Rapid Adaptation of Video Game AI

Sander Bakkes, Pieter Spronck, and Jaap van den Herik

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.**

I. Introduction

Over the last decades, modern video games have become increasingly realistic with regard to visual and auditory presentation. However, game AI has not reached a high degree of realism yet. Game AI is typically based on non-adaptive techniques [1]. 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, we describe a means of rapid adaptation that is inspired by the human capability to solve problems by generalising over previous observations in a restricted problem domain.

The outline of this paper is as follows. First, we discuss related work in the field of adaptive 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 test 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.

II. Related Work

In this section we discuss related work with regard to (A) entertainment and game AI, and (B) difficulty scaling.

Sander Bakkes, Pieter Spronck and Jaap van den Herik are affiliated to the Tilburg centre for Creative Computing (TiCC), Tilburg University, The Netherlands (phone: +31 13 466 8118; fax: +31 13 466 2892 ; email: {s.bakkes, p.spronck, h.j.vdnherik}@uvt.nl).

A. Entertainment and Game AI

The purpose of a typical video game is to provide entertainment [1], [2]. 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 [3], [4]. 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 [5]. 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 [5].

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 effective 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 [6].

Consistency of computer-controlled characters within 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 [5], causing human players to assume intelligence where none existed [6]. 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 [7]. In practice, game AI in most complex games still is not consistent within the game environment, and exhibits what has been called 'artificial stupidity' [6] rather than artificial intelligence. To increase game consistency, and thus the entertainment value of a video game, in our research we foremost strive to create an optimally playing game AI, as suggested by Buro and Furtak [7]. 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 [5].

B. Difficulty Scaling

Difficulty scaling is the automatic adaptation of a game, to adapt the *challenge* a game poses to the skills of a human player [8]. When applied to game AI, difficulty scaling

usually aims at achieving an "even game", i.e., a game wherein the playing strength of the computer and the human player match.

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. Many researchers and game developers consider game AI, in general, to be entertaining when it is difficult to defeat [9]. Although for strong players that may be true, novice players will not enjoy being overwhelmed by the computer. For novice players, a game is most entertaining when the game is challenging but beatable [10].

The only means of difficulty scaling implemented in many games, is typically provided by a "difficulty setting", i.e., a discrete parameter that determines how difficult the game will be. The purpose of a difficulty setting is to allow both novice and experienced players to enjoy the appropriate challenge the game offers. Usually the parameter influences opponents' strength and health. Very rarely the parameter influences opponents' tactics. Consequently, even on a "hard" difficulty setting, opponents exhibit inferior behaviour, despite their high physical strength and health. In addition, it is hard for the player to estimate reliably the difficulty level that is appropriate for himself. Finally, difficulty settings are discrete, which implies that they cannot possibly be fine-tuned to be appropriate for each player.

In recent years, researchers have developed advanced techniques for difficulty scaling of game AI. Hunicke and Chapman [11] explore difficulty scaling by controlling the game environment (i.e., the number of weapons and power-ups available to a player). Demasi and Cruz [12] use coevolutionary algorithms to gradually teach game characters how to behave. Spronck [8] uses weights assigned to possible game actions, to determine dynamically whether or not predictably strong game actions should be executed.

III. Approach

For game AI to be challenging, as well as 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 [13], [14], 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.

