Monte-Carlo Search Techniques in the Modern Board Game Thurn and Taxis

Frederik Christiaan Schadd

Master Thesis DKE 09-29

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science of Artificial Intelligence
in the Department of Knowledge Engineering
of Maastricht University

Thesis committee:

dr.ir. J.W.H.M. Uiterwijk
dr. M.H.M. Winands
dr. F. Thuijsman
J.A.M. Nijssen, M.Sc.

Maastricht University
Department of Knowledge Engineering
Maastricht, The Netherlands
December 20, 2009

Preface

The report you are reading is my master thesis, performed at the Department of Knowledge Engineering of Maastricht University in the field of Artificial Intelligence. The main subject of the thesis is the investigation of Monte-Carlo techniques when applied to the modern board game Thurn and Taxis, which is a non-deterministic board game with imperfect information for 2 to 4 players.

Several people have made this thesis possible to whom I wish to express my gratitude. First and foremost, I wish to thank my supervisor, dr. ir. Jos Uiterwijk, for his constant support during the entire research and his lectures during my studies. I also wish to thank prof. dr. Jaap van den Herik, dr. Mark Winands and Jahn-Takeshi Saito, MSc., for their lectures during the course of *Intelligent Search Techniques*. Additionally, I wish to thank my brother Maarten Schadd, MSc., for introducing me to the game Thurn and Taxis and competing against my AI. And last but not least, I wish to thank you for reading my master thesis.

Frederik Schadd
Maastricht, December 2009

Abstract

Modern board games present a new and challenging field when researching search techniques in the field of Artificial Intelligence. These games differ to classic board games, such as chess, in that they can be non-deterministic, have imperfect information or more than two players. While tree-search approaches, such as alpha-beta pruning, have been quite successful in playing classic board games, by for instance defeating the then reigning world champion Gary Kasparov in Chess, these techniques are not as effective when applied to modern board games.

This thesis investigates the effectiveness of Monte-Carlo Tree Search when applied to a modern board game, for which the board game Thurn and Taxis was used. This is a non-deterministic modern board game with imperfect information that can be played with more than 2 players, and is hence suitable for research. First, the state-space and game-tree complexities of this game are computed, from which the conclusion can be drawn that the two-player version of the game has a complexity similar to the game Shogi. Several techniques are investigated in order to improve the sampling process, for instance by adding domain knowledge.

Given the results of the experiments, one can conclude that Monte-Carlo Tree Search gives a slight performance increase over standard Monte-Carlo search. In addition, the most effective improvements appeared to be the application of pseudo-random simulations and limiting simulation lengths, while other techniques have been shown to be less effective or even ineffective. Overall, when applying the best performing techniques, an AI with advanced playing strength has been created, such that further research is likely to push this performance to a strength of expert level.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Decision making is a challenging branch of research in the area of artificial intelligence (AI). Ever since the development of the computer, researchers strived to find ways that enable computers to make smart decisions when faced with complex problems. It is common to use simplified versions of existing problems in order to develop approaches that can solve these problems. An example is the traveling salesman problem [18], which is commonly used as benchmark for developed search or optimization techniques. For many of these problems, one can prove that they are NP-Complete [21]. Most of these problems, however, are not quickly solvable, or with current state-of-the-art technology not solvable at all. It is then necessary to apply methods that compute a solution that is as good as possible within computational time constraints. A field where these methods are vividly researched is the field of board games. Board games such as Chess or Go have a high complexity [23] such that computing an optimal response during a game is infeasible.

In this chapter, we will give a brief overview of the history of computer-games research in section 1.1. Next we will explore current developments in the field of modern board games in section 1.2. In section 1.3 we give the problem statement and the research questions of the thesis. Finally, in section 1.4 we will provide an overview of the thesis.

1.1 History of Computer-Games Research

Research in the field of game AI started in the mid 1940's with mostly theories how a computer might be able to play a game of chess. Alan Turing was the first to write a computer program that was able to play a full game [32], though it never actually did run on a computer but was executed manually. He described the principle of Minimax search, though this method can be traced back to a paper published in 1912 by Ernst Zermelo [26]. After this humble start, many researchers devoted their time to the development of search techniques, of which

a large proportion have been incorporated into chess-playing AIs. The most well known technique in this area is Alpha-Beta pruning [26]. Consequently, numerous other games have been investigated by researchers as well, which lead the development of other search techniques, such as Expectimax search [19, 26], Proof-Number search [33] and Monte-Carlo search techniques [11].

Eventually, advances in both techniques and computer hardware allowed computer programs to play at a very high level. In 1997 a long set milestone has been achieved when the chess computer Deep Blue defeated the then reigning world champion Gari Kasparov [9]. However, there are still challenges that remain, since the applied techniques do not work equally well for every game. For instance, the game of Go still remains a challenge due to its high complexity and the fact that it is difficult to create an evaluation function for this game.

1.2 Developments in Research on Modern Board Games

The field of modern board games unfortunately is not as heavily researched as the field of classic board games, let alone the game of chess. The game of Go has received notable attention from researchers. Though Go might not be a modern board game, the research performed in this field regarding Monte-Carlo (MC) search has lead to techniques which in turn can be applied to modern board games. Many of the current developments are not based on the standard Monte-Carlo Search, but rather on Monte-Carlo Tree Search (MCTS) [11]. This search technique combines Monte-Carlo sampling with the construction of a search tree, in order to gain the advantages of both techniques. Building on the success of MCTS, researchers investigated ways how to improve the sampling process. By combining elements of the multi-armed bandit problem [20], researchers were able to incorporate the exploration-exploitation aspect of this problem into the MCTS technique. This lead to the development of the popular UCT-search (Upper Confidence bounds applied to Trees) by Kocsis and Szepesvári [22]. In 2008, a computer player based on this approach managed for the first time ever to beat a world class Go player in a handicap match [13].

During the past years, many improvements have been developed for the UCT search and Monte-Carlo method in general. For instance, Chen and Zhang [15] assessed how adding domain knowledge to the sampling process can result in a much stronger AI. Chaslot et al. [12] described progressive strategies regarding how domain knowledge can also be added to the selection mechanism of the UCT algorithm. Using this approach, the UCT algorithm not only selects promising moves based on previous search results, but also based on domain knowledge. Their paper also assessed how progressive unpruning can help a MC player perform well under severe computational time constrains. For the game of Go there also exists a multi-player version, for which several algorithms

are applicable, including UCT [10]. However, an increasing amount of improvements for Monte-Carlo search methods also implies an increasing amount of parameters, such that the tuning of these parameters is crucial for the performance of the AI. While some parameters can be set using expert knowledge, the majority of the used parameters needs to be optimized using some Machine-Learning approach. However, calculating the fitness of a set of parameters in the field of game research is done by having the AI compete a set number of times and investigating how often the AI has won. This implies that learning approaches which utilize the gradient of an analytic expression cannot be used. Chaslot et al. [14] suggested applying the Cross-Entropy method in order to learn important parameters. Applying this method to their Go AI *Mango*, they observed that this approach indeed increased the performance of their AI.

The game of Poker has also received considerable attention. Poker is a non-deterministic multi-player game with imperfect information. While classic search techniques failed to competitively play Poker, it was shown that Monte-Carlo techniques did have potential to play Poker well [4]. In 2002, a strong Poker player was created by Billings et al. [5], though world-class level play has yet to be achieved. An important aspect which arises in games where estimating the imperfect information is crucial, like Poker, is opponent modeling. By modeling the behavior of the opponent, one can use this information to estimate the hidden information of the opponent. In Poker, this can be done by statistical weighting after each action [3]. With increasing amounts of stochastic elements in games, it becomes more and more difficult to evaluate whether a certain move was actually a good move in order to assess the strength of a player. This problem can be tackled by applying perfect-information hindsight analysis [2].

Apart from the previously mentioned games, unfortunately only few modern board games have been investigated. Using simulation-based move evaluations and a trained rack evaluator, Sheppard [28] has shown that it is possible to create a world-class Scrabble player. The principle of learning also received some attention in game research. The game of Backgammon can be played at a world-class level by self-teaching neural networks [31]. The board game Settlers of Catan is a modern board game, which has been approached in various ways in the past. Szita et al. [30] concluded that MCTS can be used to create a strong Settlers of Catan player. Reinforcement Learning can also be applied to learn basic strategies for this game [25], resulting in a performance that is comparable to a human player. Another approach which has been investigated in Settlers of Catan is a task-based multi-agent-based system. Utilizing this approach, Branca and Johansson [8] created an AI that demonstrated strategic game play promising enough to encourage further research.

1.3 Problem Statement and Research Questions

Monte-Carlo techniques have been shown to be quite successful when applied to certain classic board games, such as Go [15] and games with chance, such as Poker [5]. This thesis will investigate its application potential in the field of modern non-deterministic board games. Hence, the problem statement of this thesis is the following:

To what extent can Monte-Carlo techniques be applied in order to create a competitive AI for modern non-deterministic board games with imperfect information?

In order to give a conclusive answer, several research questions will be investigated. The first research question considers the game used for this research:

What is the game complexity of the game Thurn and Taxis?

To answer this question, the complexity of the game Thurn and Taxis needs to be computed or estimated.

The second research question is the following:

To what extent is it possible to use domain knowledge in order to improve the Monte-Carlo sampling process?

To answer this question, several techniques to incorporate game knowledge into the Monte-Carlo method will be investigated regarding their effectivity and efficiency.

The third research question of this thesis is:

To what extent does the addition of constructing a search tree contribute to the strength of a Monte-Carlo AI?

To answer this question, Monte-Carlo Tree Search will be investigated and its strength including several enhancements will be evaluated.

1.4 Overview of the thesis

The general outline of this thesis is structured as follows:

- Chapter 1: This chapter provides an introduction to the research field of artificial-intelligence techniques applied to classic and modern board games. Additionally, the problem statement and research questions will be given.

- Chapter 2: This chapter provides an introduction to the board game Thurn and Taxis and its historical origin.

- Chapter 3: The game of Thurn and Taxis will be analyzed in this chapter. Both the state-space complexity and the game-tree complexity will be calculated as accurate as possible and the resulting values are compared to the complexities of other games.

- Chapter 4: This chapter describes the applied search techniques and possible improvements.

- Chapter 5: In this chapter, the results of the performed experiments will be presented and evaluated.

- Chapter 6: This chapter presents the conclusion of this research and answers the previously stated research questions and problem statement. Additionally, several suggestions for future research will be given.

Chapter 2

Thurn and Taxis

In this chapter, we will provide a summary of the origin of the game Thurn und Taxis in subsection 2.1 and provide an explanation of the rules of the game in subsection 2.2.

2.1 Origin of the Game

Thurn and Taxis is a board game published by the German publisher Hans im Glück in 2006, which won the critics award 'Game of the Year' in 2006. The game is named and themed after the princely house of 'Thurn und Taxis', which is the known founder of the modern postal service. The rise of this house came when in 1490 a man named Francesco Tasso was tasked by Emperor Maximillian I. to create a steady postal route between Brussels and Innsbruck [1]. As gratitude, Francesco Tasso was granted the noble status of the German 'Reichsadel' in 1512, and was from then on known as Franz von Taxis. The continuous expansion of the postal network led to a remarkable reach across Western Europe, such that at the end of the 17th century the postal network of the then known 'Thurn und Taxis' family reached across 200,000 km^2. With the expansion of the postal network also followed an increase of social status of the family, causing the family to raise to the status of princedom, issued by Emperor Franz I. The fall of the Holy Roman Empire in 1806 also caused the cease of the 'Thurn und Taxis' postal system, however in 1815 during the Congress of Vienna the family was granted postal jurisdictions for some parts of Germany again. This lasted until 1867, when Reich Chancellor Bismarck initiated a transition, such that the 'Thurn und Taxis' postal service became a federal service in 1871, marking the end of the 'Thurn und Taxis' postal service.

The game board consists of a map featuring the general area of Bavaria, stretching from Austria and Switzerland in the south to the northern border of Bavaria, which can be seen in figure 2.1. All visible cities are connected to each other via roads. Left of the map is a display of six spots on which cards are placed. When a player picks a card, a new card is placed onto that position from a shuffled stack, giving the game non-deterministic traits. Also, the hand of a player is not visible to other players, thus making this a game with imperfect information. However, most of the missing information can be approximated by simply remembering the cards that another player picked up, a task which is easily performed by a computer. There is a possibility in the game however to draw a random card from the stack, meaning that a randomly drawn card is not known to any opponent. Up to 4 players can play this game. For this research we will restrict ourselves to 2 players in order to prevent unnecessary complications. In the course of the game, points can be earned in various ways and the player who has the most points in the end wins the game.

Figure 2.1: The board of the game Thurn and Taxis.

2.2 Rules of the Game

In this section, we will describe the rules of the game Thurn and Taxis.

2.2.1 Structure of a Turn

During his turn, a player needs to perform two mandatory actions and can do one optional action as well:

- A player **must** draw a card.
- A player **must** attach a card to his route. (see 2.2.4)
- A player **may** close his route and start a new one.

In addition to these actions, a player may choose one of four assistants, which each improve one of the other three actions. The official rules state that an assistant is chosen during the turn of the player, however due to design reasons this rule was altered such that a player may choose an assistant only at the beginning of his turn. The primary reason for this alteration is that it allows equal computation time for assessing the feasibility of each assistant. However, the alteration of this rule does not substantially change the game play, since, regarding the outcome of a turn, there is no difference between choosing an assistant at the moment he is needed or at the beginning of a turn.

2.2.2 Ending Conditions

The game has two ending conditions. Once one of these conditions is met by a player, each other player who did not play a move during the current round is allowed one last turn after which the game ends. One winning condition is the consumption of houses of a player. Each player has access to 20 houses, which he places on the board by closing routes. Once a player has used all his houses the game ends. The other winning condition is the carriage of a player. Each player has a carriage, which is essentially a status symbol of a player. When a player closes a route, it is possible to upgrade his current carriage to the next level, which in turn makes the carriage worth more points for the player. Once a carriage has reached its maximum level, the game ends as well.

2.2.3 Drawing Cards

Next to the game board, there is a special display on to which 6 cards are openly placed. At the beginning of his turn, after choosing an assistant, a player must either pick one of these cards from the display, or choose to draw a random card from the stack instead. If a player chooses to draw a card from the display, then this card is immediately replaced by another card from the stack. There are only 3 cards of each city in the game and once the stack is empty, the discard pile is shuffled and becomes the new stack.

2.2.4 Constructing Routes

The essence of the game is the construction and completion of routes, where a route is represented as a series of cards in a specific order. During the game, a player draws cards that all represent certain cities to fill up his hand, from which he can choose a card to attach to his current route. The route is laid openly in a sequential order on the table, so any player can see what a certain player is constructing at the moment. When a player wants to attach a card to his route, he must place the card to the left or right end of his current route, meaning he is not allowed to insert a card in between two already played cards. A player must also keep in mind the connections of the city to which he is attaching a new city card. Each city has a certain series of connections to other cities, which are all visible on the game board as roads. Hence, a player can only attach city A to city B, if there is a direct connection between A and B. It is also not allowed to use any city twice when constructing a route, prohibiting the construction of cycles and branching routes.

Let us view an example for route construction. Suppose a player has a route consisting of the three adjacent cards in figure 2.2 and wishes to connect the card Stuttgart to this route.

Figure 2.2: An example route of a player.

When inspecting the connections of the cities, depicted in figure 2.1, one can see that there is a road directly connecting Mannheim with Stuttgart. Hence the player would be allowed to attach this card to the card Mannheim. However, there is no direct connection between Freiburg and Stuttgart, meaning that the player would not be allowed to attach Stuttgart to Freiburg.

If there is no card present in a player's route, then he can simply play any card in order to start a new one. Also, if the player has no card in his hand that he can attach to his current route, he must scrap his route and start a new one. This means that all the cards in the current route are discarded and placed on the discard pile. The scrapping of a route does not allow the player to place houses to the board, nor does it allow an upgrade of the carriage. Hence it is important to avoid scrapping of one's own route as much as possible, in order to avoid wasting the played cards.

2.2.5 Assistants

There are four assistants in the game which a player can choose to enlist during his turn, with each assistant having a different function:

Postal Carrier: Instead of only adding one card to his route, a player may add two cards instead.

Postmaster: Instead of drawing one card, a player may draw two cards instead. After the first card is drawn, a new card is placed in its position before the second one is drawn. If a player has no cards on his hand at the beginning of his turn, he must enlist the assistance of the Postmaster.

