OPPONENT MODELING IN REAL-TIME STRATEGY GAMES

Frederik Schadd, Sander Bakkes and Pieter Spronck
Universiteit Maastricht
MICC-IKAT
P.O. Box 616
NL-6200 MD Maastricht
The Netherlands
e-mail: f.schadd@student.unimaas.nl, {s.bakkes,p.spronck}@micc.unimaas.nl

ABSTRACT

Real-time strategy games present an environment in which game AI is expected to behave realistically. One feature of realistic behaviour in game AI is the ability to recognise the strategy of the opponent player. This is known as opponent modeling. In this paper, we propose an approach of opponent modeling based on hierarchically structured models. The top-level of the hierarchy can classify the general play style of the opponent. The bottom-level of the hierarchy can classify specific strategies that further define the opponent's behaviour. Experiments that test the approach are performed in the RTS game SPRING. From our results we may conclude that the approach can be successfully used to classify the strategy of an opponent in the SPRING game.

INTRODUCTION

In computer gaming, real-time strategy (RTS) is a genre of simulated wargames which take place in real time. In RTS games, the player needs to construct a base and build units for the purpose of destroying the opponent. The opponent is either a human player, or a player controlled by an artificial intelligence (AI). Each unit-type has particular strengths and weaknesses. To effectively play an RTS game, the player has to utilise the right units in the right circumstances.

An important factor that influences the choice of strategy, is the strategy of the opponent. For instance, if one knows what types of units the opponent has, then typically one would choose to build units that are strong against those of the opponent. To make predictions about the opponent's strategy, an AI player can establish an opponent model. Many researchers point out the importance of modelling the opponent's strategy [2, 3, 9, 10, 12, 14], and state that opponent models are sorely needed to deal with the complexities of state-of-the-art video games [8].

Establishing effective opponent models in RTS games, however, is a particular challenge because of the lack of perfect information of the game environment. In classical board games the entire board is visible to the player; a player can observe all the actions of the opponent. Hence, assessing the opponent's strategy and building an opponent model is possible in principle, for instance by using case-based reasoning techniques [1]. In RTS games, however, the player has to deal with imperfect information [5]. Typically, the player can only observe the game map within a certain visibility range of its own units. This renders constructing opponent models in an RTS game a difficult task. In this paper we will investigate to what extent models of the opponent's strategy can be established in an imperfect-information RTS-game environment.

The outline of this paper is as follows. We will first introduce the concept of opponent modeling. Then, our approach to establish effective opponent models in RTS games will be discussed. Subsequently, our implementation of the approach will be presented. The experiments that test our approach are described next, followed by a discussion of the experimental results. Finally, we provide conclusions and describe future work.

OPPONENT MODELING

In general, an opponent model is an abstracted description of a player or a player's behaviour in a game [8]. Opponent modeling can be seen as a classification problem, where data that is collected during the game is classified as one of the available opponent models. A limiting condition is the fact that in RTS games, these classifications have to be performed in real-time, while many other computations, such as rendering the game graphics, have to be performed in parallel. This limits the amount of available computing resources, which is why only computationally-inexpensive techniques are suitable for opponent modeling in RTS games.

Preference-based modeling is a commonly used computationally-inexpensive technique [4]. The technique identifies the model of an opponent by analyzing the opponent's choices in important game states. Due to the visibility limitations in RTS games, however, it is common that choices of the opponent cannot always be observed.

In the present research we use SPRING, illustrated in

Figure 1: Screenshot of the Spring game environment. In the screenshot, airplane units are flying over the terrain.

Figure 1, which is a typical and open-source RTS game. A Spring game is won by the player who first destroys the opponent's 'Commander' unit. We used the freely-available 'AAI' artificial intelligence player [15] to combat opponents.

APPROACH

A straightforward approach to opponent modeling is the following. First, a number of possible opponent models is established, then the confidence level of each opponent model is calculated, and finally, the opponent model with the highest confidence level is selected. An enhancement of this approach is to apply an hierarchical ordering on the possible opponent models [7]. This *hierarchical approach* allows the division of a relatively complex classification problem, into several relatively simple classification problems. In addition, the hierarchical approach makes it possible to use different classification methods in each level or the hierarchy. For establishing opponent modeling in RTS games, we follow the hierarchical approach.

