Universiteit Leiden
The Netherlands

Opleiding Informatica

Comparing Different Agents in the Game of Risk

Jimmy Drogtrop

Supervisors:
Rudy van Vliet & Jeannette de Graaf

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
`www.liacs.leidenuniv.nl` 07/08/2018

Abstract

In this thesis, we compare the agents Knapsack, Angry, Evaluate and Random and decide the order of strength between them for two-player Risk without neutral factions. We start by stating the rules for this game and compare them with the original board game rules. Then, we choose the engine to use for a project like this. After this has been decided, we describe the agents and come across factors that decide their traits. This requires us to take a brief look at the aggressiveness of agents, and the effect of planning moves in advance against the effect of greedy agents. Eventually this brings us to the conclusion that the order of strength is Knapsack2D without greedy $<$ Random $\approx$ Knapsack3D without greedy $<$ Knapsack2D with greedy $\approx$ Knapsack3D with greedy $<$ Evaluation $<$ Angry.

Contents

Chapter 1

Introduction

1.1 Introduction to the Project

Risk is a very popular game in which players try to conquer countries on a world map using troops generated from the owned countries. There are a lot of different strategies to play the game, most of them strong in different situations. Previous work on Risk covers a broad variety of subjects to give insights into this kind of complex games.

One popular direction in previous work is the creation of generally strong agents to satisfy different experienced players for commercial purposes. The complexity of the game and the massive number of decisions still form a challenge in creating agents and keep it interesting to research them. Some of these challenges are an infinite state-space due to having no limit on the number of armies available [Wol05] or the difficulty of calculating future reward for making specific decisions and calculating expected reward of an invasion due to the chance involved in battles [Wol05].

Another direction of research is collecting useful insight into sub-parts of the game, trying to find facts in math or common logic for some situations of the game to build agents or human game strategies upon. For instance, many Risk players know the value of blocking an enemy continent, where the success-rate to win a battle is less known [Ber11]. This shows that aggressive agents can be strong; while ignoring other aspects of the game, win probabilities could make engaging in battle before being attacked attractive. There are also many ad-hoc argumentations to build upon specific statements and theories. In most cases, we search for a balance between these statements, sometimes requiring us to leave out useful information. We try to keep that in mind.

Then there is also the human side of the game with topics like diplomacy and manipulating. There are even agents in digital multi-player Risk that play together. Although this touches my initial motivation to start this project, which was trying to state something about human-controlled Risk, we will not focus on that. It is not relevant for this project.

One thing to keep in mind is that the effectiveness of different strategies also depends on the number of players. In two-player Risk without neutral factions enemy conquest always leads to the player losing countries, while this is not the case when the enemy attacks a third player. Therefore, it might be more attractive to wait for other players to weaken each other while gaining strength yourself in Risk with multiple players than with just two players. Taking a look at two-player Risk might be useful for gaining insight into the endgame of the game.

In this project, we take a closer look at these directions of research for two-player Risk without neutral factions. We are creating a couple of agents, some of them using math and common logic. We start by selecting the platform we would like to use for running agents and provide a guide for people also trying to write agents for Risk. After that we want to see what insights we can find by having some selected agents compete against each other one-on-one in pairs. The used agents are Angry, Random, Evaluation and some variations of a Knapsack agent. We hope that these match-ups result in useful insights. During our project we also encountered some interesting considerations. Think about the aggressiveness, creating plans in advance and looking ahead, greedy agents and their difference for two-player and multiple-player Risk.

We study the effectiveness of the Angry, Random, Evaluation and Knapsack-agents in two-player Risk without neutral factions. We expect the following order of strength with these agents: Knapsack2D without greedy $<$ Random $<$ Evaluation $<$ Knapsack3D without greedy $<$ Knapsack2D with greedy $\leq$ Knapsack3D with greedy $<$ Angry. This is largely based on the idea that controlled and balanced aggressiveness pays in two-player Risk without neutral factions and the idea that balanced and informed choices are more effective than greedy approaches (with the used agents). However, it might be hard to isolate the actual reasons for the effectiveness of one agent over another in a game as complex as Risk.

1.2 Thesis Overview

Chapters 2 to 4 contain the practical preparations for this project. Chapter 2 presents the rules of the original game and some popular variations opposed to the rules preserved in this project. Chapter 3 explains the engine that was used and how to use it for similar projects. Chapter 4 introduces the used agents.

After the practical preparations and an explanation of the environment of the project, we describe how we conducted our experiments, and present and discuss the results in Chapter 5. Then, Chapter 6 contains the conclusion of the main research topic of this project, which is the effectiveness of the different agents, and the conclusion of the minor topics in this project. Finally, we discuss possible future work.

Chapter 2

Rules

Because Risk is such a popular game, it is played in many different ways. It is therefore important to specify in detail how we are playing Risk during this project. Overall, we try to stay close to the original board game. As a base, we use the two-player Risk Domination (passive) version.

2.1 Set-up

In the original two-player domination game, two controlled players and four neutral factions are used. The game ends when the countries of the controlled enemy are taken. The players can use the neutral factions as active allies, but in the passive two-player version, the neutral factions will only function as a buffer. We will use neither of these versions. We have chosen for two-player Risk to create a direct confrontation between two agents. This is also why there will be no neutral factions. The result is a map that contains only the two players. We will use the classic Risk map.

The classic Risk map (Figure 2.1) contains our world divided into 42 countries. It consists of six continents, varying in number of countries. Normally, both players get nine randomly assigned countries, while the neutrals get six, also random. As there will be no neutrals, both players get 21 randomly chosen countries. One variation to the original game involves picking countries in turns, or the pickCountry phase, but random pick is standard for the original two-player Risk and we stick with that. On every country, an army/troop is placed from the player it belongs to. During the whole game, a country will always contain at least one army.

Now, a fixed number of troops is given to the players, who then distribute them over owned countries in turns. Troops that are placed within a turn can always be distributed amongst different owned countries, also in later stages of the game. This is the starting phase of the game in which the players start planning their strategies and goals. The way this starting phase will work differs largely for different Risk-versions, as there are many choices to make in how to perform this part of the game. The person that starts placing troops is the person whose turn it was to pick the next country. If the countries where distributed randomly, the starting player is

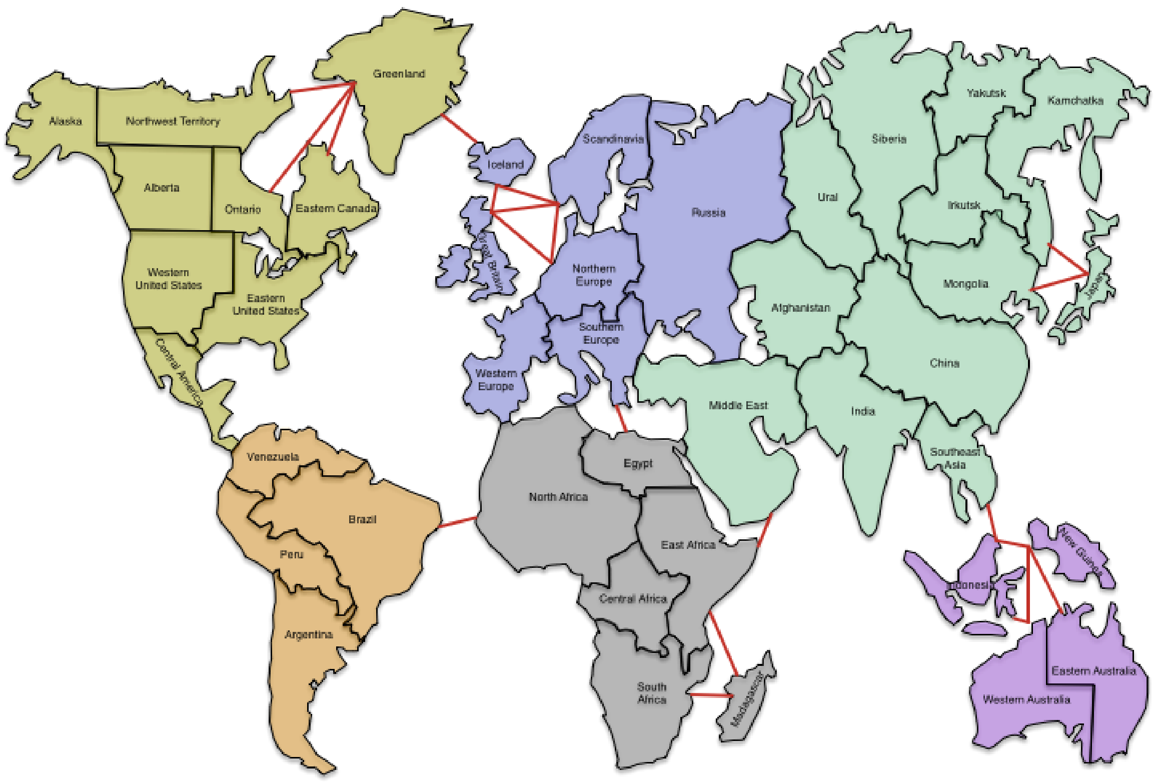

Figure 2.1: The classic Risk map. Source: `https://www.graffletopia.com/stencils/455`.

also picked randomly. However, for testing purposes, we select the starting player in advance.

In some versions (not ours) the player who starts last gets a boost in number of troops. Sometimes, the players place one troop in turn, in other versions three. In the original two-player versions, both players place three armies on their own countries and one in a neutral one. In the engine used by us, four troops are assigned every turn. If the player does not have four troops left, he will assign the remaining troops. The troops can be distributed over owned land in any way the player wishes. In our case however, we choose to distribute these troops randomly for reasons that will become clear later in this thesis. The total number of troops to assign also differs. In the original game, this number depends on the number of players, using 35 troops for three players, subtracting five for every extra player. In the two-player versions, neutrals get 18 troops and players get 27 troops. In the used engine, players get 40 troops in total, divided into one for every country and nineteen to distribute freely. We also stick with that.

The first troops have been placed and the set-up of the board is complete. Now, the battle can begin.

2.2 Gameplay

Every turn, a player goes through four phases:

- Card hand-in or card phase.
- Place troops or placeArmies phase.
- Invasions or attack phase.
- Fortifying (migration/reinforcement) phase.

Some phases will be entered at an irregular basis. The card phase is also entered after an opponent has been defeated. After the fortification, a reinforcement card will be drawn when at least one country was conquered. This can be called another phase, but we will not do so. During invasions, when a country is conquered, we reach the moveArmiesIn phase (making this a subphase of invasions). The game ends when only one player remains.

2.2.1 Card Hand-in

This phase is called as a stage before the troop placement phase and also after an opponent has been defeated, but is only relevant when someone has three or more cards. If you conquer at least one country in a turn, you get one reinforcement card at the end of that turn. If you play with more than two players and you defeat an opponent, you get all of his cards. You can have at most four cards in your hands. In most versions of the game, and also ours, if you have more than four cards at some point, you have to hand in your cards at that moment. This means that the cards have to be handed in either during the troop placement phase at the start of a turn or directly after defeating an opponent. The troops you get from these cards have to be placed immediately.

The cards have a country and a cannon-unit, horse-unit or infantry-unit on it. To get a reinforcement boost, you have to match one of each unit, or you must match three of the same unit. Some versions also have wildcards, which can be used for any type of unit. Also, some versions give the additional troops only for some countries, if that country is included on one of the cards you handed in. The original game has wildcards, our version does not use them and also does not use limiting the card bonus to some countries. Figure 2.2 displays the possible sets to hand in.

Now, the number of troops that reinforcement cards give depends on the used system. For example, the two-player and multi-player version use the following ladder: 4,6,8,10,15... and then increasing five every time until 60 is reached, in which case you can restart at four or stay at 60. The used engine also uses this ladder, but it can be changed to other ladders. What makes it interesting is that these numbers are shared amongst the players, so if we hand in cards to get four troops, the next player that hands in his cards will get six troops. Therefore, it might be smart to keep your cards for a later round.