Administrator: A player may remove the 6 cards, which are currently on the display, and replace them by 6 new random cards from the stack.

Cartwright: A player may upgrade his carriage, even if the length of his closed route is up to 2 cities short of the necessary length.

Enlisting an assistant is not mandatory, hence a player may opt not to enlist any assistant as well.

2.2.6 Closing Routes

Once a player has constructed a route of length 3 or higher, he may choose to close his route. Closing a route causes the removal of the current route, which is subsequently placed on the discard pile. It however enables the player to distribute houses on to the cities that were enclosed in the route. Special rules apply when selecting the cities which are to receive a house, making it difficult for a player to construct a route that allows each city within the route to receive a house. Hence, if the route in its entity does not fulfil these rules, the player cannot place a house on each city of his route and hence must make a strategic decision, choosing which of the cities receive a house and which do not. Every city in the game has a color, which corresponds with the colored region where the city is located in. The selection of cities must fulfil one of two criteria regarding their colors:

- The selected cities must all be of the same color.
- The selected cities must all be of a different color.

Once the houses are placed, the player checks if he is eligible for points, for which he was not eligible before. First, the player checks, whether he managed to put a house on each city of the predetermined colored areas. These areas consist of the multi-colored regions light and dark green, light and dark blue, orange and red and two mono-colored regions grey and purple. If this is the case then he receives points from the stack of the corresponding region. There also exists a colored region that does not have a corresponding point stack, which has the color black. The point stacks of the different regions contain points in decreasing order, meaning the player to first place a house on each city of a region will receive more points than the second player that does so. The second player receives more than the third player and so on until the stack is empty. A player can only receive points for completing a certain region once.

Other than for completing a region, a player also receives points if his finished route had a length of 5 or greater. For the lengths 5, 6 and 7 there are three different stacks and if a player completes a route of one of these three lengths, then he receives points from the corresponding stack. Unlike the region stacks, it is possible here to receive points from the same stack multiple times. If a stack is emptied and a player should receive points from the emptied stack, then he receives points from the next lower grade stack instead.

There are also two special ways to receive points. One is by causing an ending condition of the game. The player who does this receives a bonus point for doing so. The other way is by receiving points from a special stack. A player is eligible for points of this stack if he manages to put at least one house in all the differently colored regions of the board. Like the region stacks, a player can only receive points from this stack once.

2.2.7 Carriages

Each player acquires and upgrades a carriage during the course of the game. A carriage is worth points for the player and also serves as ending condition, if a player manages to upgrade his carriage to the highest possible level. There are five different levels of a carriage, with their costs ranging from 3 to 7. If a finished route of a player is longer than the cost of his current carriage, then he can upgrade his carriage to the next higher level, yielding a carriage that is worth more points for the player than his previous carriage. Each player starts the game with no carriage. Since each route must have a length of at least 3, the first route that a player completes in a game immediately yields him a carriage of cost 3. However, as the costs of the subsequent carriages increases, it becomes increasingly difficult to upgrade one's carriage. Figure 2.3 displays all the available carriage cards in ascending order. For each card, the required minimum route length is depicted in the top right corner and the amount of points which the card is worth is depicted in the lower left corner.

Figure 2.3: All the available carriage cards in ascending order.

2.2.8 End of the Game

Once the last turn has been played, all players count the points that they have received. The final score of a player p can be obtained in the following way, where c_p is the value of the carriage of player p, n_p is the total amount of points player p received from all the stacks in the game and h_p is the amount of houses that player p did not use:

$$score_p = c_p + n_p - h_p \tag{2.1}$$

The player with the highest score wins the game.

Chapter 3

Analysis of the Game

When performing research on a game, it is good to estimate the difficulty of the game by calculating its complexity as accurately as possible. The most common applied complexity measures are the state-space complexity and game-tree complexity. In section 3.1 the state-space complexity of the game Thurn and Taxis will be computed. The computation of the game-tree complexity will be presented in section 3.2. Finally, in section 3.3 the computed complexities of Thurn and Taxis will be compared to the complexities of other games.

3.1 State-Space Complexity

The state-space complexity of a game indicates the total amount of unique board positions possible for the game in question. If a game has a small state-space complexity, then over the course of multiple games it is more likely that a certain state is visited more than once. Encountering already visited game states gives the advantage that a player, be it human or computer, can use the gained knowledge of the previous visits to make better decisions. For instance, if at a position a certain move leads to a loss, then this knowledge can be used to, for instance, avoid this move for future revisits of this position. In the extreme case, if the state-space complexity is small enough, game knowledge or even a best response can be stored for each position of the game. However, for most classic and modern board games this is not possible due to their state-space complexities being too large.

Calculating the exact state-space complexity of the two-player variant of the game Thurn and Taxis is very complicated. Hence, an upper bound of the state-space complexity will be computed instead. The game of Thurn and Taxis can be separated into three components which determine the state-space complexity. First, there is the board itself encompassing all options to place houses on it. The second component consists of the cards and all possibilities in which these cards can be distributed amongst the players, the stack, the discard pile

and the display. Last, the point stacks and carriages need to be considered. However, the states of some point stacks are directly dependent on the board position of the houses. Hence, the calculation of the amount of possible states of these stacks is incorporated into the calculation of all possible house positions.

In a two-player game, a city on the board can have 4 different states. There can be no house on it, only a house of the first player, only a house of the second player or a house from both players. A quick estimate of all possible house positions would hence be $4^{22} = 1.7592 \times 10^{13}$, where 22 is the total amount of cities in the game. When wanting to incorporate the dependent point stacks into this calculation, it becomes necessary to separate this calculation according to the different dependent point stacks. For almost all house states of a given area on the board, there is only one possible state for its corresponding point stack. If none of the players is eligible for points of this area, then there is only one legal state of the points stack, being the state where all the points are still on the stack. Likewise, if only one player is eligible for points of this stack, then there is only one state possible as well, where the eligible player has received the highest amount of points from the stack with the remaining points still located on the stack. The only exception here is if more than one player are eligible for points of the stack, in which case either the first or second player could have received points from the stack first, which means that his opponent received less points from the stack than himself. Inspecting the board in figure 2.1, one can see the differently colored areas and their corresponding point stacks. There are 3 point stacks in the game which require houses on 3 cities, which can be of a different color, one stack which requires houses on 4 cities and one stack which requires as much as 8 houses. For each of these stacks, there is only one partial board position where the corresponding point stack can have different states, the position where both players have houses on all the necessary cities. Thus, the amount of possible house positions p_h including dependent point stacks can be computed as follows:

$$p_h = (4^3 + 1)^3 \times (4^4 + 1) \times (4^8 + 1) \times 4 = 1.8502 \times 10^{13} \tag{3.1}$$

Note that the multiplication by 4 in the end is necessary, since this represents the different states of the black region, which does not have a corresponding point stack and consists only of one city (Lodz). Also this calculation includes positions, where players have placed more than 20 houses on to the board, which is illegal. It is possible to adjust the calculated value by subtracting the amount of illegal house positions p_i from the calculated value. This value can be calculated in the following way. Let $H(x)$ denote the set of all possible house placing possibilities of one player on the entire board using exactly x houses, $Q(a, b)$ the amount of different positions possible with respect to the point stacks, given

the house placements a and b and let $E(n, S)$ be the n'th element of the set S, then p_i can be calculated as follows :

$$\begin{aligned} p_i = \quad & 2 \times \sum_{a=21}^{22} \sum_{b=0}^{|H(a)|} \sum_{x=0}^{22} \sum_{y=0}^{|H(x)|} Q(E(b, H(a), E(y, H(x))) - \\ & \sum_{\alpha=21}^{22} \sum_{\beta=0}^{|H(\alpha)|} \sum_{\gamma=21}^{22} \sum_{y=\delta}^{|H(\gamma)|} Q(E(\beta, H(\alpha), E(\delta, H(\gamma))) \\ = \quad & 276880008 \end{aligned} \tag{3.2}$$

The first term of this equation calculates all positions where at least one player has too many houses on the board. Multiplying this by 2 results in the total amount of illegal positions, where all positions where both players have too many houses on the board are counted double. Hence the second term is needed, which subtracts all positions where both players have an illegal position from the first term again, resulting in the total number of illegal positions. Subtracting the amount of illegal house positions from the amount of total possible positions, the new value for p_h becomes:

$$\begin{aligned} p_h &= (4^3 + 1)^3 \times (4^4 + 1) \times (4^8 + 1) \times 4 - p_i \\ &= 1.85017 \times 10^{13} \end{aligned} \tag{3.3}$$

Next, it is necessary to calculate the number of possibilities in which all the cards can be distributed among the game. There are 66 cards in the game of Thurn and Taxis, three for each city. These cards can be located on the stack, the discard pile, the display or in the hand or route of one of the players. There are however limits to these distribution possibilities. First, the display can only have 6 cards placed on it. Also a route of a player can only have at most one card per city in it. However, given a set of unique cards, it is possible for some sets of cards that there are more than one permutations of this set, which comprise a legal route. For instance, if there is a legal route with the cards A,B,C,D, then there is also a legal route D,C,B,A which is a mirror of that route. Also, given a set of cards, it is possible to form different routes which are not mirrors of each other. If a route A,B,C,D is possible then, due to the irregular connection between the cities, this does not mean that a route B,A,D,C is not possible. Hence, one should not inspect a set of cards and compute whether a route is possible, but rather compute all possibilities of a given set.

Utilizing a tree-search algorithm, for each set of cards the total amount of possible routes has been computed. Calculating the sum of all these values thus

yields the total amount of legal routes possible for a player, which is 23274077. One can calculate the amount of possible card distributions by inspecting each possible state of the display and calculating the amount of possibilities in which the remaining cards can be distributed amongst the stack, discard pile and the hands of the players, and multiplying this by the total amount of legal routes possible for each player. This calculation is an overestimation, since it does not consider the cards already in the routes of the players when calculating the remaining distribution possibilities. Ideally, one would want to traverse all combinations of displays and legal routes of the players and then for each of these combinations calculate how the remaining cards can be distributed amongst the rest of the game. This is unfortunately a very computationally expensive process which would require a massive amount of computational power in order to perform this calculation. However, the proposed overestimation should give a decent indication of what the real complexity of this component is.

The permutations of the cards, when distributing a set amount of cards over the remaining 4 locations, do not matter. A player can arrange the cards in his hand at will and the only moment when the discard pile is used is when the cards are reshuffled into the stack. Also, the state of the stack is hidden, since the cards are supposed to be drawn at random. The different amounts of possible distributions for each legal amount of cards of a single type over the remaining 4 locations in a two player game can be seen in table 3.1.

# Cards	Possibilities
0	1
1	4
2	10
3	20

Table 3.1: Amount of possible card distributions in the game for each possible amount of cards of the same type, excluding the display and the player's routes.

Let c denote a vector, where c_x is the amount of cards of type x which need to be distributed. Let D be the set of unique vectors of length 22,which is the total amount of cities in the game, where for each vector the sum of all elements inside this vector is exactly 60, and each element of this vector has an integer value larger than or equal to 0 and smaller than or equal to 3. These vectors represent all the different card combinations which still need to be placed, after 6 cards have been located on the display. Note that the permutation of the cards on the display does not matter, as it has no effect on the game. Thus $|D|$ denotes the total amount of states in which the display can be in. Also, let $W(y)$ denote the amount of unique possibilities in which y cards of the same type can be placed in the game, as seen in table 3.1 and let $E(n, S)$ be the n'th element of the set S. Then the total amount of possibilities p_c in which all cards

can be placed in the game for two players can be computed as follows:

$$p_c = 23274077^2 \times \sum_{d=1}^{|D|} \prod_{y=1}^{22} W(E_y(d, D)) = 7.9351 \times 10^{46} \tag{3.4}$$

Last, there are still point stacks and the carriages remaining, which can all have different states as well. The different states that a carriage can be in are fairly simple to deduct. There exist 5 different types of carriages, ranging from level 3 to 7. Also, there can be no carriage at all, which is the initial state of the carriage for each player. Hence, a carriage can have 6 different states, yielding 6^2 different states for a two-player game. Next, there are the point stacks which reward players for completing routes of certain lengths. There are different stacks for routes of lengths 5, 6 and 7, where a player can receive points from the stack for routes of length 7 at most four times, for 6 at most three times and for routes of length 5 only at most two times. Since a player can receive points from these stacks more than once during the game, the amount of different states a route stack can be in, where x is the maximum amount of times a player can receive points from this stack before it is empty, can be computed as follows:

$$\sum_{a=0}^{x} 2^a = 2^{x+1} - 1 \tag{3.5}$$

Taking into account the lengths of the three route-point stacks, the total number of different states that these three stacks can be in is:

$$(2^{2+1} - 1) \times (2^{3+1} - 1) \times (2^{4+1} - 1) = 3255 \tag{3.6}$$

The last two remaining point stacks, which need to be accounted for are the point stack for having a house in each colored area of the game and the point stack for causing the ending condition. For both of these stacks a player can only receive points from them at most once during the game. The stack containing points for having a house on all the different colored regions can have 5 different states. There is one state, where no player has received points and all the points are still located on the stack. Then there are two states where only one player has received points from this stack, one for each player. If points have been distributed from this stack two times, then there can only be two states as well, since a player can only receive points once. Hence the second player to receive points must be the opponent of the player who received points first, yielding a total of 5 states of this stack. Note that the state where first player one, then player two received points and the state where first player two, then player one

received points are different, since the player who first receives points from the stack gets more points than his opponent who subsequently does so. The stack for causing the ending condition has the additional limitation, that there is only one point to distribute. Hence, this point can either still be on the stack, or in the possession of player one or two, yielding a total of 3 different states.

Note that simply multiplying the state-space complexity by the amount of states that these last two stacks can be in is an overestimation, since this for instance counts states where a player received points for having houses in each colored region, while not actually having done so.

Now that the complexities of all the components have been determined, one can obtain a good estimate of the state-space complexity by multiplying these with each other.

$$\begin{aligned} p_h \times p_c \times 6^2 \times (2^{2+1} - 1) \times (2^{3+1} - 1) \times (2^{4+1} - 1) \times 5 \times 3 = \\ 1.85017 \times 10^{13} \times 7.9351 \times 10^{46} \times 36 \times 3255 \times 5 \times 3 = \\ 2.5805 \times 10^{66} \end{aligned} \tag{3.7}$$

Given this approximation of the state-space it becomes clear, that Thurn and Taxis has a considerable complexity.

3.2 Game-Tree Complexity

The game-tree complexity of a game is defined as the number of visited leaf nodes if one were to determine the value of the initial game position with a full-width tree search. This amount corresponds with the total amount of different games which can be played. In general, it is very hard to accurately estimate this complexity, let alone to specifically compute it. However, the general performed practice is to compute a rough estimate, by determining the average game length d and the average branching factor b of the game in question. A good estimate of the game-tree complexity of the game is then given by b^d.

For the game Thurn and Taxis, the average game length can be determined by playing a set of test games and record the game lengths of each game. Unlike other games however, the length of a game of Thurn and Taxis is proportionate with the playing strengths of the players. This is visible when inspecting the average game lengths of three different AIs engaged in self-play, displayed in figure 3.2. The random AI as well as the MC AI played 900 games in self play, where as the MCTS AI performed 700 self-play games.

AI	Simulation Technique	Average game length
Random	-	846.9
MC	Random	67.8
MCTS	Pseudo-Random	56.1

Table 3.2: Average game lengths for three different AIs, measured during self-play in 900 test games for the Random and MC AI and 700 test games for the MCTS AI.

First it is notable that games performed with randomly playing AIs have an extremely high average game length. This is due to the fact that it is possible to play Thurn and Taxis for an infinite amount of turns, if both players play bad enough. If, for instance, a player constantly scraps his own route, either deliberately or by being forced to due to bad card-drawing choices, then he will never close a route, and hence never cause the ending condition by either upgrading his carriage or placing all his houses on to the board. This peculiarity will be further discussed in subsection 4.4.1. An AI using standard MC -search has a significantly shorter average game length. Playing mainly routes of length 3, the AI plays in a very greedy way, looking for short-term point gains. This, however, leads to few carriage upgrades, causing the game to end via the ending condition of placing all one's houses on the board. This condition is in general harder to achieve than upgrading one's carriage to the maximum level. The AI using MCTS does not look only for short-term gains, playing longer routes in order to upgrade its own carriage and possibly receiving points for playing routes of length 5 or longer. Even though its goals are more long-term oriented, this playing-style allows the AI to finish the game earlier by upgrading the carriage to the maximum level.

