Data-Driven Decision Theory for Player Analysis in Pacman

Ben Cowley, Darryl Charles, Michaela Black, Ray Hickey

School of Information and Computer Engineering, University of Ulster, Coleraine
University of Ulster, Cromore Rd, Coleraine, Northern Ireland, BT52 1SA
{cowley-b, dk.charles, mm.black, rj.hickey}@ulster.ac.uk

Abstract

Computer and videogames have been described using several formal systems – in this paper we consider them as Information Systems. In particular, we use a Decision Theoretic approach to model and analyse off-line, data from Pacman™ players. Our method attempts to calculate the optimal choices available to a player based on key utilities for a given game state. Our hypothesis in this approach is that observing a player's deviation from the optimal choices predicted can reveal their play preferences and skill, and thus form a basic player classifier. The method described builds on work done in [Cowley *et al* 2006], increasing the scope and sophistication of the model by decreasing reliance on supervision. The downside is a consequent performance hit, which prevents real-time execution of the modelling algorithm. In this paper we outline the basic principle of the Decision Theoretic approach and discuss the results of our evolution toward data-driven classification.

Introduction

The approach described herein is grounded in a body of research on games as formal systems (systems of information), and we can summarise this as follows. There are several definitions of games and gaming in the literature, from a variety of sources such as ludologists [Wolf & Perron 2003], game designers [Crawford 2002] and game studies researchers [Salen & Zimmerman 2003]. However, the academic study of commercial computer and video games (hereafter called games) is still a very new area, especially as regards understanding and systematising the relationship between games and players. In particular more study is required into game *dynamics* – i.e. the interaction that arises between a player and game's formal system. A formal system (or game *mechanics*) defines a gameplay logic which can be expressed as a mathematical model. Game play in such a model occurs as a (usually) deterministic, emergent process of action sequences belonging to one or more players. As this process equates to the player's exploration of the game's possibility space it also correlates strongly to their overall experience. In other words, often how mechanics give rise to dynamics is the biggest influence on a player's experience [Salen & Zimmerman 2006], and it is improvement of experience that is the main goal of player modelling. Therefore our goal in this paper is to advance the investigation into player modelling using reductionist formal methods.

Predictive Player Modelling

Predictive player modelling works by considering the player's in-game goals as equivalent to some target function of the game state and calculating this function using in-game player-related data (similar to [Thue & Bulitko 2006]). Our overall aim is develop this method into a player classifier, with validation based on the Demographic Game Design (DGD) [Bateman & Boon 2005] typology of players. If successful, Decision Theoretic modelling would provide a low-level building block to be used so that a game's play structure may automatically adapt to the preferences and skills of each individual player.

The authors developed this approach inspired by considering games as Information Theory systems, especially the principle that Decision Theory [Gmytrasiewicz & Lisetti 2000] can be used to model the choices that players make on the basis of available information. This decision theoretic approach is *descriptive* rather than *prescriptive* because although one can quite accurately calculate the optimal choices that a player should take given the current game state and the most obvious and observable utilities, players generally do not play in an optimal way. This may be due to their lack of experience or their personal playing style.

Previously, the authors implemented predictive player modelling in [Cowley *et al* 2006], using a pre-defined set of states for classification of the player's utility, which had the disadvantage of having a low granularity. This resulted in poor accuracy when classifying the player's state, which increased the uncertainty in the prediction function. The current implementation follows a different approach – to take account of all relevant features of the current state, so that utility calculation is built on a weighted feature vector. This reduces the impact of arbitrary design choices, so classification of state is more unsupervised and data driven. The main aim of this paper then is to investigate whether such a switch from pre-defined states to data-driven utility calculation improves the predictive power of the Decision Theoretic method when the weighted features are themselves predefined.

The paper continues in the next section with a description of our Pacman test-bed implementation, while section 3 explains the Decision Theoretic formula used, and how it can be applied to Pacman in a data driven fashion. In Section 4 we present results from a series of experiments run on game data collected from gamers with a broad cross-section of experience, and this leads the reader to the conclusions and final remarks on the applicability of the method and possible future work.

