UPPSALA
UNIVERSITET

UPTEC F 20063
Examensarbete 30 hp
December 2020

Zero-Knowledge Agent Trained for the Game of Risk

Simon Bethdavid

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Zero-Knowledge Agent Trained for the Game of Risk

Simon Bethdavid

Recent developments in deep reinforcement learning applied to abstract strategy games such as Go, chess and Hex have sparked an interest within military planning. This Master thesis explores if it is possible to implement an algorithm similar to Expert Iteration and AlphaZero to wargames. The studied wargame is Risk, which is a turn-based multiplayer game played on a simplified political map of the world. The algorithms consist of an expert, in the form of a Monte Carlo tree search algorithm, and an apprentice, implemented through a neural network. The neural network is trained by imitation learning, trained to mimic expert decisions generated from self-play reinforcement learning. The apprentice is then used as heuristics in forthcoming tree searches. The results demonstrated that a Monte Carlo tree search algorithm could, to some degree, be employed on a strategy game as Risk, dominating a random playing agent. The neural network, fed with a state representation in the form of a vector, had difficulty in learning expert decisions and could not beat a random playing agent. This led to a halt in the expert/apprentice learning process. However, possible solutions are provided as future work.

Handledare: Mika Cohen & Farzad Kamrani
Ämnesgranskare: André Teixeira
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F 20063

Populärvetenskaplig Sammanfattning

Ett felsteg i en militär operation kan medföra enorma konsekvenser. Under de senaste seklerna har militära taktiker och strategier utvecklats med hjälp av krigsspel, en form av strategispel. Spelen är utformade för att efterlikna världsliga händelser. Genom att simulera olika krigshandlingar kan man utvärdera varierande krigsföring och stridförfarande i helhet och detalj. Syftet är att hitta svagheter i operationen men även förstå konfliktens omständigheter.

I en värld där maskininlärning blomstrar och med inspiration från DeepMinds utveckling av AI, AlphaGo vidare AlphaZero, som uppnådde övermänskliga prestationer i spelen Go, schack och Shogi, samt utvecklingen av Expert Iteration till spelet Hex, uppstod en naturlig fråga i militär planering; går det att utveckla en agent till krigsspel med liknande algoritmer? En agent som automatiserar analysen i krigsspel kan rekommendera taktiska åtgärder som den mänskliga spelaren kan ha missat. Detta är vad forskningsprojektet vid Totalförvarets forskningsinstitut avser att undersöka. Den här uppsatsen är begränsad till ett krigsspel, Risk. För forskningsprojektet är spelet Risk ett förstasteg mot mer realistiska krigsspel.

Algoritmen i uppsatsen efterliknar AlphaZero och Expert Iteration. I grunden är dessa förstärkningsinlärning och använder Monte Carlo trädsökning som vägleds av ett neuronnät. Neuronnätet tränas för att imitera hur trädsökningen tar beslut, för att sedan vägleda trädsökningen i framtiden. Uppsatsen testar därmed om en Monte Carlo trädsökning kan anpassas till att spela Risk, samt ifall en algoritm som liknar AlphaZero/Expert Iteration kan anpassas till att spela Risk. Det visar sig att en Monte Carlo trädsökning, kan till viss del, anpassas för att spela Risk, men neuronnätet har problem att lära sig trädsökningens beslut och därmed sätter stop för träningsprocessen men det finns eventuella lösningar.

Contents

List of Acronyms

FOI Swedish Defence Research Agency

ML Machine Learning

SL Supervised Learning

RL Reinforcement Learning

IL Imitation Learning

TD Temporal Difference

MCTS Monte Carlo Tree Search

NN Neural Network

CNN Convolutional Neural Network

GNN Graph Neural Network

ReLU Rectified Linear Unit

MAE Mean Absolute Error

MSE Mean Squared Error

EXIT Expert Iteration

MDP Markov Decision Process

UCT Upper Confidence bounds applied to Trees

PUCT Polynomial Upper Confidence bounds applied to Trees

Chapter 1

Introduction

For the past couple of centuries, military tactics and strategies have been developed and evaluated with the help of kriegspiel (wargames). The spark that ignited the usage of wargames was when Prussia defeated France in the Franco-Prussian War [1]. In contrast to popular abstract strategy games, e.g., chess and Go, which have no connection to reality or current political situation, wargames try to simulate historical or current warfare in the form of a strategy game. Wargames are used within military planning to evaluate tactics or strategies in detail, with the purpose to identify vulnerabilities within a military plan and understand the circumstances of the conflict.

Fast forward to the end of the 20th century and the rise of computers, which introduced AI/agents to strategy games. Ever since Deep Blue [2] in 1997 achieved superhuman performance and beat the grandmaster Kasparov in chess, searching for an agent to achieve similar performance in the game of Go has not been the most straightforward task. The complexity of the game encapsulates it all. Chess has approximately 10^{123} [3] possible move sequences while Go has approximately 10^{360} [3], to compare it to a relatively comprehensible number, there are approximately 10^{80} particles in the observable universe. Superhuman performance in the game of Go was achieved by Silver et al. [4] at DeepMind when their agent, AlphaGo, beat a former World Champion Lee Sedol $4-1$ [5]. DeepMind continued their development and generalised AlphaGo, leading to AlphaZero, which achieved superhuman performance in other strategy games (chess and Shogi) [6]. Following the success of AlphaZero, a natural question within military planning arose: can similar algorithms be applied to wargames? By automating the analysis of wargaming, recommendations for strategies and tactics that human players might have overlooked, could be provided.

This thesis is done in collaboration with the Swedish Defence Research Agency (FOI). FOI conducts research in defence, safety and security for the whole society. As the Swedish Armed Forces is a client to FOI, testing if it is possible to automate the analysis of wargaming by developing an AI is a subject of interest.

1.1 Background

1.1.1 AlphaGo

AlphaGo was the first AI to defeat professional players in the strategy game Go. Silver et al. accomplished this by combining supervised learning (SL) and reinforcement learning (RL) in a Monte Carlo tree search (MCTS) algorithm. The training procedure for AlphaGo was split into two parts, first, conduct training from 29.4 million moves, 160000 games played by professional human players, secondly, through self-play RL over several months [7].

1.1.2 AlphaGo Zero & Expert Iteration

The development of AlphaGo did not stop; the agent was a success but relied on the existence of large datasets. This had some drawbacks, to name a few, large datasets of expert played moves were expensive and limited the performance of the agent [7]. Hence Silver et al. developed a new agent, AlphaGo Zero, which only trained through self-play RL, no human knowledge. After 72 hours of self-play, AlphaGo Zero was evaluated against AlphaGo, and defeated AlphaGo $(100 - 0)$. Silver et al. later generalised AlphaGo Zero [6], named it AlphaZero and tested it on the games, chess, Shogi and Go. AlphaZero had no additional domain knowledge except the rules of the game, i.e., zero-knowledge, showing that superhuman performance can be achieved through a general RL algorithm. The algorithm of AlphaGo Zero and AlphaZero utilised MCTS to progress the game. The game data was then fed into a neural network (NN) for generalisation, and the NN was in return used as heuristics in future tree searches.

The same year (2017) as Silver et al. developed AlphaGo Zero; a similar algorithm developed independently by Anthony et al. [8] called Expert Iteration (EXIT) was published. The algorithm was applied to the game HEX, achieving state of the art performance. The details of EXIT and AlphaZero are further discussed in Section 3.6.

1.2 Thesis Purpose

With the success of EXIT and AlphaZero this thesis will act as a first step for FOI in the analysis of how well the decision making in wargaming can be automated by the help of zero-knowledge trained agents. The thesis has been limited to one specific game, Risk. Risk is not the ultimate objective for the research project at FOI, but a first step towards more realistic wargames. Risk was chosen because it is widely known, has some prestige within the wargaming community, and is a game where algorithms as EXIT and AlphaZero are yet to be tested. This thesis will hence investigate the feasibility of how well a similar algorithm to EXIT/AlphaZero can (be adapted to) play Risk. More specifically, since both EXIT and AlphaZero are composed of an MCTS (expert) combined with a NN (apprentice), this thesis will evaluate:

- How well an MCTS can (be adapted to) play Risk?
- How well the expert combined with the apprentice can (be adapted to) play Risk?

1.2.1 Related work

For a popular game such as Risk, there is surprisingly only a small amount of scientific publications of agents trained for the game, and none regarding zero-knowledge trained agents. Most agents created for Risk use handcrafted, human knowledge to choose optimal moves.

In *An Intelligent Artificial Player for the Game of Risk* [9], Wolf introduced a *basic* agent. The agent made its decision based on the move that yielded the game state with the highest expected reward. Specifically, the agent performed a one-step lookahead, simulated all legal actions, compared their resulting states with a handcrafted evaluation function, and chose accordingly. Only performing a one-step lookahead resulted in significant disadvantages. There were, for example, no connections between placing armies and conquering territories/continents. Wolf hence introduced an *enhanced* agent which, in addition to the basic agent, used a handcrafted function that tried to evaluate battles for conquering continents. Wolf later applied temporal difference (TD) learning [10] on the enhanced agent. The goal was to modify the evaluation function such that it converged to the function that returned the actual probability of winning the game from a given state. Wolf's findings showed that the enhanced player significantly surpassed the basic player, but the TD learned player only increased the enhanced player rating by an average 20%.

In *An Automated Technique for Drafting Territories in the Board Game Risk* [11], Gibson et al. presented a technique for the initial draft phase of Risk. They used MCTS combined with an evaluation function and showed that their drafting technique improved the performance against the most vigorous opponents in the clone version of Risk, Lux Deluxe.

Glennn Moy and Slava Shekh [12], investigated if AlphaZero could be applied to the wargame Coral Sea. They encountered many challenges such as, problem representation and hardware limitations. By combining heuristic knowledge with the AlphaZero algorithm, they were able to train a model which outperformed the heuristics used to train it.

Since there is no EXIT or AlphaZero agent implemented for the entire game (Risk), together with their success in other strategy games, an investigation of how well an MCTS and an expert (MCTS) combined with an apprentice (NN) can (be adapted to) play Risk is motivated.

1.2.2 Thesis Outline

Chapter 2 describes the rules of Risk, which will be played between two players. In Chapter 3, the overall background theory needed for the thesis is described. Chapter 4 presents the thesis implementation and methodology, how the reinforcement model is linked to Risk. The results are presented in Chapter 5, which are then discussed in Chapter 6. Chapter 7 concludes the thesis.

Chapter 2

The Game of Risk

Risk is a turn-based strategy game where the whole world is at stake and can be played between two to six players. The objective is global domination, to conquer all continents by eliminating the other players. How this is accomplished is up to the player, forming and dissolving alliances is of choice but not necessary. In the standard version, the game map is a simplified political map of the world (Figure 2.1). This chapter will describe the essentials of the game. The formalism of the RL algorithm for Risk is presented in Section 3.2.1.

