Proceedings of the First
Artificial Intelligence and Interactive
Digital Entertainment Conference
(AIIDE-2005)

Semi-Automated Gameplay Analysis by Machine Learning

**Finnegan Southey, Gang Xiao,
Robert C. Holte, Mark Trommelen**
University of Alberta

John Buchanan
Electronic Arts

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 largely automating the analysis of game behaviour in order to augment the abilities of game designers. The SAGA-ML (semi-automated gameplay analysis by machine learning) system is demonstrated in a real commercial context, Electronic Arts' FIFA'99 Soccer title, where it has identified exploitable weaknesses in the game that allow easy scoring by players.

Introduction

Modern computer games are very complex constructions, sophisticated software systems involving many parameters for the graphics, simulation, and artificial intelligence (AI). The virtual worlds portrayed in many games are three-dimensional, have rich physics models, and may feature dozens or even hundreds of game agents, each of which may have a wide range of properties. Computer games may have many different levels or maps and there are often a variety of objectives, constraints, rules and options in different sections 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 properties 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. 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 *playtesters* are employed to rigorously exercise the game, catching remaining bugs and fine-tuning its behaviour.

In the context of computer games, AI is usually associated with computer-controlled opponents and allies. While this is certainly a key application, 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 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 cannot replace the human designer but can only assist, helping them make 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 also with 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 recent research efforts applied to Electronic Arts' FIFA Soccer title. While discussed in this specific context, the approach is general and treats the game engine as a black box. We describe the overall methodology and our current implementation, the reusable part of which we call SAGA-ML (semi-automated gameplay analysis by machine learning). Our chief concern in this paper is to describe the over-

all approach rather than the details of the underlying machine learning. We address practical issues relevant to industry developers who may be interested in adopting the methodology. We conclude by noting significant industry interest in our work, as well as the possibilities 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. One important aspect of good gameplay is appropriate difficulty. A game that is too hard is frustrating, while too little challenge can be boring. In multi-player games, it is important that the game be fair, offering no player an intrinsic advantage. Internal consistency in the game world is also important. Even in fictional worlds, players expect some sort of logic to apply.

The designer must shape and regulate the gameplay throughout the game (e.g., by controlling difficulty and varying the rules and objectives). It would be naive to think that this 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 spare time to redevelop such tools for each project. The tools must also be easy to use. Many designers are not programmers, so it is not reasonable to expect them to write or understand complex scripts to guide the process.

We will now outline the *gameplay management* task. By decomposing this process more formally, we can address the components one by one in our research.

gameplay metrics: The aspects of gameplay of interest to the designer. These are the game aspects we attempt to measure. Typical *metrics* 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 of variance in any one of these is also of interest (highly variable outcomes may indicate an inconsistency, a bug, or a poorly chosen parameter). Typically, we want to evaluate a metric given some initial *game state* and a set of *player actions* taken from that initial state.

goals: The targets the designer wishes to achieve. Having identified a metric, the designer must decide what values are acceptable. For example, the designer's goal might be that the game should take less than 8 hours, or that the player can win with no more than 40% of health remaining. Techniques that help designers formally express goals would allow software to reason about goals and how to achieve them. Much of the time, however, the designer is balancing an array of loosely defined goals.

analysis: Once metrics have been identified, they must be measured. Analysis is the problem of estimating values for a set of metrics by testing the game's behaviour. It is the focus of our current research efforts and this paper.

adjustment: Given the metrics, the ability to analyze, and a set of goals, adjustments must be made to the game to achieve the desired effect. Currently, most adjustment is done by hand. It may be possible to automate parts of the process, with the designer setting and refining goals, and the software working to achieve them. This is a key part of the overall problem and is very difficult, but we will not consider it deeply at present. Before one can adjust effectively, one must be able to analyze.

visualization: The results of analysis must be presented clearly to the designer. Methods for expressing goals and machine learning techniques for summarizing data are areas where AI has much to offer, but a substantial part of visualization consists of user interface design. We have worked extensively on a visualization tool for our current case study and consider this a very important part of the problem, but one in which our expertise can have only limited impact. Ultimately, game developers will best understand how to interact with their game.

