003.
- Cordova, D.I. and M.R. Lepper. 1996. "Intrinsic motivation and the process of learning: beneficial effects of contextualization, personalization, and choice." *Journal of Educational Psychology*, no 88: 715-730.
- Crookall, D. 1995. "A guide to the literature on simulation/gaming." In *Simulation and gaming across disciplines and cultures: ISAGA at a watershed*, D. Crookall and K. Arai, eds. Sage, Thousand Oaks, CA.
- Crookall, D., R.L. Oxford and D. Saunders. 1987. "Towards a reconceptualization of simulation: From representation to reality." *Simulation/Games for Learning*, no. 17: 147-171.
- Crookall, D. and D. Saunders. 1989. "Towards an integration of communication and simulation." In *Communication and simulation: from two fields to one them*, D. Crookall and D. Saunders, eds. Multilingual Matters, Clevedon, UK.
- Csikszentmihalyi, M. 1990. *Flow: The Psychology of Optimal Experience*. Harper and Row, New York.
- Csikszentmihalyi, M. and J. Nakamura. 1989. "The dynamics of intrinsic motivation: a study of adolescents." In *Research*

- on motivation in education, R. Ames and C. Ames, eds. vol. 3. Academic Press, San Diego, CA.
- Cunningham, D., T.M. Duffy and R. Knuth. 1993. "Textbooks of the future." In *Hypertext: A psychological perspective*, C. McKnight, ed. Ellis Horwood Publishing, London, 19-49.
- Deci, E.L. and R.M. Ryan. 1985. *Intrinsic motivation and self-determination in human behavior*. Plenum, New York.
- Drubach, D. 2000. *The Brain Explained*. Prentice-Hall, Upper Saddle River, N.J.
- Eaglestone, B. and J.M. Baptista Nunes. 2004. "Pragmatics and practicalities of teaching and learning in the quicksand of database syllabuses." *Journal of Innovations in Teaching and Learning for Information and Computer Sciences*.
- EUCIP. 2003. *EUCIP (European Certification of Informatics Professionals) Core Syllabus*. March 2003. http://www.eucip.com/DownloadFiles/Core_Syllabus_March_2003.pdf (23 July 2004).
- Forman, E. and J. McPhail. 1993. "Vygotskian perspectives on children's collaborative problem-solving activities." In *Contexts for learning. Sociocultural dynamics in children's development*, E.A. Forman, N. Minick and C. Addison Stone, eds. Oxford University Press, Oxford.
- Gance, S. 2002. "Are constructivism and computer-based learning environments incompatible?" *Journal of the Association for History and Computing*, V(1), May 2002.
- Garris, R., R. Ahlers and J.E. Driskell. 2002. "Games, motivation, and learning: A research and practice model." *Simulation and Gaming*, 33(4): 441-467.
- Grabinger, R.S. and J.C. Dunlap. 1995. "Rich environments for active learning: a definition." *Association for Learning Technology Journal*, 3(2): 5-34.
- Guthrie, J.T., P. Van Meter, A. McCann, A. Wigfield, L. Bennett, C. Poundstone, M.E. Rice, F. Fabisch, B. Hunt and A. Mitchell. 1996. "Growth of literacy engagement: changes in motivations and strategies during concept-oriented reading instruction." *Reading Research Quarterly*, 31, 306-332.
- Jonassen, D.H. 1994. "Thinking technology: Toward a constructivist design model." *Educational Technology*, 34(3): 34-37.
- Katzeff, C. 2000. "The design of interactive media for learners in an organisational setting – the state of the art." In *Proceedings for NordiCHI 2000* (Stockholm, Oct. 23-25).
- Kolb, B., R. Gibb and C.L.R. Gonzalez. 2001. "Cortical injury and neuroplasticity during brain development." In *Toward a theory of Neuroplasticity*, C.A. Shaw and J.C. McEchern, eds. Taylor and Francis, New York.
- Kriz, W.C. 2003. "Creating effective learning environments and learning organizations through gaming simulation design." *Simulation and Gaming*, 34(4): 495-511.
- Kroenke, D. 2003. *Database Processing: Fundamentals, Design, and Implementation*. Prentice Hall.
- Lederman, L.C. and F. Kato. 1995. "Debriefing the debriefing process." In *Simulation and gaming across disciplines and cultures: ISAGA at a watershed*, D. Crookall and K. Arai, eds. Sage, Thousand Oaks, CA.
- Leemkuil, H.T., de Jong, R. de Hoog and N. Christoph. 2003. "KM QUEST: A collaborative Internet-based simulation game." *Simulation and Gaming*, 34(1): 89-111.
- Lepper, M.R. and M. Hodell. 1989. "Intrinsic motivation in the classroom." In *Research on motivation in education*, R. Ames and C. Ames, eds., vol. 3. Academic Press, San Diego, CA.
- Malone, T.W. 1981. "Towards a theory of intrinsically motivating instruction." *Cognitive Science*, no 4: 333-369.
- Malone, T.W. and M.R. Lepper. 1987. "Making Learning fun: A Taxonomy of intrinsic motivations for learning." In *Aptitude, learning and instruction. Volume 3: Conative and affective process analysis*. Lawrence Erlbaum, Hillsdale, N.J, 223-253.
- Mohtashami, M. and J.M. Scher. 2000. "Application of Bloom's Cognitive Domain Taxonomy to Database Design." In *Proceedings of ISECON (Information Systems Educators Conference)* (Philadelphia, Nov. 9-12).
- Postman, N. and C. Weingartner. 1971. *Teaching as a Subversive Activity*. Penguin, London.
- Prensky, M. 2001. *Digital game based learning*. McGraw-Hill.
- Ricci, K., E. Salas and J.A. Cannon-Bowers. 1996. "Do computer-based games facilitate knowledge acquisition and retention?" *Military Psychology*, 8(4): 295-307.
- Savery, J.R. and T.M. Duffy. 1995. "Problem based learning: An instructional model and its constructivist framework." *Educational Technology*, 35: 31-38.
- Schank, R. 1997. *Virtual learning: a revolutionary approach to building a highly skilled workforce*. McGraw-Hill
- Senge, P. 1990. *The fifth discipline: The Art and Practice of The Learning Organization*, Doubleday, New York.
- Squires, K., H. Jenkins and the Games-To-Teach Team. 2003. Designing educational games: design principles from the Games-To-Teach project." *Educational Technology*, September-October.
- Wilson, B. 1996. *Constructivist learning environments: Case studies in instructional design*. Educational Technology Publications, New Jersey.
- Whishaw, K., B. Kolb, and I. Whishaw. 2003. *Fundamentals of Human Neuropsychology (5th ed.)*. Freeman.
- Yazici, S., T. Boyle and T. Khan. 2001. "Towards a multimedia learning environment for object oriented design." In *Proceedings of the 2st Annual Conference of the LTSN Centre for Information and Computer Sciences I* (London, Aug. 28-30).

USING BUGS AND VIRUSES TO TEACH ARTIFICIAL INTELLIGENCE

Peter Cowling¹, Richard Fennell², Robert Hogg²,
Gavin King³, Paul Rhodes¹, Nick Sephton²

¹University of Bradford, Dept. of Computing, Bradford BD7 1DP.
E-mail: [P.I.Cowling|P.C.Rhodes}@Bradford.ac.uk](mailto:{P.I.Cowling|P.C.Rhodes}@Bradford.ac.uk)

²Black Marble Ltd., Business & Innovation Centre, Angel Way, Listerhills,
Bradford BD7 1BX. E-mail: [Richard|Robert|NSephton}@blackmarble.co.uk](mailto:{Richard|Robert|NSephton}@blackmarble.co.uk)

³Microsoft Ltd., Microsoft Campus, Thames Valley Park,
Reading RG6 1WG. E-mail: gking@microsoft.com

KEYWORDS

Games, Teaching, AI, Virus, Terrarium Academic.

ABSTRACT

In this paper we wish to add further to the growing body of evidence that games provide a good platform for teaching advanced computing concepts. We describe our experiences of teaching a group of 45 final year undergraduate students at the University of Bradford in the module “Artificial Intelligence for Games”. The students were required to create artificial intelligence for a turn-based board game called Virus, and for a real time artificial life environment, Terrarium Academic. Easy-to-use APIs and sophisticated graphical clients were developed for each game and a server application allowed student AI to compete in a round-the-clock tournament. We describe our experiences in developing and using the Virus game and Terrarium Academic, focussing particularly on what lessons we have learned that could usefully be applied in other teaching that employs games as a medium of instruction. Student feedback, and the quality of student work, suggests that the module was highly successful in its aims, especially in teaching advanced artificial intelligence and programming ideas to students, some of whom were rather nervous about their ability in both of these areas before starting the module.

INTRODUCTION

A significant number of students pursue a course of study in computing motivated in part by their interest in computer games. In spite of a financial structure that keeps rewards for game developers lower than for other computer industries, and a distinct lack of job security, many students desire to join the computer games industry on graduation. Even where students do not wish to become games developers, they are often keen computer (and non-computer) game players. Teachers can tap into this enthusiasm for games in order to motivate students to learn related topics. Since computer games are often at the “bleeding edge” of computer software, using a wide variety of techniques, a very broad range of advanced software topics can be motivated by considering computer games. Moreover, games are starting to be perceived by the research community as a useful test bed for investigating novel AI approaches (Schaeffer 2001) (Laird and van Lent 2001).

Here we will discuss our observations and experience of using games to motivate students in the design and implementation of artificial intelligence techniques. A cohort of 45 students studied for the final year undergraduate “Artificial Intelligence for Games” module at the University of Bradford. This is a fairly large cohort for an entirely optional module, which lends some evidence to the idea that students were interested in the games content of the module. Within the first couple of weeks of the module there was a small increase in the number of students, with students drifting from other options. Content was delivered using only a small number of traditional slides and a lot of discussion of real AI, using the Virus and Terrarium academic platforms to write examples, often in real time within a lecture. The modest size of the cohort allowed for significant and useful question and answer sessions in lectures and this was used extensively. A large number of links and documents were placed in a module web page which was very regularly updated. The module was assessed entirely by two pieces of coursework, described in detail in the following sections. It is arguable that students are also motivated to study courses which do not involve written examinations. However, it was made clear to all students prior to signing up for the option, that the coursework would involve writing two significant pieces of software, and learning a new language, C# (pronounced C-sharp), and development platform, Visual Studio.NET (VS.NET) in order to do this. It was also made clear that all coursework would be marked by a single person, and that all cases of plagiarism would be detected and severely punished. While it is clearly impossible to catch all cases of plagiarism, we have not detected any instances. Student feedback suggests that this up-front attitude to describing the skills required and the ground rules was appreciated by students. Some students were concerned that they did not have the requisite programming and AI skills. These students were simply given further information on the course content and coursework deliverables. Several of the students whose previous marks showed weakness in programming and AI opted to take the module, with most of them doing reasonably well and some achieving excellent results.

The paper is structured as follows. In the next two sections we will discuss the Virus and Terrarium Academic games and the APIs that were developed together with the server which allowed each student’s AI to compete against the other student’s AI and AI written by the instructors. These sections consider students’ feedback and lessons

learned which may be of use to other instructors using games as a teaching medium. Finally we give conclusions which highlight the effectiveness and drawbacks of using games to teach AI and programming topics.

THE VIRUS GAME

Rules

The Virus game is a two player game of perfect information, played on an 8x8 board (as in chess, draughts or Othello). There are two players, black and white, who play alternately, starting with black. Initially the board is set up as in Fig. 1.

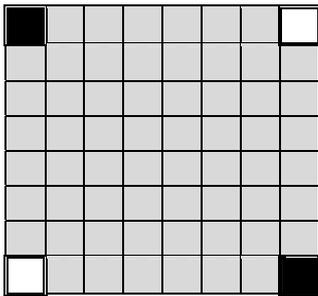


Fig.1. The Virus game starting position.

Two types of moves are available at each turn. The first type of play involves *growing* a new piece adjacent to an existing piece of the same colour (Fig. 2). The second type of play involves *moving* a piece to another square a distance exactly 2 squares away (via an empty square) (Fig. 3). Note that squares are considered adjacent if they adjoin or if they share a corner. In either case all opposing pieces next to the moved piece change colour.

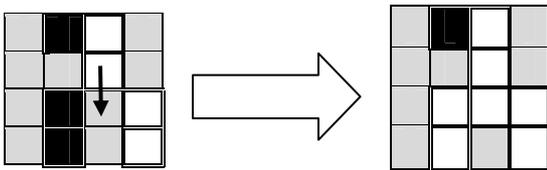


Fig. 2. The white piece grows.

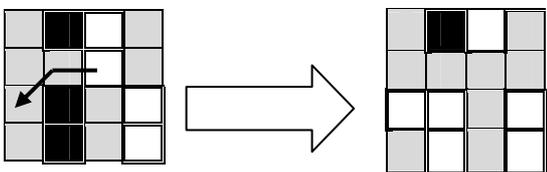


Fig. 3. The white piece moves

Play continues until neither player can move, or until one player has no pieces left, when the player with the most pieces wins the game.

Teaching Using the Virus Game

The Virus game was chosen as a game with very simple rules, but having complex game play. Whilst it is hard to quantify that complexity, at least in terms of branching factor the Virus game appears to have similar (although

arguable higher) complexity than draughts (Schaeffer et al. 1992) and Othello (Buro 2002). The advantage of using the Virus game over better known games is that the opportunity for plagiarism by finding software on the web is eliminated. Indeed checking that the web resources available for the Virus game were limited was an important preparatory step. It also meant that module web pages were able to actively link to selected sources of information on Othello, draughts, chess and other games.

Students were required to write a board evaluation function for use in a search of the move tree. Software for searching the move tree (minimax with alphabeta pruning (Knuth and Moore 1975)) was written, and students had to write a function which returns a value for each position which represents the probability of the black player winning in the given position. Since one requirement was that games between student AI had to run very quickly, the tree was searched to three moves ahead, requiring evaluation of around thirty thousand positions per move in the middle game. Initial concerns from the academics in the team over the speed of C# proved unfounded, and very fast tree searches of a speed approaching that of native C++ code were observed. It proved effective to present this situation to students as “the software looks forward three moves, your evaluation function looks forward the rest of the game”. However, it became apparent after a couple of weeks that there was confusion amongst the weaker students as to whether the evaluation function itself makes a move. A clearer explanation that the evaluation function is but a part of the AI player was needed sooner and more repeatedly. However, all students overcame this misunderstanding within a couple of weeks. To allow a reasonable amount of time per board evaluation and at the same time ensure that games ran quickly on the server, a total of 10 seconds was allowed for each AI player to complete all its moves in a game. Any AI player that used more time than this automatically lost. The aim was to teach students techniques of minimax tree search, positional analysis, pattern matching and evolutionary board tuning by using binary board representations and binary operators such as AND, OR and NOT.

Students implemented their board evaluation function by writing a class MyPlayer which inherited from our VirusPlayer class, and overriding the GetEstimatedScore() function of that class. Although students had little or no familiarity with Visual Studio .NET or C#, since they had previously used the Java programming language, they were all able to handle this and understand the very basic debugging features of VS.NET following a couple of hours of instruction in labs with Black Marble and Prof Cowling. Although several students reported some concern that they would have to learn a new computer language and environment as well as the AI techniques on this module, at the end of the module the students expressed that they had found it useful to familiarise themselves with C# and VS.NET during their degree course, and that it had not slowed them too considerably.

Sponsorship from Microsoft and development by Black Marble made it possible to write a sophisticated client for the Virus game (Fig. 4), which allowed students to learn strategies by playing against each other and against simple AI players. It also allowed students' AI players to play against each other. There were some issues of trust here,

which were presented early on to students, due to the possibility of decompilation for C#. However, once students were made aware of this possibility, they only traded AI players with other trusted colleagues and this does not appear to have caused instances of plagiarism.

Black Marble also wrote a server application for the Virus game, which allowed a competition to run continuously between student AI players, maintaining a league table that permitted students to continuously monitor the relative performance of their AI player against those of the other students. This proved to be a huge motivator for the majority of students, especially since the algorithm for determining which two AI players should play next was designed so that any AI uploaded to the server would get a game almost immediately. This resulted in many students working very hard into the early hours to improve their AI players. Student feedback was very positive on this aspect of the module. AI players were uploaded by FTP. The students could complete a “nickname” field for each AI player, which proved very useful in further raising the interest level of students. The competition for the silliest nickname was almost as fierce as that for the best AI! In addition to student players, the two instructors on the module wrote AI players of varying strengths (some designed to be at the bottom, middle and top of the league). After the first couple of weeks of familiarisation with the module and the environment a benchmark player was added, which simply counted (number of black pieces) – (number of white pieces) and reported the piece advantage of the black player as the position evaluation. Students were given plenty of notice that beating this player would be a requirement for a “pass” in terms of AI performance.

Assessment was 30% for performance on the server and 70% for a short written design document and a printout

of source code. A one page document was prepared which described precisely what was required, without giving a restrictive marking scheme. The emphasis of that document was on originality first, league performance second and good software engineering and documentation third.

Over 600,000 games of the Virus game were played on the server over a period of four weeks. It was very satisfying to see the benchmark player start high in the league, descend to mid table after a couple of weeks, and finish rock bottom at the time of coursework submission.

Since software development took place in parallel with teaching, students had to tolerate some uncertainty as development encountered snags. Giving students a complete picture of what was going in when snags occurred arguably increased the enthusiasm and involvement of the student body, as they felt involved in a real development process to create a complex piece of software.

Student coursework demonstrated a lot of originality and coding ideas. One of the pleasant surprises was the emergence of a student “add-on” community, with students creating a test server for local testing of AI players, optimising code for low level bit counting functions, developing a GUI for generating board shapes using the binary representation of the client software, and recreating functions which were in parts of the code not made available to students. Student approaches demonstrated a good deal of originality and insight and included: using very fast bit operators and low level code optimisation for speed, notions of safety, mobility, limited look-ahead, pattern-matching, flood-fill, degrees of ownership, positional scores, parameter weighting of features, opening book analysis, biological ideas such as surface area over volume, enclosure, game stage analysis, randomness and terminal node checking. This list cannot do full justice to the effectiveness and originality of student approaches, but should give a flavour. All students submitted a coursework on time and weaker as well as strong students were motivated to do well.

TERRARIUM ACADEMIC

Artificial Life has been an interesting area for the investigation of Artificial Intelligence for some time now, inspired by the work of Conway (Berlekamp et al. 1982), Reynolds (Reynolds 1987) and others. Terrarium is an artificial life environment created by Microsoft developers (<http://www.gotdotnet.com/Terrarium>), where the aim is to give software agents (which represent bugs and plants) intelligence and effective real time behaviour to survive and multiply in the environment created by other bugs and plants. The software has an interface that renders the Terrarium world observable using rather beautiful animated graphics. Initially it was a spare time project for the developers, but more recently it has been embraced by Microsoft as a .NET sample application, and indeed Microsoft ran a highly successful international Terrarium competition. Terrarium is itself an interesting tool to use directly as a tool for teaching AI in real time environments, but since the Terrarium environment is extremely rich and complex, and significant levels of software engineering skill are needed to understand the API and create effective and original Terrarium bugs, it was considered too difficult for

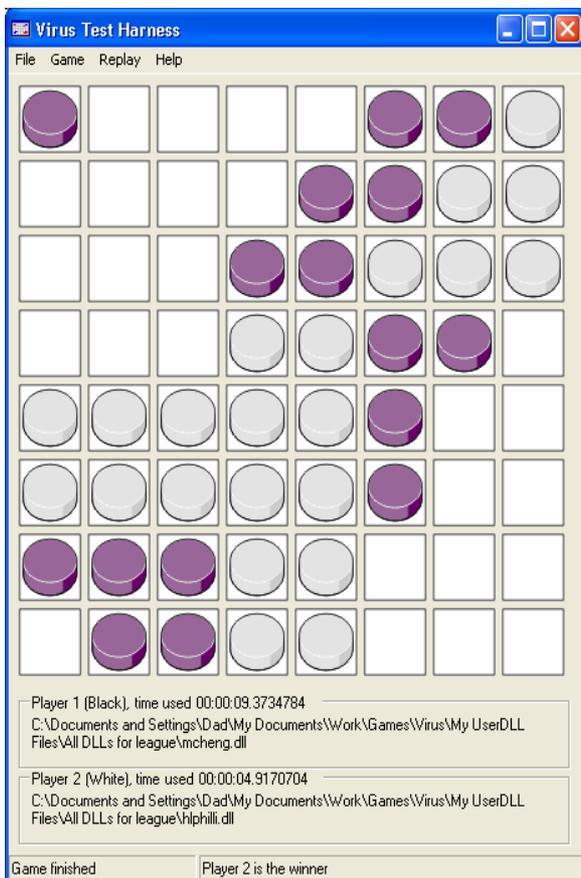


Fig.4. The Virus game client

one half of a module given to final year undergraduates in Computing. Moreover, the source code for significant numbers of Terrarium bugs is available on the Internet.

Simplification was needed to the API, and also the measure of success for a bug needed to be simpler, so that observing a bug would more readily give information as to how that bug could be enhanced. To that end, Terrarium Academic was created, by extending the existing Terrarium beta source code (Fig. 5). Terrarium Academic replaced reproduction and the notion of survival of the fittest with a points system that rewarded collection of points from fixed goals (which the creatures had to find) and used rocks to



Fig.5. The Terrarium Academic graphical client

define a variety of maze-like environments for the bugs so that path finding would be an essential activity for a successful bug (Mika and Charla 2002). Terrain building and loading were added as functions. The dual energy and injury system of Terrarium was replaced by a single system for injury. Finally, plants were regarded as a way to cure injury and were made into indestructible, fixed objects (again to encourage mapping and pathfinding). Each bug in Terrarium Academic is called once per “tick” so that each student bug was required simply to override the DoAction() function of a basic bug object. Students did not have to handle other events. The aim was to teach multi-level planning, rule-based and state-based system ideas, pathfinding, opponent modelling and real-time software agent ideas.

Again, a server application was developed by Black Marble, which functioned in much the same way as for the Virus game, as a Terrarium environment without a graphical representation, that recorded the results of each Terrarium run in an SQL database. An important difference is that each Terrarium game can take several minutes to run, so that students were unable to immediately see the effectiveness of a submitted bug. While students were certainly happy with this setup, many wished for the immediacy of the Virus game server, and this is an issue which it would be desirable to address by allowing bugs to be directly added to the Terrarium environment as they are uploaded to the server.

The course instructors wrote a variety of bugs for students to test against, since observation of their own bugs interacting with other bugs using the graphical Terrarium

Academic client proved to be the most effective way of refining bugs.

Students found the Terrarium API much harder to get to grips with than the corresponding API for the Virus game, in spite of our simplifications and instruction. Several approaches were used to aid their understanding, by far the most effective of which was providing students with full source code for four bugs (including one very highly functional bug with mapping and pathfinding capabilities). It was made clear to students that no credit would be given for reproducing the functionality in these bugs, and that they would have to write the action planning code in order to use the tools provided in the bugs given. All of the students used state based and rule based approaches to do this, with most agents having a sensible fixed plan, such as attack any adjacent bug, otherwise go to or search for a goal. Some of the more advanced bugs had more sophisticated strategies based upon a correct supposition that most of the other bugs would have a relatively simple fixed behaviour, using methods designed to beat common behaviour, such as hiding until most bug combat had finished, or searching the map for a secluded spot with a goal in it. All students again submitted coursework, which varied from reasonable to excellent. Student feedback suggested that although all students found this coursework harder than the Virus coursework, they also found it more rewarding due to the obvious link with AI in use in the video game industry.

CONCLUSION

Student feedback for the Virus game and Terrarium, and the quality of coursework submitted, strongly support the idea of games as a tool to teach advanced AI and programming concepts. Of course, some things could have been done better. Since the application was being written as teaching progressed, refinement of the API was limited, however students obtained excellent knowledge of the process of working with beta code, which changed over time, written by other developers. The time and effort required for this detracted to an extent from the time available for implementing AI. This balance will shift back with recent refinement to the APIs of both the Virus game and Terrarium Academic. Developing support software alongside lectures was time consuming and difficult, but it meant that instructors and developers were very well aware of student problems and issues. Two of the lectures were given over to unscripted question and answer sessions which were a great success given the common issues being faced by students and teachers.

Overall, students appreciated being asked to think creatively “outside the box” and rewarded for creativity, originality and effort in a well-structured coursework. All of the students’ feedback, with one notable exception, was very positive with many comments to the effect that some students found this the most interesting and rewarding experience of their University career. The one exceptional student who vocally protested about the non-traditional methods of the module in a question and answer session found that all of the other students argued very vigorously against this.

We would like to see games used as a medium for teaching AI and software engineering (and other topics)

more widely. To this end we aim to make available, free of charge, the clients and servers for the Virus game and Terrarium for teaching use in other Universities and teaching institutions. Please contact Professor Cowling (P.I.Cowling@bradford.ac.uk) for more information. There is the potential for inter-University competition or competition at national and international levels using refinements of the tools described in this paper. We are currently investigating effective ways to set up such competitions.

Peter Cowling has been fascinated by games and game AI since his father bought a Commodore Pet computer in the late 1970s. In addition to the work described in this paper, he is currently investigating generic approaches to game tree search in perfect information board games, artificial intelligence for deck construction and game play in collectible card games and learning approaches for real time video game agents.

He lives in the Yorkshire countryside with his wife and three sons.

REFERENCES

- E.R. Berlekamp, J.H. Conway, R.K. Guy. 1982. *Winning Ways for Your Mathematical Plays, Vol. II*. Academic Press.
- Knuth, D.E., Moore, R.W. 1975. "An analysis of alpha-beta pruning." *Artificial Intelligence* 6: 293-326.
- M. Buro. 2002. "Improving Heuristic Mini-Max Search by Supervised Learning." *Artificial Intelligence* 134, nos. 1-2: 85-99.
- D.E. Knuth, R.W. Moore. 1975. "An analysis of alpha-beta pruning." *Artificial Intelligence* 6: 293-326.
- J. E. Laird, M. van Lent. 2001. "Human-Level AI's Killer Application Interactive Computer Games." *AI Magazine*, Summer 2001.
- M. Mika, C. Charla. 2002. "Simple, Cheap Pathfinding." In *AI Game Programming Wisdom*, S. Rabin, ed. Charles River Media, Hingham, MA, USA.
- C.W. Reynolds. 1987. "Flocks, Herds, and Schools: A Distributed Behavioral Model." *Computer Graphics* 21, no. 4: 25-34.
- J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu and D. Szafron. 1992. "A World Championship Caliber Checkers Program." *Artificial Intelligence* 53, nos. 2-3: 273-290.
- J. Schaeffer. 2001. "A Gamut of Games." *AI Magazine*, Fall 2001.

BIOGRAPHY

Peter Cowling (<http://www.inf.brad.ac.uk/~picowlin>) is a Professor of Computing at the University of Bradford. He leads the Modelling Optimisation Scheduling And Intelligent Control (MOSAIC) research centre (<http://mosaic.ac>), whose main research interests lie in the investigation and development of new modelling, optimisation, control and decision support technologies, which bridge the gap between the theory and practice of optimisation, using artificial intelligence and operational research techniques. These approaches have been used in a variety of real world problems involving the planning and scheduling of personnel and production, in collaboration with commercial and industrial partners. He has sat on the programme committees of over 20 international scientific conferences in the past 5 years, and has published over 40 refereed journal and conference papers.

TEACHING CONSOLE GAMES PROGRAMMING WITH THE SONY PLAYSTATION2 LINUX KIT

Henry S Fortuna

School of Computing and Creative Technologies

University of Abertay Dundee

Bell Street, Dundee DD1 1HG,

Scotland, UK

E-mail: h.s.fortuna@abertay.ac.uk

KEYWORDS

Games, Console, Programming, PlayStation2.

ABSTRACT

In May 2002, Sony Computers Entertainment Europe released the PlayStation2 Linux Development Kit providing educational establishments with a means for teaching native code development on this current generation of console. Games console programming is fundamentally different from programming games on PC based platforms and is a valuable skill to be mastered by any graduate seeking employment as a programmer in the Computer Games Industry. In order to obtain optimal performance from a console, detailed knowledge and understanding of the system hardware is required. The PlayStation2 contains several processors which operate in parallel, and both the programming and synchronisation of these processors is essential. Program code is developed for the PlayStation2 Linux Kit using a range of generic and proprietary tools. The GNU C++ compiler is used to create high level game code, an assembler and/or inline assembly is used to create custom high-speed routines, a Vector Command Line preprocessor is used to develop low level code for the Vector Unit processors and the Graphics Synthesizer is configured to render to a television or monitor. These techniques and tools are introduced to students to provide them with a realistic insight into modern console game development.

INTRODUCTION

The PlayStation2 Linux kit (Linuxplay Web Site, 2004) is added to a standard PlayStation2 (PS2)

transforming it into a Linux workstation which can be used for many purposes including the development of native console game code. The kit consists of a hard disk drive, a keyboard and mouse, an Ethernet network adapter, cables, Linux operating system and development software.

The kit was originally released with two main methods of code development for graphics/games applications: an implementation of OpenGL called PS2GL, and a low level development library called libps2dev (Playstation2-linux Web Site, 2004). PS2GL did not utilise any of the advanced hardware within the PS2 and provided a development experience very similar to using OpenGL on a standard PC. Using PS2GL did not reflect the development methods being used by professional PS2 developers and this method was not pursued by the author.

Development under libps2dev provided some access to the console hardware including the vector units (VUs). However, the major disadvantage of libps2dev is that it did not provide access to the Direct Memory Access Controller (DMAC) which is a key component, central to providing high performance from the PS2. Under libps2dev, the function of the DMAC was emulated, leading to non-optimal performance from the console.

In November 2002, shortly after the release of the kit, the SPS2 Direct Access Development Environment (Osman, 2002) was released. SPS2 is a low level development library providing direct access to the PS2 hardware and unlike the other development libraries, it has been updated several times since its initial release. A significant feature of SPS2 is that it provides direct access to the DMAC, thus allowing the programmer to utilise the full power and performance of the PS2. SPS2

also provides a development infrastructure similar to that of the professional development kit, with code being developed under SPS2 only requiring minor modification to run under the professional development kit. It was for the reasons of similarity of experience with the professional development kit, and superior performance, that SPS2 was adopted as the development environment/library for games development courses at the University of Abertay Dundee.

