AN MCTS AGENT FOR TICKET TO RIDE

Carina Huchler

Master Thesis DKE 15-18

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Faculty of Humanities and Sciences
of Maastricht University

Thesis committee:

Dr. Mark H. M. Winands
Dr. ir. Jos W. H. M. Uiterwijk

Maastricht University
Department of Knowledge Engineering
Maastricht, The Netherlands
July 8, 2015

Preface

This thesis was written at the Department of Knowledge Engineering of Maastricht University. This thesis discusses the implementation of Ticket to Ride in the framework of Monte-Carlo Tree Search. I would like to express my thanks to several people. First of all, I would like to thank my supervisor Dr. Mark Winands, for his help during my thesis. He gave me a lot of insightful advice and helpful remarks in writing my thesis. I would also like to thank my boyfriend and fellow student, Emanuel Oster, for helping me to find my bugs and supporting me throughout the thesis. Finally, I would like to give my gratitude to my family and friends for supporting me during my thesis. I want to thank them not only for playing countless games of Ticket to Ride, but also for giving me moral support.

Carina Huchler
Aachen, July 2015

Abstract

The implementation of AI players for games exists at least since 1956 where the first chess player MANIAC was implemented. Since this time AI players got more and more attention. For over 14 years now Monte-Carlo Tree Search (MCTS) got investigated.

This thesis explains Ticket to Ride and the implementation of the game in the Monte-Carlo Tree Search (MCTS) framework. However, the game engine also had to be implemented as well as several enhancements for the MCTS algorithm. In the game each player plays a train building company connecting certain cities as fast as possible. To know which routes to build each player gets certain destination tickets, which he has to fulfill until the game ends. These destination tickets are only known to the player himself.

Ticket to Ride imposes several problems, which have to be considered. The board itself and each possibility a player has each round are known. The game also contains imperfect information, because a player does not necessarily know the cards of his opponents as well as the destination tickets, he needs to fulfill. The game also contains chance events when a player chooses to draw a card or new destination tickets.

This thesis includes a more detailed description of the rules of Ticket to Ride as well as a calculation of the state-space complexity and game-tree complexity. MCTS, the handling of imperfect information and chance events and several other enhancements to MCTS are discussed. This enhancements are the Single-Observer, Multiple-Observer, determinized UCT, voting, progressive unpruning, progressive bias, early termination of the playout step and an ϵ-rule based strategy.

Since no usable implementation of the game exists, the problems and certain specifications of the implementation have to be discussed as well. The conditions under which the experiments were performed are discussed as well. The results show that the combination of progressive unpruning and progressive bias works best with using either the Single-Observer or the Multiple-Observer as its base.

The remaining enhancements were tested as well, but showed no improvement over the combination of the two progressive approaches. Some enhancements like the determinized UCT, the voting player or the playout strategy performed even worse than its Single-Observer variant.
It is also shown how well imperfect information is handled in the case of Ticket to Ride by comparing it to two different types of the cheating player. Both of them are able to access the knowledge of all players. One cheating player also has fixed chance events. The Multiple-Observer has a similar result when playing against the cheating player. Both agents were tested with and without the combination of progressive bias and progressive unpruning. Both agents were able to win $\sim 40\%$ of all games. Therefore both agents perform well considering their disadvantage.

Contents

Preface iii

Abstract v

Contents vii

1 Introduction 1
- 1.1 Games and AI . . . 1
  - 1.1.1 History of AI players . . . 1
  - 1.1.2 Search Algorithms . . . 1
  - 1.1.3 Success of Monte-Carlo Search as an AI player . . . 2
- 1.2 Ticket to Ride . . . 3
- 1.3 Problem Statement and Research Questions . . . 3
- 1.4 Thesis Outline . . . 3

2 Ticket to Ride 5
- 2.1 Rules . . . 5
- 2.2 Strategy . . . 7
  - 2.2.1 Counting Cards . . . 7
  - 2.2.2 Choosing Destination Tickets . . . 7
  - 2.2.3 Claiming Tracks of Importance to Opponents . . . 8
  - 2.2.4 Building a Continuous Route . . . 8
- 2.3 Stand Alone Editions and Expansions . . . 8
- 2.4 Complexity . . . 8
  - 2.4.1 State-Space Complexity . . . 8
  - 2.4.2 Game-Tree Complexity . . . 9

3 Monte-Carlo Tree Search 11
- 3.1 Flat Monte-Carlo . . . 11
- 3.2 Monte-Carlo Tree Search . . . 11
- 3.3 Upper Confidence Bounds for Trees . . . 12
- 3.4 Enhancements . . . 12
  - 3.4.1 Progressive Unpruning . . . 12
  - 3.4.2 Progressive Bias . . . 13
  - 3.4.3 Number-of-Visits-Dependent Strategy . . . 13
  - 3.4.4 Determinization . . . 13
  - 3.4.5 Cheating MCTS . . . 13
  - 3.4.6 Single-Observer MCTS . . . 13
  - 3.4.7 Multiple-Observer MCTS . . . 14
  - 3.4.8 Determinized UCT . . . 15
  - 3.4.9 Early Terminations . . . 15
  - 3.4.10 ϵ-based strategies . . . 15

4 MCTS in Ticket to Ride **17**
4.1 Nodes . . . 17
4.1.1 Actions . . . 17
4.2 Choosing Destination Tickets . . . 17
4.2.1 Start of the game . . . 17
4.2.2 During the game . . . 18
4.3 Hidden Information . . . 18
4.3.1 Train Cards . . . 18
4.3.2 Destination Tickets . . . 19
4.4 Chance . . . 19
4.5 Upper Confidence Bounds for Trees . . . 20
4.6 Selection Strategy . . . 20
4.6.1 Heuristic Knowledge . . . 20
4.6.2 Progressive Bias And Unpruning . . . 21
4.7 Playout Policies . . . 21
4.7.1 Random . . . 21
4.7.2 Early Termination . . . 21
4.7.3 ϵ-Rule Based Strategy . . . 21
4.8 Finding the Shortest Path . . . 22

5 Experiments and Results **23**
5.1 AI Players . . . 23
5.2 Experimental Setup . . . 23
5.2.1 Player Setup . . . 23
5.2.2 Confidence Bounds . . . 24
5.2.3 Environmental Setup . . . 24
5.3 Single Observer . . . 24
5.4 Cheating Player . . . 25
5.5 Enhancements . . . 26
5.5.1 FSO vs. Enhancements . . . 26
5.5.2 Improving Flat Monte-Carlo with Enhancements . . . 26
5.5.3 SO vs. Enhancements . . . 27
5.5.4 Combinations . . . 28
5.6 Multiple-Observer MCTS . . . 28
5.7 Multiple Trees MCTS . . . 29
5.7.1 SO vs. Determinized UCT and Voting . . . 29
5.7.2 Determinized UCT . . . 29
5.7.3 Voting . . . 30
5.8 Early Termination Approach . . . 31
5.9 Number of Visits vs. Ratio . . . 32
5.10 Combining PUB with other Approaches . . . 32
5.11 Cheating Player vs. Different Agents . . . 33
5.12 Completing destination tickets . . . 34

6 Conclusion and Future Research **35**
6.1 Thesis summary . . . 35
6.2 Answering the Research Questions . . . 36
6.3 Answering the Problem Statement . . . 37
6.4 Future Research . . . 37

References **39**

Chapter 1

Introduction

This chapter shows the usage of AI in games. It gives a brief introduction to search algorithms and the imperfect-information game Ticket to Ride. It also introduces the research questions and gives a brief overview of this thesis.

1.1 Games and AI

Over the last few years game-playing agents have become more and more popular. You did not play games only at home with your friends and family or in your local game shop, but also on your computer or smart phone against other persons all over the world but also against the computer itself.

For playing against the computer, intelligent agents have to be developed. Such agents can have different goals. Such an agent can either play the game as good as possible or it tries to reach a certain kind of level to play against.

These agents were developed for different kinds of games like chess, checkers, Go or poker. The more common AI methods are search algorithms. Most of these techniques need an evaluation function. The better such an evaluation function classifies each possible movement, the better it plays the game. Such evaluation functions can be hard to find.

1.1.1 History of AI players

In 1956 the first chess program, called MANIAC, was able to defeat a novice player in a simplified chess game (Douglas, 1978). Since then more agents were implemented. The supercomputerDeep Blue was the first who was able to defeat a world champion under normal chess tournament conditions against Garry Kasparov in 1997. In this match Deep Blue had to play six times, Deep Blue won 3.5–2.5 (Campbell, Hoane Jr., and Hsu, 2002). Around this time people focused more on researching on alternatives for AI engines for other games.

Since then Go has become popular as a research topic. Building a strong engine has turned out to be difficult, because Go requires a more complex evaluation function than chess. Monte-Carlo Evaluations was first introduced as a Go player by Brügmann (1993). With this introduction an evaluation function was no longer required. 13 years later it was further developed as Monte-Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006, Coulom, 2007a). Since then more players were implemented, which were even able to compete against humans on expert level. In 2014 the MCTS program Crazy Stone won against the 9 dan player Norimoto Yoda (Wedd, 2015). Norimoto Yoda had a handicap of 4 stones.

1.1.2 Search Algorithms

There are many different search algorithms. A search algorithm is normally used to fulfill an objective. These objectives can have different forms. It is either used to find a path to an objective or to find a winning state. There exist different kinds of search algorithms. In this section a brief overview about seven different search algorithms is given: Minimax, $\alpha\beta$-pruning, $\max^n$, Expectimax, Dijkstra, A^* and

MCTS.

Minimax
Minimax is an adversarial search algorithm (Von Neumann and Morgenstern, 1944). The algorithm gives each leaf node a value by using a heuristic function. The algorithm assumes that each player plays as best as possible. It does so by assuming that the root player or max player maximizes the results, while the other player tries to minimize it. At each node the best value according to the belonging player gets chosen.

$\alpha\beta$-pruning
$\alpha\beta$-pruning improves the Minimax algorithm by pruning branches. A branch is pruned when it is proven that it cannot influence the upcoming decisions (Knuth and Moore, 1975).

max^n
The max^n algorithm (Luckhart and Irani, 1986) can be used for games with more than two players. It works similar to the Minimax algorithm. The biggest difference is that each player has his own value, and that each player maximizes it.

Expectimax
Expectimax (Michie, 1966) is used for games with a chance element. It changes the Minimax algorithm such that it also contains a chance node. The Expectimax algorithm uses the chance element to calculate the value of a node by multiplying the values of the children with their probabilities.

Dijkstra
Dijkstra is a single agent graph search algorithm (Dijkstra, 1959). It tries to find the path with the lowest cost between a starting node and each other node. Each node, except for the starting node, starts with a cost for the path equal infinity. The algorithm starts at the starting node. It updates all nodes, which are neighbors and were not visited yet. The updated cost is equal to cost of the distance. It then chooses the node with the lowest cost and removes it from the list of non-visited nodes. This node is then the current node and all neighbors are updated with the tentative distances.

***A****
A^* is a single agent pathfinding algorithm. It is used to find the path with the lowest cost between two nodes in a graph. To find such a path it estimates at each node the total cost of movement and tries to keep it as low as possible. The graph tries to build a way over the nodes, which have the lowest estimated movement cost (Hart, Nilsson, and Raphael, 1968).

Monte-Carlo Tree Search
MCTS is a best-first search algorithm (Coulom, 2007a). The algorithm is based on simulating the game multiple times and then choosing the most promising option as your move or turn. The algorithm builds a tree. Each node contains the information for the choosing process.

1.1.3 Success of Monte-Carlo Search as an AI player

One of the first successful Monte-Carlo search was called GIB. It was a bridge player implemented by Ginsberg (2001). It competed in the 1998 and the 2000 World Computer Bridge Championships. It won all games 1998 and all but one in the 2000 championship.
A successful implementation of MCTS for an Euro style board game was done by Szita, Chaslot, and Spronck (2010) in Settlers of Catan. The MCTS-10,000 player performed the best with winning 49% of all played games against several other players.
It also has shown some success in handling imperfect information (Cowling, Powley, and Whitehouse, 2012). They have tested several games with imperfect information. The games tested were Lord of the Rings: The Confrontation (LOTR:C), Dou Di Zhu and the Phantom (4,4,4) game. The approach was also tested against humans in the case of LOTR:C. There it was able to win 16 of the 32 games played the human player. When the player switched sides the AI player, the AI player was able to win 14 of the 32 games.

It also worked well for several other games like Amazons (Lorentz, 2008) and Scotland Yard (Nijssen and Winands, 2012).

1.2 Ticket to Ride

In this thesis the game of Ticket to Ride (TTR) is investigated. Each player takes over the job of a train building company in America and tries to build train tracks to connect certain cities as fast as possible. Ticket to Ride is a Euro style board game. This means that it uses rather simple rules, has only indirect player interaction and no one gets eliminated before the game ends. The game was invented by Alan R. Moon and was published 2004 by Games Of Wonder. It has won several awards and the rules have been translated into 11 other languages. The game also has several spin-offs, where you can play the game in other settings with normally only a few more rules. These rules can involve other ways to get victory points or special building rules for tracks. Different expansions can change the number of players, but two to five players is the most common one.

Each player has destination tickets on his hand, which are unknown to his opponent. A player can also draw train cards. These cards are also only known to him. Therefore TTR imposes several problems to the MCTS algorithm. These problems are imperfect information as well as chance events.

1.3 Problem Statement and Research Questions

The goal of this thesis is to implement an MCTS player, which is able to play Ticket to Ride as strong as possible. This leads to the following problem statement:

How can an MCTS player for TTR be developed?

To address the problem statement, this thesis answers the following three research questions.

Are Monte-Carlo methods applicable to Ticket to Ride?
It has been shown that Monte-Carlo methods work for many different types of games (Browne *et al.*, 2012), but it has to be tested that it also works for the imperfect-information game Ticket to Ride. It has to be tested as well, if MCTS is able to handle the stochastically of TTR.

How can MCTS handle the imperfect-information game in the domain of TTR?
Only a little amount of research has been done on handling imperfect-information in MCTS. This makes it an interesting topic to investigate further, how the imperfect information influences the performance of the game. This will be done, by testing several strategies, which are used to handle imperfect information. The tested strategies are the Single-Observer, the Multiple-Observer and determinized UCT. These strategies were all introduced by Cowling *et al.* (2012). The last tested strategy is the voting approach (Nijssen and Winands, 2012).

