Semi-Automated Gameplay Analysis

Finnegan Southey and Robert Holte
Dept. of Computing Science, University of Alberta

Abstract

While presentation aspects like graphics and sound are important to a successful commercial game, it is likewise important that the *gameplay*, the non-presentational behaviour of the game, is engaging to the player. Considerable effort is invested in testing and refining gameplay throughout the development process. We present an overall view of the *gameplay management* problem and, more concretely, our recent research on the *gameplay analysis* part of this task. This consists of an *active learning* methodology, implemented in software tools, for partially automating the analysis of game behaviour in order to augment the abilities of game designers.

Introduction

From its earliest days, computing science has played an increasing role in the realm of games. Traditional games like chess were among the early problems addressed by artificial intelligence (AI) research, while computer graphics and simulation have given rise to a whole new genre of games, now comprising a multi-billion dollar industry. We will refer to this new class as simply *computer games*. We distinguish these from traditional games, which may be played by computers but do not require one.

Compared with traditional games, modern computer games are very complex constructions, involving many parameters for the simulation and AI. The virtual worlds portrayed in many games are three-dimensional and have complicated physics models. Such worlds are effectively continuous spaces, and player controls are often similarly continuous (e.g. analog controllers). Another key difference from traditional games is that there is a much wider variation in scenarios. Chess is always played on the same board, and most traditional games use a widely accepted set of rules with only minor variations. By contrast, computer games may have many different "levels" or "maps", including some developed by the players themselves. There are often a variety of rules and options, and even the objectives and constraints may change in different portions of the game. The number of players often varies too, especially when playing on the Internet, and there may be a mixture of human and computer-controlled players.

This variability poses difficult problems for developers. They must expend considerable effort to ensure that the game is "well-behaved" across the widest possible range of scenarios. Beyond the already difficult task of ensuring that the game runs reliably (e.g. no crashes, graphical glitches, or networking problems), they must ensure that the simulation is plausible and consistent, that artificial opponents and allies behave reasonably, and, most importantly, that the game is enjoyable for the player. We will refer to this range of objectives as the *gameplay* of a game.

Gameplay is a popular term used throughout the industry. Lacking any precise definition, it typically refers to the behaviour of a game (e.g. the rules, difficulty, consistency, fairness, goals, and constraints), rather than the presentation of the game (e.g. graphics, artwork, and sound). Gameplay contributes much to the enjoyability of a game and considerable effort is invested in designing and refining gameplay. This problem is nothing new to game developers. Quality assurance and *playtesting* are fundamental parts of the development process, evaluating gameplay along with other game aspects. For the most part, these evaluations are made by continuously playing the game during development and adjusting accordingly. Toward the end of the development cycle, many human testers are employed to rigorously exercise the game, catching remaining bugs and fine-tuning its behaviour.

When people consider artificial intelligence in the context of computer games, the most common association is with computer-controlled opponents and allies. While this is certainly a key application of AI methods, it is not the only area in which AI can contribute to computer games. Our recent research has been directed toward what we call *semi-automated gameplay analysis*, developing techniques to partially automate the process of evaluating gameplay during development in order to reduce the burden on designers and improve the quality of the final product. To this end, we have developed sampling and machine learning techniques that automatically explore the behaviour of the game and provide intelligible summaries to the designer. We use the term *semi-automatic* to reinforce the notion that such tools can not replace, but can only augment the human designer, helping them to make the complex decisions about what constitutes enjoyable gameplay. This analysis is a key component in the larger task of *gameplay management*, which deals not only with the analysis of gameplay, but the visualization of analysis results and the adjustment of the game's parameters and design.

In the following sections, we will briefly characterize the overall gameplay management task and its role in the development process, and then focus on gameplay analysis, presenting two current research efforts aimed at providing suitable techniques. Details regarding the sampling methods and machine learning techniques used in these two projects will be provided, and we will conclude with some discussion about the wider gameplay management task and the corresponding possibility for future research and improvements to computer games.

