ONLINE LEARNING FROM OBSERVATION FOR INTERACTIVE COMPUTER 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

ABSTRACT

The research presented in this paper describes an architecture, which enables an agent to predict an observed entity's actions (most likely a human's) online. Case-based approaches have been utilised by a number of researchers for online action prediction in interactive applications. Our architecture builds on these works and provides a number of novel contributions. Specifically our architecture offers a more comprehensive state representation, behaviour prediction and a more robust case maintenance approach. The proposed architecture is fully described in terms of interactive simulations (specifically first person shooter (FPS) computer games); however it would be applicable to other interactive applications, such as intelligent tutoring and surveillance systems. We conclude the paper by evaluating our proposed architecture and discussing how the system will be implemented.

INTRODUCTION & MOTIVATION

Action games, in particular first person shooters (FPS) are one of the most popular game genres on the market today (Bauckhage and Thurau, 2004). These games usually involve running around and using deadly force against an enemy (Laird and Lent, 2001). Examples include Half Life, Quake and Unreal. AI opponents in actions games are becoming more sophisticated and are beginning to approximate the game play of human players (Laird and Lent, 2001); however non-player characters (NPCs) (agents) in these types of games tend to rely on pre-programmed scripts and finite state machines to achieve this behaviour. In fact almost all commercial computer game AI (regardless of genre) is controlled by scripts or finite state machines (Cass, 2002). The use of these techniques can result in poor AI, which is predicable, less believable and can be exploited by the human player (Spronck *et al*. 2003). This reduces the enjoyment for the player and can result in them preferring human controlled opponents (Schaeffer, 2001). Unfortunately human opponents are not always available or appropriate, so continuing to improve computer game AI is a desirable endeavour. In addition with the on going developments in computer hardware, more processing time (Cass, 2002) is now available for a computer game's AI. In the future games developers will be able to employ more sophisticated academic AI techniques, to make the NPCs in their computer games more human-like and responsive to the game player.

Learning from observation is one such sophisticated AI approach. Traditionally human behaviour models have been developed by acquiring knowledge from subject matter experts (SMEs) (Fernlund, 2002). This is a difficult and time-consuming process, which learning from observation attempts to automate. In AI research there is no hard and fast definition for the term "learning from observation". Most literature refers to it as a method of learning the behaviour of another entity by observing its actions (Fernlund, 2002). This is a broad description and means that learning from observation can be applied to many types of applications / problems, using a variety of learning techniques. In the area of computer games there have been a number of learning from observation research efforts. Lent and Laird implemented their symbolic KnoMic system with the Soar architecture, in an air combat domain and have created a reactive Quake II bot (Laird and Lent, 2001). Games researchers have also examined sub-symbolic approaches to the problem. One of the most interesting is Thurau et al. (2004) system, which makes use of artificial neural network (ANN) techniques to model human behaviour in order to create a reactive NPC for Quake II. This system learns offline from recorded Quake II game sessions, between human players. McGlinchey (2003) also uses a similar approach to create more humanlike opponents for the Pong game. A system called GoCaps (Game observation capture) (Alexander, 2002) also learns by observing play sessions between humans, but rather than from recorded game sessions the system learns in a training mode and is directly controlled by a human player / expert.

All these approaches produce believable behaviour for NPCs through learning from observation. However, these types of systems still encounter the same limitations as current FSM and scripting techniques. This is highlighted by McGlinchey (2003) who states that humans may act differently in identical situations, but his system is always completely deterministic. In addition Laird found that games developers were impressed by his behaviour modelling system, but would invariably ask, "Does it anticipate the human player's actions?" (Laird, 2001). Enabling an agent to predict and adapt its behaviour through learning from observation (I.e. rather that simply using observations to learn, we want to model another entity's behaviour in order to predict and adapt) would overcome the limitations facing traditional and some sophisticated computer game AI approaches. As a result the primary goal of our research is to explore and evaluate how learning from observation can be used to develop agents for interactive applications (specifically FPS computer games), which are able to intelligently adapt their behaviour in order to improve their chances of success in cooperative or adversarial situations.

We begin this paper by defining the computer game scenario that will form the focus of our research. After that we discuss background and related work. We then take the first steps to achieving our goal by developing an architecture inspired by existing case-based prediction research and proposing a number of novel contributions which improve these types of systems. Finally we conclude the paper by evaluating our

architecture and discussing how the proposed system will be implemented.

COMPUTER GAME SCENARIO

The test-bed for our adaptation research is FPS computer games. We will initially focus our work on applying action and behaviour adaptation to combat situations in these types of games (as illustrated in Figure 1); however future work will incorporate adaptation at task / tactic and goal levels.

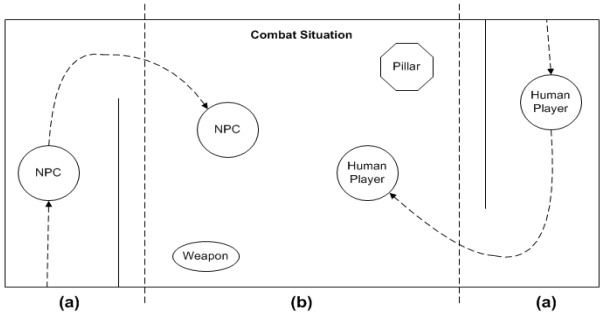

Figure 1: Computer Game Scenario - A combat situation between an NPC and a human player.

