Opponent Modelling in Wargus

Bachelor Thesis Business Communication and Digital Media
Faculty of Humanities
Tilburg University

Tetske Avontuur
Anr: 282263

Supervisor: Dr. Ir. P.H.M. Spronck

Tilburg, December 2010

Abstract

In most games, the goal is to defeat the opponent. In order to do this, a player has to anticipate the moves and strategy of the opponent. To discover what the opponent's strategy is, the player has to build a model of his opponent: an opponent model. This is an abstracted description of the opponent and his strategy, based on the opponent's behaviour in the game.

The problem statement that guides the research in this thesis is: *To what extent can a high level opponent model be constructed, using a classification algorithm, based on the behaviour of the player for a modern real-time computer strategy game?*

We built three opponent models for the real-time strategy game Wargus, based on the opponent's actions, using classification algorithms. To incorporate the three opponent models in the game, three scripts were written that each represented an opponent model. The actions of the opponents were registered for points in time in the game. This data was processed by a data mining program using five different classifiers. The resulting models were then applied on a test set.

We found that of the five classifiers tested, IBk and Jrip performed best. 10-fold cross-validation showed that IBk achieved an accuracy of 98.2%, and Jrip and accuracy of 91.6%. On the test set, they achieved an accuracy of 67.0% and 67.4% respectively, which is significantly higher than the frequency baseline of 42.2%.

From our experiments we conclude that it is possible to construct a high level opponent model for Wargus. The IBk and Jrip algorithms proved to be most suitable for the construction of such opponent models.

Preface

I am very lucky to be able to write this thesis on a subject I would have never thought of being able to choose: opponent modelling. This subject has little to do with my studies corporate communication and digital media but I am always up for a challenge.

Most of the time, I liked working on my thesis although it was a bit boring watching the computer do the same thing over and over again for a day or two. This thesis has taken a bit more time than I planned because my life was and is very busy and I started an internship while my thesis was not yet finished.

I would really like to express many thanks to Pieter Spronck for guiding me through the process of writing my thesis. There were a lot of things I did not know on this topic and the field of data mining was completely new to me but Pieter taught me a lot about these things. His advice and feedback were very helpful and he helped me a lot with some programming issues.

Furthermore, I would also like to thank my parents for supporting me and giving me the opportunity to get the most out of my studies. Of course I also want to thank my friends and my boyfriend for all the support they gave me during coffee and lunch breaks between writing my thesis, next to all the things I do with them, that makes studying in Tilburg a great and fun experience.

Tetske Avontuur

Table of Contents

1. Introduction

The goal of most games is to defeat the opponent. In order to do this, the rules of the game have to be mastered and applied strategically. Computers can do this by means of algorithms. For classical board games such as chess, checkers and scrabble, algorithms have been developed that play very strong games, irrespective of the circumstances (Schaeffer, 2000). This is possible because these games are deterministic, two-player, perfect-information games (Schaeffer, 2000). This means that one player wins at the expense of the other player who loses. Both players have complete information about the status of the board and of all the moves that have been made.

Modern games such as card games, modern board games and computer games are different from classical board games. They are often multiplayer games which means that there are three or more players. Information is imperfect because nobody knows the entire state of the board or the cards at hand of the opponents. In these games, only using algorithms is not enough to play a strong game. It is also important to gain knowledge about the opponent's strategy: the general play style that determines the opponent's actions and behaviour (Van den Herik, Donkers, & Spronck, 2005).

When a player has knowledge about the strategy of the opponent, this knowledge can be used to try to predict the opponent's future actions and change the player's own behaviour to affect the outcome of the game (Carmel & Markovich, 1996). This way, weaknesses of the opponent's behaviour can be exploited (Buro, 2003).

Knowledge about the opponent's strategy can be gained by observing and describing its behaviour (Billings, Pap, Schaeffer & Szafron, 1998). This is called opponent modelling: *making an abstracted description of a player or of a player's behaviour in a game* (Schadd, Bakkes & Spronck, 2007).

1.1 Problem Statement and Research Questions

In practice, opponent modelling is mainly used for simple, turn-based two-player board games because these games have a relatively large state-action space. Real-time strategy games however are highly complex, involve a high amount of actions, and have a relatively small state-action space, compared to board games.

In most research on opponent modelling in real-time strategy games, only a few action features of the opponent are used for classification. In the present research, the goal is to see to what extent it is possible to construct and classify an opponent model, based on all possible actions that can be registered. Therefore, the problem statement is:

To what extent can a high level opponent model be constructed, using a classification algorithm, based on the behaviour of the player for a modern real-time computer strategy game?

In order to solve the problem statement, three research questions have to be answered. The research questions are:

1. *What is a suitable high level opponent model for a real-time computer strategy game?*
2. *What is a suitable classification method for a high level opponent model for a real-time computer strategy game?*
3. *To what extent can an opponent model accurately classify computer-controlled opponents, based on their behaviour?*

1.2 Outline

In Chapter 1 of this thesis, an introduction is given to opponent modelling. In Chapter 2, background information is given on opponent modelling, opponent classifying, classifiers, real-time strategy games and the RTS game Wargus will be discussed. In Chapter 3, the experimental setup is discussed. It contains information about the features used to construct the opponent model, how the data was collected, the classification of the opponent models and the conduction of the tests. In Chapter 4, the results of these tests are described. In Chapter 5, the results will be discussed. Finally, Chapter 6 contains the conclusions.

2. Background

This chapter describes some of the advances in the field of opponent modelling, classifiers, strategy games and Wargus. Section 2.1 elaborates on the advances in the field of opponent modelling. Section 2.2 discusses the five algorithms that are used for our experiment.

