Artificial Intelligence for Adaptive Computer Games

Ashwin Ram, Santiago Ontañón, and Manish Mehta
Cognitive Computing Lab (CCL)
College of Computing, Georgia Institute of Technology
Atlanta, Georgia, USA
{ashwin, santi, mehtama1}@cc.gatech.edu

Abstract

Computer games are an increasingly popular application for Artificial Intelligence (AI) research, and conversely AI is an increasingly popular selling point for commercial games. Although games are typically associated with entertainment, there are many "serious" applications of gaming, including military, corporate, and advertising applications. There are also so-called "humane" gaming applications for medical training, educational games, and games that reflect social consciousness or advocate for a cause. Game AI is the effort of going beyond scripted interactions, however complex, into the arena of truly interactive systems that are responsive, adaptive, and intelligent. Such systems learn about the player(s) during game play, adapt their own behaviors beyond the pre-programmed set provided by the game author, and interactively develop and provide a richer experience to the player(s).

The long-term goal of our research is to develop artificial intelligence techniques that can have a significant impact in the game industry. In this paper, we present a list of challenges and research opportunities in developing techniques that can be used by computer game developers. We discuss three Case-Based Reasoning (CBR) (Aamodt & Plaza 1994), (Kolodner 1993) approaches to achieve adaptability in games: automatic behavior adaptation for believable characters; drama management and user modeling for interactive stories; and strategic behavior planning for real-time strategy games.

Introduction

Computer games have been classified as the "Human-level AI's Killer Application" (Laird & van Lent 2000). State-of-the-art computer games recreate real-life environments with a surprising level of detail. These environments are usually populated with many characters (allies or enemies) that require human-level intelligence and exhibit believable behaviors. However, even though there have been enormous advances in computer graphics, animation and audio for games, most of the games contain very basic artificial intelligence (AI) techniques. AI programming wisdom series (Rabin 2002; 2004) provides a good overview of current state of the art AI techniques used in the game industry. As a result of simplistic approaches for AI, the whole atmosphere created by the game can be broken when the game and characters situated within it behave in a non-believable manner. On the other hand, creating richer experiences requires a great deal of engineering effort on the part of game developers.

The development of AI techniques for computer games also impacts several areas outside the game industry. Autonomous characters can be used in any anthropomorphic interface, and have already been employed in a variety of applications including as instruction agents in training environments (Lester & Stone 1997), as presentation agents for giving slide presentations (Lester & Stone 1998), and as guide agents on websites (Isbister & Doyle 2003). Interactive plots (drama management agents) can be employed in education and training environments (Isbister & Doyle 2001). There is also great interest in applying lessons from game design to the design of "serious games" for use in military and corporate applications (Sawyer 2003).

In recent years, interest in applying AI techniques and in particular CBR approaches to computer games (Aha, Molineaux, & Ponsen 2005; Cheng & Thawonmas 2004) has seen a notable increase. There have also been conferences and workshops dedicated particularly to Game AI (e.g., AI-IDE and workshops on game AI at ICCBR 2005 (Aha & Wilson 2005) and IJCAI 2005 (Aha, Munoz-Avila, & van Lent 2005)). The vast majority of this work, however, focuses on small subproblems within a computer game (small tactical-level problems, coordination, path planning, etc.) or is not situated within a real game. Although this research provides interesting solutions and ideas, it cannot be directly applied by computer game companies. As computer games are being developed by increasingly large project teams with increasingly tight timelines, game developers do not have the necessary cycles needed to try and transition these techniques to their own games. One of the long-term goals of our work is to reduce the transition effort needed in applying academic AI techniques in real games. Further, we want to ease the effort in developing more complex AI for computer games to make them more adaptive and appealing to the player.