(a) The NPC (and the human player) are opponents and are moving through the game environment in search of items, such as weapons.
(b). When the NPC comes in close proximity to an opponent it enters a combat situation and has to decide what actions to perform.

When an NPC is not in a combat situation it executes long-term plans, goals and certain tactics, such as avoiding combat in order conserve health (Figure 1 (a)). High-level goals include searching for items (i.e. weapons), navigating to a location, searching for opponents or capturing a flag. When the NPC enters an area near where an opponent is located, Figure 1 (b), the NPC in effect enters a combat situation and has to decide quickly in real time what actions, behaviours and tactics to take. Actions can be considered atomic events (e.g. move left, move forward). Behaviours can be classed as identifiable components or building blocks (Khoo *et al*. 2003) of a tactic / task. For example the 'Camp' tactic could consist of 4 behaviours, find hidden location, select weapon, wait for target and shoot target (Lent and Laird, 1999), some of which could have sub-behaviours. Also it's worth noting that behaviours can be classified in a number of other ways, such as stimulus-responses (i.e. reactive behaviours), however this is not the context in which they are used in this paper. Tactics can be described as smart, localized situation handling, which depend on anticipating an enemy and a broad understanding of a scene (e.g. the layout of the environment) (Bauckhage and Thurau, 2004). Examples of tactics include strafing, camping and sniping.

It would be desirable for an NPC in this type of situation to be able to adapt and reason about an opponent's actions, behaviours and tactics in order to improve its decision making and chances of survival. For example predicting an opponent a few time-steps into the future could allow an NPC to adapt its path planning or targeting, by providing possible future locations to aim for. This would be especially useful in chase situations, where an NPC has to catch or kill an opponent. Inferring an opponent's tactics would allow an NPC to adapt its playing style, for example if an opponent prefers to snipe, the NPC could adapt its behaviour to prefer cautious tactics and to search for sniper locations. Any approach would however have to operate in real time, as humans do, in order to provide useful adaptation.

BACKGROUND & RELATED RESEARCH

In order for our proposed system to adapt successfully at different levels of behaviour (i.e. goal, task or action) and be able to integrate with existing frameworks we will make use of a multi-layered adaptation approach. In general, system decision models can be categorised / organised into a hierarchy that follows human behaviour models, such as the psychological hierarchy of human behaviour (Hollnagel, 1994 in Thurau *et al*. 2004) or cognitive models (Funge et al. 1999 in Dinerstein and Egbert 2005). Kerkez and Cox (2001) define prediction in terms of local (predicting the agent's immediate course of action) and global (predicting an agent's goals and plans). Dinerstein and Egbert (2005) make use of a layered behaviour model, which has goal selection on the top, task selection on the second level and action selection on the bottom. For our research we will follow this style of behaviour model, however initial work will focus on action and behaviour prediction. Formulating and integrating other layers will be discussed in future work; however each layer in our proposed system will offer different levels of adaptation, using appropriate learning techniques, which reflect the temporal granularity of the level.

The research described in this paper expands on existing work in the area of intelligent online behaviour adaptation. Work in this field has made use of a variety of approaches, such as prediction (Dinerstein *et al*. 2005), user modelling (Schaeffer *et al*. 1999), anticipation (Laird, 2001), reinforcement learning (Spronck *et al*. 2003) and plan recognition (Kerkez and Cox, 2003). We propose to achieve action and behaviour adaptation through prediction and plan recognition approaches. We will make use of incremental case-based techniques to model an observed entity's actions in order to predict their behaviour. The use of these techniques is not new, Dinerstein *et al*. (2005), Fagan and Cunningham (2003) and Kerkez and Cox (2001) have all utilised them, however they allow us to take an observation based approach to modelling actions. It is worth noting that there have also been many other researchers who have explored case-base reasoning, plan recognition and prediction; however the works identified above are the key research efforts that come closest to meeting the requirements of online action prediction in interactive computer games. In addition a number of online adaptation systems have been successfully implemented in real-time games, for example Spronck *et al*. (2003). However the majority of these works have focused on a single layer adaptation approach and do not solve the issue of action prediction, which we are addressing.

Plan Recognition

Plan recognition (Mao and Gratch, 2004) systems could be classed as being able to learn from observation, as the recogniser observes agents and infers their behaviour. However in a large number of systems, the plan library is specified *a priori*, by some external agent who is often the

system designer (Kerkez and Cox 2001). Existing work in the area, that addresses this issue and is applicable to our target domain of FPS games includes Kerkez and Cox (2001) and Fagan and Cunningham (2003). Kerkez and Cox focus on developing an incremental case-based approach to plan recognition which can deal with incomplete plan libraries and can create new plans based on observations. In addition they introduce the idea of storing intermediate states as part of a plan and address the problem of having too much state information by incorporating an indexing scheme. The system is successful in the blocksworld, logistics and Extended-STRIPS domains; however it is limited to a discrete environment state representation and is not used to predict human behaviour / plans.

