Analyzing Player Behavior in Pacman using Feature-driven Decision Theoretic Predictive Modeling

Ben Cowley, Darryl Charles, Michaela Black and Ray Hickey

Abstract—**We describe the results of a modeling methodology that combines the formal choice-system representation of decision theory with a human player-focused description of the behavioral features of game play in Pacman. This predictive player modeler addresses issues raised in previous work [1] and [2], to produce reliable accuracy. This paper focuses on using player-centric knowledge to reason about player behavior, utilizing a set of features which describe game-play to obtain quantitative data corresponding to qualitative behavioral concepts.**

I. Introduction

With this paper we present the results from a novel real-time player modeling algorithm that attempts to predict each move that a human player makes in Pacman. These predictions are compared with actual player activity. We suggest that the results of this comparison form a model, and after enough instances of play, analysis of this model should form a valuable picture of player behavior. Whether they follow the predicted path or deviate from it, some insight should still be possible.

Predictive player modeling works by considering the player's in-game goals as equivalent to some target function of the game state and calculating this function using observed player data [3], [4], [1]. Consideration of the player's goals or *utilities* is central to this; broadly speaking the predictive modeler will use real-time observation of play habits to determine player preference for competing potential game states. In [5] we discussed how games can be described using formal or semi-formal structured specification systems. By extension of this approach, we can use a decision theoretic formulation to model the short-term elements of the game that are important to a player – the player's utilities – and predict the player's next actions. That is, we can calculate a ranking for all the choices available to the player, based on the utilities associated with the game states produced by each choice.

In other work [4], preference learning has been unsupervised, building the classes or features of preference through on-line learning. By contrast, in our algorithm definitions of player utilities in Pacman will be based on the game-play features described in section II.

Ben Cowley is with the Centre for Knowledge and Innovation Research, Helsinki School of Economics, Fredrikinkatu 48a 9krs, 00100 Finland. (phone: +358 40 353 8339; e-mail: zenphd@gmail.com).

Darryl Charles, Michaela Black and Ray Hickey are with the School of Information and Computer Engineering, University of Ulster, Coleraine, BT52 1HQ, United Kingdom. e-mail: {dk.charles; mm.black; rj.hickey}@email.ulster.ac.uk

A. Previous Work

Our original inspiration in formulating game-play in terms of Decision Theory came from [6]. Their approach combines a rational, utility-maximizing agent design, with a taxonomy of emotional states drawn from the literature [7]. Emotions are implemented as transformations of the decision-making situation described for the rational agent, thus empowering the agent to manipulate its own decision-making process, and effect non-rational behavior. In order to facilitate our approach, we have replaced the emotional part of the model with utility calculation functions – in this paper they are derived from the features of play behavior described in the next section.

This provides the major novelty of the approach – although predictive player modeling has been done before, for example in [3], as far as we know there are no other attempts to incorporate knowledge gleaned from data mining of players' recorded games into the utility or preference calculation functions of the modeler.

Our previous work had two main problems: the non-empirical calculation of utilities in both papers, and the tree-based approach of [2] for searching the game's possibility space. This evaluated temporal-sequences in *post-hoc* fashion, estimating qualities (like utility) of single states, and accumulating the estimations. For the first problem, it is clear that utilities should draw on observed behavior if possible, but the second problem may be less clear.

One essential difference between Pacman and games like Chess or Draughts (where look-ahead trees have been tried and proven [8]) becomes apparent when evaluation of the utility of states is attempted. The locations of entities (the sum of which is referred to as a player's *position* in board games), in Pacman is not enough to fully describe the subtleties of the game because the game is not turn-based, and so the relative timings of parallel deterministic event sequences is important to play mechanics. Our features describe player behavior across multiple states, as in most cases the core behavior that a feature is trying to capture cannot be reasoned about from the perspective of a single state. Thus the estimation of most features has an inherent time-series structure and solves the second problem.

Incorporating these features required a fundamental change to the algorithm because the original used *post-hoc* utility calculation. This means that the calculation of utility at a given depth only referenced the single state from the tree node at that depth. Thus, the relationship that the referenced state had with the features from the previous or next states had to be inferred, if it was calculated at all.

This may work for game playing AI or bots, that have the aim of *beating* the player, but when the aim is to *model* the player some account must be made of the player's point of view. A human player's cognition has a strong time-sequence orientation [9]. Also during faster, more stressful game-play some players may begin to focus on a select number of **evolving** features, and disregard the full scope of detail of the current state [10]. Thus in order to model the player our algorithm had to be altered; its operation is described in section III.

In section IV we describe how the experiment was prepared and data collected. Then in section V we give the results of the tests, and describe how valuable player-related information can be extracted from the modeler's output. Finally section VI provides our conclusions.

II. Features Of Game-Play

Ultimately player utility represents the value of a process of play, for example the process of collecting all the Dots in an area, or getting a Pill then chasing Ghosts. Utility for a process is found by considering the sequence of states in which the process occurs, programmatically extracting some knowledge of what the player has been doing in that sequence, and applying some valuation to that knowledge.