Determining the average branching factor for the game of Thurn and Taxis is more complicated than for a classic board game, since here a turn of a player does not consist of a single decision, where one can simply record the average complexity of this decision, but rather of a series of decisions that a player has to make. These decisions, for instance the selection of an assistant, do often not seem complex to a player, but combined still create a vast amount of possibilities in which a turn can end. Figures 3.1, 3.2 and 3.3 display the frequencies of recorded branching factors for the performed self-play games of a random AI, MC-AI and MCTS-AI, respectively.

It is clear that Thurn and Taxis is game with a quite irregular branching factor. While positions with a branching factor ranging from 10 to as much as 1000 are the most common, positions with a branching factor well above that have been recorded occasionally as well. Also, the occurrences of measured branching factors spike whenever it is a multiple of 30 or 35. This can be attributed to the structure of the game. Since a player commonly has 5 assistant-selection

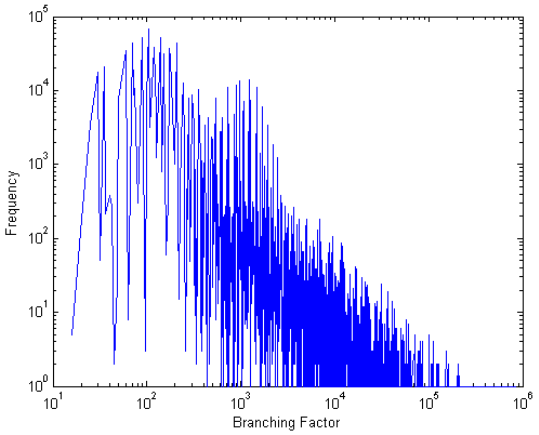

Figure 3.1: Frequency of branching factors recorded in 900 test games, played by a random AI in self-play.

options, consisting of the 4 assistants and the option of not choosing an assistant at all, and 7 card-selection options, consisting of the 6 cards on the display and the options of choosing a random card, then the total amount of different choices that the player can make in this turn will be a multitude of 35. Branching factors of a multitude of 30 are common as well, since it is a regular occurrence that a display contains two cards of the same type, yielding only 6 card selection options instead of 7. Branching factors which are not a multitude of 30 or 35 occur less frequent, since these require less common game states to occur, for instance a display containing only 3 different card types, or only having one assistant-selection option, due to the fact that the player has no cards in his hand. The complexity of a turn increases considerably, if the Postmaster is chosen, since then a player has to select a second card to draw, which again multiplies the amount of different outcomes of a turn by a factor of up to 7. After a player has done these selections, he must still attach a card to his route and choose whether or not to close his route. If his route is too small to be closed, then there usually are few options to choose from. However, the amount of available options quickly becomes larger if a player can close his route, considering the different house-placement options and the possibilities in which a player has to discard a set amount of cards from his hand. All these factors considered, the complete branching factor of a turn can get very large.

The different recorded average branching factors for the three AIs can be seen in table 3.3.

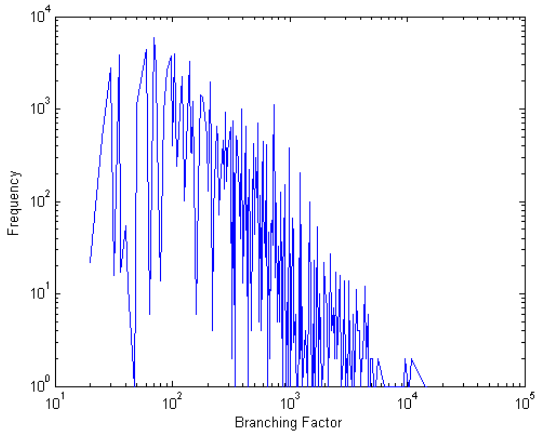

Figure 3.2: Frequency of branching factors recorded in 900 test games, played by a MC-AI in self-play.

AI	Simulation Technique	Average branching factor
Random	-	495.5
MC	Random	207.9
MCTS	Pseudo-Random	879.3

Table 3.3: Average branching factors for three different AIs, measured during self-play in 900 test games for the Random and MC AI and 700 test games for the MCTS AI.

It is notable that for each of the three AIs a considerable average branching factor has been measured during the test games. However, for the MC AI the average branching factor is actually significantly lower than for the other two AIs. This can be attributed to the playing-style of the AI, caused by the fact that it applied random simulations, which makes the AI play in a greedy way. Since the AI mostly closes its route at its shortest length, the AI is faced less frequently with positions where closing its route is a legal option. For instance, assuming that per turn only one card is added to a route each turn, then the MC AI will be faced with a position where it can close its route at most each third turn, since a route needs to be at least three cities long so it can be closed. The MCTS AI, however, prefers to play longer routes and hence does not always immediately close its route when presented with this possibility. Thus, it will have the option to close its route more frequently and hence be faced with positions which have a higher branching factor more frequently as well. The randomly playing AI is faced with a situation where it can close its route even less frequent. However, since it plays randomly, it can encounter very complex

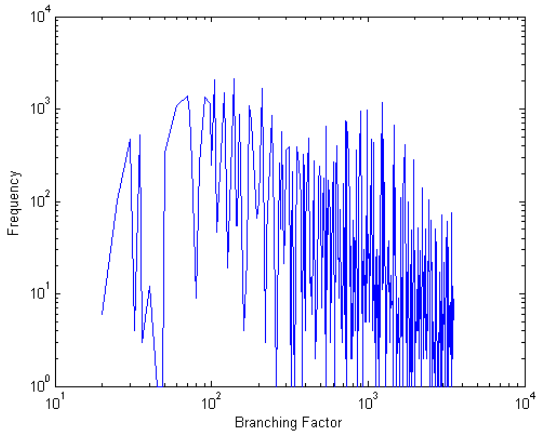

Figure 3.3: Frequency of branching factors recorded in 700 test games, played by a MCTS-AI in self-play.

situations which the other AIs avoid. This can be seen in figure 3.1, where positions with a low branching factor are the most common, but positions with an extremely high branching factor have been recorded as well. A more detailed analysis of the playing-styles of the different techniques can be viewed in sections 5.1 and 5.2.

Since MCTS is the strongest performing approach, its results are more likely to represent the actual decision complexity of the game. Hence, these values will be used for the estimation of the game-tree complexity of Thurn and Taxis. However, given the current available values, one can only estimate the game-tree complexity if Thurn and Taxis were a non-deterministic perfect information game. Hence, an additional measure is needed such that the estimation incorporates the elements of chance and hidden information as well. The component which makes Thurn and Taxis a non-deterministic game with imperfect information is the stack, due to the fact that its state is hidden. If one were to know the state of the stack, then one would know at each position which cards would be placed on to the display if a player draws a card or chooses the Administrator. One would also know the hidden information of the game, since if a player draws a random card from the stack and one knows the state of the stack, then one would essentially know which card the player receives. Thus, in order to determine the game-theoretical value of the root of the game, one would need to determine it for each state which the stack can have.

Thurn and Taxis contains 66 cards, 3 of each type. Thus, it is necessary to determine all the possible arrangements of these cards inside the stack. This is essentially a problem of determining all possible permutations with limited

repetition of each type. For the game of Thurn and Taxis, this amount can be calculated as follows.

$$\prod_{x=1}^{22} \binom{(22-x+1)\times 3}{3} = 4.1357 \times 10^{75} \tag{3.8}$$

Now that the amount of initial states which the stack can have is known, the game tree complexity of Thurn and Taxis can be estimated as follows.

$$4.1357 \times 10^{75} \times 879.3^{56.1} = 6.0623 \times 10^{240} \tag{3.9}$$

3.3 Comparison to other Games

In order to gain some perspective in how complex Thurn and Taxis is, one can compare its state-space complexity and game-tree complexity to the complexities of other games [34] [27] [24]. These can be viewed in table 3.4.

Given the complexities of the different games displayed in table 3.4, one can conclude that in terms of complexities the 2-player version of Thurn and Taxis is the most similar to the game Shogi. Given these dimensions, it is likely that the game-tree complexities of the 3- and 4-player versions of Thurn and Taxis lie in a range similar to the game Go. Note that the computed state-space complexity of Thurn and Taxis is an overestimation and that the true state-space complexity is likely a few powers of 10 smaller than the current overestimation. Since its complexities are vastly higher than the game of Chess one can conclude that it is not possible to solve the game Thurn and Taxis within the next decade.

Game	State-Space Complexity	Game-Tree Complexity
Awari	10^{12}	10^{32}
Checkers	10^{21}	10^{31}
Chess	10^{46}	10^{123}
Chinese Chess	10^{48}	10^{150}
Connect-Four	10^{14}	10^{21}
Dakon-6	10^{15}	10^{33}
Domineering (8x8)	10^{15}	10^{27}
Draughts	10^{30}	10^{54}
Fanorona	10^{21}	10^{46}
Go (19x19)	10^{172}	10^{360}
Go-Moku (15x15)	10^{105}	10^{70}
Hex (11x11)	10^{57}	10^{98}
Kalah(6,4)	10^{13}	10^{18}
Khet	10^{48}	10^{125}
Nine Men's Morris	10^{10}	10^{50}
Othello	10^{28}	10^{58}
Pentominoes	10^{12}	10^{18}
Qubic	10^{30}	10^{34}
Renju (15x15)	10^{105}	10^{70}
Shogi	10^{71}	10^{226}
Thurn and Taxis (2-Players)	10^{66}	10^{240}

Table 3.4: State-space complexities and game-tree complexities of various games and Thurn and Taxis.

Chapter 4

Search Techniques

In this chapter, we will explain the investigated search methods that have been applied. In section 4.1 the basic principle of MC search is explained. Section 4.2 will cover the MCTS approach and its advantages over standard MC search. The UCT enhancement will be explained in section 4.3. Finally, we will cover all heuristical enhancements and applications of domain knowledge in section 4.4.

4.1 Monte-Carlo Search

The Monte-Carlo method relies on extensive random sampling inside some domain. After the sampling process is completed, some sort of computation is performed on the set of samples, such that an estimate of the desired result is computed. The more samples are drawn, the more accurate the result will become. Monte-Carlo methods are well suited in environments which include uncertainty and chance, since they rely on random sampling.

One can for instance use Monte-Carlo methods in order to compute the surface of a circle. By defining a square, such that the circle is located inside the square, one can then draw many random points from inside this square. An estimate of the surface can be obtained, by multiplying the surface of the square with the fraction of random points which were located inside the circle.

When applied to games, the Monte-Carlo method samples randomly played out games for each move that is possible at the current state. Then, the game results are computed for each sample and the move which produced the highest average game score amongst its samples is selected as the best move. The result of such a game can have different representations. For instance, the result for many classic board games can be represented as a zero-sum result. For other games, which can have various degrees of victory such as Thurn and Taxis, a different system of result representation might be more appropriate. Since

the result of a game of Thurn and Taxis is point-based, one might just provide these results instead. Subsection 4.2.4 contains further explanation why a point-based result representation is more advantageous and contains the description of the applied method, which is used to represent results of simulated games for both MC Search and MCTS. MC Search has several distinct advantages over standard tree search. First, MC Search only requires the rules of the game, which includes a way of computing the result of the game. These rules are already defined by the game itself and only need to be implemented. Tree-search methods however require extensive domain knowledge such that an estimate of a mid-game position can be computed. Additionally, tree-search methods perform increasingly worse if there are random or unknown elements present in the game, since these elements increase the search depth that one must search, in order to reach a set amount of turns. Another advantage is that tree-search methods do not perform well with games that have a large branching factor, since it becomes very computationally expensive if one wants to reach appropriate search depths. While an increasingly large branching factor also hinders the MC Search, the MC Search is not as gravely affected as tree-search methods. MC Search methods can easily cope with large branching factors by just increasing the amount of sampling, or applying some domain knowledge during the sampling process.

MC Search, however, does have a disadvantage when compared to tree-search methods. This severe disadvantage is that MC Search lacks the tactical insight which a search tree possesses. Assume a move, which would lead to a certain win if the game would be continued in a special way. A search tree would recognize this move as a good move and probably play it. The MC Search, however, does not see this advantage, since the score of this move would be comprised of the random playouts of bad and good moves, thus obfuscating the potential that this move possesses.

After a move has been selected to be simulated, another step needs to be performed before this can be done. Thurn and Taxis is a game of imperfect information, thus it is necessary to take this into account. The basic way to deal with this problem is to inspect all possible states of this hidden information according to their probability of occurrence, such that the score of the move is the average over all simulations performed for each possible state of the imperfect information. Figure 4.1 shows an illustration of the Monte-Carlo search process when incorporating imperfect information.

There exist ways to improve this method. For instance one can use opponent modeling [3] in order to only simulate perfect information game states which are likely to occur according to the opponent model.

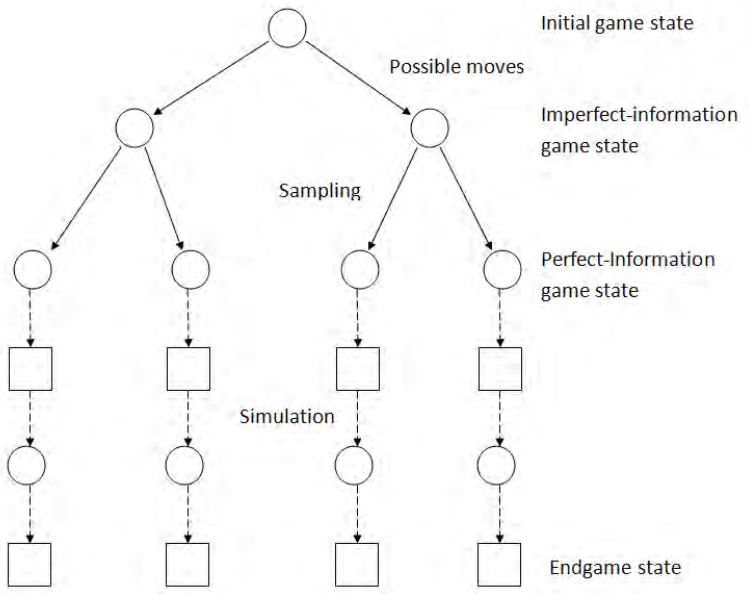

Figure 4.1: Illustration of the Monte-Carlo search process in a game with imperfect information.

4.2 Monte-Carlo Tree Search

Due to the lack of tactical insight of MC Search, discussed in section 4.1, a way was needed to resolve this issue. MCTS [11] combines the advantages of standard MC search with a search tree. Since Monte-Carlo sampling is still the essence of this search method, no heuristical board evaluation methods are required. However, the samples are used to construct a search tree, such that after sufficient samples have been drawn a tactical decision can be made. The basic algorithm of MCTS can be divided into four different steps, called selection, expansion, simulation and backpropagation. These tasks are carried out repeatedly until there is no more search time available. Then the final move needs to be selected based upon the result of the constructed search tree. In the following subsections, the different tasks will be explained.

4.2.1 Selection

During selection the search tree is traversed until a leaf node is reached. This task controls to what extent promising moves are investigated more extensively than less promising moves, similar to Variable Depth Search [17]. Figure 4.2 displays an example of such a traversal of a search tree.

In essence, this part of the algorithm controls the balance between exploration and exploitation. Exploration is the task of investigating nodes, which are barely visited and possibly have a bad score from previous simulations. The goal of explorations is to find unknown nodes, which potentially have a high value for the player. On the contrary, exploitation is the task of investigating nodes which already have been visited and have received an at least promising score. This is to approximate the true value of this node better, such that eventually the best node can be selected. How to handle the trade-off between exploration and exploitation will be further explained in section 4.3.

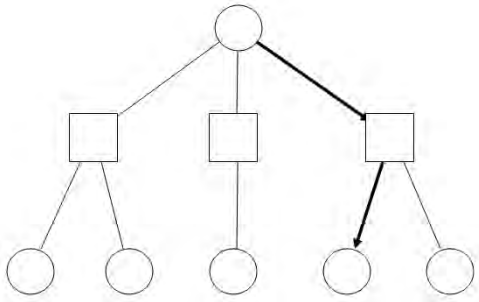

Figure 4.2: Illustration of the selection step in an example Monte-Carlo search tree

4.2.2 Expansion

The expansion task has the responsibility of expanding the search tree. Given the node which has been selected during the selection task a certain amount of child nodes are added to it. Figure 4.3 shows an example of the expansion process.

Several ways exist how to tackle this task. The most commonly used way is to create a single node during each expansion step. A good strategy is also to fully expand a given node, if it has been visited a set amount of times. This strategy has been used for instance in the Go program *Mango* [12].

