Master Thesis
Software Engineering
Thesis no: MSE-2005:05
March 2005

A Multi-Agent System for playing the board game Risk

Fredrik Olsson

Department of
Systems and Software Engineering
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden

This thesis is submitted to the Department of Systems and Software Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:

Author: Fredrik Olsson
E-mail: fredrik.olsson@spray.se

University advisor:
Stefan Johansson
Department of Systems and Software Engineering
E-mail: stefan.johansson@bth.se

Department of
Systems and Software Engineering
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden

Internet : www.bth.se/tek/aps
Phone : +46 457 38 50 00
Fax : +46 457 271 25

Abstract

Risk is a game in which traditional Artificial-Intelligence methods such as for example iterative deepening and Alpha-Beta pruning can not successfully be applied due to the size of the search space. Distributed problem solving in the form of a multi-agent system might be the solution. This needs to be tested before it is possible to tell if a multi-agent system will be successful at playing Risk or not.
In this thesis the development of a multi-agent system that plays Risk is explained. The system places an agent in every country on the board and uses a central agent for organizing communication. An auction mechanism is used for negotiation. The experiments show that a multi-agent solution indeed is a prosperous approach when developing a computer based player for the board game Risk.

Keywords: Multi-Agent System, Board Games, Distributed Problem Solving, Risk

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Risk is a turn based board game with dice and cards. The game has a huge search space, which has turned the computerized players (bots) to use strategy instead of relying on tree searches. The standard way to create a Risk bot today is to create a monolithic solution.
This is where Multi-Agent Systems (MAS) comes into the picture. MAS uses the concept of autonomous agents to solve problems. Each of these agents will have its own state and make its own decisions. The agents will also communicate with each other and the surrounding environment.
It might be difficult to beforehand know whether MAS is a successful solution or not. By testing MAS solutions in different situations it might be possible to get a better understanding of its potentials.
There have been some previous attempts at making Multi-Agent Systems for playing board games such as, but not limited to, Chess [4], Stratego [16] and Diplomacy [7]. There has been an attempt at making a Risk bot using a neural network [8], but no attempt has yet been made on making a MAS for playing Risk.

1.1 Problem Definition

Is a MAS solution a prosperous approach when creating a Risk bot?
What would the architecture of such a system look like?

1.2 Scope and Limitations

The scope of this thesis is to test how well a MAS Risk bot performs against centralized bots. It is not within the scope to test and/or evaluate if the MAS solution developed is the best possible. Neither is the aim to evaluate how any bot mentioned performs against human players.

1.3 Method

First a prestudy of the domain will be performed to help with design of the MAS architecture. After this a Risk bot will be implemented and evaluated.

The evaluation will be done by running a series of experiments on the bot. The experiments will, among other things, show how well the bot compares to monolithic solutions.

1.4 Outline of the thesis

In the next chapter the game of Risk will be explained and after that in Chapter 3, the design and implementation of MARS will be explained and described in detail. MARS is the bot that is being developed as the main part of this thesis. In Chapter 4 and Chapter 5 this bot will be tested in an experiment and the results presented. In the chapter after that, Chapter 6 the previous mentioned results will be discussed. All the above chapters will be summed up in Chapter 7, the conclusion. In the last chapter, Chapter 8, possible ways to develop MARS further will be suggested.

Chapter 2

Domain

2.1 Risk

Risk is a board game by Hasbro Inc[1]. This board game is all about conquering the world. The game is played on a somewhat simplyfied version of a world map, as can be seen in Figure 2.1. If you are familiar with Risk you may move on to the next section.
A Risk game includes the following items.

- A board
- A deck of cards where each card represents both a country and one of three symbols (cannon, horseman, foot soldier).
- Five dice, whereof three are offensive and two are defensive.
- Armies.

In Risk all armies are of the same type and they are all equal. It is the amount of armies one has that makes the difference.
In the game there can be anywhere from two to six players.
A game of Risk starts out by randomly dividing the countries among the players. This is done by shuffeling the cards and dealing them out to the players. After this the turn order is decided by the rolling of the die. The start up phase is ended by the players taking turns in placing their initial armies onto their countries.
Then it is time for the game phase. In this phase each player takes turn to do their moves. Once a player is finished and does not want to do any more moves, it is the next players turn.
A turn consists of four parts which are performed in order. These are:

- Trade in cards
- Place armies
- Attack
- Fortify

[1]Risk is a registered trademark by Hasbro Inc http://www.hasbro.com.

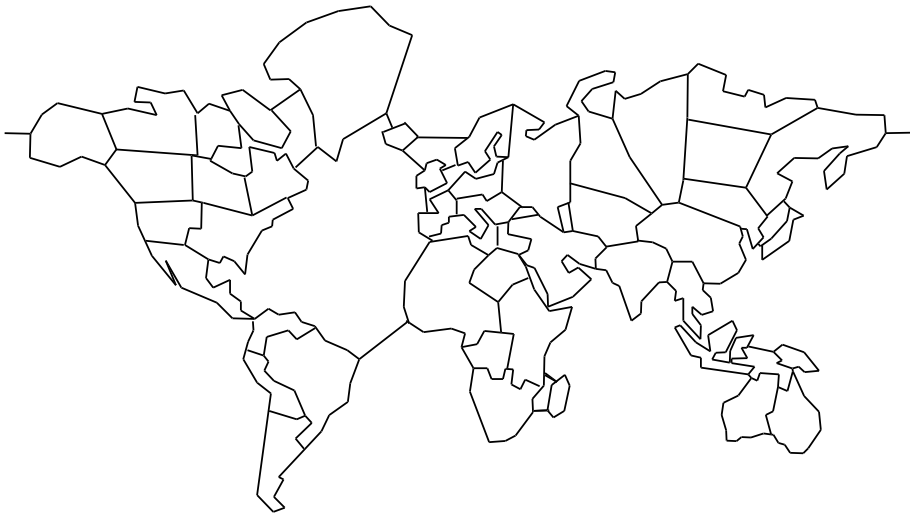

Figure 2.1: World map used in Risk.

2.1.1 Trade in cards

During the game the player will receive cards. Each card has one of the three symbols (Cannon, Horseman, Foot Soldier). There is also two jokers in the deck. A joker can represent any one of the three symbols. If you have three cards with the same symbol or one of each, you can trade them in. You may never end the trade in phase with more than four cards, which means as long as you have five or more cards you must trade in cards. Note that it is always possible to do a trade in if you have five or more cards.
What you get when trading in depends on which rule set you are using. The 1999 rules state that the first trade in done gives the player four extra armies and every trade in after that increases the value by two [5], until twelve is reached, thereafter 15 is given and the increasment is five.

2.1.2 Place armies

At the beginning of a placement phase the player is given a number of armies based on the number of countries that are owned by the player. This number of armies given can be calculated by dividing the number of owned countries by three. The minimum number of armies given is three.
The player will also receive bonus armies for every whole continent owned by the player. Different continents give different number of bonus armies.

- North America 5
- South America 2
- Europe 5
- Africa 3
- Asia 7

- Australia 2

The player now has a number of armies given to him due to trade in of cards and/or ownership of countries. These armies must now all be placed in one or several of the countries owned by the player.

2.1.3 Attack

Once a player has placed all armies the attack phase is entered. In this phase the player may, from any by the player owned country, which has more than one army, attack any neighboring country owned by an opponent. There is no limit to the number of attacks that can be made. The outcome of an attack is determined by the dice. The attacker chooses the number of armies to attack with and rolls that number of dice, but never more than three. The defender rolls a die for each of the armies in the defending country, but never more than two. The highest attacking die is matched with the highest defending die. The one with the lowest value of the two looses an army. In case of a tie the attacker looses. If the defender rolled two dice, the process is repeated with the last defending die and second highest attacking dice.
If the defender runs out of armies the attacker has conquered that country and must decide on the number of armies to bring in into the new country. This number must be no less than the number of dice that were rolled by the attacker in the last attack.
If the defender has no more countries on the board, the player is out of the game. The game continues until there is only one player left in the game.

2.1.4 Fortify

At the end of every turn there is a fortification phase. This phase is sometimes refered to as the "Free Move". This is where the player can move his armies. In the rules there are two different rules when it comes to fortification [5]. One standard rule and one for experts. The standard rule states that the player may move any number of armies from one, and only one, country to one, and only one, neighboring country. This severely limits the troop movement. The expert rule makes it a little bit more easy by allowing any movement as long as it is within a cluster. A cluster is a continuous chain of countries with the same owner. The only exception is that no country may have zero armies after the fortification is done.
There is also a common house rule, listed in the Risk FAQ [12], which is a more realistic version of the expert rule. It states that during the fortification phase any army may move to any neighboring country, but it may only move a total of one step during a turn. As before, no country may have zero armies at the end of the phase.

2.2 Limitations

There are certain limitations beyond the game's rule set that must be taken into consideration. This to make it fair for all parts when going from a real life board game to a computerized game.
Players may not cooperate in any way. No player may in any way signal what

their intentions are in such a way that it affects the actions of the other players. To ensure this, there is to be no communication between the players.
To avoid that a single match of Risk takes too long it is needed to put a limit on the length of a match. This limit will be set to 100 rounds of play, where a round means when every player still in the game has had their turn.

Chapter 3

MARS

MARS, or Multi-Agent Risk System, is the name given to the bot that is the focus of this thesis. This chapter aims to explain how the bot is designed and how it works.

3.1 Environment

The environment that the bot will operate in is within the game server. It is a trusted environment without security issues. All the agents can be assumed to always be up and running and not make any malicious actions.
The game is turn based which means that the agents will not have to run the whole time, but rather only when it is the bots turn to act.

3.2 Multi-Agent System Architecture

When designing a system like this, there are three main focus points.

- Agent distribution: How many agents and what do they control?
- Agent communication: How will the agents communicate?
- Agent negotiation: How will the agents negotiate?

The following three sections will find a solution to each of the three questions above.

3.2.1 Agent Distribution

What should be the range of an agent in MARS? How many agents are needed? The sections below will try to answer these questions by examining the possible solutions.

