Monte Carlo Tree Search for Risk

Christoffer Limér, Erik Kalmér, Mika Cohen
Kungshamra 3/1317
SWEDEN

limer@kth.se
ekalmer@kth.se
mika.cohen@foi.se

ABSTRACT

The idea of using artificial intelligence to evaluate military strategies is relevant for a large number of governments today. With programs like Alpha Zero beating world champions in games of ever-increasing complexity, military adaptations are probably not far away, if they are not in use already. Part of these programs' recent success is due to a heuristic search algorithm called Monte Carlo Tree Search. In this project, we explored the possibility of using this algorithm to build a program capable of playing the strategy board game of Risk at a high level. The complexity and stochastic dynamic of the game demanded the use of chance nodes and aggressive gameplay limitations, known as action-pruning. By changing the conditions and game environment of the algorithm, we observed performance differences, mainly in that simulation length improved performance. We suggest that the created program, optimized with correct algorithm parameters, has the potential of playing Risk at ahigh skill level.

1.0 INTRODUCTION

Military strategies and tactics have long been evaluated and tested through the use of board games. One of the early examples being the Greeks playing a board game of war named Petteia in the 5th century BC. In the mid-17thcentury, Chess enthusiasts in Prussia started to develop more complex strategy games based on Chess, which ultimately led to what is now known as the grandfather of all modern military war games, Kriegsspiel. During the interwar period, interest in war games peaks, specifically in Germany, where the Treaty of Versailles restricted the use of actual military exercises [1]. The use of war games continued throughout the 20th century, a late example being the Pentagon's use of the board game Gulf Strike during the Gulf War [2]. With the introduction of electronic computers and the concept of artificial intelligence (AI) in the mid-20th century, the prospect of using computers as opponents or aid in strategy games seemed increasingly plausible. By the 1960s, a program had been produced that could play automated Chess against a human opponent. However, it would take until 1997 before a Chess program officially beat the world champion, Garry Kasparov [3]. Despite the advances in Chess-programs, more advanced strategy games have proven problematic to conquer. In March 2016, the computer program AlphaGo sparked headlines across the globe by beating the professional Go player Lee Sedol in a series of Go matches [4]. Such a program was thought by some experts to be a decade away [5]. Go is challenging in part due to the enormous amount of moves that must be considered. In computer programs, these are typically constructed as trees representing sequences of moves that proliferate with depth. These trees then have to be searched for a favorable move. One effective search algorithm used in the AlphaGo program is Monte Carlo Tree Search (MCTS) [6]. With the promising results of using this algorithm, there search objective of this project was to explore how well an MCTS-agent could play the strategy board game Risk.

1.1 Why Risk?

Risk is a turn-based strategy game of military conquest. The board depicts a political map of Earth divided into territories that two to six players fight for control over. Armies are gained and used to attack and capture adjacent enemy territories until only one player remains. Conflict is resolved using die rolls, meaning that unlike the game Chess, some degree of luck dictates the outcome. This random element makes for an exciting dynamic that needs to be taken into consideration when developing the MCTS-agent. Since its release in the late 1950s, Risk has remained one of the world's most popular strategic board games and has considerable renown in the strategy board game community. In 1995 the game was inducted into Academy of Adventure Gaming Arts and Design Hall of Fame, confirming its significance. Since the late 1980s, computer and video game versions have been released with various rules. The AI in those games has mostly been underwhelming, never achieving a skilled human level of play. The use of MCTS to improve this ability seems to be uncharted territory. With its essential role in its genre, a fascinating mix of strategy and risk-taking, Risk makes for a sound choice.

1.2 Research topic

How well can a program using the Monte Carlo Tree Search-algorithm, play the strategic board game of Risk? Using some simplifying limitations to the game, the goal of this project is to explore how well an MCTS-agent could perform in the game of Risk. We use a set of experiments to measure the performance of different implementations of the MCTS-algorithm, to see how they compare and which parameters that has the largest effect.

2.0 RISK RULESET

Throughout its existence, several rule changes and adaptations have been used to simplify or change the game dynamic. For this project, we have chosen to use the original rules with some common adaptations. A game with three or more players was deemed unnecessarily complicated, and an adversarial approach of two players was chosen. The ruleset for a two-player match differs from a regular game in that a neutral player is present with armies of its own, but this player can only defend, not attack. We use the simplified rules of no secret mission cards and" blitz-attacks," where each attack is simulated to the end as is standard on the mobile and PC versions of the game. In order to implement the MCTS-algorithm more efficiently, the reinforcement card type that is given for successfully acquiring a new territory is not hidden from the opponent. An example of an empty game board is shown in Figure 1.

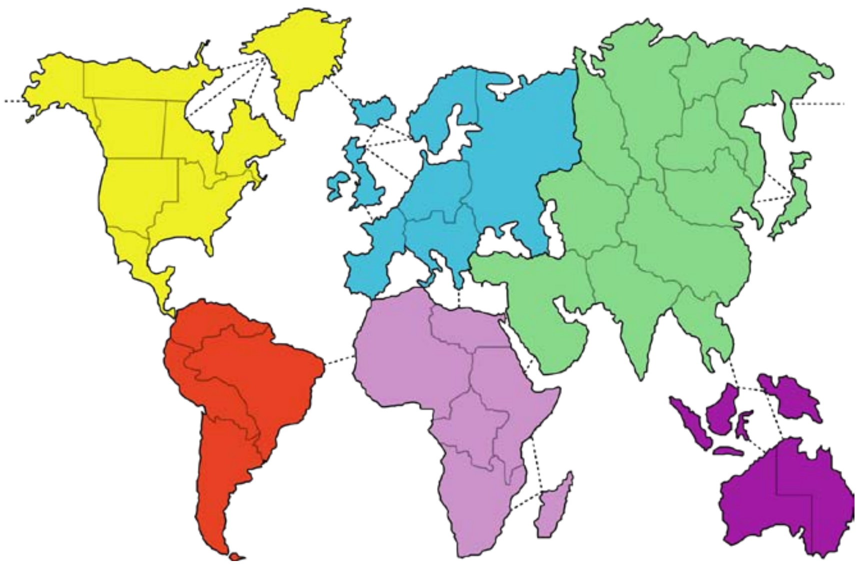


Figure 1 An empty game board of Risk

The game board depicts a world map of Earth divided into 42 territories, on six continents, that are color-coded. In our rule adaptation, each of the three players is given 14 territories, and 28 armies spread randomly across them. Players then take turns with three phases per turn, starting with the reinforcement phase. In this phase, players get to reinforce one territory with armies according to the number of territories and continents they control. If they possess a correct combination of unit cards, those could be exchanged for an additional amount of armies. Directly after the reinforcement choice, the opposing player chooses a neutral territory and places half the amount of armies the reinforcing player received. Next is the attacking phase. As long as a controlled territory borders an enemy territory and has at least two armies, it can attack, possibly resulting in the attacked territory changing owner. Depending on the ratio of armies between the territories, two to five dice may be needed. If a player manages to conquer at least one enemy territory during their turn, a unit card is earned, which can be exchanged in the next fortifying phase. The attacking phase can continue for as long as there are possible candidates, or the player decides they are done. Next is the fortifying phase, where the player may select one territory to reinforce from another, as long as they are connected through other controlled territories. If one player manages to conquer all the enemy player's territories (excluding the neutral player), it is declared the winner.

