_DE_DE

hochschule mannheim

Eine Künstliche Intelligenz für ein rundenbasierendes Strategiespiel mit versteckten Informationen am Beispiel des Pokèmon Showdown

Florian Kutz

Bachelor-Thesis

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

Studiengang Informatik

Fakultät für Informatik

Hochschule Mannheim

15.11.2019

Betreuer

Prof. Jörn Fischer, Hochschule Mannheim

Prof. Dr. Thomas Ihme Hochschule Mannheim

Kutz, Florian:
Eine Künstliche Intelligenz für ein rundenbasierendes Strategiespiel mit versteckten Informationen am Beispiel des Pokèmon Showdown / Florian Kutz. –
Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2019. 54 Seiten.

Kutz, Florian:
An artificial intelligence for a round based strategy game with hidden information at the example of Pokèmon Showdown / Florian Kutz. –
Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2019. 54 pages.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 15.11.2019

Florian Kutz

Abstract

Eine Künstliche Intelligenz für ein rundenbasierendes Strategiespiel mit versteckten Informationen am Beispiel des Pokèmon Showdown

Künstliche Intelligenzen (KIs), die Spiele auf dem Niveau professioneller Spieler spielen, haben großen Einfluss auf die jeweilige Spielgemeinschaft. Bei Pokémon-Kämpfen gibt es noch keine professionelle KI. In dieser Arbeit wird die Herausforderung und die bisherigen Fortschritte in dem Bereich analysiert und ein Prototyp namens Talonbot entwickelt, der einer professionellen KI möglichst nahe kommt. Der Prototyp ist eine Abwandlung von AlphaZero. In Tests gegen andere KIs zeigt sich, dass Talonbot unter den Testbedingungen besser spielt als alle anderen. Spiele gegen Versuchspersonen zeigen jedoch, dass die KI noch nicht professionell spielt.

An artificial intelligence for a round based strategy game with hidden information at the example of Pokèmon Showdown

Artificial intelligences (AIs), which play games on a level of professional players, have great influence on the individual game community. There's no such AI for Pokémon battles. This paper analyses the challenge and the progress made to create a professional AI for Pokémon battles. A prototype called Talonbot is developed with the goal to become close to a professional AI. The algorithm is AlphaZero with some adaptions. Test battles against other AIs show, that Talonbot is better than other AIs in the test environment. However, battles against human players show, that the AI does not play professionally.

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen des Spiels **4**
2.1 Pokémon Showdown im Kontext der Pokémon-Spielreihe 4
2.1.1 Die Spielreihe Pokémon 4
2.1.2 Pokémon Showdown 4
2.2 Einstieg in das Kampfsystem 5
2.3 Artenvielfalt 5
2.4 Attacken 5
2.5 Generationen 6
2.6 Statuswerte 8
2.7 Regelwerke 8
2.8 Kampfverlauf 8

3 Grundlagen erwägter Algorithmen **11**
3.1 Minimax 11
3.2 Monte-Carlo Tree Search (MCTS) 13
3.3 Neuronale Netze 14
3.4 AlphaGo 16
3.5 AlphaGo Zero und AlphaZero 17
3.6 Neural Fictitious Self-Play 17

4 Problemanalyse **18**
4.1 Problem und Implementierungsziel 18
4.2 Herausforderungen 19
4.2.1 Unperfekte Information 19
4.2.2 Normal-Form 20
4.2.3 Zufallsereignisse 21
4.2.4 Zorua und Zoroark 21

5 Verwandte Arbeiten **22**
5.1 KIs auf Github 22
5.2 KI basierend auf neuronalen Netzen 23
5.3 Bachelorarbeit von Miquel Llobet Sanchez 24

5.4 Minimax-KI von Robert A. Mills 24
5.5 Zusammenfassung 25

6 Eigener Implementierungsplan 26
6.1 Setting 26
6.2 Verwendete Algorithmen 27
6.3 Teamanalysierer 27
6.4 Teamsimulationen 28
6.5 Verwaltung der simulierten Teams 30
6.6 Implementierung des MCTS 31
6.7 Einsatz von neuronalen Netzen 34

7 Implementierung 35
7.1 Architektur und Codebasis 35
7.2 Verwendung von Servercode 36
7.3 Replizierung des Kampfstatus 37
7.4 Teamsimulator und MCTS 38
7.5 Wahrscheinlichkeitsberechnung 38

8 Evaluation 42
8.1 Ealuation der neuronalen Netze 42
8.2 Evaluation der Lauffähigkeit 43
8.3 Evaluation der Spielerstärke 45
8.3.1 Evaluation gegen andere KIs 45
8.3.2 Evaluation gegen menschliche Spieler 47
8.4 Auswertung des Fragebogens 49

9 Ergebnisse und Ausblick 52

Abkürzungsverzeichnis v

Tabellenverzeichnis vi

Abbildungsverzeichnis vii

Quellcodeverzeichnis viii

Literatur viii

Online Literatur ix

Kapitel 1

Einleitung

In der Herausforderung, KIs für Spiele auf hohem Spielniveau zu entwickeln, gibt es bereits große Erfolge. Ein Beispiel ist der Algorithmus AlphaGo von DeepMind, welcher basierend auf Simulationen des weiteren Spielverlaufs entscheidet. Dieser Algorithmus ist in der Lage, einen Weltmeister im Brettspiel Go zu schlagen. [41] Der von AlphaGo weiterentwickelte Algorithmus AlphaZero ist auch in der Lage, sich selbst Schach und Shogi beizubringen und auf dem Niveau professioneller Spieler zu spielen. Dabei verlässt sich die KI nicht auf die menschliche Erkenntnisse über das jeweilige Spiel, sondern trainiert sich selbst auf professionelles Niveau mit anfangs keinerlei Wissen über vorteilhafte Spielzüge und Stellungen. [39] Der Erfolg dieser KIs geht über ihre hohe Spielstärke hinaus. Auf der Webseite von DeepMind wird betont, dass professionelle Go-Spieler von dem Erfolg und der Spielweise ihrer KI inspiriert werden. AlphaGo nutzt unkonventionelle Strategien, die funktionieren. Davon inspiriert versuchen menschliche Spieler, die Strategien zu übernehmen oder komplett neue zu entwerfen, um so ihre eigene Spielweise weiter zu verbessern. [4] Dieser Effekt ist jedoch auf die paar Spiele limitiert, für die es eine solche KI gibt, die das Spiel auf professionellem Niveau spielen kann. Bei anderen Spieler-gegen-Spieler (PvP)-Spielen kann ein gleicher Effekt erfolgen, wenn eine erfolgreiche KI für sie implementiert wird.