Fagan and Cunningham (2003) also focus on developing an incremental case-based approach to plan recognition, which is based on Kerkez and Cox's work. They implement their system relatively successfully in a Space-Invaders computer game and model human behaviours, however the environment in their system is limited to three possible states. In more complex environments such as FPS or real time strategy games this limitation would not be practical.

Prediction

One of the most interesting approaches to prediction / anticipation in computer games has been Lairds' (2001) system for Quake II. In this work prediction is achieved by creating an internal representation of an opponent, by estimating what it is sensing and could have in its working memory. The system then predicts behaviours by using its own knowledge to select what the enemy would do, based on the estimated internal representation. This approach is successful, however it does not model human behaviour and predicts based on its own knowledge.

The research that has been most influential on our work is Dinerstein and Egberts' (2005) adaptation architecture. In particular their action prediction layer (Dinerstein *et al*. 2005) has proven very interesting. The system in their research has been designed specifically for use in real-time interactive simulations, it provides the ability to model an observed entity and predict their future actions. In order to achieve this, the system records state-action pairs of an observed entity and stores them in a case library. To predict an observed entity's actions the system finds close matches in the plan library to its current state. Once close match(es) have been found their associated actions are used to determine a predicted state. This state is then used to find new close matches in the plan library. The process is repeated for a number of time steps until a prediction is made. This system works well, however its accuracy is low when implemented in action games, because it relies on a compact state representation for fast performance. Also it has a coarse state replacement approach, which limits its ability to retain useful cases and the state space is divided into regions which limits its ability to learn novel actions. In addition at an action level the system only provides prediction of future locations. Prediction of behaviours at this level could enable an observing agent to more accurately infer future locations and behaviours.

As well as developing a robust online adaptation system Dinerstein *et al*. (2005) list 4 characteristics that are necessary for an adaptation system to work in practice. These characteristics are similar to Spronck *et al*. (2003) 4 requirements for online learning. The most important of these characteristics is speed, as slow learning, adaptation and predict techniques would render the system pointless. In order for our work to be applicable in practice we will adhere as closely as possible to all the requirements proposed by these authors.

PROPOSED ARCHITECHURE

We propose to build upon existing incremental case-based approaches to modelling an observed entity's actions, in order to predict their behaviour. Our architecture combines aspects of previous research in the area, including Dinerstein *et al*. (2005), Fagan and Cunningham (2003), Kerkez and Cox (2001) and makes a number of novel contributions to improve the capabilities of the system. The proposed architecture is described in terms of interactive simulations (specifically FPS computer games), however it would be applicable to other interactive applications that allow states and actions to be observed and defined (e.g. intelligent tutoring and surveillance systems).

During an encounter in an interactive simulation between an observing agent and an observed entity (mostly likely a human) the agent will record state-action pairs of the environment and entity's actions. Cases will be represented as single state-actions pairs for short-term action prediction and groups of state-action pairs for prediction of the observed entity's current behaviour (i.e. getting a nearby item). This will allow the system to predict at an action level and a behaviour level. The next section details the proposed state and action representation for our prediction system. The sections after that discuss how actions and behaviours will be predicted and how cases will be maintained.

Representation of States and Actions

Most plan recognition systems only include start states and goals states in their plans (Kerkez and Cox, 2003) and assume that observed actions are discrete, instantaneous and come one at a time (Kaminka and Avrahami, 2004). Dinersteins' *et al*. (2005) case-based prediction system can represent continuous or discrete variables, however it relies on a compact state representation, which helps maintain fast learning. In complex environments this will result in the state space being approximated, which may lead to significant state features being left out of the representation in order to maintain speed. For our system we propose a dual state representation, consisting of primary and secondary state definitions. The primary state space will be relatively compact and be used for the main searching and indexing of the system. Additional (or secondary) state information will be stored in environment views that correspond to specific primary states and will be used to provide a more comprehensive match of states stored in the case library to query states. Primary states will be used to determine a shortlist of states that are closest to the query state, once identified their secondary states will be compared. The state or states that have the closest combined primary and

