Hierarchical Opponent Models for Real-Time Strategy Games

F.C. Schadd

June 13, 2007

Abstract

Real-time strategy games compose a large part of modern game industry. Simulating human behavior is a crucial aspect for the artificial opponents of these games, in order to provide a good experience for the user. To achieve this, the AI should ideally be able to recognize the strategy of the human player and adapt to it. This concept is known as opponent modeling. This paper describes an approach of opponent modeling using hierarchical structured models. As classifiers, two different approaches have been used for the different hierarchical levels, in order to test the effectiveness of this approach. The first classifier uses fuzzy models whereas the second classifier is a modification of the concept of discounted rewards from game theory. It is observed that the fuzzy classifier shows a stable convergence to a correct classification with high confidence rating. The experiments regarding the discounted reward approach revealed, that for some sub-models this approach showed a similar well conversion as the fuzzy classifier. However, for other sub-models only mediocre results were achieved with a late and unstable convergence. We conclude that this approach is suitable for real-time strategy games and can be reliable for each sub-model with further research.

1 Introduction

Computer games are a popular branch of software development. One part of this branch consists of real-time strategy (RTS) games. In these kind of games, the player needs to construct a base and build units for the purpose of destroying the opponent. The other player can be also human or an AI. The goal of the game is to destroy the base of the opponent. All units have strengths and weaknesses, so it is crucial to choose the right units in order to have better chances of winning. Since the chosen type of units is very important, we can define a strategy as the chosen type of built units combined with how the player operates these units.

An important factor that influences the choice of strategy is the strategy of the opponent. If one knows what types of units the opponent has, then typically one would choose to build units that are strong versus those from the opponent. A method of representing information of the enemy is known as opponent modeling. A model of the opponent is a generalization of his strategy. While strategies can have many small variances, they can be generalized into a small amount of representative strategies, which are referred to as player models. These player models are constructed and matched against the in-game data of the opponent. The model that has the largest resemblance of the behavior of the opponent is chosen as the actual model of the opponent. The AI can now develop a counter-strategy based on the confidence levels of the various player models.

A problem with creating opponent models in RTS games is the amount of available information. In classical board games the entire board is visible to every player. Thus a player can always observe the actions of his opponent and predict his strategy. RTS games, however, only provide imperfect information [2] to the player. Though it is common to have complete knowledge about the structure of the battlefield, a player can only observe that battlefield within a certain visibility range of its own units. If the units of the opponent are outside this range, then these units are not visible to the player. Hence constructing opponent models in an imperfect information RTS game is a difficult task.

This paper examines the following research question:

To what extent can opponent modeling be applied in order to predict player strategies for modern real-time strategy games with imperfect information?

The outline of this paper is as follows. In section 2 the principle of hierarchical opponent models and the applied classifiers for the different hierarchical levels will explained. Section 3 presents the performed experiments and their results. Section 4 discusses the strengths and weaknesses of the chosen approach and possible improvements. The conclusions are presented in section 5.

1.1 Background

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. Therefore it is possible to apply standard machine-learning techniques such as Neural Networks. However, a limiting condition is the fact that these calculations have to be performed in real-time, while many other computations, like the rendering of the game graphics,

have to be performed simultaniously. This limits the amount of available computing resources, which is why many machine learning techniques are not suitable for this problem.
A basic issue with opponent modeling is the structure of the models. An initial idea of a structuring would be enlisting all the possible models and calculate all the confidence levels. However, a hierarchical way of ordering [4] is also possible. This approach is used within this research and details about this will be explained later on.
Another way of matching in-game data to models is by a preference based approach [1]. Such an approach identifies the model of a player by analyzing the player's choices in important game states, which can yield a lot of information about the opponent. However, in a real-time strategy game it is hard to define a state since the game field can be different each time the game is played. Also, due to the visibility limitations in RTS games, it is possible, that the reaction of a player in a certain state cannot be observed.
As experimental game, the game Total Annihilation: Spring has been chosen due to several reasons. Firstly since it contains the key elements of every RTS game: base construction with base defenses, land, sea, air units and super weapons. Secondly it has been chosen since it is completely open source, which makes implementing additional features very convenient. A screenshot displaying a typical game situation is shown in figure 1. In order to collect typical in-game information, an AI basis was needed. For this purpose, the 'AAI'-AI [6] has been used.