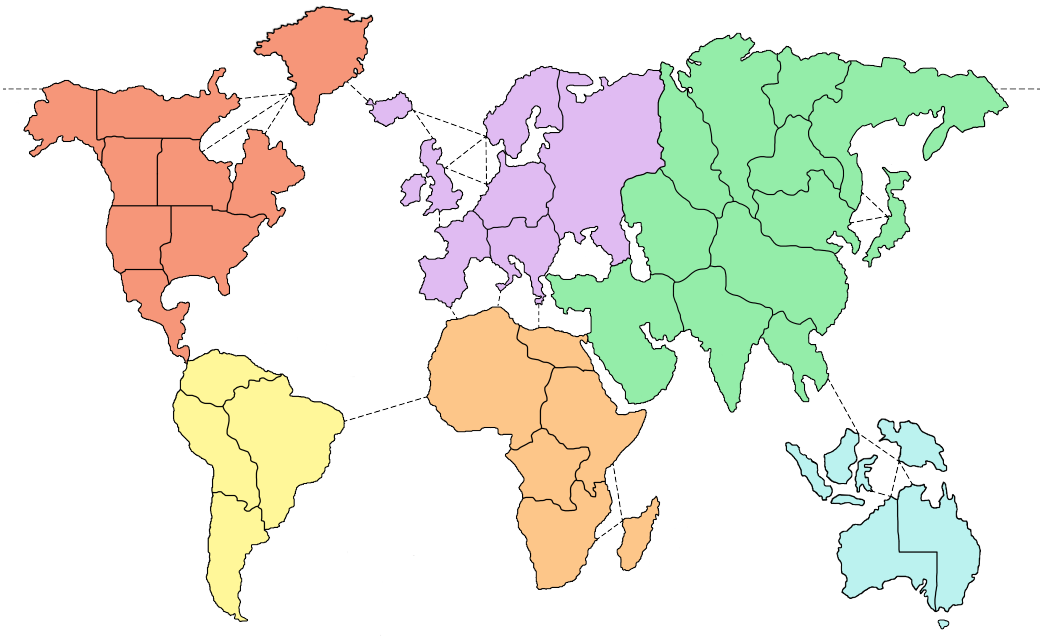


Figure 2.1: Visualisation of the Risk game map.

2.1 Rules

The game's main objective is to conquer the world, which consists of six continents split into 42 territories. Depending on the number of players, the rules will differ. As the thesis is conducted by playing the game between two players, the two-player rules will be explained. The reason behind choosing two-player Risk is that when playing with more than two active players, there might occur alliances or diplomatic play, which will not be taken into account when developing the agent.

When playing the two-player game, a neutral player is added, which does not attack or receive reinforcements. The neutral player's purpose is to act as a buffer between other players. The neutral player also introduces more strategic play. The other players have to evaluate the worth of attacking a player that will not come back to haunt them after they have conquered its territories.

After the initial setup, the game has two phases, the initial draft phase and the game phase.

2.1.1 Initial Draft Phase

The game has 42 cards excluding two wild cards. Each card represents a territory on the map. The game starts with each player getting randomly assigned territories by dealing out the cards (disregarding the wild cards) and reinforcing the territories with one unit.

In the next step, each player places 22 units onto its territories. This step is divided into ten turns. Each turn, the player places two units onto any one or two of its owned territories and one neutral unit onto neutral owned territory. The initial phase is concluded when all ten turns have been played.

In most computer/mobile implementations of the game, the standard option is to auto initialise the draft phase, i.e., skip the phase and let a random generator produce the outcome.

2.1.2 Game Phase

Each turn is split up into four different actions, and the order of play is always the same.

Receive troops: Three units is the minimum number of units the player can receive each turn, but this number increases depending on how many continents and territories the player occupies. Each continent is worth a different amount of units depending on the difficulty to retain it. The number of units received from owning territories is the total number of territories held divided by three.

There is a third way the player can receive extra troops, by handing in cards acquired from successful attacks. Aside from the cards representing a territory, they also have a symbol printed on them, an infantry, cavalry or a canon. The collected cards are hidden to the opposing player and handing in three of a kind or one of each at the start of the turn yields additional troops. There are two different rulesets for card bonuses.

Fixed rules: Playing with the fixed ruleset, each combination of three cards has a fixed value. Three infantries yield four units, three cavalries six, three canons eight and one of each ten units.

Progressive rules: Playing with the progressive ruleset, each time a combination of cards is handed in, the number of troops received increases. The first combination yields four troops, second six, third eight. After six combinations, the number of troops is increased by five instead of two.

The two rulesets have one thing in common, owning a territory represented on the card to be handed in, yields an additional two troops to that territory. To make the game more strategic, the chosen ruleset for this thesis is the fixed ruleset.

Place troops: Place the troops that the player has received onto territories they own in order to conquer new territories or defend against incoming attacks.

Attack: This is an optional action. To attack, the source of attack needs to have more than one unit, as one unit needs to be left behind to control the territory in the case of a defeat. The targeted territory needs to either be neighbouring or connected via the rules. The attacker chooses the number of units to attack with, attacking with more than one unit yields extra dice to roll, up to three dice. The targeted territory defends with its units, defending with more than one unit yields an additional die to roll. Each die represents one unit. Rolling the dice, highest rolls are compared, and players lose one unit for each lower die. An equal die

roll is in favour of the defender. If the attack is successful, the attacker takes control over the territory with the remaining army that launched the attack. The player can choose to stop any attack mid-fight, attack multiple territories in succession, and stop whenever it feels right. The only time the player has to stop the attack action is when there is no source to attack from.

When attacking a neutral territory, the opposing player rolls the neutral player's dice. If there was one successful attack, a territory was conquered, the player draws a card from the deck.

Fortification: Moving armies between territories can be essential in wars, but as the attack action, this is an optional move. The player can only fortify once each turn, from one source to one target. The source needs to have more than one unit, as one unit needs to be left behind controlling the territory. The target territory needs to be ruled by the player. A target territory can be any territory that has a connected path to the source, where the path is territories occupied by the player.

The turn ends after the fortification action.

The game is played until one player has global domination or when one player surrenders.

Chapter 3

Theoretical Background

All beings learn new things through some input. We either watch other people practise something or spend an endless amount of hours in subjects, to get some knowledge that we can pass down. A machine learning (ML) algorithm is not that different, in the sense that it needs to be trained to operate successfully. This training is executed by supplying the program with data such that it can gather information from it through, e.g., observation or simulation. In general, an ML algorithm tries to learn a mathematical function that relates inputs and outputs, and to make predictions without implicitly being programmed to forecast the prediction. This process can be split into three different fields: SL, unsupervised learning and RL.

The ML algorithm of AlphaZero and EXIT is a RL algorithm. It consists of an MCTS, used in a RL setting, combined with a NN, trained by utilising SL. Below is a description of SL, RL, MCTS, NN, and finally, how MCTS, together with a NN, yield AlphaZero and EXIT.

3.1 Supervised Learning

SL is a term that consistently reappears in ML. The concept of SL is that a model's mathematical function maps the input data to a labelled output, map x to y. Hence SL requires labelled input-output pairs. The goal of SL is to develop a model which generalises to unseen input data, i.e., accurately map unseen input data to predict the output. Annotating the data is usually performed by a "supervisor" or an expert.

There is a substantial amount of research dedicated to SL, and many different algorithms developed. Two well-known and frequently used algorithms in SL

are linear regression and k-nearest neighbour. In linear regression, the goal is to model the relationship between x and y with a linear mathematical function. The k in k-nearest neighbour stands for how many neighbours to check the distance to before classifying a new data point to a group. The distance between two data points can be calculated by, e.g., the Euclidean distance formula. An excellent read for this subject where the algorithms, including others, are discussed more thoroughly is *Supervised Machine Learning* by Andreas Lindholm et al. [13].

Another SL algorithm that has caught considerable attention in recent years is NNs and deep neural networks. There is a lot of time and theory devoted to how and why NNs work and a good read is *Deep Learning*, written by Ian Goodfellow et al. [14]. NNs are further discussed in Section 3.5, as they play a central role in this thesis.

3.2 Reinforcement Learning

In *Reinforcement Learning: An Introduction* [10], Richard Sutton and Andrew Barto dive into the world of RL and explain many different methods of use. A general description of RL is that the objective is to develop an agent that tries to maximise a reward function by taking actions in its surrounding environment.

The main difference between RL and SL is that the agent is not provided labelled data, input/output pairs. It is not told which action to take to maximise the reward; instead, it needs to discover which actions will yield the most reward. However, SL can be applied to RL and is discussed further in Section 3.3. Unsupervised learning [14], in short, tries to find structures in the data as it is unlabelled, which can be useful in RL but is not the solution to maximising the reward function. RL is, therefore, often thought to be its own field in ML.

One of the challenges that arises with RL is finding the balance between wandering in uncharted territory (exploration) and current knowledge (exploitation). To formalise the interactions between an agent and its environment, i.e., the RL problem, a Markov decision process (MDP) is formulated. A generic description of an MDP is presented in Section 3.2.1.

3.2.1 Markov Decision Process

An MDP is a formalisation of sequential decision making, for every discrete time step, the agent observes its surrounding environment before making a decision. The following description of finite MDPs is a short summary from Sutton and Barto's book [10].

In many scenarios, the interactions between an agent and its environment can be discretised into a sequence of time steps. For every time step t, the agent receives some depiction of the environment's state $S_t \in \mathcal{S}$. In the game of Risk, a state S_t is any possible board layout in the set $\mathcal{S}$ from the game. From state S_t, the agent selects an action $A_t \in \mathcal{A}(s)$, where A_t is an action in the set of all legal actions $\mathcal{A}(s)$ given $S_t = s$. The actions in Risk include, receive troops, place troops, attack and fortify, all explained in Section 2.1.2. By performing action A_t, the agent receives a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and the state is progressed to $S_{t+1} \in \mathcal{S}$. This leads to the sequence,

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, ..., \tag{3.1}$$

moreover, in finite MDPs, $\mathcal{S}, \mathcal{A}, \mathcal{R}$ have a finite number of elements. The random values S_t and R_t only depend on the previous state S_{t-1} and action A_{t-1} (the Markov property). Therefore, S_t and R_t have well defined probability distributions. The dynamics of the MDP can hence be described by,

$$p(s', r \mid s, a) := Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}, \tag{3.2}$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, $a \in \mathcal{A}$. By using marginalisation, the state-transition probabilities can be acquired,

$$p(s' \mid s, a) := Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a). \tag{3.3}$$

Similarly, the expected rewards for taking a specific action is derived by,

$$r(s, a) := \mathbb{E}\left[R_t \mid S_{t-1} = s, A_{t-1} = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a). \tag{3.4}$$

Goals and Rewards

Informally, the agent's long-term goal is to find a policy (defined in Section *Policy and Value Function* below) that maximises the total reward it receives, not necessarily maximising the immediate reward. It is essential to know that the reward is the developer's way of communicating to the agent what they want it to achieve and not how.

There are two different branches when it comes to the agent's task, episodic and continuous tasks. An episodic task is a task that has a beginning and an end, while a continuous task does not have a terminal state. For this thesis, where the agent is to play Risk, the task is episodic. It begins with a new game and ends when a terminal state is reached, i.e., when a winner of the game has been decided. This also corresponds to the reward the agent receives. Similar to EXIT and AlphaZero, the only time the agent receives a reward is when a terminal state is reached. The reward is ± 1 for victory or loss respectively, hence the goal of the agent is to maximise this reward.

Policy and Value Function

The only way an agent can affect the reward it will receive is by choosing a sequence of actions. Hence, the behaviour of an agent can be described as the probability distribution of actions that the agent might take in each state, which is termed *policy function*,

$$\pi(a|s) = Pr\{A_t = a \mid S_t = s\}. \tag{3.5}$$

The *value function* is defined as the expected return when starting in s and following policy π, that is,

$$v_\pi(s) := \mathbb{E}_\pi \left[\sum_{k=0}^{n} R_{t+k+1} \,\middle|\, S_t = s \right], \ \forall s \in \mathcal{S}, \tag{3.6}$$

where $\mathbb{E}_\pi[\cdot]$ is the expected value of a random variable when the agent follows the policy π, and R_{t+n+1} is the reward after the final state.