The research described in this paper is a step towards this objective. This research focuses on the development of reasoning and learning techniques in the context of current state-of-the-art computer games. These techniques can allow non-AI experts to define behaviors for characters that can then be adapted to different situations and individual players, thereby reducing the development effort required to address all contingencies in a complex game. Specifically, we are interested in *adaptive games*, i.e., games that can adapt themselves to unforeseen situations. For a more elaborate discussion on adaptivity in computer games see (Spronck 2005). In our viewpoint, Adaptive games can perform the following functions:

- Improve the player experience, since an adaptive game can adapt to each individual player to better fit his or her playing style and goals, and
- Reduce the development effort, since if a game is able to adapt itself, the developers require less effort trying to foresee all possible situations.

The rest of the paper is organized as follows. First, we present a brief analysis of the requirements of different game genres from an AI perspective. Following this, we identify a set of challenges that computer games present for the AI community. Next, we discuss three CBR approaches for adaptive games: automatic behavior adaptation for believable characters, drama management and user modeling for interactive stories, and strategic behavior planning for real-time strategy games. We finish with conclusions and future research directions.

Requirements for Game AI

In previous work, Laird and van Lent (2000) analyzed different game genres, and the AI challenges that each presents. In their report, they considered the following types of games: action, role playing, adventure, strategy games, god games, individual and team sports games. In addition to those genres, we would like to consider two additional categories, namely, interactive drama (Mateas & Stern 2003) and educational games (Rieber 1996). Interactive dramas have a strong plot behind them that the author wants to communicate to the player, but where the player may have a strong influence on the plot. A key difference with the classical "adventure" genre is that adventures have a scripted plot, while interactive dramas are more open-ended and adapt to the player interaction as the story unfolds. Educational games have an additional rhetorical goal of teaching some particular content to the player.

By analyzing the range of possible applications of computer game AI to different applications and game genres, we identify two different levels at which AI can be applied: 1) individual characters AI, with the goal of producing more intelligent or believable behaviors, and 2) a global AI that watches over the game or game-player interaction, influencing the directions that the game is taking. Thus, we can talk about *character-level AI* and *game-level AI* (the second being referred in some papers as the *Drama Manager* (Nelson *et al.* 2006) or as the *Director* (Magerko *et al.* 2004)).

Different applications and game genres require a different mix of these two kind of AIs. For instance, real-time strategy games rely mainly on a game-level AI that controls all the units, while the individual unit behaviors can be scripted. Role playing games, on the other hand, require believable character-level AI to provide an interesting player experience. Interactive dramas requires a mix of both kinds of AI: individual characters that are believable and a drama manager that leads the plot by guiding the individual characters to take actions that can make the drama advance. Educational applications of gaming also require a game-level AI, similar to the drama manager, which monitors the interaction of the game as it unfolds, easing or complicating the tasks according to the learner's expertise level, thereby making sure that educational purpose of the game is being met.

Each game genre presents particular requirements for character level and game level AI. For instance, god games usually require the game-level AI to solve resource allocation problems and solve long-term strategy problems, while interactive drama requires the game-level AI to adapt the story according to the player interactions in a way that it is more appealing to the player (thus, the latter requires user modeling and story planning). Moreover, adventures, interactive dramas and other genres with embodied characters usually require believability and natural language generation.

In the following section, we summarize a list of interesting challenges that computer games pose in general to the AI community.

Challenges in Computer Game AI

Let us briefly describe some of the main issues that arise when developing artificial intelligence for computer games. This list is not exhaustive, but is intended to give a flavor of the kind of problems that real computer games pose to the AI community.

- *Complex decision spaces*: Most state-of-the-art computer games involve complex strategic (real-time strategy games) or believable behaviors (interactive dramas). Both kind of behaviors share the characteristic of having huge decision spaces, and thus traditional search-based AI techniques cannot be applied. Learning techniques or higher level representations are required to deal with such complex games. Traditionally, computer games use handcrafted strategies coded by the game developers, but these tend to be repetitive, and players easily find holes and exploit them.
- *Knowledge engineering*: Even assuming that strategies or behaviors are handcrafted, authoring these behavior sets in a game requires a huge human engineering effort. Game developers have to encode all the knowledge they have about a domain (either to achieve a strategic behavior or a believable human behavior) in some sort of behavior language.
- *Authoring support*: Hand crafted behaviors are, ultimately, software code in a complex programming language, prone to human errors. The behavior errors could be in the form of program "bugs" or not achieving the desired result. Tools are needed to support story authors, who are typically not artificial intelligence experts, to author behaviors in a computer programming language.