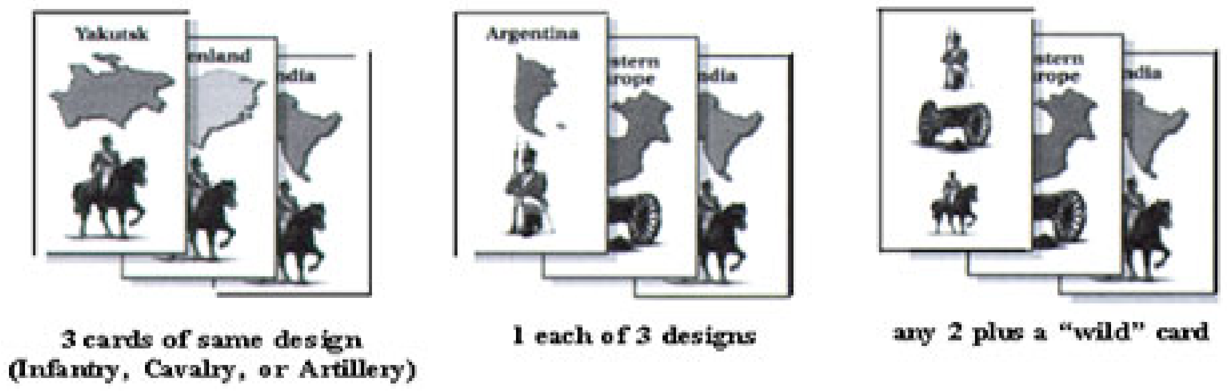

Figure 2.2: Possible card hand-in sets including wildcards. Source: `http://www.ultraboardgames.com/risk/game-rules.php`.

2.2.2 Place Troops

The number of troops you get each turn depends on the version of Risk you are playing. You must place all the troops you receive immediately. You cannot save them for later. Troops can be received from three sources: number of countries, owned continents and reinforcement card boosts. The reinforcement cards boosts are awarded in the card phase instead of the troop placement phase. The number of troops received from the number of countries and continent bonuses is called the income of a player.

- Number of Countries. In the original game and in the used engine, the player gets $(\#\mathit{Owned\ countries})/3$, but at least three troops. In many variants, this number is fixed, giving for example five troops for any number of countries.
- Continents. On the classic map, six continents exist. For owning a full continent at the start of a turn, troops are awarded according to Table 2.1. If a player is assigned a full continent in the starting phase, he will get an army boost in most versions of the game.

Continent	#Countries in continent	#Troops reward
North-America	9	5
South-America	4	2
Europe	7	5
Africa	6	3
Asia	12	7
Australia	4	2

Table 2.1: Troops rewarded for owning a continent.

2.2.3 Invasion

You can attack a country from an owned connected country. This is called an invasion. In this invasion the country you attack is the target and the country that is owned by you is called the source or the starting country. We also have paths, which we will get to later. These paths include the starting country, a final target, and also contain the enemy countries needed to be able to attack the final target. A connected country is a neighbouring country, which is sometimes connected using a sea route. Battles are part of an invasion and this is where the success of an attack is decided. An invasion consists of one or more battles. During the invasion phase, you can do as many invasions and battles as you like.

A battle is executed using a dice game, where the attacker uses at most three armies and at least one army, even if the attacker has more troops in his country. One army must always remain in the attacking country. The defender uses a maximum of two armies. Most game engines, also ours, deploy the maximum number of troops (with a maximum of three for attackers) they can afford to send to battle. In the original board versions, the player can send out fewer troops than possible, even though this would not be beneficial in most cases due to a reduction in winning chance. This is very different for one particular variation of the game in which the defending player can send out his troops after the dice roll of the attacking party, possibly limiting his loss. One die is used for every deployed army. All dice are thrown at the same time. When the dice are thrown, the highest numbered dice and (when applicable) the second highest dice are compared. In each comparison, the highest die wins and kills an army of the opponent. When tied, the defender wins.

After every battle, the attacker may abort the invasion. A new invasion with the attacking country and to the defending country is allowed, even another invasion between these countries. The attacker can keep invading countries connected to his own countries. A conquered country must directly be claimed by the remaining troops used in the invasion. This is called the moveArmiesIn phase. You can then send troops from the conquering country with them. Remember, one army MUST stay in each country at all times. The minimum number of troops to send on is the number of dice used in a battle, meaning that when there are more than four armies in the attacking country, we have to pass on at least three troops. In the original rules, we have to send on the troops that survived the battle. This is effectively the same, as a country can only be conquered when all dice comparisons are in favour of the attacker, thus the attacker did not lose any troops in battle. It is permitted to attack with a country you have just conquered.

2.2.4 Fortifying

In the original game, you can perform one migration from an owned country to a directly connected owned country. During this phase you can move as many movable troops to a neighbouring country as you wish. The movable troops are all troops that have not been moved from another country to their current country in this very turn. There are some version of Risk in which the player can make more than one migration, which is also the case in the used engine.

Chapter 3

Game Engine

For a game as popular and well-known as Risk, there is a commercial value to be gained by writing a Risk game or engine for recreational purposes or for research. Therefore, many different Risk games exist. Most of them are not made for experimenting with agents, do not have the necessary features, are too different from the original board game or do not satisfy the needs of a useful engine in some other way. After these considerations, three choices remained, each one with its own pros and cons.

3.1 Requirements

My personal, original goal of gaining insight in the original domination board version was important in deciding the requirements for an engine. For that reason, it was important that the rules would remain close to the board version. I came up with the following list of requirements that needed consideration:

1. Custom number of agents.
2. Possibility of using the classic Risk map.
3. Random, turn-based pick and predetermined assignment of countries.
4. At least one army must remain in a country at all times.
5. Fixed number of troops to be assigned in the set-up phase, similar to the number used in the board game.
6. Distributing the armies in turns, with the number of troops being the same as used in the original board game.
7. Option for fixed or randomly picked starting agent.
8. Easy editing of the used agents.

9. Easy simulation of games.
10. Choice whether or not to use reinforcement cards, and when activated the ability to pick the original troop ladder.
11. Possibility to continue attacking from a country you conquered this turn.
12. Possibility to stop an invasion and later continue that same invasion.
13. Ability to use less than the maximum number of troops in an invasion.
14. Possibility to execute at most one migration.
15. Possibility of neutral factions, passive and active.
16. Uneven assignment of countries for playing with neutral factions.

The engine environment is also important. Take for example the SDK, software development kit, and documentation used to write your own agent and tweak the game engine itself. Every SDK has different code available, and when more functions of code are already included by the engine, you have to write less code yourself. However, it is not desirable to have many useless or unclear functions that clutter your overview. The framework for an agent must be easy to use. It can also be very helpful to have an active community and forum from which you can get new insights or get help when you get stuck using a specific code-function. For the same reason, many example agents can be helpful. It can also save a lot of time if the agents for the experiments already exist. Testing should also be quick, as we will be executing many simulations.

Then there is also the graphical aspect and coding language. It is way easier to improve an agent of which you can see its moves clearly, while maintaining overview of the whole play field. The negative side of this may be a reduction in performance. Coding language is not really an issue most of the time, as many of these engines use easy to use languages.

3.2 Engine Choice

After taking the requirements into account, the best options that remained were writing an own engine or using one of the existing engines, *Domination by Yura* or *Lux Delux*.

Own Engine

Writing an own engine has many advantages. The most important one is its flexibility, as you decide exactly how the game is played. Therefore, you are not bound to unchangeable rules used by an existing engine. You also know exactly what every coded function does, and are hardly limited by the way the engine works when writing your agents.

However, there are several reasons not to write an engine yourself. There are already many engines available, and it could be time consuming to write the engine itself, but also to write some coded function SDK-agents do have access to. Then, there is no community aimed at your engine that can help with coding problems or providing popular agents to test against. Another reason could be that it is easier to share your agents when they are written using a skeleton people already know. Even though these reasons are mostly time and community-related, the graphical display can also be a reason not to write the engine yourself. It would have been extremely time consuming to write a clear graphical display in which errors in the agents could easily be detected, in addition to the non-graphical program.

Domination by Yura

Domination [Mam03] was written by Yura Mamyrin and can be downloaded for free. It offers an SDK to develop your own agents and maps, a lot of statistics during battles, some pre-made functions that speed up the process of writing your own agents and a few agents that come with the game. The community also offers some extra agents too. Agents are written in Java.

Lux Delux

Lux Delux [Sil02a] offers most features Domination also offers. However, Lux Delux is more extensive on some aspects. This comes at a cost, namely that the game is not free, but it does have a trial version in which a few games can be played for free. Our license was bought through Steam [Ste15]. Lux Delux might cost a little more effort to reach the level at which you are able to write your own agent due to setting up the environment and an arguably more complex structure of the program. Lux Delux agents are written in Java.

Compared to Domination, Lux Delux also contains a plug-in library with many maps and agents. It contains an SDK to develop your own agents and maps just as Domination has, and also contains some statistics. It has more really useful pre-made functions and a bigger availability of agents. It has more game options, making it easier to set up the game the way required for a project like this.

Eventual Decision

Lux Delux seemed to fit my needs most, as it allowed me to easily set up the options for playing the game in a setting close to the original game, which for me was a goal in picking the engine. Lux Delux also seemed more extensive in its functionality, cutting in the time necessary to develop agents and functions. The big availability of agents for testing against many different enemies was another reason to pick Lux Delux over the alternatives. It did not satisfy all requirements (in particular, it did not satisfy requirements 13, 14 and 16), but did satisfy the ones important for executing our experiments in an environment representative for comparisons with the original board game. We used Lux Delux version 6.56.

3.3 Writing Agents

To get started with Lux Delux, we have to set up some things to quickly load agents into the engine. It is assumed that the SDK [Sil02b] provided by Lux Delux is downloaded. The SDK comes with some useful files, amongst which some instructional ones like the readme_AI.html-file that can help with setting up your environment. It is advised [Sil02c] to copy an existing agent with the ".java" extension into a new file, for instance "random.java", and edit the file to get your agent. The documentation also advises to "pick one that extends SmartAgentBase if this is your first bot" to arm you with some extra functionality. For comprehension of agent interaction, the readme_AI file advised to look at the agent of Angry to get a clear picture of how the agents work. The java-files are located at "*SillysoftSDK\src\com\sillysoft\lux\agent*"

After writing an agent, we want to place it into the engine. To get the agent eventually into the engine we need to have two things installed; Apache Ant [Pro99] to turn java files into class files and Java Developer Tools [Ora] to run Ant. After the class files are created they have to be placed in the folder from where new agents are loaded into the engine. As this will be done on a regular basis, it pays to automate the process. Probably the easiest way to do this is by editing the build.xml file provided in the SDK folder to copy files directly to the folder from where the agents are loaded into the engine. Instead, I created bat scripts to do so, in order to split tasks. These can be found in Appendix A.

3.4 Agent Structure

Lux Delux agents use a few functions that the engine calls to run the game. These functions are the phases an agent passes through; the skeleton of an agent. The functions follow the phases of the game as explained Chapter 2.

```
int pickCountry() // Decide which country to pick in the setup stage
void placeInitialArmies( int numberOfArmies ) // The placement of troops in the setup stage
void cardsPhase( Card[] cards ) // The card phase

void placeArmies( int numberOfArmies ); // The spending of an agent's income
void attackPhase(); // The attacking of an agent
int moveArmiesIn( int cca, int ccd); // The movement of troops after a conquest between
                                     // the Attacking country and the Defending country
void fortifyPhase(); // Movement of troops after completing all invasions of a turn
```

Lux Delux also provides iterators over the countries of, amongst other things, agents or continents. Some helpful functions that Lux has built in are functions that can be used in a variety of cases. This includes functions to find shortest paths, cheapest paths, weakest enemies, neighbours of a country, attackable neighbours and many other useful information to build agents quickly. Documentation can be found in the Lux_AI_javadocs.html-file of the SDK.