3.0 TREE GENERATION

In a board game, players typically take turns selecting moves in hopes of improving one's position and ultimately win the game. These sequences of moves can be constructed as trees in which one game state has branches leading to more states. Using the game of Tic-Tac-Toe as an example, we can illustrate how one part of the tree would look like playing as X shown in Figure 2

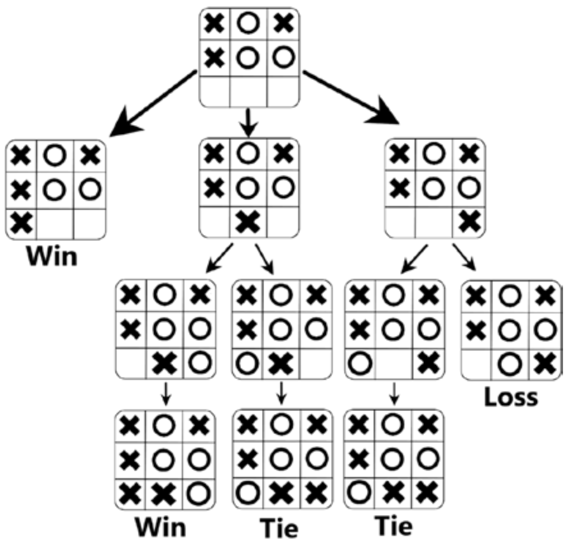


Figure 2 Example of tree construction in Tic-Tac-Toe

Each square represents a game state known in the tree as anode. These nodes are classified by their position in relation to each other much like a family tree. The top node is known as a parent node to the three nodes below it, and these three are known as child nodes to the one above it. Two of these nodes have children of their own and are thereby both parent-and child nodes depending on perspective. If a node does not have any child nodes, it is known as a leaf node, as it is the last state of a tree branch. In the example of Tic-Tac-Toe, generating a complete tree is achievable due to the low number of possible states and the fact that one can only play a specific move once. There are nine possible initial moves, which brings the maximum number of games to 9! (362880). Taking the games that take less than nine moves to resolve into account, that number drops to 255168, and this number is generally known as the game-tree complexity. With all games calculated, a perfect strategy resulting in a win or at least a tie is possible. However, many board games have much higher complexity, with the number of moves being practically infinite. If the tree generation is done as the game progresses, a decision on what node path to create and explore must be made. Choosing a move that results in victory several tree levels ahead when the opponent controls the moves at every other node level is not trivial.

4.0 MCTS THEORY

MCTS is a heuristic search algorithm that expands the tree with the use of simulations to estimate the lasting potential of a move. The algorithm is particularly useful at turn-based games with perfect information and no uncertainty in the game mechanics (full information of the opponent and all moves lead to an assured state), such as Chess, Go, Tic-Tac-Toe and Connect 4. The algorithm can, however, be applied to more complex games such as Risk with techniques explained in section V. When the algorithm first encounters a new node in the tree, it plays a simulated game from that state and then valuates the outcome. Throughout many simulations, nodes with a better outcome are weighted and are more likely to be explored and expanded, while weak performance nodes are abandoned. Generally, the MCTS-algorithm can be divided and explained in four phases, depicted in Figure 3.

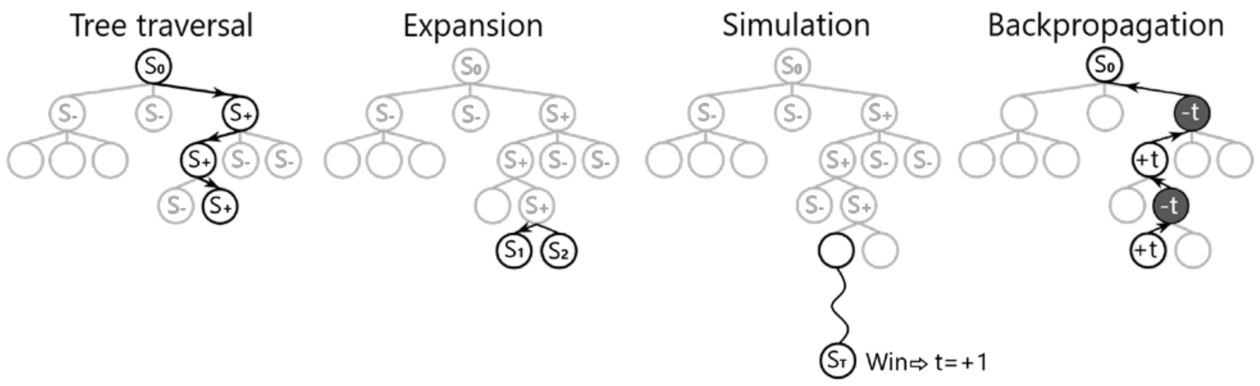


Figure 3: MCTS phase logic

4.1 Tree traversal

Tree traversal or the selection phase evaluates each child-node using the UCB1 function (Equation 1) and moves to the node with the highest value. This process continues until it reaches a leaf node and is described with pseudocode in Figure 4. In Figure 3, the highest valued child-state is noted as S_+ and the other as S_-.

Equation 1: UCB1-function

$$UCB1_i = \overline{V_i} + C\sqrt{\frac{\ln N}{n_i}}$$

- V_i = Empirical valuation mean (t_i/n_i)
- n_i = Number of rollouts (simulations) for node i
- t_i = Sum of all valuations of node i
- N = Total number of simulations for parent-node
- C = Constant (exploration parameter)

UCB1 has two terms, exploitation ($\overline{V}_i$), which is high for nodes with high valuations t (e.g. 1 for win and 0 for loss), and exploration ($C*\sqrt{\ln(N/n_i)}$), which is high for nodes with few visits. The exploration term is essential because Vi has a high variance at a low number of visits, and because nodes deeper in the tree generally have higher valuations *t*. C is essential to choose correctly because a too high value will prevent the MCTS from converging, and a too low value may cause convergence to the wrong node. The theoretical value of C is $\sqrt{2}$ [7] but is usually chosen empirically.

```
while node is not leaf node do
    for all child nodes of current node do
        calculate UCB1($S_i$);
    end
    traverse to child node with highest $UCB1$
end
```

Figure 4: Tree traversal

4.2 Rollout

If the leaf-node has not been visited, the MCTS simulates random moves from the current state (S) until it reaches a terminal state (S_T), which is then valuated as t; this process is called a rollout. Choosing random moves in the rollout will probably result in a reasonable valuation, but by simulating strategical moves have shown to significantly improve the MCTS performance [8]. Note that this requires knowledge of some strategies of the game.