Pokémon ist eine bekannte und erfolgreiche Serie von Videospielen. Die Spiele der Hauptreihe sind Rollenspiele (RGPs). In den Spielen erlebt der Protagonist ein Abenteuer, in dem er häufig gegen computergesteuerte Gegner kämpft. Diese Kämpfe werden mit Wesen namens *Pokémon* ausgetragen, die von dem Spieler beziehungsweise dem Computer gesteuert werden. Eine Seite hat bis zu sechs Pokémon und verwendet in der Regel eines gleichzeitig. Die Kämpfe verlaufen in

Runden und in jeder Runde können die Kontrahenten ihr Pokémon gegen ein anderes austauschen oder das aktuelle Pokémon eine Attacke einsetzten lassen. Die Entscheidungen der Trainer geschehen gleichzeitig ohne Wissen der gegnerischen Wahl. [33]

Es gibt auch die Funktion, dass Kämpfe zwischen zwei menschlichen Spielern ausgetragen werden können, wenn beide eine Version des Spiels haben. Diese PvP-Kämpfe bilden ein Interesse für einen großen Teil der Spieler. Diese Spieler nutzten nicht nur die Funktion, sondern organisieren sich auch auf der Webseite https://www.smogon.com/. Hier werden zusätzliche Regeln festgelegt, mit denen die Spiele fairer werden und Strategien werden unter den Spielern ausgetauscht, mit denen sie sich verbessern. Es werden auch Turniere ausgetragen, beispielsweise im Video Game Championships (VGC)-Format. [5]

Häufig werden Spiele nicht mit den Hauptspielen ausgetragen sondern mit *Pokémon Showdown*. Pokémon Showdown simuliert PvP-Kämpfe und wirbt damit, weniger umständlich zu sein, als die Spiele der Hauptspielreihe. Es ist mit einem Internet-Browser spielbar oder mit einer lokalen installierbaren Anwendung. [34, 29]

Für PvP-Kämpfe in Pokémon gibt es noch keine professionelle KI, welche die Spielgemeinschaft so aufmischt, wie AlphaGo die Go-Spieler. KIs für das Spiel gibt es durchaus, zum Beispiel die computergesteuerten Gegner in den Hauptspielen. Jedoch sind Kämpfe während dem Abenteuer nicht vergleichbar mit PvP-Kämpfen. Die Pokémon, die man während dem Abenteuer bekämpft haben Level, die mit dem Verlauf des Abenteuers ansteigen, während die eigenen Pokémon stetig hochleveln. Die Level der Pokémon der Kontrahenten sind selten gleich und viele Gegner haben nicht einmal die maximale Anzahl von Pokémon zur Verfügung. Die Gegner in den Spielen sind darauf ausgelegt, dass man als Spieler mehrere hintereinander besiegen kann. Außerdem gibt es bei den Kämpfen gegen sie von beiden Seiten die Option, sein Pokémon mit Items zu heilen. Bei PvP-Kämpfen gibt es die Option nicht. Noch dazu gibt es Hinweise, dass die KI schummelt, da sie auf Informationen zugreift, die ein menschlicher Spieler nicht hat. [45] Folglich ist die KI, die in den Hauptspielen verwendet wird, ungeeignet, um neue Strategien zu generieren, die inspirieren.

Pokémon-Kämpfe beinhalten einige Schwierigkeiten für die Entwickklung einer KI, beispielsweise die versteckten Informationen. AlphaGo und AlphaZero funktionieren bei Spielen, in denen beide Spieler perfekte Informationen über das Spiel

haben, was bei Pokémon Showdown nicht der Fall ist. Darum wird es generell interessant, wie eine Entwicklung einer KI für Pokémon Showdown funktioniert. Die Erkenntnis kann auch auf andere Spiele mit den ähnlichen Problemen wirken.

In dieser Arbeit werden die Herausforderungen, die die Entwicklung einer professionellen KI schwierig machen, näher analysiert. Ferner gibt es einen Überblick über bestehende KIs. Im Rahmen dieser Arbeit wird ein Prototyp einer KI implementiert, mit AlphaZero als Ziel. Die Entwicklung wird in dieser Arbeit dokumentiert mit Fokus auf die Bewältigung der vorher identifizierten Herausforderungen. Zum Schluss wird der Prototyp evaluiert, indem er gegen andere KIs spielt und gegen menschliche Spieler. So wird die Spielerstärke eingeschätzt und die Schwächen des Prototyps werden offengelegt.

Kapitel 2

Grundlagen des Spiels

Bevor eine KI für das Spiel geschrieben werden kann, müssen zuerst die Spielregeln bekannt sein. In diesem Abschnitt werden deshalb die Hintergründe und Spielregeln von Pokémon Showdown erklärt. Da die Spielregeln zu komplex sind, um komplett erklärt zu werden, beschränkt sich dieses Kapitel auf das Nötigste. Für weitere Informationen wird auf Quellen verwiesen.

2.1 Pokémon Showdown im Kontext der Pokémon-Spielreihe

2.1.1 Die Spielreihe Pokémon

Pokémon ist eine Reihe an Videospielen von Gamefreak. In den Hauptspielen erlebt ein Spieler ein Abenteuer, in dem man regelmäßig in Kämpfe gegen computergesteuerte Gegner verwickelt wird. Das Kampfsystem ist rundenbasiert und wird in diesem Kapitel noch näher beschrieben. Es gibt auch die Möglichkeit, in dem gleichen Kampfsystem gegen andere menschliche Spieler zu spielen, doch dies ist oft umständlich. [32]

2.1.2 Pokémon Showdown

Pokémon Showdown ist eine Simulation dieser PvP-Kämpfe. Diese Simulation wird über einen Web-Browser gespielt und ist unter der Domäne https://play.pokemonshowdown.com/ erreichbar. Alternativ gibt es auch eine lokale Anwendung dieses Spiels.

Die Webseite oder lokale Anwendung dient als Client, der mit dem Server kommuniziert. Der Server simuliert die Kämpfe zwischen zwei Clients. [34]

2.2 Einstieg in das Kampfsystem

Bei Pokémon-Kämpfen treten immer zwei Spieler gegeneinander an, die jeweils bis zu 6 Wesen namens *Pokémon* kontrollieren. Die Spieler werden *Pokémon-Trainer* oder kurz Trainer genannt. Die Pokémon, die ein Trainer bei einem Kampf einsetzt, werden zu einem sogenannten Pokémon-Team oder kurz Team zusammengefasst. In den meisten Kämpfen haben beide Trainer zu jedem Zeitpunkt jeweils ein Pokémon des eigenen Teams auf dem Kampffeld. Der Kampf verläuft in Runden und in jeder Runde wählen beide Trainer gleichzeitig ihre Aktion aus, ohne die gegnerische Zugwahl zu kennen. [32]

2.3 Artenvielfalt

Pokémon sind von Tieren der realen Welt inspiriert. Dementsprechend gibt es auch eine Artenvielfalt. Pokémon teilen sich wie Tiere auch in *Spezies* ein. Die Spezies eines Pokémon definiert sein Aussehen und Eigenschaften im Kampf. Diese Spezies sind in einem virtuellen Lexikon namens *Pokèdex* aufgelistet und durchnummeriert. Für bestimmte Spezies existieren auch verschiedene Formen mit der geichen Pokédex-Nummer. Auch verschiedene Formen wirken sich auf die Kampfeigenschaften aus. [35]

2.4 Attacken

Jedes Pokémon kann bis zu vier verschiedene *Attacken* beherrschen. Die vier erlernten Attacken der verwendeten Pokémon werden vor einem PvP-Kampf festgelegt. Wenn ein Pokémon in einer Runde nicht ausgewechselt wird, kann sein Trainer eine Attacke wählen, die es in der Runde ausführt. Die verschiedenen Attacken bieten verschiedene Vorteile im Kampf.