Our opponent models will describe the strategy of a player. We define a strategy as *the general play style combined with the player's choice of units built.* The most defining element of an opponent's strategy is the general play style. We therefore place the general play style at the top of the hierarchy. Each play style has its own subcategories that further define behavioural characteristics.

For instance, if it is known that the opponent follows an aggressive general play style, a logical response would be to improve one's defenses. If also the opponent's choice of units is known, the defenses can be specialised to be effective against those specific units.

IMPLEMENTATION

This section discusses our implementation of hierarchical opponent modeling in RTS games. In our implementation, we establish a hierarchy consisting of two levels. The top-level of the hierarchy classifies the opponent's general play style. The bottom-level of the hierarchy classifies the opponent's choice of units built.
In the Spring game we discriminate between an aggressive, and a defensive play style. For an aggressive play style we discriminate at the bottom level between predominantly using the following four unit-types: (1) K-Bots, (2) Tanks, (3) Ships, and (4) Airplanes. Each unit-type has specific strengths and weaknesses, and is therefore used to execute a particular strategy. For instance, K-Bots are relatively fragile but can cross mountains, and are therefore useful for a strategy against an opponent which attempts to exploit chokepoints between mountains. Tanks can only manoeuvre on plain terrain but are relatively sturdy, and are therefore useful for a strategy against an opponent who constructs strong defenses.
For a defensive play style we discriminate at the bottom level between the following three building preferences: (1) Super Weapon, (2) Tech, and (3) Bunker. These three building preferences are commonly observed in actual Spring games.
Figure 2 displays the hierarchy of the opponent models. The hierarchy defines the following strategies.

- Aggressive→K-Bot. The opponent will attack early and will typically use K-Bot robots.
- Aggressive→Tanks. The opponent will attack early and will typically use tanks.
- Aggressive→Air. The opponent will attack early and will typically use airplanes.

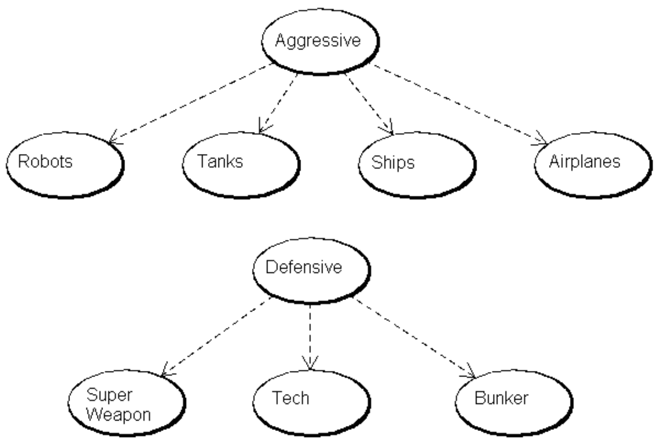

Figure 2: Hierarchy of the opponent models.

- Aggressive→Sea. The opponent will attack early and will typically use ships.

- Defensive→Super weapon. The opponent will attempt to construct a super weapon (e.g., a ballistic missile).

- Defensive→Tech. The opponent will attempt to reach a high technology level in order to have quick access to superior units.

- Defensive→Bunker. The opponent will construct a massive wall of static defenses around his base so that he has time to construct an army.

Top-level Classifier

A way of performing opponent-model classifications in a computationally inexpensive fashion, is by using fuzzy models [16]. Fuzzy models create models of several classifiable classes based on a single numerical feature. The choice of the numerical feature is crucial, as it should allow to discriminate between the defined classes.

In our hierarchy, the top level classifier has to discriminate between the classes 'aggressive' and 'defensive'. An aggressive player typically will spend a large part of game time attacking. A defensive player, on the other hand, typically will use most of the game time for preparing an army, and only needs a small amount of the game time for an actual attack. As a result, an appropriate numerical feature to discriminate between the two classes would be the relative amount of game time that the opponent spends attacking.

We define an attack as the observed loss of a player's own units. When a loss of units is detected, then the time span around this moment can be regarded as the time that an attack took place. Because information about a player's own units is always available, this definition is suitable for use in an imperfect information environment.

An example of a top-level player model is shown in Figure 3. The figure illustrates the confidence of the opponent following an aggressive and defensive play style, as a function of the percentage of game time spent on attacking.