Pacman Test-bed Implementation

The Pacman implementation used was created by the authors and represents an interpretation of the original Namco game, rather than a clone. Thus we provide a description of the game mechanics below. Pacman is a linear and relatively simple game when compared to modern computer games, yet its possibility space is still far too complex to search without heuristics. Below we describe how the game was broken down into its constitutive elements and fitted into the utility calculation formula (Initial Caps are used to describe game entities and their actions; 'the player' and 'Pacman' are inter-changeable). This enabled our implementation of the formula within the test-bed code.

- The game world is a 2D graph, where the nodes of the graph can be Pill, Dot or empty. This constitutes a level.
- Pacman and the Ghosts (the in-game actors) Move between nodes along two axes, horizontal and vertical.
- Ghosts move randomly, starting in a central box area. Random movement was chosen as the Ghosts' controller to make it simpler to calculate the probability of future states in the look-ahead tree (since random movement gives an even probability distribution to all future states)
- Pacman Eats Dots & Pills when he Moves over them.
- Pacman only Eats Ghosts when he has Eaten a Pill within the last *t* game cycles, otherwise Ghosts Eat Pacman and he loses a Life.
- Eaten sprites re-spawn at their original start node, unless Pacman has run out of lives, then the game is over.
- Pacman must Eat all Pills and Dots to finish, whereupon the level ends whether or not the player was still within the *t* cycles started from recently Eating a Pill.
- Ghosts are permeable and do not interact with Dots or Pills or obstruct each other.
- Points are scored as follows: 10 per Dot, 50 per Pill, (100 $\times$(2 ^ Num of Ghosts)) for each Ghost Eaten after a Pill.

The above is a description of Constituative rules [Salen & Zimmerman 2003] – the formal rules that comprise the game mechanics [Hunicke *et al* 2004]. Our approach uses descriptive Decision Theory to model the Constituative rules, and by this method approximately model the game dynamics – i.e. how the player interacts with the game (as described by Operative rules [Salen & Zimmerman 2003]).

Pacman and Decision Theory

To see why we use Decision Theory, consider the situation where Pacman is unconstrained in a level – for example *if there were no Ghosts*. Time is never a constraint in Pacman, so without Ghosts the path chosen by the player would be unimportant, as any explorative path would result in the same utility – collecting all the dots. There would be no unforeseeable or surprising events, no challenge. Pacman could follow any path through the level, and the level would end when the path does. In this game, there are no meaningful choices to be made, which by some definitions [Salen & Zimmerman 2003] would mean this was no longer even a game. Gameplay is created by choosing between utilities – e.g. save lives by evading (or increase points by hunting) Ghosts.

In fact, in most games the mechanics of play are concerned with choosing the action which maximises a utility function, from a set of actions situated in an evolving possibility-space. This explanation of gameplay is very evident in the literature, both before and after the advent of computer games:

> A game is a form of art in which…players…make decisions in order to manage resources through game tokens in pursuit of a goal. [Costikyan 1994]

Indeed most formal work on games playing, whether it be Game Theory [von Neumann & Morgenstern 1947] or board-game playing computer programs [Bell 1972], relies on predicting the decisions of opponents based on estimation of their utility. The problem in translating this for modern computer games is that firstly, the possibility space is often very large and states and features cannot be precisely defined, and secondly *most are not turn-based*. This last invalidates many optimisations & heuristics from the research on board game playing programs [Bell 1972].

Our method follows [Gmytrasiewicz & Lisetti 2000] in formulating gameplay in terms of Decision Theory. Information Theory, and by extension Decision Theory, can be viewed as a formulation of the uncertainty of making non-trivial choices, and so can be used to describe the choices that create gameplay. Thus in our implementation the current information, from the game state and the heuristics on player utility, is used to make predictions of the choices of which nodes it is possible for Pacman to move to. We can fit the choices, and their utility weights, to a formula using Decision Theory. Thus we can predict quite accurately the behaviour of a rational, utility-maximising player. The list of these predictions (i.e. the actions of a rational player) will give an example of a near-optimal path.