Jedes Pokémon kann abhängig von seiner Spezies und Form nur eine bestimmte Menge an Attacken erlernen. Diese Menge wird *Movepool* genannt. In Ausnahmefällen wird der Movepool von weiteren Faktoren beeinflusst. [6]

2.5 Generationen

Die Spiele der Spielreihe Pokémon sind in *Generationen* aufgeteilt, die nach der Reihenfolge ihres Erscheinens nummeriert sind. Zum Zeitpunkt der Erstellung dieses Subabschnitts (28.Januar 2019) gibt es sieben Generationen. Während dieser Bachelorarbeit wird eine achte Generation angekündigt, die erst zum etwaigen Zeitpunkt der Fertigstellung erscheint. Darum ist über sie wenig bekannt. [33]

Jede Generation führt neue Pokémon-Spezies ein, sowie neue Attacken und neue Kampfmechaniken. Mit neuen Attacken ändern sich auch die Movepools der Spezies. In der Tabelle 2.1 ist eine Übersicht über die Generationen gegeben. Hier ist gelistet, welche Spiele der Hauptreihe zu der Generation gehören, wie viele neue Pokémon-Spezies aus der Generation stammen, wie viele neue Attacken eingeführt werden und welche wichtigen Erneuerungen des Kampfsystems unternommen werden.

Alle gelisteten Spiele haben den Prefix „Pokémon“, der in der Tabelle ausgeblendet ist.

Die angeführten Erneuerungen des Kampfsystems werden in der jeweiligen Quelle näher erklärt. In jeder Generation gibt es noch weitere kleinere Änderungen, die hier der Übersichtlichkeit wegen nicht angeführt sind.

Durch die Änderungen des Kampfsystems in den Generationen hat jede Generation eine oder mehrere eigene Versionen des Kampfsystems. Eine Version meint hier eine Menge an verfügbaren Spezies, Attacken und Kampfmechaniken. Je höher die Generationennummer, desto komplizierter ist in der Regel das Kampfsystem. Mit Pokémon Showdown können Kämpfe in jeder erschienenen Generation simuliert werden. [34]

Nr.	Spiele	Neue Spezies	Neue Attacken	Wichtige Erneuerungen des Kampfsystems	Quelle
1	Rot Blau Gelb Stadium	151	165	Premiere des Kampfsystems Typeffektivitäten Volltreffer 5 Statuswerte Determinierende Werte(DV) Stat Experience(Stat.Exp) Statuswertveränderungen Statusprobleme	[10]
2	Gold Silber Kristall	100	86	Die Typen Stahl und Unlicht Spezialwert-Aufteilung Trageitems Pokémon-Geschlechter Schillernde Pokémon Freundschaftswert Wetter Entry Hazards	[11]
3	Rubin Saphir Smaragd Feuerrot Blattgrün	135	103	Fähigkeiten Wesen DV-Reform Fleiß-Punkte(EV) Doppelkämpfe	[12]
4	Diamant Perl Platin HeartGold SoulSilver	107	113	Physisch-Spezial-Aufteilung Video Game Championships	[13]
5	Schwarz Weiß Schwarz 2 Weiß 2	156	92	Versteckte Fähigkeiten Dreierkampf	[14]
6	X Y Omega Rubin Alpha Saphir	72	66	Typ Fee Mega-Entwicklung Feldeffekte	[15]
7	Sonne Mond Ultra Sonne Ultra Mond Let's Go Pikachu Let's Go Evoli	88	83	Z-Angriffe	[16]
8	Schwert Schild	?	?	Dynamax Gigantamax	[17]

Tabelle 2.1: Die Tabelle gibt eine Übersicht darüber, wie sich das Kampfsystem von Pokémon über die Zeit entwickelt hat. Die Spielreihe ist in Generationen aufgeteilt und jede Generation hat zugehörige Spiele, die hier aufgelistet sind. Die Erneuerungen des Kampfsystems sind in Stichpunkten aufgelistet. Genaueres über die Erneuerungen steht in den Quellen.

2.6 Statuswerte

Jedes Pokémon hat verschiedene numerische Statuswerte (Stats), welche die Kampfeigenschaften des Pokémon beeinflussen.

Die Stats eines Pokémon sind primär von den *Basiswerten* seiner Spezies und Form abhängig. Die Stats eines Pokémon können vor einem Kampf dauerhaft im begrenzten Rahmen angepasst werden. [43]

Jedes Pokémon hat zu Beginn eines PvP-Kampfes seine maximale Anzahl Kraftpunkte (KP), der durch einen Stat bestimmt wird. Nimmt ein Pokémon Schaden, verliert es KP, wird es geheilt, gewinnt es KP. Die häufigste Ursache von Schaden sind Attacken gegnerischer Pokémon, die direkt Schaden verursachen. Es gibt auch andere Schadensquellen. Sinken die KP eines Pokémon auf 0, wird das Pokémon kampfunfähig. [24]

2.7 Regelwerke

Pokémon Showdown bietet Kämpfe in verschiedenen *Regelwerken* an. Jedes Regelwerk ist einer Generation zugehörig. Bei fast jedem Regelwerk gibt es zusätzliche Einschränkungen der Regeln. Beispielsweise beinhalten die meisten Regelwerke eine Regel, dass man nicht mehrere Pokémon derselben Spezies verwenden darf. Ob man seine sechs Pokémon selber wählen kann oder sie zufällig generiert werden, hängt ebenfalls vom Regelwerk ab. [27]

2.8 Kampfverlauf

Abbildung 2.1 skizziert den Ablauf eines Kampfes. Vor einem Kampf findet bei beiden Trainern immer eine *Teamauswahl* statt. In der Teamauswahl werden die Pokémon gewählt, die ein Trainer einsetzt, einschließlich der Anpassungen der Statuswerte der Pokémon, der Auswahl der Trageitems, Fähigkeiten und Attacken der Pokémon. Wenn das Regelwerk ein RandomBattle ist, übernimmt ein Zufall-Algorithmus die Teamauswahl, bei anderen Regelwerken können die Trainer mit dem Teambuilder manuell ihre Teams zusammenstellen. [36]