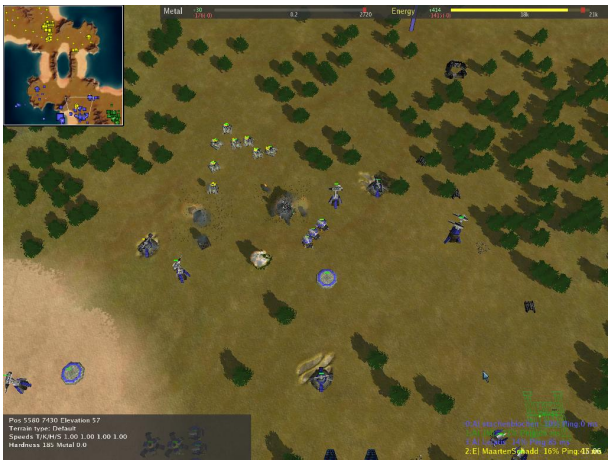

Figure 1: Screenshot of the RTS game Total Annihilation: Spring.

2 Approach

This section explains the applied concept of hierarchical opponent models and the specific implementations. In section 2.1 the general concept of hierarchical opponent models will be explained. Section 2.2 describes the used approach of the top level classifier and section 2.3 explains the applied bottom level classifier.

2.1 General Concept

The first decision that needed to be made was how to structure the models. Here a hierarchical ordering [4] was chosen, as this allowed the division of the general classification problem into sub-problems. Additionally it allows the usage of various and more appropriate classification methods in the different hierarchical levels.
In order to define the models of the opponent one must first define what a strategy is. An appropriate definition of a strategy for RTS games is: 'A strategy is the combination of the general play style of the player combined with his choice of built units'. With this definition in mind, it is logical to place the general play styles at the top of the hierarchy since this is the most crucial attitude of an opponent. For example, if one knows that an opponent has shown aggressive behavior, then it is logical to improve ones own base defenses as it relatively unimportant with which units exactly he will attack. The type of units, that the opponent uses in order to attack will only influence the type of defenses which is to be constructed, but will not influence the choice of constructing defenses itself. The general play styles here are 'aggressive' and 'defensive', where each style has its own subcategories. Figure 2 shows the chosen hierarchy of the player models. We will now provide a short description for each of the different considered strategies:

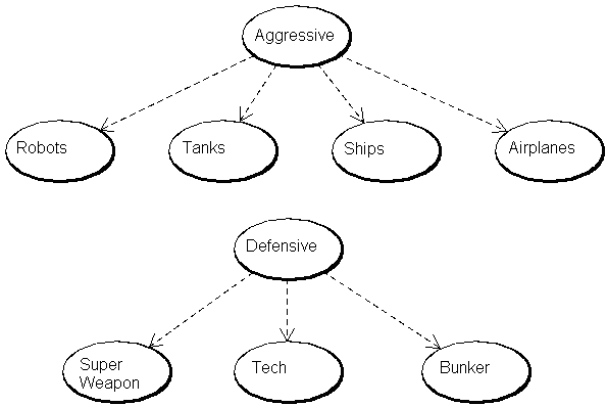

Figure 2: Hierarchy of the considered player models

- Aggressive→Robots - The opponent will attack early and often using robots, also referred to as K-Bots
- Aggressive→Tanks - The opponent will attack early and often using tanks
- Aggressive→Air - The opponent will attack early and often using airplanes

- Aggressive→Sea - The opponent will attack early and often using ships
- Defensive→Super weapon - The opponent will attempt to construct a super weapon (such as an atomic bomb)
- 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 constructs a massive wall of defensive towers around his base so that he has time to construct an army

2.2 Top Level Classifier

A crucial aspect when choosing a classification method here is speed. The AI must be able to perform the classification in real-time leaving enough system resources for other subsystems. A fast way of performing classifications is using fuzzy models [7], which is the approach that has been used during this research for the top level classifier. Fuzzy models create the models of the several classes based on a single numerical feature. The choice of the numerical feature is crucial. In general, one should be able to define the several classes using this feature.

