UNIVERSITÄT BASEL

A Learning AI for the game Risk using the TD(λ)-Algorithm

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence Group
http://ai.cs.unibas.ch

Examiner: Prof. Malte Helmert
Supervisor: Gabriele Röger

Manuela Lütolf
manuela.luetolf@stud.unibas.ch

24.6.2013

UNI
BASEL

Table of Contents

Abstract

Risk is a popular board game where players conquer each other's countries. In this project, I created an AI that plays Risk and is capable of learning. For each decision it makes, it performs a simple search one step ahead, looking at the outcomes of all possible moves it could make, and picks the most beneficial. It judges the desirability of outcomes by a series of parameters, which are modified after each game using the TD(λ)-Algorithm, allowing the AI to learn.

1 Introduction: What is Risk?

My goal in this project was to create an artificial intelligence (AI) which plays the game Risk, a game depicting a map of the world, with the goal of conquering the whole map. Originally, Risk was a board game, but there have been digital versions of it, with AI computer opponents. I had a look at some of these, and as is to be expected, they are usually plan-based, with their playing strategies hardcoded into them. The AI I created in this project, by constrast, uses a learning algorithm to adjust its behaviour.

1.1 Rules of Risk

Each country is owned by one of the players, with a certain number of troops on it.
At the start of the game, during the “setting up stage”, the players take turns placing one army on an empty country, making themselves the owner, until there are no countries left. Then the players take turns distributing their spare armies onto countries they own. (How many armies they may place depends on the number of players.) After that, a player performs the following actions on his turn, always in the same order:

1. Trading stage: The player may have earned cards from previous attacks. If he has a matching set of three, he may exchange these for extra armies. The first player to trade in a set receives 4 armies, the next one 6, and the number increases with each traded set. If a player has 5 cards, he must trade.

2. Placing stage: The player decides which of his countries he wants to place extra troops on. How many troops he may place depends on how many countries and continents he owns: For every three countries he owns, he gets an extra army, with the minimum being 3. Owning all of South America or Australia gives 2 extra armies, owning Africa gives 3, owning all or North America or Europe is worth 5, and finally, owning Asia gives 7 extra armies. This is of course in addition to armies he might have gotten from a card stage.

3. Attack stage: Once all his armies are positioned, he can use his countries to attack and conquer neighbouring countries, the success of which is decided by dice: The attacker may use one die if he has two armies, two if he has three, or three dice if he has more.

The defender may use one die if he has one army, or two otherwise. The attacker's highest die roll is compared to the defender's highest die roll, and (if both used at least two dice) his second highest with the defender's second highest. If the defender's result is equal or higher, the attacker loses one army, otherwise the defender loses one. This means that, if both players have at least two dice, the result is either that the attacker loses two armies, that the defender loses two armies, or that they each lose one. Once the defender has no more troops, the attacker owns the country and moves his troops over. If the attacker instead is reduced to one army, he can no longer attack. A player may make as many attacks as he wants and is capable of with his armies.

4. Fortifying stage: Once the player has chosen to end his attack phase, he may move one set of armies from one country he owns onto an adjacent one.

1.2 Variations

- Instead of letting the player chose their countries at the beginning of the game, the countries may also be distributed at random. In a physical game, this is done by drawing cards. I implemented a version without autoplacement, as this is more interesting.
- Instead of aiming to conquer the whole world (known as domination), players may also receive secret missions, such as "Conquer Africa and Europe", which allow them to win the game. Another alternative has the players chose one of their countries as their capital, which must then be conquered. I worked with the domination version, as I am more familiar with it.
- Rather than have the card sets increase in army value with each trade, the value can also be fixed, or depend on what kinds of cards are traded. I worked with the increasing set in mind.
- Risk can be played with alternate maps, with different countries and continents. I focused on the classical map of the real world.

In chapter 2, I will introduce some work other people have done towards creating Risk AIs. Chapter 3 explains the mathematical basis and the implementation of the learning algorithm I used to let the AI get smarter, and in Chapter 4 I will analyse the results.

2
Related Work

2.1 Yura Domination

Domination [4] is an huge open-source implementation of the Risk game by Yura Mamyrin, written in Java. It provides countless features, such as alternate maps and the possibility to create your own map, translations into many languages and instructions on how to add your own translation, the possibility of playing against others over the internet, and AI players labelled “Easy”, “Average” and “Hard”. I used this program as a framework for my own AI.

2.1.1 Basic workings of the existing AIs

- The most simple AI is called AISubmissive. It is completely passive, and never attacks. It serves simply as a framework for other AIs to extend from, and the author advises programmers who want to create their own AI to make it extend AISubmissive so that it follows all the rules correctly. If people play over the internet and one player is disconnected, his part will be taken over by an AISubmissive so that the others can finish.

- In order for an AI to work, it must contain certain methods corresponding to the stages mentioned in the last chapter. For example, getPlaceArmies() should return the player’s choice of army placement.

- When I first downloaded the code at the beginning of my project, the various AIs were located in separate classes. There was later an update (Revision 1167) which completely changed the structure. The methods of all three AIs are now in the same class, with various if-clauses distinguishing their behaviour from each other. The skills of AIEasy and AIAverage are reduced in some places by adding a probability that they will take random actions, unlike AIHard, which always performs the smartest move the program knows, and performs more checks to see if certain moves are a good idea or if they will weaken him.

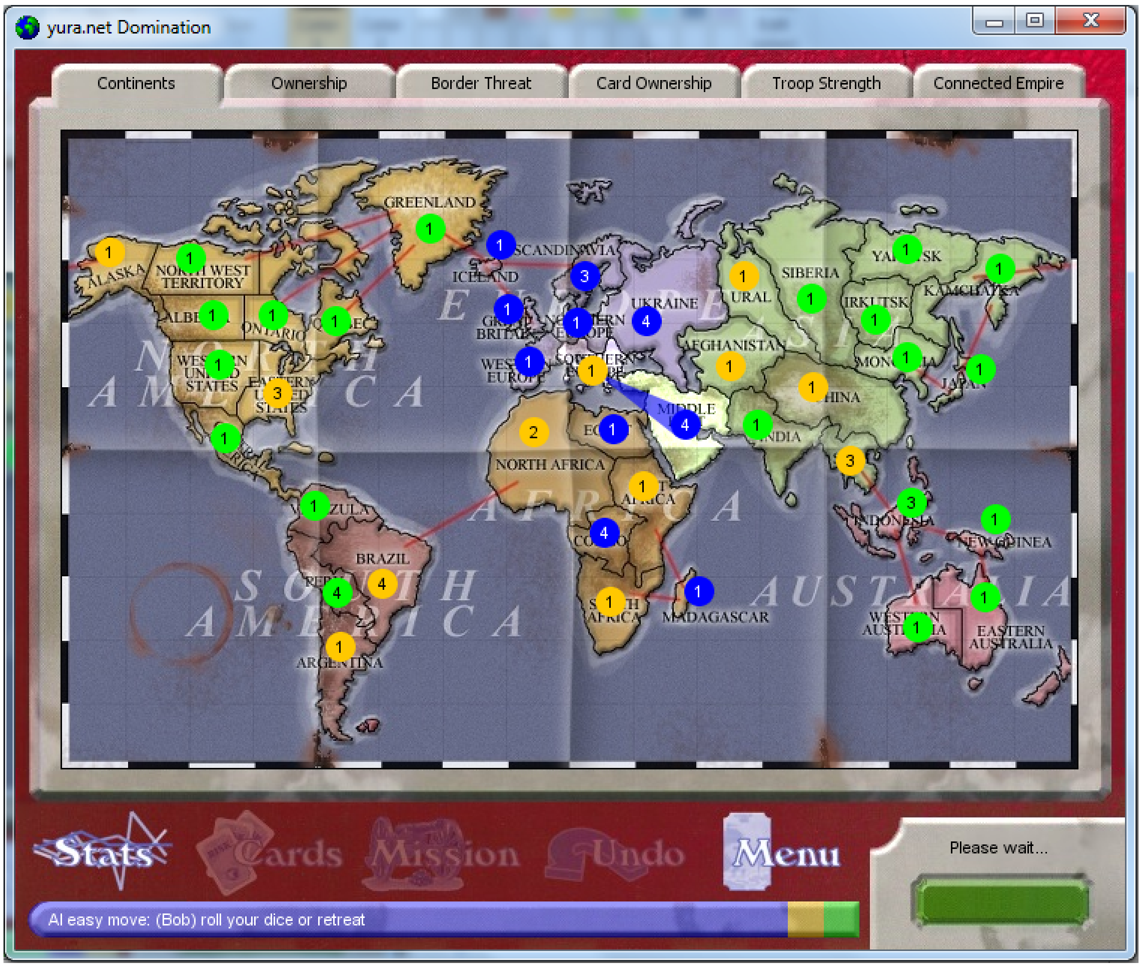

Figure 2.1: Screenshot of the Domination game. Here with the blue player attacking Southern Europe from the Middle East

2.1.2 AI behaviour

General

- The AIs understand the concept of a "common threat", that is, if one player becomes too powerful, they will all focus on stopping him first.
- Each turn, an AI will decide if it should be more aggressive. It also calculates its own "defenseValue" and compares it with the other player's "attackValue", which influences if they will keep pressing an attack, or if they will be quicker to retreat.
- Before making an attack, the method isGoodIdea() will be queried, which analyses the cost and benefit, among other things.