PS2 ARCHITECTURE

Figure 1 shows the main internal components and data pathways that exist within the PS2. The main

4k of data memory and can be used in either Micro- or Macro-mode. In Macro-mode, VU0 acts as a second co-processor for the main CPU. In micro-mode VU0 executes its own micro-program independently from the main CPU and can be used in this mode for physics and other intensive in-game calculations. The vector units are connected to the data bus via their associated vector unit interface (VIF). The VIFs are intelligent microcontrollers which interpret the data sent to them using special instructions embedded in the data called VIFCodes.

The Direct Memory Access Controller (DMAC) is responsible for transferring data between main memory and the various processors and scratchpad

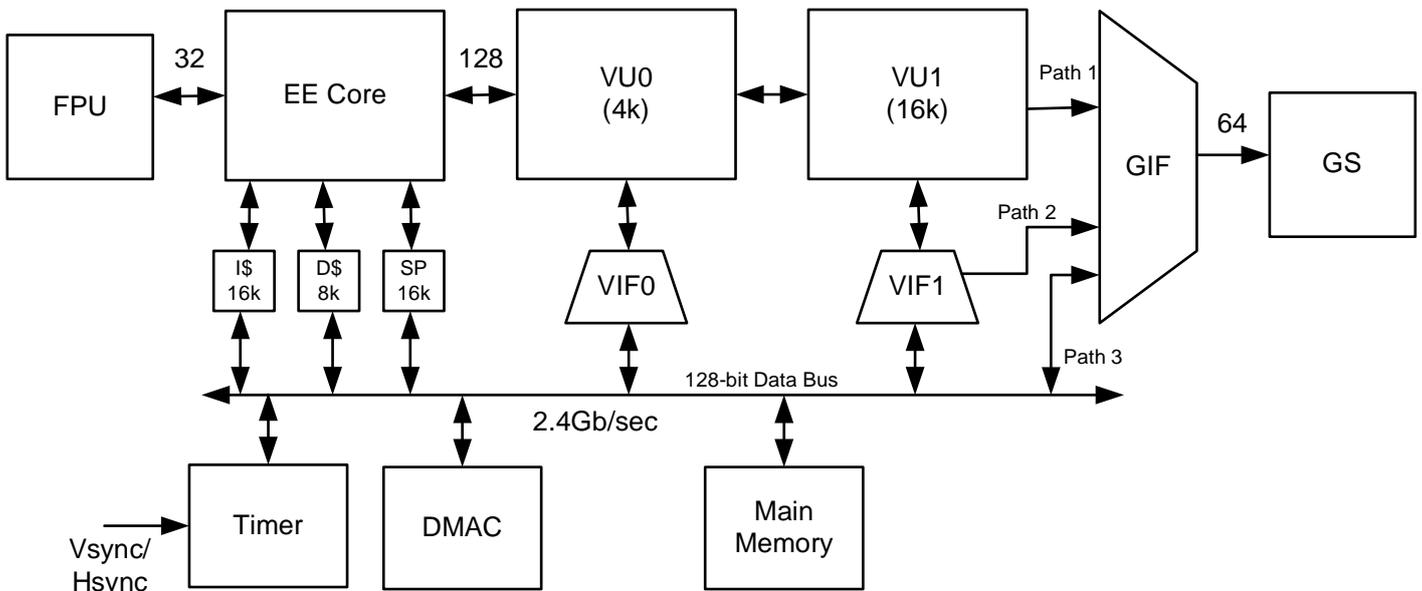


Figure 1

‘Emotion Engine’ (EE) core is a MIPS IV custom processor operating at 300MHz. A 32-bit floating-point unit (FPU) is connected to the EE Core and acts as a coprocessor. Two 128-bit vector processing units are present, VU0 and VU1. VU1 contains 16k of program memory and 16k of data memory and operates in Micro-mode, independently from the main CPU core. VU1 is connected directly to the graphics interface (GIF) which is used to unpack data and send it directly to the graphics synthesiser for rendering. VU1 is mainly used for vertex transformation, lighting and clipping. VU0 contains 4k of program memory and

memory. Correct utilisation of the DMAC is fundamental to obtaining high performance from the PS2. Data transfer is over a 128-bit bus which operates at a maximum transfer speed of 2.4 Gbytes per second.

Three paths exist through the GIF to the graphics synthesiser. Path 1 is from VU1 micro-memory, Path 2 is from VIF1 and Path 3 is from the main data bus. Although there is flexibility in the use of each data path, the recommended function of the data paths is as follows (Sony Computer Entertainment Europe, 2001a). Path 3 is for

loading image data into texture memory within the GS. Path 2 can also be used to upload texture data and for the setting of configuration registers within the GS. Path 2 transfers are convenient, in that they provide inherent synchronisation between texture data and vertex data. Path 1 is the main geometry path for transferring transformed vertex data to the GS for rendering.

DEVELOPMENT PROCESS AND TOOLS

Several methods for games application development are possible with the PS2 Linux kit, with the arrangements adopted by the author being illustrated in figure 2

The development station consists of a PlayStation2

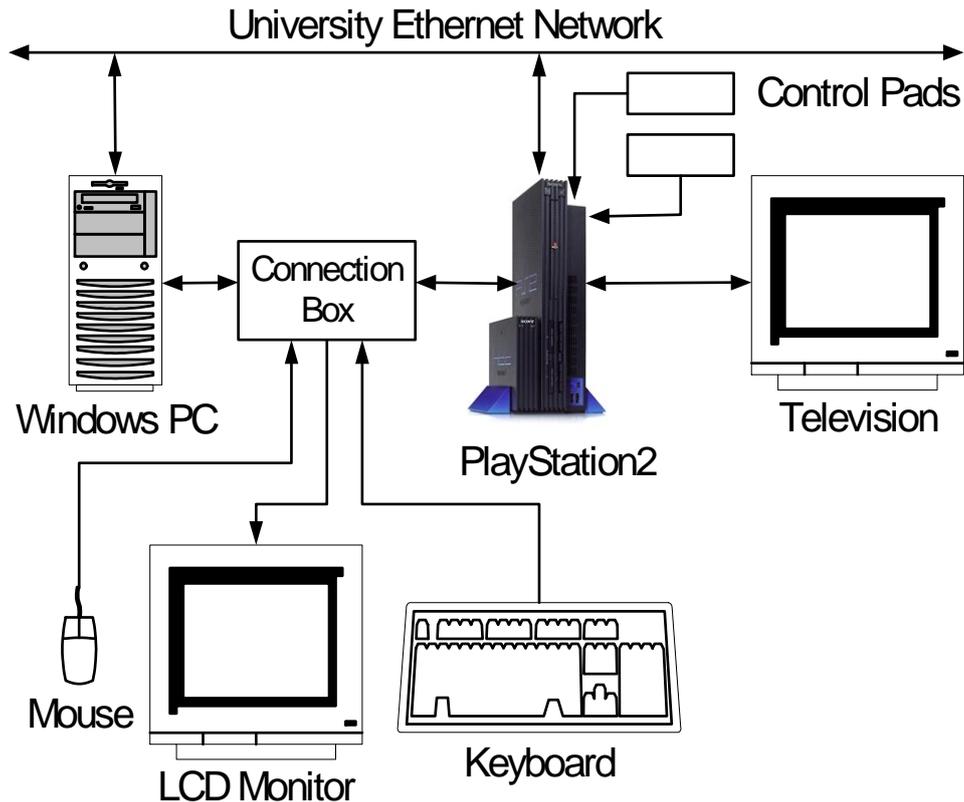


Figure 2

A typical rendering process involves uploading texture data from main memory to the GS via path 2/3 and untransformed vertex data to VU1 micro memory via VIF1. A VU1 micro program transforms, lights and clips the vertex data then sends it over Path 1 to the GS for rendering. Many of these operations are carried out in parallel and are synchronized with the use of appropriate VIFCodes embedded within the vertex and texture data.

Linux kit, Windows PC, two PS2 controllers, dual input LCD monitor, keyboard, mouse, television and peripheral connection box. The dual input LCD monitor is used to display the video output from either the PC or the PlayStation2. The single keyboard and mouse are switchable between the PC and the PS2 using an interconnection box and suitable leads. Both the PC and the PS2 are connected to the University Ethernet network and communicate with each other via the TCP/IP and Server Message Block protocols. Graphics output from the PlayStation2 can be directed to the television for prototyping games applications at the correct resolution and size. This arrangement significantly reduces the amount of equipment

needed per development station and maximises the utilisation of the student laboratory.

Students are free to select a development method that is comfortable to them, but an arrangement found to be successful is as follows. The Linux file system is made available to the PC via a Samba server running on the PS2. Files on the PS2 can be created and edited using a suitable text editor (such as UltraEdit or Visual Studio) running on the PC. From either a Telnet or SSH session from the PC to the PS2, programs can be compiled and executed on the PS2, with graphics output being directed to either the LCD monitor or television. Debug output from the program is sent via the Telnet/SSH session to a console window on the PC. Students store their project code on the main University file servers making this arrangement a robust, effective and secure development environment.

Several tools are used in order to develop games applications under PS2 Linux. Core game code is written in either C or C++ and GNU C and C++ compilers are shipped with the kit. The vector units are proprietary chips (Sony Computer Entertainment Europe, 2001b) which are programmed at assembly level. Both vector units have two execution units (upper and lower) which operate in parallel, leading to assembly code which is written in two parallel streams. This assembly code is compiled to native VU object code with a VU assembler (ee-dvp-as) shipped with the kit. The pairing and scheduling of VU assembly code is relatively complex for students with limited experience of assembly language, but a Vector Command Line (VCL) preprocessor which is shipped with the kit is available to help generate VU code. VCL takes a more traditional single stream of assembly language instructions and from that generates the dual stream of VU assembly code with correct scheduling, pairing and optimisation of instructions. The output from VCL is then compiled with ee-dvp-as to produce the vector unit object code.

The operation of the VUs can be remotely monitored and debugged using a visual debugger (Osman, 2003) running on either a Windows or Linux PC. A server program runs on the PS2 kit monitoring the execution of the VU micro program

under control of the debugger client running on the PC. Full control of the execution of the micro program is obtained together with access to both program and data memory. Using the debugger it is possible to single step the execution of the VU code and observe output on the television/monitor on a frame-by-frame basis.

TEACHING METHODS

Students studying on the BSc Computer Games Technology course gain access to the PlayStation2 Linux kits for the first time in their second year where they study a full module in Console Game Programming. The module introduces topics such as the internal structure and organisation of a games console, the structure and organisation of a games program, and the tools necessary to create and import media content for games. By the end of this module students will understand how consoles are structured and organised and the methods and techniques that are necessary in order to program consoles effectively

Students entering the third year have a solid grounding in console architecture and programming and it is from this background that the PlayStation2 Linux kit is used to introduce the design and construction of console based 3D games engines. Students develop and use the mathematical routines that are necessary for implementing a 3D engine and generate code that interacts directly with the 3D console hardware such as the vector units. By the end of this module students will have created a small prototype 3D game engine and understand the structure, organisation, development and use of modern 3D games engines.

Students undertake a Group project in their third year and an individual honours project in their fourth and final year of study. The PlayStation2 Linux kit is available as a platform to undertake these projects. It provides access to a modern console for testing and evaluating algorithms, techniques and ideas.

A further theme that the PlayStation2 Linux kit is well suited to exposing is that of network programming and gaming. The Playstation2 Linux

kit is supplied with a 10/100 Base-T Ethernet interface network adaptor and a Linux operating system with full network support. Under Linux, the Berkeley Sockets API provides access to the TCP/IP protocol suite that is the backbone of the Internet. The PlayStation2 Linux kit can therefore be used in the teaching of network theory and practice and more specifically in the design and implementation of network computer games. Using the kit it is possible for students to design and create network enabled computer games which have global access through the Internet

CONCLUSIONS

This article has reviewed the internal structure and organisation of the PlayStation2 Linux development kit and has demonstrated how the kit can be applied to teaching and learning on a wide range of topics within Computer Games Technology courses. In practice, the kit has been found to be highly motivational for students and is an invaluable tool for the in-context teaching of Computer Games Technology.

REFERENCES

- Linuxplay WebSite, 2004
<http://linuxplay.com>
‘Information on the PlayStation2 Linux Kit.’”
- Osman S, 2002. ‘A Development Library for Linux (for Playstations2)’
<http://playstation2-linux.com/projects/sps2>
- Osman S, 2003. ‘Sauce's Visual VU Debugger’
<http://playstation2-linux.com/projects/sps2>
- Playstation2-linux Web Site, 2004
<http://playstation2-linux.com/coding-on-playstation2.php>
‘Console game development options for the PlayStation2 Linux Kit.’”
- Sony Computer Entertainment Europe, 2001a. EE Core User’s Manual, 5th Edition
- Sony Computer Entertainment Europe, 2001b. VU User’s Manual, 5th Edition.

Learning and Adaptation in Games

| | |
|---|------------|
| Jankovic, L. and Chiong, R. (Invited Paper) Investigation of strategy dynamics using prisoner's dilemma problem | 371 |
| Björnsson, Y, Hafsteinsson, V., Jóhannsson, Á and Jónsson, E. Efficient use of reinforcement learning in a computer game | 379 |
| Pfeiffer, M. Reinforcement learning of strategies for Settlers of Catan | 384 |
| Ponsen, M. and Spronck, P. Improved adaptive game AI with evolutionary learning | 389 |
| Tambellini, W, Sanza, C. and Duthen, Y. High-level decision learning for non-player characters in video games | 397 |
| Thurau, C., Sagerer, G. and Bauckhage, C. Imitation learning at all levels of game-AI | 402 |
| Fyfe, C. and Wang, T.D. Cooperative population based incremental learning | 409 |

INVESTIGATION OF STRATEGY DYNAMICS USING PRISONER'S DILEMMA PROBLEM

Lubo Jankovic* and Raymond Chiong**

*InteSys Ltd,
University of Birmingham Research Park,
Vincent Drive, Birmingham B15 2SQ, UK
E-mail: L.Jankovic@e-intesys.com

**School of Computer Science,
The University of Birmingham,
Birmingham B12 2TT, UK
E-mail: raychiong@hotmail.com

KEYWORDS

Prisoner's dilemma, game strategy, cellular automata

ABSTRACT

The paper investigates games strategies on the basis of experiments with Prisoner's Dilemma problem. The objective is to determine which strategies have the best chance of winning. Although some strategies, such as tit-for-tat, emerge as better than others in some cases, it appears that there is no overall winning strategy, but that success or failure of individual strategies depends on strategies adopted by a population of opponents. The winning strategy will therefore change dynamically, and will need to be determined while the game in progress. Based on the results of this work, a strategy engine for games development is proposed, and a future development of strategy middleware is discussed.

INTRODUCTION

Finding a winning strategy may not be a trivial task even in a two player game, but it can be extremely hard in a multi-player game. Developers are particularly interested in the quality of games, which will be directly proportional to the quality of gameplay and associated strategies. In this paper we use *Prisoner's Dilemma* problem to investigate success of different strategies in games, in order to learn how to design games which can be played and won at various proficiency levels by a range of players.

After in-depth analysis of theoretical work in the field, we will adopt experimental approach to evaluating different strategies. Using Cellular Automata as suitable mechanism for emergent modeling and simulation, we will conduct experiments, analyse results, and draw conclusions on whether or not there are winning strategies and how to design them into a game.

Prisoner's Dilemma

The story of the *Prisoner's Dilemma* begins with two prisoners who are arrested for committing a crime and are being interrogated. Due to insufficient evidence for a conviction, the two prisoners have been isolated and offered a deal. Under such arrangement, should both prisoners refuse to confess and choose to remain silent, both would receive a small punishment. On the other hand, should any one of the prisoners choose to confess and accuse the other,

while the other prisoner remains silent, the prisoner who confesses would be freed and full punishment would be given to the prisoner who remains silent and was accused by the other. Under the circumstance whereby both prisoners confess and accuse the other, both would receive a heavy punishment but slightly less severe than the full punishment. It is necessary to note that throughout the decision making process of both prisoners, they are neither allowed to communicate with one another nor given to know the decision of the other.

The game becomes more interesting when prison sentences are added as payoffs. For instance, if both the prisoners remain silent (cooperate) without confession, they will be each sentenced for a year in prison. If one confesses (defects) and accuses the other one, the accused one will be sentenced for five years in prison whereas the confessed one will go free. Symmetrical result applies under the reverse scenario. However, if both accuse one another, both will go to prison for three years instead of five years in full. The whole scenario is summarised in Table 1 below using game theoretic notation, where (0, 5) means the first prisoner gets zero years in jail and the second one gets five years.

| | | Prisoner 2 | |
|------------|--------------------|--------------------|------------------|
| | | Cooperate (silent) | Defect (confess) |
| Prisoner 1 | Cooperate (silent) | (1, 1) | (5, 0) |
| | Defect (confess) | (0, 5) | (3, 3) |

Table 1 The payoff matrix

The game of *Prisoner's Dilemma* becomes more complicated when it is played more than once, which Axelrod (1984) called the *Iterated Prisoner's Dilemma*. Instead of using rationality, the prisoners are allowed to form strategies based on the other person's previous moves in an *Iterated Prisoner's Dilemma*. Therefore, competing strategies emerge in order for prisoners to find a winning way. There are several common strategies such as the vicious strategy of universal defection called All-Defect (All-D), naive strategy of unconditional cooperation called All-Cooperate (All-C), or a strategy of come-on initial cooperation followed thereafter by vicious defection (C-then-All-D). However, one of the most well known strategies is the strategy called Tit-for-Tat (TFT). TFT cooperates on the first move and thereafter simply repeats its opponent's move on the previous round. It has established a

reputation on both experimental and theoretical grounds as particularly robust (Axelrod 1980a, 1980b, 1984; Axelrod & Hamilton 1981). The implication behind this *Iterated Prisoner's Dilemma* is that we often face certain situations or dilemmas in our daily life over and over again. Many a time, we must make fresh decision of what we are going to do each time and make use of the correct strategy at the right time.

ANALYSIS OF STRATEGIES

In this section we will explore a little further different kinds of strategies in *Prisoner's Dilemma*. This is necessary since one of the main objectives of our study is to investigate the winning strategies for games.

| Strategies | Description |
|------------|--|
| All-C | Always cooperates. |
| All-D | Always defects. |
| TFT | Cooperates on the first move and then plays what its opponent played on the previous move. |
| Spiteful | Cooperates until the opponent defects, then defects all the time. |
| Pavlov | Cooperates on the first move and then cooperates only if the two players made the same move. |
| TF2T | Cooperates except if opponent has defected twice consecutively. |
| Random | Randomly plays, cooperates with probability 1/2. |
| soft_majo | Plays the opponent's most used move and cooperates in case of equality (first move considered as equality). |
| per_ddc | Plays periodically [defect, defect, cooperate]. |
| per_ccd | Plays periodically [cooperate, cooperate, defect]. |
| mistrust | Defects, then plays opponent's move. |
| per_cd | Plays periodically [cooperate, defect]. |
| hard_tft | Cooperates except if opponent has defected at least one time in the two previous move. |
| Slow_tft | Plays [cooperate, cooperate], then if opponent plays two consecutive time the same move plays its move. |
| hard_majo | Plays opponent's majority move and defects in case of equality (first move is considered to be equality). |
| prober | Begins by playing [cooperate, defect, defect], then if the opponent cooperates on the second and the third move continues to defect, else plays tit-for-tat. |

Table 2 Major strategies in Iterated Prisoner's Dilemma

In the early days, the only rational and best strategy for the classic *Prisoner's Dilemma* was simply to defect all the time, as it was the safest to defect regardless of what action the opponent took. As mentioned in the previous section, various kinds of strategies only started to emerge since the tournament organised by Axelrod in the 1980s with an aim to study strategies that would encourage behaviour of cooperation in *Iterated Prisoner's Dilemma*. When Axelrod initiated his first *Iterated Prisoner's Dilemma* tournament, only around fourteen strategies had been proposed. Today, there are numerous strategies used in the *Iterated Prisoner's Dilemma*. Among the most common are All-C, All-D, TFT, Spiteful, Pavlov, TF2T and Random. There are also some strategies which are less common, but have been used in several prominent experiments of the *Iterated Prisoner's Dilemma* in the past, such as soft_majo, per_ddc, per_ccd, mistrust, per_cd, hard_tft, slow_tft and hard_majo. Table 2 summarises the major strategies that have been mentioned.

TFT, a very simple strategy and overall winner of all Axelrod's *Prisoner's Dilemma* tournament, has established its reputation among all other strategies as the most robust strategy available in *Prisoner's Dilemma* game on both experimental and theoretical grounds (Axelrod 1980a, 1980b, 1984; Axelrod and Hamilton 1981). TFT is simple in the sense that it cooperates on its first move and follows opponent's moves subsequently. According to Axelrod, it has three characteristics that account for its impressive performance:

1. It is nice (cooperates on the first move).
2. Retaliatory (punishes defection in the prior move with defection).
3. Forgiving (immediate return to cooperation after one C of the adversary).

Although TFT has been claimed as the best strategy in *Prisoner's Dilemma* for the past twenty years since Axelrod's tournaments, a lot of game theorists in this field have never ceased to depose its status with endless research to investigate more winning strategies. Someone even protested that logically speaking TFT can only come to a draw scenario, and it can never win or score more points than its opponent in a single game since its first move was always to cooperate (Sober & Wilson, 1998).

In the late 1980s, Boyd and Lorberbaum (1987) showed that no deterministic strategy is evolutionarily stable in the *Iterated Prisoner's Dilemma*. Lorberbaum (1994) extended the instability proof to mixed strategies as well. In mixed strategies, the strategies generate moves probabilistically depending on the opponent's previous move.

In early 1990s, Nowak and Sigmund (1993b) reported that Pavlov had outperformed TFT in evolutionary *Iterated Prisoner's Dilemma* games embedded with certain random disturbances such as 'noise'. According to Nowak and Sigmund, Pavlov was better than TFT because it can correct occasional mistakes and prevents invasion of strict cooperators by exploiting them. However, they did not claim Pavlov to be the best because in their experiments, Pavlov lost to All-D.

Later in mid 1990s, Beaufils, Delahaye and Mathieu (1997) proposed Gradual as a better strategy than TFT given its three important qualities, namely its kindness (it does not begin with defect), reactivity (it defects when the opponent has defected) and forgiveness (it goes back to cooperate after punishment). They further derived four simple principles of doing well in an *Iterated Prisoner's Dilemma* game based on their experimental results:

1. Don't be envious.
2. Don't be the first to defect.
3. Reciprocate both cooperation and defection.
4. Don't be too clever.

It is necessary to note that there is strategy like All-C, which cooperates on every move no matter what the partner does. This kind of strategy will eventually be defeated since the opponent would have no incentive to cooperate, but to defect in order to earn the greater payoff on every move it makes. In our opinion, the reason All-C exists perhaps is to show its

simplicity to the opponent and hopeful that its opponent will appreciate it and in turn to cooperate as well.

In conclusion, the various strategies in *Prisoner's Dilemma* have formed a simple yet profound inspiration to our every walk of life and all humankind experience. In the next section, we will discuss the characteristics of the cellular model we use in our current research study, which was based on the experimental simulations by Grim (1997).

METHODOLOGY

As last section has introduced the subject area and outlined the literature on the *Prisoner's Dilemma*, this section aims to provide an understanding on the research approach we adopted in our study and the methodology we used in our experiments. Generally speaking, the research approach we will adopt is a cellular automata model. There are several examples from the previous work done with the use of cellular automata in *Spatial Prisoner's Dilemma*, such as Axelrod (1984); Nowak and May (1992, 1993); Mar and St Denis (1994); and Grim (1996, 1997). We have largely followed the guidelines and ways of encoding different strategies in *Prisoner's Dilemma* game based on the implementation of Grim (1996, 1997) to ensure the validity of our study.

Cellular Automata in Spatial Prisoner's Dilemma

Recently, cellular automata have been used very often as a modeling tool in various researches, including the research in the field of *Prisoner's Dilemma*. The introduction of spatial dimensions in *Prisoner's Dilemma* have greatly linked cellular automata together with *Spatial Prisoner's Dilemma*, in which the interactions between cells among their neighbourhood could be simulated and the dynamics of their behaviour could be observed in various ways. The cellular automata model for *Prisoner's Dilemma* was presented first by Axelrod (1984), and thereafter an investigation into the dynamics of such model was done in Nowak and May (1992, 1993). Grim (1997); and Mar and St Denis (1994) also reported on a cellular automata model in their study of undecidability in *Spatial Prisoner's Dilemma*.

In 1984, Axelrod laid the foundation of the use of cellular automata in *Prisoner's Dilemma* with an analysis into a cellular automaton in which all cells are occupied with players of different strategies. In Axelrod's model, players get their points based on the payoffs against their neighbouring cells respectively, and copy more successful strategies from their neighbors.

While Axelrod studied a large number of possible strategies in his tournaments, Nowak and May (1992) focused on players that behaved only either cooperatively or defectively. They investigated various configurations in *Spatial Prisoner's Dilemma* at the initial stage and then studied the dynamics of behaviour that evolve among the interactions between cells. In their model, the state of a cell is determined by a set of four rules:

1. Strategies which are cooperators and were cooperators in the previous generation.
2. Strategies which are defectors and were defectors in the previous generation.
3. Strategies which are cooperators but were defectors in the previous generation.
4. Strategies which are defectors but were cooperators in the previous generation.

In this paper, we report on a model where different strategies are represented as cellular automata based on the paper of Grim (1997). As Grim studied the undecidability in *Spatial Prisoner's Dilemma*, our emphasis will be on the establishing winning strategies in multi-agent interaction through the results produced by the model. The next section will describe the model we built for experimental purposes.

Cellular Automata Model

Our model is a simple Java program implemented in the form of cellular automata to simulate agents of different strategies in *Spatial Prisoner's Dilemma*. In this program, agents representing different strategies are distributed on two-dimensional grid. The grid has overlapping edges, which means that every cell on the grid has eight immediate neighbouring cells including those at the edges of the grid.

Each agent will occupy one cell on the grid. Different colours are used for the cells to represent each individual strategy, which occupies that particular territory. Each cell is designed to compete against its immediate neighbours with its own strategy. Scores are calculated as each cell progressively competes with its immediate eight neighbouring cells (see Figure 1), and the winning cell will inhabit the territory of the losing cell by replacing its colour during the process.

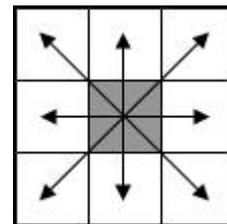


Figure 1 Spatial interactions among neighbouring cells

The grid simulates a society of prisoners interacting in a spatial game with different strategies adopted. Every prisoner is due to play with the eight neighbouring prisoners, and the payoffs they get represent their fitness. The following two sections will discuss the two major implementation issues in our model.

Strategy Encoding

In our Java program, the cellular automata that represent different strategies are encoded as binary strings. For instance, all the strategies are represented in the program as 3-bit binary numbers with 0 and 1 corresponding to

defection and cooperation. We selected eight most common strategies for our experiments based on the implementation by Robert Rothenburg for Grim's paper (1997). The eight strategies mentioned have been represented from 000 to 111, or 0 to 7, as shown in Table 3.

| Strategies | Binary representation | Colour representation |
|------------|-----------------------|-----------------------|
| All-D | 000 | Red |
| Random | 001 | Yellow |
| Pavlov | 010 | Gray |
| Prober | 011 | Orange |
| Gradual | 100 | Magenta |
| Mistrust | 101 | Blue |
| TFT | 110 | Cyan |
| All-C | 111 | Green |

Table 3 Representation of strategies in binary numbers and colours

As for the output of the program, we use eight different colours to represent eight different strategies we have particularly selected for this study. Table 3 also shows the representation of colours for different strategies.

Fitness Function

In our program, we determine the fitness of each cell or agent by assuming that the agent with highest payoff score is the fittest. On the other hand, the agent with lowest payoff score naturally will be the weakest. This means that the fitness scores that each cell accumulated will determine which agent will inhabit other's cell and which agent will 'die off'. Generally, our model aims to simulate the *Prisoner's Dilemma* game in spatialised form for a specified number of rounds among the agents. By keeping track the fitness score of each agent, we will be able to evaluate the eventual winner evolves from these given rounds of play.

EXPERIMENTS AND RESULTS

Iterated Prisoner's Dilemma Simulation

In our first experiment, we conducted a simulation for TFT and 10 other strategies selected from the *Iterated Prisoner's Dilemma*. It was a simple round-robin simulation where each strategy met with all other strategies. This experiment was designed for each strategy to play with one another iteratively for 50 rounds with the payoffs as shown in Table 4. A total final score will be calculated for each strategy from the sum of scores achieved based on the payoffs after each round of play.

At the end of this experiment, we measured the strength of each strategy according to its overall performance during the simulation as reflected in the total score it has accumulated from the payoffs. This experiment followed closely the ways Axelrod (1984) conducted his tournaments for *Iterated Prisoner's Dilemma* in the early 1980s. The simulation program used for this experiment was acquired from

<http://www.iterated-prisoners-dilemma.net> based on Richard Dawkins's book (1989).

| | | Strategy 2 | |
|------------|-----------|------------|--------|
| | | Cooperate | Defect |
| Strategy 1 | Cooperate | 3, 3 | 0, 5 |
| | Defect | 5, 0 | 1, 1 |

Table 4 The payoffs table for the first experiment

Table 5 summarises the results of our first experiment. The results were produced on the basis of a round-robin simulation, after each strategy had played against all other strategies including itself. Each row of the table represents the score of each strategy after playing against its opponent appeared on top of the column. The score was based on 50 rounds of iterative play. The eventual winner was determined on the basis of total score that each strategy had acquired after a complete round-robin.

From Table 5 we can observe that TFT was the eventual winner of this simulation experiment, in which it scored the highest. However when we look closer at each individual encounter between TFT and other strategies, TFT in actual fact had never won a single encounter throughout the simulation; It drew eight times and lost two times. It is necessary to note that it actually lost to All-D and Prober. However, TFT eventually won the overall game by having accumulated the higher payoffs. If we had allowed the experiment to be conducted in a league form which each strategy ranked according to the number of wins and loses, TFT would have turned out to be at the bottom of the league.