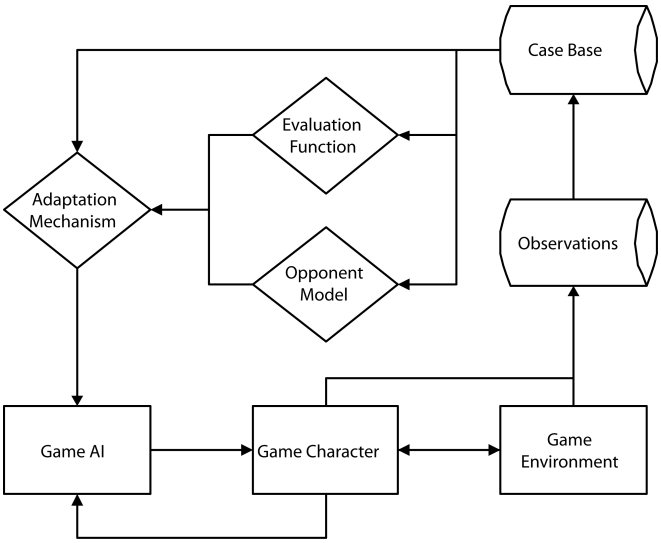

Fig. 1. Rapidly adaptive game AI (see text for details).

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 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

Fig. 2. Screenshot of the SPRING game environment. In the screenshot, airplane units are flying over the terrain.

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 [16]. As far as we know, rapidly adaptive game AI has not yet been implemented in an actual video game.

IV. IMPLEMENTATION

This section discusses our implementation of rapidly adaptive game AI. We subsequently discuss (A) the game environment in which we implement rapidly adaptive game AI, (B) the established evaluation function, and (C) an adaptation mechanism inspired by the case-based reasoning paradigm. Previously established opponent models [17] will be incorporated in future research.

A. Game Environment

The game environment in which we implement rapidly adaptive game AI, is the actual video game SPRING [18]. SPRING, illustrated in Figure 2, is a typical and open-source RTS game, in which a player needs to gather resources for the construction of units and buildings. The aim of the game is to use the constructed units and buildings 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.

Modern RTS games typically progress through several distinct phases as players perform research and create new buildings that provide them with new capabilities. The phase of a game can be straightforwardly derived from the observed traversal through the game's tech tree. A tech tree is a directed graph without cycles that models the possible paths of research a player can take within the game. Traversing the tech tree is (almost) always advantageous, yet there is a cost for doing so in time and game resources. In SPRING, three levels of technology are available. At the start of the game, a player can only construct Level 1 structures and Level 1 units. Later in the game, after the player has performed the required research, advanced structures and units of Level 2 and Level 3 become available.

B. Evaluation Function

To exhibit behaviour consistent within the game environment presented by modern video games, game AI needs the ability to assess accurately 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.

In previous research we discussed an approach to generate automatically an evaluation function for game AI in RTS games [19]. The approach incorporated TD (Temporal Difference) learning [20] to learn unit-type weights for the evaluation function, to reflect the actual playing strength of each unit type. 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*. Our evaluation function incorporates two evaluative terms, the term v_1 that represents the material strength and the term v_2 that represents the Commander safety.

Previous research performed in the SPRING environment has shown that the accuracy of situation assessments are closely related to the phase of the game in which they are made [21]. To distinguish phases of the SPRING game, we map tech levels to game phases and distinguish between when tech levels are "new," and when they are "mature," as indicated by the presence of units with a long construction time. This leads us to define the following five game phases.

- **Phase 1:** Level 1 structures observed.
- **Phase 2:** Level 1 units observed that have a build time $\geq$ 2,500.
- **Phase 3:** Level 2 structures observed.
- **Phase 4:** Level 2 units observed that have a build time $\geq$ 15,000.
- **Phase 5:** Level 3 units or Level 3 structures observed.

Results of experiments to test the established evaluation function 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. Note that this is not a trivial result, for two reasons. First, the evaluation function is tuned to make predictions that are good during a large part of the game, not only at the end, and hence it will trade prediction accuracy at the end of the game for higher prediction accuracy earlier

in the game. Second, if the goal of the game was to destroy all the opponent's units, a correct prediction would be easy to make at the end. However, the goal is to destroy the opponent's Commander, and we found that it sometimes happens that a player who is behind in material strength can win, often because the opponent's Commander makes a high-risk move, such as attacking strong enemy units on its own. An evaluation function that is based on comparison of material strength and Commander safety cannot take such moves into account other than allowing for their general statistical likelihood.