Gameplay Management

It is difficult to characterize the gameplay management task precisely. This is largely due to the fact that it inevitably involves some human judgement. "Enjoyability" is essentially impossible to quantify.

An integral but complex part of enjoyability is the gameplay offered. The designer often wants to vary the gameplay throughout the game (e.g. by varying difficulty, the rules, or the objectives). It is naive to think that this problem can be addressed without human intervention. However, computational tools can reduce the human burden, allow more extensive testing, and search in systematic ways that may be tedious or unintuitive to a human. Where possible, these tools should be applicable to many games. This re-usability is important since industry development cycles have little time to spare to develop such tools every time. The tools must be easy to use. Many designers and testers are not programmers so it is not reasonable to expect them to write or understand complex scripts to guide the process.

Some of the basic aspects of gameplay management include:

dimensions: What aspects of gameplay is the designer interested in? Some interesting aspects include the time to complete part of the game, the "cost" (e.g. health, game money, resources) involved, and the probability of succeeding. The amount variance in any one of these is also of interest (highly variable outcomes may indicate an inconsistency or bug in gameplay). All of these are aspects which the designer may wish to analyze or constrain.

goals: What goals does the designer have in mind? They may wish to limit gameplay along some dimensions (e.g. the game should usually take less than 8 hours, or the player should be able to win by only a narrow margin). They may wish to prioritize goals.

analysis: Once dimensions have been identified, how may they be evaluated? To what extent can they be quantified and estimated? This is a hard problem all by itself and forms the focus of our current research efforts and this paper.

adjustment: Given the dimensions and the ability to analyze parts of the game, what may be done to achieve the goals set by designers? Currently, most of the adjustment is done by hand. It may be possible to automate parts of the process, with the designer refining goals and the software working to achieve them. This is a key part of the problem, and very difficult, but one which we will not consider deeply at present. Before one can adjust, one must be able to analyze.

visualization: How can goals and the results of analysis be expressed by, and presented to, the designer? While all aspects of gameplay management will be heavily game dependent, this aspect may be the most so. The method for expressing goals is a good AI problem, but other parts of the visualization seem more suited to user interface design and, ultimately, the game developers will best understand how to interact with their game. We concern ourselves with visualization only to the extent necessary to demonstrate our ideas, while recognizing that this is a significant part of the problem.

Gameplay Analysis

As noted above, our current research efforts are focused on the task of *gameplay analysis.* The dimensions to be evaluated will be game dependent, but the analysis methods are one part that may be transferred from one project to another. Our methodology embraces three basic tools used in many branches of AI research: *abstraction*, *sampling*, and *machine learning*.

Abstraction

Any game engine has far more variables that we can reasonably expect to interpret. When evaluating the game engine, we are interested in the dimensions we are analyzing (e.g. time elapsed, resources consumed, success/failure, etc.) but we may also need to set the conditions we are interested in sampling (e.g. position, speed, initial resources, etc.). This is a large and complex body of information, and will need to be reduced and/or transformed in order to be useful. Therefore, we seek to abstract the raw game engine state into some smaller set of *features* that we can more easily interpret and control. While many interesting features will be game dependent and designed by hand, the AI literature offers considerable research on how to automate abstraction by automatically discovering important features of the data, either by selecting important variables from amongst many (*feature selection*) or by mapping the high-dimensional game state to some lower-dimensional set of features (*dimensionality reduction*). We have not explored automated abstraction at present, but it is likely to be a useful tool. All of our present work uses hand-coded abstractions to provide our features.