Which enhancements can improve the MCTS player?
It also has to be tested, which enhancements to the implemented player improve its effectiveness. The enhancements tested in this thesis are progressive unpruning (Chaslot *et al.*, 2008 and Coulom, 2007b), progressive bias (Chaslot *et al.*, 2008) and an ϵ-rule based strategy.

1.4 Thesis Outline

The outline of this thesis is:

Chapter 1 gives an introduction to the thesis. It starts with introducing the usage of AI in games. This also covers the success of MCTS in different games. Then it gives an overview of different search algorithms used in this thesis. Afterwards the general idea and principles of Ticket to Ride are explained. This is followed by the problem statement and research questions.

Chapter 2 is about Ticket to Ride. The rules of the game are explained and different strategies are shown. It gives a brief introduction of the different stand-alone editions and expansions for the game. The chapter closes with the calculation of the state-space complexity and game-tree complexity.

Chapter 3 gives an introduction in MCTS. It begins with explaining Flat Monte-Carlo. Afterwards the enhancement of Flat Monte-Carlo, Monte-Carlo Tree Search, is explained. Then the selection strategy Upper Confidence Bounds for Trees is closer analyzed. Then the different enhancements, which were used in this paper, are introduced. These enhancements contain progressive unpruning and progressive bias as well as determinization, cheating MCTS, Single- and Multiple-Observer, determinized UCT, early termination and ϵ-based strategies.

Chapter 4 shows the connection of Ticket to Ride and MCTS. This chapter shows the special implementations, which had to be done, such that Ticket to Ride can be analyzed by MCTS. Therefore the special node structure is discussed first. Then implementation of choosing destination tickets, an important part of Ticket to Ride, is explained. Afterwards it is explained how the different types of hidden information are handled.
The chapter also shows the implementation of UCT and explains the specifications of different enhancements. Therefore the heuristic knowledge used for progressive bias and progressive unpruning is explained. Different playout policies are also discussed. These playout policies cover the random playout, early termination and the used ϵ-rule based strategy. It closes with an explanation on how to calculate the shortest path during a game.

Chapter 5 describes the experiments which are performed. It starts with introducing the general players. Afterwards the experimental setup is shown. This is done by not only showing the setup of the different used players, but also by introducing confidence bounds, which is later used to validate the results. It also gives the specifications of the environmental setup. Afterwards the different experiments are shortly explained and there results are given and explained.

Chapter 6 concludes the thesis. A brief summary of the results of the thesis is given. Then the research questions and the problem statement are answered. The thesis closes with a proposal of future research possibilities.

Chapter 2

Ticket to Ride

This chapter explains the rules and strategies of Ticket to Ride. It gives a brief overview over the different standalone editions and expansions of Ticket to Ride and examines its complexity.

2.1 Rules

Ticket to Ride can be played by two to five players in the original game. The game consists of a gaming board (see Figure 2.1), 12 train cards in eight different colors each, 14 locomotives counting as any color, 30 destination tickets and 45 trains for each player. All colored cards and the locomotives are shuffled. Together they form one stack of train cards. The destination tickets are shuffled as well. Each player gets dealt four train cards and three destination tickets, from which each player has to keep at least two. The destination tickets that are not chosen return to the bottom of the destination tickets stack. Also five open train cards are visible to all players at any time. The goal of the game is to gain the most victory points. A player can get victory points in the three following ways. First the player can build a track between to cities, second the player can build a continuous path between the two listed cities from a destination ticket and third a player can own the longest continuous path. This is a bonus objective, which can change in different game modes. A player can also lose points. A player loses points if he has not fulfilled a destination ticket at the end of the game. The player loses the value, which is written on the ticket. During each turn a player has up to three options. He can draw cards, build a track between two cities or take new destination tickets.

Drawing and Choosing Cards
A player can obtain up to two cards in his turn. A player can either choose from the cards visible to all players, or a player can draw hidden cards from the card stack. If the player chooses a visible card, the card is immediately replaced by the top most card of the card stack. If the player chooses a locomotive card openly, he cannot draw another card. A player also cannot choose a locomotive as his second openly drawn card. Another special rule is that as soon as three locomotives can be openly chosen, all open train cards get replaced. The open cards from before go onto the discard pile.

Claiming a Track
A player can build a track by paying the number and color of a track visible on the board. If the color of a track is gray, the player can choose any color to build this road. The player has to give the number of cards visible on the board of the chosen color or can replace any number of train cards with the same number of locomotives. Afterwards this player puts the same number of trains on the claimed track. If a track has two options (see Figure 2.2), he can choose one of them and puts his train on the chosen color. The other track can no longer be claimed in a game except this game is played by more than three players. The player has to use the remaining color for building the connection. If a track is claimed, this connection is gone and cannot be claimed by another player. If two cities have two connections, the same player can only claim one of them when playing with four or more players. The player also immediately

Figure 2.1: Two tracks connecting two cities

Number	Points
1	1
2	2
3	4
4	7
5	10
6	15

Table 2.1: Points for claimed tracks. Left side of the table states, how many trains a player needs for a track. On the right side the according victory points are shown.

gains the victory points for the track he claimed. The number of victory points is determined by the number of trains the player needs to build the track (see Table 2.1).

Draw Destination Tickets

Figure 2.2: Two tracks connecting two cities

A player can also draw three destination tickets. He has to choose one but he can keep two or all three of them.

The game starts to end if one player has less than three trains left at the end of his turn. Afterwards each player, including the player who owns less than three trains, has one turn left. In the end all destination tickets are revealed and the bonuses or penalties for destination tickets are added to the

victory points gained by claiming tracks. At last the longest continuous path is calculated and each player who has it gets 10 additional victory points. The player with the most victory points wins. If one or more players are tied for the most victory points, the player with the most fulfilled destination tickets wins, if there is still more than one player the player who holds the longest continuous path wins. The rules can be also found online at the Days of Wonder homepage (Moon, 2015).

2.2 Strategy

2.2.1 Counting Cards

A good strategy is to keep a good overview over the remaining available cards. To do so, the player should pay attention to the cards other players draw openly or pay while claiming tracks. Each type of card, like locomotives or white trains, only occurs a certain number of times and can so influence the tracks a player can claim. The cards, which are known to the player, are open cards, discard pile, his own cards and chosen cards by the opponents. With this knowledge the player can make an assumption, which cards are on the hand of the other player and which cards are still in the stack. As a consequence the player can estimate for which tracks he can still get enough cards and which color the player should choose.

2.2.2 Choosing Destination Tickets

For the game it is quite important, which kinds of tracks a player takes during which point in the game.

At the Beginning of the Game

The first destination tickets a player chooses are important, because they influence the tracks a player is going to build during the game. A player should take one of the more valuable destination tickets. Destination tickets with a high value lead to a wide distribution over the board, and so it is more likely that destination tickets, a player gets later in the game, are easier or already complete. If one of the other routes is close by and can be incorporated into the first destination ticket, it should be chosen as well. This way a player can fulfill more destination tickets while building fewer tracks. If the last destination ticket is not connectable to the other destination tickets a player should not take it, so he can draw new destination tickets earlier in the game and is able to get destination tickets, which are nearer to his built path and as such easier to build.

During the Game

A player can follow two different strategies for choosing destination tickets during the game. In the first strategy the player takes new destination tickets as early as possible, to incorporate more destination ticket objectives when he chooses which tracks to claim. The other strategy is to choose new destination tickets as soon as the player fulfilled his existing destination tickets or a player has no possibility to complete the remaining destination tickets. Both strategies have advantages and disadvantages. The advantage of choosing destination tickets early in the game is that a player can make a better plan of which tracks to take. The disadvantage is that the player may distract himself too much, and is not able to fulfill the bigger destination tickets until the game ends. The advantage of choosing destination tickets after a player completed all other destination tickets is that the player can focus on completing this new drawn destination tickets. Another advantage could be that the player gets destination tickets, which are easily or already fulfilled. This strategy also has a disadvantage. The disadvantage is that the player may not be able to fulfill these new drawn destination tickets, because other players finish the game.

At the Last Turn of the Game

At the last turn a player has the same options as in any other turn during the game. If a player has suffered great losses due to unfilled destination tickets, but has built his tracks in such a way that it covers a lot of cities, it could be plausible to draw new destination tickets, to cover the losses he made due to unfilled destination tickets. But this strategy contains gambling due to the fact that you can only estimate, which destination tickets are left and which destination tickets are owned by other players.

2.2.3 Claiming Tracks of Importance to Opponents

Another strategy is to claim tracks, which are of great importance to other players. Such a track could be a track, which connects two different paths. Opponents then have to estimate of how much value this track is for the opponent and if it is worth paying the resources and spending a turn. The advantage of such a move is that the opponent has to spend more resources to connect these two paths and maybe does not gain the possibility to do so. The disadvantage is that the player has to give up resources of his own and that the track could be of no to little value for the other player.

2.2.4 Building a Continuous Route

The last strategy is that a player tries to build a track connecting to one of his routes, so that an opponent does not build such tracks to the player's disadvantage. This is especially true for the games with two or three players, because all tracks can only use one of their connections. In games with more players, more possibilities become available for certain connections. It can be plausible to skip bigger tracks, if the player is sure that such tracks are not useful for other players. Most players would not build such tracks, because the resources, such a move costs, are too high and could be of more value to the player elsewhere.

2.3 Stand Alone Editions and Expansions

After the first game has been released, four different stand-alone editions were published. The game also has six expansions. Two of them would be so called "Mini Expansions". All other games contain more rules and new content. Most of the expansions cover new maps, like the Europe map in the "Ticket To Ride Europe" stand-alone edition. This stand-alone edition adds a new map, new destination tickets and train stations as new game elements. This edition also adds new rules. These cover ferries, tunnels and train stations. It also adds a new kind of destination ticket, which a player can only get in the beginning. These destination tickets are the "longest destination ticket" and are worth the most points. One of the new rules cover train stations, which allow players to use one track that is owned by another player. This is a fourth option a player can take on his turn. Each train station is worth 4 victory points at the end of the game, if it is still not placed. Some of the games also have different or more bonus objectives. One of them is to have the most fulfilled destination tickets. This thesis only covers the original "USA Map".

2.4 Complexity

This section discusses the state-space and the game-tree complexity (Allis, 1994) for the two-player variant of Ticket to Ride.

2.4.1 State-Space Complexity

The state-space complexity is the number of legal board configurations, which can be reached from the initial position of the game. The calculation for Ticket to Ride depends on the tracks, which are built and the destination tickets on the hands of the players. The USA map has 78 different tracks. The distribution of tracks for a certain number of trains can be seen in Table 2.2. At the end of a game not all tracks are necessarily built. Since a wide variety of track combinations is possible, an upper bound for track configurations can be calculated by 3^{78}. Here 3 represents the different possibilities for a track. A track can be built by the first player, the second player or be unoccupied. The total number of tracks is 78.

One player can also own up to 28 destination tickets while the other one only owns 2. Any other combination is also possible as well. The total number of combinations can be calculated as following:

$$\sum_{n=2}^{28} \binom{30}{n} \times \sum_{l=2}^{30-n} \binom{30-n}{l}$$

This leads to $\sim 5.76 \times 10^{17}$ possibilities for destination tickets. This number has to multiplied by the number of possible tracks configurations. The number of possible track configurations is $3^{78} \approx 1.64 \times 10^{37}$.

Number of Trains	Number of Tracks
1	5
2	25
3	17
4	14
5	8
6	9

Table 2.2: Number of tracks for a certain number of trains.

This leads to the following total number of board configurations: $\sim 9.46 \times 10^{54}$. This is an overestimation, because it is not possible to build all tracks in the actual game. To compare the state-space complexity, chess was chosen. Shannon calculated this number to be $\sim 10^{43}$ (Shannon, 1950). Therefore the state-space complexity of TTR is more than $\sim 10^{11}$ higher than chess.

2.4.2 Game-Tree Complexity

The game-tree complexity of a game is the number of leaf nodes in the solution search tree of the initial position of the game. To calculate an exact number is often not possible. An estimation can be made in normal cases. To estimate the average number of turns is needed. The average number of possibilities for each player on each turn, the average branching factor, is also required. There are three different possibilities a player can do each turn.

- Draw or choosing new train cards
- Claim a road
- Draw new destination tickets

When the player chooses to draw or select new train cards, he can choose between five open cards or the topmost card of the stack. As a second card it is possible that a player cannot choose another card, because he took a locomotive as his first card. He also cannot choose a locomotive as his second card. For choosing cards a player has at most 36 options, if no locomotives are visible or become visible as a player chooses new cards. The least number of options a player can have for drawing cards is 18 while 2 locomotives are visible and no more locomotives become visible while choosing cards.

The number of tracks a player can claim is highly dependent on the train cards a player has on his hand. For the number of different possibilities a player can choose from, it is also important, which tracks were claimed, because it decreases the number of tracks a player can possibly build. A total number of 78 different tracks exists. The possibility that a player can build all tracks at the same time is unlikely.

Finally, a player has the possibility to gain new destination tickets. When he draws destination tickets, he has 7 different options to choose from. He can either choose 1 destination ticket (3 possibilities), two destination tickets (3 possibilities) or all three destination tickets (1 possibility).

