Rapidly Adapting Game AI

Sander Bakkes Pieter Spronck Jaap van den Herik

Tilburg University / Tilburg Centre for Creative Computing (TiCC)
P.O. Box 90153, NL-5000 LE Tilburg, The Netherlands
{s.bakkes, p.spronck, h.j.vdnherik}@uvt.nl

Abstract

Current approaches to adaptive game AI require either a high quality of utilised domain knowledge, or a large number of adaptation trials. These requirements hamper the goal of rapidly adapting game AI to changing circumstances. In an alternative, novel approach, domain knowledge is gathered automatically by the game AI, and is immediately (i.e., without trials and without resource-intensive learning) utilised to evoke effective behaviour. In this paper we discuss this approach, called 'rapidly adaptive game AI'. We perform experiments that apply the approach in an actual video game. From our results we may conclude that rapidly adaptive game AI provides a strong basis for effectively adapting game AI in actual video games.

1 Introduction

Over the last decades, modern video games have become increasingly realistic with regard to visual and auditory presentation. Unfortunately, game AI has not reached a high degree of realism yet. Game AI is typically based on non-adaptive techniques [18]. A major disadvantage of non-adaptive game AI is that once a weakness is discovered, nothing stops the human player from exploiting the discovery. The disadvantage can be resolved by endowing game AI with adaptive behaviour, i.e., the ability to learn from mistakes. Adaptive game AI can be established by using machine-learning techniques, such as artificial neural networks or evolutionary algorithms. In practice, adaptive game AI in video games is seldom implemented because machine-learning techniques typically require numerous trials to learn effective behaviour. To allow rapid adaptation in games, in this paper we describe a means of adaptation that is inspired by the human capability to solve problems by generalising over a limited number of experiences with a problem domain.

The outline of this paper is as follows. First, we discuss the aspect of entertainment in relation to game AI. Then, we discuss our approach to establish rapidly adaptive game AI. Subsequently, we describe an implementation of rapidly adaptive game AI. Next, we describe the experiments that apply rapidly adaptive game AI in an actual video game, followed by a discussion of the experimental results. Finally, we provide conclusions and describe future work.

2 Entertainment and Game AI

The purpose of a typical video game is to provide entertainment [18, 12]. Of course, the criteria of what makes a game entertaining may depend on who is playing the game. Literature suggests the concept of immersion as a general measure of entertainment [11, 17]. Immersion concerns evoking an immersed feeling with a video game, thereby retaining a player's interest in the game. As such, an entertaining game should at the very least not repel the feeling of immersion from the player [9]. Aesthetical elements of a video game, such as graphics, narrative and rewards, are instrumental in establishing an immersive game-environment. Once established, the game environment needs to uphold some form of *consistency* for the player to remain immersed within it [9]. Taylor [17] argues that a lack of consistency in a game can cause player-immersion breakdowns.

The task for game AI is to control game characters in such a way that behaviour exhibited by the characters is consistent within the game environment. In a realistic game environment,

realistic character behaviour is expected. As a result, game AI that is solely focused on exhibiting the most *challenging* behaviour is not necessarily regarded as realistic. For instance, in a typical first-person shooter (FPS) game it is not realistic if characters controlled by game AI aim with an accuracy of one hundred per cent. Game AI for shooter games, in practice, is designed to make intentional mistakes, such as warning the player of an opponent character's whereabouts by intentionally missing the first shot [10].

Consistency of computer-controlled characters with a game environment is often established with tricks and cheats. For instance, in the game HALF-LIFE, tricks were used to establish the illusion of collaborative teamwork [9], causing human players to assume intelligence where none existed [10]. While it is true that tricks and cheats may be required to uphold consistency of the game environment, they often are implemented only to compensate for the lack of sophistication in game AI [4]. In practice, game AI in most complex games still is not consistent with the game environment, and exhibits what has been called 'artificial stupidity' [10] rather than artificial intelligence. To increase game consistency, and thus the entertainment value of a video game, we agree with Buro and Furtak [4] that researchers should foremost strive to create the most optimally playing game AI possible. In complex video-games, such as real-time strategy (RTS) games, near-optimal game AI is seen as the only way to obtain consistency of the game environment [9]. Once near-optimal game AI is established, difficulty-scaling techniques can be applied to downgrade the playing-strength of game AI, to ensure that a suitable challenge is created for the player [15].