4.3 Expansion

If a node is a leaf node and has been evaluated by a rollout, new child nodes are created for each available move of the current state (S). The new child nodes then update their states (S_i) based on the corresponding move *i*. After all states (S_i) have been updated, the MCTS moves to a random child (usually the first (S_1)). It can be useful to limit the number of available moves if they are unreasonable, to make the search more efficient; this is known as action-pruning.

4.4 Backpropagation

After a rollout, the MCTS backpropagates the valuation (*t*) through a chain of parent nodes until it reaches the original state (S_0), shown in Figure 3. In each parent node, the number of visits increases by one, and tis added to the node's total valuation. Note that t alter sign after every iteration, illustrated in Fig. 3, because it is the other player's turn and, therefore, evaluates that branch negatively. In simple games as Tic-Tac-Toe, the player switches after each move, but in more advanced games, a turn can be structured as a sequence of many moves (e.g., Risk). The Backpropagation process is explained with pseudocode in Figure 5, where the valuation (*t*) is inverted if the parent node is on the other player's turn (current player $\neq$ parent player).

```
**Data:** $t$ is the valuation of the leaf-node
**while** *node has a parent(not $S_0$)* **do**
    **if** *current player $\neq$ parent player* **then**
        t = -t;
    **end**
    node value += t;
    node visits += 1;
    node = parent node;
**end**
```

Figure 5: Backpropagation

4.5 Flowchart of MCTS

In order to illustrate the algorithm further, flowcharts were constructed and depicted in Figure 6 and Figure 7. One iteration is started and ended at the first state in the tree (S_0). The process proceeds until it reaches a determined number of iterations, where the moves of the first state (S_0) are studied. The child node with the highest number of visits is desired, and the corresponding move is then played in the real game.

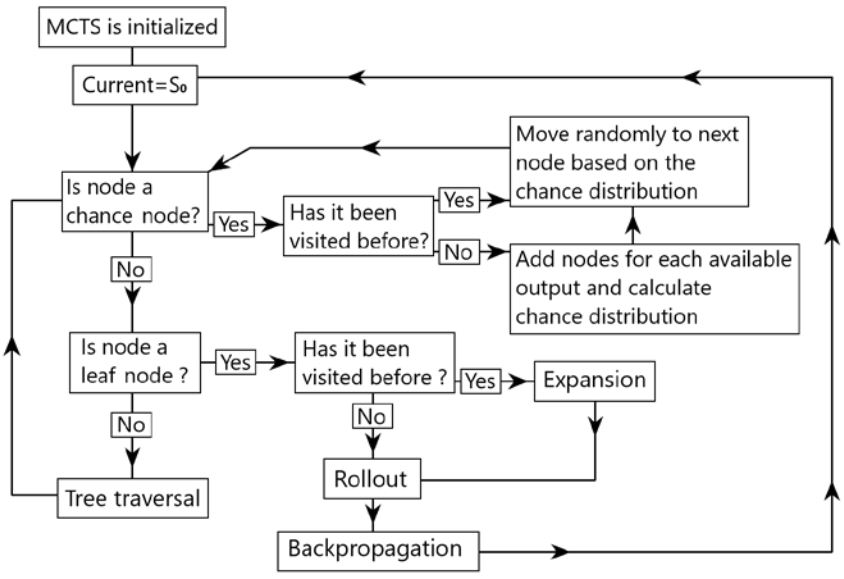

MCTS is initialized
Current=S₀
Is node a chance node?
Yes
No
Has it been visited before?
Yes
No
Move randomly to next node based on the chance distribution
Add nodes for each available output and calculate chance distribution
Is node a leaf node ?
Yes
No
Has it been visited before ?
Yes
No
Expansion
Tree traversal
Rollout
Backpropagation

Figure 6 MCTS flowchart

For nondeterministic games, implementing chance nodes is a possible solution for the MCTS. Unlike regular nodes that use the UCB1-formula, chance nodes traverse to a child node with a distinct probability distribution. For a die roll, the probability is uniform, but chance nodes can adopt any discrete distribution. The flowchart is then updated to accommodate for the chance nodes shown in Figure 7.

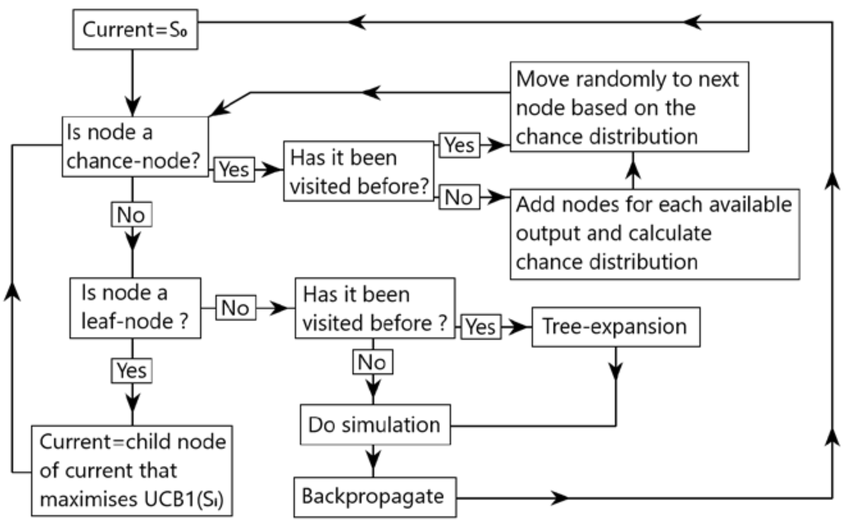

Current=S₀
Is node a chance-node?
Yes
No
Has it been visited before?
Yes
No
Move randomly to next node based on the chance distribution
Add nodes for each available output and calculate chance distribution
Is node a leaf-node ?
No
Yes
Has it been visited before ?
Yes
No
Tree-expansion
Do simulation
Current=child node of current that maximises UCB1(Sᵢ)
Backpropagate

Figure 7 MCTS flowchart with chance-nodes

5.0 MCTS FOR RISK

Risk is a complex game with elements of chance, many steps in each turn, and with many possible moves per step. it is beneficial to know the number of possible moves and outcomes after a turn, known as the branching factor. For Connect 4, the branching factor is the number of non-full columns (1-7), but in multi-step games, it grows exponentially for each step.