The options a player has each turn, can be seen in Figure 2.3. The formula to calculate the game-tree complexity is in general $GTC = c^l \times b^l$. c is the average number of chance nodes, b is the average branching factor and l is the average length of the game. In most games the decision nodes and chance nodes are separated from each other, but in Ticket To Ride decision nodes and chance nodes are mixed. This changes the formula to $GTC = (d+c)^l$. d is $\sum_{n=1}^{5} d_i$, c is the sum of c_1 and c_2 and l is still the length of the game. These values were gained by self-play experiments, so these values are a statistical estimation of the values. If a player chooses to draw destination tickets, the number of possibilities is $d_3 = 7$. The player can select cards openly, by either choosing two colored cards or choosing one locomotive. The player has on average $\sim$11.34 options to choose two cards(d_1) or ~ 0.43 options to take a locomotive d_2. The combination of c_1 and d_4 represents the option, where the player chooses a card and draws a card from the stack. c_1, choosing a card and afterwards drawing a card, is on average ~ 66.01. d_4, drawing a card first and afterwards choosing a card, is about ~ 71.19. Since c_1 and d_4 contain a lot of similar results and d_4 is not commonly used in actual game play, it is not included completely in the calculation. Since d_4 and c_1 contain many similar results the difference is used. The difference is 5.18. d_5 is the average number of tracks a player can build on his turn. d_5 is $\sim$99.23. This number counts each color a track

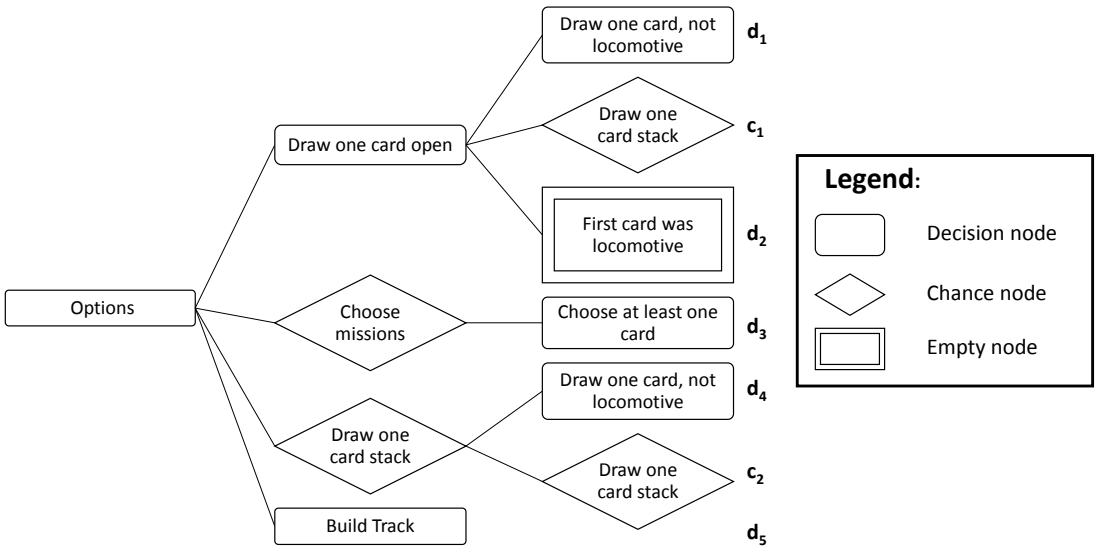

Figure 2.3: Options for a player

Var.	Action Types	Possibilities
d1	Choosing two cards	11.34
d2	Choosing a locomotive	0.43
d4	Drawing one card and then choosing one card	71.19
d3	Choosing destination tickets	7
d5	Building a track	99.23
c1	Choosing one card and then drawing one card	66.01
c2	Drawing two cards	124.61

Table 2.3: Possibilities for different action types

can be build from separately. c_2 represents the possibilities a player has, if he draws two hidden cards. This number is on average ~ 124.61. Finally, the length of the game is not fixed, because it ends after a certain condition is met. The average number of turns is ~ 91.70. An overview over the other values can also be seen in Table 2.3. This leads to $c1 + c2 + \sum_{n=1}^{5} d_i \approx 313.79$, so the Game-Tree Complexity is $GTC = 313.79^{91.70} \approx 8.75 \times 10^{228}$. This value can also be compared to chess. Shannon calculated this number to be 10^{120}. The game-tree complexity of TTR is therefore 10^{118} times higher than the game-tree complexity of chess.

Chapter 3

Monte-Carlo Tree Search

This chapter explains Monte-Carlo Tree Search (MCTS). At first the basis of MCTS, flat Monte-Carlo, is explained. Then the MCTS algorithm itself is explained. Afterwards the Upper Confidence Bounds for Trees, also called UCT, is shown and the enhancements used for this paper are explained.

3.1 Flat Monte-Carlo

Flat Monte-Carlo is an algorithm which samples the possible actions of a given state (Brügmann, 1993). The algorithm does not need an evaluation function or domain knowledge to work. The algorithm starts with determining the possible actions from the current state of the situation. The algorithm has a time limit or a certain number of simulations as a restriction for simulating but other restrictions can be possible as well. The algorithm then simulates a game for each possible movement. It first does the chosen action and then does random movements until the game is finished. This is repeated for all other actions. Each action gets either an equal number of simulations or equal time to do so. After each simulation the action adds one to the number of simulations and one to the number of wins, if the simulation ended in a win for the current player. In the end the action with the highest win rate is chosen as the best possible action.

3.2 Monte-Carlo Tree Search

MCTS is an expansion of Flat Monte-Carlo (Coulom, 2007a and Kocsis and Szepesvári, 2006). MCTS uses a tree to save the actions and their results. As first step a root node is created. Afterwards four different parts of the algorithm are repeated until a given time has passed or a certain number of iterations is reached. The four parts are selection, expansion, simulation and backpropagation. These four parts can also be seen in Figure 3.1.

Selection. Starting from the root node, the algorithm selects one of its children by using the chosen selection policy. This is repeated until a leaf node is reached.

Expansion. The node, which was chosen in selection, gets a number of children equal to the number of possible actions by the chosen node. Each of these children represents another action from the possible actions.

Playout. The game uses a policy until the game is finished. This policy can be to choose random actions, but more informed policies are possible as well.

Backpropagation. The chosen node and its parents get updated with the results of the simulation. The visit counter of each node is increased. If the result is a win for the player belonging to a current

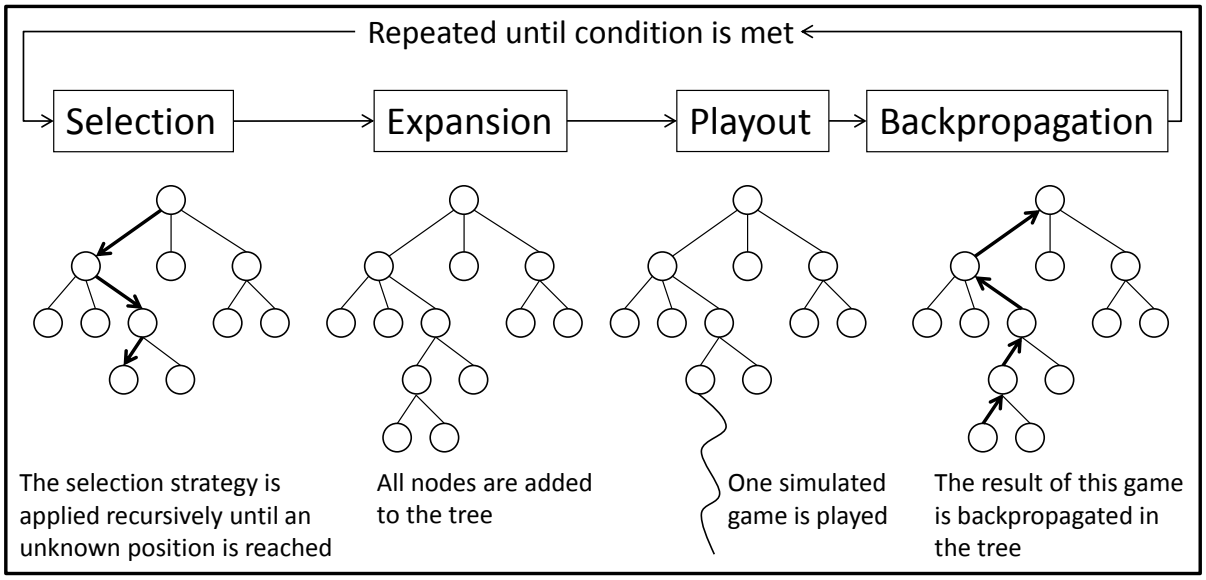

Figure 3.1: MCTS

node the number of wins is increased as well.

The action which is chosen at the end can be determined by different algorithms. One of them chooses the node with the most visits.

3.3 Upper Confidence Bounds for Trees

Upper Confidence Bounds for Trees or UCT is a selection policy for MCTS (Kocsis and Szepesvári, 2006). This policy uses the following function:

$$\max_{i \in A_p} \frac{s_i}{n_i} + C\sqrt{\frac{\ln n_p}{n_i}}$$

s_i is the total score and n_i is the number of visits of node i. i is a child of p. C is a constant. This constant is tweaked by experiments. A_p contains all children of p. To select a node the UCT value for each node is calculated. The node with the highest UCT value is selected as the most promising node.

3.4 Enhancements

MCTS can be enhanced in many ways. This thesis covers the enhancements used for the implementation. These enhancements include progressive unpruning, handling imperfect information and early terminations.

3.4.1 Progressive Unpruning

Progressive unpruning is used to reduce the branching factor (Chaslot *et al.*, 2008 and Coulom, 2007b). Overtime more and more nodes become available as well. Progressive unpruning uses domain knowledge to assign values to nodes. How many nodes are available gets calculated with the following formula.

$$T = A \times B^{k-k_{init}}$$

The value k states the number of nodes which are available. The value k starts with the value of k_{init}, which is 5. A is 50 and B is 1.3 as the paper of Chaslot *et al.* (2008) states. k gets increased until T is smaller than the number of simulations.

3.4.2 Progressive Bias

Progressive bias is another approach, which is based on heuristic knowledge (Chaslot *et al.*, 2008). Similarly to progressive unpruning, this approach also tries to guide the selection step into earlier selecting nodes which have a high heuristic value. The enhancement of the selection step is made through an addition to the UCT formula.

$$\max_{i \in A_p} \frac{s_i}{n_i} + C\sqrt{\frac{\ln n_p}{n_i}} + \frac{H_i}{n_i}$$

H_i represents the heuristic knowledge from the action in node i. This value is divided by the number of times the node is visited, so the influence of the heuristic knowledge gets smaller over time. Therefore nodes of higher importance get evaluated more closely earlier in the game.

3.4.3 Number-of-Visits-Dependent Strategy

In the two earlier sections progressive unpruning and progressive bias were discussed. Both strategies can be enhanced by starting at a later point in the simulation. This point is given by a threshold T. The number of visits of the selected node has to be at least equal to this threshold to apply the chosen selection strategy to the node. As long as the threshold is not reached, the selection step uses another given selection strategy. For example this selection strategy could be UCT. The threshold T has to be found empirically.

3.4.4 Determinization

Determinization is an approach to handle imperfect information (Frank and Basin, 1998). Determinization converts a game of imperfect information into a game with perfect information. It uses the structure of the game and the history of the actual game, to fill in the hidden information. In each iteration, one of the determinizations is randomly chosen as the current state, which is used for all four parts of the MCTS algorithm.

Determinization has two disadvantages which were explained by Frank and Basin (1998). The first disadvantage is called strategy fusion. For each determinization a best action exists. The problem of strategy fusion is to find one action which is best for the information set which combines all possible determinizations. The second disadvantage non-locality describes the possibility that a determinization may be unlikely in an actual game. For example, if the opponent has built certain tracks, which would fulfill a certain destination ticket, it is likely that the opponent has this destination ticket. In determinization it could happen that a player would get another destination ticket, which represents an unlikely state. Even if determinization has several disadvantages, it works for several games like Skat (Buro *et al.*, 2009), Bridge (Ginsberg, 1999) and Phantom Go (Cazenave, 2006).
To explain determinization, Ticket to Ride can be used for an example. The train cards from the beginning and the train cards which are drawn from the stack by the opponents are unknown to the player. The number of train cards for each color is known to all players. It is also known which cards are in the discard pile and which open cards are chosen by the opponents. With this information a stack of cards can be created. Each opponent then gets a number equal to his hidden cards randomly from this stack. The remaining cards then build the new stack for the current simulation.

3.4.5 Cheating MCTS

Cheating MCTS is used for games with imperfect information. This strategy does not try to handle imperfect information, but uses all information even if this is normally not possible for the player. Cheating MCTS cannot only be used to handle imperfect information, but it also can be used to gain information about the upcoming chance events. This could be done by fixing upcoming chance event.

3.4.6 Single-Observer MCTS

The Single-Observer enhancement is an enhancement for games with imperfect information (Cowling *et al.*, 2012). For each simulation a random determinization is chosen. This enhancement changes the selection step. In the selection step the algorithm disables all moves, which are not possible. It also adds new moves, if they become available through the new determinization. This can also be seen Figure 3.2.

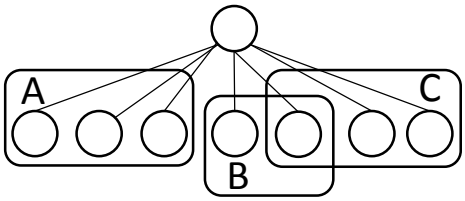

Figure 3.2: Possible moves for different determinizations

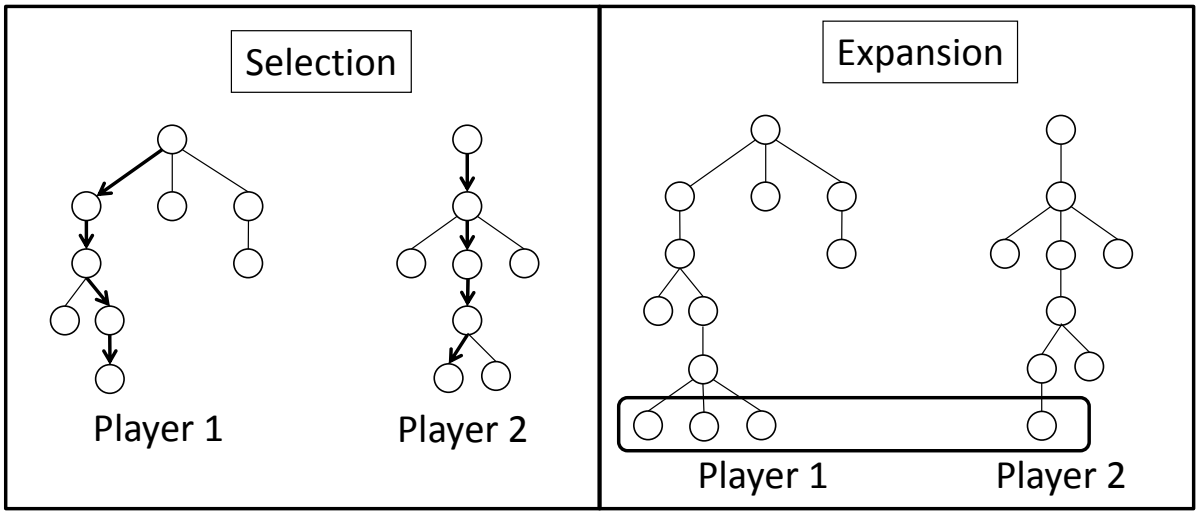

Figure 3.3: MCTS algorithm

The figure shows the possible moves for the three different determinizations A, B and C. As can be seen in the picture some moves are possible for more than one determinization. Single-Observer MCTS has a disadvantage, because of the high number of different determinizations a lot of nodes have to be added to the tree near the root. The high branching factor can lead to a smaller depth of the tree (Cowling *et al.*, 2012).