Chapter 4

Agents

The goal of this project was to compare several Risk agents and see if we could gain insight into the effectiveness of these agents for a game like Risk. To do this, I picked the existing Angry agent and wrote three myself: Random, Evaluation and Knapsack.

Some notes on the agents: the pickCountry phase, the initial armies phase and the card phase are handled in a way that is similar for that phase for all used agents. For the experiments we use randomly distributed countries in the set-up phase, resulting in never actually using the pickCountry phase. This is also the case for the initial troop placement, where in most experiments these troops will be randomly placed. If the initial troop placement is not random, the initial distribution of armies simply calls the function of the troop placement phase. The card phases depend on the engine to do the work for them; the engine will take over if the agent has five or more cards after returning from this phase, automatically cashing in sets until the agent has at most four. The fortify phase is the same for all self-written agents, but different for Angry. During the fortify phase, we check every owned country with at least two troops. If it has no enemy neighbours, we check if there is an owned neighbour with enemy neighbours. If there is one, we move all our movable troops towards it. The behaviour of these four phases is fully described for the (Angry) agent provided by the SDK for the sake of completeness of the descriptions. The description will not be repeated for the self-written agents using similar executions of the phases.

The structure of Angry was used as a base for the other agents written during this project. Angry does not extend smartAgentBased. This means that most functions are written by me or provided by the engine instead of the smartAgentBased-file.

4.1 Angry

The Angry agent makes use of the idea that countries with many enemy neighbours have more options during the attack phase, hopefully resulting in more valid paths and conquering more countries before getting stuck because there are no valid targets left. The agent does not concern itself with continents except when manual picking of countries in the set-up stage is enabled. However, as mentioned earlier, the picking of countries is done randomly in our experiments. It tries to improve income solely by having more countries.

Motivation for Choice

There were several reasons to pick the Angry agent for experimenting. Because all agents were built from the angry.java file, some phases from angry are used in other agents. This also meant the agent was going to be described anyway, making the step to taking it into the experiments smaller. This reason is not good enough on itself, but there were other reasons. For instance, Angry is a simple agent, but it uses the interesting idea of keeping its options open. In some way this counters the problem of having a big group of troops got stuck in some country during the attack phase because there are no more adjacent enemy countries to attack. Angry also considers getting income only through country bonus (and card bonus), which can be interesting to compare to some other agents that concentrate on continent bonuses to find the importance of both sources of income. Being an agent provided by the engine itself is a bonus.

Phases

When picking countries (which happens randomly in our experiments) in the set-up stage, the original version of the agent first picks the smallest continent with no troops in it. When there are no empty continents left, it will place his army in a country of the smallest continent that has an unowned country left, trying to get some continents. The chosen country will preferably be a country that has neighbours that are also owned. If it cannot find one, we pick a country that has the fewest neighbouring countries. The placement of armies in the set-up stage happens in the same way armies are placed in every normal placement phase.

When we get to a card phase we do nothing. Just as with the other agents, the card phase will be handled by the engine.

In the placeArmies phase, we choose an owned country with the highest number of enemy neighbours. There, we place all of our troops. In the attack phase, we iterate over all (owned) countries with at least two troops in it. For every country, we look at its weakest neighbour, if it has one, and attack it if we have more troops. We fight the full invasion until the enemy is conquered or the agent is out of troops to attack with. If we made at least one attack, we repeat the process; if not, we know that the remaining options for attacks are not good enough. If we conquered a country, we place all troops that we can move on the conquered country, unless the attacking country has more enemy neighbours. In that case, we leave the troops we do not have to move, in the attacking country.

After the attack phase we reach the fortify phase where we go through all countries. We check if we own a country with movable armies that has an owned neighbour with more enemy neighbours. If that is the case, we move our movable armies, which are all troops (that were not already moved) except one, to that country. If we find a country that we own that has no enemy neighbours, we move its movable armies to a random owned neighbour, hoping to find a country with enemy neighbours at some point.

Predictions/Traits

Even though Angry is very simple, it is also quite strong against simpler agents. Concentrating troops may cause this agent to leave its flanks exposed, but due to its aggressive nature (trying to take many countries and/or attack with more risk), it will be able to easily try to take these back, possibly a good tactic for succeeding against less aggressive, simple agents. For attacking, concentrating troops gives us a big probability of winning an invasion, while keeping options open makes sure this agent can continue attacking as long as it has the troops to do so. It also picks the weakest neighbour, meaning we possibly lose less troops, having more troops for further conquest. However, the greedy choice of the weakest enemy is not necessarily the best choice. It is conceivable that the pure focus on aggressiveness will make it easy to take countries to recover income lost to enemy conquest. By doing so, Angry has a big chance to be one of the best agents tested in this thesis.

4.2 Random

The self-written Random agent does not run completely random as the name might suggest. There is a preference for placing armies in countries that already have troops in it or have enemy neighbours. The agent also uses the win probability (as explained later) to decide whether or not to continue an invasion, rather than tossing a coin. There is a preference for continuing an invasion if the probability to win the invasion is higher than when the invasion was started. In moving the troops in after having conquered a country, there is a 50% probability that the full army is sent to the conquered country. However, every possible troop placement, every possible attack and every number of troops to send into a conquered country has a chance of being picked.

Motivation for Choice

The random agent was originally picked as a bottom-line. Every reasonable agent should be able to beat a random agent when multiple games are played. Because a totally random agent would not fit a game as complex and with so many different choices and decisions as Risk well, it would be totally destroyed by simple agents. For this reason, the agent is a bit stronger than a totally random agent.

Phases

When placing the armies, the random agent will place troops on an owned country one army at a time. For every country, we keep track of how many armies it contains, or remember the value 1 if it has no enemy neighbours, using a variable named deservingArmyValue. The probability a country has of being picked is proportional to deservingArmyValue. We keep picking countries this way until all armies are placed.

In the attack phase, we compile a list of all possible attacks from owned countries, where an owned country needs at least two armies to be able to attack. We then randomly pick one of the attacks from the list to start an invasion. For every invasion, we calculate the probability of winning the invasion. Then, for every subsequent battle, we recalculate the probability to win the invasion. Every time our probability of winning is not bigger than the original win probability, we reconsider if we want to continue attacking. The invasion is also stopped when we are out of troops to attack with or conquered the target country. If we stop the invasion, we will not reconsider it again. If a country has been conquered, its possible attacks are added to the list. We check every possible attack until there is not a single eligible one left.

If we have conquered an enemy country and call the moveArmiesIn function, the number of troops that will be moved in will be, both with a 50% probability of being picked, either the full movable army or a random number between the maximal number of troops we can move in and the minimum number. This means that sending on the full army has a probability of more than 50% to be picked.

DeservingArmyValue

The reason for the deserving army approach is simple; we want to keep the agent random, which is why placing all troops on one location was not an option. We also wanted to keep armies together as much as possible to have a higher probability of winning invasions. We wanted to place them preferably on border locations to be able to use them the same turn. There are many other ways to balance the armies between the countries, possibly more random or more concentrated. We chose our method as an agreeable balance between randomness and performance.

Win Probability Calculation

The probability of winning an invasion is calculated using a probability calculator provided by Rudy van Vliet, the first supervisor for this project. The calculator takes the number of troops for the attacking and the defending country, including the unmovable army in the source that cannot be used in battle in order to keep at least one army in the source, and calculates the probability of winning that invasion. The calculator assumes that for every battle, both the attacker and the defender use the maximum possible number of dice, given their numbers of troops.

The probability of winning an invasion is calculated from:

1. all possible outcomes of the dice roll for the first battle of the invasion,
2. the probability of each of these outcomes,
3. the consequences of these outcomes for the number of troops of both the attacker and the defender, and
4. the probability of winning the rest of the invasion with the remaining troops.

Step four makes the calculation recursive. In practice, the calculation is carried out with bottom-up dynamic programming.

Predictions/Traits

It is conceivable that Random will be stronger than a completely random agent, but will still be too weak due to the random troop placement which is so important for Risk. This will cause concentrations of troops to appear far less often than desirable, resulting in lower probability of winning an invasion. The result would be that Random competes in relatively fewer profitable match-ups. While having more of a chance than a totally random agent due to its knowledge of winning probabilities, it will probably be not enough to win from most agents.

4.3 Evaluation

The Evaluation agent awards points for some specific situations, the most important ones being blocking enemy continent bonuses and gaining continent bonuses. The troop placement or attack with the highest score will be executed.

Motivation for Choice

Evaluation was picked to test some common statements about Risk, like the importance of blocking continent bonuses or taking less risk when the opponent has a higher income or more countries. The idea was that Evaluation would make it very easy to quickly change balance between awarding factors, making it easy to test these different statements. In practice, this was too hard to achieve, for different reasons:

- Enemy agents have different weaknesses and strengths on which different balances of the awarding factors would work better.
- The balance between awarding factors could tip easily due to the complexity of Risk, even when using few factors.

Even though this made it hard to achieve its original purpose, Evaluation performed better when some rewarding factors were removed. The result was an agent focused on performance rather than on testing statements about the game. This agent, with a fixed balance of rewarding factors, was kept for the experimenting phase. The used version was an agent mainly focusing on blocking and gaining continent bonuses. Nevertheless, it may still be used to draw some conclusions from its original goal, like examining the effect of blocking a continent bonus.

Phases

In the troop placement phase, every owned country starts with one evaluation point. Then, we award points for the possibility of conquering a continent and for blocking an enemy continent bonus from that country (which will be explained in the paragraph Continent Bonuses). We also subtract one point if the country has no enemy neighbours. We place our troops on the best scoring country, and do this one troop at a time until all troops are placed.

In the attack phase, we evaluate pairs of [owned country, enemy neighbour], where the owned country should contain at least two troops. During the invasion phase, conquered countries with neighbouring enemies are added to the list of pairs. If we find a country that has less than two troops or no enemy neighbours, we remove it from the list.

Every pair starts with one evaluation point. We award points for the possibility of conquering a continent and blocking an enemy continent bonus again. Additionally, we subtract one point for countries if we have less than three troops but our enemy has more than two troops. The attack of the pair [owned country, enemy country] with the highest score will be executed. We keep seeking the best possibility until we have no score of more than three points, which will actually never happen, or our list is empty.

When we have conquered a country, we move armies in the same way the Angry agent does. We place all troops that we can move on the conquered country, unless the attacking country has more neighbours. In that case we leave the troops we do not have to move, in the attacking country.

Continent Bonuses

There are still a few things that need further explanation. The points awarded for blocking a continent and possibly taking a continent are awarded if the owned country being checked is on the cheapest path towards that continent, but in the attack phase the checked enemy country must also be on this path.

For the points awarded for blocking an enemy continent bonus, we check all continents. We award points for every continent that awards an enemy a continent bonus for which the owned country (and during the attack phase [owned country, enemy country]) is on the cheapest path to take that bonus away. The cost of a path will be explained in paragraph pathCost. If the cost of the path $\leq$ the total number of troops to be placed $+$ the usable number of troops in the checked country, we add the continent bonus to the evaluation points, even though we can place the deployable troops only once in a phase. We also add one point if the country concerned is on a cheapest path to block an enemy continent bonus at all, as we rather choose a country with potential to be on a cheapest path than a random country in the case that very few points are awarded during a phase.

