Jannis Römermann

Evaluation und Vergleich verschiedener Entscheidungsalgorithmen zur Optimierung von Tennisstrategien

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
INF FAKULTÄT FÜR INFORMATIK

Intelligent Cooperative Systems
Computational Intelligence

Evaluation und Vergleich verschiedener Entscheidungsalgorithmen zur Optimierung von Tennisstrategien

Bachelorarbeit

Jannis Römermann

24. Juni 2025

Betreuender Professor: Prof. Dr.-Ing. habil. Sanaz Mostaghim

Betreuer: Carlo Nübel, M.Sc.

Jannis Römermann: *Evaluation und Vergleich verschiedener Entscheidungsalgorithmen zur Optimierung von Tennisstrategien*
Otto-von-Guericke Universität
Intelligent Cooperative Systems
Computational Intelligence
Magdeburg, 2025.

Abstrakt

Tagtäglich treffen wir eine Vielzahl an Entscheidungen, wobei selbst die kleinste Änderung große Auswirkungen hervorrufen kann. Besonders im Sport wird dies einem bewusst, da in Sportarten wie Tennis gezeigt wird, wie selbst ein einzelner Schlag den ganzen Spielverlauf ändern kann. Eine dieser Fragen, die sich Tennisspieler stellen müssen, ist, in welche Richtung sie spielen wollen. Professionelle Spieler können zwar aufgrund ihrer Erfahrung und ihres Trainings dies gut einschätzen, trotzdem stellt sich die Frage, ob es eine optimale Strategie existiert, mit der die Spieler ihre Leistung verbessern können. Daher werden in dieser Arbeit zwei Algorithmen vorgestellt, deren Ziel es ist, eine solche Strategie zu entwickeln. Dazu wird das Framework Match Point AI genutzt, welches Tennismatches basierend auf den Schlagrichtungen simuliert. Es werden Bots basierend auf der Monte-Carlo-Simulation und auf dem Rolling Horizon Evolutionary Algorithm entwickelt, die dann in mehreren Matches miteinander verglichen und deren entwickelte Strategien analysiert werden. Dabei werden vorher die Parameter des RHEA-Algorithmus analysiert, welche Kombination am besten geeignet ist.

Inhaltsverzeichnis

Abbildungsverzeichnis **V**

Tabellenverzeichnis **VII**

Quellcodeverzeichnis **IX**

Abkürzungsverzeichnis **XI**

1 Einleitung **1**

1.1 Motivation 1

1.2 Forschungsfragen 2

1.3 Struktur 3

2 Stand der Forschung **5**

2.1 Entscheidungsalgorithmen 5

2.1.1 Entscheidung 5

2.1.2 Algorithmus 6

2.1.3 Maschinelles Lernen 6

2.2 Markov Decision Process 9

2.3 Monte-Carlo-Simulation 11

2.4 Rolling Horizon Evolutionary Algorithm 14

2.4.1 Rolling Horizon 14

2.4.2 Evolutionärer Algorithmus 15

2.5 Match Point AI 19

2.5.1 Tennisregeln 19

2.5.2 Framework 20

2.5.3 Bots 22

3 Methodik **25**
3.1 Software 25
3.2 Bots 25
3.2.1 Random-Bot 25
3.2.2 MCSim-Bot 26
3.2.3 RHEA-Bot 27
3.3 Experimente 29
3.3.1 Hyperparameter Optimierung 30
3.3.2 Turnier zum Vergleich von Monte-Carlo-Simulation und RHEA 30

4 Ergebnisse **33**
4.1 Hyperparameter Optimierung von RHEA 33
4.2 Vergleich Monte-Carlo-Simulation und RHEA 40
4.3 Analyse gefundener Spielstrategien 44

5 Fazit und Ausblick **49**

Literaturverzeichnis **51**

Abbildungsverzeichnis

2.1 Beispiel Markov Prozess 10
2.2 Verlauf Monte-Carlo-Simulation 13
2.3 Beispiel eines Rolling Horizon Ablaufs 15
2.4 Schema evolutionärer Algorithmus 16
2.5 Richtungs Codierung 20
2.6 Match Point AI Interface 21
2.7 Anteile von Spielern am Average-Stat-Bot 22

3.1 Darstellung Monte-Carlo-Simulation 26
3.2 Beispiel Individuum 27

4.1 Häufigkeitsverteilung der Ballwechsellängen 33
4.2 Vergleich Kombinationen gegen Random-Bot 34
4.3 Darstellung Gewinnrate zur durchschnittlichen Berechnungszeit 37
4.4 Konfidenzintervalle von Populationsgröße/Simulationsanzahl . . 40
4.5 Matches Zusammenfassung 44

Tabellenverzeichnis

4.1 Ergebnisse Hyperparameter Optimierung 36
4.2 Durchschnitte für Horizonte 37
4.3 Durchschnitte für Populationsgrößen 38
4.4 Durchschnitte für simulationsanzahl 38
4.5 Dreifaktorielle ANOVA der Parameter 39
4.6 Vergleich Gewinnrate - Random 40
4.7 Tukey HSD - Random 41
4.8 Vergleich Gewinnrate - Average 41
4.9 Tukey HSD - Average 42
4.10 Vergleich Gewinnrate Djokovic 42
4.11 Tukey HSD - Djokovic 43
4.12 Vergleich durchschnittliche Ballwechsellänge 43
4.13 Vergleich: durchschnittliche Wahrscheinlichkeit pro Richtung . . 45
4.14 MCSim - Nach Richtung des Gegners 45
4.15 RHEA - Nach Richtung des Gegners 45
4.16 Average - Nach Richtung des Gegners 45
4.17 MCSim - nach Spielzustand 46
4.18 RHEA - nach Spielzustand 47
4.19 Average - nach Spielzustand 47

Quellcodeverzeichnis

1 Monte-Carlo-Simulation 27
2 RHEA 29

Abkürzungsverzeichnis

EA = Evolutionärer Algorithmus
KNN = Künstliches neuronales Netz
MDP = Markov Decision Process
MCSim = Monte-Carlo-Simulation
MCTS = Monte-Carlo Tree Search
NTBEA = N-Tuple Bandit Evolutionary Algorithm
RHEA = Rolling Horizon Evolutionary Algorithm

1 Einleitung

"Champions keep playing until they get it right." -Billie Jean King[1]

Dies trifft nicht nur auf Tennisprofis zu. Auch für Maschinen ist es hilfreich, etwas so oft zu wiederholen, bis sie eine gute Lösung gefunden haben.

1.1 Motivation

Wenn man an Sportarten wie Tennis denkt, denkt man in erster Linie an die physische Komponente. Man denkt an das Training, das benötigt wird, um so schnell wie möglich den Ball rechtzeitig zu erreichen und diesen dann mit aller Kraft über das Netz zu schlagen. Doch dies ist nicht alles. Auch die taktische Komponente ist sehr wichtig, um einen Vorteil gegenüber dem Gegner zu haben. Damit sind viele Aspekte verbunden, wie zum Beispiel die eigene Positionierung. Aber auch Ballkontrolle spielt eine bedeutende Rolle. Diesbezüglich hat die USTA (United States Tennis Association) fünf Merkmale definiert, nach denen die Ballkontrolle beurteilt werden kann: Richtung, Geschwindigkeit, Weite, Höhe und Drehung [2]. In dieser Arbeit wird sich auf die Schlagrichtung konzentriert. Natürlich sind taktische Entscheidungen auch eng mit den körperlichen Fähigkeiten verbunden, damit es überhaupt möglich ist, die Taktik auszuführen [3]. Bei Profisportlern kann man allerdings davon ausgehen, dass das nötige Training bereits vorhanden ist und es nur noch eine Frage der Entscheidung ist, wohin man spielt. Doch woher weiß man, was die beste Möglichkeit ist? Dazu gibt es mehrere Wege. Die Erfahrung spielt zunächst eine große Rolle, da man mit der Zeit seine eigenen Stärken und Schwächen kennenlernt und auch das Verhalten der Gegner versteht. Außerdem besteht die Möglichkeit, vergangene Spiele von Gegnern oder von einem selbst zu betrachten, doch dies ist aufwendig und braucht viel Zeit. Eine weitere Alternative ist es, mithilfe von Algorithmen gute Strategien zu ermitteln

und diese dann zu analysieren. Dies spart Zeit und gibt einem die Möglichkeit, Zusammenhänge zu finden, die man sonst nicht beachtet hätte. In dieser Arbeit ist es das Ziel, zwei Algorithmen, Monte-Carlo-Simulation und RHEA, zu implementieren, die mit Simulationen versuchen, die beste Schlagrichtung in einer bestimmten Situation zu ermitteln, und das später gegen verschiedene Gegner zu testen, wie gut es funktioniert und ob sie brauchbare Strategien entwickeln.

1.2 Forschungsfragen

- **Frage 1**: Was sind die optimalen Parameter für den RHEA-Algorithmus für Match Point AI?

  Der RHEA-Algorithmus nutzt verschiedene Parameter, die zum einen Auswirkungen auf die Qualität des Ergebnisses haben als auch auf die Dauer der Berechnung haben können. Diese Parameter sind die Populationsgröße, Simulationsanzahl, Individuumlänge, Crossover-Rate und Mutationsrate. Während Crossover-Rate und Mutationsrate festgelegt sind, um die Menge der Experimente gering zu halten, da die Experimentenmenge exponential zu den möglichen Parametern ist, wird hier nur die Kombination aus den anderen drei Parametern untersucht. Dabei wird hauptsächlich auf die Siegrate der Kombinationen geachtet, aber es wird auch auf die zeitliche Effizienz geschaut. Die beste Kombination wird dann für die weiteren Experimente verwendet.

  Hypothese 0: Es existieren signifikante Unterschiede bezüglich der Gewinnrate und der zeitlichen Effizienz zwischen den Parameterkombinationen.

  Hypothese 1: Es existieren keine signifikanten Unterschiede zwischen den Kombinationen.

- **Frage 2**: Wie schneiden die beiden Entscheidungsalgorithmen, Monte-Carlo-Simulation und RHEA, gegen verschiedene Gegner ab?

  Das Framework bietet zwei verschiedene Bots unterschiedlicher Qualität gegen die gespielt werden kann. Einmal Average-Stat-Bot, der die durchschnittlichen Werte aller Spieler beschreibt, und den Novak-Djokovic-Bot, welche die Werte des besten Spielers aus der Datenmenge beschreibt.

Zusätzlich wird ein Random-Bot implementiert. Beide Entscheidungsalgorithmen sollen gegen diese drei Bots spielen und es wird deren Qualität bezüglich verschiedener Merkmale verglichen. Diese Merkmale sind die Siegrate für Punkte und die Anzahl der Schläge. Zusätzlich sollen beide Bots nochmal gegeneinander spielen.

Hypothese 0: Es gibt signifikante Unterschiede in der Siegrate zwischen den Bots.

Hypothese 1: Es gibt keine signifikanten Unterschiede in der Siegrate zwischen den Bots.

- **Frage 3**: Können die beiden Algorithmen eine Spielstrategie gegen den Novak-Djokovic-Bot entwickeln?

  Die getroffenen Entscheidungen während den Matches lassen sich statistisch auswerten, wobei dies eine Strategie des Bots widerspiegelt. Es soll getestet werden, ob sich die Strategien beider Algorithmen voneinander unterscheiden und ob Gemeinsamkeiten oder Unterschiede zum Durchschnittsspieler bestehen.

  Hypothese 0: Beide Algorithmen entwickeln die gleiche Strategie.

  Hypothese 1: Die Algorithmen entwickeln verschiedene Strategien.

1.3 Struktur

Diese Arbeit ist in fünf Kapitel unterteilt. Nach dem ersten Kapitel, welches die Motivation, die Untersuchungsfragen und diese Zusammenfassung umfasst, kommt ein Überblick über den aktuellen Stand der Forschung. Dort wird zunächst auf die grundsätzlichen Definitionen von Entscheidungen und Algorithmen eingegangen. Danach wird auf maschinelles Lernen und damit auf verschiedene Beispiele und Einsatzmöglichkeiten dessen eingegangen, um alternative Varianten zur Entscheidungsfindung kennenzulernen. Anschließend werden die genutzten Ansätze, Monte-Carlo-Simulation und RHEA, erklärt und auch hier bereits abgeschlossene Studien erläutert. Da als Testumgebung das Match Point AI Framework genutzt wird, wird dann noch dies erklärt und die bereits vorhandenen Bots vorgestellt. Im dritten Kapitel wird die Methodik vorgestellt. Dabei werden die neu erstellten Bots genau erklärt und wie die Algorithmen funktionieren. Dann werden die Experimente vorgestellt,

mit denen die Untersuchungsfragen beantwortet werden sollen. Zum Schluss wird noch auf die verwendete Software eingegangen. Im vierten Kapitel werden die Ergebnisse aus den Experimenten untersucht und es wird auf die Untersuchungsfragen und die Hypothesen eingegangen. Im abschließenden fünften Kapitel wird die Arbeit nochmal zusammengefasst, welche Erkenntnisse man gewonnen hat und inwiefern man die Arbeit in der Zukunft erweitern kann und welche Möglichkeiten für weitere Studien in diesem Themengebiet sich ergeben.

2 Stand der Forschung

In diesem Kapitel wird auf den aktuellen Stand der Forschung geblickt. Dabei wird auf die Definitionen und Beispiele für Entscheidungsalgorithmen, Markov-Prozesse, Monte-Carlo-Simulation und RHEA eingegangen. Weiterhin wird das verwendete Framework Match Point AI erklärt.

2.1 Entscheidungsalgorithmen

Ein Entscheidungsalgorithmus ist ein sehr oberflächlicher Begriff, da darunter jeglicher Algorithmus fällt, bei dem eine Entscheidung getroffen wird. Entsprechend ist das Themengebiet und die verschiedenen Anwendungen und Methoden breit gefächert. In dieser Arbeit werden einige Beispiele an Entscheidungsalgorithmen gegeben, wovon zwei, Monte-Carlo-Simulation und Rolling Horizon Evolutionary Algorithm, kurz RHEA, implementiert und miteinander verglichen werden.

Vorher werden wichtige Begriffe definiert und es wird auf einige Beispiele eingegangen, die die vielfältige Anwendung solcher Algorithmen, hier konkret im Bereich der Sport- und Spielanalyse und Simulation, verdeutlichen.

2.1.1 Entscheidung

Eine Entscheidung ist die Auswahl eines Ergebnisses aus mehreren Optionen. Dabei zielt man meistens darauf ab, die bestmögliche Entscheidung zu treffen. Dieser Prozess kann recht simpel sein oder sehr kompliziert, abhängig von der Menge der Optionen, der Anzahl der Kriterien, nach denen man entscheidet, und der Perspektive, von der aus man die Kriterien betrachtet.

- **Auswahl:** Weiterhin gibt es verschiedene Arten von Entscheidungen. So kann man eine Auswahl aus einer größeren Menge an Alternativen

wählen, wie es in dieser Arbeit der Fall ist in dem man sich für eine von drei möglichen Spielrichtungen entscheidet.

- **Akzeptanz:** Außerdem kann man sich zwischen Akzeptanz und Verweigerung entscheiden, zum Beispiel, ob man einen bestimmten Spieler in seinem Team spielen lässt oder nicht.
- **Evaluation:** bedeutet, dass man den Wert von etwas bestimmt, um eine bereits entschiedene Handlung zu unterstützen. Wie wenn man bei Sportwetten auf ein bestimmtes Team setzen will und sich entscheiden muss, wie viel man bereit ist einzusetzen.
- **Konstruktion:** Hier ist das Ziel die bestmögliche Lösung für ein Problem zu entwickeln unter Berücksichtigung der bekannten Einschränkungen. Zum Beispiel in dem man eine gute Spielstrategie entwickelt [4].