Bottom-level Classifier

The bottom-level classifier has to discriminate between the subcategories that further define behavioural characteristics. The dynamic nature of RTS games implies that a player's strategy may change during the game. Therefore, the bottom-level classifier needs to emphasize recent event more than past events. To achieve this, the principle of discounted rewards is applied.

Theoretical Background

The concept of discounted rewards origins from the principle of repeated matrix-games [6]. A player can make a choice between several actions and depending on the effectiveness of an action, a reward is received. In the field of repeated matrix-games, the most important considerations are how to select the initial strategy and how to evaluate if a deviation from the initial strategy is needed. A strategy in repeated matrix-games can be a simple sequence of actions, which is played repeatedly. It can also include certain rules for cases that take the actions of the enemy into consideration. In order to determine whether a deviation from the original strategy is feasible, the expected rewards of the game with and without deviation are calculated. Here the discount factor is applied, since it is assumed that a player prefers to receive a reward early in time rather than in the distant future. Hence, rewards in the future are valued less. This valuation is expressed by multiplying the future reward with the discount factor. The more distant the reward is, the more often it is multiplied with the discount factor. If δ is the discount factor and π_t is the reward received at time t, then the expected reward can be computed as follows [6]:

$$(1-\delta) * \sum_{i=1}^{\infty} \pi_i * \delta^{i-1} \tag{1}$$

When the expected rewards are calculated, a selection mechanism typically will select the strategy with the highest expected reward.

Applied Modification

Analogously to calculating the expected rewards, for discriminating between strategies we need to calculate the confidence value of the opponent applying a particular strategy. The confidence value is calculated on the basis of observations during game events. In classical matrix-games, a game event would be one move of both players.

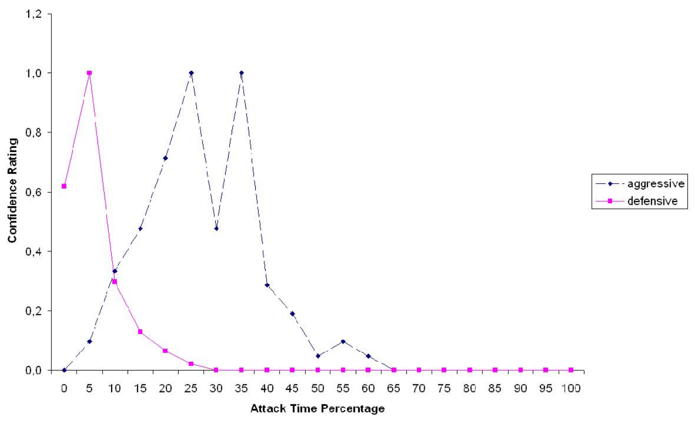

Figure 3: Example of a top-level player model.

Since RTS games do not operate in moves, the term of an event must be redefined. When playing against an aggressive opponent, the occurrence of an attack is a suitable event since the player can then observe the army of the opponent.
When playing against a defensive opponent, however, an attack is not a suitable event because a defensive opponent will rarely attack and hence not many observations can be made. Also, waiting for an attack of a defensive opponent is typically an unsuccessful game strategy. When playing against a defensive opponent, it is typical that the player will 'scout' around the base of the opponent. Since scouting will provide improved information about the opponent's actions, a scout event is a suitable moment for calculating confidence values against a defensive opponent.
When an event is detected, the confidence values of each possible strategy will be updated according to the observed game information. If δ is the discount factor, $\psi_{s,t}$ the belief that the opponent uses strategy s at event t, ranging between 0 and 1, π the total reward added at each event and i the most recent event, then the confidence c_s that the opponent uses strategy s is computed as follows:

$$c_s = \sum_{t=i}^{0} \psi_{s,t} * \pi * \delta^{i-t} \qquad (2)$$

The parameter $\psi_{s,t}$ is acquired by inspecting all visible units and structures during event t. Each unit or structure has a value representing a tendency to a certain strategy. The unit-tendency values were determined by the experimenter, using his own knowledge of the game [13]. To give three examples of unit-tendency values: (1) a common defensive tower has a relatively small tendency towards an opponent using the defensive bunkering strategy, (2) a super-weapon building has a relatively high tendency towards the defensive-super-weapon strategy, and (3) an amphibious tank has a tendency towards both the aggressive tank and the aggressive sea strategy.