All-D, on the other hand, had won eight of its individual encounters and drawn twice – to itself and mistrust. It is worth mentioning that All-D had indeed beaten TFT by 104 over 99 in their individual encounter. Based on this performance, All-D could be said to be a relatively strong strategy, if not the best. However, the final total result sprung a surprise as it ranked second from the bottom and only performed slightly better than Random. This proved that mutual defection, although could win over the opponent in short-term, does not bring success in the long run due to its low payoff (one point).

We noticed that the 'nice' strategies such as All-C and TF2T came out second and third based on their final total score. It is necessary to note that All-C cooperates unconditionally regardless of opponent's moves and TF2T cooperates at the first two moves before following the opponent's previous moves subsequently.

| Strategies | All-C | All-D | TFT | Spiteful | Pavlov | TF2T | Random | Gradual | mistrust | Prober | Total Score |
|------------|-------|-------|-----|----------|--------|------|--------|---------|----------|--------|-------------|
| All-C | 150 | 0 | 150 | 150 | 150 | 150 | 87 | 150 | 147 | 138 | 1272 |
| All-D | 250 | 50 | 104 | 54 | 150 | 58 | 162 | 110 | 50 | 54 | 1042 |
| TFT | 150 | 99 | 150 | 150 | 150 | 150 | 110 | 148 | 125 | 123 | 1355 |
| Spiteful | 150 | 49 | 150 | 150 | 147 | 150 | 159 | 49 | 53 | 51 | 1108 |
| Pavlov | 150 | 25 | 150 | 32 | 150 | 150 | 117 | 150 | 101 | 128 | 1153 |
| TF2T | 150 | 48 | 150 | 150 | 150 | 150 | 109 | 150 | 147 | 54 | 1258 |
| Random | 192 | 22 | 110 | 29 | 102 | 109 | 110 | 103 | 125 | 110 | 1012 |
| Gradual | 150 | 35 | 148 | 54 | 150 | 150 | 118 | 150 | 146 | 132 | 1233 |
| mistrust | 152 | 50 | 125 | 53 | 101 | 152 | 130 | 151 | 50 | 110 | 1074 |
| Prober | 158 | 49 | 128 | 56 | 128 | 59 | 120 | 147 | 110 | 141 | 1096 |

Table 5 The results of round-robin simulation of Iterated Prisoner's Dilemma

This result could prove that rationality does not count for all in real-life, but cooperative nature and forgiveness towards others may bring positive consequences. Pavlov, as commented by Nowak and Sigmund (1992, 1993a) to be better than TFT, did not do particularly well in our simulation. Gradual, which Beaufils, Delahaye and Mathieu (1997) claimed to have outperformed TFT, ranked in the middle of the chart.

Spatial Prisoner's Dilemma Simulation

We conducted our second experiment with a simulation of *Spatial Prisoner's Dilemma*. We used a simple Java program to simulate various strategies as cellular automata, with each strategy occupying the same quantity of cells in the beginning of the simulation. This simulation was designed to allow individual strategy to play against its eight immediate neighbouring cells, with winning strategy evolved and inhabited the cell of the losing strategy.

As mentioned before, each strategy is represented by a different colour. The process of a winning strategy inhabiting a losing strategy was represented with the change of colours among the cells. The simulation was repeated until the evolution of strategies was stabilised. This would differ according to the size of the grid that holds the cells.

As for our simulation, we noticed that the evolving strategies became stable after 15 to 20 rounds of play. Figure 2 shows the distribution of the population before the game, namely the round zero. As depicted by Figure 2, there were eight different strategies represented by eight different colours (see Table 3) on the grid before the start of the game. The colours were randomly distributed but with even quantities.

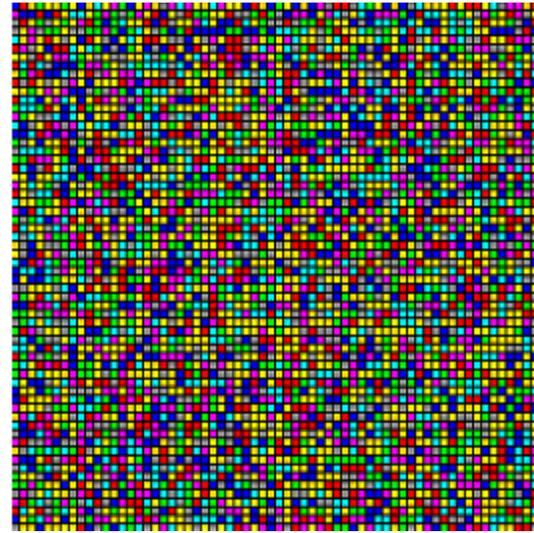


Figure 2 Distribution of strategies before the game

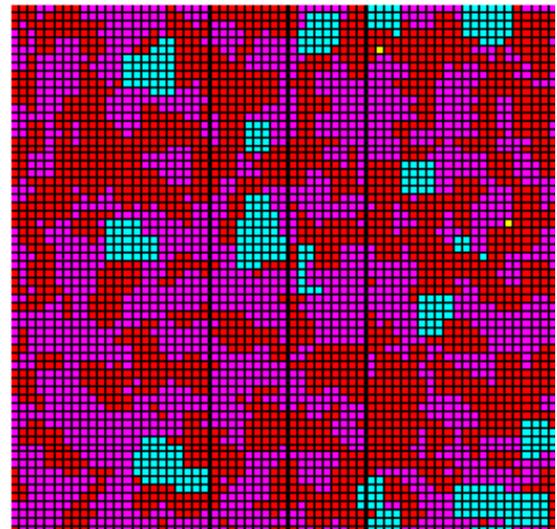


Figure 3 Distribution of strategies after three rounds

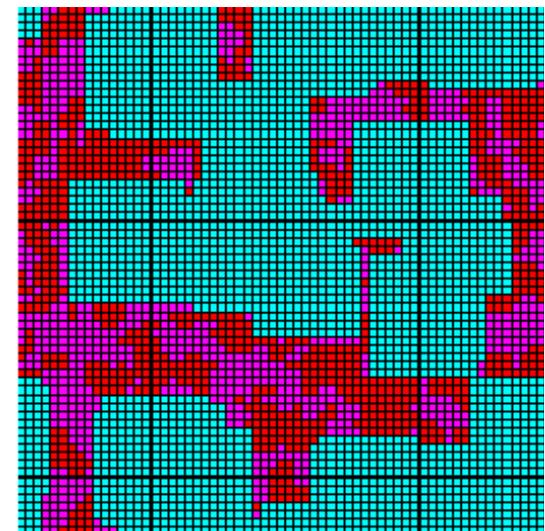


Figure 4 Distribution of strategies after eight rounds

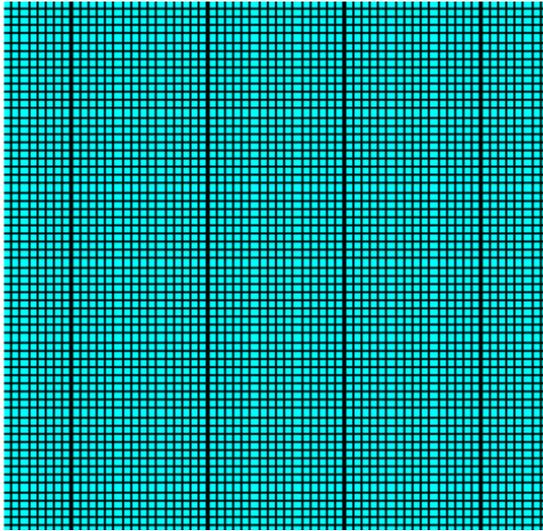


Figure 5 TFT after fifteen rounds

After three rounds of play, Figure 3 shows that only four strategies remained on the grid – Gradual (magenta), All-D (red), TFT (cyan) and Random (yellow). We noticed that TFT was not performed particularly well at this stage, as it was overshadowed by Gradual and All-D. Random survived round but was expected to ‘die off’ very soon due to its limited territory.

When it reached round eight, TFT started to dominate, with Gradual and All-D began to shrink, as shown in Figure 4. Based on the evolutionary pattern and growth rate, we predicted at this stage that TFT would continue to grow and evolve as the eventual winner in this experiment.

As shown in Figure 5, TFT conquered the whole grid in round fifteen and emerged as the eventual winner as predicted. We repeated this same simulation for five times to observe if there would be any possible variation to the results of our experiment.

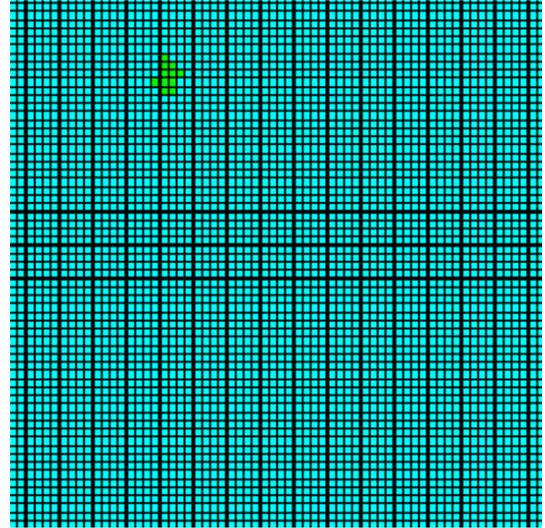


Figure 6 A unique result with TFT and All-C both stabilized

The repeated simulations have shown the same results throughout, except on one occasion when TFT and All-C (green) both have become stabilised in the end with All-C occupies less than 1% of overall cells. This unique result is depicted in Figure 6.

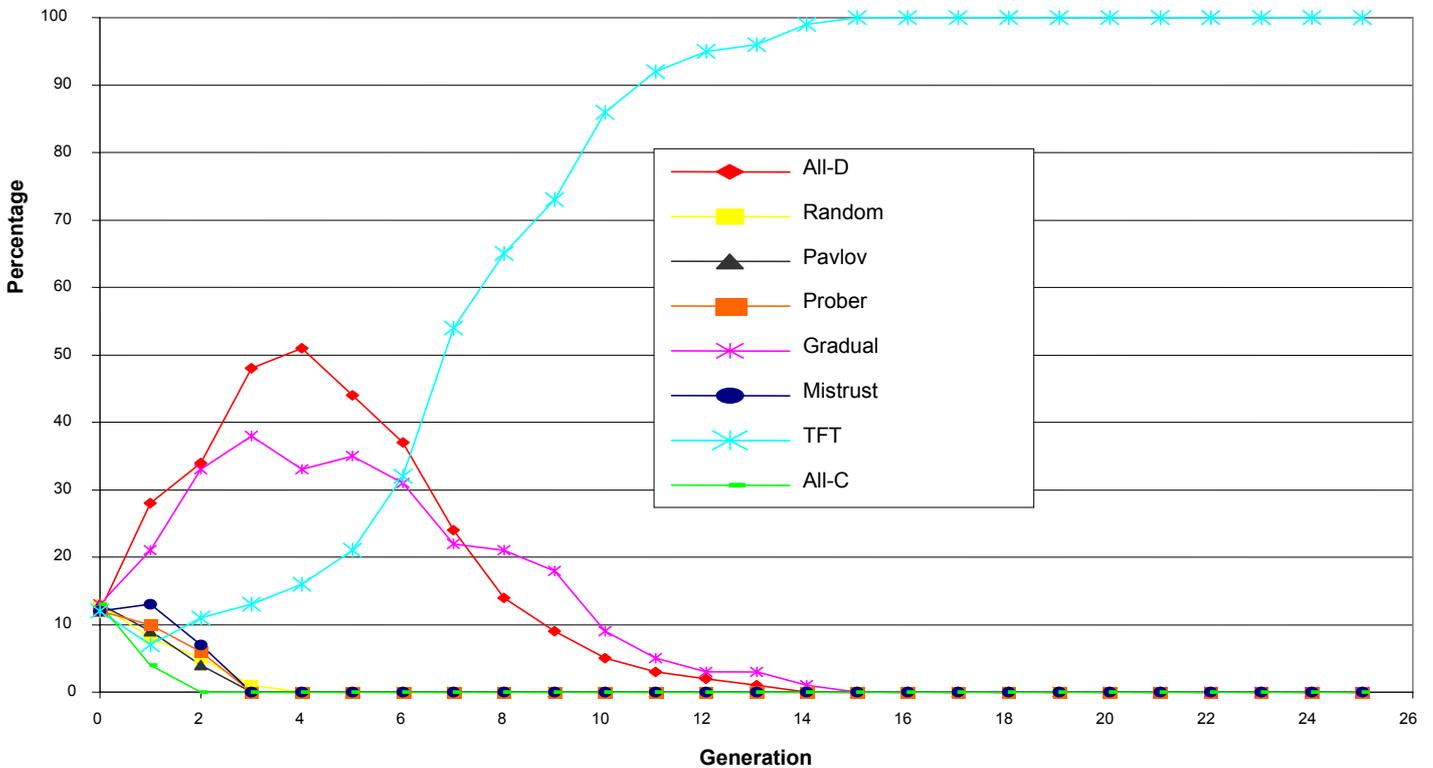


Figure 7 Percentage of evolving strategies over 25 generations

The results from our second experiment have clearly shown that TFT has outperformed all the other strategies, except in certain rare occasions. We also noticed that Gradual and All-D have similar evolution at the beginning rounds, but would start to shrink in size after TFT has begun to take control. Figure 7 shows percentage of evolving strategies over 25 generations.

DISCUSSION

We demonstrated once again in our second experiment that TFT was again the overall winner. Based on the results, we observed that TFT sprung no surprise to colonise the spatialised game of Prisoner's Dilemma played on a grid in a given 25 rounds. However, we also noticed that TFT did not do well in the first few rounds of play. The reason TFT did not do well at the initial stage of the experiment could be caused by its nature in cooperating at its first move. TFT came back strongly after 5-8 rounds due to its strength in accumulating higher scores. This again concurs with our previous two conclusions.

Another interesting result we learned from our second experiment was that, in one particular simulation, All-C survived the 25 rounds of play together with TFT. This is a rare case. From this particular incident we wish to conclude that TFT has the tendency to perform rather well when it played against strategies similar to its nature. All-C survived because of the advantage of playing with TFT. Since it is TFT's nature to follow the opponent's subsequent moves after the first move, this eventually helped All-C to gain higher scores while playing together with it.

Therefore, we could conclude from this second experiment that there is a group effect in Spatial Prisoner's Dilemma. TFT won the play because there were plenty of similarly 'nice' strategies playing in the simulation. Consider a population with a lot of defectors, it would be difficult for TFT to win over them because of its slight 'disadvantage' in its first move to cooperate. The optimal strategy in such a population would be to defect every time, yet TFT loses on the very first turn.

In this context, we believe that it is not always true to say any given strategy is the best, as we have learned from our study that there is no single best strategy for the Iterated Prisoner's Dilemma game. Whichever strategy comes out to be the best would depend on its opponent's strategy or the strategies among the population, yet players obviously would not be able to know the opponent's strategy until after their encounter with each other.

We therefore believe that continuous evaluation of the game is necessary in order to enable us to determine the best strategy on the fly, while the game is in progress. This approach would enable dynamic change of strategies of one agent in response to strategies adopted by the population of opponents. This method would be the basis for a strategy engine for games, which we named "strategy middleware". It would be used in a similar way as physics engines,

working as a separate soft co-processor alongside the game engine.

We believe that strategy middleware would help developers create games in which the strategy would neither be hard coded nor based on learning algorithms, but it would be continuously developed and re-assessed using Prisoner's Dilemma Problem. In single player games, this could help to set the strategy dynamically when playing against the computer. In multiplayer games, the strategy middleware could give advice to individual users, in live situation when facing dynamically changing strategies of multiple opponents. In either case, developers would not need to develop strategies. Instead, they would use services of strategy middleware.

CONCLUSIONS AND FUTURE WORK

In this paper we investigated different games strategies using Prisoner's Dilemma problem. We adopted experimental approach based on cellular automata, where the cells represented a population of agents each playing against their immediate neighbourhood.

Our work showed that there is no best strategy, but which strategy wins depends on the complementary strategies adopted by the population of opponents, and duration of the game. We conclude that the winning strategy will not be static, but that it will change as strategies of the opponents change.

Adapting the game to a population of players at varying levels requires dynamic re-evaluation of strategies while the game is in progress. This requires a strategy engine that will look at what the population of players does and choose suitable strategies in order to achieve different levels of the game.

This can be achieved through a future development and implementation of strategy middleware that would help developers with strategy services, in a similar way as physics engines help with physics services. Prisoner's Dilemma problem could be used as the basis for this development. We believe that this approach to game strategy would open new possibilities for games.

REFERENCES

- Axelrod, R. (1980a). Effective Choice in the Prisoner's Dilemma. *Journal of Conflict Resolution*, 24, 3-25.
- Axelrod, R. (1980b). More Effective Choice in the Prisoner's Dilemma. *Journal of Conflict Resolution*, 24, 379-403.
- Axelrod, R. (1984). *The Evolution of Cooperation*. New York: Basic Books.
- Axelrod, R., & Hamilton, W.D. (1981). The evolution of cooperation. *Science*, 211, 1390-1396.
- Beaufils, B., Delahaye, J.P., & Mathieu, P. (1997). Our Meeting With Gradual: A Good Strategy For The Iterated Prisoner's Dilemma. In *Proceedings of the 5th International Workshop on the Synthesis and*

- Simulation of Living Systems, Artificial Life V* (pp. 202-209). MIT Press.
- Boyd, R., & Lorberbaum, J.P. (1987). No pure strategy is evolutionary stable in the repeated prisoner's dilemma game. *Nature*, 327, 58-59.
- Dawkins, R. (1989). *The Selfish Gene* (2nd ed.). Oxford University Press.
- Delahaye, J.P., & Mathieu, P. (1995). Complex strategies in the iterated prisoner's dilemma. In A. Albert (Ed.), *Chaos and Society* (Vol. 29, pp. 283-292). Amsterdam: IOS Press.
- Grim, P. (1996). Spatialization and generosity in the stochastic prisoner's dilemma. *BioSystems*, 37, 3-17.
- Grim, P. (1997). The Undecidability of the Spatialized Prisoner's Dilemma. *Theory and Decision*, 42, 53-80.
- Lorberbaum, J.P. (1994). No strategy is evolutionary stable in the repeated prisoner's dilemma. *Journal of Theoretical Biology*, 168, 117-130.
- Mar, G., & St. Denis, P. (1994). Chaos in Cooperation: Continuous-Valued Prisoner's Dilemmas in Infinite Valued Logic. *International Journal of Bifurcation and Chaos*, 4, 943-958.
- Nowak, M.A., & May, R.M. (1992). Evolutionary games and spatial chaos. *Nature*, 359, 826-829.
- Nowak, M.A., & May, R.M. (1993). The spatial dilemmas of evolution. *International Journal of Bifurcation and Chaos*, 3, 35-78.
- Nowak, M.A., May, R.M., & Sigmund, K. (1995). The arithmetics of mutual help. *Scientific American*, 272, 76-81.
- Nowak, M.A., & Sigmund, K. (1992). Tit-for-Tat in heterogeneous populations. *Nature*, 355, 250-252.
- Nowak, M.A., & Sigmund, K. (1993a). A strategy for win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game. *Nature*, 364, 56-58.
- Nowak, M.A., & Sigmund, K. (1993b). Chaos and the evolution of cooperation. *Proceedings of the National Academy of Sciences, USA*, 90, 5091-5094.
- Sober, E. & Wilson, D.S. (1998). *Unto Others. The Evolution and Psychology of Unselfish Behavior*. Cambridge, MA: Harvard University Press.

EFFICIENT USE OF REINFORMENT LEARNING IN A COMPUTER GAME

Yngvi Björnsson, Vignir Hafsteinsson, Ársæll Jóhannsson, and Einar Jónsson
Reykjavík University, School of Computer Science
Ofanleiti 2
Reykjavik, Iceland
E-mail: {yngvi, vignirh02, arsaelltj02, einarj02}@ru.is

KEYWORDS

Computer game-playing, Mobile games, Artificial intelligence, Reinforcement learning.

ABSTRACT

Reinforcement learning methods are not yet widely used in computer games, at least not for demanding online learning tasks. This is in part because such methods often require excessive number of training samples before converging. This can be particularly troublesome in mobile game devices where both storage and CPU are limited and valuable resources. In this paper we describe a new AI-based game for mobile phones that we are currently developing. We address some of the main challenges of incorporating efficient on-line reinforcement learning methods into such gaming platforms. Furthermore, we introduce two simple methods for interactively incorporating user feed-back into reinforcement learning. These methods not only have the potential of increasing the entertainment value of games, but they also drastically reduce the number of training episodes needed for the learning to converge. This enhancement made it possible for us to use otherwise standard reinforcement learning as the core part of the learning AI in our game.

INTRODUCTION

Reinforcement learning methods have in recent years increased in popularity and are now-a-days used for solving various demanding learning tasks. Although they have been used successfully for years in for example classic board game-playing programs (Tesauro 1994), they have only recently caught the interest of the commercial game community (Evans 2002; Manslow 2004). However, one of the main criticisms these methods have met is their lack of efficiency. That is, many trials are often needed before the learning converges, rendering them practically inapplicable in fast paced game environments where many trials are a luxury one cannot afford. For any on-line learning method to be applicable in practice in games it needs to be *fast*, *effective*, *robust*, and *efficient* (Spronck 2003).

In this paper we address some the challenges of incorporating efficient on-line reinforcement learning techniques into gaming environments. We are using reinforcement learning as the central component in an AI-based game that we are developing for mobile phones — a platform where efficiency is of a paramount importance, in part because of limited CPU power. We describe the main learning method used in our game, and introduce and

experiment with two enhancements that allow us to incorporate user feedback interactively into the learning process. Not only does this allow the user to take on a more active role in the development of the game characters, but also drastically reduces the number of training episodes needed for the learning to converge.

These enhancements made it feasible for us to use reinforcement learning as the central learning component in our mobile game. Although the game is still in early stages of development, we have finished prototypes of the components that most heavily rely on learning (the athletic training events). We used them to demonstrate the feasibility of reinforcement learning when augmented with the new learning enhancements.

The remainder of the paper is structured as follows. In the next section we give an overview of the game, and thereafter we explain in detail the learning procedure used in the athletic training event module. We next provide experimental results, and finally conclude and discuss future work.

GAME DESCRIPTION

The game consists of a (pet) creature in a virtual home environment inside a mobile phone or PDA. The objective of the game is to raise a creature with desirable characteristics. The player (user) decides which characteristics are important, and he or she has various ways of interacting with the creature to help shape its character. This is in the spirit of games like *Black & White*. For example, a player might want to raise the creature to be a good athlete, and would therefore have the creature do activities that develop athletic skills, such as swimming or running. Alternatively, the player might prefer more of a thinking creature, and instead have it read books and solve puzzles. In addition to these training activities, the user can show off the creature's skills by having it participate in competitions like athletic or mind-game events. These competitions can be either single-player (e.g. racing against time) or multi-player (e.g. competing against a friend's creature).

Game Architecture

The game is written in Java2 Micro Edition (CLDC 1.1 and MIDP 2.0). Most new phones do or will support these new standards. We have tested the game on a Nokia 6230 mobile phone.

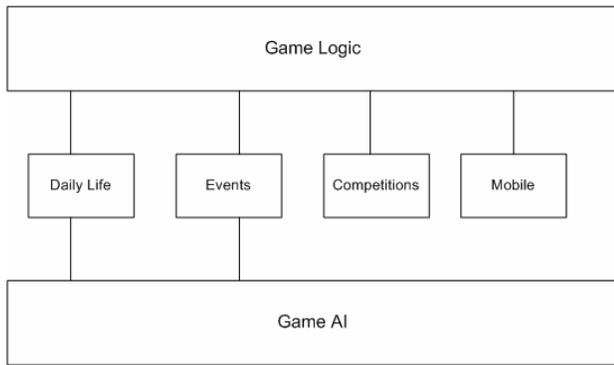


Figure 1. The main modules of the game

The main modules of the game are shown in Figure 1. These modules are loosely coupled and all communications between them are done via interfaces.

Daily-life

This module focuses on the creature's principal needs, such as eating, drinking, sleeping, and social interaction. The food has various attributes; it can be healthy, tasty, fattening etc. Healthy and fattening food types have physical impact on the creature, where as attributes such as taste affect the creature's mental condition (we're not too happy when we need to eat something that tastes bad).

Events

This module contains various sub-games and challenges where the user can train the creature in various skills and activities and benchmark its performance. These include physical activities such as sprinting and swimming events, and simple mind-games like 9-men-morris. The more training the creature gets in an activity the better it becomes. Not only do the physical or mental attributes improve, but the creature also learns the best strategy for the given activity, for example a good strategy in 9-men-morris, or how to pace its speed during a 200 meter sprinting event. During the training the user may play the role of a personal trainer by giving advices, for example by playing a game against it or telling it how to pace its speed during a training run.

Competitions

Players can sign their creatures up for competitions and tournaments, either single or multi-player. The primary purpose is to show-off abilities of the creature, but good results in such competitions (such as 1st prize in a sprinting competition) can provide the player with money and rare valuable items. In a competition the creature's performance is decided by its unique combination of personality, abilities, and skills acquired through training and other previous interactions with the user. Thus it is important to train the creature well before entering a competition.

Mobile

All mobile device specific code, such as graphics, communications, menus and other user interface components are kept in a separate module. This enforces transparent design which will enable the game to be extended to other gaming platforms.

Game AI

Machine learning plays a central role in this game. The main objective of the game AI is to support creatures that evolve and adapt based on the player's preference. There are several attributes that make one creature distinct from the next, both physical (such as strength, dexterity, constitution, fat and health) and mental (such as intelligence, wisdom, patience and social behavior). The way the user interacts with the creature affects how these different attributes develop. The exact learning tasks are somewhat different and can be coarsely divided in two.

In the *Daily-life module* the task of the learning is to allow the various characteristics of the creatures to adapt and develop in response to the user's input. The user is here primarily in a parenting role. The creature, when left unattended, starts exploring and doing things on its own. For example, it could: go hang out around the eating bowl looking for food, go playing, eat your slippers, give you flowers, break your favourite vase, or even poop on the floor. It is up to the user to raise the creature in a way such that it acts responsibly, for example by rewarding it when it does good things and punishing it when it does undesirable things, e.g. eating unhealthy, staying up late, or avoiding exercise and social interactions. These are all activities that negatively affect the creature's health and might possibly eventually lead to its death.

In the *Events module*, on the other hand, the creature learns from experience by repeatedly performing athletic or mind game activities. The more it practices the better it gets at these activities. There is no input necessary from the user. However, if the user wants she can act as a coach and give advice. The creature takes notes of and uses the advice to speed up its learning curve. In this paper we will focus on the learning in this module.

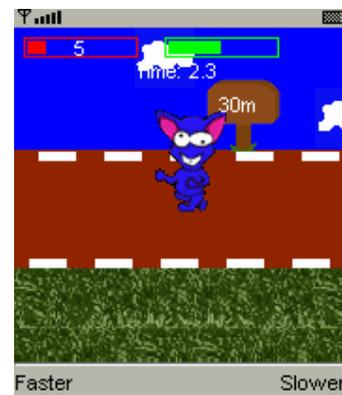


Figure 2. A scene from the Sprinting sub-game

LEARNING

In here we describe the learning method used in the game's athletic training events. All the different sport training events (currently only sprinting and swimming) use the same underlying *reinforcement learning* based methodology. The task is to learn the optimal racing strategy for the given event, possibly with some learning input from the user. The best strategy will differ from one creature to the next because of different physical characteristics (e.g. weight and maximum stamina). For the same reason the optimal racing strategy may differ from one week (or day) to the next, because the physical condition of the creature might have changed in that timeframe.

In the reinforcement learning paradigm an agent learns by interacting with its environment. The agent has a goal that it tries to achieve and while striving towards this goal the agent takes actions that affect the environment. The agent senses the environment and thus knows the consequences of its actions, both in terms of detecting the current state of the world, and also by receiving a numerical *reward* in a response to each action. The strategy that the agent follows for deciding how to act in different situations is called the agent's *policy*. The agent seeks to learn the *optimal policy*, that is, the policy that maximizes the overall reward it receives in the long run.

Reinforcement learning problems are typically formulated as Markov Decision Processes (MDPs). Important world properties are used to represent unique states in the MDP. The underlying assumption is that each state represents all properties of the environment that are important for decision making in that state (Sutton and Barto 1998). The actions the agent takes give the agent an immediate reward and transfer it to a (possibly) new state. These state transitions are not necessarily deterministic.

Formulating the Learning Problem

We can formulate the sprinting game as a MDP. The 200-meter long running course is divided into 20 consecutive 10-meter zones. The creature makes a decision at the beginning of each zone how fast to run. It can choose between four progressively faster speeds: walking, jogging, running, and sprinting. Each running speed affects the creature's endurance differently. The creature's endurance is represented with 21 discrete levels, ranging from 0 (minimum endurance) to 20 (maximum endurance). The endurance decreases 2 levels each zone sprinted, decreases 1 level when running, does not change when jogging, and replenishes by 1 level when walking. The endurance can though never exceed twenty nor become less than zero. If the endurance becomes zero the creature can not run anymore, but must instead walk the next zone to replenishing its endurance. Over all, the decision of how fast to run through a zone depends on which zone the creature is in and its remaining level of endurance. This formulation of the running game allows us to represent it as a (cycle-free) Markov-Decisions Process (MDP), as shown in Figure 3.

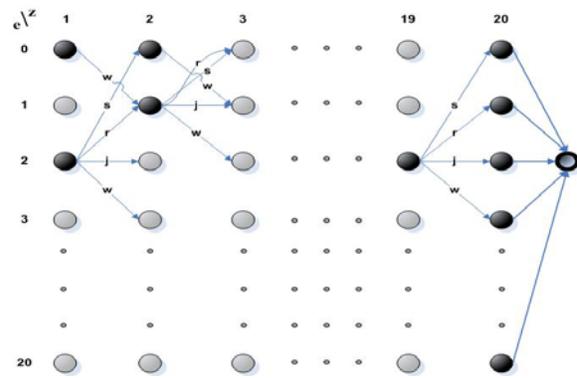


Figure 3. Sprinting game as MDP

Each state is uniquely represented by a (z, e) pair, where z is the current zone (1-20) and e the remaining endurance level (0-20). There are in total 421 different states (21 states for each of the 20 zones plus one final state). The transitions between the states represent the running speed actions, each resulting in us ending in the next zone, although with different remaining endurance. The actions are deterministic and the reward returned by each action is the negation of the time it takes to run the zone at the given speed. The states we draw actions for are shown as dark colored in Figure 3; the light colored states will also have analogous actions although they are not shown. There are 4 possible actions in each state (walk, jog, run, sprint), except in states where endurance is 0 where there is only one possible action (walk).