2.1 Applications of Opponent Modelling

As described in Chapter 1, opponent modelling is *making an abstracted description of a player or of a player's behaviour in a game* (Schadd et al., 2007). However, it is not just making a description of any player, it is making a description of an *opponent* player. This is different from making a player model, where the player that is described can also be a friendly player. The goal of opponent modelling is gaining knowledge about the opponent's strategy to exploit its weaknesses. The goal of player modelling is not only to examine the opponent's role, but also the companion's role where the AI is the friend of the player and has to behave human-like (Bakkes, Spronck, & Van der Herik, 2009).

The topic of opponent modelling has been researched in both classical and modern games. In Subsection 2.1.1, the history of opponent modelling in classical games is described. In Subsection 2.1.2, opponent modelling in modern computer games is described. Subsection 2.1.3 describes strategy games. Subsection 2.1.4 discusses the real-time strategy game Wargus that was used for our research.

2.1.1 Opponent Modelling in Classical Games

This subsection gives an introduction on the field of opponent modelling in classical games. A few important studies in this field are discussed to give an indication of the advances in this field.

Two of the first studies on opponent modelling was performed by Carmel and Markovich (1993) and Iida, Uiterwijk and Van der Herik (1993), both studying minimax. Their algorithm uses a search tree and assumes that players minimize possible losses and maximize minimal wins. Carmel and Markovitch (1993) generalized this algorithm and developed the M* algorithm. In addition to the original minimax theory, it takes into account the depth of the opponent's minimax search.

A strategy for more easy games was examined by Egnor (2000) who studied Roshambo, or rock-paper-scissors. This is an imperfect information game where players compete with each other by choosing rock, paper, or scissors. Rock defeats scissors, paper defeats rock and scissors defeat paper. Egnor (2000) broke these rules down in 6 possible strategies. Then, he developed a meta-strategy that used past moves to predict future moves. Based on this prediction, the algorithm chooses one of the 6 strategies. The meta-strategy contained three features. First, it uses random guess because nobody can defeat a completely random player. Second, it uses frequency analysis. By looking at the past moves of the opponent, the algorithm can predict which moves are most likely to occur in the future. Third, it uses history matching which takes past sequences of moves to predict future move sequences.

Another game in which opponent modelling is essential is poker. This is an imperfect information card game. There are different hands one can have and some are stronger than others. If you know what is in the hand of the opponent, you can use this knowledge to defeat him or fold your hand to avoid a defeat. Davidson, Billings, Schaeffer and Szafron (2000) created an opponent modelling system which uses neural networks to predict the moves of the opponents and the strength of their hands.

2.1.2 Opponent Modelling in Modern Computer Games

This subsection describes some of the advances in the field of opponent modelling in modern computer games. This is relevant for this research because our research environment is based on a modern computer game.

In modern computer games, the main goal of opponent modelling is making the game more interesting for players (Bakkes et al., 2009). In these games, opponent modelling is difficult because the environment is complex, there is little time for observation, and information is often imperfect because the map is only partially visible (Bakkes et al., 2009).

There are several approaches to opponent modelling in modern computer game. Here, we list three approaches. First, the actions of opponents can be modelled (Ledezma, Aler, Sanchis and Borraja, 2009). Ledezma et al. (2009) constructed their own classifier to construct a model in a RoboCup challenge. They first built a general classifier to label the actions of the opponent. Then, an opponent is observed and his behaviours are labelled and classified based on the general classifier. After this, a model of the opponent is constructed based on the classified observations. Second, the preferences of opponents can be modelled. The preference is the source of the opponent's actions (Bakkes et al., 2009). Third, opponent modelling can be implemented by using case based reasoning. Here, an extensive database of played games is stored. The behaviour of players in a new game is matched with the games in the database. This way, future actions of the opponent can be predicted and the computer can adjust its strategy to the play style of the player. All these approaches use opponent classifying: data is extracted from games and used to build opponent models. Opponents in the game are then classified as one of these built opponent models (Bakkes et al., 2009).

2.1.3 Strategy Games

For games, a strategy is defined *as the complete plan of action that describes what a player will do under all possible circumstances* (Davis, 1997). Hence, a strategy game is a game where the planning of actions is important. The best known subgenre in strategy games is that of the war game. In war games, usually the goal is to defeat the army of the opponent. The present research is based on a war game.

Strategy games can be turn-based or real-time. In turn-based strategy games, players have to take turns. Examples of these games are board games like chess, or computer games like Civilization IV.

Unlike turn-based strategy games, in real-time strategy games, players can perform actions simultaneously. This means that there is little time for elaborating. In real-time strategy games, players have to gather resources, and build units and structures with the goal to destroy the opponent. The game Utopia, developed in 1982, is often named as the first real-time strategy game (Barron, 2003). However, the first games that really defined the genre were developed ten years later between 1992 and 1998. These games were Dune II and Command & Conquer by Westwood Studios, and Warcraft and Starcraft by Blizzard (Barron, 2003). They set the standard for modern RTS games.

2.1.4 Wargus

Wargus is the real-time strategy game that we use in the present research. It is a modified version of the real-time strategy game Warcraft II (Blizzard, 1995) and it runs on the Stratagus open-source gaming engine. The two games are virtually the same, with Wargus having more modern options for multiplayer, setting rally points, and using self-written AI scripts. Because Wargus is based on Warcraft II, the latter game will be discussed in the section below.