EXPERIMENTS

This section discusses the experiments that test our approach. First we test the top-level classifier, and then the bottom-level classifier. We finish the section by providing a summary of the experimental results.

Top-level Experiment

As discussed in the previous section, the top-level classifier requires the AI to detect if an attack took place. This is implemented as follows. The AI will register all visible units every N seconds, where N is the size of the time window. The detection algorithm will scan each frame for lost units. If the amount of lost units is above a certain threshold, then each frame inside the analysed time window is labeled as a moment in which the opponent was attacking.
Two parameters determine the accuracy of the attack-detection algorithm. The first parameter is the size of the time window. The second parameter is the unit threshold, which is a percentage value. We will first analyse the sensitivity of the parameter settings on the obtained confidence values. Next, we will discuss the obtained confidence values as a function over the game time.

	Time Window		
Threshold	20 seconds	30 seconds	40 seconds
10%	94,92166	99,2271	93,64321
15%	99,99083	99,57988	96,22472

Table 1: Average defensive confidence values against a defensive opponent

	Time Window		
Threshold	20 seconds	30 seconds	40 seconds
10%	45.10261	76.27533	57.83488
15%	33.18209	52.74819	53.14834

Table 2: Average aggressive confidence values against an aggressive opponent

Sensitivity Analysis
For each configuration of parameters, the AI was matched ten times against an aggressively playing AI and ten times against a defensively playing AI. As opponent the 'NTAI'-AI [11] was chosen, since it is relatively straightforward to implement one's own strategy into this AI. For the matches where an aggressive opponent is required, the 'NTAI'-AI was configured with an aggressive strategy. Analogously, for the matches where a defensive opponent is required, the AI was configured with a defensive strategy. For the time window, the sizes of 20, 30 and 40 seconds were tested. For the unit threshold the percentages with value 10% and 15% were tested.
For each match played, the aggressive and defensive confidence values were recorded at each time point. When the match was played, the average confidence values of the match were calculated. Note that as in the first five minutes of a game a player rarely performs an action that would reveal his strategy, these first minutes were not taken into account.
Table 1 shows the defensive confidence values of matches against an defensive opponent, and Table 2 shows the aggressive confidence values of matches against an aggressive opponent.
The obtained confidence values reveal that all configurations of the top-level classifier perform well when

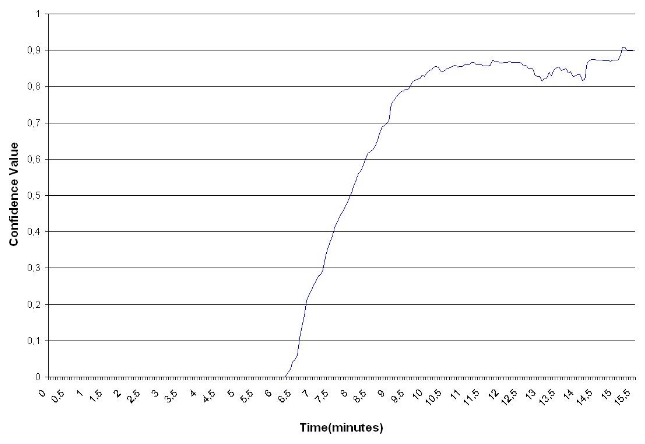

Figure 4: Average confidence value over time against an aggressive opponent

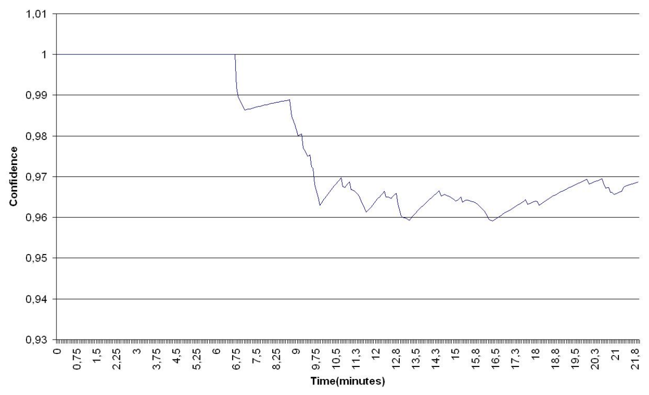

Figure 5: Average confidence value over time against a defensive opponent

recognizing defensive players. The best configuration obtained a confidence value of nearly 100%. When recognizing aggressive players, the obtained confidence values are lower in comparison, as will be elaborated upon shortly. The best configuration, with as parameters a units-lost threshold of 10% and a time window of 30 seconds, obtained a confidence value of 76%. These obtained configurations will be used for the remainder of this research.