Task

Having an agent for every task that is needed, such as defensive, offensive, fortification, etc. Risk is turn based and each turn has a four different phases. Any one of these phases would most likely be represented by a Task agent. This

would result in not that many agents and about one or two agents active at the time.

Continent

The only reason for the inclusion of continents in the game is because of the army bonus acquired for holding a whole continent.
Having an agent per continent would result in large amounts of internal activity, as it would, in most cases, contain both its own and hostile countries.

All Countries

No major internal separations except for a few different tasks. Most, if not all, actions are on the country level. One country attacks another, armies are moved from one country to another, etc.
With this distribution each agent could function sufficently with only its local environment available. This means that it only sees and communicates with the direct neighbors.

Own Countries

The only difference between this and All Countries is that with having agents only in one's own countries it would be needed for each agent to have a larger world view. This would be needed for an agent to be able to plan attacks further than one step away.

Army

This would result in a large number of agents. All armies are equal and are worth the same. The only way two armies differ is depending on what country they are placed in.

Discussion

The task alternative would result in a system that would be rather close to a centralized system and therefore not a valid choice since the aim is to make a MAS. Choosing continent would lead to too much internal activity. By removing army as a valid choice, due to the share amount of agents it might conclude to as well as the units being identical, the list is down to two.
The final two have more in common and the solutions would be rather similar. The only real difference would be in their ways to handle enemy countries. The All Countries solution seems more uniform as the alternative suffers from being unable to communicate with countries owned by an enemy.

Conclusion

As most events are on the country level, it seems like a logical choice to have one agent per country. Since information is needed even from enemy controlled countries the choice is to have agents in All Countries.
One possible extension to this would to have an agent per continent as well but this would only result in one extra step when communicating between agents

and have no obvious advantages. Due to the nature of the game, being turn based and every turn having different phases, even the all countries choice seems to have some of the Task solution in it.

Cards

Risk involves some cards that need to be managed. Having an agent per card seems awkward since cards come and go and are paired up in a rather simple way. Another option would be to have a single agent managing the cards, a Card agent.

3.2.2 Agent Communication

In the previous section it was decided that one agent per country would be the best solution. This means that MARS will have at least 42 agents, as there are 42 countries in Risk. From this the next questions will be
How is the interaction between agents and the game server handled?
How will the agents communicate with each other?
As a standard Risk bot is monolithic, it is safe to assume that any game server found will need to communicate through a single point. This can be solved by introducing a Mediator agent [6, 13]. The Mediator will create the country agents and handle all communication between them. This means that the country agents need not know about the other country agents and have one single point for communication. The Mediator is also a layer of abstraction for the game server, letting the game server see MARS as a single bot instead of a bunch of agents.

3.2.3 Agent Negotiation

The typical case of negotiation that will take place in MARS will be to settle disputes such as:

- Who needs this army the most?
- What countries should we attack?
- In what order should these attacks be performed?

Every agent will most likely have their own opinion on what the answer should be, based on their local environment. One way to solve the problem would be to implement some form of auction mechanism. The auction type to be used is a first price sealed bid auction [17].
This will work in such a way that the central agent will put out a request for bids. Upon receiving such a request every country agent will reply by sending a bid. If the bid is a winning bid the agent will receive a message saying that it won the bidding.
This system will be the basis for every negotiation in MARS.

3.3 Design

MARS has three different types of agents, as can be seen in Figure 3.1. The following sections will take a closer look at these and explain how they work.

After that there are three more sections which explain other important functions such as parameters, country evaluation and dice odds.

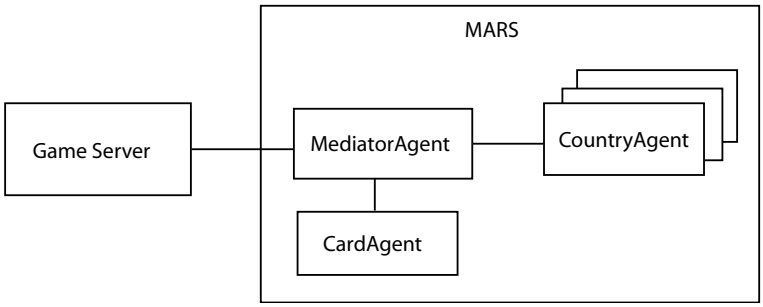

Figure 3.1: MARS Architecture.

3.3.1 Mediator Agent

The Mediator agent is the center of MARS. Every message sent goes through this agent. It receives the calls from the game server about what phase the game is in and handles and coordinates the other agents active in the phase. Generally this follows the pattern shown in Figure 3.2. The Mediator requests

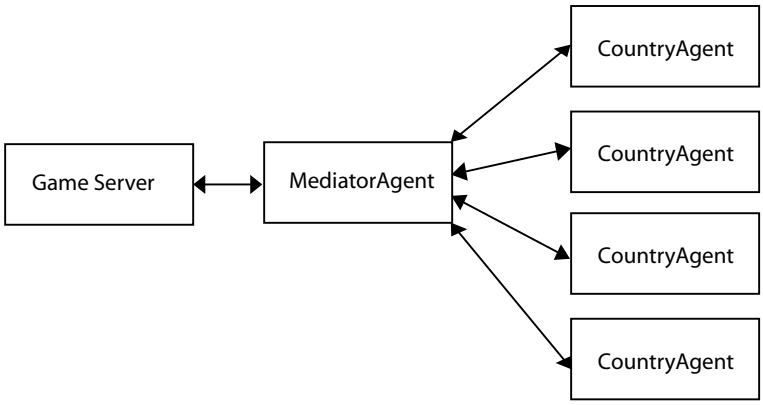

Figure 3.2: Mediator Agent's role in communication.

bids from a set of agents and then selects the winner, thereafter the actions of the winning agent are performed. This is explained in detail in Section 3.4.

3.3.2 Country Agent

In every country on the board there is a country agent. This agent manages that country and its scope is the immediate neighborhood. It will do as it is told by the mediator. When not doing anything it will wait for a request from the mediator. The typical request is a request for a bid, in which case the country agent will calculate a bid and send it back to the mediator.
The type of activity in the country agent depends on what phase the game is

in and if the country is owned by the player or an opponent.
Exactly what the agent does in the different phases is explained in Section 3.4.

3.3.3 Card Agent

There is only one Card agent running at any time. The agent is not in communication with any country agent as the trade in phase is not connected to any of the other actions. The Card Agent is quite simple, but adaptive. Its actions are based on a cost and reward system where a trade in has a default value of five. There is then a cost assigned to using each card where it costs more to use a joker than a regular card. The cost of using a joker is such that it is not rewarded to use two jokers in a trade in as this is considered a waste. In case there are more than five cards it will find the best combination of trade ins based on the highest total reward. This system can by changing the rewards and costs be changed to fit the different possible rules, as the ones described in Section 2.1.1.

3.3.4 Parameters

To make MARS highly configurable and at the same time lessen the affect of the developers own values, there is a high usage of parameters. All parts of MARS rely on parameters when doing calculations. The parameters used can be seen in Table 3.1 and Table 3.2.
Many of the parameters are set by the rules of the game, such as
CARD_COUNTRY_BONUS = 2 which is the number of bonus armies one gets if the country on a traded in card is owned by the player. Others are set to what seems to be a logically good value, such as WIN_PERCENTAGE_FIRST = 0.25 which is how high the odds for a successful attack must be until a country is conquered[1].
There are three main parameters that take a larger part in determining the behaviour of MARS. These are WIN_PERCENTAGE,
PLACE_OFFENSIVE_BONUS and PLACE_DEFENSE_BONUS. The last two are in turn connected to each other in such a way that increasing one will have the same affect as decreasing the other and vice versa. This means that there are in fact only two main parmeters in MARS.

3.3.5 Country Evaluation

In Risk, not all countries are the same. Some countries might be very valuable to have, while others may not. At the same time this value might change over the course of a game. Due to this it is important to have an accurate way of calculating this value. The value will be used to know what countries to attack and what countries to defend.
The value of a country consists of two parts. A static part and a dynamic part. The static part is the same for all countries in the same continent and is based on the continents bonus value V_b, size V_s and the number of borders V_{nb}. The static value V_{sv} of a country is $V_{sv} = V_b/(V_s * V_{nb})$.
The dynamic part is based on what the immediate neighborhood looks like. Things that affect the dynamic value are

[1]Taking a country each turn is important because it is a prerequisite to getting a card

Parameter	Value	Explaniation
PLACE_DEFENSE_BONUS	3.5	Bonus multiplier, higher value makes the bot more defensive.
PLACE_OFFENSIVE_BONUS	170	Bonus multiplier, higher value makes the bot more offensive.
WIN_PERCENTAGE	0.7375	The percentage needed to be sure of a victory.
WIN_PERCENTAGE_FIRST	0.25	The value for WP before a country is taken.
WIN_PERCENTAGE_OVERKILL	0.90	The percentage at which assigning more armies is considered overkill.
CARD_SET_VALUE	5	The reward for turing in a set
CARD_USAGE_COST	1	The cost of using a card
CARD_USAGE_JOKER_COST	2.5	The cost of usin a joker
CARD_COUNTRY_BONUS	2	The reward for owning the country on a card
KILL_ELIMINATET_BONUS	20	The value of eliminating an opponent
KILL_CARD_VALUE	5	The value of each card stolen
FORTIFY_TRANSPORT_COST	2	The cost of an army not making it to its destination
GOAL_LENGTH	5	Maximum length of a goal path

Table 3.1: The non-country value related parameters used in MARS.