Wie gesagt, ist das Hauptziel der Bots, die bestmögliche Schlagrichtung auszuwählen, aber es werden auch andere Arten von Entscheidungen getroffen. So werden die Bots auch evaluiert, um ihre Stärken und Schwächen einschätzen zu können, und sie können auch genutzt werden, um eine Strategie gegen bestimmte Spieler, in diesem Fall Novak Djokovic, zu entwickeln.

2.1.2 Algorithmus

Ein Algorithmus ist eine eindeutige Prozedur zur Lösung eines Problems. Diese bestehen üblicherweise aus einer Menge an Befehlen, die nacheinander bearbeitet werden und dabei eine Reihe an Eingabewerten annehmen und Ausgabewerte wieder zurückgeben [5]. Die Art der Probleme, die gelöst werden können, ist vielfältig, darunter befinden sich auch Entscheidungsprobleme. Um so ein Problem zu lösen, können die Parameter fest beschrieben werden, sodass bei der gleichen Eingabe auch immer das gleiche Ergebnis herauskommt. Es ist auch möglich, dass sich die Parameter abhängig von dem aktuellen Status anpassen können und sich in die Richtung entwickeln, um das bestmögliche Ergebnis zu finden. Dies geschieht unter anderem im Bereich des maschinellen Lernens.

2.1.3 Maschinelles Lernen

Es gibt verschiedene Wege, um maschinelles Lernen zu definieren. In einer Studie des Europäischen Parlaments zu Entscheidungsalgorithmen wird es als

eine KI-Komponente beschrieben, die basierend auf einer großen Datenmenge selbstständig lernen kann. Dabei beobachtet das Programm seinen aktuellen Zustand oder lernt basierend auf vorbereiteten Trainingsdaten, um Gemeinsamkeiten zu entdecken, Vorhersagen zu treffen oder eben um Entscheidungen zu fällen. Der Algorithmus erstellt basierend darauf einen neuen Algorithmus, den man als Modell bezeichnet.

Dabei wird das maschinelle Lernen üblicherweise in drei Klassen unterteilt:

1. Supervised Learning: Die Daten sind bereits bestimmten Kategorien zugeordnet. Das Modell lernt dadurch die Gemeinsamkeiten der Kategorien und kann neue Daten, die nicht zum Trainingssatz gehörten, den Kategorien zuordnen.

2. Unsupervised Learning: Im Gegensatz zum supervised Learning sind hier die Datensätze nicht bereits kategorisiert. Ziel ist es aus den Trainingsdaten Gemeinsamkeiten zu identifizieren und so die Kategorien selbst zu ermitteln.

3. Reinforcement Learning: Hierbei wird durch das Feedback aus der Umgebung gelernt. Es wird eine Aktion innerhalb der Umgebung gewählt und darauf geachtet, ob dies ein Erfolg oder ein Fehlschlag war. Ziel ist es das positive Feedback zu maximieren [5].

Innerhalb dieser Arbeit wird sich auf Reinforcement Learning konzentriert, da das Framework die Möglichkeit bietet, Feedback in Form von Punkten zu geben oder ob man ein Spiel gewinnt oder nicht.

Beispiele für maschinelles Lernen im Sport

Es ist keine Neuheit, dass man maschinelles Lernen auch im Sport- und Spielbereich einsetzt. Denn während es natürlich auch um Stärke, Reaktionsgeschwindigkeit, Talent und Intuition geht, kann man oft konkrete Parameter ermitteln, die sich so quantifizieren lassen, dass man mit ihnen zum Beispiel die Siegwahrscheinlichkeit eines Spielzugs bestimmen kann. So bietet es sich an, maschinelles Lernen zu nutzen, um damit zu trainieren, Spielstrategien gegen bestimmte Gegner zu identifizieren oder schlicht in Testumgebungen zu benutzen, um die Grenzen und Möglichkeiten von intelligenten Algorithmen zu analysieren.

Eines der am beliebtesten Spiele, um die Leistungsfähigkeit von intelligenten Algorithmen zu testen, ist Schach. Bereits 1997 hatte das Schachprogramm Deep Blue gegen den Schachchampion Garry Kasparov gewonnen. Damals wurde allerdings hauptsächlich durch Brute-Force-Methoden die Spielzüge berechnet. 2017 hatte das Schachprogramm AlphaZero gegen Stockfish, eines der stärksten Schachprogramme überhaupt, gewonnen. Dabei hatte es im Gegensatz zu seinen Vorgängern nicht dadurch gewonnen, dass es so viele Spielzüge wie möglich berechnet hat, sondern indem es 4 Stunden lang durch Reinforcement Learning das Spiel gelernt hat. Das ist insofern beeindruckend, dass Schach ein so komplexes Spiel ist, dass es bis heute nicht möglich ist, alle möglichen Spielzüge zu berechnen, geschweige denn zu speichern. Dass es nur durch eine so relativ kurze Lernsession möglich ist, gegen eines der besten Schachprogramme zu gewinnen, zeigt das Potenzial, das maschinelles Lernen mit sich bringt [6].

Schach ist allerdings ein vorrangig strategisches Spiel. Die hohe Komplexität wird hierbei durch die immens hohe Anzahl an möglichen Zügen verursacht. Anders sieht es bei Sportarten aus, bei denen nicht nur strategisches Denken, sondern auch körperliches Geschick zum Einsatz kommt. Eine solche Sportart wäre zum Beispiel Tischtennis, bei dem man nicht nur entscheiden muss, wie man spielt, sondern man auch die Fähigkeiten haben muss, um die Schläge effektiv auszunutzen. Um dort die bestmöglichen Leistungen zu erzielen, bedarf es eines Trainings, das sowohl auf die Stärken und Schwächen des Spielers als auch auf die Anforderungen der Sportart zugeschnitten ist. Mit einem solchen Problem beschäftigte sich eine Studie aus 2021, wobei hier ein Entscheidungsbaum berechnet wurde, welcher Spieler anhand verschiedener Attribute klassifizieren sollte. Ein Entscheidungsbaum ist dabei eine Baumstruktur, wo jeder Knoten ein Attribut darstellt, welches abgefragt wird. Abhängig vom Ergebnis wird zum nächsten Knoten übergegangen. Die Blattknoten geben dann die Klassifizierung. Die Ergebnisse der Studie zeigen, dass ein solcher Entscheidungsbaum sehr akkurat ist und die Ergebnisse, die er liefert, objektiv und einfach zu verstehen sind. Dies kann Trainern die Möglichkeit geben, die Stärken des eigenen, aber auch des gegnerischen Spielers zu analysieren und somit gute Entscheidungen bei der Erstellung des Trainingsplans zu treffen [7].

Wenn man an Sport denkt, denkt man nicht nur an die Ergebnisse einzelner Spiele, sondern auch an Turniere und somit an Platzierungen. Diese bieten zum einen die Möglichkeit für Spieler und Trainer zu planen, gegen welchen Gegner sie spielen werden. Zum anderen spielen sie eine wichtige Rolle, wenn es um Sportwetten geht, zum Beispiel bei der Fußballweltmeisterschaft. Da-

bei werden die Entscheidungen meist auf Basis der früheren Erfolge getroffen. Dies bildet auch eine gute Grundlage für maschinelles Lernen, um zukünftige Platzierungen vorherzusagen. Diesbezüglich wurde auch 2014 ein Experiment beschrieben, bei dem mehrere intelligente Algorithmen miteinander verglichen wurden, unter anderem künstliche neuronale Netze, kurz KNN. Ein KNN besteht dabei aus mehreren Ebenen, welche wiederum aus Knoten bestehen. Die Knoten einer Ebene sind nicht miteinander verbunden, aber mit denen aus den Ebenen darüber und darunter. Die erste Ebene nimmt die Eingabedaten entgegen und die letzte Ebene gibt die Ausgabewerte aus. Dazwischen sind sie mit einer Summenfunktion und einer Aktivierungsfunktion verbunden. Dabei werden die Eingaben über die Summenfunktion gewichtet zusammengerechnet, während die Aktivierungsfunktion darauf basierend bestimmt, ob der Knoten eine Ausgabe liefert oder nicht. Das KNN lernt auf Basis vergangener Fußballspiele und soll zum einen die Ergebnisse zukünftiger Spiele bestimmen, als auch auf diese Ergebnisse wetten. Dabei zeigte sich, dass die KNN gute Vorhersagen trafen, mit einer durchschnittlichen 50% Richtigkeit und somit der zweitbeste Algorithmus nach MHITS waren und mit dem meisten Einkommen ein gutes Ergebnis bei den Wetten erzielten [8].

Diese Beispiele geben nur einen kleinen Überblick über die verschiedenen Anwendungsmöglichkeiten von Entscheidungsalgorithmen im Sport.

2.2 Markov Decision Process

Nach dem Buch „Reinforcement Learning: State-of-the-Art“ von Wiering und Van Otterlo wird ein Markov Decision Process, kurz MDP, wie folgt definiert:

A Markov decision Process is a Tuple <S,A,T,R> in which S is a finite set of states, A a finite set of actions, T a transition function defined as T: $S \times A \times S \rightarrow$ [0,1] and R a reward function defined as R: $S \times A \times S \rightarrow$ R [9].

State: Ein State (Zustand) ist eine Beschreibung aller wichtigen Eigenschaften eines Systems. Ein Zustand ist ein Element der Menge S.

Action: Eine Action (Aktion) ist ein Element der Menge A. Damit kann der Zustand des Systems kontrolliert werden. Alle Aktionen, die auf einen Zustand angewendet werden können, werden beschrieben mit A(s). Dies ist eine Teilmenge der Gesamtmenge A.

Transition-Function: Beim Ausführen einer Aktion a von einem Zustand s, geht das System in den Zustand s' über. Dies wird durch die Transition-Function T(s,a,s') beschrieben. Dabei gibt es die Wahrscheinlichkeit an, dass das System in den Zustand s' landet, nachdem a auf den Zustand s angewendet wurde. So gilt zum Beispiel, dass bei T(s,a,s') = 0 der Übergang von s zu s' durch a nicht möglich ist.

Reward-Function: Die Funktion beschreibt die Belohnung, die man in einem Zustand bekommen kann. Dabei gilt R: S $\rightarrow$ |R. Des Weiteren gibt es zwei weitere Definitionen, R: $S \times A \rightarrow$ |R und R: $S \times A \times S \rightarrow$ |R. Die erste gibt eine Belohnung durch das Ausführen einer Aktion auf einen Zustand und die zweite beim Übergang von einem Zustand in einen anderen.

Das nachfolgende Beispiel stellt einen Teil des Markov-Prozesses aus einem Tennisspiel von Match Point AI dar.

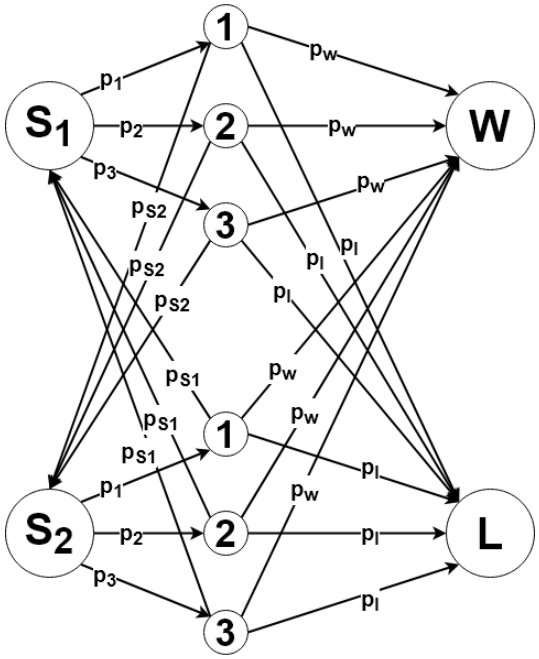

Abbildung 2.1: Beispiel Markov Prozess

Dieser Markov-Prozess stellt einen Ausschnitt aus den Schlagwahrscheinlichkeiten von Match-Point-AI dar. Dabei gibt es verschiedene Zustände, abhängig von der Seite, auf der man spielt, und welcher Spieler den Aufschlag hatte. Dabei wird nochmal zwischen Aufschlägen und normalen Schlägen unterschieden.

In diesem Beispiel stellen die Zustände S_1 und S_2 zwei Spieler bei einem normalen Schlagabtausch dar, nachdem der Aufschlag und die Erwiderung gespielt wurden. Für jeden Zustand wird mit einer Wahrscheinlichkeit von p_1, p_2 und p_3 eine Richtung gewählt, in die gespielt wird. Dabei sind die Schlagrichtungen die auszuwählenden Aktionen. Wurde sich für eine Aktion entschieden, so landet man mit der Wahrscheinlichkeit pS_1/pS_2, p_W und p_L im nächsten Zustand. Dies ist eine Transaktion. So ist es möglich, aus dem Zustand S_1 über die Aktion 1 in den Zustand S_2 zu wechseln, also T(S_1,1, S_2). Erreicht man die Zustände W oder L, wobei W für Gewinnzustand und L für Verlustzustand steht, so sind dies Endzustände und der Ballwechsel ist beendet. Abhängig vom aktuellen Zeitpunkt im Spiel gibt es für einen Sieg eine bestimmte Anzahl von Punkten und für eine Niederlage keine Punkte. Ein Beispiel für so eine Abfolge wäre:

$$S_1 \rightarrow 1 \rightarrow S_2 \rightarrow 3 \rightarrow S_1 \rightarrow 2 \rightarrow W = \text{Reward in Punkten}$$

Wie viel man genau bekommen kann, wird später im Abschnitt zu Match Point AI genauer erklärt.

2.3 Monte-Carlo-Simulation

Mit zwei Würfeln kann man jede Zahl zwischen zwei und zwölf erhalten, doch welche Zahl man bekommt, kann man nicht garantiert vorhersagen, da es zufällig ist. Oft kann man das Ergebnis nicht vorher bestimmen, da es zufällig ist oder das System dahinter unbekannt ist und daher zufällig erscheint. Dennoch kann man herausfinden, welches Ergebnis am wahrscheinlichsten ist. Wenn man nur oft genug würfelt, findet man heraus, dass man die Zahl sieben am häufigsten bekommt. Somit lässt sich eine Wahrscheinlichkeitsverteilung aller Zahlen ermitteln. Bei diesem Beispiel könnte man dies nun vorher bestimmen, da sieben von allen Zahlen aus den meisten Kombinationen besteht. Doch oft sind Systeme zu komplex, um die Wahrscheinlichkeiten rein rechnerisch zu ermitteln. Und dieser Ansatz, aus einem zufälligen System durch eine Vielzahl von Versuchen ein konkretes Ergebnis zu ermitteln, bildet die Grundlage für die Monte-Carlo-Simulation.

Eines der ersten bekannten Anwendungen dieses Ansatzes war das Experiment „Buffon's Needle“ vom französischen Wissenschaftler Georges Louis LeClerc, Comte de Buffon (1707-1788). Dabei konnte er beweisen, dass man näherungsweise die Kreiszahl Pi bestimmen kann, indem man viele Nadeln auf einen

linierten Untergrund warf, wobei der Abstand der Linien der Länge der Nadeln entsprach. Die Wahrscheinlichkeit, dass eine Nadel auf einer Linie landet, ist $2/\pi$ [10].