Warcraft II is a fantasy game that is set in a medieval land. There are two races to choose from: humans and orcs. The latter are humanoid troll-like creatures that are aggressive, not very intelligent and rather ugly. The story of the campaign mode in the game is placed in the land of Azeroth where humans and orcs fight each other. The campaign contains 24 scenarios: in 12 scenarios, the player is human, and in the other 12 the player is orc.

There is also a multiplayer option, which is modernized in Wargus. Here, a player can play against other humans or computers. At the start of the game, everyone begins with at least one peon, which is a worker that can gather resources and build structures. With those structures, the player can build an army of units and upgrade them. The goal is to defeat the enemy with this army.

In this thesis, the real-time strategy war game Wargus is used to test to what extent it is possible to build an opponent model, based on actions, for a real-time strategy modern computer game. The reasons for choosing Wargus are described in Section 3.1.

2.2 Classifiers

In the present research, we build an opponent model based on actions. In order to do this, we need a classification algorithm. We use five classification algorithms to test to what extend it is possible to accurately classify opponents. The classifiers are: (1) the IBk classifier, (2) the SMO classifier, (3) the Naive Bayes classifier, (4) the J48 classifier, and (5) the Jrip classifier. These classifiers are explained below.

IBk is an instance based k-nearest-neighbour algorithm. This means that it places instances in a 1-dimensional space. It then places new instances based on similarity with the previously placed instances. The class is used to predict this similarity (Witten & Frank, 2000).

The sequential minimal optimization algorithm, or SMO algorithm, is an optimized version of the SVM algorithm. It places instances in a 2-dimensional space and separates them into classes by means of vector lines. In vector space, linear vector lines can separate instances where in normal space, normal linear lines cannot separate instances into classes (Witten & Frank, 2000).

The Naive Bayes classifier calculates the chance that a new case belongs to a certain category, based on the cases that are already classified. This concept is called prior probability. The more instances there are in one class, the higher the change that a new case belongs to that class. (StatSoft Inc, 2010).

J48 is a greedy algorithm that uses information gain to build a decision tree. Greedy means that the algorithm tries to follow the most optimal path. The algorithm uses a top-down approach. It splits the data, based on the attribute that is best at discriminating between instances. The nodes that are formed are split into smaller nodes by adding the attribute that is second best at information gain. This continues until all data is split. The branches between the nodes contain the rules that have to be met to be placed in a certain node. When the model is built, it will try to classify new data by following the tree. (Witten & Frank, 2000).

The Jrip algorithm is which is a version of the Ripper algorithm (Pedersen, 2008). Jrip uses a bottom-up approach. It takes all the instances that belong to a certain class and then makes rules that divide the instances into the right class. (Pedersen, 2008).

3 Experimental Setup

The goal of our experiment is to develop three opponent models for opponents in Wargus and test them to examine to what extent the opponent models are able to accurately classify new instances. Section 3.1 discusses the reasons for choosing Wargus.

Our experiment consists of three steps. First, three opponent models were built for Wargus. Section 3.2 describes this. Second, the three opponent models battled each other on a specially designed map and then were validated based on the training set. For each game, the data was collected by writing down all values of the features that are described in Appendix II for every 70 or 80 game cycles. Then, the resulting data set was analyzed using five different classifiers. Section 3.3 discusses this. Third, a test set was created which was used to test the validated opponent models to examine to what extent the opponent models were able to classify the data. Section 3.4 describes this. Section 3.5 elaborates on the construction and validation of the models.

3.1 Wargus

To develop and test opponent models, the game Wargus was used. There are three reasons for this. First, this game has the possibility to register everything the players build, destroy and gather. Second, there was a modified version of this game available that allowed two computer players to play against each other without the need of a human player present in the game. Finally, it was easy to develop the AI scripts because the game is not very complex, and the programming language Lua in which the game is written is easy to understand. More information on Wargus can be found in Appendix I.

3.2 Opponent model

Three high-level example opponents were developed to test whether it is possible to classify opponent models. The three models make out three different strategies: aggressive, defensive and neutral. These strategies are implemented in the experiment by developing three scripts that are executed by an AI-player. These scripts are based on the land attack script that was obtained from the installation folder of Wargus. This script contains a build and attack plan for a land battle, and no sea units are built.

The land attack script was chosen the basis for the three scripts and altered for this experiment. This script was chosen because it has a balance between defensive and offensive play. It was altered because it also made the choice of building twelve towers at ineffective moments. The instructions to build those towers were removed and new instructions on different placed were added to build five towers. This changed the land attack script into the neutral AI script.

To differentiate in strategy between the three opponent models, small adjustments were made to the script of the neutral opponent model (B). All three models build defensive and offensive forces. In its attack forces, the aggressive player (A) builds one unit of each type more than the neutral and defensive players. In its defensive forces, its builds one unit of each type less than the neutral player and two units of each type less than the defensive player. The defensive player (C) builds the same amount of attack forces as the neutral. In its defensive forces, this player builds one more unit of each unit type more than the neutral script and two more than the aggressive script. This is shown in Table 3.1. The defensive AI also builds one guard tower more than the other players and attacks the other players one time less than the other two scripts.

The first column of Table 3.1 lists the opponent models. The second column describes the type of force: defensive or offensive. The third column lists the number of units built per unit type more or less in relation to the neutral player. The fourth column describes the amount of towers more or less in relation to the neutral player. The fifth column describes the number of attacks more or less in relation to the neutral player.

