Modelling Goal-directed Players in Digital Games

David Thue and **Vadim Bulitko**
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada, T6G 2E8
{dthue | bulitko}@cs.ualberta.ca

Abstract

The pursuit of a viable model of player behaviour has gained momentum in research in recent years, and it is beginning to attract the attention of the designers of next-generation digital games. In this paper, we present a novel enhancement to player modelling that is well-suited to the digital role-playing and puzzle game industries, titled Goal-Directed-Player Modelling, in which state abstraction based on a player's goals is used to improve the performance of a classifier for predicting player actions. We survey a set of related research, formally introduce a method for Goal-Directed-Player Modelling, and present empirical results which clearly show the ability of Goal-Directed-Player Modelling to greatly improve the accuracy of a simple, online, low cost incremental classifier to a level near those of more advanced and complex offline methods.

Introduction

Current single-player digital games are generic; regardless of who is playing, the player's experience is largely the same. While techniques such as scripting allow the player to choose to experience various alternative sequences of events in a game, they only aim to accommodate a handful of over-generalized player groups; often, only "good" and "evil" playing styles are considered. For example, in BioWare's *The Knights of the Old Republic*, the player is presented with moral choices: should he accept a poor citizen's payment for a service rendered, refuse the payment altogether, or intimidate the citizen into paying more (BioWare Corp. 2003)? Although such choices help influence the outcome of the story, little attempt is made to tailor a game's operation to players on an individual level. To achieve this improvement and make a digital game more player-specific, player information must be gathered, interpreted, and used. This task, known as *player modelling*, can be achieved in a variety of ways, some of which will be presented later in this paper.

Motivation

Although player modelling is of theoretical interest in itself, its potential uses in digital games are particularly compelling. Many of these uses are based on increasing the individual player's enjoyment of the game, as follows.

Dynamic Levels of Difficulty In goal-directed digital games such as first-person shooters or role-playing games, the player's enjoyment of the game is often directly related to the perceived difficulty of the goals he achieves. Setting goals that maintain a balance between being both challenging and achievable is a difficult aspect of game design; most current games defer this selection to the player, offering the choice of a "difficulty level" via an in-game menu. Unfortunately, the number of available levels is often small, making adjustments too coarse; a player may find one difficulty setting far too easy, and the "next hardest" setting impossibly difficult. Given a model that predicts the player's actions and, thereby, his likelihood of achieving his current goal, the level of difficulty of that goal may be adjusted in a more detailed manner, and independently of other goals. For example, consider a hypothetical role-playing game that can predict that the player will attack a pack of monsters and easily defeat them. With dynamic levels of difficulty, the difficulty of defeating the monsters could be increased using the methods that are traditionally implemented in the difficulty-level selector, but instead of applying this increase uniformly to all goals in the game, the difficulty of only the current, excessively easy goal could be increased, at a resolution much finer than those presented via in-game menus.

Helpful NPCs In combat-oriented games such as ArenaNet's *Guild Wars*, non-player characters (NPCs) can be allied to the player in battle, aiding in defeating any opponents that are encountered (ArenaNet 2005). Having effective assistance allows players to accomplish more difficult goals, which heightens their sense of achievement and enjoyment of the task. Unfortunately, the "intelligence" of NPCs in current digital games is often governed by a pre-specified set of rules, similar to *if player health* $< 50\%$*, heal player* or *if player attacks target, attack target*. Although these rules can easily lead to reasonable behaviour, they rarely lead to effective behaviour; perhaps a player's strategy relies on a special ability that is available only when his health is below 10%, making the first rule above counter-productive. *Star Wars: Republic Commando*, a first-person squad-based action game heralded for its ally AI, uses several simple strategies that allow the AI-controlled squadmates to be useful to a human player (Lucas Arts 2005; Gamespot 2005). The communication mechanism, though

ingenious in its simplicity, requires the player to give explicit instructions to his squadmates by pressing various buttons; no inference of player intentions is apparent during gameplay, and the heavily scripted squadmate behavior is independent of the player's gameplay style or personality. By identifying player strategies via player modelling, complementary strategies for NPCs could be learned or evolved.