Sampling

A game's internal variables, combined with the player's strategies and controls, form a very large space. Games often include a random component, so that, even given identical initial conditions and player actions, outcomes may differ on repeated trials. Analyzing the entire space is clearly infeasible. While running the game simulation without graphics or speed limits allows for much faster analysis, we still need to confine our analysis to only small parts of the game at a time,

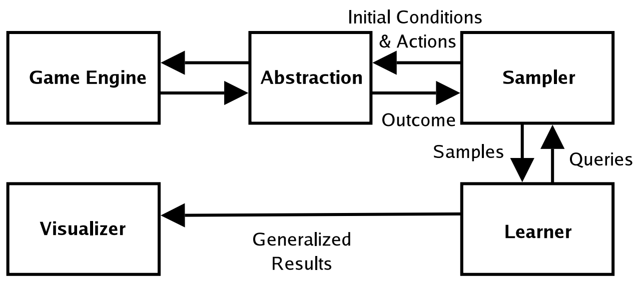


Figure 1: Architecture for sampling/learning analyzer. The "Sampler" and "Learner" components are not game-dependent.

and even then, can only explore part of the search space. We must approximate our evaluations using *sampling*.

Machine Learning

The raw data from sampling is too large and too complicated to understand readily. *Machine learning* techniques can generalize the results of the data, constructing an abstract *model* of the game's behaviour and allowing us to visualize overall results rather than specific instances. It also allows us to make predictions about previously unsampled points in the space and test them, or to suggest new sample points to confirm our general conclusions or extend them to new regions. This use of the learned model to direct sampling (also known as *queries*) in order to improve the model is known as *active learning*.

Figure 1 shows the interaction between the components described here, the game engine, and the visualizer.

Current Research

Soccer Viz

The Soccer Viz project uses active learning to analyze scenarios in the Electronic Arts (EA) soccer game, FIFA'99[1] (e.g. a single shooter shooting at the goal). In this context, a sample point consists of an initial state fed into the game engine and a subsequent sequence of actions. For example, the initial state might have the shooter at position (x_s, y_s) on the field and the goalie at position (x_g, y_g). The shooter then kicks the ball toward point (x_k, y_k)[2]. For this relatively simple scenario, we already have a six-dimensional space. Each sample produces an outcome, scoring or non-scoring, which is recorded. Because FIFA'99 includes some randomness, each point is sampled 10 times and all of the outcomes recorded.

[1]EA has made the source code available to us for this purpose and the project is in the process of being extended to work on FIFA 2004.

[2]In this description we use Cartesian coordinates, but the abstraction used to control and interpret FIFA'99 is in terms of polar coordinates (e.g. the distance and angle between shooter and goalie). Polar coordinates better express the relative positions required to score and hence result in better learned rules. This is part of the hand-crafted abstraction made for FIFA'99.

First we will describe the algorithm in general terms before describing the specific components. The process is iterative. On the first step, a fixed number of points are uniformly randomly selected from the space and sampled by running them in the game engine. We then learn a set of rules from the sampled points. Next, we use an active learning method to decide which samples to collect next, collect the sample and repeat to process. This can continue until we have exhausted available time and resources, or until our learned rules cease to change.

Rule Learners *Rule learning* methods attempt to construct a set of *rules* that predict an output based on inputs. For example, in the soccer domain, such a rule might be: IF the shooter is within 5 metres of the goalie AND the angle between the shooter and goalie is between $30°$ and $40°$ AND the goalie is within 1 metre of the centre of the goal THEN the probability of scoring is 70%.

From a given set of samples, a set of rules will be learned which describe the general behaviour of the game in this specific scenario. These rules are already easier to understand than the raw six dimensional data-points, but visualization tools can make them clearer still, and are better suited to designers with little or no programming experience. We will shortly give an example of such a visualization tool.

We have used two "off-the-shelf" rule learning methods in our framework. The first is C4.5 (Quinlan 1994), a well-known decision tree learning algorithm that is capable of rendering the learned tree into a set of rules suitable for our purposes. The second is SLIPPER (Cohen & Singer 1999), a more recent method that learns rules directly and is based on an earlier rule learner, RIPPER (Cohen 1995), and the AdaBoost learning algorithm (Freund & Schapire 1997). We refer the interested reader to the relevant publications for details of these algorithms. For our purposes here, all that is important is to understand that a set of rules has been learned using readily available software, and that different learning algorithms can be used within our analysis tool so long as they produce reasonable rules.