Confidence Value over Time

Using the obtained parameter values, we match the AI fifty times against an aggressive opponent, and fifty times against a defensive opponent. We again used the 'NTAI'-AI as an opponent. For each game, we record the confidence values as a function over time. The average confidence values of all test games against an aggressive opponent are displayed in Figure 4.

In the figure we observe that the average confidence value is low in the beginning of the game. This is due to the fact that the opponent is hardly able to attack at this stage of the game, since he needs to construct a base first. Therefore, one can safely disregard the confidence values of the beginning of the game. After approximately seven minutes of game time, the average confidence value increases until it stabilizes at approximately 85%.

A similar effect can be observed when examining the average confidence value over time of the games against a defensive opponent, as displayed in Figure 5. In the beginning of the game, the confidence values are nearly 100%. This is because the enemy does not attack in the beginning of the game. The top-level classifier will therefore respond with the maximum defensive confidence value during this game stage. One observes that after about six minutes of game time, the average confidence value stabilizes between 96% and 97%.

Bottom-level Experiment

For testing the bottom-level classifier, the 'NTAI'-AI has been configured such that it resembles each of the specific aggressive opponent models. However, since NTAI is not able to adequately play the defensive strategies, a human opponent was chosen to play against the 'AAI'-AI [15], by following the defensive strategies. For each opponent model, ten experimental matches have been performed. A correct classification of the top-level classifier is assumed for this experiment. For the discounted reward algorithm, the parameters were set to $\delta = 20\%$ and $\pi = 0.8$ by the experimenter. We test the bottom-level classifier's ability to discriminate between the K-Bot and tanks aggressive sub-model, and between the bunker, tech and super weapon defensive sub-model.

Aggressive Opponent

Figure 6 shows the average K-Bot confidence over time of an opponent using the aggressive K-Bot strategy as well as the average Tank confidence over time of an opponent using the aggressive tank strategy. It is observed that both confidence values eventually approximate a value over 90%. We note that the average confidence of the aggressive tank-strategy increases more slowly and at a later stage than the average confidence of the aggressive K-Bot-strategy. This can be explained by the fact that producing tanks requires more resources, and therefore more game time will be needed to attack with tanks.

Defensive Opponent - Bunker

Figure 7 displays the confidence values obtained against an opponent using the defensive bunker-strategy. We observe that the bunker confidence rating increases rapidly after approximately five minutes of game-time. Over time the obtained average confidence value is 83%. The instabilities that occur after 35 minutes of game-time can be explained by the fact that at this moment the AI has discovered structures that may also be used

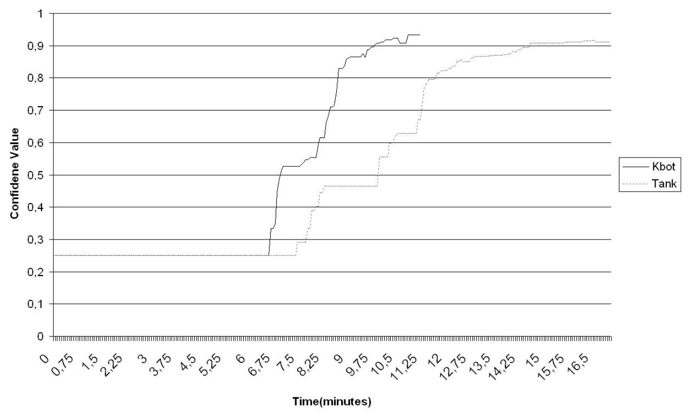

Figure 6: Average confidence value over time for the aggressive sub-models.

by an opponent using the tech-strategy.