Knowing what the player does in a sequence of play requires codifying all the pertinent things they are able to do within the closed system of the game. For example, a player can usually move pieces or an avatar. The *way* they move very often has qualifiers, such as quick or aggressive movement. These can be codified as *behavioral features*.

In more detail, data from players is described by in-game variables, logged at significant points of change in the game state. 'Significant' in this case is defined as any action of any of the game agents which affects the state of game *and* cannot be interpolated from adjacent states in the log. Extracting features involves identifying player 'behaviors' over sequences of one or more game state variables. Behaviors can be defined as macro level (i.e. more than one state change) game play actions within repeating situations. In other words, they are patterns of player action that could be observed to be consistent across repetitive game situations by a hypothetical third party observer.

Our process for deriving features was to work from the general to the specific, a top-down approach that maximizes the chance of defining all the features necessary to express distinct player behaviors. The first step was to create a list of high-level behavioral *traits* that can be expressed, either positively or negatively and to varying degrees, in most games and by most players. The list comprises Aggression, Caution, Planning, Decisiveness, Thoroughness, Control Skill and Resource Hoarding, and is repeated with detail below. Having this list, we tried to posit distinct features of play in Pacman that represented a behavioral aspect from the list, as many as it seemed were plausible. A group of aggregate features was also specified to hold simple counts such as number of lives gained.

The set of high-level traits was specified with regard to a generic concept of play in contemporary games, and thus applicable to any given player and game. To suitably limit the mode of investigation, the physical reactions of players (e.g. facial, galvanic skin response, other biometrics) were not recorded – the only concern was how player actions reflect their interaction with the mechanics of a game. In studying how players interact with games we covered systems of play and modes of interaction [11], as well as generic motivations for play [12] and player type [13]. Considering these models together gave us a picture of the forms of behavior that can be expressed by most players in many games. The list was not designed to be complete but only be comprehensive enough to cover Pacman and yet still be generic; it is presented in no particular order:

Aggression: describes forward and hasty action with respect to opponents or obstacles. In Pacman this must relate to how consistently and riskily a player chases after *all* the ghosts after eating a Pill.

Caution: describes how much the player guards their avatar, lives, or other representation of failure risk. In Pacman, this might be how much distance they *consistently* keep between Pacman and the ghosts when not in the Hunting state.

Planning: is achieving higher level goals with a premeditated, linked series of actions. A Pacman example could be luring Ghosts near to a Pill, then eating them all.

Speed: primarily, this is how fast the player completes discrete segments of game play, be they levels, objectives or even moves (as in chess); secondarily, whether they divert from the 'quickest-possible' approach for any reason.

Chasing high score/optimal result: this would often be a meta-behaviour, of replaying the game to gain ever-higher scores. Classically, this is the key motivator in Pacman.

Decisiveness: describes players who don't backtrack, or make oscillating-type movements. This goes to skill of using the controls also, but the distinction is that decisiveness operates on a macro scale.

Thoroughness: is attempting to do or see everything available, also solving puzzles by exhaustion of combinations. This is hard to define in Pacman, since the only way to progress is to collect everything.

Control Skill: means fine, precise and detailed command of control schemes, such as accuracy in a shooter or knowledge of hotkeys in an RTS. In Pacman, this is mostly how well corners are turned and Ghost near-misses are handled, and perhaps also how quickly local sections of Dots are cleared.

Resource hoarding: games are often described as having an 'economy': ammo and health in combat games, items and virtual currency in RPG's are just two examples. Resources in Pacman are hard to delimit from points, but lives and fruit are valid examples. Pills are often hoarded as 'insurance', to deploy at strategic moments.

Under each of these generic headings, we developed a set of specifications of features of game play that pertain directly to Pacman, along with the group of simple aggregate features. Developing specifications involved first observing patterns of play in Pacman games, so that the various patterns of play performed in the game were known to us. The intent of the player in performing each pattern of play was associated with a generic behavior as listed above. A description of the pattern of play was written in natural language based on its observation and associated generic behavior. Having specified patterns of play, feature development then entailed finding and observing patterns in the logs of played games, and deconstructing their operation to the most atomic level possible so that each pattern of play could be codified as an algorithm.

Working from the first principles of these specifications, features were implemented in C++ in our Pacman game engine. Thus reasoning about the specification of a behavioral feature inspired the algorithm which derived that feature. Most of the features were derived by the same basic algorithmic procedure; that is, iterating through the states from a raw data log file the algorithm searches for constraint harness parameters to satisfy. The constraint harness is a set of logic statements, and the parameters are simply numeric thresholds. An example is the initial constraint for the first Aggression feature, *A1_Hunt Close To Ghost House*:

```
if( PacAttack && (
myDist(xGhost[0],yGhost[0],xHse,yHse) < 3 ||
myDist(xGhost[1],yGhost[1],xHse,yHse) < 3 ))
```

This simply says execute the 'then' portion of the 'if' statement when Pacman is Hunting the Ghosts, and one Ghost is within a manhattan distance of 3 from the centre of their House. Within the 'if' statement is more logic to induce whether Pacman is closing in on the House, and what happens when he gets there. If the right conditions are satisfied then the function increments a counter which is returned as the final result once the function has iterated through all the raw data. More complicated functions might look at a wider spread of the raw data, and try to meet more complicated constraints, but the algorithmic pattern is much the same for most features.