For the points awarded for possibly taking a continent, every single continent that is not already owned will once again be checked for the owned country considered (and [owned country, enemy country] during the attack phase). The points are awarded for every continent for which the specified country is on the cheapest path towards it. There are two possible cases we take into account when calculating these evaluation points. The first case is a cheapest path of length one, meaning that the specified country already is in the continent being checked, and the second case is a longer cheapest path, in which case we first have to reach the continent, see Figure 4.1.

```
// EnemyCountry != null: we only need to check for these exceptions when in attack Phase.
// If cheapestRoute.length == 1, we are in the target continent. Check if enemy is also in it.
// If cheapestRoute.length > 1, we check the cheapest path to the target continent, and
// check if enemyCountry is on it.

if(enemyCountry != null){
  if(cheapestRoute.length == 1){
    if(enemyCountry.getContinent() != currentCountry.getContinent()) // Different continents
      continue;
  } else if(cheapestRoute.length > 1) {
    if(enemyCountry != countries[cheapestRoute[1]])
      continue;
```

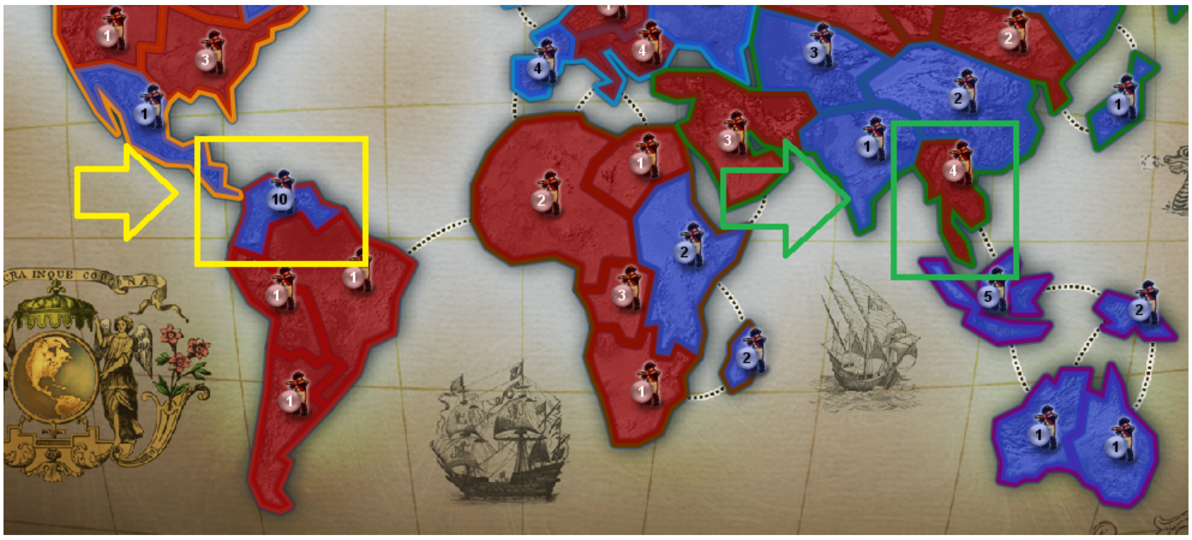

Figure 4.1: A screenshot of a game in Lux Delux. Here we see the two possible cheapestPath to a continent situations. The first one is marked in yellow and starts in Venezuela with the blue agent. Length of cheapestPath to South America is one, as the path only contains Venezuela itself. The second one is marked in green and starts in Siam with the red agent. The length of cheapestPath is two here, the list containing Siam and Indonesia respectively in the path towards Australia as continent. CheapestPath always returns at most one path, even if two paths have the same length.

```
13      }
14    // Make sure currentCountry has enemy neighbors
15    } else if(currentCountry.getHostileAdjoiningCodeList().length == 0){
16      totalEvalPoints = 0;
17      System.out.println("No neighbors for "+currentCountry);
18      break;
19    }
```

There are two cases in which the concerned continent will never lead to evaluation points during the attack phase (as seen in the code when line 6 evaluates to true).

- The first one appears if our country is already in the continent we are trying to take (thus the length op cheapestRoute is one). The specified enemy country must also be in this continent in order to be able to make progress in taking this specific continent.
- The other one appears when the length of our cheapest path is bigger than one. In this case, we want the enemy country concerned to be the first enemy country on the cheapest path to this continent.

If one of these cases applies, we do not have to check this continent for this country or pair of [owned country, enemy country] any more; it will not yield any evaluation points for taking a continent bonus.

Our last case appears when we do this check from the placeArmies phase (line 15).

- In this case, an enemy country is not specified, but we do know that if the checked country has no enemy neighbours it cannot possibly be the cheapest option to conquer an enemy continent from. Continents cannot be taken from this country, which is why we will return zero points.

If we do not find any of these scenarios, we start the calculation of the evaluation points we award for possibly taking a continent from a specified country (and specified enemy if we do this check from the attack phase).

The current implementation only uses the cheapest path to a continent when checking a continent to block or take. At most one cheapest path will be returned when searching for this cheapest path. This means that there will be only one cheapest path per continent that will be considered for awarding evaluation points.

To calculate the evaluation points for a continent for the specified country (and possibly enemy country):

1. We take the number of troops we already possess in a continent, add the number of deployable troops to it, which is zero in the attack phase, and subtract the number of enemy troops.
2. We subtract the number of countries in the continent. This way, we subtract the unmovable army for every friendly country and the army we have to leave in conquered countries for every enemy country.
3. We subtract the path cost for the cheapest route, whatever the length might be. This function also calculates the buffer we want to have to statistically always have the advantage in battles. The buffer we use is two.
4. Note that if the cheapest path has a length of more than one, the cost of the last country in the cheapest path was subtracted twice: once in the pathCost and once when counting enemy troops within the continent. For this reason, we re-add the points we subtracted.
5. Add the continent bonus as a reward for taking a continent.

Because we also count the troops we already have inside the continent, the number of points awarded can become very high. Therefore, we impose to have a maximum on the points we can award. For every continent, the maximum awarded evaluation points is equal to the continent bonus. The score for taking a continent will also be set to zero if it gets beneath zero.

pathCost

Then we have the calculation of the cost of a path, which we deducted from the evaluation points in the function it is called from. If the length of a checked path is only one, it contains an owned country within the continent we are checking. This implies that we are currently considering to take the continent, because blocking a continent bonus is applicable only to continents that are fully owned by the enemy. We return the value two (the buffer) to statistically have the advantage in battles.

In other cases, we do a simple calculation. The first country in the cheapest path is always an owned one, so we subtract its movable troops from our path cost. For every country after the first one, we add one to the cost for every troop we have to leave behind in that country and the number of (enemy) armies it contains for the cost to take that country. Finally, we add the buffer of two points to the cost.

Predictions/Traits

Evaluation is an agent with strong traits. Not placing troops on countries without enemy neighbours helps to keep attacking options open. Also, blocking continent bonuses is an effective way to take income from an opponent to make sure he does not get too strong. However, Evaluation does not take much risk in its path cost calculations, not utilizing all chances but also not overestimating itself. It is less careful with taking risk when it cannot find a good opportunity to get a continent or block a continent bonus.

Taking continents or picking countries when there are no paths valid through our evaluation could be smarter. When Evaluation does not find a clear path towards getting or blocking continent bonuses it ends up in a country with enemy neighbours (if there is one). If there are multiple options with the same evaluation value, it picks the first one it encounters. During tests the agents performed well, but the balance between awarding factors is always delicate. This is especially true with the reward of blocking income, effectively removing income of the enemy, and the reward for taking a continent where the enemy still has a chance for blocking your income when he retakes that continent.

It is conceivable that Evaluation will be strong enough to win from the weaker agents, as the agent is not very strong but has a very powerful trait. Evaluation will also do well against enemies taking continents, as it can stop continent bonuses that would make enemies too strong. However, it will probably not be strong enough to be the best agent.

4.4 Knapsack 2D and 3D

The idea underlying the knapsack agents is that there are different (paths to) enemy countries that we might try to conquer, each with a certain profit and a certain cost, while having a limited budget. The agents use the knapsack theory to pick the most profitable paths for the available budget.

Here, the cost of a path is our estimation of the path cost using the calculations found in the paragraph PathCost above. The profit is the number of enemy countries obtained by successfully taking the path. Thus, we aim to maximize the number of countries we own. When the path we are checking is a path to an owned country, it will get a profit value of -1. Finally, the budget consists of the troops that can be used by the agent: the number of troops to be placed, and, for the 3D version, troops available in starting countries of paths.

Our dynamic programming implementation for the knapsack problem calculates the most profitable solution for a subset of the available countries. Then, it checks if adding another country will result in a more profitable solution. Both versions of the knapsack agents have access to a greedy post-processing phase, which can be performed after the knapsack plan has been executed.

Motivation for Choice

Knapsack agents perform very well in situations with a limited budget and many choices to spend it on. Knapsack agents are definitely not a perfect fit for Risk. In Risk we have many different paths to a destination and path combinations, path branching, continent bonuses, overlap between paths, not to mention risk calculation and a chance-element in battles. We still wanted to see how well we could make a knapsack agent perform under these circumstances.

Phases

In the knapsack agents we implemented, most planning happens in the placeArmies phase of the agent, where we plan in advance which attacks we want to do and how we pass on troops if we have won an invasion. In the agents, we run the knapsack function and end up with a set of chosen paths. After that we run the post-processing phase where we do three things:

1. Check if one of the chosen paths is a subpath of another (and if so, delete the shortest one of the two).
2. Check if two paths have a common subpath somewhere.
3. Calculate how many troops to pass on if an invasion was won.

After that, we place the number of troops we estimated to need at the start of the chosen paths. Then we place the troops we have left from the knapsack together with troops we have left from post-processing. We distribute them over the start countries of the chosen paths, or just put everything on the owned country with the highest number of enemy neighbours if we did not have enough troops to conquer any enemy country. The last thing we do is turn the resulting chosen paths into an attack plan.

After the placeArmies phase we reach the attack phase where we execute the plan we have just created. After executing the plan, the agents have the possibility to use a greedy approach for the troops they have left in their countries. The greedy approach attacks the weakest enemy of each owned country that has at least two troops. This enemy may have at most two troops more than the owned country. Conquered countries are added to the list. If an owned country was not conquered in the greedy stage, it must have at least four armies to be added to the list. This sets a minimum for the number of troops we want our country to have before it is used by the greedy stage.

If a country has been conquered and we have enough troops left, we would in most cases follow the plan we created in the placeArmies phase. There are some exceptions. If we are in the greedy stage, we sent on our full army (except the mandatory one to leave behind). Both in the greedy stage and in the knapsack stage, if the target has no enemy neighbours but our source-country has at least one enemy neighbour left, we do not send any additional troops. If we do not have enough troops to complete our passOn plan, we send the troops we can send. Also note that the minimum number of troops to send on is the number of dice used in a battle, which is especially relevant in this agent. In most cases the passOn plan leaves only two troops, the buffer to statistically have the advantage, in the last country. However, the engine will still have to send on three troops when it takes a country when using three dice. This rule also has to be enforced during the greedy stage.

PathCost

The cost of a path is a rough estimate of how many troops would be needed to take the path. We do not take into account the exact probability of winning an invasion, which in some cases would reduce the number of troops needed. We do this to keep the calculation simple. For every enemy army we encounter in our path, we assume we also lose one. For every enemy country we have to leave behind one army. We also need to leave one army in our starting country. This army is also added to the cost. In particular, if we use a starting country multiple times for different paths, we add this army to the cost of every path. The last factor is that we always want to have the advantage in battle, adding a buffer of two troops. If we are checking an owned country, we simply return a cost of zero.