Depending on the creature characteristics a different MDP might be created. For example, if the creature is overweight the sprinting action might be disabled, or the stamina might drain or replenish at a different phase. Also, the walking or running speed (the rewards) may differ from one creature to the next or by how well the creature is conditioned. The optimal racing strategy may therefore differ from one run to the next, and therefore it may be non-trivial for the user to figure out the correct policy each time.

Q-learning

For learning the optimal policy we use a well-known reinforcement learning algorithm, *Q-learning* (Watkins 1989). The pseudo-code of the algorithm is shown in Fig. 4.

```

Initialize all  $Q(s, a)$ 
Repeat (for all running episode):
  Set  $s$  to be a starting state
  Repeat (for each step in run):
    Choose  $a$  from  $s$  using agent policy
    Take action  $a$ , observe  $r, s'$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
     $s \leftarrow s'$ 
  Until  $s$  is goal state
End
  
```

Figure 4. The Q-learning algorithm

The action-value function $Q(s, a)$ gives for each state the value of each of its actions. This is the function we must approximate. Once the learning is finished an optimal policy is easily achieved simply by greedily choosing in each state the action with the highest $Q(s, a)$ value.

In the beginning the $Q(s, a)$ function is initialized with arbitrary values. Next we execute many learning episodes from a given starting state; in our case a 200-meter sprint from a starting position (any zone 1 state can be a start state, depending on the initial endurance of the creature). For each episode a decision is made in each step (zone) what action to take, that is, how fast to run. We use an ϵ -greedy strategy to pick an action, that is, ϵ part of the time a random action is taken, but otherwise the best action is taken. The best action is the one with the currently highest $Q(s, a)$ value. It is necessary to occasionally take locally non-optimal actions for exploration purposes, otherwise we risk getting stuck in local optima and never finding an improved policy. This is the reason for using the ϵ -greedy strategy. Typically one gradually decreases the exploration rate as more and more episodes are executed. The update rule

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

gradually updates the $Q(s, a)$ action-values until they converge. The constant α is an adjustable step size parameter, and γ is a discount factor of future rewards (not used in our case, that is, set to 1.0). The term r is the current reward, s and a are the current state and action respectively, s' is the next state and a' an action from that state. The $\max_{a'} Q(s', a')$ function returns the value of the currently best action from state s' . The Q-learning algorithm assures convergence to optimal values (in the limit) given that the basic Markov properties hold and the α parameter is set appropriately. For a more detailed discussion of Q-learning see for example (Sutton and Barto 1998).

Improving the learning speed

The standard Q-learning algorithm was able to learn the optimal running strategy in our domain within a couple of thousand episodes. However, there were two problems with this approach:

- The first is that this is far too slow convergence for the algorithm to be practical in our application domain. Even a few hundred episodes would be excessive.
- The second problem is that the player itself has no control over the learning and is simply a passive observer. This could potentially make long learning sessions uninteresting.

To overcome the above problems we designed the game such that the user is allowed to participate in the decision process, effectively taking on the role of a coach by giving feedback. This not only allows the user to take on a more active role in the game, but the player's input can additionally be used to reduce the convergence time of the Q-learning algorithm.

During a run the user observes the creature and can overwrite its decisions. In other words the user can tell it to run either slower or faster. We record the user's (last) preferred action in each MDP state encountered. There are two different ways we can use the user's feedback to speed up the learning.

Imitation Runs

During a many episode training process we periodically rerun a running episode coached by the user, taking the user preferred actions where applicable. These reruns are done in the background and are transparent to the user. Given that the user gave good advice, the creature will get better result sooner and thus starts adapting the good strategy. However, if the advice was not good the creature might get temporarily sidetracked, but then gradually moves away from the user's strategy.

Bonus Reward Points

In this approach extra reward points are awarded to the user-preferred action during regular running episodes. On one hand, this approach has the benefit of not requiring extra reruns. On the other hand, it can be potentially dangerous because we are changing the learning problem. Because of the bonus reward the total reward is not anymore the negation of the total running time. Despite this, given that the feedback is useful, Q-learning can still learn an optimal policy (and that quickly!). Conversely, giving bad user advice can delay the learning process and even possibly prevent optimal policy to be learned. Note, however, that this does reflect a real life scenario where bad advice from a coach is harmful if always followed blindly.

EXPERIMENTS

This section gives the experimental results of a comparison study between standard Q-learning and Q-learning augmented with the enhancements proposed above: *imitation runs* and *bonus reward*. We ran the enhanced Q-learning methods every fifth episode (instead of a regular episode).

At the start of a 200-meter sprint the creature has full endurance (level 20). The time it takes to traverse a zone when walking, jogging, running, and sprinting are 1.5 s., 0.8 s., 0.4s. and 0.1s., respectively. When using an optimal strategy (running all but the last zone where one sprints) the running time for the 200 meters will be 7.7 s. The ϵ parameter is set to 0.15 and α to 1.0. We ran the three different learning approaches until convergence was reached observing how long it took them to find an optimal policy. The performance of the three methods is shown in the following graphs. Each data point represents the running-time average of 10 independent runs (tie-breaks between equally good actions are broken randomly; therefore we base each data point on several runs).

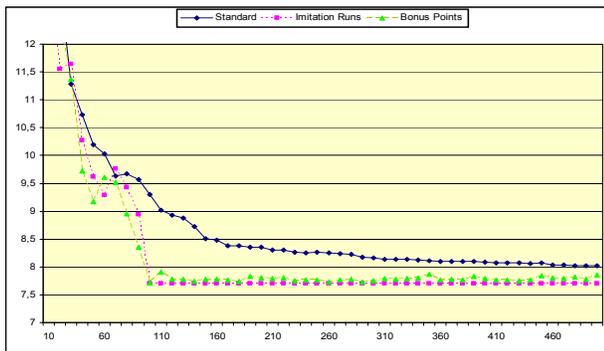


Figure 5. Coach providing helpful advice

Figure 5 shows how the learning progresses when the user provides useful advice (the user performs the optimal running sequence). Both the enhanced Q-learning methods converge faster than the standard method. Within 100 episodes they have discovered the optimal policy. The standard Q-learning needs about 2000 episodes for that. *This is a twenty fold reduction in the number of episodes.* We ran similar experiments with slightly different values for the ϵ and α parameter, but all yielded similar results.

We were also curious to know what happens if the coach provides useless advice. We ran the same set of experiments, but now with a random strategy interleaved every fifth episode. The result is shown in Figure 6. Now all methods perform similar, although the *Imitation Run* method is slightly worse to start with. However, within 300 runs they all have converged to an equally good policy and within about 2000 episodes to an optimal one (same as standard Q-learning). The reason why random advice does not seem to hurt is that Q-learning is a so-called *off-policy* algorithm, that is, it can learn a good policy while following an inferior one.

Finally, we experimented with the case where the user deliberately advises a bad policy (walk all the way). In this case both the enhanced algorithms started out really badly but were eventually able to recover, although it took somewhat longer this time. We did not expect beforehand that the *Bonus reward* approach would be able to recover because the learning problem has been changed. However, because the enhanced Q-version is executed only 20% of the time and the reward bonus is relatively small, the change in total reward is small enough not to affect what is an optimal policy. However, this is not necessarily always the case, and one must be careful using this approach in situations where there are several policies similar in quality.

CONCLUSIONS

In this paper we introduced and experimented with techniques for incorporating user-guided feedback into reinforcement learning. The proposed techniques can drastically reduce the number of training episodes necessary for convergence. They are also robust against the case where the user provides useless feedback. In our game these

techniques contributed to the success of the learning by making the learning algorithm converge much faster. This is

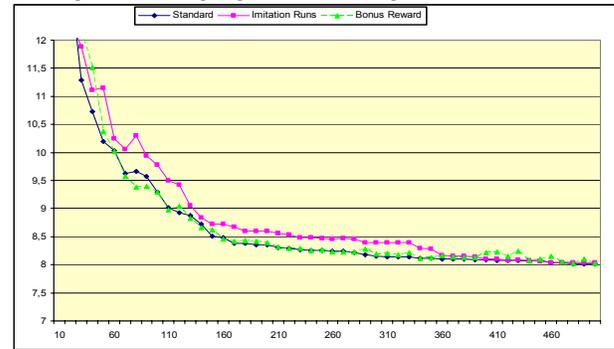


Figure 6. Coach providing random advice

in our view especially important in mobile games not only because of limited computing resources, but also the fact that typical user game-playing sessions on mobile devices are generally much shorter than on other game platforms

As a future work we are investigating other ways of incorporating and reusing user feedback, as well as measuring how well the techniques scale up to larger and more complex learning tasks.

REFERENCES

- Evans, R. 2002. "Varieties of Learning" In *AI Game Programming Wisdom*, Steve Rabin, eds. Charles River Media Inc., Hingham, Massachusetts.
- Manslow, J. 2004. "Using Reinforcement Learning to Solve AI Control Problems" In *AI Game Programming Wisdom 2*, Steve Rabin, eds. Charles River Media Inc., Hingham, M.A., 591-601.
- Spronck, P.; Sprinkhuizen-Kuyper, I.; and Postma E. 2003. "Online Adaptation of Game Opponent AI in Simulation and in Practice". In *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON 2003)*, EUROSIS, Belgium, 93-100.
- Sutton, R. S. and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts.
- Tesauro, G. J. 1994. "TD-Gammon, a self-teaching backgammon program, achieves master-level play." *Neural Computation*, 6(2): 215-219.
- Watkins, C. J. C. H. 1989. "Learning from Delayed Rewards." Ph.D. thesis, Cambridge University.

BIOGRAPHY

The first author is an associate professor at the School of Computer Science, Reykjavik University. He holds a Ph.D. degree in computer science from the University of Alberta, Canada, specializing in AI. Prior to moving to Reykjavik University he was for several years a research scientist with the GAMES group at the University of Alberta working on AI in games. The GAMES group has industrial ties with both Electronic Arts and Bioware Inc. The other co-authors are students at the School of Computer Science, Reykjavik University.

REINFORCEMENT LEARNING OF STRATEGIES FOR SETTLERS OF CATAN

Michael Pfeiffer
Institute for Theoretical Computer Science
Graz University of Technology
A 8010, Graz
Austria
E-mail: pfeiffer@igi.tugraz.at

KEYWORDS

Reinforcement Learning, Self-play, Model Trees, Settlers of Catan

ABSTRACT

In this paper we study the application of machine learning methods in complex computer games. A combination of hierarchical reinforcement learning and simple heuristics is used to learn strategies for the game Settlers of Catan (© 1995 by Kosmos Verlag, Stuttgart) via self-play. Since existing algorithms for function approximation are not well-suited for problems of this size and complexity, we present a novel use of model trees for state-action value prediction in a sophisticated computer game. Furthermore we demonstrate how a-priori knowledge about the game can reduce the learning time and improve the performance of learning virtual agents. We compare several different learning approaches, and it turns out that, despite the simplicity of the architecture, a combination of learning and built-in knowledge yields strategies that are able to challenge and even beat human players in a complex game like this.

INTRODUCTION

Developing programs that play games has always been a major challenge for artificial intelligence research. In many classical board or card games computers have reached the level of human grandmasters, or even outperformed them. On the other hand, the computer game industry has often ignored current developments in AI. Instead they rely mainly on heuristic rules, which require a lot of a-priori knowledge by the AI designer, and are also very inflexible. Machine learning provides a variety of tools to tackle these problems, and since Samuel's pioneering work (Samuel 1959) in the 1950's, researchers have used learning very successfully for games like backgammon, chess or poker (see (Fürnkranz 2001) for an overview). There are, however, only very few commercial computer games that make use of machine learning, and only few people are trying to close the gap between research and industry.

One of the most impressive applications of machine learning in game playing is Tesauro's TD-Gammon (Tesauro 1995), which used *reinforcement learning* to master the game of backgammon through self-play. This means the program uses only very little knowledge from its designer, but rather learns to approximate an evaluation function with an artificial neural network from the outcome of thousands of training games against a copy of itself. After more than one million training games and the inclusion of some backgammon specific knowledge, TD-Gammon reached a playing strength that surpasses most human grandmasters. In this paper we demonstrate how Tesauro's approach can be modified and applied to even more complex games, which are more related to commercial strategy games.

For this purpose we decided to study the game *Settlers of Catan*. Due to the monumental size of the problem, we need to think about ways to shorten the required learning time. We do this by using a hierarchical learning architecture, and by incorporating simple a-priori knowledge. For learning and representing the state-action value function we use model trees (Quinlan 1992), which are better suited for representing discontinuities and local dependencies than neural networks, as used by Tesauro.

REINFORCEMENT LEARNING

Reinforcement learning (*RL*) is used to learn strategies for an agent through interaction with the environment. RL problems are usually formulated as *Markov Decision Processes (MDPs)*, where at every time step t the agent perceives the state of the environment s , chooses and executes an action a , receives a *reward signal* $r(s, a)$, and finds itself in a new state $s' = \delta(s, a)$. The task of the agent is to find a *policy* $\pi(s, a)$, that is, a mapping from states to actions, so as to maximize the *cumulative (discounted) reward* over time. The *discount factor* $\gamma \in [0, 1]$ thereby describes the present value of future rewards.

The quality of a policy is measured by a *value function* $V^\pi(s)$, which is defined as the expected discounted return if we start from state s and follow policy π (Sutton and Barto 1998). An *optimal* policy π^* is then defined as any policy satisfying $V^{\pi^*}(s) \geq V^\pi(s)$ for all policies π and all states s . It is also useful to define the *Q-function* $Q^\pi(s, a)$ as the expected return if we take action a in state s , and thereafter follow policy π . Thus, knowing $Q^* = Q^{\pi^*}$ is sufficient to find an optimal policy, because then we can simply choose $\pi(s) = \arg \max_a Q^*(s)$ at any state s .

The idea behind most RL approaches is *policy iteration*: starting with an arbitrary policy, the agent first evaluates the Q-function of the current policy, and then improves the policy by selecting new actions that are greedy with respect to the current estimation of Q . All this is done while the agent interacts with the environment and receives numerical rewards or punishments. The best-known RL algorithms are *Q-learning* and *SARSA* (Sutton and Barto 1998).

Obviously the policy to which the algorithm converges is highly dependent on the reward signal. For games, the most natural choice of rewards is to give the agent zero reward for intermediate moves, a positive reward (e.g. +1) for winning the game and a negative (e.g. -1) or zero reward for losing. The resulting state value function then approximates the probability of winning from this state, and can be used like an evaluation function for heuristic search methods. By choosing different reward models, the designer may bias the resulting policies, e.g. by also rewarding important subgoals. This may speed up the learning process, but also bears the risk of learning policies that are not optimal in the original reward model.

Self-play is used for learning strong policies in adversarial domains. The agent learns while playing against itself, and therefore has to face increasingly stronger opponents. The major drawback of this method is that without sufficient *exploration* of the state and strategy space, the players only learn to counter a very small set of policies. This problem is particularly severe for deterministic games like chess or Go, while e.g. the dynamics of backgammon appear to be perfect for this co-evolutionary approach (Pollack and Blair 1998). Since Settlers of Catan is also a highly stochastic game, the use of self-play seems justified.

APPROXIMATION WITH MODEL TREES

In the above formulation of RL, a value for $Q(s, a)$ has to be learned for every possible state-action pair, which is impossible for very large, potentially even infinite,

state spaces. One solution to overcome this problem is to define the current state using a finite number of *features*, and to approximate the Q-function as a function of a finite-dimensional parameter vector. Linear and neural network approximators, which are trained via gradient descent, are most frequently used.

Even though these methods have been successfully applied for RL tasks, we found that they have certain drawbacks that make them less suitable for complex game domains. The discrete nature of board games produces many local discontinuities in the value functions. On the other hand, linear or neural network approximators tend to smooth the value function globally. It is also often the case that the importance of certain state features changes in different situations, which is e.g. impossible to represent with linear approximators. As TD-Gammon (Tesauro 1995) has shown, neural networks can cope with all these difficulties, which of course also exist in a game like backgammon. However, the price to pay is an undesirably high number of training games which is needed before reasonable results can be obtained. Therefore it is justified to look for alternatives which are faster at learning local models and discontinuities.

In (Sridharan and Tesauro 2000) it was shown for a smaller scenario, that tree-based approximators have all the desired properties and require less training time than neural networks. So we decided to use *model trees* for function approximation in this experiment. Model trees are similar to decision trees, but instead of predicting discrete classes, they predict real valued functions. Model trees recursively partition the feature-space into regions, by choosing one attribute as a split criterion at every level of the tree. In the leaves of the trees, regression models are trained to predict a numerical value from the features of an instance. Most model tree algorithms use linear models, but in principle any function approximator can be used.

In this paper we used a variant of Quinlan's *M5* algorithm (Quinlan 1992) to learn model trees. This algorithm first grows a tree by selecting splitting criteria so as to minimize the target variable's variance, and then builds linear regression models in the nodes. Finally the tree is pruned, which means that sub-trees are replaced with leaves, as long as the prediction error of the resulting tree does not exceed a certain threshold.

In this context a separate model tree was trained for every action, and the target variable was the Q-value of this action for the current state. The main disadvantage in using model trees for value function approximation is that there is currently no algorithm for online training. To refine the predictions of a model tree, we

must therefore rebuild it from scratch, using not only the new training examples, but also the old ones, that were used for the previous model trees.

We followed the ideas of (Sridharan and Tesauro 2000), in which regression trees (model trees with constant predictions in the leaves) performed very well in a much simpler environment. The goal was to develop an offline RL algorithm, suitable for learning via self-play. First a number of training matches is played, using the current model trees to define the policy. The stored game traces (state, actions and rewards) and the current estimation of the Q-functions are used to calculate the approximate Q-values of the new training examples. Also, SARSA-like updates of the older training examples are calculated, to reach convergence in the self-play process. A new model tree approximation of the Q-functions is built from the whole updated training set, and these trees are then used to play the next set of training games. Even though there is not much experience with using model trees in reinforcement learning, and no convergence results exist, we found that this algorithm yielded promising results for a complex game like this.

SETTLERS OF CATAN

Klaus Teuber's *Settlers of Catan* is probably the most popular modern board game in the German-speaking area. The island of Catan consists of 19 hexagonal land fields of different types, which are randomly placed at the beginning of the game. A typical arrangement of the board is shown in Figure 1.

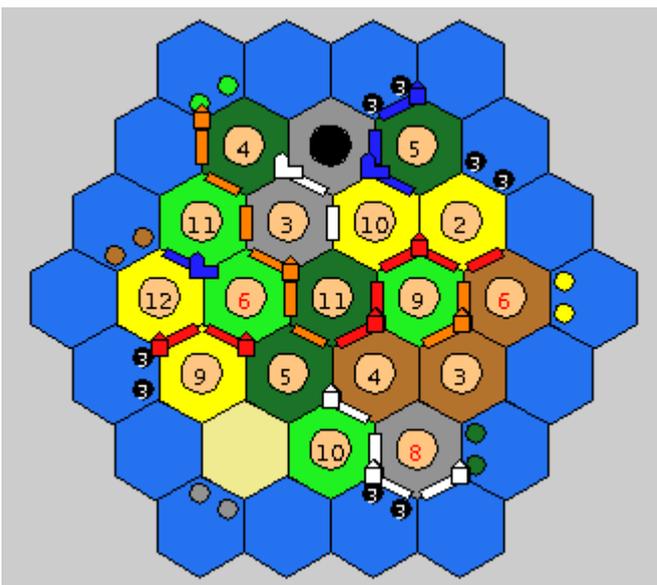


Figure 1: A typical situation in a Settlers of Catan game (screenshot from the Java simulation)

Each field is characterized by the resource it produces and the production probabilities of the field. The island is surrounded by an ocean and nine ports, which do not produce anything, but can be used for trading. Four players are colonizing the island by building roads, settlements and cities on the corners and edges of the hexagonal fields.

The players' settlements produce resources, which can be used to further expand the colony. Since only certain combinations of resources can be used for building, and players usually do not possess all of them, they have to negotiate with their opponents. The players are awarded victory points for their buildings and other special achievements. The first player to reach ten victory points wins the game.

LEARNING STRATEGIES FOR SETTLERS

The complexity of the rules and the dynamics of Settlers of Catan raise questions, that usually do not appear in classical board games. Here e.g. we have to deal with 3 opponents, and each of them can execute an arbitrary number of actions, which makes MiniMax approaches almost impossible. There is an element of chance for resource production, and interaction with the opponents is required for negotiation. Players can choose from a large action-set, and they have to balance long-term and short-term decisions, always depending on the performance of their opponents. All this places Settlers of Catan among the most complex games for which learning a full game strategy via self-play RL has ever been tried.

To make this task feasible, we used a *hierarchical RL* approach, in which the whole strategy was divided into smaller *behaviors*, for which independent policies were learned. The high-level policy first selects a behavior, and this behavior's policy chooses one of several low-level actions. Each policy (high- or low-level) is defined by a set of model trees, which approximates the action values for any given state. Given these approximate action values a controller would choose the next action with some mix of exploitation (i.e., choosing the highest valued action) and exploration.

The high-level policy receives only positive rewards for winning the game, i.e., at every time step the reward for choosing a particular behavior is zero, only at the end of the game the winning agent receives a reward of +1. The other agents also receive a reward at the end of a match, which is scaled between 0 and 1 according to their victory points. The low-level policies are only rewarded for reaching the sub-goal of the behavior, so there is only an indirect connection

between high-level and low-level rewards. The collected data (states, actions and rewards) from the training games is used to train the high-level policy and all low-level policies independently. We used *a-priori* knowledge to map *low-level* actions to *primitive* actions, which are the ones that can actually be executed on the board. E.g. the low-level action `build-settlement` is mapped to a specific board position by a heuristic algorithm. However, the design of this heuristics could be simplified significantly, because the outcome of a move could be evaluated with respect to the learned value functions. This was especially useful for assessing trades with the opponents, because the profitability could naturally be estimated by the resulting change in the value function, without relying on an economic model.

Since good state-representation can significantly improve the performance of RL controllers with value-function approximation, we mainly used *high-level features*, which are calculated to summarize important facts about the current board situation, rather than feeding raw board positions into the learning algorithm.

Four different approaches of hierarchical RL were used in order to learn Settlers of Catan strategies. The *feudal* approach is inspired by the concept of feudal learning (Dayan and Hinton 1993). It lets the high-level policy select a new behavior at every time-step, and tries to learn both the high-level policy and the behaviors simultaneously. In the *module-based* approach the high-level policy changes only when the sub-goal of the last selected behavior is reached. The *heuristic* approach used a hand-coded high-level policy, but learned the low-level behaviors via self-play. And finally the *guided* approach used the same heuristic high-level strategy during learning, but then devised its own high-level policy from this experience.

EXPERIMENTS AND RESULTS

First, 1000 random games were played to provide initial training examples for the model tree learning algorithm described above. Then we ran between 3000 and 8000 training games for each approach, updating the policy after every 1000 games. The policies were evaluated in games against random players, previously learned strategies, and human opponents. Since there is no real benchmark program for Settlers of Catan, testing against a human was the only way to assess the real playing strength of the learned policies.

The learning and testing environment was written entirely in Java (Pfeiffer 2003). Due to the complexity of the game and the huge amount of training data

(about 1 GB per 1000 games), the learning time was very high. Playing 1000 games in our simulation environment took about one day on a 1.5 GHz dual-processor machine. Another 20 to 24 hours were needed to train the model trees on the same computer, so the number of training matches that could be played was limited.

The results of the *feudal* approach were not satisfactory, mainly because the agent could not learn to stick to one high-level strategy when it was selected. Overfitting in the learning of model trees also occurred, which resulted in huge trees and even caused a decrease of performance for larger training sets.

The *module-based* approach outperformed the feudal strategies, but still it was not competitive in games against humans. We found that the low-level behaviors made almost the same choices, that a human would have made in the same situation with that goal in mind. The high-level strategy however was still very poor. We also could not avoid overfitting, even though this time the trees were pruned more aggressively.

Using a simple *heuristic* high-level policy in combination with learned behaviors, the virtual agents for the first time were able to beat a human opponent. This *heuristic* method was a significant improvement over the previous approaches. The amount of prior knowledge used for this method is still very small, compared to "classical" rule-based solutions. The high-level behavior plays a role in the selection of target positions on the board, e.g. for placing settlements and roads. This knowledge alone made the heuristics outperform most of the previous approaches, even with random low-level action selection, which can be seen in Figure 2.

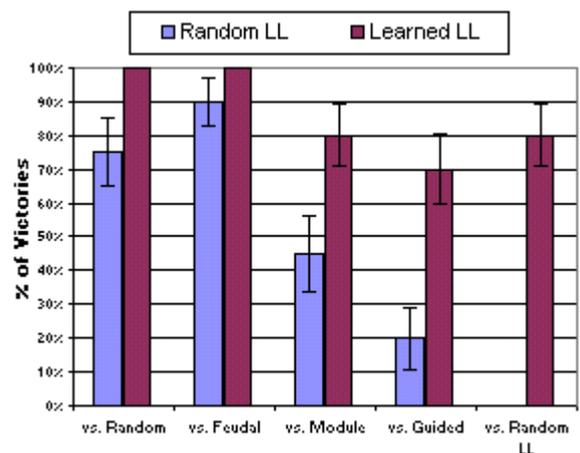


Figure 2: Performance of heuristic high-level strategies, using random or learned action selection in the low-level (LL), against other approaches (during 20 games)

Figure 2 also clearly indicates that learning is responsible for the greatest improvements in playing strength, because the trained policy wins most of the games against players who use random low-level policies, as well as against all previous approaches.

We also conducted a last experiment, in which the agents were *guided* by the heuristic high-level policy during learning. As expected, these agents did not reach the level of the heuristic agents, but their performance against humans is slightly better than that of the module-based strategies. The sub-optimal high-level strategy, caused by a lack of negative training examples (i.e., wrong choices which cause punishment instead of reward), is probably the main reason for these results.

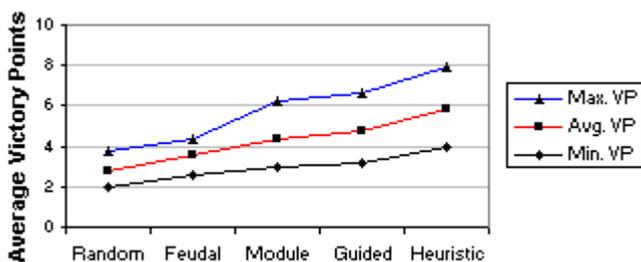


Figure 3: Performance of different strategies against a human opponent (during 10 games)

To obtain a rough estimate of the real playing strength of the different strategies, the author, who considers himself a Settlers of Catan expert, played 10 matches against the best strategies from each approach. The results of the matches are shown in Figure 3. There were always three computer players playing against the human, so the three lines show the average number of victory points of the worst, average and best artificial player. Note that a score of 10 would mean a win in every game, so the maximum of 8 points, achieved by the best *heuristic* player, indicates a pretty good performance. Actually the *heuristic* player managed to win 2 out of 10 games against the author, and at least one of the agents came close to winning in every match. Demonstration matches against other human players of various playing strengths have confirmed the competitiveness of the best learned strategies, but these results have not been included in the above statistics.

Summarizing, we can say that although the agents did not learn to play at a grandmaster-level, like in TD-Gammon, the results are encouraging, considering the complexity of this game.

CONCLUSION

In this paper we have demonstrated how reinforcement learning can be used to learn strategies for a large-scale

board game. A combination of new and well-tried learning methods was used to make this problem feasible. We see this as a first step to apply advanced machine learning techniques for even more complex computer games. This research has shown that the combination of learning with prior knowledge can be a promising way to improve the performance, and ultimately also the human-like adaptiveness of agents in computer games.

ACKNOWLEDGEMENTS

The work was partially supported by the Austrian Science Fund FWF, project # P17229-N04, and PASCAL, project # IST2002-506778, of the EU.

REFERENCES

- Dayan, P. and Hinton, G.E. 1993. "Feudal Reinforcement Learning" In *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, San Mateo, CA, 271-278
- Fürnkranz, J. 2001. "Machine Learning in Games: A Survey." In *Machines that Learn to play Games*, J. Fürnkranz and M. Kubat, eds. Nova Scientific Pub., Huntington, N.Y., 11-59
- Pfeiffer, M. 2003. "*Machine Learning Applications in Computer Games*." MSc Thesis, Graz University of Technology
- Pollack, J.B. and Blair, A.D. 1998. "Co-Evolution in the successful Learning of Backgammon Strategy." *Machine Learning* 32, no. 1, 225-240.
- Quinlan, J.R. 1992. "Learning with Continuous Classes." In *Proceedings of the 5th Australian Joint Conference on AI*. World Scientific, Singapore, 343-348.
- Samuel, A.L. 1959. "Some Studies in Machine Learning using the Game of Checkers." *IBM Journal on Research and Development* 3, 211-229.
- Sridharan, M. and Tesauro, G.J. 2000. "Multi-agent Q-Learning and Regression Trees for Automated Pricing Decisions." In *Proceedings of the 17th International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 927-934.
- Sutton, R.S. and Barto, A.G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA
- Tesauro, G.J. 1995. "Temporal Difference Learning and TD-Gammon." *Communications of the ACM* 38, 58-68.

IMPROVING ADAPTIVE GAME AI WITH EVOLUTIONARY LEARNING

Marc Ponsen
Lehigh University / Computer Science & Engineering
19 Memorial Drive West
Bethlehem, PA 18015-3084 USA
mjp304@lehigh.edu

Pieter Spronck
Maastricht University / IKAT
P.O. Box 616, NL-6200 MD Maastricht,
The Netherlands
p.spronck@cs.unimaas.nl

KEYWORDS

Games, artificial intelligence, real-time strategy, dynamic scripting, evolutionary algorithms.