A rational player formulates its decision making situation in terms of a finite set, A, of the alternative courses of action, or plans, it can execute. A member of A, say a_i, can be thought of as a plan consisting of consecutive actions extending to the future time t_{ai} (see section **Tree**

Search). An action plan occurs in a state-delimited world, formalised as the set of all possible states of the world, *S*. In Pacman, the current state is known, and thus our probability function represents the uncertainty of the player as we project forward in time – resulting in a probability distribution **P**(*S*) over the state space *S*. This projection function is expressed *proj* : *S* × A -> **P**(*S*), so that projecting the results of action a_i given the current state *S* gives the probability of the projected states, $proj(S, a_i) = P_i(S)$. Our Utility function *U* encodes the desirability of the projected states to the player. *U* maps a state to numerical output, $U : \mathbf{S} \rightarrow \mathbb{R}$. Thus by calculating (Formula 1) we predict a plan *a** that a player should perform to maximise their utility:

$$a^* = ArgMax_{a_i \in A} \sum_{s \in S} proj_i^j U(s^j) \qquad \mathbf{1}$$

Where $proj_i^j$ is the probability assigned by $P_i(S)$ to the state s^j. We can derive from this formula the algorithm for dynamic predictions of optimal utility actions in a game. Our definition here is closely related to that of stochastic processes, and in particular to Markov decision processes [Boutilier, Dean, & Hanks 1999], but it makes explicit the decision problem the agent is facing by enumerating the alternative action sequences the agent is choosing among.

Data Driven Decision Theory

Tree Search. The goal of implementing (Formula 1) is to search the utility-weighted future-moves tree. This tree is built by finding all possible combinations of positions which the in-game actors, Pacman and four Ghosts, can occupy when they move one step; and then iterating that calculation for a computationally tractable number of steps. The tree will branch by a factor corresponding to the number of nodes adjacent to each actor. Our test-bed game map has 187 navigable nodes – of these 150 have 2 adjacent nodes, 30 have 3 adjacent nodes, and 7 have 4 adjacent nodes. Thus the average branching factor will be roughly $^{20}/_{25}.2^5 + {}^{4}/_{25}.3^5 + {}^{1}/_{25}.4^5 \approx 2.54^5$.

Look-ahead is done by building the tree of possible future states up to a ply depth congruent to the number of moves Pacman might need in order to complete a meaningful action. In other words, players make plans with strategic goals, but each plan also equates to a sequence of player's low-level choices. Decision Theory considers these sets of choices - thus a plan produced by our prediction function should be bounded by what is a reasonable number of moves ahead that the player could consider. Since players do not know the Ghost movement control functions, which here are in any case random, it will be difficult for them to predict Ghost movement very far in the future (at least until they become quite expert). This means that we have a boundary constraint (beyond computational limitations) on how many moves of the player's plan we should try to predict. We have posited that the ideal tree depth is 6, as that is the width of one grid square when our test-bed game map is partitioned into a 3 ×3 grid. This grid division is significant because, in 8 of 9 grid squares, there is an attractor for the activity of the in-game actors. There are the 4 Pills, the 2 re-spawn points, and the 2 teleport gates (all standard elements of Pacman). When the distribution of each actor's movement is plotted, peaks can be seen around these attractors. Unfortunately, a depth of 6 produces an intractably large computation under the current implementation and was scaled back to a depth of 3 pending algorithmic optimisation (see **Future Work** Section).

In building the tree the algorithm explores the game's possibility space, ranking each possible state by calculating the utility to the player of a set of features (as defined below), and can tell us the how far off the 'spine' of the space is the player's move. Picture this as the exploration of a chreode [Kier *et al* 2002] of optimal play, defining optimality as the maximal of a utility function.

Feature Specification. Ultimately, utility in (Formula 1) is for a process – e.g. the process of collecting all the Dots in an area, or getting a Pill then Eating Ghosts. Process utility must be calculated by summing state utility, and the utility of a state is judged by global features.

Ideally, features of a Pacman state would be defined by (ideally, unsupervised) Machine Learning (ML) of the patterns in gameplay logic through observation of play. The problem with this approach, as with any attempt at learning through observation of play, is that it is difficult to obtain a sufficiently large corpus of state data from observed games. So in substitution for this method we look to expert opinion to specify our features, which luckily in Pacman are not too numerous.