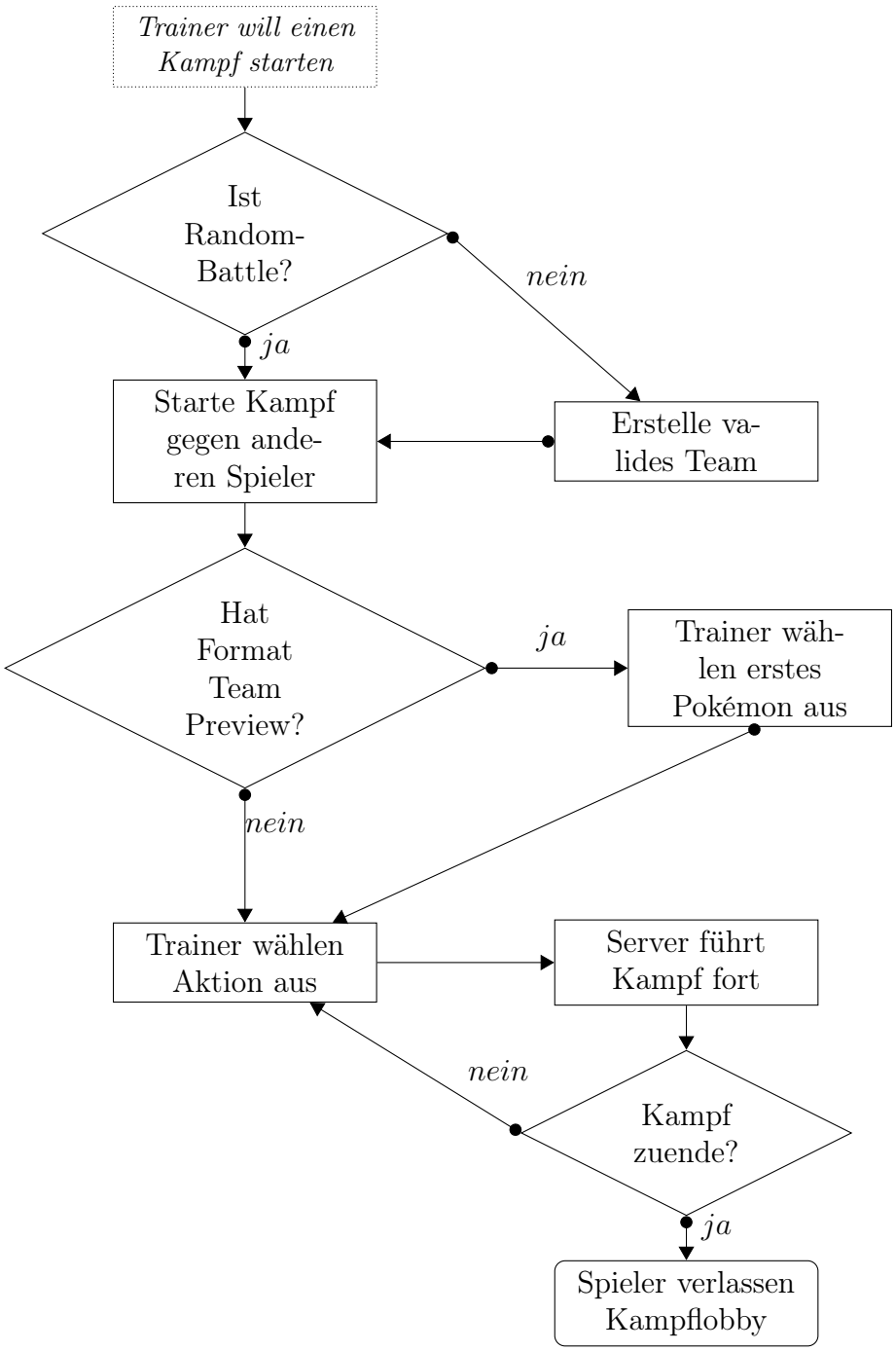

Abbildung 2.1: Wenn ein Spieler einen Kampf haben will, muss er zuerst ein valides Pokémon-Team erstellen oder ein vorher erstelltes auswählen, vorrausgesetzt das Regelwerk, in dem dieser spielen will ist kein RandomBattle. Bei RandomBattles werden Teams automatisch generiert. Dann startet der Trainer den Kampf. Der Server startet dann einen Kampf zwischen zwei Trainern. Wenn das Regelwerk eine TeamPreview vorsieht, sehen beide Trainer zu Beginn des Kampfs eine Teilinformation des gegnerischen Teams und wählen ein Pokémon des eigenen Teams aus, mit dem sie anfangen. In einer Runde des Kampfes müssen beide Trainer gleichzeitig ihre Aktion auswählen ohne die Wahl des Gegners zu sehen. Wenn beide Aktionen gewählt sind, wird die Runde vom Server ausgeführt, bis wieder Entscheidungen von den Trainern erforderlich sind. Dies wird so lange fortgeführt, bis ein Spieler gewonnen hat.

Sobald die Teams erstellt sind, beginnt ein Kampf zwischen zwei Spielern. Ein Kampf kann abhängig vom Regelwerk auf verschiedene Weisen beginnen. Bei einigen Regelwerken wird das erste eingesetzte Pokémon der beiden Trainer vor dem Kampf bei der Teamauswahl gewählt ohne Wissen über das gegnerische Team. Andere Regelwerke sehen eine sogenannte *Team Preview* vor, in der beide Trainer einige Informationen über das gegnerische Team erhalten und dabei das erste Pokémon auf dem Kampffeld wählen. [44]

Der Kampf verläuft in Runden. In jeder Runde müssen beide Trainer gleichzeitig einen Aktion auswählen, ohne die Zugentscheidung des Gegners zu kennen. Ein Trainer hat dabei mehrere Zugmöglichkeiten: Man kann eine der bis zu vier Attacken des eigenen Pokémons auf dem Feld auswählen, oder es mit einem der anderen Pokémon aus dem Team austauschen. Sobald beide Trainer eine Aktion ausgewählt haben, führt der Server den Kampf fort bis die nächste Entscheidung getroffen werden muss oder der Kampf entschieden ist. [27]

Ziel eines Pokémon-Kampfs ist es, alle Pokémon des gegnerischen Teams kampfunfähig zu machen, bevor die eigenen Pokémon kampfunfähig sind. Wenn ein Pokémon kampfunfähig wird, kann es für den Rest des Kampfes nicht mehr verwendet werden. Sein Trainer kann am Ende der Runde ein anderes Pokémon aus seinem Team wählen, welches bei der nächsten Runde auf dem Feld sein wird. [32]

Kapitel 3

Grundlagen erwägter Algorithmen

In diesem Abschnitt werden verschiedene Algorithmen erklärt, die für eine Pokémon Showdown-KI in Frage kommen.

3.1 Minimax

Minimax ist ein Algorithmus, der rundenbasierte PvP-Spiele spielen kann. Abbildung 3.1 visualisiert den Prozess am Beispiel Tic-Tac-Toe. Wenn der Algorithmus einen Zug macht, analysiert er für die nächsten Runden, welche Verläufe möglich sind. Dabei werden entweder alle möglichen Ausgänge analysiert, was nicht immer möglich ist, oder nur die Möglichkeiten der nächsten x Runden. x steht hier für eine natürliche Zahl, die von der Implementierung des Algorithmus abhängt. ([38, 2])

Dadurch entsteht ein Verlaufbaum über die nächsten x Runden. Unter der Annahme, dass sowohl die KI selbst, als auch der Gegner immer den optimalen Weg gehen wird, wird im Verlaufbaum analysiert, welcher Zug welchen Zustand in x Runden zur Folge hätte.

Wenn der komplette, restliche Verlauf nicht berechnet werden kann und nur die nächsten x Runden analysiert werden, wird eine Funktion benötigt, die einen nicht-finalen Spielzustand bewertet. Die Wertung sagt aus, wie hoch in dem Zustand die Gewinnchance für die KI ist.