- How much of this continent do we own V_{cp}
- How many friendly neighbors V_{fn}
- How many armies do the friendly neighbors have V_{fnu}
- How many enemy neighbors V_{en}
- How many armies do the enemy neighbors have V_{enu}
- How many continents does this country border V_{cb}
- Do we own the whole continent except this country V_{oc}
- Does an enemy own the whole continent V_{eoc}

Where V_{oc} and V_{eoc} have the value one if the whole continent is owned and the value zero if not. Using the values mentioned above, the parameters from Table 3.2 and the notation from Equation 3.1 the value of a country V_c can be defined as shown in Equation 3.2.

$$P_x = V_x * C_x \tag{3.1}$$

$$V_c = P_{sv} + P_{fn} + P_{fnu} + P_{en} + P_{enu} + P_{cb} + V_b * (V_{cp} + P_{oc} + P_{eoc}) \tag{3.2}$$

Parameter	Value	Explaniation	Den. by
CV_STATIC_BONUS	70	Bonus multiplier, used to adjust the static value of a country	C_{sv}
CV_FRIENDLY_NEIGHBOR	1.2	The value of having a friendly neighbor	C_{fn}
CV_ENEMY_NEIGHBOR	-0.3	The value of having a enemy neighbor	C_{en}
CV_FRIENDLY_NEIGHBOR_UNIT	0.05	The value of each neighboring friendly unit	C_{fnu}
CV_ENEMY_NEIGHBOR_UNIT	-0.03	The value of each neighboring enemy unit	C_{enu}
CV_CONTINENT_BORDER_BONUS	0.5	Bonus for bordering another continent	C_{cb}
CV_OWN_CONTINENT_BONUS	20	Bonus for owning the whole continent	C_{oc}
CV_ENEMY_OWN_CONTINENT	4	Enemy owns the whole continent	C_{eoc}

Table 3.2: The country value related parameters used in MARS. COUNTRY_VALUE is abbreviated to CV in the parameter name column.

3.3.6 Dice Odds

A large part of the odds calculations in Risk is based on dice. For every army knocked off the board at least two dice must have been rolled. Since all conflicts in Risk are solved with the die, one can gain an advantage by knowing the odds. The following two sections will explain how the odds were calculated and the third will explain how it was actually used in MARS.
With a battle is meant a single roll of one to three dice by the attacker and one to two by the defender. This process is usually repeated when the defender and attacker both have multiple armies. Several consecutive battles between the same two countries is here referred to as a conflict.

Battle

For a battle it is possible to do a complete search and retrieve the exact odds [11]. This is possible since there are a rather limited number of outcomes. A maximum of five dice will be used, the attacker can use anything from one to three dice. and the defender one to two. This gives six different starting scenarios.
By calculating the outcome of every possible dice combination, the odds in Table 3.3 and Table 3.4 are given.

Conflict

Using the odds from the previous section it is possible to get the odds for all possible outcomes of an entire conflict. What is needed is the odds that for example, the attacker will win an attack with six armies against five defending

	attack:1 defence:1	attack:2 defence:1	attack:3 defence:1
Attacker wins:	15/36 (42%)	125/216 (58%)	855/1296 (66%)
Defender wins:	21/36 (58%)	91/216 (42%)	441/1296 (34%)

Table 3.3: Dice odds for battles with one defending die

	attack:1 defence:2	attack:2 defence:2	attack:3 defence:2
Attacker wins:	55/216 (25%)	295/1296 (23%)	2890/7776 (37%)
Defender wins:	161/216 (75%)	581/1296 (45%)	2275/7776 (29%)
Lose one each:		420/1296 (32%)	2611/7776 (34%)

Table 3.4: Dice odds for battles with two defending dice

armies.
This will be used very often and therefore needs to be fairly fast. The first attempt solved the problem by using recursion. This proved very slow when calculating with larger armies.
Another way is to solve the problem in a more visual way. Imagine a 2D grid where the size of each side corresponds with the number of armies for the attacker and defender. Set the top left square to have a value of 100. The values in the grid are the odds that that specific situation will occur. With the start situation always being 100%. Now using the odds from Table 3.3 and Table 3.4 it is possible to fill in the rest of the values in the grid. Starting in the top left corner and taking it one step at a time towards the bottom right corner. In Table 3.5 shows such a grid for a conflict where the attacker has six armies and the defender has five. The empty squares are situation that will never occur. In Table 3.5 it is possible to for example see that there is a 14.68% chance that a six versus five conflict will end four to zero. In such a grid the sum of the

	6	5	4	3	2	1	0
5	100.00		29.26		8.56		3.84
4		33.58		19.65		8.52	6.35
3	37.17		33.02		18.21	2.17	9.78
2		24.96		25.69		13.97	10.41
1	13.81	4.70	22.25	7.57	15.35	10.02	5.85
0	9.11	12.38	14.68	14.54	8.88	4.18	

Table 3.5: Grid showing the odds for the different outcomes, in percent, in a six versus five conflict.

bottom row will contain the odds that the attacker will win. In the same way the sum of the column on the far right will contain the odds that the defender will win the conflict.

3.4 Actions

During each turn there are four phases. The rules and how they work are described in Chapter 2. In this section it will be explained what happens in MARS during each phase and how the agents work together.

3.4.1 Trade in cards

At the beginning of each turn the card agent will be asked if it is favorable[2] to trade in any cards. If it is, or must done due to having five or more cards, a trade in will be performed.
The strategy will be different depending on which of the rules is used. If the main rule with increasing value for every trade in is used it is better to wait as long as possible before trading in cards. While if the house rule with constant value for a trade in i used it is no use waiting and more favorable to trade in as fast as possible.

3.4.2 Place armies

This phase starts by the Mediator receiving a call from the game server that it is to place a certain number of armies.
The Mediator first sends out a message to all agents that are not owned by MARS. These agents will then tell all their neighbors about the value of owning that specific country and how many armies are defending it. When it receives such a message from a neighbor it will append its values and send the message on. The message will be discarded if the chain of values is longer than GOAL_LENGTH. This is to reduce runtime as the total number of goals grows exponentialy with the maximum length of a goal[3]. When a country that is owned by MARS receives such a list it will store it in the goal list. The list will contain all possible goals that that country can possibly achieve.
After this each agent that is in a country owned by MARS will recieve a request to bid for a certain number n armies. Each of these agents will then calculate both offensive and defensive bids for zero to n armies. A bid for zero armies is included to get bids for goals that require no extra armies. Only goals that win a bidding will be persued, therefore it is important to include bids for zero armies. Goals that can be fulfilled with no extra armies include situations where a country already has a large amount of armies.
The agent now iterates from zero to n and in every step it checks the entire goal list to see what the chances are to accomplish each goal p, what the goal is worth w, how well will it be able to defend the final goal d. This can be summed up as Equation 3.3. For the best offensive bid b_o it will also remember the goal associated with that bid.

$$b_o = \max_{i=0}^{n} \begin{cases} p_i w_i d_i & i = 0 \\ \frac{p_i w_i d_i}{i} & i \geq 1 \end{cases} \tag{3.3}$$

The agent will then again iterate from zero to n. For every step the agent will calculate how well it can defend itself from its current enemy neighbors d with

[2]Based on the cost and reward system explained in Section 3.3.3.
[3]Given that the cluster of enemy countries is large enough.

the current armies m plus the number of armies to bid for. This combined with the value of the country v gives us the best defensive bid b_d as can be seen in Equation 3.4.

$$b_d = \max_{i=0}^{n} \begin{cases} v * d_{(i+m)} & i = 0 \\ \frac{v * d_{(i+m)}}{i} & i \geq 1 \end{cases} \tag{3.4}$$

Each bid value is on a per army basis. This makes them easy to compare.
Once all this is done, the agent will take the maximum value of b_o and b_d and submit it to the Mediator. The Mediator will go through all bids received and select the best one. It will then tell the winning agent that it won and assign it the number of armies. This process can be seen in Figure 3.3. If this number is less than n the process will be repeated with the remaining armies.

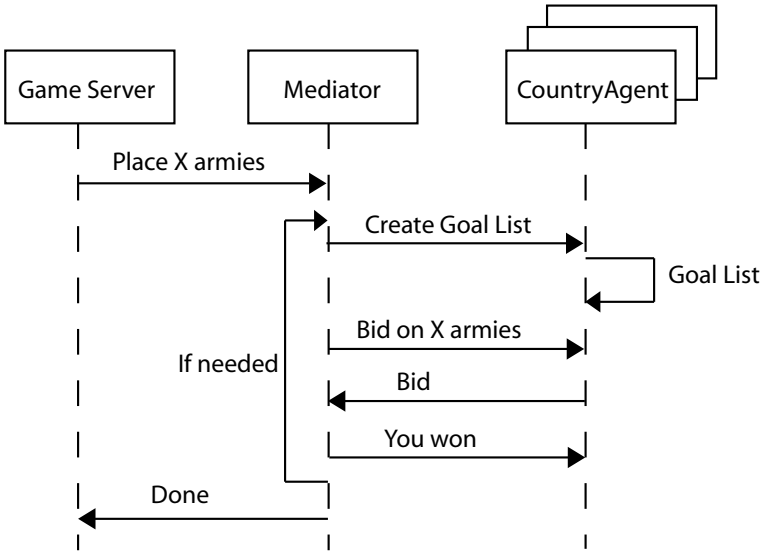

Figure 3.3: Communication during placement phase.

3.4.3 Attack

Once the attack phase has started the Mediator will ask all eligible agents to submit bids for attacking. To be eligible an agent needs to be in a country owned by MARS, have an enemy neighbor and have more than one army.
The agent with the highest success odds, as long as it is higher than WIN_PERCENTAGE, will get allowed to perform its attack. If the attacking country wins the conflict it will send the goal path to the conquered agent, which will accept it as its own goal.
The process is repeated as long as the winning bid is high enough. Figure 3.4 shows the process.