We applied these features across only a few projected states of play, and constructed a mapping from the current game state to one weighted feature vector for each path in the look-ahead tree. These vectors represent the player's possible choices, and evaluating their utility allowed us to predict the player's most likely choice. The entire list of features used is described in Appendix A.

III. ALGORITHM OPERATION

The changes to our previous work described above implied an update to our modeling algorithm, reflected in the new decision theoretic equation (1), below. We replaced the summation of utilities for each state in a sequence with a single utility for the whole sequence, corresponding to one game-play feature. This update radically simplifies the formula by removing the requirement to specify the utility of every state. All that is needed is the sequences of states that comprise a possible course of action, which is equal to a single path from the look-ahead tree. However many of the original definitions [2] are still relevant to understanding, so a brief description follows.

A rational player's decision making situation involves picking from a finite set A of the alternative courses of action (or plans), that are available to execute. A member of A such as a can be thought of as a plan consisting of consecutive moves extending to the future time t_a. The (computational) limit on this look-ahead time will be t_{max} – in our case t_a will almost always equate t_{max} since for any one plan, only Pacman's death or the end of the level will result in a cessation of planning. Such an action plan occurs in a state-delimited world, formalised as the set of all possible states of the world S. So each a is a sequence of states $s \in S$, starting with a state 'adjacent' to the current state and ending with s^{t_a}. Since all the states considered in each decision-making situation are limited by t_{max} they form a subset of S which we call S_t. To obtain the necessary ordering of s when selecting from S_t so that the sequence a makes sense (since S_t is unordered), we identify each state by its distance in the future, i.e. s^t. In terms of the look-ahead tree, a is a path and s^{t_a} (the last state in a) is a leaf which can uniquely identify the path.

In Pacman only the current state is known: no **current** information is hidden but future states depend on stochastic elements, so it is a game of semi-perfect information. 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 temporal projection is expressed as $proj: S \times A \rightarrow P(S)$; which means that the action a given the current state s^0 results in the probability of the projected states $proj(s^0, a) = P_a(S)$. Specifically, p_a^t will be the probability assigned by $P_a(S)$ to the state s^t.

This paper retains the definitions established in previous papers, but alters the application of utility. The utility function $util$ now applies to an entire sequence-of-states or plan a, rather than just one state: encoding the desirability to the player of the projected sequence and mapping it to numerical output $util: S \rightarrow R$. f is a member of the set F of all implemented features, and $util$ will choose different features f_a depending on the states in plan a (i.e. some features are not relevant in some states). $util$ maps states to a real number, and the summed output of multiple f_a gives the utility score of a plan a: the best scoring plan predicts the next move.

$$a_t^* = ArgMax(a \in A)\, util\left(\sum_{f \in F} f\,(a)\right) \quad (1)$$

We implemented this equation using a look-ahead tree – described in the schematic in figure 1 below – next we explain some aspects of how this works in Pacman.

A. The Tree of Future Moves

The goal of implementing equation (1) is to search the tree of utility-weighted future moves. Nominally, this tree would be built by finding all possible combinations of positions which the in-game actors can occupy when they move one step, and then iterating that calculation for a computationally tractable number of steps. Having enough steps required us to improve the execution speed of the algorithm. Standard methods for pruning a game theoretic look-ahead tree, such as the alpha-beta heuristic, are less applicable to time-based games. By considering the look-ahead activity from the player perspective, we conceived an effective yet simple optimization. From the player's point of view, future ghost positions are a best a probability distribution: thus we do not need to calculate exhaustive look-ahead trees. Reduction of the data space dimensionality can be achieved by transforming the combination of Pacman and the Ghosts' possible moves into the set of Pacman's possible moves plus a probability distribution over the Ghosts' possible moves with respect to Pacman's moves.