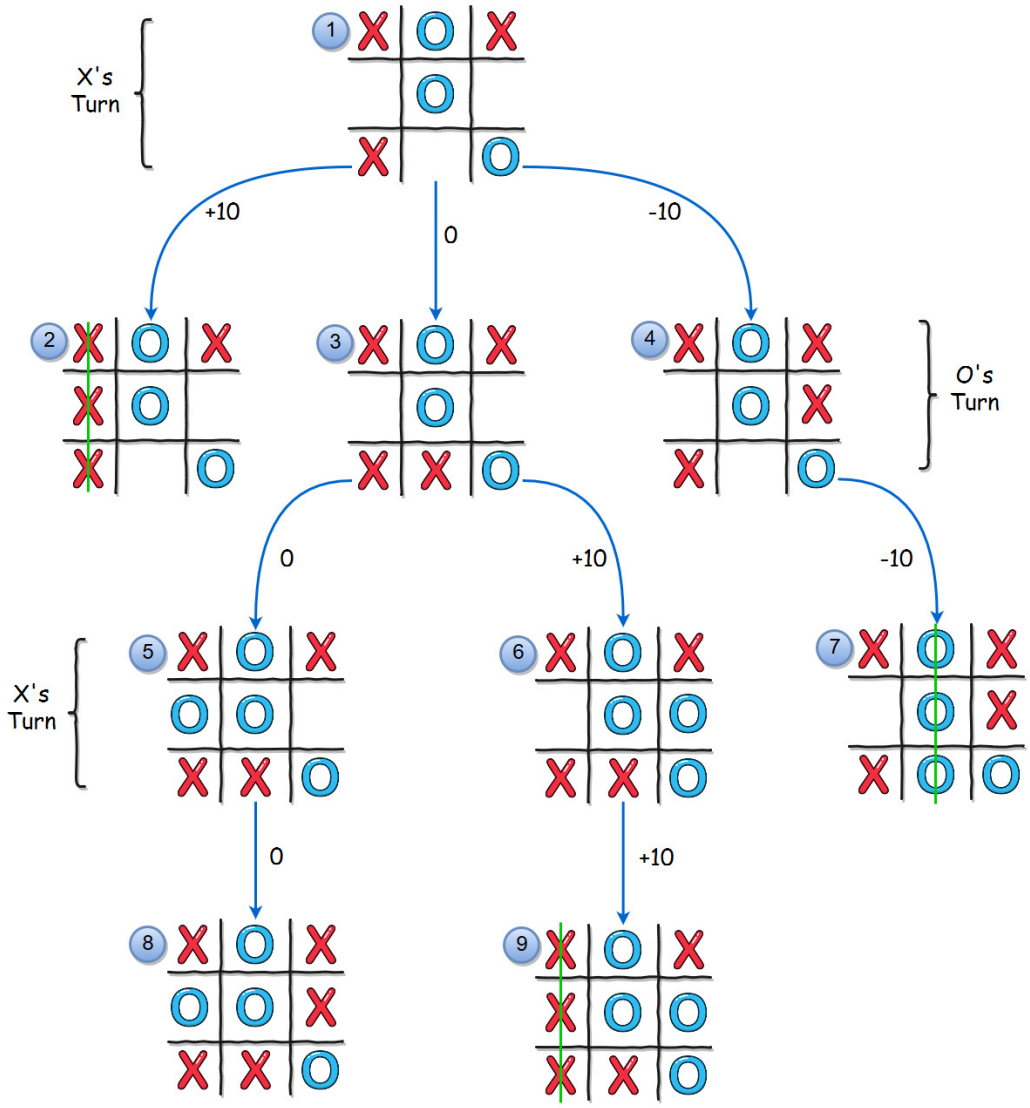

Abbildung 3.1: Hier wird anhand des Beispielspiels Tic-Tac-Toe die Funktionsweise von Minimax dargestellt. Die finalen Zustände werden hier mit +10 bewertet bei einem Sieg von X, -10 bei einem Sieg von O und 0 bei einem Gleichstand. Der aktuelle Zustand ist Zustand 1 mit Spieler X am Zug. Der Spieler hat drei Zugoptionen. Wenn das Kreuz links gesetzt wird, ist das Spiel in Zustand und X hat gewonnen. Wenn das Kreuz unten gesetzt wird, ergibt sich der nicht-finale Zustand 3, von dem aus weitere Zustände folgen. Bei Zustand 9 gewinnt X. Da 9 der einzig mögliche Folgezustand von Zustand 6 ist, ist die Wertung beider Zustände 10. Das gleiche gilt für 8 und 5, da bei 8 jedoch ein Gleichstand herrscht, sind beide Wertungen 0. 5 und 6 sind Folgezustände von Zustand 3. Da bei Zustand 3 O am Zug ist, ist davon auszugehen, dass hier der Zustand mit der schlechtesten Wertung folgt. Das ist Zustand 5. Folglich hat für Minimax der Zustand 3 die gleiche Wertung wie Zustand 5, also 0. Bei Zustand 4 gibt es die Folgezustände 5 und 7, jedoch ist in der Abbildung die Verbindung zu Zustand 5 ausgeblendet. Da O am Zug ist, wird auch für Zustand 4 die schlechteste Wertung der Folgezustände übernommen, und zwar die Wertung -10 von Zustand 7. Schließlich wählt Minimax für X den Zug mit dem besten Folgezustand. Da Zustand 2 die höchste Wertung hat, wird somit das Kreuz links gemacht. [38]

Bei nicht-finalen Zuständen geht die KI davon aus dass der Spieler, der den Zug in dem Zustand macht, den für sich besten Zug ausführt. Der Zustand erhält dadurch die höchste Wertung der Kindknoten, wenn die KI am Zug ist, oder die niedrigste, wenn der Gegner zieht. Finale Zustände erhalten als Wertung den Spielausgang und nicht-finale Zustände mit der Tiefe x werden mit einer Funktion bewertet, die den durchschnittlichen Ausgang bewerten soll und von der Implementierung abhängt. Auf diese Weise ermittelt der Algorithmus von hinten nach vorne die Wertungen der Züge, bis schließlich die Wertungen der direkten Kindknoten des aktuellen Zustands bekannt sind. Die KI wählt schließlich den Zug, dessen Resultat der Zustand mit der höchsten Gewinnwahrscheinlichkeit ist. ([38, 2])