In addition, experimental results showed that 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 and that the proposed approach is suitable for generating evaluation functions for highly complex video games, such as RTS games. Therefore, we incorporate the established evaluation function in the implementation of our rapidly adaptive game AI.

C. 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 an offline means to index collected games, and performs an offline clustering of observations. To ensure that game AI is effective from the onset of a game, it is initialised with a previously observed, successful game strategy. For online action selection, a similarity matching is performed that considers six experimentally determined features. The adaptation mechanism is algorithmically described below, and is subsequently discussed in detail.

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

//Initialisation of game AI
B1. Establish the (most likely) strategy of the
    opponent player.
B2. Determine in which parameter-band values
    this opponent strategy can be abstracted.
B3. Initialise game AI with an effective strategy
    observed against the opponent with the most
    similar parameter-band values.

//Online action selection
C1. Use game indexes to select the N most similar
    games.
C2. Of the selected N games, select the M games
    that best satisfy the goal criterion.
C3. Of the selected M games, select the most
    similar observation.
C4. 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 [22]. 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 C3): 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) positions captured, (5) economical strength, and (6) unit count. Similarity is defined by a weighted sum of the absolute *difference* in features values. The weighted sum for both clustering of observations and similarity matching is calculated as follows:

$$\begin{aligned} similarity &= ((1 + \text{diff}(phase\_of\_the\_game)) \\ &* (0.5 * \text{diff}(unit\_count))) \\ &+ \text{diff}(material\_strength) \\ &+ \text{diff}(commander\_safety) \\ &+ \text{diff}(positions\_captured) \\ &+ \text{diff}(economical\_strength) \end{aligned}$$

As observations are clustered, calculating the similarity between observations is relatively computationally-inexpensive. This is important, as similarity matching must be performed online.

Initialisation of game AI (B1-B3): To intelligently select the strategy initially followed by the game AI, we first determine which strategy the opponent is likely to use. In our game environment, perfect information of the opponent is available when loading the opponent game AI (i.e., the settings of all opponent parameters, indicating the strategic preferences, are known). We use this information to determine the initial strategy of the game AI. In a typical game-playing setting, however, such information is not directly available, but has to be established via statistical learning teachniques such as opponent modeling. This will be investigated in future work. When the opponent strategy has been established, we determine in which parameter bands [23] the opponent strategy can be abstracted. We define three bands for each parameter, 'low', 'medium' and 'high'. We subsequently initialise the game AI with an effective strategy observed against the most similar opponent. We consider a strategy effective when in previous play it achieved a set goal criterion (thus, the game AI will never be initialised with a predictably ineffective strategy), and consider opponents strictly similar when the observed values of the parameter bands are identical.

Action selection (C1-C4): 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.

V. 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.

A. 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 open-source game AI, which the author called 'AAI' [24]. 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 player, we used the original AAI game AI. We found 27 parameters that define the strategic behaviour of the game AI. The concerning parameters determine the game strategy on a high, strategic level, and not so much on a low, tactical level. The parameters are described in the Appendix.

For collecting observation, we simulate different players competing with different players, by for each game pseudo-randomising the strategic parameters of both players. This results in randomly generated strategic variations of predictably reasonable behaviour (and not fully random strategic behaviour). The 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.

We acknowledge that the amount of offline storage should be low for our approach to be considered practical for implementation in a game-production setting. We therefore store game observations in a lightweight fashion, by for each game observation only abstracting the position and unit-type of each unit. This abstraction, of approximately 3 KB per observation, provides a powerful basis for deriving observational features. Accordingly, a case base was built from 213.005 observations of 325 games, resulting in case base consisting of 679 MB of uncompressed observational data. Approaches are available to further reduce the size of the case base, such as offline data compression and subsequent online data decompression [25] and automatic condensation of the case base [26], however these lie outside the scope of the present research.

