Carlo Nübel

Match Point AI: A Novel Reinforcement Learning Framework for Evaluating Data-Driven Tennis Strategies

FAKULTÄT FÜR
INFORMATIK

Intelligent Cooperative Systems
Computational Intelligence

Match Point AI: A Novel Reinforcement Learning Framework for Evaluating Data-Driven Tennis Strategies

Master Thesis

Carlo Nübel

March 11, 2024

Supervisor: Prof. Dr.-Ing. habil. Sanaz Mostaghim

Advisor: Dr.-Ing. Christoph Steup

Advisor: Sebastian Mai, M.Sc.

Carlo Nübel: *Match Point AI: A Novel Reinforcement Learning Framework for Evaluating Data-Driven Tennis Strategies*
Otto-von-Guericke Universität
Intelligent Cooperative Systems
Computational Intelligence
Magdeburg, 2024.

Abstract

Monte-Carlo Tree Search (MCTS) is a Reinforcement Learning (RL) algorithm that is capable of optimizing different types of decision-making processes. It has been applied to a wide range of board and computer games as well as real-world decision-making problems. The shot direction selection in a tennis match is an interesting decision-making problem to which the MCTS algorithm has not yet been applied. In this thesis, we present the tennis match simulation environment *Match Point AI*, in which different adaptations of the MCTS algorithm compete against real-world data-driven bot strategies. This thesis aims to gain further insight into the MCTS algorithm's ability to solve this decision process. We also discuss, whether *Match Point AI* is a suitable environment for this purpose.

Firstly, we test our environment for its ability to generate reasonable shot-by-shot tennis data by comparing the generated results with real-world tennis data. Although the agents and bots are limited in their choices compared to real tennis, we are still able to generate realistic shot-by-shot tennis data from which we can deduce plausible strategies used by the agents to make decisions. By comparing different selection and decision policies, as well as various parameter settings in the MCTS algorithm, we then evaluate which algorithmic design is best to solve the shot selection decision process in tennis. We found that agents using decision policy *Greedy* perform significantly better than agents using *Upper Confidence Bounds for Trees (UCT)*. We also see that agents with selection policy *Random* perform significantly better in *Match Point AI* against the Djokovic Bot than agents using either selection policy *UCT* or *Greedy*.

Match Point AI has a lot of potential for optimization, some of which will require more and better real-world tennis data. As the environment is improved, the generated results and thus the insight into tennis and the MCTS algorithm will become more valuable. There are also parameter settings as well as selection and decision policies that need to be tested in this simulator, to further broaden our understanding of the algorithm's abilities.

Contents

List of Figures

List of Tables

List of Acronyms

MCTS Monte-Carlo Tree Search
UCT Upper Confidence Bounds for Trees
AI Artificial Intelligence
RL Reinforcement Learning
DF Degrees of Freedom
MDP Markov Decision Process
DP Dynamic Programming
TDL Temporal Difference Learning
MC Monte-Carlo

1 Introduction

The men's professional tennis tour has been dominated by just three players for the past two decades, earning them the nickname *the Big Three*. Between 2003 and 2023 the Big Three won 66 out of 80 Grand Slams, the world's biggest tennis tournaments. While Roger Federer has recently retired and Rafael Nadal has been plagued by injury for the past year, in 2023, the last of the Big Three, Novak Djokovic, has once again won three of the four Grand Slams, despite his, for a professional tennis player, advanced age. Many tennis fans are wondering when the new generation will ascend to the tennis throne. However, they seem to have yet to find a way to defeat Novak Djokovic. Even though the Big Three kept winning and winning, many areas of our world have changed dramatically over the last 20 years. Particularly in the field of Artificial Intelligence (AI), rapid advances have opened up new possibilities in almost every field, including sports. With this thesis we want to see if this progress has led to the possibility of applying AI to the decision-making processes in the sport of tennis, to gain insight into strategies and shot placement, and perhaps even to help young tennis players in their quest to defeat Novak Djokovic.

1.1 Motivation

The emergence of sports data analytics can be attributed to the increased use of advanced technologies and powerful computing capabilities, which enable the collection, processing, and analysis of large amounts of sports performance data. The data itself can already provide deep insights into athletic performance, tactical patterns, and performance optimization, which can then be used to achieve the ultimate goal in any sport: to win. In recent years, there have been many advances at the intersection of the fields of AI and sports. The application of AI algorithms to sports data analysis has rapidly improved the quality and quantity of the insights gained into the field of sport. Today, the applications of AI in sports are diverse, ranging from improving the fan

experience, for example by generating automated video highlights, to helping coaches to better understand the strategies and decision-making processes in their respective sport [43]. If we look at the application of RL algorithms to decision-making problems in general, we find many different application areas. RL algorithms have been successfully applied to navigation problems in robotics and to optimize decision-making processes in autonomous driving, various game-playing scenarios, as well as in sports. A widely used RL algorithm is the MCTS algorithm. The development of MCTS as it exists today is the result of the collaborative efforts of several researchers over time. The algorithm originates from the Monte-Carlo (MC) method, which was used to solve complex mathematical integrals using random numbers. The method dates back to the late 1940s [42]. It is named after the city of Monte-Carlo, because one of the inventors of the method, Stanley Ulam, had an uncle, who regularly borrowed money from his relatives to go to the casinos in Monte-Carlo to gamble [41]. In 2006 two important contributions were made to the algorithm. First, Rémi Coulom introduced a framework that combined the MC method with a tree search algorithm. This was then applied to a 9x9 *Go* playing framework, which went on to win a computer-*Go* tournament [12]. Secondly, Kocsis and Szepesvári introduced the UCT algorithm, which can be used to effectively deal with the trade-off between exploration and exploitation, which is a fundamental challenge in the field of RL and will be addressed later in this thesis. MCTS is probably best known for its achievements in the game of *Go*, where it was used in combination with deep neural networks to defeat the former world champion Lee Sedol in 2016 [55]. MCTS is a heuristic search algorithm that uses a decision tree to simulate and explore potential outcomes of the given decision-making problem. The main objective of this thesis is to evaluate the MCTS algorithm when it is applied to the decision-making processes in the sport of tennis. To excel in tennis, athletes require outstanding levels of fitness, highly developed motor skills as well as great strategic thinking and decision-making capabilities. These decision-making skills are of great interest to this thesis, as they are the aspect of a player's performance that we hope to improve by applying MCTS to tennis. However, the complexity of tennis is far too great to consider all aspects of the sport when applying the MCTS algorithm to it. To still be able to apply MCTS to tennis a tennis match simulation environment called *Match Point AI* has been created, in which the key elements of tennis are modelled. In this environment, two main approaches to playing tennis were implemented. On the one hand side, there

are different adaptations of the MCTS algorithm, which can learn strategies in the tennis simulator by playing against the opponents. To find useful strategies with this approach, the opponents have to show a behaviour that is close to real-world tennis strategies. Otherwise, the strategies found by the MCTS algorithm will not be related to real-world tennis. To achieve this, two different fixed-strategy bots were implemented, whose behaviour is based on probabilities from a real-world tennis dataset. With this setup, the different adaptations of the MCTS algorithm are tested in *Match Point AI* against the two different data-driven bot strategies in a variety of simulation matches. The resulting data of those experiments is then analyzed to see which strategies and shot patterns are found by the MCTS algorithm and to gain insight into which adaptation is best suited to solve the decision-making problems in tennis.

1.2 Research Questions and Hypotheses

The main objective of this work, as stated in the introduction, is to evaluate the MCTS algorithm when it is applied to the decision-making process in tennis. Additionally, we test *Match Point AI* in its ability to generate realistic shot-by-shot tennis data. This data is then used to analyze the tennis strategies found by the MCTS algorithm. To achieve this, we address the following research questions and test the corresponding hypotheses.

- **RQ1:** Is *Match Point AI* a suitable environment to gain insights into strategies in tennis, and where are its strengths and limitations?
    - H_{01}: There is no significant difference in the total number of points won by the MCTS agents in their matches against the Average Bot compared to their total points won in matches against the Djokovic Bot.
    - H_{11}: There is a significant difference in the total number of points won by the MCTS agents in their matches against the Average Bot compared to their total points won in matches against the Djokovic Bot.
- **RQ2:** What are the most frequent shot patterns, found by the MCTS agent, at the beginning of a rally when it is playing against the Djokovic Bot?

- **RQ3:** How does using the different selection policies *UCT*, *Greedy* and *Random* influence the performance of the MCTS algorithm?
  - H_{02}: There is no significant difference in the total number of points won by the MCTS agents when using the selection policies *UCT*, *Greedy* and *Random* to play against the two Bot strategies.
  - H_{12}: There is a significant difference in the total number of points won by the MCTS Agents when using the selection policies *UCT*, *Greedy* and *Random* to play against the two Bot strategies.
- **RQ4:** When using *UCT* as a selection policy in the MCTS Algorithm, how do different action selection policies, namely *UCT* and *Greedy*, influence the MCTS agents performance in *Match Point AI*?
  - H_{03}: There is no significant difference in the total number of points won by the MCTS agents when using the selection policies *UCT*, *Greedy* and *Random* to play against the two Bot strategies.
  - H_{13}: There is a significant difference in the total number of points won by the MCTS agents when using the selection policies *UCT*, *Greedy* and *Random* to play against the two Bot strategies.
- **RQ5:** How do different parameter settings of the MCTS algorithm influence its performance when it is applied to *Match Point AI*?
  - H_{04}: There is no significant difference in the total number of points won by the MCTS agent using the *UCT* selection policy and *Greedy* decision policy when using the C-values $\sqrt{2} - 0.5$, $\sqrt{2}$ and $\sqrt{2} + 0.5$.
  - H_{14}: There is a significant difference in the total number of points won by the MCTS agent using the *UCT* selection policy and *Greedy* decision policy when using the C-values $\sqrt{2} - 0.5$, $\sqrt{2}$ and $\sqrt{2} + 0.5$.
  - H_{05}: There is no significant difference in the total number of points won by MCTS agent using the *UCT* selection policy and *Greedy* decision policy when using 5, 10, and 15 simulations.
  - H_{15}: There is a significant difference in the total number of points won by the MCTS agent using the *UCT* selection policy and *Greedy* decision policy when using 5, 10 and 15 simulations.

Answering these questions will further broaden our understanding of the MCTS algorithm in general and especially in its application to decision-making processes that have deterministic and stochastic components, like the shot direction selection process found in tennis. It will also contribute to the sport of tennis. By analyzing the resulting shot-by-shot data generated by *Match Point AI*, it is possible to gain insight into successful strategies against specific real-world tennis players.

1.3 Thesis Structure

Chapter 2 will begin with an introduction to RL, in which we will cover important basics of the field. We then go on to introduce a few selected RL algorithms and present studies, in which those algorithms have been applied to different problems. After that, we will explain the basic principles and architecture of the MCTS algorithm, along with common alteration possibilities. Building on the knowledge about MCTS, we then showcase its diverse application in related research items across the domains of board and computer games, real-world decision processes, and sports. From those studies, we derive our research gap. This chapter concludes by giving a summary of different aspects of the sport of tennis that are relevant to this study and clarifying some terminology. In chapter 3, our methodology and the experiments conducted for this thesis will be explained. We begin by introducing the tennis match simulation environment *Match Point AI* and explain how an MCTS based agent is capable of making decisions within this environment. We then delve into the data analysis process employed to develop real-world data-driven fixed-strategy bots. After outlining the experiments, which are conducted to address the research questions, we then explain the methods utilized for the statistical analysis of our experiments and finally conclude the chapter by giving an overview of the software employed in this thesis. We present the results of the conducted experiments in chapter 4 and evaluate the results. Furthermore, we will address each research question and test the hypotheses. Chapter 5 then will conclude this thesis with a summary of the main aspects of this work followed by a discussion of the possibilities for future work on this topic.

2 Related Work

This chapter will start with a short introduction to the field of RL, home to the MCTS algorithm. We will cover the most important basics of the field and discuss commonly used algorithms and their application possibilities. These fundamentals are based on the book *Reinforcement Learning: An Introduction* [59] and on the content of the lecture on *Computational Intelligence in Games* at the Otto-von-Guericke Universtät Magdeburg by Sanaz Mostaghim. The chapter will go on to introduce the basics of the MCTS algorithm, including its four phases and different selection and decision policies. We then introduce studies in which the MCTS algorithm is applied to different fields, such as board and computer games, real-world problems, and sports. We continue to present related studies in which different modifications have been made to the algorithm and where the influence of those modifications on the algorithm's performance is analyzed. Here we distinguish this thesis from these existing studies and identify our research gap. This chapter concludes with a short introduction to the sport of tennis and clarifies a few important terms that will be used throughout this thesis.

2.1 Reinforcement Learning

The field of RL can be placed in the intersection between AI and decision-making in general. As with many of the algorithms in the field of Computational Intelligence, RL algorithms are inspired by dynamics observable in nature. The approach to learning is based on the way a child or dog learns, which is trial and error. By running back to its owner when called, a dog gets rewarded with a treat. That way, when called again, it will again run back to its owner, expecting a reward. Intelligent agents using RL algorithms can learn to make decisions to achieve specific goals by interacting with their environments the same way as children or dogs. Figure 2.1 shows a basic RL framework. An agent in this framework can apply actions to its environment and with it, cause

a change to it. That change can either move the agent closer to its goal, yielding him a reward, or deter him away from the goal, resulting in a punishment, which is usually done with a negative reward.

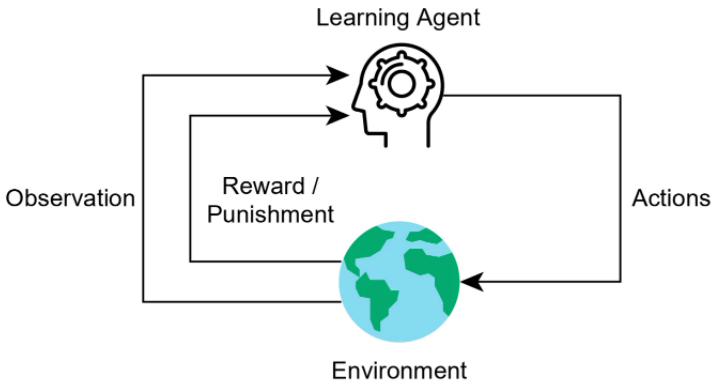

Figure 2.1: Components of a RL Framework [14]

This iterative interaction of an agent with the environment mirrors the trial and error learning by real-world humans and animals. Using this trial and error approach, RL agents can explore the possible actions in their environment to find the actions yielding the highest rewards. In every state s of the environment, the agent can select an action a from the set of all legal actions A_s in that state. The agent employs a mapping that associates each state with probabilities to determine the choice of the next action in that specific state. This probability distribution is often referred to as a *policy*, commonly denoted as π. During the interactions with the environment, the agent tries to optimize this policy, by changing it based on the rewards it receives.