3.2 MCTS

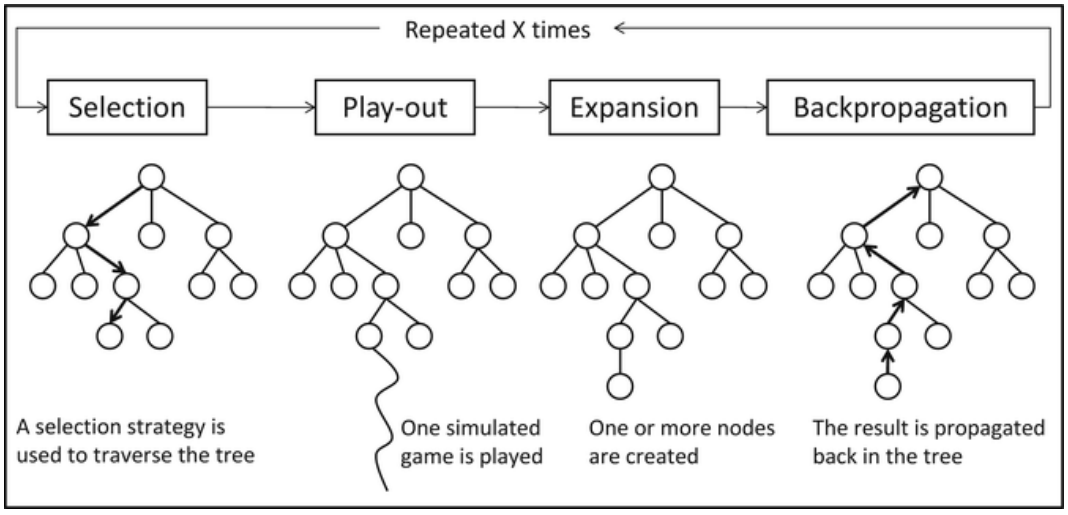

Abbildung 3.2: Diese Abbildung von Winands et. al skizziert den Entscheidugsprozess Monte-Carlo Tree Search. Die Bäume stellen dabei die möglichen zukünftigen Spielverläufe dar, soweit diese geladen sind. Jeder Knoten steht für einen Zustand im Spiel, auf den mehrere mögliche Zustände folgen können. Der Wurzelknoten an der Spitze steht für den aktuellen Zustand. Der Prozess wird aufgerufen, wenn die KI eine Zugentscheidung treffen soll. Je nach Entscheidung der KI wechselt das Spiel in einen der Folgezustände des Wurzelknotens. Um zu entscheiden, welche Entscheidung für die KI ideal ist, wird ein Prozess X mal iterativ ausgeführt. Dieser Prozess beinhaltet die vier Schritte Selektion, Play-Out, Expandierung und Backpropagation. Bei der Seletion wird ein Knoten im geladenen Baum ausgewählt, indem vom Wurzelknoten aus zufällige Kindknoten ausgewählt werden. Dabei können auch nicht geladene Knoten erfasst werden. Beim Play-Out wird von dem vorher gewählten Knoten aus ein Spiel simuliert. Dabei werden weiter zufällige Kindknoten gewählt, bis ein finaler Zustand erreicht ist, bei dem ein Gewinner feststeht. Bei der Expandierung werden ein oder mehrere Knoten der Simulation dem geladenen Baum hinzugefügt. Zum Schluss wird der Gewinner des Play-Outs in alle geladenen Knoten gespeichert, die auf dem Pfad des Play-Outs liegen. Nachdem der Prozess X mal ausgeführt ist, kann der Algorithmus bestimmen, welcher Kindknoten die größten Gewinnchancen für die KI hat und wählt die entsprechende Zugoption. [47]

Monte-Carlo Tree Search (MCTS) ist ein Algorithmus der Minimax ähnelt. Genau wie Minimax ist MCTS in der Lage, rundenbbasierte Spiele zu spielen, indem es den Verlaufbaum analysiert.

Im Gegensatz zu Minimax ist MCTS nicht auf eine Evaluationsfunktion angewiesen. Stattdessen wird ein Zustand mit Simulationen bewertet. In einer Simulation erweitert der Algorithmus den Verlaufbaum von einem Zustand bis zu einem finalen Zustand, in dem ein Gewinner feststeht. Der Ausgang wird für jeden Knoten gespeichert, der auf dem Pfad der Simulation liegt. Statt mit einer klassischen Evaluationsfunktion wird ein Zustand anhand des Anteils der Simulationen bewertet, in denen die KI von dem Zustand aus gewinnt. Der Prozess, in denen MCTS den Verlaufbaum aufbaut und die Simulationen ausführt, wird in 6.1 dargestellt.

Auch dieser Algorithmus wählt schließlich den Zug, dessen Folgezustand die höchste berechnete Gewinnwahrscheinlichkeit hat. ([47, 2, 9, 7])

Für den MCTS gibt es Anpassungsmöglichkeiten, damit er für verschiedene Spiele optimiert wird. Es existiert eine Übersicht von Browne et. al. über bestehende Implementierungsformen. [7]

3.3 Neuronale Netze

Ein *neuronales Netz* ist ein Algorithmus im Bereich des maschinellen Lernens. Es ist von dem Aufbau des menschlichen Gehirns inspiriert und kann Korrelationen von Informationen erkennen.

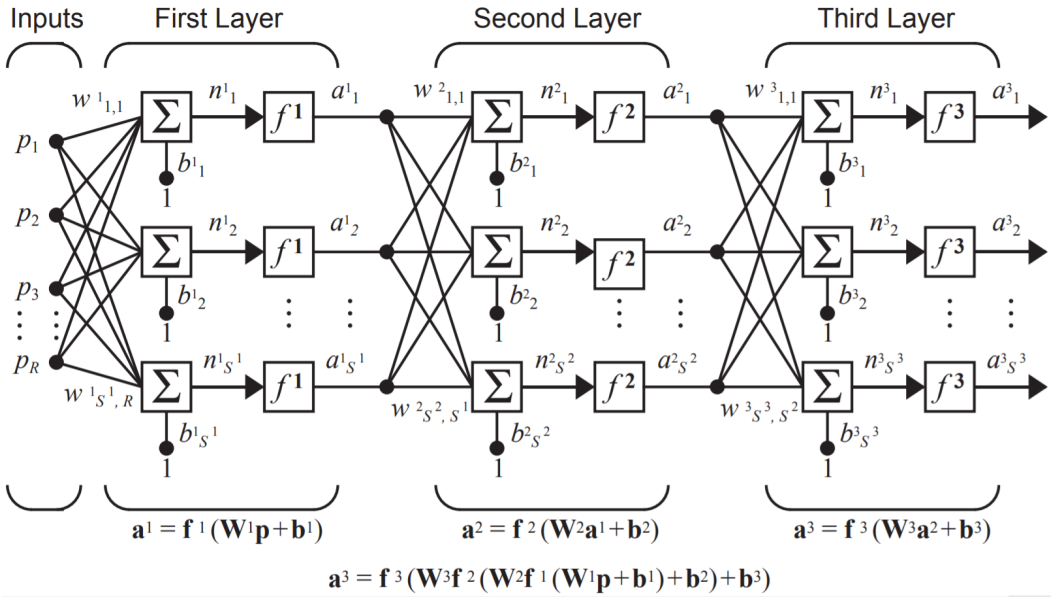

Abbildung 3.3: Die Eingabe wird als Vektor mit den Werten p_1, p_2, p_3, ..., p_R übergeben. Die Neuronen der ersten Schicht multiplizieren die Werte des Eingabevektors mit Gewichten $w^1_{n,p}$. Die Gewichte sind dabei von dem Neuron n und dem Eingabeindex p abhängig. Auch eine Biaseingabe mit dem konstanten Wert 1 wird in jedem Neuron n mit einem Gewicht b^1_n multipliziert. Die Produkte eines Neurons werden addiert und einer Funktion f^1 übergeben. Die Ergebnisse der Funktionsaufrufe bilden einen neuen Vektor mit den Werten a^1_1, a^1_2, ..., a^{11}_s. Dieser Vektor ist der Ausgabevektor der ersten Schicht und zugleich der Eingabevektor der zweiten Neuronenschicht. Alle Neuronenschichten verarbeiten den Eingabevektor auf die gleiche Weise wie die erste Neuronenschicht. Dabei hat jede Schicht ein eigenes Set an Neuronen und Gewichten, sowie eine eigene Funktion f. Auch die Größe des Eingabevektors und die Anzahl der Neuronen einer Schicht s können sich zwischen den Schichten unterscheiden, für eine Schicht bleiben diese aber konstant. Die Ausgabe der letzten Schicht ist zugleich die Ausgabe des neuronalen Netzes. Diese besteht in diesem Beispiel aus den Werten a^3_1, a^3_2, ..., a^{33}_s. [18]

Abbildung 3.3 zeigt, wie ein neuronales Netzwerk typischerweise aufgebaut ist. Der Bestandteil von neuronalen Netzen sind Neuronen, die Eingangsparameter erhalten. Ein Neuron multipliziert jeden Eingangsparameter mit dem jeweiligen Gewicht w des Parameters. Die Produkte werden addiert und die Summe wird einer Funktion f übergeben, die von der Implementierung abhängt. Das Ergebnis a wird vom Neuron ausgegeben.