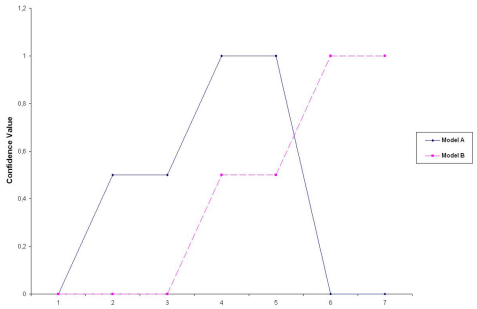

Figure 3: An example of a fuzzy model with 2 classes

Figure 3 shows an example of a fuzzy model. The general concept of the classes 'aggressive' and 'defensive' are how often an opponent attacks. An aggressive player would attack early and often, in order to put pressure on the opponent and disrupt him. A defensive player on the other hand would attack in a later stage of the game, concentrating his energies into one single attack. Hence a defensive player will rarely initiate an attack. As a result, a feature representing the frequency of attacks is an appropriate choice. An initial idea of such a feature would be the average amount of attacks per minute. However, using this feature would result into an update of the confidence rates only once a minute. A better feature which has been chosen was the relative amount of game time, that the opponent spent attacking, which can be updated constantly. An aggressive player will spend a large part of game time attacking. A defensive player on the other hand uses most time preparing an army and only needs a small amount of time for the actual attack.

For our research, we define an attack by the loss of ones own units. Usually an attack should be defined as a movement of an enemy group of units towards the own base. Such a detection, however, would require perfect information, which is not available. Instead the purpose of an attack has been applied as a definition, which is the destruction of enemy units. Hence, when a loss of units is detected, then the time span around this loss can be viewed as an attack. This is a logical choice of definition, easy to implement and moreover is suitable in imperfect information environments where data about the own units is always available. Therefor this approach to detect an attack was chosen.
The implemented AI registers all visible units each five seconds. The algorithm scans a time frame for lost units and if the final amount is above a certain threshold, then each discrete time moment inside the analyzed time frame is labeled as a moment in time, which the opponent spend attacking. This time window is shifted forward through the observation until each discrete time moment is labeled. The sensitivities

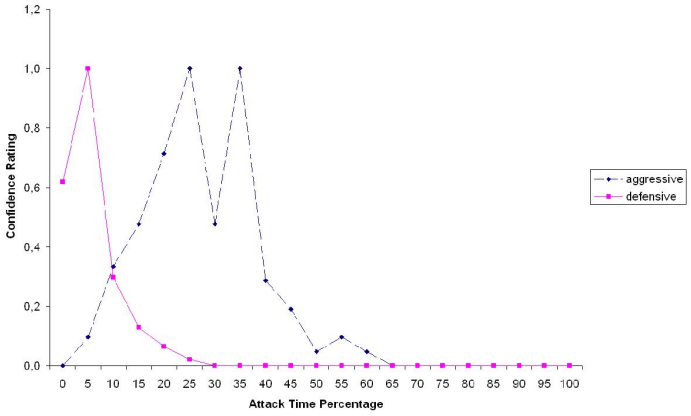

Figure 4: Constructed Player Models from 100 Example Games per Model

of the threshold and the size of the time frame are discussed in section 3. In the end, the percentage of time moments that are labeled as an attack moment is used to construct the models. Figure 4 shows the constructed models using data of 100 recorded matches against an aggressive and defensive player each.

2.3 Bottom Level Classifier

For the lower-level classifier, it would be necessary that it somehow emphasizes recent events more than past ones, since if recent observations change with respect to past ones, then this could mean a change in strategy which needs to be accounted for. Therefore a modification of the principle of discounted rewards [3] has been applied.

Theoretical Background

The concept of discounted rewards origins from the principle of repeated matrix-games [3]. There, a player can make a choice between several actions and depending on the choice of the opponent a reward is received. Choosing strategies and evaluating if a deviation from the original strategy is needed are important calculations in this field. A strategy in matrix games can be a simple sequence of actions, which is repeatedly played. 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 a player would prefer receiving 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[3]:

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

When the two expected rewards are known then according to those rewards it is decided whether the player should deviate from his original strategy.

Applied Modification

For opponent models it is not necessary to calculate some form of expected reward, but a confidence value. Hence not the rewards when using different strategies are calculated, but their confidence values. Basically, this calculation iterates through certain events in time and multiplies a value with the discount factor. In classical matrix games such an event would be one move of both players. Since RTS games do not operate in moves, the term of an event must be redefined. For aggressive opponents an attack would be an appropriate event since the player can then observe the army of the opponent. For defensive opponents, however, this would be an inappropriate choice since defensive players rarely attack and hence not many observations can be made. Additionally, waiting for an attack of a defensive player only to find out what strategy he uses, is most likely a fatal decision, since that particular attack will defeat the AI. Therefore an attack event alone is not sufficient, which is why some new event must be added.
Naturally, if one discovers that the opponent is playing defensively, then it is logical to scout the actions of the opponent. This action is also implemented by the modified AI, so that we can choose a scout-event as additional event.
Even though we are not computing rewards in the classical matrix games context, one should not discard the idea of a reward. Whenever a event is registered and new confidence values are computed, it is necessary to detect what strategy the opponent seems to be using at this time. That strategy should be rewarded with an increase of confidence rating. 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:

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

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. For example does a super weapon-building have a high tendency rating towards the defensive-super-weapon strategy, while a small defensive tower shows a small tendency towards an opponent using the defensive bunkering strategy. When all tendency values are collected, they are normalized to 1 so that the confidence reward can be distributed respectively.
In order to cope with the start phase of the game, where no events are registered yet, a default event is added at the beginning of the game, giving each strategy the same belief. This should represent the lack of information at this stage of the game such that each strategy has the same confidence rating.
Attack events are registered the same way as they are by the approach described in section 2.2. However, it remains arguable how to detect a scout event. It is possible to define a scout event by simply detecting sudden visibility gains. This would allow the use of data which is not collected by scout units, but also by other units that achieve visibility on to enemy structures. A more natural way would be to let the AI trigger such an event when it sends out a scout unit. That way, it would not be necessary to design an algorithm that detects a scout action of the own AI. For the duration of the research, the latter approach has been chosen.

3 Experiments

This section will cover the performed experiments with the discussed approach. Section 3.1 will cover a brief sensitivity analysis in order to find reasonable parameters for the top-level classifier. Section 3.2 contains the evaluation of a broad experiment with the top-level classifier using the best found parameters in Section 3.1 and Section 3.3 will cover all the experiments using the bottom-level classifier.

3.1 Sensitivity Analysis

As discussed in section 2.2, there are two crucial parameters influencing the attack detection algorithm. The first parameter is the time window size which is shifted through the time frames. The second parameter is the unit threshold, which is a percentage value. If the percentage of lost units in the considered time frame is above this threshold, then each time

frame inside the analyzed time window is considered an attack frame. For the time window, the sizes of 20, 30 and 40 seconds have been considered. For the unit threshold the percentages with value 10% and 15% were tested. The fuzzy models were constructed each match using the same parameters.
For each possible configuration, the AI was matched 10 times against an aggressively playing AI and 10 times against a defensive playing AI. As opponent representing both the aggressive and defensive player, the 'NTAI'-AI [5] was chosen, since it very simple to implement one's own strategy into this AI. This makes this AI ideal to test out this opponent modeling approach. Hence, for the matches where an aggressive opponent is required, the 'NTAI'-AI was configured with an aggressive strategy. Similarly, for the matches where a defensive player is represented, the AI was configured with a defensive strategy.
For each match played, the aggressive and defensive confidence values were recorded at each time point. After the match, the average confidence values of the entire match were calculated. However due to a weakness of the approach, which will be discussed in section 4, the information of the first 5 minutes will not be taken into account. This is due to the fact that in the first minutes of the game it is barely possible to perform any action that would reveal one's own tactic to the opposing player. This influences the test results negatively.
Table 1 shows the aggressive confidence values of matches against against an aggressive opponent and table 2 shows the defensive confidence values of matches against an 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 1: Average aggressive confidence values against an aggressive opponent

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

Table 2: Average defensive confidence values against an defensive opponent

As can be seen from table 2 all configurations of the fuzzy classifier perform very well when recognizing defensive players. However, their performance drastically differs when recognizing aggressive players. Only one configuration showed an acceptable performance when recognizing aggressive opponents, which had a units-lost threshold of 10% and a time window of 30 seconds. Hence this configuration was chosen for further analysis for the top level classifier.

3.2 Top Level Experiment

In this section, a similar test as in section 3.1 is performed, however only the resulting best configuration from section 3.1 is used. The AI is matched 50 times against an aggressive and 50 times against a defensive AI. Like in section 3.1, the 'NTAI'-AI was used to oppose the AI. Throughout each match, the computed aggressive and defensive confidence values have been recorded at each time. In order to analyze the development of the confidences during the game, the average confidence values of all test games over time have been computed, displayed in figure 5 for an aggressive opponent.