2.1.1 Markov Decision Processes

An important concept in the field of RL is the Markov Decision Process (MDP). A RL task, in which the agent's current state is only dependent on the previous state and the action taken in that state, is called a MDP. Imagine a robot navigating through a grid world for example. The robot can move one cell at a time, either up, down, left, or right. His current position in any state is then only dependent on his previous position and the direction he chose to go to. This example fulfills the Markov property and is a MDP. As a counterexample, let us assume the grid world of our robot is changing randomly. At any given moment, a cell can be blocked off and no longer be available to the robot. This

would violate the Markov property because the robot's state now is not only dependent on its previous state and action but also on this randomly changing environment.

2.1.2 Reward and Return

A reward can be implemented in different ways. It depends on the problem the RL algorithm is facing, as well as the algorithm that is chosen to optimize the problem. Looking at the game of chess for example, when moving a chess piece to a tile where it can be captured by an opponent's piece, the reward should penalize that action in some way, for example by punishing the agent with a negative reward. When the chosen action is capturing one of the opponent's pieces, that action could be rewarded with a positive value of $+1$. The capturing of a queen can then also be rewarded with a higher value than the capturing of a pawn for example. Before giving rewards for an agent's actions it is important to know the problem and to find reasonable rewards for specific actions and states. Now, when we want to quantify the quality of an action in a state of a game, it is often not enough to only consider the impact on the environment in the next time step after the action is played. It is also important to consider all or at least some of the future states that can be reached in the environment when applying an action to the current state of the environment. Therefore, it is possible to look at a whole episode of a game. An episode starts at the current game state and ends when the game is over or at a predefined depth of the game. The episode contains all states and actions that are played from the current game state to the end of the game. The *Return* of an action is different from its reward and it uses the episodes of a game. When calculating the expected return for an action in a game, the expected return G_t of that action is the sum of all rewards R at each time step t to the last time step in the game T, as shown in formula 2.1.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.1}$$

The agent would then try to maximize this expected return when it has to choose the next action. Another approach to calculating the return is called *Discounted Return*. With this approach, all the rewards are weighed, based on how far in the future they would be received. Rewards of actions at the

beginning of the episode would get larger weights than those at the end of an episode of the game, as shown in formula 2.2.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{2.2}$$

The reward at *t+1* is fully included in the return because receiving it is guaranteed. This does not hold for the rewards after *t+1*, because they might never be received by the agent. This is done using the discounting factor γ, a value between 0 and 1. Receiving a reward is less and less likely the further we look into the future, which is exactly the aspect, that the discounted return approach addresses.

2.1.3 State Value functions vs. Action Value functions

A state value function, denoted by $v_\pi(s)$, addresses the question of how desirable it is to be in a specific state s whilst following a policy π. This desirability is determined by using the discounted return of that state s under policy π as shown in formula 2.3.

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} \mid s_t = s \right] \tag{2.3}$$

Value functions can be seen as a measure of potential future rewards that might be gained by being in a specific state following a policy.

An action-value function, denoted by $q_\pi(s, a)$, addresses the question of how desirable it is to choose action a in state s following policy π. This value q is also calculated using the discounted return as shown in formula 2.4

$$q_\pi(s, a) = \mathbb{E} \left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} \mid s_t = s, a_t = a \right] \tag{2.4}$$

Both the state value function and the action value function are fundamental aspects of RL, because of their ability to provide a quantitative measure of expected cumulative future rewards that are attainable by an agent in specific state and state-action pairs.

2.1.4 RL Algorithms and their Applications

- **Dynamic Programming (DP):** DP can be utilized to solve complex problems by breaking down the problem into simpler sub-problems. Those sub-problems then are solved and their solutions are combined to solve the overlying problem. Problems, to which this approach can be applied, must fulfill the Markov Property and we need to have full knowledge of the environment, which means we must know all the states that can be reached in a game. All of the algorithms in DP share the goal of converging to an optimal policy, which maximizes the expected reward in any given state of the underlying MDP. There are two main approaches commonly used in dynamic programming, iterative policy evaluation [25] and value iteration [5]. The iterative policy evaluation algorithm typically consists of two parts, which are repeated iteratively until the policy converges to an optimum: policy evaluation and policy improvement. Starting with an arbitrary policy, this policy is evaluated and then improved in one iteration of the algorithm. The value iteration algorithm on the other hand starts with an arbitrary value function and iteratively improves that. Since their invention, there have been a lot of adaptations to those algorithms. In [51], approximate modified policy iteration has been applied to the game of *Tetris*. Offline policy evaluation has been used in educational games to improve student engagement in the classroom [39] and policy iteration algorithms have been used to compute value functions and equilibrium strategies in model-based and model-free settings [20]. A cooperative policy iteration algorithm was used in solving decision problems in graphical games [70]. Value iteration algorithms also have been applied to various decision-making processes. In [15], value iteration has been applied to agents, learning to use devices in the electrical power system control. Furthermore, a data-driven value iteration pitch controller for wind turbines was introduced in [38]. A stable value iteration algorithm was applied to a two-player zero-sum game in [57] and the algorithm was utilized to find solutions for the scheduling problem for multiclass queuing networks [8].

- **MC Method:** In contrast to the DP algorithms, to apply the MC Method it is not necessary to have complete knowledge about the environment. Instead, this method uses random sampling of states, actions, and rewards to gain experience about the environment and it can learn optimal policies

based on that collected knowledge. To apply this method, a model of the underlying RL problem is necessary and it is important to note, that this method can only learn from complete episodes, meaning that it requires the agent to interact with the environment until the episode terminates, before it can update the estimated rewards of the value functions or policies. The MC method has been applied to many fields, for example, to study the rates of chemical reactions on surfaces [3], or in medical physics, where it was used to determine the efficiencies of gamma-ray detectors [47]. In [10], a framework for MC planning is introduced, *MCPlan*, which then was applied to the real-time strategy game *Capture the Flag* and tested in different settings. There is also a MC bayesian approach, introduced in [66], which was applied to weighted voting and bin-packing games. There are other applications of the method as well, for example in the field of material sciences [58, 65] and Economics and Finance [28].

- **Temporal Difference Learning (TDL):** One main difference between the MC method and TDL is that TDL does not need to complete an episode to update the value estimates of a state. Instead, in TDL, the value estimate for a state is updated only based on one sample transition from one state to the following state. This has the advantage, that TDL can be applied to RL problems with infinite horizons, meaning problems, where episodes do not naturally terminate or take too long to terminate. It can also be more efficient compared to the MC method, especially when the environment has long episodes. There is a long list of games to which TDL has been applied. There is for example *Connect6* [73], the game *Shogi* [4] and *Backgammon* [64], in all of which agents utilizing TDL were successful in playing against their opponents. In the mobile game *2048*, different adaptations of the TDL approach in combination with artificial neural networks were able to achieve good results [61]. The TDL approach was also applied to real-world problems. In [29] for example, TDL was used in combination with approximate dynamic programming to create an efficient smart home energy management system.

These algorithms all have different strengths and weaknesses. In the shot direction selection process in tennis, an episode consists of all the shots in a rally until it terminates. Therefore, we only can reward actions after a rally terminates, otherwise, we do not know who won the point. Also, we do not have complete knowledge of the environment, because there are just too many

possible shot sequences in tennis rallies. This is why we decided to apply the MCTS algorithm and not one of the other RL algorithms to the shot direction selection problem in tennis.

2.2 Basic Principles of MCTS

The MCTS algorithm uses a decision tree to simulate and explore possible outcomes of actions by iteratively traversing, expanding, and updating the tree. This chapter will start by explaining game trees in general and the relevant terminology of the nodes found in game trees. We will then go on to explain the four phases typically found in the basic version of the algorithm and present different policies that can be used in these phases.

2.2.1 Game Trees

When the MCTS algorithm is applied to a game, and there have been a lot of games that the algorithm has been applied to, then the decision tree representing the game is called a game tree. The game tree consists of nodes and edges. The nodes represent game states, whereas the edges represent the transitions between game states. In a game of *Tic Tac Toe* for example, a game state would be the current state of the 9x9 board with a representation of the crosses and circles that have been played up until that point. An action in this game can be placing a cross or a circle into an open field, which then would change the game state. In figure 2.2 part of an example game tree is displayed.

When starting a new turn-based game with the two players A and B and player A having the first move, then player A would have three possible actions to choose from: *a1*, *a2* and *a3*. By choosing an action, *a1* for example, the game would transition from game state *S0* to *S1*. In this example, player B would then have the choice between the actions *a1* and *a3*. The game ends as soon as a terminal game state T is reached. A terminal game state is a state, from which there are no actions to choose from, meaning one of the players has won the game or there is a draw. If for example, player B would choose action *a3* from the current game state *S1*, then the terminal game state *T4* would be reached and the game would terminate. An episode in this context consists of all the state-action pairs from the start to the end of the game. How the MCTS Algorithm can utilize game trees like this to optimize the decision process in

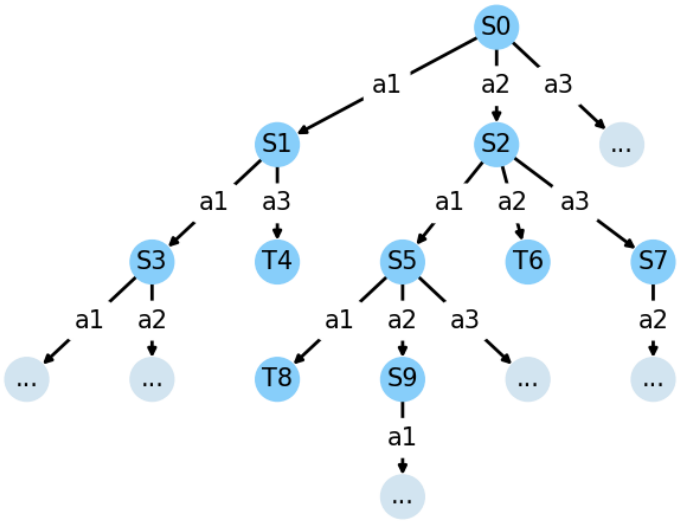

Figure 2.2: Example Game Tree

the game will be the focus of the following chapters. Therefore, it is important to introduce the following terms:

- **Root Node:** A root node is the node in the game tree that represents the game state from which the next action has to be played. When player A started the game by choosing between actions *a1*, *a2* and *a3*, the root node was the node with the game state *S0*. Assuming player A chooses action *a1*, for the turn of player B the root node would change to the node with game state *S1* because that is the node from which player B would have to choose an action.

- **Leaf Node:** A leaf node is a node in the tree representing a game state from which not all possible actions have been played yet. Assuming that in each game state in the example tree in figure 2.2 there are three possible actions *a1*, *a2* and *a3*, then the node which represents game state *S1* would be one of the leaf nodes because action *a2* has never been played.

- **Expansion Node:** An expansion node represents a previously unvisited game state that can be reached from a leaf node and is important during the expansion phase of the MCTS algorithm, as explained in the next chapter.

- **Terminal Node:** A terminal node represents a terminal game state, from which no action can be played. Meaning, the game is over and a player won or there is a draw. Examples are the nodes with the game states *T4*, *T6*, and *T8* in the example game tree in figure 2.2.

2.2.2 Four Phases of the Algorithm

The MCTS algorithm typically consists of four phases, namely the Selection Phase, Expansion Phase, Simulation Phase, and the Backpropagation Phase. Each Phase alters the game tree, or traverses through the tree to find specific nodes. The phases are shown in 2.3. Nodes with a R represent the root nodes of the tree, L stands for the leaf node, nodes with an E in them are expansion nodes and the letter T represents terminal nodes.

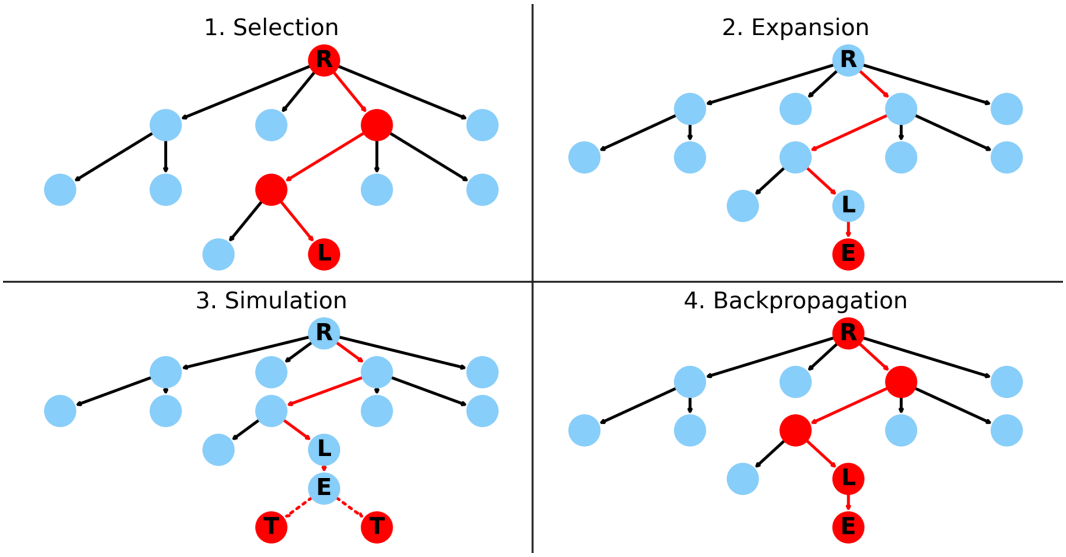

Figure 2.3: Four Phases of the MCTS Algorithm

- **Selection Phase:** The selection phase is the first phase of the algorithm. During this phase, starting from the root node the algorithm traverses through the already existing tree until a leaf node is found. It chooses the next child node based on the selection policy. There are different approaches to the selection policy, which are introduced in the next chapter.
- **Expansion Phase:** Starting from the leaf node, one or more previously unexplored nodes are added to the game tree. These new nodes are

child nodes of the leaf node, that have not been visited yet. If the leaf node is already terminal, the algorithm usually jumps directly to the backpropagation phase, because a terminal node can not have any child nodes.

- **Simulation Phase:** When there is at least one expansion node, the algorithm simulates several episodes from the expansion node, either until a terminal node is reached, or until the computational budget is used up. Usually during the simulation, the actions are picked randomly from the possible legal moves in each state. There also exist adaptations to the algorithm that use semi-random simulation policies, which pick actions that have done well in previous simulations [56]. When the actions of the opponent during the simulation are not picked at random, but rather based on the actual behavior of the opponent, it is often referred to as guided or informed simulation.
- **Backpropagation Phase:** For each simulation of a game during the simulation phase the tree is backpropagated through and the statistics of the visited nodes during the simulation are updated. The statistics can include the number of visits and the estimated reward value of that node. These statistics can then be used in the decision strategy of the algorithm, which decides which action to pick for the actual game. This is the output of one iteration of the algorithm.

2.2.3 Selection Policies

To pick the next action during the selection phase the algorithm follows a selection policy. This policy determines how to traverse through the game tree starting from the root node until a leaf node is found. There are different approaches to this. One important aspect of those approaches is the trade-off between exploration and exploitation. The exploration describes the ability of the algorithm to discover unvisited or only rarely visited states of the game. This can help to find potentially better actions and is especially important in games where the game states offer a large number of legal actions. It also helps to counter the uncertainty of the statistics that are gathered during the simulation and backpropagation phases of the algorithm. In games with dynamically changing environments, exploration also can help to adapt quickly to those changes. The counterpart to this is called exploitation. Exploitation uses

the information that already has been gathered by the algorithm to favor the actions that have proven to be successful in the game. Exploiting good actions is essential for the algorithm to converge to the optimal paths in the game tree. When favoring exploration over exploitation there is a risk of converging only slowly to optimal solutions and wasting computational resources in the process. On the other hand, favoring exploitation over exploration can get the algorithm stuck in local optima by converging too quickly without exploring all the possible actions and it can lead to a loss in adaptability to changing environments. To handle this trade-off, different approaches exist.