Gameplay Analysis

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

While our framework is intended to be independent of game, and even of genre, we illustrate the approach with our implementation for Electronic Arts' FIFA'99 Soccer game.[1] In particular, we have developed a general purpose analysis tool (SAGA-ML) and a FIFA-specific visualization tool (SoccerViz). The aim is to identify "sweet spots" (manoeuvres that can be used to repeatedly score goals with unreasonable ease) and "hard spots" (situations where it is too difficult to score) in various FIFA scenarios. In this case, the metric of interest is the probability of scoring.

Electronic Arts' FIFA'99 Soccer

We have used SAGA-ML to analyze scenarios in the Electronic Arts (EA) soccer game, FIFA'99. Games are generally too complex to be analyzed in their entirety. It makes sense to analyze certain specific scenarios, especially when the game itself distinguishes between scenarios (e.g., different levels, minigames, and control modes). We have explored a small set of scenarios in FIFA but here we will only consider the *corner kick* and the *shooter-goalie* scenarios.

In the corner kick scenario, the ball has rolled across the endline and was most recently touched by the defenders. It must be kicked back in from the corner by a member of the offense. This is a key opportunity for the player's computer controlled teammates to score. Figure 1 shows a screenshot of this scenario in the FIFA engine. The player, shown at the bottom right, must pick a spot for the ball to land when they

[1] Electronic Arts provided source code for their FIFA'99 AI testbed, including gameplay, physics, and AI, but limited graphics.

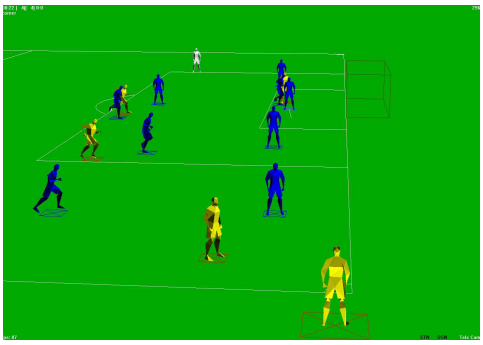

Figure 1: The corner kick scenario in FIFA'99.

Figure 2: The shooter-goalie scenario in FIFA'99.

kick. The success or failure of the player's teammates in scoring depends on where the ball lands. The specific metric we consider is a single number, the probability of scoring given that a single teammate touches the ball after the player kicks it (otherwise the ball might continue in play for some time before scoring). The initial game state is trivial in this case; we only require that we be in the corner kick scenario. The player action is the target for the ball, (x_k, y_k), giving us a two-dimensional space to search.

In the shooter-goalie scenario, shown in Figure 2, the player controls a shooter placed somewhere near the goal and shooting into it, with the goalie placed to defend the goal. All other players are absent from the scenario. This tests the ability of the player to shoot and the goalie to block shots on goal, a critical part of the game. The metric is a single number, the probability of scoring from the player's kick. More formally, the initial game state of the scenario has the shooter at position (x_s, y_s) on the field and the goalie at position (x_g, y_g). The player action is that the shooter kicks the ball toward point (x_k, y_k). This means we must search a six-dimensional space for this scenario.

Overview of SAGA-ML

The overall architecture of the approach is shown in Figure 3. The *game engine* is treated as a black box and SAGA-ML interacts with it through an *abstraction* layer. This layer is game-specific and translates game-specific data and function calls to an abstract state format used by SAGA-ML. The *sampler* component uses the abstraction layer to evaluate situations by running the game with an initial state and

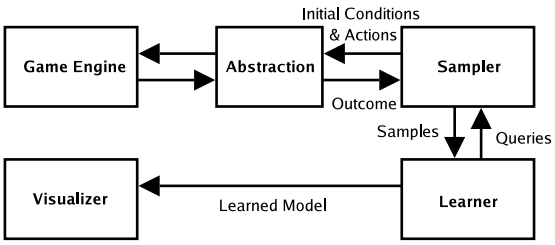

Figure 3: Architecture for the sampling/learning analyzer. The "Sampler" and "Learner" components are not game-dependent.