ABSTRACT

Game AI is defined as the decision-making process of computer-controlled opponents in computer games. Adaptive game AI can improve the entertainment provided by computer games, by allowing the computer-controlled opponents to fix automatically weaknesses in the game AI, and to respond to changes in human-player tactics online, i.e., during gameplay. Successful adaptive game AI is based invariably on domain knowledge of the game it is used in. Dynamic scripting is an algorithm that implements adaptive game AI. The domain knowledge used by dynamic scripting is stored in a rulebase with manually designed rules. In this paper we propose the use of an offline evolutionary algorithm to enhance the performance of adaptive game AI, by evolving new domain knowledge. We empirically validate our proposal, using dynamic scripting as adaptive game AI in a real-time-strategy (RTS) game, in three steps: (1) we implement and test dynamic scripting in an RTS game; (2) we use an offline evolutionary algorithm to evolve new tactics that are able to deal with optimised tactics, which dynamic scripting cannot defeat using its original rulebase; (3) we translate the evolved tactics to rules in the rulebase, and test dynamic scripting with the revised rulebase. The empirical validation shows that the revised rulebase yields a significantly improved performance of dynamic scripting compared to the original rulebase. We therefore conclude that offline evolutionary learning can be used to improve the performance of adaptive game AI.

INTRODUCTION

Traditionally, commercial game developers spend most of their resources on improving a game's graphics. However, in recent years, game developers have begun to compete with each other by providing a more challenging gaming experience (Rabin 2004). For most games, challenging gameplay is equivalent to having high-quality game AI (Laird 2000). Game AI is defined as the decision-making process of computer-controlled opponents. Even in state-of-the-art games, game AI is, in general, of inferior quality (Schaeffer 2001, Laird 2001, Gold 2004). It tends to be predictable, and often contains weaknesses that human players can exploit.

Adaptive game AI, which implies the online (i.e., during gameplay) adaptation of the behaviour of computer-controlled opponents, has the potential to increase the quality of game AI. It has been widely disregarded by game developers, because online learning tends to be slow, and can lead to undesired behaviour (Manslow 2002). However, academic game AI researchers have shown that successful adaptive game AI is feasible (Demasi and Cruz 2002,

Johnson 2004, Spronck, Sprinkhuizen-Kuyper and Postma 2004a).

To ensure the efficiency and reliability of adaptive game AI, it must incorporate a great amount of prior domain knowledge (Manslow 2002, Spronck, Sprinkhuizen-Kuyper and Postma 2004b). However, if the incorporated domain knowledge is incorrect or insufficient, adaptive game AI will not be able to generate satisfying results. In this paper we propose an evolutionary algorithm to improve the quality of the domain knowledge used by adaptive game AI. We empirically validate our proposal by testing it on an adaptive game AI technique called "dynamic scripting", used in a real-time strategy (RTS) game.

The outline of the remainder of the paper is as follows. First, we discuss RTS games, and the game environment selected for the experiments. Then, we discuss the implementation of dynamic scripting for RTS games, followed by a discussion of the implementation of an evolutionary algorithm that generates successful tactics for RTS games. The achieved results are used to show how the tactics discovered with an evolutionary algorithm can be employed to improve the original dynamic scripting implementation. Finally, we draw conclusions and indicate future work.

REAL-TIME-STRATEGY GAMES

RTS games are simple military simulations (war games) that require the player to control armies (consisting of different types of units), and defeat all opposing forces. In most RTS games, the key to winning lies in efficiently collecting and managing resources, and appropriately distributing these resources over the various game elements. Typical game elements in RTS games include the construction of buildings, the research of new technologies, and combat.

Game AI in RTS games determines the tactics of the armies controlled by the computer, including the management of resources. Game AI in RTS games is particularly challenging for game developers, because of two reasons: (1) RTS games are complex, i.e., a wide variety of tactics can be employed, and (2) decisions have to be made in real-time, i.e., under severe time constraints. RTS games have been called "an ideal test-bed for real-time AI research" (Buro 2003).

For our experiments, we selected the RTS game WARGUS, with STRATAGUS as its underlying engine. STRATAGUS is an open-source engine for building RTS games. WARGUS implements a clone of the highly popular RTS game WARCRAFT II. While the graphics of WARGUS are not up-to-date with today's standards, its gameplay can still be considered state-of-the-art. Figure 1 illustrates WARGUS. The figure shows a battle between an army of "orcs", which



Figure 1: Screenshot of a battle in WARGUS.

approach from the bottom right, and an army of “humans”, which attempt to defend a base consisting of several buildings.

ADAPTIVE GAME AI IN RTS GAMES

Game AI for complex games, such as RTS games, is mostly defined in scripts, i.e., lists of rules that are executed sequentially (Tozour 2002). Because the scripts tend to be long and complex (Brockington and Darrah 2002), they are likely to contain weaknesses, which the human player can exploit. Because scripts are static they cannot adapt to overcome these exploits. Spronck *et al.* (2004a) designed a novel technique called “dynamic scripting” that realises the online adaptation of scripted opponent AI. Experiments have shown that the dynamic scripting technique can be successfully incorporated in commercial Computer RolePlaying Games (CRPGs) (Spronck *et al.* 2004a, 2004b).

Because the game AI for WARGUS is defined in scripts, dynamic scripting should also be applicable to WARGUS. However, because of the differences between RTS games and CRPGs, the original dynamic scripting implementation cannot be transferred to RTS games unchanged. In this section a dynamic scripting implementation for the game AI in RTS games is designed and evaluated. The basics of dynamic scripting are explained first. Then, we highlight the changes made to dynamic scripting to apply it to RTS games, and discuss the implementation of dynamic scripting in WARGUS. The implementation is evaluated, and the evaluation results are discussed.

Dynamic Scripting

Dynamic scripting is an online learning technique for commercial computer games, inspired by reinforcement learning (Russel and Norvig 1995). Dynamic scripting generates scripted opponents on the fly by extracting rules from an adaptive rulebase. The rules in the rulebase are manually designed using domain-specific knowledge. The

probability that a rule is selected for a script is proportional to a weight value that is associated with each rule, i.e., rules with larger weights have a higher probability of being selected. After every encounter between opponents, the weights of rules employed during gameplay are increased when having a positive contribution to the outcome, and decreased when having a negative contribution. The size of the weight changes is determined by a weight-update function. To keep the sum of all weight values in a rulebase constant, weight changes are executed through a redistribution of all weights in the rulebase. Through the process of punishments and rewards, dynamic scripting gradually adapts to the human player. For CRPGs, it has been shown that dynamic scripting is fast, effective, robust and efficient (Spronck *et al.* 2004a).

Dynamic Scripting for RTS games

Our design of dynamic scripting for RTS games has two differences with dynamic scripting for CRPGs. The first difference is that, while dynamic scripting for CRPGs employs different rulebases for different opponent types in the game (Spronck *et al.* 2004a), our RTS implementation of dynamic scripting employs different rulebases for the different states of the game. The reason for this deviation from the CRPG implementation of dynamic scripting is that, in contrast with CRPGs, the tactics that can be used in an RTS game mainly depend on the availability of different unit types and technologies. For instance, attacking with weak units might be the only viable choice in early game states, while in later game states, when strong units are available, usually weak units will have become useless.

The second difference is that, while dynamic scripting for CRPGs executes weight updates based on an evaluation of a fight, our RTS implementation of dynamic scripting executes weight updates based on both an evaluation of the performance of the game AI during the whole game (called the “overall fitness”), and on an evaluation of the performance of the game AI between state changes (called

the “state fitness”). As such, the weight-update function is based on the state fitness, combined with the overall fitness. The use of both evaluations for the weight-updates increases the efficiency of the learning mechanism (Manslow 2004).

Dynamic Scripting in WARGUS

We implemented the dynamic scripting process in WARGUS as follows. Dynamic scripting starts by randomly selecting rules for the first state. When a rule is selected that spawns a state change, from that point on rules will be selected for the new state. To avoid monotone behaviour, we restricted each rule to be selected only once for each state. At the end of the scripts, a loop is implemented that initiates continuous attacks against the enemy.

Because in WARGUS the available buildings determine the unit types that can be built and technologies that can be researched, we decided to distinguish game states according to the type of buildings possessed. Consequently, state

changes are spawned by rules that comprise the creation of new buildings. The twenty states for WARGUS, and the possible state changes, are illustrated in Figure 2.

We allowed a maximum of 100 rules per script. The rulebases for each of the states contained between 21 and 42 rules. The rules can be divided in four basic categories: (1) build rules (for constructing buildings), (2) research rules (for acquiring new technologies), (3) economy rules (for gathering resources), and (4) combat rules (for military activities). To design the rules, we incorporated domain knowledge acquired from strategy guides for WARCRAFT II.

The ‘overall fitness’ function F for player d controlled by dynamic scripting (henceforth called the “dynamic player”) yields a value in the range $[0,1]$. It is defined as:

$$F = \begin{cases} \min\left(\frac{S_d}{S_d + S_o}, b\right) & \{d \text{ lost}\} \\ \max\left(b, \frac{S_d}{S_d + S_o}\right) & \{d \text{ won}\} \end{cases} \quad (1)$$

In equation (1), S_d represents the score for the dynamic player, S_o represents the score for the dynamic player’s opponent, and $b \in [0,1]$ is the break-even point. At the break-even point, weights remain unchanged.

For the dynamic player, the state fitness F_i for state i is defined as:

$$F_i = \begin{cases} \frac{S_{d,i}}{S_{d,i} + S_{o,i}} & \{i = 1\} \\ \frac{S_{d,i}}{S_{d,i} + S_{o,i}} - \frac{S_{d,i-1}}{S_{d,i-1} + S_{o,i-1}} & \{i > 1\} \end{cases} \quad (2)$$

In equation (2), $S_{d,x}$ represents the score of the dynamic player after state x , and $S_{o,x}$ represents the score of the dynamic player’s opponent after state x .

The score function is domain-dependent, and should successfully reflect the relative strength of the two opposing players in the game. For WARGUS, we defined the score S_x for player x as:

$$S_x = 0.7M_x + 0.3B_x \quad (3)$$

In equation (3), M_x represents the military points for player x , i.e. the number of points awarded for killing units and destruction of buildings, and B_x represents the building points for player x , i.e. the number of points awarded for training armies and constructing buildings.

After each game, the weights of all rules employed are updated. The weight-update function translates the fitness functions into weight adaptations for the rules in the script. The weight-update function W for the dynamic player is defined as:

$$W = \begin{cases} \max\left(W_{\min}, W_{org} - 0.3 \frac{b-F}{b} P - 0.7 \frac{b-F_i}{b} P\right) & \{F < b\} \\ \min\left(W_{org} + 0.3 \frac{F-b}{1-b} R + 0.7 \frac{F_i-b}{1-b} R, W_{\max}\right) & \{F \geq b\} \end{cases} \quad (4)$$

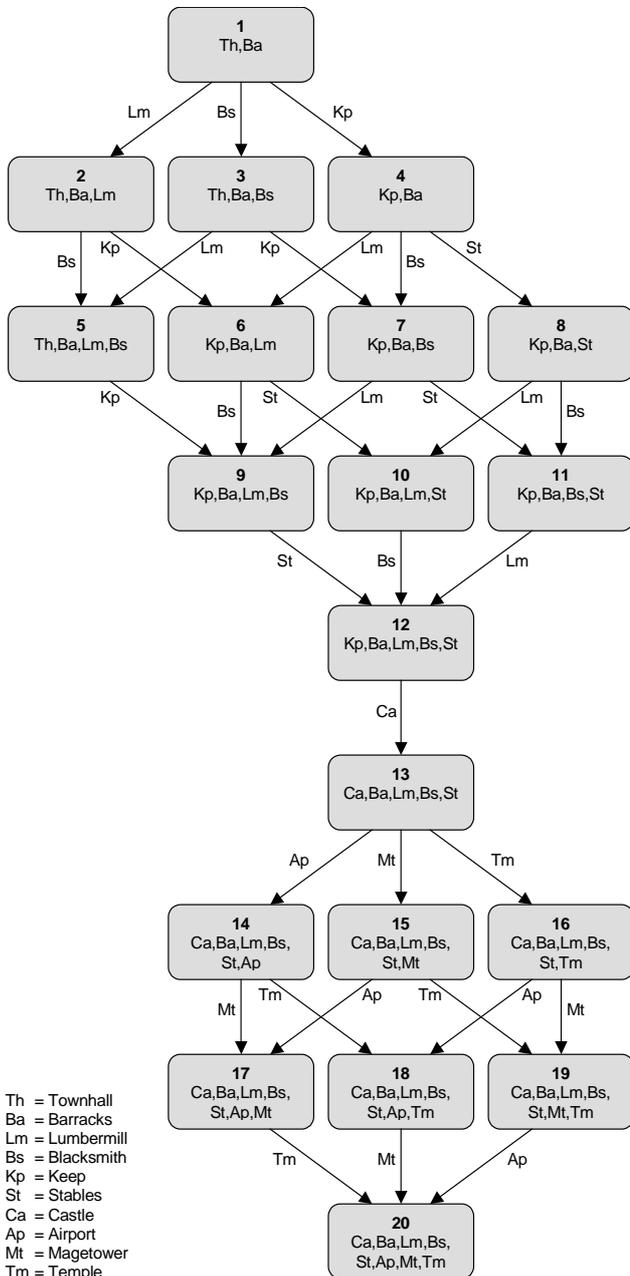


Figure 2: Game states in WARGUS.

In equation (4), W is the new weight value, W_{org} is the original weight value, P is the maximum penalty, R is the maximum reward, W_{max} is the maximum weight value, W_{min} is the minimum weight value, F is the overall fitness of the dynamic player, F_i is the state fitness for the dynamic player in state i , and b is the break-even point. The equation indicates that we prioritise state performance over overall performance. The reason is that, even if a game is lost, we wish to prevent rules from being punished (too much) in states where performance is successful. In our simulation we set P to 175, R to 200, W_{max} to 1250, W_{min} to 25 and b to 0.5.

Evaluating Dynamic Scripting in WARGUS

We evaluated the performance of dynamic scripting for RTS games in WARGUS, by letting the computer play the game against itself. One of the two opposing players was controlled by dynamic scripting (the dynamic player), and the other was controlled by a static script (the static player). Each game lasted until one of the players was defeated, or until a certain period of time had elapsed. If the game ended due to the time restriction (which was rarely the case), the player with the highest score was considered to have won. After the game, the rulebases were adapted, and the next game was started, using the adapted rulebases. A sequence of 100 games constituted one test. We tested four different tactics for the static player:

1. **Small Balanced Land Attack (SBLA):** The SBLA is a tactic that focuses on land combat, keeping a balance between offensive actions, defensive actions, and research. The SBLA is applied on a small map. Games on a small map are usually decided swiftly, with fierce battles between weak armies.
2. **Large Balanced Land Attack (LBLA):** The LBLA is similar to the SBLA, but applied on a large map. A large map allows for a slower-paced game, with long-lasting battles between strong armies.
3. **Soldier's Rush (SR):** The soldier's rush aims at overwhelming the opponent with cheap offensive units in an early state of the game. Since the soldier's rush works best in fast games, we tested it on a small map.
4. **Knight's Rush (KR):** The knight's rush aims at quick technological advancement, launching large offences as soon as strong units are available. Since the knight's rush works best in slower-paced games, we tested it on a large map.

To quantify the relative performance of the dynamic player against the static player, we used the 'randomization turning point' (RTP). The RTP is measured as follows. After each game, a randomization test (Cohen 1995; pp. 168–170) is performed using the fitness values over the last ten games, with the null hypothesis that both players are equally strong. The dynamic player is said to outperform the static player if the randomization test concludes that the null hypothesis can be rejected with 90% probability in favour of the dynamic player. The RTP is the number of the first game in which the dynamic player outperforms the static player. A low value for the RTP indicates good efficiency of dynamic scripting.

If the player controlled by dynamic scripting is unable to statistically outperform the static player within 100 games,

the test is aborted. For the SBLA we ran 31 tests. For the LBLA we ran 21 tests. For both the SR and KR, we ran 10 tests.

Results

The results of the evaluation of dynamic scripting in WARGUS are displayed in Table 1. From left to right, the table displays (1) the tactic used by the static player, (2) the number of tests, (3) the lowest RTP found, (4) the highest RTP found, (5) the average RTP, (6) the median RTP, (7) the number of tests that did not find an RTP within 100 games, and (8) the average number of games won out of 100.

| Tactic | Tests | Low | High | Avg | Med | >100 | Won |
|--------|-------|-----|------|-----|-----|------|------|
| SBLA | 31 | 18 | 99 | 50 | 39 | 0 | 59.3 |
| LBLA | 21 | 19 | 79 | 49 | 47 | 0 | 60.2 |
| SR | 10 | | | | | 10 | 1.2 |
| KR | 10 | | | | | 10 | 2.3 |

Table 1: Evaluation results of dynamic scripting in RTS games.

From the low values for the RTPs for both the SBLA and the LBLA, we can conclude that the dynamic player efficiently adapts to these two tactics. Therefore, we conclude that dynamic scripting in our implementation can be applied successfully to RTS games.

However, the dynamic player was unable to adapt to the soldier's rush and the knight's rush within 100 games. As the rightmost column in Table 1 shows, the dynamic player only won approximately 1 out of 100 games against the soldier's rush, and 1 out of 50 games against the knight's rush. The reason for the inferior performance of the dynamic player against the two rush tactics is twofold, namely (1) the rush tactics are optimised, in the sense that it is very hard to design game AI that is able to deal with them, and (2) the rulebase does not contain the appropriate knowledge to easily design game AI that is able to deal with the rush tactics.

The remainder of this paper investigates how offline evolutionary learning can be used to improve the rulebase to deal with optimised tactics.

EVOLUTIONARY TACTICS

In this section we empirically investigate to what extent an evolutionary algorithm can be used to search for effective tactics for RTS games. Our goal is to use offline evolutionary learning to design tactics that can be used to defeat the two optimised tactics described in the previous section, the soldier's rush and the knight's rush. In the following subsections we describe the procedure used, the encoding of the chromosome, the fitness function, the genetic operators, and the achieved results.

Experimental Procedure

We designed an evolutionary algorithm that evolves new tactics to be used in WARGUS against a static player using the soldier's rush and the knight's rush tactics. The evolutionary algorithm uses a population of size 50, representing sample solutions (i.e., game AI scripts). Relatively successful solutions (as determined by a fitness

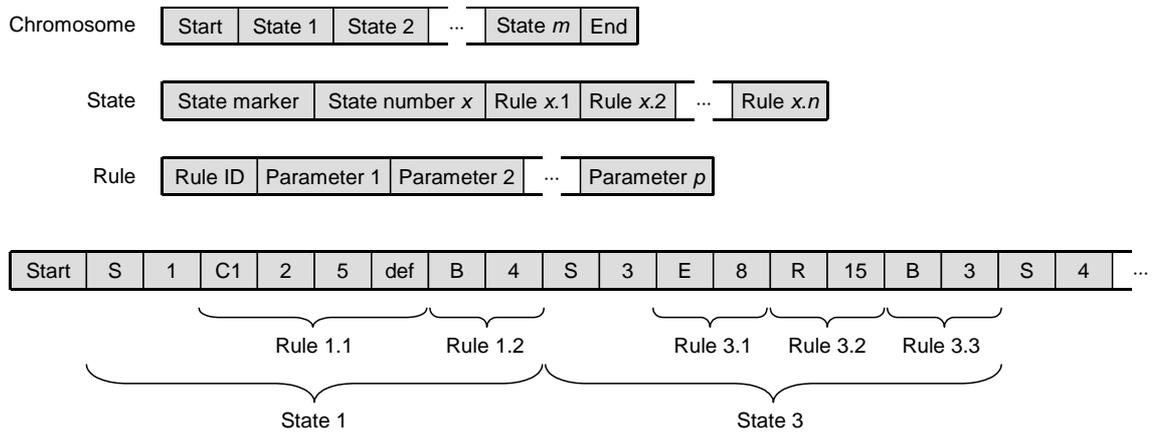


Figure 3: Design of a chromosome to store game AI for WARGUS.

function) are allowed to breed. To select parent chromosomes for breeding, we used size-3 tournament selection (Buckland 2004). This method prevents early convergence and is computationally fast. Newly generated chromosomes replace existing solutions in the population, using size-3 crowding (Goldberg 1989).

Our goal is to generate a chromosome with a fitness exceeding a target value. When such a chromosome is found, the evolution process ends. This is the fitness-stop criterion. We set the target value to 0.75 against the soldier's rush, and to 0.7 against the knight's rush. Since there is no guarantee that a solution exceeding the target value will be found, the evolution process also ends after it has generated a maximum number of solutions. This is the run-stop criterion. We set the maximum number of solutions to 250. The choices for the fitness-stop and run-stop criteria were determined during preliminary experiments.

Encoding

The evolutionary algorithm works with a population of chromosomes. In the present context, a chromosome represents a game-AI script. To encode a game-AI script for WARGUS, each gene in the chromosome represents one rule. Four different gene types are distinguished, corresponding to the four basic rule categories mentioned in the previous section, namely (1) build genes, (2) research genes, (3) economy genes, and (4) combat genes. Each gene consists of a rule ID that indicates the type of gene (B, R, E and C, respectively), followed by values for the parameters needed by the gene. Of the combat gene, there are actually twenty variations, one for each possible state, each with its own parameters. The genes are grouped by states. A separate marker (S), followed by the state number, indicates the start of a state.

The chromosome design is illustrated in Figure 3. A schematic representation of the chromosome, divided into states, is shown at the top. Below it, a schematic representation of one state in the chromosome is shown, consisting of a state marker and a series of rule genes. Rule genes are identified by the number of the state for which they occur, followed by a period, followed by a sequence number. Below the state representation, a schematic representation of one rule is shown. At the bottom, part of an example chromosome is shown. Chromosomes for the initial population are generated randomly.

By taking into account state changes spawned by build genes, it is ensured that only legal game AI scripts are created. A more detailed description of the chromosome design can be found in (Ponsen 2004).

Fitness Function

To measure the success of a game AI script represented by a chromosome, the following fitness function F for the dynamic player d , yielding a value in the range [0,1], is defined:

$$F = \begin{cases} \min\left(\frac{GC}{EC} \cdot \frac{M_d}{M_d + M_o}, b\right) & \{d \text{ lost}\} \\ \max\left(b, \frac{M_d}{M_d + M_o}\right) & \{d \text{ won}\} \end{cases} \quad (5)$$

In equation (5), M_d represents the military points for the dynamic player, M_o represents the military points for the dynamic player's opponent, and b is the break-even point. GC represents the game cycle, i.e., the time it took before the game is lost by one of the players. EC represents the end cycle, i.e. the longest time a game is allowed to continue. When a game reaches the end cycle and neither army has been completely defeated, scores at that time are measured and the game is aborted. The factor GC/EC ensures that losing solutions that play a long game are awarded higher fitness scores than losing solutions that play a short game.

Genetic Operators

To breed new chromosomes, four genetic operators were implemented. By design, all four genetic operators ensure that a child chromosome always represents a legal game-AI script. Parent chromosomes are selected with a chance corresponding to their fitness values.

The genetic operators take into account "activated" genes. An activated gene is a gene that represents a rule that was executed during the fitness determination. Non-activated genes can be considered irrelevant to the game-AI script the chromosome represents. If a genetic operator produces a child chromosome that is equal to a parent chromosome for all activated genes, the child is rejected and a new child is generated.

1. **State Crossover** selects two parents, and copies states from either parent to the child chromosome. State crossover is controlled by “matching states”. A matching state is a state that exists in both parent chromosomes. Figure 2 makes evident that, for WARGUS, there are always at least four matching states, namely state 1, state 12, state 13, and state 20. State crossover will only be used when there are least three matching states with activated genes. A child chromosome is created as follows. States are copied from the first parent chromosome to the child chromosome, starting at state 1 and working down the chromosome. When there is a state change to a matching state, there is a 50% probability that from that point on, the role of the two parents is switched, and states are copied from the second parent. When the next state change to a matching state is encountered, again a switch between the parents can occur. This continues until the last state has been copied.
2. **Rule Replace Mutation** selects one parent, and replaces economy, research or combat rules with a 25% probability. Building rules are excluded, both for and as replacement, because these could spawn a state change and thus could possibly corrupt the chromosome.
3. **Rule Biased Mutation** selects one parent and mutates parameters for existing economy or combat rules with a 50% chance. The mutations are executed by adding a random integer value in the range [-5,5].
4. **Randomization** generates a random new chromosome.

Randomization had a 10% chance of being selected during evolution. The other genetic operators had a 30% chance.

Results

The results of ten tests of the evolutionary algorithm against each of the two optimised tactics are shown in Table 2. From left to right, the columns show (1) the tactic used by the static player, (2) the number of tests, (3) the lowest fitness value found, (4) the highest fitness value found, (5) the average fitness value, and (6) the number of tests that ended because of the run-stop criterion.

| Tactic | Tests | Low | High | Avg | >250 |
|--------|-------|------|------|------|------|
| SR | 10 | 0.73 | 0.85 | 0.78 | 2 |
| KR | 10 | 0.71 | 0.84 | 0.75 | 0 |

Table 2: Evolutionary algorithm results.

The table shows surprisingly high average, highest, and even lowest solution-fitness values. Therefore, it may be concluded that offline adaptive game AI was successful in rapidly discovering game-AI scripts able to defeat both rush tactics used by the static player.

IMPROVING ADAPTIVE AI

In the first experiment, we discovered that our original implementation of dynamic scripting did not achieve satisfying results against the two rush tactics. In the previous section we evolved new tactics that were able to defeat the two rush tactics. In the present section we discuss how the evolved tactics can be used to improve the rulebases employed by dynamic scripting, to enable it to deal with the

rush tactics with more success. First, we discuss observations on the evolved tactics. Then, we discuss the translation of the evolved tactics to rulebase improvements. Finally, we evaluate of the new rulebases by repeating the first experiment with the new rulebases.

Observations on the Evolved Tactics

About the solutions evolved against the soldier’s rush, the following observations were made. The soldier’s rush is used on a small map. As is usual for a small map, the game played by the solutions was always short. Most solutions included only two states with activated genes. Basically, we found that all ten solutions counter the soldier’s rush with a soldier’s rush of their own. In eight out of ten solutions, the solutions included building a “blacksmith” very early in the game, which allows the research of weapon and armour upgrades. Then, the solutions selected at least two out of the three possible research advancements, after which large attack forces were created. These eight solutions succeeded because they ensure their soldiers are quickly upgraded to be very effective, before they attack. The remaining two solutions overwhelmed the static player with sheer numbers.

About the solutions evolved against the knight’s rush, the following observations were made. The knight’s rush is used on a large map, which enticed longer games. On average, for each solution five or six states were activated. Against the knight’s rush, all solutions included training a large number of “workers” to be able to expand quickly. They also included boosting the economy by exploiting additional resource sites after setting up defences.

Almost all solutions evolved against the knight’s rush worked towards the goal of quickly creating advanced military units, in particular “knights”. Seven out of ten solutions achieved this goal by employing a specific building order, namely a “blacksmith”, followed by a “lumbermill”, followed by a “keep”, followed by “stables”. Two out of ten solutions preferred a building order that reached state 11 as quickly as possible (see Figure 2). State 11 is the first state that allows the building of knights.

Surprisingly, in several solutions against the knight’s rush, the game AI employed many “catapults”. WARCRAFT II strategy guides generally consider catapults to be inferior military units, because of their high costs and considerable vulnerability. A possible explanation for the successful use of catapults by the evolutionary game AI is that, with their high damaging abilities and large range, they are particularly effective against tightly packed armies, such as groups of knights.

Improving the Rulebase for Dynamic Scripting

Based on our observations we decided to create four new rules for the rulebases, and to (slightly) change the parameters for several existing combat rules.

The first new rule was designed to be able to deal with the soldier’s rush. The rule contained the pattern that was observed in most of the tactics evolved against the soldier’s rush, namely a combination of the building of a “blacksmith”, followed by the research of several upgrades, followed by the creation of a large offensive force.

The second rule was designed to be able to deal with the knight’s rush. Against the knight’s rush, almost all evolved

solutions aimed at creating advanced military units quickly. The new rule checks whether it is possible to reach a state that allows the creation of advanced military units, by constructing one new building. If such is possible, the rule constructs that building, and creates an offensive force consisting of the advanced military units.

The third rule was aimed at boosting the economy by exploiting additional resource sites. The original rulebases contained a rule to this end, but this rule was invariably assigned a low weight. In the evolved solutions we discovered that exploitation of additional resource sites only occurred after a defensive force was built. The new rule acknowledged this by preparing the exploitation of additional resource sites with the building of a defensive army.

The fourth rule was a straightforward translation of the best solution found against the knight's rush. Simply all activated genes for each state were translated and combined in one rule, and stored in the rulebase corresponding to the state.

To keep the total number of rules constant, the new rules replaced existing rules, namely rules that always ended up with low weights. Besides the creation of the four new rules, small changes were made to some of the existing combat rules, by changing the parameters to increase the number of units of types clearly preferred by the solutions, and to decrease the number of units of types avoided by the solutions. Through these changes, the use of "catapults" was encouraged. More details on the original and revised rulebases can be found in (Ponsen 2004).

Evaluating the Improved Rule-base in Wargus

We repeated the first experiment, but with dynamic scripting using the new rulebases, and with the values of the maximum reward and maximum penalty both set to 400, to allow dynamic scripting to reach the boundaries of the weight values faster. Table 3 summarises the achieved results. The columns in Table 3 are equal to those in Table 1.

| Tactic | Tests | Low | High | Avg | Med | >100 | Won |
|--------|-------|-----|------|-----|-----|------|------|
| SBLA | 11 | 10 | 34 | 19 | 14 | 0 | 72.5 |
| LBLA | 11 | 10 | 61 | 24 | 26 | 0 | 66.4 |
| SR | 10 | | | | | 10 | 27.5 |
| KR | 10 | | | | | 10 | 10.1 |

Table 3: Evaluation results of dynamic scripting in RTS games using improved rulebases.

A comparison of Table 1 and Table 3 shows that the performance of dynamic scripting is considerably improved with the new rulebases. Against the two balanced tactics, SBLA and LBLA, the average RTP is reduced by more than 50%. Against the two optimised tactics, the soldier's rush and the knight's rush, the number of games won out of 100 has increased enormously. Since we observed that dynamic scripting assigned the new rules large weights, the improved performance can be attributed to the new rules.

Note that, despite the improvements, dynamic scripting is still unable to statistically outperform the two rush tactics. The explanation is as follows. The two rush tactics are 'super-tactics', that can only be defeated by very specific counter-tactics, with little room for variation. By design,

dynamic scripting generates a variety of tactics at all times, thus it is unlikely to make the appropriate choices enough times in a row to reach the RTP. A possible solution to this shortcoming of adaptive game AI, is to allow it to recognise that an optimised tactic is used, and then oppose it with a pre-programmed "answer" without activating a learning mechanism. Note, however, that since the existence of super-tactics can be considered a weakness of game design, a better solution would be to change the game design before the release of the game, to make super-tactics impossible.