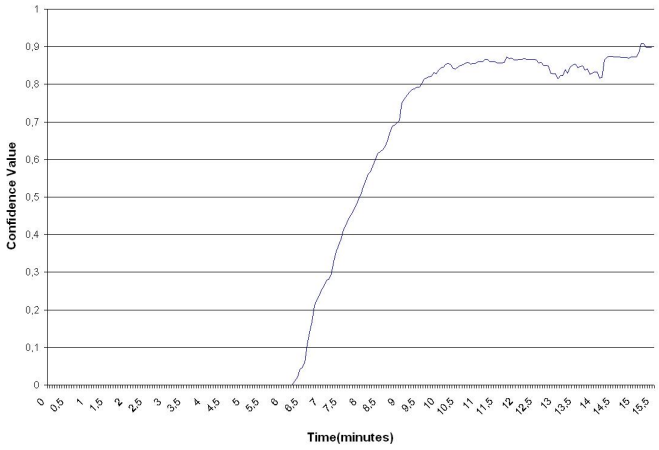

Figure 5: Average Confidence Value over Time against an aggressive opponent

As can be observed, the average confidence value is very low in the beginning of the match. This is due to the fact, that the enemy is hardly able to attack at this stage of the fight, since he needs to construct a base first. Therefor one should disregard the confidence values of the beginning of the game. After roughly 7 minutes of game time, the average confidence value drastically increases until it stabilizes at a confidence value of approximately 85%.
A similar, but reversed effect can be observed when examining average confidence value over time of the matches against a defensive opponent, displayed in figure 6. During the initial game stage, the confidence values are close to 100%. This is because the enemy does not attack in the beginning of the game. Therefore the fuzzy models will respond with the maximum defensive confidence value during this game stage. However you can see that after 6 minutes of game time, the average confidence value descends until it stabilizes between 96% and 97%.
From this test we can conclude that, given sufficient game

time, the confidence values will stabilize at proper values.

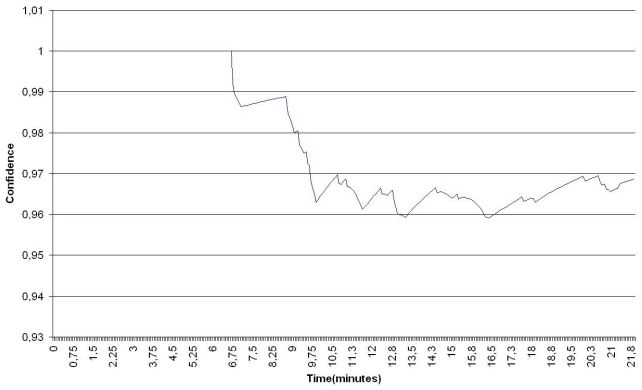

Figure 6: Average Confidence Value over Time against an defensive opponent

3.3 Bottom Level Experiment

For this experiment, the 'NTAI'-AI has been configured such that it resembles each of the specific aggressive player models. However, since the NTAI was unable to play the defensive strategies up to the desired standard, a human opponent was chosen to play against the 'AAI'-AI using the defensive strategies. For each player model, 10 experimental matches have been performed, during which at each time step all the confidence values have been recorded. A correct classification of the top-level classifier is assumed for this experiment. For the duration of the experiment, the parameters $\delta = 20\%$ and $\pi = 0.8$ have been chosen. The experimentally determined unit tendency values used to construct the parameter $\psi_{s,t}$ in formula 2 are presented in appendix A. As you can see in appendix A, the belief values of the aggressive sub-models barely differ. Thus it is sufficient to only test a part of all the sub-models to see, whether this approach can classify the aggressive sub-models. The defensive sub-models however each require testing. Figure 7 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.

As can be observed do both confidence values approximate a value near 90% after sufficient time. However, it is noticable that the average confidence of the aggressive tank-strategy increases more slowly and at a later stage than the average confidence of the aggressive kbot-strategy. This is due to the fact that tanks cost more ressources to produce. Because of that the opponent needs more time before he can attack.

For the defensive player models, each model is analyzed separately. Figure 8 displays the development of the different confidence values against an opponent using the defensive bunker-strategy. Here we see that the bunker confidence rating increases rapidly after approximately 5 minutes of game-