In simple terms, features would be things like: the A* or Manhattan distance between Pacman and each Ghost; number of Pills, Lives or Points. Each of these features will have a relatively simple numerical representation describing its state in the context of the game engine.

The conceptual form of most of the state features is that of a vector of opportunity for Pacman, determined by the capacity for meaningful action and weighted by the consequent risk/reward ratio. Definitions of heuristics for the calculation of utility of state features begin with the primary utility – Score. Since the scoring function of Dot collection is a one-to-one mapping between potential and actual Points, the major deciding feature of a state associated with scoring is the potential for Hunting Ghosts. The antecedent for Ghost Hunting is for Pacman to Eat a Pill, so unless Pacman has just done this, the weight associated with Ghost Hunting will be inversely proportional to the A* distance from Pacman to the nearest Pill.

This serves to balance off this weighted feature against that of Ghost avoidance and Dot collection, so that aggressive tactics are only predicted if a Pill is sufficiently close that the distance to it doesn't offset the predicted payoff of aggression. Some heuristic features used were:

- Ghost proximity (here measured by A*) and distribution – these correlate to Pacman's risk in normal play, and to

reward when he has just Eaten a Pill and is invulnerable.

- Individual dots are weighted by adding the distribution density of adjacent dots, and by the inverse of their distance to Pacman.
- Risk or the number of lives left to Pacman.

Utility Calculation. The ideal metric for the calculation of feature utility weights in any game would be the difference between the value of some metric for the current state and the absolute value for this metric in the final state. The final state is the winning state (maximum utility in a non-binary outcome game). Knowing this final state, one must simply calculate which next state decreases the difference by the greatest amount – i.e. maximises player utility. In practice, obtaining this final state metric requires the calculation of the entire game possibility space, which is almost always an intractable calculation. Even TicTacToe, (i.e. noughts and crosses), has a possibility space on the order of 10,000 moves and counter-moves.

Therefore, it is necessary to specify metrics that only look ahead a few steps, and instead of taking as their target the final state utility these metrics use heuristic measures of utility. Such measures reflect the two major utility indices, Points and Lives, and variables which affect them. Calculating potentiality of scoring is important, since it can cut down on the amount of lookahead necessary. For example if, when moving in one direction Pacman comes closer to some Ghosts, his utility will be higher if he also comes in proximity to a Pill. This is because the *potentia*l for scoring was raised, whatever the *actual* future state of the game.

As each possible state at a time-step is ranked by its utility contribution to the path to which its parent belongs, we navigate the tree of states along the path of highest utility in order to 'back-up' the prediction of which next Pacman move is optimal. In concept, this is a similar methodology to [Samuel 1950]'s checkers player, which specifies a polynomial of weighted feature vectors.

Our utility function was implemented as a static (non-learning) evaluation of Pacman's scoring potential. Thus each feature would be weighted by a combination of its contribution to the Score, and its potentiality of occurrence. If Pacman were in proximity to a Pill, and all four Ghosts, this would have the utility of the potential score gained by eating all four Ghosts, reduced by some constant factor of the distance to, first the Pill, and then the Ghosts. However if Pacman had already eaten a Pill, the algorithm would not reduce the utility of Eating all four Ghosts by the distance to the nearest Pill. Of course, there is a risk factor involved with this hunting behaviour, since the timer on the effect of the Pill wears off after a fixed period. It is unlikely that a player would be exactly aware of this timer's duration, so in chasing all the Ghosts they risk switching back to vulnerability right beside a Ghost.

With implementations of features and utility as described above, our current implementation of (Formula 1) can be described with the following pseudo-code:

```
->Read new game state
  ->Build possible future states tree
  ->For each direction open to Pacman
    ->Calculate utility of child states
    ->Update Pacman position = direction
    corresponding to highest utility
  ^-Iterate up to max tree depth
^-Iterate for all game states
```

Data Driven Analysis Experiment

The aim of this paper was to investigate the potential of our data driven Decision Theoretic approach as the foundation for a player classifier, based on comparative analysis of predicted and actual behaviour. The manner in which this would work is as follows.