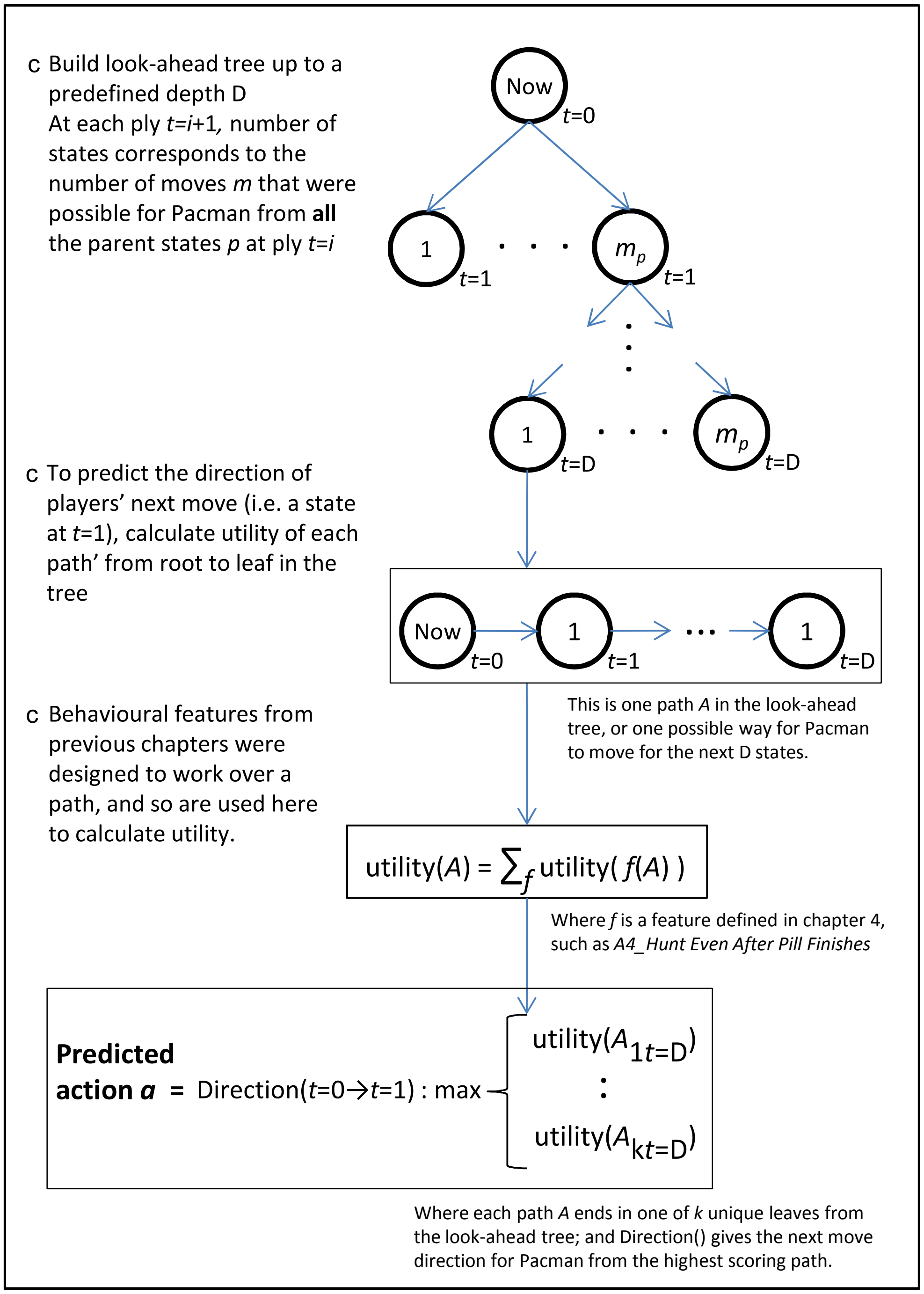
c Build look-ahead tree up to a predefined depth D
At each ply t=i+1, number of states corresponds to the number of moves m that were possible for Pacman from **all** the parent states p at ply t=i
Now
t=0
1
t=1
. . .
m_p
t=1
1
t=D
m_p
t=D
c To predict the direction of players' next move (i.e. a state at t=1), calculate utility of each path' from root to leaf in the tree
Now
t=0
1
t=1
...
1
t=D
This is one path A in the look-ahead tree, or one possible way for Pacman to move for the next D states.
c Behavioural features from previous chapters were designed to work over a path, and so are used here to calculate utility.
utility(A) = ∑_f utility(f(A))
Where f is a feature defined in chapter 4, such as A4_Hunt Even After Pill Finishes
Predicted action a = Direction(t=0→t=1) : max
utility(A_1t=D)
utility(A_kt=D)
Where each path A ends in one of k unique leaves from the look-ahead tree; and Direction() gives the next move direction for Pacman from the highest scoring path.

Fig.1. Schematic of the operation of the Decision Theoretic feature-based predictive player modeling algorithm. Multiple features contribute to the utility of each path in the tree, to evaluate action ***a***.

A random selection from this distribution is what drives Ghost movement in the game engine. Thus the functionality was readily available to calculate the Ghost positions in the look-ahead tree. We chose to select the highest probability position automatically. Since the choice mechanism for 'real' Ghost movement was stochastic, stochastic choices in building the look-ahead tree could cause the predicted moves to diverge from actual movement, more than necessary. Although it was not necessary to predict exact Ghost locations, using approximately the right locations would improve the relevance of many of the features calculated. Also, all probabilities of future states were rendered equal due to the way the look-ahead tree was calculated with respect to Ghost movements.