Samplers The part of our research that has received the most attention is the sampling. The literature on active learning offers many algorithms for deciding where to sample next, chiefly differing in the heuristics they use to identify "interesting" regions. We have implemented several samplers which fall into three broad categories.

1. Uncertainty Sampling (Lewis & Gale 1994): Given a binary outcome (e.g. scoring or not scoring) and rules that predict a probability for the outcome, it's clear that the regions of greatest uncertainty (containing a small number of conflicting samples, or none at all) will assign a probability of 0.5 to both outcomes. Uncertainty sampling examines the learned rules to determine the regions where such predictions are made and recommends additional samples there. This method has some significant problems. If the probability of the outcome for that region truly is 0.5, uncertainty sampling will continue to allocate samples there even though it has actually discovered the true probability and should sample elsewhere.

2. Query By Committee (QBC) (Seung, Opper, & Sompolinsky 1992): This is a collection of techniques with a common theme. Instead of learning a single predictor, one collects several different predictors (the *committee*) and trains them all using the available data. The results are then examined to determine those regions where the committee members disagree the most and new samples are selected from those regions. Clearly, there are many possible QBC algorithms, but we have chosen two existing algorithms to implement.

   (a) Query by Boosting (Abe & Mamitsuka 1998): We mentioned boosting (Freund & Schapire 1997) in the previous section in the context of rule learners but it is applied somewhat differently. The basic idea is to associate a *weight* with each sample, indicating the relative importance of correctly predicting the outcome of that sample. A predictor is then trained with the weighted sample set. For those samples where this new predictor makes a mistake, the weights are increased. For those where it is correct, the weights are decreased. Then a new predictor is trained using the new weighted sample set, and the process repeats. Boosting is useful because the set of predictors can be combined to form a higher quality predictor. However, in the QBC context, we use the set of predictors as a committee. Even if the same algorithm is used to train each predictor, the differences in the weights allow the predictors to differ as well.

   (b) Query by Bagging (Abe & Mamitsuka 1998): Bagging (Breiman 1996) is a method for obtaining different predictors, suitable for a committee, even when using the same algorithm to learn each predictor. The idea (similar to boosting) is to take the available samples and generate a new sample set by selecting them randomly and with replacement. This means that duplicates of some samples may arise in the new set and that some of the samples may be missing. By generating a new set in order to train each predictor, the predictors can now be different and are suitable for forming a committee.

3. Region-Based Sampling: Whereas the preceding sampling techniques are effectively “off-the-shelf”, the three techniques presented here were developed by ourselves, based on our intuitions about sampling and its relationship to the regions identified by the rules.

   (a) Minimum Density Sampling: Each rule learned from the data corresponds to a volume in the sample space containing the samples which form the prediction of that rule (i.e. the probability). A very simple approach to the active learning problem is simply to ensure that we have sufficient evidence in all learned regions. We therefore consider the *density* of sample points in the rule’s region (i.e. the number of samples / region volume) and add points sampled uniformly within the region until the density is at some minimum level, specified by a parameter.

   (b) Boundary Extension: Each rule bounds a region for which it makes a prediction. The strategy here is to sample points just beyond the boundary of a rule to see if we can extend its boundaries. Figure 2 shows a rule’s region (which is predicting a positive outcome) and then the same region with the proposed boundary extension samples. The size of the extension is specified by a parameter.

   (c) Counter-Example Sampling: Suppose a region contains a set of sample points where the overwhelming majority of outcomes are the same, but a very few counter-examples exist, as in Figure 3. Counter-example sampling attempts to determine if these oddities in the sample are just a matter a chance or represent some small opposing region contained within the larger region which should be covered by its own rule. This is done by sampling points in the immediate neighbourhood of the counter-example.