5.1 Branching factor for Risk

For Risk, the branching factor depends on which step it is currently on, fortify, attack, or mobilize, where the attack- and mobilize steps are also highly dependent on the board state. Itis, therefore, difficult to estimate the branching factor as one value, or even as three separate values for each step. Note that the attacking- and mobilize step can be played multiple times in a row, which result in a higher game-tree complexity. The game-tree complexity is defined as the combinations of all possible moves throughout the whole game; in other words, the number of ways the game can be played. Examples for other games are shown in Table 1. With all steps and chance nodes taken into account, our estimate puts Risk at a larger game-tree complexity than that of Go, but at the very least of comparable size.

Table 1 Combinatory for games 18[9]

Game/system	Game-tree complexity (as log to base 10)	Average game length	Branching factor
Chess	123	70	35
GO	360	150	250
Tic-Tac-Toe	5	9	9

Despite MCTS, not needing to explore the whole game-tree to converge, the current game-tree complexity is too large for MCTS to explore any reasonable strategy. It is unclear what the highest tolerated branching factor is to create a reasonable MCTS, but it can be reduced significantly with action-pruning. Action-pruning is the idea of limiting the available moves that the MCTS can do, usually, only irrelevant moves, but sometimes even reasonable moves have to be pruned to reduce the branching factor.

5.2 Action Pruning for Risk

In this project, we study the effect on the MCTS with the following Action pruning.

5.2.1 Fortify all

In the fortifying phase, it is common for high-level players to fortify all units at one territory; this is in part because there is a statistical advantage in attacking with many units from the same territory than once from multiple territories.

5.2.2 Advance all units after an attack

Advance all units after an attack: After a successful attack, the player chooses how many units to advance to the captured territory. If there are equal or less than four units left from the attacking territory, all but one has to advance; otherwise, there are (u_i-1) possible moves. The branching from” Advancing troops” grows by ($\prod N i = (u_i-3)$). This can be reduced significantly by limiting how many choices of units that can advance. By forcing the MCTS to advance all units, the branching reduces to one but can also limit the MCTS from establishing strategies. Three magnitudes of this action-pruning are shown in Table 2, with one, two, and three number of possible moves.

Table 2 Limitations after an attack

Limitations	Number of moves	Branching effect
Advance all	1	1
Advance all or 3	2	2^N
Advance all, 2/3 or 3	3	3^N

Where N is the number of attacks in a turn. We have chosen to use the middle alternative (advance all or three units, where 3 is the least units that the player can advance).

5.2.3 Mobilize all

When mobilizing from a territory, there are (u_i-1) number of moves but can be limited, similar to” Advancing troops.” For our Simulations in section VII, we use the limitation to mobilize all, but less restrictive limitations can also be used, as shown in Table 3.

Table 3 Limitations when mobilizing

Limitations	Number of moves	Branching effect
Mobilize all	1	1
Mobilize all or half	2	2
Mobilize all, 2/3 or all	3	3

5.2.4 Limit available territories when fortifying and mobilizing

In the fortifying-and mobilizing-phase, it is sensible to reenforce territory-fronts (territories adjacent to an enemy), with the main exception when defending a continent. Utilizing this to reduce branching, the MCTS can only mobilize to-and to fortify territories that are adjacent to an enemy, or territories that can block continents (with territory index [0, 2, 8, 11, 15, 20, 21, 23, 35, 38], as seen in Figure 13).

5.2.5 Only attack with a statistical advantage

When attacking a territory, it is simulated until one army is defeated. The probability of a prominent outcome is determined by the probability distribution of all outcomes of the attack, as shown in Figure 8. Therefore, we can limit the MCTS only to consider moves where the probability of a positive outcome is larger than a negative outcome. A neutral outcome is when both players lose the same number of units. We have although chosen not to implement this in the final MCTS-algorithm because we think it limits the MCTS-agents from finding hidden strategies.

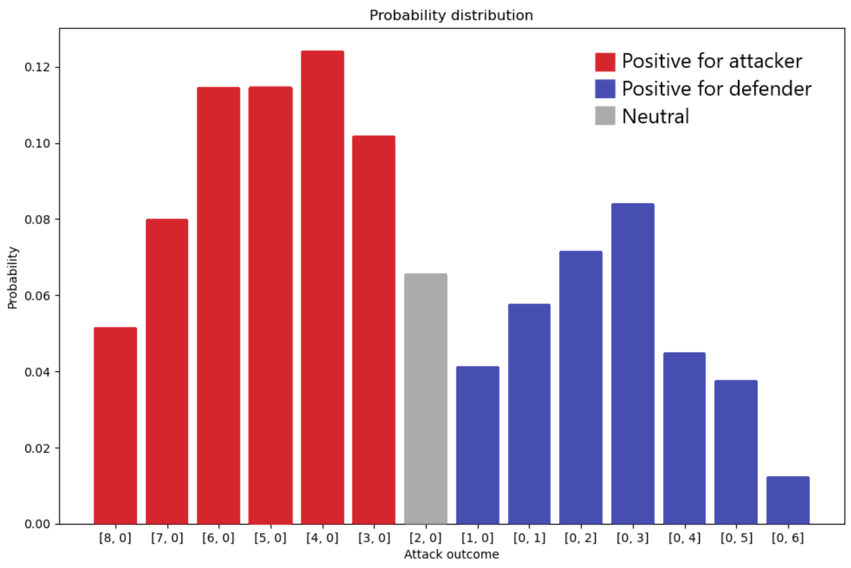


Figure 8: Probability outcome of an 8 vs 6 attack

5.3 Chance-nodes

Chance-nodes are used when attacking as the different outputs shown in Figure 8 and for the cards shown in Figure 9. When dealt a card after a turn, there is a uniform distribution of the tree cards, Infantry, Cavalry and, Artillery.

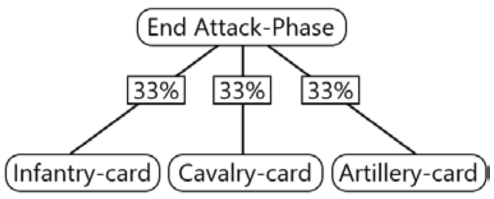


Figure 9 Chance-node for cards

Chance nodes traverse to a child based on a defined distribution; this is problematic if the number of child nodes (outcomes) are high because it prevents the MCTS from converging. After an attack, there are N_a+N_d (number of attacking and defending units) number of child nodes but can be reduced using output-clustering, where the number of attacking and defending units is Na and Nd. Output-clustering is the idea of merging similar outputs, e.g., winning: [4-0], [5-0], and [6-0] is all estimated as winning [5-0], shown in Figure 10.