By branching only for the potential moves of Pacman, and not the Ghosts, we reduce tree branching by a factor of at least ~2.25 for every Ghost. The tree branching factor corresponds to the number of possible moves for all actors considered at each step, which is equal to the connectedness of the square the actor is in. Our test-bed game map has 182 navigable squares. Of these, 143 are connected to 2 squares; 32 are connected to 3 squares; and 7 are connected to 4 squares. Therefore, if we consider the future moves of a number of actors i, the average branching factor would be given by equation (2):

$$^{143}/_{182} \cdot 2i + {}^{32}/_{182} \cdot 3i + {}^{7}/_{182} \cdot 4i \approx 2.252i \qquad (2)$$

We only consider the human player, so the nominal average branching factor equals 2.25. In a given level or game players will traverse each map square more than once, and the frequency of traversal will vary across the map. By their nature, the junction squares will be traversed more often, which implies the average branching factor will be higher than given by equation (2). Indeed in later tests when we analyzed the time taken at each depth of tree search, every increase of depth by 1 multiplied the time taken to calculate the algorithm for a single state by ~2.75. That implies that the actual branching factor is ~2.75 on average.

Finally, in look-ahead tree terms equation (1) specifies the tree of potential next states up to a given depth and uses behavioral features adapted to work over short sequences of states, to return a utility value for an entire path of the future moves tree. Thus the predicted next move, the first in the highest-scoring path a, corresponds to the direction that leads to the highest utility and is the final output of our algorithm. This was tested as follows.

IV. Experimental Set up

Data for the experiment was composed of the logged games of 37 volunteer players from among the under-graduate classes in the host university. They were also given a short survey to ascertain the following personal details.

- Sex
- Age
- Gaming habit, choice of: *Hardcore*, *Casual*, *No Idea*
- Pacman Experience, choice of: *Newbie*, *Beginner*, *Intermediate*, *Expert*
- Top 3 game-play styles enjoyed; unrestricted answer
- One style of game play hated; unrestricted answer

The last two questions were not restricted to pre-defined game styles, because it was felt that player understanding of the terms describing styles of game play is so variable that better results would be obtained by allowing players to use their own terms and interpreting them categorically.

A. Game Description

Although it is a common game, the Pacman test bed used was an interpretation of the original Namco game rather than a clone, and so below we describe our version. In all versions of Pacman the goal is to move around the game level and obtain all the collectables, thus progressing to the next level. Points are awarded for collectables and eating Ghosts. Our version of Pacman's game play is described in the following list (Initial Caps are used to describe game entities and their actions in this description; 'the player' and 'Pacman' are inter-changeable terms).

The game world is a 20x20 matrix which constitutes a level, and each element of the matrix can be Wall, Pill, or Dot.

Pacman and the Ghosts (the in-game actors) Move about the level along two axes, horizontal and vertical.

Pacman Eats Dots & Pills when he Moves over them.

When a Dot or Pill is Eaten, it's 'element' becomes empty.

By Eating a Pill, Pacman switches the game state from one where he is vulnerable to the Ghosts (the normal state), to one where he is able to Eat the Ghosts (the Hunt state).

If Pacman collides with a Ghost, he can Eat the Ghost if in Hunt state, otherwise Pacman is Eaten and loses a Life.

Eaten actors re-spawn at their original start point, unless Pacman has run out of lives, when 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.

Points are scored as follows: 10 per Dot, 50 per Pill, (100*(2 ^ Number of Ghosts Eaten)) per Ghost Eaten after a Pill.

Ghosts start in a central box area called their 'House'.

Ghosts are permeable and do not interact with Dots or Pills or obstruct each other.

Difficulty is measured by speed of Ghost movement, probability distribution for their movement direction, and length of Hunts conferred by Eating Pills. Ghost speed increases every 5 levels starting in the 2nd level. Probability changes 5 times after the initial setting, at the start of levels 2, 4, 7, 11 and 16. Length of Hunts is a continuous function of the level number and quantity of Pills left in that level. This lends Ghosts some reactivity to the number of times they've been Hunted, as they run away for less time each Hunt.

Ghosts move according to a pseudo-random probabilistic control function based on Pacman's relative location and the (increasing) difficulty level. In the normal game state, the probability of moving closer to Pacman is higher than the probability of moving to any other adjacent point. The direction is reversed when the game is in the Hunt state.

V. Test Results

The logged games of 37 players were tested off-line in order to allow full exploration of parameter combinations, which under certain settings required up to 32 hours of computation[1]. If testing had been performed during actual play, computational constraints would never have allowed any deep look-ahead tree searching. The exploratory analysis of the parameters of the algorithm formed the initial stage of testing, and is explained in greater detail in [14].

Once we established optimal values for algorithm settings, and determined the features to be included in utility calculation, we tested the set of archived games. In order to allow algorithm settings time to converge, we excluded from testing any game that had only two levels or less. To illustrate the performance of the algorithm over a single game, figure 2 shows the running accuracy total that is obtained every state by the formula: *number of correct predictions / number of states*.