3.4.7 Multiple-Observer MCTS

Multiple-Observer MCTS was introduced by Cowling *et al.* (2012). In this approach each player has a separate tree. Each tree contains the information from the corresponding player's point of view. The actions of the other players are represented by one node. The algorithm expands the tree for each player simultaneously. The changes of Multiple-Observer MCTS can be seen in Figure 3.3. In each simulation a random determinization is chosen. From this determinization it gets determined which moves are possible. This type of enhancement changes the selection step, the expansion step and the backpropagation step. The Multi-Observer works similar to the Single-Observer, but instead of selecting a move in one tree, it is done in all trees. The same depth of the tree represents the same step. In the trees of the other players, the turn of the player is represented by only one node.

In the selection step the current player chooses a node. In the other trees the placeholder node is selected. The current player then changes to the next player and the same steps are repeated until a leaf node is reached.

In the expansion step the current player expands the selected node as usual. Each other tree is expanded by the placeholder node.

In the backpropagation step each tree gets updated as usual. Multiple-Observer MCTS fixes the disadvantage of Single-Observer MCTS. All nodes of the opponents are only represented by one node, so the total branching factor is smaller than in Single-Observer MCTS and less nodes are available in total.

3.4.8 Determinized UCT

Determinized UCT is another approach to fix the disadvantage of Single-Observer MCTS (Cowling *et al.*, 2012). Determinized UCT uses a fixed number of determinizations. Each of these determinizations is randomly chosen out of all possible determinizations at the beginning. Every determinization has its own tree, so that the branching factor does not change throughout the simulations. Instead of choosing a new determinization after each iteration, the next fixed determinization is chosen. The tree is also changed correspondingly. The rest of the algorithm works like the normal MCTS algorithm except for the choosing step. In the choosing step all children from all roots are combined. If two nodes correspond to the same move, the nodes add up the visits and the score of the node. The node with the most visits is chosen as the action, which is played. Other choosing methods are possible as well. One other possibility is that each tree gets one vote. The action which has the most votes is then chosen. If some moves are tied for most votes, the action with the most number of visits is chosen (Nijssen and Winands, 2012). Another possibility was explored by Fern and Lewis (2011). The algorithm, called Ensemble UCT, chooses the node with the highest average score over all trees. Therefore the following formula has to be maximized.

$$Q_{RP}(s,a) = \frac{\sum_i Q^{(i)}(s,a) \times n^{(i)}(s,a)}{\sum_i n^{(i)}(s,a)}$$

$Q^{(i)}(s,a)$ represents the score of action a in state s in tree i. $n^{(i)}(s,a)$ represents the number of visits of the same node.

3.4.9 Early Terminations

The early termination takes place in the simulation step. In the normal simulation step the game is simulated until the end. For early terminations the game is either played for a certain number of turns or until a certain condition is met. An example for the usage of an evaluation function to terminate the simulation, is given by Winands and Björnsson (2010). For the game of Amazons it has been shown that it enhances the algorithm if the simulation is cut off after a certain number of turns, the game state is then evaluated with an evaluation function to give a result for the current playout (Lorentz, 2008).

3.4.10 ϵ-based strategies

The ϵ-greedy strategy enhances the selection of nodes in the simulation step. ϵ has a value higher than 0 and smaller than 1 ($\epsilon \in]0,1[$). The strategy follows the strategy (1-ϵ) of the times, but does a random move in all other times. The random action could be a move, which follows the strategy, but it can be any other move as well. This ϵ has to be tweaked empirically.

ϵ-greedy strategy
The ϵ-greedy strategy analyses each action after certain criteria. After each action is analyzed the best action is performed.

ϵ-rule based strategy The ϵ-rule based strategy is similar to the ϵ-greedy strategy. But rather than choosing the best node, it chooses an action based on expert domain knowledge.

Chapter 4

MCTS in Ticket to Ride

This chapter explains integration of MCTS in Ticket to Ride.

4.1 Nodes

A node has to know all its children as well as the children which are possible at the moment. For each information set different determinizations exist. Due to this it can happen that a player can build a track in one determinization, but not in another one. A node also has to save the number of visits and wins.

4.1.1 Actions

A node in Ticket to Ride has to represent three different actions a player can take, but it has only to represent one of these actions at a time.

- Choose and/or draw cards
- Choose destination tickets
- Build one track

Each of these actions has to be represented in a special way. The action of choosing or drawing a card is represented by a number. This value represents the card's position within the visible train cards. If the value is equal to the size of the array, a card is drawn. Choosing a destination ticket can also be represented by a number. For this the node can save up to three values between 0 and 2. If a destination ticket is chosen, the player gets three destination tickets to choose from. The saved values represent the position of the destination tickets in this list. The last action is to build a track. This action can also be represented by a number. Each track has a certain ID by which it can be referenced. If a node represents this type of action, it is necessary to know which colors can be used by the player to build this track. All actions need a list to store information, so each different type of action can reuse that list. The action also stores information about its type and the player who performs it.

4.2 Choosing Destination Tickets

This section describes the process of choosing destination tickets at the beginning and during the game. The tickets at the start of the game are evaluated and chosen directly. The tickets, which are chosen during the game, are checked, if they are fulfilled and then evaluated with MCTS.

4.2.1 Start of the game

At the start of the game each player is handed three destination tickets. From these tickets the player has to choose at least two. If one ticket is not chosen, it gets returned to the bottom of the destination

tickets stack. These tickets have to be chosen right away. The selection of at least two tickets can be separated in several steps. The engine always chooses only two tickets. First the destination tickets the player got, are evaluated by the number of points a player scores, if they fulfill them. The destination ticket with the highest score is chosen as the first destination ticket. The tracks on the shortest path of this destination ticket are virtually built for the further analysis of the two remaining destination tickets.

The shortest path for the remaining two destination tickets are calculated separately. These paths are then compared to the path of the already chosen destination ticket. Each track, which exists in both lists, is removed. The number of removed tracks is counted for each destination ticket separately. After each track is checked, the destination ticket, which shares the most tracks with the first destination ticket, is chosen. If both destination tickets have removed the same number of tracks, the destination ticket with fewer tracks to build is chosen. If two paths share some or most of their track, a longer continuous path can be built. This helps later in the game to integrate more destination tickets in the network of trains a player has.

If no distinct answer is found, the destination ticket with the higher value is chosen. In the case that both destinations have the same value, the last destination ticket is chosen. This ticket can still be considered random, because the destination tickets a player get are random and as such are in a random order.

4.2.2 During the game

As soon as all destination tickets are fulfilled, a player should draw new destination tickets, especially if the end of the game is not imminent. The possible combinations of destination tickets, if he takes new destination tickets first, is at least 2925. The general formula to calculate this number is

$$total = \frac{(30 - nr)!}{(27 - nr)! \times 3!}$$

nr represents the number of destination tickets, which are known by the player. The value 2925 represents a two-player game, where each player has chosen his starting hand. The calculation of this number also considers to have no knowledge about the destination tickets of the opponent other than the number. nr is in this case equal 6.

If the opponent's destination tickets are known, $nr = 3$, the number of combinations is still 2024. Drawing destination tickets is unreliable, if a player tries to predict which destination tickets he would get. If a player has all his destination tickets fulfilled, new destination tickets are drawn immediately, if it is not his last round of the game.

The player draws three new destination tickets as his next action. From these tickets he has to take at least one, but he can keep any number. Unfulfilled tickets still give negative points at the end of the game.

All destination tickets are checked, if they are fulfilled, because a player built tracks, which belong to this destination ticket. The unfulfilled drawn tickets generate the new actions by creating a separate action for each possible combination. Up to seven different actions are possible when no destination ticket was already fulfilled, otherwise fewer combinations are possible. To these actions the fulfilled destination tickets are added, such that the destination tickets are added to the player's hand when the action is selected. If at least one of the drawn destination ticket is fulfilled, an action which includes only fulfilled destination tickets, is created as well. These actions are then evaluated with MCTS.

4.3 Hidden Information

In Ticket to Ride it is unknown which cards were secretly drawn by the opponents. It is also unknown which destination tickets the opponents have chosen. These are two different types of hidden information and therefore have to be handled differently. The process of filling in this hidden information is also called determinization.

4.3.1 Train Cards

The stack and train cards drawn from it by the opponent are unknown to the player. These train cards can be estimated. This process is called determinization.

To find out these train cards, the player can use the deck of train cards. The deck represents all train cards, which are on the hands of all players, open-lying, the stack and the discard pile. All cards, which were chosen openly, the cards to choose from, and are on the discard pile are removed from the deck. For a determinization for a certain player, his unknown cards have to be assigned. The remaining cards have to be in the stack or on the opponents' hands. This deck is called hidden cards for the rest of this thesis. It is possible to narrow down the cards of the opponent even further. As soon as the stack is empty, all cards, which are on the hands of all opponents, are known to the player. All cards are now either in the discard pile, open-lying or on the hands of a player. In a game with two people a player knows for certain which cards are owned by the opponent. In a game with more than two players, a player only knows the cards that have to be on the hands of all opponents.
As an example a two-player game of TTR is considered, which both players have drawn cards directly from the stack, so these cards are only known by the players. Player 1 has drawn two red and a yellow card and Player 2 has drawn a green and also a yellow card. As soon as the stack is empty, all cards of the opponent are known by the player. Player 1 would need to look in the discard pile and at the open-lying cards, eliminate these cards from the possible deck of cards, and then do the same with his own cards and the cards which were chosen by the opponent openly. The remaining cards have to be the cards which were secretly drawn by the opponent. This procedure is easily adaptable for more than two players, but then a player has to assign these cards randomly to the hands of his opponents.
In the determinization each opponent gets assigned cards equal to his number of cards which he has drawn from the stack. If some of these cards are known, because the stack was empty at one point, this knowledge is used first. The remainder of cards is randomly assigned from the hidden cards. The remaining hidden cards are then shuffled and become the new stack. To use the example from above, Player 1 knew that Player 2 has a green and a yellow card on his hand. In the next round he drew another card from the stack. In the determinization a green and a yellow card is assigned to the hand of Player 2, because he has not played these cards yet. He also gets a random card from the hidden cards, because Player 1 has no information about this card. The remaining hidden cards become the new stack.

4.3.2 Destination Tickets

Each player also has destination tickets. These destination tickets are known to the player, but unknown to his opponent. The number of destination tickets is also known. Each destination ticket is unique. A game has always the same destination tickets, so the destination tickets which exist are known as well. The destination tickets of the player are subtracted from all possible destination tickets. The remaining destination tickets are possible for the opponents. Three different strategies have been implemented to determine which destination tickets an opponent could have. The first strategy randomly assigns the destination tickets. The last two strategies use the tracks the opponent has built. The destination tickets which would be fulfilled get selected. Both strategies use these destination tickets as the destination tickets the opponent has. If these destination tickets are more than the opponent should have, the second strategy takes the destination tickets with the highest value. The third strategy chooses these destination tickets randomly. If the opponent has too few destination tickets, the second strategy takes the destination tickets with the lowest value from the remaining destination tickets. The third strategy chooses these destination tickets randomly from the remaining ones.

4.4 Chance

Ticket to Ride has two different chance events. One chance event happens when a player draws new destination tickets. The other type of chance event happens, if the player chooses or draws a train card. Both chance events are handled similarly. At the beginning of each iteration a new game situation is created. The cards which are determined to be in the stack and the remaining destination tickets are shuffled separately. If the stack should be empty, the discard pile gets shuffled and becomes the new stack. This is the reason why this implementation has no chance nodes and therefore does not calculate each chance as it happens.

In the special case of the cheating MCTS, discussed in Subsection 3.4.5, chance events are fixed. The chance events are fixed by not shuffling the stack of train cards or mission tickets, if a new board for a simulation is created. Therefore the chance events are fixed until the stack of train cards is empty. The

discard pile is then shuffled randomly to create a new stack. From this point the chance events are no longer known to the player regarding the discard pile. The destination ticket stack gets never shuffled, so this information is always known to the cheating MCTS.

4.5 Upper Confidence Bounds for Trees

Upper Confidence Bounds for Trees (UCT) as described in Section 3.3 needs some adjustment before it can be implemented. Two different problems have to be further examined. The first problem happens if a node was never visited yet. The second problem occurs if more than one child has the same UCT value.

To handle the first problem, the method to calculate the UCT value returns 1.

For the second problem, a random value is added to the UCT value. Therefore a random value between 0 and 1 is divided by 10k. This random value does not have an influence on the actual UCT value, but allows to differentiate between the different nodes.

4.6 Selection Strategy

This section describes the different strategies, which were implemented to improve the selection step. The used heuristic knowledge is explained first, because the selection strategies are based on it.

4.6.1 Heuristic Knowledge

Progressive bias and progressive unpruning depend on heuristic knowledge. Therefore each action gets a certain value. These values need some game knowledge which has to be calculated. Two different types of information are important here. The first information contains all tracks, which still have to be built, to fulfill the destination tickets of a player. The second information stores the train colors, which are needed for the tracks, to fulfill the destination tickets of a player.

An overview of these values can be seen in Table 4.1. The nodes are first ordered by the type of

Action	**Condition**	**Value**
Track	Destination ticket contributing	$x \times 10$
Track	Not contributing	-10
Card	Track contributing	4
Card	Locomotive	1
Card	Drawn and contributing	4
Card	Not contributing	-4
Card	Maybe not contributing	-4

Table 4.1: Heuristic values for different actions

action. All three actions are theoretically possible, but they are not all possible together. Therefore a player can choose either to build tracks and choose and/or draw train cards; or choose destination tickets. The destination tickets are not closer examined and are all evaluated this the same value.

Track

The number of times a track is needed to fulfill destination tickets is used to calculate the heuristic value for track building actions. If a track is needed at least once, it gets a value of $x \times 10$, where x is the number of times, the track is needed. If this number is 0, the action gets a value of -10.

Card

If an action involves drawing cards, each card that is drawn is evaluated separately. If the action only chooses to take a locomotive, the action gets a value of 1. If the action takes a card, which is needed to build a track, 4 is added to the value of the action. If the action draws a card from the stack, it is important which cards could be in the stack. If only cards, which would contribute to a destination ticket, are possible the action's value gets a bonus of 4. If the action draws or chooses a card, which cannot contribute to a destination ticket, the value of the action gets a penalty of 4. The same penalty also gets applied, if the possibility exists that a drawn card does not contribute to a destination ticket.

Drawing two cards which help to build such tracks gets a value of 8. Choosing or drawing only one card that will contribute to the possible tracks will give an action a value of 0. If only a locomotive is chosen, the action gets a value of 1. If an action chooses two cards, which might not or does not contribute to destination ticket tracks, the action gets a value of -8.