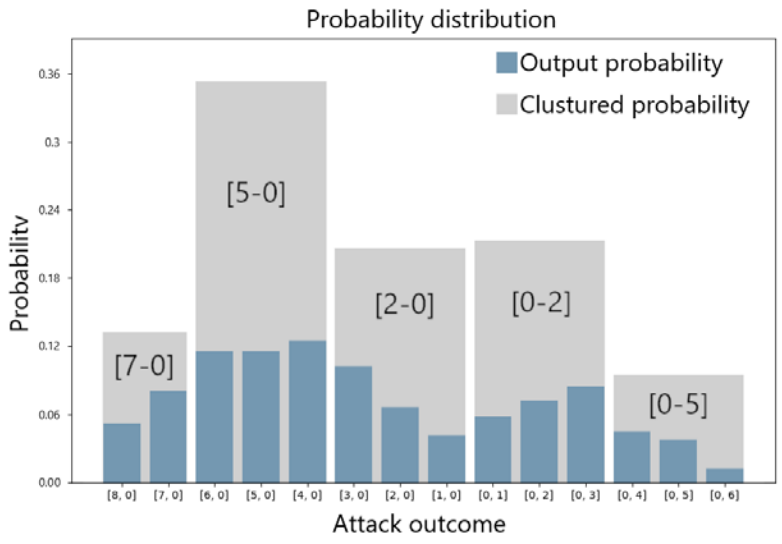


Figure 10: Chance-node clustering for the 8 vs 6 attack

5.4 Rollout and valuation t

Valuating a node is done with a rollout, where random moves are played until we can evaluate the resulting state. For Connect-4, we could not evaluate a state (game board); instead, we simulated to the end and returned the valuation as 1 for a win, -1 for a loss, and 0 for a tie. However, on Risk, we can approximate a state's valuation as the number of units you control relative to the other player's units. Instead of simulating random moves to the end, we only simulate a specific number of turns, which we refer to as the cut-off; before making the approximated valuation. When studying the variance of the valuation with different cut-off, we see a linear correlation between variance and cut-off shown in Figure 11, which is reasonable since it determines how many random moves are played before the valuation.

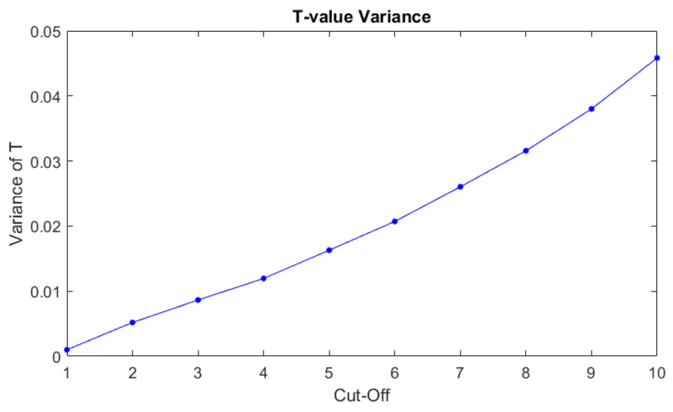


Figure 11: Variance of 80 000 simulated valuations *t* with different *cut-off*.

5.5 UCB1-constant

The UCB1-constant C determines how the MCTS-algorithm traverses in the tree, and it is important to choose correctly. If C is too large, the tree expands more uniformly and prevents the MCTS from converging, but with a too-small C, the MCTS may converge to the wrong move, shown in Figure 12.

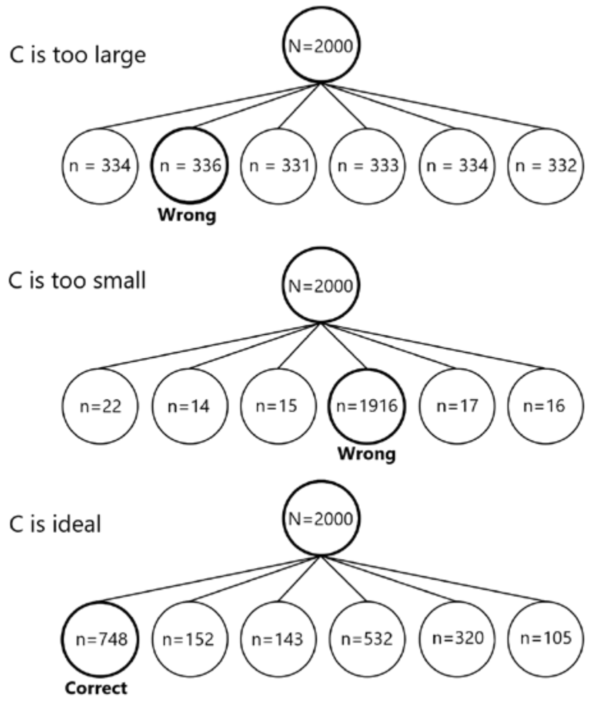
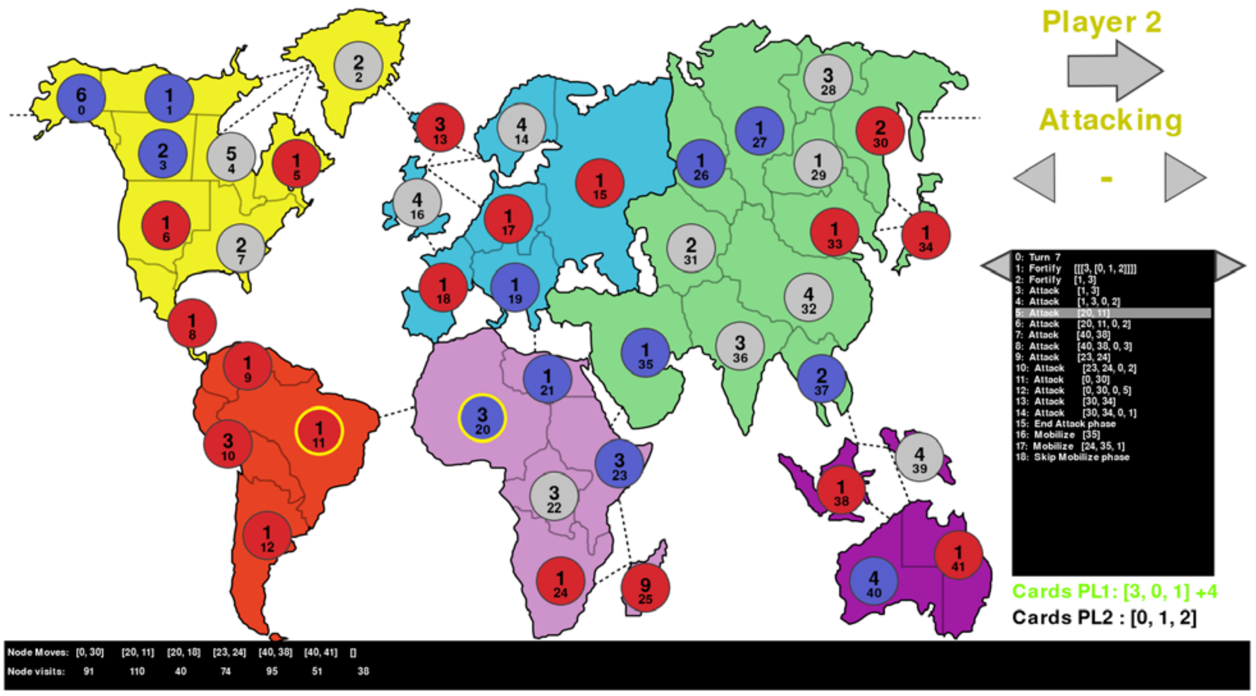