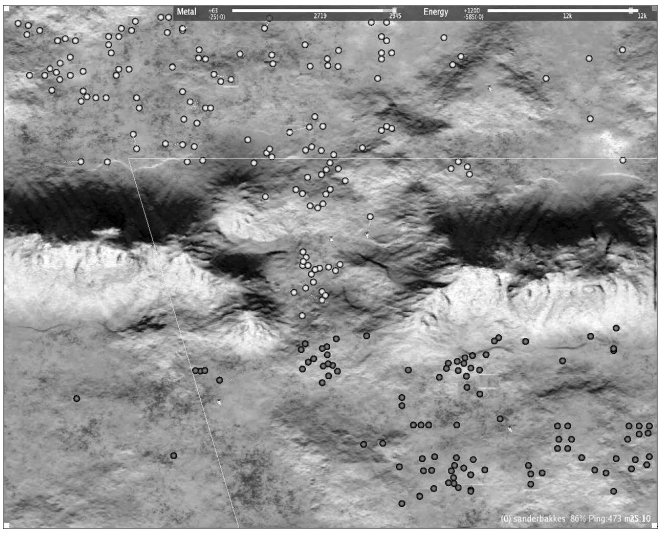

Fig. 3. Two game AI's pitted against each other on the map 'SmallDivide'. In the screenshot, the white player has captured the centre position.

We collect observations from games played on the map 'SmallDivide'. This map is also used for adaptation experiments. The map, illustrated in Figure 3, is a symmetrical map without water areas. All games are played under identical starting conditions.

For offline clustering of observations, k is set to ten per cent of the total number of observations. Before the game starts, the initial strategy is determined. While the game is in progress, action selection is performed at every phase transition. The parameter N for action selection is set to 50, and the parameter M is set to 5. The game action is expressed by the configuration of the 27 parameters of strategic behaviour.

B. Performance Evaluation

To evaluate the performance of the rapidly adaptive game AI, we determine to what extent it is capable of adapting effectively to game circumstances. We perform three different experiments. First, we test to what extent the rapidly adaptive game AI is capable of adapting to the original AAI game AI, set to play in a medium playing strength. Second, we test to what extent the rapidly adaptive game AI is capable of adapting to previously unobserved opponents, which is simulated by pitting the game AI against the original AAI game AI, initialised with randomly generated strategies. Third, as a form of difficulty scaling, we test to what extent the rapidly adaptive game AI is capable of upholding a tie when pitted against the original AAI game AI, also set to play in a medium playing strength.

For each of the first two experiments, we perform a trial where the rapidly adaptive game AI is set to win the game (i.e., obtain a positive fitness value). For the third experiment, we set the rapidly adaptive game AI to uphold a tie (i.e, maintain a fitness value of 0, while never obtaining a fitness

TABLE I

EFFECTIVENESS OF RAPIDLY ADAPTIVE GAME AI.

Opponent	#Games	Goal achv.	Goal achv. (%)	Impr. (%)
Original AAI	25	20	80%	30%
Random	100	58	58%	8%

TABLE II

UPHOLDING A TIE WITH RAPIDLY ADAPTIVE GAME AI.

Opponent	#Games	Time to uphold tie	Variance in fitness
Original AAI	25	37 min. (15 min.)	2.04 (0.59)
Random	100	35 min. (13 min.)	2.00 (0.82)

value less than -10, or greater than 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 25 times, except the trial where the rapidly adaptive game AI is pitted against randomly generated opponents, which is repeated 100 times.

C. Results of Game Adaptation

Table I gives an overview of the results of the first and second experiments performed in the SPRING game. The experiments concerned the adaptation ability of the rapidly adaptive game AI. The results presented in the first row of the table reveal that when pitted against the original AAI game AI, set to play in a medium playing strength, the rapidly adaptive game AI effectively obtains a victory (80% of the experimental runs). This result indicates that rapidly adaptive game AI is effective in play against the original AAI game AI. Figure 4 displays the obtained fitness value as a function over time of two typical experimental runs.