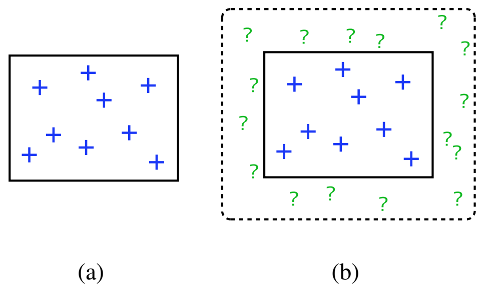


Figure 2: Boundary-Extension Sampling: (a) Original region (b) Boundary-Extension samples

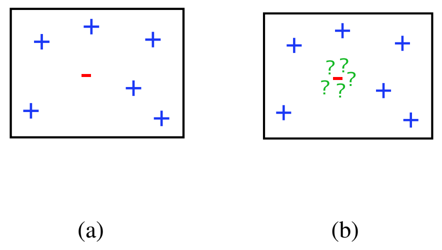
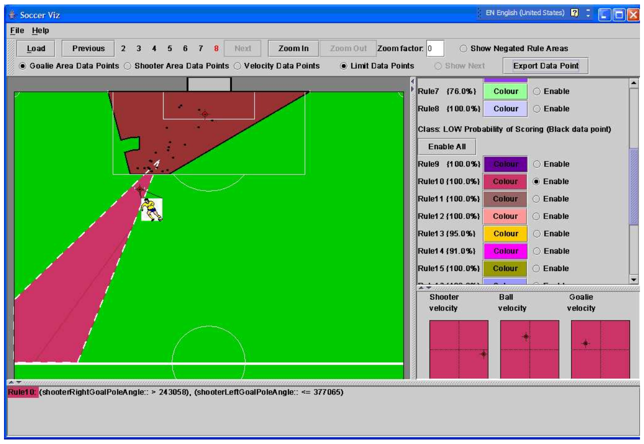


Figure 3: Counter-Example Sampling: (a) Original region (b) Counter-Example samples

While the details of the various samplers are of research interest, happily they are transparent to the designer using the tool. All of the above methods have been implemented but we do not yet have a full comparative study to determine which combination of rule learners and sampling techniques is best. However, our experience with the C4.5 learner in combination with our own Region-Based sampling techniques shows that they produce sensible rules which we can examine in the real game engine to confirm our understanding of what is going on. We envision that the usage of this tool will consist to a large extent of the designer looking for



Rule7 (76.0%) Colour Enable
Rule8 (100.0%) Colour Enable
Class: LOW Probability of Scoring (Black data point)
Enable All
Rule9 (100.0%) Colour Enable
Rule10 (100.0%) Colour [✓ Enable]
Rule11 (100.0%) Colour Enable
Rule12 (100.0%) Colour Enable
Rule13 (95.0%) Colour Enable
Rule14 (91.0%) Colour Enable
Rule15 (100.0%) Colour Enable

Rule10: (shooterRightGoalPoleAngle:: > 243058), (shooterLeftGoalPoleAngle:: <= 377065)

Figure 4: Soccer Viz display of a learned rule which predicts a low probability of scoring. The area outlined near the goal shows goalie positions covered by the rule. The triangle extending out into the field shows shooter positions covered by the rule. Black dots show sampled shooter positions where no goal was scored. One particular shooter position sample and the corresponding goalie position are shown.