Obtaining a Player Model

Before a player model can be used to increase the player's enjoyment of a game, the model must be obtained. The remainder of this work investigates this task, and is organized as follows. First, the problem of player modelling is formalized for a general game scenario. Second, a set of related research is surveyed. Third, a novel approach to player modelling is proposed and analyzed, and empirical results are presented and discussed. Finally, a body of future work is proposed, and conclusions are given.

Problem Formulation

In general, a digital game has four primary components: a world, W, a set of attributes, A, which describe the world, a set of values, V, and a set of value-changing actions, Q. Specifically, every attribute $a \in A$ represents *information about some aspect of the world*; examples of attributes could include the position of a book, the mood of its reader, and the amount of light reaching its pages. Every attribute, a, can assume one of several values, $v \in V_a \subseteq V$, which describe the current configuration of a, $Value(a) = v$. The state of the world is defined as a set of pairs specifying the current value of each attribute in the world: $State(W) = \{(a, v) | \forall a \in A, \exists v \in V_a, Value(a) = v\}$. When performed, each action $q \in Q$ changes the values of one or more attributes in $A_q \subseteq A$: $\forall a \in A_q, q(Value(a) = v) \rightarrow Value(a) = v'$, where $v, v' \in V_a$ and $v \neq v'$. As time progresses, actions from Q are applied to sets of attributes in A, changing the values of those attributes and thus the state of the world.

For example, consider the images shown in Figure 1. There are two attributes, $A = \{ball.position, ball.colour\}$, two possible values for each attribute, $V_{ball.position} = \{0, 1\}$, and $V_{ball.colour} = \{black, white\}$, and two actions, $Q = \{push, paint\}$. The $push$ and $paint$ actions operate on the $ball.position$ and $ball.colour$ attributes, respectively. From the initial world state shown in Figure 1 a), where $Value(ball.position) = 0$ and $Value(ball.colour) = white$, either action in Q may be chosen. Figure 1 b) shows the result of the $push$ action: the value of $ball.position$ has changed to 1. Figure 1 c) shows the result of the $paint$ action: the value of $ball.colour$ has changed to $black$.

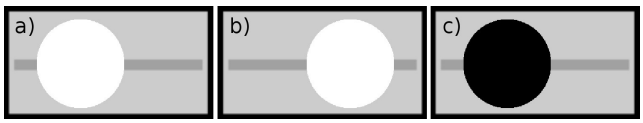


Figure 1: Three possible world states. a) is the initial state, b) is the state after action $push$ is taken in a), and c) is the state after action $paint$ is taken in a).

In single-player games, the player is responsible for determining which action to perform at any given time, based on a perceived subset of the current state of the world: $Player(U) \rightarrow q \in Q$, where $U \subseteq State(W)$. In digital games, the player must indicate their desired action to the game; given this ability to observe the actions chosen by the player when presented with various states of the world, digital games have the opportunity to *model* the actions of the player; that is, given a subset of the current world state, $U \subseteq State(W)$, determine which action the user will take: $\forall q \in Q, Model(U) \rightarrow q \Leftrightarrow Player(U) \rightarrow q$.

In Machine Learning, player modelling is the task of learning a classifier; that is, a function that assigns every subset of the world state to one of n classes, where each class uniquely represents one of n available player actions. From the perspective of Reinforcement Learning, player modelling resembles policy iteration; a policy represents a mapping from world states to actions, and at each timestep the policy is evaluated and improved. The incremental nature of policy iteration makes it well-suited to modelling players *during* gameplay. In terms of Heuristic Search, player modelling represents the learning of a heuristic to guide the search agent through a space of states and actions, along the same path that the player would take, given the opportunity to do so. It is this distinction that separates player modelling from the typical applications of methods in the above fields: instead of learning optimal or near optimal solutions to a problem, player modelling seeks to learn how to *tackle* the problem in the same way that the player being modelled would; whether or not a solution is found is often irrelevant.

This paper focuses on a particular subfield of player modelling: modelling *goal-directed* players. Henceforth, a goal is defined to be a subset of a state of the world, $G \subseteq State(W)$, such that the player's choice of actions will be guided by his desire to bring the attributes in G, $A_G \subseteq A$, to the corresponding values specified in G. An approach to modelling the behaviour of goal-directed players is presented later in this paper.