- *Unanticipated situations*: It is not feasible to anticipate all possible situations and player strategies that can encountered during game play. This makes it difficult to craft believable behaviors that react in an appropriate manner to these unforeseen circumstances and player actions.
- *User-specific adaptation*: Different players may enjoy different strategies to fight against (in the case of real-time strategy games), or different styles of story telling (in the case of interactive dramas), different types of story development, different kinds of character behaviors and interactions, or different educational problems. As game designers begin to include user modeling capabilities, the AI strategy and behavior must, in turn, be adaptable based on the user model.
- *Replayability and variability*: A player might get bored of seeing the same strategies and behaviors again and again. Although simple variability can be achieved through stochastic selection of behaviors or strategies from a large repository, this increases the authoring burden. Furthermore, random selection begs the question of true interestingness.
- *Rhetorical objectives*: It is possible, even likely, that human-engineered behaviors or strategies do not achieve the game's objectives adequately, especially in realistic, scaled-up domains or applications. These objectives could range from entertainment to education, training, etc. Thus, the game has to realize that the objectives are not being met on a per-use basis, and adapt accordingly. For example, a particular user may be getting bored, or not learning the intended lesson.

A conclusion that can be drawn from the previous list is that not only can games benefit from better AI techniques, but AI can also benefit from the challenges that computer games provide. In the remainder of this paper, we will present three research projects that contribute to achieving a subset of these challenges. The three projects have the same underlying theme of (a) easing the effort involved in developing computer game AI and (b) making them more adaptive and appealing to the player.

The first project focuses on a framework for runtime behavior adaptation that aims at removing the responsibility of the authors to foresee all possible situations that can be encountered during the game. The second project aims also at reducing the effort of defining strategic behaviors for real-time strategy games by extracting behavioral knowledge from experts using a system that can automatically learn new behaviors from example of game play sessions. Finally, the third project aims at making the games more adaptive to the players by using player modeling. The approach predicts which sequence of events will be more appealing for each player and influences the game accordingly through the drama manager.

Behavior Modification for Believable Characters AI

In interactive games, embodied characters typically have their own personalities, affecting the way they act in the game. Authors usually create such characters by writing behaviors or scripts that describe the characters' reaction to all imaginable circumstances within the game world. This approach of authoring characters presents several difficulties. First, when authoring a character's behavior set, it is hard to imagine and plan for all possible scenarios it might encounter. Given the rich, dynamic nature of game worlds, this can require extensive programming effort. Second, over long game sessions, a character's static behavioral repertoire may result in repetitive behavior. Such repetition harms the believability of the characters. Third, when behaviors fail to achieve their desired purpose, characters are unable to identify such failures and will continue to exhibit them. Ideally, we want a self-adapting behavior set for characters, allowing characters to autonomously exhibit their author-specified personalities in new and unforeseen circumstances, and relieving authors of the burden of writing behaviors for every possible situation.

To address these issues, we have developed an approach in which agents keep track of the status of their executing behaviors, infer from their execution trace what might be wrong, and perform appropriate revisions to their behaviors. This approach to runtime behavior transformation enables characters to autonomously adapt during execution to changing game situations, taking a first step towards automatic generation of behavior that maintains desired personality characteristics. Our approach is related to plan revision (Cushing & Kambhampati 2005), with the added complexity that failure detection and behavior modification must be performed during execution, enabling the game to continue seamlessly from the player's perspective. This section presents an overview of such approach; for more details see (Zang *et al.* 2007).

Behavior Transformation System

Our game scenario consists of two embodied characters named Jack and Jill. They are involved in a game of Tag, implemented in Unreal Tournament (Epic Games 2004), where they chase the character who is "It" around the game area. The system (see Figure 1) is composed of a reactive layer which handles the real-time interactions, and a reasoning layer responsible for monitoring the character's state and making repairs as needed.