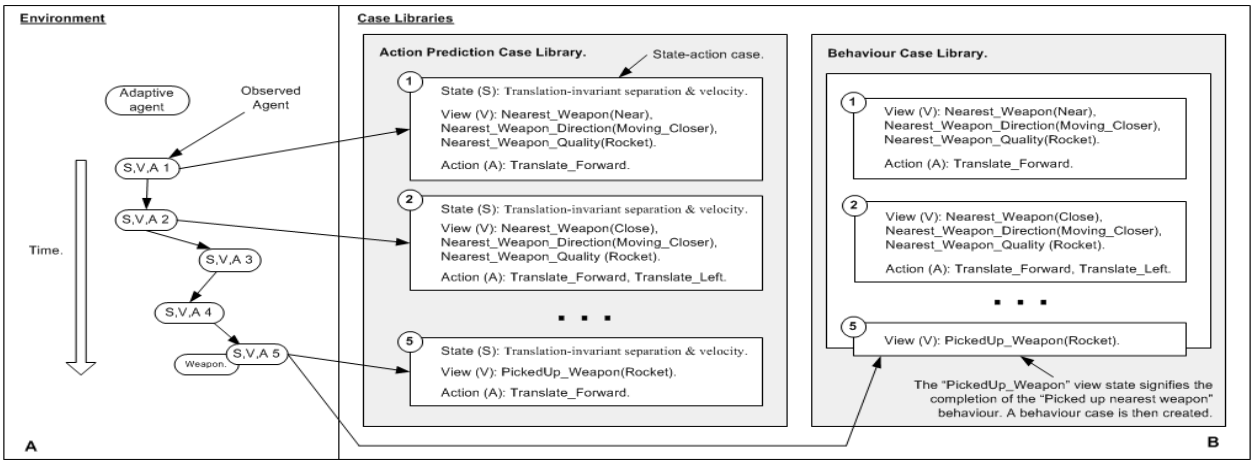

Figure 2(a): An encounter between an adaptive agent and an observed entity. As the entity moves towards a weapon the state of the environment and its action are recorded by the observing agent and stored in cases.

Figure 2(b): States and actions are stored in the case library. A behaviour case is generated when a view state, which signifies the completion of a behaviour occurs, the proceeding view-action pairs are linked together to form a behaviour case.

secondary match to the query state will be used for prediction.

Real values will represent the primary state space in our proposed system and sets of ground literals will represent the secondary state space. Dinerstein *et al*. (2005) represent their state space as a real-valued, *n*-dimensional feature vector, where a state is a point within the feature space. They propose the use of the translation-invariant separation and velocity of the observed entity to their nearest opponent to represent the state space of action games. We will also make use of this approach for primary state information. However it is worth noting that states can be represented by any relevant environment properties, using either discrete or continuous variables.

This state representation alone has proved to be relatively successful in previous work; however it lacks the detail needed to fully represent complex environments. To incorporate a more comprehensive representation, whilst still maintaining speed, we propose the use of environment views, which would consist of additional state information surrounding the observed agent. The use of views is inspired by Suliman *et al.* (2002) cognitive map and knowledge base navigation system. Each view in our proposed system will be made up of ground literals (i.e. Nearest_Weapon(Close)) and be encoded in a hierarchical representation, based on an object's distance from the observed entity. For example a pistol, which is far away from the observed entity, would have a low significance within the view, whereas a rocket that is close would have a high significance. Ordering state information within views will allow close matches to the current view to be found quickly and ranked in order of significance.

Actions in our system will be made up of an agents actuators (i.e. move forward, move left, jump). Table 1 illustrates an initial set of possible actuators an observed agent could possess and their possible outputs. An action in our proposed system would be made up of these actuators. By using this approach we will be able to capture the continuous parallel nature of actions in dynamic FPS domains. The use of actuators is based on the output action vector used by Khoo *et al*. (2002) in their behaviour based control mechanism for the Half Life action game.

Actuator	Possible Outputs
Rotate	Rotate_Left / Rotate_Right / No_Rotation
Translate	Translate_Forward / Translate_Back / No_Translation
Strafe	Strafe_Left / Strafe_Right / No_Strafe
Pitch	Pitch_Forward / Pitch_Backwards / No_Pitch
Shoot	True / False
Jump	True / False
Duck	True / False

Table 1: Possible actuators an observed agent can possess.

Case Learning

Cases in our proposed system will take the form of state-action pairs and groups of view-action pairs, which form behaviour cases. When discussing state-action pairs in this paper 'state' refers to both primary and secondary (i.e. views) state information, whilst view-action pairs refers to just secondary state information. State-action pairs will be recorded online and stored in a case library, at *t* time steps as an encounter between an agent and an observed entity progresses. Figure 2(a) illustrates this process; Figure 2(b) contains examples of state-action cases and a behaviour case. A behaviour case is generated when a view state, which signifies the completion of a behaviour occurs. The proceeding view-action pairs are then linked together to form a behaviour case. The view-action pairs represent the states and actions that lead to the completion of a behaviour.

As state-action pairs are observed useful cases are recorded in the case library, which is hierarchically partitioned to enable fast lookup and retrieval of cases. Duplicate and nearly duplicate cases will not be automatically added to the case library, however a primary state can have a number of distinct secondary states (views) associated to it, each referring to an action. This allows a state to predict different actions, depending on which view most closely matches the query state, as illustrated in Figure 3. When duplicate or nearly duplicate cases are observed, but do not replace the current case, their view state will be added to the current primary state if it is different to existing views for that state.

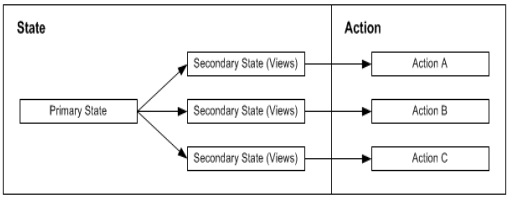

Figure 3: A primary state can link too many secondary states, which each link to an action.