Im Verlauf der Zeit wurden viele verschiedene Studien auf Grundlage dieser Idee durchgeführt, doch die Monte-Carlo-Simulation in der heutigen Form wurde erst in den 1940er Jahren von unter anderem Stanley Ulam entwickelt. Der Name wiederum kommt daher, dass Stanleys Onkel des Öfteren Geld von Verwandten auslieh, um im Monte-Carlo-Casino in Monaco Glücksspiele zu spielen [11].

Diese Methode lässt sich normalerweise in vier Schritte unterteilen:

1. **Generierung des statischen Modells**: Zuerst muss das Modell bestimmt werden, indem die Berechnungen stattfinden. Dieses Modell sollte so gut wie möglich die reale Umgebung darstellen. Dies kann man ermöglichen, indem man die bereits bekannte Daten analysiert oder mit ausgewählten Eingabe- und Ausgabevariablen deren mathematischen Beziehungen bestimmt.

2. **Wahrscheinlichkeitsverteilung der Eingabevariablen**: Dies ist der zufällige Teil des Modells. Basierend auf historischen Daten werden die Verteilungen der einzelnen Variablen bestimmt.

3. **Generierung zufälliger Eingaben**: Basierend auf diesen Verteilungen werden nun zufällige Werte berechnet. Für einen Eingabesatz bekommt jede Variable einen Zufallswert. Diese Eingabe wird auf das statische Modell angewendet und es wird ein Ausgabesatz berechnet. Dies wird oft wiederholt und immer wieder wird eine neue zufällige Eingabe bestimmt. Die Ausgabesätze werden dabei gesammelt.

4. **Analyse der Ergebnisse**: Wenn man genug Daten gesammelt hat, werden diese statistisch aufbereitet und analysiert. Das Ergebnis kann dann als Grundlage für die Entscheidung genommen werden.(Raychaudhuri, 2008)

Monte-Carlo-Simulationen werden in vielen verschiedenen Bereichen genutzt. Es findet Anwendung in Bereichen wie Biologie, Medizin [12], Quantenphysik [13], Ingenieurwesen [14] oder Management [15]. Aber auch im Sport- und Spielbereich findet man Studien zur Nutzung von Monte-Carlo-Simulationen. So kann dieser Ansatz effektiv in Spielen wie Skat genutzt werden, um schnell

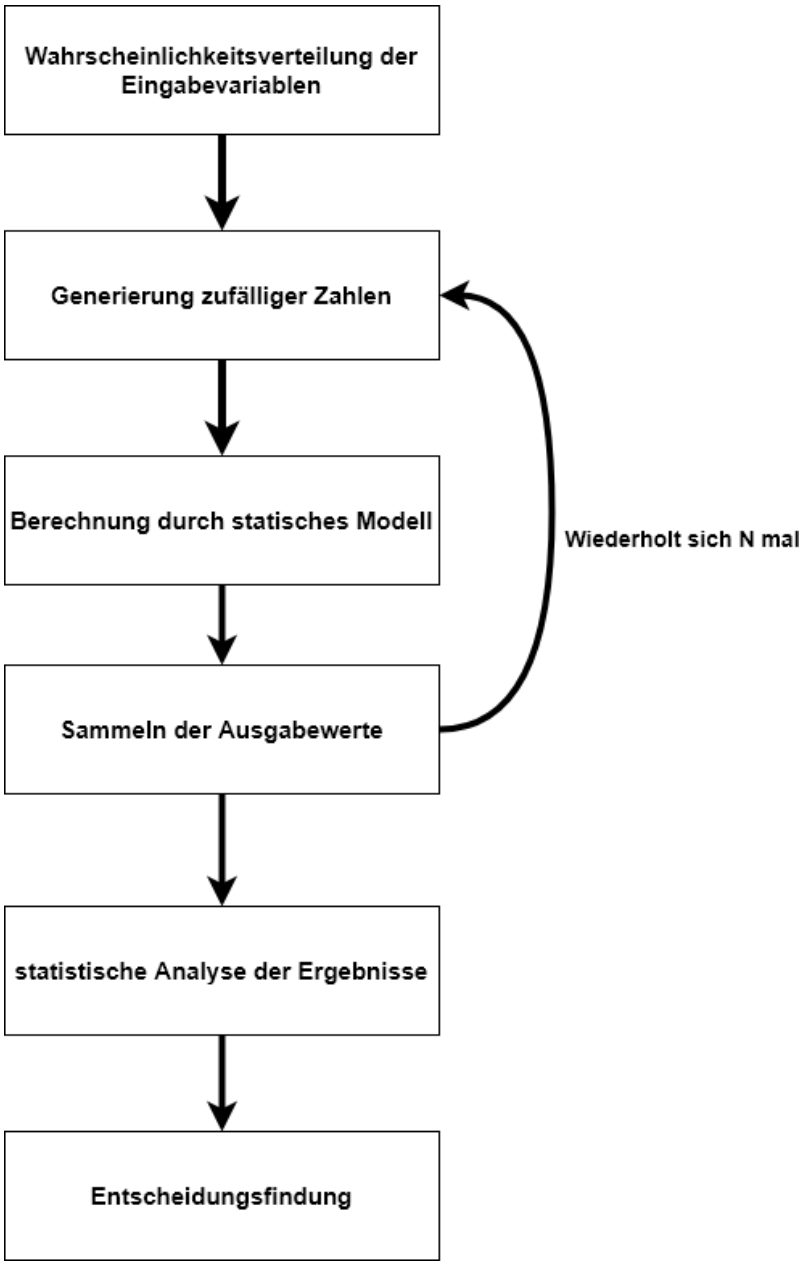

Abbildung 2.2: Verlauf Monte-Carlo-Simulation

gute Ergebnisse zu simulieren [16]. Es wurde in einer anderen Studie auch dazu genutzt, um die Spannungskurven von verschiedenen Sportarten zu analysieren. So wurde ein Simulationsmodell, welches die Punktesysteme von Basketball, Tennis und Tischtennis darstellt, erstellt. Darauf wurden verschiedene

Spiele abhängig von den Gewinnwahrscheinlichkeiten der Spieler/Teams simuliert und verschiedene Variablen gemessen, die im Zusammenhang mit Spannungsmomenten stehen, wie die Länge des Spiels oder die Zahl und Position von Wendepunkten. So hat man unter anderem herausgefunden, dass Basketball mehr ans Fernsehen angepasst ist, während Tennis und Tischtennis unberechenbarer sind. Dies bildet eine gute Grundlage, um Spannungsbögen zu analysieren, auch außerhalb vom Sport [15]. Die Monte-Carlo-Simulation kann auch genutzt werden, um die Gewinner eines Tennismatches vorherzusagen. In einer Studie wurde ein Modell entwickelt, welches die Wahrscheinlichkeit nutzt, ob ein Spieler die Punkte erzielt oder der Gegner. Damit konnten dann ganze Spiele simuliert werden. Zusätzlich wurden zwei Modelle entwickelt, in denen Aspekte wie Müdigkeit oder die Siegwahrscheinlichkeit eines Spielers, nachdem dieser den Lauf seines Gegners beendet hat, eingearbeitet wurden. Es hat sich ergeben, dass die Simulationen, auch wenn sie sich zu 70% sicher waren, nur eine Korrektheit von 67% hatten und dass das Einarbeiten weiterer Variablen nur einen geringen Effekt hatte. Allerdings übersteigen auch andere Modelle, wie beim maschinellen Lernen, nicht die 70% Korrektheit, wodurch sich zeigt, dass Monte-Carlo-Simulationen auch eine akzeptable Variante bei Vorhersagemodellen sind [17].

2.4 Rolling Horizon Evolutionary Algorithm

Der Rolling Horizon Evolutionary Algorithm, oder kurz RHEA, setzt sich aus dem Ansatz des rollierenden Horizonts und einem evolutionären Algorithmus zusammen.

2.4.1 Rolling Horizon

Bei vielen Problemen im ökonomischen, mathematischen und vielen anderen Bereichen muss man zukünftige Variablen in die aktuelle Planung mit einbeziehen, um ein vernünftiges Ergebnis zu erzielen. Dabei wird bis zu einem bestimmten Zeitpunkt, dem Planungshorizont, eingeplant. Dabei wird der Zeitraum in einzelne Zeitabschnitte unterteilt, für die jeweils Entscheidungen getroffen werden müssen. Ein Ansatz wäre es, einen Plan für den gesamten Zeitraum zu erstellen, doch stellt dies ein großes Problem dar, da es zu viele Variablen gibt, um es in einer akzeptablen Zeit zu lösen. Um dies zu umgehen, kann man die Planung in mehrere Iterationen unterteilen, die nur einen

kleineren Planungshorizont besitzen. Ist ein Zeitabschnitt fertig verplant, wird der Horizont um einen Zeitabschnitt in die Zukunft verschoben.

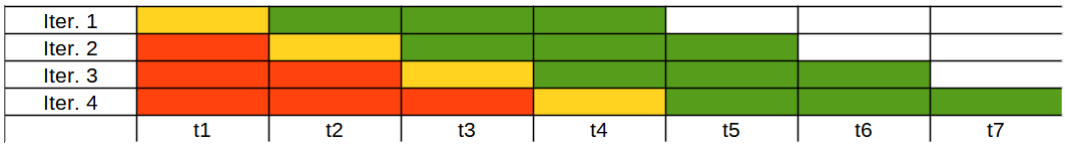

Abbildung 2.3: Beispiel eines Rolling Horizon Ablaufs

In diesem Beispiel wird pro Iteration für drei Zeitabschnitte in der Zukunft geplant. Gelb markiert den aktuellen Abschnitt und Grün die, die für die Planung mit einbezogen werden. Ist ein Abschnitt abgearbeitet, wird dieser rot markiert. Dadurch wird effektiv ein Plan erstellt, ohne zu großen Berechnungsaufwand und unter Bezug näherer zukünftiger Variablen [18].

2.4.2 Evolutionärer Algorithmus

Ein evolutionärer Algorithmus, kurz EA, ist ein Ansatz zur Lösung von Optimierungsproblemen, wobei sich an der natürlichen Evolution orientiert wird. Dementsprechend ist es das Ziel, aus einer großen Menge verschiedener Lösungen durch stetige Ausmusterung und Veränderung eine gute Lösung zu finden. EAs lassen sich also vor allem durch drei Merkmale charakterisieren.

1. Population-basiert: Die Menge an Lösungen wird als Population bezeichnet. Ähnlich wie in der natürlichen Evolution, verändert sich diese Population und passt sich mit der Zeit an das Problem an, womit es somit aus möglichst guten Lösungen besteht. Ein Bestandteil, welches eine solche Lösung repräsentiert wird als Chromosom bezeichnet. Es ist so kodiert, dass das Chromosom wiederum in Gene unterteilt werden kann, die dann verändert werden können. Die Population zu einem Zeitpunkt ist die Generation.

2. Fitness-orientiert: Um die Qualität, hier Fitness genannt, eines Chromosoms herauszufinden, muss eine Funktion bestimmt werden, die dies berechnet. Ziel dabei ist es die Lösung mit der höchsten Fitness zu finden und weiter zu verwenden.

3. Variation-getrieben: Lösungen können sehr komplex sein, was sich auch in der Kodierung der Chromosomen widerspiegelt. Daher ist es möglich,

dass das aktuell beste Individuum nicht das beste aller möglichen Lösungen ist. Um dies zu umgehen, werden die Chromosomen mit jeder Generation leicht verändert um die Möglichkeit zu eröffnen, andere womöglich bessere Lösungen zu finden [19].

Innerhalb einer Generation durchläuft die Population mehrere Schritte, um sich besser an das Problem anzupassen:

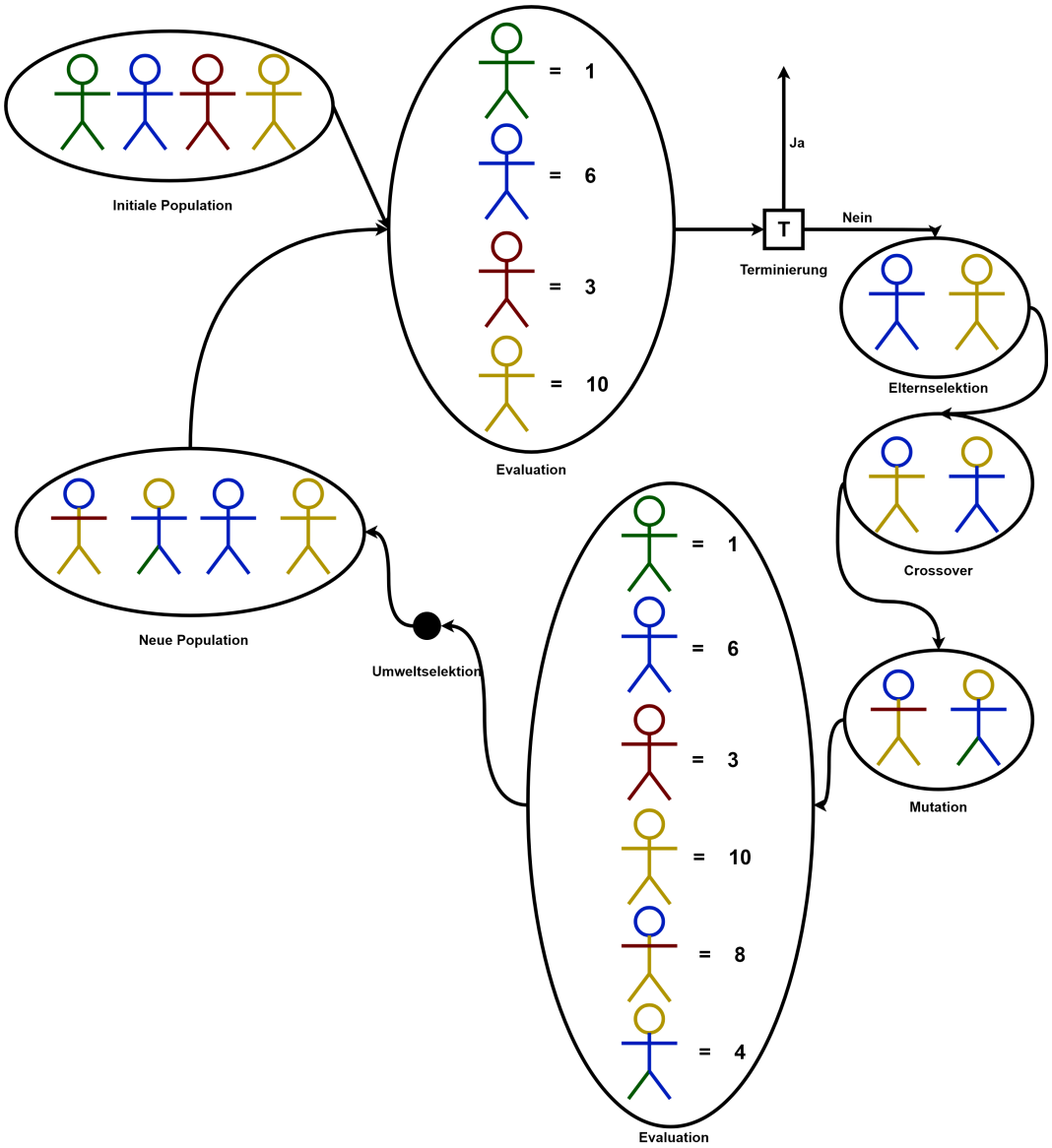

Abbildung 2.4: Schema evolutionärer Algorithmus

- Initialisierung: Zu beginn eines EAs benötigt es eine Ausgangspopulation aus der dann alle anderen Lösungen resultieren. Dazu werden P Individuen zufällig generiert, die dann in die erste Generation aufgenommen werden. P ist dabei die Populationsgröße
- Evaluation: Hier wird die Qualität jedes Individuums in der Population ermittelt. Dazu wird die Fitnessfunktion verwendet. Wie diese aussieht ist vom Problem, auf dem optimiert wird, und der Kodierung der Chromosome abhängig
- Selektion: Im nächsten Schritt werden basierend auf dieser Fitness Individuen rausgesucht und miteinander gepaart um die nächste Kindgeneration zu erstellen. Es gibt verschiedene Möglichkeiten, wie diese ausgewählt werden. Zum einen gibt es die Roulette-Selektion. Hierbei wird der prozentuale Anteil eines Individuums an der Gesamtfitness berechnet. Dies entspricht der Wahrscheinlichkeit, mit der das Individuum ausgewählt wird. Bei der Turnierselektion werden mehrere Individuen zufällig ausgewählt, von denen die besten ausgewählt werden. Bei der Ranglisten-Selektion ist die Wahrscheinlichkeit abhängig von der Ranglisten Platzierung der Individuen. Neben diesen gibt es noch einige weitere Selektionsmethoden.
- Crossover: Nachdem die die Elterngeneration ausgewählt wurde, werden diese gepaart. Dazu wird zuerst die Crossover-Operation mit einer Wahrscheinlichkeit enstprechend der Crossover-Rate ausgeführt. Auch hier gibt es verschiedene Möglichkeiten. Generell werden hier Gene zwischen den Chromosomen ausgetauscht. Beim Ein-Punkt-Crossover wird ein zufälliger Punkt ausgewählt an denen die Chromosome in zwei Teile geteilt und dann ausgetauscht werden. Beim Zwei-Punkt-Crossover wird es in drei Teile geteilt. Dies kann man so weiter führen, wie es Schnittstellen gibt. Beim Uniform-Crossover gibt es für jedes Gen die Wahrscheinlichkeit von 50%, dass diese ausgetauscht werden.
- Mutation: Nach dem Crossover durchlaufen die Individuen eine Mutation. Für jedes Gen besteht eine Wahrscheinlichkeit gleich der Mutationsrate, dass es sich zu einem anderen gültigen Wert ändert.
- Neue Population: Nachdem sich eine Kindergeneration gebildet hat, wird eine neue Population ausgewählt. Dabei kann die neue Population nur aus den Kindern bestehen oder aus einer Kombination aus Kindern und Eltern.

- Terminierung: Nachdem eine neue Population gewählt wurde, wird der Prozess erneut durchlaufen, bis eine Terminierungsbedingung erreicht ist. Dies kann dadurch bestimmt werden, indem man eine maximale Anzahl an Generationen definiert. Man kann auch eine Zeit festlegen, sodass der Algorithmus beendet wird, sobald die Zeit abgelaufen ist. Es ist auch möglich die Bedingung an einer bestimmten Fitness fest zu machen. Erreicht ein Individuum diese Fitness wird es beendet. Oder man misst die Differenz zwischen der Fitness der einzelnen Generationen. Verbessert sich die Fitness nicht mehr nennenswert wird es terminiert [20][21].

Nachdem man den Algorithmus terminiert hat, ist die Population, die man am Ende hat, die Lösungsmenge. Daraus kann man dann eine Lösung auswählen, zum Beispiel indem man das Chromosom mit der besten Fitness nimmt.

Bei RHEA wird der Ansatz des Rolling Horizon mit einem evolutionären Algorithmus kombiniert. Dies bedeutet, dass jedes Individuum ein Plan ist, dessen Länge dem Planungshorizont H entspricht. Ziel ist es, zu jedem Zeitpunkt den besten Plan für die nahe Zukunft zu finden und dann den ersten Schritt der gefundenen Lösung auszuführen.

Evolutionäre Algorithmen können in vielen verschiedenen Bereichen eingesetzt werden, in denen es Optimierungsprobleme gibt. So kann es im Management, Ingenieurwesen oder im Bereich der künstlichen Intelligenz genutzt werden [22]. Es lassen sich auch im Sportbereich Anwendungsmöglichkeiten finden. 2019 wurde zum Beispiel ein EA genutzt, um mit dessen Hilfe die beste Aufstellung einer Basketballmannschaft herauszufinden. Dabei wurde eine Teamaufstellung als ein Vektor dargestellt, wobei die Spieler durch Zahlen repräsentiert wurden und ein Eintrag im Vektor die Position des Spielers darstellte. Die Fitness wurde durch den Wert aller Spieler, was sich durch deren Leistung in vergangenen Spielen bestimmen lässt, und deren Kosten berechnet. Ziel war es, ein Team mit einem hohen Wert und geringen Kosten zu finden. Zwar zeigt die Studie, dass der Algorithmus gute Ergebnisse liefert, allerdings gibt es viele Aspekte, die nicht beachtet werden, wie zum Beispiel die Verträge der Spieler. Deshalb kann die Arbeit eines Coaches bei der Entscheidungsfindung bezüglich der Teamaufstellung nicht komplett ersetzt werden, trotzdem bietet der Algorithmus eine gute Unterstützung, um solche Entscheidungen zu erleichtern [23]. Auch wurde ein evolutionärer Algorithmus im Zusammenhang mit Tennis genutzt. Dabei wurde ein Evaluationsmodell entwickelt, welches die Leistung von Spielern während eines Tennismatches berechnet. Um die opti-

malen Parameter für das Modell zu finden, wurde ein EA genutzt [24]. RHEA wurde hingegen noch gar nicht im Tennis oder generell im Sport genutzt. Die meisten Anwendungsbeispiele für RHEA finden sich im Computerspielbereich. So wurde 2013 der RHEA-Algorithmus mit einem MCTS-Algorithmus basierend auf dem Physical Travelling Salesman Problem verglichen. Dabei wurde gezeigt, dass RHEA genauso gute, mitunter sogar bessere Ergebnisse liefert als MCTS [25]. In einem anderen Beispiel von 2022 wurde RHEA auf 20 verschiedene Spiele angewendet, wobei es das Ziel war, mithilfe eines N-Tuple Bandit Evolutionary Algorithm(NTBEA) die Parameter von RHEA für jedes Spiel zu optimieren. Denn mit der Zeit und über verschiedene Papers hinweg wurden immer mehr Parameter zu RHEA hinzugefügt, wobei diese meist von den Wissenschaftlern ausgewählt wurden. Der Lösungsraum für die Parameter kann aber als ein Optimierungsproblem für sich verstanden werden. Dabei zeigt sich, dass durch den Einsatz von NTBEA die Qualität der Ergebnisse bei einigen Spielen verbessert hat und dass ähnliche Spiele auch ähnliche Parameterkombinationen bevorzugen [26].

2.5 Match Point AI

In diesem Abschnitt wird das Framework Match Point AI [27] beschrieben.

2.5.1 Tennisregeln

Match Point AI benutzt das normale Punktesystem, welches im Tennis üblich ist [28]. Dabei beginnt ein Schlagabtausch, welcher als Rally bezeichnet wird, mit dem Aufschlag eines Spielers und endet, wenn ein Spieler einen Punkt macht. Ein Punkt wird erzielt, indem man entweder einen Schlag gewinnt, also so spielt, dass der Gegner den Ball nicht erwischt, oder der Gegner einen Fehler macht, indem dieser ins Aus oder ins Netz spielt. In diesem Fall bekommt der Gegner einen Punkt. Macht man bei einem Aufschlag einen Fehler, dann kann man den Aufschlag wiederholen. Macht man dann wieder einen Fehler, dann geht der Punkt an den Gegner. Der Aufschlag wird abwechselnd von der linken und der rechten Seite auf die gegenüberliegende Seite ins Aufschlagsfeld gespielt. Der Gegner antwortet mit dem Rückschlag, auch Return genannt. Danach folgt ein normaler Schlagabtausch, bis ein Spieler einen Punkt macht. Die Punkte werden als 15, 30 und 40 gezählt. Macht ein Spieler den vierten

Punkt und hat somit zwei Punkte Vorsprung, gewinnt dieser ein Spiel. Haben beide Spieler drei Punkte, also 40:40, dann nennt man das Deuce und es wird so lange weitergespielt, bis ein Spieler zwei Punkte Vorsprung hat. Man muss mindestens 6 Spiele mit zwei Spielen Vorsprung gewinnen, um einen Satz zu gewinnen. Steht es 6 zu 6, dann nennt man das Tie-Break und das nächste Spiel ist entscheidend. Wer dies gewinnt, gewinnt den Satz. Wenn man zwei Sätze gewinnt, gewinnt man dann das Match.

2.5.2 Framework

Ob man ein Tennismatch gewinnt, hängt von verschiedenen Faktoren ab, die sich im Verlauf eines Matches ändern können. So kann sich die Müdigkeit der Spieler über den Verlauf des Spiels auf ihre Leistung auswirken. Es haben auch die Unterstützung der Fans oder der Stress kurz vor Ende eines Spiels, Sets oder Matches einen Einfluss auf den Zustand und die Konzentration der Spieler. Auch technische Aspekte wie die Art der Schläge, die Position der Spieler oder die Geschwindigkeit des Balls spielen eine Rolle. Dadurch ist Tennis ein sehr komplexes Spiel, weshalb es nahezu unmöglich ist, es akkurat zu simulieren. Aus diesem Grund und um eine gute Grundlage zu schaffen, mit der die Bots arbeiten können, konzentriert sich das Framework nur auf die Schlagrichtungen.

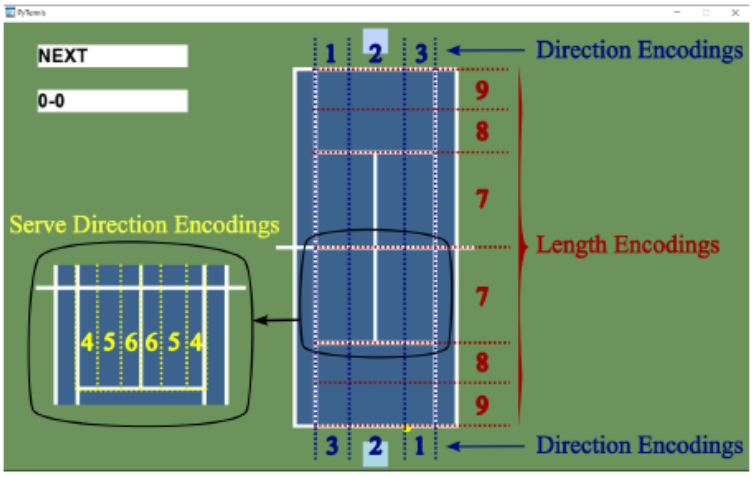

Abbildung 2.5: Richtungs Codierung [27]

Dazu wird das Spielfeld in drei Teile geteilt. Es wird jeweils zu 30 Prozent für die linke und rechte Seite und 40 Prozent für die Mitte geteilt. Des Weiteren wird das Aufschlagsfeld in drei Teile geteilt. Um das von den normalen Schlägen zu unterscheiden, wird dies extra kodiert. Dies entspricht der Kodierung aus dem Match Charting Project [29], welches auch als Grundlage für die Bots dient. Wenn ein Spieler am Zug ist, kann dieser sich zwischen den drei Schlagrichtungen 1,2 und 3 entscheiden, beziehungsweise 4, 5 und 6, wenn man aufschlägt. Normale Schläge bekommen zusätzlich noch eine Längenkodierung, welche allerdings nur zur Visualisierung dient. Nachdem eine Schlagrichtung ausgesucht wurde, wird bestimmt, ob der Schlag gewinnt, verliert oder weiter an den Gegner geht. Die Sieg- und Verlustwahrscheinlichkeiten sowie die Wahrscheinlichkeiten für die Richtungen sind stets vom gegenwärtigen Spielzustand abhängig. Dabei gibt es jeweils einen Zustand für die beiden Seiten des Aufschlags. Wenn der erste Aufschlag fehlschlägt, so gibt es für den zweiten Aufschlag zwei Zustände. Dann gibt es zwei Zustände, abhängig davon, ob man auf einen ersten oder auf einen zweiten Aufschlag reagiert. Weitere Schläge werden danach unterschieden, ob sie in dem Schlagabtausch einen ersten oder zweiten Aufschlag hatten oder ob sie darauf reagiert haben. Diese sind wiederum danach unterteilt, aus welcher Richtung der Ball kam. Die Daten dazu stammen aus dem Match Charting Project [29]. Das Interface zu Match

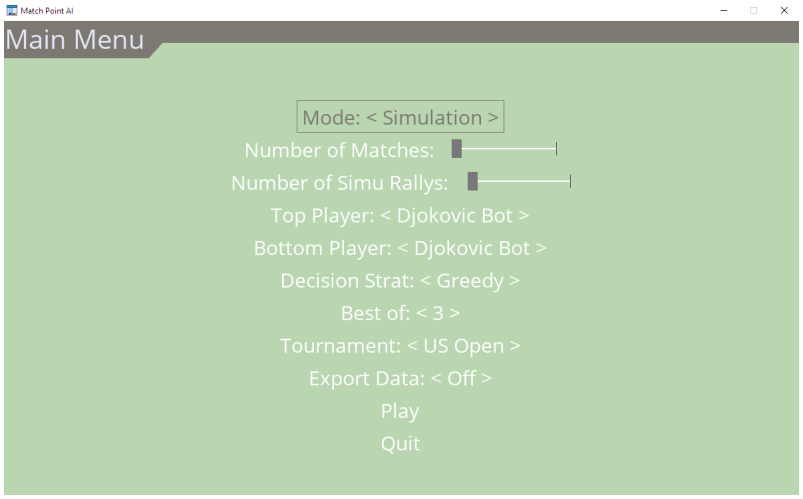

Abbildung 2.6: Match Point AI Interface [27]

Point AI bietet verschiedene Einstellungsmöglichkeiten. Man kann zwischen einem Simulationsmodus und einem manuellen Modus wechseln. Weiterhin kann

man die Anzahl der Matches, die beiden Spieler, die getestet werden, die maximale Anzahl der Sets pro Match und ob die Daten in einer csv-Datei exportiert werden, festlegen.

2.5.3 Bots

Für Match Point AI wurden im Rahmen der Masterarbeit bereits drei Bots erstellt. Der erste Bot basiert auf dem Tennisspieler Novak Djokovic. Er ist einer der besten Spieler der Welt und hält mehrere Rekorde. Das Ziel dieses Bots ist es, einen herausfordernden Gegner für die zu testenden Algorithmen anzubieten. Die Daten für diesen Bot stammen aus dem Match Charting Project. Dabei wurden 19.325 Ballwechsel von Djokovic analysiert. Wie bereits beschrieben, wurden daraus 252 Wahrscheinlichkeiten für die Schlagrichtung, Sieg und Verlust für Aufschläge, Returns und normale Schläge berechnet. Neben dem Djokovic-Bot wurde ein Average-Stat-Bot entwickelt, welcher auch 252 Wahrscheinlichkeiten besitzt. Dazu wurden alle 295.354 Ballwechsel genutzt, wobei 420 Spieler teilgenommen haben. Die untere Abbildung zeigt die 10 Spieler mit dem größten Anteil an diesen Daten.