Table 3.1 The differences in amount of units and towers built and number of attacks per unit type between the three models

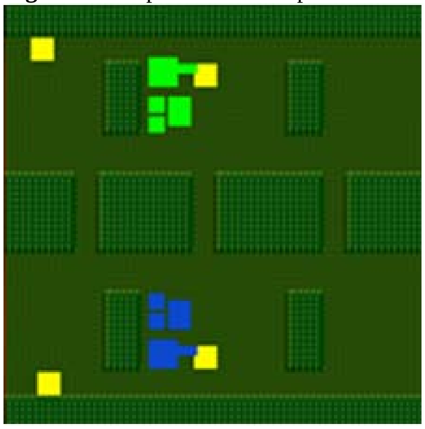
Model	Force	Amount of units	Amount of towers	Number of attacks
Aggressive (A)	Defensive	- 1 unit	+ 0 towers	+1 attacks
	Attack	+ 1 unit	+ 0 towers	+1 attacks
Neutral (B)	Defensive	+ 0 unit	+ 0 towers	+ 0 attacks
	Attack	+ 0 unit	+ 0 towers	+ 0 attacks
Defensive (C)	Defensive	+ 1 unit	+ 1towers	+ 0 attacks
	Attack	+ 0 unit	+ 1towers	+ 0 attacks

3.3 Constructing the training set

To generate a training set, the models have to battle each other. Because there is a machine versus machine option in the version of Wargus that we used, it is possible to have two computers competing with each other.

For this experiment, a special map was created where all battles take place. The upper and lower half of the map are mirror images of each other. The map contains no water. The map is displayed in Figure 3.1.

Figure 3.1 Map used in the experiment



🟨 Gold Mine

🟩 Buildings player 0

🟦 Buildings player 1

▩ Trees

Every game, there are two players fighting. One player is placed at the top of the map and the other is placed at the bottom. All players start half of the games at the top of the map and half of the games at the bottom of the map to remove bias in outcome caused by map placement.

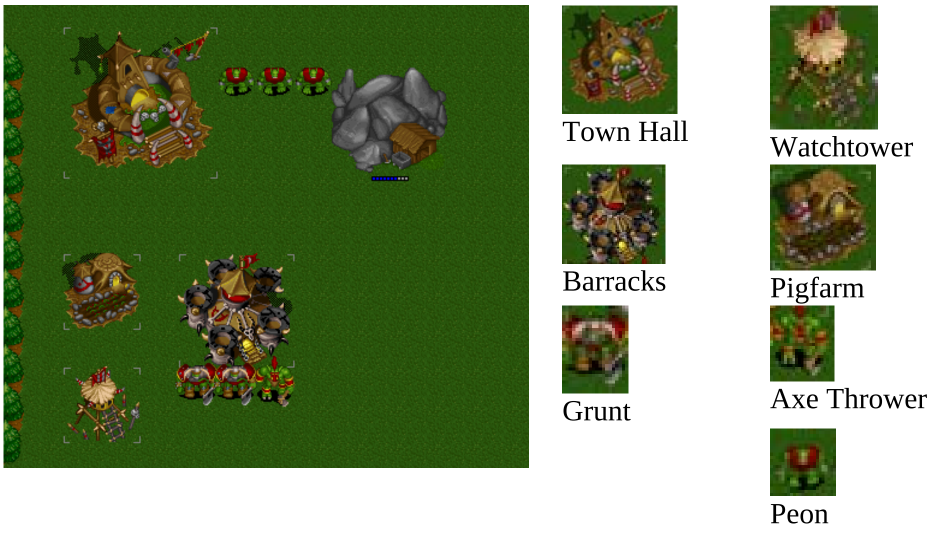
The battle scheme of is displayed in Table 3.2. All games are orc versus orc to avoid any imbalance between the players.

Table 3.2 Battle scheme

Top player	Bottom player
Aggressive (A)	Aggressive (A)
Aggressive (A)	Neutral (B)
Aggressive (A)	Defensive (C)
Neutral (B)	Aggressive (A)
Neutral (B)	Neutral (B)
Neutral (B)	Defensive (C)
Defensive (C)	Aggressive (A)
Defensive (C)	Neutral (B)
Defensive (C)	Defensive (C)

Each player begins the game with one town hall, one barracks, one farm, one guard tower, three workers named peons, two close ranged units named grunts and one ranged unit named axe thrower. Workers can gather recourses and build and repair buildings. The melee and ranged units are used for fighting. The unit and building placement for the player at the upper part of the map is shown in Figure 3.2. This set-up is mirrored for the player at the lower part of the map. All units and buildings are explained in Appendix I.

Figure 3.2 Placement of buildings and units at the start of the game



There are 9 different games to be played. Each game is played 3 times which results in a total of 27 games. The output data of these games consists of a list of 45 features that can be found in Appendix II. Data is collected for every 50 game cycles. The 27 games result in a data set with a total of 52506 instances. This is the training set.

Five different classifiers were used for model building: IBk, SMO, Naive Bayes, J48, and Jrip. The open source data mining program Weka was used for the validation and the program's standard parameters for these classifiers were used.

3.4 Constructing the test set

To test whether the opponent models are able to correctly classify new instances, a test set is built. We wanted to see whether the classifiers would still be able to accurately classify the A, B, and C opponent models if they were put against new opponents.

Three new opponent models were constructed by altering the neutral model. Then, they were put in a battle against the three previously built opponent models. The three new models are called D, E, and F. Model D builds one unit per unit type less than the neutral model and the same amount of attack forces. Model E builds one more unit per unit type more in his defensive force than the neutral player. In addition, this model builds one less unit per unit type in its attack force. Finally, model F builds the same amount of defensive forces as the neutral player and builds one unit per unit type more in its attack forces. The difference in amount of units built per type in relation to the neutral script is shown in Table 3.4. There were no differences in number of towers built or number of attacks.