Algorithm 1 SAGA-ML Algorithm

1. Collect initial uniform random sample, S.
2. Learn model M from samples in S.
3. If the M is unchanged from the previous model, or we are out of time, pass M to to the visualizer and exit.
4. Else, examine model M to determine where new samples should be placed (active learning).
5. Collect additional samples and add them to S.
6. Goto 2.

a sequence of actions, and then observing the outcome. The *learner* uses the data gathered by the sampler to construct a concise *model* (or *summary*) of the game's behaviour. The learner may then request more samples to refine its model. The model is passed to the game-specific *visualizer* for the designer to evaluate. The algorithm is summarized in Algorithm 1. We will now discuss each of these components in greater detail using our FIFA scenarios to illustrate.

Abstraction

Games have far more variables than we can reasonably expect to interpret. When evaluating the game, we are interested in the metrics we are analyzing (e.g., goals scored, time elapsed, etc.) but we also need to set up the initial game state (e.g., position, speed, resources, etc.) and player actions (e.g., move, shoot, buy) we want to sample. The abstraction layer restricts and transforms the raw *game variables* to form the corresponding *game state*, *player actions*, and *metrics* used by SAGA-ML. These are three vectors of numbers exchanged between SAGA-ML and the abstraction layer to control and observe the game.[2]

SAGA-ML uses machine learning to summarize and generalize data from the game engine. The raw game variables in an engine must serve several purposes. Aside from deciding gameplay issues, the engine must also use those variables to render the graphics, compute physics simulations, etc. Different representations are often well-suited to one of these tasks but not to another. The developer must weigh the tradeoffs involved and choose the representation offering the best overall performance. In many cases, the game's

[2]The TIELT framework offers a standardized abstraction layer for machine learning in games, which we are currently investigating (see http://nrlsat.ittid.com).

raw variables may not be directly suited to gameplay decisions, thus requiring substantial code to make those decisions. Simply handing those variables to the SAGA-ML system and expecting it to learn a sensible model may cause problems. Therefore, the abstraction should provide the features that are really used to make the decisions. Developers of the gameplay code will necessarily be aware of these features and it is likely that the gameplay code can be reused to export suitable features through the abstraction layer.

For example, FIFA stores player positions as x-y coordinates on the field. But the actual outcome of a shot on goal is determined by the angle between shooter and goalie, the distance between them, and the relationship of the shooter to the goalposts. If the learner only sees the x-y coordinates, it may not be able to deduce the underlying rules used by the game engine or it may learn an overly complex set of rules. Learning is most likely to be successful if you provide appropriate features. It is also easier to develop visualizations if the features allow for readily comprehensible models.

In our research capacity, outside the game's real development, the construction of the abstraction layer involved a lot of exploration of the codebase. This burden would be much less for developers actively working on the game in normal development. Indeed, it may simply be an extension of existing gameplay code, or of a testing harness. Furthermore, the usual benefits of adding any kind of testing code will apply, encouraging developers to think about how their code works and design clean interfaces.

Sampling

A game's internal variables, combined with the player's strategies and controls, form a very large space. Even within a single scenario in the game, with an abstraction already applied, the space will often be impossible to explore completely. Furthermore, games often include a random component, so that, even given identical initial game states and player actions, outcomes may differ on repeated trials. Exhaustive analysis is clearly infeasible. We must approximate our evaluations using *sampling*.

Systematic sampling (e.g., picking points at regular intervals on a soccer field) may fail to catch some anomaly because corresponding regularities may occur in the logic controlling the game (e.g., if you sample at 4 pixel intervals across the width of the soccer field, you will miss a bug that occurs only with odd numbered pixel locations). Uniform random sampling is a better choice and has attractive statistical properties. With such large spaces, however, we may need to sample more intelligently, but we will return to this issue later when we discuss *active learning*.

One might guide the sampler toward regions that are likely to occur. This requires some initial model of the player and game engine, and can automatically rule out very unlikely situations. While this may have advantages, it must be applied very carefully because it may prevent us from finding problems if our initial model is inaccurate. Some minimal amount of uniform random sampling should be kept to guard against this.