3.4.4 Fortify

The fortification phase is a little bit more complicated. It starts out by the Mediator retrieving a list of all owned clusters. A cluster is a chain of all

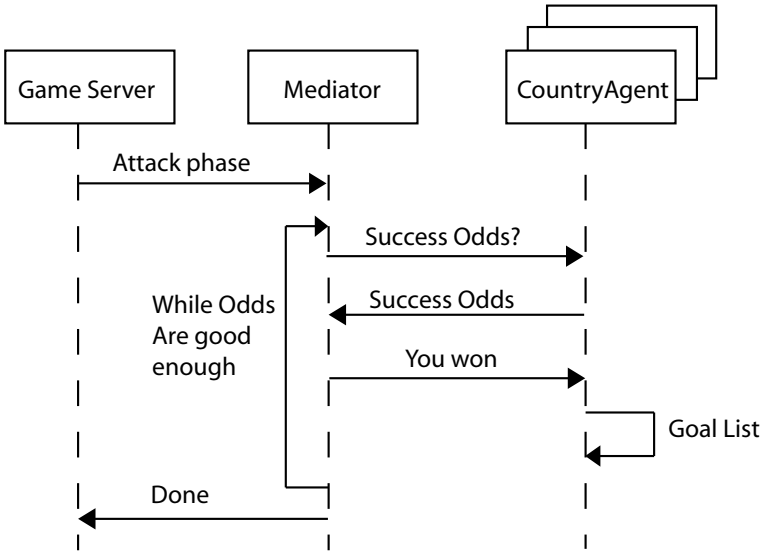

Figure 3.4: Communication during attack phase.

connected countries with the same owner.
Then the Mediator will step through every country in the cluster and see if any one of them has any armies to spare. The number of armies a country can spare depends on its environment, how strong the enemies are, if the country is a border or if the country is on the border between continents. When this is done the Mediator has a list of all possible sellers in the cluster.
For each seller in the list there will be an auction, almost like the one in during the placement phase. This time only countries in the same cluster as the seller will be able to bid. The seller will also bid on its own spare armies to make sure that the country that is in the most need of them gets them.
There is one more level of difficulty as each army is only allowed to move one step. To solve this each bid also includes a transport cost. This cost is based on how many armies can be moved all the way to the destination and how many moves are needed. This information is needed both for calculating the transport costs and to do the actual moving of the armies. It is a difficult task to solve but luckily it is a well studied task, since a max-flow algorithm may be used. It calculates, in our case, how many armies can move from one country to another. The max-flow algorithm used in MARS was developed by Edmonds and Karp [3] and is a breadth first implementation of the Ford-Fulkerson algorithm [15]. It works by repeatedly doing a breadth first search to find the shortest path, until there are no more paths.
This whole process is repeated first for every seller in the cluster and then for every cluster. The process can be seen in Figure 3.5.

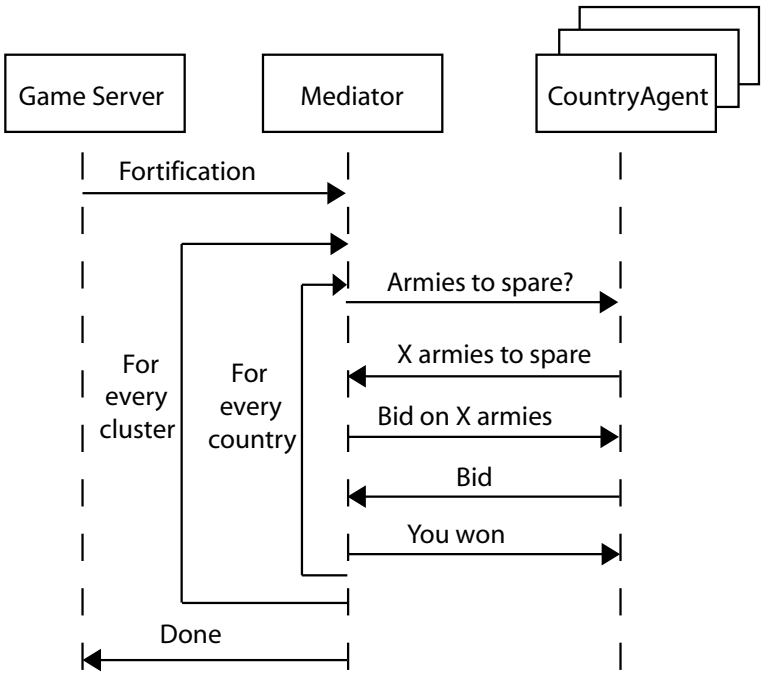

Figure 3.5: Communication during fortification phase.

Chapter 4

Experiments

To test MARS we will need to run some experiments that will show how well it performs. Performance can be classified in many different ways, but when it comes to Risk the main thing is to win, as the game is all about world conquest and winning. The experiments will also test other issues such as speed and stability.
Risk is a game where chance, in the form of dice and cards, plays a specific role. The outcome of a game is also affected by what strategies the opponents use. To stop chance from being a major factor in the outcome of the experiments, it is needed to play many matches. These matches take time to run and it is therefore important to find a balance between running enough matches and still being within a reasonable time frame.
The experiments are divided into two different parts. The first part will aim to set the main parameters used to an optimal setting. These parameters are the ones that are described in Section 3.3.4. The second part is to actually measure how well MARS performs compared to existing monolithic solutions.
This chapter explains how the experiments will be performed and what the circumstances are. The outcome of the experiments are then presented in the next chapter.

4.1 Experiments Setup

As stated in Section 3.3.4, MARS relies on parameters to ensure that the actions taken are not affected by the developers own values. To further ensure this it is vital that the parameter values are set in such a way that the bot performs its best. This will be done in the first experiment, the Parameter Optimization.
When the parameters are set, MARS is ready for the second and final experiment. The goal of this experiment is to test and compare the performance of MARS with the other bots. Read more about this experiment in the Performance section.

4.1.1 Environment

To test MARS it is first needed to find a suitable environment for implementation. This environment must include as many as possible of the criteria. The

following items are needed:

1. A Risk game
2. Possibility to add an externally developed bot
3. A variety of existing bots

A wide selection of Risk games can be found for almost any platform. Almost all of these are official games developed by or with the approval of Hasbro. These games are aimed for the casual gamer do not fill criteria number two.
A possible option would be to implement not only MARS but also the game environment in which it operates. However this does leave criteria number three unfulfilled.
A solution is to look at the Risk clones. A clone is a game that contains most functions of the original but has a different name, to for example avoid legal issues. Some of these Risk clones serve a different audience than just the casual gamer. With the ability to add bots, new levels and themes they have found their own niche. Among these Risk clones the four below where found to allow third party development of bots.

JavaRisk

JavaRisk [10] is not really a Risk clone as it is only roughly based on Risk. This means that its focus lies not on being a Risk game but rather on adding new game play to the game. JavaRisk has three different bots as default with at least one more bot [1] being available.

JRisk

JRisk [9] is a fully featured Risk game that follow the function of the original very well. The game only has two computerized players.

KsirK

KsirK [2] is a another Risk like game. According to the documentation there is only one bot available and it chooses its actions randomly.

Lux

Lux [14] is a Risk game by Sillysoft[1]. As it is a commercial product it seemed more complete than any of the other three. Lux even has a plugin manager that allows for easy installation of third party developed components. In Lux there are as default twelve different bots and with one more being available via the plugin manager.
Lux does not totally comply with the current Risk rules. There are two minor differences between the two. These affect:

- The value of cards traded in
- Fortification

[1] http://www.sillysoft.net

In Lux, using the default settings, all traded in cards will give the player the same amount of extra armies. This amount is five. The value of the trade in stays the same and is not based on some kind of sequence.
When it comes to fortification Lux uses the house rule mentioned in Section 2.1.4.

Conclusion

Lux was chosen to be the platform, on which to perform the experiments, because it was the best match based on the criteria. The other three just do not have enough bots. The only thing that is not optimal with Lux is that the rule set is not exactly the same as the current Risk rules but the differences are minor and does not affect the experiment.
Lux allows and encourages development of bots. It has a wide variety of bots with it from the start and there even exists bots developed by enthusiasts.

4.1.2 The Opponents

In the experiments there are 13 different bots. One of these 13 is MARS, which is thoroughly described in Chapter 3. The other twelve will be listed here with a short analysis of each. The first eleven in the list are included with Lux by default. The last one, Nefarious[2], was developed by an enthusiast named Paul Brotherton. Nefarious was acquired through the use of Lux's plugin manager, a system for downloading additional components into the game.
The following analyses are done by reading the source code to figure out how the bots behave. Most of these behaviors are stated as comments in the source code. The code comes with the LuxAgent SDK[3] and is freely available. The exception to this is Nefarious, where the source is not available. In this case a decompiler was used to retrieve the source. The downside to this is that the comments are lost.
Each bot is ranked and put into one of three categories. These are Easy, Medium and Hard. This ranking is done by Sillysoft. It is not stated how the ranking is performed.
The bots all inherit from each other or some kind of help class. There are two help classes that a bot can inherit from. It is either LuxAgent which is a very basic class without any actual code in it, or SmartAgentBase which inherits from LuxAgent but has help functions to help the developer with things such as finding weak targets or lucrative goals. Figure 4.1 shows how the bots are related by inheritance.

Angry

Inherits from: LuxAgent
Lux Difficulty: Easy
Angry is a very simple bot that has attacking as its main strategy. Angry will place all armies in the country that has the most enemy neighbors. During

[2]Version 1.1
[3]Version 2 of the SDK was used.