- **Greedy:** For the *greedy* selection policy different adaptations exist. The formula 2.5 is the approach called *constant-epsilon greedy*. This approach picks the next action uniformly at random with a fixed probability *epsilon*. With probability *epsilon-1* the action with the highest expected return *Q(a)* is chosen to transition to the next node.

$$a^* = \begin{cases} \textit{Randomly choose } a \in A(s), & \textit{with probability } \epsilon \\ \arg\max_{a \in A(s)} Q(a), & \textit{with probability } 1 - \epsilon \end{cases} \tag{2.5}$$

  An adaptation of this approach is called *Decaying-Epsilon Greedy*. This is done by decreasing the *epsilon* value over time, which leads to a stronger exploration in the early stages of a game and to a better exploitation of good actions towards the end of the learning process. When setting the *epsilon* value to 0, this policy is called *Greedy* selection. In every node, the next action with the best expected return is picked. With this approach, good actions are exploited while ignoring the exploration of the game tree completely.

- **Random:** Using selection policy *Random* means that in every node the next action a^* is picked uniformly at random from all possible actions from that node *A(s)*, as shown in formula 2.6.

$$a^* = \textit{Randomly choose } a \in A(s) \tag{2.6}$$

- **UCT:** To move through the tree with the selection policy *UCT*, in every node the action with the highest *UCT* value is picked. For each node the *UCT* value is calculated as shown in formula 2.7.

$$a* = argmax_{a \in A(s)} \left(Q(s,a) + C \cdot \sqrt{\frac{\ln(N(s))}{N(s,a)}} \right) \tag{2.7}$$

N(s) denotes the number of visits of the game state s. The *N(s,a)* value represents the number of times in which action a has been picked in state s. The C-value is an application-dependent constant, which balances the exploitation and exploration of the game tree, which is considered one of the main aspects of the MCTS algorithm [48]. A higher C-value translates to a better exploration of all the existing game states, whereas a lower C-value exploits the game states that have shown good results already. Finally *Q(s,a)* also is application dependent and represents the expected return yielded by picking action a in the game state s. The *UCT* approach was first introduced in 2006 by Kocsis and Szepesvári [30].

- **Softmax:** The *Softmax* selection policy assigns a probability P to each possible action a available in the current state s. The probability is dependent on the expected return Q of each action as shown in 2.8.

$$P(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{b \in A(s)} e^{Q(s,b)/\tau}} \tag{2.8}$$

In this approach, τ is a constant that controls the trade-off between exploration and exploitation. A higher value for τ results in better exploration of the game tree, whereas a lower τ will favor the exploitation of good actions. After assigning the probabilities to all the possible actions in the current node, the next node is reached by picking the next action based on their probabilities.

To apply the MCTS algorithm to any kind of problem, a selection policy has to be chosen. This choice is dependent on the characteristics of the problem, the available information as well as the desired trade-off between exploration and exploitation. In this thesis, the algorithm is applied to the sport of tennis, which in this form has not been done before. Therefore, three different selection policies will be applied, namely *Random*, *Greedy*, and *UCT*. The algorithm's performance in winning points will be evaluated across the three presented policies to identify the most effective one for this particular problem.

2.2.4 Decision Policies

When the computational budget is used up or another termination criterion is met, the MCTS algorithm stops to run through the four phases. At this point, a final decision has to be made: from the root node, the next action for the actual game must be chosen. To make a decision the algorithm can use different decision policies.

- **Max Child/Greedy:** With this approach, the action that leads to the node with the highest expected reward is picked.
- **Robust Child:** The action that leads to the child node with the highest visit count is picked.
- **Robust-max child:** When there is a child, that has both the highest visit count as well as the highest expected reward value, that action is picked. When such a child node does not exist, the simulations continue until a child like that is found [7].

There are more possible approaches to select the final action and there is not a lot of research on the effectiveness of these decision policies. One study investigates different policies in a strategic card game *Lords of War*, to create entertaining opponents. As a side effect, they found significant differences in the playing strength of MCTS agents, using different decision policies [53]. In this thesis, two different decision policies will be evaluated. Firstly we use the *Greedy* decision policy, always choosing the action with the highest expected return. Secondly, we use *UCT* again, but this time as a decision policy, to see how it will affect the learning process of the MCTS agent when the final decision is still able to explore the possible actions in the game.

2.3 Applications of the MCTS algorithm

MCTS is a powerful heuristic search algorithm, which can be applied to a variety of decision-making processes. It has been successfully used in different turn-based board games, such as *Carcassone* [18] and *Settlers of Catan* [60]. The algorithm got a lot of attention when it was used in combination with deep neural networks in the game of *Go*, defeating various professional *Go* players [55]. MCTS also was applied to various computer games, for example, to train agents to play *Super Mario Bros* [27] and to optimize the decision

process of the hero selection in multiplayer online battle arena games [9], or in the real-time strategy game *Wargus*, where it was used to plan tactical assaults on the opponent's units [2]. MCTS was also applied to the mobile game *2048*, where it was able to effectively play the game despite the game's high uncertainty factors [62] and it was also applied to *Hearthstone: Heroes of Warcraft* to examine the effect of supervised prediction models on an MCTS AI [74]. Most turn-based board and computer games share a crucial commonality. When a player takes an action in the current game state, that action results in a singular, specific new state of the game. Take chess for instance, when moving a piece from its current tile to another there is no possibility that the piece will land on a different tile, than the chosen one. This differs from games that incorporate stochastic components. Consider the game *Battleship*, where calling out coordinates to locate an opponent's ship can lead to different subsequent game states. If you hit your opponent's ship you earn another turn. If you miss, however, it becomes your opponent's turn to call out coordinates. Real-life decision-making processes often come with a high level of unpredictability due to their stochastic elements. The MCTS algorithm still can be applied successfully to those kinds of problems. It proved to be a viable method to solve the capacitated vehicle routing problem with traffic jams and risk-aware project scheduling problem [40], as well as the airline crew pairing problem [34] and the dynamic flexible job shop scheduling problem [33]. Moreover, the MCTS algorithm produced acceptable solutions when employed in addressing a real-world structural engineering design problem [49] and showed promising results when applied to solve Planning Problems in Transportation Domains [68]. In the field of sport the algorithm was successfully used to find and compare strategies in the Olympic sport of curling [44]. It was also used in combination with artificial neural networks to optimize decision processes before and during Formula-E races [37] and even to generate plays in American football [31]. When we look into the application of the MCTS algorithm to the sport of tennis, there is only a limited number of studies related to our work. Firstly, video broadcasting data from tennis matches was used to analyze tactical information in tennis using MC simulation to find optimal strategies [63]. Even with the use of an unrelated data set to ours and a different algorithmic approach the primary aim of this study closely aligns with one of the research questions discussed in this thesis. It demonstrates that employing AI algorithms can yield valuable insights into tennis strategies. In the second study, the MCTS algorithm was implemented in a 3D tennis video game. A tennis agent utilizing the MCTS

approach was developed using a data set of pre-recorded tennis matches. The primary objective of this study was to generate a behavior for the agent that is both believable and skillful. The results show that this goal can be accomplished using the MCTS approach.
When we look into studies that compare different action selection policies of MCTS, several relevant research items can be found. MCTS was applied to the game *Tron* with a variety of different selection policies, both deterministic and stochastic. An empirical comparison was conducted. Results show that deterministic policies outperform stochastic policies, the most successful policy being the UCBI-tuned, followed by UCB1 [46]. An important difference between this study to our current work is that in *Tron*, player moves can occur simultaneously whereas in tennis players take turns to hit the ball. Another study offered two methods to improve the UCT action selection policy for deterministic games by using a moving average return function and a sufficiency threshold [21]. In the created match simulator *Match Point AI* in this thesis, tennis was modeled as a non-deterministic game, meaning there are probabilities for errors and winners involved in each shot. That is not the case in deterministic games, which distinguishes this thesis from the presented paper. Another research paper compared two different action selection policies in general game playing, namely UCT and UCT-tuned. They also examined how the number of simulations during the simulation phase influences the performance of the algorithm with the two different policies. Results show that with a lower number of simulations UCT-tuned outperforms UCT. But with a higher number of simulations both selection policies perform similar [17]. While the comparison of different parameter settings of the MCTS algorithm is also relevant to our work, the presented paper compared the different adaptations when applied to board games which differ from our tennis environment in many ways, such as its complexity and the uncertainty factors. The current project differentiates from the above-mentioned studies in other ways as well. We use shot-by-shot tennis data from the match charting project introduced by Jeff Sackmann [50]. The match charting project is a crowd-sourced effort aimed at enhancing both the quantity and quality of professional tennis data. The dataset is used widely in literature, for example, to predict tennis match outcomes by using machine learning techniques [32], to generate win probabilities during ongoing matches [19] or to analyze the distribution of rally lengths [35]. In addition to using a different data set compared to the related work, our study aims to evaluate specific shot patterns against modeled real-world tennis players and also to

compare different algorithmic designs of MCTS in its application to *Match Point AI*. To the best of our knowledge, there have been no studies yet that use the MCTS algorithm in combination with shot-by-shot real-world tennis data to gain strategic insights into tennis as well as evaluate different adaptations of the MCTS algorithm while doing so.

2.4 Tennis - Rules and Terminology

For a better understanding of the next chapters of this thesis, a short introduction to tennis is necessary. A rally in tennis is the shot sequence starting from the serve of one player until the point is over, resulting in one player gaining a point. Rallies can consist of one shot only, if the serving player hits a first-serve winner, commonly referred to as an ace, or it can be 20 or 30 shots long, depending on how many shots it takes one of the players to hit a winner or make an error. Scoring points can be accomplished by either errors committed by the opponent or the execution of winning shots by the player. Points resulting from the opponent's failure to make a successful and deliberate shot are categorized as errors. An error occurs, if the ball lands outside of the court or when it lands in the net. Conversely, points secured by a player through skillful and strategically placed shots that go beyond the opponent's reach and result in a point are considered winners. Therefore, the dynamics of tennis scoring involve a balance between capitalizing on the opponent's mistakes and actively creating opportunities for successful shots to win points. When the first serve of one of the players is an error, it is called a fault and he gets a second chance and can serve a second serve. If the second serve is also an error, it is referred to as a double fault and grants the opponent the point. Therefore, usually, the first serve is hit fast and with high risk, whereas the second serve is played more conservatively with lower risk. It is also necessary to note that players alternate in serving from two different sides of the court, depending on the score of the match. Each game starts with a serve from the right side of the court, called the deuce side, and the ball has to land in the opponent's left serving box. The next point then is started with a serve from the left side of the court, referred to as the advantage side of the court and the serve has to land in the opponent's right service box. The service boxes are designated areas of the court in which the serves have to bounce. The first shot after a successful serve is called a return. A detailed explanation of the tennis scoring system is not relevant to this thesis. It is however important to know

that the system allows a player to win a match, even though he might not have won more points than his opponent throughout the match. This phenomenon is called the Simpsons Paradox. In 2013 61.000 professional tennis matches were analysed regarding this dynamic. Results showed that 4.5% of those matches, which took place over the course of 21 years, were won by the player with fewer total points won [6]. This study later was followed up on and confirmed in 2019, where results show that the paradox occurs in 4.5% in men's professional tennis [36]. In a study aiming to predict tennis match outcomes through machine learning algorithms, findings indicated that the predictor variable "total points won" was the most dominant feature in linear models for predicting match outcomes [32]. Based on these studies we assume that winning points leads to winning matches despite the Simpsons Paradox. So we developed the MCTS algorithm with the goal of winning points instead of matches.

3 MCTS in the Tennis Simulator - Match Point AI

This Chapter presents the Tennis Match Simulation Environment, *Match Point AI*, which was created to address the research questions and hypotheses. We will illustrate how the MCTS algorithm is applied to this environment, showcasing the creation of game trees in the simulator and demonstrating how the algorithm competes against the opponents. The chapter continues with the explanation of the opponents, and the fixed strategy bots, detailing the data used to create them and showing how the bots can react to different situations in the tennis simulator. We then present the settings for the experiments conducted in *Match Point AI*, followed by showing our methods to statistically analyze the results of those experiments. This chapter concludes by crediting the software used in this thesis.

3.1 Match Point AI

In a real-world tennis match, many factors can influence the course of the match as well as the outcome of individual points. There can be external factors such as rain delays or the support by fans. On the court, there can be foot faults when serving, time violations, and medical timeouts. The outcome of a rally is of course heavily influenced by the shots in that rally, particularly the ball velocity, the player's movements and positions on the court, and the shot placement. All of these factors and many more can have an impact on the outcome and the course of rallies and matches. To be able to apply the MCTS algorithm to the sport of tennis we model tennis in a simulation environment called *Match Point AI*, which considers only the most important aspects of the sport because it is nearly impossible and for our purpose also not necessary to consider every influential aspect there is in real-world tennis. To start with, in *Match Point AI*, the players follow the typical tennis match rules. The scoring

is done using the traditional scoring system in tennis [69]. They switch serve after each game, they play tiebreaks at a score of 6:6 in a set, they serve to the correct serving box according to the score in a game and if the first serve is a fault, they get a second serve. Professional tennis players can play the ball very precisely in the direction they want. In *Match Point AI* the directions are broken down to the left or right 30% of the court and the middle 40% of the court, resulting in three possible directions for each shot. The service boxes are split into three directions as well. Players in *Match Point AI* can either serve wide, through the middle, or play a body serve directly onto the opponent's body. The direction encoding and serve direction encoding are shown in figure 3.1.