For now, we consider only uniform random sampling. In the corner kick scenario, we would simply pick random positions within the range of the kicker. The more samples we take, the better coverage we will achieve. One thing that assists us in this endeavour is the fallibility of human control. If it turns out that there is a single pixel location that always results in a goal during the corner kick, we probably don't need to worry because a human player is unlikely to be able to reliably hit this spot, even assuming they ever find it. Our sampling and the entire gameplay analysis, including learning, can afford to take a coarse-grained approach. In FIFA, we need only identify regions of high scoring probability large enough that the player can hit them repeatedly.

Realistically, sampling will always depend on available resources. As scenarios grow more complex, decisions about how many samples are "enough" will quickly be replaced by the question of how much time and computation we can afford for analysis. Making the best use of resources is the goal of active learning (see Active Learning below).

Machine Learning

The raw data from sampling is too large and too complicated to understand readily. *Machine learning* techniques can generalize the sample data, constructing an abstract *model* of the game's behaviour that takes any game state and action features as inputs and outputs a prediction for the metric of interest. This allows us to visualize overall results rather than specific data points. There are two key motivations behind our use of machine learning. The first, a human interface issue, is to provide more useful feedback to the designer. The second, a computational issue, is to use the model to decide where new samples should be taken.

Addressing the human interface issue, the model can be examined to see if it matches up with how we believe the game is supposed to behave. If we believe we built the game such that "no shooter can score on the goalie from further than 6 metres", then we can look at the learned model to see if it agrees with that statement. Similarly, we may examine the model and find general rules that we didn't anticipate or intend. A designer can then decide whether these gameplay properties are acceptable or not.

In interesting scenarios, our models are likely to be fairly complex, so looking at the model directly may be difficult. We can use visualization tools to express the model in a way more natural to the designer. If embedded in the game engine or in design and support tools, the designer has a familiar framework in which to understand what the model is saying. We will discuss the visualization question briefly and describe our own visualizer in a later section.

Turning now to the computational issue, the model will generalize to make predictions over the entire space, even in regions we have not sampled. The quality of this prediction will vary, depending on the quality of the learner, the complexity of the space, and the number of samples available. Using predictions for unsampled regions, we can decide where it would be most useful to sample next in order to learn the most about the space, thus making the best possible use of our computational resources. This is *active learning* and is discussed in its own section below.

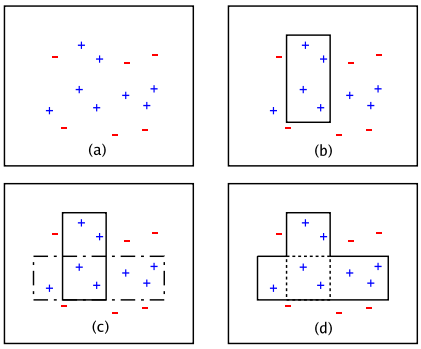

Figure 4: (a) Positive and negative samples in a 2-D space. (b) A single, positive rule learned from samples. (c) A second, overlapping positive rule. (d) Overlapping rules joined into a single positive region (dashed lines are internal boundaries from the original rules).

Rule Learners SAGA-ML uses *rule learners* for the learning component. Rule learners attempt to construct a set of *rules* that predict an output based on inputs. For example, in the shooter-goalie scenario, such a rule might be: IF the shooter is within 5 metres of the goalie AND the angle between shooter and goalie is between $30°$ and $40°$ AND the goalie is within 1 metre of the goal's centre THEN the probability of scoring is greater than 70%. The conditioning statements in these rules are expressed in terms of the features provided by the abstraction layer. From a given set of samples, a set of rules will be learned that describes the behaviour of the game. Such rules are easier to understand than the raw, multidimensional data points but visualization tools can make them clearer still and are better suited to designers with little or no programming experience.