We use A Behavior Language (ABL) as the reactive layer. ABL is explicitly designed to support programming idioms for the creation of reactive, believable agents (Mateas & Stern 2002). A character authored in ABL is composed of a library of behaviors, capturing the various activities the character can perform in the world. ABL's fast runtime execution module makes it suitable for real-time scenarios. The runtime execution module constantly senses the world, keeps track of the current game state, initiates and monitors primitive actions in the game world.

The reasoning layer consists of two components. The first component tracks long-term patterns in the character's behavior execution and detects violations of the author-specified behavior contract (see below). When a contract violation is detected, it uses the execution trace to perform blame assignment, identifying one or more behaviors that

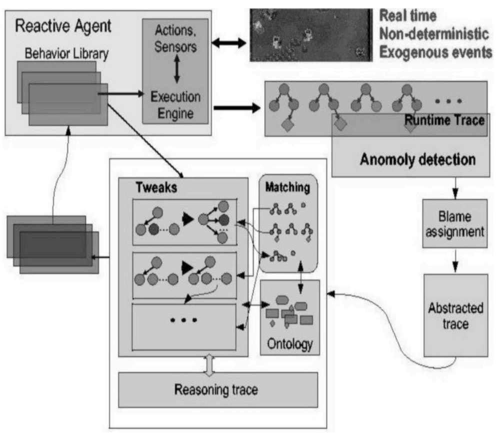

Figure 1: Architecture of our behavior transformation system.

should be changed. The second component applies behavior modification operators so as to repair the offending behaviors identified during blame assignment.

One of the essential requirements of a reasoning system responsible for runtime behavior modification is to detect when modification should be carried out. We need a way for authors to specify contracts about long-term character behavior; when the contract is violated, the reasoning layer should modify the behavior library. To accomplish this, we use a simple emotion model based on Em (Loyall 1997), an OCC (Bartneck 2002) model of emotion. Emotion values serve as compact representations of long-term behavior. The author specifies personality-specific constraints on behavior by specifying nominal bounds for emotion values. When an emotion value exceeds the bounds specified by the author, this tells the reasoning layer that the current behavior library is creating inappropriate long-term behavior and that it should seek to assign blame and change its behavior. At runtime, a character's emotional state is incremented when specific behaviors, annotated by the author, succeed or fail. The emotion increment value per behavior is defined by the author as part of specifying the character personality.

A second requirement on the reasoning module is to determine the behavior(s) that should be revised in response to a violation of the personality contract (in our case, an emotion value exceeding a bound). This blame assignment process involves analyzing the past execution trace and identifying the behavior with the maximal contribution to the out-of-bound emotion value, amortized over time, as the responsible behavior.

Once the reasoning module has detected the behavior(s) that need to be modified, the behaviors are modified using a set of modification operators. The applicability of a modification operator depends on the role the problematic behavior plays in the execution trace, that is, an explanation of how the problematic behavior contributed to a contract violation. Thus, modification operators are categorized according to failure patterns. The failure patterns provide an abstraction mechanism over the execution trace to detect the type of failure that is taking place. On an implementation level, these failure patterns are encoded loosely as finite state machines that look for patterns in the execution trace.

At runtime, the system detects when the author-provided behavior contract has been violated. Once blame assignment has determined the offending behavior, the system uses the failure patterns to explain the behavior's role in the contract violation. This involves matching each of the finite state machines associated with the failure pattern against the execution trace. The set of matching failure patterns provide an associated set of applicable behavior modification operators to try on the offending behavior. Operators are tried one at a time until one succeeds (operators can fail if the behavior they are tweaking lacks the structural prerequisites for the application of the operator). The modified behavior is compiled and reloaded into the agent, allowing the game to continue.

Future Work In order to increase the transformational power of our system, we are adding more behavior modification operators to the system. Adding additional operators has several effects. First, as the number of operators increases, the time required to reason about them and find the applicable set increases. Second, operators for more complex scenarios may have a lower success rate, requiring us to focus the search through behavior transformation space. It will become necessary for the reasoning layer to learn which operators are best applicable in which situations, such that fewer operators have to be tried. These characteristics of the problem make a case-based approach, as a form of speedup learning, very attractive. We are in the process of integrating a case-based reasoner into our system.