Related Work

A naive approach to player modelling relies on tracking user actions from every visited world state. Given a world with $|A|$ attributes, a, each having $|V_a|$ possible values, the number of possible world states is given by:

$$WorldSize = \prod_{i=1}^{|A|} |V_{a_i}|$$

When *WorldSize* is large, any given configuration of states is unlikely to reoccur in its entirety; this makes the naive approach problematic in two respects:

- information obtained for world states that *do* occur is unlikely to ever be used, and
- when a query is made about the player's next action given the current world state, it is likely that this world state has not previously occurred, meaning that no player information is available.

How can this problem be solved?

In a recent publication on adaptive game design, Charles *et al.* address the first point, presenting a method for player modelling based on *profiling* (Charles *et al.* 2005). Formally, profiling is "a technique whereby a set of characteristics of a particular class of person is inferred from past experience" (Electronic Benefits Transfer Glossary 2005). In their work, Charles *et al.* use game data to infer values for a small set of player characteristics, such as "good at solving puzzles", or "uses varied tactics"; the set of characteristics must be tuned for games in various genres. By using abstraction to form a profile of each player, information concerning similar players can be used and updated in groups. This mitigates the problem of specific world-states reoccurring infrequently, as the obtained information is distributed over a class of players. Although their technique is intuitive, Charles *et al.* fail to consider player goals; knowing that a player uses varied tactics may be useful, but knowing his current goal could let a helpful non-player character estimate *which* tactic he might use.

In recent work by Yannakakis and Maragoudakis, the second point above, that of predicting player actions in previously unseen world states, is addressed (Yannakakis & Maragoudakis 2005). Focusing on predator/prey games, Yannakakis and Maragoudakis construct their player model using game data as the input to a Bayesian Network, which allows the model to be used as long as player information is known for at least a *partial* set of current game data; matching the complete set is not required. By relaxing the set of data required to use the player model, the problem of predicting actions for previously unseen world states is reduced. Although preliminary results indicate that their method works well, Yannakakis and Maragoudakis do not seem to consider the high-level knowledge that players in predator/prey games have well-defined goals (to avoid the predators, or to catch the prey).

Albrecht *et al.* tested several Bayesian Networks for predicting the goals, actions, and locations of players interacting in a Multi-User Dungeon, a domain similar to present-day multiplayer role-playing games (Albrecht, Zukerman, & Nicholson 1998). By examining a player's actions, locations, and previously completed goal, the player's current goal (one of 20 possibilities) is predicted with a promising level of success. In addition to predicting player goals, Albrecht *et al.* attempt to predict player actions based on the estimate of their current goal, and obtain results that are visually similar to those given later in this paper. The key difference between the approach of Albrecht *et al.* and the technique we present here is that their Bayesian Networks require a significant amount of training (80% of the full data set) to obtain the results shown for tests run on the remaining 20% of the data, while our method is fully online and incremental; no prior training is required, as each action prediction is made based solely on the player data observed thus far. Additionally, the predictions made by the method of Albrecht *et al.* are made based on data gathered from all players, while our method is based solely on data gathered from the current player being modelled.

Goal-Directed-Player Modelling

As mentioned previously, this paper focuses on modelling *goal-directed* players; this task is henceforth referred to as GDP-Modelling (Goal-Directed-Player Modelling). Goal-directed player behaviour is particularly prevalent in both digital role-playing games and puzzle games. In the former, goals are provided to the player in terms of *quests*, such as "retrieve the magic sword from the dragon's lair"; in the latter, solving an 8-puzzle is a potential goal. How can knowledge of a player's goal be used in forming a player model? The key to modelling goal-directed behaviour is that the goal is known at all times. Given this information, various methods can be used to compare the current state of the world to the goal state and assess their similarity. Our hypothesis is that *the differences between the current world state and goal state can be used to predict the player's actions*.

The study of Means-Ends Analysis performed by Newell, Shaw, and Simon offers insight into this approach (Newell, Shaw, & Simon 1960). The General Problem Solver they designed was based on a study of how humans solve problems as a sequence of transformations on objects, where each transformation is applied to reduce the differences between given and target objects. In the context of digital games, goal-directed players will perform actions to reduce the number of differences between the current world state and goal state. Given this knowledge, an abstraction can be constructed that groups world states by their similarity to the goal state; the task of measuring this similarity will now be discussed.