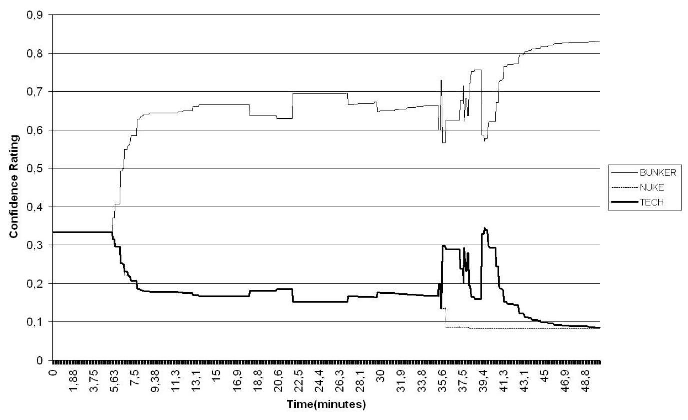

Figure 7: Average confidence value over time for an opponent using the defensive bunker-strategy.

Defensive Opponent - Tech

Figure 8 displays the confidence values obtained against an opponent using the defensive tech-strategy. We observe that for the largest part of the game, the confidence values of the bunker-strategy are higher than the confidence values of the tech-strategy. This can be explained as follows. First, we observed that scout units were destroyed before they were able to scout the base of the opponent. Second, high level units and structures that define the tech-strategy can only be constructed in later stages of the game. This implies that in the earlier stages only structures that belong to a different strategy can be observed. When at a later stage of the game the AI is able to observe structures that are characteristic for the tech-strategy, the confidence value of the tech-strategy increases.

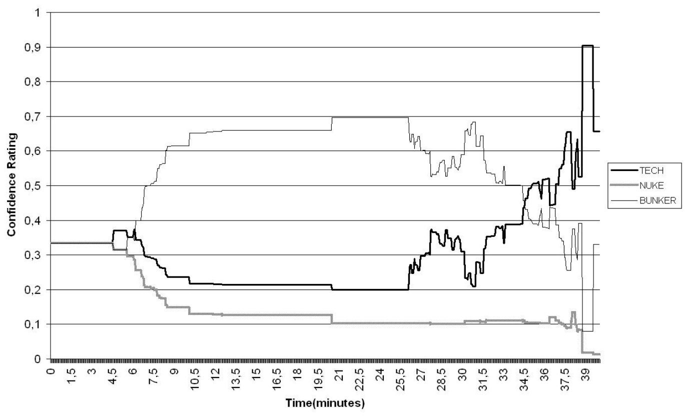

Figure 8: Average confidence value over time for an opponent using the defensive tech-strategy.

Defensive Opponent - Super Weapon

Figure 9 displays the confidence values obtained against an opponent using the defensive super-weapon-strategy. Analogously to the results obtained against an opponent using the defensive tech-strategy, we observe that the confidence values of the bunker-strategy are higher than the confidence values of the super-weapon-strategy. After approximately 25 minutes the confidence value of super-weapon-strategy steadily increases. After approximately 41 minutes, the discounted-reward algorithm temporarily decreased the confidence value because the AI did not observe super-weapon structures any more. Eventually the bottom-level classifier obtains a confidence value of 75%.

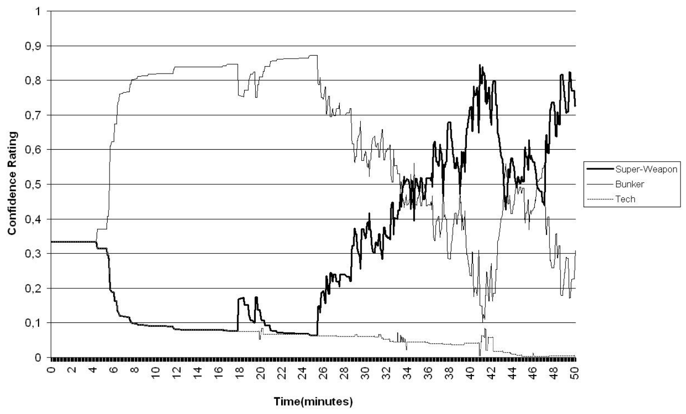

Figure 9: Average confidence value over time for an opponent using the defensive super-weapon-strategy.

Summary of the Experimental Results

Experimental results obtained with the top-level classifier show that the top-level classifier can accurately discriminate between an aggressive and a defensive player. Experimental results obtained with the bottom-level classifier show that the bottom-level classifier can accurately discriminate between the established sub-models in later stages of the game. In early stages of the game, the bottom-level classifier was not always able to accurately discriminate between the established sub-models. This is discussed next.