3 Approach

For game AI to be consistent with the game environment in which it is situated, it needs the ability to adapt adequately to changing circumstances. Game AI with this ability is called 'adaptive game AI'. Typically, adaptive game AI is implemented for performing adaptation of the game AI in an online and computer-controlled fashion. Improved behaviour is established by continuously making (small) adaptations to the game AI. To adapt to circumstances in the current game, the adaptation process typically is based only on observations of *current* gameplay. This approach to adaptive game AI may be used to improve significantly the quality of game AI by endowing it with the capability of adapting its behaviour while the game is in progress. For instance, the approach has been successfully applied to simple video games [5, 8], and to complex video games [15]. However, this appproach to adaptive game AI requires either (1) a high quality of the utilised domain knowledge, or (2) a large number of adaptation trials. These two requirements hamper the goal of achieving *rapidly* adaptive game AI.

To achieve rapidly adaptive game AI, we propose an alternative, novel approach to adaptive game AI that comes without the hampering requirements of typical adaptive game AI. The approach is coined 'rapidly adaptive game AI'. We define rapidly adaptive game AI as an approach to game AI

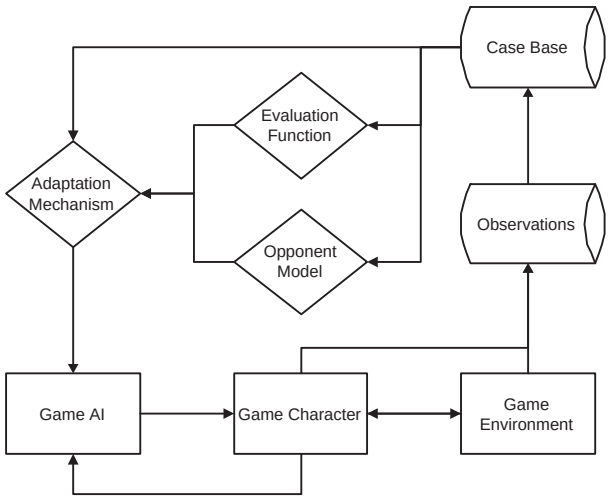


Figure 1: Rapidly adaptive game AI (see text for details).

where domain knowledge is gathered automatically by the game AI, and is immediately (i.e., without trials and without resource-intensive learning) utilised to evoke effective behaviour. The approach, illustrated in Figure 1, implements a direct feedback loop for control of characters operating in the game environment. The behaviour of a game character is determined by the game AI. Each game character feeds the game AI with data on its current situation, and with the observed results of its actions. The game AI adapts by processing the observed results, and generates actions in response to the character's current situation. An adaptation mechanism is incorporated to determine how to best adapt the game AI. For instance, reinforcement learning may be applied to assign rewards and penalties to certain behaviour exhibited by the game AI.

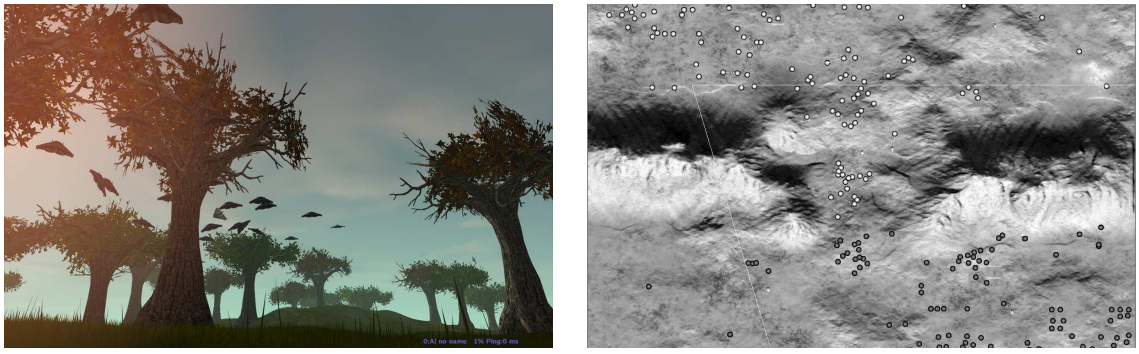
For rapid adaption, the feedback loop is extended by (1) explicitly processing observations from the game AI, and (2) allowing the use of game-environment attributes which are not directly observed by the game character (e.g., observations of team-mates). Inspired by the case-based reasoning paradigm, the approach collects character observations and game environment observations, and extracts from those a case base. The case base contains all observations relevant for the adaptive game AI, without redundancies, time-stamped, and structured in a standard format for rapid access. To rapidly adapt to circumstances in the current game, the adaptation process is based on domain knowledge drawn from observations of a *multitude* of games. The domain knowledge gathered in a case base is typically used to extract models of game behaviour, but can also directly be utilised to adapt the AI to game circumstances. In our proposal of rapidly adaptive game AI, the case base is used to extract an evaluation function and opponent models. Subsequently, the evaluation function and opponent models are incorporated in an adaptation mechanism that directly utilises the gathered cases.