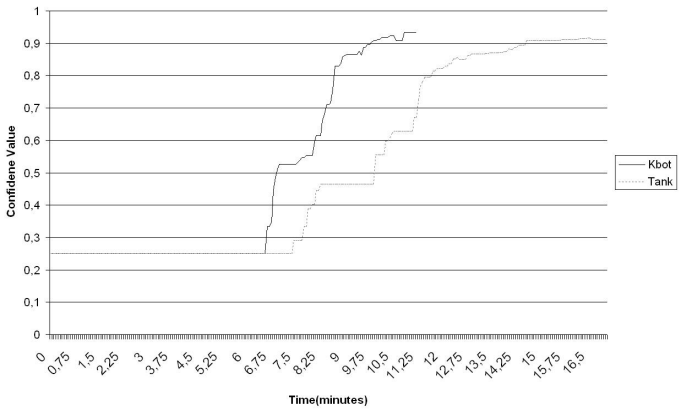

Figure 7: Average Confidence Value over Time for the aggressive sub-models.

time. For the majority of the game the confidence rating stabilizes at a value of roughly 65%. The instabilities occuring after 35 minutes of game-time indicate that at this time period the AI discovers structures that are also used by an opponent using the tech-strategy. However the value stabilizes again at approximately 40 minutes of game time.

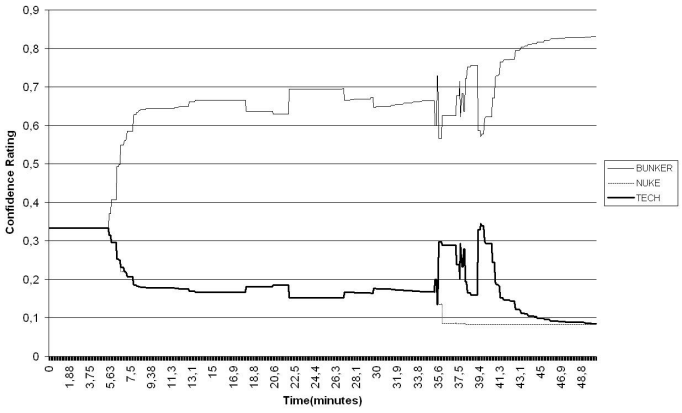

Figure 8: Average Confidence Value over Time for an opponent using the defensive bunker-strategy.

Figure 9 shows the development of the different confidence values of the test games against an opponent, that uses the tech-strategy. Here, the confidence value of the bunker-strategy does dominate during the majority of the game with a value of approximately 65%, whereas the confidence value of the tech strategy remains mostly at 20%. This can be explained by two circumstances. The first is that the AI is unable to penetrate the base defenses of its opponent with its scout-units in order to observe the enemy base, which results in only defensive facilities being observed. The second is that the high level units and structures that define the tech-strategy can only be constructed in later stages of the game. This means that in the earlier stages only structures that be-

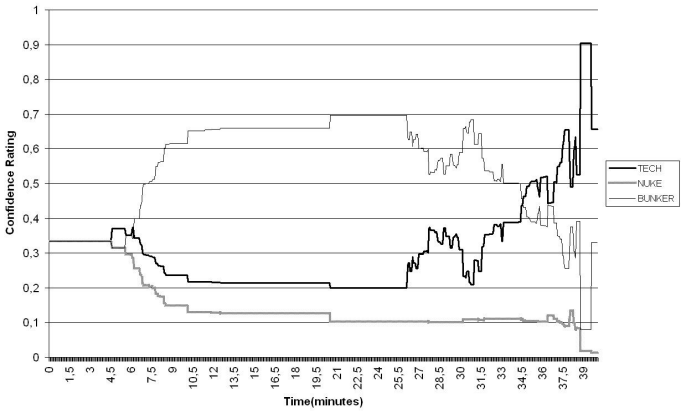

Figure 9: Average Confidence Value over Time for an opponent using the defensive tech-strategy.

long to a different strategy can be observed. You can see that after approximately 25 minutes of game time, the confidence value of the tech-strategy and the bunker-strategy becomy unstable. Moreover does the tech-strategy confidence value slowly start to increase. Likewise does the bunker-strategy confidence value drop at the same time. This means that at this stage of the game the AI is able to observe structures that clearly belong to a player using the tech-strategy, which results in the slow ascendance of the tech-confidence value. Also we can observe that the game ends with a high, but unstable tech-confidence value of roughly 65%.
Figure 10 shows the confidence development of the last experiment, which is against an opponent using the defensive Super-Weapon-strategy.