Case-Based Planning for Strategy Games

AI techniques have been successfully applied to several computer games such as checkers, chess or Othello (Schaeffer 2001). However, in many computer games traditional AI techniques fail to play at a human level because of the characteristics of the vast search spaces this games require. For that reason, game developers need to invest significant effort in hand-coding specific strategies that play at a reasonable level for each new game.

For instance, previous research has shown that real-time strategy games (RTS) such as Wargus (a clone of the popular commercial game Warcraft II) have huge decision spaces (Aha, Molineaux, & Ponsen 2005; Buro 2003). In this section we present an architecture that uses case-based planning (Hammond 1990) to deal with such complex games.

In previous work, we have applied case-based reasoning (CBR) to RTS games (Sharma *et al.* 2007a). The idea there was to define a set of high level actions, and let a CBR system learn when each should be applied. In this section, we discuss a different approach that addresses the complexity of this domain by extract behavioral knowledge from expert demonstrations (i.e., an expert plays the game and our system observes). Then, at performance time, a case-based planning engine retrieves suitable behaviors observed from the expert and adapts them to the current game state. Adaptation is required since the game state may be different from

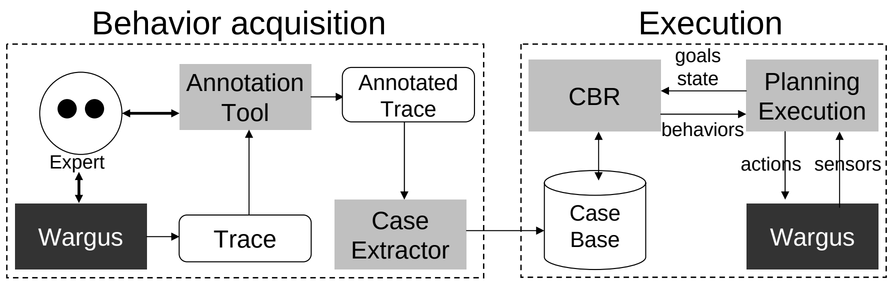

Figure 2: Overview of our case-based behavior acquisition and planning architecture.

the one in which the behavior was originally demonstrated, with, for example, a new map, different units, or a different objective.

One of the main contributions of this approach is that it enables the game developers to specify the AI behavior just by demonstration, i.e., instead of having to code the behavior using a programming language, the behavior can be specified simply by *demonstrating* it to the system. If the system shows an incorrect behavior in any particular situation, instead of having to find the bug in the program and fix it, the game developers can simply demonstrate the correct action in the particular situation. The system will then incorporate that information in its case base, thereby improve its behaviors in the future.

Case-Based Behavior Learning in Wargus

Figure 2 shows an overview of the process used to learn behaviors from expert demonstrations. The process is divided into two stages:

- *Behavior acquisition*: Where a set of cases are extracted from an expert trace.
- *Execution*: Where the cases extracted are reused to play the game.

The first step in the process involves an expert providing a demonstration to the system of how to play the game. As a result of that demonstration, the system obtains a game trace consisting of the set of actions executed during the game. The next step is to annotate the trace. For this process, the expert uses a simple annotation tool that allows him to specify which goals he was pursuing with each particular action.

Next, as Figure 2 shows, the annotated trace is processed by the *Case Extractor* module, that encodes the strategy of the expert in this particular trace in a series of cases. A case stores a sequence of actions that an expert used in a particular situation to achieve a particular goal. Notice that from a single trace many cases can be extracted. For instance, if the expert destroyed multiple towers of the enemy, we can collect multiple cases on how to destroy a tower.