3.3 Imitation Learning

The real world inspires the concept of imitation learning (IL), as intelligent species will often imitate others to develop new skills. There are many examples of this behaviour, e.g., when a lion cub learns to hunt, it tries to mimic every move the mother takes. An interpretation of IL is that, IL in ML is SL applied to RL. The goal is to have an apprentice solve the MDP with the help of an expert. Instead of maximising a specified reward function, the apprentice mimics the expert's policy π^* by learning from labelled data, generated from the expert. Therefore, the input data x for the IL model in this thesis is board states, and the target labelled data y is the expert policy for the corresponding states. When developing a zero-knowledge agent, the expert is to have no additional domain knowledge except the rules of the game.

3.4 Game tree

A game tree consists of nodes where each node represents a state of the game. Traversing between nodes requires that an action is applied. Searching a game tree is hence equivalent to an MDP sequence (eq. 3.1). In Figure 3.1 a game tree of Tic-Tac-Toe is illustrated, played from the perspective of x and where 1 represents a win, 0 a draw and -1 a loss. By expanding and searching a game tree, an RL agent can maximise the reward it will receive. If, e.g., a Tic-Tac-Toe game state is represented as a nine character string containing the numbers $1-9$ and x, o instead of numbers where the players have played, the first state in Figure 3.1 (before *Action 1* is played) would be represented as the string `oox4x6ox9`. Depending on the type of agent, the state-transition probabilities (eq 3.3) will differ, using the simple case of a uniform random agent, the state-transition probabilities for *Action 1* are,

$$\begin{aligned} &p(s_2 = \texttt{oox4xxox9} \mid s_1 = \texttt{oox4x6ox9}, a_1 = 6) = \\ &p(s_2 = \texttt{oox4x6oxx} \mid s_1 = \texttt{oox4x6ox9}, a_1 = 9) = \\ &p(s_2 = \texttt{ooxxx6ox9} \mid s_1 = \texttt{oox4x6ox9}, a_1 = 4) = \frac{1}{3}. \end{aligned} \tag{3.7}$$

For a more intelligent player, by searching the game tree, the optimal action would be: $a_1 = 4$, as it results in a minimum reward of 0 (draw), hence the state-transition probability function is different compared to the uniform random agent. In the case of Tic-Tac-Toe, when the agent searches the game tree, i.e., transitioning between states, the table is turned for the agent after each action played. Moreover, the agent will use its own policy but act as the opposing player and choose an optimal action for the opponent, hence why the game tree (Figure 3.1) includes the opposing player's actions. In a self-play game, the opponent of the agent is itself. There are numerous tree search algorithms one can employ, but for games such as chess, Go and Risk with high tree complexities, i.e., many different move sequences, MCTS is commonly used.

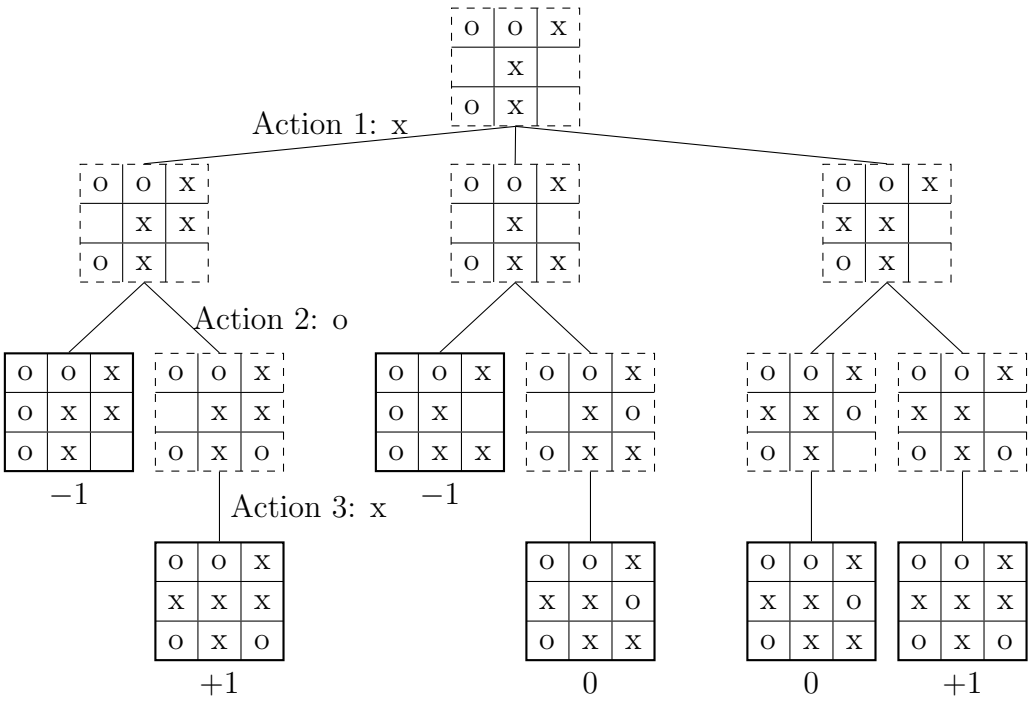
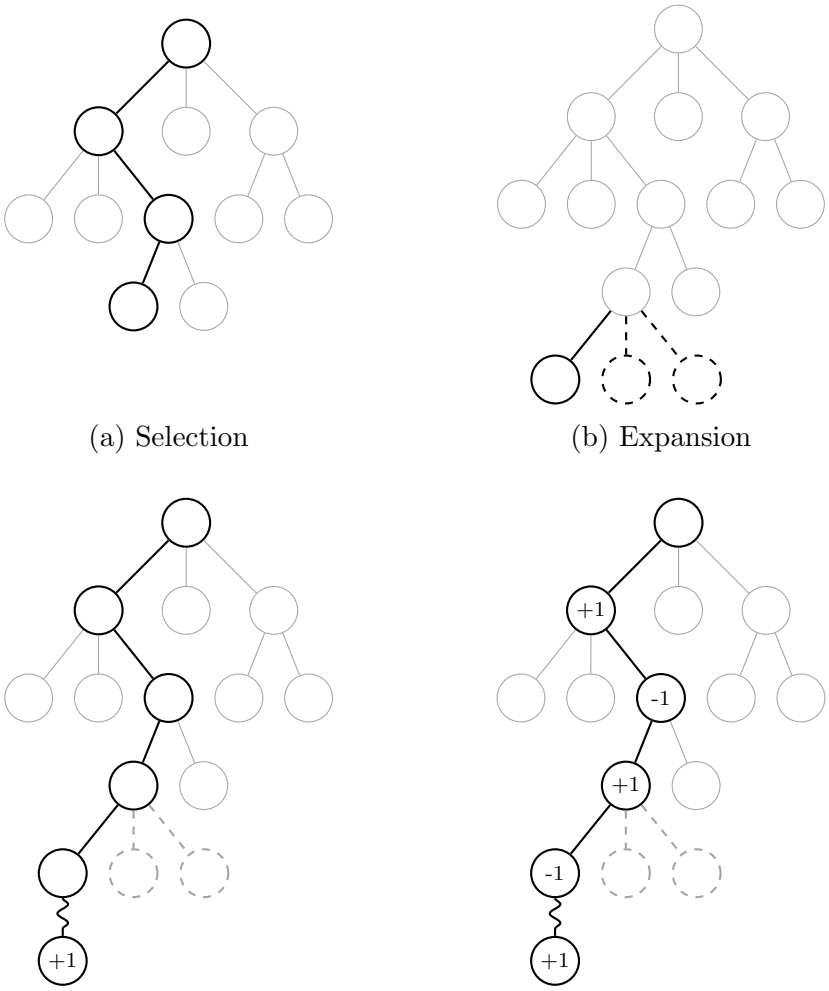


Figure 3.1: Tic-Tac-Toe game tree illustration.

3.4.1 Monte Carlo Tree Search

To value each node (state) in the tree, MCTS uses game simulations and expands the tree in the directions which have shown more promising results. MCTS can be divided into four stages, *selection*, *expansion*, *rollout* and *back-propagation* (Figure 3.2).



(a) Selection (b) Expansion

(c) Rollout (d) Backpropagation

Figure 3.2: MCTS phase illustration.

Selection

The first simulation initialises the tree by creating the root node, i.e., a node representing the current state of the game. Every forthcoming simulation starts from the root, nodes are then selected according to a tree policy until a leaf node is reached (Figure 3.2a). A leaf node is either a terminal node or a node from which no simulations has been made. The tree policy defines the balance between exploration and exploitation: the balance between long-term reward, exploring to improve the knowledge about each action, and immediate reward, exploiting the knowledge of which action currently has the largest estimated value. The most common tree policy used is to

choose the node which has the maximum Upper Confidence bounds applied to Trees (UCT) [15],

$$\mathrm{UCT}(s,a) = \frac{\beta(s,a)}{n(s,a)} + c_b\sqrt{\frac{\log n(s)}{n(s,a)}}, \tag{3.8}$$

where s is the state which the node represents, a the action taken to get to this state, $\beta(s,a)$ the cumulative reward acquired by simulations passing through this node, $n(s,a)$ the number of simulations passed through the node, c_b the exploration constant and $n(s)$ the number of simulations passed through the (current) node's direct parent. The first term (eq 3.8) represents the value of the node and controls the exploitation as it is correlated to the cumulative reward. The second term controls the exploration (possible long-term reward) and encourages searches of less-visited nodes. Each time a node is visited, both the numerator and denominator of the exploration term increase, decreasing its contribution. However, as any other child of the parent node is visited, the numerator increases, hence the exploration value of less-visited siblings increases. The value of c_b is empirically chosen and is different for each game.

The tree is hence traversed by selecting an action according to,

$$a = \mathrm{argmax}(\mathrm{UCT}(s,a)). \tag{3.9}$$

Expansion

When the search reaches a non-terminal leaf node, all legal actions of the node are expanded (Figure 3.2b). If there is no specific information, the default policy is that an action is chosen from uniform random and applied to get the corresponding state for rollout. The dashed lines and nodes in Figure 3.2b are the expanded legal actions and states which are yet to be visited.

Rollout

After adding the newly created node to the search tree, a rollout is played from the state until a terminal state is reached (Figure 3.2c). The simplest case, default policy, is to perform the rollout by choosing actions from uniform random. However, more carefully crafted rollout policies, e.g., incorporating domain knowledge, can improve the algorithm's performance [16].

Backpropagation

After the rollout, the result is evaluated and backpropagated through all the traversed nodes and updates their statistics. The statistics of a node are most commonly, the number of times it has been selected (i.e., visit count) and the total reward, which corresponds to results from rollouts. In Figure 3.2d, the result from a rollout is backpropagated and alternates between ± 1 for every node. This is only for games that alternate player after each move, e.g., Tic-Tac-Toe. In games such as Risk, where a player can conduct multiple actions before the game switches player, the result is backpropagated accordingly.

Chance Nodes

A non-deterministic action can be, e.g., dice rolls, and the outcome of non-deterministic action is often called a chance node [17]. When the MCTS encounters a chance node in the selection phase, it draws from the provided stochastic distribution to choose the next node instead of using the UCT formula (eq. 3.8). In the expansion phase, the MCTS expands all possible random outcomes.

Final Act

A common way to choose an action when all game simulations are completed, i.e., the tree search is concluded, is to select the action of the root that has been visited the most.

3.5 Neural Network

The original inspiration for artificial NNs emerged from how neurons in a brain function. Over the years, NNs have seen increased popularity in ML, especially when modelling non-linear relationships.