PassOn

Suppose that our knapsack has resulted in a collection of enemy countries to take and that we have converted those countries to a list of chosen paths. We then get to the post-processing stage. In this stage we also create the passOn plan. The passOn object is an array containing for every country the previous country in the path (or -1 for owned countries and unused countries)f, and the number of troops we want to have in that country according to the plan, see Table 4.1 for an example. Pointing to the previous country will work, as the path is always a cheapest path, meaning that there will not be two different paths to the same country, thus the previous country is always uniquely determined.

As an example, suppose that we own China (with country code 34), that the enemy owns Siam (32), Indonesia (3), Western Australia (2), New Guinea (1) and Eastern Australia (0), and that he has one army in each of these five countries. Also, suppose that our list of chosenPaths contains Indonesia, Western Australia, New Guinea and Eastern Australia in that order.

1. We start by adding the path of Indonesia through Siam from China to our passOn object. We add the buffer of two to the last country, resulting in Indonesia now having the value two in the third row. Note that it is possible that the engine forces us to leave three armies at the end of the path; we ignore this in our calculations. The second row will point to Siam. Then we go to the next country, which is Siam. we add the cost of taking the previous country in this path (Indonesia) and the value of that country to the value of Siam. The result in Siam is $2 + 2 + 0 = 4$. For China we do the same calculation, resulting in the value 6.

2. We now want to add the second path, towards Western Australia, for which the previous country is Indonesia. After adding the buffer to the index of Western Australia, we see that Indonesia is already in some path. In particular, we see that Indonesia is the end of another path and we can simply delete

Indonesia from the chosenPaths list. We call this case 4 in the context of our post-process; the current path is an extension of an earlier path. We then do some calculations to make sure that the new path gets updated with the correct values from the added piece of the path. Important for case 4 is that we have to relocate the buffer from the previous last node to the new last node of the path in order to not count the buffer twice.

3. We now handle New Guinea. Once again we encounter a country we passed before, namely Indonesia. However, Indonesia is not a country in chosenPaths as we have deleted it in the previous step. We call this case 2, a split point or branch of the current path from an earlier path. Case 2 differs from case 4 only in not having to relocate the buffer. The new piece of the path still has to be calculated into the path to the starting country.

4. The last country to add is Eastern Australia, which is again a case 4 situation, as the path to Eastern Australia extends the path to Western Australia.

We did not encounter case 1 and case 3 in this example. They are less complex. Case 3 involves two paths exclusively sharing a starting country. Nothing special happens there, as we can simply add the cost of the new path to the starting node as if we where doing this for a disjoint path. Case 1 is the opposite of case 4, the new path being a subset of an existing path. In this case, we can just remove the new path from the chosenPaths list. The final state of this example is depicted in Figure 4.2 and Figure 4.3.

Country:	0	1	2	3	4	...	31	32	33	34	35	...	41
Previous Country:	2	3	3	32	-1	...	-1	34	-1	-1	-1	...	-1
Troops:	2	2	4	10	0	...	0	12	0	14	0	...	0

Table 4.1: Example of the passOn table. On the first row, Country, we see all 42 countries. On the second row, we see the Previous country in the cheapest path to this country. The third row, troops, stands for the number of troops we would want to pass on to this country. The ... represent countries with the default values -1 for the second row representing having no previous country or an owned country, and 0 for the third row meaning trying to passOn no troops to this country. Countries representing the numbers in the table are: Eastern Australia (0), New Guinea (1), Western Australia (2), Indonesia (3), Argentina (4), India (31), Siam (32), Afghanistan (33), China (34), Ural (35) and Yakutsk (41).

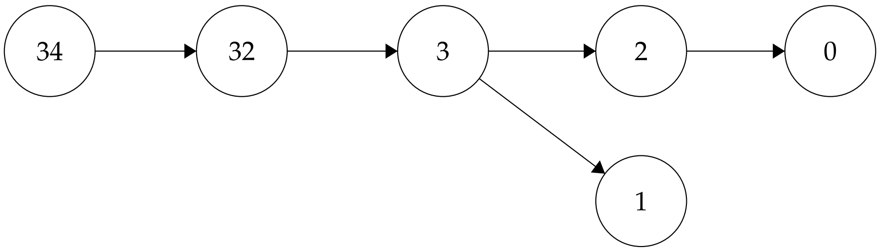

Figure 4.2: Graphical representation of the used countries of the passOn data structure. The countries represented here are China (34), Siam (32), Indonesia (3), Western Australia (2), Eastern Australia (0) and New Guinea (1). For each country, its previous country is displayed on its left.

Attack Plan

The attack plan object consists of two rows and resizes the number of columns to the number of invasions derived from the chosenPaths. For every column, the first row displays the source of an invasion and the

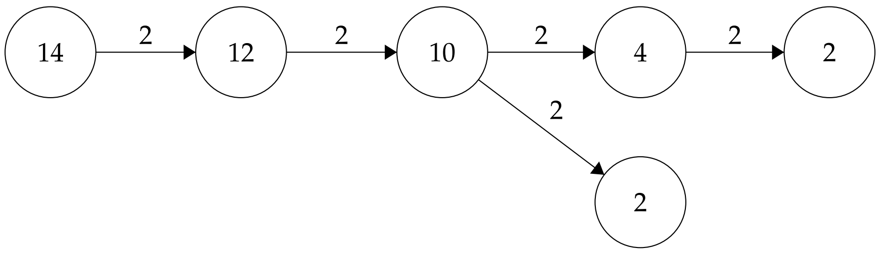

Figure 4.3: Graphical representation of the used values in row two and three of the passOn data structure. For each country, its previous country is displayed on its left. The number in the nodes represents the number of troops we want to pass into that country according to the plan. The number on the arrows represent the troops taking that country will cost, including the troop required for claiming the previous country. The number in the starting node is the number of troops required for this plan.

second row displays the target. It is possible that an invasion appears multiple times in this data structure. This does not matter for the passOn plan, as that plan derives the number of troops to pass on after the unique invasions. The attack phase will just skip the invasions that already took place. The attack plan for the example given earlier can be found in Table 4.2.

China (34)	Siam (32)	Indonesia (3)	China (34)	Siam (32)	Indonesia (3)	Western Australia (2)
Siam (32)	Indonesia (3)	New Guinea (1)	Siam (32)	Indonesia (3)	Western Australia (2)	Eastern Australia (0)

Table 4.2: The attack plan for the example given earlier, in which only New Guinea and Eastern Australia were not removed from chosenPaths. The first row presents the source of the attack, the second row the target. Every column represents an invasion.

Knapsack agents, 2D and 3D

There are some differences between the 2D version and the 3D version of the knapsack agent, but the main difference is in the knapsack itself. The 2D version takes into account our budget in the form of the number of armies available for placing, and the countries we may wish to conquer. The 3D version adds another dimension, keeping track of the number of troops used from the start of the current cheapest path, as those armies can be used only once. This makes the 3D version capable of using the troops that are already present on owned countries. In order to use this third dimension, we must keep track of some extra parameters and do some more pre-processing. The main differences apart from the knapsack itself are that for the 3D version, we sort the countries by starting country (of their cheapest path), and that we only investigate a list of enemy countries instead of skipping owned countries during the calculation. Note that the third dimension has the size of the largest concentration of troops on an owned country. This implies that for starting countries with fewer troops, we do not use the full third dimension. We will not read in the indexes in the table that are not used to solve this. The agents are similar again after the final choices have been converted to a list with the chosen paths, right before we start the post-processing of these choices.

We have two tables, the knapsackTable and the pickedCountryTable. The knapsackTable keeps track of the maximum reward with the given resources, for a given subset of the countries. The resources are a number of deployable troops smaller than or equal to the total number of deployable troops this turn, and in case of the 3D agent, a number of troops smaller or equal than the total number of troops available in the starting country. The pickedCountryTable keeps track of the countries picked for this best score with the available resources, with the value one representing a picked country. One thing to keep in mind for implementation is that row and column 0 represent the case in which there are no countries used and we have no budget available.

Knapsack agents, 2D

In the 2D version of the knapsack function, we iterate over all countries. For every country we iterate over the number of armies available. We also consider index 0, as it is theoretically possible that we can buy a path of cost 0. In practice, however, this will not happen, as an owned path gives us a reward with a value of -1, which is not profitable. Every other path will lead to an enemy country and will both have positive cost and profit.

In every iteration we check if our weight, which is the cost of the path, does not surpass our budget, which is the number of troops we have available for deployment. We also check if we are actually investigating a path to an enemy country. If this is not the case, the current country cannot improve the score that we already have for this number of deployable troops.

If our budget suffices and we check an enemy country, we calculate the potential better score by adding the value of using the current path to the best score for the budget minus the cost of the current path. We compare the total value to the best value using only the earlier countries. If the new value is higher, then the path to the current country is picked in the most valuable collection of paths.

To find the actual best solution for the full set of countries, we go to the index for the last row and column. We will go up rows until we find a 1 in the pickedCountryTable. We move the cost of that country to the left and go up rows until we find a 1 again. We will repeat the process until no more countries can be bought. An example is given in Figure 4.4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0 :	0	0	0	0	0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
1 :	0	0	0	0	0	0	0	0	3	3	4	4	4	4	4	4	4	4	7	7	7	7	7	7	7	7	7	7
2 :	0	0	0	0	0	0	0	0	3	3	4	4	4	4	4	4	6	6	7	7	7	7	7	7	7	7	10	10
3 :	0	0	0	0	0	0	2	2	3	3	4	4	4	4	5	5	6	6	7	7	7	7	8	8	9	9	10	10
4 :	0	0	0	0	0	0	2	2	3	3	4	4	4	4	5	5	6	6	7	7	7	7	8	8	9	9	10	10
...																												
31:	0	0	0	0	0	0	2	2	3	3	4	4	4	4	5	5	6	6	7	7	7	7	8	8	9	9	10	10
32:	0	0	0	0	1	1	2	2	3	3	4	4	4	4	5	5	6	6	7	7	7	7	8	8	9	9	10	10
33:	0	0	0	0	1	1	2	2	3	3	4	4	4	4	5	5	6	6	7	7	7	7	8	8	9	9	10	10
...																												
41:	0	0	0	0	1	1	2	2	3	3	4	4	4	4	5	5	6	6	7	7	7	7	8	8	9	9	10	10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0 :	-	-	-	-	-	-	-	-	-	-	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1 :	-	-	-	-	-	-	-	-	1	1	-	-	-	-	-	-	-	-	1	1	1	1	1	1	1	1	1	1
2 :	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	-	-	-	-	-	-	-	-	1	1
3 :	-	-	-	-	-	-	1	1	-	-	-	-	-	-	1	1	-	-	-	-	-	-	1	1	1	1	-	-
4 :	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
...																												
31:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
32:	-	-	-	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
33:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
...																												
41:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 4.4: Example of a 2D knapsack execution. The dotted line represents repeating results. Rows represent the newly added countries to the solution. Columns represent the numbers of troops available for deployment. The first table is the knapsackTable, displaying the best found reward for the countries from zero to the used row. The second table is the pickedCountryTable, displaying the used countries from the countries from zero to the used row with a 1 for picking the country and a "-" for not picking the country. To find the final solution, we move up in the last column to eventually find a 1 at row 2 column 27 costing 8 troops (the cost is not found in these tables). We now move the cost, 8 troops, to the left (ending in column 19) and go up rows until we find a 1 again. We find it immediately at [1, 19] with cost 8. Repeating the process, we also find country zero at [0, 11]. Our chosenPaths will contain countries 0, 1 and 2, with a total reward of 10 newly obtained countries. This reward may be reduced in the post-processing phase. Note that in both tables, row 0 corresponds to country 0. In our implementation, there is another row before this row, corresponding to no countries, at all.

Knapsack agents, 3D

The 3D knapsack agent is more complicated than the 2D version, but the idea is largely the same. The 3D version is based on a model that was provided by supervisor Rudy van Vliet.