4.6.2 Progressive Bias And Unpruning

Progressive bias and progressive unpruning use the heuristic knowledge to select nodes during the selection step. The values are calculated once for each node. Progressive bias uses heuristic knowledge during the UCT calculation, while progressive unpruning uses the value to prune children.

A threshold for the number of visits can be given. This threshold has to be reached in a parent node, before progressive unpruning or progressive bias is used on its children. Otherwise it uses another selection strategy such as UCT. This strategy is given at the initialization.

4.7 Playout Policies

The further discussed policies were implemented to improve the simulation step. This section discusses the random policy, the early termination policy and the strategy policy.

4.7.1 Random

The most common playout policy is the random playout. All actions a player can take are determined and one of them is randomly chosen. Due to the fact that the outcome of acquiring new destination tickets is highly unpredictable, each action which chooses new destination tickets is not added as a possible move and only other actions can be chosen.

4.7.2 Early Termination

As discussed in Subsection 3.4.9 the playout can be terminated earlier than the end of the game. If the early termination is used, the playout ends after 45 moves which is about half of a normal game. The game is then rated as a regular game. This means it is either rated as a win or a loss for the corresponding player.

4.7.3 ϵ-Rule Based Strategy

Instead of doing random plays during the simulation step, these steps can follow certain rules. This is called a playout strategy. In this master thesis an ϵ-rule based strategy is used. The playout strategy follows the strategy $(1 - \epsilon)$ of the times and does a random play ϵ of the times.

Each round, if the playout strategy is used, the tracks, which are needed to fulfill the current destination tickets are calculated and saved for all players. Each player has his own list with his own tracks. Each round the playout strategy looks for the track, which needs the highest number of train cards and belongs to the list of the current player. If a buildable track is found, which needs six cards, the selection ends early and the action to build the track is performed. If no track is found, the playout compares the buildable tracks to the tracks of the other player. The strategy then searches for the smallest track for which the player has the necessary cards. If a track is found, which needs only one card, the track is built immediately. After all tracks were compared and no track was found that fits the criteria, cards are drawn or chosen. All possibilities, which either choose or draw up to two cards, are calculated and one of them is randomly performed. If no cards were drawn or chosen, one of the tracks which is buildable, is randomly executed. If the playout needs to select a random action, all possible actions except choosing new destination tickets are found. One of these actions is then randomly selected and afterwards performed.

Fulfilling destination tickets has the advantage that the simulation is more accurate, because this would happen in the actual game as well. It is also preferred to build tracks for the player itself over tracks, which the opponent needs. The reason for building tracks, which are needed by opponents, is that it can

be helpful to block opponents to fulfill their destination tickets. However, a player only wants to invest as little as possible in such a strategy. Building tracks also ends the simulation faster, and so a higher number of simulations is possible.

This type of playout policy has another option for an early termination. The termination should be performed, if the list of tracks to build is empty. This is a point during the game where new destination tickets are chosen. It is a good point to end the playout, because choosing destination tickets in a simulation is unpredictable (see Subsection 4.2.2). Therefore no meaningful results can be obtained from choosing new destination tickets during the playout strategy and the current game situation can still be evaluated.

4.8 Finding the Shortest Path

During the game the calculation of the shortest path is used for different strategies like progressive bias, progressive unpruning or the playout strategy. The strategies depend on the knowledge to know, which tracks have to be built to accomplish destination tickets. Dijkstra and A^*, briefly discussed in Section 1.1.2, were used to find the shortest path. The infrastructure of railroads or map of tracks is stored as an undirected graph. In this graph the nodes represent the cities and the edges represent the tracks. The cost of an edge is equal to the number of trains a player needs to build that track. For further analysis it is also important to know, if a track was already built and if so, which player build the track.

Before a game starts, the graph is evaluated and the lowest connection cost from one city to each other city is calculated. After this is done for all cities, this information is stored for later analysis. A^* uses the stored information to plot the shortest path from one city to another. Therefore Dijkstra was used as an underestimation for A^*. Instead of an heuristic function the A^* algorithm uses the evaluation of the Dijkstra algorithm as its heuristic knowledge. The A^* algorithm considers, if a track was built by the player the information is considered to find a shorter path. The algorithm also takes into account, if a track was built by an opponent. A^* then tries to find an alternative shortest path.

The algorithm does not consider, if the player actual has enough resources to build all tracks. This means, that if a player should not own enough trains, which he needs to claim a track, a shortest path would be returned.

Chapter 5

Experiments and Results

This chapter shows the different AI players, which were implemented. It also explains the setup of the experiments. Then the different experiments are explained and the results are analyzed.

5.1 AI Players

In general, five different types of AI Players were implemented. The five players are Cheating Player, Single Observer, Multiple-Observer MCTS, determinized UCT and voting. All of these players can be enhanced by the different enhancements discussed in Chapter 4.

A special player, which was implemented, is a combination of progressive unpruning and progressive bias. The agent works in the following way. First the nodes are pruned with progressive unpruning. Afterwards the node is selected with progressive bias. All information about progressive unpruning can be found in Subsection 3.4.1. Progressive bias has been explained in Subsection 3.4.2.

5.2 Experimental Setup

In most cases, each experiment was performed at least 200 times. If an experiment included more than 200 games, this information is included in the experiment. Each player started in an equal number of games, so if 200 games were played, each player started 100 times. If it is not otherwise stated, the MCTS algorithm got 2 seconds of thinking time. The other thinking time, which has been used, is 4 seconds.

This section will also discuss the undefined parameters of different players. Confidence bounds are explained as well. Next, the environment in which the experiments were performed is shown.

5.2.1 Player Setup

The used values for the different constants are discussed below. All agents are simulating until the end of the game, if not otherwise stated.

The UCT algorithm, see Section 3.3, uses a C value of $\sqrt{2}$. The progressive unpruning (see Subsection 3.4.1) and progressive bias (see Subsection 3.4.2) approaches have a threshold of 0. This means that both strategies directly start their approach after a node is selected the first time. The constants, which are used to determine how many nodes are not pruned in the progressive unpruning selection step, were set as discussed in Subsection 3.4.1. Therefore A was set to 25 and B to 1.3. The value for k_{init} is 5. The heuristic value of progressive bias is not modified. The voting and determinized UCT approaches shuffle the card stack each time when a new board for a simulation is created. This way the cards in the hands of the other player are always the same, but the train card stack and the mission stack are

shuffled. The playout strategy uses the early termination strategy discussed in Subsection 4.7.3 and does not necessarily play until the game is finished. The ϵ value of the ϵ-rule based strategy is 0.05.

5.2.2 Confidence Bounds

To give the confidence bound of an experiment the following formulas are used

$$s(w) = \frac{\sqrt{w \times (1-w)}}{\sqrt{n}}$$

and

$$b(w) = z_{\%} \times s(w)$$

$s(w)$ is the standard error. w is the winning rate from any of the both players. For this thesis a confidence bound of 95% is chosen, so $z_{\%}$ is 1.96.

5.2.3 Environmental Setup

All experiments were performed on a CL211LOA2-B transtec CALLEO 211L AMD Opteron Server. The server has the following specifications:

- 2 x AMD Dual-Core Opteron F 2216, 2.4 GHz, 95 Watt (max. 2 Opteron Socket F Processors)
- 8 GB DDR2 DIMM, reg. ECC (4 DIMMs, max. 32 GB, 16 DIMMs)
- NVIDIA nForce4 2200 Professional chipset
- 2x PCI-E x8 slots via riser card (full height, half length)
- 80 GB hot-swap SATA hard drive, max. 2 hot-swap hard drives
- DVD-ROM
- onboard dual Broadcom BCM5721 Gigabit Ethernet
- onboard XGI Z9s VGA 32 MB DDR2 graphics
- 1 U rackmount chassis incl. mounting rails
- 500 Watt power supply

5.3 Single Observer

For this experiment, different types of MCTS agents were tested against the Single Observer (SO). The results of these tests can be seen in Table 5.1. A total of four experiments were performed. Each experiment was performed with 2 and 4 seconds thinking time. The first experiment shows the performance of the SO versus Flat Monte-Carlo with usage of determinization (FSO).
For most games the FSO performs less than the SO. In this case the FSO performs better for both thinking times. Similar observations were obtained by Browne (2013), were the Flat Monte-Carlo Player also beats the MCTS Player. He showed that if the number of simulations is higher, the Flat Monte-Carlo Player starts to perform worse and the MCTS Player starts to perform better, so that at a certain point the MCTS player is better than Flat Monte-Carlo Player. A similar phenomenon may occur here as well, because the results appear to be better for the 4 seconds thinking time than the 2 seconds thinking time.

It was also tested, if narrowing down the number of determinizations has an influence on the performance of the SO. Therefore the SO was tested against the SO with the worst mission enhancement (WM) and the worst mission random enhancement (WMR). Both enhancements were explained in Subsection 3.4.4. As the results show, the SO-WM agent performs better than the SO with 2 seconds thinking time. The SO-WMR player has a slightly lower performance against the SO than the SO-WM. If both agents have a thinking time of 4 seconds, they both win about 50% of the played games. Therefore they enhance the

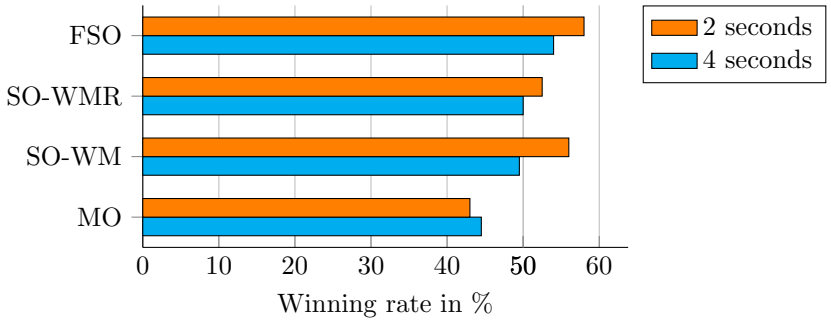

Figure 5.1: Visualization of SO results

agent only, if fewer determinizations are done and as such each simulation has a higher influence on the end result. The last agent tested is the Multiple-Observer (MO). The MO performs worse than the SO for 2 seconds thinking time. If the players used 4 seconds thinking time no significant difference could be shown.

A visualization of the results is shown in Figure 5.1. The figure shows the result of the player in percent. The player is the player stated in the description next to the bar. The figure compares the result of the same player for the two different thinking times.

Player	**2 sec**		**4 sec**	
	SO %	**SO Nr.**	**SO %**	**SO Nr.**
SO vs. FSO	42 ± 6.84	84	46 ± 4.88	184
SO vs. SO-WMR	47.5 ± 6.92	95	50 ± 6.93	100
SO vs. SO-WM	44 ± 6.88	88	50.5 ± 6.93	101
SO vs. MO	57 ± 6.86	114	55.5 ± 6.89	111

Table 5.1: Single Observer

5.4 Cheating Player

As another experiment, the Cheating Player (CP) was tested against the Flat Monte-Carlo Player, which knows the cards of the other player (FCP). It was also tested, if knowing the stack changes something. This means that the player knows beforehand, which cards will be drawn or which cards are the replacement cards for chosen cards. This player gets the additional description of U, which stands for unshuffled. Both results can be seen in Table 5.2. The CP-U player played 400 games to get significant results. The results were visualized in Figure 5.2. The visualization shows the results from the view of the Flat Monte-Carlo player.

In both cases the Flat Monte-Carlo Player performs better than the corresponding Cheating Player. The CP has a winning rate of 40.5%. The CP-U performs slightly better. CP-U has a winning rate of 45.5% against FCP-U.

Pl. 1	**Pl. 2**	**Pl. 1 %**	**Pl. 2 %**	**Pl. 1 Wins**	**Pl. 2 Wins**
FCP	CP	59.5± 6.8	40.5± 6.8	119	81
FCP-U	CP-U	56± 4.86	44± 4.86	224	176

Table 5.2: FCP vs. CP

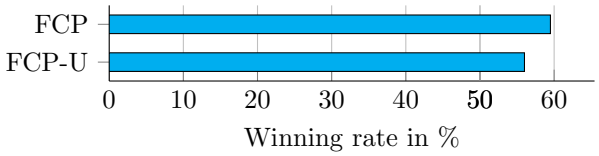

Figure 5.2: Visualization of FCP vs. CP results

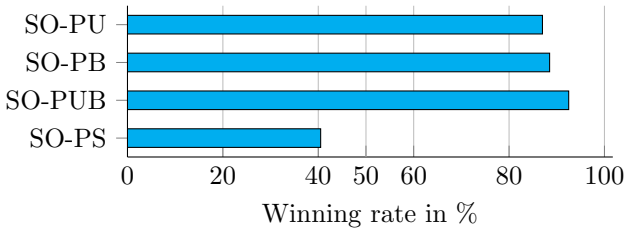

Figure 5.3: Visualization of FSO vs. SO with Enhancements results

5.5 Enhancements

In this section the four different enhancements are closer analyzed. They are compared to different agents and also compared with each other.

5.5.1 FSO vs. Enhancements

The enhancements, which are tested in this section, are progressive unpruning (PU, see 3.4.1), progressive bias (PB, see 3.4.2), the combination of progressive unpruning with progressive bias (PUB) and a ϵ-rule based playout strategy (PS, see 4.7.3). The heuristic values, which were used for the progressive enhancements, can be seen in Subsection 4.6.1. All enhancements were tested against the FSO first, to see if they enhance the playout of the SO in such a way that the SO is able to defeat the FSO on average. As can be seen in Table 5.3, most enhancements, except SO-PS, are able to defeat the Flat Monte-Carlo agent in more than 80% of the games. SO-PS loses over 70% of the time when playing against the Flat Monte-Carlo player. The visualization of the results, visible in Figure 5.3, shows that SO-PUB performs best against the FSO. SO-PB wins 88.5% of all games. SO-PU performs a little bit less than the other two progressive approaches with winning only 87% of all games.