Figure 12: Illustration of how the UCB1-constant affects the tree traversal

The C-constant counteract state-valuations (t) with high variance because it reduces the effect of an incorrect valuation. In our case for Risk, the C-constant was chosen empirically to 0.5.

6.0 USER INTERFACE

Using the picture shown in Figure 13, we can exemplify a post-match review using our user interface of two MCTS-agents playing the game. For every territory on the game board, there is a filled circle consisting of a color and two numbers. Red color represents player one while blue represents player two; the neutral player is present in white. The top number shows how many armies are present, and the lower number shows the territory index. The current player is shown in the upper right corner, together with the current action phase, just below. Further down is a black box containing lines of text, where each line represents a move made by the agent. In the picture, we are looking at an attack from territory twenty on territory eleven by player one, as shown by the yellow frame around the engaging circles. Highlighting the move in the box shows another black box along the bottom of the picture. This box shows the nodes the agent has considered combined with the number of visits (more means better move potential) per node. In this example, the node furthest to the left has the highest number of visits resulting in the chosen attack, while the middle node (an attack from twenty to twenty-one) was considered the worst alternative. The node furthest to the right with empty brackets represents the choice of not attacking at all and continue to the next phase. In the bottom right corner, each player has a line showing their current unit cards in order of infantry, cavalry, and artillery, with the green number being the additional number of armies earned if traded in. The user interface considerably helps in analyzing the agent’s choices post-game and can also be used to play as a human against an agent directly.



```
0:  Turn 7
1:  Fortify    [[[3, [0, 1, 2]]]]
2:  Fortify    [1, 3]
3:  Attack     [1, 3]
4:  Attack     [1, 3, 0, 2]
5:  Attack     [20, 11]
6:  Attack     [20, 11, 0, 2]
7:  Attack     [40, 38]
8:  Attack     [40, 38, 0, 3]
9:  Attack     [23, 24]
10:  Attack     [23, 24, 0, 2]
11:  Attack     [0, 30]
12:  Attack     [0, 30, 0, 5]
13:  Attack     [30, 34]
14:  Attack     [30, 34, 0, 1]
15:  End Attack phase
16:  Mobilize  [35]
17:  Mobilize  [24, 35, 1]
18:  Skip Mobilize phase
```

```
Node Moves:  [0, 30]   [20, 11]   [20, 18]   [23, 24]   [40, 38]   [40, 41]   []
Node visits:    91        110        40         74         95         51       38
```

Figure 13: Risk UI

7.0 EXPERIMENT

This section explains the experiments that we have done. Our main focus is to determine how certain parameters affect the MCTS performance both on Connect-4 and Risk. The experiments consisted of the following:

For Connect-4 test how:

- The number of rollouts affects performance.
- The MCTS performs against human players.

For Risk test how:

- The number of rollouts affects performance.
- Cut-off affects performance.
- The best MCTS-agent performs against human players

To measure the performance of an MCTS-agent, we simulated multiple matches of two MCTS-agents with different parameters to see how they perform relative to each other. We will refer to this as" a set of matches." Example: an MCTS-agent with 500 rollouts plays 50 games against an MCTS-agent with 1000 rollouts. One test consists of multiple sets of matches, illustrated in Figure 14 where all agents play a set against all other agents.

7.1 Connect-4

7.1.1 Test how the number of rollouts affects performance

Test how the number of rollouts affects performance: To evaluate how the number of rollouts affects performance, we let 12 MCTS-agents play a set of 20 matches against one another. Both MCTS agents in each set begin half of the 20 games since it is favorable to start in Connect 4. To accomplish this, we need to simulate (144−12)/2 = 66 sets illustrated in Figure 14, resulting in 20∗66 = 1320 simulated games. We illustrate the result as the total win rate (wins/total played matches) of an agent shown in Figure 15.

	100	200	300	400	500	1000	2000	3000	4000	5000	10000	15000
100		set-1	set-2	set-3	set-4	set-5	set-6	set-7	set-8	set-9	set-10	set-11
200			set-12	set-13	set-14	set-15	set-16	set-17	set-18	set-19	set-20	set-21
300				set-22	set-23	set-24	set-25	set-26	set-27	set-28	set-29	set-30
400					set-31	set-32	set-33	set-34	set-35	set-36	set-37	set-38
500						set-39	set-40	set-41	set-42	set-43	set-44	set-45
1000							set-46	set-47	set-48	set-49	set-50	set-51
2000								set-52	set-53	set-54	set-55	set-56
3000									set-57	set-58	set-59	set-60
4000										set-61	set-62	set-63
5000											set-64	set-65
10000												set-66
15000												

Figure 14: Illustration of the simulation setup for Connect 4, the rows and columns corresponds to the number of rollouts of that MCTS-agent.

7.1.2 Test how the MCTS performs against human players

To measure how the MCTS stack up against human players, we let four people play a set of 10 matches against two different MCTS-agents with 500 and 1000 rollouts. Of the 10 matches in a set, the MCTS-agents start 5 of them, and the human player starts the other 5.