If we think about the rules in a two-dimensional space, such as the corner kick scenario, then the rules describe rectangles where our prediction is positive (true) or negative (false) (this is just an example, predictions could be more complex, e.g., involving probabilities). Figure 4 (a) shows the results, positive and negative, of a set of samples. Figure 4 (b) shows a single, positive rule learned from these samples represented as a rectangle (it is easy to see how an IF-THEN style rule about x-y coordinates can be drawn as a rectangle). Figure 4 (c) shows a second, overlapping positive rule learned from the same data. If we have several rules that overlap, but all agree in their prediction, then we can merge them together to form what we call a *region* (as shown in Figure 4 (d)). This idea extends to higher-dimensional data, where the regions are composed of hyper-rectangles.

We have tried two "off-the-shelf" rule learning methods. The first is C4.5 (Quinlan 1994), a well-known decision tree learning algorithm capable of producing a suitable set of rules. The second is SLIPPER (Cohen & Singer 1999), a more recent method that learns rules directly. We refer the interested reader to the relevant publications for details. It is only important to understand that we can learn a set of rules with readily available software and that SAGA-ML will use any rule learner. While SAGA-ML uses rule learners, the overall methodology extends to other learning algorithms. We have not explored alternatives, but possibilities include *support vector machines*, *Bayesian networks, neural networks*, *nearest-neighbour* methods, and *clustering*.

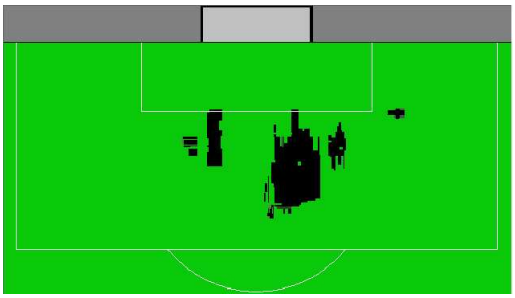

Figure 5: Positive rules learned for corner kick scenario.

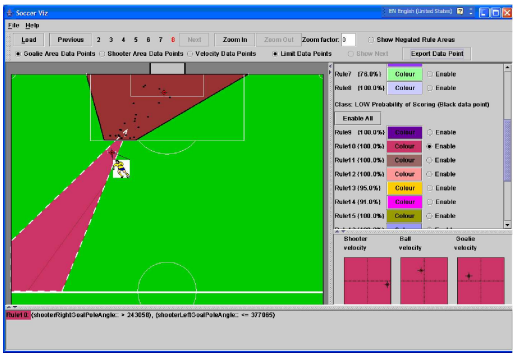

Figure 6: SoccerViz display of a rule predicting a low probability of scoring. Shading near the goal shows goalie positions covered by the rule. The triangle in the field shows shooter positions covered by the rule. Black dots show sampled goalie positions where no goal was scored. One particular shooter/goalie position sample is shown. This sample was exported to the game engine to obtain Figure 2.

Figure 5 shows positive rules learned for the corner kick scenario (the kicker is placed in the top right corner of the field). Note the large shaded region centred on the right goalpost. By kicking the ball into this area, the player has a greater than 40% chance of scoring. This is quite high and represents a "sweet spot" in the game. By watching what happens when the ball is kicked into this region, we see that a nearby defender reacts poorly, running away from the ball instead of intercepting it. The display of the rules and the interface for running the simulations to see what is going on are handled by our SoccerViz visualizer. Figure 6 shows a single negative rule for the shooter-goalie scenario, along with the rest of the SoccerViz interface.

Active Learning

The literature on active learning offers many algorithms for deciding where to sample next, chiefly differing in how they identify "interesting" regions. We have explored several samplers which fall into three broad categories: *uncertainty sampling* (Lewis & Gale), *query by committee* (QBC) (Seung, Opper, & Sompolinsky 1992), and our own method, *decision boundary refinement* sampling. A detailed discussion of these methods is beyond the scope of this paper, but we will briefly describe our own active learning method, which was used to produce the results shown here.