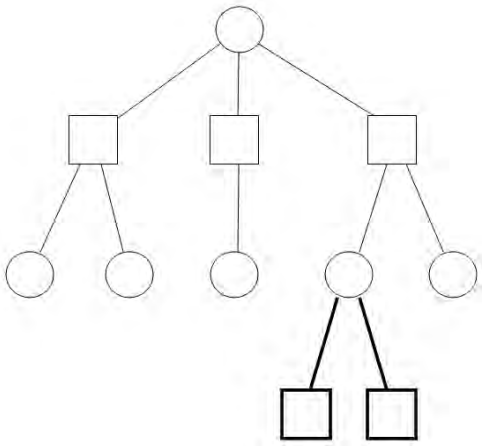

Figure 4.3: Illustration of the expansion step in an example Monte-Carlo search tree.

4.2.3 Simulation

Simulation is the task of finishing the game from the current node using self-play. This task is similar to the simulations performed during standard MC Search. Figure 4.4 shows an example of the simulation step.

The move selection during the simulation step is as in standard Monte-Carlo search completely random. It is possible to improve this task by choosing moves during the simulation according to a pseudo-random distribution or selecting moves using heuristics. The advantages and disadvantages of these improvements will be discussed in subsection 4.4.3.

4.2.4 Backpropagation

After the results of a simulation have been obtained, these have to be forwarded through the search tree to the top. This process is called backpropagation. Whenever a simulated game k of leaf node L is finished, its result R_k is added to the total result count of that leaf node and each node which was traversed in order to reach it. Figure 4.5 shows an example of the backpropagation step.

For zero-sum games like Chess, the result R of a game is represented by the numbers 1, 0 and -1, where 1 means a win, 0 a draw and -1 a loss. If n_L is the amount of times node L has been visited, the value v_L of node L can be computed by taking the average outcome of all simulations that traversed this node:

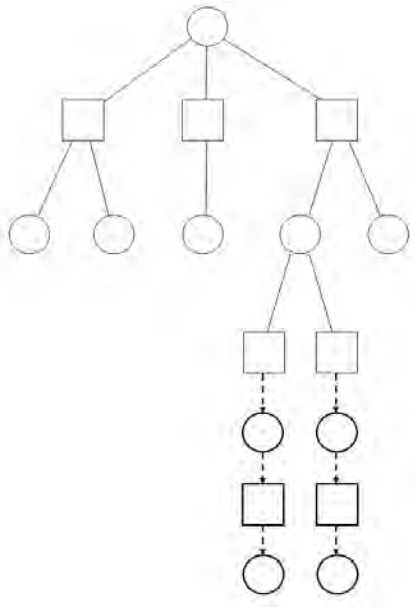

Figure 4.4: Illustration of the simulation step in an example Monte-Carlo search tree.

$$v_L = (\sum_k R_k)/n_L \tag{4.1}$$

However, this representation of a score is not suited for non-zero-sum games, like Thurn and Taxis. In the game of Chess a win through the use of a certain move has the same value as a win through a different one. Since the win evaluation of Thurn and Taxis is point based, however, two moves leading to a win can have different values, since a win with a lead of 10 points is more worth than a win with a lead of only 1 point. Hence, it would be better if the AI could differentiate between these moves. Problems with this representation also arise in multiplayer games. Due to the zero-sum representation of a result, the AI would assume that every opponent is trying to minimize the score of the AI, resulting in the investigation of moves during the selection step which possibly are unlikely to be played. This in essence leads to an unintentional paranoid assumption [29] during the search.

Rather than propagating zero-sum results through the tree, it would be better suited for this game to use the actual point-based results instead. That way one can decide, based upon the scores of the game, how the tree is traversed. Hence the results of a simulated game k are represented as the set $\{p \epsilon P : R_{k,p}\}$, where P is the set of players and $R_{k,p}$ the score of player p at the end of game

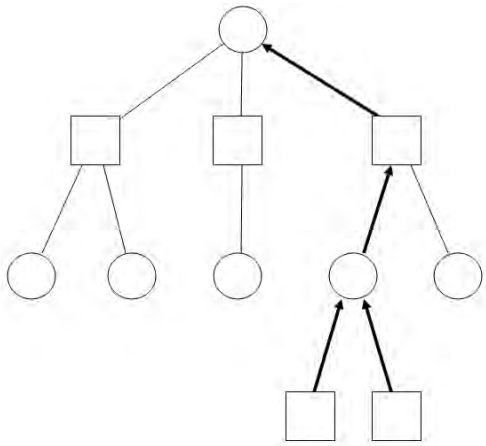

Figure 4.5: Illustration of the backpropagation step in an example Monte-Carlo search tree.

k. This set is then propagated through each node that was traversed in order to reach the current leaf node.

Next, a way is needed to calculate the value of a node. However, since Thurn and Taxis is a multiplayer game, there cannot be a single value for a node, but rather a value for each player. One can represent the value of a node for a player simply as his average score for each playthrough, but that would likely cause the AI to ignore moves which might not be so beneficial for the player himself, but rather devastating for his opponents. A better way to represent the value of the node is by the score difference between the player and the highest-scored opponent. When given this representation, moves that lead to a high value can either be moves that benefit the player or hinder the opponents. Hence, for this game, if n_L is the amount of times node L has been visited, $R_{p,L}$ the total amount of points accumulated by player p over all simulations which traversed node L and O_p the set of opponents of player p, the value $v_{p,L}$ of node L for player p is defined as:

$$v_{p,L} = \min_{o \epsilon O_p} \left(\frac{R_{p,L}}{n_L} - \frac{R_{o,L}}{n_L} \right) \tag{4.2}$$

This equation in essence calculates the average score difference between the player and his highest-scored opponent.

4.2.5 Final Move Selection

Eventually, when there is not enough search time available, a move corresponding to a child of the root needs to be selected, which is considered the best

possible child according to some criteria based upon the search results. There exist several criteria [12] which can be applied when selecting the child which is to be selected:

Max child: Select the child with the highest value.

Robust child: Select the child with the highest visit count.

Max-Robust child: Select the child which has both the highest visit count and the highest value. If there is no max-robust child at the moment, it is better to continue the search until a max-robust child is found rather than returning a child with a low visit count [16].

Secure child: Select the child which maximizes a lower confidence bound.

4.2.6 MCTS with Imperfect Information and the Element of Chance

Standard, MCTS is unable to cope with games that contain imperfect information or the element of chance. Hence, an adjustment needs to be made in order to make this possible. Similar to Expectimax search [26], this can be done by adding chance nodes to the search tree. During the selection step, this node would select a child not according to the selection method used elsewhere in the tree, but randomly according to the chance of occurrence of each child node. Likewise, when the expansion step is performed on this node, the new child nodes are created at random and not according to the standard expansion strategy. Figure 4.6 displays an example tree which utilizes chance nodes.

4.3 UCT Selection Strategy

When selecting a child from a node to investigate, one is basically faced with a resource-allocation problem. Ideally, one would only want to investigate promising nodes. But one can only identify promising nodes without using heuristics, by investigating these while their values are still unknown. So in essence, one would want to invest enough computational resources in unknown child nodes, such that one can see which nodes are worth additional resource investments. This problem is very similar to the Multi-Armed Bandit (MAB) problem. In the MAB problem, a person is faced with a set of MAB's, which are gambling devices, and is tasked to maximize the reward he receives from those machines with a set amount of money. Each of these machines follows a set stochastic distribution when calculating the reward of one activation, such to identify the machine with the highest expected reward and invest as much money as possible into this machine. These two contradicting actions are known as the exploration-exploitation dilemma. While exploiting the most promising options, one basically wants to invest into lesser promising options as well, ensuring that no possibly better option is missed.

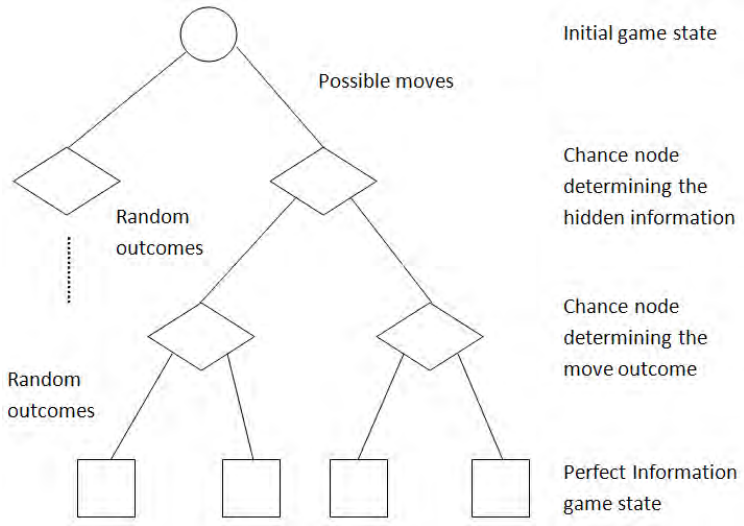

Figure 4.6: Illustration of a MCTS tree which incorporates hidden information and the element of chance

Inspired by this problem, Kocsis and Szepesvári [22] developed the UCT selection strategy, which has been very popular amongst researchers since its development. If I is the set of child nodes reachable from node p, v_i the value of node i, n_i the visit count of node i and n_p the visit count of node p, then the UCT selection strategy selects the node k that satisfies the following equation:

$$k = \arg\max_{i \epsilon I}(v_i + C * \sqrt{\frac{\ln n_p}{n_i}}) \tag{4.3}$$

Here, C is a constant, which needs to be tuned manually. Additionally, it is common in practice not to apply the UCT selection until a node has been visited a set amount of times T [16], where the value T needs to be determined manually as well. If a node has a lower visit count than this threshold, then a node is selected according to the simulation strategy. Note, that this is the definition for two-player games. However, since in multi-player games a node cannot have a single value, an adaption to the selection strategy is needed. Using the definition of the value of a node presented in equation 4.2, the UCT selection strategy can be adapted in the following way in order to cope with the multi-player environment: If I is the set of child nodes reachable from node p, q_p the player that has to move at node p, $v_{q_p,i}$ the value of node i for player q_p, n_i the visit count of node i and n_p the visit count of node p, then the adapted UCT selection strategy selects the node k that satisfies the following equation:

$$k = \arg\max_{i \epsilon I}(v_{q_p,i} + C \times \sqrt{\frac{\ln n_p}{n_i}}) \tag{4.4}$$

4.4 Heuristic Enhancements

In this section, several enhancements for both the MC Search and MCTS will be investigated.

4.4.1 Limiting Simulation Lengths

Monte-Carlo techniques heavily rely on the sampling process. The most straightforward way to improve a Monte-Carlo based AI is simply enabling it to draw more samples by some means. One way of doing so is by limiting the amount of turns which are played during the simulations. The idea behind this technique is, that the additional samples more than make up for the loss of quality, which the samples suffer when being cut down. Tree searches cannot search to such extents which Monte-Carlo techniques are able to simulate, for instance full games. If they were able to do so, then this would correspond to playing optimal and further research would not be necessary. Hence, tree-search approaches look for advantageous positions as far ahead as possible. Even though this limitation exists, tree-search approaches have been shown to be able to play at master level for many games [7]. Thus, if tree-search approaches are able to play at master level given their limited hindsight, then the gain of extra simulations probably outweighs the degradation that the samples suffer given the shortened simulation lengths.

While limiting simulation lengths may only be optional for some games, for other games this technique can be a necessity. Take for instance the game of Go. Even if one were to play completely random, the game length does not substantially increase. For games like Thurn and Taxis, however, it is possible to play in such a way that the game never ends. For this game, the game length increases if a player plays poorly. While a game of two average human players can take as long as 70 turns, early experiments have shown that games played by two random players can last as long as a thousand turns. With such an increase of game length, it becomes infeasible to perform random simulations with no restrictions. This issue raises the question why random play-outs require so much time compared to games played by human players. The cause of this issue lies within the way the game progresses. The progression of the game Thurn and Taxis depends on the players playing good moves, or at least avoiding bad moves. In order for a carriage to increase in its level and eventually trigger the ending condition, a player must finish a route of certain length in order to upgrade his carriage. In the game of Chess, if for instance the queen of the opponent has no cover, then the good move of capturing the queen only depends on a single decision that needs to be made. However, for Thurn and Taxis a good move depends not on a single decision, but on a series of decisions

split over several turns which all need to be made correctly in order for a good move to emerge.

Let us investigate an example in order to demonstrate how unlikely it is that a good move is played through random decisions in Thurn and Taxis. Assume that it is the goal of the random AI to play any route of length 3. Also, assume that on average there are two cards on the display which are of use and the player has no cards in his hand. Then the chance of playing this route depends on several factors. First, the AI needs to randomly pick three cards which can be connected into a route. Since it is assumed that on average two of the six cards on the display are of use and the drawn cards are immediately replaced by a new random card, the chance of picking three cards which are of use is $(\frac{1}{3})^3$ or $\frac{1}{27}$. Second, the AI needs to correctly attach the three cards while not choosing to scrap the current route. The first card can be played with no problem since it is assumed that the route of the AI is empty. The second card can be attached to either side of the first one. However, if the AI randomly chooses to scrap the current route, then the first card will be removed and the route cannot be finished using the three cards of this example. Thus, there is only a 1/2 chance that a random AI will successfully attach the second card. The third card, in most cases, can only be attached to one of the two played out cards. So the AI must attach the third card at the correct end of the route and also choose to close the route in order to achieve its goal. It is of course possible to continue expanding the route, but this will likely lead to the random AI scrapping it later on. Hence, the AI has the choice between attaching the third card to the incorrect side and scrapping his route, attaching it to the correct side while not closing the route and attaching it to the correct side while closing the route. Thus, the chance of successfully attaching the third card and closing the route is 1/3. Taking all these factors into consideration, one can estimate the chance c of completing a route of length 3 as the following:

$$c = \frac{1}{27} \times 1 \times \frac{1}{2} \times \frac{1}{3} = \frac{1}{162} \tag{4.5}$$

Note that there are other factors, which can decrease the chance of randomly playing a good move as well. First, the assistants have an influence on the chance of successfully constructing a route. The Administrator, for example, can be a necessity or liability, depending on the game state. If, for instance, there are cards which are of use on the display, then it would be bad to choose the Administrator, since this may result in a display with no usable cards at all. If this would be the case and the AI has no other usable cards in his hand, then it would be forced to scrap its current route. On the contrary, if there are no usable cards on the display to begin with, then the AI must choose the Administrator in the hope that there are usable cards among the new display. Choosing the Postal Carrier can be a bad choice, too. If a player cannot accumulate two attachable cards after choosing the Postal Carrier during his turn, then he would be forced to scrap his current route as well, since he cannot attach

at least one card to his route. The opponent(s) can also be an influence. Since in order to complete any route at least two turns are required, any opponent can hinder the player in between his turns by drawing the cards that the player needs himself, or by replacing the display using the Administrator.

One can conclude that the chance of completing any route of length 3 using random play is very small. In order for the game to progress and eventually finish, however, longer routes with at least one with length of at least 5 in conjunction with the Cartwright are required, in order to cause the game to end via the carriage ending condition. It is easy to deduce that the chance of such a move occurring using random play will be exceptionally small. Such a move may not be necessary, however, since it also possible to cause the game to end by consuming all houses which are available. This can even be achieved by solely playing routes of length 3. Attempting to end the game this way using random play will be increasingly unlikely, however, the further the game progresses. If, for instance, the random AI needs to place one last house to cause the game to end, then playing any route will not suffice to cause the game to end. The AI must play a route that incorporates at least one of the cities, which do not have a house yet. The chance of this occurring is a fraction of the chance of any route being completed.

It is evident that the chance of randomly playing a move which causes the game Thurn and Taxis to end is exceptionally small. Hence, random playouts during the simulation take a very long time to finish, causing a severe lack of samples for a Monte-Carlo based AI. Therefore, limiting the simulation length is necessary in order to accumulate a sufficient amount of samples.

The execution of this technique is fairly straightforward. When performing a simulation, a turn-limit is given, indicating the amount of turns which are to be simulated. After this amount has been reached, the simulation is aborted. However, since the game has been stopped before the ending condition has been reached, it is not possible anymore to return zero-sum results. Hence, a different measure needs to be supplied, like a heuristic evaluation of the game-state. Fortunately, this is not an issue for the game Thurn and Taxis, since it is not a zero-sum game due to its point-based scoring system. Hence, when a simulation has ended, a point-based result is returned which is based on the score of the AI and its opponent(s). How this value is calculated can be seen in equation 4.2. In order to avoid simulations that run extremely or even indefinitely long, one should apply this technique as well, albeit with a generous limit such that the simulations are at least as long as the average game length. For this research, a maximum length of 100 turns is applied globally, which is well above the average game length, estimated in subsection 3.2, ensuring that samples are generated at a consistent rate.