The approach to rapidly adaptive AI is inspired by the human capability to reason reliably on a preferred course of action with only a few observations on the problem domain. Following from the complexity of modern video games, game observations should, for effective and rapid use, (1) be represented in such a way that stored cases can be reused for previously unconsidered situations, and (2) be compactly stored in terms of the amount of retrievable cases [1]. As far as we know, rapidly adaptive game AI has not yet been implemented in an actual video game.

4 Implementation

This section discusses our proposed implementation of rapidly adaptive game AI. In the present research we use Spring [7], illustrated in Figure 2(a), which is a typical and open-source RTS game. In Spring, as in most RTS games, a player needs to gather resources for the construction of units and buildings. The aim of the game is to defeat an enemy army in a real-time battle. A Spring game is won by the player who first destroys the opponent's 'Commander' unit.

We subsequently discuss (1) the evaluation function, (2) the established opponent models, and (3) an adaptation mechanism inspired by the case-based reasoning paradigm.



(a) (b)

Figure 2: Two screenshots of the Spring game environment. In the first screenshot, airplane units are flying over the terrain. In the second screenshot, an overview is presented of two game AI's pitted against each other on the map 'SmallDivide'.

4.1 Evaluation Function

To exhibit behaviour consistent with the game environment presented by modern video games, game AI needs the ability to accurately assess the current situation. This requires an appropriate evaluation function. The high complexity of modern video games makes the task to generate such an evaluation function for game AI a difficult one.

Previous research discussed an approach to automatically generate an evaluation function for game AI in RTS games [3]. The approach incorporates TD-learning [16] to learn unit-type weights for the evaluation function. Our evaluation function for the game's state is denoted by

$$v(p) = w_p v_1 + (1 - w_p) v_2 \tag{1}$$

where $w_p \in [0 \dots 1]$ is a free parameter to determine the weight of each term v_n of the evaluation function, and $p \in \mathbb{N}$ is a parameter that represents the current *phase of the game*. In our experiments, we defined five game phases and used two evaluative terms, the term v_1 that represents the material strength and the term v_2 that represents the Commander safety. Our experimental results showed that just before the game's end, the established evaluation function is able to predict correctly the outcome of the game with an accuracy that approaches one hundred per cent. Additionally, the evaluation function predicts ultimate wins and losses accurately before half of the game is played. From these results, we concluded that the established evaluation function effectively predicts the outcome of a SPRING game. Therefore, we incorporated the established evaluation function in the implementation of our rapidly adaptive game AI.

4.2 Opponent Models

An additional feature of consistent behaviour in game AI is the ability to recognise the strategy of the opponent player. This is known as opponent modeling. In the current experiment, we will not yet incorporate opponent modeling, for first the effectiveness of the adaptation mechanism will be established in dedicated experimentation.

However, previous research already discussed a successful approach for opponent modeling in RTS games [13]. In the approach, a hierarchical opponent model of the opponent's strategy is established. The models are so-called fuzzy models [19] that incorporate the principle of discounted rewards for emphasising recent events more than earlier events. The top-level of the hierarchy can classify the general play style of the opponent. The bottom-level of the hierarchy can classify strategies that further define behavioural characteristics of the opponent.

Experimental results showed that the general play style can accurately be classified by the top-level of the hierarchy. Additionally, experimental results obtained with the bottom-level of the hierarchy showed that in early stages of the game it is difficult to obtain accurate classifications. In later stages of the game, however, the bottom-level of the hierarchy will accurately predict the opponent's specific strategy. From these results, it was concluded that the established approach for opponent modeling in RTS games can be successfully used to classify the strategy of the opponent while the game is still in progress.