When we consider buying a path, we will first try to spend the troops on the starting country of the path. As mentioned, we iterate over a list of enemy countries sorted on starting country instead of over all countries. We also keep track of the number of troops present in the starting country of the path. As was the case with the number of armies in 2D, it is theoretically possible to pay for a path of cost zero with the troops from a starting country, so we again evaluate this possibility. However, now it is more than theory, as it is very common that we used all of the troops in the starting country, and still want to pay for a path using the deployable troops.

Or that we used all of the deployable troops, and want to pay for a path with the troops in its starting country. To find the best score, we loop over the enemy countries, the deployable troops, and the troops present in the starting country that we currently need. It might seem incorrect to use the number of armies in the starting country instead of the number of *movable* armies in the starting country. Recall, however, that we included the cost of leaving troops in the first country in the path cost calculation.

Once again, we first check if we have enough budget, now consisting of the number of deployable troops plus the troops present in the starting country of the path, to buy the current path at all. We do not have to check if the checked country is owned as the list of countries we check now only contains enemy countries. If we do not have enough budget, we will use the best score without this country.

The third dimension complicates a bit where we jump to in order to find this best score. We have to make sure that we have the right index for that dimension if this is the first occurrence of the current starting country, as the troops that are already present in different starting countries are completely independent. If this is the first time we check this starting country, the index for the third dimension we search for is set to the number of troops in the previous starting country. Otherwise, we use the current index in the third dimension as we might have used troops from the starting country already.

If we do have the budget to buy a path, we check the potentially better score including this path. This happens in three steps:

1. The first one involves calculating the shares of the budget we use to buy the path. In particular, how many deployable troops and how many troops from the starting country do we use. Before we start using deployable troops, we first want to use all troops in the starting country. The reason is that the deployable troops can be used for all paths, whereas the troops in the starting country can be used only for paths from that starting country.

2. In step two, we calculate the score we are going to compare the potentially better one with. The correct index to use for the knapsackTable depends on whether this is the first occurrence of the starting country in the same way as it did in handling insufficient budget to buy the path at all and finding the best score to use there. This time, we also have to calculate the potentially better score based on the first occurrence situation. If this is the first occurrence, we can use the same index for the third dimension in both the potentially better score and the best score until now, as we use totally independent starting country troops, having no overlap. If, on the other hand, this is not the first occurrence of a starting country, we have to account for troops in the third dimension the same way we did with the number of deployable troops in the 2D agent; we look at the best profit we can make after subtracting the cost of adding the current country. In both cases we do this also for the second dimension, which is the number of deployable troops. Also in both cases, we add the value of the new path to the value of the best score for the budget left after buying that path, and compare the result to the value without the path.

3. The last step is a short one: inserting the new best score for the given resources in the knapsackTable and filling in the pickedCountryTable accordingly.

Finding the actual best solution works in a similar way as with the 2D version, but also uses the third dimension for the troops in the starting country of a path. When we consider choosing (a path to) an enemy country, the costs get subtracted from the third dimension before they get subtracted from the number of deployable troops left. Also, the index of the third dimension checked is variable, depending on the number of troops in the starting country of a path.

Predictions/Traits

The knapsack agents are possibly our smartest agents tested. They plan in advance, do path cost calculations, and try to take as many countries as they possibly can. While being smart in the knapsack part, the agent does have a number of weaknesses. The most important one is Knapsack not fitting Risk perfectly, which means we have to post-process the result of the knapsack, loosing performance by doing so. Another important one is while planning ahead to get the best plan of conquest, it is not that aggressive due to not taking much risk in the calculation of the path cost. This is worsened by the agent planning in advance, following plan even if things go wrong or better opportunities arise. Knapsack also ignores continent bonuses, in some cases it does nothing with troops leftover after post-processing the knapsack results and it does not pass on troops in the best way possible.

Even though these agents have a number of weaknesses, Knapsack is able to pick profitable paths and see the bigger picture with a conquest plan instead of picking the best option in a greedy way, which is an advantage. However, it presumably has too many weaknesses to be the best agent.

Chapter 5

Experiment Evaluation

In this chapter we explain the way we conducted match-ups/experiments. After that introduction, we present the used match-ups with the hypothesis and results of those match-ups. In the discussion, we use the results to reflect on the hypotheses.

The experiments can be divided into three groups. Group one focuses on the Knapsack agents, trying to find the better and worse performing ones. The second group compares agents to the bottom-line agent: Random. The third group measures the effectiveness of the other agents against each other and against Knapsack agents.

5.1 Execution Environment of Match-ups

The execution of the experiments happens with the chosen engine, as presented in Chapter 3. Every simulation is started by hand and takes only a few seconds to complete. Afterwards the results are registered, in particular: which agent won, and in which turn did this agent win.

The number of turns is measured as it is very easy to do, and it may say more about each individual simulation than just who won. For example, the number of turns can say something about how much easier it is for one agent to win than for the other. It can say something about how equally matched two agents are. The number of turns can say something about the stability of a match-up. If an agent always wins in approximately the same number of turns, it may indicate that the winning agent can steadily recover from most situations against its opponent, even when having an unfavourable country set.

However, it does not say enough about match-ups to get a clear image of how they went. The big problem is that a high number of turns can also be caused by an agent just performing worse for some situations in the game, for example with finishing the opponent or when having slight advantage. There are just too many factors that are hard to isolate. The overall conclusion is that the number of turns, while interesting to be taken into account, is open for too many different interpretations to actually draw conclusions from. Therefore, we will not discuss it. However, we will keep track of it as it does not take much effort to do so.

For the simulations themselves: we run agents against each other a number of times, following the rules as presented in Chapter 2. Both agents have the starting turn in half of the total number of simulations of a match-up.

Every simulation starts from a randomly generated board set-up. This random start combined with the complex nature of the game could be a pitfall, as it might be hard to distinguish the actual reasons for the effectiveness of an agent from having luck with the starting situation. However, it is more fair than having the agents pick their countries, as we do not want to mark an agent as more effective if it just picks its countries smarter during the set-up stage. We will avoid this pitfall by having enough simulations before actually concluding something and being more careful with statements we make.

A similar issue appears with the distribution of the troops over the countries during the set-up stage. Our initial plan was to let the agents handle the troop placement in the set-up stage as a normal troop placement phase. The reason was that random troop placement may yield starting configurations more favourable for one agent than for the other, introducing another factor that influences the result of a game. On the other hand, if one of the agents has a worse troop placement phase, the agent with the better troop placement phase will always have the advantage in the start situation, regardless of its other phases (whether better or worse). In fact, the question is: what is more important for testing the effectiveness of two agents against each other, the phases after the set-up or the full agent. This resulted in the decision to test both forms of initial troops distribution in the first match-up performed, but that match-up was not so important for the decision. Eventually, we decided to use random troop placement. The reason was that it seemed more fair to let both agents have an equal chance of having a better starting situation. Hence, as an answer to the earlier question: we focus more on the phases after the set-up of the game to see how strong an agent is.

In order to state that an agent is stronger than another one, we require a few things:

1. In a direct confrontation, an agent must be significantly stronger with a confidence interval of 95%. This means that when 40 simulations are executed, the stronger agent has to win more than 65.8% of the time. When 100 simulations are executed, the stronger agent has to win more than 60% of the time.

2. Indirect experiments did not contradict the direct order of strength. Thus, when involving a third agent, a rock-paper-scissors situation is not allowed.

3. When a direct experiment is not conducted, we assume being stronger works transitive, even though transitivity may not always be applicable, especially when one agent has strengths targeting the weaknesses of another agent.

In case the experiments cannot prove the order of strength, we conclude that two agents are possibly equally strong.

5.2 Execution of Match-ups

5.2.1 Order of Strength Knapsack Agents

The first group of experiments were for match-ups of one Knapsack agent against another Knapsack agent, in order to measure the order of strength of the knapsack agents. The match-ups are described in Table 5.1.

Agent 1	Agent 2	Total Number of Simulations	Additional Reason for Experiment, Besides Agent Strength
Knapsack2D	Knapsack2DG	40	Test the effect of greedy stage with Knapsack agent to counter not being able to use troops in starting countries
Knapsack3D	Knapsack3DG	100	Test the effect of greedy stage with Knapsack agent combined with using troops in starting countries
Knapsack2DG	Knapsack3D	100	Compare the effect of using troops the same turn with using them in Knapsack calculations next turn
Knapsack2DG	Knapsack3DG	100	Insight into using a greedy stage
Knapsack2D	Knapsack3D	40	

Table 5.1: Match-up of Knapsack agents, the number of simulations conducted for each match-up, and an additional reason for the match-up. The G in agent names represents the word Greedy.

Hypotheses and Results

I expected the following winners for the five match-ups:

- Knapsack2D against Knapsack2D with Greedy: Knapsack2D with Greedy.
- Knapsack3D against Knapsack3D with Greedy: Knapsack3D with Greedy.
- Knapsack2D with Greedy against Knapsack3D: Knapsack2D with Greedy.
- Knapsack 2D with Greedy against Knapsack 3D with Greedy: Knapsack3D with Greedy.
- Knapsack2D against Knapsack3D: Knapsack3D.

The results can be found in Table 5.2.

Agent 1	Agent 2	Total Number of Simulations	Total Percentage of Wins by Agent 1	Start Turn for Agent 1: Wins by Agent 1	Start Turn for Agent 2: Wins by Agent 1
Knapsack2D	Knapsack2DG	40	0%	0	0
Knapsack3D	Knapsack3DG	100	31%	29	2
Knapsack2DG	Knapsack3D	100	62%	43	19
Knapsack2DG	Knapsack3DG	100	54%	41	13
Knapsack2D	Knapsack3D	40	5%	1	1

Table 5.2: Results from the simulations for the match-ups of two Knapsack agents. Note that in all cases, both agents had the starting turn in half of the simulations. The G in agent names represents the word Greedy.

5.2.2 Order of Strength against Bottom-line Agent

The second group of experiments were for each agent against the Random agent. Every reasonable agent should be able to beat the Random one. The match-ups are described in Table 5.3.

Agent 1	Agent 2	Total Number of Simulations	Additional Reason for Experiment, Besides Agent Strength
Random	Evaluation	40x2	Test the effect of random vs. chosen troop distribution in set-up stage
Random	Angry	40	
Random	Knapsack2D	40	
Random	Knapsack3D	40	
Random	Knapsack2DG	100	
Random	Knapsack3DG	40	

Table 5.3: Match-ups of the random agent against all other agents, the number of simulations conducted for the match-up, and an additional reason for the match-up. For Evaluation, we run the simulations once for random troop distribution and once for chosen troop distribution during the set-up stage. The G in agent names represents the word Greedy.

Hypotheses and Results

I expected the following winners for the six match-ups:

- Random against Evaluation: Evaluation.
- Random against Angry: Angry.
- Random against Knapsack2D: Random.
- Random against Knapsack3D: Knapsack3D.
- Random against Knapsack2D with Greedy: Knapsack2D with Greedy.
- Random against Knapsack3D with Greedy: Knapsack3D with Greedy.

The results can be found in Table 5.4.

Agent 1	Agent 2	Total Number of Simulations	Total Percentage of Wins by Agent 1	Start Turn for Agent 1: Wins by Agent 1	Start Turn for Agent 2: Wins by Agent 1
Random	Evaluation	40*	22.5%	8	1
Random	Evaluation	40**	12.5%	3	2
Random	Angry	40	15%	5	1
Random	Knapsack2D	40	92.5%	19	18
Random	Knapsack3D	40	50%	17	3
Random	Knapsack2DG	100	28%	27	1
Random	Knapsack3DG	40	25%	8	2

Table 5.4: Results from the simulations for the match-ups of the random agent against all other agents. Note that in all cases, both agents had the starting turn in half of the simulations. For Evaluation, two different environments were used. The first one having a random troop distribution during the set-up stage (*) as used with every experiment, the other one using the chosen troop distribution during the set-up stage (**), meaning the set-up stage used in the regular troop placement phase. The G in agent names represents the word Greedy.