Measuring Similarity

Two types of measures of similarity can be used: domain-independent measures, and domain-dependent measures. Hamming Distance is an example of a domain-independent measure, being the number of substitutions that are required to transform one sequence of objects into another (Hamming 1950). For example, given two world states represented as bit vectors, 100010 and 110000, the Hamming Distance between them is 2; for the vectors to match, the second and fifth bits from the left must be changed. A possible domain-dependent measure is "Count Distance"; in a world where every attribute has the same set of possible values, the difference between the number of attributes in both G and W_G having a particular value can be compared. The previous two bit vectors each contain 2 ones, so their Count Distance is 2 - 2 = 0. In a digital role-playing game, Geographic Distance could be used as a domain-dependent measure. Abstracting world states based on the geographic proximity of their components could help in predicting the player's route and destination during travel in-game, allowing enjoyable challenges to be automatically placed along the way.

Building and Applying the Abstraction

Once a set of similarity measures has been chosen, a compact, abstract world may be constructed as follows. Let each measure of similarity define an abstract attribute of the world. Partition the range of possible values for each measure of similarity into a small set of ranges, and let each

range define an abstract value of the abstract attribute that represents that measure of similarity[1]. Given the current, non-abstract state of the world and the state expressing the player's current goal, the corresponding abstract world state can be found by computing the value of each measure of similarity between the world and the goal, and finding where each value falls in the set of ranges defined for that measure; this gives the current abstract value for every abstract attribute, thus defining the current state of the abstract world.

The Algorithm

Pseudocode for a modelling algorithm that uses goal-based abstraction is presented in Figure 2. The sole input to the method consists of a specification of the abstraction, that is, a set of similarity measures, each having a small set of ranges that separates the possible values for that measure. The main loop begins on line 1, and execution of the loop's body will repeat until play ends. On lines 2 and 3, both the goal world state and the current world state are obtained, and on line 4 they are compared; if they are equal, then the player has achieved the desired goal, and execution returns to line 2, where a new goal world state is obtained. Otherwise, execution proceeds to line 5, where the computation of the current abstract world state begins by initializing it to be empty. From line 6 to line 8, each measure of similarity (with the included set of value ranges) is used to compute the abstract value held by the measure's corresponding abstract attribute, and these attribute/value pairs are added to the abstract world state. On line 10, the abstract world state is used as the input to a Machine Learning classifier, which returns a prediction of the action that the player is about to take. Line 11 indicates the position in the algorithm where predicted actions might be used, perhaps to make a simple challenge more difficult, employ a complementary strategy, or offer help in solving a puzzle. On lines 12 and 13, the player's action is obtained (via traditional user-input) and used along with the current abstract world state to update the classifier with a new training instance. The player's action is applied to the world on line 14, and the resulting world state is obtained on line 15 and set as the current world state.

Theoretical Analysis

In terms of computational cost, the two most expensive components of the GDP-Modelling algorithm appear on lines 7 and 10 in Figure 2: using the measures of similarity to obtain abstract values, and using the classifier to predict an action. The following is an analysis of the first of these processes; relative classifier complexity will be discussed later on.

The computational complexity of computing the similarities between the world and goal states depends largely on each measure of similarity individually. For example, the computations of Hamming Distance and Count Distance are both in $\Omega(|A_G|)$ and $O(|A|)$, where $|A_G|$ is the number of attributes whose values are specified in the goal world state, and $|A|$ is the total number of attributes in the world state.

[1]The size of the set of ranges affects a trade-off between reducing the effective size of the game world and having good resolution in the abstraction.

GDP-Modelling

Input: *mos* ← measures of similarity
mos.ranges ← value ranges for each measure

```
1  while the player is still playing do
2    gws ← goal world state
3    cws ← current world state
4    while cws ≠ gws do
5      aws ← ∅
6      for each mos do
7        abstractValue ← mos.AssessSimilarity(cws, gws)
8        aws ← aws ∪ {(mos, abstractValue)}
9      end for
10     predictedAction ← classifier.Classify(aws)
11     Use(predictedAction)
12     playerAction ← ObtainPlayerAction()
13     classifier.Update(aws, playerAction)
14     ApplyToWorld(playerAction)
15     cws ← current world state
16   end while
17 end while
```

Figure 2: The GDP-Modelling Algorithm.