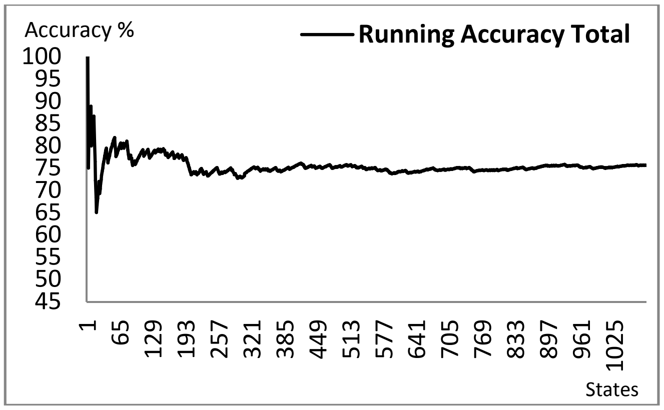

Fig.2. Running total for prediction algorithm accuracy on a single game.

As this figure shows, accuracy varies for several hundred states (the average number of states per level was 327) and then settles into a small range. Since it drops from 100% in the first few states, it is capable of high accuracy immediately and does not have to learn accurate predictions. The accuracy results for all games in the archived test-set are shown in figure 3, giving a clear impression of the overall level of accuracy achieved, and the variation between games that can occur even for a single player. The final accuracy from all games was 70.5%: this represents a lift of ~26% from the default value of ~44% that would be obtained by random guesses of next direction at every position on the game map. The output of the tests gave the raw data necessary to begin interpretation of the behavior of players.

A. Player Observations

The core of our approach involved predicting player actions and comparing these predictions with actual player activity. The validation of these predictions forms a model with the ultimate aim that over enough instances of play this model would embody valuable information on player behavior. The value of this information is based on the features describing game-play, which the modeler used to reason over players' actions. The log of the player's game can thus be composed into a set of occurrences of the features describing their behavior, and the patterns of occurrence can be interpreted to give an insight into the player at an even higher level.

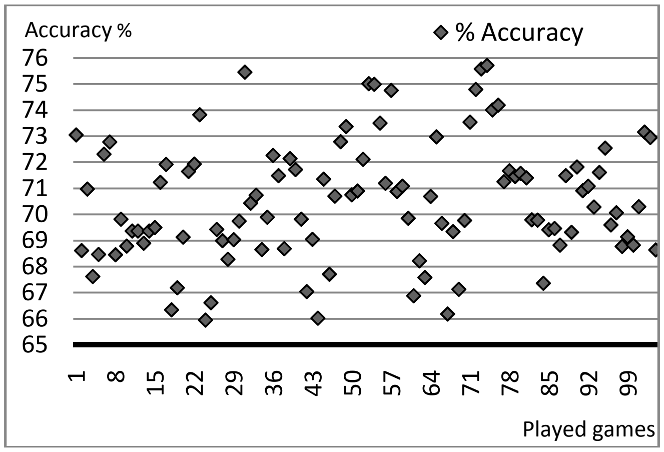

Fig.3. The best accuracy scores for the 105 archived games from 37 players.

Categorical partitions of the players, based on the self-reported survey data Gamer kind, Pacman experience, Age and Sex, all correlate negatively with prediction accuracy as shown in table 7.8. Only Age is anywhere near a strong correlation though. Two thirds, or 6 of the players aged over 25 years are in the lowest 10 predicted accuracies.

Table 1 Spearman coefficients and p-values for correlations of categorical and accuracy data.

Player Data Category	Spearman Coefficient	2-tailed p-value
Sex	-0.21	0.2143
Age	-0.39	0.0158
Pacman Experience {Newbie, Beginner, Intermediate, Expert}	-0.19	0.2630
Gamer Kind {No Idea, Casual, Hardcore}	-0.12	0.4778

Aside from the immediate value of accurate predictions, a brief analysis of the trends of behavioural features as recorded during algorithm operation should contribute to the player knowledge agenda. The analysis was performed on one of the archived games chosen randomly. Accuracy of prediction for this game was 70%. First we compared the firing frequency of features. Certain features that previously had been proven important, mainly concerning Hunting-related behaviours, could now only fire relatively infrequently. For example, the highest ranking feature in table 7.3 was *A4_Hunt Even After Pill Finishes,* yet this now fired ~5% as often as the lower ranked *C5_Moves With No Points Scored* (which could fire in any mode). Since

[1] At depth of 9, look-ahead tree search took ~14 seconds/state; test games were about 9000 states.

predictions are based on reported values, feature importance becomes dependent on frequency of firing. Thus the most important features were those relating to Points or lack of Points, and distance to the Ghosts. The value of features also provided information. For instance, for the analyzed game the distance measure during *Hunt* mode had an average of 13.49, while the distance at other times averaged 8.93 (both Manhattan distance). This suggests a player who prefers to get *further away* from the Ghosts during *Hunt* mode, which seems counter-intuitive until we recall the strategy of using the Ghost repelling effect of Pills to win breathing space to collect the Dots. Such analysis of feature trends could improve their utilization in future work.

VI. Conclusions

We have described a method for in-game real-time player modeling based on a Decision Theoretic predictive algorithm, and implemented in our Pacman test-bed. It was tested on players with a range of experience of playing Pacman, and the results examined to determine new valuable information on the behavior of players.