7.2 Risk

7.2.1 Test how the number of rollouts affects performance

To evaluate how the number of rollouts affects performance, we let four agents with a different number of rollouts play a set of 20 matches against one another, similar to the setup onConnect-4. In each set, 10 starting states are played twice, with both agents starting one of them; this is to reduce the effect of favorable starting positions. All other parameters are constant, including the cut-off of the agents. We tested this twice, once with cut-off 6 and once with cut-off 2, shown in Figure 16 and Figure 17.

7.2.2 Test how cut-off affects performance

To evaluate how the cut-off affects performance, we let four MCTS-agents with cut-off= [2, 4, 6, 8] play (16−4)/2 = 6 sets, with 40 starting states in each set, resulting in 2∗40∗6 = 480 simulated matches. We tested this with 500 rollouts, and the resulting win rate of the agents is shown in Fig. 15.We also tested how cut-off affect performance when the calculation time is constant (15 seconds), shown in Figure 19.This is because both the cut-off and number of rollouts are proportional to the total calculation time, so increasing the cut-off then decreases the number of rollouts on the same calculation time. We, therefore, need to find a balanced cut-off in order to optimize the MCTS.

7.2.3 Test how the best MCTS-agent performs against human players

To evaluate how the MCTS performs against human players we (the authors) played against tree MCTS-agents, with cut-off= 10 and rollouts = [500, 5000, 15000]. Each set against the MCTS-agents consist of 20 played matches with10 starting states, alternating the starting player every match. This data is very sparse but can give us some indication about the MCTS potential.

8.0 RESULTS

The results of the simulations explained in chapter 7.0 “experiment” is presented here. Starting with the performance difference between the number of rollouts for Connect 4, as presented in Figure 15.

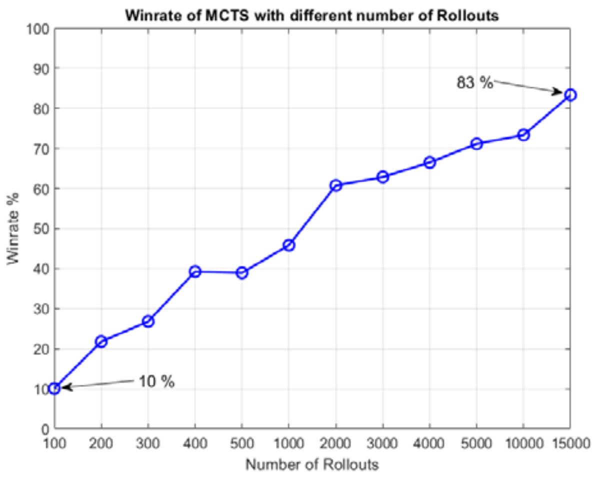


Figure 15: MCTS performance for Connect-4 with different amounts of rollouts.

The data show a correlation between the number of rollouts and the win rate. The agent with 100 rollouts having a mere 10% of its matches won while the agent with 15000 rollouts has an almost 83%-win rate. For agents playing Risk, shown in Figure 16, the correlation persists, but with less significance.

However, when we did the same study but reduced the cut-off to 2, no tendency toward correlation was seen. The win rate appears to be static at 50%, shown in Figure 17. Note: The cut-off only directly affects the valuation *t* that is used in the USB1-formula; this indicates that a low cut-off gives a bad valuation, which causes the MCTS-algorithm to converge to the wrong node. When we then tested how the cut-off affected performance, as seen in Figure 18, we saw that a higher cut-off correlates to a higher performance (win rate) on the interval cut-off= [2-8], further strengthening that conclusion. When we did the same test but kept the calculation time constant we can see a performance peek with cut-off=10, indicating that 10 is the optimal value for our MCTS-algorithm.

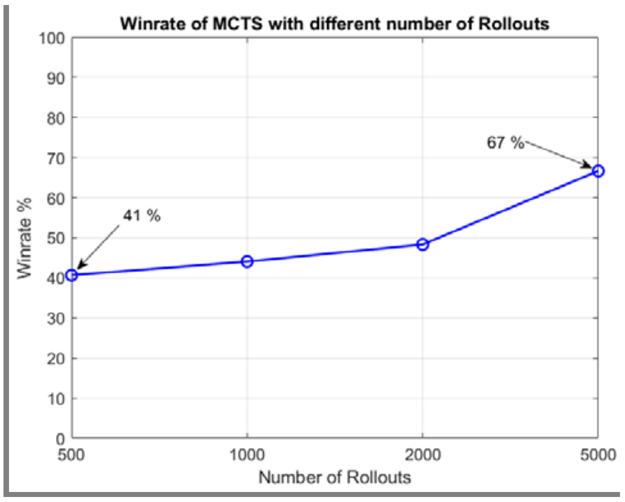


Figure 16: MCTS performance for Risk with different amount of rollouts. All agents have cut-off = 6.

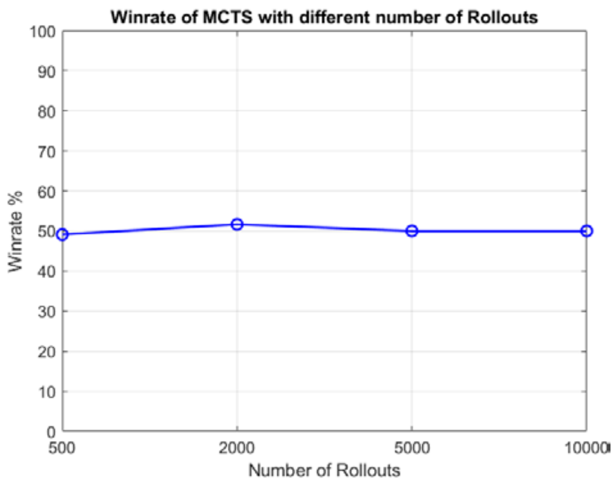


Figure 17: MCTS performance with different amounts of rollouts. All agents have cut-off = 2.

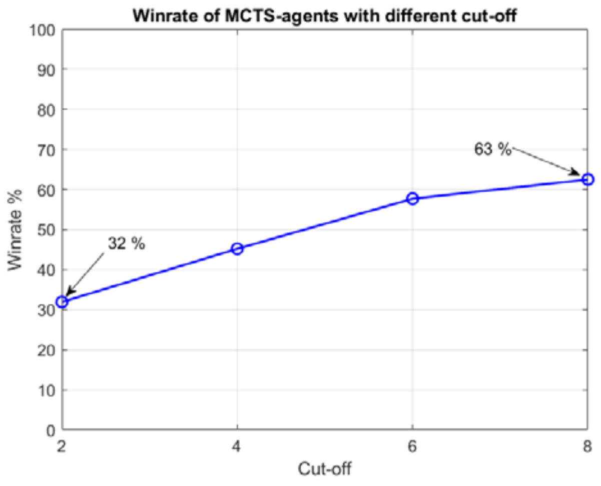


Figure 18: MCTS performance on Risk with different cut-offs. All agents have 500 rollouts.