In addition, the results presented in the second row of the concerning table reveal that when pitted against the original AAI game AI, initialised with randomly generated strategies, the rapidly adaptive game AI improves on the effectiveness of strictly randomised games played without adaptation mechanism (i.e., a verified effectiveness of 50%). The obtained improvement in effectiveness is 8%. This result indicates that even in randomised play against the AAI game AI, the rapidly adaptive game AI is able to find effective strategies in the case-base.

D. Results of Difficulty Scaling

Table II gives an overview of the results of the third experiment. The experiment concerned the difficulty scaling ability of the rapidly adaptive game AI. The results presented in this table reveal that when pitted against the original AAI opponent, the rapidly adaptive game AI is capable of upholding a tie for a relatively long time (37 minutes on average), while at the same time maintaining a relatively low variance in the fitness value that is strived for (2.04 on average).

Comparable difficulty scaling results are obtained when the rapidly adaptive game AI pitted against the opponents with randomly generated strategies. The table reveals that when pitted against opponents with randomly generated strategies, the rapidly adaptive game AI is able to uphold a tie for a relatively long time (35 minutes on average), while at the same time maintaining a relatively low variance in the fitness value that is strived for (2.00 on average).

VI. DISCUSSION

In the experiments that test our implementation of rapidly adaptive game AI, we observed that the game AI was well able to achieve a victory when pitted against the original AAI game AI, set to play in a medium playing strength. We noticed that the rapidly adaptive game AI was able to find in the case base a strategy that could effectively defeat the original AAI game AI. As the original AAI game AI is not able to adapt its behaviour, the rapidly adaptive game AI could exploit its discovery indefinitely. Note that in some cases, the rapidly adaptive game AI did not win the game, despite it exhibiting strong behaviour. Such outliers cannot be avoided due to the inherent randomness that is typical to video games. For instance, in the SPRING game, the most powerful 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.

In addition, we observed that even in play with randomised strategic parameter-values, the rapidly adaptive game AI is able to find effective strategies in the case base, and was thereby able to improve on the randomised performance by 8%. This is a satisfactory result. As randomised play may be considered a simulated way to test the game AI against previously unobserved opponents, naturally, the question remains if the performance in randomised play can be further enhanced. We discuss two approaches to enhance the performance in play with randomised strategic parameter-values.

First, note that our case-base currently consists of observations collected over 325 games. For randomised play, determined by 27 pseudo-randomised behavioural parameters, it would be beneficial to collect more games in the case base in order to increase the probability of it containing effective game strategies. 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 effective strategies.

Second, we observed that the final outcome of a SPRING game is largely determined by the actions performed in the beginning of the game. This exemplifies the importance of initialising the game AI with effective behaviour. In order to do so, one needs to accurately determine the opponent one will be pitted against. In video-game practice, (human) game opponents do not exhibit behaviour as random as in

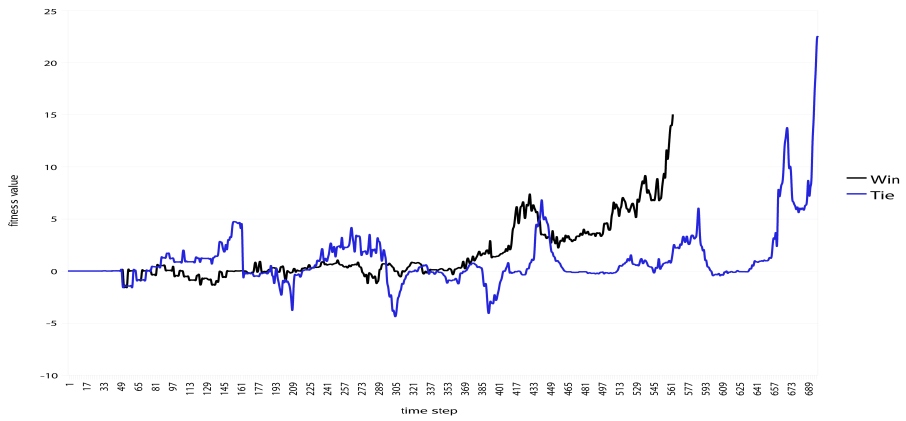

Fig. 4. Obtained fitness values as a function over time, when pitted against the original AAI game AI. The figure displays a typical experimental result of (1) the rapidly adaptive game AI set to win the game, and (2) the rapidly adaptive game AI set to uphold a tie.