Feature weights can be specified for individuals to match patterns observed in their play/associated with their type. This allows a form of player classification corresponding to observed play styles [Bateman & Boon 2005]. As play progresses, well-chosen features should be reflecting the play decisions that the player needs to make. If their decisions are all potentially equally likely, then whatever they decide to do indicates what their *play preference* is. The *weights* of those features that reflect play decisions will therefore correspond to these same play preferences – in future work, adjusting weights to maintain high accuracy of predictions could allow the method to be used to a learn a player model in real-time.

To see how the weights of gameplay features could reflect player preferences, we can examine some of the common play activities in Pacman with reference to the DGD typology. Dot collection is the default activity, and unless the player is in a hurry to finish, should be of quite low weight compared to Pills and Ghosts. How the player regards Ghost-hunting has a substantial effect on their choices in most situations. If not too interested in achievement, they may play through the whole level as a dot clearing exercise, crossing Pills simply because they're in the way – corresponding to a Wanderer-style of play. Next is more moderate hunting behaviour, the Manager style, where the player waits for a few Ghosts before Eating a Pill, but no risks are taken by chasing all four. Then there's the Conqueror's approach, carefully hoarding each Pill until all four Ghosts are in proximity, looking for the maximum 1550 points that would be gained from Eating one Pill and four Ghosts. These styles are taken from player typological research [Bateman & Boon 2005].

Such activities are naturally arising out of the mechanics of game play, and as such represent an inherent dynamic of the game. This is why the player's pursuit of these activities was chosen to define the states in the authors' first implementation of Decision Theoretic Pacman – yet as was reported, such an approach failed to give accurate results. Instead of abandoning or ignoring the observed existence of such states, we would look to find classification of states arising from the more sophisticated data driven Decision Theoretic approach described herein,

as opposed to defining them ourselves. In other words, we envision a classifier, built on top of the new data driven approach, which can translate data from a state-tree look-ahead model into analogues of player typologies for use in reasoning about player behaviour.

Experimental Design. To create this type of classifier, we first need to test the validity of predicting the actions of a single type of player. Setting up the choice of features and balancing weights was done by hand, guided by expert opinion and the player type descriptions of the DGD [Bateman & Boon 2005]. This was thought to be sufficiently accurate as the Pacman feature set and the player type under investigation are both relatively simple.

In placing the analysis step off-line, using recorded game state data, we remove what seems to be a vital component of player modelling – the ability to react in real-time to the learning curve of the player. However, in order to validate the method, the authors consider it a worthwhile first step to take the most complex game state data, and analyse it using the least parsimonious methodology. This serves several ends.

Firstly, we can establish a worst-case scenario estimation of computational time and resource consumption in the current test-bed. Algorithm analysis can then explore in detail the most computationally expensive sub-routines for the process of optimisation. Secondly, basing the method on a higher dimensional data space will help prevent getting a balance of feature weights that is over-fitted to our training data set.

Given these considerations, testing the method involved using data sets from full-featured, complete one-level games of a single player typed as Conqueror under the DGD player typology. That is, one player aimed for high scores in one complete level with four Ghosts and four Pills. Look ahead was set at 3, so that state trees that branch on the order of 2^5 times at each node would have a magnitude (or count-of-leaves) of approximately 2^{15}. Data sets averaged around 300 states, the number of moves it took our normally skilled player to complete a level.

Player movement is enabled by mapping the keyboard arrow keys [↑,↓,→,←] to integers from 0 to 3 respectively. Thus the function which implements the Decision Theory formula above, and whose domain is the set of game states, has the range [0..3]. A one-to-one comparison between prediction function output and actual moves made in the training data produced the results in (Chart 1 and 2).