4.3 Adaptation Mechanism

In our approach, domain knowledge collected in a case base is utilised for adapting game AI. To generalise over observations with the problem domain, the adaptation mechanism incorporates a means to index collected games, and performs a clustering of observations. For action selection, a similarity matching is performed that considers six experimentally determined features. The adaptation process is algorithmically described below.

```
//Offline processing
A1. Game indexing: to calculate indexes for all stored games.
A2. Clustering of observations: to group together similar observations.

//Online action selection
B1. Use game indexes to select the N most similar games.
B2. Of the selected N games, select the M games that best satisfy the goal criterion.
B3. Of the selected M games, select the most similar observation.
B4. Perform the action stored for the selected observation.
```

Game indexing (A1): We define a game's index as a vector of fitness values, containing one entry for each time step. These fitness values represent the desirability of all observed game states. To calculate the fitness value of an observed game state, we use the previously established evaluation function (denoted in Equation 1). Game indexing is supportive for later action selection, and as it is a computationally-expensive procedure, it is performed offline.

Clustering of observations (A2): As an initial means to cluster similar observations, we apply the standard k-means clustering algorithm [6]. The metric that expresses an observation's position in the cluster space is comprised of a weighted sum of the six observational features that also are applied for similarity matching. Clustering of observations is supportive for later action selection, and as it is a computationally-expensive procedure, it is performed offline.

Similarity matching (A2 and B3): To compare a given observation with another, we define six observational features, namely (1) phase of the game, (2) material strength, (3) commander safety, (4) positional footprint, (5) economical strength, and (6) unit count. Similarity is defined by a weighted sum of the absolute *difference* in features values.[1] As observations are clustered, calculating the similarity between observations is relatively computationally-inexpensive. This is important, as similarity matching must be performed online.

Action selection (B1-B4): Using the established game indexes, we select the N games with the smallest accumulated fitness difference with the current game, up until the current observation. Subsequently, of the selected N games, we perform the game action of the most similar observation of the M games that satisfy a particular goal criterion. The goal criterium can be any metric to represent preferred behaviour. For instance, a preferred fitness value of 0 can represent challenging gameplay, as this implies that players are equally matched. Naturally, we have to consider that performing actions associated to similar observations may not yield the same outcome when applied to the current state. Therefore, to estimate the effect of performing the retrieved game action, we straightforwardly compensate for the difference in metric value between the current and the selected observation.

5 Experiments

This section discusses experiments that test our implementation of rapidly adaptive game AI. We first describe the experimental setup and the performance evaluation, and then the experimental results.

5.1 Experimental Setup

To test our implementation we start collecting observations of games where two game AIs are posed against each other. Multiple SPRING game AIs are available. We found one game AI which was open source, which we labelled 'AAI' [14]. We enhanced this game AI with the ability to collect game observations in a case base, and the ability to disregard radar visibility so that perfect information on the environment was available. As opposing game AI, we used AAI itself. We found 27 parameters that define the strategic behaviour of the game AI.[2] To simulate different players competing with different players, for each game the strategic parameters of both players are pseudo-randomised. The data collection process was as follows. During each game, game observations were collected every 127 game cycles, which corresponds to the update frequency of AAI. With the SPRING game operating at 30 game cycles per second, this resulted in game observations being collected every 4.233 seconds. Of each game observation, the position and unit-type of every unit is abstracted. The games were played on the map 'SmallDivide', illustrated in Figure 2(b), which is a symmetrical map without water areas. All games were played under identical starting conditions. Accordingly, a case base was built from 130552 observations of 200 games, resulting in a total of 392 MB of uncompressed observational data. Note that approaches are available for offline data

[1] The weights for both clustering of observations and similarity matching are as follows: $(1 + phase\_of\_the\_game) * ((0.5 * unit\_count) + material\_strength + commander\_safety + positional\_footprint + economical\_strength)$.

[2] Three examples of these parameters are AIRCRAFT_RATE (determines how many airplane units the AI will build), MAX_MEX_DEFENCE_DISTANCE (maximum distance to base where the AI defends metal extractors), and MAX_SCOUTS (maximum of units scouting at the same time). The authors happily provide a full list of the parameters on request.

	Trial runs	Goal achieved	Goal achieved (%)
Goal(win)	14	12	86%
Goal(lose)	15	12	80%

Table 1: Effectiveness of rapidly adaptive game AI.

	Average	Standard deviation
Time to uphold tie	32 min.	12 min.
Variance in fitness value	0.16	1.95

Table 2: Rapidly adaptive game AI applied for upholding a tie.

compression and subsequent online data decompression [2], but these lie outside the scope of the present research.