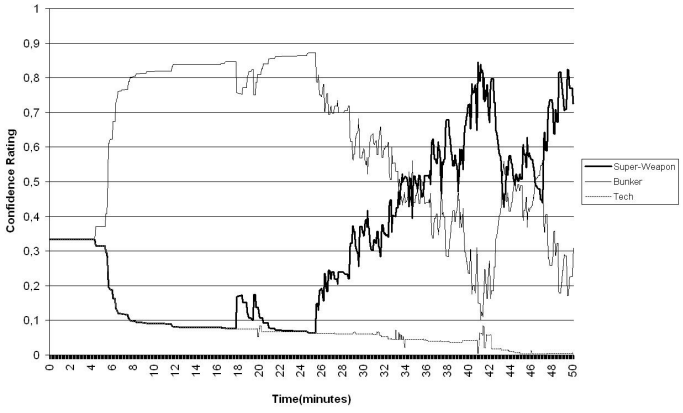

Figure 10: Average Confidence Value over Time for an opponent using the defensive Super-Weapon-strategy.

A similar development as in figure 9 can be observed in figure 10. Here, the bunker-strategy confidence value is also high during the first stages of the game, which is due to the same reasons that were explained earlier. At roughly 25 minutes of game-time, the super-weapon confidence value starts to increase. However, after a peak at approximately 40 minutes of game-time the confidence value starts to descend until it is roughly equal to the confidence value of the bunker-strategy. This can be explained by the dynamics of the discounted reward-approach. If at this stage of the game only defensive structures are observed, then it is logical that the bunker-strategy confidence value starts to increase while the super-weapon confidence value drops. However, we can observe that the experimental games ended with an average super-weapon confidence rating of approximately 75%.
We can conclude from the performed experiments, that the tested approach performs well on the aggressive sub-models, but rather mediocre on certain defensive sub-models. The fact that the opponent is classified correctly at the end of the game is not sufficient for an opponent modeling approach, since it would not allow the AI to react upon the classification. A correct classification must occur in earlier game stages, such that there is enough time for the AI to develop and execute a counter strategy.

4 Discussion

We have seen good results when testing the fuzzy models for the top level classifier. However there are also weaknesses, of which some got discovered during the experiments. For one, the fuzzy classifier is unable to cope with the lack of information during the beginning of the game, which resulted in a 0% confidence value for an aggressive player and a 100% confidence value for a defensive player during the beginning of each game. Secondly, the way how attacks are registered has a theoretical weakness, that can cause false positives as well as false negatives. If for example the opponent would attack and the AI would apprehend the attack without loosing many units or structures, then this attack might not be classified as an attack since the amount of lost units did not reach the threshold of lost units. As example for the false positives, we have the AI itself that attacks. If the AI looses too many units during an attack, then this time period may be classified as an attack of the opponent, even though he was only defending his base during this time period.
As further improvements for the fuzzy classifier, we suggest to apply some kind of system that copes with the lack of information during the beginning of a match. Additionally, the attack detection algorithm needs to be changed, such that it registers attacks as a group movement of enemy units towards the enemy base instead of a loss of units.
The evaluation of the experimental games testing the low-level classifier revealed that for certain sub-models the approach worked well. For other sub-models a misclassification during the majority of the match can be observed. Unlike the top-level classifier, no sensitivity analysis regarding the unit tendency values or the parameters δ and π have been performed. Therefor, it is possible that better results can be achieved using a different set of parameters. As another fu-

ture improvement we suggest, that the classification of the top-level classifier should influence the scouting behavior of the AI. For both the aggressive and defensive opponent the AI, in its current status, applies the same scouting behavior. However it would be necessary that the AI would scout more aggressively if a defensive opponent is recognized. With a more aggressive scouting behavior it would be possible to classify the defensive sub-models correctly at earlier stages of the game. A more aggressive scouting behavior, however, does not guarantee that during each scout event every crucial structure can be observed, since scout units can always be destroyed before reaching the center of the enemy base. This means, that the essential strategy defining structures can only be observed during a part of the scout events, which is the reason for the instability of the confidence values at later stages of the game. As future improvement we suggest to emphasize those scout events during which the essential structures are observed, so that the confidence value of the correct model does not descend rapidly after a few failed scouting attempts.

5 Conclusion

This paper concludes that hierarchical opponent model can be applied to RTS-games. It is observed during the experiments that, after sufficient game-time, the fuzzy classifier correctly classifies the opponent with a high and stable confidence rating. Regarding the modified approach of discounted rewards, that is used as low-level classifier, different results are observed for each sub-model. For all the aggressive sub-models and one defensive sub-model an early and stable convergence towards a correct classification with high confidence rating can be observed. The remaining sub-models however only converge at late game-stages to a correct classification in an unstable way, while during the game-time before the convergence the opponent is classified wrongly. After all, the used approach for the bottom level classifier did show potential and can be developed into a more reliable approach with more research.