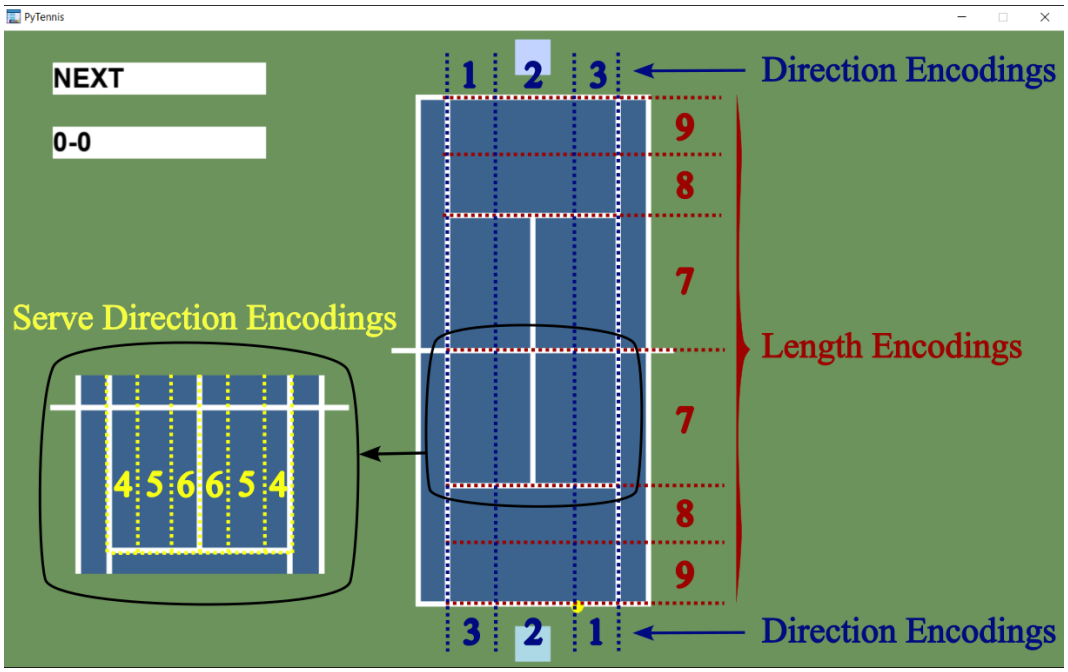

Figure 3.1: Tennis Shot Encoding

This encoding is also used by the match charting project [50] and therefore had to be used in the simulator as well. Otherwise, our player's behavior could not have been based on the match charting project dataset. When it is one of the player's turns to choose an action in an ongoing rally, he can choose to play the ball in the three different directions, encoded as *1*, *2*, and *3*. If it is a player's turn to serve, he can choose between the action encodings *4*, *5*, and *6*. The shot length encoding is used for visualization purposes. It is based on the match charting project dataset as well. Still, it does not influence

the winner and error probabilities of individual shots and the depth can not actively be chosen by the players in *Match Point AI*. Tennis is modelled as a non-deterministic game. When a player chooses a direction to play the ball in there is always the possibility that the shot will be an error or a winner. The probabilities for an error or a winner are different depending on the current game state. A game state in *Match Point AI* includes the direction of the opponent's previous shot, it considers which player was opening the rally with a serve and whether that serve was a first or a second serve. To summarize, each shot in *Match Point AI* consists of an active choice of direction of one of the players and a stochastic component, whose probabilities are dependent on the current game state. The probability of the return being an error is for example higher when the previous serve was a first serve instead of a second serve because the second serve is usually easier to return than a first serve. There are certain important aspects in real-world tennis, that could not be taken into account in *Match Point AI*. In a real-world tennis match, the ball's velocity and the position and movement of the opponent on the court impact the player's decision to play the next ball in a specific direction. These aspects are not included in the dataset and therefore were not modeled in *Match Point AI*. We also created the simulator under the assumption, that factors like rain delays, foot faults, and medical timeouts do not have a large enough impact on outcomes of rallies and matches for us to consider them in *Match Point AI*.

3.2 MCTS agents in Match Point AI

This chapter illustrates how an agent based on the MCTS algorithm can play tennis matches in *Match Point AI*. The MCTS algorithm runs through four phases as introduced in subsection 2.2.2, utilizing a game tree. An illustration of a game tree during one simulated match in *Match Point AI* is presented in figure 3.2.

A rally always starts at the red node at the top of the tree. The players take turns to play a shot in a rally, represented by the blue and green nodes. Each node represents one shot in a rally. The shot encoding is explained in figure 3.1. In this tree however, we have more than just the encoding of the depth and direction of a shot, but also whether or not the shot was a winner, which is encoded as a *, or an error, encoded by *e*. After each point in a match, meaning as soon as a rally terminates, we start over at the red node at the top of the

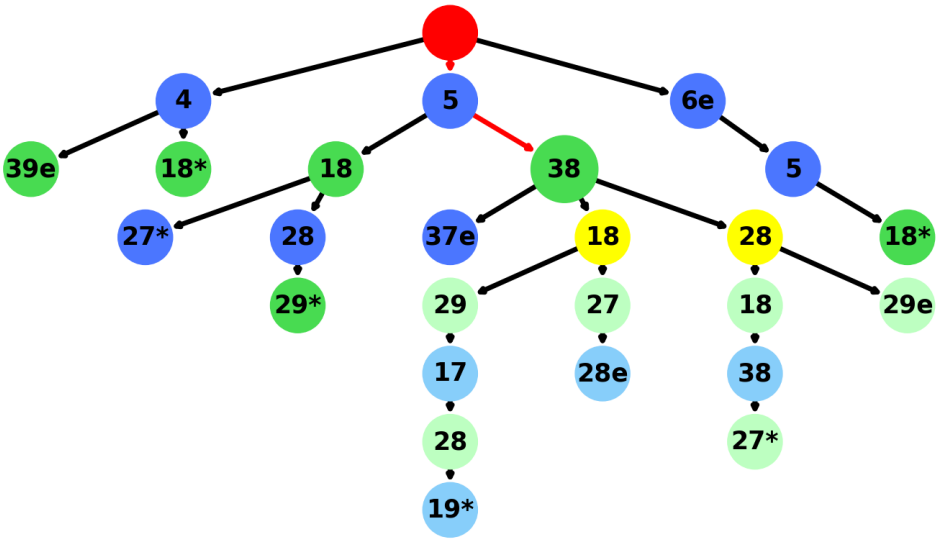

Figure 3.2: MCTS Game Tree in the Simulation Environment

tree. In this particular tree, we have already terminated some rallies. When we traverse the tree by always choosing the left node for example, the rally consists of two shots, a first serve in direction *4* by the blue player followed by a return error by the green player, *39e*. If we go through the tree by always choosing the right path, the rally consists of three shots, a first serve fault by the blue player (*6e*) followed by a successful second serve (*5*) and finally a winner, played by the green player (*18**). With this tree design, we can record every rally in a match, or even in several matches, in a single tree, which then can be used by the MCTS algorithm.

Suppose we are in an ongoing rally in a match simulation. The rally up to this point has been a successful first serve by the blue player with direction encoding *5* and the green player made a successful return, encoded as *38*. This rally corresponds to the red path leading to the enlarged green node in figure 3.2 with the encoding *38*. From this node, it is the turn of the MCTS agent to choose a direction to play the ball in. At this point, the first phase of the algorithm begins. During the selection phase, the algorithm starts at the root node, which in this case is the enlarged green node with encoding *38*, and traverses through the tree using the selection policy until a leaf node is found. In this case, the root node is the same as the leaf node because not all directions have been played from this node yet, but instead, only direction *3* has been played so far.

This means that in this situation the algorithm would jump to the second step, the expansion phase. When the root node is not the same as the leaf node the algorithm traverses through the tree until it finds a leaf node, using its selection policy. During the expansion phase, the algorithm adds all unexplored shot directions to the tree. This is showcased in the example tree by the yellow nodes. From these yellow nodes the third phase, the simulation phase, begins. In our example tree, we only run two simulations per expanded node, as shown by the lighter-colored nodes. Two simulations per expanded node is a very small number and was chosen only to keep the figure from getting too large. During the simulation phase, the MCTS agent and the bot take turns adding shots to the simulated rally until a terminal node is reached. Our MCTS agents play their shots by randomly choosing a direction to play the ball to. For the fixed-strategy bot, an informed simulation phase is performed. This means that we know everything about the opponent's behavior. For the simulation phase, we do not have to randomly add an opponent's shot, but we get the next shot from the actual bot with the actual bot's behavior and add it to the simulated rally. The informed simulation has the advantage that the MCTS algorithm learns the opponent's behavior directly during the simulation phase and can then make an informed decision based on that knowledge. For every time that a simulated rally reaches a terminal node the algorithm performs the fourth phase, the backpropagation phase. It is also possible that the expanded node is terminal. For that case, no simulations are run from the expanded node, because the rally already is terminated. Here, the algorithm would skip the simulation phase and jump directly to the backpropagation phase. During this phase, the visit counts and win counts of the relevant nodes are updated and the new UCT values are calculated. When running 15 simulations for an expanded node for example and 10 of those rallies are won by the MCTS agent, the visit count of the corresponding expansion node would be set to 15 and the win count to 10. Finally, when the predefined number of simulations for all the expansion nodes is reached the algorithm makes its final decision using the predefined decision policy. The MCTS algorithm applied to *Match Point AI* in this form can be seen in the following pseudo-code. Lastly, the MCTS agents have the same probability of making errors or winners in a rally, as the Average Bot. The origin of these probabilities will be explained in the next chapter.

Algorithm 1 Monte Carlo Tree Search (MCTS)

```
1: Input: root, budget
2: Output: next action (direction to play the next ball in)
3:
4: function MCTS(root, budget)
5:     for i ← 1 to budget do
6:         node ← root                                          ▷ Selection Phase
7:         while node is fully expanded and node is not terminal do
8:             node ← SelectionPolicy(node)
9:         end while
10:        leaf ← node                                          ▷ Expansion Phase
11:        if leaf is terminal then Backpropagation(leaf)
12:        else
13:            unexplored ← all actions not yet played from leaf
14:            for all action in unexplored do
15:                Create expansion for action
16:            end for
17:            for all node in expansions do                    ▷ Simulation Phase
18:                result ← Simulate(node)
19:                while node is not root do                    ▷ Backpropagation Phase
20:                    Update node statistics according to result
21:                    node ← parent of node
22:                end while
23:            end for
24:        end if
25:    end for
26:    return DecisionPolicy(root)
27: end function
28:
29: function Simulate(node)
30:    while node is not terminal do
31:        if MCTS Agents turn is True then
32:            Randomly choose action from available actions
33:        else Get action from Opponent                        ▷ Informed Simulation
34:        end if
35:        Create new node by applying action
36:    end while
37:    return result of simulation from terminal state
38: end function
```

3.3 Data-Driven Bot Strategies

The MCTS algorithm needs an opponent against which it can learn to play the game. Therefore, we implemented two different bots, which follow different fixed strategies in tennis. To create these strategies, real-world tennis data from the match charting project [50] was analyzed. Between the second of January 2017 and the 23rd of November 2023, the match charting community collected shot-by-shot tennis data for 295.354 rallies of singles matches in professional men's tennis. This data includes 30 different information points about each rally tracked in the dataset, including for example the tournament at which the match was played, the two opposing players, the scores, and each shot in each rally. This is done using a tennis shot encoding, where for each attribute of a shot a number, letter, or character is assigned. A detailed description of every possible shot encoding can be found in the match charting project. Based on this data two different bot strategies were created. This was done by looking at different situations in a match to make it as realistic as possible with the given data. Therefore, we first differentiate between three main shot types: the serve, the return, and all the other shots.

- **The Serve:** The serve can be either from the advantage side or the deuce side of the court. The players alternate between serving from the two sides after each point. This difference is taken into consideration because the serve through the middle might be more difficult for some players when serving from the deuce side compared to the serve through the middle from the advantage side. It is also possible that the player either has to hit a first serve or a second serve. The second serve often is played with a lower risk compared to the first serve, which has an effect on the placement as well as on the error and winner probabilities of the shot. For each of those scenarios, the players can choose in which direction to serve, to the left, through the middle, or to the right side of the service box. The dataset was analyzed to see with which probability a player would play in which direction for each of those scenarios, meaning there are 12 possible scenarios (2 sides to serve from * 2 different serve types * 3 different directions) for which the probabilities were extracted. When the bot has to serve in the *Match Point AI* it uses these probabilities to choose the direction of the serve. For each of those 12 scenarios, we then extracted the error and winner probabilities. Because each shot can

be a winner, an error, or neither, a total of 36 different probabilities are included in each bot for their behavior when serving.

- **The Return:** The direction of the return and the corresponding error and winner probabilities are dependent on the previous serve. That means the probabilities are different for the 12 serve scenarios described above. For each of those 12 scenarios, a return can be hit into the 3 directions of the court, to the left, through the middle, and to the right. So there is a total of 36 (3 directions * 12 serve scenarios) different shots for the return. Each of those shots again has the potential to be an error or a winner, which means a total of 108 different probabilities were extracted from the dataset to create the return behavior for each bot.
- **Normal Shot:** For the other shots in a rally after the return a new set of probabilities was extracted from the dataset. Firstly, the next shot in a rally is influenced by who was serving in that rally. When a player is serving, that player has the first chance to dictate the point and be offensive, whereas the returning player usually is the more defensive player, because he has to start the rally by reacting to the opponent's serve. This dynamic does not hold however for second serves. Rallies after a second serve usually are more balanced, with both players having chances to dictate the rally and be aggressive. Another factor influencing a shot in a rally is the direction of the previous shot. So we again have 12 different situations in a rally (2 possible serving players * 2 serve types * 3 directions of the previous shot), which are influencing the direction of the next shot and the winner and error probabilities. From each of the 12 described situations, the shot can be played in three different directions. When we include the error and winner probabilities for each of those 36 shots, again 108 different probabilities were extracted from the dataset and used to create each bot's shot behavior during a rally after the return.

3.3.1 Novak Djokovic Bot

Novak Djokovic undoubtedly is one of the greatest tennis players of all time. Currently ranked Number 1 in the world, he holds many records, such as the most grand slam wins and most weeks as the world's number one ranked player. To create a tennis bot that is based on Djokovic's behavior on the tennis court and then letting MCTS agents learn to win points against him

poses an interesting challenge. In the specified time frame, a total of 19.325 rallies were tracked by the match charting community, in which Djokovic was playing. From these rallies, a total of 252 different direction, error, and winner probabilities for serves, returns, and normal shots were extracted as described above, and based on these probabilities the Djokovic Bot's behavior in *Match Point AI* was created.

3.3.2 Average Bot

To create the behavior of the Average Bot the same 252 probabilities for shot direction, errors, and winners were extracted from the dataset. This time all of the 295.354 rallies, tracked in the specified time frame, were used for the data analysis. In those rallies, a total of 420 professional men's tennis players participated. Figure 3.3 shows the composition of the Average Bot with a focus on the 10 players, which had the biggest impact on the Average Bot's behavior. Rafael Nadal for example was part of 5.4% of all the rallies in that dataset.