Given the value of each abstract attribute (similarity measure), the calculation of each corresponding abstract value (similarity value range) is in $\Theta(\log |AbstractV_{AbstractA}|)$; that is, the calculation is bounded by the log of the number of abstract values that the abstract attribute can take on.

Empirical Evaluation

To evaluate the GDP-Modelling algorithm, a set of player choices in a goal-directed setting was required. To this end, a simple, web-based puzzle game was created, titled *Dot Master* (Thue 2005). Hosted on a remote web server, Dot Master was accessible to anyone using a modern web browser; players who navigated to the Dot Master web page were immediately presented with the game, as shown in Figure 3.

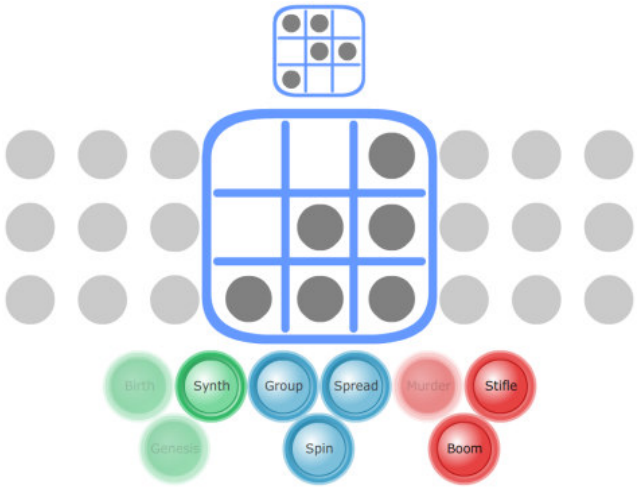
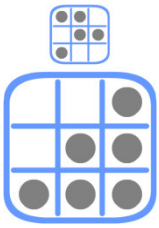


Figure 3: The Experimental Testbed: Dot Master.

Measure of Similarity	Value Ranges
Hamming Distance	$1, \quad 2, \quad [3,5], \quad [6,\infty)$
Count Distance	$(-\infty,-2], \quad -1, \quad 0, \quad 1, \quad [2,\infty)$
Group Distance	$(-\infty,-1], \quad 0, \quad [1,\infty)$

Table 1: Measures of Similarity and Value Ranges.

The goal given to each player was to use the buttons along the bottom to manipulate the central 3 by 3 grid of dots in an attempt to match the configuration of dots given in the small grid at the top of the screen. Nine possible actions existed, though the dynamics of the game resulted in not all actions being available for selection at all times. After every player action choice, the inner 'while' loop of the GDP-Modelling algorithm was run, and a 5-tuple of results (player ID, GDP-selected action, randomly-selected action, player-selected action, abstract world state) was saved to the server. The measures of similarity and accompanying value ranges that were used in this experiment are given in Table 1. Hamming Distance and Count Distance have been discussed previously; Group Distance is a domain-dependent measure referring to the difference in number of groups between the world and goal states, where a group is defined to be either one dot or a set of dots that are 4-adjacent. For both Count Distance and Group Distance, differences were computed as *worldValue* − *goalValue*. Given that there are 4, 5 and 3 abstract values for each of the Hamming, Count, and Group Distance measures respectively, the total size of the abstract world is $4 \times 5 \times 3 = 60$, while the size of the non-abstract world and goal, having 18 attributes (dots) with 2 values each (on or off), is $2^{18} = 262144$. An example of the goal-based state abstraction process is shown in Figure 4; five world dots must be changed to match the goal, and the world has both one more dot and one less group than the goal. Once the values for the similarity measures are computed, they are fitted into the ranges in Table 1.



Abstract Attribute	Value	Abstract Value
Hamming Distance	5	$[3,5]$
Count Distance	1	1
Group Distance	-1	$(-\infty,-1]$

Figure 4: Example Abstraction: on the left are the world and goal states, and on the right is the abstract world state.