Setting up stage All AIs base their decision on which countries and continents to own on a score they give each country. A continent is rated by the number of starting armies it gives, the number of territories it contains and how many border countries it has. Furthermore, if the AI notices that another player has got almost all the countries of one continent, it will place a troop there to block him. AIEasy simply autoplaces (i.e. it lets the program place the troops at random).

Troop placement and attack stage The AI will check which troops it can attack, and sort them based on which target has the shortest route, i.e. which target can be reached by passing the fewest countries. It will also check if there is a continent it can conquer, or a continent that belongs to another player, which it can "break", meaning steal one of its countries, to prevent the player from getting the starting bonus the next round.

Tactical move When moving armies at the end of the turn, the AI will move its troops toward its borders, where they can serve as defense. If there is a "common threat", the AI will move its troops off continents it doesn't want.

Rolling the dice While rolling the dice during an attack, the AI will check every three rolls if the attack is still a good idea, or if it has lost too many troops.

Trading cards Card trades are delayed where sensible, provided the player isn't too weak and really needs the reinforcement troops.

2.2 Lux Delux

Lux Delux [1] is another Risk game written in Java, although this one is not free. I therefore didn't play the game, but only looked at the code of their AIs. One useful gaming tip which I read on their website was: Instead of focusing too much on conquering continents, one should focus on spreading one's largest cluster. A cluster is a connected group of countries owned by the same player. Strengthening the borders of this cluster, and expanding it bit by bit, is a good way to play. This idea was the biggest influence Lux Delux had on my work.
Lux has a colorful variety of AIs, as illustrated in Figure 2.2. A lot of them aren't directly aimed at playing well, but instead having an interesting personality.
Chimera is simply a wrapper or disguise for another AI (the "backer"), chosen at random, so that the players don't know which AI they are dealing with until the game is over, when it is revealed.
SmartAgentBase is the basis for most of the AIs. One thing notable about AIs extending this one, is that they have a memory, and remember which continents they intended to conquer etc. When choosing which continent to conquer, a factor is the ratio of my troops to the troops of my enemies on it. When deciding which country in a continent to own, they pick one that is closest to countries they already own. If the AI doesn't own any countries in the continent yet, if finds the "cheapestRouteFromOwnerToContinent". SmartAgentBase and its descendants know a variety of attack strategies, such as "attackEasyExpand", which checks for weak enemies around the cluster borders that can be defeated, "attackFillOut", which conquers "islands" of enemies completely surrounded by me, and "attackConsolidate", in which a country is conquerred that borders two of my countries, thus reducing the

number of countries at the border. “Going Hogwild” means attacking as much as possible, usually because the AI has found that its armies far outnumber the enemies. This AI will also recognize enemies as common threats that must be defeated first before attacking the others. It also has a method for attacking along a specific path.
Communist distributes its troops evenly among all its countries. It will first kill all the players that are not of the type “Communist”, and then the “Communists” will start attacking each other.
Pixie: As described in the code’s comments: “Pixie examines each continent and focuses on promising ones. She picks countries in continents that have the fewest border points.” [1]
EvilPixie: Unlike Pixie, who waits with cashing in her cards for as long as she can, EvilPixie makes the trade as soon as she can. Before she starts conquering continents each turn, she checks if there is a dominant player that needs to be stopped. She will try to kill any players she outnumbers 5:1.
Cluster This AI and its descendants focus on expanding their biggest cluster of countries, as mentioned earlier.
HumanFriendly: Never attacks countries owned by human players.
Noisy: Comments on everything it does, to test the chat.
Shaft and **Quo**, descendants of Cluster, differ from their parent very little in terms of strategy. They just have a different order regarding when and how often they try to expand borders, fill out islands, consolidate borders, etc.
Yakool: “If an enemy looks too strong all [his] efforts are put into killing him.” [1]. Yakool attacks if he considers himself too weak and needs to strengthen, or if he considers himself very strong and thinks he might be able to conquer the world.
Boscoe is “Yakool with a slowed down attack strategy” [1], and **Bort** is slower still. (It tries to do only one attack per turn).
Vulture uses another AI as a “backer”, much like Chimera does, except it only uses backers of the type “Cluster”, and it doesn’t let its backer make all the decisions. It will check each turn if there is a player it can kill within reach. It will also check if it outnumbers all the other players and should hence “run hogwild”.
Killbot extends Vulture, but uses only **BetterPixies** as a backer (whose strategy is similar to that of Pixie).

Besides these programs, which implement the complete Risk play, there has also been a study just tackling the beginning of the game.

2.3 “An Automated Technique for Drafting Territories in the Board Game Risk”

This is a paper by Richard Gibson, Neesha Desai, Richard Zhao [5]. It explores the setting up stage, in which the players chose which countries to own initially, and how likely a choice will make the player’s chances of winning. It was tested on Lux Delux.
The authors of this paper used the Monte Carlo tree search algorithm UCT [3] to simulate

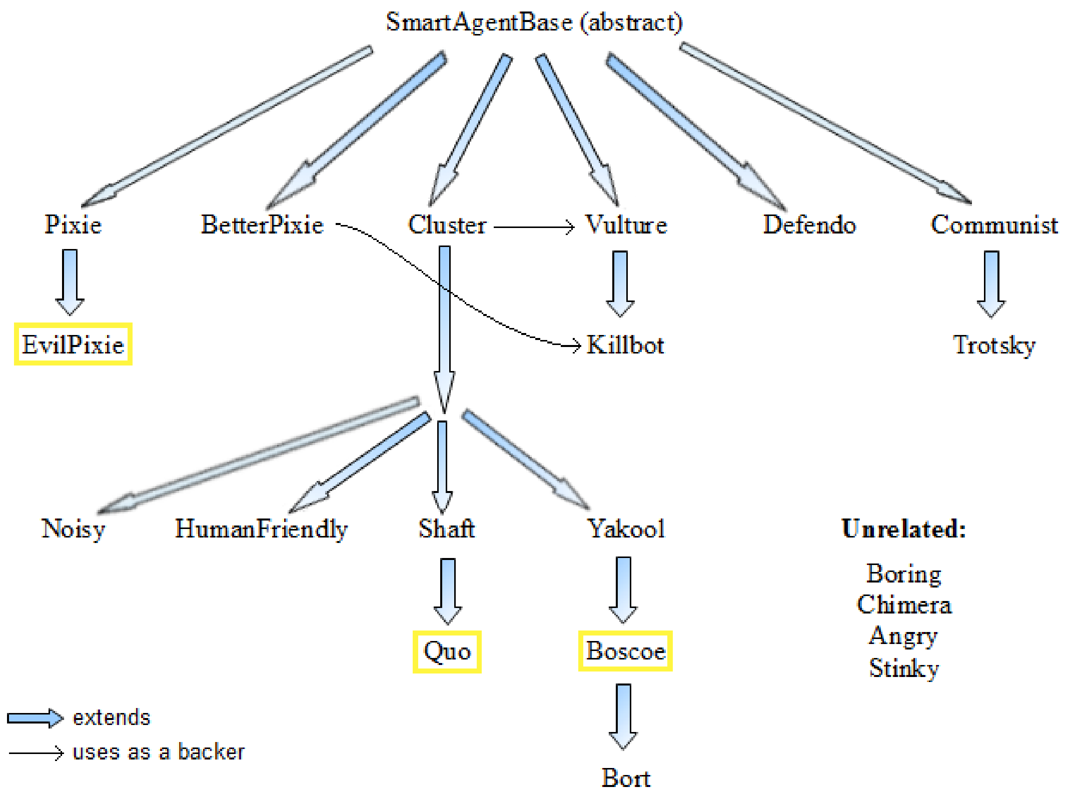

Figure 2.2: Diagram of the various types of AI in Lux Delux and how they extend each other. Outlined in yellow are the ones the paper by Gibson, Desai and Zhao labels as the strongest.