Similarity assessment of views will be achieved through ground literal matching. Since ground literals in our proposed system are strictly defined, the easiest approach to computing correspondence is to match the ground literals in the query view to ground literals in stored views. Even though this approach is simplistic it has been shown to perform these types of classification tasks fairly well (Kibler and Aha, 1987). Once ground literal matches within a query view and a case view have been found the degree of the match for each pair will be computed, based on its importance. Importance of a ground literal depends on its context within a view; as a result ground literals will be given multiple assignments of importance depending on the query view. For example the ground literal 'Nearest_Weapon = Pistol' would have low importance if an observed entity currently had a better weapon. But it would have high importance if the observed entity had no weapon. Implementation of this process will be achieved through a simple algorithm, such as the one used in the REMIND's system (Kolodner, 1993), which is illustrated in the equation below. This algorithm enables both the importance and degree of the match to be represented as values between 0 (poor matches) and 1 (close matches). *Wi* in the equation is the importance of ground literal *i*, *sim* is the similarity function, which determines the similarity between the query case f_i^Q and the retrieved case f_i^R.

$$\frac{\sum_{i=1}^{n} w_i \times sim(f_i^Q, f_i^R)}{\sum_{i=1}^{n} w_i}$$

Short-term prediction of actions will be based on Dinerstein *et al.* (2005) prediction method. In their approach they predict the next action for each agent in the environment; this is then used to determine the next state for that agent, which in turn is used to predict the next action. This iterative process continues for *n* time steps in the future. Depending on the confidence in the case-library and the closeness of states to the query state, their associated actions can be generalised or "blended" into one combined action. The actions are generalised using k-nearest neighbour and bound minimax searches. This allows the non-determinism of human decision-making to be represented in the behaviour model. Views (secondary state information) will be incorporated into short-term prediction in order to give more accurate matches to the current state and to provide a guide for predicted states. The generalisation and prediction techniques described above do not allow views to be fully incorporated as they rely on determining predicted states through generalisation of actions. Views contain additional state information, which may or may not be affected / related to an observed entity's actions or generalised actions. As a result the next secondary state in a prediction can not be determined from an observed entity's actions. However it is possible to use views as a guide, as predictions that are close to the current time step would have views that are similar to the current view. These views would be given a higher priority over views that are different. The closeness priority would then be reduced, as the predictions get further away from the current time step.

The prediction techniques described above allow the actions of an observed entity to be predicted, however they do not provide any detail about the observed entity's current behaviour. For example, is an observed entity trying to reach the nearest weapon or are they are trying to escape through a door. Behaviour prediction of this type would provide an observing agent with this information and allow it to reason more accurately about an observed entity's future states and actions. In order to achieve this we propose to group view-action pairs together to form behaviour cases. This approach is akin to classical plan recognition techniques, such as the ones used by Kerkez and Cox (2001) and Fagan and Cunningham (2003). Each behaviour case will represent a possible behaviour an observed agent could be undertaking and will be stored in a separate case library to state-action pairs. The completion of behaviours will be identified by significant view state information. For example, the ground literal "*Picked_up_weapon*" would signify the observed entity had completed the behaviour "*getting_nearby_weapon*". Behaviours and their significant goal states will be defined, by subject matter experts (SMEs) (i.e. game players). The SMEs will specify possible behaviours an NPC could undertake and significant goal states for that behaviour, which will make up the significant view. This approach to defining behaviours could be considered contrary to our objective of developing a learning from observation system, however we feel that behaviours and goals must be clearly defined in order for behaviour prediction to perform well. In addition this approach to defining goals is consistent with existing work in the area, for example Lent and Laird (2001) use observations and annotations made by SMEs.

Each set of significant goal states for a behaviour will be unique, in order to allow the system to differentiate between the completion of behaviours. Table 2 illustrates possible behaviours and significant goal states that could be identified by SMEs. Once behaviours and goal states have been defined the recogniser can learn plans for these behaviours online by observing an entity's states and actions and storing them in the behaviour case plan library.

Behaviour	Significant Goal States (Unique)
Goto_Near_Weapon	Collected_New_Weapon
Charge_Opponent	Close_To_Opponent / Shooting
Runing_for_cover	Far-off / Moving_Away / Cover_Close

Table 2: Possible behaviours of an observed agent and their significant goal states.

The recogniser identifies that a behaviour has been achieved when all its significant goal states have been met. The stream of view-action pairs recorded for the last *n* time steps will be linked together to form a behaviour case. The new case

represents an observed plan for that behaviour. The recogniser can then identify future occurrences of that plan and thus the behaviour the observed agent is trying to achieve. The appropriate number of states and actions needed in a behaviour case to enable plan recognition will be determined during our research. To prevent saturation of the behaviour case library only a limited number of cases for each behaviour will be stored. Older cases will be deleted when new cases are observed. Case deletion is discussed in more detail in the next section.