Pl. 1	**Pl. 2**	**Pl. 1 %**	**Pl. 2 %**	**Pl. 1 Wins**	**Pl. 2 Wins**
FSO	SO-PU	13 ± 4.66	87 ± 4.66	26	174
FSO	SO-PB	11.5 ± 4.42	88.5 ± 4.42	23	177
FSO	SO-PUB	7.5 ± 3.65	92.5 ± 3.65	15	185
FSO	SO-PS	70.5 ± 6.32	40.5 ± 6.32	141	59

Table 5.3: FSO vs. SO with Enhancements

5.5.2 Improving Flat Monte-Carlo with Enhancements

To see, if the enhancements help to enhance the performance of the SO, the enhancements were tested against the Flat Monte-Carlo, which was also improved by the same enhancements. The results for these experiments can be seen in Table 5.4. The progressive approaches played 600 games in total, when they had 2 seconds thinking time, to get clearer results. The approaches with 2 seconds thinking time do not give a clear result. The enhanced version of FSO seems to performs less than the enhanced SO, but these results are not clear. The SO-PS performs similarly to the FSO-PS with winning 49.5% of all games. To get clearer results both approaches got more thinking time, so the thinking time was increased to 4 seconds.

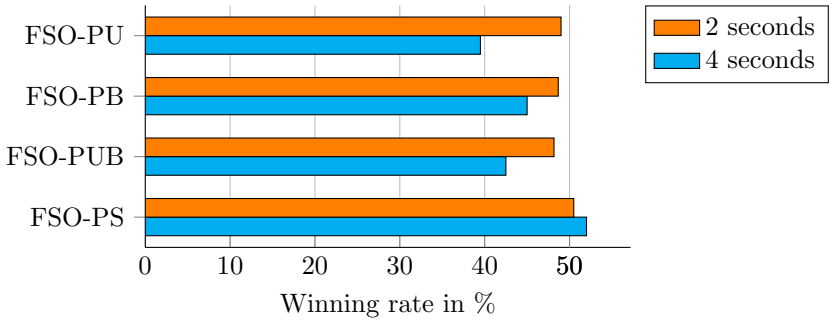

Figure 5.4: Visualization of FSO vs. SO both with enhancements

With 4 seconds thinking time the progressive unpruning approach performs best against its flat variant. The other progressive variants have a slightly smaller winning rate, but both win more than 50% of the time. PB wins 55% of all games and PUB wins 57.5% of all games. The performance of the playout strategy remains the same.
The visualization of the games can be seen in Figure 5.4. The results are presented in the same way as in Section 5.3.

Player	**2 sec**		**4 sec**	
	SO %	**SO Nr.**	**SO %**	**SO Nr.**
SO-PU vs. FSO-PU	51 ± 4	306	60.5 ± 6.78	121
SO-PB vs. FSO-PB	51.33 ± 4	308	55 ± 4.88	220
SO-PUB vs. FSO-PUB	51.83 ± 4	311	57.5 ± 6.85	115
SO-PS vs. FSO-PS	49.5 ± 6.32	99	48 ± 6.92	96

Table 5.4: SO vs. FSO both with Enhancements

5.5.3 SO vs. Enhancements

This section shows the results of the SO, which plays against the different enhancements. As can be seen in Table 5.5, all enhancements, except the PS, perform better than the SO without enhancements. It was expected that the default SO performs worse than the progressive approaches. This was expected, because the default FSO lost the majority of the games against the SO enhanced with progressive approaches. However, the default FSO was able to win against the default SO more than 50% of all games. While it was expected that the heuristic enhancement performs better than the SO, it was not anticipated that the difference would be that significant. The worst performing enhancement, PU, wins 88% of the games. PB wins 95% of the games and PUB even wins 99.5% of the 200 games.
PS performs significantly worse than the SO with winning only 32.5% of all games.

To get a clearer overview over the results, the results were visualized in Figure 5.5. The results show the results in percent for the enhanced players.

Pl. 1	**Pl. 2**	**Pl. 1 %**	**Pl. 2 %**	**Pl. 1 Wins**	**Pl. 2 Wins**
SO	SO-PU	12 ± 4.5	88 ± 4.5	24	176
SO	SO-PB	5 ± 3.02	95 ± 3.02	10	190
SO	SO-PUB	0.5 ± 0.98	99.5 ± 0.98	1	199
SO	SO-PS	67.5 ± 6.49	32.5 ± 6.49	135	65

Table 5.5: SO vs. SO with enhancements

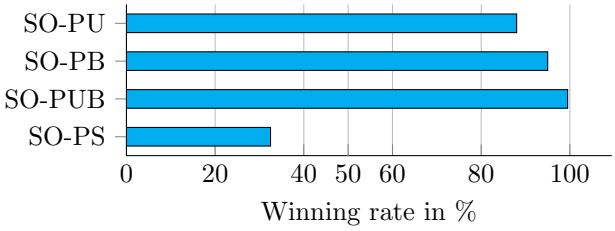

Figure 5.5: Visualization of SO against different enhancements

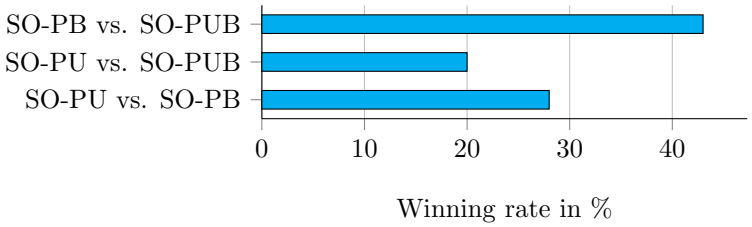

Figure 5.6: Visualization of different enhancements playing against each other

5.5.4 Combinations

In this section, the three working enhancements were tested against each other. The results of these experiments can be seen in Table 5.6. The results show that progressive unpruning loses against progressive bias over 70% of the games and against the combination of progressive unpruning and progressive bias about 80% of the games. Progressive bias and progressive unpruning performs slightly better (57% win ratio) than progressive bias. All in all PUB wins more than 50% of the games against PU as well as against PB. These results are in line with the results of Subsection 5.5.3.
The results of the different enhancements playing against each other also can be seen in Figure 5.6.

Pl. 1	**Pl. 2**	**Pl. 1 %**	**Pl. 2 %**	**Pl. 1 Wins**	**Pl. 2 Wins**
SO-PU	SO-PB	28 ± 6.22	72 ± 6.22	56	144
SO-PU	SO-PUB	20 ± 5.54	80 ± 5.54	40	160
SO-PB	SO-PUB	43 ± 6.86	57 ± 6.86	86	114

Table 5.6: Testing enhancement against each other in the SO agent

5.6 Multiple-Observer MCTS

Multiple-Observer MCTS was also tested with several enhancements and tested against the same enhancements used in the Single Observer. All experiments were done with a thinking time of 2 and 4 seconds. The results of this experiment are shown in Table 5.7. The PU approach for 2 seconds and the PB for 2 seconds played 400 games. The remaining experiments were performed 600 times.
The experiment shows that only MO-PB has some advantage over its SO variant. The SO-PB wins only 43.25% of the played games with 2 seconds thinking time. This advantage over the SO seems to become smaller when using a thinking time of 4 seconds. SO-PB wins 49.17% against MO-PB with 4 seconds thinking time. SO-PU wins ~ 50.33 of all games against the MO-PU with 4 seconds thinking time. The MO-PUB has no advantage with 2 seconds thinking time. SO-PUB wins 50.83% of its games against the MO-PUB. This changes if both have a thinking time of 4 seconds. SO-PUB wins 46.5% of all games under the same remaining conditions. For an easy comparison between the different thinking times, Figure 5.7 can be used.

Player	2 sec		4 sec	
	SO %	**SO Nr.**	**SO %**	**SO Nr.**
SO-PU vs. MO-PU	46.75 ± 4.89	187	50.33 ± 4	302
SO-PB vs. MO-PB	43.25 ± 4.86	173	49.17 ± 4	295
SO-PUB vs. MO-PUB	50.83 ± 4	305	46.5 ± 3.99	279

Table 5.7: SO vs. MO both with Enhancements

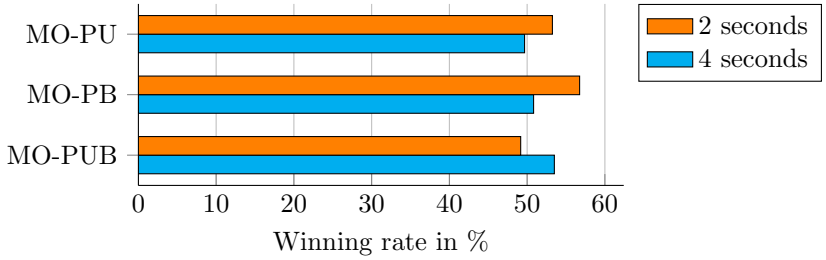

Figure 5.7: Visualization of MO vs. SO both with enhancements

5.7 Multiple Trees MCTS

In this section the two different strategies, which use more than one tree to map their results, are discussed. Both strategies were explained in Subsection 3.4.8. First both approaches are tested without any enhancements. Determinized UCT is then enhanced with progressive unpruning and progressive bias. Finally, the voting player is enhanced with all three progressive approaches.

5.7.1 SO vs. Determinized UCT and Voting

Determinized UCT (DZ) and voting (V) was tested with 1, 5, 10 and 20 trees and played against the SO with the same enhancements. It has to be considered that both approaches when they are using only one tree are the same. This also means that for each tree only a certain randomly calculated determinization was chosen. Determinized UCT with 1 and 10 trees played 400 games instead of the usual 200 games. The results can be seen in Table 5.8 and Figure 5.8.
The results show that the SO performs clearly better than all voting agents. The determinized UCT also seems to perform better than the voting player, but in most cases this improvement is rather small. The only approach, where the voting player performed better than the determinized UCT against the SO was for 20 trees. The voting player was able to win 41.5% of all games, while the determinized UCT player was only able to win 38.5% of all games. The determinized UCT player was also not able to perform significantly better than the SO. Determinized UCT performs best with using only one tree, in which case no player performs significantly better than the other one.

Player	Determinized UCT		Voting	
	SO %	**SO Nr.**	**SO %**	**SO Nr.**
SO vs. 1 Tree	50.75 ± 4.9	203	58 ± 6.84	116
SO vs. 5 Trees	57.5 ± 6.85	115	58.5 ± 6.83	117
SO vs. 10 Trees	56.25 ± 4.86	225	62.5 ± 6.71	125
SO vs. 20 Trees	61.5 ± 6.74	123	58.5 ± 6.83	117

Table 5.8: SO vs. Determinized UCT and Voting

5.7.2 Determinized UCT

In this subsection, determinized UCT is closer analyzed. The enhancements, which were tested, are progressive unpruning or progressive bias. The results are shown in Table 5.9.
The results show, that SO-PB performs clearly better than all DZ players. These results can be compared

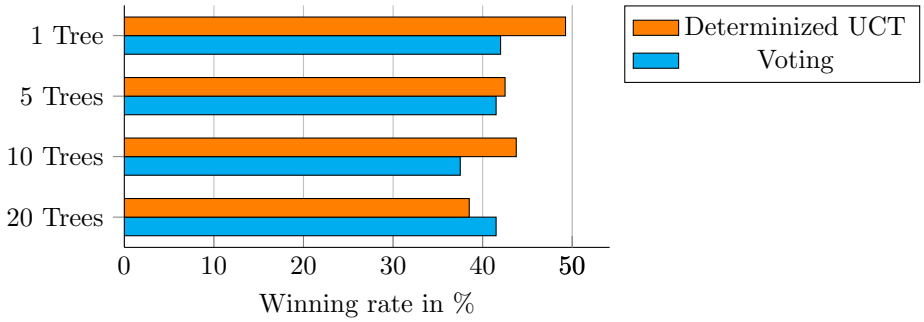

Figure 5.8: Visualization of the results of DZ and V vs. the SO

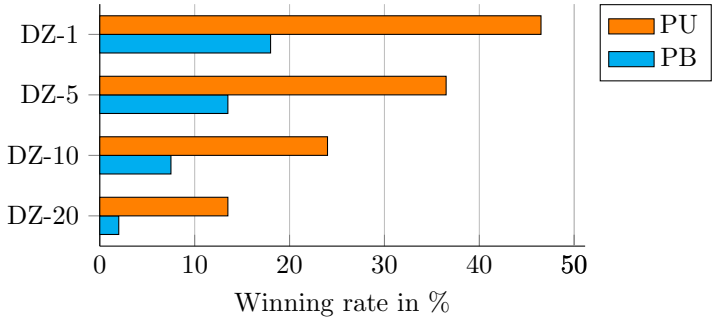

Figure 5.9: Visualization of DZ vs. SO both with enhancements

to the results of the SO versus the DZ player, seen in Table 5.8. Since the SO won at most 62% of all games against the DZ player, when using no enhancements, it can be shown that PB improves the SO more than all DZ players. The SO-PB was able to win at least 82% of all played games against the DZ-PB players.
SO-PU performs better than the DZ agents in most cases. DZ-1-PU wins 46.5% of all games and so does not perform significantly worse than SO-PU, but it also does not perform better. For 10 and 20 trees the PU strategy performs better for the SO. It can be seen in the table that PU in general performs better for the DZ agents than PB.

Player	**Progressive Unpruning**		**Progressive Bias**	
	SO %	**SO Nr.**	**SO %**	**SO Nr.**
SO vs. DZ-1	53.5 ± 6.91	107	82 ± 5.32	164
SO vs. DZ-5	63.5 ± 6.67	127	86.5 ± 4.74	173
SO vs. DZ-10	76 ± 5.92	152	92.5 ± 3.65	185
SO vs. DZ-20	86.5 ± 4.74	173	98 ± 1.94	196

Table 5.9: SO vs. DZ both with enhancements

5.7.3 Voting

The voting agent was also enhanced. The enhancements are the same as the determinized UCT agent, but progressive unpruning and bias was also added to test their performance. The remaining experimental setup was the same as for determinized UCT. Table 5.10 shows the experiments, which were done for progressive unpruning. The results for progressive bias and progressive unpruning with progressive bias are represented by Table 5.11. Figure 5.10 shows all results with 2 seconds thinking time.
The general results are similar to the results from determinized UCT. All voting agents perform strictly worse than the SO. Progressive unpruning was also tested with 4 seconds thinking time. The voting player with 5 trees performed slightly better than before, but not significantly. It is also important to notice that SO-PB performs less against the V agents than the DZ agents. Though the results of PU

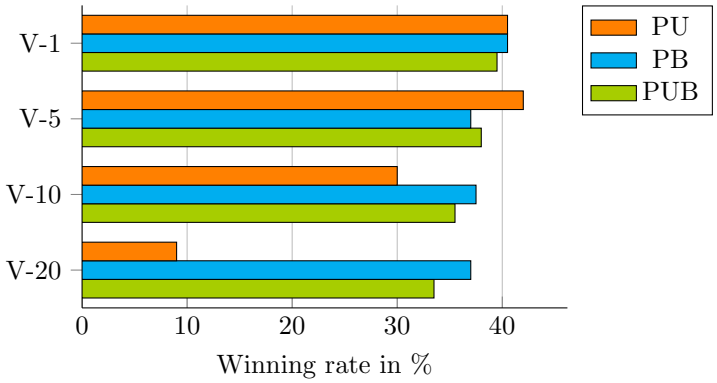

Figure 5.10: Visualization of V vs. SO both with different enhancements results

strategy does not show a significant difference for either the DZ or the V player.
The results show less difference for the voting variant, when using different numbers of trees. V-PB only has a winning difference of 7 games, while DZ-PB has a winning difference of 32 games. V-PU still has a difference of 66 games and is so equal to the variance of the determinized variant.