Risk games given a certain starting distribution of countries, and thereby decided which countries were the most favourable to own at the start. What had the most influence on my work was the features they used:

- For each continent, the number of countries owned
- When it's the player's turn
- Number of distinct enemy territories bordering on mine (enemy neighbors)
- Number of pairs of owned adjacent territories (friendly neighbors)

These features were evaluated by randomly assigning the 42 countries to players, playing 100 games, and thereby assigning values to each of these features.
Their results showed, among other things, that placing one's first country on Europe was most beneficial. But the value for countries in a continent doesn't always increase. For example, having 6 countries in Europe was calculated as being more beneficial than having 7. It seems obvious to me that this can only be true for the initial placement stage, and not for the game thereafter, or the AI would refuse to attack another European country after it had 6.

3
AIParam: The parameterized AI

3.1 Basic workings

Rather than performing complex planning ahead like the other Risk AIs, my AI simply "blindly" looks at all the possible moves it can make, evaluates the worth of each outcome, then picks the move with the most benefit. In order to evaluate a state reached by an action, it has a list of features which describe the game, such as "Number of countries I own in Europe" or "starting armies I will receive with the countries I currently own". The AI evaluates each of these features for itself, and for each of its enemies (though I put the main focus on two players, due to time constraints) . Each of these features is paired with a weight, hereafter sometimes referred to as parameter, which indicates how much the feature is counted. These parameters are modified by the TD(λ) algorithm (see below) to allow the AI to learn where to place its priorities.

Initially, I considered letting the AI do a tree search for all the possible outcomes of its actions which reached until the end of its turn, perhaps even until the end of the opponent's next turn. However, a quick calculation showed that this wasn't very feasible, as there were just too many possibilities. Deciding which countries to place armies on, for instance, assuming the player owns 20 countries, and assuming this is a stage later in the game where he has 30 armies to place, gives one a possibility of 20^{30} placements.
Therefore, I made my AI look ahead only a single step.

3.2 The TD(λ) algorithm

The TD(λ) algorithm was originally described by Richard Sutton in the paper "Learning to Predict by the Method of Temporal Differences" [6]. I used the paper "TDLeaf(λ): Combining Temporal Difference Learning with Game-Tree" by Jonathan Baxter, Andrew Tridgell and Lex Weaver [2] to guide me on it, as it already applies the algorithm to games.

3.2.1 Theory

The idea is that an agent is given a possible number of actions it can perform each turn, and it will chose the action leading to the most favourable new state with regard to an evaluation function $J(x, w)$. Here $x = x_1, \ldots x_N$ is the sequence of states encountered in the game, which in the case of Risk would include information such as who owned which country during a certain turn, etc. The variable w represent the vector of weights that need to be learned. At the end of the game, a reward is given, usually +1 for success, and -1 or 0 for failure.

The "TD" stands for "temporal difference", which refers to the difference in the reward predicted between two states:

$d_t = J(x_{t+1}, w) - J(x_t, w) \quad \text{for } t \in \{1, \ldots, N-1\}$

For ease of notation, it is assumed that $J(x_N) = r(x_N)$, i.e. the last state has the value of the reward $r(x_N)$ for winning or losing. Therefore we can add the reward at the end as the last state. Then the parameter vector w is updated according to the formula:

$w := w + \alpha \sum_{t=1}^{N-1} \nabla J(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right],$

where α controls the learning rate, $\lambda \in [0, 1]$ controls the extent to which temporal differences propagate backwards in time, and $\nabla J(x_t, w)$ is the vector of partial derivatives of J with respect to each parameter.

For the evaluation function $J(x, w)$, I needed a simple way of evaluating the current game situation which has a derivative that is easy to calculate. I therefore used a simple multiplication: The AI has a vector $f(x) = (f_1(x), \ldots, f_M(x))$ of the aforementioned feature functions, which each describe one aspect of the game by returning a simple number, and a vector of weights $w = (w_1, \ldots w_M)$. To calculate the current score $J(x_t, w)$, it multiplies the values of these two vectors and adds them up, and, since the result should lie between 0 and 1, to match the possible rewards given at the end, enters the result into the sigmoid function.

The sigmoid function is defined as:

$$\sigma(t) = \tfrac{1}{1+\exp(-t)}$$

Inserting the sum of features gives:

$$J(x, w) = \frac{1}{1 + \exp(-\underbrace{\sum_{j=1}^{M} w_j f_j(x)}_{t})} \tag{3.1}$$

The advantage of the sigmoid function is that it is easy to derive, which is of course needed for the $\nabla J(x_t, w)$ part of the update function.

The derivative of the regular sigmoid function is:

$$\frac{d\sigma(t)}{dt} = \sigma(t)(1 - \sigma(t)) \tag{3.2}$$

To derive the formula by one specific weight w_i, we extract the weight in question:

$$J(x,w) = \frac{1}{1+\exp(-\sum_{\substack{j=1\\ j\neq i}}^{M} w_j f_j(x)) \cdot \exp(-w_i f_i(x))} \tag{3.3}$$

Using the chain rule to combine (3.3) and (3.2), we can find the derivative by a specific parameter:

$$\begin{aligned}\frac{d}{dw_i}J(x,w) &= \frac{d}{dw_i}\frac{1}{1+\exp\left(-\sum_{j=1}^{M} f_j(x)\cdot w_j\right)}\\ &= \frac{1}{1+\exp\left(-\sum_{j=1}^{M} f_j(x)w_j\right)}\left(1-\frac{1}{1+\exp\left(-\sum_{j=1}^{M} f_j(x)\cdot w_j\right)}\right)\cdot f_i(x)\end{aligned}$$

As a result, each individual parameter w_i is updated with the formula

$$w_i := w_i + \alpha\sum_{t=1}^{N-1} J(x,w)(1-J(x,w))\cdot f_i(x)\cdot\left[\sum_{j=t}^{N-1}\lambda^{j-t}d_t\right]$$

3.2.2 Implementation

I called the AI I implemented “AIParam”, referring to the parameters it tunes. One thing worth noting is that, in my Risk AI, a decision is made more than once per game turn: first in the trading stage (deciding whether to trade cards or not), then in the placing stage (deciding where to put extra armies), the attack stage (deciding which enemy countries to attack), the moving stage (after a battle has been won and a new country has been conquerred, how many troops should be moved to it), and finally, the fortifying stage, where the player may move one group of troops between countries he owns. As far as the TD(λ)-algorithm is concerned, each of these stages represent a “turn”, where a decision is evaluated, and which is later used in the algorithm to improve the weights.

When it makes a decision, AIParam creates objects called GameEvaluations, which represent the hypothetical state the game would be in if a certain command was given. These GameEvaluations are able to evaluate themselves, with the method totalEvaluationOfCurrentGame representing the function $J(x,w)$, and the weights stored in the class EvaluationParam.
The class Autoplay.java starts a series of automatic Risk games without a GUI, and lets the AI learn, by updating the parameters after every game.

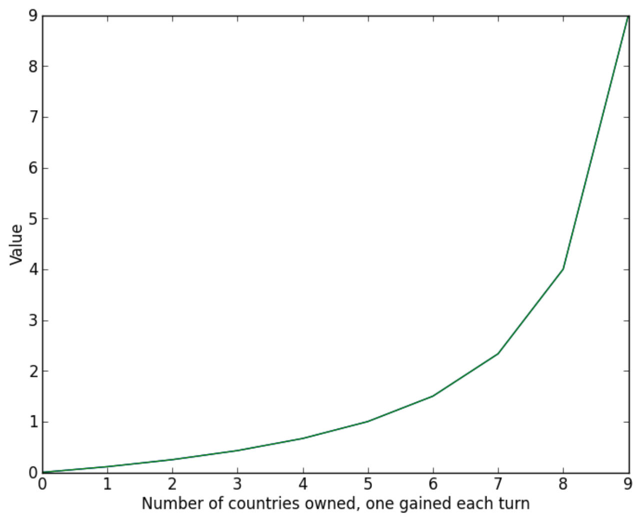

Figure 3.1: Graph evaluating how much each additional country in North America is worth, using the formula $\frac{countriesIOwn}{countriesIDontOwn+1}$. The graph's gradient increases, as required.