To assess the value of the GDP-Modelling algorithm's goal-based abstraction, several popular Machine Learning classifiers were deployed on the Dot Master testbed with two sets of input: approximately 4000 player actions with the set of full (non-abstract) world and goal states in which they were taken, and the same set of actions with the set of corresponding abstract world states. Three of the classifiers used were run using the Weka Data Mining Software, a free tool containing a host of different classifiers (Witten & Frank 2005). These classifiers were: K-Nearest Neighbour (K=3), Decision Stump, and Naive Bayes. A fourth classifier, called

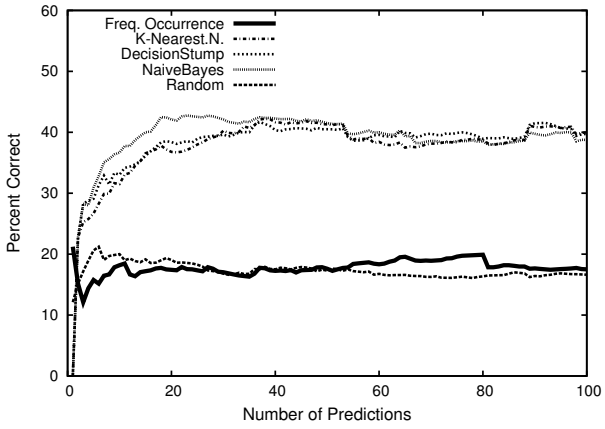


Figure 5: Classifier performance using no abstraction

"Frequent Occurrence", was added directly to the Dot Master testbed, consisting of a simple most-frequent-occurrence classification choice with random initialization; for any input world state, from the actions that can be performed in that state, the one taken from that state most frequently in the past is returned, and if no actions have been observed for a given state, a random prediction is made. Given its direct integration into the testbed, the Frequent Occurrence classifier was implemented to be online and incremental. With every player action, the classifier's model is refined, and a prediction of the next action is generated. Because the classifiers chosen from Weka were implemented as offline methods, simulating an online implementation required re-initializing each classifier after every action, using all previous actions as a training set to predict the next action. As a baseline, a classifier that returns a random prediction from the set of possible actions, called "Random", was also run.

Figure 5 shows the results for the first input set, where the full world and goal states were used. Figure 6 shows the results for the second input set, where abstract world states were used. Each line in the figures represents the performance over time of one of the five classifiers, in terms of the percent of predictions made that were correct, that is, where the predicted action matched the action taken by the player. Each data point represents an average over predictions for a set of players, beginning with 33 players for the first action. Classifier performance is shown for 100 predictions and actions, but many players stopped playing before this point. In light of this, all averages were taken relative to the number of players that were still playing at the timestep in which the action was taken.

Discussion

Comparing Figures 5 and 6, several results are apparent: the Weka-based classifiers, K-Nearest Neighbour, Decision Stump, and Naive Bayes, perform similarly with respect to one another in both graphs, and although all three appear to gain a small boost in performance (between 3% and 5%) when GDP-Modelling is applied, these increases are not statistically significant. In contrast, a large, statistically significant increase in performance is obtained by the Frequent Oc-

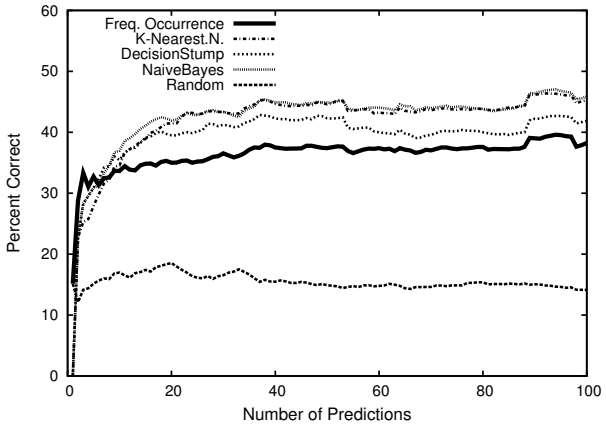


Figure 6: Classifier performance using GDP-Modelling

currence classifier, improving in accuracy almost twofold, and moving from predicting almost completely randomly to being competitive with the more advanced classifiers from Weka. As expected, the Random classifier generally remains between accuracies of 12% and 20%, reflecting random predictions based on a variable number of action choices available in a given state.