CONCLUSIONS

We set out to show that offline evolutionary learning can be used to improve the performance of adaptive game AI, by improving the domain knowledge that is used by the adaptive game AI. We implemented an adaptive game AI technique called "dynamic scripting", which uses domain knowledge stored in rulebases, in the RTS game WARGUS. We tested the implementation against four manually designed tactics. We observed that, while dynamic scripting was successful in defeating balanced tactics, it did not do well against two optimised rush tactics. We then used evolutionary learning to design tactics able to defeat the rush tactics. Finally, we used the evolved tactics to improve the rulebases of dynamic scripting. From our empirical results we were able to conclude that the new rulebases resulted in significantly improved performance of dynamic scripting against all four tactics.

We draw three conclusions from our experiments. (1) Dynamic scripting can be successfully implemented in RTS games. (2) Offline evolutionary learning can be used to successfully design counter-tactics against strong tactics used in an RTS game. (3) Tactics designed by offline evolutionary learning can be used to improve the domain knowledge used by adaptive game AI, and thus to improved performance of adaptive game AI.

Future Work

It can be argued that a game is entertaining when the game AI attempts matching the playing strength of the human player, instead of defeating the human player. In parallel research, techniques have been investigated that allow dynamic scripting to scale the difficulty level of the game AI to match the human player's skill, instead of optimise it (Spronck, Sprinkhuizen-Kuyper and Postma 2004c). In future work we will add difficulty-scaling enhancements to dynamic scripting in RTS games. We will also test dynamic scripting in RTS games played against humans, to determine if adaptive game AI actually increases the entertainment value of a game.

In the present research, the translation of the evolved solutions to improvements in domain knowledge was done manually. Because the translation requires understanding and interpretation of the evolved solutions, it is difficult to perform the translation automatically. Nevertheless, in future work we will attempt to design an automated mechanism that translates tactics evolved by offline evolutionary learning into an improved rulebase for dynamic scripting. The addition of such a mechanism would enable us to completely automate the process of designing successful rulebases for dynamic scripting.

REFERENCES

- Brockington, M and M. Darrah. 2002. "How *Not* to Implement a Basic Scripting Language." *AI Game Programming Wisdom* (ed. S. Rabin), Charles River Media, Hingham, MA, pp. 548–554.
- Buckland, M. 2004. "Building better Genetic Algorithms." *AI Game Programming Wisdom 2* (ed. S. Rabin), Charles River Media, Hingham, MA, pp. 649–660.
- Buro, M. 2003. "RTS Games as Test-Bed for Real-Time AI Research". *Proceedings of the 7th Joint Conference on Information Science (JCIS 2003)* (eds. K. Chen *et al.*), pp. 481–484.
- Cohen, R.C. (1995). *Empirical Methods for Artificial Intelligence*, MIT Press, Cambridge, MA.
- Demasi, P. and A.J. de O. Cruz. 2002. "Online Coevolution for Action Games." *GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation* (eds. Q. Medhi, N. Gough and M. Cavazza), SCS Europe Bvba, pp. 113–120.
- Gold, J. 2004. *Object-Oriented Game Development*, Addison-Wesley, Harrow, UK.
- Goldberg, D.E. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, Reading, UK.
- Johnson, S. 2004. "Adaptive AI: A Practical Example." *AI Game Programming Wisdom 2* (ed. S. Rabin), Charles River Media, Hingham, MA, pp. 639–647.
- Laird, J. E. and M. van Lent. 2000. Human-Level AI's Killer Application: Computer Game AI. *Proceedings of AAAI 2000 Fall Symposium on Simulating Human Agents*, Technical Report FS-00-03. AAAI Press 2000, pp. 80–87.
- Laird, J.E. 2001. "It Knows What You're Going To Do: Adding Anticipation to a Quakebot." *Proceedings of the Fifth International Conference on Autonomous Agents* (eds. J.P. Müller *et al.*), ACM Press, Montreal, Canada, pp. 385–392.
- Manslow, J. 2002. "Learning and Adaptation." *AI Game Programming Wisdom* (ed. S. Rabin), Charles River Media, Hingham, MA, pp. 557–566.
- Manslow, J. 2004. "Using reinforcement learning to Solve AI Control Problems." *AI Game Programming Wisdom 2* (ed. S. Rabin), Charles River Media, Hingham, MA, pp. 591–601.
- Ponsen, M. 2004. *Improving Adaptive AI with Evolutionary Learning*. MSc Thesis, Delft University of Technology.
- Rabin, S. 2004. *AI Game Programming Wisdom 2*. Charles River Media, Hingham, MA.
- Russel, S. and J. Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Pearson Education, Upper Saddle River, NJ.
- Schaeffer, J. 2001. "A Gamut of Games." *AI Magazine*, Vol. 22, No. 3, pp. 29–46.
- Spronck, P., I. Sprinkhuizen-Kuyper, and E. Postma. 2004a. "Online Adaptation of Game Opponent AI with Dynamic Scripting." *International Journal of Intelligent Games and Simulation* (eds. N.E. Gough and Q.H. Mehdi), Vol. 3, No. 1, University of Wolverhampton and EUROSIS, pp. 45–53.
- Spronck, P., I. Sprinkhuizen-Kuyper, and E. Postma. 2004b. "Enhancing the Performance of Dynamic Scripting in Computer Games." *Entertainment Computing – ICEC 2004* (ed. M. Rauterberg), LNCS 3166, Springer-Verlag, pp. 273–282.
- Spronck, P., I. Sprinkhuizen-Kuyper, and E. Postma. 2004c. "Difficulty Scaling of Game AI." *Proceedings of the GAME-ON 2004 Conference*. (To be published)
- Tozour, P. 2002. "The Perils of AI Scripting." *AI Game Programming Wisdom* (ed. S. Rabin), Charles River Media, Hingham, MA, pp. 541–547.

HIGH-LEVEL DECISION LEARNING FOR NON-PLAYER CHARACTERS IN VIDEO GAMES

William TAMBELLINI
KYNOGON
12 rue Chaussée d'Antin
75009 Paris FRANCE

E-mail : William.Tambellini@kynogon.com

Cédric SANZA and Yves DUTHEN
IRIT Laboratory
118 route de Narbonne
31062 Toulouse CEDEX FRANCE
E-mail : sanza@irit.fr

KEYWORDS

Classifier Systems, Genetic Algorithms, Behavior Learning, Video Games, Non-Player Characters, High-Level Decision.

ABSTRACT

Among evolutionary learning techniques, one can find Classifier Systems (CS) and Genetic Algorithms (GA). GA generally evolve simple data structures, series of information unit. CS deal with condition-action structures.

By consequences, CS are well-suited for perception-action decision learning.

In non-scripted video games, both Non Player Characters (NPC) and players are a full freedom. In this case, NPC need efficient high-level decision system. It will decide why the NPC will adopt behaviors such as wandering, fleeing, following, shooting, hiding, etc.

Non-scripted games do need efficient high-level decision rules. These rules are difficult, tiring and time-consuming to write by hand. We propose instead to force NPC learn high-level behavior.

For this, we use CS to obtain rules of behavior thanks to an off-line learning session. This article describes what using CS for NPC requires. Among others, it needs to choose and build perceptual and action genes and chromosomes. Furthermore, it requires to choose fitness for rewarding NPC.

Then we propose an example of application for a simple video game.

INTRODUCTION

Computer Games (CG) present two kinds of entities in the point of view of Artificial Intelligence : Player Characters (PC) and Non Player Characters (NPC). NPC are the ones controlled by the core game engine. Decision making for NPC uses :

- any hard coded decisions like in "case" C instruction ;
- any simple scripted rules base as in LUA language ;
- any high-level decision system.

In Action\Adventure games, NPC can play several roles regarding the player but they are above all enemies towards players.

In this case, NPC enemies have two kinds of situation. In scripted games such as Half-Life solo mode or Max Payne, they are present to participate to a written scenario and do not have to interfere in the story. In this case, they have a short life time and no liberty of move nor decision.

On the contrary, in Multiplayer modes as "Deathmatch" or "Team Deathmatch", or "Shoot Them Up" games as Unreal Tournament or Quake III, NPC and PC are symmetric. NPC

have the same role as players, have a full freedom and a longer life time. We propose to work on this kind of NPC.

DECISION MAKING FOR NON-PLAYER CHARACTERS

To provide interaction between PC and NPC, CG developers have to tell NPC which basic behavior to adopt according to the situation. Here is a simple example of rule:

```
if ((DistanceToEnemy<30)&&(life<50)
|| (!MedikitNear&&EnemyStrong)
&& (nMunitions<10&&!MunitionPackNear)
|| (Armor<20)&&(EnemyArmor>80))
then (Flee(NearestEnemy))
```

Fig.1 : Example of a rule for NPC in Shoot games

A set of rules could be mathematically assimilated to a Finite State Machine (FSM) (Houlette et al. 2003). Today, mainly three ways are used to implement such a technique :

- write the rules in a "hard-coded" way, i.e. in the source code ;
 - put the set of rules outside the code, i.e. in a script. It allows developers not to recompile engine for each modification of behaviors rules ;
 - use a graphical user interface software to easily build the FSM and export it in the desired language ;
- Some drawbacks of such techniques are explained in (Manslow 2002) :
- limiting the scope of the problem to a finite set of entities is dangerous ; if the system becomes complex, you can easily forget a state ;
 - defining a complete FSM is an extremely long process ;
 - when system complexity grows up, developers should expect to see an even greater list of events and states to support it. As logic errors are common in programming, event errors are common in FSM. Large and complex FSM can easily become a nightmare to modify and tune. However, tuning is the hottest game production time because it determines if the game will be funny or not.

Rather than building the whole rules base manually, we propose to force NPC to learn how to play. Resulting behavior should be optimal, coherent and intelligent. After the learning session, we could export these rules to any language that game designers could modify.

LITTERATURE REVIEW

Learning for video games

The bibliographic study from (Kirby 2003) shows that learning for CG, independently of the used technique (Neural Network, Reinforcement Learning or Genetic Algorithms), leaves CG developers baffled.

Our issues show two different learning tasks. The first is the one made during the game, called **on-line**, and the second is the one made before game commercial release, realized by developers during production sessions, called **off-line** learning.

The first one suffers from a maximum of negative critics from CG developers. Mostly all uses of online learning resulted in unplayable games with bad gameplays or made NPC learn just a little or not at all (Manslow 2002). However, (Spronck et al. 2003) shows that this could be useful for dynamic scripting in Role Playing Games.

On the other side, off-line learning techniques are difficult to tune and to use to get the desired behavior. Nevertheless, this kind of learning is safe because it minimizes the appearance of non-desired behaviors during the game. This can be obtained by letting the game designer delete, select and reinforce the rules of his choice after learning session. Off-line learning also presents advantages to enhance NPC behavior and to find holes in the game engine (Spronck et al. 2002).

Off-line learning has to build dynamically and automatically an efficient set of rules. It should be much more pleasant for the game designer to use this offline learning rather than writing each rule manually.

In this article, our purpose is mainly to obtain efficient behavior rules after a more or less long off-line learning session. For this, we have chosen the evolutionary algorithm called Classifier Systems (CS).

Classifier Systems and Genetic Algorithms

Genetic Algorithms (GA) and Classifier Systems (CS) has been introduced by (Holland 75). GA emphasize the use of a "genotype" that is decoded and evaluated. A population of different genotype is then mixed using evolutionary Darwin theory to find the best subjects.

Whereas genotypes in GA are often simple data structures, as a list of parameters for example, CS deal with rule

structures ; each Classifier being a 'condition-action' rule.

Let consider the simple model of CS of Fig.2. In evolutionary algorithms, each subject of the population is a solution trial. In the CS point of view, each subject of the population is a solution part of the system. This means that the genomes in GA is represented by each subject. In CS, it is represented by each Classifier (see Fig.2).

As a consequence, each part of a classifier, condition and action, represents a chromosome. Each chromosome contains some genes. Each gene represents a perception or action unit. These genes can contain different kinds of data : boolean, integer, float, interval, etc.

A CS is then a system handling a pool of n classifiers (Holland et al. 99). To interact with the environment, CS clients send and receive messages (a, b, c, ...) to and from the CS via Message Board, recording perceptions and actions history. The client will compare this perception chromosome to the condition part of each classifier and send the best action chromosome corresponding to the perception. Each classifier has a fitness value, i.e. a note which indicates his strength to solve the problem (between -200 & 200 in Fig 2). When different classifiers reply to the same perceptions, CS use a bidding system to get the wining subject.

At the initialization time, the whole rules are chosen randomly. During learning, the Environment acts according to the rules of the CS using bid system for concurrent rules. When the Environment detects that a client has done a good or bad action, it rewards or punishes the last classifiers used by this client (Reward from the Environment in Fig.2).

To make the whole population evolve, during the learning session, we have to call the GA function to mix the population of classifiers in order to create a new generation (at left in Fig.2). It will kill the worst, keep the best, and produce probably better new subjects, here classifiers. This evolution uses recombination mechanisms. (Buckland 2003) lists the main mechanisms needed :

- selection using elitism and tournament techniques ;
- duplication using reproduction rules ;
- mutation which locally change one or several genes of a chromosome ;
- recombination with simple or multiple crossover points, which combine two or more chromosomes between them.

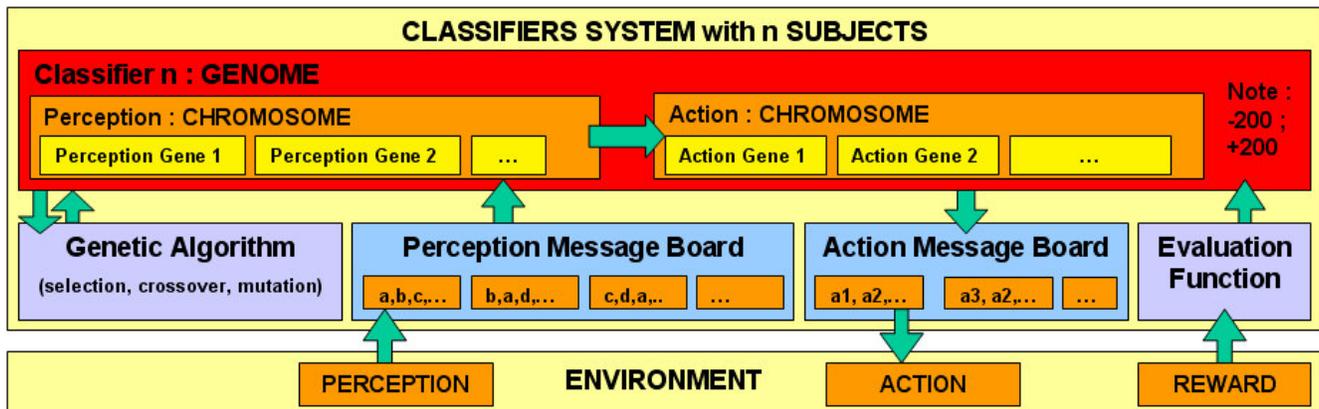


Fig 2 : A simple Classifier System architecture

Genetic Algorithms, Classifier Systems, and Video Games

(Robert et al. 2002) presents first works using CS for NPC for the action selection. This work has been particularly used for Massively Multi-players Online Role Playing Games, MMORPG. Unfortunately, it does not really help us as it uses personal, unusual and undocumented CS. Despite this, (Robert et al. 2003) uses this kind of CS in Multiplayer games. Even if the kind of CS used in this work differs too much from ours, it deals with the same issues.

(Demasi et al. 2003) presents works on both online and offline learning, but using GA, not CS. Nevertheless, they build their chromosomes in the same way CS are modeled : <condition><action>. Whereas we propose to work only on pure offline learning, it deals with both learning types even on mixing online and offline evolutions. Moreover, it proposes to animate the player agent with both hand-made rules data and human internet data, which is unfeasible in our case. To finish, it has the same point of view for basic perception (like distance to enemy) and high level behavior (like chase player) as our experiment. All these points can help us with the global issues of this kind of works, but do not help us about pure CS choices.

(Sanza et al. 2000) and (Sanza et al. 2001) simulate a soccer game. The NPC have to decide at each time what to do, on the basis of both an individual fitness value and a collective reward. This last is evaluated relatively to the efficiency of the whole team. Here, each football player has his own CS because each player has a different goal (attack, defense,...). We have construct our experiment with help of this work.

(Heguy et al. 2002) present a generic CS for a real-time task learning. Even if this work mainly deals with an artificial life purpose, the use of CS in dynamic environments for behaviors learning is similar to our goal. It is used here in a virtual basketball simulation. This work has the same issues as the other ones : team fitness and CS modeling.

In spite of those works, and according to the CG bibliography, unclear trials of using GA and CS for CG let consider that the use of these techniques is not obvious in this context.

And yet, CS have a major advantage on others learning techniques. One of the problems which appear for behavior learning is result readability. Indeed, game designers want to keep control over behavior rules for tuning purpose. It means that the results have to be understandable and modifiable by user. Precisely, CS have the advantage of giving understandable results, contrary to Neural Networks for example. Indeed, neural networks concentrate their knowledge, during and after learning sessions inside a structure topology (number of hidden layers, number of neurons per layer...), and inside synapse weights. This structure is totally unreadable if one want to know which weight controls which behavior component.

For CS, the knowledge is concentrated in the set of rules. Originally, each classifier is just a bit concatenation as « 01011001->1110101 ». But after the translation work, each rule could be easily understood as “if condition then action”. Example: « if Distance to enemy lower than 100 and Enemy busy, attack enemy » which is very understandable.

To conclude, literature review shows that behavior learning is a delicate subject when applied in video games. Moreover, CS have been rarely applied in their basic theory. We

propose here to clarify the use of CS in a simple video games.

APPLICATION

Here is an example of application of CS for a simple game (fig.2) :

- a world, kind of maze with tunnels and holes ;
- several termites, which can see, walk and attack Hunter ;
- a hunter, which can see, run and attack termites ;



Fig.3 : one hunter versus several termites

The termite score increases when they kill Hunter and Hunter wins when it kills all the termites.

We use for this work RenderWare Artificial Intelligence (RWAI 2004). RWAI separates entity actions by low and high level decisions. The low level deals with entity direction, speed and shooting parameters. The high level decision, called Brain structure, only selects global behavioral agents as Wander, Flee, Attack, etc. This work deals with high level decision only.

Then Hunter and termites could have the choice between the following behaviors :

- wandering using a random destination point ;
- attacking any visible entity ;
- following any visible entity ;
- fleeing or hiding from an entity ;

Game engine has been set to provide an inequality between termite and Hunter hits. The hunter kills a termite in just one shot. When a termite hit the hunter, this last loses only a quarter of his life. This will force the termite to search for a better rule than : “If Hunter Visible then Attack Hunter”.

In this application, Hunter entity has a hard coded behavior which is :

"If Termite(s) seen then Attack Nearest Termite else Wander"

It means that only the termites use CS decision system.

We have now to explain what is required to apply CS in this simple game.

(Sweetser 2003) explains the main difficulties to use Evolutionary Algorithms for games :

- representing possible solutions in genes, chromosomes and genomes ;
- finding an adequate fitness function for rewarding good behaviors ;

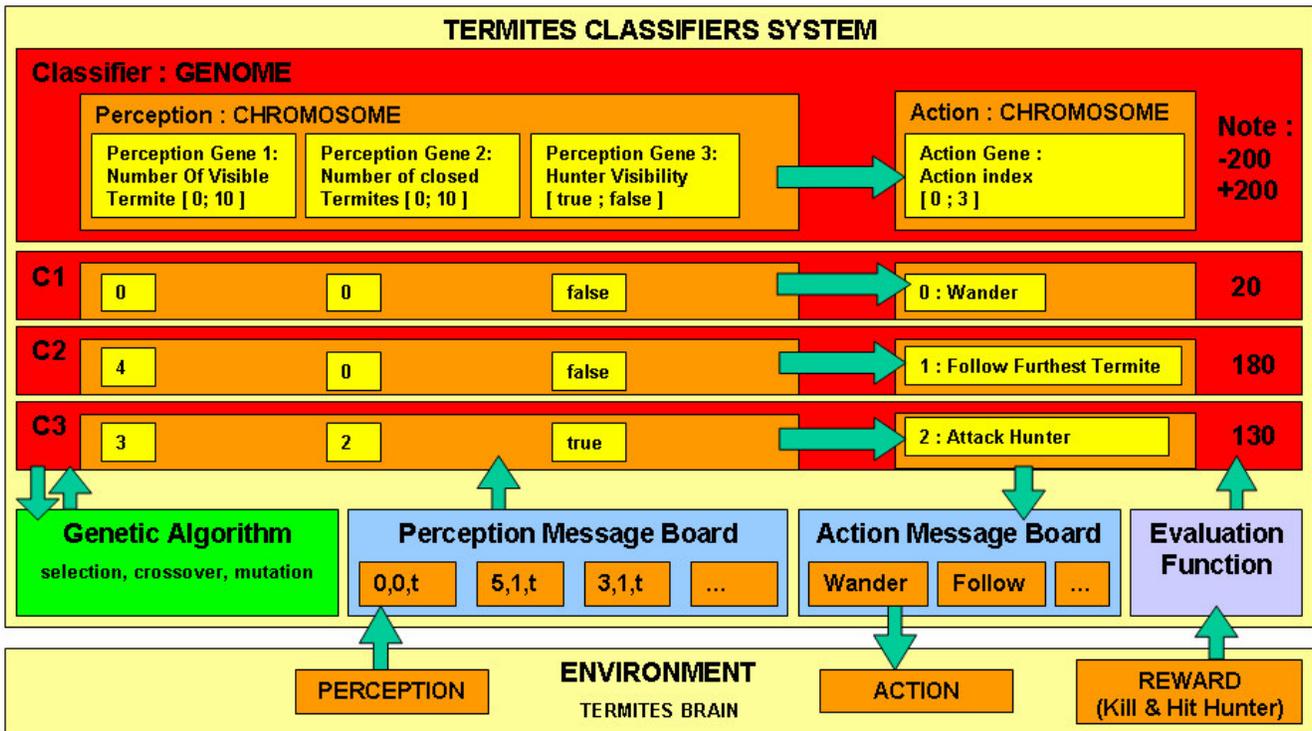


Fig. 4 : Classifier System architecture applied to a simple video game

In our application, the first difficulty is to select perceptual genes. Main dangers are to put termites in **under perception** (not enough perception to solve problems) or **upper perception** (too much perception genes).

Fig.4 presents the modeling we propose for the termites classifier representation. We have tried to minimize number of perceptual genes (only 3 in this experiment) by choosing the most important : number of visible termites, number of closed termites & hunter visibility. The Fig.4 only shows 3 examples of classifier (C1, C2, C3) but you can easily imagine 40 rules as in our experiment. This choice of perception chromosome has been made with good reasons. Indeed, after writing hand made rules, we have checked that this perception allow the termites to win as much as Hunter. So this verification gives us a reference on the possible efficiency of the learning.

The second difficulty is to manage the fitness function. We decide to reward the termite Messages Board only when this termite hurts the Hunter. We show here that the fitness function depends on the problem. This means that each game has his own fitness function even if it could often be similar ("Kill enemy" for example). Here, we choose the easier way to reward actions. As the termites have to hurt the hunter many times to kill it, we could think that rewarding hurting could lead to his death.

In this example of application, CS has to discover and keep the best rules that make termites kill hunter. For this, reward has been set to 50 and happens only when a termite hurts the Hunter. Number of classifier has been set to 40.

Score measure equals to score of termites divided by score of Hunter. Learning session is required for letting CS evolve. Testing session is necessary to calculate set of rules efficiency. Both sessions last 30 minutes. At initialization, the Hunter always starts from the same point, the termites start from random points in the world.

MAIN RESULT

We will show here preliminary results of this experiment in its first version.

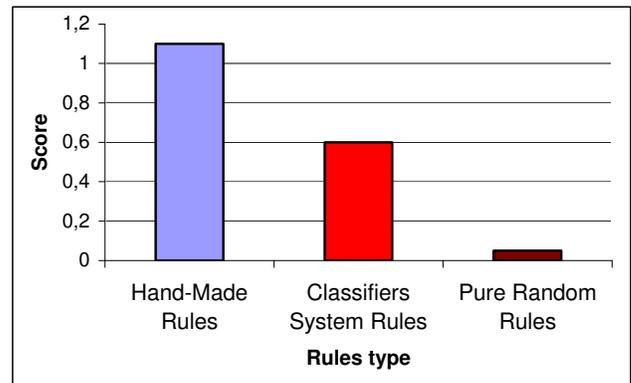


Fig. 5 : Mean results using different rules types

A summary of our main result is presented in Fig.5. We see that Hand Made Rules do not exceed learned ones (mean obtained after 10 testing sessions).

After learning sessions, CS effectively gave efficient rules like this one (C3 in Fig. 4):

```
"If    NumberVisibleTermite = 3
      and NumberOfClosedTermite = 2
      and HunterVisible = true
then  Attack Hunter"
with  strength=130
```

We can also see that learning rules are better than pure random rules, showing that learning session has been useful to find efficient rules even if not perfect.

DISCUSSION

We have shown that using CS for NPC decision systems needs several requirements. The main difficulties are building well-chosen genes, and getting an adequate fitness function using rewards and punishments.

There are probably two reasons for the results we obtained.

The first one is about “non-deterministic” systems. It seems that the game we used is not deterministic. That means a same action in same conditions does not always result in the same consequence. This is explained by uses of random (such as in the “Wander” behavior). The results could be also explained by pure game engine noises like physics between entities, processor unit load, etc. In other words, external parameters make the consequences of the NPC decision vary.

This instability could explain why the CS rules hardly converge.

The second reason is caused by the fitness. Nearly every supervised learning algorithms build their knowledge thanks to an efficient fitness function, i.e. reward or punishment functions. In non-scripted video games, it seems the only reward we can do with CS is when a sub-goal has been achieved by a NPC. GA & CS are generally used with accurate and stable fitness functions. In such video games, we should only have a rare and fuzzy reward.

We have seen that the results of CS learning in a highly non-deterministic environment is quite weak.

The next step is now to work on different kinds of CS that could deal with such non-deterministic commercial video games.

REFERENCES

- Buckland M. 2003, "Building Better Genetic Algorithms" In *AI Game Programming Wisdom 2*, eds. S.Rabin, Charles River Media, 649-660.
- Demasi P., Cruz A. J. 2003 "Online Coevolution for Action Games." *Int. J. Intell. Games & Simulation* 2(2), 80-88
- Heguy O., Sanza C., Berro A., Duthen Y. 2002 "GXCS: A Generic Classifier System and its application in a Real Time Cooperative Behavior Simulations" In *The International Symposium and School on Advanced Distributed System (ISADS'02)*, 11-15
- Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*". University of Michigan Press, Ann Arbor.
- Holland, J. H., L.B. Booker, M. Colombetti, M. Dorigo, D. E. Goldberg, S. Forrest, R. L. Riolo, R. E. Smith, P. L.

- Lanzi, W. Stolzmann, S. W. Wilson 1999, "What Is a Learning Classifier System ?" In *Learning Classifier Systems*, 3-32
- Houlette, R and D. Fu. 2003. "The Ultimate Guide to FSMs in Games." In *AI Game Programming Wisdom 2*, eds. S.Rabin, Charles River Media.
- Kirby N. 2003 "Getting Around the Limits of Machine Learning" In *AI Game Programming Wisdom 2*, eds. S.Rabin, Charles River Media, 603-611.
- Manslow, J. 2002. "Learning and Adaptation" In *AI Game Programming Wisdom*, eds. S.Rabin, Charles River Media, 557-566.
- Robert, G., Portier, P. and Guillot, A. 2002. "Classifier systems as 'Animat' architectures for action selection in MMORPG". In *Gameon 2002* eds. Mehdi, Gough and Cavazza. 121-125.
- Robert, G. and Guillot, A. 2003 "MHiCS, A Modular And Hierarchical Classifier Systems Architecture For Bots". In Mehdi, Q., Gough, N. and Natkin, S. eds. *Game-on 2003*. 140-144.
- Sanza, C. ; C. Panatier ; Y. Duthen. 2000. "Communication and Interaction with Learning Agents in Virtual Soccer". In *Proceedings of Virtual Worlds 2000*, eds. J.-C. Heudin, 147-158.
- Sanza C., O. Heguy, Y. Duthen. 2001. "Evolution and Cooperation of Virtual Entities with Classifier Systems" CAS'2001, Eurographic Workshop on Computer Animation and Simulation, Springer Computer Science, ISBN 3-211-83711-6.
- Spronck, P., I. Sprinkhuizen-Kuyper and E. Postma. 2002. "Improving Opponent Intelligence Through Offline Evolutionary Learning." In *International Journal of Intelligent Games and Simulation* (eds. N.E. Gough and Q.H. Mehdi), Vol. 2, 20-27.
- Spronck, P., I. Sprinkhuizen-Kuyper and E. Postma 2003. "Online Adaptation of Game Opponent AI in Simulation and in Practice" In *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON 2003)* (eds. Q.H. Mehdi and N.E. Gough), 93-100. EUROSIS, Belgium.
- Sweetser P. 2003, "How to Build Evolutionary Algorithms for Games" In *AI Game Programming Wisdom 2*, eds. S.Rabin, Charles River Media, 627-637.

SOFTWARE

RW RenderWare, Criterion, <http://www.renderware.com>

RWAI RenderWare Artificial Intelligence, Kynogon S.A., <http://www.kynogon.com>

IMITATION LEARNING AT ALL LEVELS OF GAME-AI

Christian Thureau, Gerhard Sagerer
Applied Computer Science
Bielefeld University
P.O. Box 100131, 33501 Bielefeld, Germany
{cthureau,sagerer}@techfak.uni-bielefeld.de

Christian Bauckhage
Centre for Vision Research
York University
4700 Keele Street, Toronto M3J 1P3, Canada
bauckhag@cs.yorku.ca

ABSTRACT

Imitation is a powerful mechanism the human brain applies to extend its repertoire of solutions and behaviors suitable to solve problems of various kinds. From an abstract point of view, the major advantage of this strategy is that it reduces the search space of appropriate solutions. In this contribution, we discuss if and how the principle of imitation learning can facilitate the programming of life-like computer game characters. We present different algorithms that learn from human generated training data and we show that machine learning can be applied on different levels of cognitive abstraction.