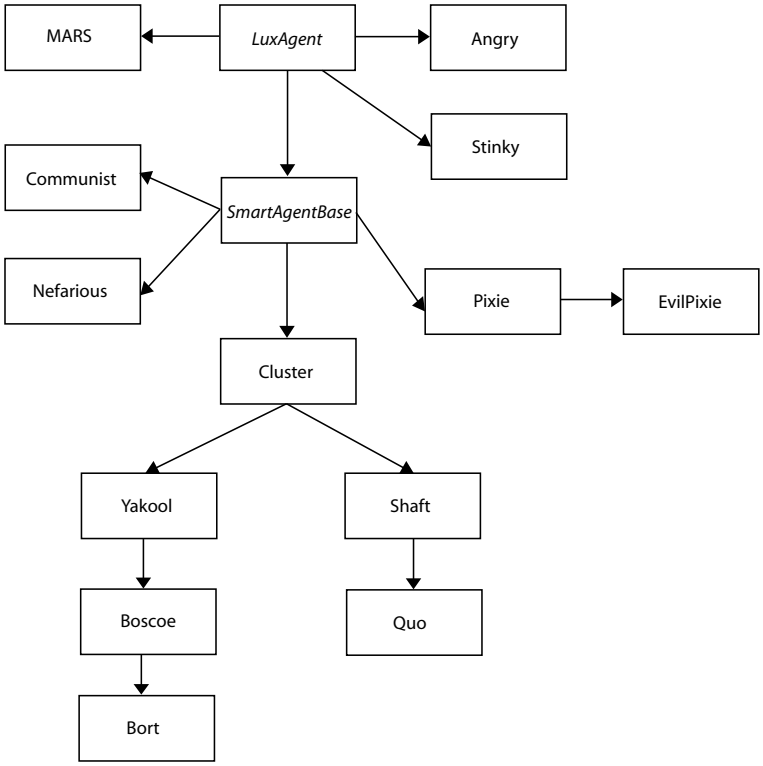

Figure 4.1: Tree showing the inheritance of all the bots.

the attack phase every country that Angry owns, will attack its weakest neighbor. But it will only attack if the attacking country has more armies than the neighbor. If it takes a country all armies will be moved to the country which has the most enemy neighbors. The same thinking goes on in the fortification phase where any country will give all its armies to any neighbor that has more enemies.

Communist

Inherits from: SmartAgentBase
Lux Difficulty: Easy
Communist uses the concept that all countries are equal. Communist will place its armies one by one, placing each one in the weakest country. The bot will attack the weakest neighbor it has on the board, but only if it has more armies than the opponent. When it comes to moving or fortification the goal is that every country should have as many armies as its friendly neighbors.
Another feature that Communist has is that it will not attack another player if it also is a Communist. At least not until there are only Communist players left in the game.
It might seem as if Communist violates the set rule that bots should not cooperation. This will only be the case if there is more than one Communist in a particular game. This scenario will not occur in the experiments.

Stinky

Inherits from: LuxAgent
Lux Difficulty: Easy
Stinky uses randomness as its guidelines. It will place all of its armies in a random country, as long as the country has enemy neighbors. The attack phase is less random as all countries will attack its weakest neighbor as long as it has more than 1.5 times as many armies as the neighbor. When moving armies into a conquered country it will move all armies into the country that has the weakest neighbor. Stinky ignores the fortification phase.

Cluster

Inherits from: SmartAgentBase
Lux Difficulty: Medium
As the name suggests Cluster is all about owning continuous chains of countries. When placing armies, it will spread them along the border of the best cluster. The clusters are rated by how big the largest owned continent is. Attacks are done by countries in the best cluster. Border countries will attack neighbors that have less armies. Moving in armies after conquering a country is decided by who has the weakest neighbor. During fortification, armies will be moved outward in the clusters towards the borders.

Pixie

Inherits from: SmartAgentBase
Lux Difficulty: Medium
Pixie is more continent oriented. At the beginning of each turn Pixie will evaluate for each continent if it is worth spending armies on. The armies are then distributed accordingly. During the attack phase it will step through these continents and perform every possible attack where it outnumbers the enemy. The number of armies to move into a conquered country is primarily based on the number of neighboring enemies the countries have and secondly on which one is closest to a wanted continent. Fortification in owned continents is done by moving armies toward the borders. If Pixie does not own the continent the armies in it will be moved towards the weaker enemies in the continent.

Shaft

Inherits from: Cluster
Lux Difficulty: Medium
Shaft improves upon the Cluster design by making the attack phase more advanced. The rest of the phases work just like Cluster. Shaft does a further analysis of what the world map would look like after it has taken one or several of the countries on the border of the cluster. With focus on making the border smaller by expanding to countries with fewer borders.

Yakool

Inherits from: Cluster
Lux Difficulty: Medium

Yakool also uses Cluster as a base but takes the process in a different direction than Shaft. Yakool does not like it when the other players get too strong, therefor it primarily attacks any opponent that is considered too dominant on the board. If no such opponent is found it uses the attack pattern of Cluster only a bit less aggressive, but makes sure to try to at least get a card.

Boscoe

Inherits from: Yakool
Lux Difficulty: Hard
Boscoe is a small addition to Yakool. It is ever less aggressive. This is done by removing some of the conditions that made Yakool attack. Boscoe still tries to get a card every round.

Bort

Inherits from: Boscoe
Lux Difficulty: Hard
Bort follows in the footsteps of Boscoe but takes it one step further, by also slowing down the expansion of the cluster. Only one country is taken every round. If a country is taken to get a card, Bort will move in less armies than its predecessors did.

EvilPixie

Inherits from: Pixie
Lux Difficulty: Hard
EvilPixie works pretty much the same way as Pixie does. There are differences in the behaviour. When placing armies EvilPixie will, unlike Pixie, take strong opponents into calculations. The same happens in the attack phase where EvilPixie will attack the player that is too strong. If no such player exists, it behaves just like Pixie. The move in strategy used after conquering a country is identical to Pixie. During fortification EvilPixie does differ from its predecessor as it is a bit more defensive and can move armies in continents it does not own to aid in defending a nearby continent it does own.

Quo

Inherits from: Shaft
Lux Difficulty: Hard
Quo is a based on Shaft and basically only adds one feature. Shaft did not focus on getting cards. As cards can be very important this seems to be the flaw that put Shaft in the Medium category instead of Hard. Quo fixes this and makes sure that if no countries are taken it will lower its standards and make a few extra attempts at getting a card.

Nefarious

Inherits from: SmartAgentBase
Lux Difficulty: Not Available
Nefarious is different because it was not developed by the same people as the

rest of the bots. When placing armies its decisions depend on what continents are owned. If it owns no continent, it will place the armies close to the continent which is easiest to take. If it does own at least one continent, it will just as Cluster spread the armies on the border of the best cluster. Just like Yakool, Nefarious will look for players that become too strong and will attack that player no matter what. If no such player was found it will slowly expand its main cluster or at least try to get a card. When conquering a country the movement will be based on what country has the most enemy neighbors. For fortification Nefarious uses the same strategy as Cluster.
Nefarious does one thing that the other eleven do not. It chats. In a networked game the players can send messages to each other during the game. Nefarious will use this to send a randomly selected message that is relevant to the situation. Thus making it seem more human. For example, if it conquers the last of another players countries it might say that it did it for the cards.

4.1.3 Parameter Optimization

Parameter Optimization with multiple parameters may be a difficult task. Especially in a environment like this where it is needed to run many tests due to the randomness of the game. Each setting might require running several hundred matches before a reliable evaluation can be made. This limits the number of setting that can be tested as running tens of thousands of matches to find the optimal settings is unreasonable within the time frame of this thesis.
Since two of the parameters are more dominating than the others there is a simple way to optimize those two. With only two parameters it is possible to create a 2D map of a range of values for the two. By finding the area of a maximum and zooming in and repeating the process, the parameters get set to an ever more optimal setting. This method does require both parameters to be continuous. Which they can be assumed to be. The step by step process is:

1. Find the minimum and maximum values of both parameters.
2. Divide each parameters range steps. Making a grid with a certain number of intersections.
3. Run the tests and fill in the values.
4. Find the area of the maximum value and set the range of the parameters to enclose it.
5. If there is a need to zoom further, goto step 2.

The values in step 3 is the number of matches MARS wins. This is because the game of Risk focuses on winning but also because these parameters have very limited effect, if any, on things such as stability and runtime. To retrieve the number of wins, MARS is set to play a certain number of matches against five other randomly selected bots.

4.1.4 Performance

Once the parameters are optimized MARS is ready to be tested for performance. Focus in this section is on three parts, wins, stability and runtime. From each

match the following data will be collected:

- What bots are in the match
- The order in which the bots were eliminated from the match
- The time it took to play the match
- Run-time errors in the form of exceptions
- If there was a stalemate or the match was not completed

A stalemate is when two or more bots are still in the game but there is no progress. The bots just build up their armies and do not attack. This is defined as a match that lasts more than 100 turns. When this happens the match is stopped and marked as a stalemate.

Wins

From the data collected a simple calculation will show how many percent of the matches of a bot it wins. Also calculated are the percentage of the other positions.

Stability

There are two ways to determine instability from the data collected. One way is by counting the number of unhandled exceptions thrown. The other is by counting the number of matches that were not completed. This happens if a bot goes into a loop without exiting from it.

Runtime

With the bots in the match and the runtime, it is possible to sum up a total of a bots matches and total runtime. With the average runtime it is possible to compare the bots to see what bots use the most CPU resources. Only completed matches will be included in the runtime calculations. In other words, matches that do not end due to stalemate or bugs will be ignored.

Experiment 1

In the first experiment MARS will be tested by playing against in matches with the other twelve bots. This list of bots consists of the ones covered in Section 4.1.2. Testing all $\frac{13!}{7!}$ different combinations of bots is unrealistic as it would take more than three years[4] to run the experiment. A way to limit the number of matches is by ignoring the order. This would bring the number of matches down to $\binom{13}{7} = 1716$. This way all the bots will get to play the same number of matches and will all play against every combination of opponents.
This experiment will show how well the bot performs against other bots of uneven difficulty.

[4] with an estimated average match time of over one minute

Experiment 2

In the second experiment another 1000 matches will be played, but this time MARS will be tested against a fixed set of opponents. The opponents will be the five best bots from Experiment 1. This in order to see how the bot perform with only the best bots.

Experiment 3

The last experiment will aim to test MARS in a more ecological way. How well does it perform in an environment with bots of the same type or that use the same strategy. This will tested by running matches with either two or three instances of MARS. The opponents will be selected randomly, with no bot selected twice in the same match. 500 matches with two instances of MARS and 500 matches with three will be run.[5]

[5] An alternative way of doing this experiment could be to do a population dynamics study where each of the six positions in a game would be selected by random from a population. The population would be based on the percentage of wins that the different bots have. However such an experiment would be affected by the fact that Communist has cooperative code.

Chapter 5

Experiments Result

This chapter will present the results of the experiments described in the previous chapter. Chapter 6, will contain all the interpretation of what and why the numbers are what they are.