Results. The results show that prediction accuracy averages at 39%, and the duration of consecutive predictions (a sign of accurate classification of activity sequences) averages 2.33 with standard deviation of 2.35. The latter figures imply that our method is not predicting long sequences of Pacman's actions, but this is belied by the significant occurrence (10%) of long and very long sequences (over 6) of correct predictions. Overall, the picture that emerges is that *some* features exist which facilitate precise prediction modelling, even to the magnitude of 19 consecutive correct predictions. The feature most associated with high frequency of correct predictions is that where Pacman Hunts Ghosts. In a state where this feature has a high utility, Pacman will have just eaten a Pill and will be relatively close to one or more Ghosts – a situation which presents an obvious high-scoring opportunity to the player and generates a clearly defined high-utility path within the prediction function.

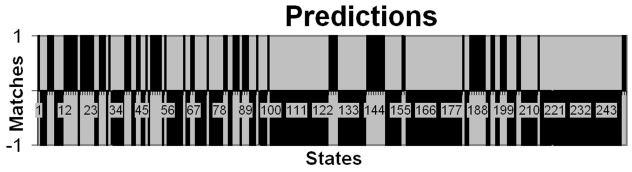

Chart 1: First 250 game states, correct predictions = 1, incorrect predictions = -1, black lines are predictions

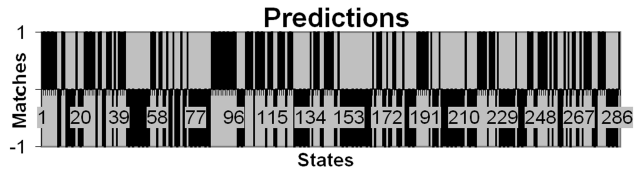

Chart 2: Another 290 game states, where accuracy ≈ 45%

On further analysis an interesting correlation becomes apparent. *Patterns* of moves, or the frequency with which directions are followed and changed, can be seen to match across a significantly higher proportion of the test results than had actual values. This is illustrated best by considering smaller portions of the data so that the correlations can be more clearly seen (Chart 3).

In this chart, we can see that the black line, representing Predictions, has a similar profile to the grey line (actual Moves), but is slightly offset in time. In other words, the decision theory function predicts the same behavioural patterns but is not getting the timing right. In graphs of the other results sets (which we have not space to show here), a similar pattern arises. This suggests the Prediction function wasn't working with a deep enough look-ahead tree. The correlation shown in (Chart 3) implies that the low accuracy of predictions can be explained as a consequence of insufficient refinement of weights under circumstances where two or more features compete closely in terms of utility. In these cases, final choice of direction will be almost arbitrary as the utility calculation it comes from is based on insufficiently refined weights or a too-shallow look-ahead tree and cannot classify the 'true' spine of the data space (the action sequence of highest utility).

Future Work. The authors' initial concern is to optimise the performance of the method. Since calculation of the look-ahead tree consumes the lion's share of execution time, we plan to apply a form of the alpha-beta heuristic to prune the tree in mid-build. Of course, the simplest optimisation is often the best – reduction of the data space dimensionality. In considering games of 3 or less Ghosts, we reduce tree branching by a power of 2 for every Ghost.

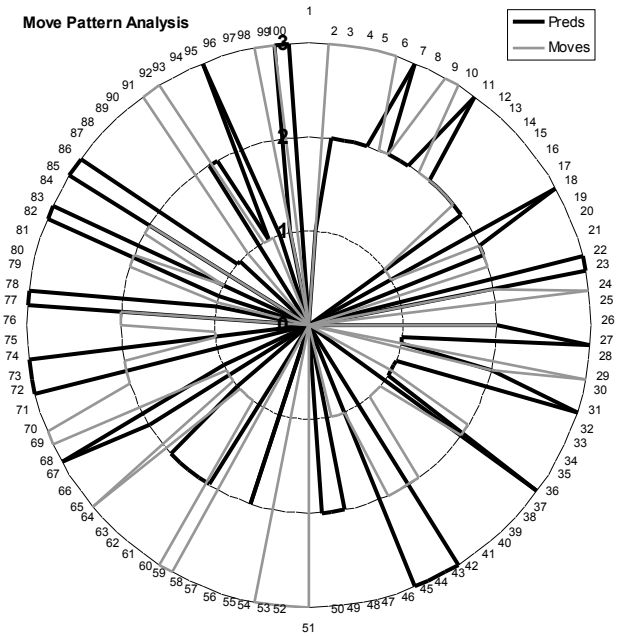

Chart 3: Radar graph - Actual Moves overlays Predictions

Conclusion

This paper details the authors' second implementation of a Decision Theory approach, and in conclusion, the data driven Decision Theoretic analysis method does not yet provide a statistically significant improvement of accuracy over previous work done in [Cowley *et al* 2006].