4.4.2 Game-Progression-based Simulation Cuts

Another way of gaining more samples when simulating games is to stop the simulation, if a player has a sufficiently large advantage over his opponents. For many games, it is possible to estimate with a high certainty that one player is winning, even though the game has not ended yet. Many professional board game players forfeit the match, if they are certain that they cannot win the match anymore, figuring that continuing the game is a futile effort. The same principle can be applied to the simulation process as well. If it is certain that a simulated match is won by a player, then one might as well stop the simulation at this point and start a new one. However, one cannot simply use the current score of the game to achieve this, since simulations which start with an already advantageous position would be stopped immediately, effectively leading to the AI making random decisions. Hence one should consider relative advantages with respect to the initial position of the simulation instead. This approach should result in an increase of samples. However, this approach requires some form of heuristic which estimates whether a player has a significant lead over his opponents or not. The applied heuristics should not be too computationally intensive, else they would cause the simulations to slow down to such an extent that the sample gain is nullified.

The difficulty of estimating a mid-game position entirely depends on the game itself. For the game of Chess, one can easily make a simple estimation of a position by looking at how many pieces of which type a player has left compared to his opponent. One can of course make this estimate more sophisticated by taking into account aspects, such as positioning, cover or threat, but adding complicated features will likely decrease the simulation speed too much. On the other hand, for the game of Go for instance it is very hard to estimate a position in general. Therefore, including this heuristic in order to gain simulation cuts may be so computationally intensive, that one would end up with fewer samples than without it. Finally, for the game of Thurn and Taxis it is fairly easy to make a reasonable estimate of a player's advantage. One can simply inspect the amount of points each player has accumulated so far and see if any player has a significant lead over his opponents. Since points are the essence of this game, the amount of points which a player has so far is the most crucial attribute of a player's position. It is possible to add other features to the heuristic, such as the current route and hand of the player, but is unlikely that the small quality gain of the heuristic would make up for the increased computation time.

4.4.3 Pseudo-Random Simulations

A different way of increasing the strength of a Monte-Carlo AI is to increase the quality of the drawn samples. From all the drawn random samples, only few are likely to represent games which would actually be played by skilled players. Though random selfplay has the advantage that it is fast and has the potential to reach the entire search space, a pure random simulation, however, is unlikely

to result in a value close to the real value of the node, since it will also choose bad moves which might turn the game in favor of a different player. Hence, all other samples can be seen as noise, which either cause the AI to over- or underestimate a move. If the ratio between good samples compared to bad ones would be higher, then this should lead to an increase in strength of the AI. Rather than utilizing random selfplay, one can simulate games by selecting moves through the application of heuristics. This would increase the quality of the drawn samples. A problem which arises when using solely heuristics when simulating games, is that the search space is limited by the heuristics. This means that potentially good moves can be ignored if the heuristic does not estimate this move correctly. Additionally, if there is more than one good move available to choose from, a purely heuristic selection will likely only choose the same move. An additional disadvantage of applying heuristics is that the constant application of heuristics slows the simulation process down by a significant amount.

Bouzy [6] proposed a compromise between random simulations and heuristic simulations. This compromise is based on selecting moves according a pseudo-random probability distribution. In this distribution, the probability of each move being chosen corresponds to a score calculated by a heuristic. This approach prevents the same move from constantly being selected when faced with the same situation, but rather allows different moves to be investigated as well. This includes moves which are bad according to the heuristic as well, although only with a small probability. Hence, the entire search space is available using this approach. The application of pseudo-random simulation has been shown to increase the strength of an AI compared to an AI using standard random simulations [6].

The Applied Heuristics

For many classic board games, a turn of a player only consists of making a single choice between a set of moves. For the game of Thurn and Taxis, however, a turn consists of several different actions. Hence, an appropriate heuristic function needs to be applied to each of these different actions. One can separate these actions into three types. First, a player needs to choose one of the four available assistants or opt not to choose any. Second, a player needs to draw a card. This action can be performed twice if the player has chosen the Postmaster during his turn. Last, a player needs to attach one or two cards to his route, depending on the choice of his assistant, and decide if he wants to close his route. If he chooses to close his route, then this action must also include a choice of cities on to which houses are placed and a selection of cards which he needs to discard, provided he has too many on his hand. The reason for this separation is, that between each of these three actions an element of chance can occur. For example, if a player draws a card from the display, a new randomly

drawn card is placed on the empty position. Or if the player chooses the administrator, then the display is replaced with six new randomly drawn cards. The actions of attaching a card to a route and closing the route can be combined into one large action, since the outcome of both actions are deterministic.

When calculating the usefulness of an assistant, each assistant needs to be assessed by a different criterion, since each assistant affects the game in a different way. The choice of not enlisting the help of any assistant always receives a low score, since it is always better to choose the Postmaster or Cartwright instead due to neither of the two assistants having a potential disadvantage. In order to assess the Postal Carrier, one needs to look at the hand of the player and the display. The Postal Carrier is only useful if a player can actually attach two cards two his route. If this is not the case, then the player would be forced to scrap his route. Hence, if the player can attach two cards to his route by using two cards which he already has on his hand, or using one card from his hand and one card from the display which he can draw during this turn, the Postal Carrier receives a high score. If this is not the case, then the Postal Carrier receives a low score. Calculating the score of the Postmaster is rather straightforward. By standard, this assistant receives a high score since it is always good in this game to draw more cards. Even if there is only one useful card to the player himself, he can always draw a card which one of his opponents needed, thus hindering their progress. The Administrator only receives a high score, if there are no useful cards for the player on the display. Else, the player would risk removing the useful cards on the display and not getting any useful cards in return. Hence, if there is at least one useful card on the display, the administrator receives a low score. The Cartwright also only receives a high score if a special condition is met. This condition takes into account the length of the current route and the size of the carriage, which the player has. The Cartwright only receives a high score if the Cartwright would cause an upgrade of the carriage of the player, whereas the lack of the Cartwright would not cause the carriage to upgrade, if the player were to close his route this turn. In every other case, the Cartwright receives a low score.

When faced with the task of drawing a card, each card on the display is scored by its usefulness to the player. If he can attach this card directly to his route, then the card receives a high score. If the card cannot be attached to the route, but to a card on the player's hand instead, the card receives a medium score. Else, the card is not useful for the player at all and hence receives a low score. The option of drawing a random card from the stack instead needs to be judged differently. Drawing a random card from the stack is usually a desperate move, only chosen when there are no better options to choose from, since it will only yield a useful card with a low chance. Hence, if none of the cards on the display have received an at least medium score, the move of choosing a random card is assigned a high score. Else, it is assigned a low score.

One can separate the third type of actions into three different subgroups. The

first consists of moves, which cause the current move to be scrapped. Moves which attach one or two cards to the route without closing it can be grouped into the second subgroup and moves which attach one or two cards and close the route can be put into the third. It is evident that moves of the first and second subgroup can easily be assessed. Scrapping the current route is hardly ever a good action, since this would mean that the effort of constructing that route has been wasted. Thus, a move which caused the route to be scrapped always receives a low score. Extending the current route is the basic action performed during the game of Thurn and Taxis, and thus always receives a medium score. Moves which cause the route to be closed require more attention. If the closing of the route would cause the carriage to be upgraded, then the move receives a high score. Else, the score of the move is proportionate to the amount of houses placed on to the field. In a regular game of Thurn and Taxis, not every closed route causes a carriage upgrade due to varying reasons. However, the more houses are placed on to the field, the more points that move yields, since each placed house is one penalty point less during the calculation of the final scores. Plus, every placed house makes it easier to be eligible for points later in the game.

4.4.4 Mechanisms for Pseudo-Random Simulations realized using Binary Heuristics

While Pseudo-Random simulations do increase the strength of a Monte-Carlo based AI, there is still a problem that remains with this approach. When calculating which move to select at a certain position during the simulation, it is necessary to first generate all possible moves at this position and then apply a heuristic function for each of these moves. Thus, compared to random selections, this method requires a considerable amount of resources in order to select a move. Hence, when utilizing pseudo-random simulations, the AI can only draw significantly fewer samples when compared to random simulations.

Ideally, one would want to draw samples at the same speed as done during random simulations, but with the samples being of the same quality than the samples drawn during the pseudo-random simulations. For this to happen, it is necessary to somehow reduce the time needed to select a move. Standard pseudo-random simulations compute the heuristic value for each move, including moves which are unlikely to be selected anyway. This basically is quite a redundant operation, but one cannot avoid computing the heuristic value of a bad move, since in order to skip the move, it is necessary to know its value.

It is evident that one cannot reduce the computation time for generating a pseudo-random move by skipping bad moves. Hence, a different way is needed to speed up the selection process. Rather than having to compute an entire probability distribution, it would be much more efficient to generate moves according to some mechanism from which a pseudo-random probability distribution emerges. If such a mechanism consumes less computation time than the

generation and evaluation of every possible move at some position, then such an approach should increase the simulation speed.

An example of such a mechanism would be to generate a random move and to replace this move with a new random move if the first move is not adequate according to some heuristic. This method requires a binary heuristic, that judges whether a randomly drawn move is kept and played, or whether a new random move should be generated instead. It is evident that moves which are adequate according to the heuristic are more likely to be played than the remaining moves, thus moves are pseudo-randomly selected. One can also show that this method requires less computation time in most circumstances than the standard pseudo-random move selection. Assume that g is the cost to generate any move, h is the cost to apply a heuristic function to any move and n_p is the number of legal moves which can be played from a certain position p. In order to apply the standard pseudo-random move selection, all possible moves need to be computed and evaluated. Thus the required computational costs c can be described as:

$$c = n_p \times g + n_p \times h \tag{4.6}$$

When using the proposed mechanism, assuming that the worst case scenario always occurs, every randomly drawn move would need to be first evaluated and redrawn. Hence, the cost c to generate a move according to the proposed mechanic can be described as:

$$c = 2 \times g + h \tag{4.7}$$

While, for standard pseudo-random generation, the computation costs increase if n_p increases, the costs of applying the proposed mechanism stay constant. One can easily see that the only time that pseudo-random move generation is faster, is when $n_p = 1$, which would result in a computational cost of $g + h$. However, if n_p has the value 2 or greater, the proposed mechanism is faster. Having $n_p = 1$ in essence corresponds to an exceptional situation, namely being forced into a single legal move by the opponent. Consequently, unless the game in question incorporates heavy aspects of such situations, one can safely assume that the proposed mechanism will be faster overall than the standard pseudo-random approach.

Note that this mechanism lacks the fine-tunability compared to standard pseudo-random selection. If, for instance, a certain move is slightly better than a different one, then the pseudo-random selection can select the better move with a slightly higher chance than the other move, corresponding to the difference in their heuristic value. On the contrary, when selecting moves using the pseudo-random mechanism, two moves, which are good enough that they will be kept

when evaluated by the binary heuristic, will have the same chance to be selected. Here, the chance for a move to be selected depends on several factors. Not only does the chance depends on whether the move is evaluated positively by the heuristic or not, but also on how many other moves are evaluated positively when inspected as well. The higher the ratio between bad moves and total moves available, the more likely it becomes that any good move is selected. Given any game position, if b is the amount of bad moves, according to the heuristic, possible at this position, n the total amount of moves possible and r the number of times a randomly chosen move is redrawn if it is not good according to the heuristic, then the chance of any good move being selected can be calculated as follows:

$$\sum_{x=0}^{r} \frac{b^x}{n^{x+1}} \tag{4.8}$$

Also, given the same definitions, the chance of any bad move being selected can be calculated as follows:

$$\frac{b^r}{n^{r+1}} \tag{4.9}$$

It is evident that, the higher the amount of redraws r is, the more likely it is that good moves are being selected and bad ones being avoided. Of course, increasing r means an increase of computation time as well, since the heuristics need to be applied more often for each position.

4.4.5 Hybrid Simulations

As mentioned in subsection 4.4.3, one reason to choose pseudo-random move selections over pure heuristics is to ensure that the entire search tree is explored. Inevitably, the simulation will eventually reach a point where it is extremely unlikely that the current position will be simulated again. Thus, it then becomes unnecessary to ensure that each part of the sub-tree can be visited, since it is likely that this particular sub-tree will be examined only once. This means, that at this point during the simulation, the computationally expensive pseudo-random move selection can be substituted by a different, preferably faster, selection method. Several options present themselves when choosing a second simulation method. One can opt for a fast simulation method, for instance random simulations, to generate a larger amount of samples. On the other hand one can choose simulations which generate samples of the same quality as pseudo-random simulations at a faster rate. Since ensuring that the complete search space is explored is not necessary at this point, these simulations can be done by pseudo-random mechanisms or even by pure heuristics.

Since it is uncertain which approach will be more successful, all the proposed hybrid simulations will be tested. Additionally, since it is uncertain at which

stage of the simulation it is necessary to switch to the second simulation method, different simulation distributions between the two simulation methods will be tested as well.

Chapter 5

Experiments

In this chapter, the results of the performed experiments for all investigated approaches will be presented. First, in section 5.1 the experiments considering standard Monte-Carlo Search and the investigated enhancements will be presented. Next, in section 5.2 the experiments to test the performance of MCTS and the effects of the investigated enhancements on MCTS will be presented. For each experiment, each AI received a total of three minutes of computation time to finish a single game. For the experiment of standard MC search against a random AI, only 50 test games needed to be performed until a conclusion could be drawn, with the AI utilizing MC Search playing 25 games as player 1 and 25 times as player 2. For every other experiment, 300 test games were conducted with each AI playing 150 games as player 1 and 150 games as player 2. Each AI uses a maximum simulation length of 100 turns, which is well above the average game length of Thurn and Taxis, unless otherwise specified.

5.1 Monte-Carlo Search and Enhancements

First, to establish a reference point for further experiments, an AI using MC Search has been tested against a randomly playing AI, with the results shown in subsection 5.1.1. Next, the effects of limiting simulation lengths are shown in subsection 5.1.2. Subsection 5.1.3 addresses the effects of game-progression based simulation cuts. Subsection 5.1.4 shows the results of utilizing pseudo-random simulations and subsection 5.1.5 shows the effects of mechanisms for pseudo-random simulations. Finally, subsection 5.1.6 addresses the effects of using hybrid simulations.

5.1.1 MC Search

For the first experiment, an AI using MC Search without any improvements has been matched against a randomly playing AI in 50 experimental games. The results of this experiment can be seen in table 5.1.

	MC Search	
	As Player 1	As Player 2
win	100	100
draw	0	0
loss	0	0

Table 5.1: Results of a standard Monte-Carlo AI versus a random AI. Values are percentages of total games played.

As can be seen, the Monte-Carlo AI won every single game. However, this result emerged not because of the possibly good performance of the standard Monte-Carlo AI, but more due to the extremely bad performance of the random AI, as discussed in subsection 4.4.1. In order to illustrate the difference of play quality between the two AIs, table 5.2 shows the move breakdown of choices each AI made when attaching a card to its route. As discussed in the end of subsection 4.4.3, a move at this stage of a turn can have three different general outcomes: the current route is scrapped, a new card is attached to the route while not closing it and the current route is closed in addition to adding a card to the route.

	MC	Random
Route scrapped	54.5%	79.7%
Card attached	28.3%	15.6%
Route closed	17.1%	4.6%

Table 5.2: Breakdown of card-attach moves per game of a MC AI compared to a random AI.

Scrapping your own route is a move which is to be avoided as much as possible in the game of Thurn and Taxis. The random AI, however, spends almost 80% of its turns scrapping its current route, either by being forced to due to bad card choices, or by just randomly selecting to scrap its current route. The MC AI does on average scrap its current route significantly less than the random AI, however it does not show signs of strong play, since scrapping its current route more than every second turn is still a sign of weak performance.

5.1.2 Limited Simulation Lengths

To test the effect of limiting the amount of played turns during simulation, standard MC AIs with varying smaller simulation limits have all been matched against a MC AI with the standard simulation limit of 100 turns. An AI with

a simulation length of 100 turns can on average draw approximately 1000 samples per second on a 2.4GHz processor when using random simulations. The amount of drawn samples per second linearly increases, as the simulation length decreases. The results of this experiment can be seen in table 5.3.