The first column of Table 3.4 lists the new opponents. The second column describes the type of force. The third column describes the difference in amount of units built per unit type in relation to the neutral script.

Table 3.4 Differences in amount of units built per unit type between the three new models and the neutral script

Model	Force	Difference amount of units built
D	Defensive	- 1 unit
	Attack	+ 0 unit
E	Defensive	+ 1 unit
	Attack	- 1 unit
F	Defensive	+ 0 unit
	Attack	+ 1 unit

Models A, B, and C are placed in battles against models D, E, and F. Each of the first three models fights each of the last three models two times: one time, they start at the top of the map and one time, the start at the bottom of the map. This way, the three models that are classified, A, B, and C, all fight three new opponents. They battle each new opponent two times, resulting in a total data set of 18 games. The data is collected for every 10 game cycles and this this results in a test set of 48304 instances.

The data of the test set was tested on the previously validated opponent models, using the same five classifiers that were used to validate the training set: IBk, SMO, Naive Bayes, J48, and Jrip. Again, this was done using Weka usin the program's standard parameters.

3.5 Model building and validation

The opponent models are built based on the training set, using a 10-fold cross validation. This means that Weka divides the data set into 10 parts. Nine parts are used as the training set and one part is used as the test set. The program repeats this process 10 times, so each of the ten parts has been used as the test set. Classification is done based on the attributes that are listed in Appendix II, with play style as the class attribute.

For the training set, each game is played 3 times because it was assumed that there would be slight differences because of randomness. All corresponding games were compared against each other to examine whether there were any differences. This proved that each game is exactly the same. Therefore, for the test set, every battle is only fought once.

To test whether the results are significant, they have to be compared with a baseline model. Because the instances are not equally distributed among the classes in this research, the performances of the classifiers have to be compared with a frequency baseline model. It calculates the error rate with the formula $1-f_{max}$, *where* f_{max} is the percentage of the instances that belong to the most frequent class. A chance baseline model ignores the fact that instances are not equally distributed but calculates the chance based on the number of classes. For example, if there are 100 instances and 2 classes, and class A contains 20 instances and class B contains 80 instances, the frequency baseline would be 80% because 80% of the instances belong to class B. The chance baseline however would be 50% because there are two classes.

Because the instances are not equally distributed among the classes, the performances of the classifiers for both the training set and the test set have to be compared with a frequency baseline model. It calculates the error rate with the formula $1-f_{max}$, *where* f_{max} is the percentage of the instances that belong to the most frequent class.

4 Results and Discussion

In this chapter, the performances of the opponent models on classification are discussed. Section 4.1 discusses the results of the validation of the opponent models. Section 4.2, describes the performances of the opponent models on the test set. Section 4.3 discusses the performance of the classifiers for the training set and test set.

4.1 Validating the opponent models

The training data was used in a 10-fold cross validation to validate the opponent models. The results have to be compared with a baseline model to see whether they performed significantly better than the baseline. In this test, the instances are not evenly distributed among the classes. The reason for this is that each instance equals the results of one recorded game cycle. Some AI's take longer to finish a game than others, depending on the features of the AI and the opponent.

The total data set contains 52506 instances and the most frequent class is class B with 18516 instances. This results in a frequency baseline of 52506/18516= 35.3%. Class A contained 16593 instances and Class C contained 17397 instances. Now, the results of the classifiers can be compared with the frequency baseline. We compare against *Fmax* instead of the error rate.

The results of the 5 classifiers are displayed in Table 4.1. The first column contains the classifier, the second column tells what percentage of the instances was correctly classified, and the third column tells what percentage of the instances was incorrectly classified. The fourth column contains Fmax.

Table 4.1 Results of the classifiers

Classifier	**Correctly classified instances**	**Incorrectly classified instances**	***Fmax***
IBk	98.2 %	1.8 %	35.3%
SMO	75.9 %	24.1 %	35.3%
Naive Bayes	51.2 %	48.8 %	35.3%
J48	98.4 %	1.6 %	35.3%
Jrip	91.6 %	8.4 %	35.3%

Three classifiers performed very well in classifying the instance. IBk, J48, and Jrip classified more than 90% of the instances were classified correctly. The SMO classifier performed less strong with 75.9% of the instances correctly classified. The Naive Bayes classifier performed worst with only half of the instances classified correctly.

All classifiers performed better than the baseline. This means that all classifiers correctly identified more instances than if they would randomly guess. The IBk and J48 classifiers performed best with respectively 98.2% and 98.4% correctly classified instances. The Jrip classifier did slightly worse than IBk and J48 but still performed very well with 91.6% correctly classified instances. The SMO and Naive Bayes classifiers performed the worst with respectively 75.9% and 51.2% instances classified correctly.

4.2 Testing the opponent models

After validating the opponent models using a 10-fold cross validation, a test set is created to test the performance of the validated opponent models in classifying new instances. The total number of instances for the test set is 48304. This seems a relatively large amount of data compared to the training set. There are two reasons for this. First, for the training set, only data for every 50 game cycles was collected. For the test set, data for every 10 game cycles was collected. Second, the games the neutral player (B) played against the new opponents, on average, were longer than the games the neutral player (B) played in the training set.

Now, the frequency baseline of the test set is calculated. The total data set contains 48304 instances and the most frequent class is class B with 20372 instances. This results in a frequency baseline of 20372/48304= 42.2%. Class A contains 14426 instances and Class B contains 13506 instances.

The results of the classifiers can now be compared to F_{max}. The results are displayed in Table 4.2. The first column contains the classifier name, the second column contains the percentage of correctly classified instances. The third column displays the percentage of incorrectly classified instances. The fourth column displays F_{max}.