5.1 Parameter Optimization

As mentioned in Section 3.3.4 there are two main parameters. These are PLACE_OFFENSIVE_BONUS (POB) and WIN_PERCENTAGE (WP).
POB is dependant on PLACE_DEFENSIVE_BONUS (PDB), but with the current setting of PDB at 3.5, POB is thought to have a valid range of 1 to 240. Because of this five steps were selected, 40, 80, 120, 160 and 200.
WP has a range that lies between WIN_PERCENTAGE_FIRST (0.25) and WIN_PERCENTAGE_OVERKILL (0.90). By selecting four different intervals inbetween we get 0.38, 0.51, 0.64 and 0.77. There will be 200 matches run with each setting. The first level, as seen in Figure 5.1, shows an optimum somewhere around POB=160 and WP=0.64. Zooming in on this by using the POB range

	40	80	120	160	200
0.38	32%	37%	41%	35%	39%
0.51	37%	38%	39%	38%	43%
0.64	37%	41%	42%	48%	38%
0.77	35%	41%	40%	41%	44%

Table 5.1: A five by four grid of the win percentage with the parameter values PLACE_OFFENSIVE_BONUS (40-200), WIN_PERCENTAGE (0.38-0.77)

130-190 and WP range 0.5425-0.7375 gives us Table 5.2. Figure 5.2 shows an area of four squares with about the same values. Columns 170 and 190, and rows 0.6725 and 0.7375. It was decided that the detail of the setting was not going to get more accurate than this when running 200 matches. One of the values is a little bit higher and was selected as the optimal setting. The selected settings are POB=170 and WP=0.7375.

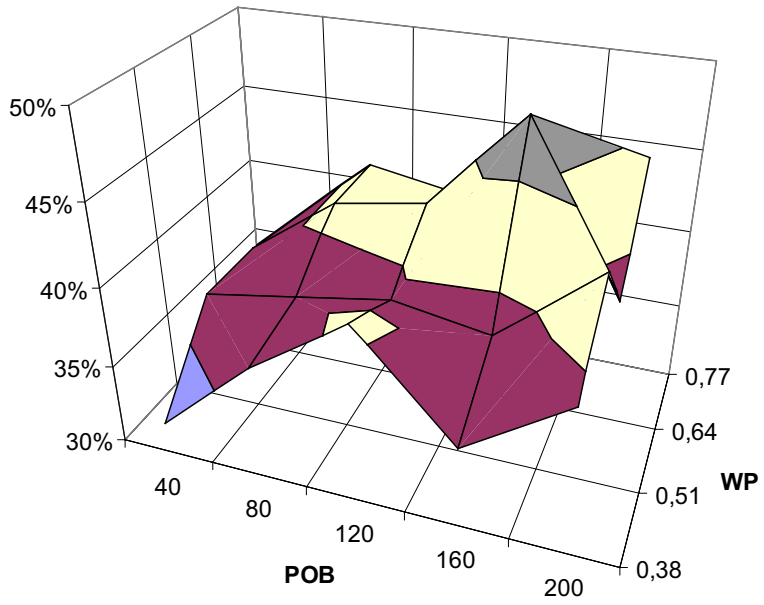

Figure 5.1: Win percentage with the parameter values PLACE_OFFENSIVE_BONUS (40-200), WIN_PERCENTAGE (0.38-0.77). See Table 5.1 for actual values.

	130	150	170	190
0.5425	41%	46%	44%	45%
0.6075	40%	34%	43%	43%
0.6725	45%	44%	48%	48%
0.7375	44%	40%	49%	46%

Table 5.2: A four by four grid of the win percentage with the parameter values PLACE_OFFENSIVE_BONUS (130-190), WIN_PERCENTAGE (0.5425-0.7375)

5.2 Performance

The performance is divided into three different experiments. The results from each one will presented in their own section.

5.2.1 Experiment 1

The first experiment involved playing 1716 matches.

Wins

MARS did come out on top in this experiment and did win more matches than any other bot, as can be seen in Table 5.3. This magnitude is even more visible in Figure 5.3 which only shows the wins.
When looking at both first and second placements, MARS scored an amazing

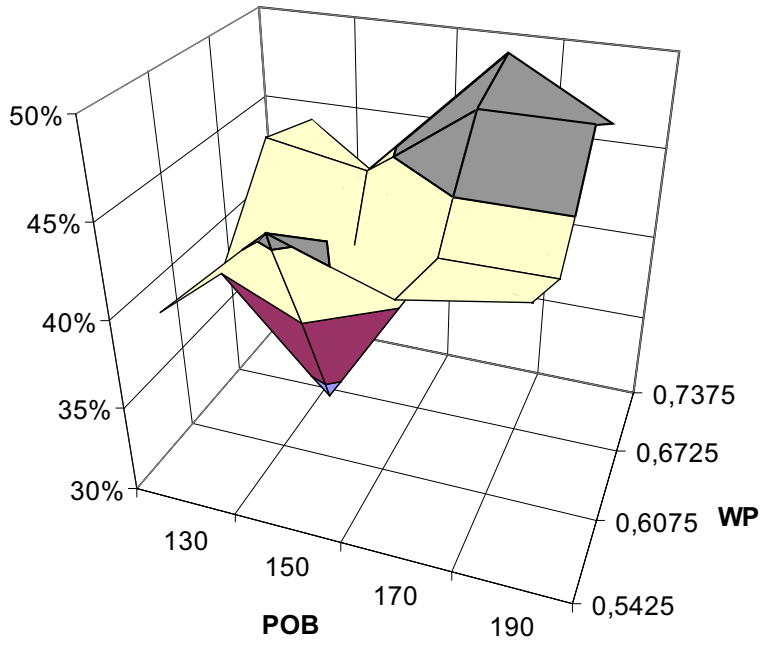

Figure 5.2: Win percentage with the parameter values PLACE_OFFENSIVE_BONUS (130-190), WIN_PERCENTAGE (0.5425-0.7375). See Table 5.2 for actual values.

507 out the total of 792 played matches. This is almost as much as Bort and EvilPixie combined.

Stability

In this experiment there were exceptions thrown in a total of 127 matches. 22 out of these were due to bug in the Card Agent. The other exceptions were thrown by EvilPixie which seems to have a bug in the fortification phase. EvilPixie threw 106 exceptions.
By summing it up, you find that there was 1 game in which both the bots threw exceptions.
The exceptions thrown were not critical and the bots could continue to play the match. Out of the 22 matches where MARS threw exceptions, it won 14.

Runtime

As it can be seen in Figure 5.4 MARS has a longer average runtime than any other bot. Shaft however does seperate itself from the rest with the second highest average runtime.

5.2.2 Experiment 2

In experiment 2, 1000 matches with the top six bots from experiment 1, were run.

Name	1st	2nd	3rd	4th	5th	6th	Stalemates
MARS	338	169	60	62	69	84	10
Bort	206	47	130	156	149	100	4
EvilPixie	205	56	105	153	138	133	2
Boscoe	184	52	142	174	157	76	7
Yakool	135	57	118	162	167	146	7
Quo	177	103	102	132	134	142	2
Pixie	118	153	102	123	139	154	3
Shaft	117	370	157	61	36	38	13
Cluster	100	137	165	161	118	105	6
Nefarious	84	115	128	141	168	156	0
Communist	16	278	199	130	85	74	10
Stinky	7	139	204	141	147	149	5
Angry	6	17	89	113	207	358	2

Table 5.3: The outcome of experiment 1. Each bot played 792 matches.

Number of Bots	Stalemates
2	8
3	8
4	5
5	1
6	1

Table 5.4: The number of bots involved in each stalemate in experiment 1

Wins

The bot that obviously had more problems with harder competition is EvilPixie which dropped from a third place in experiment 1 to a fifth place here in experiment 2. This can be seen when comparing Table 5.5 from experiment 2 with Table 5.3 from experiment 1. Figure 5.5 shows that MARS still wins more matches than any other bot.
Just as in experiment 1, MARS did very well when combining first and second placements. In experiment 2, MARS came first and second a total of 588 times. This is alot compared to Boscoe and Bort who had a total of 339 and 326 repespectively.
In this experiment there were a total of eleven stalemates. As can be seen in Table 5.6 the most common stalemate involved three bots.
In Figure 5.6 we can see that MARS plays unlike any of the other bots. Not only is the number of wins higher but the number of third and fourth places are much lower than those for the other bots.

Stability

Just like in experiment 1, all the exceptions thrown were caused by either the Card Agent bug in MARS or the fortification bug in EvilPixie. In experiment 2, a total of 130 matches had exceptions thrown. MARS threw 13 and EvilPixie threw 120. Which makes it three matches where both threw exceptions.

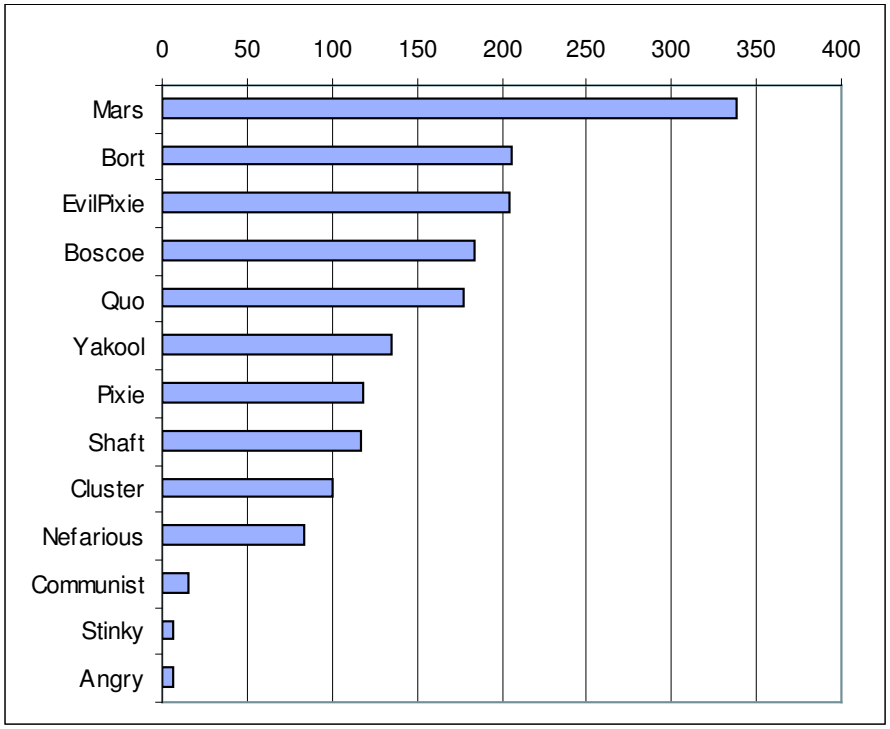

Figure 5.3: The number of wins each bot had in experiment 1. See Table 5.3 for actual numbers.