Neurons construct a NN, and each neuron can receive multiple inputs and generate a single output (Figure 3.3). Each connection between neurons has a weight. The weight of a connection factors the importance of the output from a neuron. Mathematically, a NN applies matrix multiplications in succession, and the elements in the matrices are parameters that can be optimised. Since matrix multiplications are linear and applying them consecutively is also linear, NNs would only be able to model linear relationships, but introducing non-linear activation functions between the linear layers solves this problem. As Hanin and Sellke show [18], by using the rectified linear unit (ReLU)

function as activation function (eq 3.10), any continuous function can be approximated to arbitrary precision,

$$f(x) = \max(0, x). \tag{3.10}$$

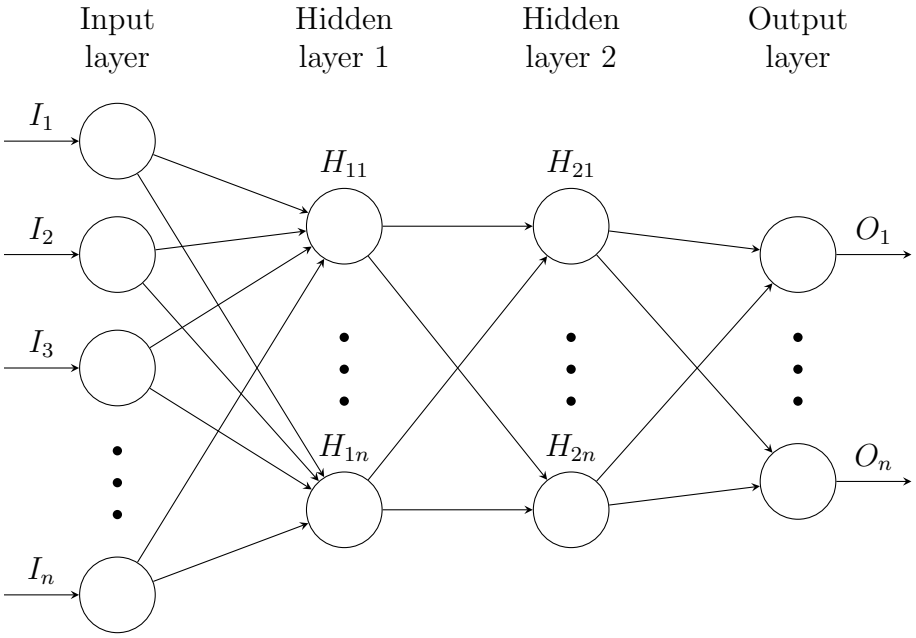


Figure 3.3: Illustration of a NN, each circle represents a neuron and the arrows in between represent the connections.

3.5.1 Loss Functions

The loss function will calculate how good, or bad the NN performs by calculating the cost, i.e., comparing the prediction (p) of the NN to the true value (y). Two common loss functions for regression problems are mean absolute error (MAE),

$$l(p, y) = \frac{1}{n} \sum_{i=1}^{n} |p_i - y_i|, \tag{3.11}$$

and mean squared error (MSE),

$$l(p, y) = \frac{1}{n} \sum_{i=1}^{n} (p_i - y_i)^2. \tag{3.12}$$

Taking inspiration from Silver et al. [6] and Anthony et al. [8], the loss functions used for this thesis are MSE and cross-entropy loss,

$$l(p, y) = -\sum_{i=1}^{n} y_i \log(p_i). \tag{3.13}$$

3.5.2 Optimisation

The optimisable parameters in a NN are the weights as they determine the importance of the output from a neuron. They are updated iteratively by minimising the loss function using a gradient method. Due to the optimisation being a gradient method, the loss function needs to be locally differentiable with respect to the weights.

The gradient method used for this thesis is the Adam optimisation algorithm [19]. Adam only requires first-order gradients and has been proven to be well suited for non-convex ML problems.

3.5.3 Dropout & Batch Normalisation

Regularisation methods are introduced to remedy the high model complexity that NNs naturally carry with them. A model with high complexity is prone to overfitting, i.e., learning a particular dataset too precisely and failing to predict new/future data. Dropout [20] is a regularisation method which randomly drops inputs into layers during the training procedure in order to reduce overfitting to specific data samples or inputs. Dropout may reduce overfitting but increases the training time due to ignoring some input features. Another performance-enhancing method is batch normalisation [21]. Batch normalisation works by normalising the layer inputs for each batch of data. This enables the usage of higher learning rates which increases the speed of training. Batch normalisation can, in some cases, also act as a regulariser. In this thesis, both dropout and batch normalisation are used.

3.6 AlphaZero & EXIT

Both AlphaZero and EXIT utilise an RL scheme with an expert and an apprentice. In many cases generating expert labelled data comes with a high cost. It is this issue that AlphaZero and EXIT solve by using an MCTS algorithm as an expert and a NN apprentice to assist the search. The theory behind it is that the expert will improve if the apprentice improves, i.e., as

long as the expert generates data that can improve the apprentice, it will implicitly improve itself (Figure 3.4). The apprentice is trained in iterations, where an iteration consists of a larger number of self-played games. To assist the MCTS, the apprentice is included in the selection phase (Figure 3.2a), i.e., by modifying the UCT formula (eq 3.8), this is further discussed in Section 3.6.2.

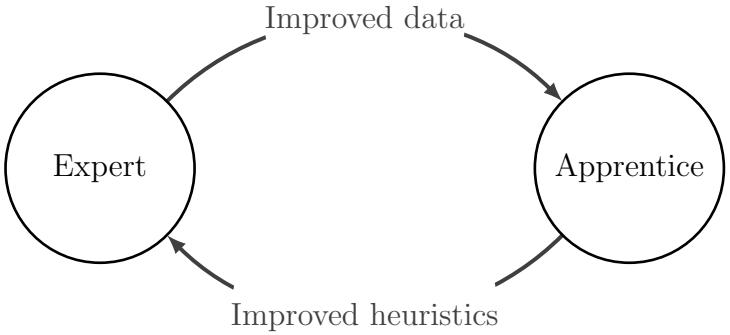


Figure 3.4: Illustration of an expert/apprentice setting.

The fundamentals of the NN architecture in both AlphaZero and EXIT consist of, first, multiple convolutional neural network (CNN) [14] layers, in conjunction with an image representation of the current state as input, secondly, a split into a policy and value head (Figure 3.5). The policy head has the MCTS policy π^* as ground truth, the search probability for each legal action of the current state. The value head has the game outcome as ground truth, ± 1.

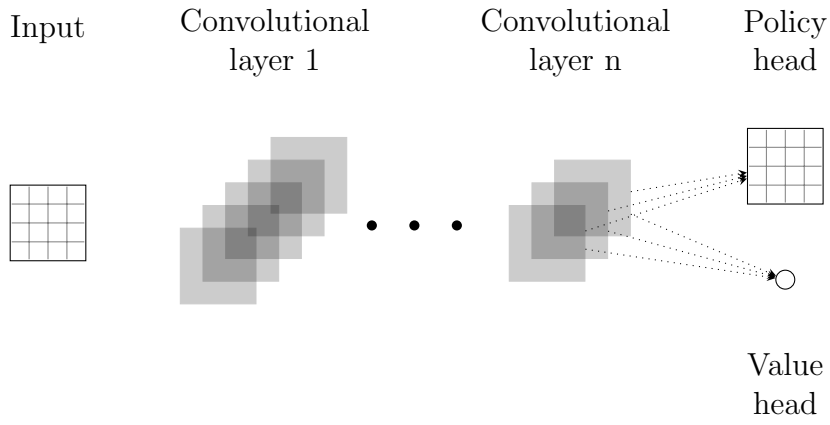


Figure 3.5: Illustration of AlphaZero and EXIT CNN architecture.

3.6.1 Data Generation

A significant difference between AlphaZero and EXIT is how data is generated.

For AlphaZero [6], the NN is trained by a self-play RL algorithm that uses the MCTS to play each move. For every state of the game, an MCTS guided by the previous iteration NN is launched, and π^* is provided. To progress the game, actions are selected according to the search probabilities provided by the MCTS. When the game is finished, the result is saved for the value head and combined with π^* an IL data sample is generated.

For EXIT [8], the data is also generated through self-play, however, instead of the MCTS, the RL apprentice plays against itself, and only the first iteration is self-played by an MCTS. To reduce the computational time, the self-play MCTS uses fewer simulations for each search than the expert. The self-play MCTS uses 1000 simulations, while the expert uses 10000. After the first iteration, the games are self-played by the most recent apprentice (NN). Every state of a game is saved, and when the game is finished, a random state is selected for the expert to evaluate. The expert, MCTS with the help of the latest iteration NN, searches and provides π^*, and together with the result of the game, a data sample is generated. The reason behind only selecting one state from each game is to ensure that there are no correlations in the dataset. Using the most recent NN to sample the states proves to have two advantages, selecting actions with the NN is faster and, in doing so, results in states closer to the distribution that the NN will visit at test time.

In EXIT, the network's value head is not activated until after $2-3$ iterations of data generation while AlphaZero activates it directly.

3.6.2 UCT

Another key difference between AlphaZero and EXIT is how the UCT formula is implemented, i.e., how the NN guides the MCTS. In AlphaZero the random rollouts of the MCTS are replaced by the value output from the NN. The standard UCT (eq 3.8), is replaced with Polynomial Upper Confidence bounds applied to Trees (PUCT) [6],

$$\text{PUCT}(s,a) = \frac{\beta(s,a)}{n(s,a)} + c_b\hat{\pi}(a|s)\frac{\sqrt{n(s)}}{n(s,a)+1}, \tag{3.14}$$

where $\beta(s,a)$ now is the backed-up value output from the NN and $\hat{\pi}(a|s)$ the policy output for action a from the NN. The original exploration term (eq 3.8), now includes the apprentice policy output, guiding the exploration to more prominent actions. EXIT, on the other hand, does not replace the random rollouts and instead only alters the UCT formula (eq 3.8),

$$\text{UCT}_{\text{PV-NN}}(s,a) = \frac{\beta(s,a)}{n(s,a)} + c_b\sqrt{\frac{\log n(s)}{n(s,a)}} + w_a\frac{\hat{\pi}(a|s)}{n(s,a)+1} + w_v\hat{Q}(s,a), \tag{3.15}$$

where w_a, w_v are weights, $\hat{\pi}(a|s)$ the policy output for action a from the NN and $\hat{Q}(s,a)$ is the backed-up average of the NN value output. Similar to AlphaZero, the idea is to introduce heuristics to the exploration and use the MCTS simulations more efficiently instead of depleting them on inferior actions. The value terms in $\text{UCT}_{\text{PV-NN}}$ are not used until the value head is activated.

3.6.3 Learning Policies

The search probabilities that the apprentice is to imitate is the visit probabilities generated by the tree search. Hence, the cross-entropy loss function is formulated as,

$$l(\hat{\pi}, y) = -\sum_a y(s,a)\log(\hat{\pi}(a|s)), \tag{3.16}$$

where $y(s,a) = \frac{n(s,a)}{n(s)}$. The defined cross-entropy loss (eq 3.16) is cost-sensitive, i.e., actions of similar strength will be penalised less severely. Moreover, it introduces a trade-off in accuracy on less important actions for increased accuracy on critical actions.