The described methodology would benefit from further tests, with greater variance in the demographic background of subjects. Of particular importance in this method is deciding the right questions to ask of the results.

A. Future Work

Three possible improvements to the algorithm suggest themselves as especially valuable. These are: extending predictions beyond the next state to a full path from the look-ahead tree; adding consideration of longer-term sequences of moves than the look-ahead tree can handle; and dynamically adjusting feature weights in response to the results of the player classifier.

Currently, the algorithm predicts only the player's next move. However, it does so by calculating the utilities of entire future paths, and these paths could form the basis of a more detailed prediction. The reason why this was not done was because in each new state, the last state's predicted path would need to be integrated with a new predicted path, or even with every path in the look-ahead tree. This would be quite a complex implementation task, and might raise many questions requiring extensive testing, hence an issue for future work. To implement the idea, the first requirement would be storage. Exactly what would be stored from the look-ahead tree with its utility weights, and how would it be updated in subsequent levels? At one extreme, an entire tree could be stored, its weights decaying to 0 in order to decrease its influence over time. The latter idea brings up the requirement for an update policy, which in turn is closely related to the need for an integration of past and current predictions. All in all, these would be complex questions.

Current results suggest that while we can calculate a utility for a short series of moves (which we could call a *plan*), often the player will be considering a much wider picture, involving more moves than it is practical to calculate a utility for. Plans of this nature, which we would call *strategies*, would require a higher level of abstraction than the look-ahead tree can represent and still be computationally viable.

If we defined medium- to long- term strategies and their utilities using ML methods, then instead of *building* weightings of all strategies possible from a position, our algorithm would *learn* some set of favorable strategies. This favors an inductive approach over a deductive one, which as well as being computationally efficient, is assumed to be more similar to human cognition and thus more suitable for a player modeler.

Having incorporated features of play, as play progresses some subset of the chosen features should be reflecting the decisions of the player. If all their possible choices at the current state appear equally likely, we may not be able to make a confident prediction, but then once they make a choice whatever they decide will indicate what their play preference was. This *a posteriori* information should be used to pick out those features in the search tree relating *only* to the direction the player actually went in. The weights of these features can be increased to reflect those play preferences that they did in fact correspond to. Adjusting weights to maintain high accuracy of predictions would allow the method to improve its player model in real-time.

Appendix A

The full list of features used is reproduced below. Descriptions in column 2 are in natural language first, and then pseudo-code. Prefix letters in the name indicate the behavioral trait the feature was supposed to represent. *A* for aggression, *C* for caution, *D* for decisiveness, *P* for planning, and *S* for the simple count of a variable. More detail is available in [14].

A1_Hunt Close To Ghost House	Counts instances where Pacman follows the Ghosts right up to their house while attacking them.
	`while(PacAtkBool = True) { if(dist(Gst{1,2}XY, HseXY) < 3 & dist(PacXY, HseXY) < 5) then Output++ }`
A4_Hunt Even After Pill Finishes	How often the player chases the ghosts until *after* the Pill effects wear off
	`while(PacAtkBool@State(t-20)){ if( dist(PacXY, Gst{1,2}XY) == 1 )then Output++ }`
A6_ Chase Ghosts or Collect Dots	Whether the player uses the Pills to chase ghosts, or just continues collecting Dots
	`while(PacAtkBool){ if( dist(PacXY, Gst{1,2}XY):decrease) then Output++}`
C1.b_Times Trapped and Killed By Ghosts	Threat perception: Pacman trapped in corridor by Ghosts, and consequently losing a life.
	`if( (`*C1.a_Times Trapped By Ghosts*`) & Lives@State(t+10):decrease ) then Output++`