The main reason for this is insufficient balancing of the weights of state features. Since game data was provided by a single player, it was not thought that weights could be learned using such a small training set. Instead, weights were set manually leading to unbalanced and indeterminate features as explained above. Another specification flaw was the random movement of the Ghosts, which doesn't correspond to player's intuition: they would tend to ascribe agency to the Ghosts.

However, in this experiment we investigated the capacity for utility-centric Decision Theory to do predictive player modelling, and we saw results which suggest that this method could indeed work as a classifier *if a learning step were introduced*. Experimental setup suffered from a common flaw of games research – over-reliance on unproven assumptions of gameplay. Certain of these assumptions could be circumvented by learning weights for utility-measuring features of the game state.

In the final analysis, it was necessary to move beyond the dependence of the authors' earlier work on pre-defined states for classification. Such dependence limits potential refinement of classification accuracy, a limitation which is circumvented by considering data from a full feature-set of each state in a look-ahead tree. Naturally, consideration of the full data space does not *guarantee* accuracy of predictions, but accuracy is made more achievable by facilitating iterative refinement of feature weights. With some promising trends apparent in the current results, it is hoped that optimisation of the prediction algorithm and learned balancing of feature weights can produce a workable player classification method.

Acknowledgements

Pacman is a registered Trademark of Namco Corp.

References

Bateman, C. & Boon, R. 2005. *21st Century Game Design*, Charles River Media, London.

Bell, A.G., 1972. *Games Playing with Computers*. London : Allen and Unwin.

Boutilier, C., Dean, T., & Hanks, S., 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11, 1-94.

Costikyan, G. 1994. I Have No Words And I Must Design. *Interactive Fantasy Journal 2*. Republished in [Salen & Zimmerman 2006].

Cowley, B., Charles, D., Black, M., & Hickey, R. 2006. Using Decision Theory for Player Analysis in Pacman. In *Proceedings of the SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, Rome, Italy, October 2006. Yannakakis, G.N. & Hallam, J. Eds. Technical Report TR-2006-2, Maersk Institute for Production Technology, University of Southern Denmark, Odense, Denmark.

Crawford, C. 2002. *Chris Crawford on Game Design*, Prentice Hall PTR, London.

Gmytrasiewicz, P. & Lisetti, C. 2000. Modelling User Emotions during Interactive Entertainment Sessions, In *Papers from the AAAI 2000 Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Tech. Report SS-00-02. AAAI Press.

Hunicke, R., LeBlanc, M., & Zubek, R. 2004. MDA: A Formal Approach to Game Design and Game Research. In *Proceedings of the Challenges in Game AI Workshop*, 19th National Conference on A.I. (AAAI '04), California. AAAI Press.

Kier, L.B., Cheng, C.K., & Testa, B, 2002. A Cellular Automata Model of Ligand Passage over a Protein Hydro-dynamic Landscape, *Journal of Theoretical Biology*, 214, 415–426.

Salen, K., & Zimmermen, E., 2003. *Rules of Play: Game Design Fundamentals*, The MIT Press.

Salen, K., & Zimmermen, E., Eds., 2006. *The Game Design Reader: Rules of Play Anthology*, MIT Press, Cambridge, Mass.

Samuel, A. L. 1950. Some studies in machine learning using the game of checkers. *IBM J. Res. & Dev.* 3 (1950), 211-229.

Thue, D. & Bulitko, V. 2006. Modelling Goal-Directed Players in Digital Games. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE)*, Laird J. & Schaeffer, J. Eds. June 20-23, Marina del Rey, California. AAAI 2006.

Von Neumann, J. & Morgenstern, O. 1947. *Theory of Games and Economic Behaviour*, Princeton University Press.

Wolf, M.J.P., & Perron, B., Eds., 2003. *The Video Game Theory Reader*, Routledge, NY.