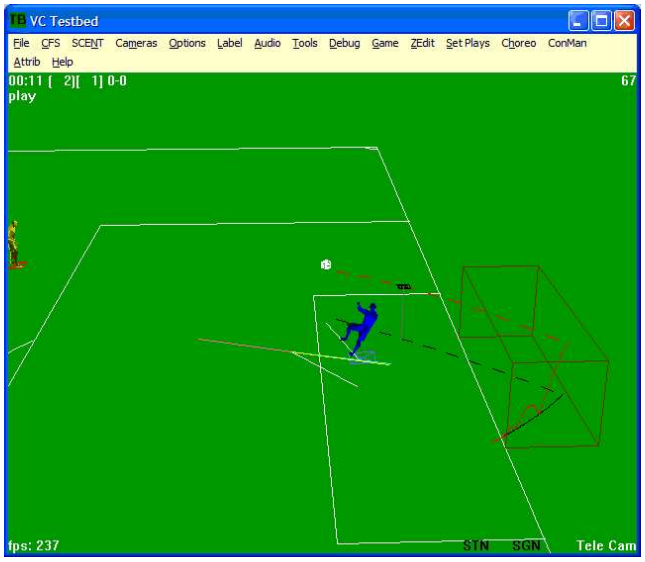

VC Testbed
File CFS SCENT Cameras Options Label Audio Tools Debug Game ZEdit Set Plays Choreo ConMan
Attrib Help
00:11 [2][1] 0-0
play
67
fps: 237
STN
SGN
Tele Cam

Figure 5: FIFA'99 game engine running a simulation of the shooter-goalie sample highlighted in Figure 4. The simulation is running in the real game engine but the graphics have been simplified for speed. Soccer Viz allows the user to directly export samples to the simulation so the designer can observe the behaviour of the real game.

peculiar rules and the running the game engine to determine whether the behaviour is reasonable. Changes to the gameplay can be made, the samples re-examined, and the new rules checked again by the designer in an iterative process. Future sampling engines should also allow the designer to direct sampling toward regions of interest.

Visualization While we are not primarily interested in the visualization aspect, which can probably be better designed and managed by game developers who have considerable graphics, user-interface, and gameplay design experience, we have developed a tool to demonstrates the results of the learning process. The Soccer Viz visualization tool is shown in Figure 4. This tool displays the rules generated by the learning and allows the user to examine the samples supporting the rule. These samples can also be directly visualized in the FIFA'99 engine (see Figure 5). This research uses the real game engine. Visualization is clearly specialized for soccer, as is the state abstraction, but the analysis tool (learner and sampling engine) is generic and should be transferable to other games. This work was presented to developers at two Electronic Arts sports game studios (Vancouver, BC and Orlando, FL) earlier this year.

RPG Assessor

Our second project, still under early development, attempts to analyze gameplay in role-playing games (RPGs) like BioWare's popular "Neverwinter Nights" title. In these games, the type of scenario is typically fixed by the designer (e.g. a fight in a room with three "monsters" and the player's character, who carries a set of different weapons). This research explores a somewhat different avenue to the soccer research where we consider richer player strategies. These strategies take the form of *policies* which specify the player's reaction to the different states encountered during the action (e.g. if all three monsters are alive, the player will attack monster A, whereas if only monsters B and C are alive, the player will choose to attack monster C, etc). Such policies are not the fixed sequences of actions used in the soccer research, but entire plans of action covering multiple possible outcomes to each stage of the battle.

As with soccer, the detailed game state (i.e. the health of the player and all her opponents, the weapons carried by each, etc.) forms a large space which we must abstract (e.g. monster A is simply dead or alive, etc). As before, the abstraction is constructed by hand, but in order to realize our policies, we must discover which abstract states are reachable from a given abstract state (e.g. in the absence of the dubious benefits of necromancy, the abstract state "Monster A is alive" is not reachable from "Monster A is dead", but the reverse transition is quite sensible). Rather than having the designer specify this large set of *abstract state transitions*, we have developed sampling mechanisms that automatically discover the transitions. This is done by starting at the initial state and systematically attempting combinations of actions until we are confident that we have observed most, if not all, of the possible transitions (extremely low probability transitions are unlikely to have profound effects on gameplay).

Given the state transitions, we can enumerate all possible player policies and sample them to determine the outcomes for each. Unfortunately, space does not allow us to to explain this process in greater detail, but the automatic transition discovery has already been developed and is currently being tested. We hope to move onto the policy sampling aspect in the near future. The products of this research will allow the designer to analyze the scenarios in their game and discover potentially undesirable behaviours (e.g. very few player strategies result in success, some particular strategy renders the encounter trivial, etc).