Like experiment 1, the exceptions thrown in experiment 2 were not critical and the bots could continue to play in the matches where exceptions were thrown.

Runtime

Due to the nature of this experiment each and every bot has the same average runtime. 1000 matches were played and the total time was 50,868 seconds. Which makes the average runtime 50.9 seconds. This is actually much lower than average runtime of MARS in experiment 1 which was 105.6 seconds.

5.2.3 Experiment 3

In the third and final experiment 1000 matches were run. Half with two instances of MARS present and half with three instances.

Wins

Experiment 1 showed that MARS wins about 43% of its matches against a selection of opponents with mixed difficulty. If MARS played equally well against any competition, ie. the win percentage would remain at 43%, it would with

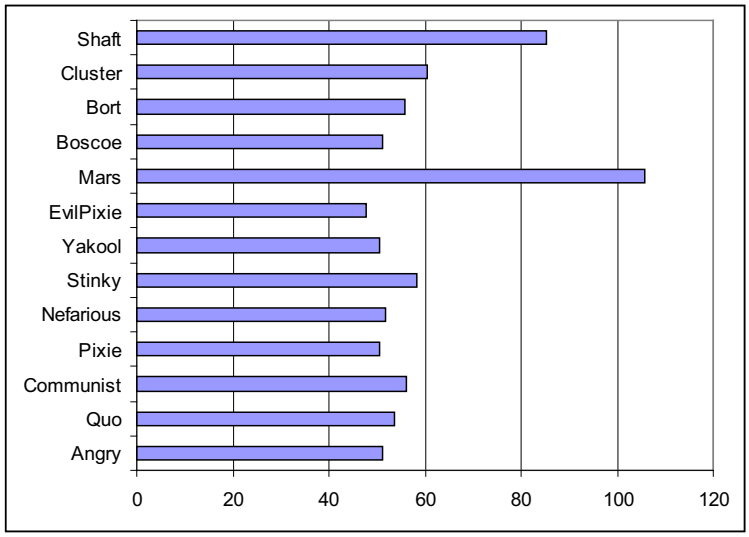

Figure 5.4: The average runtime, in experiment 1, for each bot in seconds.

Name	1st	2nd	3rd	4th	5th	6th	Stalemates
MARS	286	302	75	50	91	190	6
Boscoe	192	147	209	225	125	92	10
Bort	187	139	210	202	153	102	7
Quo	132	172	154	138	234	169	1
EvilPixie	125	124	179	178	188	200	6
Yakool	67	105	163	205	209	247	4

Table 5.5: The outcome of experiment 2. 1000 matches were played.

two MARS bots win about 68% [1] and about 81% [2] with three bots. It is visible in Table 5.7 and Table 5.8 that MARS did not quite live up to the expectations of 68% and 81%, but still managed to win 61% with two bots and 71% with three bots.

Stability

In the first 500 matches there were exceptions thrown in 32. Just like the previous experiments these were caused by MARS and EvilPixie. MARS threw 16 exceptions due to the bug in the Card Agent and EvilPixie threw 16 exceptions due to its fortification bug.
This was repeated in the second part of the experiment only now with 35 exceptions, 23 by MARS and twelve by EvilPixie.

[1] The odds that one MARS bot will not win a match is $(1 - 0.43)$. With two MARS bots the odds are then $(1 - 0.43)^2$ that neither will win. Which makes it $1 - (1 - 0.43)^2$ that one of the two will win.

[2] $1 - (1 - 0.43)^3$

Number of Bots	Stalemates
2	1
3	8
4	2
5	0
6	0

Table 5.6: The number of bots involved in each stalemate in experiment 2

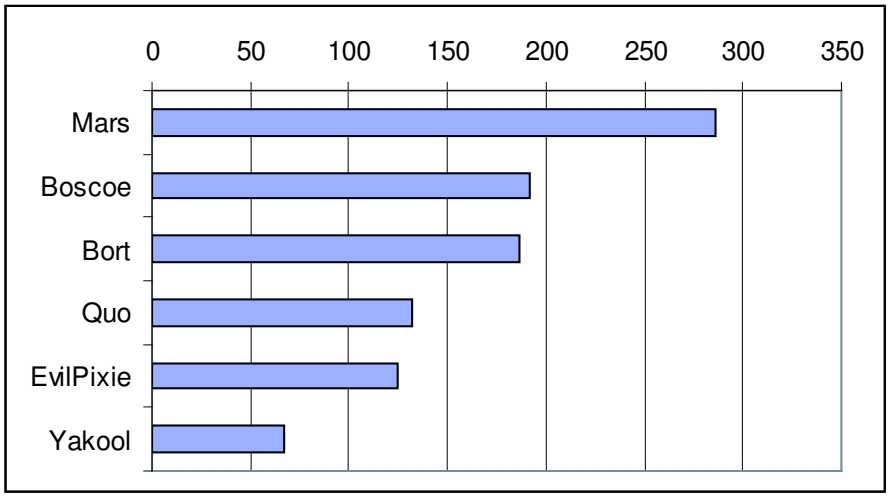

Figure 5.5: The number of wins each bot had in experiment 2. See Table 5.5 for actual numbers.

Runtime

The average runtime, in the first part, with two MARS bots was 122.8 seconds. With three MARS bots the average runtime was 128.1 seconds. The summary of the runtime from all the experiments can be seen in Figure 5.7

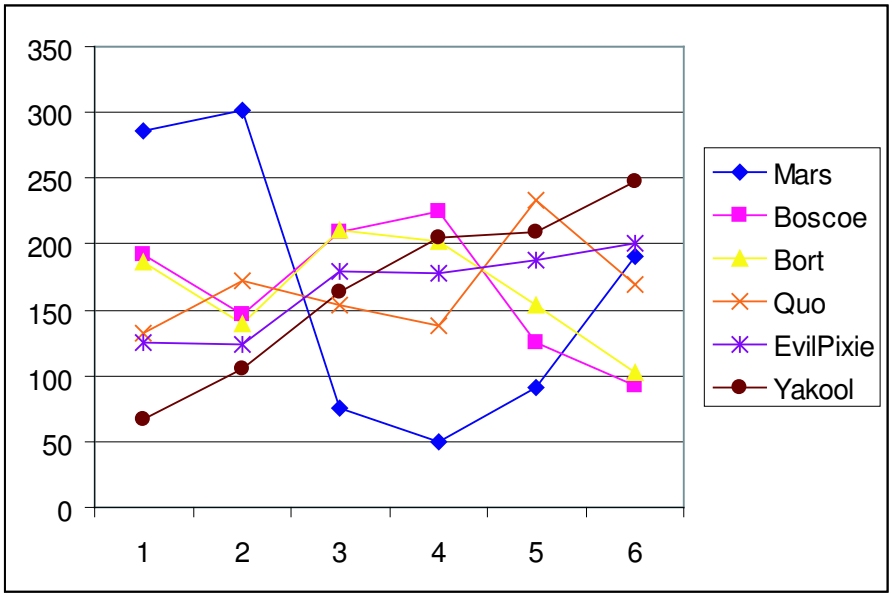

Figure 5.6: The number of first to sixth placements in experiment 2. 1000 matches were played.

Name	Matches	Win%	1st	2nd	3rd	4th	5th	6th	Stalemate
Angry	162	1.2%	2	20	21	26	36	57	0
Communist	163	6.1%	10	69	41	20	13	6	4
Quo	168	10.1%	17	14	44	44	24	25	0
Pixie	162	8.0%	13	21	29	28	45	26	0
Nefarious	167	5.4%	9	18	31	45	43	21	0
Stinky	178	0.6%	1	47	45	38	24	23	0
Yakool	187	8.6%	16	8	27	37	56	43	0
EvilPixie	170	11.8%	20	8	28	39	41	33	1
MARS	500	61.0%	305	143	108	101	122	212	9
Boscoe	153	18.3%	28	5	31	39	33	17	0
Bort	167	21.0%	35	14	38	41	26	12	1
Cluster	159	8.2%	13	29	33	35	33	15	1
Shaft	164	14.0%	23	96	20	5	3	10	7

Table 5.7: Experiment 2,500 matches with two MARS bots in each match.

Name	Matches	Win%	1st	2nd	3rd	4th	5th	6th	Stalemate
Angry	130	0.8%	1	27	28	25	26	23	0
Communist	129	8.5%	11	64	32	10	7	4	1
Quo	120	12.5%	15	14	21	31	25	14	0
Pixie	123	11.4%	14	17	14	25	30	23	0
Nefarious	121	7.4%	9	16	22	34	28	12	0
Stinky	128	0.0%	0	40	39	16	13	19	1
Yakool	117	3.4%	4	4	17	33	38	21	0
EvilPixie	120	7.5%	9	8	14	38	40	11	0
MARS	500	70.6%	353	168	204	198	238	335	4
Boscoe	135	24.4%	33	15	31	28	19	9	0
Bort	123	19.5%	24	15	31	30	12	11	0
Cluster	128	3.1%	4	29	27	29	22	17	0
Shaft	126	15.1%	19	79	19	3	2	1	3

Table 5.8: Experiment 3, 500 matches with three MARS bots in each match.