Prediction of behaviours will be accomplished by matching a history of observed view-action pairs to view-action pairs in the behaviour case library. This is however a nontrivial task, as it is not possible in FPS combat situations to clearly identify when an observed entity starts a behaviour or if it switches behaviour, without direct player feedback. Furthermore observed view-action pairs may match many intermediate view-action pairs at multiple points in multiple behaviour cases. In order to predict under these conditions we propose an efficient matching method, which continuously predicts the observed entity's behaviour and a probability, which relates to the quality of the prediction. In order to achieve this, views that match the current view will be retrieved from the behaviour case library. Fast and efficient retrieval will be obtained by indexing views through their ground literals. Once close matches have been determined they will be ranked according to the closeness evaluation function described above. The behaviour cases which contain the closest matching views will then be identified, as a possible plan (behaviour) the observed agent could be undertaking. This approach is similar to the conflict pool method used by Fagan and Cunningham (2003). In our work we will refer to these cases as the behaviour pool. At the next time step this process will be repeated and behaviour cases with matching sequences of view-action pairs will be retained in the behaviour pool, while cases that do not have a matching sequence will be removed. Any new view-action matches found in behaviour cases will be added to the behaviour pool.

The behaviour with the longest sequence and greatest number of matching states will be selected as the predicted behaviour. In addition a probability will be created which represents the quality of a prediction. For example, predicting a behaviour based on two non sequential view-action pairs in two different behaviour cases would have a very low probability of an accurate prediction. Whereas a prediction based on five sequential view-action pairs from one case would have a high probability of an accurate prediction, because it closely matches a previously observed behaviour case. The advantages of this approach are that the system can continually predict the observed entity's current behaviour and change the predicted behaviour quickly. However, the technique relies on predicting based on a history of observed view-action pairs; this may result in poor prediction if the observed entity is performing novel actions.

Case Maintenance

In order to guarantee the performance of our action prediction and behaviour case libraries, a case maintenance policy is required. Too many cases in the library can significantly slow down the system and lead to poor prediction and generalisation. In addition deleting cases (or "forgetting") is very important when learning something as non-stationary as human behaviour (Dinerstein *et al*. 2005). In Dinersteins' work the number of cases in a region of the state space is fixed and cases are selected for replacement based on their age and unimportance (i.e. their similarity to a new case being added). Kerkez and Cox (2001) and Fagan and Cunningham (2003) do not propose a limit or delete plan libraries, however they do implement an abstraction scheme to maintain fast case indexing. Both these approaches are satisfactory, however limiting the number of cases in a region of the state space can prevent the system from learning behaviours and as stated above having too many case will reduce prediction quality and speed of the system.

In order to keep the action prediction case library at optimal performance we propose to add cases if the difference between a new case and the closest case in the library surpasses a threshold. If the query case does not surpass the threshold then the library case will be evaluated using a metric. If the returned metric value is below a threshold the library case will removed and the query case added, otherwise the query case will be discarded. However, as discussed in a previous section, if the query case is discarded the associated view and action will be attached to the library case if its current associated views are different from the query view. We formally define the case replacement policy as follows:

$$\text{if } M(\mathrm{s},\mathrm{a}) < \mathrm{r} \text{ then replace all } (\mathrm{s},\mathrm{a}) \text{ with } (\mathrm{s}_t,\mathrm{a}_t).$$

Where (s,a) is a case in the case library, $(\mathrm{s}_t,\mathrm{a}_t)$ is the query case and r is a predefined threshold value. The metric *M* is defined as follows:

$$M(\mathrm{s},\mathrm{a}) = -\alpha \times \mathrm{age} + \beta \times ||(\mathrm{s},\mathrm{a}),(\mathrm{s}_t,\mathrm{a}_t)|| + \delta \times \mathrm{useful}$$

This replacement metric is based on Dinerstein *et al*. (2005) approach; however we have also incorporated "usefulness", which defines how often the case has been used in prediction. A case which is used more often will more likely be retained in the case library. For example the simple usefulness metric: $\mathrm{useful} = \mathrm{age} \div \mathrm{timesused}$ could be implemented. The time since the case was last used in prediction could also be taken into account. In addition it's worth noting that since the primary state space will be partitioned for fast lookup, it will need to be repartitioned online as states get added and deleted. This can be achieved by automatic partitioning techniques such as kd-tree (Dinerstein *et al*. 2005). However this approach may add additional processing that could slow down the system. As a result further research will be required during implementation.

The behaviour case library will be maintained by limiting the number of cases that can be stored for each behaviour type. Behaviour case replacement will be achieved by first matching all the cases in the library that result in the same behaviour as the query case. The oldest case for that behaviour, which proved least useful, will be deleted and the

query case will be added. We formally define the behaviour case replacement policy as follows:

$$\text{replace } \arg \min_{(\text{bc})_i \in b} (M(bc)) \text{ with the query case } \text{bc}_t .$$

Where b are the matching behaviour cases, $(bc)_i$ is an individual behaviour case and $(bc)_t$ is the query case. The metric *M* is defined as follows:

$$M(\text{bc})_i = -\alpha \times \text{age} + \beta \times \text{useful}.$$

Behaviour cases are replaced based on their age and usefulness. To maintain computational efficiency behaviour cases will only be compared by their behaviour type. This may result in poor case replacement; however the reduction in the cost of searching is a worthwhile trade-off, since performance is key to our system and human behaviour is non-stationary.