Player	**2 sec**		**4 sec**	
	SO %	**SO Nr.**	**SO %**	**SO Nr.**
SO vs. V-1	59.5 ± 6.8	119	56 ± 6.88	112
SO vs. V-5	58 ± 6.84	116	63 ± 6.69	126
SO vs. V-10	70 ± 6.35	140	- ± -	-
SO vs. V-20	91 ± 3.97	182	- ± -	-

Table 5.10: SO vs. V both enhanced with progressive unpruning

Player	**Progressive Bias**		**Progressive Unpruning + Bias**	
	SO %	**SO Nr.**	**SO %**	**SO Nr.**
SO vs. V-1	59.5± 6.8	119	60.5± 6.78	121
SO vs. V-5	63± 6.69	126	62± 6.73	124
SO vs. V-10	62.5± 6.71	125	64.5± 6.63	129
SO vs. V-20	63± 6.69	126	66.5± 6.54	133

Table 5.11: SO vs. V both enhanced with PB and PUB

5.8 Early Termination Approach

This section tests the early termination approach discussed in Subsection 4.7.2. Different agents were tested against each other. All of them were not enhanced by selection strategies or playout policies except SO-PUB where it is explicitly stated. In all tests the two agents perform similarly, but one agent terminates the playout step after 45 moves while the other agent simulates the game to the end. The player, which uses the termination strategy, gets a T added to its description. The results of these experiments can be seen in Table 5.12 and Figure 5.11.

In general, the two approaches seem to have the same performance. For the cheating player, the termination enhancement seems to perform slightly better than for the other agents. But this result is not significant. FSO, SO, PUB and MO win in general a similar number of games to the early termination variant.

Pl. 1	Pl. 2	Pl. 1 %	Pl. 2 %	Pl. 1 Wins	Pl. 2 Wins
CP-T	CP	55 ± 6.89	45 ± 6.89	110	90
SO-T	SO	50 ± 6.93	50 ± 6.93	100	100
FSO-T	FSO	50.5 ± 6.93	49.5 ± 6.93	101	99
SO-PUB-T	SO-PUB	51.5 ± 6.93	48.5 ± 6.93	103	97
MO-T	MO	47 ± 6.92	53 ± 6.92	94	106

Table 5.12: Termination results

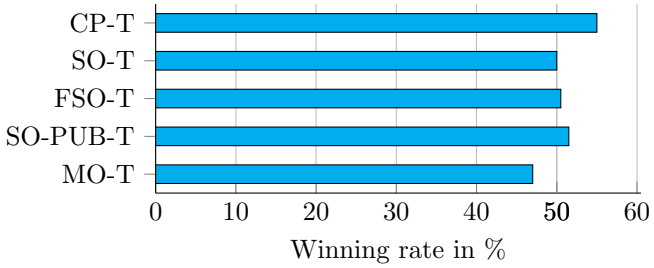

Figure 5.11: Visualization of Termination results

5.9 Number of Visits vs. Ratio

In this section, agents with different choosing strategies, which determine the move to play in the actual game, were tested against each other. One of the agents chooses the node with the most number of visits, the other agent chooses the node with the highest winning ratio. All experiments were run for 400 games instead of the normal 200 games. The players were not enhanced by another strategy. Table 5.13 shows the results of these experiments. As seen in the table, the agent, which chooses the node with the most number of visits, performs better than the agent, which chooses the node with the highest winning ratio. For the cheating player and the voting 5 agent, the difference is only small. The FSO seems to gain the most benefit when choosing a node by using the highest number of visits. The results were also visualized in Figure 5.12.

Player	Visits %	Ratio %	Visits Wins	Ratio Wins
CP	51.5 ± 4.9	48.5 ± 4.9	206	194
FSO	61.25 ± 4.77	38.75 ± 4.77	245	155
SO	56.5 ± 4.86	43.5 ± 4.86	226	174
MOMCTS	58.5 ± 4.83	41.5 ± 4.83	234	166
DZ 5	56.75 ± 4.86	43.25 ± 4.86	227	173
V5	51.5 ± 4.9	48.5 ± 4.9	206	194

Table 5.13: Number of Visits vs. Ratio

5.10 Combining PUB with other Approaches

This experiment tries to combine PUB with other approaches. All results can be seen in Table 5.14 and Figure 5.13.

PUB was tested and combined with several approaches to see, if it can be enhanced. PUB was tested against three different approaches, PUB with worst missions (SO-PUB-WM), PUB with a playout strategy (SO-PUB-PS) and Multiple-Observer PUB with a playout strategy (MO-PUB-PS).

SO-PUB-PS was also tested against its Multiple-Observer version. This experiment was also done with the enhancement of worst missions to both agents.

As the result shows, SO-PUB cannot be enhanced. The further addition of the PS to the SO and the MO first seemed to work. Afterwards it was tested, if the agent actually performs better than SO-PUB. The results show that SO-PUB performs better than MO-PUB-PS. SO-PUB and SO-PUB-PS wins

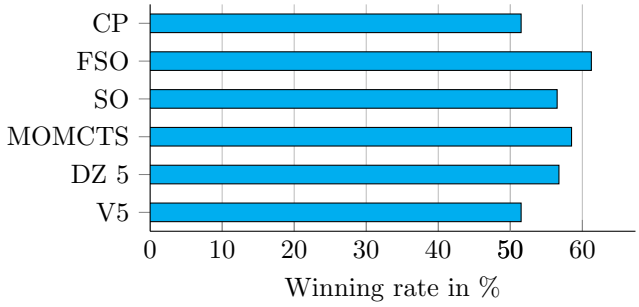

Figure 5.12: Visualization of Number of Visits vs. Ratio

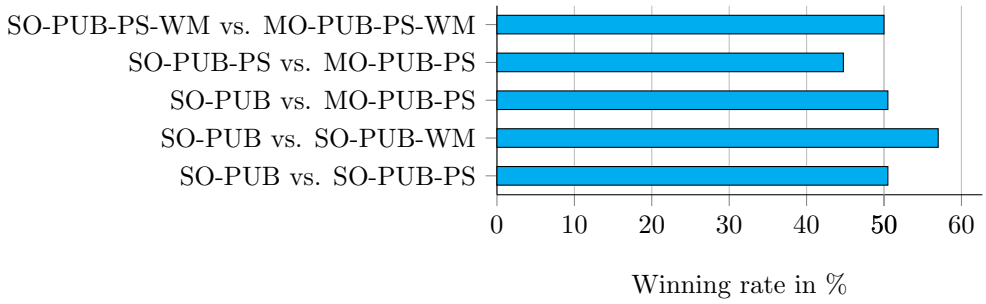

Figure 5.13: Visualization of PUB results

nearly the same number of games. SO-PUB was strictly better than SO-PUB-WM. SO-PUB-PS-WM and MO-PUB-PS-WM win, similarly to SO-PUB against SO-PUB-PS, an equal number of games.

Pl. 1	**Pl. 2**	**Pl. 1 %**	**Pl. 2 %**	**Pl. 1 Wins**	**Pl. 2 Wins**
SO-PUB	SO-PUB-PS	50.5 ± 6.93	49.5 ± 6.93	101	99
SO-PUB	SO-PUB-WM	57 ± 6.86	43 ± 6.86	114	86
SO-PUB	MO-PUB-PS	50.5 ± 6.93	49.5 ± 6.93	101	99
SO-PUB-PS	MO-PUB-PS	44.75 ± 4.87	55.25 ± 4.87	179	221
SO-PUB-PS-WM	MO-PUB-PS-WM	50 ± 6.93	50 ± 6.93	100	100

Table 5.14: Combining PUB with other approaches

5.11 Cheating Player vs. Different Agents

In this section different strategies were tested against the CP, which shuffles the stack, and CP-U, which does not shuffle the stack. The results can be seen in Table 5.15. All agents played 200 games. CP-U-PUB played 400 games against SO-PUB.
When comparing the results of the first experiment, FCP-PUB against the CP-PUB (see Table 5.2), it can be clearly seen that the PUB approach enhances the CP strategy, because both CP now perform better than the FCP. This was not the case in the previous experiment. As can be seen in the results, most CP perform clearly better than their opponents. The CP and CP-PUB win 58.5% of all games played against the SO and SO-PUB. The CP performs a little bit less against MO with winning only 57.5% of all games. CP-PUB only wins 57% of all games against MO-PUB.
The CP-U seems to work better than the CP except for CP-U-PUB plays against SO-PUB. The result shows that CP-U-PUB performs better than the SO-PUB with winning 55.25% of all games. CP-U performs best against the SO. CP-U wins 64% of all games. The CP-U wins 62.5% of all games against MO and CP-U-PUB wins only 61% of all games against MO-PUB.
A visualized part of the results can be seen in Figure 5.14. It shows that CP generally performs less than CP-U except for playing against SO-PUB.

Player	CP Shuffled		CP Fixed (CP-U)	
	CP %	**CP Nr.**	**CP %**	**CP Nr.**
CP-PUB vs. FCP-PUB	59.5 ± 6.8	119	61.5 ± 6.74	123
CP vs. SO	58.5 ± 6.83	117	64 ± 6.65	128
CP-PUB vs. SO-PUB	58.5 ± 6.83	117	55.25 ± 3.45	442
CP vs. MO	57.5 ± 6.85	115	62.5 ± 6.71	125
CP-PUB vs. MO-PUB	57 ± 6.86	114	61 ± 6.76	122

Table 5.15: CP vs. different agents

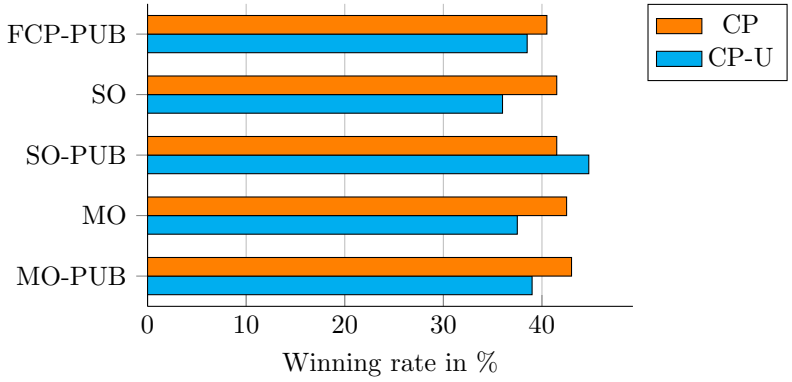

Figure 5.14: Visualization of CP vs. different agents

5.12 Completing destination tickets

During the experiments it became apparent that unimproved strategies like the SO were not able to complete destination tickets. In a normal two player game, both players should be able to fulfill at least two destination tickets most of the time.

As an example, when the FSO played against the SO with 2 seconds thinking time, the FSO was able to complete 0.5 destination tickets while the SO was able to complete 0.51 destination tickets on average. The average number of victory points for both players is 42.07. This is not a good result for any Ticket to Ride player. These results could be explained with too few simulations and the too complex structure of Ticket to Ride. Especially the first destination tickets take the longest time to complete.

As another example, when the FSO-PUB played against the SO-PUB with 2 seconds thinking time, the FSO-PUB was able to complete 3.83 destination tickets on average. The SO-PUB was able to complete 4.11 destination tickets. This suggests that the enhancements actually guide the MCTS algorithm to completing more missions. This results also in more points. Both players are able to score 96.58 victory points. The victory points of the unimproved FSO and SO are therefore more than doubled.

Chapter 6

Conclusion and Future Research

This chapter concludes the thesis. The thesis and its most important idea are summarized. Afterwards the conclusion for the thesis is drawn by answering the research questions and the problem statement. The thesis ends with the presentation of ideas for future research.

6.1 Thesis summary

An MCTS agent for Ticket to Ride tried to find an answer to the question, if an MCTS agent would be feasible for Ticket to Ride. To give an introduction in the thesis itself, the development of games with artificial players was shown. The success of MCTS was also presented. After a short introduction in the game itself, a short overview over different search algorithms was given, including MCTS. The chapter also introduced the research questions and problem statement of this thesis.

To build an MCTS agent for Ticket to Ride the rules of the game and some strategies for the game were explained. To show how complex the game is, the state-space complexity and game-tree complexity were calculated. Through this calculation an overestimation for the state-space complexity was found. It was calculated that $\sim 9.45 \times 10^{54}$ different board configurations could be possible. The game-tree complexity was also calculated. The number of possible leaf nodes also was an estimation and the value was calculated to be $\sim 8.75 \times 10^{228}$. These results were compared to chess. It was shown, that TTR is more complex then chess. The game-tree complexity is much higher with $\sim 10^{108}$ more possible leaf nodes. The possible number of board configurations differentiate by 10^{11} boards.

In the next chapter, the MCTS algorithm, as well as Flat Monte-Carlo were explained. Different enhancements were discussed. The focus was on enhancements to handle imperfect information. For this problem determinization and several other enhancements were introduced. All of the enhancements, which do not try to improve a certain step of the MCTS algorithm, are designed for determinization. The Single-Observer was the closest to the standard MCTS agent, but also other agents were shown. The other three agents were the Multiple-Observer, the determinized UCT and the voting strategy, which is similar to the determinized UCT strategy except the way the actual move is chosen. Also other enhancements were introduced. These enhancements focused on a specific step of the MCTS algorithm. The most common one is UCT, which uses information from the parent as well as the child to select a node. Progressive unpruning and progressive bias were also explained. Also two playout strategies were discussed, early termination and ϵ-based strategies.

In the fourth chapter the implementation problems of Ticket to Ride in MCTS were shown. Therefore several steps had to be explained, like the implementation of the node and its actions. Also certain problems with the mission action or generating a determinized board were explained. Some of the enhancements needed specific values, which had to be fitted to Ticket to Ride. One of them was heuristic knowledge, which was needed for progressive unpruning and progressive bias. The implementation of the

playout strategy also needed a more detailed explanation.