our experimental setup, but will typically exhibit behaviour that can be abstracted into a limited set of opponent models. Previous research has shown that even in complex RTS games such as SPRING, accurate models of the opponent player can be established [17]. We will therefore follow the expert opinion that game AI should not so much be focussed on directly exploiting current game observations, but should rather focus on effectively applying models of the opponent in actual game circumstances [27].

In addition, 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.

VII. 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 extract an evaluation function, 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 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 how our approach to rapidly adapting game AI can be improved by incorporating opponent models.

Appendix

In this appendix, we describe the 27 parameters of strategic behaviour that were used in our experiments.

- AIRCRAFT_RATE. Determines how many air units AAI will build (7 means every 7th unit will be an air unit, set to 1 to disable air units).
- AIR_DEFENCE. How often air defence units will be built.
- FAST_UNITS_RATE. Determines the amount of units that will be selected taking their maximum speed into account ($4 \rightarrow 25\%$).
- HIGH_RANGE_UNITS_RATE. Determines the amount of units that will be selected taking weapons range into account ($4 \rightarrow 25\%$).
- MAX_AIR_GROUP_SIZE. Maximum air group size.
- MAX_ANTI_AIR_GROUP_SIZE. Maximum size of anti-air groups (ground, hover or sea).
- MAX_ASSISTANTS. Maximum number of builders assisting construction of other units/buildings.
- MAX_BASE_SIZE. Maximum base size in sectors.
- MAX_BUILDERS. Maximum builders used at the same time
- MAX_BUILDERS_PER_TYPE. How many builders of a certain type may be built.
- MAX_DEFENCES. Maximum number of defences aai will build in a sector.
- MAX_FACTORIES_PER_TYPE. How many factories of a certain type may be built.
- MAX_GROUP_SIZE. Maximum group size; aai will create additional groups if all groups of a certain type are full.
- MAX_METAL_COST. Maximum metal cost, units that cost more metal will not be built.
- MAX_METAL_MAKERS. Maximum number of metal makers, set to 0 if you want to disable usage of metal makers.
- MAX_MEX_DISTANCE. Tells AAI how many sectors away from its main base it is allowed to build mexes.
- MAX_MEX_DEFENCE_DISTANCE. Maximum distance to base where aai defends mexes with cheap defence-buildings.
- MAX_SCOUTS. Maximum scouts used at the same time.
- MAX_STAT_ARTY. Maximum number of stationary artillery (like berthas).
- MAX_STORAGE. Maximum number of storage buildings.

- MIN_AIR_SUPPORT_EFFICIENCY. Minimum efficiency of an enemy unit to call for air support.
- MIN_ASSISTANCE_BUILDSPEED. Minimum workertime / buildpeed of a unit to be taken into account when.
- MIN_FACTORIES_FOR_DEFENCES. AAI will not start to build stationary defences before it has built at least that number of factories.
- MIN_FACTORIES_FOR_STORAGE. AAI will not start to build stationary defences before it has built at least that number of storage buildings.
- MIN_FACTORIES_FOR_RADAR_JAMMER. AAI will not start to build stationary defences before it has built at least that number of radars and jammers.
- MIN_SECTOR_THREAT. The higher the value the earlier AAI will stop to build further defences (if it has not already reached the maximum number of defences per sector).
- UNIT_SPEED_SUBGROUPS. AAI sorts units of the same category (e.g. ground assault units) into different groups according to their max speed (so that slow and fast units are in different groups to prevent the slower ones from arriving in combat much later) this indicates how many different groups will be made (dont set this too high).

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] P. Tozour, *AI Game Programming Wisdom*. Charles River Media, 2002, ch. The Perils of AI Scripting, pp. 541–547.