Once the cases have been extracted from the expert demonstration, the system is ready to use them in actual game play. During performance, the interleaved Planning and Execution (PE) module keeps track of the open goals. For each open goal, the PE module sends the goal and the current game state to the CBR module. In response, the CBR module selects a case from the case base that suits the goal and game state. That case is adapted to match the current game state (adjusting which units make which actions, and on what coordinates, since in the new map the stored coordinates in the case might not make sense). Once adaptation has taken place, the adapted behavior is sent to the PE module.

The PE module keeps track of the behaviors that are being executed for each goal, and ensures that each subgoal of each behavior is also satisfied by maintaining an execution tree of goals and subgoals. In addition, by using the *alive conditions* of the behaviors, it monitors whether a particular behavior is still alive, and whether it is worthwhile to continue executing it. Each time a behavior is finished (or canceled), the PE module checks whether the goal that behavior was pursuing has been achieved. If it has not been achieved, then another behavior must be created by the CBR module to satisfy the goal.

Future Work We plan to incorporate learning during performance by retaining those adapted behaviors that succeeded when applied. This will enable the system to learn with experience. The goal is to use the behaviors extracted from the expert as the starting point, and slowly improve the behavior library with experience. At any time, if the game developers see that the system is unable to succeed in a particular situation, an expert demonstration for that situation can be provided.

Drama Management in Interactive Stories

A typical problem in creating compelling story-based games, such as the interactive drama *Façade* (Mateas & Stern 2003), is to provide the player with a sense of agency during the interaction while simultaneously giving the whole experience an overall narrative structure. There is a growing interest in developing Drama Manager (DM) components that gently guide the player towards a story ending that exhibits a narrative arc. The goal of these components is to allow the player to have significant impact on what happens during the interaction, rather than following along with a pre-written script or being in control at only a few decision points.

Previous approaches to DM development have used an author-specified evaluation function to measure interestingness and narrative coherence of a particular story path (Weyhrauch 1997) and have employed a simulated player

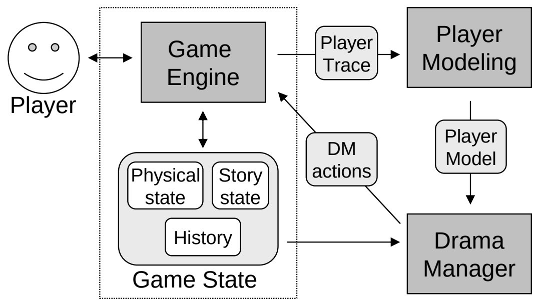

Figure 3: Basic scheme of the three modules that compose a game in the proposed approach.

model to predict the next player action during the interaction (Nelson *et al.* 2006). However, in experiential interactive systems, the player's preferences must be taken into account. This requires player models constructed during actual game play based on observations of real human players interacting with the game. In this section we will present a drama management approach that takes into account the player by using a case-based user modeling approach, using the learned models as input to the Drama Manager (for more details see (Sharma *et al.* 2007b)). The technique is implemented with the well-known interactive fiction game Anchorhead (Gentry 1998).

Integrating User Modeling with Drama Management

Our approach to drama management consists of three modules (shown in Figure 3), namely: a *Game Engine*, responsible for actually running the game and interacting with the player; a *Player Modeling* module, responsible for analyzing the actions of the current player and developing a player model; and a *Drama Management* module, influencing the development of the game and making it more appealing to the player (represented as the player model).

The game engine handles the player input (formalized as *player actions*); presents the game state to the user including information regarding the kind of actions the player can perform (via an audiovisual interface); and maintains the current game state. Specifically, the game state consists of three parts: The *physical state*, the *story state*, and the *history*. The physical state stores physical information such as the location of the characters in the game. The story state is represented as a set of *plot points* (Weyhrauch 1997). A plot point is an event that is relevant to the game story (e.g., "the librarian had a car accident"). Plot points are structured as a directed graph, where each plot point acts as a node in the graph, and the arcs represent dependencies. Finally, the history is a record of what has happened in the game since the start to the current point in time.