3.3 Problems encountered and lessons learned

3.3.1 Features over foresight

There were several times during the project where I was sceptical about the idea that an AI which does not plan ahead at all could possibly be a good player. But I learned over time that a lot of knowledge normally gained by looking ahead could also be achieved with the right Feature, measuring the relevant game situation:

Conquering a continent : How does the AI know that conquering all of a continent is beneficial, if it can't see that in the future, this will cause it to gain more starting armies? That is of course not so difficult: simply introduce a Feature measuring the starting armies one would gain. But more complicated: On the way towards conquering a continent, an AI should become more motivated to gain a continent the more countries it already owns on that continent. If it owns all but two countries, it should realize that owning all but one is much better, and owning all of them is better still. I took care of that by making sure that the Feature indicating how much of the continent is owned (NumberOfCountriesInContinent) doesn't simply return the number of countries owned, but instead uses a super-linear function to evaluate how much of the continent is owned, making the reward increase faster the closer the conquest is to complete. I changed it to $\frac{countriesIOwn}{countriesIDontOwn+1}$, with the +1 added to avoid division by zero. This essentially means that if the player owns a whole continent, he is awarded the score of owning all its countries, otherwise he receives less. Owning 5 out of the 9 countries in North America, for example, gives one only a value of approximately 0.1.

Preventing an opponent from conquering a continent : An intelligent Risk AI also needs to recognize if an opponent has taken almost all the countries in a continent, and place one of its troops on the last country to prevent him from taking it completely. As it was

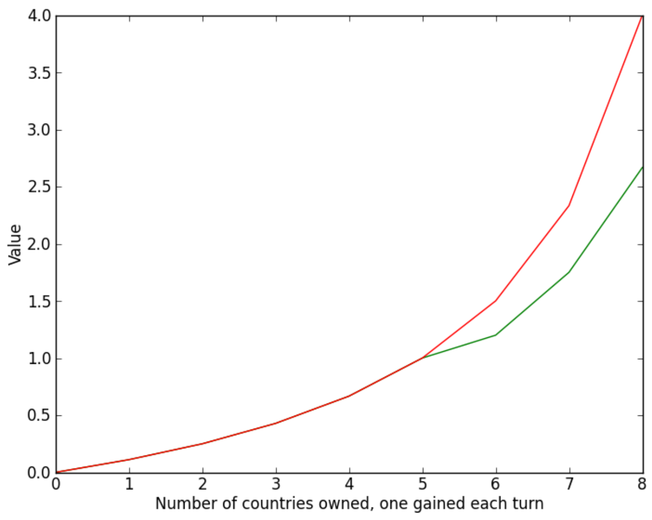

Figure 3.2: Shown in red is the score achieved by the regular conquest of America as before. Shown in green is the reduction of this score if, at turn 5, the opponent steals one of the countries.

now, the AI could recognize that, unfortunately, the opponent currently had a lot of points for owning almost all the countries. And next turn it would recognize that now that the opponent had also taken the last country, he was receiving much more points. But how to make the connection between the two without looking ahead? The trick was to make the AI realize that a game situation where the opponent owns all countries of a continent but one, and I own that last country, is more favourable than a situation where the opponent owns all but one country and the last country has no owner. I therefore modified the formula for the number of countries in continents further, $\frac{countriesIOwn}{countriesIDontOwn+1+countriesOwnedByEnemies}$ essentially counting countries owned by enemies twice as much as countries simply owned by no one.

3.3.2 Foresight only for attacks

There was only one area where I didn't get around using a little searching-ahead, and this was for the attack phase. The first version of the AI never attacked any countries that had more than two defenders, and for people familiar with the die rolling rules, it's obvious why: The AI would calculate one step ahead and conclude that, if it attacked, the best possible result would be that the opponent owned two less troops on his country. The AI had no idea that attacking the country might result in him owning it in the next turn, and all the benefits that entails. I therefore allowed the AI to plan its attacks ahead far enough to reach the end result, that is, either the country is conquerred, with the attacker having spent various possible numbers of troops, or the attacker is defeated and can't attack anymore, with the defender having spent various numbers of troops. It is still not a complete tree search, as a true tree search would include various redundant combinations such as "First I attack Alaska once. Then I attack Quebec. Then I attack Alaska again", whereas this first

calculates to the end of one attack before it considers the next.
It is conceivable that even this problem could somehow be solved with a sophisticated Feature, rather than looking ahead, but I didn't explore this further due to time constraints.

3.3.3 Strengthening the strongest cluster

When I noticed the AI was wasting some of its troops by placing them on "islands", that is, countries it owned which were completely surrounded by enemies and not in a strategically good position, I decided it was time to make use of the tip I got from Lux Delux [1], and to push the AI towards focusing on clusters. I therefore added the Features TroopsInLargest-Cluster and CountriesInLargestCluster to reward it for keeping its troops together.
Once I added these, the AI rarely blocked its opponent from gaining a continent during the setting up stage. This is because it now preferred placing its troops near friendly neighbours to increase the size of its cluster. Nevertheless, the option to prevent an opponent from gaining a continent is still there, and with the right training, the AI can learn that preventing an opponent from gaining a continent might be worth more than increasing its cluster.

3.3.4 Securing borders

Since the AI still had no particular "opinion" on which countries would be the best place to put its troops, it ended up placing its reinforcements in the middle of its territories, where they were far away from the enemy and of no use in battle. I therefore introduced the Feature BorderTroops, which exist for each country separately. It calculates - assuming the player actually owns that country - how many troops the player has there compared to how many troops are in the surrounding enemy neighbours. This caused the AI to strengthen its borders. It also had the effect that the AI gave the appearance of responding to its opponent: the opponent would place some troops on one border, and the AI would respond by placing troops on that border too, to balance the ratio again.

3.3.5 Evenly distributing the troops

The AI then tended to place all of its troops on one country it had deemed the most beneficial, and few or none on others. I made it spread out its armies more evenly by adding a square root into the formula of the BorderingTroops Feature. The important trait of the square root function is that it increases gradually, with the gradient getting smaller. This means that adding a third army into a troop with two armies is more valuable than adding a seventh army to a country that already has six troops. At an early stage, before I started using the sigmoid function and multiplication to make deriving the function easier, I considered making even the degree of the root one of the parameters that could be modified by the learning algorithm, so that the degree to which the troops are spread out or concentrated in one place could be varied. But this is not feasible with the formula I ended up using (see above), and the extra level of complexity that would have been caused by bringing this feature back wouldn't have been worth the slight benefit, so the degree of the root remains 2.

3.3.6 Normalize the Features

In the first few learning sessions, I was puzzled to discover that the AI would make an attempt to attack its opponent only in the first game, and in all subsequent games, it refused to attack at all, seemingly having unlearned the desire to attack. I finally found the cause: The values of the Features all varied from each other strongly. For instance, the AIParam player may at one point have 12 pairs of friendly neighbours, have its troops at the border of Venezuela outnumber their enemies by 1.425, and own 3 countries in Africa. Since the derivative of the score function for a specific game state depends directly on the value of the Feature, the gradient with respect to those pairs of friendly neighbours would be much higher than the gradient with respect to the BorderTroops in Venezuela. This resulted in the parameters being corrected more in regards to the former. It essentialy translates as:

- At this point in the game, I had many countries, and many friendly neighbours
- Then my score suddenly dropped very steeply
- Therefore, owning a lot of countries and a lot of friendly neighbours is a bad thing.

It then lowered these weights to negative values, essentially "blaming" them for all its losses, and as a result didn't try conquering more countries in the next game, because it thought conquest would reduce its score.
I solved this problem by making great efforts to ensure that all the Features produced similar ranges, mostly by dividing them by the maximum result the Feature could give. The number of countries a player owns, for example, is divided by the total number of countries on the map. The number of starting armies is divided by the highest number of armies one could possibly get by owning all the continents, and gaining the extra bonus of owning so many countries. (It does not include card trades. These are counted separately). Another important step to solving this was to ensure there really were enough Features to allow the AI to predict when its opponent was getting dangerous, making the drops in score less sudden. For example, adding the aforementioned evaluation of how many countries a person owns in a continent helped to lower the "surprise" when the opponent finished conquering a continent and suddenly received a big bonus.

3.4 List of Features

Total number of countries owned

This is the most obvious Feature, as the goal of the game is to conquer all the countries on the map.
During test sessions, I sometimes noticed that, rather frustratingly, AIParam had almost won, but was dragging out the game endlessly by not properly attacking the enemy. I therefore made the evaluation superlinear by adding a square, making the value of conquering extra countries strongly increase if one already owns most of

them. I normalized the number by dividing it by 42^2, 42 being the maximum number of countries one can own. So the whole formula is $\frac{n^2}{42^2}$, n being the number of countries owned.