Chapter 4

EXIT/AlphaZero Adaptation to Risk & Implementation

This thesis evaluates how well an MCTS and an expert (MCTS) combined with an apprentice (NN) can excel in the game Risk. Key differences between Risk and board games as Go and Hex are:

- Risk has non-deterministic outcomes (dice rolls, cards) and incomplete information in the form of hidden cards, none of which are present in Go or Hex.
- Go, and Hex have matrix-like boards. Hence, the boards can easily be represented by an image/matrix where each cell of the matrix is a placement on the board. The game map of Risk is similar to an undirected graph and requires an additional dimension to represent connections between territories and continents, making it far more complex to represent as an image compared to Go and Hex.
- In Go and Hex, the game alternates between the players after each action taken. In Risk, a player may conduct multiple actions in succession before switching player.

These differences impacted how both the MCTS built the search tree and how the NN was constructed.

This chapter describes the methods used to answer the research questions, how the expert/apprentice setting was implemented, how Risk needed to be action pruned [23] and how the MCTS and the NN where implemented.

4.1 Expert/Apprentice Setting

The computer cluster used was relatively small in comparison to what Silver et al. and Anthony et al. used. Due to the computational limitations, this thesis's data generation was similar to how Anthony et al. (EXIT) implemented their data generation. The first iteration was self-played by an MCTS with 300 simulations, and the expert MCTS for all iterations had 10000 simulations. The generated data was split into two sets for the NN, train and validation:

- Train — Used for training the NN and optimising the NN weights (85 %).
- Validation — Used for evaluating the performance of the NN and tuning of hyperparameters (15 %).

The main tools used to implement the game, expert and apprentice were:

- Python 3.7.6
- NumPy 1.18.5
- Pandas 1.1.4
- TensorFlow 2.1.0
- ScikitLearn 0.23.2

4.2 Risk Action Pruning

The limitations in computational resources together with Risk's high game complexity, i.e., the existence of many different move sequences, hindered the implementation of a pure zero-knowledge agent. Hence, several trivial legal actions in the game phases of Risk were diminished by following simple logical arguments based on the nature of the game.

4.2.1 Receive troops

For simplicity, the action of receiving troops through turning in cards was not a decision the MCTS evaluated. Instead, at the start of a turn, if the agent had a card combination, turned it in automatically, giving the agent more troops to place.

4.2.2 Place troops

If the player receives ten troops and occupies 20 territories, there are 10^{20} possible legal actions. This is reduced in two ways, by limiting where the player can place troops and how many:

- A player can only place troops in a territory with at least one neighbour controlled by the enemy.
- If the player has $n < 5$ troops to place, its only option is to place them all.
- If the player has $n \geq 5$ troops to place, the combinatorial explosion of all possible ways to deploy troops is managed by using hierarchical expansion (Section 4.3.3).

Limiting the player to only place troops in a territory with at least one enemy-controlled neighbour should not hinder the player's performance. There are not many cases where the superior move is to place troops in a territory surrounded by friendly neighbours. The player is most likely to place troops in a territory where it can defend its borders or from where it can acquire new territories. The player is also most likely to place multiple units in a territory, making the defence stronger and possible attacks easier, hence the action prune to how many units the player can place in a territory.

4.2.3 Attack

When attacking or defending, instead of choosing the number of units to attack/defend with, the attacker attacks with all troops except one, leaving one behind to control the source territory, the defender defends with all. This has been shown by Osborne [22] to be the most effective and statistically best way to perform an attack. To reduce the possible move sequences even further, battles cannot be interrupted, i.e., either the defender entirely defeats the attacking armies or vice versa. A similar implementation called blitz is made in the official Risk application, "RISK: Global Domination".

4.2.4 Fortification

Similar to place troops:

- A player may only fortify to a territory that has at least one neighbour controlled by the enemy.

- If the player has $n < 4$ troops in the source territory, its only option is to fortify all except one, leaving one behind to control the source territory.
- If the player has $n \geq 4$ troops in the source territory, the player is presented with two options, fortifying with all except one or half rounded up.

4.2.5 Incomplete Information

To remedy the fact that Risk has incomplete information in hidden cards, which increases the tree complexity, the cards were implemented to be visible for the agents in the state. This information is used by the MCTS in the cutoff function in the rollout (Section 4.3.1), and as input to the NN (Section 4.4.1).

4.3 MCTS

4.3.1 Cutoff in Rollout

Similar to action pruning the agent, due to the constrained computational resources, instead of the random rollout playing to a terminal state of the game and returning the game result ± 1, a trivial cutoff function that arises through the rules of the game without any further analysis was implemented. Moreover, the cutoff can also improve the algorithm's playing strength [16]. The random rollout was cutoff after n turns played. The state was evaluated based on the troops available for the players, i.e., the current troop count, territorial bonus, continental bonus, and possible card bonus. The total amount of this score for each player was normalised by the total amount of scores for both players. The scores were then shifted to be between ± 1 and returned. An example is shown in Table 4.1.

Table 4.1: Example of a rollout cutoff after n turns. The shifted value returned represents the win/loose probability for each player.

	Player 0	Player 1
Troops on map	63	42
Territory bonus	9	3
Continental bonus	4	0
Card bonus	0	10
Total	76	55
Normalised	0.58	0.42
Shifted	0.16	-0.16

4.3.2 Chance Nodes

The chance nodes in Risk appear when determining the outcome of a battle and when drawing a card. The latter was not implemented into the MCTS, and instead, if eligible, the agent just drew a card.

The implementation of a chance node for a battle outcome differed from how chance nodes are typically implemented. The number of combinations possible for a battle outcome in Risk rapidly increases as larger troops attack each other. To calculate the probability of every possible battle outcome and create a node for each result is infeasible. Hence, the distribution was unknown, and to solve this, a battle simulation was implemented, and the chance node was selected or expanded accordingly.

4.3.3 Hierarchical Expansion

The MCTS was implemented such that almost all decisions made were hierarchical [23]. Instead of giving the MCTS the legal actions of, e.g., place troops on the form, a_i = (Target, Count), the MCTS was given the decision to choose place troops target first, and then select count. This was also done for the attack phase, select attack source first, and then target. For the fortification phase, choose the fortification target first, then source and finally count. The only exception of where a hierarchical decision was not implemented was the decision not to attack or fortify. This decision was instead presented as the option `None` in attack source and fortification target. The reasoning behind the hierarchical idea is that if there is, e.g., a place troops count action that is better than others, the MCTS will converge faster towards the better move and gradually start to ignore the others,

moving deeper in the tree instead of wider. Backtracking to the place troops count options presented to the MCTS, when the player had $n \geq 5$ troops to deploy, the options were, place all, half rounded up or one.

All parameters of both the expert and self-play MCTS were empirically chosen and can be found in the appendix (Table A.1).

4.4 Neural Network

4.4.1 Input Parameters

Since the board map of Risk is more similar to an undirected graph compared to a matrix, and requiring an additional dimension to represent the connections between territories and continents, instead of representing the state as an image in conjunction with a CNN, a feedforward NN was used with an input consisting of a 134x1 vector. Likewise, when applying AlphaZero to Coral Sea, Moy and Shekh [12] use a vector representation of the board state to tackle a similar problem of the need for an additional dimension added by the property of multiple pieces being allowed in a single hex.

- elements $0 - 41$ of the vector correspond to player 0 and how many troops they have in each territory,
- elements $42 - 83$ correspond to player 1,
- elements $84 - 125$ are for the neutral player,
- the remaining elements $126 - 133$ represent the number of cards of each type the players hold ($126 - 129$ for player 0 and $129 - 133$ for player 1).

4.4.2 Output Parameters

The overall architecture of the NN was similar to both AlphaZero and EXIT, after the input, there were some shared layers which were then split into eight separate branches, one for each output to match how the MCTS operated,

- place troops target, output size 42x1,
- place troops count, output size 3x1,
- attack source, output size 43x1,
- attack target, output size 42x1,

- fortify target, output size 43x1,
- fortify source, output size 42x1,
- fortify count, output size 2x1,
- value, output size 1x1.

The 43rd output for attack source and fortification target was the decision not to attack or fortify. All outputs except value used a *softmax* activation function to generate the probability for each action. The value output used a *tanh* activation function, i.e., outputting a value between ± 1. However, before the NN outputted the probabilities, all illegal actions were filtered out, i.e., $a_{illegal} = \infty$, resulting in the softmax output $\hat{\pi}(a_{illegal}|s) = 0$. The shared layers consisted of two hidden layers before separating into branches, all individual branches had one additional hidden layer.

The training was stopped using early stopping with the patience of 30, other hyperparameter choices for the NN can be found in the appendix (Table A.2).

Chapter 5

Results

Although Risk shares some similarities with the board games Go and Hex, the differences described in the previous section implies challenging tasks when applying EXIT or AlphaZero directly to Risk. The non-deterministic outcomes, the non-matrix-like board, and that a player conducts multiple actions in succession before switching player, are all elements of increased complexity. Hence, conducting the thesis had its ups and downs, and is reflected in the results.

5.1 MCTS

The results of how well the MCTS adapts to play Risk is presented in this section.

5.1.1 Cutoff Performance

Choosing the parameter n_{turns} (Section 4.3.1) in the cutoff function had a direct impact on the playing strength of the MCTS. The parameter was empirically determined, and the optimal choice was $n_{turns} = 8$. The speedup and playing strength of MCTS with 500 simulations when using the cutoff rollout, compared to rollout to a terminal state, is shown in Table 5.1. By using the cutoff rollout, the playing strength increases, and the run time of the rollout decreases.

Table 5.1: Speedup, tournament result (240 games) and win ratio of cutoff rollout (eight turns) compared to rollout to terminal state using MCTS with 500 simulations.

Speedup	Tournament Result	Win ratio (%)
4.58	182-58	75.8%

5.1.2 Playing Strength

To measure the playing strength of the MCTS when varying the number of simulations, i.e., computational effort, tournaments consisting of 240 games each were played with an MCTS having 5000 simulations versus MCTS with 100, 500, 1000, 1500 and 2000 respectively. The tournament win ratio is presented in Figure 5.1. MCTS 5000 has a win ratio of 73.3% against the weakest opponent, MCTS 100. Overall the win ratio for MCTS 5000 decreases when it plays against stronger opponents. The weaker agent's performance increases substantially (17.1%) when switching from MCTS 100 to 500, while not so much when going from 500 to 2000 (2.9%).