5.2.3 Order of Strength Strongest Agents

The third group of experiments compare the possibly strongest agents. These experiment should result in a final order of effectiveness of agents. Table 5.5 presents the match-ups for these experiment.

Agent 1	Agent 2	Total Number of Simulations
Angry	Evaluation	100
Angry	Knapsack2DG	100
Angry	Knapsack3DG	100
Evaluation	Knapsack2DG	100
Evaluation	Knapsack3DG	100

Table 5.5: Match-up of two agents with the number of simulations conducted for the match-up. The G in agent names represents the word Greedy.

Hypotheses and Results

I expected the following winners for the five match-ups:

- Evaluation against Angry: Angry.
- Angry against Knapsack2D with Greedy: Angry.
- Angry against Knapsack3D with Greedy: Angry.
- Evaluation against Knapsack2D with Greedy: Knapsack2D with Greedy.
- Evaluation against Knapsack3D with Greedy: Knapsack3D with Greedy.

The results can be found in Table 5.6.

Agent 1	Agent 2	Total Number of Simulations	Total Percentage of Wins by Agent 1	Start Turn for Agent 1: Wins by Agent 1	Start Turn for Agent 2: Wins by Agent 1
Angry	Evaluation	100	71%	39	32
Angry	Knapsack2DG	100	68%	45	23
Angry	Knapsack3DG	100	75%	49	26
Evaluation	Knapsack2DG	100	71%	46	25
Evaluation	Knapsack3DG	100	67%	43	24

Table 5.6: Results from the simulations for the match-ups of two agents. Note that in all cases, both agents had the starting turn in half of the simulations. The G in agent names represents the word Greedy.

5.3 Discussion of Results

Here we reflect on the correctness of hypotheses we have set. It seems impossible to say with certainty why some experiments gave unexpected outcomes. We will, however, speculate on possible reasons.

Agent 1	Agent 2	Total Number of Simulations	Total Percentage of Wins by Agent 1	Start Turn for Agent 1: Wins by Agent 1	Start Turn for Agent 2: Wins by Agent 1
Angry	Evaluation	100	71%	39	32
Angry	Knapsack2DG	100	68%	45	23
Angry	Knapsack3DG	100	75%	49	26
Evaluation	Knapsack2DG	100	71%	46	25
Evaluation	Knapsack3DG	100	67%	43	24

Table 5.7: Results from the simulations for the match-ups of two agents. Note that in all cases, both agents had the starting turn in half of the simulations. The G in agent names represents the word Greedy.

Agent 1	Agent 2	Total Percentage of Wins by Agent 1	Hypothesis Winner	Actual Winner
Knapsack2D	Knapsack2DG	0%	Knapsack2DG	Knapsack2DG
Knapsack3D	Knapsack3DG	31%	Knapsack3DG	Knapsack3DG
Knapsack2DG	Knapsack3D	62%	Knapsack2DG	Knapsack2DG
Knapsack2DG	Knapsack3DG	54%	Knapsack3DG	Knapsack2DG
Knapsack2D	Knapsack3D	5%	Knapsack3D	Knapsack3D
Random	Evaluation	22.5%	Evaluation	Evaluation
Random	Angry	15%	Angry	Angry
Random	Knapsack2D	92.5%	Random	Random
Random	Knapsack3D	50%	Knapsack3D	Tie
Random	Knapsack2DG	28%	Knapsack2DG	Knapsack2DG
Random	Knapsack3DG	25%	Knapsack3DG	Knapsack3DG
Angry	Evaluation	71%	Angry	Angry
Angry	Knapsack2DG	68%	Angry	Angry
Angry	Knapsack3DG	75%	Angry	Angry
Evaluation	Knapsack2DG	71%	Knapsack2DG	Evaluation
Evaluation	Knapsack3DG	67%	Knapsack3DG	Evaluation

Table 5.8: Summary of the outcomes of the experiments: the match-up of two agents, the expected winner for the match-up, and the actual result. Green indicated a correct prediction, orange means an inconclusive result, and red means a wrong prediction. A result is inconclusive if the percentage of wins is close to 50 percent, following the rules described in Section 5.1. For Evaluation, two different environments were used. The first one having a random troop distribution during the set-up stage, which was the default for all experiments, is used here. The G in agent names represents the word Greedy.

Knapsack2D against Knapsack2D with Greedy: Presumably, the hypothesis was true because of two reasons: First of all, the knapsack phase is exactly the same for both agents. Therefore everything Knapsack2D does, Knapsack2DG(reedy) would do the same in the same situation. Thus, their knapsack phases are equally strong. The second reason is Knapsack2D not using troops in starting countries. The number of troops available to attack with each turn is rather limited, resulting in small battle plans, whereas Knapsack2DG has an additional greedy stage to use those troops. This makes Knapsack2DG able to perform more battles, which gives it a huge advantage.

Knapsack3D against Knapsack3D with Greedy: both versions can use troops leftover after the knapsack phase: the greedy version can use them the same turn and the not greedy version can use them in its knapsack the next turn. The greedy phase has the advantage that conquering more countries in the same turn leads to more income while also weakening the opponent. Also, the greedy stage takes more risk, resulting in possibly conquering more countries than using the knapsack itself the next turn. This is probably why Knapsack3D, waiting for the start of the next turn to do a better planned attack, was for that reason not strong enough to defeat Knapsack3D with greedy.

Knapsack2D with Greedy against Knapsack3D: As with the previous case, it is likely that the greedy stage taking more risk and using income the same turn gives a bigger advantage than being able to use troops in a smarter way the next turn.

Knapsack2D with Greedy against Knapsack3D with Greedy: Presumably, the hypothesis was not true because even though Knapsack3D can use starting country troops, which is, in principle, a huge advantage, the greedy phase will have used most of these troops in the previous turn already. This would explain why none of the agents could be declared significantly stronger. However, the percentage of wins suggest that Knapsack2D with Greedy is stronger. Perhaps, the Greedy approach is stronger than the Knapsack part. This would mean that troops in starting countries used by the 3D version could have been used better by the Greedy phase. The fact that Knapsack3D can use troops left in starting countries would only weaken the agent in that case.

Knapsack2D against Knapsack3D: Both agents work the same. However, Knapsack3D has the advantage, as it is able to use troops from owned countries in its calculations.

Random against Evaluation: While Evaluation is not very strong, it will recognize when Random accidentally takes a continent. It will immediately try to invade the continent to avoid a continent bonus. An enemy capable of forming a plan to take and block continent bonuses will be too much for Random's weaknesses of semi-random troop placement.

Random against Evaluation: There was also a hypothesis for the simulations of the two different types of troops distribution in the set-up phase. The hypothesis was that the weaker agent would win more with the random troop distribution than with chosen troop distribution. The reason is that the weaker agent may also have a weaker troop placement phase, thus increasing the difference between the two agents when using chosen troop distribution instead of random troop distribution. This hypothesis is weakly confirmed by the experiments (see Table 5.4).

Random against Angry: The possibly best performing agent (Angry) should have no trouble defeating the possibly worst agent. The balanced aggressiveness of Angry was probably too much for an agent trying to win through accidentally making good choices.

Random against Knapsack2D: Due to the limited troops to execute its plan and taking few risks with path cost calculations, the Knapsack2D calculations will result in few and small paths, not being able to compete against most agents, not even having a chance against the Random agent.

Random against Knapsack3D: We expected that taking few risks with path cost calculations, Knapsack3D calculations would result in few and small paths, but by using troops left in starting countries, they would have been (long) enough to win from the Random agent. In our simulations, however, both agents won half of the times. We thus have no evidence for either of the two agents to be stronger. What I assume as reason for this result is that after all, the paths chosen by the knapsack are still so short and too few that it is hard for Knapsack3D to beat the restrained-aggressive Random agent.

Random against Knapsack2DG and Random against Knapsack3DG: Using transitivity, the Knapsack agents with greedy stages should be better than Random, as Knapsack3D was equally good as Random. This does assume that Random is not particularly strong against the greedy approach used.

Evaluation against Angry: Even though Evaluation might work well against some types of agents due to its focus on continents, its weak side is its lack in selecting smart attacks when continent bonuses are not to be gained or blocked. This is especially relevant, since Angry never concerns himself with taking continents; it grabs continents accidentally.

Angry against Knapsack2D with Greedy and Angry against Knapsack3D with Greedy: As it appears from earlier experiments that the knapsack algorithm had worse results than expected in advance, and the greedy part of Angry seems stronger than the greedy part of the Knapsack agents, it should not be surprising that Angry could win from Knapsack2DG and Knapsack3DG.

Evaluation against Knapsack2D with Greedy and Evaluation against Knapsack3D with Greedy: I expected that the Knapsack agents would be too strong and smart to lose from Evaluation. However, even though the Knapsack greedy stage is very aggressive, Evaluation can counter that, to some extend, with its own aggression when taking or blocking continent bonuses is not applicable. In addition to this aggressive approach, Evaluation also focuses on continents when possible, which have a high impact on income. Probably, the actions towards continent bonuses and (limited) evaluations of good attacks were enough for Evaluation to beat the best knapsack agent with its somewhat small battle plan and leftover attacks with the greedy stage, whether that best Knapsack agent is Knapsack2D with Greedy or Knapsack3D with Greedy. Presumably, I underestimated the strength of Evaluation in case no attempt would be made to take continents or block continent bonuses.

Chapter 6

Conclusions

In this chapter, we will summarize the results from the match-ups and discuss and reflect on the results, but now from the view of the order of strength. This will lead to our discussion on the major research topic of this project, which was the effectiveness of the agents for two-player Risk-games without neutral factions. We also use the results to briefly discuss some specific properties of Risk agents: the aggressiveness, creating plans in advance and looking ahead and greedy agents. We will end with a final summary/conclusion and give suggestions for future work.

6.1 Agent Performance

We combine the results for the three experiment categories into an order of strength according to these experiments.

Knapsack

From the experiments, we could conclude that Knapsack2D is weaker than Knapsack3D, and Knapsack3D is weaker than Knapsack2DG and Knapsack3DG. We assume that Knapsack3DG is stronger than Knapsack2D by using transitivity. The result between Knapsack3DG and Knapsack2DG was inconclusive. The combined order of strength can be seen in Figure 6.1.

Random

The results from the experiments give us the following order of strength: Knapsack2D < Random ? Knapsack3D < Knapsack2DG ? Knapsack3DG ? Angry ? Evaluation as seen in Figure 6.2. The results from Random against Knapsack3D suggest that Knapsack3D is weaker than Angry, Evaluation, Knapsack3DG and Knapsack2DG.

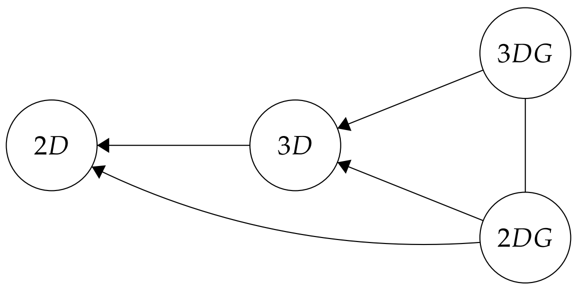

Figure 6.1: Graphical representation of the order of strength from right (strongest) to left (weakest). Arrows point from the stronger to the weaker agent. An arrow presents a direct confrontation. When no direct line is present we assume transitivity. A line between two nodes indicates a direct match-up with inconclusive result. 2D stands for Knapsack2D and 3D stands for Knapsack3D; the G is added when using a greedy stage.