Number of starting armies

Another Feature that is obvious to experienced Risk players. These are the extra armies a player is allowed to place on countries he owns at the beginning of his turn, based on how many countries and continents he already owns.
I normalized this value by dividing it by 38, the highest possible number of starting armies one can have on a regular Risk-map (excluding card trades).

Number of unique enemy neighbours

This is one of the values inspired by the paper "An Automated Technique for Drafting Territories in the Board Game Risk" [5]. An enemy neighbour is, of course, a country owned by the opponent which borders a country owned by me. The word "unique" refers to the fact that, if the enemy country borders two of my countries, it is only counted once.
I normalized this value by dividing it by the highest possible number of enemies one can own on the regular Risk map. (A rough estimate found by hand.)

Pairs of friendly neighbours

Also influenced by the "Drafting Territories" paper [5]. Friendly neighbours are a pair of countries touching each other which are both owned by the same player.
This was a Feature whose value quickly became rather high, so normalization was quite necessary, and I normalized it once again by dividing the number by the highest number of friendly neighbours a player can own, that is, the number he would have if he owned all the countries.

Number of countries in a specific continent (at the start of the game, or later)

This feature exists for each continent. One version is counted at the start of the game, during the "setting up stage", another for the rest of the game, as the paper by Gibson et al [5] not only introduced me to the idea of counting the number of countries in a specific continent, but also shows that in some cases, what is beneficial at the beginning of the game can vary from what is beneficial later. An additional reason for choosing different values at the beginning is that some of the values for countries in continents don't constantly increase. Owning 6 countries in Europe was calculated as being better than owning 7, so if the same features were used during regular gameplay, players wouldn't try conquering all of Europe.
I started out using an approximation of the values calculated in the paper, but they were soon changed significantly, both by the learning algorithm, and my own gradual modifications of the algorithm in general. For one thing, I started off using a map, giving each number of countries its own specific value. But I later turned it into a multiplication, no longer allowing for local minima and other such detailed structures.

These values are already contained fairly well in the boundaries between 0 and 1 by the formula mentioned before, where I divide the number of countries owned by the number of countries not owned.

Number of countries in largest cluster

As mentioned, a cluster is a group of countries owned by the same player which border each other.
The number is normalized by dividing it by the total number of countries, as it would be one large cluster if one owned the whole world.

Number of troops in largest cluster

The same as above, but with the number of troops rather than countries.
Normalized by dividing it by the total number of armies on the board.

Bordering troops in each country

This feature exists for each country. For those countries the player does not own, it returns 0. For those he does own, it returns $\frac{\pm\sqrt{abs(troopsInThisCountry - borderingEnemyTroops)}}{10}$.
The square root is to ensure the AI will spread his troops, instead of concentrating them in one place. The division by 10 is for normalization: I decided it was unlikely for there to be more than 100 troops in a country, the square root of which is 10.

Total troops

The total number of armies I have, divided by the total number of armies on the board overall.

Number of cards

A player suddenly cashing in a set of cards and gaining a dozen or more troops can be a game changer, so it needs to be accounted for. It has very little influence in the AI's decisions during actual game play (other than making it attack at least one country to ensure it gets a card), but it is needed for updating the parameters later. If the AI is aware the opponent is about to cash in its cards for troops, the drop in scores after he does isn't quite as extreme, and it therefore won't "punish" other feature values for this as much.
The maximum number of cards a player can own (and the minimum needed to guarantee 100% that there is a matching set of three) is 5, so to simplify things, the code makes the assumption that the opponent will cash in his cards when he has 5. The value is therefore calculated by multiplying the troops that will be received by the next trade by the number of cards owned divided by 5, as if implying that having one card is like having one fifth of the troops later gained, and this is again divided by the total number of armies currently standing on the board, to normalize it.

4 Experimental Results

4.1 Behaviour

For the beginning parameters, I entered my own estimates of what would be good values. The largest were for the number of countries owned in total and the largest cluster a player has, and the smallest values on the BorderTroops of individual countries.
When I play-tested the AI in its untrained state, I of course always defeated it, but it took me a bit of time.
The AI will usually start by trying to take South America, or, if its opponent has already taken a country there, Australia, since these are the two continents most easily taken. It will then take countries that border the ones it already owns, taking care not to break its cluster, even if this means letting its opponent take a whole continent. This behaviour is further explored below. When reinforcing its countries with armies, AIParam will mostly react to its opponent, placing armies next to countries where enemy armies were placed. As a result, two AIParam players will initially put all their armies on one border, always pushing each other to strengthen the border further in response. Once they have placed all their initial armies, however, this thins out somewhat, and the AI tends to spread its armies somewhat evenly.
Due to its complete inability to look ahead, the AI often places troops in one country but attacks its opponent from a different, poorly armed country.

4.1.1 Blocking opponents from gaining continents

The first version of the feature calculating the number of countries in a continent didn't contain any incentive to block the opponent from conquering the whole continent. The resulting behaviour can be seen in Figure 4.1: AIParam is busy taking Australia and Africa, and once it's finished it places troops elsewhere, and it completely ignores the human player taking North America.
I then modified the formula for the number of countries in a continent as mentioned in Chapter 3, making it rewarding to interfere and place an army on continents enemies are trying to get (Figure 4.1). The AI wasn't motivated enough to prevent me from conquering South America, instead preferring to take Australia for itself in the meantime. It then took

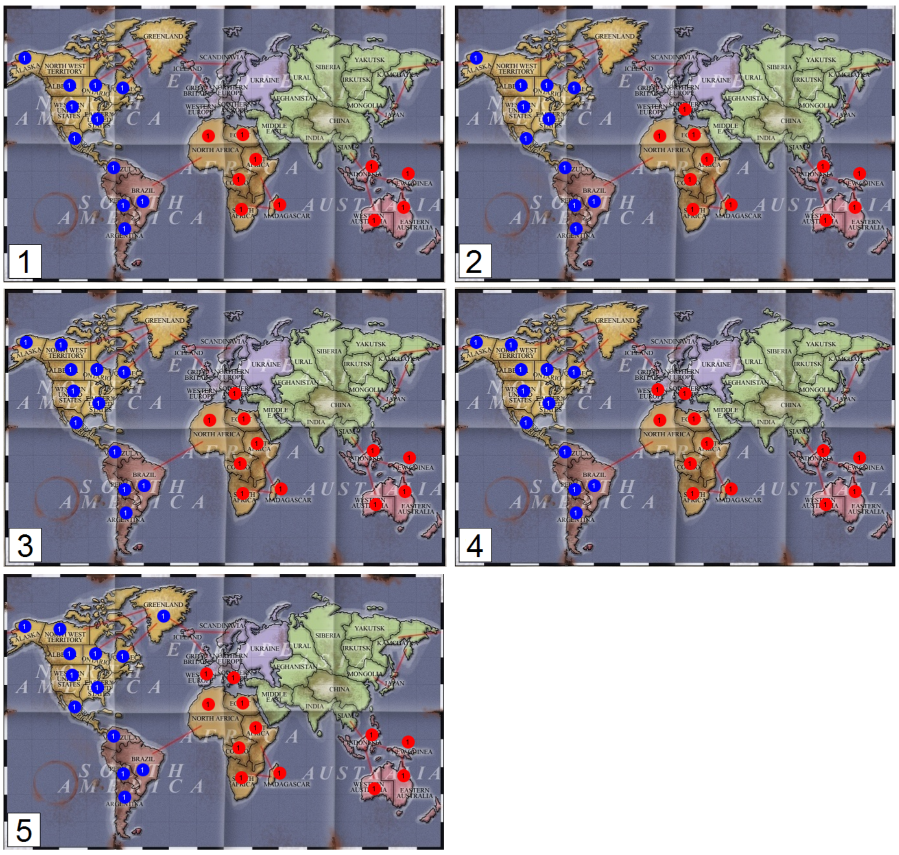

Figure 4.1: Simplest form of counting countries in a continent, with no desire to form clusters implemented yet: the blue human player is able to take North America step by step, and the AIParam doesn't prevent it. In the pictures on the left, it's the human player's turn to place an army, on the right it's the AI's turn.