Table 4.2 Comparison of the classification results with the frequency baseline for the training set

Classifier	Correctly classified instances	Incorrectly classified instances	F_{max}
IBk	67.0%	33.0%	42.2%
SMO	53.7%	46.3%	42.2%
Naive Bayes	48.1 %	51.9%	42.2%
J48	62.0 %	38%	42.2%
Jrip	67.4 %	32.6%	42.2%

All classifiers performed better than the frequency baseline. The IBk and Jrip classifiers performed best with respectively 67.0% and 67.4% instances classified correctly. J48 did slightly worse with 62.0%. The SMO classifier did worse with 53.7% correctly classified instances. The Naive Bayes performed worst, it only classified 48.1% of the instances correctly which is only slightly better than predicted by chance.

4.3 Performance of the classifiers

The performances of the algorithms were very strong for the training set. The IBk and J48 algorithms performed very well with about 98% of the instances classified correctly. However, the test results of the test set were less strong. The IBk algorithm again performed best with 67% correctly classified instances. This is notable because IBk is the least complex algorithm of the five that were used. It places new instances based on similarity with previously placed instances. A possible explanation can be that there is no randomness in the execution of the scripts written for the AI players in Wargus. A script will do exactly the same every time it is executed. The only differences that arise come from the attacks or the defence of the opponents. The cycles of each game will be exactly the same until there is an encounter with an opponent. This results in game data that is very similar for each played game. Because all game data is very similar, and the instances of the early game cycles are exactly the same for each game, IBk has little trouble placing instances based on similarity with previous instances.

For the test set, the performance of Jrip (67.4%) was higher than the performance of J48 (62.0%) although for the training set, J48 (98.4%) performed better than Jrip (91.6%). A possible reason for this is that J48 creates many rules because it uses a bottom-up approach. Jrip generated 50 rules to classify the data while J48 generated 291 leaves for a tree with a total size of 581. The focus in this research is on making a high level opponent model, which is based on the general play style of the opponent. Even when there are small differences between instances, they can still belong to the same general play style. The instances of the test set differ somewhat from the instances in the training set but both still belong to the same class. Because Jrip is less precise, these differences will be taken into account less than with J48 and instances will still be classified correctly although they are not exactly the same.

For Jrip, there was one rule that was most decisive for style C. If, at a game cycle, the opponent had 3 or more guard towers, and it's total amount of gathered wood was equal to or less than 22625, and it had 3 or more watch towers, and it had 2033 or less wood at that moment, a total weight of 7572 instances could be classified as style C, the defensive opponent model, with 0 misclassified for the training set. For the training set, a total of 17397 instances belonged to style C. This means that 7572/17397= 43.52% of the instances belonging to style C, could be classified with this one rule. This rule is logical because style C builds one more watch tower than all other styles. This watch tower is later upgraded to a guard tower so style C has one more guard tower than the other styles later in the game and one more watch tower more than the other styles earlier on in the game.

The most decisive tree leaf for J48 was based on 12 features and predicted style B with a total weight of 2028 instances with 0 instances misclassified for the training set. For a total of 18516 instances belonging to style B, 2028/18516= 10.95% of the instances of class B could be classified using this leaf.

5 Conclusions

In this chapter, the research questions and problem statement are answered. Section 5.1 discusses this. Section 5.2 discusses the limitations of this research are discussed. Section 5.3 describes the recommendations for future work.

5.1 Answering the research questions and problem statement

In this section, the answers to the research questions and problem statement are given. This section is divided into four subsections. The first three subsections each answer one research question. The final subsection discusses to what extent the problem statement is answered. The three research questions were:

1. *What is a suitable high level opponent model for a real-time computer strategy game?*
2. *What is a suitable classification method for a high level opponent model for a real-time computer strategy game?*
3. *To what extent can an opponent model be constructed that can accurately classify computer-controlled opponents, based on their behaviour?*

5.1.1 Suitable high level opponent model

The results of our research showed that registering the actions of an opponent in the real-time strategy game Wargus is a solid base to create a high level opponent model for this environment. Chapter 3 explains the method that was used for building hour opponent model.

5.1.2 Suitable classification method

Using classifiers is a suitable method to create an opponent model but not all classifiers used in this research are equally effective. The IBk and Jrip classifiers showed the best results for the real-time strategy game Wargus and turned out to be most suitable for this environment. Both classifiers were able to create solid opponent models and were able to accurately classify them in new situations. Only considering building opponent models, IBk and J48 performed best but when these opponent models have to classify new data, IBk and Jrip turn out to be most suitable for Wargus. This is argumented in Section 4.3.

The Naive Bayes and SMO classifiers proved to be less effective for this environment as they performed lower than the other three classifiers. They were able to build opponent models but did not do as well in classifying opponents in a new situation.

5.1.3 Classifying computer-controlled opponents

This research demonstrated that computer-controlled opponents can be accurately classified for the real-time strategy game Wargus. Opponent models were built and validated as described in Chapter 3. The performance of the models for validation was strong. The opponent models did well at classifying the training set and also performed well on the test set. This is argumented in Chapter 4.

The extent to which an opponent model can be constructed is high. Results were very strong for the training set. The extent to which an opponent model can be used for classification of new instances is less high. This is argumented in Sectin 4.3.

5.1.4 Answering the problem statement

The problem statement is:
To what extent can a high level opponent model be constructed, using a classification algorithm, based on the behaviour of the player for a modern real-time computer strategy game?

