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 predicable, 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.

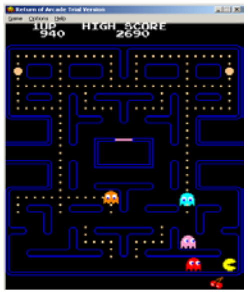

Figure 1: A screenshot of Microsoft Return of the Arcade Pac-man game.

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.

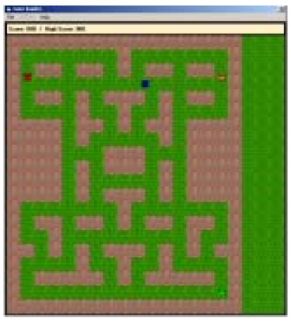

Figure 2: A Screenshot of the Gem Raider game.

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 NPCs 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.

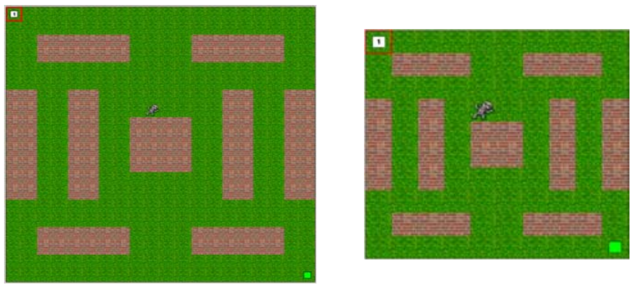

Figure 3: Screenshots of the 20x20 and 10x10 maps used to produce the results in table 1.

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 Euclidian 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.

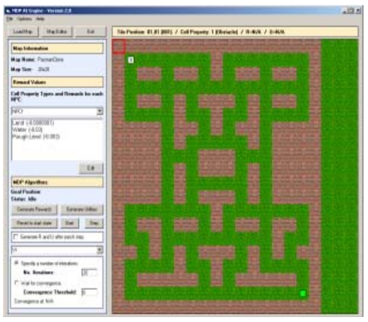

Figure 4: Screenshot of the map used to produce the results for the region experiments.

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.

DISCUSION

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 that 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=466 35&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:zSzzSzftp.ics.uci.eduzSzpubzSzCSP-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). Combinging 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).