DISCUSSION

Opponent modeling will typically be implemented in an actual RTS game for the purpose of automatically classifying the strategy of the opponent. Ideally, an accurate classification of the opponent's strategy is available relatively early in the game, at a time when the player is still able to counter the opponent's strategy.
In the experiments that test our approach, we observed that the bottom-level classifier was not always able to accurately discriminate between the established submodels in an early stage of the game. This phenomenon can be explained by the fact that, typically, the AI cannot directly observe units and structures that are characteristic for a particular bottom-level strategy. To observe these units and structures, the AI relies on scouting.
A straightforward approach to achieve improved results, therefore, is to adapt the AI's scouting behaviour dependent on the need for information of the opponent's activities. For instance, in competition against an aggressive opponent, scouting is relatively unimportant. In competition against a defensive opponent, however, intensive scouting is vital. Analogously, to emphasise the information obtained during a scout event, one may choose to adapt the parameters of the delayed reward algorithm dependent on the top-level classification.

CONCLUSION

In this paper we proposed an approach for opponent modeling in RTS games. In the approach, a hierarchical opponent model of the opponent's strategy is established. 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. Experiments to test the approach were performing in the RTS game Spring. Our experimental results show 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 show 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 classify between specific strategies of the opponent. From these results, we may conclude that the approach for opponent modeling in RTS can be successfully used to classify the strategy of the opponent while the game is still in progress.
For future work, we will incorporate a mechanism to adapt the scouting behaviour of the AI dependent on the top-level classification of the general play style of the opponent. Subsequently, we will investigate to what extent the classification of the opponent's strategy can be used to improve the performance of a computer-controlled player, and will investigate how new models to describe the opponent's strategy can be established automatically.

ACKNOWLEDGEMENTS

The authors wish to extend their gratitude to Marcel Ludwig and Alexander Miesen for being the human opponent in the experimental games. Furthermore, we extend our gratitude to the anonymous reviewers for their insightful feedback.
This research was funded by a grant from the Netherlands Organization for Scientific Research (NWO grant No 612.066.406).

REFERENCES

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

[2] Bruce Abramson. Expected outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:182–193, 1990.

[3] Hans Berliner. Search and knowledge. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 77)*, pages 975–979, 1977.

[4] Jeroen Donkers and Pieter Spronck. Preference-based player modeling. In S. Rabin, editor, *AI Programming Wisdom 3*, chapter 8.4, pages 647–659. Charles River Media, 25 Thomson Place, Boston, Massachusetts 02210, 2006.

[5] Robert Gibbons. *A Primer in Game Theory*, chapter 2.3B. Pearson Education Limited, Edingburgh Gate, Harlow, Essex CM20 2JE, England, 1992.

[6] Robert Gibbons. *A Primer in Game Theory*, chapter 2. Pearson Education Limited, Edingburgh Gate, Harlow, Essex CM20 2JE, England, 1992.

[7] Ryan Houlete. Player modeling for adaptive games. In S. Rabin, editor, *AI Programming Wisdom 2*, chapter 10.1, pages 557–566. Charles River Media, 10 Downer Avenue, Hingham, Massachusetts 02043, 2004.

[8] Jaap van den Herik, Jeroen Donkers, and Pieter Spronck. Opponent modelling and commercial

games. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (eds. Graham Kendall and Simon Lucas)*, pages 15–25, 2005.

[9] Donald Knuth and Ronald Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[10] Richard Korf. Generalized game trees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 89)*, pages 328–333. Detroit, MI, August 1989.

[11] Tom Nowell. AI:NTAI. Creator of the game AI 'NTAI', http://spring.clan-sy.com/wiki/AI:NTAI, 2007.

[12] Arthur Samuel. Some studies in machine learning using the game of chechers, ii - recent progres. *IBM Journal*, 11:601–617, 1967.

[13] Frederik Schadd. Hierarchical opponent models for real-time strategy games. Bachelor thesis. Universiteit Maastricht, The Netherlands, 2007.

[14] Jonathan Schaeffer, Joseph Culberson, Norman Treloar, Brent Knight, Paul Lu, and Duane Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53:273–289, 1992.

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

[16] Michael Zarozinski. An open-fuzzy logic library. In S. Rabin, editor, *AI Programming Wisdom*, chapter 2.8, pages 90–101. Charles River Media, 20 Downer Avenue, Suite 3, Hingham, Massachusetts 02043, 2002.