The answer to the problem statement is that it is possible to construct a high level opponent model for the modern real-time computer strategy game Wargus. The IBk and Jrip algorithms proved to be most suitable for the construction of opponent models for Wargus. To examine whether it is possible to construct opponent models for other real-time computer strategy games, further research is needed.

5.2 Limitations

This study has four limitations. First, Wargus is a fairly simple strategy game. More modern strategy games are often more complex and that might make it harder to classify opponents based on their actions. Second, only one race, the orc, was used to play with in the game. Using other races might change the output of the actions while the underlying strategies are the same. This might be a problem for games like Starcraft (Blizzard, 1998) and Starcraft II (Blizzard, 2010) because the available races have to be played in very different ways. Third, the scripts that were used to create the training set and the test set were exactly the same. This makes it easier to classify the instances of the test set because all actions are the same. It would be interesting to test the classification performance in a situation where the models are slightly different, but still follow the same general strategy. Fourth, the game uses perfect information. Every action of the opponent is taken into account, while in a real game, it is only possible to see the opponents actions by scouting, unless the computer would cheat to see the actions of its opponent.

5.3 Future research

This research shows that using opponent actions to classify opponents is a valid approach to opponent. There are, of course, still recommendations for future work.

First, more scripts could be generated for each opponent model to create slight variations in the output of the actions while the scripts still follow the same general play style. Furthermore, different races could be used that follow the same strategy to test to what extent the data would be classified correctly. The layout of the game map is also a factor that can influence the course of a game. The same opponent models could battle each other on different maps. Another recommendation for future work is to create more opponent models that have smaller differences between them and test whether classifiers then still are able to accurately classify instances. Finally, results might be improved by modifying some of the output features to make the data more useful. Using a function like InfoGain and GainRatio might also help to improve the results. These functions increase the importance of attributes that are more important for classifying and reduce the importance of attributes that are less important for classifying (Den Teuling, 2010).

References

Adams, E. (2009). *Fundamentals of game design*. Berkeley, CA: New Riders.

Bakkes, S. C. J., Spronck, P. H. M., and Van den Herik, H. J. (2009). Opponent Modelling for Case-based Adaptive Game AI. *Entertainment Computing, 1(1),* 27-37.

Bakkes, S., C., J., Spronck, P., H. M., & Van den Herik, H. J. (2009). Rapid and Reliable Adaptation of Video Game AI. *IEEE Transactions on Computational Intelligence and AI in Games, 1(2),* 93-104.

Barron, T. (2003). *Strategy game programming with DirectX 9.*Plano, TX: Wordware Publishing, Inc.

Billings, D., Papp, D., Schaeffer, J., & Szafron, D. (1998). Opponent modeling in poker. Proceedings *of the Fifteenth National Conference on Artificial Intelligence,* 493–
498.

Buro, M. (2003). Real-time strategy games: A new AI research challenge. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* 1534-1535.

Carmel, D. & Markovitch, S. (1993). Learning Models of Opponent's Strategy in Game Playing. *Proceedings of AAAI Fall Symposium on Games: Planning and Learning,* 140-147.

Carmel, D., & Markovitch, S. (1996). Learning Models of Intelligent Agents. *AAAI/IAAI, 1,* 62-67.

Davis, M., D. (1997). *Game Theory: a Non-Technical Introduction.* Mineola, NY: Dover Publications Inc.

Davidson, A., Billings, D., Schaeffer, J., & Szafron, D. (2000). Improved opponent modeling in poker. *Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI'2000),* 1467–1473.

Den Teuling, F., (2010). *Player modelling in Civilization IV.* Retrieved from http://arno.uvt.nl/show.cgi?fid=105933

Egnor, D. (2000). Iocaine Powder. *ICGA Journal, 23(1),*33-35
Schaeffer, J. (2000). The Games Computers (and People) Play. *Advances in Computers 50,* Academic Press, 189-266.

Ledezma, A., Aler, R., Sanchis, A., & Borrajo, D. RoboCup 2004: Robot Soccer World Cup VIII. *Lecture Notes in Computer Science, chapter Predicting Opponent Actions by Observation, 3276,* 286-296.

Pandolfini, B. (2003). *Pandolfini's ultimate guide to chess.* New York, NY: Fireside Books.

Pedersen, T. (2008). Learning high precision rules to make predictions of morbidities in discharge summaries. *Proceedings of the i2b2 Workshop on Challenges in Natural Language Processing for Clinical Data*.

Schadd, F., Bakkes, S. C. J., & Spronck, P. H. M. (2007). Opponent Modeling in Real-Time Strategy Games. *8th International Conference on Intelligent Games and Simulation (GAME-ON 2007),* 61-68.

Spronck, P. H. M. (2005). *Adaptive Game AI.* Retrieved from http://ticc.uvt.nl/~pspronck/pubs/ThesisSpronck.pdf

StatSoft, Inc. (2010). *Electronic Statistics Textbook. Tulsa,* OK: StatSoft. WEB: http://www.statsoft.com/textbook/

Van den Herik, H. J., Donkers, H. H. L. M., and Spronck, P. H. M. (2005). Opponent Modelling and Commercial Games. *Kendall, G. and Lucas, S. (Eds.), Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05),* 15–25.

Witten, H. I., Frank, E. (2000). Data mining: practical machine learning tools and techniques with Java implementations. San Diego, CA: Academic Press.