Die Neuronen im Netz sind in mehrere Schichten aufgeteilt. Die Inputinformation wird in ihre Bits p aufgeteilt und den Neuronen der ersten Schicht als Input gegeben. Die Ausgabe der Neuronen einer Schicht wird als Input der Neuronen der nächsten Schicht weitergereicht. Die Ausgabe der letzten Schicht ist die endgültige Ausgabe. Bei diesem Beispiel handelt es sich um ein feedforward-Netz. Das heißt, dass die Neuronen nur vorwärts gerichtet verbunden sind und keine Ausgabe zu einer vorherigen Schicht geleitet wird.

Ziel eines solchen Netzwerks ist es, aus den Eingabeinformationen Ausgabeinformationen zu generieren. Dabei muss der Programmierer die genaue Funktion nicht kennen, mit der die Ausgabeinformation genau berechnet wird. Laut Hornik et al. ist es mit solchen Netzen möglich, jede Funktion ungefähr abzubilden. [21]

Damit das möglich ist, wird das neuronale Netz vorher mit Testdaten trainiert, bei denen Eingabe und Ausgabe bekannt sind. Das neuronale Netz verarbeitet die Eingabe der Daten. Die Ausgabe des neuronalen Netzes wird mit der Ausgabe der Testdaten abgeglichen und die Gewichte w innerhalb des neuronalen Netzes so angepasst, dass die Ausgabe eher der gewünschten Ausgabe entsprechen wird. Dieser Trainingsprozess ist essenziell für neuronale Netze, da ohne sie die Höhe der Gewichte und folglich die Ergebnisse geraten und nutzlos wären. ([18], [25])

3.4 AlphaGo

AlphaGo ist ein Algorithmus, der darauf ausgerichtet ist, das Spiel Go auf professionellem Niveau zu spielen. Der Algorithmus baut auf MCTS auf und beinhaltet zwei neuronale Netzwerke. Beide bekommen als Input den Status des Spiels zu einem Zeitpunkt.

Das erste Netz gibt aus, welche Züge mit welcher Wahrscheinlichkeit gespielt werden. Dadurch werden die Zugwahrscheinlichkeiten für die Simulationen des MCTS berechnet. Das Netz lernt aus Spielen professioneller, menschlicher Spieler.

Das zweite neuronale Netz gibt die Gewinnwahrscheinlichkeit einer Seite aus. Trainiert wird es mittels Spielen der KI gegen sich selbst. Dieser Ansatz stammt vom *Reinforcement-Learning*, bei dem durch wiederholtes Spielen die KI lernt, welche Zustände eher zum Ausgang eines Sieges führen. [22] Dieser Lernprozess geschieht bei AlphaGo in diesem neuronalen Netz.

Anstatt selbst zu spielen, werden die neuronalen Netze mit Spielverläufen von professionellen, menschlichen Spielern trainiert. Das macht sie besonders effektiv im Spiel gegen diese professionellen Spieler. Außerdem müssen dank dem neuronalen Netz die Simulationen nicht bis zum Spielende berechnet werden, was den Algorithmus beschleunigt. [9, 8]

3.5 AlphaGo Zero und AlphaZero

AlphaGo Zero ist ein ähnlicher Algorithmus wie AlphaGo. Anstatt von menschlichen Spielern zu lernen, lernt AlphaGo Zero von Spielen von AlphaGo gegen sich selbst, da AlphaGo das Spiel Go bereits besser als alle Menschen spielt. Die beiden neuronalen Netzwerke werden zu einem zusammengefasst. Beim Training des neuronalen Netzwerks muss es basierend auf dem Spielstatus den Zug der AlphaGo-KI und den Spielausgang ausgegeben. Während der Algorithmus AlphaGo Zero spielt, werden die Züge durch Reinforcement Learning gespielt, anstatt durch Simulationen. Wenn anschließend das neuronale Netz durch Spiele von AlphaGo Zero gegen sich selbst spielt, spielt die KI dadurch immer besser.

AlphaZero ist ein fast identischer Algorithmus mit einem Unterschied zu AlphaGo Zero. AlphaZero nimmt als Startpunkt KIs mit MCTS, anstatt menschliche Spieler. Dadurch braucht der Algorithmus mehr Iterationen, um auf professioneller Ebene spielen zu können, kann aber auf dem Weg dahin neue Strategien entwickeln. [39, 40, 9, 41, 8]

3.6 Neural Fictitious Self-Play

Ficticious Self-Play (FSP) ist ein Spiel-Algorithmus, mit dem Spiele in der Normal-Form gespielt werden können. Jede Spielentscheidung der Gegner wird von dem Algorithmus gespeichert, sodass aus der jeweiligen Historie abgelesen werden kann, welchen Zug der jeweilige Gegner mit welcher Wahrscheinlichkeit ausführt. Basierend auf dem Wissen wird der eigene Zug gezielter auf den Gegner gespielt. Dieser Algorithmus ist sehr effektiv im Spiel Poker. [20]

Bei einem Neural Ficticious Self-Play (NFSP) werden die gegnerischen Züge mittels neuronalen Netzen vorhergesagt, die mit ihrer Historie trainiert werden. Ferner gibt es hier ein zweites neuronales Netz zur Bestimmung der optimalen Konter-Strategie gegen die vermuteten gegnerischen Züge. [20]

Kapitel 4

Problemanalyse

In diesem Kapitel wird festgelegt, welches Problem in dieser Arbeit gelöst werden soll und auf welches Ziel hingearbeitet wird. Ferner wird ein Weg zu dem Ziel aufgezeigt und analysiert, welche Herausforderungen die Spielregeln für diesen Weg bieten.

4.1 Problem und Implementierungsziel

Ziel ist, eine KI für das Spiel Pokémon Showdown zu entwickeln, die mit wenigen Anpassungen alle Regelwerke auf professionellem Niveau spielen kann. Professionelles Niveau meint, dass sie jeden Spieler in mindestens 50% der Kämpfe besiegen kann. Da das Spiel stochastisch ist, ist eine Gewinnuote von 100% ausgeschlossen, wenn der Gegner gut genug spielt. Idealerweise sollte die KI auch eigens Strategien entwickeln, von der idealerweise menschliche Spieler lernen können. Da Strategien häufig in der Teamauswahl geplant und entwickelt werden, muss die KI dafür eigenständig Teans zusammenstellen können, anstatt vorgefertigte Teams zu verwenden.