Simulation length limit	75		50		25		10		5	
As Player	1	2	1	2	1	2	1	2	1	2
win	57.3	54.6	85.3	79.3	97.3	98.0	100	100	100	100
draw	4.0	3.3	2.0	1.3	0.0	0.6	0	0	0	0
loss	38.6	42.0	12.6	19.3	2.6	1.3	0	0	0	0

Table 5.3: Results of various MC AIs with different simulations lengths playing against a MC AI with a standard simulation length of 100 turns. Values are percentages of total games played.

One can see that shortening the simulation lengths in favor of more samples greatly increases the performance of the MC AI. For the very short simulation limits of only 10 and 5 turns, the MC AI even won every single game against the standard AI. It is notable that there is a tendency that an AI plays slightly better when it is the first player. The most likely reason for this is that the game favors the player who has to move first, since he can choose the most promising cards from the display at the very first turn, possibly leaving the opponent with sub-standard options to choose from. In order to further investigate the performance of the AIs with a low simulation length, a further experiment has been conducted. Here, an MC AI using a simulation length of only 10 turns has been matched against AIs using simulation lengths of 25 and 5 turns. The results of this experiment can be seen in table 5.4.

Simulation-length limit	25		5	
As Player	1	2	1	2
win	17.3	10.6	50.0	49.3
draw	2.0	1.3	1.3	3.3
loss	80.6	88.0	48.6	47.3

Table 5.4: Results of MC AIs with simulations lengths of 25 and 5 turns playing against a MC AI with a simulation length of 10 turns. Values are percentages of total games played.

As can be seen, the AI with a simulation length of 25 turns performed considerably worse than the AI with a simulation length of 10 turns, having lost more than 80% of the games. However, reducing the simulation length from 10 to 5 turns only produced a negligible gain. Further experimental runs

would be required in order to be able to rule out random fluctuations as cause of this slight difference in the outcome of the experiment. Hence, for further experiments a simulation limit of 10 turns is applied in order to enable a larger hindsight for the AI.

5.1.3 Game-Progression-based Simulation Cuts

To test the effectiveness of game-progression based simulation cuts, for the rest of this thesis referred to as 'SimCut', two MC AIs using a simulation length of 10 turns have been matched against each other, with only one of these AIs using SimCut. For this experiment, simulations were aborted when one player gained a lead of at least 5 points over his opponent, representing a value of approximately a fifth of the total points accumulated during a game of Thurn and Taxis. This value is of course still subject to optimization. Table 5.5 displays the result of this experiment.

	SimCut	
As Player	1	2
won	52.6	41.3
draw	4.0	6.6
loss	43.3	52.0

Table 5.5: Results of a MC AI using SimCut against a normal MC AI, both using random simulations. Values are percentages of total games played.

It is evident that both AIs performed approximately equal to each other. Even when using a generous score difference of 5 points, the AI using SimCut did not manage to draw significantly more samples than the standard AI, drawing approximately 12628 and 12771 samples per second respectively. Since both AIs are already drawing samples at approximately the same rate, testing higher values of the score difference threshold would be futile. A higher threshold would lead to fewer samples drawn per second, due to the higher computational overhead combined with a decreased amount of simulation cuts occurring. It is questionable whether this approach is causing any simulation cuts at all, since random simulations take a long time to cause any significant changes on the board due to the bad level of play, as discussed in section 4.4.1. Section 5.1.4 will examine, whether a simulation approach which is more likely to encounter different game positions has any effect on SimCut.

5.1.4 Pseudo-Random Simulations

For the experiment regarding pseudo-random simulations, two MC AIs with a simulation length limit of 10 turns have been matched against each other, where one AI uses random simulations and the other uses pseudo-random simulations. The results of this experiment can be seen in table 5.6.

	Pseudo-Random	
As Player	1	2
won	96.0	95.3
draw	2.0	2.0
loss	2.0	2.6

Table 5.6: Results of a MC AI using pseudo-random simulations against a MC AI using random simulations. Values are percentages of total games played.

As can be seen, the inclusion of pseudo-random simulations vastly increased the performance of the AI. Even though it was only able to draw significantly fewer samples per second than the AI using random simulations, sampling 1972 samples per second using pseudo-random simulation has shown that the increase of quality of the samples more than made up for the loss of quantity.

It is also interesting, that the application of pseudo-random simulations greatly altered the playing style of the AI. To illustrate this change, table 5.7 displays the average amount of routes, which each AI closed during a game, divided into their respective lengths.

Route length	Random	Pseudo-Random
3	8.386	6.616
4	0.206	0.810
5	0.016	0.900
6	0.000	0.316
≥ 7	0.000	0.040

Table 5.7: Average amount of routes closed during a game of Thurn and Taxis of two AIs using different simulation methods.

It is apparent that an AI using random simulations focuses on playing routes with length 3 and rarely plays a route of greater length. Hence, its way of triggering the end condition of the game is to consume all its available houses. Since it hardly plays routes of length 5 or greater, the AI is extremely unlikely to upgrade its carriage to its maximum level. This behavior is likely caused by the simulations hardly playing longer routes and hence causing the AI to choose moves that will win the game by playing shorter routes.

On the other hand, the AI using pseudo-random simulations has a larger emphasis on longer routes, which in turn leads to more carriage upgrades. Since on average it only plays 0.04 routes of length 7 or greater, it mostly upgrades its carriage to the maximum level by closing a shorter route in combination with the Cartwright, which is the easiest way to do so.

Since pseudo-random simulations select more knowledgeable moves than just random simulations, it is likely that the application of SimCut would result in more simulations cuts, generating an increase in generated samples per second. To test this, an experiment has been performed, matching an AI using pseudo-random simulations against an AI using pseudo-random simulations applying SimCut with a generous threshold of 5 points score difference to cause a simulation cut. The results of this experiment can be seen in table 5.8.

	SimCut	
As Player	1	2
won	47.3	48.6
draw	1.3	7.3
loss	51.3	44.0

Table 5.8: Results of a MC AI using SimCut against a normal MC AI, both using pseudo-random simulations. Values are percentages of total games played.

Just as in section 5.1.3, the addition of SimCut only leads to an insignificant gain of simulations per second, 2183 per second with SimCut as opposed to 2090 without, causing the AIs to play at an approximately equal level. Since the AI applying SimCut using a low score-difference threshold, with the intention of causing many simulation cuts, did not draw significantly more samples, experiments using higher thresholds can be omitted, since an AI applying SimCut with a more general threshold will only be able to draw fewer samples per second, due to the overhead calculations that SimCut generates.

5.1.5 Mechanisms for Pseudo-Random Simulations

In order to test the effectiveness of this improvement, an experiment has been performed with an AI using a pseudo-random simulation mechanism competed against an AI using random simulations. Both AIs used a simulation-length limit of 10 turns and the AI using a pseudo-random simulation mechanism only redrew a move once if it was not suitable according to its used heuristics. Table 5.9 displays the result of this experiment.

The experiment has shown, that the AI using a pseudo-random simulation mechanism is superior to the AI using only random simulations. However, comparing the results of table 5.9 with the results of table 5.6 reveals, that this approach did not create results superior to the results of standard pseudo-random simulations.

To verify this, an experiment has been performed with an AI using a pseudo-random simulation mechanism playing against an AI using standard pseudo-random simulations. The results of this experiment can be seen in table 5.10.

	Mechanism	
As Player	1	2
won	64.6	58.6
draw	2.6	2.6
loss	32.6	38.6

Table 5.9: Results of a MC AI using a pseudo-random simulation mechanism against a MC AI using random simulations. Values are percentages of total games played.

	Mechanism	
As Player	1	2
won	20.0	29.3
draw	3.3	4.0
loss	76.6	66.6

Table 5.10: Results of a MC AI using a pseudo-random simulation mechanism against a MC AI using pseudo-random simulations. Values are percentages of total games played.

This experiment verifies, that the application of a pseudo-random simulation mechanism did not have the desired effect. While, with on average 4844 simulations per second, the AI was able to draw significantly more samples than the AI using standard pseudo-random simulations, drawing only 1972 samples per second on average, it still lost most of the games. The only explanation for this is, that the quality of the drawn samples is inferior to the quality of samples generated by pseudo-random simulation to such an extent, that even the superior quantity of samples could not compensate for the loss in quality.

This of course raises the question why the quality of samples is reduced. The cause of this problem is most likely, that the emergent distributions of move probabilities do not sufficiently resemble the probability distributions of standard pseudo-random simulations. Thus, bad moves are chosen more often during the simulation, which in turn degrades the quality of the samples. It is possible to improve the emergent distributions, by increasing the amount of times a randomly chosen bad move is being redrawn. Hence, an additional test has been performed, where the mechanism would redraw a randomly chosen move up to two times instead of one. Table 5.11 displays the results of this test.

Surprisingly, enabling the simulation to redraw a bad move twice actually decreased the performance of the AI. With on average 3719 samples per second

	Mechanism-2 Redraws	
As Player	1	2
won	10.0	9.3
draw	2.0	0.6
loss	88.8	90.0

Table 5.11: Results of a MC AI using an enhanced pseudo-random simulation mechanism against a MC AI using pseudo-random simulations. Values are percentages of total games played.

the pseudo-random mechanism drew approximately 1100 samples per second less than the mechanism using only one redraw. It still drew significantly more samples than an AI using pseudo-random simulations.

5.1.6 Hybrid Simulations

When performing a hybrid simulation, combining two different simulation techniques, then one has to consider the percentage of assigned simulation turns which each technique is allowed to simulate. For the experiments, three different distributions have been examined, assigning the first simulation method 80%, 50% and 20% of the total amount of turns to be simulated, leaving the other method with the remaining turns. First hybrid simulations, where the second part of the simulations where performed at random have been tested, where each simulation had a total length of 10 turns. The results of this test can be seen in table 5.12.

	Hybrid-Random					
Pseudo-Random turn percentage	80%		50%		20%	
As Player	1	2	1	2	1	2
won	38.6	48.0	32.0	30.0	5.3	9.3
draw	4.6	4.0	4.0	1.3	1.3	2.6
loss	56.6	48.0	64.0	68.6	93.3	88.0

Table 5.12: Results of a MC AI using hybrid random simulations against a MC AI using pseudo-random simulations. Values are percentages of total games played.

While the hybrid simulations with a large percentage performed comparably to its opponent which used strictly pseudo-random simulations, none of hybrid-random simulations displayed any performance gain, leading to the conclusion that using random simulations are not a viable option for hybrid simulations.

Next, a test using the same setup and hybrid simulations using pseudo-random mechanisms, using only one redraw of the randomly drawn move, as second simulation method has been performed. Table 5.13 displays the results of this experiment.

Pseudo-Random turn percentage	Hybrid-Mechanism					
	80%		50%		20%	
As Player	1	2	1	2	1	2
won	44.6	43.3	37.3	30.0	17.3	11.3
draw	2.6	2.6	2.6	1.3	2.6	2.0
loss	52.6	54.0	60.0	68.6	80.0	86.6

Table 5.13: Results of a MC AI using hybrid simulations with pseudo-random mechanisms against a MC AI using pseudo-random simulations. Values are percentages of total games played.

Just like Hybrid-Random simulations, none of the variations were able to outperform the AI using pseudo-random simulations. Both hybrid approaches, using random moves or pseudo-random mechanisms for simulations display a similar pattern regarding their performance. Both play almost on par with the opponent if the second simulation method is only executed a small amount of times, while the performances of both AIs decrease as the assigned turn percentage for the second simulation method increases.

As last experiment, a hybrid simulation method was tested which used pure heuristic move selection as second simulation method. Table 5.14 displays the result of this experiment.

From the experiment it becomes evident that none of the variations managed to outperform the AI using pseudo-random simulations.

5.2 MCTS and Enhancements

This section will tackle all experiments which were performed regarding the effectiveness of MCTS. Subsection 5.2.1 contains the brief experiments, which were run to find suitable parameter values for C and T. In subsection 5.2.3 the results of the experiments regarding Limited Simulation Lengths will be presented and subsection 5.2.4 presents the effects of using game-progression based simulation cuts. The effects of using pseudo-random simulations and mechanisms for pseudo-random simulations will be displayed in subsection 5.2.5

Pseudo-Random turn percentage	Hybrid-Heuristic 80%		50%		20%	
As Player	1	2	1	2	1	2
won	40.6	44.6	28.0	30.0	7.3	10.0
draw	3.3	2.6	1.3	1.3	1.3	2.6
loss	56.0	52.6	70.6	68.6	91.3	87.3

Table 5.14: Results of a MC AI using hybrid simulations with pure heuristic simulations against a MC AI using pseudo-random simulations. Values are percentages of total games played.

and 5.2.6 respectively. Last, subsection 5.2.7 presents the effects of using various types of hybrid simulations.

5.2.1 MCTS Parameters

MCTS uses two parameters, C and T. C is a mandatory parameter, describing how much the selection strategy should emphasize exploration or exploitation. The full expansion threshold T, however, is not a mandatory parameter for the expansion strategy, but it is applied in most applications of MCTS [12].

For each parameter, three rough values were examined to gain insight at what range of values the optimal values of C and T could lie. First, an experiment has been performed using three different values for C: 0.5, 1.5 and 2.5 using an initial value 10 for T. Note that values larger than 1 are necessary to be examined, since Thurn and Taxis is not a zero-sum game, meaning the value of a node can be bigger than 1 or lesser than -1. For each of these values, 50 test games have been performed against a standard Monte-Carlo AI, with both AIs using random simulations. The results of the experiment can be seen in table 5.15.

C	0.5		1.5		2.5	
As Player	1	2	1	2	1	2
won	36.0	40.0	32.0	16.0	38.7	54.8
draw	4.0	0.0	8.0	0.0	3.2	12.9
loss	60.0	60.0	60.0	84.0	58.1	32.3

Table 5.15: Results of various values for C for the selection method of UCT. Tested against a MC AI with both AIs using random simulations. Values are percentages of total games played.

One can see from table 5.15 that the AI using MCTS played the strongest with a high value for C, allowing more exploration than the other values.

Next, for C=2.5, three different values for T were tested: 5, 10 and 15. For each of these values, an AI using MCTS was matched in 50 test games against an AI using standard Monte-Carlo search, with both AIs using random simulations. The results of these tests can be seen in table 5.16.

T	5		10		15	
As Player	1	2	1	2	1	2
won	43.3	50.0	38.7	54.8	40.0	46.6
draw	0.0	0.0	3.2	12.9	6.6	3.3
loss	56.6	50.0	58.1	32.3	53.3	50.0

Table 5.16: Results of various values for T for the selection method of UCT. Tested against a MC AI with both AIs using random simulations. Values are percentages of total games played.

The performed experiments, that different values of T do only slightly alter the performance of the AI. Overall, it appears that for T=5 the AI has performed the strongest.

For the subsequential experiments, the values C=2.5 and T=5 have been applied.

5.2.2 Final Move Selection

There exist different strategies for a MCTS AI, when selecting the final move 4.2.5. For the course of this research, the two basic methods have been investigated: Max Child and Robust Child. Both approaches were matched against a standard MC AI, with both AIs using random simulations. The results of this experiment can be viewed in table 5.17.

It is evident from the experiment that the Robust-Child selection method outperforms the Max-Child method by a small margin. Hence, this selection method was used for the remainder of this research.

5.2.3 Limited Simulation Lengths

This subsection discusses the effects of limiting simulation lengths during MCTS. Several limits have been investigated and compared against an AI using the most effective limit of 10 turns from subsection 5.1.2. For each investigated limit 300

Selection Method	Max Child		Robust Child	
As Player	1	2	1	2
won	36.6	46.6	43.3	50.0
draw	5.6	6.6	0.0	0.0
loss	56.6	46.6	56.6	50.0

Table 5.17: Results of two different final-move-selection methods. Tested against a MC AI with both AIs using random simulations. Values are percentages of total games played.

test games have been played.

Simulation-length Limit	5		25		50	
As Player	1	2	1	2	1	2
won	66.1	61.1	18.6	20.6	11.3	7.3
draw	2.8	3.9	2.0	0.6	0.6	0.6
loss	31.1	35.0	79.3	78.6	88.0	92.0

Table 5.18: Results of a MCTS AI utilizing several simulation-length limits applied to random simulations playing against a MCTS AI using a simulation-length limit of 10. Values are percentages of total games played.

While for higher limits similar trends are visible compared to the results in subsection 5.1.2, there is a difference regarding the performance of the simulation limits 5 and 10. For standard MC Search both AIs indicated an equal playing strength, however, there is a significant difference in performance when applied to MCTS. Here, a simulation limit of 5 turns outperforms a limit of 10 turns. This is can be explained by the fact that a shorter simulation length produces more results, which in turn allows the MCTS to further converge to a better solution. Hence, the extra amount of samples is better utilized than in a standard MC Search.