Introduction

Considering the past two decades, we can clearly observe a coevolution of commercial computer games, computer graphics and networking. However, for the time being, this dynamic between game industry and academic research seems rather an exception than the rule. Although the potential for another coevolution is obvious, behaviour programming for game characters and artificial intelligence (AI) or machine learning (ML) research hardly inspired each other.

Of course, ideas from academia have entered game AI programming. But it is striking, though, that the two most prevalent AI techniques in game programming, i.e. the A* algorithm and finite state machines, are somewhat old-fashioned. A*-search for pathfinding was introduced more than three decades ago (Hart et al. 1968) and finite state machines with outputs even date back to the 1950s (Mealy 1955, Moore 1956)

The main reason why more recent results just merely influence game character programming is that algorithms that would produce the versatility and flexibility of human players are still not available. So far, research on autonomous agents mainly focused on robots that navigate through the physical world. And even though roboticists recognise the importance of *learning from demonstration* in order to constrain the search space spanned by this task (Schaal 1999), it's fair to say that uncontrollable environmental dynamics and sensor noise still consume more intellectual efforts than techniques for behaviour representation and learning.

However, this situation is about to change as the AI and ML communities are beginning to discover the merits of computer games (Amir & Doyle 2002, Laird 2001, Le Hy et al. 2004, Nareyek 2004, Sklar et al. 1999, Yannakakis & Hallam 2004, Spronck et al. 2002)

In this paper, we will discuss the possible impact of *imitation learning* techniques for computer games. To this end, we will briefly summarise behavioural, neurobiological and AI and ML perspectives on imitation learning. Then we will identify different levels of human behaviour that occur in computer games and consequently will require algorithmic solutions. Following an idea discussed in (Bauckhage et al. 2003), we will thus report on different techniques of analysing the network traffic of multiplayer games in order to realize game agents that exert human-like behaviour learned from examples.

Imitation Learning for Games

Numerous behavioural science experiments document that infants endeavour to produce a behaviour previously demonstrated to them. Some psychological studies on imitation learning even suggest that infants devote most of their time to the imitation of observed behaviours (Rao & Meltzoff 2003).

Obviously, imitation requires a mechanism to map percepts onto actions. And indeed, neurophysiological examinations indicate that there are particular brain areas specialised in imitation (Hietanen & Perrett 1996). After finding neurons that were specific to the execution of goal related limb movements, a connection to imitation came with the discovery of mirror neurons (at least in the brain of macaque monkeys) which are active during the observation as well as the execution of a task (Kohler et al. 2002).

Backed by these results from behavioural science and neurobiology it is no surprise that the idea of imitation learning is getting ever more popular in AI and robotics. Current robotics research on imitation learning mainly concentrates on sub-symbolic techniques like neural networks and fuzzy control. However, it is well known that such low-level approaches do not scale well to situations of many degrees of freedom. As a consequence, the concept of movement primitives was introduced to encode complete temporal behaviours (Fod et al. 2002, Schaal et al. 2003). In fact, there



Figure 1: Training sample generation at a “lan-party”

is evidence that the human brain uses movement primitives to produce goal oriented actions since using such primitives considerably reduces the number of parameters that must be learned (Thoroughman & Shadmehr 2000). Movement primitives are thus closely related to mirror neurons some of which are believed to be high-level representations of behaviour.

The capabilities of the human brain of course extend to virtual worlds and learning how to play a computer game is a process of training and imitating experienced players. In the following, we will thus argue that –similar to robotics– learning from demonstration can provide an avenue to behaviour programming for computer game characters. Unlike present day robotics, however, computer games provide an excellent testbed for the learning of complex behaviours. For most genres, the behaviour of game characters will be composed of *reactive*, *tactical* and *strategic* decisions. While the latter, higher cognitive aspects are still widely neglected in robotics (since the problem of sensor noise is so predominant), computer games allow to study them rather effortlessly.

To substantiate our arguments, we will base the discussion on the example of ID software’s game QUAKE II[®]. First, we shall describe what kind of *percepts* this game provides as input for imitation learning. Then, we will present algorithmic solutions for different kinds of behaviour.

Why Quake ?

QUAKE II[®] is a so called *Ego* or *First-Person-Shooter*. Figure 2 shows a typical game situation the way a human player views the gaming world, from a first-person perspective.

The main goal of the game is to get the most points in a fixed time span. Points are gained by shooting enemy players (so far we are not using team based game modifications, therefore all other players are opponents).

The game takes place in a 3D gameworld which is loosely based on the real world. The human players can move around freely, their actions are bound to the game physics, which are also based on the real world, not all places are always

reachable.

Different types of items (weapons, armor, health packages) are distributed at fixed positions around the map. If an item is picked up, it reappears about thirty seconds later at the same position. From these item positions arises one strategic component of the game. Winning is a lot easier by smart item control, get the best items for yourself and only allow the most weak weapons and armors to be picked up by your enemies. But there are of course endless possibilities of different strategies, which all might lead to a successful gameplay.

Winning a game does not only depends on good aiming, smart playing is often a better way to success - maybe that is the reason why even game bots playing with superhuman aiming are still beaten by good, experienced human players¹.

So why would we want to use such a complex game for imitating human controlled game agents? Simpler games, more focused onto one aspect of human acting could maybe provide better results with less effort, besides, by using a home-made game we could avoid the troubles of interacting with someone else’s (mostly closed source) software.

There are basically two answers to that question. First, imitating humans wandering around a 3D virtual world and performing tasks of different complexity is not only of interest for the gaming community and might provide further insights on the modeling of human behaviours in general. The level of abstraction compared to the real world is a lot lower than in Chess, Tetris or PacMan. In addition, unlike in the real world, we have perfect sensor data. The second point is, that by using a commercially successful game, we have access to an incredible huge database of records of humans playing the game – these so called demo files can be recorded by ourselves (Figure 1 shows a possible location for demo recording) or just downloaded from numerous sites on the internet. Having an almost unlimited amount of training samples, showing nothing less than humans performing complex tasks in complex environments is a unique (so far almost unrecognized) setting.

The training data or demo files we are dealing with are records of the network traffic. They contain information about the exact locations (x, y, z) the player assumed, nearby items and other players. Temporary entities like sounds, and flying projectiles are also included. There is no need for a visual analysis of a game scene, since all necessary information is already available on a cognitive higher level. The same applies to the player actions, they are included as simple velocity and position vectors.

Categories of Behavior

Our current model for the imitation of a human player’s behaviour in QUAKE II[®] consists of three separate layers as

¹<http://botchallenge.com> offers a competition on who is the fastest human player in beating *nightmare* (which is somehow equal to superhuman) skilled game-agents in the game QUAKE III[®]

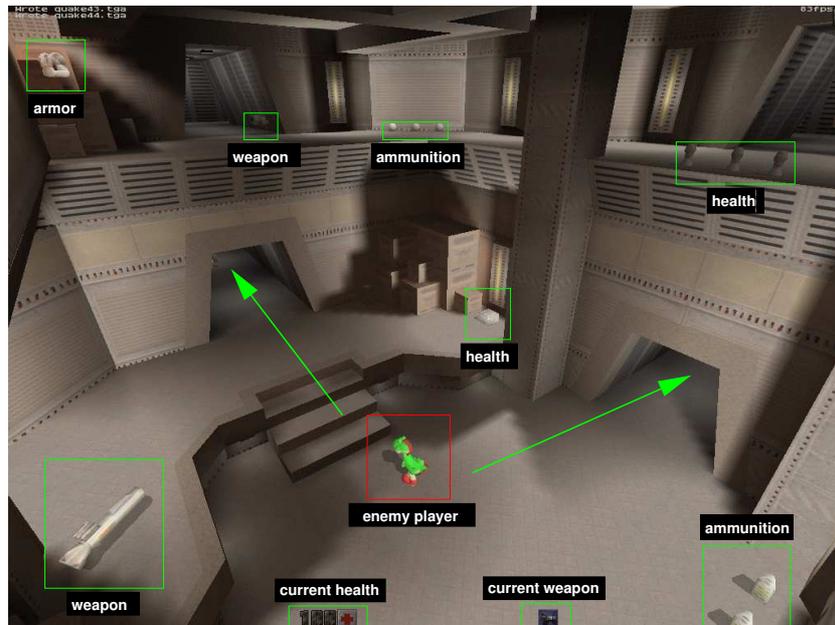


Figure 2: A typical situation in the FPS QUAKE II[®], augmented by some entity descriptions.

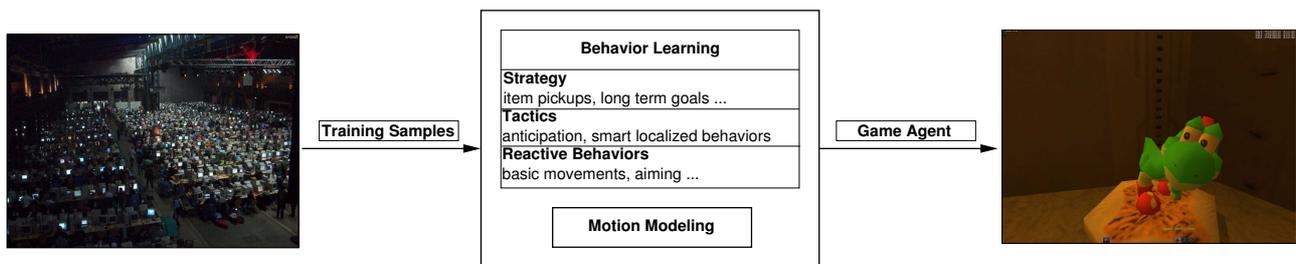


Figure 3: A model for generation of human-like game agents by imitating human play styles.

visualized in Figure 3 (since the model is rather abstract, it's transfer to other games is straightforward). The decision for this design was based on observations as well as on a widely believed psychological hierarchy of human behaviour (Hollnagel 1994).

On top-most level, there are *strategic* behaviours which are used to achieve long term goals. In our case, besides the obvious goal of winning the game, several other (sub)goals were identified. Most strategies are targeted at securing important areas and important items. Still there is a constant change in specific subgoals which depends on the current game-state (e.g. if a player is low on health, health refilling items might be more attractive than the most powerful weapon). The player who manages to control the important places of the map, with the most valuable items, will greatly enhance his chances to win the game.

The second layer represents *tactical* behaviours. Tactics usually are defined by a smart, localized situation handling. While the strategy tells the player about the next important region on a map, the tactics are responsible for evading pos-

sible threats on the way. Since tactics highly depend on anticipation of an enemy player's movement, a broader understanding of a scene is necessary. Prominent examples include the laying of traps or securing of areas by putting it under constant fire.

The most basic layer is given by *reactive* behaviours. Here we find simple reactions to audio-visual percepts. This includes movement, jumps, but also aiming and shooting on an enemy player as well as the prediction shots based on audible cues.

Although this discrimination of behaviours is rather strict and suggests a selection of one active layer of behaviours, they are to be understood as concurrent sets of behaviour. However, unexperienced players are more likely to concentrate on only one aspect. They will either engage in combat or look for better items, whereas experienced players have no problems in improving their strategic position while being in combat (often combat itself holds many tactical parts as well, e.g. taking cover or avoiding small passages). A classification of behaviours into categories makes the whole problem

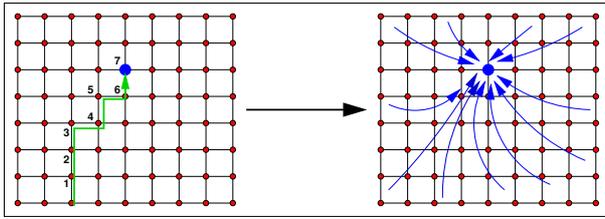


Figure 5: The left figure shows an observed movement pattern of a human player in the topological map representation, the right figure shows the corresponding computed potential field forces.

of imitating humans a lot easier and finally manageable.

Apart from behaviour learning we discovered that the problem of motion modeling is another integral part for imitation learning. In early experiments, we noticed that, although the actions seemed reasonable, the motion of our artificial agent looked jerky and were more robot- than life-like. Moreover there is a strong coupling between the actions a human player performs and the actions he is motion-wise capable of doing. Artificial agent's usually do not have such restrictions, they can instantly turn and even aim perfect on long ranges. Therefore the imitation of a human controlled agent's motion is necessary and poses another challenge.

On the one hand, we thus need to find approaches for learning by observation what to do in a given game situation. On the other hand, we have to find the appropriate, life-like motion for doing so. An integration into a consistent framework would be desirable.

Learning Strategies

In (Thureau et al. 2004a), we presented an approach for learning strategic behaviours. It realized goal oriented movement in 3D, including situation dependent item pickups and area control.

The approach consists of two parts. First, a topological representation of the 3D gaming world is learned by means of applying a *Neural Gas Algorithm* to the positions $\vec{p} = [x, y, z]$ a human player held during a match. Neural Gasses extended the popular k-means clustering algorithm and are especially suited for building topological representations (Martinez & Schulten 1991). In our case, the result is similar to the widely used waypoint-maps. But since it is data-driven, it provides an elegant and accurate way of generating waypoint maps corresponding to human gameplay. Figure 4 shows an exemplary topological representation.

In a second step, *Artificial Potential Fields* are placed into the topological map. These potential fields then guide the game-bot. Since strategies change according to game-states, the training samples are clustered in the state space of the agent yielding prototypical game states. Here the game-state includes information about the current items of the player and his health and armor values.

For each such state a potential field force distribution is computed which recreates movement patterns typically observed in that state, Figure 5 shows a simple example. Changes in the internal state of the agent cause switches among the field forces and thus will lead the agent to other, more attractive locations and items, thereby implicitly defining situative sequential item pickups as long-term goals.

To cope with known flaws of potential field approaches, namely local minima and weak potential field forces on certain parts of the map, we make use of *Avoiding the Past Pheromone Trails* (cf. (Balch & Arkin 1993)). These trails reinforce a chosen direction and drive the agent out of local minima. The obtained results were convincing. Strategic behaviours, situation dependent item-pickups or preferences for certain map areas could be learned and convincingly reproduced.

Learning Tactics

Tactical behaviours were described as a smart localized behaviour. However, learning or imitating such behaviours is not an easy task. It requires a broader understanding of a situation. For example, the observation of a human player ambushing an enemy player is itself a rather simple sequence of actions. But in order to imitate it, it wouldn't be enough to copy the action sequence. Instead, underlying contextual prerequisites to activate tactical behaviour have to be extracted. In the given example the enemy player might have been entering a room with only one exit, thereby giving the opportunity to ambush him. But learning such an understanding for situations provides a great challenge.

Right now we are searching for an appropriate representation of such more abstract scene understanding. Unfortunately, up to now there are only little known techniques of how to approach that topic. Moreover, since we prefer a data-driven approach to the imitation of tactical behaviours, we do not want to label sample sequences or rely on common game AI methods like finite-state machines or scripting. Recent work thus examined the application of *Mixture of Experts* architecture (Jordan & Jacobs 1994). In particular, we studied the context dependent handling of different weapon types. First results are encouraging, i.e. Mixture of Experts architectures were observed to learn the handling of different weapons; further experiments are necessary though.

Learning Reactive Behaviors

In (Thureau et al. 2003), we presented an approach for learning purely reactive behaviours (note that potential fields are often referred to as *reactive* behaviours, but since we effectively model long-term goals with them, we discussed them above). In QUAKE II[®], reactive behaviours can be characterized as a direct functional mapping of game states $\vec{s}_t, \vec{s}_{t-1} \dots \vec{s}_{t-n}$ onto player reactions \vec{a} . Typical reactive behaviours include aiming, shooting or dodging projectiles.

At first the training sample set is separated by letting a *Self Organizing Map* (SOM) (Ritter et al. 1992) unfold itself into

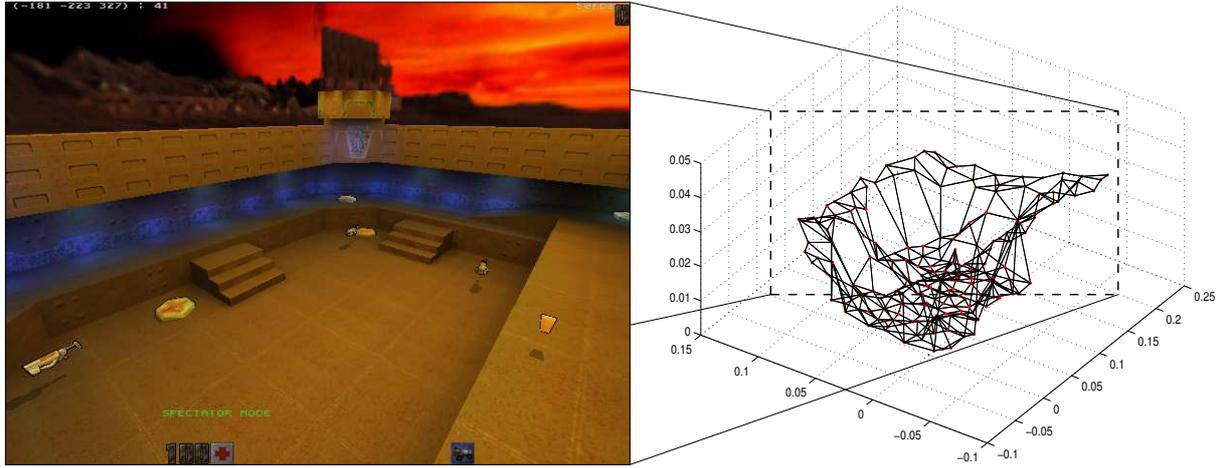


Figure 4: A 3D map and its topological representation as an outcome of a Neural Gas algorithm

the game-state space (in (Thureau et al. 2003), the game-state space consisted of the agent’s position and an enemy’s position). Training samples are then assigned to a corresponding SOM Neuron thus separating the data into different clusters. In a second step, we trained *Multi Layer Perceptrons* (MLP) for each cluster. Figure 6 provides a graphical presentation of the described approach.

For each new game situation, the most similar prototypical game-state from the SOM-Neurons is selected and its corresponding MLP used for behaviour generation. Thereby simple reactive behaviours, aiming or movement in 3D, could be learned. However, even after introducing time-dependent *Time-Delayed-Neural-Networks* the approach is limited to ad-hoc reactions to game-states. Long-term goals and planning of actions should be taken care of by strategical or tactical behaviour models.

Motion Modeling

Although the approaches presented so far reproduce human behaviours, the game agent’s motion often appears jerky and can be easily distinguished from the smooth motions of a human player. An artificial game agents motion is thus an integral part of its appearance, for a realistic impression it should be as human-like as possible.

Our approach to motion modeling from imitation is based on movement-primitives (Fod et al. 2002). Movement-primitives, as the basic building blocks of movement, can be derived from a game agent’s motion vectors using *Principal component analysis* (PCA). Thereby individual movement-primitives for individual players are obtained. To discretize a player’s motion, prototypical motion vectors are extracted from the projections of the training-sample motion vectors onto the eigenvectors (or movement-primitives), using a k-means algorithm. Every single training sample motion vector can now be described by a more general action primitive \vec{v} . Usually a number of up to 800 action primitives is sufficient and covers the set of possible motions very well. Choosing

less action primitives results in a more choppy movement, however, larger numbers of action primitives didn’t lead to an observable smoother recreation of movements.

Complex movements result from sequencing action primitives. Since the *right* sequencing of action primitives, for imitating a human controlled game agent’s motion, is given in the training sample set, probabilities for the execution of an action primitive can be extracted. Two transition matrices are computed. One expresses interdependencies between action primitives, the other expresses localized dependencies, based on the position of the player in the topological representation.

Given these matrices, the next action primitive to execute is chosen according to a roulette wheel selection over the probabilities for all action primitives. The probability for the execution of a single primitive \vec{v}_i can be denoted as:

$$P_{\vec{v}_i} = \frac{P(\vec{v}_i|\vec{v}_l, w_k)}{\sum_{u=1}^n P(\vec{v}_u|\vec{v}_l, w_k)} = \frac{P(\vec{v}_i|\vec{v}_l)P(\vec{v}_i|w_k)}{\sum_{u=1}^n P(\vec{v}_u|\vec{v}_l)P(\vec{v}_u|w_k)}$$

where $P(\vec{v}_i|w_k)$ denotes the probability of executing action primitive \vec{v}_i for topological node graph node w_k , and $P(\vec{v}_i|\vec{v}_l)$ denotes the probability of executing action primitive \vec{v}_i as a successor of action primitive \vec{v}_l . These probabilities can be extracted from the training samples by inspecting the observed action primitive sequence. Since all probabilities can be computed in advance, the approach is computationally rather inexpensive, managing to execute up to 20-30 action primitives a second in our MATLAB[®] QUAKE II[®] client. A detailed analysis can be found in (Thureau et al. 2004b).

Using this approach, we managed to imitate (or recreate) complex sequences of motion. This includes not only simple movements, but also jumps over ledges and the infamous rocket jump (a maneuver, where a player fires a rocket on the ground and jumps at the same time, thereby reaching otherwise unreachable places – considered an experienced

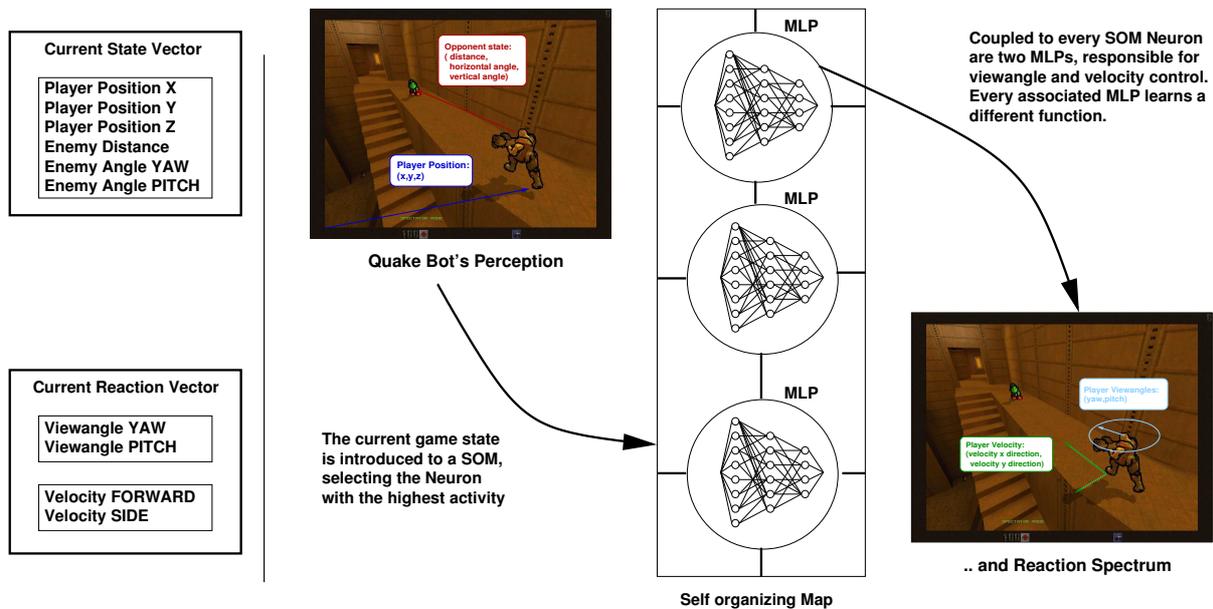


Figure 6: Architecture for learning situation dependent reactive behaviour, establishing a direct mapping of state space variables onto player actions.

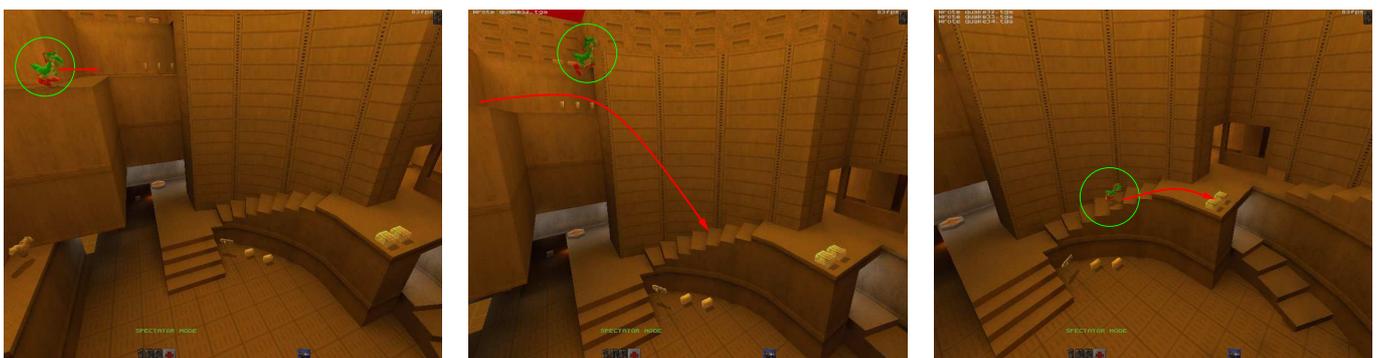


Figure 7: Game-agent performing a long jump by means of movement primitives

player's move). Figure 7 shows the artificial agent performing a long jump to an otherwise not reachable region. The created movements appeared smooth and paid attention to the observed human player's style of movement, creating indeed life-like motion.

Conclusion

In order to create more human-like computer game agents, we proposed the usage of imitation learning. Thereby following a general trend in robotics towards imitation learning and following evidences in psychology on the importance of imitation for behaviour development in infants. Unlike other approaches in the field of machine learning in games, we concentrate on the highly complex genre of Ego or First-Person-Shooter games. These game types provide us with an incredibly huge amount of training samples - records of

human player's, being downloadable from the Internet. We presented a comprehensive approach for the imitation of a human controlled game agent for a FPS game.

First, we identified different behavioural layers, namely strategic, tactical and reactive behaviours. Besides the behaviour learning, we clarified the importance and influence of motion modeling on a life-like appearance of an artificial game character. Finally, we outlined suited approaches for behavioural learning in each of the mentioned layers and sketched our recent work on the topic of motion modeling.

Since human behaviour during a game is very complex and rich of problems when it comes to machine learning, an artificial player that integrates all the techniques presented above was not yet realized. However, from the results discussed in this paper, it is reasonable to conclude that imitation learning is a well suited method for behaviour generation of artificial game characters. Concerning individual aspects of game

play, our game-bots outperform conventional bots which are driven by finite state machines or similar architectures. Bots that learned by imitation definitely stay closer to what a human player is doing since they consequently rely on observations of human players. Therefore, we are convinced it is worthwhile to further pursue this topic in order to see how far imitation learning will bring game characters.

Acknowledgements

This work was supported by the German Research Foundation (DFG) within the graduate program “Strategies & Optimization of Behaviour”.

REFERENCES

- Amir, E. & P. Doyle. 2002. Adventure Games: A Challenge for Cognitive Robotics. In *Proc. Int. Cognitive Robotics Workshop*. Edmonton, Canada: .
- Balch, T. & R. Arkin. 1993. Avoiding the past: A simple but effective strategy for reactive navigat. In *Proc. IEEE Int. Conf. on Robotics and Automation*.
- Bauckhage, C., C. Thureau & G. Sagerer. 2003. Learning Human-like Opponent Behavior for Interactive Computer Games. In *Pattern Recognition*. Vol. 2781 of LNCS Springer-Verlag.
- Fod, A., M.J. Mataric & O.C. Jenkins. 2002. “Automated Derivation of Primitives for Movement Classification.” *Autonomous Robots* 12(1):39–54.
- Hart, P.E., N.J. Nilsson & B. Raphael. 1968. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” *IEEE Trans. on Systems Science and Cybernetics* 4(2):100–107.
- Hietanen, J.K. & D.L. Perrett. 1996. “Motion sensitive cells in the macaque superior temporal polysensory area: response discrimination between self-generated and externally generated pattern motion.” *Behavioral Brain Research* 76:155–167.
- Hollnagel, E. 1994. *Human Reliability Analysis: Context & Control*. Academic Press.
- Jordan, M. I. & R. A. Jacobs. 1994. “Hierarchical mixtures of experts and the EM algorithm.” *Neural Computation* 6:181–214.
- Kohler, E., C. Keysers, M.A. Umiltà, V. Gallese L. Fogassi & G. Rizzolatti. 2002. “Hearing Sounds, Understanding Actions: Action Representation in Mirror Neurons.” *Science* 297:846–848.
- Laird, J.E. 2001. “Using a Computer Game to develop advanced AI.” *IEEE Computer* pp. 70–75.
- Le Hy, R., A. Arrigioni, P. Bessière & O. Lebeltel. 2004. “Teaching Bayesian behaviours to video game characters.” *Robotics and Autonomous Systems* 47(2–3):177–185.
- Martinez, T. & K. Schulten. 1991. A Neural Gas Network learns Topologies. In *Artificial Neural Networks*. Elsevier Science Publishers B.V.
- Mealy, G.H. 1955. “A Method for Synthesizing Sequential Circuits.” *Bell System Technology J.* 34:1045–1079.
- Moore, E.F. 1956. Gedanken-experiments on Sequential Machines. In *Automata Studies*. Number 34 in “Annals of Mathematical Studies” Princeton University Press pp. 129–153.
- Nareyek, A. 2004. “Computer Games – Boon or Bane for AI Research.” *Künstliche Intelligenz* pp. 43–44.
- Rao, R.P.N. & A.N. Meltzoff. 2003. Imitation Learning in Infoants and Robots: Towards Probabilistic Computational Models. In *Proc. AISB 2003 Convention: Cognition in Machines and Animals*. Aberystwyth, UK: .
- Ritter, H., T. Martinetz & K. Schulten. 1992. *Neural Computation and Self-Organizing Maps*. Addison-Wesley.
- Schaal, S. 1999. “Is Imitation Learning the Route to Humanoid Robots?” *Trends in Cognitive Sciences* 3(6):233–242.
- Schaal, S., J. Peters, J. Nakanishi & A. Ijspeert. 2003. Learning Movement Primitives. In *Proc. Int. Symposium on Robotics Research*. Siena, Italy: .
- Sklar, E., A.D. Blair, P. Funes & J. Pollack. 1999. Training Intelligent Agents Using Human Internet Data. In *Proc. 1st Asia-Pacific Conference on Intelligent Agent Technology (IAT-99)*.
- Spronck, P., I. Sprinkhuizen-Kuyper & E. Postma. 2002. Improving Opponent Intelligence through Machine Learning. In *Proc. 3rd Int. Conf. on Intelligent Games and Simulation (GAME-ON’02)*.
- Thoroughman, K.A. & R. Shadmehr. 2000. “Learning of action through adaptive combination of motor primitives.” *Nature* 407:742–747.
- Thureau, C., C. Bauckhage & G. Sagerer. 2003. Combining Self Organizing Maps and Multilayer Perceptrons to Learn Bot-Behavior for a Commercial Computer Game. In *Proc. GAME-ON*.
- Thureau, C., C. Bauckhage & G. Sagerer. 2004a. Learning Human-Like Movement Behavior for Computer Games. In *Proc. 8th Int. Conf. on the Simulation of Adaptive Behavior (SAB’04)*.
- Thureau, C., C. Bauckhage & G. Sagerer. 2004b. Synthesizing Movements for Computer Game Characters. In *Pattern Recognition*, ed. C.E. Rasmussen, H.H. Bühlhoff & et. al. Vol. 3175 of LNCS Springer.
- Yannakakis, G. N. & J. Hallam. 2004. Evolving Opponents for Interesting Interactive Computer Games. In *Proc. 8th Int. Conf. on the Simulation of Adaptive Behavior (SAB’04)*. pp. 499–508.