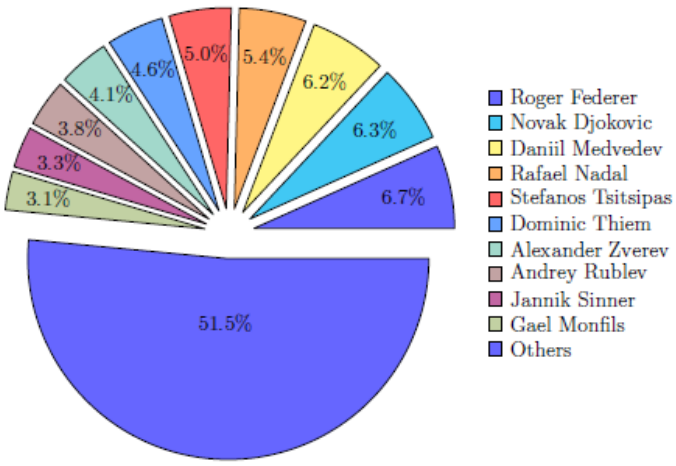

Abbildung 2.7: Anteile von Spielern am Average-Stat-Bot [27]

Neben den beiden Bots, die auf festen Strategien basieren, wurde ein Bot entwickelt, welcher einen Monte-Carlo-Tree-Search-Algorithmus nutzt. Dabei wird ein Baum erstellt, wo ein Knoten eine Schlagrichtung repräsentiert. Wenn der MCTS-Bot am Zug ist, wird ein Blattknoten gesucht, für den noch nicht

alle Richtungen gespielt wurden. Danach wird der Knoten um die noch nicht gespielten Richtungen erweitert. Für jede Richtung finden mehrere Simulationen statt, bei denen abwechselnd der MCTS-Bot zufällig und der Gegner basierend auf dessen Strategie einen Schlag auswählen. Dies geschieht solange, bis ein terminierender Zustand erreicht ist. Basierend auf den Ergebnissen der Simulationen werden die Werte aller Knoten, die zum Blattknoten führen, aktualisiert. Dann wird sich für eine Richtung basierend auf der Strategie des MCTS-Bots entschieden.

3 Methodik

In diesem Kapitel wird auf die genutzte Software, die Entwicklungen und die Experimente eingegangen.

3.1 Software

Für die Entwicklung wird Python mit der Version 3.10 genutzt. Neben den Standardmodulen wurden folgende Module genutzt: Für die Datenverarbeitung wurden die Module `pandas`(2.0.0) und `NumPy`(1.24.2) genutzt. Für statistische Berechnungen gab es `SciPy`(1.15.2) und `statsmodels`(0.14.4). Zur visuellen Darstellung wurde `matplotlib`(3.7.1) genommen. Weitere Hilfsmittel für Berechnungen und grafische Darstellungen waren LibreOffice(25.2.2.2), yEd(3.21.1) und Drawio(27.1.5).

3.2 Bots

In diesem Abschnitt werden die für die Experimente entwickelten Bots vorgestellt.

3.2.1 Random-Bot

Da die Entscheidungsalgorithmen unabhängig von einer vordefinierten Schlagwahrscheinlichkeit agieren sollen, wie es zum Beispiel der Average Bot aus Match Point AI besitzt, soll zuerst ein Zufalls-Bot implementiert werden, der zum einen als Vorlage für die folgenden Agenten dient sowie zu Vergleichszwecken genutzt werden kann. Der Zufalls-Bot übernimmt dabei die Gewinn- und Verlustwahrscheinlichkeiten aus dem Average-Stat-Bot. Dabei werden die Wahrscheinlichkeiten für die Schlagrichtungen durch eine 33%-ige Schlagwahrscheinlichkeit ersetzt.

3.2.2 MCSim-Bot

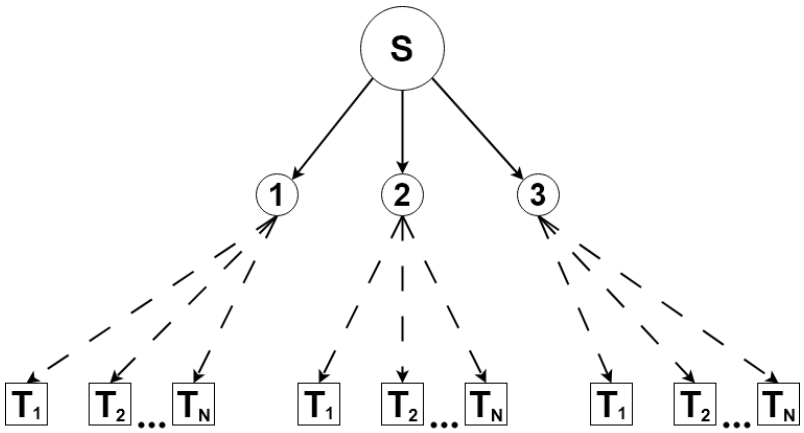

Abbildung 3.1: Darstellung Monte-Carlo-Simulation

Dieser Bot setzt den Monte-Carlo-Simulationsansatz um. Wenn der Bot am Zug ist, befindet er sich dann im Zustand S. Von dort muss er sich für eine Schlagrichtung als Aktion entscheiden. Dazu werden für jede Möglichkeit N Simulationen durchgeführt. Innerhalb der Simulation werden die Gewinn-/Verlustwahrscheinlichkeiten des Average-Stat-Bot genutzt, damit er möglichst flexibel gegen verschiedene Gegner spielen kann. Dadurch ist er wahrscheinlich im Nachteil gegen sehr gute Spieler wie Novak Djokovic, sollte aber im Vorteil gegen den Durchschnittsspieler sein. Die Schlagwahrscheinlichkeiten hingegen sind gleichverteilt, um eine hohe Bandbreite an Lösungsmöglichkeiten zu ermöglichen. Eine Simulation wird durchgespielt, bis ein Terminierungszustand T erreicht wird. Dies passiert, wenn der Spieler gewinnt oder verliert. Sollte er gewinnen, bekommt er einen Punkt. Verliert er, bekommt er nichts. Dadurch hat jede Schlagrichtung am Ende zwischen 0 und N Punkten. Schlussendlich wird sich für die Option mit den meisten Punkten entschieden. Sollten mehrere Richtungen gleich viele Punkte haben, wird sich zwischen denen zufällig entschieden und die gefundene Lösung wird gespielt.

Algorithm 1 Monte-Carlo-Simulation

```
Input: Rally, Score
Output: Shot
function MCSIM(Rally, Score)
    gamestate ← GETGAMESTATE(Rally, Score)
    direction ← GETDIRECTION(Rally, Score)
    for shot ← 1...3(4...6) do
        winsPerDirection ← SIMULATE(shot, gamestate, direction)
    winner ← Shotdirection with most wins in winsPerDirection
function SIMULATE(shot, gamestate, direction)
    wins ← 0
    while i < Simulationcount do
        shot ← Check if shot wins or loses
        newShot ← shot
        while no one won yet do
            opponentShot ← RANDOMSHOT(newShot, gamestate,
            direction)
            if opponentShot wins/loses then
                Break
            newShot ← RANDOMSHOT(newShot, gamestate, direction)
        if opponentShot loses or newShot wins then
            wins ← wins + 1
    return wins
```

3.2.3 RHEA-Bot

Neben der Monte-Carlo-Simulation wird in dieser Arbeit auch der RHEA-Algorithmus untersucht.

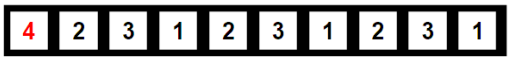

Abbildung 3.2: Beispiel Individuum

Wenn der Bot dran ist, wird eine Population entsprechend der Populationsgröße erzeugt. Ein Individuum dieser Population stellt einen Plan der nächsten Schläge bis zum Horizont H dar. Die Schläge der Individuen werden entsprechend der Regeln zufällig erzeugt. Das bedeutet, wenn der Bot den Aufschlag

hat, ist der erste Wert 4, 5 oder 6. Bei allen anderen Schlägen wird zwischen 1, 2 oder 3 gewählt. In der Abbildung 3.2 wird ein Beispiel eines solchen Individuums gezeigt. Die Schlagrichtung, die schlussendlich ausgewählt wird, ist rot markiert. Danach wird die Population evaluiert und die Fitness der Individuen bestimmt. Dazu durchläuft jeder Plan N Simulationen. Für jeden Schlag im Plan wird basierend auf den Gewinn- und Verlustwahrscheinlichkeiten aus dem Average-stat-Bot bestimmt, ob der Schlag siegreich oder ein Fehler war. Wenn es keins von beiden war, dann wird für den Gegner vom Zufalls-Bot ein Schlag ausgewählt. Beginnt der RHEA-Bot mit einem Aufschlag und macht einen Fehler in der Simulation, dann wird der Schlag in der gleichen Richtung wiederholt. Gewinnt das Individuum oder der Gegner macht einen Fehler, bekommt die Simulation einen Punkt. Ist die Simulation bis zum Horizont H durchgelaufen oder der Gegner hat gewonnen, dann gibt es keinen Punkt. Zur Evaluierung wird die Anzahl der gewonnenen Simulationen gezählt. Danach folgt die Selektion. Alle Individuen werden zu Paaren für die nächste Generation zusammengestellt. Wie ein Paar entsteht, wird mit einer Turnierselektion entschieden. Dazu werden zweimal zwei Individuen aus der Population ausgewählt und deren Fitness miteinander verglichen. Die besseren zwei Individuen werden gepaart und die anderen beiden kommen zurück in die Population. Sollten zwei Individuen die gleiche Fitness haben, dann wird zufällig gewählt. Dies geschieht solange, bis alle Individuen gepaart wurden. Die letzten beiden in der Menge werden direkt zusammengeführt. Darauf folgt ein Ein-Punkt-Crossover. Es wird basierend auf der Crossover-Rate entschieden, ob dieser ausgeführt wird. Wenn sich dazu entschieden wurde, wird ein zufälliger Punkt entlang der Individuenlänge ausgewählt, der sie in zwei Teile teilt, welche dann ausgetauscht werden. In der darauffolgenden Mutation wird für jedes Gen basierend auf der Mutationsrate entschieden, ob sich der Wert in einer der beiden anderen Schlagrichtungen ändert. Wurden Paare vom Crossover oder der Mutation nicht betroffen, bleiben sie der Nachkommensmenge erhalten. Diese Menge bildet dann die neue Population und wird wieder evaluiert, um den Prozess zu wiederholen, bis eine festgelegte Generationsanzahl erreicht wurde. Von der letzten Generation wird dann vom besten Individuum der erste Schlag als Ergebnis gewählt.

Algorithm 2 RHEA

```
1:  Input: Rally, Score
2:  Output: Shot
3:  function RHEA(Rally, Score)
4:      population ← CREATE_POPULATION(Rally, Score)
5:      Population_mit_Punkten ← EVALUATION(Rally,
        Score,Population)
6:      while i < Generationen do
7:          eltern ← PARENT_SELECTION(Population_mit_Punkten)▷ Tur-
            nierselektion
8:          crossover ← CROSSOVER(eltern)          ▷ Ein-Punkt-Crossover
9:          mutation ← MUTATION(crossover)
10:         Population_mit_Punkten ← NEUE_POPULATION(Rally,
            Score,mutation)
11: function CREATE_POPULATION(Rally, Score)
12:     gamestate ← GETGAMESTATE(Rally, Score)
13:     population ← []
14:     while i < Populationsgroesse do
15:         individuum ← []
16:         while i < individuenlaenge do
17:             individuum ← zufälliger Shot basierend auf gamestate
18:         population ← individuum
        return population
19: function EVALUATION(Rally, Score,Population)
20:     population_mit_punkten ← []
21:     while i < Populationsgroesse do
22:         wins ← 0
23:         SIMULATION(Rally, Score,population[i])
24:         if Simulation gewonnen then
25:             wins ← wins + 1
26:         population_mit_punkten ← (population[i],wins)
        return population_mit_punkten
```

3.3 Experimente

In diesem Abschnitt werden die Experimente vorgestellt.

3.3.1 Hyperparameter Optimierung

In diesem Experiment werden die möglichen Hyperparameter für den RHEA-Bot untersucht. Dabei wird sich auf die drei Werte Populationsgröße, Individuenlänge und Anzahl der Simulationen konzentriert. Für jeden Parameter werden drei Werte festgelegt, wobei dann jede mögliche Kombination untersucht wird, womit es dann 27 Kombinationen sind. Um gute Kandidaten für die Parameter auszuwählen, werden zur Vorbereitung ein paar Experimente betrieben. Dazu wird testweise für alle drei Parameter die Werte 10,20 und 30 gegeben und 200 Matches gegen den Djokovic-Bot gespielt. Darauf basierend soll bestimmt werden, ob diese drei Werte für die Populationsgröße und Simulationsanzahl ausreichen, oder ob sie noch weiter angepasst werden müssen. Zur Ermittlung der Individuenlänge werden zwei Average-Stat-Bots für 310 Matches gegeneinander spielen. Damit wird dann die durchschnittliche Länge eines Schlagabtausches und die Standardabweichung ermittelt. Somit werden hierfür als Werte die Durchschnittslänge $\pm$ Standardabweichung genommen. Um die Qualität einer Kombination zu ermitteln, wird auf die Gewinnrate der Ballwechsel und die zeitliche Effizienz geschaut. Es werden Ballwechsel genommen, da die Gewinnrate für die Punkte zu zufällig ist, während sie für Sets und Matches nicht aussagekräftig genug ist. Ballwechsel bieten dafür ein gutes Mittelmaß. Für die Zeitmessung wird die durchschnittliche Dauer eines Funktionsaufrufs zur Ermittlung eines Schlages für den RHEA-Bot genutzt. Die Zeit wird auf einem Computer mit einem AMD Ryzen 5 3500 X Prozessor gemessen. Jede Kombination durchläuft 31 Matches, für die die Gewinnrate und zeitliche Effizienz gemessen werden. Um dies auszuwerten, wird die Paretofront ermittelt, um die besten Kandidaten zu bestimmen, von denen dann einer ausgewählt wird.

3.3.2 Turnier zum Vergleich von Monte-Carlo-Simulation und RHEA

Im zweiten Experiment werden die beiden Bots Monte-Carlo-Simulation, kurz MCSim, und RHEA gegen verschiedene Bots spielen, um danach ihre Leistung miteinander zu vergleichen. Für den MCSim-Bot wird eine Simulationsanzahl von 200 pro Entscheidung festgelegt. Die beiden Bots spielen nacheinander gegen den Average-Stat-Bot, Novak-Djokovic-Bot, Random-Bot und anschließend gegeneinander für jeweils 200 Matches. Zum Vergleich spielen auch

der Random-Bot und der Average-Stat-Bot in den Turnieren mit. Gegen den Novak-Djokovic-Bot tritt außerdem noch der MCTS-Bot an. Dabei wird die Gewinnrate für die Punkte und die durchschnittliche Länge der Ballwechsel gemessen. Die Ergebnisse werden danach analysiert und miteinander verglichen. Zusätzlich wird bei den Matches gegen den Djokovic-Bot die Spielstrategie gemessen. Dazu wird für jede Schlagart und jede Richtung mitgezählt, wie oft sie ausgewählt wurde. Danach wird für jede Schlagrichtung deren Anteil an der Schlagart bestimmt und in Prozente umgerechnet. Dadurch hat man für jede Schlagart eine Wahrscheinlichkeitsverteilung dafür, in welche Richtung der Bot spielen könnte. Es soll damit untersucht werden, ob mithilfe der Bots sich optimale Strategien gegen Djokovic finden lassen. Die gefundenen Strategien werden dann mit dem Average-Stat-Bot verglichen.