For clustering of observations, k is set to ten per cent of the total number of observations. For action selection, the $N = 50$ games with the smallest fitness difference with the current game are selected. Subsequently, the game action of the most similar observation in the $M = 5$ games that best satisfy a defined goal criterion, is selected for direct execution. The game action is expressed by the configuration of the 27 parameters of strategic behaviour. Action selection is performed in the beginning of the game, and at every phase transition.

5.2 Performance Evaluation

To evaluate the performance of the rapidly adaptive game AI, we determine to what extent it is capable of adapting effectively when in competition with the original AAI game AI. We define three goals for adaptation, namely (1) winning the game (positive fitness value), (2) losing the game (negative fitness value), and (3) upholding a tie (fitness value of 0, with a fitness difference of at most 10). To measure how well the rapidly adaptive game AI is able to maintain a fitness value of 0, the variance in fitness value is calculated. A low variance implies that the rapidly adaptive game AI has the ability to consistently maintain a predefined fitness value. All experimental trials are repeated 15 times, except the trial to test the rapidly adaptive game AI, which due to a game-engine crash was repeated 14 times.

5.3 Results

Table 1 and Table 2 give an overview of the results of the experiments performed in the Spring game. Figure 3 displays the obtained fitness value as a function over time of three typical experimental runs. The results reveal that the rapidly adaptive game AI can effectively obtain a victory (86% of the experimental runs), and can effectively lose the game when this is desired (80% of the experimental runs). Subsequently, the results reveal that the rapidly adaptive game AI is capable of upholding a tie for a relatively long time (32 minutes on average), while at the same time maintaining a relatively low variance in the fitness value that is strived for. From these results, we may conclude that rapidly adaptive game AI can be used for effectively adapting game AI in an actual video game.

6 Discussion

In the experiments that test our implementation of rapidly adaptive game AI, we observed that the game AI was not always able to achieve the set goal. A first explanation is that our implementation performs action selection only when a transition in game phase is detected. Though this setup is effective for most games, more moments of action selection may be needed when circumstances are changing rapidly. A second explanation is that our case base, built from 200 games, may still not contain an adequate amount of relevant observations. As rapidly adaptive game AI can be expected to be applied in the playtesting phase of game development, and predictably in multi-player games, the case base in practical applications is expected to grow rapidly to contain a multitude of relevant observations. A last explanation is that eventual outliers cannot be avoided due to the inherent randomness that is typical to video games. For instance, in the Spring game, the most powerful

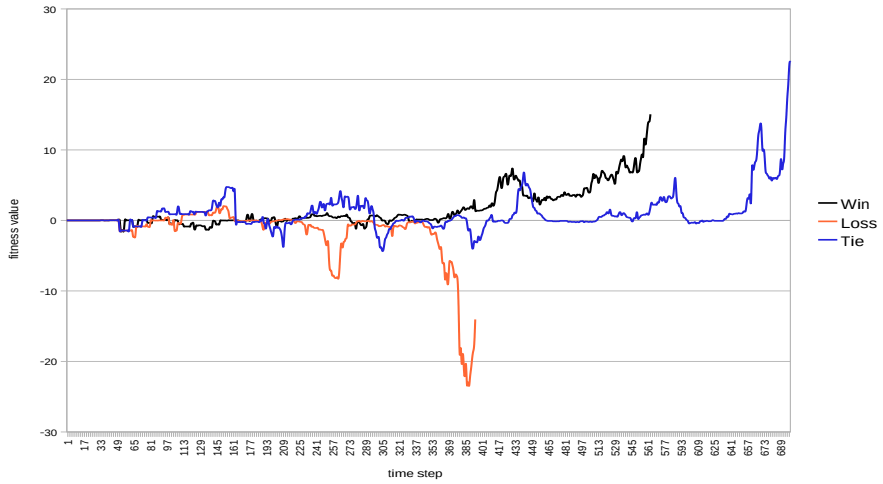


Figure 3: Obtained fitness values as a function over time. The figure displays a typical experimental result of (1) the rapidly adaptive game AI set to win the game, (2) the rapidly adaptive game AI set to lose the game, and (3) the rapidly adaptive game AI set to uphold a tie.

unit is able to destroy a Commander unit with a single shot. Should the Commander be destroyed in such a way, the question would arise if this was due to bad luck, or due to an effective strategy of the opponent. For game AI to be accepted as effective players, one could argue, recalling the previously mentioned need for consistent AI behaviour, that game AI should not force a situation that may be regarded as the result of lucky circumstances.