Decision boundary refinement sampling was developed specifically to deal with rule learners. As shown in Figure 4 (d), a set of rules can create a region. To ensure that the boundary of this region is correct, new samples are randomly placed on both sides of the boundary within a small margin, specified by a parameter. These additional samples will provide evidence confirming or refuting the current boundary. Sampling near the boundaries between overlapping, agreeing rules is wasteful. This means we cannot simply sample at rule boundaries but must identify the regions to do effective sampling. Fortunately, this is quite straightforward.

The active learning methods are of research interest, but happily their presence is transparent to the designer using the tool. We have implemented all of the above methods and will publish a comparison in an upcoming paper focused on the machine learning issues. For now, we observe that the C4.5 rule learner combined with decision boundary refinement sampling produces sensible rules and that active learning in general helps scale the approach to higher dimensions.

Visualization

Visualization is an important aspect of gameplay management, and one which is best handled by game developers who have considerable graphics, user-interface, and gameplay design experience. Nevertheless, we have developed a tool to demonstrate how SAGA-ML usage might feature in real game development. The SoccerViz visualization tool is shown in Figure 6. This tool displays the rules generated by the learning as regions on a 2-D soccer field and allows the user to examine the samples supporting the rule. These samples can also be exported to the FIFA'99 engine (see Figure 2). This visualization is clearly specialized for soccer, and some customization was required for the different scenarios. We believe that, in industry practice, much of the visualization could be incorporated into the real game engine, or into its associated design tools, in a cost-effective manner.

Usage

SAGA-ML could be used in a variety of ways during game development. We will briefly mention a few possible strategies, which may be combined to meet developer needs.

- Desktop Analysis: The designer interacts with SAGA-ML through the visualizer a few times a day, requesting analyses as changes are made. Small sample sets are used to get a quick impression of game behaviour.
- Batch Analysis: Large-scale sampling, run overnight or on weekends. Used to evaluate milestone builds or more frequently if resources are available. Results are examined by individuals or teams to check progress.
- Regression Testing: Regular analysis using an established set of sample points (instead of choosing new ones) to ensure that established behaviours are not lost. Models learned from previous tests are kept for comparison. It may be possible to automatically detect drastic changes in models and alert developers.
- Retrograde Analysis: Once small scenarios have been analyzed, they can be combined to form a larger scenario. While this larger scenario may be too large to analyze directly, the models learned for its components can be used to make predictions without running the actual game engine. Preliminary results in this area look promising.

Evaluation

SAGA-ML and SoccerViz have been used to analyze four different scenarios in FIFA'99, identifying various sweet and hard spots. However, the real question is whether commercial game companies find this technology useful. This work was recently presented to developers at two Electronic Arts sports game studios (Vancouver, BC and Orlando, FL) where it was very well received. EA Vancouver has since ported the abstraction layer to the FIFA'2004 engine so we can extend our research. More significantly, SAGA-ML and SoccerViz are now being used in-house at EA Vancouver during development of the FIFA Soccer series. We regard this step as a substantial validation of our approach. The research was also well received at the industry's Game Developers Conference (Southey *et al.* 2005).

Conclusions

The gameplay management task presents tough challenges but our contacts with the industry indicate a strong inclination to pursue them. It will undoubtedly form a substantial portion of our future research. Our current work on gameplay analysis has revealed many interesting AI research problems and our experiments have uncovered real gameplay issues in a commercial product. The software has been adopted for internal use by a major game software developer. Future directions for analysis include allowing the user to guide sampling, new active learning methods, new learning algorithms, new scenarios and new games. Another key area is moving from sampling fixed sequences of actions to policies that specify actions for different situations. With good analyzers in place, we can look forward to the adjustment aspect of the task, tuning game parameters automatically to meet designer-specified goals. We believe this work will be rewarding to both the AI research community and game developers, and that it offers another avenue for AI research to interface with computer games.

References

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

Lewis, D. D., and Gale, W. A. A sequential algorithm for training text classifiers. In *Proceedings of SIGIR-94*.

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.

Southey, F.; Holte, R. C.; Xiao, G.; Trommelen, M.; and Buchanan, J. 2005. Machine Learning for Semi-Automated Gameplay Analysis. In *Proceedings of the 2005 Game Developers Conference (GDC-2005)*.