While for random simulation an extremely short simulation-length limit produced the best results, a slightly different result was apparent when examining different simulation lengths when applied to pseudo-random simulations. The results of the experiment can be viewed in table 5.19.

While limiting simulation lengths is also successful when applied to pseudo-random simulations, it is notable that this approach causes the AI to perform weaker if the simulations are shortened to an extensive level.

Simulation Limit	5		25	
As Player	1	2	1	2
won	31.8	18.1	6.0	13.6
draw	4.5	4.5	0.0	1.5
loss	63.6	77.2	94.0	84.8

Table 5.19: Results of a MCTS AI utilizing several simulation-length limits applied to pseudo-random simulations playing against a MCTS AI using a simulation-length limit of 10. Values are percentages of total games played.

Given these results, the remaining experiments utilizing MCTS applied a simulation limit of 5 turns for random simulations and 10 turns for pseudo-random simulations, unless specified otherwise.

5.2.4 Game-Progression-based Simulation Cuts

To test the effectiveness of Game-Progression-based Simulation Cuts (SimCut) on MCTS, an experiment has been performed in which an MCTS AI applying SimCut with a score difference threshold of 5 points on random simulations played against a MCTS AI using just random simulations. The results of this experiment can be seen in table 5.20.

	SimCut	
As Player	1	2
won	44.6	44.0
draw	5.3	1.3
loss	50.0	54.6

Table 5.20: Results of a MCTS AI applying SimCut on random simulations against a MCTS AI using random simulations. Values are percentages of total games played.

Similar to the results of the experiment in subsection 5.1.3, the application of SimCut did not enable the AI to draw significantly more samples, with both AIs drawing approximately 6460 samples per second. It is apparent that the sample gain, caused by aborting simulations which are in advantageous positions, only manages to compensate for the computational costs invested in monitoring the game progression. One needs to consider, that with a shortened simulation length of only 5 turns and the application of random simulations, it is unlikely that an advantageous position will be simulated. To investigate these possible reasons, a second experiment has been performed, where both AIs have a significantly higher simulation-length limit. An experiment has been performed, in

which an MCTS AI using both random and pseudo-random simulations played an AI of the same configuration, with the only difference being that one of the playing AIs applied SimCut. Every AI applied a simulation-length limit of 50 turns, giving the simulations ample room to reach an advantageous game state for any player in order to achieve the SimCut threshold. The average amount of simulations per second have been recorded during the performed test game, visible in table 5.21.

Simulation Method	No SimCut	SimCut	Sample Gain
Random	1905.3	1935.2	1.57%
Pseudo-Random	182.1	260.5	43.1%

Table 5.21: Average amount of simulations per second for two different simulation methods with and without SimCut.

It appears that even with a very high simulation-length limit, SimCut only generates an insignificant increase of generated samples. However, when applied to pseudo-random simulations, SimCut generated an increase of generated samples of 43.1%. It is notable that the AI applying pseudo-random simulations and applying SimCut performed stronger than its counterpart without applying SimCut, which can be attributed to the fact that at such high simulation-length limits either of the two AIs only draw few samples such that any sample gain leads to an increase in playing strength.

To investigate whether SimCut also causes a sample gain at lower simulation lengths when applied to pseudo-random simulations, an experiment has been performed under lower simulation lengths, measuring the average amount of drawn samples of a MCTS-AI with and without applying SimCut. The results of this experiment can be seen in table 5.22 and 5.23.

Simulation-length Limit	No SimCut	SimCut	Sample Gain
25 turns	358.5	463.3	29.23%
10 turns	1253.9	1369.9	9.25%

Table 5.22: Average amount of simulations per second for a MCTS-AI applying pseudo-random simulations at different simulation-length limits with and without SimCut.

It appears that the more strict the simulation limit is, the less effective SimCut becomes, leading to an increase of only 9.25% more samples per second at a simulation-length limit of 10 turns. Even though there still exists an advan-

Simulation Length	25-SimCut		10-SimCut	
As Player	1	2	1	2
won	71.1	60.0	52.2	37.7
draw	2.2	2.2	5.6	2.2
loss	26.7	37.8	42.2	60.0

Table 5.23: Results of a MCTS AI applying pseudo-random simulations in self-play with one AI applying SimCut. Values are percentages of total games played.

tage in drawn samples per second at low simulation-length limits when applying SimCut, it appears that the actual performance of the AI degrades when doing so. When applied to longer simulations, SimCut leads to a larger increase in drawn samples per second, as well as an increase in playing strength. The latter can be substantiated by the fact that at these simulation-length limits, only relatively few samples are drawn and hence any increase in samples is likely to increase the playing strength as well.

Overall, one can conclude that SimCut is ineffective when applied to random simulations. When applied to pseudo-random simulations, SimCut does produce an increase in drawn samples per second, however this effect diminishes with further limited simulation lengths, to the extent that it only has little effect under the best found simulation-length limit. Additionally, when applied under the best found simulation length, SimCut actually appeared to decrease the playing strength of the AI.

5.2.5 Pseudo-Random Simulations

This subsection discusses the effect of applying pseudo-random simulations on MCTS. For this experiment two MCTS AIs played against each other in 300 test games, one AI using pseudo-random simulations and the other using random simulations. Table 5.24 displays the results of this experiment.

	Pseudo-Random	
As Player	1	2
won	98.0	97.4
draw	0.6	0.0
loss	1.3	2.6

Table 5.24: Results of a MCTS AI applying pseudo-random simulations against a MCTS AI using random simulations. Values are percentages of total games played.

The results indicate that also when applied to MCTS, pseudo-random simulations clearly outperform random simulations. This means that MCTS using pseudo-random simulations converges faster to a good solution than MCTS using random simulations, even though it can only draw significantly fewer samples.

5.2.6 Mechanisms for Pseudo-Random Simulations

In order to test the effect of mechanisms for pseudo-random simulations on MCTS, several settings regarding the amount of redraws of randomly chosen moves for the proposed mechanism have been investigated. For each investigated setting, an AI applying the proposed mechanism played an MCTS AI using pseudo-random simulations. The results of this experiment can be seen in table 5.25:

Number of redraws	1		2		3	
As Player	1	2	1	2	1	2
won	8.6	9.3	13.3	16.0	16.6	20.0
draw	2.0	1.3	2.0	1.3	2.0	4.6
loss	89.3	89.3	84.6	82.6	81.3	75.3

Table 5.25: Results of a MCTS AI applying a pseudo-random mechanism against a MCTS AI using pseudo-random simulations. Values are percentages of total games played.

It is evident from table 5.25, that the AI applying the mechanism for pseudo-random simulations cannot match the performance of the AI using pseudo-random simulations. Comparing its performance to table 5.24 one can see that this approach performed better than the AI applying simple random simulations. Additionally, one can see that increasing the number or redraws of random samples increases the performance of this approach, contrary to the results in subsection 5.1.5, where an increase of redraws led to a decreased performance of the AI. However, this increase in performance is not significant enough to compete with the performance of the AI using pseudo-random simulations. Despite the lesser performance of this approach, one can conclude from this and previous experiments that sacrificing sample quantities in favor of adding domain knowledge to the simulations is a superior approach when compared to simulation techniques which draw many samples, but only apply little to no domain knowledge, like for instance random simulations.

5.2.7 Hybrid Simulations

This section describes the effects that hybrid simulations have on MCTS. To investigate this approach, several experiments have been performed, matching

the different proposed approaches for hybrid simulations against an AI applying pseudo-random simulations. The first performed experiment matched an MCTS AI applying hybrid random simulation against an MCTS AI applying pseudo-random simulations. Various turn percentages for the hybrid simulations have been investigated. The results of this experiment can be seen in table 5.26.

Pseudo-Random turn percentage	Random 80%		50%		20%	
As Player	1	2	1	2	1	2
won	27.3	42.0	14.6	21.3	32.3	32.7
draw	7.3	4.0	0.6	2.0	2.7	2.0
loss	65.3	54.0	84.6	76.7	66.0	65.3

Table 5.26: Results of a MCTS AI using hybrid random simulations against a MCTS AI using pseudo-random simulations. Values are percentages of total games played.

As one can see, at a setting where only few turns are played via random simulations, the AI performs the strongest, albeit still weaker than the opponent applying pseudo-random simulations. Also, it is notable that at a low pseudo-random turn percentage, the performance of the AI is better than when applying a medium percentage, indicating that at this point the quantity of the samples starts to compensate for the loss of quality. However, this compensation is not sufficient for the AI to outperform its opponent, but almost causes the AI to perform similar to the AI applying hybrid random simulations with a high pseudo-random turn percentage.

Next, an experiment has been performed regarding the performance of hybrid simulations applying the proposed mechanism for pseudo-random simulations. For the mechanism, a maximum of 3 redraws has been applied, following the results of subsection 5.2.6. Again, different pseudo-random turn percentages have been investigated. The results of this experiment can be seen in table 5.27.

The results of the experiment show that applying the proposed pseudo-random mechanism to hybrid simulations instead of random simulations did not lead to a significant change in the performance of the AI. Like the previous experiment, the AI with the highest pseudo-random turn percentage had the highest performance among the explored percentages.

In the third experiment, an approach to hybrid simulations has been used, in which the simulations were finished by pure heuristic move selection. Different

Pseudo-Random turn percentage	Hybrid-Mechanism					
	80%		50%		20%	
As Player	1	2	1	2	1	2
won	37.1	37.1	27.3	29.3	20.0	28.7
draw	3.8	1.3	4.0	2.7	1.3	2.0
loss	59.1	61.6	68.7	68.0	78.7	69.3

Table 5.27: Results of a MCTS AI using hybrid simulations applying the pseudo-random mechanism against a MCTS AI using pseudo-random simulations. Values are percentages of total games played.

hybrid turn percentages have been examined, while playing against an MCTS AI applying pseudo-random simulations. The results of this experiment can be seen in table 5.28.

Pseudo-Random turn percentage	Hybrid-Heuristic					
	80%		50%		20%	
As Player	1	2	1	2	1	2
won	37.3	56.0	23.3	34.7	6.2	14.8
draw	2.0	4.7	0.7	3.3	0.0	3.1
loss	60.7	39.3	76.0	62.0	93.8	82.1

Table 5.28: Results of a MCTS AI using hybrid simulations with pure heuristic simulations against a MCTS AI using pseudo-random simulations. Values are percentages of total games played.

While most of the results are similar to those of the previous experiments, it is notable that with a high pseudo-random turn percentage the AI managed to slightly outperform its opponent while playing as player 2. However, given the performance of the AI as player 1, one can conclude that this simulation approach performs weaker than pseudo-random simulations, albeit only by a small margin. Additionally, the performance drops drastically when the pseudo-random turn percentage reaches a low level, indicating that the few turns which are simulation using pseudo-random simulations are not sufficient to ensure that the entire search spaced is explored, leading to a biased search.

Overall, one can conclude that the examined hybrid simulation approaches are inferior to pseudo-random simulations. However, given the right configuration this approach can deliver a performance almost on par with pseudo-random simulations.

5.3 Comparison between Monte-Carlo Search and MCTS

After investigating both MC Search and MCTS and the effects of various techniques to improve both approaches, an evaluation is needed to assess whether the addition of a search tree to the Monte-Carlo approach has indeed resulted in an improvement of performance. In order to achieve this, an experiment has been performed where the highest performing configuration of a MC AI has been matched against the highest performing configuration of a MCTS AI. Both AIs applied pseudo-random simulations with a simulation-length limit of 10 turns and a total of 800 test games have been played. The results of this experiment can be seen in table 5.29.

	MCTS	
As Player	1	2
won	54.8	52.2
draw	3.6	2.2
loss	41.6	45.6

Table 5.29: Results of a MC AI playing against a MCTS AI. Both AIs applied pseudo-random simulations. Values are percentages of total games played.

The results indicate that the addition of a search tree to the Monte-Carlo approach indeed leads to an increase in performance, albeit only by a small margin. The fact that the performance increase is only small can have several reasons. First, one reason could be the fact that Thurn and Taxis is a non-deterministic game with imperfect information. Hence, it is likely that MCTS needs considerably more search time in order to converge to the best move. Second, even if a move, which has been determined to be the best move in some position, is played, the resulting position might be not advantageous at all, due to the non-deterministic elements of the game. Even if the said move is optimal, the outcome of the non-deterministic elements can still render the outcome of this move to the disadvantage of the player who played it. This means that even though an AI makes better decisions than some other AI, the overall playing strength would not increase as significantly as with for example non-deterministic perfect-information games.

While it is now established that MCTS performs better, an interesting insight would also be how it played better. For this several game specific elements have been analyzed. First, figure 5.30 displays the average amount of routes of different lengths which each AI closed during a game.

Route length	MC	MCTS
3	6.322	6.288
4	0.750	0.857
5	0.681	0.671
6	0.338	0.397
≥ 7	0.050	0.054

Table 5.30: Average amount of routes of specific lengths closed during a game of Thurn and Taxis of a MC AI and a MCTS AI.

With the exception of routes of length 5, the analysis indicates that the MCTS AI prefers routes of longer length slightly more than the MC AI does. While playing routes of a longer length are potentially worth more points, especially at lengths 5 or longer, longer routes are also more risky to construct since one might run out of cards to attach while constructing it, thus forcing oneself to scrap the incomplete route. However, given the fact that the MCTS AI actually did close more routes of larger lengths indicates that it managed to cope with this risk better than the MC AI does.

Another indication of playing strength is the choices a player makes, when selecting an assistant. To illustrate this, figure 5.31 displays the average amount each assistant has been chosen during a game for both AIs.

Assistant	MC	MCTS
None	1.67	1.51
Postal Carrier	5.05	5.48
Postmaster	9.19	10.77
Administrator	7.24	5.61
Cartwright	3.73	3.51

Table 5.31: Average amount of times each assistant has been selected during a game for a MC and MCTS AI.

It is noteworthy that the MC player on average chooses the Administrator more than the MCTS player. The Administrator is commonly used as last resort if there are no usable cards on the display, meaning that a lower average number of uses means that the AI anticipated possible future cards on the display better, and hence had to resort to the Administrator less often. The Administrator can also be used as a disruptive measure, ensuring that an opponent cannot draw cards on the display which might be to critical importance to him. Hence, this assistant still has a potential use, even if the upcoming cards are adequately anticipated. The fact that the administrator has been selected less by the MCTS

player means, that it could select other assistants more often, helping the player to progress more quickly through the game. It was able to draw more cards due to a higher use of the Postmaster, but also was able to lay these cards quicker on the board due to a slightly more frequent use of the Postal Carrier. However, there likely remains room for improvement, since the MCTS player on average selected no assistant 1.51 times during a game. It remains highly doubtful whether choosing no assistant is a good move at all, since one might as well choose the Postmaster instead to draw one extra card, be it just to ensure that the opponent is unable to draw it.

5.4 MCTS against Human Players

In order to gain some perspective how well a MCTS AI actually plays the game Thurn and Taxis, a comparison is needed against an already established player, which can be both human or AI. To establish the playing strength of the developed AI, it has been matched against two human players. The first tested human player, the experimenter himself, plays Thurn and Taxis on and expert level. The second test against a human player has been performed against current reigning Dutch board game champion Maarten Schadd, which enables the comparison against a player at champion level. 20 matches were played against the expert player, while the AI competed in a best-of-five challenge against M. Schadd. The overall results of the matches against the expert player can be seen in table 5.32 while the detailed result of the three challenge games can be seen in table 5.33.

	MCTS	
As Player	1	2
won	0	0
draw	0	0
loss	10	10
average score difference	12.8	11.6

Table 5.32: Results of a MCTS AI playing 20 matches against an expert human player.

The results indicate that the AI neither plays at champion nor expert level, having lost every game. However, the result of the second game against the champion player, a loss with a difference of only one point, indicates that the AI had good chances to win against the champion player, suggesting an at least advanced playing strength of the AI. A similar conclusion can be drawn when inspecting the results against the expert player. The AI lost with an average score difference of roughly 12 points, from which one can conclude that the AI

As Player	Match 1		Match 2		Match 3	
1	Champion	37	MCTS	24	Champion	34
2	MCTS	16	Champion	25	MCTS	17

Table 5.33: Scores of a MCTS AI competing in a best-of-five match against a human player at champion level.

did not play at the same level as the expert, but not significantly weaker as well. The average score differences against the expert player indicate the same playing strength. Overall, the average game length of these experimental games was approximately 45 turns, which in comparison to the measured average game length of 56.1 turns means that the human players needed roughly 5 turns less to cause the ending condition. A 5 turn difference can be equated roughly to a difference of needing one route less to finish the game.