Figure 4 illustrates a high-level overview of the proposed prediction architecture; it is inspired by Kerkez (2003) system design. The inputs for the system come from observed environment events and the outputs are action and behaviour predictions. A world interface (top-right in Figure 4) converts environment inputs into state-action pairs (including views). Once inputs have been properly formatted the system retrieves close matches to the inputs from the case libraries. The action prediction library returns cases, which are close to the query state-action pair. The behaviour case library returns cases which contains similar views to the query view. Once close matches have been found they are then passed to the behaviour pool and actions prediction modules. These are independent processes and will run in parallel (i.e. different threads) or sequentially one after the other. The action prediction module will generalise actions and determine a new predicted state based on the generalised action. The new state will be passed back to the retrieval module in order to determine matches to the predicted state. This process is repeated for *n* time steps into the future. Once complete the predicted state is passed to an agent. The behaviour pool module determines if sequential view-action pair matches have occurred. Once this has been accomplished the behaviour prediction module determines which behaviour the observed entity is most likely to be performing by reviewing which behaviour has the longest sequence and greatest number of matching states in the behaviour pool. Once the predictions have been made the storage and forgetting module updates the case libraries. The module first determines if the newly observed state-action pair should be added or should replace an existing case in the action predication library. It then determines if a behaviour case should be created. If a behaviour case needs to be created it will be added to the case library and the oldest and least useful case for the newly created cases behaviour type will be removed.

DISCUSSION & CONCLUSION

Being able to anticipate and infer an observed entity's actions is an important feature in the development of more sophisticated and responsive agents for interactive simulations. In this paper we have discussed the concept of learning from observation, online adaptation and have proposed an action prediction architecture. The proposed system builds upon existing incremental case-based research in the area and makes a number of novel contributions to improve the capabilities of the system. Specifically the architecture offers a more comprehensive state representation, behaviour prediction and a more robust case maintenance approach. These modifications should improve this type of system by enabling it to give more accurate and comprehensive predictions.

Action and behaviour prediction as proposed would be very useful in FPS games. It could be used in a variety of situations, such as combat, chase and partner prediction. For example in combat situations an NPC could infer an enemy's future position and their current behaviour, such as "get nearby weapon". This would enable an NPC to improve its action selection, by allowing it to adapt its path planning or targeting systems. However the reactive nature of action games does present a number of challenges, as human opponents are going to continually react and adapt their behaviour in response to an NPC's actions. For example a human player could observe an NPC firing at a nearby weapon; this may result in them adapting their behaviour by not picking up the weapon. This could be considered a useless prediction, since the NPC did not hit their enemy, however denying an enemy powerups in this way has advantages in that the opponent will continue to fight with a less powerful weapon. Having said this adaptation and prediction is only useful when an NPC can take advantage of it. If firing at nearby weapons takes time and leaves an NPC open for attack it is not a worthwhile adaptation. However if

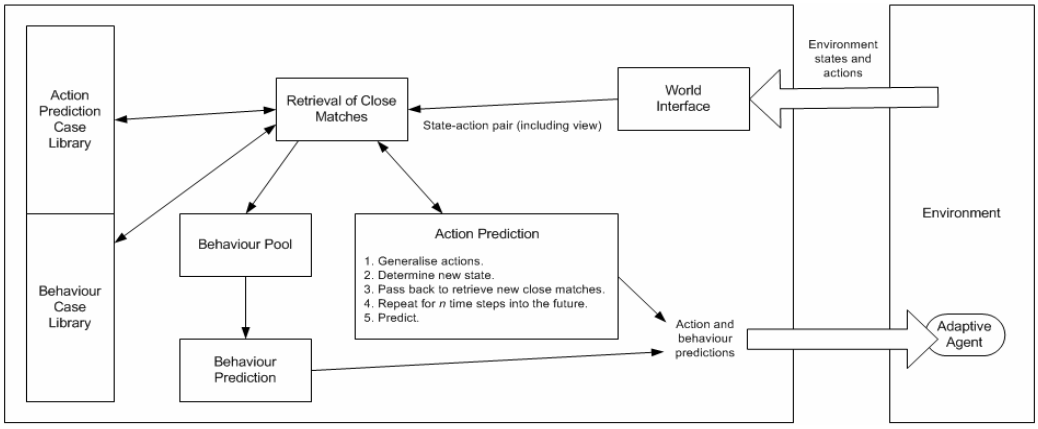

Figure 4: High-level overview of the proposed system architecture.

the adaptation takes at short amount of time and the NPC is protected by cover there would be more chance of success. In other situations, such as when chasing an opponent, an NPC would have more time to adapt if an enemy is facing away, as changes in targeting and path planning would not be observed.

We have yet to implement and empirically evaluate our proposed architecture. As a result we are unable to determine the appropriateness of our modifications / improvements to existing incremental case-based approaches. In particular it is difficult to determine if our proposed system will meet the speed requirements for online learning and prediction in interactive simulations, such as computer games. However our work is grounded in existing online prediction and adaptation research, as a result the architecture should meet speed requirements.

An Initial implementation will take the form of a 2D custom environment, which represents a combat situation in an FPS game. To begin with a simple tracking agent will be implemented, with which we will incorporate our prediction system. This is similar to Dinerstein *et al*. (2005) initial implementation. Human players and simple navigation / planning agents (e.g. an A* agent) will be used to control observed entities. If our initial implementation is successful we will link our system to an existing FPS computer game such as Quake II or Unreal Tournament. In addition further work will include developing task and goal adaptation layers for our architecture.