C2.a Average Distance to Ghosts	Average distance the player keeps from ghosts, when not on a Pill.
	`if(!PacAtkBool) then Output = SumAcrossRecords( manhattanDist(PacXY, Gst{1,2}XY) / NumRecords`
C2.b Average Distance During Hunt	Average distance the player keeps from ghosts, when in Hunt mode.
	`if(PacAtkBool) then Output = SumAcrossRecords( manhattanDist (PacXY, Gst{1,2}XY) / NumRecords`
C3_Close Calls	How often player comes very close to a ghost, when not on a Pill, and doesn't die afterwards!
	`if( dist(PacXY, Gst{1,2}XY)@State(t-5) == 1 & Lives@State(t-5) == Lives) then Output++`
C4_Caught After Hunt	Whether chasing the ghosts until after the Pill wears off costs a life
	`if( PacAtkBool@State(t-1) & !PacAtkBool & (Lives > Lives@State(t+15))) then Output++`
C5_Moves With No Points Scored	Count traversals of empty space/visits to squares without Dots
	`if( !PacAtkBool & Dots=Dots@State(t-1) & Lives=Lives@State(t-1) & Pills = Pills@State(t-1) ) then Output++`
C7_Killed at Ghost House	Counts if the player dies collecting Dots around the ghost house
	`if( dist(PacXY,HseXY) < 5 & !PacAtkBool & Lives > Lives@State(t+10) ) then Output++`
Cherry Onscreen Time	Sum of Cherry Boolean flag – gives total time onscreen
D2 Pacman Vacillating	Oscillating movement with no ghosts near.
	`if( PacXY != PacXY@State(t+2,3) & PacXY@State(t+8,9) & PacXY = PacXY@State(t+4..7) & PacXY@State(t+10) ) then Output++`
P1_Waits Near Pill to Lure Ghosts	How often the player waits beside a Pill to lure the Ghosts in – predicates of sub-features below depend on satisfaction of this one
	`if(Cycles-Cycles@State(t-5) > 1000 & dist(PacXY,PillXY)<8 & dist( PacXY, Gst{1,2}XY) < dist(PacXY,Gst{1,2}XY) @State(t-5)) then Output++`
P1.a_Lure: Count Moves While Waiting for Ghosts	During execution of the feature above, this counts how much Pacman moved after he got close to the Pill, while the ghosts were still far away
	`Predicate cannot be expressed concisely here`
P1.c_Lure: Number Ghosts Eaten After Lure	Counts how many Ghosts Pacman eats after doing a lure
	`while( PacAtkBool ){ if( Points > Points@State(t-1)+99 ) then Output++ }`
P1.d_Lure: Caught Before Eating Pill	Records if Pacman fails to complete the lure, because he was caught before eating the Pill
	`if( Lives < Lives@State(t-1) ) then Output++`
P4.a Average Speed Hunting 1st Ghost	Count moves taken on average when Ghost 1 is Hunted and caught
	`while( PacAtkBool ){ if( Points <= Points@State(t-1)+99 ) then Output++ }`
P4.b Average Speed Hunting 2nd Ghost	Count moves taken on average when Ghost 2 is Hunted and caught
	`while( PacAtkBool ){ if( Points <= Points@State(t-1)+199) then Output++ }`
Points_Max	Max Points value
S2.a_Lives Gained	Count number of times Pacman gained lives – i.e. ate a Cherry
S2.b_Lives Lost	Count number of times Pacman lost lives – i.e. caught by ghost
S4_Teleport Use	Count the number of times Pacman uses a teleporter.

REFERENCES

[1] B. Cowley, D. Charles, M.M. Black, and R.J. Hickey, "Using Decision theory for Player Analysis in Pacman," Proceedings of the SAB Workshop on Adaptive Approaches to Optimizing Player Satisfaction, Roma, Italy: 2006, pp. 41-50.

[2] B. Cowley, D. Charles, M.M. Black, and R.J. Hickey, "Data-Driven Decision theory for Player Analysis in Pacman," Proceedings of the Optimizing Player Satisfaction Workshop, Stanford University, Stanford, Ca: AAAI Press, 2007, pp. 25-30.

[3] D. Thue and V. Bulitko, "Modeling Goal-Directed Players in Digital Games," Proceedings of Artificial Intelligence and Interactive Digital Entertainment 06, Stanford, CA, USA: AAAI Press, 2006, pp. 86-91.

[4] J. Donkers and P. Spronck, "Preference-based Player Modelling," AI Game Programming Wisdom 3, Hingham (MA): Charles River Media, 2006, pp. 647-659.

[5] B. Cowley, D. Charles, M. Black, and R. Hickey, "Toward an understanding of flow in video games," ACM Comput. Entertain., vol. 6, 2008, pp. 1-27.

[6] P.J. Gmytrasiewicz and C.L. Lisetti, "Modeling users' emotions during interactive entertainment sessions," Proceedings of AAAI 2000 Spring Symposium Series. 20-22 March 2000, Stanford, CA, USA: AAAI Press, 2000, pp. 30-35.

[7] A. Ortony, G.L. Clore, and A. Collins, The cognitive structure of emotions, New York: Cambridge Uni Press, 1988.

[8] A. Samuel, "Some studies in machine learning using the game of checkers," IBM Journal of R&D, vol. 3, 1959, pp. 229, 210.

[9] R. Penrose, The emperor's new mind : concerning computers, minds, and the laws of physics, Oxford; New York: Oxford Uni Press, 1989.

[10] N. Baumann, R. Kaschel, and J. Kuhl, "Striving for unwanted goals: stress-dependent discrepancies between explicit and implicit achievement motives reduce subjective well-being and increase psychosomatic symptoms," *Journal of Personality and Social Psychology*, vol. 89, Nov. 2005, pp. 781-99.

[11] K. Salen and E. Zimmerman, Rules of play : game design fundamentals, London: MIT, 2004.

[12] R. Caillois, Man, play, and games : Translated from the french by Meyer Barash, New York: Free Press of Glencoe, 1961.

[13] C. Bateman and R. Boon, 21st century game design, London: Charles River Media, 2005.

[14] B. Cowley, "Player Profiling and Modelling in Computer and Video Games," thesis submitted at University of Ulster, Coleraine, 2009.