6 Acknowledgments

We would like to thank Pieter Spronck, Jos Uiterwijk and Sander Bakkes for their support during this research. We would also like to thank Sander Bakkes for providing his modification of the 'AAI'-AI on which we based our research. Additionally we would like to thank Marcel Ludwig and Alexander Miesen for being the human opponent for half of experimental games testing the defensive sub-models.

References

[1] Donkers, J. and Spronck, P. (2006). Preference-based player modeling. *AI Programming Wisdom 3* (ed. S. Rabin), Chapter 8.4, pp. 647–659. Charles River Media, 25 Thomson Place, Boston, Massachusetts 02210.

[2] Gibbons, R. (1992a). *A Primer in Game Theory*, Chapter 2.3B. Pearson Education Limited, Edingburgh Gate, Harlow, Essex CM20 2JE, England.

[3] Gibbons, R. (1992b). *A Primer in Game Theory*, Chapter 2. Pearson Education Limited, Edingburgh Gate, Harlow, Essex CM20 2JE, England.

[4] Houlete, R. (2004). Player modeling for adaptive games. *AI Programming Wisdom 2* (ed. S. Rabin), Chapter 10.1, pp. 557–566. Charles River Media, 10 Downer Avenue, Hingham, Massachusetts 02043.

[5] Nowell, T. (2007). Ai:ntai. Creator of the game AI 'NTAI', http://spring.clan-sy.com/wiki/AI:NTAI.

[6] Seizinger, A. (2006). Ai:aai. Creator of the game AI 'AAI', http://spring.clan-sy.com/wiki/AI:AAI.

[7] Zarozinski, M. (2002). An open-fuzzy logic library. *AI Programming Wisdom* (ed. S. Rabin), Chapter 2.8, pp. 90–101. Charles River Media, 20 Downer Avenue, Suite 3, Hingham, Massachusetts 02043.

A Unit Tendency Values

	Model Tendency					Model Tendency			
Unit Name	K-Bot	Tank	Air	Ship	Unit Name	K-Bot	Tank	Air	Ship
Eagle			1		Flea	1			
Brawler			1		Fark	1			
Liche			1		Zipper	1			
Dragonfly			1		Zeus	1			
Freedom Fighter			1		Maverick	1			
Hawk			1		Fido	1			
Banshee			1		Sharpshooter	1			
Lancet			1		Fatboy	1			
Peeper			1		Archangel	1			
Phoenix			1		Invader	1			
Sabre			1		Scarab	1			
Tsunami			1		Eraser	1			
Albatross			1		Bantha	1			
Tornado			1		Razorback	1			
Thunder			1		Marauder	1			1
Blade			1		Vanguard	1			
Stiletto			1		Podger		1		
Archer				1	Jeffy		1		
Millenium				1	Flash		1		
Conqueror				1	Pincer		1		1
Skeeter				1	Stumpy		1		
Crusader				1	Shellshocker		1		
Epoch				1	Janus		1		
Decade				1	Samson		1		
Lurker				1	Triton		1		1
Piranha				1	Panther		1		
Serpent				1	Bulldog		1		
Swatter		1		1	Gremlin		1		
Pelican	1			1	Luger		1		
Anaconda		1		1	Merl		1		
Wombat		1		1	Penetrator		1		
Bear		1		1	Phalanx		1		
Skimmer		1		1	Seer		1		
Peewee	1				Hammer	1			
Rocko	1				Warrior	1			
Jethro	1								

Table 3: Unit tendency values for the aggressive sub-models

	Model Tendency		
Structure Name	Tech	Super-Weapon	Bunker
Retaliator		100	
Big Bertha		100	
Vulcan Canon		100	
Experimental Gantry	150		
Bantha	100		
Vanguard	100		
Marauder	100		
Razorback	100		
Adv. Aircraft Plant	25		
Adv. Vehicle Plant	25		
Adv. K-bot Lab	25		
Adv. Shipyard	25		
Nano Turret	5		
Dragon's Eye			1
Dragon's Teeth			1
Dragon's Claw			1
Light Laser Tower			1
Beamer			1
Sentinel			1
Guardian			1
Defender			1
Pack0			1
Chainsaw			1
Keeper			1
Ambusher			1
Pit Bull			1
Annihilator			1
Flakker NS			1
Mercury			1
Protector			1

Table 4: Unit tendency values for the defensive sub-models