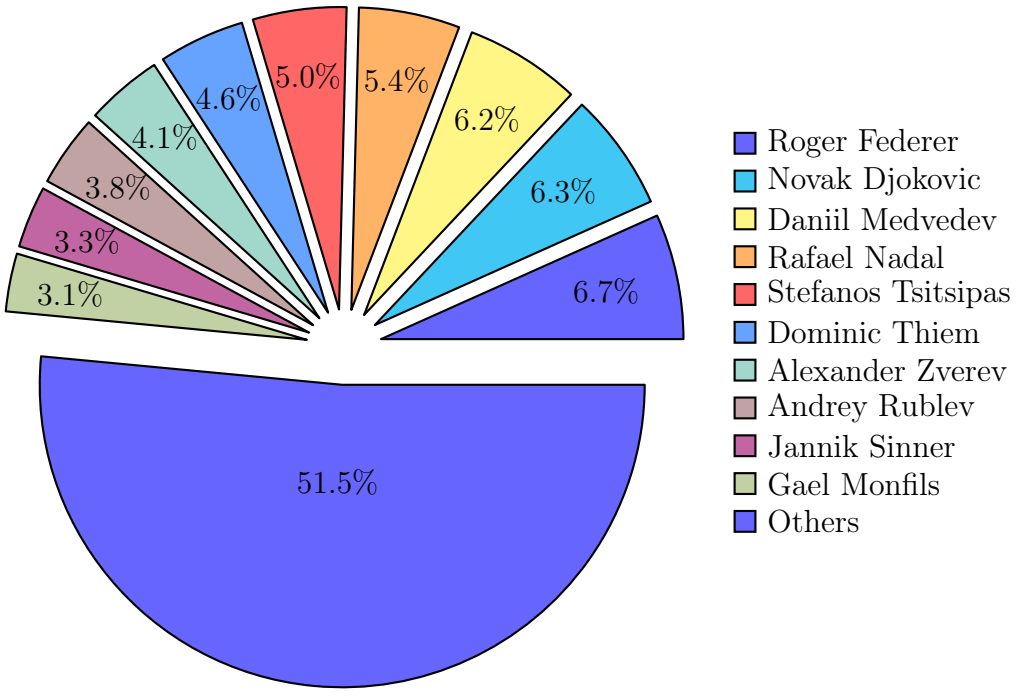

Figure 3.3: Composition of the Average Bot

3.4 Experiments

Match Point AI enables us to conduct numerous match simulations with diverse settings, offering a wide range of possibilities for exploration and analysis. For the research questions in this thesis, three sets of experiments were conducted:

- **Experiment Set 1:** To gain insight into the capability of *Match Point AI* to generate reasonable tennis data we first simulate 500 matches between the Average Bot and the Djokovic Bot. The resulting shot-by-shot data from these simulations allows for the analysis of important aspects. The Djokovic Bot should win most of the points and matches against the average bot and the average rally length in the simulated rallies should be comparable to real-world tennis rally lengths. By comparing these aspects between the simulated results and the real-world data we gain insights into whether or not *Match Point AI* is a suitable environment to retrieve insights about strategies in tennis.
- **Experiment Set 2:** To evaluate the performance of different selection and decision policies in the MCTS algorithm, we simulate matches, in which MCTS agents with different selection and decision policies play matches against the two bot strategies. Three different selection policies are used combined with two different decision policies, which means there are six different MCTS agents playing against both the Average Bot and the Djokovic Bot. In this set of simulations, there are 12 different match-ups and in each match-up, we simulate 200 matches. With the results of those experiments, we can evaluate the effectiveness of different selection and decision policies against the two bots. Specific parts of the results from this set of experiments are also utilized to perform the shot pattern analysis.
- **Experiment Set 3**: To analyze the effect different parameter settings have on the performance of the algorithm, we simulate matches with the MCTS agent using *UCT* as selection and *Greedy* as decision policy. We adapt this agent by using three different C-values as well as three different counts of simulations conducted during the simulations phase. These nine adapted agents then play against the Average Bot in simulations of 100 matches per adaptation.

For each rally in each simulation match in *Match Point AI*, we track the information shown in table 3.1.

Information	Relevance
Point Winner	Main parameter for all performance evaluations
Score	Determine serve position (Advantage or Deuce Side)
Selection Policy	Performance evaluation of different Selection Policies
Decision Policy	Performance evaluation of different Decision Policies
Opponent	Performance evaluation against specific players
Parameter settings	Performance evaluation of different Parameter settings
Rally length	Suitability of *Match Point AI*
Serving player	Shot pattern analysis
Shot directions	Shot pattern analysis

Table 3.1: Information tracked for each Rally during the Experiments

3.5 Statistical Analysis of the Results

The research questions in this thesis are answered by analyzing the results generated in the experiments. In addition to descriptive statistics, we conduct different kinds of tests on the resulting data to analyze it for statistically significant relationships. The main method used to verify the hypotheses is the Chi-Squared test [45] with an alpha of 0.05. This test is often used to examine if there is a significant association between two categorical variables. A pairwise comparison is conducted as a post-hoc test in cases, where the initial Chi-Squared test shows significant relationships between the categorical variables. For the post-hoc tests, a Bonferroni-Holm correction is used to account for the family-wise error rate [24]. To gain insights into the effect size of a significant relationship in the data we calculate and interpret Cramer's V [13], using formula 3.1.

$$V = \sqrt{\frac{\chi^2}{n \cdot \min(k-1, r-1)}} \tag{3.1}$$

χ^2 is the Chi-Squared value, n translates to the total number of observations, and k and r are the numbers of categories for the first and the second variable. A larger value for Cramer's V translates to a greater statistical connection between two variables. The interpretation of Cramer's V is dependent on the Degrees of Freedom (DF) of the data that is analyzed. DF are calculated as shown in formula 3.2, with the DF and the columns and rows of the contingency table, c and r.

$$\mathrm{DF} = (r-1) * (c-1) \tag{3.2}$$

Table 3.2 shows how an effect size can be interpreted based on the dataset's DF [11].

Degree of freedom	**Small**	**Medium**	**Large**
1	0.10	0.30	0.50
2	0.07	0.21	0.35
3	0.06	0.17	0.29
4	0.05	0.15	0.25
5	0.04	0.13	0.22

Table 3.2: Effect size for the Chi-Squared test, Cramer's V and its interpretation [11]

3.6 Software

To create *Match Point AI*, analyze the dataset, and implement the MCTS algorithm as well as conduct the statistical analysis of the results, the programming language Python (Version 3.11.6) [71] is used. In addition to the standard modules of that Python version, the following packages are utilized. *Match Point AI* is created using the package *PyGame* (Version 2.5.2) [54]. The data analysis is performed using the Python packages *Pandas* (Version 2.1.1) [72] and *NumPy* (Version 1.26.1) [23]. To visualize game trees during the different phases of the MCTS algorithm, the packages *NetworkX* (Version 3.2) [22] and *Matplotlib* [26] are used. Finally, the statistical analysis of the results is done using the library *statsmodels* (Version 0.14.1) [52].

4 Results and Evaluation

The upcoming chapter will present and evaluate the results of the conducted experiments. We address each research question and their corresponding hypotheses and examine our findings. The chapter begins by presenting the strengths and limitations found in *Match Point AI*. We then continue by showing the most frequent shot patterns found by one of the MCTS agents at the beginning of a rally and those pattern's success rates. Subsequently, we will analyze the influence of different selection and decision policies on the performance of the MCTS algorithm when it is applied to the decision-making problem of shot direction selection in tennis. Lastly, we will investigate the effects of adaptations made to two specific parameters in the algorithm and how these adaptations influence the algorithm's performance in winning points in *Match Point AI*.

4.1 Strengths and Limitations of Match Point AI

While conducting the experiments for this thesis and designing the different bots and agents, we observed several limitations *Match Point AI* and the Bots and Agents have. To apply the MCTS algorithm to the sport of tennis we first modeled tennis as a non-deterministic game. A model of something is never as accurate as the real thing. Here are a few of the biggest compromises that were made during the design of *Match Point AI*.

- **Ball velocity:** We do not consider the ball velocity in *Match Point AI*. This is due to the fact, that in the dataset we use, no information about ball velocity is tracked for the shots. This is a limitation because it is safe to say that the ball velocity has an impact on the error and winner probabilities of each shot. In a real-world tennis match, the professional players can choose how much risk they take in each shot by hitting the

ball slow or fast. In some situations it might be advantageous to hit a slow ball to break the opponent's rhythm for example, in other situations it might make sense to play a really fast ball, increasing both the winner and error probabilities of that shot. In *Match Point AI* the bots and agents do not have the option to choose to play a fast or a slow ball, because we do not have access to data on which this decision could be based.

- **Player Positions:** It can have a large impact on the choice of direction of the ball when you can see at what position on the court the opponent is currently at or in which direction of the court he is currently moving. It might for example be beneficial for a player to play the ball against the opponent's direction of movement. In the dataset, we do not have information about the player's position or movement on the court, which means we can not include these aspects in the shot direction selection process of the bots and agents.
- **Shot Selection:** In our shot direction selection processes, neither the bots nor the agents have the option to choose between different shot types like slice or topspin from the baseline, drop shots, lobs, attacking the net, and playing volleys, overheads or choosing between different types of serves like kick or slice serves. In terms of their choices in a simulated tennis match, our players are very limited compared to the choices a real-world tennis player has in a match. However, the players in *Match Point AI* can choose the direction in which they want to play the ball for each normal shot and every serve and return. Normal shots in this context include every forehand and backhand shot from the baseline. In figure 4.1 we show how many of the shots, tracked in the real-world dataset between 2017 and 2023, were normal shots compared to other shot types like slices, volleys, drop shots, lobs and other shots like overhead smashes or half volleys.

  In 80.77% of all the shots tracked in the dataset, the shot type was either a forehand or a backhand groundstroke. This is the reason for only including the direction selection of groundstrokes into *Match Point AI* and not the different shot types.
- **Fatigue:** In our probabilities for making an error or hitting a winner when playing a shot, we do not take the player's fatigue into account. During a rally in a real-world tennis match, one of the players might

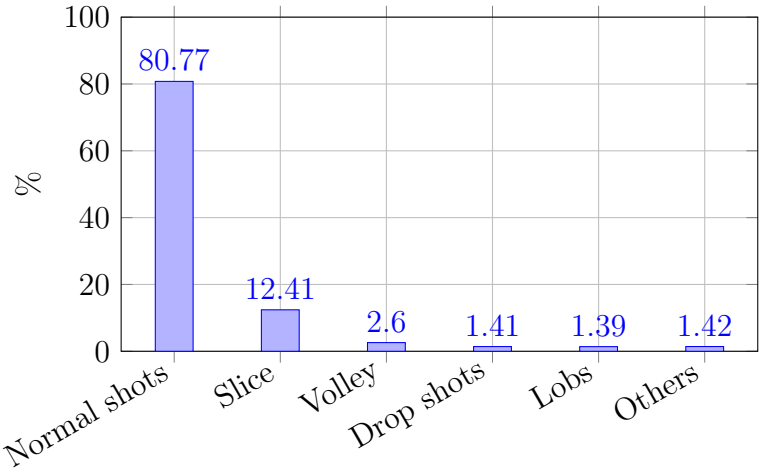

Figure 4.1: Shot Type Distribution of the Real-World Dataset

switch directions between every shot, making his opponent run, while the player plays every shot comfortably from the same position. In a real-world tennis match, this dynamic can influence the error probabilities, because the more exhausted player is more likely to make an error. This influence can not be extracted accurately enough from the dataset and therefore, we did not include this aspect in *Match Point AI*.

These compromises are the biggest limitations in *Match Point AI* and with them, a lot of the dynamics of a real-world tennis match are lost. The main reason for most of these compromises is a lack of information. By including more dynamics of a real-world tennis match, the results generated by *Match Point AI* would be more valuable and meaningful for real-world tennis. However, when modeling something as complex as the sport of tennis, making compromises is a necessity. It is not possible and in our case also not necessary to consider every aspect of the real sport of tennis, which will be shown in the following sections.

The results of the experiment set 1, in which the Average Bot competes in 500 matches against the Djokovic Bot, show a match win rate of the Djokovic Bot of 82.6% (413 wins versus 87 losses). The real-world Novak Djokovic played a total of 394 Matches on the professional men's tennis tour between 2017 and 2023, which is the time frame from which we analyzed the data set to extract the probabilities for the Djokovic Bot. Out of those 394 matches, Novak Djokovic won 336 matches which translates to an 85.3% match win rate, a difference to

our experiment of 2.7%. Even with all the compromises made to model tennis in *Match Point AI*, these results show a realistic difference in performance between the Average and the Djokovic Bot in the simulator and it speaks to the capability of *Match Point AI* to generate valuable and reasonable tennis data. The difference of 2.7% can be caused by two factors. Firstly, the Average Bots name is misleading. Almost 50% of the rallies analyzed for the Average Bots behavior originate from matches of current and former top ten players as we showed in figure 3.3. This means, that the extracted probabilities for the Average Bot represent the behavior of a player that is better than average. But with the real-world Djokovic playing as well as he does and reaching the finals of almost every tournament he enters, he actually faces a lot of the other top 10 players. So the Average Bots behaviour potentially represents the usual opponent of Novak Djokovic quite well. Secondly, the probabilities used in the Average Bot also include rallies of Novak Djokovic himself. This means, during our first set of experiments, the Djokovic Bot also plays against himself, at least to a certain degree. This could also explain the 2.7% difference in the match-win rates. Considering the characteristics of tennis, which are not modeled in *Match Point AI*, in addition to these two aspects, a difference in match win performance of 2.7% between the Djokovic Bot and the real Novak Djokovic is very small.

In table 4.1 we show the overall results of experiment set 2, in which different adaptations of the MCTS algorithm compete against both fixed-strategy bots. Each point win rate and match win rate pair corresponds to a simulation consisting of 200 matches played in *Match Point AI*. The discussion about the different performances regarding the selection and decision policies of the MCTS agents will follow in a later section, but from these results, we can already see interesting dynamics regarding our first research question.

The first observation we want to discuss is that the point win percentages seem low when compared to the match win percentages. When looking at the UCT/Greedy MCTS agents result when it is playing against the Average Bot for example, we see that the agent wins 51.99% of all the points and that corresponds to a 71% match win rate. When we look at the point- and match-win percentages of some of the men's real-world top tennis players, however, we can see that those small point-win percentage differences do make the difference in real-world tennis as well. Figure 4.2 shows the match win data [16] and point win data [67] from five professional tennis players for all matches played between 2017 and 2023. The ranking was taken on the 9th

MCTS Agents vs. Fixed Strategy Bots	**Opponent 1: Average Bot**		**Opponent 2: Djokovic Bot**	
Selection Policy/ Decision Policy	Point Win Rate [%]	Match Win Rate [%]	Point Win Rate [%]	Match Win Rate [%]
UCT/Greedy	51.99	71.00	49.65	46.50
UCT/UCT	50.10	50.50	46.48	13.50
Random/Greedy	52.14	69.50	49.03	39.00
Random/UCT	49.84	46.50	48.77	36.00
Greedy/Greedy	51.58	66.00	49.34	41.50
Greedy/UCT	49.99	47.50	45.84	15.00

Table 4.1: Win Percentages of MCTS Agents against Fixed Strategy Bots (Win Rates of MCTS Agents out of 200 Matches)

of January 2024. This shows us, that the point-win to the match-win ratio of the matches played in the match simulator between the MCTS agents and the bots is plausible. Furthermore, we observe, that the point and match win rates of the MCTS agents seem higher when they play against the Average Bot compared to the point and match win rates when playing against the Djokovic Bot.

Player	**Ranking**	**Point Wins [%]**	**Match Wins [%]**
N. Djokovic	1	54.86	85.86
D. Medvedev	3	52.71	74.25
A. Zverev	6	52.57	72.03
C. Ruud	11	50.57	64.67
J. Struff	21	49.86	49.52

Table 4.2: Real-World Player Statistics

This is to be expected because we saw in the results of the 500 match simulations in the experiment set 1, in which the two bots played against each other, that the Djokovic Bot performed better than the Average Bot. The results of a Chi-Squared test show a significant relationship between the total points won by the agents and their opponent (p=0.0000). The agents win significantly more points against the Average Bot than against the Djokovic Bot with a small effect size (V=0.0267). With this result we can reject our hypothesis H_{01}

and accept the alternative Hypothesis H_{11}, stating, that there is a significant difference in the total points won by the MCTS Agents against the two bots.