4 Ergebnisse

4.1 Hyperparameter Optimierung von RHEA

Vor dem eigentlichen Experiment wurden einige Tests durchgeführt, um zu ermitteln, mit welchen Werten für die Parameter gearbeitet wird. Zuerst wird sich mit dem Horizont der Individuen befasst. Der Horizont soll so gewählt sein, dass er die Länge eines Ballwechsels optimal abbildet. Ist der Horizont zu klein, wird die Simulation zu früh beendet, um einen Gewinner festzustellen, wodurch es als Verlust gewertet wird. Ist der Horizont zu groß, wirken sich die letzteren Gene nicht auf die Simulation aus. Wenn Crossovers oder Mutationen in diesem Bereich stattfinden, haben sie somit keinen Einfluss auf die Qualität des Individuums. Um das Problem zu umgehen, wurden zwei Average-Stat-Bots miteinander getestet und die Länge der gespielten Ballwechsel gemessen.

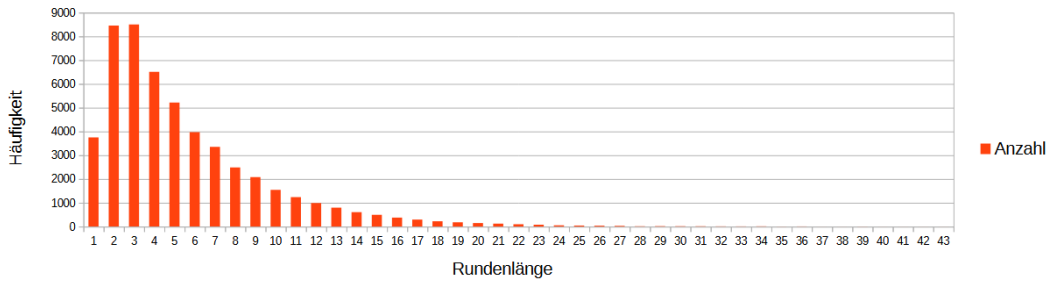

Abbildung 4.1: Häufigkeitsverteilung der Ballwechsellängen

In der Abbildung 4.1 sieht man, dass die Längen der Ballwechsel beim Tennis oft sehr kurz sind und innerhalb der ersten 2 bis 3 Schlägen entschieden werden. Insgesamt wurden 51.737 Ballwechsel gespielt. Aus dieser Menge hat sich eine durchschnittliche Spiellänge von 5,527 Schlägen ergeben. Die Standardabweichung dessen beträgt 3,242. Wenn man dies auf den Mittelwert drauf addiert

beziehungsweise abzieht, erhält man für den linken Rand 2,284 und für den rechten Rand 8,769. Aufgerundet erhält man die Werte 3, 6 und 9. Dies sind gute Parameterwerte, da sie den Großteil der Fälle abdecken, ohne zu große Überlänge zu haben. Für die Werte für die Parameter Simulationsanzahl N und Populationsgröße L wurden mehrere Kombinationen getestet, um einen Überblick zu erhalten, wie sehr sich die Qualität unterscheidet. Dazu wurden folgende Kombinationen für (N,L) getestet: (5,6); (10,10); (20,20); (30,30). Zusätzlich wurde zur Referenz der Djokovic-Bot getestet. Alle Beispiele haben 31 Matches gegen den Random-Bot gespielt, wobei die Siegrate für die Punkte, Spiele, Sets und Matches gemessen wurde.

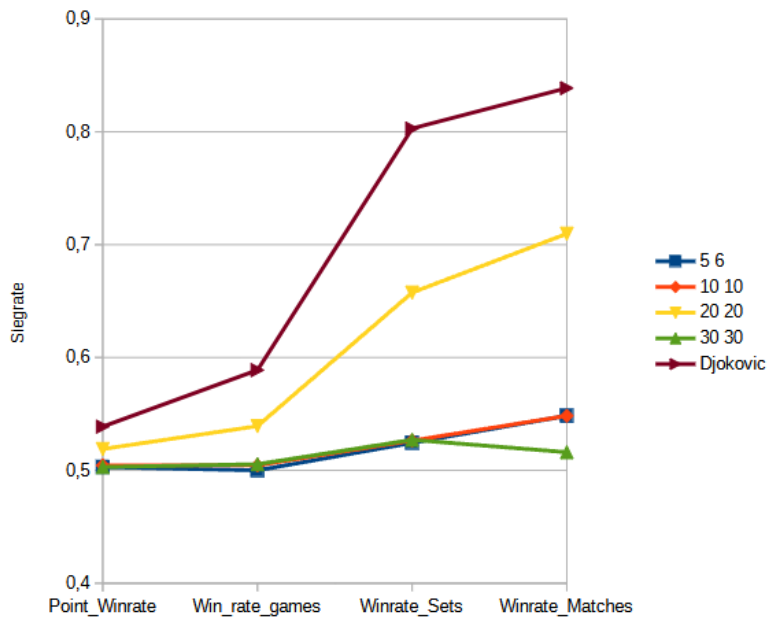

Abbildung 4.2: Vergleich Kombinationen gegen Random-Bot

In der Abbildung 4.2 kann man schon mehrere Dinge erkennen. Durchschnittlich wurden für jeden Test 4.885 Punkte erspielt, was erstmal eine gute Datenmenge darstellt. Trotzdem kann es zu Abweichungen in den Ergebnissen kommen, wie man es bei der Kombination für (20,20) erkennen kann, welche sich von den anderen drei RHEA-Tests abhebt. Bereits bei mehreren Tests im Verlaufe der Experimente konnte man erkennen, dass für die gleichen Voraussetzungen unterschiedliche Ergebnisse ermittelt wurden, was sich durch die höchst komplexe und zufällige Natur von Tennis erklären lässt. Daher darf

man auf kleine Abweichungen nicht zu viel Wert legen, und in diesem Beispiel schlussfolgern, dass (20,20) die beste Kombination ist. Des Weiteren lässt sich feststellen, dass die Gewinnraten für höhere Spielebenen extremer werden, je weiter sie sich von 50% entfernen. So hat in dem Fall (20,20) eine Punkte-Gewinnrate von 0,512, aber eine Match-Gewinnrate von 0,71. Bei Djokovic fällt dies nochmal deutlicher aus. Auch hier gibt es Abweichungen, wie man es bei (30,30) erkennen kann. Und schlussendlich sieht man, dass der Djokovic-Bot, welcher eigentlich im Vergleich zum Random-Bot weitaus besser ist, nur einen geringen Unterschied in der Punkte-Gewinnrate im Vergleich zu den anderen Bots hat. Der Unterschied zwischen dem Djokovic-Bot und dem schlechtesten Ergebnis bei den Punkten beträgt gerade mal 3,58%. Dies deutet an, dass höhere Kombinationen als (30,30) wohl kaum viel bessere Ergebnisse erzielen. Außerdem kommen höhere Kombinationen mit deutlich schlechterer Berechnungsdauer einher. Während ein Match mit der Kombination(5,6) gerade mal einige Sekunden brauchte, benötigt (30,30) bis zu 10 Minuten, was eine deutliche Verschlechterung für eine geringe Ausbeute ist. Daher bleiben die Parameter für die Simulationsanzahl und die Populationsgröße bei 10, 20 und 30.

Populationsgröße	**Simulationsanzahl**	**Horizont**	**Gespielte Matches**	**Gewinnrate Punkte**	**durchschnittliche Zeit**
10	10	3	31	0,507166	0,068965
10	10	6	31	0,503189	0,083318
10	10	9	31	0,492930	0,085019
10	20	3	31	0,527094	0,135886
10	20	6	31	0,521203	0,172972
10	20	9	31	0,516716	0,179199
10	30	3	31	0,518404	0,201381
10	30	6	31	0,504613	0,243081
10	30	9	31	0,522193	0,253016
20	10	3	31	0,523043	0,134975
20	10	6	31	0,513239	0,164275
20	10	9	31	0,510576	0,171057
20	20	3	31	0,499915	0,267023
20	20	6	31	0,506663	0,324091
20	20	9	31	0,516300	0,337929
20	30	3	31	0,516152	0,398455
20	30	6	31	0,512603	0,484332
20	30	9	31	0,523426	0,502942
30	10	3	31	0,510787	0,202625
30	10	6	31	0,521397	0,245407
30	10	9	31	0,513744	0,262077
30	20	3	31	0,512000	0,428963
30	20	6	31	0,506663	0,506457
30	20	9	31	0,504597	0,505699
30	30	3	31	0,512975	0,604024
30	30	6	31	0,512966	0,752617
30	30	9	31	0,524539	0,749484

Tabelle 4.1: Ergebnisse Hyperparameter Optimierung

Nach der Ausführung des Experiments erhielt man diese Ergebnisse, die man in der Tabelle 4.1 sieht.

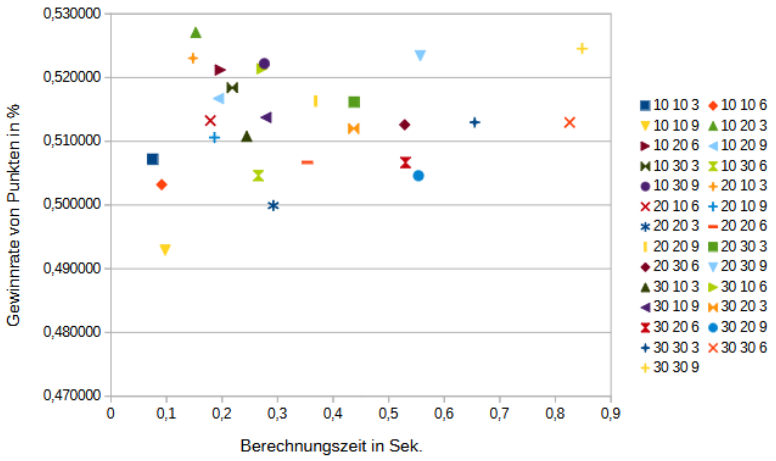

Abbildung 4.3: Darstellung Gewinnrate zur durchschnittlichen Berechnungszeit

Wie man in der oberen Abbildung 4.3 erkennen kann, gibt es große Unterschiede zwischen den verschiedenen Kombinationen bezüglich der zeitlichen Effizienz. Hierbei liegen die Ergebnisse zwischen 0,075 und 0,848 Sekunden, woraus sich schließen lässt, dass die Parameterwerte einen direkten Einfluss auf die Berechnungsdauer eines Schlages haben. Bei der Gewinnrate ist es hingegen nicht so eindeutig. Die meisten Werte liegen zwischen 51% und 52% mit einem Durchschnittswert von 51,31%. Dabei gibt es auch ein paar, die besonders hervorstechen, wie die Kombination (10,10,9), welche mit 49,29% das niedrigste Ergebnis erzielte und neben (20,20,3) als einziges sichtbar unter 50% liegt. Dem entgegen liegt (10,20,3), was das höchste Ergebnis erreicht hat mit 52,71% und mit 0,197 Sekunden eine relativ gute Effizienz hat. Somit könnte man daraus schließen, dass dies die beste Kombination ist, welche für die weitere Verwendung genutzt werden kann. Doch da die Ergebnisse so nah beieinander liegen, ist es nötig, sich genauer anzusehen, inwiefern sie zufällig sind und wie sich die Auswahl der Parameter darauf auswirkt.

Horizont	**Gewinnrate**	**durchschnittliche Zeit**
3	0,514171	0,295602
6	0,511341	0,360270
9	0,513891	0,373329

Tabelle 4.2: Durchschnitte für Horizonte

Populationsgröße	Gewinnrate	durchschnittliche Zeit
10	0,512612	0,174109
20	0,513546	0,339028
30	0,513245	0,516065

Tabelle 4.3: Durchschnitte für Populationsgrößen

Simulationsanzahl	Gewinnrate	durchschnittliche Zeit
10	0,510675	0,174729
20	0,512298	0,342029
30	0,516430	0,512444

Tabelle 4.4: Durchschnitte für simulationsanzahl

Dazu wurden zunächst die Durchschnittswerte der Werte für die drei Parameter berechnet. Diese werden in den Tabellen 4.2, 4.3 und 4.4 dargestellt. Dabei fällt erstmal auf, dass es bei der Gewinnrate keine nennenswerten Unterschiede existieren. Sie liegen alle zwischen 51% und 51,5%. Einzig die Simulationsanzahl 30 sticht dabei leicht heraus. Dennoch kann man daraus nicht ableiten, dass die Parameterwerte einen wirklichen Einfluss auf die Qualität haben. Bei der Berechnungszeit sieht dies allerdings anders aus. Bei der Populationsgröße und der Simulationsanzahl steigt die Berechnungszeit proportional zu den Werten an. Dies lässt sich auch mathematisch erklären, da pro Aufruf eine feste Anzahl Simulationen durchlaufen wird, die sich berechnen lässt aus: Generationenanzahl * Populationsgröße * Simulationslänge. Abweichungen können dadurch entstehen, da neben den Simulationen auch Crossover und Mutation stattfinden, welche aber nur geringen Einfluss haben. Außerdem müssen Simulationen nicht immer bis zum Ende durchlaufen, sondern können früher abbrechen. Diesbezüglich unterscheiden sich die Ergebnisse für den Horizont. Insgesamt nähern sie sich dem Gesamtdurchschnitt von 0,34 an, was bedeutet, dass der Horizont einen geringeren Einfluss auf die zeitliche Effizienz hat. Außerdem besteht zwischen 3 und 6 ein größerer Unterschied als zwischen 6 und 9. Wenn man sich nochmal die Tabelle 4.1 ansieht, so erkennt man, dass ein Großteil der Ballwechsel nicht länger als 3 sind, was bedeutet, dass bei einem Horizont von 9 die meisten Simulationen schon früher enden und die ganze Länge des Individuums nicht vollständig genutzt wird. Daraus ergibt sich auch ein geringerer Berechnungsaufwand. Um zu sehen, ob es relevante

Unterschiede zwischen den Parametern und ihren Interaktionen bezüglich der Gewinnrate gibt, wird eine dreifaktorielle ANOVA [30] genutzt. Dazu wird das Pythonpaket `statsmodels` verwendet. Dies ergibt folgende Ergebnisse:

Parameter	**sum_sq**	**df**	**F**	**PR(>F)**
Populationsgröße (P)	0.000127	2.0	0.046531	0.954537
Simulationsanzahl (S)	0.004914	2.0	1.802579	0.165534
Horizont (H)	0.001357	2.0	0.497683	0.608124
P:S	0.025206	4.0	4.623575	0.001074
P:H	0.004038	4.0	0.740732	0.564344
S:H	0.010517	4.0	1.929149	0.103636
P:S:H	0.00875	8.0	0.802487	0.600491
	1.103962	810.0		

Tabelle 4.5: Dreifaktorielle ANOVA der Parameter

In der Tabelle 4.5 sieht man, dass für ein Signifikanzniveau von 1% nur die Interaktion von Populationsgröße und Simulationsanzahl die mindestens einen signifikanten Unterschied aufweist. Für die anderen Interaktionen und die Parameter als solche ist es wahrscheinlich, dass Unterschiede eher zufällig entstanden sind. Um herauszufinden, wo dieser Unterschied besteht, werden die Konfidenzintervalle der Kombinationen mit einem Konfidenzniveau von 95% berechnet. In der Abbildung 4.4 sieht man, dass ein deutlicher Unterschied zwischen (10,10) und (10,20) existiert. Demnach kann man annehmen, dass die Kombination (10,20) besser ist als (10,10). Mit Blick auf 4.3 unterstützt dies die Entscheidung, die Parameterkombination (10,20,3) für die weiteren Experimente zu verwenden.