REFERENCES

Alexander Thor. (2002) GoCap: Game Observation Capture *in* Steve Rabin (ed) *AI Game Programming Wisdom*, Charles River Media: WHERE, pp. 579 - 585.

Cass, S. (2002) Mind Games, IEEE Spectrum pp. 40-44, December 2002.

Dinerstein, J. and Egbert, P. (2005) Fast multi-level adaptation for interactive autonomous characters. ACM Trans. Graph. 24, 2 (Apr. 2005), 262-288.

Dinerstein, J., Ventura, D. and Egbert, P. (2005) Fast and Robust Incremental Action Prediction for Interactive Agents, Computational Intelligence, Vol. 21, No 1, pp. 90-110, 2005.

Fagan, M. and Cunningham, P. (2003) Case-based recognition in computer games". Technical Report TCD-CS-2003-01 Trinity College Dublin Computer Science Department.

Fernlund, H. (2002) Learning by Observation – New Improvements. <http://www.ida.his.se/ida/conf/PromoteIT 2002/4C1Fernlund.pdf >.

Kaminka, G., Veloso, M., Schaffer, S., Sollitto C., Adobbati, R., Marshall, A., Scholer, A., Tejada, S. (2002) GameBots: a flexible test bed for multiagent team research, Communications of the ACM, v.45 n.1, January 2002.

Kerkez, B. (2003) *Incremental Case-Based Plan Recognition with Incomplete Plan Libraries*. PhD Thesis, Wright State University.

Kerkez, B. and Cox, M. (2001) Incremental Case-Based Plan Recognition Using State Indices, Cased-based Reasoning Research and Development: Proceedings of 4th International Conference on Case-Based Reasoning (ICCBR 2001), Aha, D.W., Watson, I., Yang, Q. eds., pp. 291-305, Springer-Verlag, 2001.

Kerkez, B. and Cox, M. (2003) Incremental case-based plan recognition with local predictions. Int. J. Artif. Intell. Tools: Architectures, languages, algorithms 12, 4, 413-463.

Khoo, A., Dunham G., Trienens, N. and Sood, S. (2002) Efficient, Realistic NPC Control Systems using Behavior-Based Techniques. AAAI tech. report SS-02-01, Menlo Park, CA: AAAI Press.

Khoo, A. (2003) *Implementing Efficient Joint Beliefs on Multi-Robot Teams*. PhD Thesis, Evanston, Illinois.

Kibler, D. and Aha, D. (1987). Learning representative examplar of concepts: Initial case study. In *Proceedings of 4th International Workshop on Machine Learning*, (pp.24-30). San Mateo, CA: Morgan Kaufmann.

Kolodner, J. (1993) *Case-based Reasoning*. Morgan Kaufmann Publishers Inc, 340 Pine St, 6th floor, San Fransisco, CA 94104, USA, 1st edition, 1993.

Laird, J. (2001) It Knows What You're Going To Do: Adding Anticipation to a Quakebot. *Proceedings of the Fifth International Conference on Autonomous Agents,* pp. 385-392. Available from: <http://ai.eecs.umich.edu/people /laird/ papers/Agents01.pdf>.

Laird, J. and van Lent, M. (2001) Human Level AI's Killer Application. Interactive Computer Games. Artificial Intelligence Magazine, 22(2), pp. 15-25.

Lent, M. and Laird J. (1999) Developing an Artificial Intelligence Engine. In Proceedings of the Game Developers Conference 1999, San Jose, USA, 1999.

Lent, M. and Laird, J (2001) Learning procedural knowledge through observation. In: International conference on Knowledge capture, Victoria, British Columbia, Canada, ACM Press (2001) 179–186.

Mao, W. and Gratch, J. (2004) Decision-Theoretic Approach to Plan Recognition, ICT Technical Report ICT-TR-01-2004.

McGlinchey, S. (2003). Learning Of AI Players From Game Observation Data. *GAME-ON 2003 4th International Conference on Intelligent Games and Simulation* (eds. Quasim Medhi, Norman Gough and Stephane Natkin), pp. 106-110.

Schaeffer, J. (2001) A gamut of games. *AI Magazine*, vol. 22 nr. 3, pp. 29-46.

Schaeffer J., Billings D., Peña L., Szafron D. (1999). *Learning to Play Strong Poker*. *Department of Computing Science University of Alberta* [Online]. 02 Aug 1999 [cited 28 Mar 2003]. Available from: <http://www.cs. ualberta.ca/~darse/Papers/ML99.html>.

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.

Suliman, H., Mehdi, Q.H. and Gough, N.E. (2002) Virtual agent using a combined cognitive map and knowledge base system. *ISCA 9th International Conference*, Boston 2002, PP 65 – 70. ISBN 1-880843-43-9.

Thurau C., Bauckhage C. and Sagerer G (2004) Imitation Learning at all Levels of Game-AI, *Proc. 5th Int. Game-On Conf. CGAIDE'2004*, Microsoft Reading UK, pp 402-408, ISBN 0-9549016-0-6.