Indeed, in the Knapsack experiments, we already concluded that Knapsack3D is weaker than Knapsack3DG and Knapsack2DG.

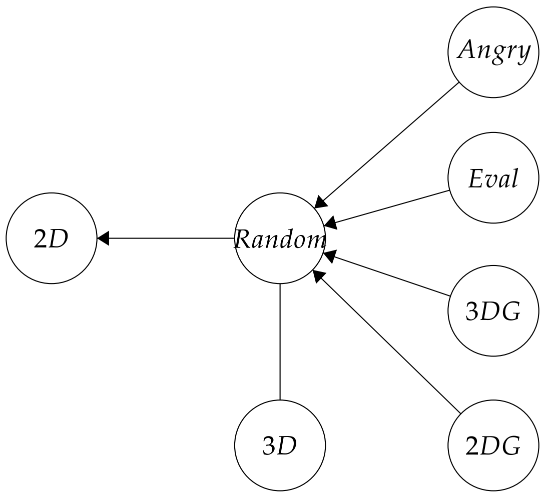

Figure 6.2: Graphical representation of the order of strength from right (strongest) to left (weakest). Arrows point from the stronger to the weaker agent. An arrow presents a direct confrontation. When no direct line is present we assume transitivity. A line between two nodes indicates a direct match-up with inconclusive result. 2D stands for Knapsack2D and 3D stands for Knapsack3D; the G is added when using a greedy stage. Eval stands for Evaluation.

Possibly Strongest Agents

We draw two conclusions from the experiments for the possibly strongest agents:

1. Angry would defeat its opponent in all of its match-ups. This implies that Knapsack2DG ? Knapsack3DG ? Evaluation $<$ Angry. To conclude that Angry is also better than the other agents, we should show that the weakest agents of Evaluation, Knapsack2DG and Knapsack3DG are stronger than those other agents (and assume transitivity).

2. Evaluation would be the second best agent according to the results. Again, this holds if the weakest of the agents Knapsack2DG and Knapsack3DG is stronger than those other agents (and we assume transitivity). Knapsack2DG ? Knapsack3DG $<$ Evaluation $<$ Angry.

These results can be seen in Figure 6.3.

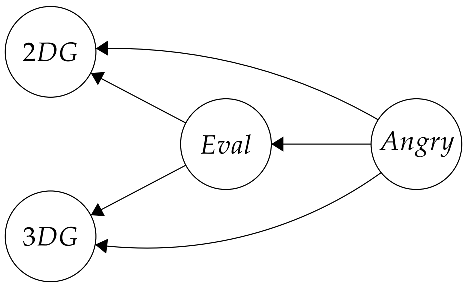

Figure 6.3: Graphical representation of the order of strength from right (strongest) to left (weakest). Arrows point from the stronger to the weaker agent. An arrow presents a direct confrontation. 2D stands for Knapsack2D and 3D stands for Knapsack3D; the G is added when using a greedy stage. Eval stands for Evaluation.

Combination of Experiment Categories

To obtain an order of strength over all agents used, the separate 'partial' orders of strength can be combined. The result is graphically depicted in Figure 6.4. The resulting order of strength includes the inconclusive results. This means that with the pairs Knapsack3D - Random and Knapsack3DG - Knapsack2DG, it is possible that one agent is better than the other, but our experiments did not establish that. We therefore assume that these pairs are of similar strength levels. Recall that it is possible that against unused agents, a lower strength agent would perform better than a higher strength agent because of exploiting weaknesses. Figure 6.4, however, shows the order of strength for the agents we considered in this project.

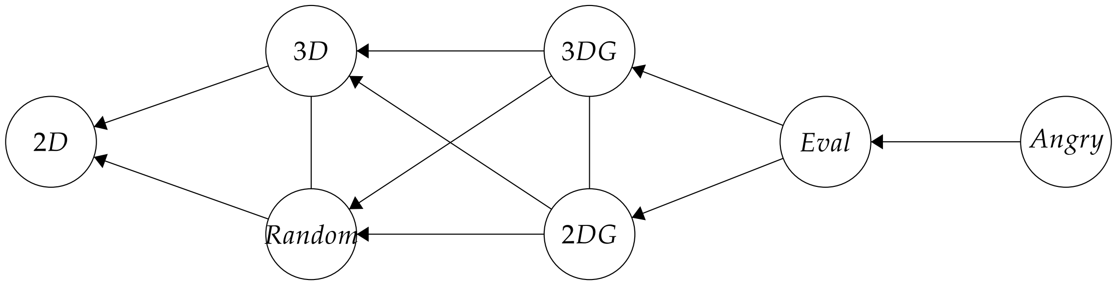

Figure 6.4: Graphical representation of the order of strength from right (strongest) to left (weakest). Arrows point from the stronger to the weaker agent. An arrow presents a direct confrontation. When no direct line is present we assume transitivity. A line between two nodes indicates a direct match-up with inconclusive result. 2D stands for Knapsack2D and 3D stands for Knapsack3D; the G is added when using a greedy stage. Eval stands for Evaluation.

6.2 Specific Properties of Agents

We can use the results from our experiments to discuss three specific properties that agents may or may not have. The properties and corresponding agents are:

Planning in Advance: Knapsack agents.
Greedy: Evaluation, Angry for picking the neighbour to attack and the Greedy stage of Knapsack.
Aggressive: Angry and Evaluation. Not Aggressive: Knapsack agents without Greedy.

The results indicate that planning in advance is inferior to greediness. In our case, the greedy agents are also the more aggressive agents. However, it is very hard exact impact of these properties on the effectiveness of the agents. Therefore, we do not make too firm statements about the observed correlation between these properties and the strength of an agent.

6.3 Final Conclusion

For this project we tried to find an order of strength for some agents for two-player Risk with neutral factions. The results cannot be used for multiple-player Risk, as other aspects play a role in that game.

We expected the order of strength to be Knapsack2D without greedy $<$ Random $<$ Evaluation $<$ Knapsack3D without greedy $<$ Knapsack2D with greedy $\leq$ Knapsack3D with greedy $<$ Angry. Most relations in this expected order of strength were confirmed by our experiments. However, the experiments did not provide evidence for the hypotheses that Knapsack3D is stronger than Random and that Knapsack3DG is stronger than Knapsack2DG. Instead, they indicated that Knapsack3D and Random are equally strong, and that Knapsack3DG and Knapsack2DG are equally strong. Moreover, Evaluation turned out to be the second best agent. This resulted in the order of strength shown in Figure 6.4: Knapsack2D without greedy $<$ Random $\approx$ Knapsack3D without greedy $<$ Knapsack2D with greedy $\approx$ Knapsack3D with greedy $<$ Evaluation $<$ Angry.

The results suggest that aggressive and greedy play styles should be preferred when playing against some weaker and simpler agents in two-player match-ups without neutral factions. We must, however, be careful not to make too firm statements on this, as this was not the focus of the project. The focus was the order of strength, and the discussion and results on play styles was merely an observation for the current agent set.

6.4 Future Work

In this project, we came across a lot of possible improvements for agents, found some potentially useful metrics to use, and made some observations.

It would definitely be worthwhile to dive further into our agents. Improving agents is an infinite process. There are always improvements possible for the agents used, as Risk is a game about balance between different complex factors that influence the effectiveness of a choice. This is certainly true for an evaluation agent, but still, it would be interesting to see the effect of adding new awarding factors in this agent while creating a good balance between these factors. It is also true for other ways of handling leftover troops in the Knapsack agents, for example by using a knapsack excluding specific country combinations to prevent overlapping paths, or by creating a better back-up plan when things are not going according to the passOn table, for example by adding a function to decide the number of troops to send on.

The number of attacks (within a turn) could be an interesting metric to say something about aggressiveness, but it may be hard to convert this into actual useful data. During the experiments, some notable things happened with the number of turns it would take to win a game. They lead to an idea for yet another metric to decide the strength of an agent: point of return. When a weaker agent has a more favourable country set, how many countries or income should a stronger agent have to still be able turn the game into a win. At some point, when a weaker agent has gained a big income advantage, it may become extremely difficult for a stronger agent to recover. It can be very interesting to examine how this metric would work out.

Last but not least, it would be a great challenge to generalize the agents and experiments to multiple-player Risk.

References

[Ber11] N. Berry. Risk analysis. http://datagenetics.com/blog/november22011/, 2011.

[Mam03] Y. Mamyrin. Yura's domination. http://domination.sourceforge.net/, 2003.

[Ora] Oracle. Oracle. http://www.oracle.com/technetwork/java/index.html.

[Pro99] The Apache Ant Project. Apache ant. https://ant.apache.org/, 1999.

[Sil02a] Sillysoft. Lux delux download. http://sillysoft.net/lux/, 2002.

[Sil02b] Sillysoft. Lux delux sdk. https://sillysoft.net/sdk/, 2002.

[Sil02c] Sillysoft. Lux delux writing your own ai. http://sillysoft.net/wiki/?WritingYourOwnAI, 2002.

[Ste15] Steam. Lux delux on steam. https://store.steampowered.com/app/341950/Lux_Delux/, 2015.

[Wol05] M. Wolf. An intelligent artificial player for the game of risk. Master's thesis, TU Darmstadt, Knowledge Engineering Group, Darmstadt, Germany, 2005. http://www.ke.tu-darmstadt.de/bibtex/publications/show/1302.

Appendix A

After the class files are created they have to be placed in the folder from where new agents are loaded into the engine. As this will be done on a regular basis, it pays to automate the process. The following bat scripts can help to do so.

Take in mind that the scripts are only examples. It all depends on how the hard drive-folders on your computer are set up and on your operating system. We will describe how to do this for Microsoft Windows.

From the location where you keep your bat scripts, move to the sillysoftSDK folder. Call ant there; it will see the build.xml file located in here and build class files from the java files.

```
cd ...\SillysoftSDK
call ant
```

We now want to move the class file to the folder from where the engine loads its agents, and start the game to actually load the agent into the engine. Because the engine loads all agents at start-up, the engine must be restarted to load an edited bot. We use random as the example agent. The dots stand for the common path to this location.

```
copy "...\SillysoftSDK\build\com\sillysoft\lux\agent\random.class"
"...\Steam\steamapps\common\Lux Delux\Support\Agents"

cd Risk
start "" "...\Steam\steamapps\common\Lux Delux\Lux Delux.exe"
```

As mentioned earlier, it matters how your hard drive was set-up. It also matters whether the game licence was obtained from steam or through Sillysoft's (owners of Lux Delux) own official website, as the location of the agents-folder is reached in a different way.

Now, it is also handy to have quick access to the log-file. With the last script, the log-file can be moved to another location.

```
copy "...\Steam\steamapps\common\Lux Delux\Support\log.txt" "...\SillysoftSDK"
```

Appendix B

For readers trying to read some of the tables that use country codes or for people that wish to experiment with Risk agents themselves, it may be useful to know the country codes and the continent codes used in the engine:

Continent Code	Continent Name
0	Australia
1	South-America
2	Africa
3	North-America
4	Europe
5	Asia

Country Code	Country Name
0	Eastern Australia
1	New Guinea
2	Western Australia
3	Indonesia
4	Argentina
5	Brazil
6	Peru
7	Venezuela
8	South Africa
9	Congo
10	East Africa
11	Madagascar
12	North Africa
13	Egypt
14	Mexico
15	Western United States
16	Eastern United States
17	Western Canada
18	Ontario
19	Quebec
20	Alaska
21	Northwest Territory
22	Greenland - Nunavut
23	Iceland
24	Scandinavia
25	Great Britain
26	Western Europe
27	Northern Europe
28	Southern Europe
29	Ukraine
30	Middle East
31	India
32	Siam
33	Afghanistan
34	China
35	Ural
36	Siberia
37	Mongolia
38	Japan
39	Irkutsk
40	Kamchatka
41	Yakutsk