Wenn man nochmal auf die anfängliche Untersuchungsfrage schaut, bestätigt sich die Hypothese 1. Wenn es um die Berechnungszeit geht, gibt es deutliche Unterschiede zwischen den Kombinationen und man kann den Einfluss der Parameter auf die Dauer nachvollziehen. Bei der Gewinnrate allerdings scheint es sich eher zufällig zu verteilen, nur zwischen den Interaktionen (10,10) und (10,20) fällt ein Unterschied auf. Basierend auf den Ergebnissen wird für die weiteren Experimente die Parameter Populationsgröße=10, Simulationsanzahl=20 und Horizont=3 genommen.

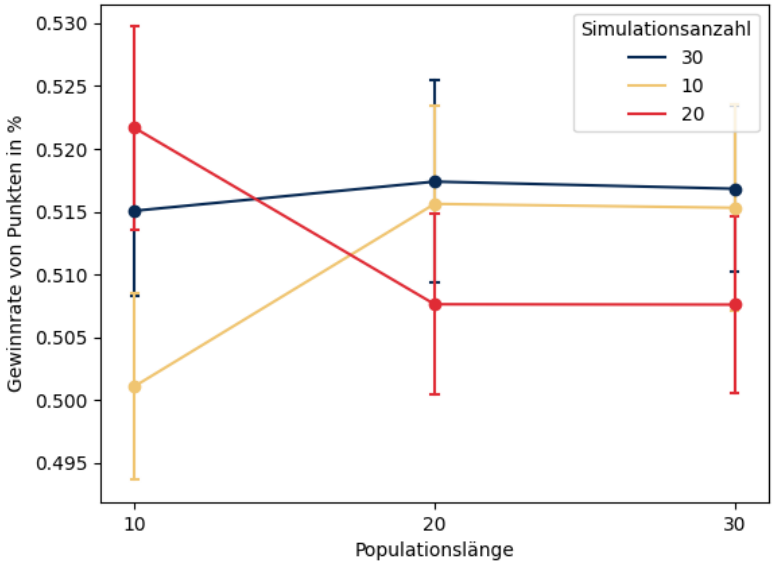

Abbildung 4.4: Konfidenzintervalle von Populationsgröße/Simulationsanzahl

4.2 Vergleich Monte-Carlo-Simulation und RHEA

Nachdem die Parameter für den RHEA-Bot festgelegt wurden, tritt dieser gegen den Monte-Carlo-Simulations-Bot in mehreren Matches gegeneinander an. Zum Vergleich wurden auch der Average-Stat-Bot und der Random-Bot getestet. Zuerst haben alle vier Bots in 200 Matches gegen den Random-Bot gespielt.

Bot	Punkte Gewinnrate	Match Gewinnrate
Random	0,497742	0,49
Average Stat	0,507123	0,56
MCSim	0,525363	0,8
RHEA	0,5118213	0,66

Tabelle 4.6: Vergleich Gewinnrate - Random

Aus den Ergebnissen der Tabelle 4.6 lässt sich eine klare Rangfolge ablesen. Demnach hat der MCSim-Bot am besten abgeschnitten, gefolgt vom RHEA-Bot, dann dem Average-Stat-Bot und dem Random-Bot. Wie zu erwarten hat-

te der Random-Bot eine etwa 50%-ige Gewinnrate, da dieser gegen sich selbst spielt und somit beide Seiten zufällig spielen. Dies spiegelt sich auch in der Match-Gewinnrate wider. Entsprechend hat der Average-Stat-Bot ein besseres Ergebnis, da dieser eine bestimmte Strategie verfolgt. MCSim und RHEA hingegen simulieren mithilfe des Random-Bots, daher haben diese bessere Gewinnchancen. Dies zeigt sich besonders beim MCSim-Bot, wodurch dieser 80% der Matches gewinnt. Es besteht dennoch die Chance, dass diese Unterschiede unterschiedlich sind, daher werden die Werte mit Tukey HSD[31] getestet.

group1	**group2**	**p-adj**	**reject**
MCSim	RHEA	0.0009	True
MCSim	averag-stat	0.0	True
MCSim	random	0.0	True
RHEA	average-stat	0.5545	False
RHEA	random	0.0005	True
average-stat	random	0.0439	True

Tabelle 4.7: Tukey HSD - Random

Man kann in der Tabelle 4.7 erkennen, dass zwischen fast allen Gruppen signifikante Unterschiede bestehen. Die einzige Ausnahme besteht zwischen RHEA und Average-Stat-Bot, wobei hier die Nullhypothese nicht verworfen werden kann.

Bot	**Punkte Gewinnrate**	**Match Gewinnrate**
Random	0,488828	0,41
Average Stat	0,502011	0,52
MCSim	0,516552	0,69
RHEA	0,505783	0,53

Tabelle 4.8: Vergleich Gewinnrate - Average

Im Spiel gegen den Average-Stat-Bot sehen die Ergebnisse in 4.8 ähnlich aus, auch wenn die Ergebnisse generell schlechter sind. Dies ist nachvollziehbar, da dieser Bot generell besser spielt als der Random-Bot. Das erkennt man auch daran, dass der Random-Bot öfter verliert als gewinnt. Die Entscheidungs-Bots sind aber weiterhin im Vorteil.

group1	group2	p-adj	reject
MCSim	RHEA	0.0076	True
MCSim	averag-stat	0.0001	True
MCSim	random	0.0	True
RHEA	average-stat	0.6755	False
RHEA	random	0.0	True
average-stat	random	0.0005	True

Tabelle 4.9: Tukey HSD - Average

Auch beim Tukey-Test 4.9 erkennt man, dass zwischen den Ergebnissen signifikante Unterschiede bestehen, außer zwischen dem RHEA-Bot und dem Average-Stat-Bot. Zusätzlich spielt hier zum Vergleich auch der MCTS-Bot

Bot	Punkte Gewinnrate	Match Gewinnrate
Random	0,451833	0,07
Average Stat	0,464358	0,12
MCSim	0,478456	0,27
RHEA	0,470588	0,19
MCTS	0,480994	0,3

Tabelle 4.10: Vergleich Gewinnrate Djokovic

mit. Als nächstes wird gegen den Novak-Djokovic-Bot gespielt. Dieser stellt im Gegensatz zu den anderen Bots die größte Herausforderung dar, da dieser neben einer besseren Spielstrategie auch bessere Siegwahrscheinlichkeiten hat. Dem entsprechend gewinnen sämtliche Bots nur recht selten, wie man in der Tabelle 4.10 sieht. Wie in den Wettbewerben vorher bleibt die Rangliste hier gleich, wobei der MCTS sich auf dem ersten Platz befindet. Dabei stechen der MCTS und MCSim-Bot dadurch heraus, dass sie beide über ein Viertel der Matches gewonnen haben, was im Vergleich zu den anderen ein sehr gutes Ergebnis ist. Dabei stechen der MCTS und MCSim-Bot dadurch heraus, dass sie beide über ein Viertel der Matches gewonnen haben, was im Vergleich zu den anderen ein sehr gutes Ergebnis ist.

group1	group2	p-adj	reject
MCSim	MCTS	0.9388	False
MCSim	RHEA	0.1183	False
MCSim	averag-stat	0.0002	True
MCSim	random	0.0	True
MCTS	RHEA	0.0139	True
MCTS	average-stat	0.0	True
MCTS	random	0.0	True
RHEA	average-stat	0.3213	False
RHEA	random	0.0	True
average-stat	random	0.0014	True

Tabelle 4.11: Tukey HSD - Djokovic

Die Ergebnisse des Tukey-Tests 4.11 sind ähnlich zu den vorherigen Ergebnissen, außer dass der MCTS-Bot dazugekommen ist. MCTS hebt sich stark von den anderen Bots ab, mit Ausnahme von dem MCSim-Bot, bei dem die Nullhypothese nicht verworfen werden kann. Weiterhin haben sich der RHEA-Bot und der MCSim-Bot so weit angenähert, dass auch hier kein signifikanter Unterschied mehr besteht.

Bot	durchschnittliche Ballwechsellänge
Random	5,495402
Average	5,604052
MCSim	5,279143
RHEA	5,218589

Tabelle 4.12: Vergleich durchschnittliche Ballwechsellänge

Neben den Gewinnchancen wurde auch die durchschnittliche Länge der Ballwechsel gemessen, was in 4.12 zu sehen ist. Es wird deutlich, dass der Average-Stat-Bot die längsten Ballwechsel hat. Man kann annehmen, dass dies daran liegt, dass in der Realität Tennisspieler weniger risikobereit sind und eher sicherer spielen, wodurch die Ballwechsel wiederum länger werden. Die Bots hingegen sollen stets in die Richtung mit der höchsten Gewinnwahrscheinlichkeit schlagen, wobei dann allerdings auch eine hohe Wahrscheinlichkeit bestehe, den Punkt zu verlieren. Dadurch sind deren Ballwechsel kürzer.

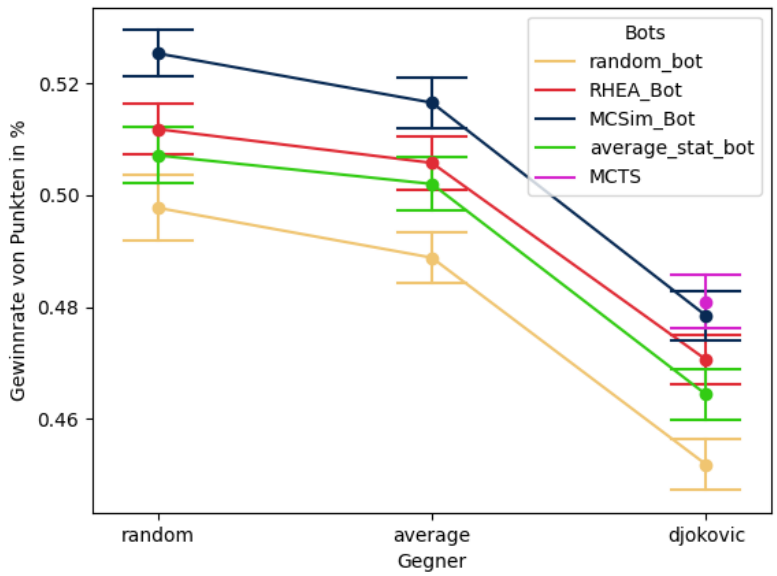

Abbildung 4.5: Matches Zusammenfassung

In der Abbildung 4.5 lassen sich zusammenfassend die Punktegewinnraten aus den Matches erkennen. Um auf die anfängliche Frage einzugehen, wie die beiden Bots gegen verschiedene Gegner abschneiden, so bestätigt sich Hypothese 0, da der MCSim-Bot bessere Ergebnisse erzielt als der RHEA-Bot. Weiterhin überbietet er deutlich den Average-Stat-Bot und kann gut mit dem MCTS-Bot mithalten. Auch wenn der RHEA-Bot nicht an diese Ergebnisse rankommt, so liegt dieser etwa auf einer Ebene mit dem Average-Stat-Bot, sogar leicht besser, auch wenn kein signifikanter Unterschied besteht. Auch bezüglich der Performance überbietet der MCSim-Bot den RHEA-Bot. Mit einer Generationsanzahl von 30, Populationsgröße von 10 und einer Simulationsanzahl von 20 finden bei einem Durchgang also 6000 Simulationen statt. Beim MCSim-Bot sind es nur 200 Simulationen. Somit zeigt sich, dass dieser Bot eine gute Alternative zum MCTS-Bot ist und eine gute Grundlage für weitere Forschungen darstellt.

4.3 Analyse gefundener Spielstrategien

Bei den Spielen gegen den Djokovic-Bot wurden neben der Gewinnrate auch die Spielstrategie gemessen. Dazu wurde für die beiden Bots für jeden Spielzustand und jede Schlagrichtung deren Wahrscheinlichkeit gemessen.

Bots	Links	Mitte	rechts
MCSim	45,15	20,31	34,54
RHEA	42,96	21,52	35,52
Average	29,27	35,31	35,64

Tabelle 4.13: Vergleich: durchschnittliche Wahrscheinlichkeit pro Richtung

In der Tabelle 4.13 kann man erkennen, dass die generelle Strategie von MCSim und RHEA recht ähnlich ist. Bei beiden wird oft nach links oder rechts gespielt, wobei links bevorzugt wird. Dies lässt sich damit erklären, dass die Seiten eine höhere Gewinnwahrscheinlichkeit bieten, da diese für den Gegner schwerer zu erreichen sind. Allerdings stellen sie ein Risiko dar, da die Gefahr besteht, ins Aus zu spielen. Da die meisten Tennisspieler Rechtshänder sind, sind die stärksten Schläge Vorhand-Schläge, die diagonal über das Netz gehen, weshalb eher nach links gespielt wird. Beim Average-Bot hingegen ist es eher gleichmäßig verteilt, wobei vor allem nach rechts und in die Mitte gespielt wird.

Richtung des Gegners	Links	Mitte	rechts
Links	53,11	16,96	29,93
Mitte	50,52	16,50	32,98
Rechts	35,51	23,08	41,41

Tabelle 4.14: MCSim - Nach Richtung des Gegners

Richtung des Gegners	Links	Mitte	rechts
Links	41,76	21,07	37,17
Mitte	45,83	18,64	35,53
Rechts	43,59	21,82	34,59

Tabelle 4.15: RHEA - Nach Richtung des Gegners

Richtung des Gegners	Links	Mitte	rechts
Links	35,05	34,87	30,08
Mitte	26,39	38,96	34,65
Rechts	19,11	38,35	42,55

Tabelle 4.16: Average - Nach Richtung des Gegners

Die Tabellen 4.14, 4.15 und 4.16 zeigen die Spielstrategien der Bots abhängig von der Schlagrichtung des Gegners. Das bedeutet, wenn der Gegner nach links spielt, landet der Ball auf der rechten Seite des Spielers. Beim MCSim-Bot fällt auf, dass mit einer Wahrscheinlichkeit von über 50% nach links gespielt wird, wenn der Ball von links oder aus der Mitte kommt. Wenn der Ball von rechts kommt, wird hingegen bevorzugt nach rechts gespielt. Da die meisten Tennisspieler Rechtshänder sind, könnte man daraus schließen, dass sie nach links am sichersten schlagen. Im linken Spielbereich ist es wiederum am besten, nach rechts zu spielen. Der RHEA-Bot ist im Vergleich dazu etwas ausgeglichener, obwohl hier auch links bevorzugt wird. Auffällig hierbei ist aber, dass RHEA auch dann mehrheitlich nach links spielt, auch wenn der Ball von rechts kommt. Der durchschnittliche Spieler schlägt meistens in die Mitte, außer wenn der Ball von rechts kommt. Dies könnte daran liegen, dass die meisten Tennisspieler wohl eher mit der Rückhand dann spielen, weshalb es dann sicherer ist, diagonal nach rechts zu spielen.