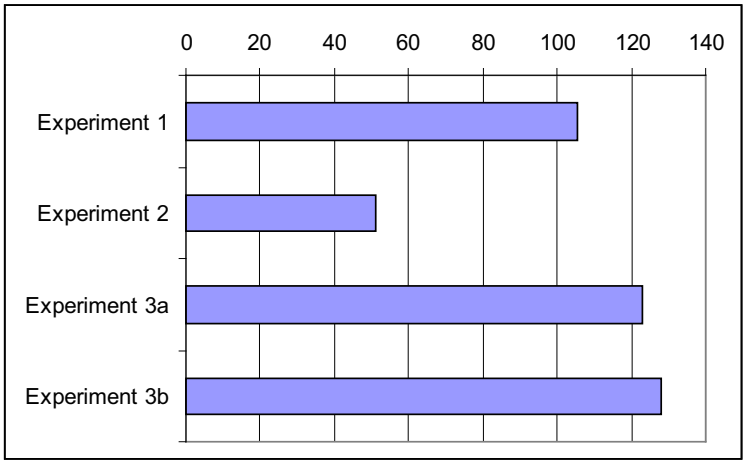

Figure 5.7: Average runtime in seconds per game, with MARS, in the different experiments.

Chapter 6

Discussion

6.1 Ranking

The fact that MARS did place sixth more times than third, fourth, or fifth, I believe is a sign of its aggressive nature. It will early on fight very hard to get that first continent. It either succeeds and places first, second or it fails and places sixth. The alternative to placing sixth being a slow death and placing fifth, fourth or even third.
The high number of second places, especially during experiment 2 where it faced better opponents, might be an indication that it needs an alternative strategy for when there are only two opponents left.

6.2 Runtime

All runtime experiments showed that MARS is much slower than its fellow bots. Experiment 2 runtimes were a bit faster since none of the passive bots were there to drag on and extend the runtime. To the passive bots I include Shaft, Communist and to some extent even Cluster. What is odd is that in experiment 3 the runtime did not double or triple[1]. As can be seen in Figure 5.7 going from one MARS bot to two increases the runtime from 106sec to 123sec, while adding one more bot only increases the runtime to 128sec. This might be because when there are two or more MARS bots they will many times knock each other out first, because of fighting for the same continent.

6.3 Stability

Even though MARS occasionally did throw exceptions it managed to continue play and win some of those matches. This shows that the bug is not critical. The bug should due to its nature, being a ArrayIndexOutOfBounds exception, be easy to find and fix. This should obviously have been found and fixed before the experiments, but was missed. Most likely due to its low rate of occurrence, of less than 3%.

[1] Compared to experiment 1

6.4 Parameter Optimization

Parameter Optimization showed 49%, but when running experiment 1 MARS only won 43%. This shows an obvious flaw in the parameter optimization. I believe that the technique is correct, but that running 200 matches with each setting just is not enough. Running more than 200 matches was unrealistic due to the scope of the thesis.
One way to possibly improve the technique would be to have MARS play only against the other five bots from experiment 2. This to lessen number of variables that affect the outcome. Also by using a smaller grid and instead playing more matches. Instead of having a five by four grid and playing 200 matches, as the first step was, maybe have three by three and play 900 matches. Playing this many matches at each intersection is possible since the average runtime in experiment 2 was about half of that in experiment 1.

6.5 Multi-Agent System

To what extent is the success of MARS due to it being a MAS?
Could all this have been done with a monolithic solution?
It is difficult to tell exactly how much of MARS's success is due to the it being a MAS. The usage of agents did divide the problem into smaller, simpler problems and the much of the strength of this solution lies in thinking on the country level instead of the world level.
There is nothing in this solution that is impossible to perform by a monolithic bot. Therefore it is theoretically possible to create such a bot. It is my opinion that such a system would merely simulate the multi-agent aspects of MARS. This raises the question wether such a system should be classified as a monolithic solution or as a MAS? The answer to such a question lies far beyond the scope of this thesis.

6.6 Evolutionary Viewpoint

What will happen in the future if all bots use the principles of MARS as a basis? The results did not reach the win percentage that was calculated, based on its performance in experiment 1. This is most likely due to higher level of competition. Namely MARS itself. When comparing the results in experiment 1 and experiment 2 it is visible that the win percentage goes down a bit when playing against better opponents. Since MARS is the best bot in the tests, it is only logical that playing against itself will lower the percentage. Because of this I believe that experiment 3 showed that MARS is strong from an evolutionary viewpoint.

6.7 Methodology

One possible flaw of the method used lies in the selection of the opponents. The system used was chosen much due to the quantity of opponents, in the hope that quantity would also mean diversity. What if one of the other systems has a bot that is better? Also, there is no way to know for how long these results

will stand. For example, during the course of the experiments a new version of Nefarious was released. It is uncertain how MARS performs against this improved opponent.

6.8 Strengths and Weaknesses

Fortification worked very well, due to the usage of a maxflow algorithm. There are no numbers to support this explicitly but the fortification might very well be one of the reasons that MARS performed so well.
Another strength in the system is the auction mechanism used, this turned out to be a very stable way to distribute armies based on who needs them the most. Using a mediator agent simplified the auction mechanism as it would have been much more complicated if it was completely distributed. The mediator agent also made debugging easier since every message passed through this single point. At the same time this might have made the system somewhat slower, since a message sent between two country agents is sent twice. First from the source to the mediator and then from the mediator to the destination.
A weakness with the system is the way that goal data is gathered. Spreading every possible goal path to every possible country[2] seems like a suboptimal solution.

[2]Every possible own country, on the border of a cluster of enemy countries

Chapter 7

Conclusion

Creating a MAS solution in a world of monolithic, strategy focused bots was a new direction. This new way of thinking showed great success. It outperformed the competition in every category except runtime. The higher rate of wins came at the cost of CPU-time, despite this it showed that when developing a Risk bot a MAS solution is indeed prosperous.

Chapter 8

Future Work

8.1 Cooperate

Cooperation between MARS bots might be a way to further improve the bot. How would it perform in experiment 3 if it did cooperate with other MARS bots?

8.2 Aware of Different Opponents

MARS currently treats all opponents the same, but what would happen if it was aware of how the other players play.
For example, if we know that red player is tied up in a conflict on the other side of the board, we can gain from this by attacking it since we know he will most likely avoid a two front war.
Or, with knowledge of how the opponents act. If we know that a specific player only attacks the weakest neighbor, we can avoid being attacked by simply making sure we have one army more than the weakest neighbor.

8.3 Improved Parameter Optimization

As was shown in the experiments the current parameter optimization is a bit lacking, which means that there is room for improvement. If the current technique is to be used, one needs to run more than the 200 matches with each setting that were run in the parameter optimization.
An alternative could be dynamic parameters, that adapt to what the world or immediate neighborhood looks like.

8.4 More Opponents

Even though MARS was tested against twelve other bots, most of these came from the same place. Several were just extensions of others. Finding out a way to test it against more opponents such as the ones found in commercial Risk games would give a better view on how well it really performs.
A large scale test against humans would also be interesting.

8.5 Compare to Other MAS Bots

What does this have in common with other MAS bots for different board games? Is there a common strategy used? Is it possible to develop a MAS that can play different board games using the same principles?

Bibliography

[1] Danny Beardsley, Matt Killam, Brendan McElligott, Rory Strawther, and Luke Winkenbach. Javarisk v2.0 - an enhanced computer opponent. `http://counties.csc.calpoly.edu/~team_8fk/F04/` URL last visited on 2005-02-02.

[2] Gael de Chalendar. Ksirk v1.1-3. `http://home.gna.org/ksirk/` URL last visited on 2005-02-02.

[3] J. Edmonds and R. M. Karp. Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

[4] Henric Fransson. Agentchess - an agent chess approach. Master's thesis, Blekinge Institute of Technology, 2003. MSE-2003:02.

[5] Hasbro. *Risk - The Game of Global Domination*, 1999.

[6] S. Hayden, C. Carrick, and Q. Yang. Architectural design patterns for multi-agent coordination, 1999. Proceedings of the International Conference on Agent Systems '99. (Agents'99). Seattle, WA, May 1999. Available WWW .

[7] Fredrik Håård. Multi-agent diplomacy. Master's thesis, Blekinge Institute of Technology, 2004. MCS-2004:20.

[8] David Keppler and Edward Choi. An intelligent agent for risk. `http://www.cs.cornell.edu/boom/2001sp/choi/473repo.html` URL last visited on 2005-02-02., 2000.

[9] Paul Kinlan. Jrisk v1.0.7.5. `http://sourceforge.net/projects/risk/` URL last visited on 2005-02-02.

[10] Sebastian Kirsch, Christian Domsch, Dirk Engberg, and Shawn Krug. Javarisk v2.0.92. `http://sourceforge.net/projects/javarisk/` URL last visited on 2005-02-02.

[11] Kennedy Lemke. Risk board game dice odds. March 1999. `http://www.plainsboro.com/~lemke/risk/` URL last visited on 2005-02-02.

[12] Owen Lyne, Leon Atkinson, Peter George, and Don Woods. Risk - frequently asked questions v5.51. `http://www.maths.nott.ac.uk/personal/odl/riskfaq.html` URL last visited on 2005-02-02.

[13] S. Shu and D. Norrie. Patterns for adaptive multi-agent systems in intelligent manufacturing, 1999. In Proceedings of the 2nd International Workshop on Intelligent Manufacturing Systems, pages 67–74, Leuven, Belgium, september 1999.

[14] SillySoft. Lux v4.3. `http://sillysoft.net/` URL last visited on 2005-02-02.

[15] Andrew S. Tanenbaum. *Computer Networks; 3rd edition.* Prentice Hall, 1996.

[16] Caspar Treijtel. Multi-agent stratego. Master's thesis, Delft University of Technology, October 2000.

[17] M. Wooldridge. *An Introduction to MultiAgent Systems.* John Wiley and Sons Ltd, 2001. page 134.