Africa as before, and it was only once it had gained all the countries in Africa (and I had all the countries in North America but two) that it intervened. When I gave it the chance, it also placed a second blocking army in North America, as the algorithm gives me less points for a continent the more countries in it are owned by enemies.
With the introduction of the cluster Feature, however, giving the AI the desire to collect

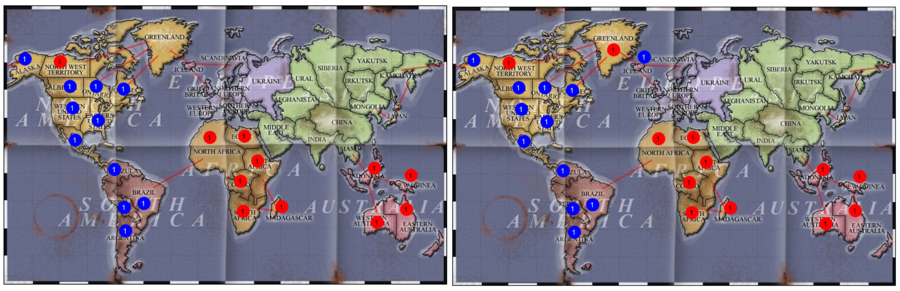

Figure 4.2: Behaviour after modifying the formula for the countries in a continent: After taking all of Africa, the AI blocks the human from taking North America, instead of moving on to Europe as it did in the previous example.

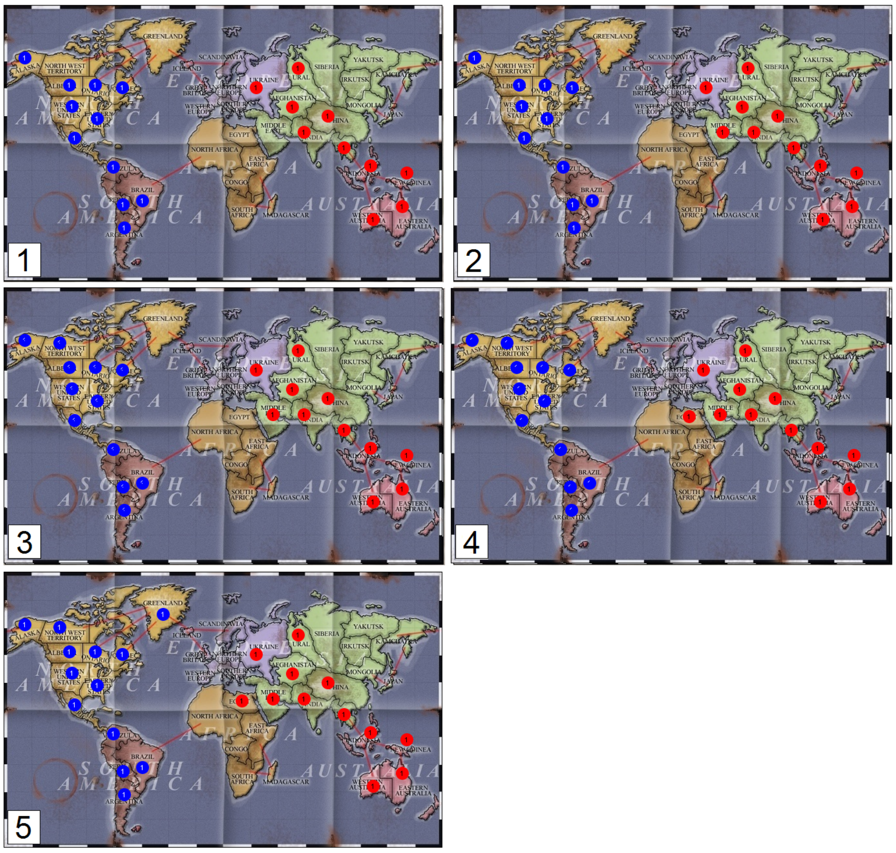

Figure 4.3: After the introduction of clusters, the AI takes care to make all the countries it takes touch each other, even if this means it has to let its opponent gain continents.

countries that touch each other, it stopped blocking its enemy from getting continents (unless it was able to do so by taking a country bordering its cluster). As seen in Figure 4.3, after conquering Australia, the AI is no longer willing to separate from the cluster it has created so far to start conquering Africa, and will instead spread into Asia seemingly without a specific goal. And even though it is now not preoccupied with conquering a specific continent, it doesn't prevent its opponent from getting North America.

4.2 Learning

With the default parameters it started off with, the AI can defeat the game's preexisting "AI Hard" player an average of 3 times out of 100 games.

4.2.1 Untrained

The maximum number of countries the AI conquers without being trained is 24 on average (21 being the minimum for 2 players, which it receives at the beginning, and 42 being the maximum, the number of countries on the map). Figure 4.4 shows the maximum number of countries it managed to conquer in each game in a session without learning.

The maximum score it achieves averages to 0.579694409, 1 meaning a victory, 0 meaning

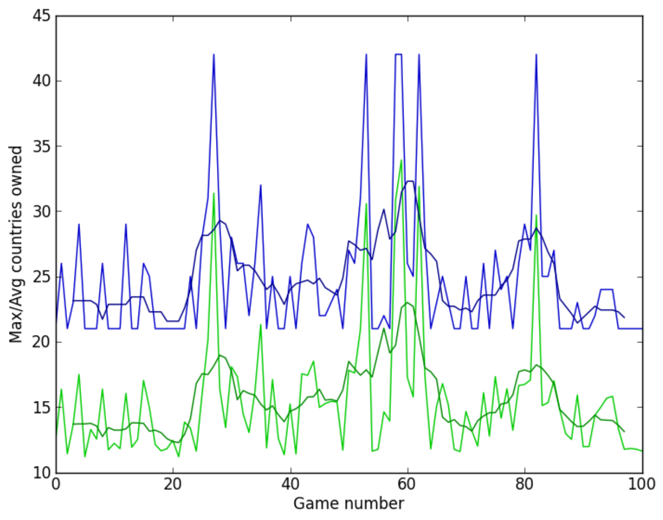

Figure 4.4: In blue: the maximum number of countries the AI gained in each of 100 games, without learning. In green: the average number of countries it had in that game. In dark colours: their respective central moving average over 7 points. 42 countries indicates a victory.

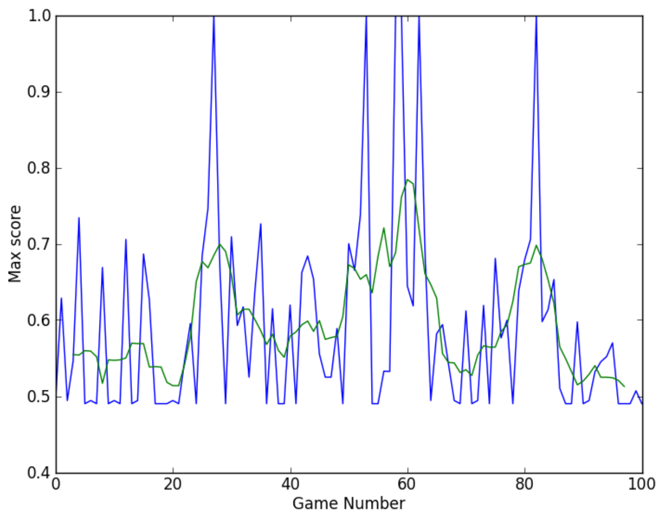

Figure 4.5: Highest score the AI achieved in each game, without learning. Central moving average over 7 points in green. 1 indicates a victory.

defeat, and 0.5 meaning the game situation is evened out between the players, which is the case at the beginning. The scores can be seen in Figure 4.5.

Figure 4.6 gives a sample of the length a game takes. Games where the AIParam wins take between 323 and 1380 turns, around 626 on average. (Note: in this context, a "turn" means one of the stages within a player's turn, such as army placement, attack stage, etc.) In games where it loses, the AI survives for 112 to 721 turns, with the average being 188. This suggests that, if the AI is winning, it takes unnecessarily long to finish off its opponent. I also witnessed a few games where it seemingly had almost won, but fooled around long enough for its opponent to recover and get back to power. I suspect a large part of this is due to the fact that the AI can not look ahead to the attack phase when it places its troops. This problem was even stronger before I introduced the Feature “BorderTroops”, ensuring it at least placed extra armies on the outside of its cluster and not the inside.