We found the rapidly adaptive game AI to be able to uphold a tie for a relatively long time, while at the same time maintaining a relatively low variance in the fitness value that is strived for. This ability may be regarded as a straightforward form of difficulty scaling. If a metric can be established that represents the preferred level of challenge for the human player, then in theory the rapidly adaptive game AI would be capable of scaling the difficulty level to the human player. Such a capability provides an interesting challenge for future research.

7 Conclusions and Future Work

In this paper we discussed an approach to establish rapidly-adaptive game AI. In the approach, domain knowledge is gathered automatically by the game AI, and is immediately (i.e., without trials and without resource-intensive learning) utilised to evoke effective behaviour. In our implementation of the approach, game observations are collected in a case base. Subsequently, the case base is used to abstract an evaluation function and opponent models, and gathered cases are directly utilised by an adaptation mechanism. Results of experiments that test the approach in the SPRING game show that rapidly-adaptive game AI can effectively obtain a victory, can effectively lose the game when this is desired, and is capable of upholding a tie for a relatively long time. From these results, we may conclude that the established rapidly adaptive game AI provides a strong basis for effectively adapting game AI in actual video games.

For future work, we will extend the established rapidly-adaptive game AI with a means to scale the difficulty level to the human player. Subsequently, we will investigate if our approach to rapidly adapting game AI can be improved by incorporating opponent models.

Acknowledgements. This research is funded by a grant from the Netherlands Organization for Scientific Research (NWO grant No 612.066.406) and is performed in the framework of the ROLEC project.

References

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1), March 1994.

[2] Samir Abou-Samra, Claude Comair, Robert Champagne, Sun Tjen Fam, Prasanna Ghali, Stephen Lee, Jun Pan, and Xin Li. Data compression/decompression based on pattern and symbol run length encoding for use in a portable handheld video game system. US Patent 6416410, 2002.

[3] Sander Bakkes and Pieter Spronck. *AI Game Programming Wisdom 4*, chapter Automatically Generating Score Functions for Strategy Games, pages 647–658. Charles River Media, Hingham, MA., U.S.A., 2008.

[4] Michael Buro and Timothy M. Furtak. RTS games and real-time AI research. In *Proceedings of the BRIMS Conference*. Arlington VA, 2004.

[5] Pedro Demasi and Adriano J. de O. Cruz. Online coevolution for action games. *International Journal of Intelligent Games and Simulation*, 2(3):80–88, 2002.

[6] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[7] Stefan Johansson, Jelmer Cnossen, and Tomaz Kunaver. Spring game engine. http://spring.clan-sy.com/, 2007.

[8] S Johnson. *AI Game Programming Wisdom 2*, chapter Adaptive AI: A Practical Example, pages 639–647. Charles River Media, Inc., Hingham, MA, 2004.

[9] Ronni Laursen and Daniel Nielsen. Investigating small scale combat situations in real-time-strategy computer games. Master's thesis, Department of computer science, University of Aarhus, Denmark, 2005.

[10] L. Liden. *AI Game Programming Wisdom 2*, chapter Artificial Stupidity: The Art of Making Intentional Mistakes, pages 41–48. Charles River Media, Inc., Hingham, MA, 2004.

[11] Lev Manovich. *The Language of New Media*. The MIT Press, Cambridge, Massachusetts, U.S.A., 2002.

[12] Alexander Nareyek. AI in computer games. *ACM Queue*, 1(10):58–65, 2004.

[13] Frederik Schadd, Sander Bakkes, and Pieter Spronck. Opponent modeling in real-time strategy games. In Marco Roccetti, editor, *Proceedings of the GAME-ON 2007*, pages 61–68, 2007.

[14] Alexander Seizinger. AI:AAI. Creator of the game AI 'AAI', http://spring.clan-sy.com/wiki/AI:AAI, 2006.

[15] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.

[16] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[17] Laurie N. Taylor. Video games: Perspective, point-of-view, and immersion. Masters thesis, Graduate Art School, University of Florida, U.S.A., 2002.

[18] Paul Tozour. *AI Game Programming Wisdom (ed. Rabin, S.)*, chapter The Perils of AI Scripting, pages 541–547. Charles River Media, 2002.

[19] Michael Zarozinski. *AI Game Programming Wisdom*, chapter An Open-Fuzzy Logic Library, pages 90–101. Charles River Media, 2002.