In figure 4.2 we compare the rally length distribution of matches tracked in the real-world dataset with the rally length distribution in the resulting shot-by-shot data generated by the simulated matches of the MCTS agents against the Average Bot. The blue bars represent the rally length distribution in matches tracked in the real-world dataset and the red bars show the rally length distribution of the simulated matches in *Match Point AI*.

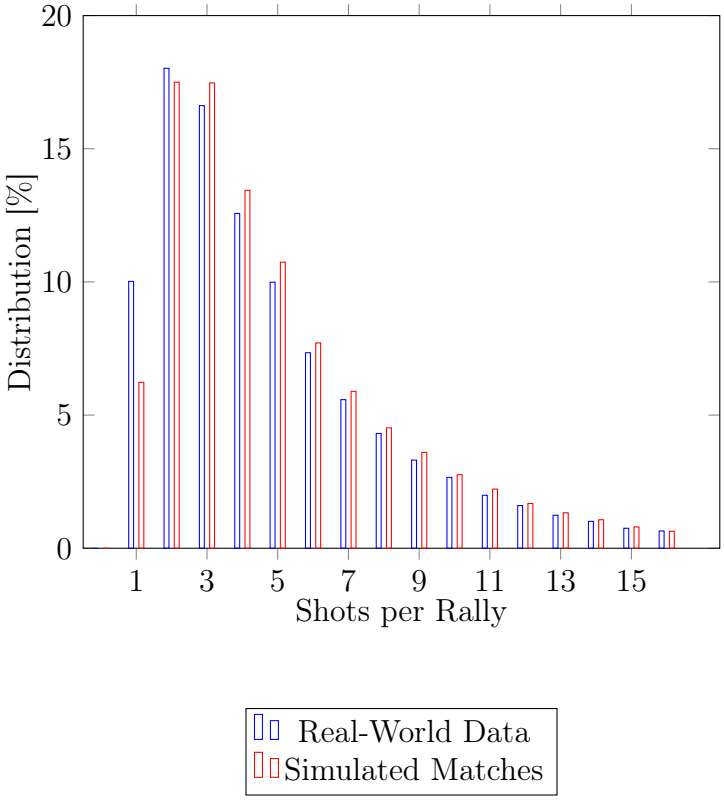

Figure 4.2: Rally Length Distribution

When comparing the rally length distribution in the real-world matches with the the rally length distribution in the simulated matches of the MCTS agents against the Average Bot, we see that in both cases the most frequent rally length is two shots. In real-world matches, a rally has only two shots in 18.02% of all rallies, and in our simulated matches, a rally is only two shots long in 17.5% of the cases. Furthermore, when looking at rallies with more than two shots we can see that in both the simulations and the real-world matches,

longer rallies occur less and less often. This is plausible because, with each shot in a rally, there is a probability with which a shot terminates the rally, in both the simulation matches and also in real-world tennis. For rallies with more than two shots, we can observe that the distribution of those rallies has slightly higher lengths in the simulated matches compared to the real-world rally lengths. This can be explained by the shot selection compromise we made during the development of *Match Point AI*. Players in *Match Point AI* are not able to play drop shots, volleys, and overheads for example, which are aggressive shots, that are more likely to terminate a rally than normal shots. Therefore it is only natural, that rallies in *Match Point AI* are slightly longer than rallies in real-world tennis. The results of a Mann-Whitney U Test on the rally distributions show no significant difference between the real-world and simulation rally lengths ($p=0.895$). This indicates, that the rallies generated in the simulation matches in *Match Point AI* are reasonably long compared to real-world tennis rallies. However, we want to address one last observation in figure 4.2. Rallies that only consist of one shot occur in real-world tennis in 10.02% of all rallies whereas one-shot rallies in our simulations only occur with 6.23%. One-shot rallies only consist of a first-serve winner, also called an ace. As soon as the returning player hits the ball, that would count as the second shot in the rally, even if the return is an error. This means that in real-world tennis it is easier to hit an ace, than in *Match Point AI*. This difference can be caused by different aspects. On the one hand side, it is likely that not being able to choose between different serves like kick serves or slice serves can contribute to this difference. Also, the serving player in *Match Point AI* does not know where the returning player positions himself before the serve. On the other hand side, the serving player can not choose to take more risks at specific serves by altering the serve speed or even by playing a first serve as a second serve, which real-world players can do. All these aspects can cause this difference in rally length distribution for one-shot rallies. Even though we made compromises while developing *Match Point AI*, we can conclude that the generated results show reasonable and plausible shot-by-shot tennis data. However, by including more aspects of real-world tennis matches into *Match Point AI* and by expanding the choices of the players in it, the generated data would be more valuable for gaining insight into real-world tennis strategies.

4.2 Analysis of Shot Patterns in Rally Openings

In the second research question, we address the possibility of gaining insight into specific shot sequences in tennis, specifically in their occurrence and their point win rates against specific players in *Match Point AI*. Because there are so many possible shot sequences and scenarios in a rally of a tennis match, we only look at a few specific situations. First, we are only looking at matches from the MCTS agent using selection policy *UCT* and decision policy *Greedy*, with the parameter settings of 10 simulation rallies and a C-value of $\sqrt{2}$. Furthermore, we only consider matches, in which this agent played against the Djokovic Bot. We then only look at rallies, in which the MCTS agent was serving. This data then is split into two sets, the first one containing all the rallies, in which the agent was serving a successful first serve and the second dataset containing the rallies, which start with a successful second serve. This split is necessary because whether a player hits a successful first serve or only a second serve has a big influence on a rally's dynamic and outcome. In 2022 for example, Novak Djokovic won 76% of all rallies after a successful first serve and only 54% after a successful second serve [67]. We then only look at rallies, in which there are at least two shots after the successful serve, and then split these rallies again, dependent on whether the serve was hit from the deuce side or advantage side of the court. This difference was considered because of the shot encoding. When a serve is hit in direction *4* from the deuce side, it would land in a right-handed opponent's forehand, whereas when it is hit in direction *4* from the ad side, it would land in that opponent's backhand (this dynamic can be seen in figure 3.1). This can influence the rally that follows and therefore is considered in this shot pattern analysis. We end up with four datasets containing different amounts of rallies, as shown in table 4.3, each representing one of the four scenarios of interest.

Serve Type	**Serve Side**	**Total Rally Count**
First Serve	Deuce Side	3.437
	Advantage Side	3.779
Second Serve	Deuce Side	2.070
	Advantage Side	2.008

Table 4.3: Total Rally Count for Matches where MCTS Agent is serving against Djokovic Bot

For every serve, the MCTS agent can choose between the three directions *4*, *5*, and *6*. After that, the return and the third shot can be played in directions *1*, *2* and *3*. The following four tables show the frequency with which specific shot patterns were detected, as well as their point win rates. The MCTS agent chooses the direction of the first and second serves as well as the direction of the third shot. The return direction is chosen by the Djokovic Bot. This holds for all of the following four tables. At this point, it is necessary to note, that the point win rates are derived from very different sample sizes due to our resulting data. Therefore we can not compare them directly. It is in the nature of the MCTS algorithm to converge to specific actions. Those actions usually yield higher rewards and therefore are visited more often than actions with less promising rewards. This behavior results in unbalanced sample sizes. The point win rate can still be an indicator of the algorithm's capability of finding successful shot patterns, not by comparing them with other win rates, but by viewing them individually with regard to their shot pattern. Table 4.4 covers the scenario in which the MCTS agent serves a successful first serve from the deuce side of the court.

First Serve Direction (Frequency)	**Return Direction (Frequency)**	**3rd Shot Direction (Frequency)**	**Point Win Rate [%]**
4 (3.358)	1 (976)	1 (3)	0.00
		2 (19)	47.37
		3 (954)	53.35
	2 (2057)	1 (238)	55.04
		2 (268)	47.76
		3 (1551)	52.93
	3 (325)	1 (175)	57.71
		2 (26)	34.62
		3 (124)	44.36
5 (10)	-	-	-
6 (69)	-	-	-

Table 4.4: Direction Frequency and Point Win Rate after First Serve from Deuce Side

Direction *4* is chosen 3.358 times, whereas direction *5* and *6* are only chosen 10 and 69 times. This shows a very fast convergence favoring direction *4* over

the other directions. One could argue, that directions *5* and *6* are not being explored enough and therefore it is interesting to see the influence of the C-value, the exploration-exploitation trade-off parameter, in the *UCT* formula on this convergence. The influence of different C-values will be discussed in chapter 4.5. This fast convergence of the serves can be seen in all of our four scenarios. It is worth mentioning, however, that the frequency in the following tables does not include rallies that were simulated in the fourth phase of the algorithm. That means that the rally $4 \rightarrow 1 \rightarrow 1$ in table 4.4 actually has been played three times in a match, but in the simulation phase, this shot sequence probably has been visited hundreds of times. Both first serve scenarios converge to direction *4* (Table 4.4 and 4.5), whereas both second serve scenarios converge to direction *6* (Table 4.6 and 4.7). Due to the sample size of the actions, that the algorithm did not converge to, we continue by only analyzing the shot patterns following the serve directions *4* for first serves and *6* for second serves because these are the shot direction that the algorithm chose the most frequent. After the first serve, table 4.4 shows the return directions of the Djokovic Bot. Because this is based on a fixed strategy, the frequency of those return directions is the result of the probabilities for this situation, which were extracted from the real-world dataset. The third shot direction is again chosen by the MCTS agent. We can see a very clear convergence to direction *3* when the Djokovic Bot was returning in direction *1*. When Djokovic was returning in direction *2*, we see the same convergence in direction *3*. When he was returning in direction *3* however, we can see that the MCTS agent did not converge to one direction but was mixing directions *1* and *3*.

Table 4.5 represents the rallies, in which the MCTS agent serves a first serve from the advantage side. For all three return directions of the Djokovic Bot, direction *2* was chosen by the MCTS agent the least. Return direction *1* was followed up by the agent with a mix of shots in directions *1* and *3*. The same holds for the third shot when the Djokovic Bot played the return in direction *2*. For return direction *3*, the agent converged to direction *1*. But the agent mixed in a good amount of shots in direction *3*.

First Serve Direction (Frequency)	Return Direction (Frequency)	3rd Shot Direction (Frequency)	Point Win Rate [%]
4 (3.173)	1 (127)	1 (69)	40.58
		2 (13)	53.85
		3 (45)	55.56
	2 (1.248)	1 (546)	48.54
		2 (183)	38.80
		3 (519)	49.71
	3 (1.798)	1 (1153)	44.23
		2 (206)	37.38
		3 (439)	30.98
5 (62)	-	-	-
6 (544)	-	-	-

Table 4.5: Direction Frequency and Point Win Rate after First Serve from Advantage Side

Table 4.6 represents the rallies, in which the agent is opening with a second serve from the deuce side. This time, the choice of direction of the serve is converging to direction *6*. We can see a clear convergence of the MCTS agent's behavior, favoring shots in direction *3* for rallies in which the Djokovic Bot is returning in direction *2*. When the bot is returning in direction *3*, we again can find a mix of shots in directions *1* and *3*. For returns in direction *1* the agent plays most of the shots in direction *2*, as well as a fair amount of shots in direction *3*.

Finally, in 4.7 the shot patterns are shown for rallies, in which the MCTS agent was serving a second serve from the advantage side of the court. For every return direction of the Djokovic Bot in this scenario, we have a favorite shot direction of the MCTS agent. But in each case, the agent is mixing in a fair amount of shots in other directions. In case the bot returns in direction *1*, the agent plays most shots in direction *2* while mixing in shots in direction *1*. Returns in direction *2* are answered most commonly by shots in direction *1* while mixing in shots in direction *2* as well as *3*. For returns in direction *3*, we can see that the agent plays a majority of shots in direction *1*, while still mixing in shots in direction *3*. In this scenario, the agent never truly converges to a single shot direction regardless of the return direction of the bot.

Second Serve Direction (Frequency)	Return Direction (Frequency)	3rd Shot Direction (Frequency)	Point Win Rate [%]
4 (23)	-	-	-
5 (17)	-	-	-
6 (2.030)	1 (185)	1 (22)	63.64
		2 (100)	44.00
		3 (63)	53.97
	2 (1.292)	1 (31)	58.07
		2 (57)	59.65
		3 (1.204)	51.66
	3 (553)	1 (281)	52.31
		2 (10)	30.00
		3 (262)	50.00

Table 4.6: Direction Frequency and Point Win Rate after Second Serve from Deuce Side

To summarize our findings of the shot pattern analysis, we show the most frequent shot patterns per scenario in figure 4.3. An interesting observation from this is, that the most frequent shot patterns after a first serve of the MCTS agent is followed up by the shot, that presumably makes the opponent run the most.

On the left court, for example, the serve is to the left of the left service box, followed by the third shot to the far right of the court. The same happens on the middle left court in that figure. The serve is going to the far right of the right service box followed by a shot to the far left of the court. This shot combination is certainly not new in the world of tennis. Playing the ball to the open court, far away from the opponent, and trying to make the opponent run to the ball is common sense in tennis because exhaustion and fatigue as well as having to play a ball under pressure leads to more errors. This analysis shows, that the MCTS algorithm is capable of finding reasonable strategies for ball placement in tennis when it is applied to *Match Point AI*.