When assessing the value of GDP-Modelling, it is important to recall that the Frequent Occurrence classifier is both online and incremental, and benefits from a simple implementation with low computational cost; this makes it particularly well-suited for embedding in commercial games where only a small portion of CPU cycles is available for in-game AI. Additionally, because the Frequent Occurrence classifier stores only the frequency of player actions and predicts the action with highest frequency, its computational and storage requirements are independent of the number of player actions observed. The ability of GDP-Modelling to improve the performance of the Frequent Occurrence classifier to a level of accuracy near those of significantly more costly offline classifiers is encouraging, and given that the process of building and applying the abstraction is also simple and inexpensive, the Frequent Occurrence classifier enabled by GDP-Modelling is an efficient, viable solution to the online prediction of player actions.

Future Work

Although the given results of the study of GDP-Modelling are promising, further research is required. While the related work presented in this paper has significant merit in itself, it would be interesting to extend the methods therein with the idea of Goal-Directed-Player Modelling presented in this work. Given the substantial benefit obtained by the Frequent Occurrence classifier from GDP-Modelling, it would be interesting to devise and test additional simple and inexpensive classifiers. An investigation of the performance of GDP-Modelling-enhanced classifiers in other domains is needed to verify their general ability to predict player actions, and the implementation of a system that uses GDP models to adjust game difficulty dynamically or generate helpful non-player characters remains as future work.

Conclusion

The study of player modelling for digital games is gaining popularity, and the results shown by Yannakakis and Maragoudakis, Charles *et al.*, and Albrecht *et al.* are most promising. In this paper, we have offered insight into the differences in these methods for player modelling, and investigated the subfield of modelling *goal-directed* players. We have proposed a novel enhancement to player modelling that is particularly relevant to the digital role-playing and puzzle game industries, titled Goal-Directed-Player Modelling, in which state abstraction is used with knowledge of a player's goals to improve the accuracy of a simple, incremental, low cost online classifier that learns to predict player actions as gameplay progresses, with an accuracy that is competitive with advanced offline classification techniques.

Acknowledgements

We greatly appreciate the support of the many players of Dot Master, and offer our thanks for the thoughtful suggestions from our reviewers. Special thanks are due to Nicholas D'Autremont, both for hosting Dot Master and assisting with server-side operations.

References

Albrecht, D. W.; Zukerman, I.; and Nicholson, A. 1998. Bayesian models for keyhole plan recognition in an adventure game. In *User Modeling and User-Adapted Interaction 8(1-2), Special Issue on Machine Learning in User Modeling*, 5–47. Kluwer Academic Publishers.

ArenaNet. 2005. Guild Wars. http://www.guildwars.com/.

Charles, D.; Kerr, A.; McNeill, M.; McAlister, M.; Black, M.; Kcklich, J.; Moore, A.; and Stringer, K. 2005. Player-centred game design: Player modelling and adaptive digital games. *Digital Games Research Conf. 2005* 285–298.

Hamming, R. W. 1950. Error-detecting and error-correcting codes. *Bell System Tech. Jrn.* 29(2):147–160.

BioWare Corp. 2003. Star Wars: Knights of the Old Republic. http://www.lucasarts.com/products/swkotor/.

Electronic Benefits Transfer Glossary. 2005. http://www.utexas. edu/lbj/21cp/ebt/glossary.htm.

Gamespot. 2005. Star Wars: Republic Commando, A Review. http://www.gamespot.com/pc/action/ starwarsrepubliccommando/review.html.

Lucas Arts. 2005. Star Wars: Republic Commando. http://www.lucasarts.com/games/swrepubliccommando/.

Newell, A.; Shaw, J.; and Simon, H. A. 1960. Report on a general problem solving program. In *Proceedings of the Int'l Conference on Information Processing*, 256–264.

Thue, D. 2005. Dot Master. http://wasteofspace.ca/~dotmaster/.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical machine learning tools and techniques*. San Fransisco: Morgan Kaufmann, 2nd edition.

Yannakakis, G. N., and Maragoudakis, M. 2005. Player modeling impact on players entertainment in computer games. *Springer-Verlag: Lec. Notes in Comp. Sci.* 3538:74.