In the fifth chapter, the introduced players and enhancements were tested. Therefore the setup of the experiments was described. All values from certain constants of different players, were stated as well. Afterwards different experiments were performed and evaluated.

The experiment showed, that MCTS was not necessarily working better than Flat Monte-Carlo. In most cases the addition of the tree could enhance a player. Therefore it was decided to enhance both players even further. The addition of progressive unpruning, progressive bias or a combination of both, seemed to make both players at least equal. The Single-Observer with one of the progressive enhancements seemed to work better than its Flat Monte-Carlo variant when having a thinking time of 4 seconds.

Through the experiments it can be concluded as well that several approaches like the voting player or the determinized UCT player do not work when they are enhanced with one of the progressive approaches. Multiple-Observer seemed to work for progressive bias and progressive unpruning with 2 second thinking time. Otherwise it did not seem to make a difference except for the combination of both progressive approaches, which worked better, when it had 4 seconds thinking time.

It was also tested, if the actual selection of the move would be better, if the node with the highest winning ratio would be chosen. It was shown that most players could not be improved with this choosing strategy. For the cheating player and the voting 5 player it is not proven that one strategy is better than the other one. At the moment it seems that neither strategy improves the cheating nor the voting 5 player.

The early termination could only bring a small enhancement to the cheating player. Otherwise the termination approach did not work better for most approaches except Multiple-Observer, which performed worse. Overall it was found that progressive unpruning combined with progressive bias worked best, so it was tested combined with several other approaches. In the end, it was not clear if progressive unpruning combined only with progressive bias works best, but other different enhancements with which it was tested, did not give a boost in performance or made the agent perform worse.

The two approaches to choose missions during the determinization seemed to work better for 2 seconds thinking time. When changing the thinking time to 4 seconds both agents performed as well as the Single-Observer. It was tried to add the better working approach to the combination of progressive unpruning combined with progressive bias. This approach was the one who chooses the destination tickets such that it would be worse for the momentarily active player. This player was tested against the same player without the worst missions enhancement. The more enhanced player performed significantly worse.

The cheating player was the last player, which was tested. The cheating player was tested in two different ways. One player knew the stack the other one did not. Both agents performed better than the Single-Observer and the Multiple-Observer they were tested against. But a unexpected result was obtained when both agents were enhanced with the combination of the progressive approaches. There the player which knew the stack, did not have a necessarily better performance against the Single-Observer. Both types of cheating players were tested against the Flat Monte-Carlo player. Both were not able to defeat the Flat Monte-Carlo more than 56% percent of the time, which is unusual. If both strategies were enhanced with the combination of progressive unpruning and progressive bias, the cheating player was able to defeat the flat cheating player about 59.5% of all games.

6.2 Answering the Research Questions

The research questions were given in Section 1.3. These questions are answered first, before an answer to the problem statement is given.

Are Monte-Carlo methods applicable to Ticket to Ride?
Monte-Carlo methods seem to work in Ticket to Ride. Without any enhancement the Flat Monte-Carlo seemed to work best for a thinking time of 2 seconds. If the player should not cheat, determinization had

to be implemented such that the implemented player would have the same preconditions as a humanoid player.

How can MCTS handle the imperfect-information game in the domain of TTR?
To test how well MCTS handles imperfect information several strategies were tested against the cheating MCTS player. The Single- and Multiple-Observer perform worse than the cheating player. This result was expected, because the cheating player knows the cards of his opponent and can block the player. The Single-Observer and Multiple-Observer win 42% of all games against the cheating player, which is shuffled. The Single-Observer and Multiple Observer performed worse against the cheating player with non-shuffled stacks by winning only 39% of all played games. Considering the disadvantage of the Single-Observer and Multiple-Observer, they still perform well against both types of cheating player.

Which enhancements can improve the MCTS player?
As discussed before, progressive bias, progressive unpruning and the combination of both can enhance MCTS. The Multiple-Observer enhances MCTS only under certain conditions. The playout strategy was not able to enhance the MCTS player. Terminating the playout early as well as using the voting or determinized UCT player, do not enhance the MCTS player.

6.3 Answering the Problem Statement

This section will answer the following question, which was first introduced in Section 1.3:

How can a MCTS player for TTR be developed?

Developing an MCTS player for Ticket to Ride was not as easy as it first may have seemed to be. Several facts and conditions have to be kept in mind. One of the bigger problems was that Ticket to Ride has hidden information and chance events. Therefore determinization has to be used, so MCTS has the same conditions as a human. Determinization had several problems of its own, which needed further investigation. Therefore several other strategies had to be closer analyzed, like the Single- and Multiple-Observer. Through the high variety of possible missions and the game length, MCTS seemed not to be able to fulfill missions on its own. Several approaches were tested improve the performance. The combination of progressive unpruning and progressive bias seems to work the best. A further improvement of this approach could not be shown in this thesis. The best player seems to be progressive bias combined with progressive unpruning either for the Single-Observer or the Multiple-Observer. Neither Single-Observer nor Multiple-Observer seems to perform better than the other when enhanced with the progressive combination. To enhance the best player even more seemed to be difficult, because the more enhanced player either performed worse or there was no significant difference.

6.4 Future Research

This section describes ideas for future research. Several strategies use constants. These constants can be tweaked, as discussed in the corresponding parts. The C value of UCT is used in most strategies and therefore would be a good choice. Progressive unpruning, progressive bias and the combination of both could also be an interesting choice to tune the parameters, because progressive bias combined with progressive unpruning is the best strategy and could maybe be enhanced by testing different values for the T constant. The values for A and B from the progressive unpruning could also be a good choice. Tuning the ϵ parameter of the rule-based strategy could maybe increase the performance. Testing more trees for determinized UCT and the voting strategy does not seem to be plausible, because it seemed to perform worse the more trees were used.

The second idea for future research possibilities could be to test the game with three players or more. In theory the game can be played with up to five players. This thesis covers only the two player variant. It is important to notice that with more than three players, the game rules slightly change because more tracks become available. In general, the game with more than two players is more difficult, because it is harder to analyze in advance. Especially getting cards can become more difficult, since several players

may need the same color. It also gets more likely that at least two players need to build the same track. Therefore multiple tracks were introduced when playing with more than three players.

Similar to the idea of playing the game with more than two players, would be to test the strategies with different expansions or stand-alone editions. Most expansions can only be played on a certain board, so they add most of the time only one or two rules. This changes the strategies a human player would use and it would be interesting to see, if MCTS is able to adapt to this change of ruling and is able to perform similarly. An interesting edition could be the Europe variant, where a player is able to use tracks of other players then placing certain pieces on the map. Another expansion, which was designed for two or three players, was the Netherlands expansion. It also uses double tracks, but players can build on both of them, even with only two or three players instead of the normal four or five players.

Another possibility would be, to test the strategies against different human players to see if the strategies also work against them. Until now it was only tested against Monte-Carlo players and it would be interesting to know, if MCTS is able to win against a human player.

The last idea would be to change the selection of the first destination tickets. The tickets could also be chosen with domain knowledge. Therefore a test of all different combinations of destination tickets is necessary. Each pair of destination tickets would be tested against different other combinations multiple times. For each test, the combination of the beginning tickets and the end score of the player has to be saved. This knowledge would be used to choose the two tickets with the highest total score.

References

Allis, L. V. (1994). *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. thesis, University of Limburg, Maastricht, The Netherlands. [8]

Brügmann, B. (1993). Monte Carlo Go. Technical report, Physics Department, Syracuse University. [1, 11]

Browne, C. (2013). A Problem Case for UCT. *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 5, No. 1, pp. 69–74. [24]

Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intellig. and AI in Games*, Vol. 4, No. 1, pp. 1–43. [3]

Buro, M., Long, J. R., Furtak, T., and Sturtevant, N. R. (2009). Improving State Evaluation, Inference, and Search in Trick-Based Card Games. *IJCAI* (ed. C. Boutilier), pp. 1407–1413. [13]

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

Cazenave, T. (2006). A Phantom-Go Program. *Advances in Computer Games* (eds. H. J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H. H. L. M. J. Donkers), Vol. 4250 of *Lecture Notes in Computer Science*, pp. 120–125. Springer Berlin Heidelberg. [13]

Chaslot, G. M. J.-B., Winands, M. H. M., van den Herik, H. J., Uiterwijk, J. W. H. M., and Bouzy, B. (2008). Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, Vol. 4, pp. 343–357. [3, 12, 13]

Coulom, R. (2007a). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *Computers and Games* (eds. H. J. van den Herik, P. Ciancarini, and H. H. L. M. J. Donkers), Vol. 4630 of *Lecture Notes in Computer Science*, pp. 72–83. Springer Berlin Heidelberg. [1, 2, 11]

Coulom, R. (2007b). Computing "Elo Ratings" of Move Patterns in the Game of Go. *ICGA Journal*, Vol. 30, No. 4, pp. 198–208. [3, 12]

Cowling, P. I., Powley, E. J., and Whitehouse, D. (2012). Information Set Monte Carlo Tree Search. *IEEE Trans. Comput. Intellig. and AI in Games*, Vol. 4, No. 2, pp. 120–143. [2, 3, 13, 14, 15]

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik 1*, Vol. 1, No. 1, pp. 269–271. [2]

Douglas, J. R. (1978). Chess 4.7 Versus David Levy. *Byte Magazine*, Vol. 3, No. 12, pp. 86–93. [1]

Fern, A. and Lewis, P. (2011). Ensemble Monte-Carlo Planning: An Empirical Study. *ICAPS* (eds. F. Bacchus, C. Domshlak, S. Edelkamp, and M. Helmert), AAAI. [15]

Frank, I. and Basin, D. A. (1998). Search in Games with Incomplete Information: A Case Study Using Bridge Card Play. *Artificial Intelligence*, Vol. 100, Nos. 1–2, pp. 87–123. [13]

Ginsberg, M. L. (1999). GIB: Steps Toward an Expert-Level Bridge-Playing Program. *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99*, pp. 584–589. [13]

Ginsberg, M. L. (2001). GIB: Imperfect Information in a Computationally Challenging Game. *J. Artif. Intell. Res. (JAIR)*, Vol. 14, pp. 303–358. [2]

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107. [2]

Knuth, D. E. and Moore, R. W. (1975). An Analysis of Alpha–Beta Pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293–326. [2]

Kocsis, L. and Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. *Machine Learning: ECML 2006* (eds. J. Fürnkranz, T. Scheffer, and M. Spiliopoulou), Vol. 4212 of *Lecture Notes in Computer Science*, pp. 282–293. Springer Berlin Heidelberg. [1, 11, 12]

Lorentz, R. J. (2008). Amazons Discover Monte-Carlo. *Computers and Games* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *Lecture Notes in Computer Science*, pp. 13–24. Springer Berlin Heidelberg. [3, 15]

Luckhart, C. and Irani, K. B. (1986). An Algorithmic Solution of N-Person Games. *AAAI*, pp. 158–162, Morgan Kaufmann. [2]

Michie, D. (1966). Game-playing and Game-learning Automata. *Advances in Programming and Non-Numerical Computation, Leslie Fox*, Vol. 47, No. 7, pp. 183–200. [2]

Moon, A. R. (2015). Ticket to Ride. `http://cdn1.daysofwonder.com/tickettoride/en/img/tt_rules_2013_en.pdf`. [Online; accessed 22.02.2015]. [7]

Nijssen, P. and Winands, M. H. M. (2012). Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard. *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 4, No. 4, pp. 282–294. [3, 15]

Shannon, C. E. (1950). XXII. Programming a Computer for Playing Chess. *Philosophical Magazine (Series 7)*, Vol. 41, No. 314, pp. 256–275. [9]

Szita, I., Chaslot, G., and Spronck, P. (2010). Monte-Carlo Tree Search in Settlers of Catan. *Advances in Computer Games* (eds. H. J. van den Herik and P. Spronck), Vol. 6048 of *Lecture Notes in Computer Science*, pp. 21–32. Springer Berlin Heidelberg. [2]

Von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press. [2]

Wedd, N. (2015). Human-Computer Go Challenges. `http://www.computer-go.info/h-c/`. [Online; accessed 09.02.2015]. [1]

Winands, M. H. M. and Björnsson, Y. (2010). Evaluation Function Based Monte-Carlo LOA. *Advances in Computer Games* (eds. H. J. van den Herik and P. Spronck), Vol. 6048 of *Lecture Notes in Computer Science*, pp. 33–44. Springer Berlin Heidelberg. [15]

List of Tables

2.1 Points for claimed tracks. Left side of the table states, how many trains a player needs for a track. On the right side the according victory points are shown. 6
2.2 Number of tracks for a certain number of trains. 9
2.3 Possibilities for different action types 10

4.1 Heuristic values for different actions 20

5.1 Single Observer 25
5.2 FCP vs. CP 25
5.3 FSO vs. SO with Enhancements 26
5.4 SO vs. FSO both with Enhancements 27
5.5 SO vs. SO with enhancements 27
5.6 Testing enhancement against each other in the SO agent 28
5.7 SO vs. MO both with Enhancements 29
5.8 SO vs. Determinized UCT and Voting 29
5.9 SO vs. DZ both with enhancements 30
5.10 SO vs. V both enhanced with progressive unpruning 31
5.11 SO vs. V both enhanced with PB and PUB 31
5.12 Termination results 32
5.13 Number of Visits vs. Ratio 32
5.14 Combining PUB with other approaches 33
5.15 CP vs. different agents 34

List of Figures

2.1 Two tracks connecting two cities 6
2.2 Two tracks connecting two cities 6
2.3 Options for a player 10

3.1 MCTS 12
3.2 Possible moves for different determinizations 14
3.3 MCTS algorithm 14

5.1 Visualization of SO results 25
5.2 Visualization of FCP vs. CP results 26
5.3 Visualization of FSO vs. SO with Enhancements results 26
5.4 Visualization of FSO vs. SO both with enhancements 27
5.5 Visualization of SO against different enhancements 28
5.6 Visualization of different enhancements playing against each other 28
5.7 Visualization of MO vs. SO both with enhancements 29
5.8 Visualization of the results of DZ and V vs. the SO 30
5.9 Visualization of DZ vs. SO both with enhancements 30
5.10 Visualization of V vs. SO both with different enhancements results 31
5.11 Visualization of Termination results 32
5.12 Visualization of Number of Visits vs. Ratio 33
5.13 Visualization of PUB results 33
5.14 Visualization of CP vs. different agents 34