Second Serve Direction (Frequency)	Return Direction (Frequency)	3rd Shot Direction (Frequency)	Point Win Rate [%]
4 (176)	-	-	-
5 (182)	-	-	-
6 (1650)	1 (235)	1 (79)	26.58
		2 (121)	31.41
		3 (35)	22.86
	2 (953)	1 (511)	49.51
		2 (265)	40.38
		3 (177)	49.15
	3 (462)	1 (327)	45.87
		2 (2)	50.00
		3 (133)	51.13

Table 4.7: Direction Frequency and Point Win Rate after Second Serve from Advantage Side

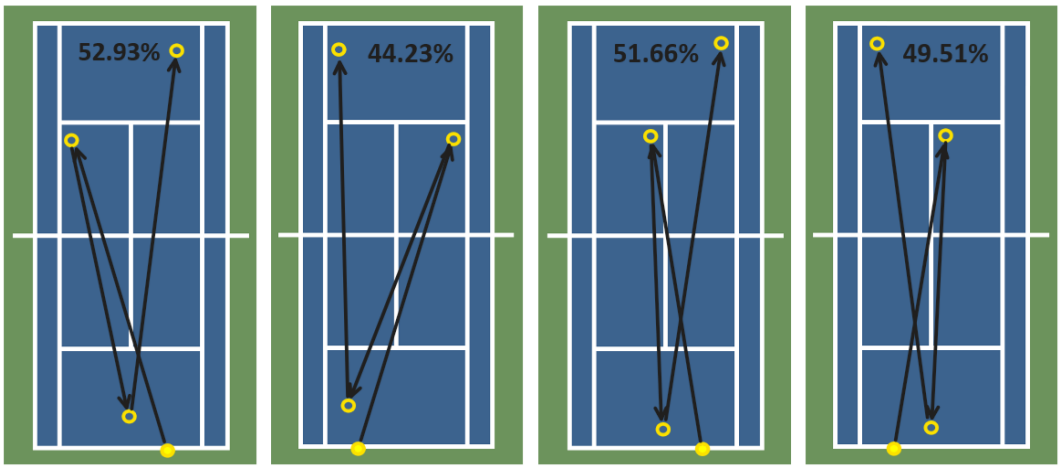

Figure 4.3: Most frequent Shot Patterns and Point Win Rates (left to right): first serve from deuce side, first serve from advantage side, second serve from deuce side, second serve from advantage side)

4.3 Evaluation of different Selection Policies

In our third research question, we analyze the influence of different selection policies, utilized in the selection phase in the MCTS algorithm, on the effectiveness of winning points in *Match Point AI* against the two bots. Therefore, we analyze the results of the experiment set two, especially in the relationship between the point win rate of the MCTS agents and the three different selection policies: *Random*, *Greedy* and *UCT*. At first, this is done without considering the opponent or the decision policy. A Chi-Squared test was conducted and the results show a significant relationship (p-value=0.0028) between the utilized selection policy and the point win rate. Therefore the null hypothesis H_{02} can be rejected and the alternative hypothesis H_{12} is accepted, which means there is a significant relationship between the point win rate of the MCTS agent and its selection policy. As a post-hoc test, we then perform a pairwise comparison of the three different selection policies to see between which policies the significant difference can be found. A Bonferroni-Holm correction was used to account for the family-wise error rate. With a corrected significance value of 0.0017, there is a significant difference between the two selection policies *Greedy* and *Random*. Furthermore, the effect size of this relationship is analyzed according to Cramer's V with a DF of 2. The result of that test shows a small effect size (V=0.0105) in this relationship. Examining these findings alongside the data presented in table 4.8, we can conclude that the algorithm demonstrates significantly better performance in winning points within *Match Point AI* when employing selection policy *Random* compared to the use of selection policy *Greedy*.

Selection Policy	**Greedy**	**Random**	**UCT**
MCTS Agent	61.989	64.267	62.158
Bots	63.865	64.435	63.117

Table 4.8: Total Point Wins using different Selection Policies

No significant differences were found in the performance of the MCTS algorithm when comparing the selection policies *UCT* and *Random* or *UCT* and *Greedy*. The tests up to this point were conducted on a dataset representing match simulations of the MCTS agents against both the Average Bot as well as the Djokovic Bot. To analyze, if the influence of the three selection policies is different depending on the two opponents, this dataset is split into two separate

datasets. One containing all the simulated matches, that the different MCTS agents played against the Djokovic Bot and the other dataset with all simulation matches against the Average Bot. Two separate Chi-Squared tests were then conducted on these datasets. This enables us to further analyze the influence of different selection policies on the performance of the MCTS agents when they are playing against a specific bot. The total point wins of the different MCTS agents using the three selection policies against the two fixed-strategy bots is shown in table 4.9.

Opponent	**Average Bot**			**Djokovic Bot**		
Selection Policy	Greedy	Random	UCT	Greedy	Random	UCT
MCTS Agent	32.702	32.754	32.337	29.287	31.513	29.821
Bot	31.739	31.503	31.007	32.126	32.932	32.110

Table 4.9: Total Point Wins of MCTS Agents using different Selection Policies against different Opponents

The results of the Chi-Squared test conducted for the dataset with the matches against the Average Bot show no significant relationship between the total point wins and the selection policy in use (p=0.5299). For the dataset containing all the matches against the Djokovic Bot, results of the Chi-Squared test show a significant relationship between selection policy and point winner (p=0.0001). We then again tested this relationship in pairwise comparisons of all the selection policies with the Bonferroni-Holm correction and found significant differences between selection policy *Random* and *Greedy* as well as *Random* and *UCT*. In both cases, the *Random* selection policy performed significantly better against the Djokovic Bot. To analyze the effect sizes of the significant differences, we again calculated Cramer's V with a DF of 2. A small effect size is found between selections policies *Greedy* and *Random* (V=0.0359) as well as between *Random* and *UCT* (V=0.0303). The pairwise comparisons show no significant relationship between the selection policies *Greedy* and *UCT*.

4.4 Evaluation of Decision Policies

The next research question aims at evaluating the influence of the different decision policies *Greedy* and *UCT* on the performance of the algorithm to win

points against the two bots. In table 4.10 the total points won by the agents utilizing the two decision policies against both bots are displayed.

Decision Policy	**Greedy**	**UCT**
MCTS Agent	96.893	91.521
Bots	94.588	96.829

Table 4.10: Total Point Wins using different Decision Policies.

Another Chi-Squared test was conducted on this data and a significant relationship between the point wins and the decision policy was found (p=0.0000). With this result we can reject the null hypothesis H_{03} and accept the alternative hypothesis H_{13}, stating that there is a significant relationship between the total points won and the utilized decision policy. With the calculation and interpretation of Cramer's V, we can also say, that this relationship has a small effect size (V=0.0216). In combination with the results in table 4.10 we can see, that the decision policy *Greedy* is performing significantly better compared to decision policy *UCT*. We then again split the dataset into two separate sets to see if this relationship is dependent on the opponent of the MCTS agents. The first dataset only consists of the results of the matches against the Average Bot and the second only contains the results of the matches against the Djokovic Bot. In table 4.11 the total point wins of the MCTS agents using the different decision policies against the two bots is shown.

Opponent	**Average Bot**		**Djokovic Bot**	
Decision Policy	Greedy	UCT	Greedy	UCT
MCTS Agent	48.828	48.965	48.065	42.556
Bot	45.243	49.006	49.345	47.823

Table 4.11: Total Point Wins of MCTS Agents employing different Decision Policies against different opponents

For both of these datasets, we then perform Chi-Squared tests, to analyze the differences in the performance of the decision policy against the two bots. We found a significant relationship between the point wins and the decision policy in the dataset containing only the matches against the Average Bot (p=0.0000). When looking at Cramer's V we can see that there is a small effect in this relationship (V=0.0193). Against the Average Bot, the decision policy

Greedy is performing significantly better compared to the Decision Policy *UCT*. This significant relationship is also found in the dataset with the results of the matches against the Djokovic Bot. The Chi-Squared test shows a significant difference in using the different decision policies *Greedy* and *UCT* (p = 0.0000) when playing matches against the Djokovic Bot. This relationship again has a small effect (V=0.0226), in which the *Greedy* approach performs significantly better compared to the *UCT* policy.

4.5 Parameter Dependency of the MCTS Algorithm

To analyze the influence of different parameter settings on the MCTS algorithms performance, we use the MCTS agent employing the selection policy *UCT* and decision policy *Greedy*. The choice of the selection policy is due to the adaptability of the UCT formula. Changing the C-value in that formula influences the trade-off between exploration and exploitation. In our experiment set three, we compare three different C-Values: $\sqrt{2} - 0.5$, $\sqrt{2}$ and $\sqrt{2} + 0.5$ in their influence on the MCTS agents ability to win points in *Match Point AI*. The decision policy is chosen, because it showed a better performance compared to the *UCT* decision policy in our previous analyses. Another interesting parameter is the number of simulations the algorithm runs through during the simulation phase. For this parameter, we also test three different values: 5, 10, and 15. Each of those nine different, parameter adapted, MCTS agents competes in 100 matches against the Average Bot in our experiment set three. From analyzing the resulting data, we evaluate the influence of different parameter settings on the MCTS algorithm's performance. Table 4.12 shows the resulting point and match win rates of the adapted MCTS agent against the Average Bot.

Just looking at those results, it is already possible to see a slightly better point win rate for a C-value of $\sqrt{2} + 0.5$ compared to the other C-values. The number of simulations on the other hand does not seem to influence the point win rates in a specific way. To confirm or reject these observations, we again conduct Chi-Squared tests, using the contingency table 4.13, where A represents the MCTS agent and B the fixed strategy bot.

MCTS UCT/Greedy Agent		Average Bot	
C-Value	Number of Simulations	Point Win Rate [%]	Match Win Rate [%]
$\sqrt{2} - 0.5$	5	51.51	68.00
	10	51.34	64.00
	15	50.53	55.00
$\sqrt{2}$	5	51.07	66.00
	10	50.70	58.00
	15	51.71	63.00
$\sqrt{2} + 0.5$	5	51.98	72.00
	10	52.47	77.00
	15	52.26	75.00

Table 4.12: Win percentages of MCTS UCT/Greedy Agent with parameter adaptations against Average Bot

N Simus	**5**		**10**		**15**	
Point Winner	A	B	A	B	A	B
C-Value						
$\sqrt{2} - 0.5$	8.720	8.210	8.404	7.965	8.320	8.146
$\sqrt{2}$	8.313	7.968	8.158	7.933	8.318	7.771
$\sqrt{2} + 0.5$	8.425	7.784	8.511	7.711	8.016	7.324

Table 4.13: Contingency Table Parameter dependency

First, we look at the relationship between point win rate and the number of simulations. The result of the Chi-Squared test does not show a significant influence of the different simulation numbers on the point win rate (p=0.1075). With these results, we can accept our null hypothesis H_{05} and reject its alternative H_{15}. This can also be seen in figure 4.4, where the blue and red curves, representing 5 and 10 simulations, have a very similar trajectory. The green curve has a slightly different trajectory, but the difference is not significant as shown by the Chi-Squared test.

In figure 4.4 we can also confirm our observation, that the point win rate for a C-value of $\sqrt{2} + 0.5$ seems higher compared to the other two C-Values. To analyze this relationship we conduct another Chi-Squared test. The results show a significant relationship between C-value and point win rate (p=0.0009). The

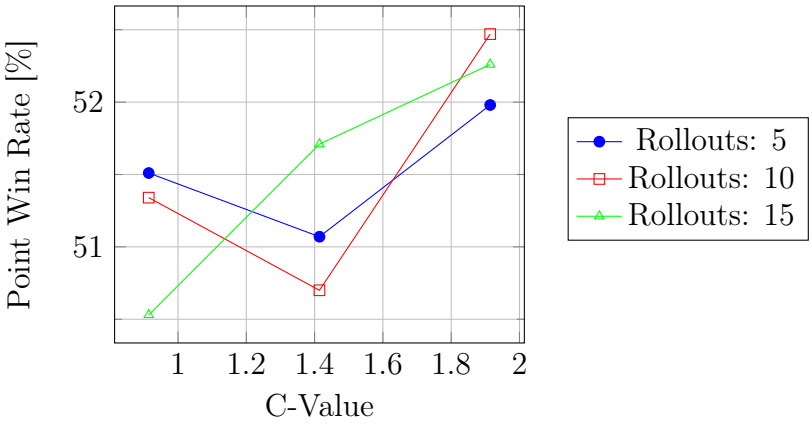

Figure 4.4: Point Win Rates of different *UCT/Greedy* MCTS Adaptations against the Average Bot

null hypothesis H_{04} can be rejected and we accept our alternative hypothesis H_{14}. As a post-hoc test, we then perform a pairwise comparison with Bonferroni-Holm corrected significance values. Results of the pairwise comparison show significant differences between the C-values $\sqrt{2}+0.5$ and $\sqrt{2}$ as well as $\sqrt{2}+0.5$ and $\sqrt{2}-0.5$. The effect size (DF=2) of the difference between the C-Values $\sqrt{2}+0.5$ and $\sqrt{2}-0.5$ is found to be a medium effect size (V=0.0727), and the same holds for the effect size between $\sqrt{2}+0.5$ and $\sqrt{2}$ (V=0.0734). We can conclude, that for the application of the *UCT/Greedy* MCTS algorithm to *Match Point AI*, the number of simulations tested does not have a significant influence on the performance of the MCTS algorithm. The algorithm does show a significantly better performance in winning points in the simulated matches against the Average Bot for a C-value of $\sqrt{2}+0.5$, compared to $\sqrt{2}$ and $\sqrt{2}-0.5$.

5 Summary and Future Work

Starting with an introduction, in which we state our motivation and goal as well as the research questions and their hypotheses, this thesis continues with an introduction to RL and presents different algorithms in the field and their applications to various problems. We then go on to present the basic principles of the MCTS algorithm and introduce related studies, in which the algorithm has been applied to board and computer games, real-world problems, and sports. Here we point out, where our research gap lies and continue by giving a short introduction to the sport of tennis, in which we explain important aspects and some terminology relevant to this thesis. To apply different adaptations of the MCTS algorithm to tennis, this thesis then presents the newly developed tennis match simulation environment, *Match Point AI*. We showcase which aspects of real-world tennis are included in *Match Point AI* and which were neglected and why. The thesis then continues by describing the MCTS agents and the different policies and adaptations they can utilize in *Match Point AI*. The MCTS algorithm needs an opponent against which it can learn to play tennis. Therefore we created two different fixed-strategy bots, whose behaviour in *Match Point AI* is based on situation-dependent probabilities, which were extracted from a real-world shot-by-shot tennis dataset. These Bots can be considered as training partners to our MCTS based agents. Before presenting our results, we outline the experiment settings used to address the different research questions and then explain the methodology utilized for the statistical evaluation of the generated results.

In the results of our first research question, we discuss the strengths and limitations of *Match Point AI*. The biggest limitation here is missing data. Information about ball velocities of tennis shots and player positions and movements on the court are not tracked in the dataset we used. It is safe to say that *Match Point AI* would generate better data in terms of both quality and quantity when more precise data is included in the design of the environment and the bot's behaviors. For our purpose of trying to apply MCTS to tennis,

however, *Match Point AI* proved to be a suitable environment. Despite its limitations, *Match Point AI* can generate valuable shot-by-shot tennis data, when we simulated matches between the MCTS agents and the fixed-strategy bots. In the resulting data, a plausible relationship between point and match win rates can be found. When we let the Djokovic Bot play against the Average Bot, his match win rate is close to the real-world Djokovic's match win rate in the same time frame. Also, the rally length distribution between the generated data and the real-world data is not significantly different. With the analysis conducted to answer our second research question, we found the most frequent shot patterns at the beginning of rallies, in which the MCTS *UCT/Greedy* agent was serving against the Djokovic Bot. The results show the shot directions, which are chosen most often by the agent, and the corresponding point win rates. The agent's strategy seems to put the opponent under pressure, by letting him run from one corner of the court to the other without playing the ball in the same direction twice in a row. We also discover that in some scenarios, the algorithm converges not only to one direction, which would be easy for an opponent to anticipate, but utilizes mixed strategies by varying the shot directions in similar situations. We can conclude that the MCTS agent's behavior is converging to specific actions in specific situations. With that, he is able to find strategies in *Match Point AI*, which are plausible strategies for real-world tennis as well. This also confirms our answer to research question one, in which we explain why *Match Point AI* can generate realistic tennis data. In the third research question, we evaluate different adaptations of the MCTS algorithm. We specifically analyze the algorithm's performance, when it is using the three different selection policies, *UCT*, *Greedy*, and *Random*. In matches against the Djokovic Bot, the MCTS agent won significantly more points when utilizing the *Random* selection policy compared to the two other selection policies *UCT* and *Greedy*. In matches against the Average Bot, however, there is no significant difference in the points won by the agents using the different selection policies. Similarly to research question three, in research question four we evaluate different policies utilized by the MCTS agents. This time we compare the decision policies *Greedy* and *UCT* in their ability to win points against the Bots. Results show, that against both Bots, the agents using decision policy *Greedy* win significantly more points than the agents using decision policy *UCT*. In research question five we adapt two parameters of the MCTS *UCT/Greedy* agent and test the adapted agents against the Average Bot. Firstly, we use three different values for the number of simulations, which

the algorithm conducts in the simulation phase. Our results showed, that this adaptation has no significant influence on the point win rate of the agent. Secondly, we use three different C-values in the *UCT* formula. Here we find a significantly better performance in winning points of the MCTS agent when it is using a C-value of $\sqrt{2}+0.5$, compared to the lower C-values tested. This result indicates, that in our test case, a preference for exploration over exploitation in the selection phase yields a better performance of the MCTS agent in *Match Point AI*.