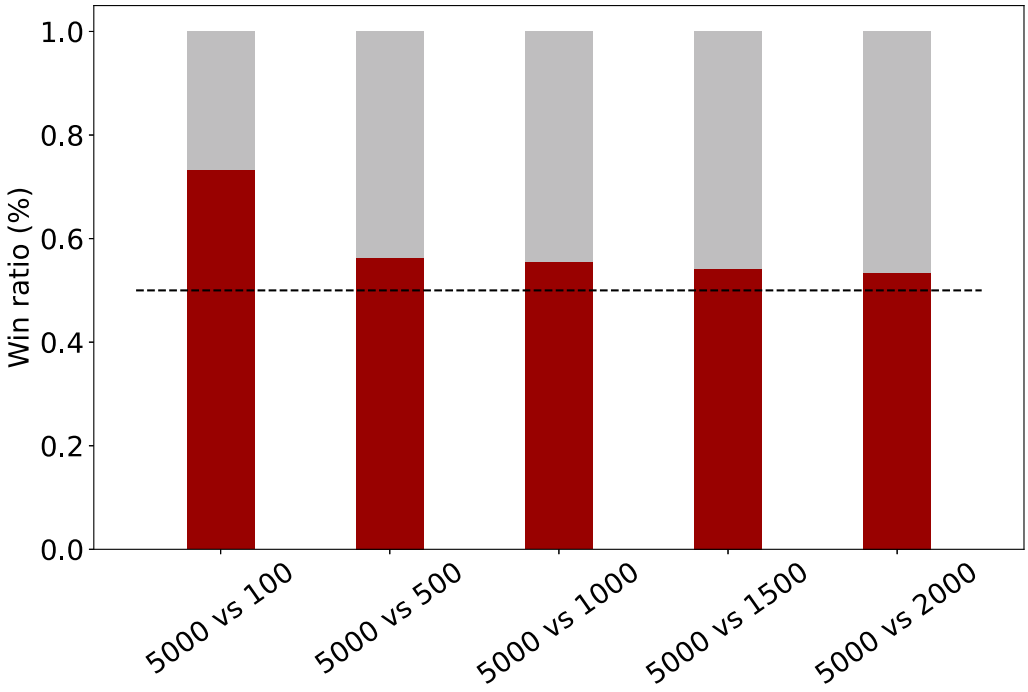
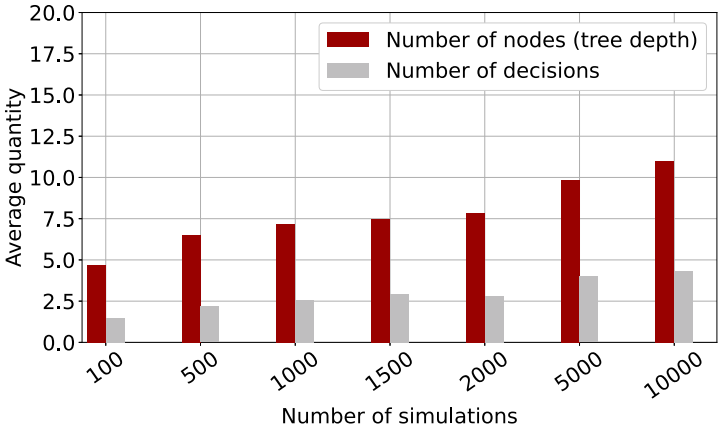


Figure 5.1: Win ratio between different MCTS simulations, 240 game tournament, the red bars represents the 5000 simulation MCTS while the grey bars represents its opponent.

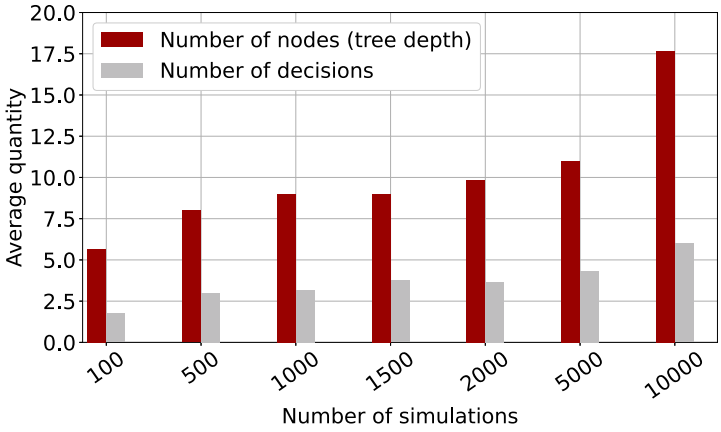
5.1.3 Search Depth

The search depth of an MCTS shows how well it adapts to play the game. In Figure 5.2 the average tree depth and the number of decisions made by MCTS 100, 500, 1000, 1500, 2000, 5000 and 10000 are shown in different parts of the game, early, mid, and late. The tree depth is the furthest level/node reached by the MCTS during a search. A decision is counted as either: i) a place troops decision, selecting both a place troop target and count, ii) an attack decision, selecting both attack source and target, or iii) a fortification decision, selecting fortification target, source and count. The big difference between the number of nodes and the number of decisions is a result of the hierarchical expansion. It takes multiple nodes to complete a decision. Therefore, the playing strength of the MCTS can be misinterpreted by solely focusing on tree depth. Interestingly, the tree depth is quite similar for 500, 1000, 1500, and 2000, especially in the early game. The number of decisions taken by each agent (including MCTS 5000) is even more similar.

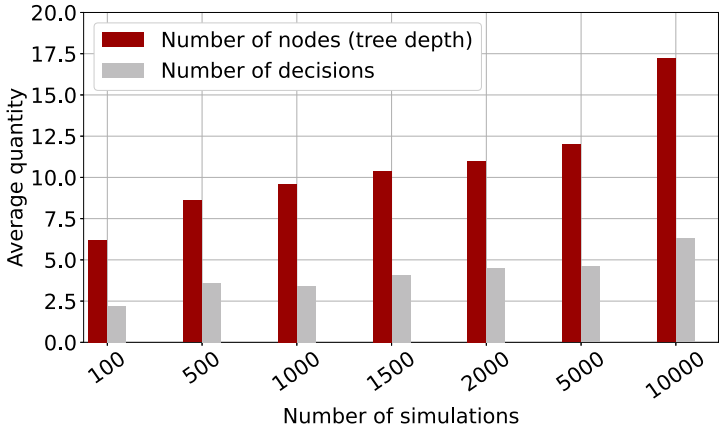
Furthermore, the average number of player switches in the deepest path of the search tree is another measure of interest. It implicitly indicates the behaviour of the MCTS. It shows if the MCTS chooses to explore its own attack options more rather than continue and further investigate the countermoves of the opposing player. As Tables A.4, A.5 and A.6 show it is not common for the MCTS's to examine the entire turn of the opposing player as the average number of player switches is rarely greater than one.



(a) Early game



(b) Mid game



(c) Late game

Figure 5.2: Plots showing the depth of the search tree at different stages of the game.

5.2 EXIT/AlphaZero

This section presents the performance of the apprentice in an expert/apprentice setting.

5.2.1 First Iteration Apprentice Performance

In Figure 5.3 the outputs of the loss function from NN_0 are shown after the first iteration of data generation, a dataset size consisting of 10305 data samples. As the individual plots show, apart from place troops count, attack source, and value, the validation loss does not decrease with more epochs of training. The aggregated loss decreases, but by observing the individual outputs shows that the aggregated loss follows the curves of the value loss.

To evaluate if NN_0 could guide the MCTS better than random heuristics, a tournament consisting of 240 games was played between NN_0 and an agent playing randomly. For reference, a tournament between MCTS 100 and a random agent was also played. Table 5.2 shows that NN_0 (10305 data samples) had hard times defeating the random agent, having a win ratio of 27.1% while it was an easy match for MCTS 100, having a 98.8% win ratio.

Table 5.2: Table showing the results from the 240 game tournaments between NN_0, MCTS 100 and the random agent, as well as the win ratio for NN_0 and MCTS 100.

Tournament	Result	Win Ratio (%)
NN_0 vs Random	65-175	27.1
MCTS 100 vs Random	237-3	98.8

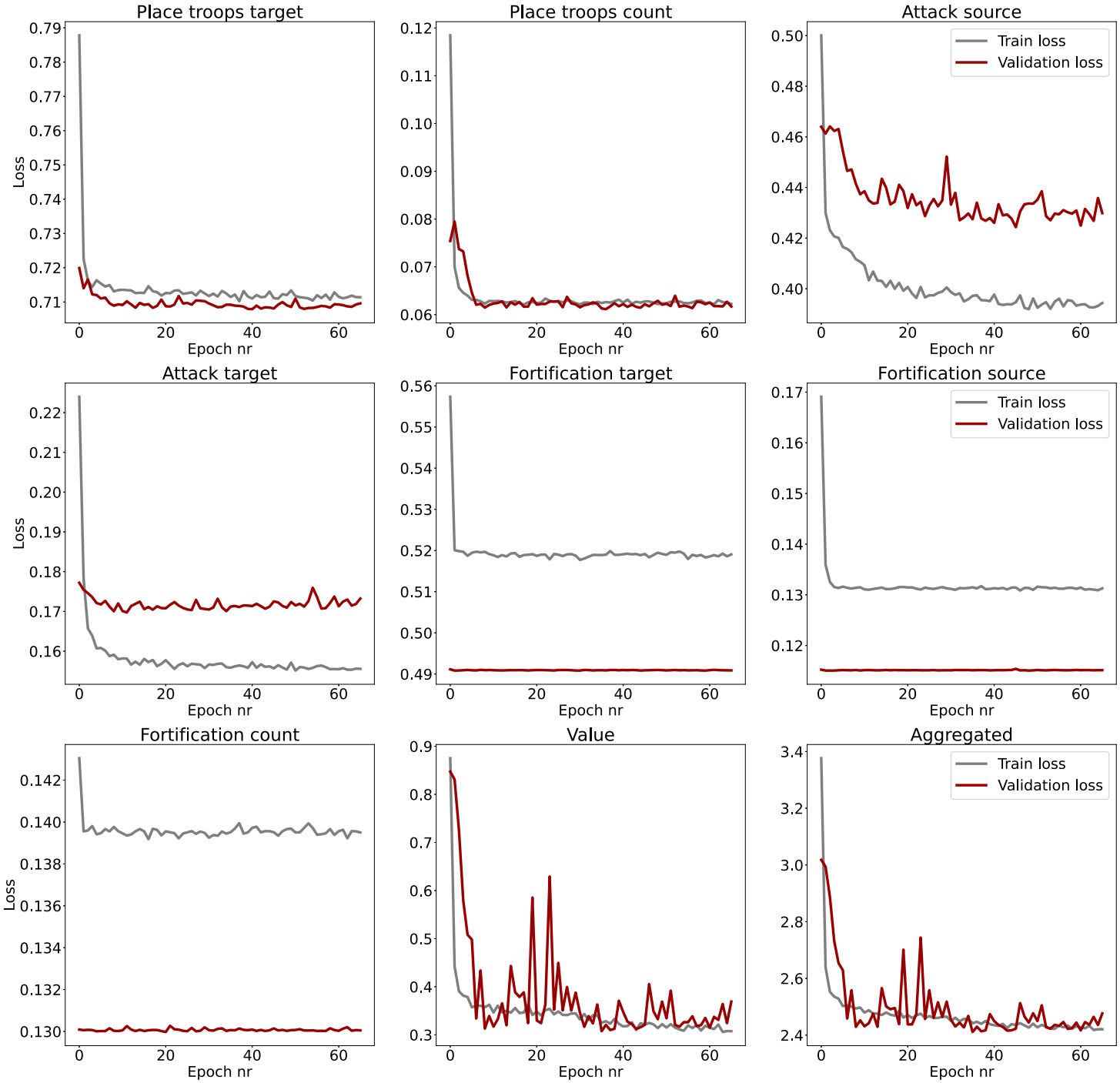


Figure 5.3: Plots showing all individual output losses of NN_0, as well as the aggregated loss from all outputs for the first iteration of data generation.

Expansion of the First Iteration

Considering NN_0 had hard times learning the expert's policy and had a lower performance than a random playing agent, an expansion of the first iteration, generating 14375 data samples, was carried out to see if more data solved the issue. As the new data samples extended the first iteration, the data generation was carried out in the same manner, MCTS 300 for self-play and an expert with 10000 simulations. The new NN, NN_{01}, had its weights randomly initialised and was trained on all data generated, 24680 data samples. Figure 5.4 compares the training loss for NN_0 and NN_{01}, while Figure 5.5 compares the validation loss. As Figure 5.4 and 5.5 show, there is no significant difference between NN_0 and NN_{01} in the learning process.

In Table 5.3 the tournament result, 240 games, between NN_{01} versus the random agent is shown. NN_{01} (31.6%) had a slightly higher win ratio than NN_0 (27.1%) versus the random agent.