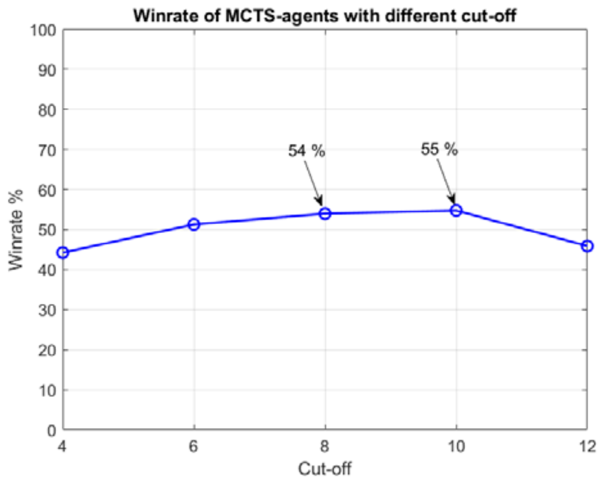


Figure 19: MCTS performance on Risk with different cut-offs. All agents have the same calculation time of 15 seconds.

8.1 MCTS performance against human players

When we tested how the MCTS performs against human player on Connect-4, the data indicated that the MCTS with only 1000 rollouts could outperform all the participants, shown in Table 4.

Table 4: Participants Win rate against the MCTS on Connect-4.

Rollouts / Participant	500 Win rate	1000 Win rate
1	30 %	20 %
2	50 %	30 %
3	30 %	10 %
4	60 %	40 %

For Risk we tested the performance of 3 MCTS agents against one participant (one of the authors), with the results shown in Table 5.

Table 5: Participant win rate against the MCTS on Risk

Rollouts	500	5000	10000
Win rate	90 %	70 %	40 %

The MCTS with 500 rollouts only won 10% of the matches while the MCTS with 15000 won 60%, outperforming the participant on that test sample.

9.0 DISCUSSION

The considerable correlation between the number of rollouts and the win rate for the simulations regarding Connect 4 (Figure 15) was expected. The interesting question would be if the data regarding Risk were comparable. Answering this would be harder than initially thought. Quantifying the performance of an MCTS-agent for Risk was not trivial, as the outcome of a single match has a great deal of luck involved. Therefore, even when making the optimal decisions in every game, a win rate of 100% is not guaranteed. To make the agent effective, careful choosing of the MCTS parameters was needed. One particularly interesting parameter was the cut-off choice. When we investigated if the cut-off had any impact on the win rate, the effect of using a high cut-off was substantial, as shown in Figure 18. In our initial experiment, we used the cut-off value of 2, which was believed to be the best value, due to the low variance, shown in Figure 11. However, it was the one that performed the worst. A new question emerged concerning the choice of cut-off, could the cut-off possibly be too low for the increased number of rollouts to take effect? To evaluate this, we ran another experiment with a cut-off of 6 between four agents. The result shown in Figure 16 confirms this new idea, that given a higher cut-off, the agent with more rollout performs better.

10.0 CONCLUSION

Sets of experiments have shown a correlation between roll-out and performance, and between cut-off and performance, where both rollout and cut-off are proportional to the total calculation-time. With the result from the experiments on Risk (Figure 16 and Figure 17), we can conclude that our MCTS needs a sufficiently large cut-off to fully utilize the effect of more rollouts. It is hard to quantify how well the MCTS agent can perform against human players without a large sample of played matches. The time to gather this data is considerable, so some concessions were made to at least get an indication of the agent’s ability. Out of the 20 games we played against the best MCTS, with 15000 rollouts shown in Table 5, 60 % were lost to the agent. Even though the data is sparse, this indicates that the potential of an MCTS-agent playing on a high level can be considered plausible.

11.0 FUTURE WORK

By testing the optimized MCTS agent against a larger group of human Risk players, a more definite conclusion as to the agent's skill level could be made. A next development step could be to implement the MCTS in an Expert Iteration (EXIT), where the MCTS is used to train a network that can make fast, reasonable decisions, and combined with the MCTS decision process, further improving the performance.

Possibilities for decision support:

- **Synthetic players in educational wargaming**

  Relieve people from roles and tasks in wargaming scenarios by letting AIs play those roles with equal skill. This enables smaller teams to play larger wargames dependent on more parallel decisions.

- **Evaluations in analytical wargaming (mainly for high-level AIs)**

  The AI can evaluate action alternatives and give insight into what those actions will lead to or why one action is significant over another.

  The AI can do a complete simulation of the scenario to evaluate how prominent your current setup is, or how likely you are to "win".

12.0 ACKNOWLEDGEMENT

The authors would like to thank the Swedish Defence Research Agency who provided excellent guidance, feedback, and engaging discussions throughout this project. The authors would also like to thank Farzad Kamrani, whose knowledge considerably helped in developing the agent.

REFERENCES

[1] Charles Homans .(2020,Apr.) Wargames: A short history. FOREIGN POLICY (FP), Washington, USA. [Online]. Available: https://foreignpolicy.com/2011/08/31/war-games-a-short-history/

[2] (2020, Apr.) Wargames and the 1991 iraq war. [Online]. Available: https://www.strategypage.com/wargames-handbook/chapter/9-7-iraq.aspx

[3] (2020, Apr.) A brief history of computer chess. [Online]. Available: https://thebestschools.org/magazine/brief-history-of-computer-chess/

[4] (2020, Apr.) Artificial intelligence: Google's alpha go beats go master lee se-dol. [Online]. Available: https://www.bbc.com/news/technology-35785875

[5] (2020, Apr.) Google deepmind computer alphago sweeps human champ in go matches. [Online]. Available: https://www.cbc.ca/news/technology/go-google-alphago-lee-sedol-deepmind-1.3488913

[6] D. Silver, A. , Huang, C. Maddison, and et al., "Mastering the game of Go with deep neural networks and tree search, "Nature, vol. 529, p.484–489, Jan. 2016.

[7] P. Auer, N. Cesa-Bianchi, and P. Fisher, "Finite-time Analysis of the Multiarmed Bandit Problem, "Machine learning, vol. 47, pp. 235–256,2002.

[8] Y. Wang and S. Gelly, “Modifications of uct and sequence-like simulations for monte-carlo go,” in2007 IEEE Symposium on Computational Intelligence and Games, 2007, pp. 175–182.

[9] L. V. Allis, Searching for Solutions in Games and Artificial Intelligence. CIP- Gegevens Koninklijke, 1994.