The Player Modeling Module (PMM) constructs player models based on the feedback provided by players at the end of each game. This feedback contains player opinions on the game, including the parts they enjoyed and those that were not interesting from their perspective. The goal is to capture the interestingness rating for the story elements encountered by the player during the game episode. At the end of each interaction, the PMM stores this player feedback along with the corresponding trace of player actions during the game. In particular, we use a case-based reasoning approach for this module, with each player experience being stored as a separate case in a case base.

During a particular game episode, the PMM utilizes this information to compare the ongoing player trace maintained by the game engine with player traces collected during previous interactions with different players. The feedback from players with the closest matching traces are combined to form an estimate of the stories that the current player is most likely to prefer. The underlying assumption behind this approach is that if the current player's actions follow a pattern that closely resembles those of certain players who have previously played this game, then their interestingness rating for stories would also closely match. The PMM uses this assumption to estimate the stories the current player is likely to enjoy. In general the player model stored in the PMM contains information about a player's ratings for the locations, plot points, or sequences of plot points in the game.

Once the player module is generated, it is given to the Drama Manager Module (DMM), whose role is to guide the development of the story according to both the player model and the author-specified story guidelines. Specifically, the DMM selects appropriate *drama manager actions* (DM actions) at each point in the game. These actions represent the things that the drama manager can carry out to influence the game, e.g., "prevent the player from entering the library by locking the door" or "hinting to the player that an object is important by making one of the characters in the game talk about it." DM actions are specified by the game author in the same way that he specifies the player actions that the player might take at each moment of time in the game. Given the set of possible directions in which the story might unfold (as a function of the player's selected actions), the drama manager plans a set of DM actions to guide the player towards story directions that are likely to be more appealing to him, according to the player module and to author specified story guidelines.

Future Work We plan to perform extensive player evaluation to validate the case-based player modeling module and the drama manager module. We also plan to expand the player modeling module to generate player action models that can predict the actions a particular player is likely to take in given situations. Finally, we plan to move from the text-based game Anchorhead to real-time 3D systems, where the complexity of drama manager is increased considerably.

Conclusions

In this paper, we discussed a set of challenges that state-of-the-art computer games pose to the artificial intelligence community. Developing AI techniques that can deal with the complexity of computer games is a big challenge, but has the potential to have a big impact in several areas including entertainment, education, and training.

Our main goal is to develop AI techniques that can ease the effort of incorporating AI in computer games to make them more adaptive and appealing to the player. We call such games *adaptive games*. In this paper, we introduced three of our current research thrusts aimed at creating adaptive games via the application of case-based reasoning techniques.

We share the vision of other computer game AI industry people (Rabin 2004; Woodcock 2005) that game AI will be the next revolution in the gaming industry. After the impressive advances in audiovisual presentation and networking capabilities, the next step in computer games is to incorporate advanced AI techniques that can achieve the goal of having truly adaptive games, increasing the level of believability and immersion. To achieve this goal, the gaming community needs new techniques, approaches and tools that allow them to easily specify, develop, and incorporate AI in their games.

Acknowledgements

The authors would like to thank all the people who contributed to the projects discussed in this paper: Michael Mateas, author of the ABL language and collaborator on the behavior modification project, Peng Zang for behavior modification, Manu Sharma, Michael Holmes, Arya Irani and Charles Isbell for real-time strategy games, and Mark Nelson and David Roberts for their help with the drama management work. We are grateful to DARPA for their funding under the Transfer Learning and Integrated Learning programs.

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications* 7(1):39–59.

Aha, D. W., and Wilson, D. 2005. Computer gaming and simulation environments: Papers from the ICCBR workshop. Technical Report, DePaul University, Chicago, IL.

Aha, D.; Molineaux, M.; and Ponsen, M. 2005. Learning to win: Case-based plan selection in a real-time strategy game. In *ICCBR'2005*, number 3620 in LNCS, 5–20. Springer-Verlag.

Aha, D. W.; Munoz-Avila, H.; and van Lent, M. 2005. Reasoning, representation, and learning in computer games: Papers from the IJCAI workshop. Technical Report AIC-05-127, Washington DC: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence.

Bartneck, C. 2002. Integrating the occ model of emotions in embodied characters. In *Proceedings of the Workshop on Virtual Conversational Characters: Applications, Methods, and Research Challenges*.