The Future of Gameplay Management

Our gameplay management framework is based on our discussions with commercial game developers. We have connections to several companies, including Electronic Arts (the world's largest game developer and publisher making sports and many other games), BioWare (top-ranked role-playing games), and Relic Entertainment (award-winning real-time strategy games). In this paper we have presented only our current research which is focused on analysis. We believe this is necessarily the first step because the tasks of visualizing and adjusting gameplay effectively will be tied to obtaining the right data for the designer and future tools, and summarizing it in useful ways.

Looking forward, there are two fundamental ways these ideas could be applied, which we will call *offline* and *online.* Offline applications impact only the development of the game. Analysis and adjustments are performed during the development, but the final product does not use these methods. More ambitiously, online methods would take effect while playing the final game. Understandably, developers are concerned about online methods because they can impact the player experience at a point when they longer have control. We believe that online methods will ultimately be important but will need to be well-understood and predictable.

We will now identify some specific tasks, some of which are handled entirely by hand at present and some of which are rarely, if ever, used.

Offline Methods

- identifying *sweet spots*, specific situations or player strategies that render the game too easy
- identifying *hard spots*, situations that are too difficult and where only very specific player strategies are effective
- *balancing* opposing forces to ensure a fair contest
- hand-tuning simulation parameters (e.g. speed, damage, etc.) for difficulty level (e.g. Easy, Normal, Hard settings)
- hand-tuning opponent/ally AI for varying difficulty level
- retuning after feature changes or bugfixes
- automatically tuning parameters
- automatically learning strategies for opponent/ally AI

Online

- dynamically adjusting difficulty based on recent player performance
- analyzing situations to make opponent/ally AI decisions
- providing feedback on opponent AI vs. player performance for learning
- providing commentary, feedback, or advice to the player during the game

Potential Benefits

- augment human testing with computational resources
- designer-driven testing strategies allowing more direct control than by managing human testers
- regression suites (collections of goals and analysis strategies that may be repeatedly tested as the game changes)

Conclusions

Many of the above tasks represent ambitious goals but all of our contacts with the industry indicate a strong, widespread inclination to pursue them. They will undoubtedly form a substantial portion of our future research. Our current work on gameplay analysis shows that the task offers many interesting AI research problems and our initial results are promising. Once we have good analyzers in place, we can look forward to the adjustment aspect of the task, tuning game parameters automatically to meet goals. We believe this work will be rewarding to both the AI research community and game developers, and offers another avenue for AI research to interface with computer games.

Acknowledgements

Much of this research is due to Gang Xiao, Mark Trommelen, and Jonathan Newton. Thanks are also owed to John Buchanan and Electronic Arts. Funding agencies included IRIS, Alberta Ingenuity through the AICML, and Electronic Arts.

References

Abe, N., and Mamitsuka, H. 1998. Query learning strategies using boosting and bagging. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98).*

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Cohen, W. W., and Singer, Y. 1999. A Simple, Fast, and Effective Rule Learner. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99).*

Cohen, W. W. 1995. Fast Effective Rule Induction. In Prieditis, A., and Russell, S., eds., *Proceedings of the 12th International Conference on Machine Learning (ICML-95)*, 115–123. Tahoe City, CA: Morgan Kaufmann.

Freund, Y., and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* 55(1):119–139.

Lewis, D. D., and Gale, W. A. 1994. A sequential algorithm for training text classifiers. In Croft, W. B., and van Rijsbergen, C. J., eds., *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*, 3–12. Dublin, IE: Springer Verlag, Heidelberg, DE.

Quinlan, J. R. 1994. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann.

Seung, H. S.; Opper, M.; and Sompolinsky, H. 1992. Query By Committee. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, 287–294.