Warcraft II wiki. Retrieved from www.wowwiki.com

Appendix I Wargus

Spronck (2005) describes Wargus as follows:

> Wargus is a copy of the game Warcraft II that was released by Blizzard in 1995 and rereleased in 1999. Wargus is built on the open-soure game engine Stratagus. Stratagus was formerly known as FreeCraft, but for legal reasons the engine has been renamed. Wargus a game module for Stratagus, implemented in the high-level Lua scripting language (Ierusalimshy, de Figueiredo, and Celes, 2003). (p. 183)

Warcraft II, and thus Wargus, is a real-time strategy game that is placed in a fantasy medieval setting. Players can choose to play either with the human race or the orc race. The orc are green, troll-like creatures (Warcraft II Wiki).

Units

List of the units in the game that were included in our research (Warcraft II Wiki):

Peons
A Peon is a worker unit. It can gather wood and gold, and build and repair buildings.

Melee units
These are units that can attack other units or buildings from a small distance. These units include the:
- Grunt
- Orge
- Orge-Mage (upgraded Orge)

Ranged units
These are units that can attack other units or buildings from a large distance. These units include the:
- Axe Thrower
- Berserker (upgraded Axe Thrower)
- Catapult
- Death Knight

Air Units
Air units are units that have the ability to fly. These units include the:
-Zeppelin (unable to attack other units or buildings)
- Dragon

Structures

List of the structures in the game that were included in our research (Warcraft II Wiki):

Town Hall, Stronghold, Fortress
Gathered recourses can be brought to the Town Hall by Peons which adds them to the player's

available resources. A Town Hall can also produce Peons. The Town Hall can be upgraded into a Stronghold which can be upgraded into a Fortress.

Barracks
Barracks produce Grunts, Axethrowers, Berserkers, Catapults, Ogres and Ogre-Mages.

Pigfarm
Pigfarms provide food which is needed to train troops. It is impossible to train more troops than the amount of food that is available.

Watch Tower, Guard Tower, Cannon Tower
Watch Towers provide vision and are unable to attack. They can be upgraded into Guard Towers and Cannon Towers which are structures that can shoot enemy units.

Lumber Mill
The Lumber Mill contains upgrades for Axe Throwers and Berserkers. Peons can gather wood in this building.

Blacksmith
The Blacksmith contains upgrades for Grunts, Axe Throwers and Catapults.

Alchemist
This Alchemist produces Zeppelins.

Ogre Mound
The Ogre Mound enables the production of Ogres.

Altar of Storms
The Altar of Storms contains upgrades for Ogres. One of the upgrades is the upgrade from Ogre to Ogre-Mage.

Temple of the Damned
The Temple of the Damned enables the production of Death Knights and contains upgrades for Death Knights.

Dragon Roost
The Dragon Roost produces Dragons.

Shipyard
The Shipyard can produce ships.

Oil Refinery
When built, the Oil Refinery gives a bonus to the collection of oil

Foundry
The foundry contains upgrades for ships.

Oil Platform
The Oil Platform is built upon oil that can be found in certain places in the water.

Appendix II Table of data features used for classification

#	Feature	Meaning
1	Name	Placement
2	Style	Play style
3	GameCycle	The moment in time
4	Score	Score of the player
5	TotalNumUnits	Total number of units built during the game
6	TotalNumBuildings	Total number of buildings built during the game
7	TotalKills	Total number of enemy soldiers killed during the game
8	TotalRazings	Total number of buildings destroyed during the game
9	TotalUnits	Total number of units possessed at current game cycle
10	TotalBuildings	Total number of buildings possessed at current game cycle
11	Supply	Food supplied by farms at current game cycle
12	Demand	Food demanded at current game cycle
13	TotalGold	Total amount of gold gathered during the game
14	TotalWood	Total amount of wood gathered during the game
15	TotalOil	Total amount of oil gathered during the game
16	Gold	Amount of gold possessed at current game cycle
17	Wood	Amount of wood possessed at current game cycle
18	Oil	Amount of oil possessed at current game cycle
19	UnitGreatHall	Number of great halls possessed at current game cycle
20	UnitStronghold	Number of strongholds possessed at current game cycle
21	UnitFortress	Number of fortresses possessed at current game cycle
22	UnitTrollLumberMill	Number of lumber mills possessed at current game cycle
23	UnitOrcBlacksmith	Number of blacksmiths possessed at current game cycle
24	UnitAlchemist	Number of alchemists possessed at current game cycle
25	UnitOgreMound	Number of ogre mounds possessed at current game cycle
26	UnitAltarofStorms	Number of altars of storms possessed at current game cycle
27	UnitTempleOfTheDamned	Number of temples of the damned possessed
28	UnitDragonRoost	Number of dragon roosts possessed at current game cycle
29	UnitOrcBarracks	Number of barracks possessed at current game cycle
30	UnitOrcWatchTower	Number of watch towers possessed at current game cycle
31	UnitOrcGuardTower	Number of guard towers possessed at current game cycle
32	UnitOrcCannonTower	Number of cannon towers possessed at current game cycle
33	UnitOrcShipyard	Number of shipyards possessed at current game cycle
34	UnitOrcRefinery	Number of refineries possessed at current game cycle
35	UnitOrcFoundry	Number of foundries possessed at current game cycle
36	UnitOrcOilPlatform	Number of oil platforms possessed at current game cycle
37	UnitGrunt	Number of grunts possessed at current game cycle
38	UnitAxeThrower	Number of axe throwers possessed at current game cycle
39	UnitBerserker	Number of berserkers possessed at current game cycle
40	UnitOrgre	Number of ogres possessed at current game cycle
41	UnitOgreMage	Number of mages possessed at current game cycle
42	UnitDeathKnight	Number of death knights possessed at current game cycle
43	UnitCatapult	Number of catapults possessed at current game cycle
44	UnitZeppelin	Number of zeppelins possessed at current game cycle
45	UnitDragon	Number of dragons possessed at current game cycle