Buro, M. 2003. Real-time strategy games: A new AI research challenge. In *IJCAI 2003*, 1534–1535. Morgan Kaufmann.

Cheng, D. C., and Thawonmas, R. 2004. Case-based plan recognition for real-time strategy games. In *Proceedings of the 5th Game-On International Conference*, 36 – 40.

Cushing, W., and Kambhampati, S. 2005. Replanning: A new perspective. In *Proceedings of ICAPS*.

Epic Games. 2004. Unreal tournament 2004, http://www.unrealtournament.com.

Gentry, M. S. 1998. Anchorhead. available online at http://www.wurb.com/if/game/17.html.

Hammond, K. F. 1990. Case based planning: A framework for planning from experience. *Cognitive Science* 14(3):385–443.

Isbister, K., and Doyle, P. 2001. Toward the holodeck: Integrating graphics, sound, character, and story. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 409 – 416.

Isbister, K., and Doyle, P. 2003. Web guide agents: Narrative context with character. In Mateas, M., and Sengers, P., eds., *Narrative Intelligence*, 229–243.

Kolodner, J. 1993. *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann.

Laird, J. E., and van Lent, M. 2000. Human-level AI's killer application: Interactive computer games. In *AAAI 2000*, 1171–1178.

Lester, J., and Stone, B. 1997. Increasing believability in animated pedagogical agents. In *First International Conference on Autonomous Agents*, 16–21.

Lester, J., and Stone, B. 1998. Integrating reactive and scripted behaviors in a life-like presentation agent. In *Proceedings of the Second International Conference on Autonomous Agents*, 261–268.

Loyall, B. 1997. *Believable Agents: Building Interactive Personalities*. Ph.D. Dissertation, Carnagie Mellon University.

Magerko, B.; Laird, J.; Assanie, M.; Kerfoot, A.; and Stokes, D. 2004. AI characters and directors for interactive computer games. In *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Confercence*.

Mateas, M., and Stern, A. 2002. A behavior language for story-based believable agents. *IEEE intelligent systems and their applications* 17(4):39–47.

Mateas, M., and Stern, A. 2003. Integrating plot, character, and natural language processing in the interactive drama façade. In *Proceedings of the 1st International Conference on Technologies for Interactive Digical Storytelling and Entertainment*.

Nelson, M.; Roberts, D.; Isbell, C.; and Mateas, M. 2006. Reinforcement learning for declarative optimization-based drama management. In *AAMAS 2006*, 775–782.

Rabin, S. 2002. *AI Game Programming Wisdom*. Charles River Media.

Rabin, S. 2004. *AI Game Programming Wisdom II*. Charles River Media.

Rieber, L. P. 1996. Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational Technology Research and Development* 44(2):43–58.

Sawyer, B. 2003. Serious games: Improving public policy through game-based learning and simulation. *Whitepaper for the Woodrow Wilson International Center for Scholars*.

Schaeffer, J. 2001. A gamut of games. *AI Magazine* 22(3):29–46.

Sharma, M.; Homes, M.; Santamaria, J.; Irani, A.; Isbell, C.; and Ram, A. 2007a. Transfer learning in real time strategy games using hybrid CBR/RL. In *IJCAI 2007*, 1401–1406.

Sharma, M.; Ontañón, S.; Strong, C.; Mehta, M.; and Ram, A. 2007b. Towards player preference modeling for drama management in interactive stories. In *FLAIRS 2007*.

Spronck, P. 2005. *Adaptive game AI*. Ph.D. Dissertation, Maastricht University.

Weyhrauch, P. 1997. *Guiding Interactive Drama*. Ph.D. Dissertation, Carnagie Mellon University.

Woodcock, S. 2005. Game AI: The state of the industry, available online at http://www.gamasutra.com/features/20001101/.

Zang, P.; Mehta, M.; Mateas, M.; and Ram, A. 2007. Towards runtime behavior adaptation for embodied characters. In *IJCAI'2007*, 1557–1562.