Table 5.3: The results from the 240 game tournament between NN_{01} and the random agent, as well as the win ratio for NN_{01}.

Tournament	Result	Win Ratio (%)
NN_{01} vs Random	76-164	31.6

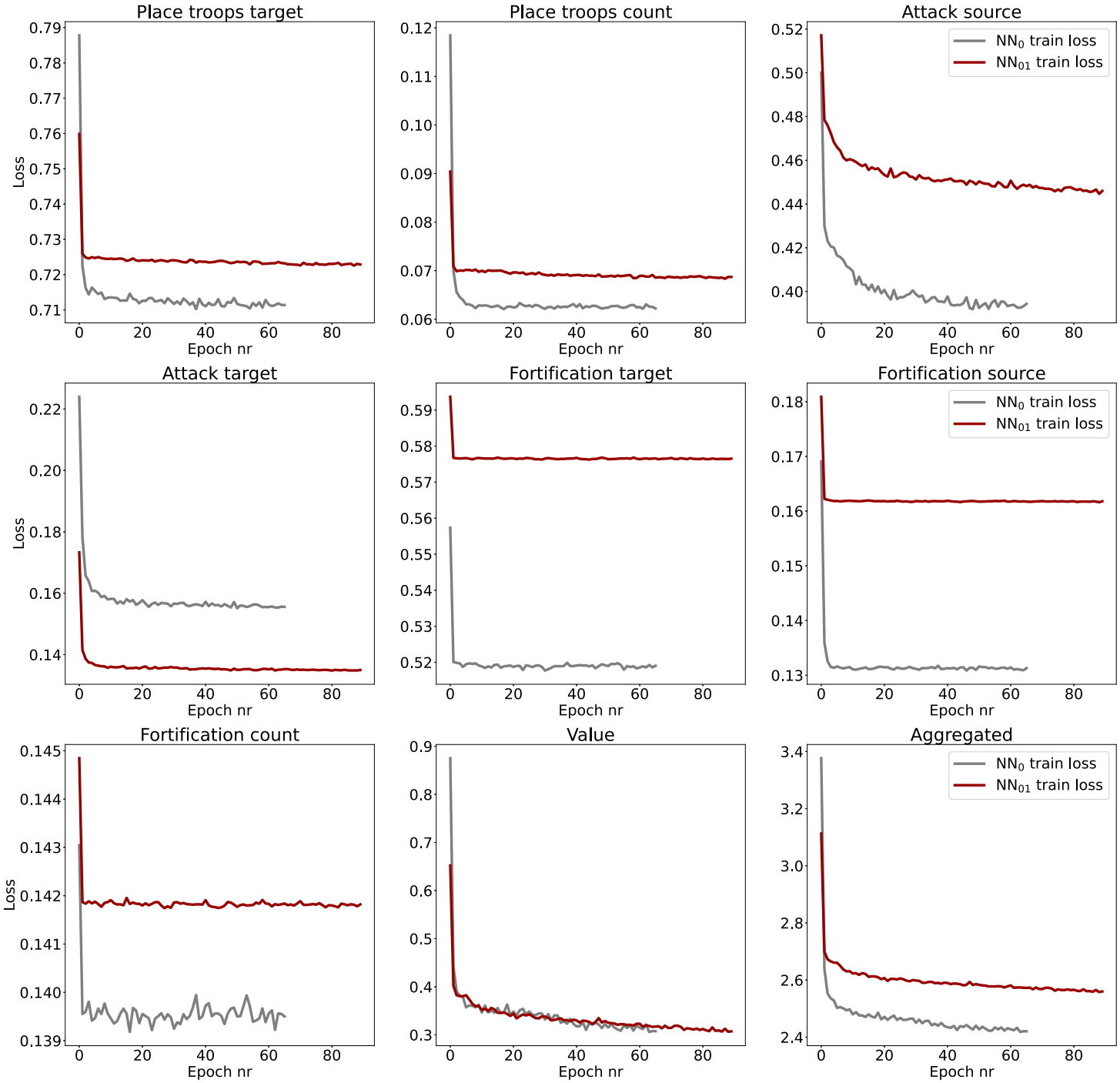


Figure 5.4: Plots showing the training loss for all individual outputs of NN_0 and NN_{01}, as well as the aggregated loss from all outputs.

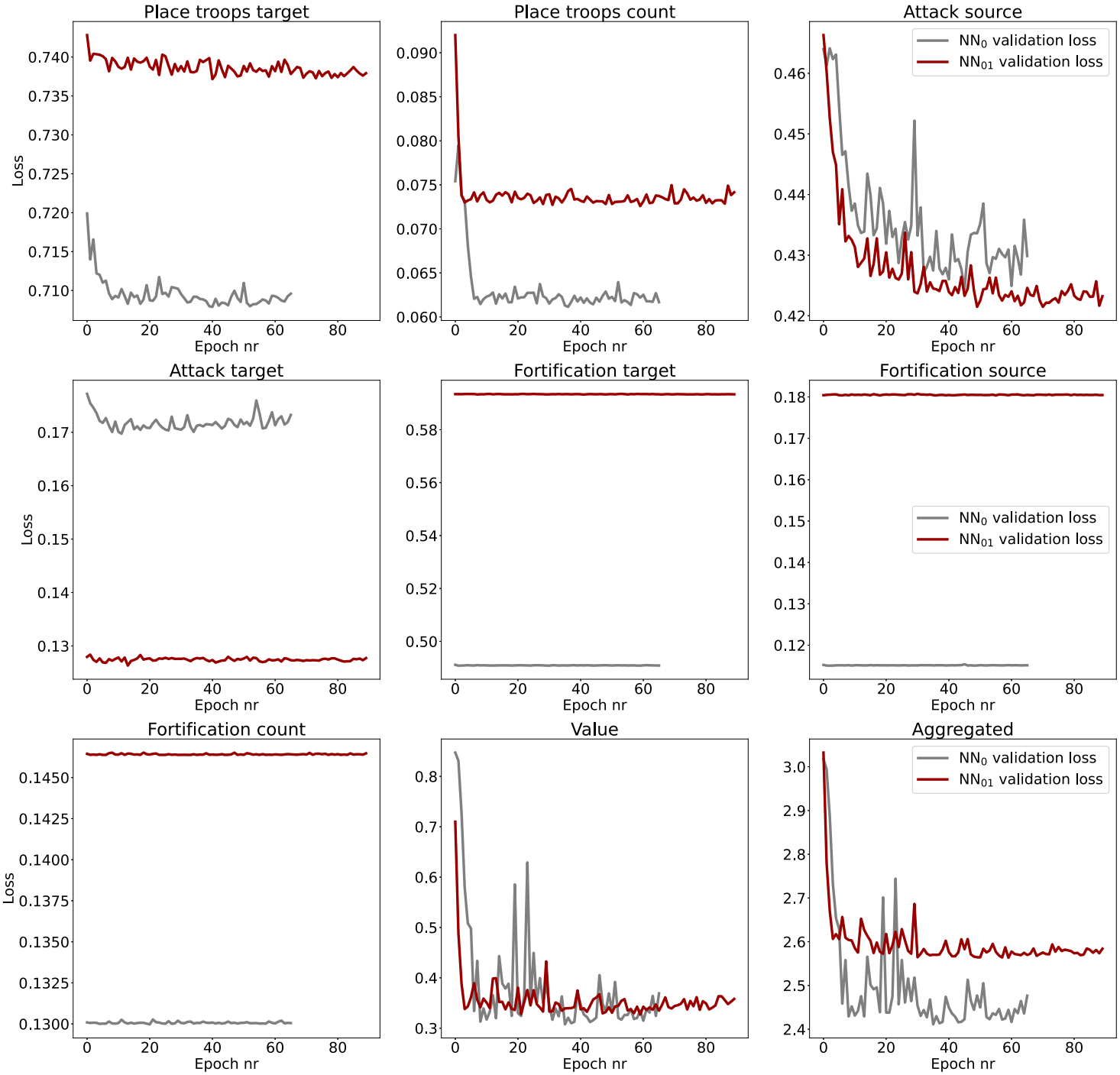


Figure 5.5: Plots showing the validation loss for all individual outputs of NN_0 and NN_{01}, as well as the aggregated loss from all outputs.

Chapter 6

Discussion

Although the results were hamstrung by the challenges of the thesis combined with the orders of magnitude smaller computational resources compared to EXIT and AlphaZero, they still carry some significance for the research project at FOI.

6.1 MCTS

The 24.2% win ratio of MCTS 500 with random rollout to terminal state over MCTS 500 with a cutoff function in the rollout, showed that an MCTS without any domain knowledge could be feasible, and there is room for improvement.

6.1.1 Cutoff Function

As the theory suggested, introducing a cutoff function for the rollout increased the playing strength of the MCTS and decreased the run time, but moved the agent away from being trained without any domain knowledge. However, the performance increase is hard to neglect.

6.1.2 Performance

MCTS 5000 was stronger than all of its opponents, but not by much, only having a 56.2% win ratio against an opponent with ten times fewer simulations per decision, i.e., MCTS 5000 taking ten times longer per decision and requiring more computational power. The playing strength of the pure MCTS can be explained by the search depth and the number of decisions taken by the agents. As the results show, the difference between the number of decisions taken by MCTS 5000 and its opponents (excluding 100)

decreased as the game entered mid and late game. Even though MCTS 5000 reached deeper in its search, it was strongest in the early game, as the difference between the number of decisions was largest. This is reflected in the scores as well. The early game is essential, as evident as it is, the better an agent sets up the early game, the easier it will have to win the whole game. The difference in the number of decisions taken between 5000 and 500 was at largest 1.83, smaller when increasing the number of simulations, and not large enough for it to dominate over its opponents (excluding 100 where the difference was relatively constant around 2.5). An obvious question arises, how can the difference be so small? One can only speculate, but as Risk involves a substantial amount of randomness, there could have been a lot of inferior moves that the MCTS chose to explore further down the tree as it did not have enough simulations to converge past them.

Another complementary explanation for the general performance could be that the MCTS in many cases had the option to explore its own attack options further, rather than investigate the countermoves of the opposing player. Therefore, explore nodes which in short-term yielded better rewards but in long-term were poor decisions. This is supported by the average number of player switches in the deepest path of the tree search. Increasing the number of simulations for the MCTS did not substantially increase the number of player switches. Demonstrating that, when given more simulations, the MCTS became more aggressive, which does not necessarily imply increased playing strength.

6.2 EXIT/AlphaZero

The results showed that the NNs had a hard time learning the expert's policy, regardless of hyperparameter choices, network width and depth, all of which were tested with random and grid searches.

6.2.1 Apprentice Performance

Increasing the dataset size of the first iteration by ~ 2.4 times, only increased the win ratio between the NN and a random agent by 4.5%, while still not being able to beat random comfortably. This was a significant issue, and the decision was made not to proceed with more iterations. The next iteration expert would be guided by the NN, guiding it to decisions that are somehow worse than random, and generating new data that is worse than the first iteration. The NNs seemed to learn the attack source policy better than

the other policies. However, when investigating how the NNs played, it was observed that they were pacifists. They had learned not to attack, which explains why they had a hard time defeating the random agent. As the apprentice rarely attacked imposed significant implications in self-play, many games never ended. Therefore, no tournament result was shown between NN_0 and NN_{01}. Most of the states that would be presented to the expert would be states that are not conventional, states that not even random agents self-playing would reach. This supported the decision to not continue with more iterations. An explanation for why the NNs learned not to attack could be that the decision, `None`, was in every attack source decision. Looking at the expert's tree statistics, the decision to not attack always had a high number of visits, which is not surprising, as a good Risk player always evaluates when to stop attacking. There was an investigation of implementing a hierarchical expansion on the decision to attack or not, similarly to fortify or not, but there were no improvements. It was observed that the MCTS (expert) became a pacifist, which resulted in the NN learning pacifist strategies.