Several game-specific tendencies can be analyzed in order to investigate to what extent the playing style of the AI is similar to that of the human competitors. First, there is the breakdown of the number of closed routes, viewable in table 5.34.

Route length	MCTS	Expert	Champion
3	5.65	2.75	1.3
4	0.75	1.65	0.6
5	0.60	0.7	1.0
6	0.20	0.5	1.0
≥ 7	0.1	0.7	1.3

Table 5.34: Average number per game of closed routes of specific lengths by the MCTS-AI, an expert and champion-level human player.

Several trends are noticeable when inspecting the average route lengths. First, the breakdown of the expert and champion-level human player differ considerably, indicating that there are several successful strategies to play Thurn and Taxis. The champion player mostly played longer routes, indicating a tendency towards receiving points for closing routes of a certain length, while the expert player has an emphasis on shorter routes, indicating a quick playing style. The breakdown of the AI however has a comparatively quite large focus on small routes with only little focus on playing long routes, where as even the expert player has a tendency to play at least one large route during a game on average. Also noticeable is a small increase in routes of length 7 as opposed to routes of length 6 for both human players. The most likely reason for this is that if a player has already invested 6 cards in a route, he might as well attach a

seventh card in order to get points from the more lucrative point stack for routes of length 7. However, the MCTS-AI fails to exhibit this behavior. Overall, the playing style of the AI vaguely resembles the style of the expert player, with a lacking tendency towards longer routes.

Next, the average assistant selection of the AI and the human players will be displayed in table 5.35.

Assistant	MCTS	Expert	Champion
None	0.65	0.25	0.3
Postal Carrier	5.25	4.6	4.3
Postmaster	10.05	9.1	8.0
Administrator	4.0	6.85	9.3
Cartwright	2.77	1.9	1.0

Table 5.35: Breakdown of chosen assistants per game of the MCTS-AI, an expert and champion-level human player.

Here, unlike the average route lengths, there is a stronger resemblance of the MCTS-AI with one of the playing styles of the human players, in this case the expert player. Both have the tendency to pick the Postmaster significantly more often than other assistants and both choose the Postal Carrier and the Administrator on a regular basis, albeit with a small difference in preference between these two assistants. Unlike the MCTS-AI or the expert player, the champion player actually has a large tendency towards the Administrator, a playing style which increases the chances that a player can draw the cards he needs, so that he can play long routes. Even though the strongest preference of the champion-level player is towards the Administrator, a strong preference towards the Postmaster exists as well, confirming that the Postmaster is indeed a strong assistant. It is also noteworthy that both the expert and champion player chose to not select any assistant only on rare occasions, indicating that this option only rarely is useful to the player. The AI exhibited a similar behavior to that of the human players in this regard, having selected no assistant only 0.65 times per game on average.

Other similarities in playing style can be seen as well when inspecting the breakdown of card-attach moves of the different players, displayed in table 5.36.

It is notable that in this comparison, the playing style of the AI only barely differs from the styles of the human competitors. The most notable difference is that the AI on average scrapped at least one route per game, where as both humans managed to avoid route scrapping better, with the champion player not

	MCTS	Expert	Champion
Route scrapped	1.6	0.6	0.0
Card attached	13.7	15.8	17.6
Route closed	7.3	6.3	5.3

Table 5.36: Breakdown of card-attach moves per game of the MCTS-AI, and expert and champion-level human player.

having scrapped a single route at all.

One last difference which is worth investigating is the difference between the players considering how often they select specific cards on average, viewable in table 5.37. While the most cards get drawn approximately once per game on average, with a few cards being drawn more often due to their strategic position, there is one striking difference between the AI and the human players, which can give an indication to why the AI performs weaker and the reason behind this. This difference concerns the card Lodz, which is rarely drawn by the AI. As a reminder, Lodz is the only city in the black colored region, and hence crucial in order to receive points for having a city in each colored region. In addition, Lodz is a difficult city to access, since it only has one connection. However, the black region in itself is not worth any points, making Lodz a city with only a long term gain, since placing a house in each colored region is usually accomplished at a later stage in the game. If one were to perform a search to determine whether drawing Lodz is the best move at a certain position, then this can only be determined if the search is deep enough. Having barely drawn Lodz at all, one can conclude that the AI did not recognize the long-term gain of Lodz, and thus regularly failed to receive points for having a house in each region. This in turn gave a regular lead of 6 points for the human players, who mostly did accomplish the placement of a house in each region.

The factor which is most likely to be responsible for the failure of detecting moves with only long-term rewards is the limiting of the simulation lengths. While this technique did increase the strength of MC Search and MCTS considerably, by drastically increasing the amount of drawn samples, it also caused simulations to stop before the benefits of long-term gains could be detected. Hence, only moves with short and mid-term gains are selected, leaving the opponent to freely benefit from long-term prosperous moves. Due to the high branching factor of the game, it is extremely unlikely that a tree-search approach would detect the advantage of these moves. Hence, an obvious solution to this problem would be to increase the simulation length combined with some way to compensate for the loss of drawn samples per second, for instance increasing the allowed computation time for the AI.

Overall, one can conclude that the AI played the game Thurn and Taxis at

an advanced level, being similar in many aspects to expert and champion-level human players, while still having some shortcomings.

Player	MCTS	Expert	Champion
Random Card	3.7	3.7	3.0
Mannheim	1.1	1.1	1.0
Carlsruhe	1.25	1.0	0.6
Freiburg	1.3	1.35	2.0
Basel	0.65	1.2	1.0
Zuerich	1.1	1.45	1.3
Stuttgart	2.0	1.2	2.3
Ulm	1.6	1.45	1.0
Sigmaningen	1.75	1.45	1.0
Innsbruck	1.0	1.2	1.6
Wuerzburg	1.1	0.95	1.6
Nuernberg	1.6	1.7	0.6
Ingolstadt	2.25	1.1	1.6
Augsburg	1.4	1.15	0.6
Kempten	1.5	1.45	1.0
Regensburg	1.65	1.5	2.3
Passau	1.2	1.35	2.0
Muenchen	2.1	1.15	1.0
Salzburg	1.3	1.35	1.3
Linz	1.1	1.1	0.6
Budweis	0.7	1.1	0.6
Pilsen	1.05	1.55	1.3
Lodz	0.35	1.25	1.0

Table 5.37: Average amount of times a card will be selected per game, by the MCTS-AI, an expert and champion-level human player.

Chapter 6

Conclusions and Future Research

In this chapter, the conclusions of this research will be presented. Section 6.1 contains the answers to the research questions and problem statement, which have been stated in section 1.3. Section 6.2 discusses the possibilities of further research, which can be conducted in this area.

6.1 Problem Statement and Research Questions Revisited

This section will discuss the answers to the problem statement and research questions. First, the research questions will be answered in subsection 6.1.1. Then, these will be used in order to answer the problem statement of this thesis in subsection 6.1.2.

6.1.1 Research Questions Revisited

What is the game complexity of the game Thurn and Taxis?

The answer to this research question was the subject of chapter 3. The state-space complexity of the 2-player version of Thurn and Taxis has been determined to be approximately 10^{66} and the game-tree complexity has been estimated at 10^{240} using the analysis of a large amount of test games, played by a MCTS-AI in self-play. Note that the state-space complexity is an overestimation, with the precise complexity likely being a few powers of 10 smaller.

Given these results, one can see that the complexities of the 2-player version of Thurn and Taxis are similar to those of the game of Shogi. Hence, one can conclude that it is unlikely that the 2-player version of Thurn and Taxis will

be solved within the next decade. However, creating a competent AI player for this game should still be possible.

To what extent is it possible to use domain knowledge in order to improve the Monte-Carlo sampling process?

In section 4.4, several techniques have been investigated to improve either the sample quality through the application of domain knowledge, or to increase the quantity of drawn samples. Two techniques were proposed to increase the sample quantity, being limiting simulation lengths and game-progression-based simulation cuts. Game-progression-based simulation cuts have been shown to be ineffective when applied to random simulations and only to have limited effects when applied to other simulation techniques. The technique is also ineffective when combined with limited simulation lengths. Limited simulation lengths proved to be an effective technique. However, as the results against human players indicate, this technique also restricts the hindsight of the AI, causing it to ignore moves which only have long-term gains.

Several techniques have been investigated to increase sample quality, being pseudo-random simulations, a mechanism for pseudo-random simulations using binary heuristics, and hybrid simulations. Each of these techniques have been shown to significantly increase the playing-strength compared to random simulations, with pseudo-random simulations being the strongest performing approach.

To what extent does the addition of constructing a search tree contribute to the strength of a Monte-Carlo AI?

The addition of a search tree, explained in section 4.2, appeared to give a small increase to the strength of a Monte-Carlo AI according to the results of section 5.3. While the addition of a tree search to the Monte-Carlo approach has been known to grant significant increases of playing strength in classic games, such as Go, in Thurn and Taxis the observed difference is small likely due to the fact that the game incorporates hidden information and elements of chance.

6.1.2 Problem Statement Revisited

Now that all the research questions have been addressed, an answer to the problem statement can be given.

To what extent can Monte-Carlo techniques be applied in order to create a competitive AI for modern non-deterministic board games with imperfect information?

It is possible to utilize Monte-Carlo techniques in order to create a competitive AI for the game Thurn and Taxis. Since during the course of this research

an AI of advanced strength has been created in a relatively short time frame, we feel it is possible that an AI using Monte-Carlo techniques can achieve an expert level or higher as well.

Monte-Carlo Tree Search utilizing the UCT selection strategy has been shown to perform stronger than standard Monte-Carlo search, albeit with a small difference. Amongst the different investigated improvements, limiting simulation lengths and pseudo-random simulations were the most promising ones.

6.2 Future Work

To this day, only little research has been done in the field of modern board games, with this thesis being the first research conducted for the game Thurn and Taxis. Hence, many areas remain where further research is due.

First, this thesis only addressed the 2-player version of the game. Hence, one area of further research would be the investigation of the 3- or 4-player version of Thurn and Taxis. There, it would be interesting to investigate whether MCTS and the possible improvements are as effective in a multi-player environment as in a 2-player one. Also, the game complexities of these multi-player variants would be needed to be determined, as the likely higher complexities will make it more difficult for an AI to perform well.

While there are likely more possible ways to add domain knowledge to the simulation process, another area which has been untouched in this research was the addition of domain knowledge to the selection step of the MCTS approach, by for instance progressive pruning [12]. Another issue that requires more research is the fact that moves which have only long-term rewards are not recognized as potential good moves, when the technique of limited simulation lengths is applied. This is a crucial element to be addressed in order to push the performance of an AI to expert level.

Bibliography

[1] W. Behringer. *Thurn und Taxis, Die Geschichte ihrer Post und ihrer Unternehmen*. Piper, 1990.

[2] D. Billings and M. Kan. A tool for the direct assessment of Poker decisions. *ICGA*, 29(3):119–142, 2006.

[3] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in Poker. In *Proceedings of the 15th International Conference on Artificial Intelligence*, pages 493–499, 1998.

[4] D. Billings, L. Pena, J. Schaeffer, and D. Szafron. Using probabilistic knowledge and simulation to play Poker. In *Proceedings of the 16th International Conference on Artificial Intelligence*, pages 697–703, 1999.

[5] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of Poker. *Artificial Intelligence*, 134(1–2):201–240, 2002.

[6] B. Bouzy. Associating domain-dependent knowledge and Monte Carlo approaches within a Go program. *Information Sciences, Heuristic Search and Computer Game Playing IV*, 175(4):247–257, 2005.

[7] B. Bouzy and T. Cazenave. Computer Go: an AI oriented survey. *Artificial Intelligence:*, 132(1), 2001.

[8] L. Branca and S. J. Johansson. Using multi-agent system technologies in Settlers of Catan bots. In *Agent-based Systems for Human Learning and Entertainment*. Blekinge Institute of Technology, 2007.

[9] M. Campbell, A. J. Hoane Jr., and F. H. Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

[10] T. Cazenave. Multi-player go. In H. Jaap van den Herik, editor, *Proceedings of the 6th International Conference on Computers and Games*, pages 50–60. Springer, 2008.

[11] T. Cazenave and B. Helmstetter. Combining tactical search and Monte-Carlo in the game of Go. In *In Symposium on Computational Intelligence and Games*, pages 171–175. IEEE, 2005.

[12] G. M. J-B. Chaslot, M. H. M. Winands, J. W. H. M. Uiterwijk, H. J. van den Herik, and B. Bouzy. Progressive strategies for Monte-Carlo tree search. In P. Wang, editor, *Proceedings of the 10th Joint Conference on Information Sciences (JCIS)*. World Scientific Publishing Co. Pte. Ltd., 2007.

[13] G. M. J-B. Chaslot, J-B. Hoock, A. Rimmel, O. Teytaud, C-S. Lee, M-W. Wang, S-R. Tsai, and S-C Hsu. Human-computer revolution 2008. *ICGA*, 31(3):179–185, 2008.

[14] G. M. J-B. Chaslot, M. H. M. Winands, I. Szita, and H. J. van den Herik. Cross-entropy for Monte-Carlo tree research. *ICGA*, 31(3):145–156, 2008.

[15] K-H. Chen and P. Zhang. Monte-Carlo Go with knowledge-guided simulations. *ICGA*, 31(2):67–76, 2008.

[16] R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, editors, *Proceedings of the 5th International Conference on Computer and Games*, volume 4630 of *Lecture Notes in Computer Science (LNCS)*, pages 72–83. Springer, 2007.

[17] R. Grimbergen. A survey of Tsume Shogi programs using variable depth search. In *Computers and Games: Proceedings CG'98. LNCS 1558*, pages 300–317. Springer Verlag, 1998.

[18] G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*, volume 12 of *Combinatorial Optimization*. Springer, 2007.

[19] T. Hauk, M. Buro, and J. Schaeffer. Rediscovering *-minimax search. In H.J. van den Herik, editor, *Computers and Games*, pages 36–50. Springer Science & Business, 2004.

[20] M. N. Katehakis and A. F. Veinott Jr. The multi-armed bandit problem: Decomposition and computation. In *Mathematics of Operations Research*, volume 12, pages 262–268. INFORMS, 1987.

[21] G. Kendall, A. Parkes, and K. Spoerer. A survey of NP-complete puzzles. *ICGA*, 31(1):13–35, 2008.

[22] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning*, pages 282–293, 2006.

[23] H. Matsubara, H. Iida, and R. Grimbergen. Natural developments in game research. *ICGA Journal*, 19(2):103–112.

[24] J.A.M. Nijssen. Using intelligent search techniques to play the game Khet. Master thesis, Maastricht University, 2009.

[25] M. Pfeiffer. Reinforcement learning of strategies for Settlers of Catan. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004.

[26] S. Russel and P. Norwig. *Artificial Intelligence*. Pearson Education International, second edition, 2003.

[27] M. P. D. Schadd, M. H. M. Winands, J. W. H. M. Uiterwijk, H. J. van den Herik, and M. H. J. Bergsma. Best Play in Fanorona leads to Draw. *New Mathematics and Natural Computation* 4, (3):369–387, 2008.

[28] B. Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*, 134(1–2):241–275, 2002.

[29] N. Sturtevant. *Multi-Player Games: Algorithms and Approaches*. PhD thesis, University of California, 2003.

[30] I. Szita, G. M. J-B. Chaslot, and P. Spronck. Monte-Carlo tree search in Settlers of Catan. In *Proceedings of the 12th Advances in Computer Games*, 2009.

[31] G. Tesauro. Programming Backgammon using self-teaching neural nets. *Artificial Intelligence*, 134:181–199, 2002.

[32] A. M. Turing, C. Strachey, M. A. Bates, and B. V. Bowden. Digital computers applied to games. In B. V. Bowden, editor, *Faster Than Thought*, pages 286–310. Pitman, London, 1953.

[33] H. J. van den Herik and M. H. M. Winands. Proof-number search and its variants. In H. R. Tizhoosh and M. Ventresca, editors, *Oppositional Concepts in Computational Intelligence*, chapter 6, pages 90–118. Springer, 2008.

[34] H. J. van den Herik, J. W. H. M. Uiterwijk, and J. van Rijswijck. Games solved: now and in the future. *Artificial Intelligence*, 134(1-2), 2002.