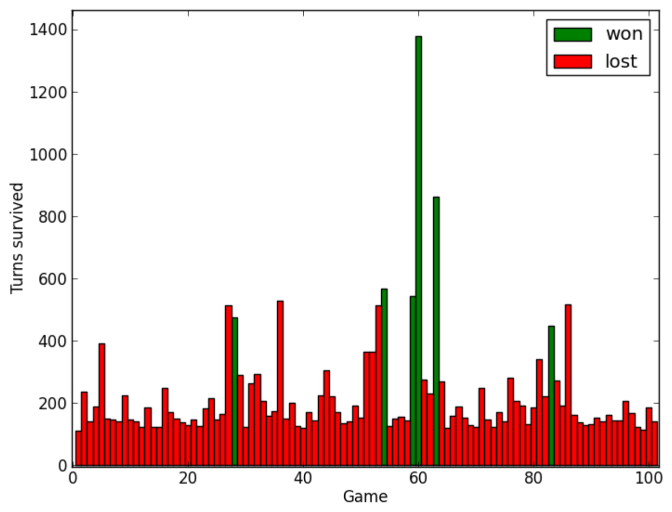

Figure 4.6: Number of turns each game lasted, without learning.

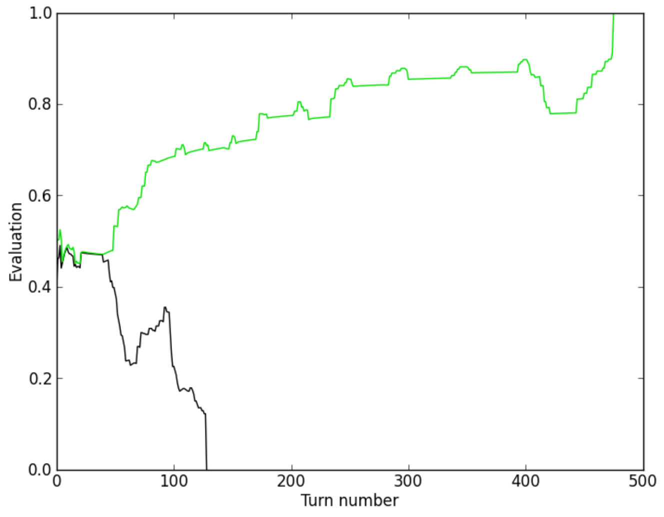

Figure 4.7: Progress of the AIParam’s score in two games, one ending in victory, the other ending in defeat.

Figure 4.7 shows how the AI's score changed over the course of two sample games. As is normal, the score starts at 0.5 at the beginning of the game. The winning curve does indeed grow rather gradually, suggesting the AI might have won sooner if it had been more aggressive. The losing curve is more to the point: the players take turns gaining territories and scores, until the opponent wins.

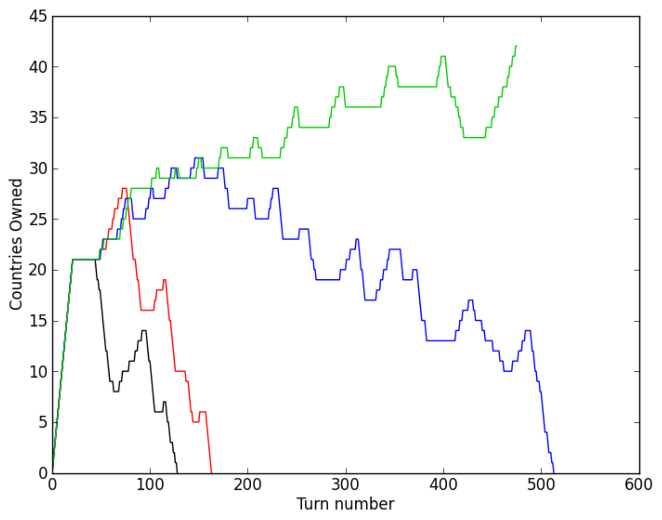

Figure 4.8: The number of countries the AI player owned in each turn, for four different games. The black curve is from the same game as the losing score in Figure 4.7, the green one is the victory mentioned before.

Figure 4.8 shows how the number of countries AIParam owns changed over the course of 4 games, one of which was won. Around turn 400 of the winning game, the AI actually had all the countries but one, yet allowed its opponent to regain 6 countries before defeating him completely. It is difficult to reconstruct the reason for this, but the evaluation function should probably put more emphasis on the importance of gaining countries when the game is almost won. And it is also likely that this was once again caused by the AI putting its extra armies in the wrong place.

For games it loses, it is a common pattern that the number of countries the AIParam owns steadily decreases, and even though it continues making bursts of attacks, it rarely succeeds in recovering all the countries it lost.

4.2.2 Trained

I trained the AI almost exclusively by letting it fight agains the hard player, since the "average" and "easy" AIs are basically just variations of "hard" with more randomness added in. I tried letting it fight against itself, but that proved impractical because the games soon became endless, making it impossible for the training to continue.

Sadly, the training seemed to make the AI worse, not better. The main cause for this appears to be that, despite the normalization mentioned in chapter 3, the algorithm still ended

up greatly reducing “good” values such as the number of countries a player owns, eventually causing the AI to avoid all behaviour that would normally be beneficial.
The parameters that were most strongly reduced were the unique neighbours of both the player and his opponent, the troops in the largest cluster of both players, and the number of their friendly neighbours. These weights were likely reduced the most because they produce the highest feature values (the players always have a lot of enemy neighbours, and a lot of troops in their cluster, etc). It would make sense for the weights marking the opponent’s state to be thus reduced, indicating that it is very detrimental to the AIParam if its opponent has many friendly neighbours. But the values for the AI’s own values were decreased just as much. This caused the AIParam to attack a lot less, since this would increase the number of friendly pairs it has, which the new weights indicate to be a bad thing. The value of owning countries in continents usually decreased slightly too, after the AI lost a few games in which it owned a lot of a certain continent, causing it to avoid conquering continents in later games.
As a result, the AI will sometimes have a few victories at the beginning of a learning session, before sinking into a failure it can’t recover from.

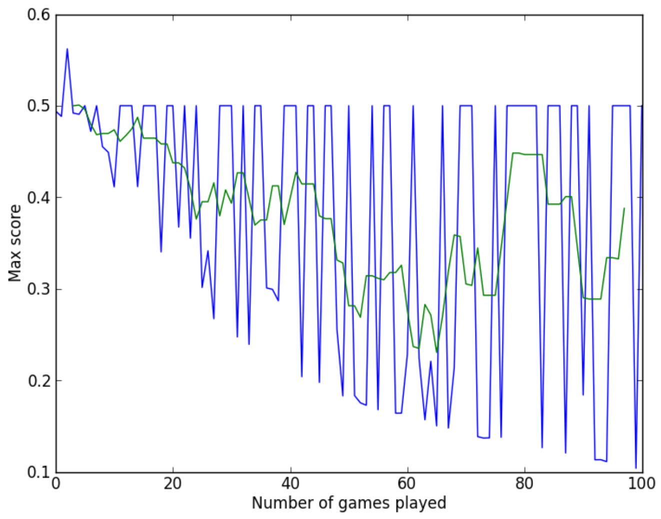

Figure 4.9: The decrease of the score during a learning session. Green is the central moving average over 7 points.

It should be noted that the score is not the best measurement of the learning progress, as the function used for evaluating the score changes progressively. As a result, a decrease in score doesn’t necessarily have to mean the AI did worse, it might also mean that it later awarded itself fewer points for the same accomplishments. Nevertheless, Figure 4.9 shows well what happens during training: after barely a dozen games, the AI has become such a poor player that its maximum score during a game never rises higher than the 0.5 it starts out with. The game decides at random which player makes the first move, and on those moves where the opponent placed first, the first score registered contained the opponent’s first army as well as the AI's, while the games where the AIParam was the first player score slightly better, since they contain a turn where only the AIParam's own army is on the board. This explains the fluctuation between two seemingly stable borders. The lower limit decreases as the weights decrease, giving a worse and worse score to games where the opponent starts.

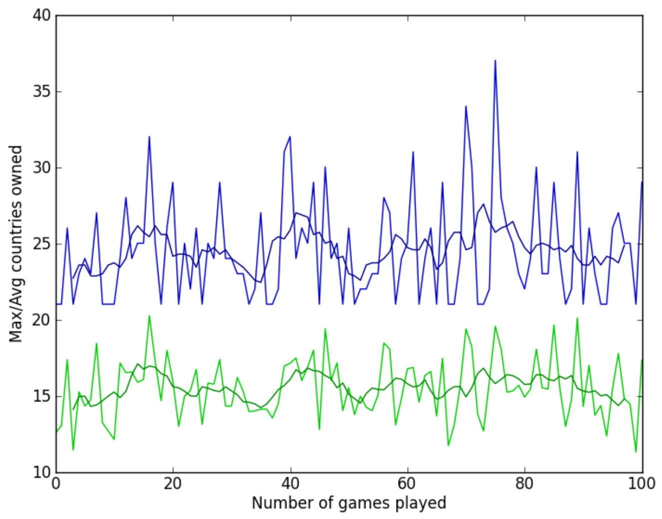

Figure 4.10: Maximum number of countries owned in a specific game during the training round in blue, average countries in green, and their respective central moving averages over 7 points. 21 is the number a player starts out with, 42 would indicate a victory (not present here).