In this thesis, we show how the MCTS algorithm can be applied to the decision-making processes in tennis, by introducing the tennis simulator *Match Point AI*. Even though the environment is capable of generating plausible tennis data, it still offers a lot of potential for future improvements. Most of these improvements, however, are only possible by including more and better tennis data. The dataset utilized for this thesis from the match charting project is a great source of tennis data and we have not yet exploited its full potential, but it also has limits. For example, we could include more shot-type options for our bots and agents such as drop shots, lobs, and volleys based on the match charting project data, but the dataset is limited when it comes to information about ball velocities or player movements and positions on the court. There are however other ways of tracking tennis data, which can already keep track of the data automatically and more precise than the crowd-sourced match charting project, such as *SwingVision* for example. *SwingVision* is an application, which can track precise shot placements, shot types, and ball velocities live, only using a smartphone camera. Another example is electronic line calling in tennis, which nowadays replaces line judges at major tournaments and already tracks ball placement down to the millimeter. The problem with these technologies is, that the resulting data is not made public. With data like that, the capabilities of *Match Point AI* would increase drastically and the bot strategies would represent the real-world player's behavior much more precisely. In this thesis, we also analyze the shot placement strategies of one of the MCTS agents against the Djokovic Bot. It would also be very interesting to see, if the agents come up with different strategies against different Bots, mirroring other player's behavior than that of Novak Djokovic, because the best strategy to win against Djokovic is probably not the best strategy to win against Rafael Nadal. The shot pattern analysis was conducted for very specific scenarios in a tennis match. Tennis and also the resulting data of *Match Point AI* offers a wide range of other scenarios, that might be interesting to explore as well. In this thesis, we then evaluate

the MCTS algorithm when it is applied to the decision-making processes in tennis. We analyze the influence of different policies and different parameter settings in the MCTS algorithm on its performance in winning points in *Match Point AI*. There are many policies and values for parameters that are yet to be tested, to truly find the best adaptation of the MCTS algorithm when applied to tennis. There also exist interesting approaches, in which MCTS is combined with evolutionary algorithms [1] or neural networks [55]. The results show that this can improve the performance of the algorithm as well. The effect of these combinations on the agent's performance in *Match Point AI* would also be very interesting to analyze.

Bibliography

[1] Hendrik Baier and Peter I. Cowling. Evolutionary mcts for multi-action adversarial games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018.

[2] Radha-Krishna Balla and Alan Fern. Uct for tactical assault planning in real-time strategy games. pages 40–45, 01 2009.

[3] C. C. Battaile, D. J. Srolovitz, and J. E. Butler. A kinetic monte carlo method for the atomic-scale simulation of chemical vapor deposition: Application to diamond. *Journal of Applied Physics*, 82(12):5705–5712, 1997.

[4] Donald F. Beal and Martin C. Smith. Temporal difference learning applied to game playing and the results of application to shogi. *Theoretical Computer Science*, 252(1):105–119, 2001. CG'98.

[5] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[6] M. Ryan Rodenberg Benjamin Wright and Jeff Sackmann. Incentives in best of n contests: Quasi-simpson's paradox in tennis. *International Journal of Performance Analysis in Sport*, 13(3):790–802, 2013.

[7] Guillaume Chaslot, Mark Winands, H. Herik, Jos Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 04:343–357, 11 2008.

[8] R.R. Chen and S. Meyn. Value iteration and optimization of multiclass queueing networks. *Queueing Systems*, 32:65–97, 1999.

[9] Zhengxing Chen, Truong-Huy D Nguyen, Yuyu Xu, Christopher Amato, Seth Cooper, Yizhou Sun, and Magy Seif El-Nasr. The art of drafting: A team-oriented hero recommendation system for multiplayer online battle arena games. In *RecSys '18: Proceedings of the 12th ACM Conference on Recommender Systems*, pages 200–208, 2018.

[10] Michael Chung, Michael Buro, and Jonathan Schaeffer. Monte carlo planning in rts games. In *IEEE Conference on Computational Intelligence and Games*, 2005.

[11] Jacob Cohen. *Statistical power and analysis for the behavioral sciences*. Lawrence Erlbaum Associates, Hillsdale, NJ, 2nd edition, 1988.

[12] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[13] Harald Cramér. *Mathematical Methods of Statistics*. Princeton University Press, Princeton, 1946. Chapter 21. The two-dimensional case.

[14] Felippe Schmoeller da Roza. Components of a reinforcement learning framework, September 2023. Published on September 12, 2023. Accessed on February 5, 2024.

[15] Damien Ernst, Mevludin Glavic, Pierre Geurts, and Louis Wehenkel. Approximate value iteration in the reinforcement learning context. application to electrical power system control. *International Journal of Emerging Electric Power Systems*, 3(1), 2005.

[16] ESPN. Tennis rankings, 2024. Accessed: 9 January 2024.

[17] Iván Francisco-Valencia, José Raymundo Marcial-Romero, and Rosa María Valdovinos-Rosas. A comparison between ucb and ucb-tuned as selection policies in ggp. *Journal of Intelligent & Fuzzy Systems*, 36(5):5073–5079, January 1 2019.

[18] Edgar Galván, Gavin Simpson, and Fred Valdez Ameneyro. Evolving the mcts upper confidence bounds for trees using a semantic-inspired evolutionary algorithm in the game of carcassonne. *IEEE Transactions on Games*, 15(3):420–429, 2023.

[19] Jacob Gollub. *Producing Win Probabilities for Professional Tennis Matches from any Score*. PhD thesis, 2019.

[20] Benjamin Gravell, Karthik Ganapathy, and Tyler Summers. Policy iteration for linear quadratic games with stochastic parameters. *IEEE Control Systems Letters*, 5(1):307–312, 2021.

[21] Stefan Freyr Gudmundsson and Yngvi Björnsson. Mcts: improved action selection techniques for deterministic games. In *Proceedings of the IJCAI-11 Workshop on General Game Playing (GIGA'11)*, pages 23–30, 2011.

[22] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[23] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[24] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[25] R.A. Howard. *Dynamic programming and Markov processes*. Technology Press of Massachusetts Institute of Technology, 1960.

[26] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[27] Emil Juul Jacobsen, Rasmus Greve, and Julian Togelius. Monte mario: Platforming with mcts. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 293–300, New York, NY, USA, 2014. Association for Computing Machinery.

[28] Corwin Joy, Phelim P. Boyle, and Ken Seng Tan. Quasi-monte carlo methods in numerical finance. *Management Science*, 42(6):926–938, 1996.

[29] Chanaka Keerthisinghe, Gregor Verbič, and Archie C. Chapman. A fast technique for smart home management: Adp with temporal difference learning. *IEEE Transactions on Smart Grid*, 9(4):3291–3303, 2018.

[30] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[31] Kennard Laviers and Gita Sukthankar. A monte carlo approach for football play generation. In *Proceedings of the 6th AAAI Conference on Artificial*

Intelligence and Interactive Digital Entertainment, AIIDE 2010, pages 150–155. AAAI, October 11–13 2010.

[32] Machine Learning. Final project report: Real time tennis match prediction using machine learning. 2017.

[33] Kexin Li, Qianwang Deng, Like Zhang, Qing Fan, Guiliang Gong, and Sun Ding. An effective mcts-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem. *Computers Industrial Engineering*, 155:107211, 2021.

[34] Yuewen Li, Xiaoling Wang, Qi Kang, Zheng Fan, and Shuaiyu Yao. An mcts-based solution approach to solve large-scale airline crew pairing problems. *IEEE Transactions on Intelligent Transportation Systems*, 24(5):5477–5488, 2023.

[35] Francesco Lisi, Matteo Grigoletto, and Mirko G Briglia. On the distribution of rally length in professional tennis matches. 2023.

[36] Francesco Lisi, Matteo Grigoletto, and Tommaso Canesso. Winning tennis matches with fewer points or games than the opponent. pages 313–324, 2019. 1 Jan. 2019.

[37] X. Liu and A. Fotouhi. Formula-e race strategy development using artificial neural networks and monte carlo tree search. *Neural Computing & Applications*, 32:15191–15207, 2020.

[38] Yang Liu, Zhanpeng Jiang, Lichao Hao, Zuoxia Xing, Mingyang Chen, and Pengfei Zhang. Data-driven robust value iteration control with application to wind turbine pitch control. *Optimal Control Applications and Methods*, 44(2):637–646, 2023.

[39] Travis Mandel, Yun-En Liu, Sergey Levine, Emma Brunskill, and Zoran Popovic. Offline policy evaluation across representations with applications to educational games. In *AAMAS*, volume 1077, 2014.

[40] Jacek Mańdziuk. *MCTS/UCT in Solving Real-Life Problems*, pages 277–292. Springer International Publishing, Cham, 2018.

[41] Nicholas Metropolis. The beginning of the monte carlo method. *Los Alamos Science Special Issue*, 15:125–130, 1987.

[42] Nicholas Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949. PMID: 18139350.

[43] Rahul Reddy Nadikattu. Implementation of new ways of artificial intelligence in sports. *Journal of Xidian University*, 14(5):5983–5997, 2020.

[44] Katsuki Ohto and Tetsuro Tanaka. A curling agent based on the monte-carlo tree search considering the similarity of the best action among similar states. In Mark H.M. Winands, H. Jaap van den Herik, and Walter A. Kosters, editors, *Advances in Computer Games*, pages 151–164, Cham, 2017. Springer International Publishing.

[45] Karl Pearson. On the criterion that a given system of derivations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(5):157–175, 1900.

[46] Pierre Perick, David L. St-Pierre, Francis Maes, and Damien Ernst. Comparison of different selection strategies in monte-carlo tree search for the game of tron. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 242–249, 2012.

[47] D. E. Raeside. Monte carlo principles and applications. *Phys. Med. Biol.*, 21:181–197, 1976.

[48] G Roelofs. Monte carlo tree search in a modern board game framework. *Research paper available at umimaas. nl*, 2012.

[49] Leonardo Rossi, Mark H. M. Winands, and Christoph Butenweg. Monte carlo tree search as an intelligent search tool in structural design problems. *Engineering with Computers*, 38(4):3219–3236, August 2022.

[50] Jeff Sackmann. Tennis match charting project. `www.tennisabstract.com/charting/meta.html`, 2023.

[51] Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate modified policy iteration and its application to the game of tetris. *J. Mach. Learn. Res.*, 16(49):1629–1676, 2015.

[52] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[53] Nick Sephton, Peter I. Cowling, and Nicholas H. Slaven. An experimental study of action selection mechanisms to create an entertaining opponent. In

2015 IEEE Conference on Computational Intelligence and Games (CIG), pages 122–129, 2015.

[54] Pete Shinners. Pygame. `http://pygame.org/`, 2011.

[55] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[56] Dennis J. N. J. Soemers, Chiara F. Sironi, Torsten Schuster, and Mark H. M. Winands. Enhancements for real-time monte-carlo tree search in general video game playing. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2016.

[57] Ruizhuo Song and Liao Zhu. Stable value iteration for two-player zero-sum game of discrete-time nonlinear systems based on adaptive dynamic programming. *Neurocomputing*, 340:180–195, 2019.

[58] Ole Stenzel, L. Jan Anton Koster, Ralf Thiedmann, Stefan D. Oosterhout, Rene A. J. Janssen, and Volker Schmidt. A new approach to model-based simulation of disordered polymer blend solar cells. *Advanced Functional Materials*, 22(6):1236–1244, March 2012.

[59] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, The MIT Press, Cambridge, Massachusetts; London, England, 2015.

[60] István Szita, Guillaume Chaslot, and Pieter Spronck. Monte-carlo tree search in settlers of catan. In H. Jaap van den Herik and Pieter Spronck, editors, *Advances in Computer Games*, pages 21–32, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[61] Marcin Szubert and Wojciech Jaśkowski. Temporal difference learning of n-tuple networks for the game 2048. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8, 2014.

[62] Jun Tao, Gui Wu, Zhentong Yi, and Peng Zeng. Optimization and improvement for the game 2048 based on the mcts algorithm. In *2020 Chinese Control And Decision Conference (CCDC)*, pages 235–239, 2020.

[63] Antonio Terroba, Walter Kosters, Javier Varona, and Cristina S. Manresa-Yee. Finding optimal strategies in tennis from video sequences. *International Journal of Pattern Recognition and Artificial Intelligence*, 27(06):1355010, 2013.

[64] Gerald Tesauro. Temporal difference learning of backgammon strategy. In Derek Sleeman and Peter Edwards, editors, *Machine Learning Proceedings 1992*, pages 451–457. Morgan Kaufmann, San Francisco (CA), 1992.

[65] Ralf Thiedmann, Ole Stenzel, Aaron Spettl, Paul R. Shearing, Stephen J. Harris, Nigel P. Brandon, and Volker Schmidt. Stochastic simulation model for the 3d morphology of composite materials in li–ion batteries. *Computational Materials Science*, 50(12):3365–3376, 2011.

[66] Sofiane Touati, Mohammed Said Radjef, and Lakhdar Sais. A bayesian monte carlo method for computing the shapley value: Application to weighted voting and bin packing games. *Computers Operations Research*, 125:105094, 2021.

[67] ATP Tour. Atp players, 2024. Accessed: 2024/01/09.

[68] Otakar Trunda and Roman Barták. Using monte carlo tree search to solve planning problems in transportation domains. In Félix Castro, Alexander Gelbukh, and Miguel González, editors, *Advances in Soft Computing and Its Applications*, pages 435–449, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[69] USTA. Tennis scoring rules, 2024. Accessed: 2024/03/07.

[70] Kyriakos G. Vamvoudakis and F. L. Lewis. Policy iteration algorithm for distributed networks and graphical games. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 128–135, 2011.

[71] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[72] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[73] I-Chen Wu, Hsin-Ti Tsai, Hung-Hsuan Lin, Yi-Shan Lin, Chieh-Min Chang, and Ping-Hung Lin. Temporal difference learning for connect6. In H. Jaap

van den Herik and Aske Plaat, editors, *Advances in Computer Games*, pages 121–133, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[74] Maciej Świechowski, Tomasz Tajmajer, and Andrzej Janusz. Improving hearthstone ai by combining mcts and supervised learning algorithms. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018.

Declaration of Authorship

I hereby declare that this thesis was created by me and me alone using only the stated sources and tools.

Carlo Nübel Magdeburg, 12.03.2024