[2] A. Nareyek, "AI in computer games," *ACM Queue*, vol. 1(10), pp. 58–65, 2004.

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

[4] L. N. Taylor, "Video games: Perspective, point-of-view, and immersion," 2002, masters thesis, Graduate Art School, University of Florida, U.S.A.

[5] R. Laursen and D. Nielsen, "Investigating small scale combat situations in real-time-strategy computer games," 2005, master's thesis, Department of computer science, University of Aarhus, Denmark.

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

[7] M. Buro and T. M. Furtak, "RTS games and real-time AI research," in *Proceedings of the BRIMS Conference*. Arlington VA, 2004, pp. 34–41.

[8] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Difficulty scaling of game AI," in *Proceedings of the GAME-ON 2004: 5th International Conference on Intelligent Games and Simulation*, A. E. Rhalibi and D. V. Welden, Eds., 2004, pp. 33–37.

[9] M. Buro, "RTS games as a test-bed for real-time AI research," in *Proceedings of the 7th Joint Conference on Information Science*. K. Chen et. al., 2003, pp. 481–484.

[10] B. Scott, *AI Game Programming Wisdom*. Charles River Media, Inc., 2002, ch. The Illusion of Intelligence, pp. 16–20.

[11] R. Hunicke and V. Chapman, "AI for dynamic difficulty adjustment in games," in *AAAI Workshop on Challenges in Game Artificial Intelligence*. AAAI Press, 2004, pp. 91–96.

[12] P. Demasi and A. J. de O. Cruz, "Anticipating opponent behaviour using sequential prediction and real-time fuzzy rule learning," in *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAMEON'2004)*, 2004, pp. 101–105.

[13] ——, "Online coevolution for action games," *International Journal of Intelligent Games and Simulation*, vol. 2(3), pp. 80–88, 2002.

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

[15] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, "Adaptive game AI with dynamic scripting," *Machine Learning*, vol. 63(3), pp. 217–248, 2006.

[16] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI Communications*, vol. 7(1), March 1994.

[17] F. Schadd, S. Bakkes, and P. Spronck, "Opponent modeling in real-time strategy games," in *Proceedings of the GAME-ON 2007*, M. Roccetti, Ed., 2007, pp. 61–68.

[18] S. Johansson, J. Cnossen, and T. Kunaver, "Spring game engine," 2007, [Online]. Available: http://spring.clan-sy.com/

[19] S. Bakkes and P. Spronck, *AI Game Programming Wisdom 4*. Charles River Media, Hingham, MA., U.S.A., 2008, ch. Automatically Generating Score Functions for Strategy Games, pp. 647–658.

[20] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988. [Online]. Available: citeseer.ist.psu.edu/sutton88learning.html

[21] S. Bakkes, P. Spronck, and J. van den Herik, "Phase-dependent evaluation in rts games," in *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC)*, M. M. Dastani and E. de Jong, Eds. Universiteit Utrecht, The Netherlands, 2007, pp. 3–10.

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

[23] R. Evans, *AI Game Programming Wisdom*. Charles River Media, 2002, ch. Varieties of Learning, pp. 571–575.

[24] A. Seizinger, "AI:AAI," 2006, creator of the game AI 'AAI'. [Online]. Available: http://spring.clan-sy.com/wiki/AI:AAI

[25] S. Abou-Samra, C. Comair, R. Champagne, S. T. Fam, P. Ghali, S. Lee, J. Pan, and X. Li, "Data compression/decompression based on pattern and symbol run length encoding for use in a portable handheld video game system," 2002, US Patent 6416410.

[26] F. Angiulli and G. Folino, "Distributed nearest neighbor-based condensation of very large data sets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19(12), pp. 1593–1606, 2007.

[27] S. Rabin, *AI Game Programming Wisdom 4*. Charles River Media, Inc., 2008, ch. Preface - What happened to learning?, pp. ix – xi, ISBN 1-584-505230.