The number of countries, seen in Figure 4.10 gives a more objective measure of success and failure. In many games, the AI doesn't gain more than the 21 countries it started with or a few more, though there are some where it got quite far. But after a certain amount of training, the AI never actually wins again.

The number of turns survived, as shown in Figure 4.11, doesn't change too much, apart from occasional extremes where the AI survives for an extremely long time before being defeated.

Its behaviour after being trained is roughly as follows: during the initial setting up stage, it will choose countries which are quite spread out, presumably because it now thinks having a large cluster is bad. One positive effect of this is that it is willing to block its opponent from gaining continents again. It will try to gain a continent whose worth has not been reduced too much by the training. Since it no longer cares about placing troops within its largest cluster, it tends to put a lot of troops into a single country in the enemy's territory, which it took to prevent him from gaining a continent. This causes the opponent a few problems before the he eventually gains the continent he was aiming for. It tries a few times to steal a country from its opponent, preventing him from getting the continent bonus, but usually it uses too few troops to do so. It then takes the opponent a few turns to get the AIParam under control and win the game.

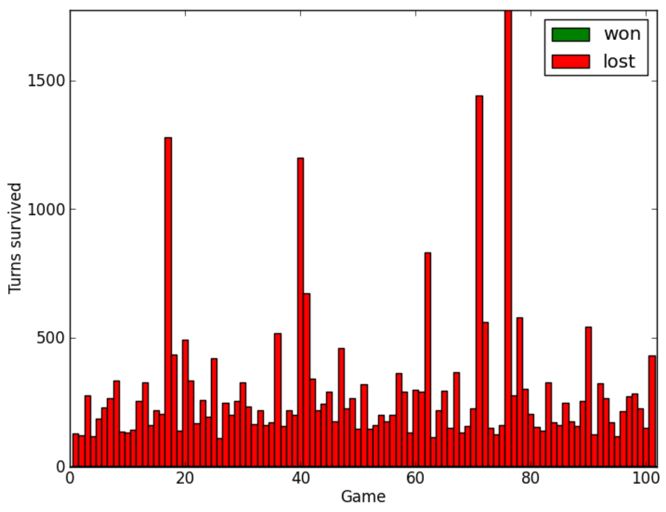

Figure 4.11: The number of turns the AI survived in this example of a 100-game training round without a victory. The giant outlier at around game number 75 goes up to 21324

5 Conclusions

5.1 Can a Risk AI work without looking ahead?

Overall, I was positively surprised at how well the AI works despite not planning anything at all. Looking ahead is definitely not as necessary as one would expect.

There are, however, a few situations where the lack of foresight is a definite disadvantage. The most obvious of which is that, during the army placement stage, the AI doesn't prepare potential attacks it will make during the attack stage. I partially solved this problem with the Feature BorderTroops, but not completely.

It is always possible that, with enough ingenuity, a Feature could solve even this problem. But the feature in question would have to be rather complex: it would have to consider the AI's current chances of conquering a certain country — encouraging it to place troops to improve these chances — and the value of conquering a that country. However, it would not be allowed to use its weights to determine the value of conquering a country, like it does in the attack stage where it actually looks ahead, because otherwise the evaluation function would no longer be a linear multiplication, and the derivative by a certain weight would change. It would therefore be forced to evaluate the worth of a country by some fixed rule that can not be modified by learning, which would already be somewhat problematic. And the mental acrobatics it would have to perform to calculate the worth a country would be rather complex, making it much simpler to just do an actual look ahead instead. In fact, the transition between this foresight-simulating feature and actual foresight would be so fluent, it would be hard to tell them apart.

It is therefore possible that the game Risk is too complex to be played completely without looking ahead, as the fact that each turn is divided into stages that depend on each other is detrimental to the success of an AI that can't plan ahead. Of relevance is also the fact that a player's turn is very long, involving many actions which require some form of coordination.

5.2 Possible limitations of TD(λ) in combination with Risk

There are some aspects where the TD(λ)-algorithm might not be adequate to train a Risk player.

For one thing, unlike chess, for which Baxter and Weaver [2] used TD(λ) to create a successful

AI, Risk has a much bigger map. This increases the possibility of unrelated events happening in different parts of the map, which the updating algorithm can't distinguish, causing it to "blame" unrelated factors.
Another potential problem might be that Risk can produce very large feature values, despite the normalization. And since the weights are updated with the derivative, which is directly linked to the feature value, this causes some weights to be changed much more strongly, until they become negative. This is unlikely to happen in chess, since it will never be the case that the chess AI had 100 queens before a big drop in score and the algorithm therefore decides to give a negative value to the queens.
On a minor note, Risk games usually take a long time, and if the players are AIs that don't yet know what they are doing, a game can potentially take forever, blocking the progress of the training.

5.3 Possible improvements

There are several approaches that could make the make the AIParam as it is now improve its skills.

5.3.1 Foresight and planning

If we were to abandon the attempt at staying pure and faithful to the learning approach, we could simply allow the AI to do a little bit of planning, at least in the areas where its usual evaluation function doesn't suffice. Increasing the scope of how far it looks ahead can work within limits on a fast computer, but a complete tree search would have too many possibilities.

5.3.2 Improving the parameters by other means

It is also tempting to play a few games and to modify the weights by hand, according to my observations of the AI's behaviour. Or it might be interesting to modify these parameters using a completely different algorithm than the TD(λ) method. For example, one could try an evolutionary algorithm, where the parameters could mutate and breed according to their success.

Adding more features is sure to be useful as well. There is a sheer unlimited number of possible features one could use to help the AI analyse the game situation better. Analoguous to the way a player can watch the AI play and adjust parameters to correct faulty behaviour, he can also add in a new feature to allow the AI to notice things it missed before.

5.3.3 Nonlinear algorithms

The TD(λ)-algorithm is the equivalent of a very simple neural network with only one node. It is linear and only allows the AI to look at the features in isolation. Using a non-linear method instead would allow it to combine features with each other in more complex ways, e.g. realizing that owning a certain country might be more valuable if it also owns another country. Some of the features I implemented already work around this a little, e.g. the "value of number of countries in continent" feature combines the worth of the continent's individual countries, making them more valuable together. Using a nonlinear, multilayer network would offer many more possibilities in this regard.

Bibliography

[1] Sillysoft Games. Lux delux. `http://sillysoft.net/lux/`, March 2013.

[2] J.Baxter, A.Tridgell, and L.Weaver. TDLeaf(λ): Combining temporal difference learning with game-tree search. In *Proceedings of the Ninth Australian Conference on Neural Networks (ACNN'98)*, 1998.

[3] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, Lecture Notes in Computer Science, pages 282–293, 2006.

[4] Yura Mamyrin. yura.net domination (risk board game). `http://domination.sourceforge.net/`, March 2013.

[5] Richard Zhao Richard Gibson, Neesha Desai. An automated technique for drafting territories in the board game risk. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2010.

[6] Richard Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

Declaration of Authorship

I hereby declare that this thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the bibliography and specified in the text.
This thesis is not substantially the same as any that I have submitted or will be submitting for a degree or diploma or other qualification at this or any other University.

Basel, date

Manuela Lütolf

Philosophisch-Naturwissenschaftliche Fakultät der Universität Basel
Dekanat

Erklärung zur wissenschaftlichen Redlichkeit
(beinhaltet Erklärung zu Plagiat und Betrug)

(bitte ankreuzen)
☐ Bachelorarbeit
☐ Masterarbeit

Titel der Arbeit *(Druckschrift)*:

Name, Vorname *(Druckschrift):* ______________________________

Matrikelnummer: ____________________

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe.

Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Diese Erklärung wird ergänzt durch eine separat abgeschlossene Vereinbarung bezüglich der Veröffentlichung oder öffentlichen Zugänglichkeit dieser Arbeit.

☐ ja ☐ nein

Ort, Datum: ____________________

Unterschrift: ____________________

Dieses Blatt ist in die Bachelor-, resp. Masterarbeit einzufügen.