Zustand	**Links**	**Mitte**	**rechts**
1. Aufschlag	39,51	16,35	44,16
2. Aufschlag	39,73	37,39	22,88
Return auf 1. Aufschlag	29,09	32,25	32,78
Return auf 2. Aufschlag	31,01	29,08	39,91
normaler Schlag mit einem 2. Aufschlag	55,18	10,89	33,93
normaler Schlag mit einem 1. Aufschlag	60,42	5,45	34,12
normaler Schlag mit einem Return auf 2. Aufschlag	49,65	16,02	34,32
normaler Schlag mit einem Return auf 1. Aufschlag	52,93	8,97	38,11

Tabelle 4.17: MCSim - nach Spielzustand

Zustand	Links	Mitte	rechts
1. Aufschlag	38,29	22,55	39,17
2. Aufschlag	38,28	30,67	31,06
Return auf 1. Aufschlag	34,97	32,25	32,78
Return auf 2. Aufschlag	41,03	24,23	34,74
normaler Schlag mit einem 2. Aufschlag	48,12	16,85	35,03
normaler Schlag mit einem 1. Aufschlag	49,95	13,19	36,86
normaler Schlag mit einem Return auf 2. Aufschlag	44,40	19,14	36,46
normaler Schlag mit einem Return auf 1. Aufschlag	45,50	16,69	37,82

Tabelle 4.18: RHEA - nach Spielzustand

Zustand	Links	Mitte	rechts
1. Aufschlag	46,72	10,31	42,97
2. Aufschlag	33,65	39,08	27,27
Return auf 1. Aufschlag	17,02	58,25	24,73
Return auf 2. Aufschlag	17,05	51,65	31,30
normaler Schlag mit einem 2. Aufschlag	31,95	28,10	39,95
normaler Schlag mit einem 1. Aufschlag	33,84	23,87	42,29
normaler Schlag mit einem Return auf 2. Aufschlag	30,69	30,31	39,00
normaler Schlag mit einem Return auf 1. Aufschlag	30,55	32,17	37,28

Tabelle 4.19: Average - nach Spielzustand

Nun wird auf die verschiedenen Spielzustände eingegangen, welche in 4.17, 4.18 und 4.19. Beim ersten Aufschlag zieht der MCSim-Bot wieder die Seiten gegenüber der Mitte vor. Beim zweiten Aufschlag hingegen wird sehr viel mehr in Richtung der Mitte gespielt. So ähnlich sieht es auch beim Durchschnittsspieler aus. RHEA schlägt dagegen beim ersten Aufschlag mehr in die Mitte als die anderen beiden, und der Unterschied dazu ist beim zweiten Aufschlag nicht so groß. Dies lässt darauf deuten, dass Spieler dazu neigen, beim ersten Aufschlag einen eher riskanteren Ansatz zu verfolgen, um einen schnellen Punkt zu erzielen. Sollte dies nicht gelingen, scheinen sie beim zweiten Aufschlag deutlich sicherer zu spielen. Wenn man den Aufschlag zurückspielt, sind die Schlagrichtungen sowohl bei MCSim als auch bei RHEA relativ gleich verteilt. Bei dem Durchschnittsspieler fällt hingegen auf, dass vor allem in die

Mitte gespielt wird. Danach folgen die normalen Schläge. Der normale Spieler schlägt gleichmäßig in alle Richtungen, allerdings wird die rechte Seite bevorzugt. Bei MCSim und RHEA gibt es aber einen klaren Vorzug von links, was bei MCSim nochmal deutlicher auffällt. Daraus lässt sich schließen, dass es eine deutlich erhöhte Gewinnchance nach links existiert, was den meisten Tennisspielern aber nicht bewusst zu sein scheint. Basierend auf diesen Strategien wurden zwei Kopien von dem Zufallsbot erstellt, denen dann die jeweilige Strategie gegeben wurde. Bei mehreren Versuchen hat sich gezeigt, dass sie eine ähnliche Leistung erzielen, wie die entsprechenden Bots in 4.5. Insgesamt kann man sagen, dass sich zwischen allen drei Strategien Unterschiede finden lassen. Dennoch sind sich die beiden Strategien für MCSim und RHEA recht ähnlich, wobei es bei MCSim mitunter deutlicher eine bestimmte Seite bevorzugt wird. Im Vergleich zum Durchschnittsspieler gibt es allerdings große Abweichungen. Im Hinblick auf die Ausgangsfrage bestätigt sich somit die Hypothese 0. Da MCSim deutlich besser als der Average-Stat-Bot performt hat, wie man in 4.5 sieht, kann man davon ausgehen, dass dies die bessere Strategie ist.

5 Fazit und Ausblick

Nach anfänglichen Recherchen hat sich herausgestellt, dass intelligente Algorithmen zwar im Sportbereich getestet wurden, unter anderem um Spielergebnisse zu analysieren, es bisher aber kaum Forschung gibt bezüglich der Richtungsbestimmung im Tennis. Daher wurden in dieser Arbeit zwei Algorithmen im Rahmen einer Tennissimulation miteinander verglichen und analysiert. Dabei wurden zum einen eine Monte-Carlo-Simulation, welche für jede Richtung Ballwechsel simuliert, um die beste Möglichkeit zu finden, zum anderen ein Rolling Horizon Evolutionary Algorithm, welcher mehrere zufällige Schlagstrategien erzeugt und innerhalb eines evolutionären Algorithmus verbessert, implementiert. Zuerst wurden die Parameter für RHEA untersucht, um das bestmögliche Ergebnis in einer akzeptablen Zeit zu finden. Während es deutliche Unterschiede bei der Effizienz gab, lagen die Gewinnraten der einzelnen Kombinationen recht nah beieinander. Um herauszufinden, ob die vorhandenen Unterschiede nicht zufällig waren, wurde eine dreifaktorielle ANOVA durchgeführt. Es ergab sich, dass es keine signifikanten Unterschiede gab, außer bei einer Kombination, welche sich zwar geringfügig, aber dennoch von den anderen abhob. Da diese auch eine vergleichsweise gute Effizienz hatte, wurde sie für die weiteren Experimente gewählt. Danach wurde RHEA mit diesen Parametern gegen verschiedene Bots getestet und mit MCSim, den Average-Stat-Bot und den Random-Bot verglichen. Dabei spielte RHEA in etwa so gut wie der Average-Stat-Bot und gewann rund ein Fünftel aller Matches gegen den Djokovic-Bot. Dies zeigt, dass RHEA zwar nicht besser spielt, aber trotzdem ein optimiertes Ergebnis liefert und es nicht zufällig ist, da es deutlich besser spielt als der Random-Bot. Der MCSim-Bot hingegen performte stets besser als die Vergleichs-Bots und ist in etwa so gut wie der MCTS-Bot. Das bedeutet, dass es mit einer Monte-Carlo-Simulation durchaus möglich ist, in einem komplexen System wie bei Tennis gute Entscheidungen zu treffen. Es zeigte sich auch, dass man mithilfe von RHEA und MCSim Strategien entwickeln kann, die sich stark vom Durchschnittsspieler unterscheiden und sogar besser als dieser spielen.

Wenn man also diese Algorithmen nicht nur auf den Durchschnitt anwendet, sondern es für einen bestimmten Spieler einsetzt, lassen sich bestimmte Verhaltensmuster erkennen und es hilft, die Stärken und Schwächen dessen zu analysieren. Außerdem lassen sich damit Strategien für Simulationsgegner entwickeln, die man auch für das Training oder für Videospiele nutzen kann. Dies muss sich nicht nur auf die Schlagrichtung beschränken. Es ist empfehlenswert, solche Simulationsumgebungen auch für andere Aspekte des Tennisspiels zu entwickeln, wie zum Beispiel die Schlaghöhe oder verschiedene Schlagstile. Weiterhin kann man auch weitere Algorithmen für dieses Framework entwickeln oder die bereits vorhandenen optimieren, zum Beispiel indem man für RHEA andere Fitnessfunktionen oder Selektionsbedingungen definiert und diese mit den anderen Bots vergleicht. Dies bietet nicht nur Einblicke in alternative Lösungsmöglichkeiten, sondern auch im Vergleich zwischen den Schwächen und Stärken unterschiedlicher Algorithmen.

Literaturverzeichnis

[1] "50 Tennis Quotes Every Player Should Know." https://quotecharms.com/blogs/tennis/50-tennis-quotes-every-player-should-know, May 2025. last accessed: 02.06.2025.

[2] "Reaction Time, Ball Control and the Essence of Tennis." https://www.tennisvancouver.com/blogs/news/reaction-time-ball-control-and-the-essence-of-tennis, Jan. 2022. last accessed: 02.06.2025.

[3] A. Kaya, "Decision making by coaches and athletes in sport," *Procedia-Social and Behavioral Sciences*, vol. 152, pp. 333–338, 2014.

[4] D. H. Jonassen, "Designing for decision making," *Educational Technology Research and Development*, vol. 60, pp. 341–359, Apr. 2012.

[5] European Parliament. Directorate General for Parliamentary Research Services., *Understanding algorithmic decision-making: opportunities and challenges.* LU: Publications Office, 2019.

[6] D. M. Duca Iliescu, "The Impact of Artificial Intelligence on the Chess World," *JMIR Serious Games*, vol. 8, p. e24049, Dec. 2020.

[7] B. Zhang, "Tactical Decision System of Table Tennis Match based on C4.5 Decision Tree," in *2021 13th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, pp. 632–635, Jan. 2021. ISSN: 2157-1481.

[8] G. Kyriakides, K. Talattinis, and S. George, "Rating Systems Vs Machine Learning on the context of sports," in *Proceedings of the 18th Panhellenic Conference on Informatics*, (Athens Greece), pp. 1–6, ACM, Oct. 2014.

[9] M. Wiering and M. Van Otterlo, eds., *Reinforcement Learning: State-of-the-Art*, vol. 12 of *Adaptation, Learning, and Optimization.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[10] R. L. Harrison, C. Granja, and C. Leroy, "Introduction to Monte Carlo Simulation," (Bratislava (Slovakia)), pp. 17–21, 2010.

[11] N. Metropolis, "The beginning of the monte carlo method," *Los Alamos Science Special Issue*, pp. 125–130, 1987.

[12] C. J. Mode, *Applications of Monte Carlo Methods in Biology, Medicine and Other Fields of Science*. BoD – Books on Demand, Feb. 2011. Google-Books-ID: S8ycDwAAQBAJ.

[13] L. Pollet, "Recent developments in quantum Monte Carlo simulations with applications for cold gases," *Reports on Progress in Physics*, vol. 75, p. 094501, Aug. 2012. Publisher: IOP Publishing.

[14] G. A. Bird, "Monte-Carlo simulation in an engineering context," *Progress in Astronautics and Aeronautics*, vol. 74, pp. 239–255, Jan. 1981. ADS Bibcode: 1981PrAA...74..239B.

[15] E. Dion, "A Monte Carlo assessment of suspense in sports and games," *Revue Jeunes Chercheurs en Management du Sport*, vol. ~, p. 8 p., Nov. 2009.

[16] S. Kupferschmid and M. Helmert, "A Skat Player Based on Monte-Carlo Simulation," in *Computers and Games* (H. J. van den Herik, P. Ciancarini, and H. H. L. M. J. Donkers, eds.), (Berlin, Heidelberg), pp. 135–147, Springer, 2007.

[17] J. Krčadinac, E. Maruševec, L. Jerković, I. Kovač, J. Zloić, A. šarčević, and M. Vranić, "Modeling Tennis Matches Using Monte Carlo Simulations Incorporating Dynamic Parameters," in *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*, pp. 1281–1286, May 2023. ISSN: 2623-8764.

[18] Z. Li and M. G. Ierapetritou, "Rolling horizon based planning and scheduling integration with production capacity consideration," *Chemical Engineering Science*, vol. 65, pp. 5887–5900, Nov. 2010.

[19] "Introduction," in *Introduction to Evolutionary Algorithms* (X. Yu and M. Gen, eds.), pp. 3–10, London: Springer, 2010.

[20] "Advanced Evolutionary Algorithms," in *Introduction to Evolutionary Algorithms* (X. Yu and M. Gen, eds.), pp. 39–132, London: Springer, 2010.

[21] "Basic Concepts," in *Evolutionary Algorithms for Solving Multi-Objective Problems: Second Edition* (C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen, eds.), pp. 1–60, Boston, MA: Springer US, 2007.

[22] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Computing and Applications*, vol. 32, pp. 12363–12379, Aug. 2020.

[23] M. Pérez-Toledano, F. J. Rodriguez, J. García-Rubio, and S. J. Ibañez, "Players' selection for basketball teams, through Performance Index Rating, using multiobjective evolutionary algorithms," *PLOS ONE*, vol. 14, p. e0221258, Sept. 2019. Publisher: Public Library of Science.

[24] J. Sun, Z. Chu, and J. Li, "Development of a Performance Evaluation Model for Tennis Players Using Event-based Scoring and Non-linear Regression with Differential Evolution Algorithm," in *2024 International Conference on Power, Electrical Engineering, Electronics and Control (PEEEC)*, pp. 1260–1264, Aug. 2024.

[25] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen, "Rolling horizon evolution versus tree search for navigation in single-player real-time games," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, GECCO '13, (New York, NY, USA), pp. 351–358, Association for Computing Machinery, July 2013.

[26] R. D. Gaina, S. Devlin, S. M. Lucas, and D. Perez-Liebana, "Rolling Horizon Evolutionary Algorithms for General Video Game Playing," *IEEE Transactions on Games*, vol. 14, pp. 232–242, June 2022. Conference Name: IEEE Transactions on Games.

[27] C. Nübel, "Match point ai: A novel reinforcement learning framework for evaluating data-driven tennis strategies," Master's thesis, Otto-von-Guericke Universität, Magdeburg, 2024.

[28] USTA, "Tennis scoring rules," 2024. last accessed: 02.06.2025.

[29] J. Sackmann, "Tennis match charting project." `www.tennisabstract.com/charting/meta.html`, 2023.

[30] P. Planing, "Zweifaktorielle Varianzanalyse (Two-Way ANOVA) – Statistik Grundlagen." `https://statistikgrundlagen.de/ebook/chapter/zweifaktorielle-varianzanalyse_two_way_anova/`. last accessed: 02.06.2025.

[31] Z. Bobbitt, "How to Perform Tukey's Test in Python." https://www.statology.org/tukey-test-python/, Dec. 2020. last accessed: 02.06.2025.

Selbstständigkeitserklärung

Thesis: Evaluation und Vergleich verschiedener Entscheidungsalgorithmen zur Optimierung von Tennisstrategien

Name: Jannis Surname: Römermann

Date of birth: 19.10.2001 Matriculation no.: 233164

I herewith assure that I have written the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or according to their meaning.

I am aware of the fact that violations of copyright can lead to injunctive relief and claims for damages of the author as well as a penalty by the law enforcement agency.

Furthermore, I confirm that I am aware that the use of content (including but not limited to text, figures, images and code) generated by artificial intelligence (AI) in the thesis must be disclosed. In those cases I have specified the AI system used, I have marked the specific sections of the thesis where AI-generated content was used, and I have provided a brief explanation of the level of detail at which the AI system was used to generate the content. I also stated the reason for using the tools.

I assure that even if a generative AI system has been used, the scientific contribution has been made entirely by myself.

Jannis Römermann Magdeburg, 24.06.2025

J. Römermann