COOPERATIVE POPULATION BASED INCREMENTAL LEARNING

Colin Fyfe and Tzai Der Wang,
Applied Computational Intelligence Research Unit,
The University of Paisley,
Scotland.

Colin.fyfe@paisley.ac.uk, Oligopoly_game@yahoo.com.tw

KEYWORDS: Evolution of cooperation, Iterated prisoners' dilemma.

ABSTRACT

We show that cooperation can be evolved in the N-person Iterated Prisoners' Dilemma using the Population Based Incremental Learning Algorithm but that the algorithm often finds a local rather than global optimum. We show how a simple extension to the algorithm enables cooperation more reliably to be found and motivate this algorithm by recent challenges to conventional evolutionary theory.

INTRODUCTION

A central thrust in modern computer games is making the computer player, the "AI", more intelligent. This is usually construed as giving the AI some adaptability – it must respond to situations within the game, learning as the game progresses. One of the central methods of modern computational intelligence is Evolutionary Algorithms, particularly the Genetic Algorithm (GA). GAs mimic the process of evolution in order to evolve good solutions to a problem. The problem is often thought of as lying in a search space and the GA blends exploration of the search space with exploitation of the solutions found so far which are better than the others. There is a balance between these two aspects: too much exploration leads to a lengthy search; too much exploitation leads to premature convergence to a less-than-optimal solution.

GAs have been used to create intelligent behaviours in game playing (see e.g. [Champanand, 2004] for some recent work). In this paper, we discuss an abstraction of the Genetic Algorithm known as Population Based Incremental Learning and use it on the standard game, the N Persons Iterated Prisoners' Dilemma. We show how an extension of the basic algorithm can be used to create nice but retaliatory rules which are best in the context of evolving cooperation.

THE PRISONERS' DILEMMA

The Prisoners' Dilemma is a well-known game from mathematical Game Theory. Two men, charged with a joint violation of a law, are held separately by the police. Each is told that,

1. *if one confesses and the other does not, the former will be given a reward and the latter will be fined.*
2. *if both confess, each will be fined.*

At the same time, each has good reason to believe that

3. *if neither confesses, both will go clear.*

The players can only choose to cooperate with each other or defect. The cooperator earns "R", *reward for mutual*

cooperation when his competitor also cooperates, and only gets penalty "S", *sucker's payoff*, when the other defects. A defector will obtain "T", *the temptation to defect*, when the competitor cooperated. However, if both players choose to defect, they both are fined "P", *the punishment for mutual defection*. There are two constraints on the payoffs that the prisoners' dilemma should satisfy,

1. $T > R > P > S$
2. $R > (S+T)/2$

This is a 2X2 non-zero sum non-cooperative game. "Non-zero sum" indicates that the benefit obtained by one player is not necessarily the same as the penalty received by another player at the same round and "non-cooperative" means that no pre-play communication is allowed between the players. [Colman, 1982] [Rapoport, 1966]

It is in each participant's best interests to defect – this maximises his/her gain no matter what the other does– but if both participants cooperate, their joint gain exceeds the sum of any possible individual gains should both or either defect.

The commonsense argument for defecting goes like this: "A prisoner's dilemma is a simultaneous choice. There is no way that your choice can affect the other player's choice. So, the situation is simple. No matter what the other player does, you're better off by defecting. That means you should defect." The commonsense argument for cooperating does like this: "The two players' situations are identical. It is unrealistic for one to expect to take advantage of the other by defecting. Assuming that the players are both rational, they should decide on the same strategy. The two realistic outcomes are mutual cooperation and mutual defection. Both prefer the cooperative outcome, so that's what they should do, 'cooperate'."

| Number of Cooperators | 0 | 1 | 2 | ... | N-1 |
|-----------------------|-------|-------|-------|-----|-----------|
| Cooperate | C_0 | C_1 | C_2 | ... | C_{N-1} |
| Defect | D_0 | D_1 | D_2 | ... | D_{N-1} |

Table 1: The payoff matrix for a single prisoner in a population of N players. There may be 0,1,...,N-1 cooperators in the remainder of the population .

Robert Axelrod [Axelrod, 1987] developed the iterated prisoner dilemma (IPD) to describe the appearance of cooperation. He has shown how a Genetic Algorithm can be used to evolve cooperation between the prisoners. In Axelrod's tournaments, there was a property, which distinguished the relatively high-scoring entries from the relatively low-scoring entries. This is the property of

| | | | | | | | |
|---------|----|-----|-----|-----------|-----------|-----------|-----------|
| Round | R1 | R2 | R2 | R3 | R3 | R3 | R3 |
| History | - | D | C | DD | CD | DC | CC |
| Length | 1 | N+1 | N+1 | $(N+1)^2$ | $(N+1)^2$ | $(N+1)^2$ | $(N+1)^2$ |

| | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| R4 |
| DDD | CDD | DCD | CCD | DDC | CDC | DCC | CCC |
| $(N+1)^3$ |

Table 2 Chromosome length for a memory of 4 rounds

being *nice*, which is to say never being the first to defect. Another property of the best decision rules is **forgiveness**.

This can be described as the property that a rule has to cooperate when the other resumes cooperation. A rule that punishes the opponent's defection is a *retaliatory* rule. For example, Tit for Tat, a rule which says just do what your opponent did in the last round, is a retaliatory rule, as it punishes the opponent's defection by defecting in the next round. However, it also a forgiving rule as it returns to cooperation with only one cooperation from the opponent.

The payoff matrix can be represented as in Table 1 which shows the gain for a single prisoner in a population of N-players. It is important to note that the return is dependent on the actions of the other N-1 players in the population. The term C_i (D_i) refers to the payoff to the current strategy if it cooperates (defects) when there are i other cooperators in the population. This payoff determines the fitness function in our simulations.

The payoff matrix of the NIPD must satisfy,

1. It pays to defect: $D_i > C_i$ for all i in $0, \dots, N-1$.
2. Payoffs increase when the number of cooperators in the population increases: $D_{i+1} > D_i$ and $C_{i+1} > C_i$ for all i in $0, \dots, N-1$
3. The population as a whole gains more, the more cooperators there are in the population: $C_{i+1} > (C_i + D_{i+1})/2$ for all i in $0, \dots, N-2$. Notice that this last gives a transitive relationship so that the global maximum is a population of cooperators.

Colman [Colman, 1982] has indicated that the NIPD is qualitatively different from the 2IPD and that certain strategies that work well for individuals in the 2-Prisoners' Dilemma fail in larger groups.

We have shown that the chromosome in Table 2 can be used to evolve solutions to the NIPD. The first bit determines if the strategy will cooperate ("1") or defect ("0") on the first round. In round 2, the decision is taken dependent on whether the strategy cooperated or defected in round 1 (second and third blocks respectively). And then in each block, the bit that determines whether to cooperate or to defect in this round is identified dependent on the number of cooperators in round 1. This may vary from 0 to N, so N+1 bits are necessary. In round 3 the decision depends on the strategy's previous decisions: DD, CD, DC, or CC. Within each block there are $(N+1)*(N+1)$ histories determined by the number of cooperators in each of the first two rounds, i.e., 00, 10, 20, 30, 01, 11, 21, and so on.

In this paper, we consider the N-player Prisoners' Dilemma game where $N > 2$. Previous experiments [Wang, 2002] have

shown that the $N=7$ is a critical value: if $N > 7$, cooperation is very difficult to achieve.

When GAs are used, many simulations [Wang, 2002] show local fitness optima in which players mutually defect at the first few rounds then mutually cooperate after that. These situations appear no matter which selection operators are used: it is an open question as to whether the problem is innate in genetic procedures.

POPULATION-BASED LEARNING

The Population-Based Incremental Learning (PBIL) method was originally proposed by Baluja [Baluja, 1994]. This approach is a combination of Genetic Algorithms and the Hill-Climbing or gradient ascent method.

The object of the PBIL is to create a probability vector which, when sampled, reveals high quality solution vectors with high probability with respect to the available knowledge of the search space.

The vector of probabilities is updated at each generation. The change in the probability vector is towards the better individuals of the population and away from the worst individuals. Each bit or component of the probability vector is analysed independently. This method does not pretend to represent all the population by means of a vector, but it rather introduces a search based on the better individuals.

The two parameters in the PBIL method that must be defined for each problem are the population size (number of samples generated) and the learning rate (LR). The process is as follows:

1. Create a vector whose length is the same as the required chromosome and whose elements are the probabilities of a "1" in the corresponding bit position of the chromosome. This vector is initialised with 0.5 in all positions.
2. Generate a number of samples (we use 100 in the simulations below) from the vector where the probability of a 1 in each bit position of each vector is determined by the current probability vector.
3. Find the fittest chromosomes from this population.
4. Amend the probability vector's elements so that the probability of a "1" is increased in the positions in which the fittest chromosomes have a 1.

This process ends when each element of the vector approaches 1 or 0. The upgrade of the probability vector is done using

$$\Delta p_i = \eta (E_{best}(chromosomes_{ji}) - p_i)$$

where Δp_i is the increment to the vector's i^{th} element, η is the learning rate, $E_{best}(chromosomes_{ji})$ is the mean value of the i^{th} chromosome bit taken over the fittest chromosomes and p_i is the probability of a 1 in position i in the current generation.

We may show that the PBIL can be used to evolve cooperation but that we often get only partial cooperation e.g. the whole population may defect at round 2 but regain cooperation at subsequent rounds. A typical result is shown in Figure 1.

The numbers of competition outcomes stay relatively steady throughout hundreds of generations, especially the number of 0-cooperators and 7-cooperators. That means the players either all mutually defect or they all mutually cooperate. We note that they also try other competition outcomes in some rounds. We check the probability vector at the end of simulation from the trial shown in Figure 1 and find that the first left bit is 0 and that only some bits at the right part of the probability vector are not 1 or 0. That means this population almost converges and the difference between strategies that individuals have is very small. However, the created individuals under this probability vector defect at the first round before finding a route to cooperation. Because of the lack of the exploration capability in the converged population, the population can not move out but sinks into this local fitness optimum and remains there throughout the generations. Since there are so many local fitness optima that the simulation could sink into in this set of simulations, we consider the original PBIL with the learning rate=0.1 is not a suitable search method to simulate the 7-player IPD game.

Also, there is a hitchhiker problem which happens in PBIL. The bits on the chromosomes of the winners which might not be used during the competition can be changed during the simulation. The reason for these individuals' success is not affected by what values they have on these bits which

have never been used. The upgrade process does not consider the factors for success in the winners and changes all the probabilities on the vector. It updates every position of the probability vector using the information from the winners even though some bits of the winners have not been used during previous competitions and might also lead to worse results if they were used. This hitchhiker problem

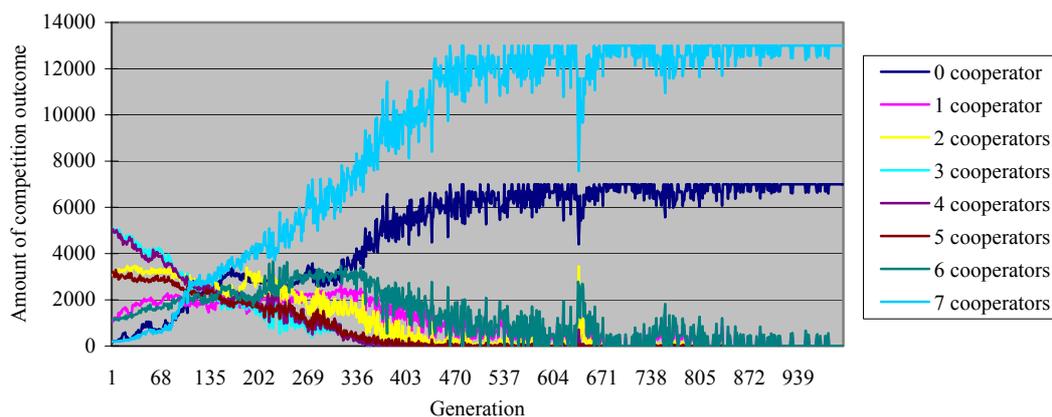


Figure 1. Convergence of the PBIL algorithm, $\eta=0.1$

decreases the performance and quality of the search.

The cooperative PBIL algorithm is designed to meet this problem.

THE COOPERATIVE PBIL ALGORITHM

Ben-Jacob [Ben-Jacob, 1998a] [Ben-Jacob, 1998b] challenges the new Darwinian view of the chromosome as static code which may be decoded to create phenotypes. Ben-Jacob notes the experimental evidence that mutations in bacteria exist prior to there being a need for or benefit from such mutations, however, while this might suggest that random mutations do exist in populations, it does not preclude non-random mutations. Ben-Jacob [Ben-Jacob, 1998b] then notes experimental evidence that bacteria may demonstrate genomic changes other than during replication. There is further evidence that a specific mutation will occur in high frequencies only when needed to remove the selective pressure and further that such mutations will not trigger any other mutations. It is the view of [Ben-Jacob, 1998a] [Ben-Jacob, 1998b] that major evolutionary changes occur in response to a population meeting paradoxical environmental conditions; Ben-Jacob contrasts paradoxes with problem solving - the normal processes of evolution may be sufficient when a population meets a problem in this environment but a paradox calls for a more directed evolution in which change is directed by the population as a whole onto individuals within the population. Let us consider that paradoxes in the NIPD are created when a prisoner wishes to respond nicely to other cooperators but retaliate against the defectors. However with only one action, each prisoner can only respond in the same way to all.

Thus we design an improved PBIL, the Cooperative PBIL Algorithm, which will use the environmental input to direct the evolutionary process but which will not cause unnecessary side effects in genes which need not be changed in response to environmental paradoxes. This improved algorithm is described below.

1. Find the highest fitness chromosomes as the winner just as in the PBIL algorithm.
2. Select representative chromosomes among the population randomly. This step and step 3 were repeated 20 times in the simulations below.

3. The winners compete against these representative chromosomes and the bits of the chromosomes used through these competitions are recorded.
4. Use the recorded bits in the chromosomes of the winners to upgrade the values of the same positions of the probability vector.
5. Recreate the new population randomly under the probabilities of the probability vector.

This procedure guarantees that only the bits which contribute to the success of the winners will be upgraded and the other unused bits will not influence the probability vectors. If the bit has not been used throughout the generations, the probability of this bit still remains around 0.5 and allows continuing exploration of these bits by the population if needed. The creation of the unused strategies encoded on these bits creates variation in the population which is not affected by the previous competitions which did not involve them.

The crucial difference between the Cooperative PBIL algorithm and the ordinary PBIL algorithm is that there is no genetic drift in this algorithm: only those bits which are actually used in determining fitness are used in modifications to the chromosome.

If we increase the learning rate, exploitation works more strongly and exploration becomes weaker. The population converges quickly and the search may improve in quality. However, if learning experience and selection moves the chromosome in a wrong search direction, the search may fall into a local fitness optimum due to weak exploration. We may expect that there are many trials in which the simulation falls into a local fitness optimum, and so we should find a suitable learning rate to keep

search works well in finding mutual cooperation. We investigate how a higher learning rate affects the appearance of mutual cooperation and which rate is the best for the simulated NIPD game.

Firstly, we use a learning rate of 0.2 to investigate the 7-IPD via the Cooperative PBIL and find that in 10 of 25 trials, we can achieve total mutual cooperation. The number of successful trials to achieve mutual cooperation is slightly lower than when we use a learning rate of 0.1. We also note the strong effect of exploitation in that many trials achieve a local fitness optimum and maintain this population of very stable strategies. 6 of these 10 simulations achieve the global fitness. There are 4 trials which achieve mutual cooperation in a local fitness optimum in which the players mutually defect at the first round or at the first two rounds. Comparing the simulations which achieve the global fitness optimum when we use 0.2 as the learning rate with that when 0.1 is used, we find that this stronger exploitation, due to the higher learning rate, does not improve the performance of the search. There are also some trials which still need several hundred generations to evolve mutual cooperation (See Figure 2). A qualitatively similar behavior is seen in Figure 3 from a different simulation with the same parameter set: there is a period of turmoil, sometimes with quieter periods embedded followed by the emergence of cooperation.

From the simulation using the PBIL with different learning rates, we can draw a graph to show the influence of the learning rate on the emergence of

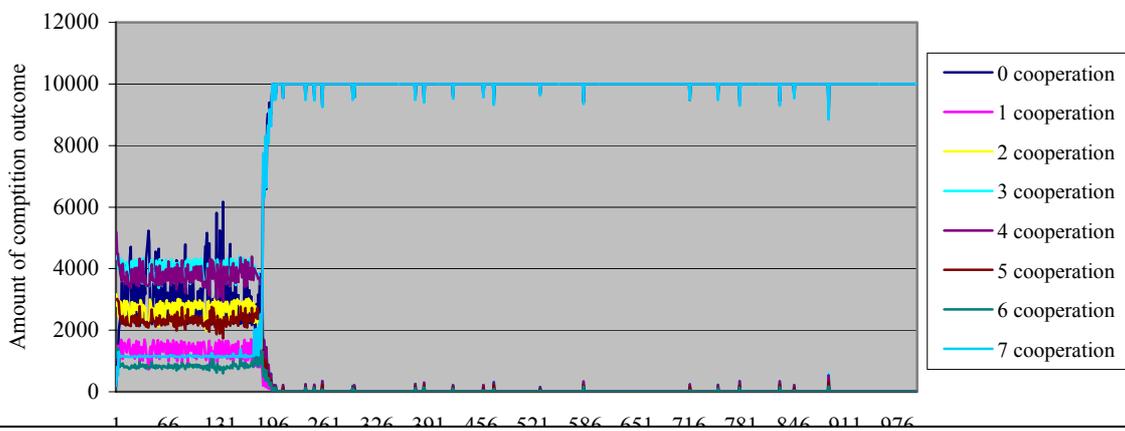


Figure 2 The cooperative PBIL, $\eta=0.2$

the balance between exploration and exploitation to keep the performance high and the quality of search satisfactory. From the simulation of the last section, we know that, with $\eta=0.1$, the

cooperation (See Figure 4). We see that too high or too low a learning rate applied to the Cooperative PBIL makes the performance worse. Too high a learning rate

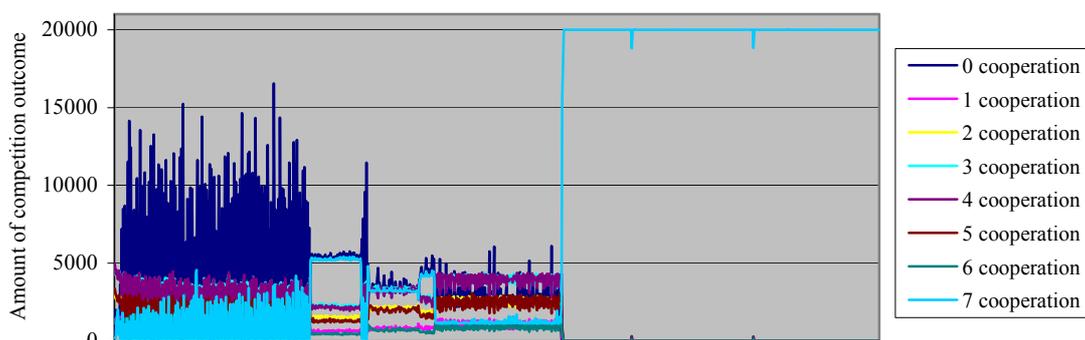


Figure 3 The cooperative PBIL, $\eta=0.2$

leads to a lack of exploration and too low a learning rate means little information is passed to the next generation. Both extremes lead to a bad performance, i.e., mutual cooperation is not achieved. However, from the experimental results, we can not say which interval of values of learning rates is optimal for a search with the PBILs. Two similar learning rates can give different results. Therefore, we do not suggest which learning rate is optimal for the investigation of the evolutionary game. However, almost half the trials achieve mutual cooperation when a learning rate of 0.5 is used for the investigation with the Cooperative PBIL. We may say that this performance is acceptable and comparable for the investigation of the 7-IPD with other evolutionary methods.

Even though the Cooperative PBIL is an improved PBIL method, the optimal learning rates for the Cooperative PBIL and the standard PBIL are different. The investigation of the NIPD shows that a good learning rate for a simulation with the Cooperative PBIL does not work well with the standard PBIL. Performance is best with the Cooperative PBIL when the learning rate is 0.5. However, with the standard PBIL, the best learning rate is 0.05.

motivated by recent challenges to evolutionary theory which considered that under crisis situations, evolution can step outside its normal boundaries in order to solve paradoxes which might not otherwise be solved within the evolutionary process.

REFERENCES

- Axelord, R., 1990. *The evolution of cooperation*, Penguin Books.
- Baluja, S., 1994. *Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning*, Carnegie Mellon University Technical Report. CMU-CS-94-163.
- Ben-Jacob, E., 1998a. *Bacterial wisdom*, Physica A, 249:553-557.
- Ben-Jacob, E., 1998b. *Bacterial wisdom, godel's theorem and creative genomic webs*. Physica A, 248:57-76.
- Champanand, A. J., 2004, *AI Game Development*, New Riders Publishing.
- Colman, A. M., 1982. *Game Theory and Experimental*

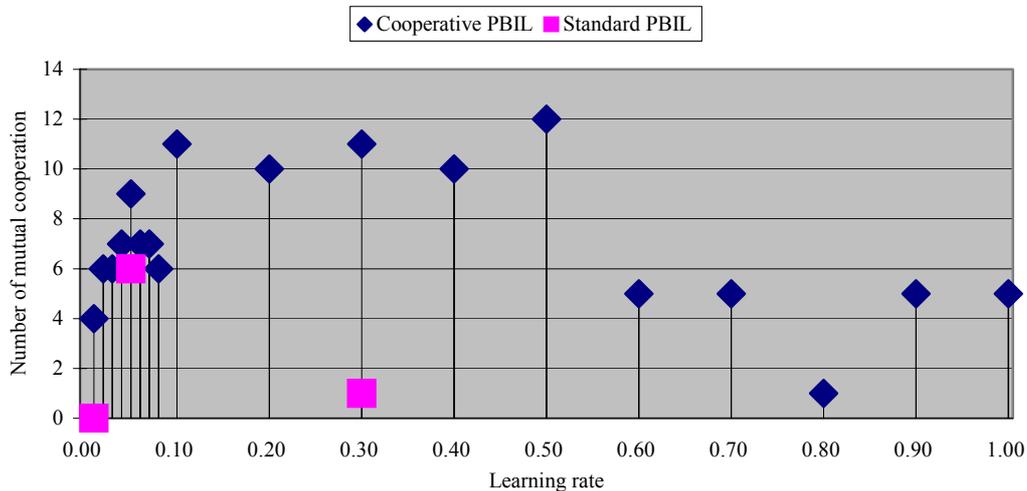


Figure 4. The number of cooperators varies with the learning rate

CONCLUSION

We have previously compared Genetic Algorithms using roulette-wheel and rank selection with Population Based Incremental Learning and shown that the last gives comparable performance with the best GA methods.

In this paper, we have amended our PBIL algorithm so that it updates only those positions on its probability vector which actually contributed to the fitness of the best chromosomes. This resulted in an elimination of the hitch-hiking problem which results in some bits being upgraded because, by chance, they happened to be a particular value on the best chromosome(s). We showed that the resulting algorithm gave improved performance in terms of the evolution of cooperation. Figure 4 shows that we do not need to be as careful in our choice of learning rate as we have to be with the standard PBIL algorithm. The algorithm was

Games, Pergamon Press, Oxford, England.

Rapoport, A., 2001. *N-Person Game Theory, Concepts and Applications*, Dove edition.

Wang, T.-D., 2002. *The Evolution of Cooperation in Artificial Communities*, PhD thesis, the University of Paisley.

Author Listing

Author Listing

| | | | |
|-------------------|-----------------|--------------------|---|
| Al-Dabass, D. | 60, 66, 89, 187 | Gold, J. | 193 |
| Andersen, P. B. | 96 | Gough, N. E. | 5, 25, 55, 122, 221, 236, 248, 263, 293, 301 |
| Anderson, D. | 221, 248, 301 | Graepel, T. | 193 |
| Anderson, E. F. | 203 | Gressier, E. | 134 |
| Arnold, S. E. | 301 | Grünvogel, S. M. | 109 |
| Arokiasamy, A. | 89 | Guha, R. K. | 41, 49 |
| Bancroft, M. | 60 | Hafsteinsson, V. | 379 |
| Bangsø, O. | 96 | Hallam, J. | 240 |
| Bartle, R. A. | 128 | Hartley, T. | 55, 293 |
| Bateman, R. M. | 170 | Hastings, E. J. | 41, 49 |
| Bauckhage, C. | 402 | Helgesen, A. S. | 306 |
| Beugnard, A. | 134 | Herbrich, R. | 193 |
| Björnsson, Y. | 379 | Hirokazu Notsu | 216 |
| Black, M. | 29 | Hiromichi Fukutake | 20 |
| Bornes, V. V. | 248 | Hogg, R. | 360 |
| Bottaci, L. | 263 | Jacobi, D. | 221, 248, 301 |
| Broekens, J. | 208 | Jankovic, L. | 371 |
| Burley, M. A. | 25 | Jensen, F. V. | 96 |
| Cant, R. | 66, 71, 187 | Jensen, O. G. | 96 |
| Cazenave, T. | 298 | Jóhannsson, Á. | 379 |
| Chabridon, S. | 134 | Jónsson, E. | 379 |
| Chang-Woo Chu | 139 | Kamphuis, A. | 285 |
| Charles, D. | 29, 163 | Kantardzic, M. | 221 |
| Cheng, D. C. | 36 | King, G. | 360 |
| Chiong, R. | 371 | Kocka, T. | 96 |
| Churchill, J. | 187 | Koichi Nijjima | 20, 216 |
| Connelly, T. M. | 352 | Krzywinski, A. | 306 |
| Cowling, P. | 360 | Kumar, P. | 263 |
| Craipeau, S. | 134 | Langensiepen, C. | 66, 71 |
| Cunningham, P. | 104 | Leen, G. | 278 |
| Da Fonseca, J. B. | 330 | Legout, M-C. | 134 |
| Davies, N. P. | 248 | Livingstone, D. | 273, 342 |
| DeGroot, D. | 208 | Macleod, A. | 268 |
| Delaney, D. | 76, 144 | Maddock, S. | 81 |
| Doboli, S. | 170 | Mańdziuk, J. | 182 |
| Doughty, M. | 338 | Marshall, D. | 76 |
| Duthen, Y. | 397 | Masuch, M. | 114, 347 |
| Ehlert, P. | 175 | McCoy, A. | 144 |
| Elmaghraby, A. | 221 | McGlinchey, S. J. | 163, 273 |
| Eun-Hee Lee | 139 | McLellan, E. | 352 |
| Fairclough, C. R. | 104 | McLoone, S. | 76, 144 |
| Fennell, R. | 360 | McMonnies, A. | 342 |
| Fortuna, H. S. | 365 | Mehdi, Q. H. | 5, 25, 55, 122, 221, 236, 248, 263, 293, 301 |
| Foster, G. | 71 | Meredith, M. | 81 |
| Francik, J. | 228 | | |
| Fyfe, C. | 259, 278, 409 | | |

| | | | |
|-------------------|--------------|-------------------|---------|
| Mesit, J. | 41, 49 | Stansfield, M. | 352 |
| Mooijekind, M. | 285 | Sung-Ye Kim | 139 |
| Mossakowski, K. | 182 | Sutherland, J. | 352 |
| Nacke, L. | 347 | Szarowicz, A. | 228 |
| Natkin, S. | 25, 109, 263 | Tambellini, W. | 397 |
| Nieuwenhuisen, D. | 285 | Thawonmas, R. | 36, 311 |
| Overmars, M.H. | 14, 285 | Thorn, D. | 150 |
| Palmer, I. | 155 | Thornton, J. S. | 317 |
| Pelton, M. | 13 | Thureau, C. | 402 |
| Periyamayagam, R. | 89 | Tzai Der Wang | 409 |
| Pfeiffer, M. | 384 | Vega, L. | 109 |
| Ponsen, M. | 389 | von Borries, V. | 221 |
| Purdy, J. H. | 317 | Ward, T. | 76, 144 |
| Qiuxia Liang | 175 | Weiqin Chen | 306 |
| Ragade, R. | 221 | Wen, Z. | 236 |
| Ramsay, J. | 352 | Wiklund, M. | 325 |
| Rhodes, D. | 66 | Williams, E. | 155 |
| Rhodes, P. | 360 | Yannakakis, G. N. | 240 |
| Röber, N. | 114 | Yao Zhai | 311 |
| Rothkrantz, L. | 175 | Yoshihiro Okada | 20, 216 |
| Sagerer, G. | 402 | Young, R. M. | 8 |
| Sanza, C. | 397 | Yuki Konno | 311 |
| Sephton, N. | 360 | Zeng, X. | 122 |
| Simatic, M. | 134 | | |
| Slater, S. | 150 | | |
| Spronck, P. | 389 | | |
| Spyridou, E. | 155 | | |