One could argue that increasing the dataset size of the first iteration or continuing with more iterations might have solved the issue. However, the computational resources required to generate a larger dataset size for the first iteration was not at hand in the time frame of the thesis. Instead of continuing with more iterations, time was dedicated to understand why the apprentice had hard times attacking and learning the other policies. It is essential to state that this is only conjectures made by observations and theory in zero-knowledge agents trained on other games. The issue may lay in how the state was represented to the network. The implemented version had the state represented as a single vector. Critical information about the map could be lost. Nothing illustrated the connections between continents and certain territories other than the order they came in. Therefore, the NN might have had problems relating what it had been trained on to what it was validated on. This is not a problem for games where the state is represented as an image, the tensors presented to the NN gives it a chance to learn the actual board layout and how different moves affect the game.

Another contributing factor to the poor performance of the apprentice could be the statistical variance in Risk. The vast number of random outcomes might have affected the training and validation data more than expected.

Chapter 7

Conclusion

This thesis was motivated by the success of EXIT and AlphaZero agents in strategy games and the lack of zero-knowledge agents in Risk. The thesis has evaluated how well an MCTS and an expert (MCTS) combined with an apprentice (NN) can (be adapted to) play Risk. In many cases, Risk is determined by non-deterministic outcomes, has large game complexity, and includes non-alternating decisions. Significant differences between the developed agent and an EXIT/AlphaZero agent include: (i) the state was represented as a vector while in EXIT/AlphaZero the state was represented as an image, (ii) computational constraints, hence the introduction of action pruning and a cutoff function in the random rollout, both not present in an EXIT/AlphaZero agent.

This thesis has shown that an MCTS algorithm can in some degree excel in a game like Risk, beating a random playing agent comfortably, but there are challenges that need to be addressed. Introducing a cutoff function in the rollout, increased the overall performance of the algorithm. Increasing the playing strength of the MCTS came at a high cost of computational time. The most effective tournament MCTS, when regarding the win ratio and computational power needed, was not the one with the most simulations, it was MCTS with 500 simulations.

With this implementation of a zero-knowledge agent trained for Risk, the apprentice had hard times learning the expert's policy and became a pacifist. This halted the whole EXIT process and stopped data generation after the first iteration. A possible explanation for this is that the state was represented as a vector, and essential game information might have been lost. Increasing the dataset size of the first iteration slightly increased the apprentice performance against a random playing agent. However, it did not resolve

the issue of guiding the expert to decisions worse than random.

7.1 Future Work

MCTS

The playing strength of the MCTS can be increased by incorporating more domain knowledge by, e.g., policy enhancements or more extensive action pruning [24]. However, as this thesis explores, in essence, zero-knowledge agents, the usage of domain knowledge is sparse. Another solution can be to examine how to adapt the MCTS even further when regarding the existence of non-alternating actions by, e.g., using domain knowledge.

Apprentice

A possible solution might still be that the first iteration needs to generate much more data. Another solution could be to represent the state, somehow, in either 2D or 3D and use a CNN. Hence, evaluate if the problem with the thesis implementation lies in information loss within the state representation. It would also be interesting to investigate the usage of a graph neural network (GNN) as the board map of Risk is more of an undirected graph. As GNNs are relatively new in ML, they lack in documentation and scientific papers, designing such a network for this thesis would be out of scope.

Bibliography

[1] Martin van Creveld, 'Wargames' : From Gladiators to Gigabytes', Cambridge University Press, New York, USA, 2013.

[2] Murray Campbell, A. Joseph Hoane Jr, Feng-hsiung Hsu, 'Deep Blue', in *Artificial Intelligence*, 2002, Vol. 134, pp. 57–83.

[3] L. Victor Allis, 'Searching for Solutions in Games and Artificial Intelligence'. PhD thesis, 1994, Univ. Limburg, Maastricht, The Netherlands.

[4] David Silver et al., 'Mastering the game of Go with deep neural networks and tree search', in *Nature*, 2016, Vol. 529, pp. 484–489.

[5] Demis Hassabis, 'What we learned in Seoul with AlphaGo', *Google The Keywoord*, 16 Mar. 2016, Available from: `https://blog.google/topics/machine-learning/what-we-learned-in-seoul-with-alphago/`.

[6] David Silver et al., 'A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play', in *Science*, 2018, Vol. 362, Issue 6419, pp. 1140-1144.

[7] David Silver et al., 'Mastering the game of Go without human knowledge', *Nature*, 2017, Vol. 550, pp. 354–359.

[8] Thomas Anthony, Zheng Tian1, David Barber, 'Thinking Fast and Slow with Deep Learning and Tree Search', in *Advances in Neural Information Processing Systems 30*, 2017, pp. 5360-5370.

[9] Michael Wolf, 'An intelligent artificial player for the game of risk, PhD thesis', Darmstadt University of Technology, Germany, 2005.

[10] Richard S. Sutton and Andrew G. Barto, 'Reinforcement Learning: An Introduction', MIT Press, 2018.

[11] Richard Gibson, Neesha Desai, and Richard Zhao, 'An automated technique for drafting territories in the board game Risk', *AIIDE*, 2010.

[12] Glennn Moy and Slava Shekh, 'The Application of AlphaZero to Wargaming', in *AI 2019: Advances in Artificial Intelligence*, 2019, pp. 3-14. Springer International Publishing.

[13] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön, 'Supervised Machine Learning', unpublished book, publish year set 2021, Cambridge University Press. Available from: `https://www.smlbook.org/`

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, 'Deep learning', MIT press, 2016.

[15] Levente Kocsis and Csaba Szepesvári, 'Bandit Based Monte-Carlo Planning', in *Machine Learning: ECML 2006*, 2006, pp. 282–293. Springer Berlin Heidelberg.

[16] Peter D. Drake and Steve Uurtamo, 'Move Ordering vs Heavy Playouts: Where Should Heuristics be Applied in Monte Carlo Go', in *Proc. 3rd North Amer. Game-On Conf.*, 2007, pp. 35–42, Gainesville, Florida, USA.

[17] Stuart J. Russell and Peter Norvig, 'Artificial Intelligence: A Modern Approach', Prentice Hall, 2009, 3rd edition, pp. 177-178.

[18] Boris Hanin, Mark Sellke, 'Approximating Continuous Functions by ReLU Nets of Minimal Width', *arXiv:1710.11278*, 2018.

[19] Diederik P. Kingma, Jimmy Ba, 'Adam: A Method for Stochastic Optimization', *arXiv:1412.6980*, 2014.

[20] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever and Ruslan R. Salakhutdinov, 'Improving neural networks by preventing co-adaptation of feature detectors', *arXiv:1207.0580*, 2012.

[21] Sergey Ioffe, Christian Szegedy, 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', *arXiv:1502.03167*, 2015, Google Inc.

[22] Jason A. Osborne, 'Markov Chains for the RISK Board Game Revisited', *Mathematics Magazine*, 2003, Vol. 76, No. 2, pp. 129-135.

[23] Gijs-Jan Roelofs, 'Pitfalls and Solutions When Using Monte Carlo Tree Search for Strategy and Tactical Games', in Steve Rabin, 'Game AI Pro 360: Guide to Tactics and Strategy', Boca Raton: CRC Press, 2019, 1st edition, pp. 217-228

[24] Cameron Browne et al., 'A Survey of Monte Carlo Tree Search Methods', in *IEEE Transactions on Computational Intelligence and AI in Games*, 2012, vol. 4, no. 1, pp. 1-43.

Appendix A

Table A.1: Parameter choices for Monte Carlo Tree Search.

Parameter	Expert	Self-play agent
Iterations	10000	300
Exploration Constant c_b	0.4	0.4
Cutoff (n turns)	8	8
NN Policy Weight w_a	N/A	N/A
NN Value Weight w_v	N/A	N/A

Table A.2: Hyperparameter choices for the neural network.

Parameter	NN
Learning Rate	0.01
Batch Size	100
Early Stop Patience	30

Table A.3: Table showing the results from each tournament, 240 games/tournament and the win ratio for the 5000 simulation MCTS.

Tournament	Result	Win Ratio (%)
5000 vs 100	176-64	73.3
5000 vs 500	135-105	56.2
5000 vs 1000	133-107	55.4
5000 vs 1500	130-110	54.2
5000 vs 2000	128-112	53.3

Table A.4: Table showing the early game tree statistics, mean and standard deviation. Number of player switches represents how many times the turn is switched to the other player in the deepest path of the search tree.

Nr of Simulations	Tree Depth	Nr of Decisions	Nr of Player Switches	Time (s)
100	4.67 (0.47)	1.5 (0.0)	0.33 (0.47)	1.59 (0.05)
500	6.5 (0.76)	2.17 (0.47)	0.33 (0.47)	8.08 (0.11)
1000	7.17 (0.37)	2.58 (0.45)	0.5 (0.5)	16.15 (0.31)
1500	7.5 (0.5)	2.92 (0.45)	0.67 (0.47)	24.89 (0.9)
2000	7.83 (0.69)	2.83 (0.24)	0.33 (0.47)	32.44 (0.5)
5000	9.83 (0.37)	4.0 (0.41)	1.0 (0.0)	82.98 (1.77)
10000	11.0 (0.82)	4.33 (0.62)	1.0 (0.0)	173.69 (1.27)

Table A.5: Table showing the mid game tree statistics, mean and standard deviation. Number of player switches represents how many times the turn is switched to the other player in the deepest path of the search tree.

Nr of Simulations	Tree Depth	Nr of Decisions	Nr of Player Switches	Time (s)
100	5.67 (0.94)	1.75 (0.25)	0.33 (0.47)	1.6 (0.05)
500	8.0 (0.82)	3.0 (0.41)	0.5 (0.5)	8.03 (0.12)
1000	9.0 (1.0)	3.17 (0.24)	0.33 (0.47)	16.54 (0.78)
1500	9.0 (1.0)	3.75 (0.56)	0.83 (0.69)	24.5 (0.65)
2000	9.83 (1.07)	3.67 (0.55)	0.67 (0.75)	32.68 (0.53)
5000	11.0 (1.0)	4.33 (0.47)	0.83 (0.69)	82.79 (1.19)
10000	17.67 (4.15)	6.0 (1.38)	0.33 (0.47)	188.05 (2.04)

Table A.6: Table showing the late game tree statistics, mean and standard deviation. Number of player switches represents how many times the turn is switched to the other player in the deepest path of the search tree.

Nr of Simulations	Tree Depth	Nr of Decisions	Nr of Player Switches	Time (s)
100	6.2 (1.17)	2.2 (0.51)	0.2 (0.4)	1.59 (0.05)
500	8.6 (1.02)	3.6 (0.8)	0.6 (0.8)	7.97 (0.15)
1000	9.6 (1.36)	3.4 (0.73)	0.4 (0.49)	16.06 (0.6)
1500	10.4 (1.36)	4.1 (1.02)	0.6 (0.49)	24.51 (0.96)
2000	11.0 (1.1)	4.5 (1.26)	0.6 (0.49)	32.37 (0.91)
5000	12.0 (1.26)	4.6 (1.24)	0.4 (0.49)	81.41 (1.53)
10000	17.2 (2.04)	6.3 (0.87)	0.4 (0.8)	165.44 (11.63)