Die Algorithmen, die vergleichbare Erfolge in einem anderen Spiel vorweisen kann, sind AlphaGo, AlphaGo Zero und AlphaZero beim Spiel Go. Diese Algorithmen sind in der Lage, professionelle Spieler zu schlagen und entwickeln neue Strateien, aus denen professionelle Spieler lernen. [39]

Aufgrund der Erfolge ist es wahrscheinlich, dass das Ziel durch die erfolgreichen Implementierung einer der Algorithmen erreicht wird. Deshalb fällt die Wahl auf diese wesensgleiche Algorithmen. Das Ziel ist eine erfolgreiche Implementierung von AlphaGo, AlphaGo Zero oder AlphaZero. Leider reichen die Ressourcen der

Arbeit nicht aus, um dieses Ziel zu erreichen. Darum wird an einem fertigen Teilprogramm gearbeitet, auf das weiter aufgebaut werden kann, um das Ziel zu erreichen.

4.2 Herausforderungen

Die Algorithmen AlphaGo, AlphaGo Zero und AlphaZero bauen alle auf MCTS auf. (Siehe Kapitel 3) Für die Implementierung eines MCTS bestehen allerdings einige Herausforderungen, die sich aus den Spielregeln ergeben. In diesem Unterkapitel werden die Herausforderungen, die das Kampfsystem von Pokémon für die Programmierung einer KI birgt, aufgezählt und erklärt. Die Erklärung erfolgt mit einem Vergleich zu Go, ein Brettspiel, das keine der genannten Schwierigkeiten mit sich bringt.

Dieser Unterabschnitt ist von anderen wissenschaftlichen Arbeiten zu dem Thema inspiriert. ([28, 27]) Im folgenden Kapitel 5 werden verwandte Arbeiten danach bewertet, wie sie mit diesen Herausforderungen umgehen.

4.2.1 Unperfekte Information

Bei Go wissen beide Spieler bei jedem Zug, wie der aktuelle Status ist. Der Status ist das Spielfeld und wo sich welche Figuren befinden. Alle Informationen sind beiden Spielern bekannt, weshalb der Status perfekt erkannt und weiterverarbeitet werden kann. [28]

Bei Pokémon sind nicht alle Informationen über den Status bekannt. Während man bei Go alles über die gegnerischen Figuren weiß, fehlt bei Pokémon Showdown Wissen über die Attacken, Statuswerte, Fähigkeiten und Trageitems des gegnerischen Teams. Diese Informationen sind relevant für den Status, da sie die Siegeswahrscheinlichkeit und die gegnerischen Zugoptionen und Entscheidungen mitbestimmen.

Miquel Llobet Sanchez beschreibt dieses Problem als Grund, weshalb ein MCTS nicht auf Pokémon Showdown angewendet werden kann. Folglich ist auch eine Implementierung von AlphaZero unmöglich. [28]

4.2.2 Normal-Form

Bei Go wird in der Extensive-Form gespielt. Das heißt, dass die Kontrahenten abwechselnd ihren Zug auswählen mit komplettem Wissen über die vergangenen Züge. Der Ausgang der eigenen Züge wird nicht durch gegnerische Entscheidungen beeinflusst, wodurch sich eine KI bei Go sicher sein kann, welcher Status nach einem bestimmten Zug erreicht wird.

Die Art und Weise, mit der in Pokémon Züge ausgewählt werden, ist einer der Gründe, weshalb laut Miquel Llobet Sanchez der AlphaZero-Algorithmus keine KI für Pokémon Showdown bilden kann. Beide Trainer wählen ihren Zug gleichzeitig ohne Wissen der gegnerischen Entscheidung aus. Diese Form von Zugwahl in einem Spiel wird "Normal-Form"genannt. Das Problem ist, dass die Entscheidung des Gegners die Folgen der eigenen Entscheidung beeinflusst. [28]

Angenommen, die KI hat ein Pokémon der Spezies Lahmus auf dem Feld, welches unter anderem die Psycho-Attacke Psychokinese und die Wasser-Attacke Surfer beherrscht. Sein Gegner hat ein Pokémon der Spezies Geowaz auf dem Feld. Wenn der Gegner sein Geowaz nicht auswechselt, ist Surfer der bessere Angriff, da Geowaz von Wasserangriffen mehr Schaden nimmt als von Psychoangriffen. Wenn der Gegner aber ein Sleimok im Team hat, kann er es einwechseln. Sleimok erhält von Psychokinese mehr Schaden als von Surfer. Wenn der Gegner zu Sleimok auswechselt, ist Psychokinese der beste Angriff. Die richtige Entscheidung hängt somit teilweise von der gegnerischen Entscheidung ab und die ist unbekannt.

Das ist nur ein simples Beispiel, um die Problematik zu veranschaulichen. In der Praxis der Pokémon-Kämpfe ist die Problematik noch viel komplizierter. Lahmus hat schließlich zwei weitere Attacken, der Gegner eventuell weitere kampffähige Pokémon und wenn Geowaz nicht ausgewechselt wird, kann es eine von vier Attacken einsetzen. Außerdem hat man selten Gewissheit über die Optionen des Gegners, da Teile des gegnerischen Teams erspekuliert sind. Der Gegner weiß häufig nicht mit absoluter Sicherheit, ob das Lahmus der KI überhaupt die Angriffe erlernt hat.

4.2.3 Zufallsereignisse

Go ist ein deterministisches Spiel, weil hier keine Zufälle stattfinden. Deshalb kann eine KI genau berechnen, in welchen Zustand das Spiel kommt, wenn sie einen bestimmten Zug vollführt.

In jedem Pokémon-Kampf gibt es dagegen Zufälle wie Volltrefferquoten, Genauigkeitswharscheinlichkeiten und einige Zusatzeffekten von Attacken treffen auch mit bestimmten Wahrscheinlichkeiten ein. Dadurch ist es unmöglich, zu wissen, wie der Kampfstatus nach einer Runde ist, selbst wenn man perfekte Information über den aktuellen Status und genaues Wissen über die gegnerische Entscheidung hätte. Laut Miquel Llobet Sanchez ist das der dritte Grund, weshalb der AlphaZero-Algorithmus nicht bei Pokémon-Kämpfen verwendet werden kann. [28]

4.2.4 Zorua und Zoroark

In der fünften Generation werden die Spezies Zorua und Zoroark eingeführt. Diese Spezies haben die Fähigkeit Trugbild, welches ihr Erscheinungsbild auf dem Kampffeld so ändert, dass sie für den gegnerischen Trainer wie ein anderes Pokémon des gegnerischen Teams aussehen. Wenn ein Gegner einer KI eines der Pokémon einsetzt, erhält die KI eine falsche Information. [27]

Dass die KI manchmal falsche Informationen erhält, kann auch als eine weitere versteckte Information begriffen werden. Manchmal ist unklar, ob das gegnerische Pokémon ein verstecktes Zorua oder Zoroark ist oder das tatsächlich erkennbare Pokémon.

Kapitel 5

Verwandte Arbeiten

Bevor eine KI geschrieben wird, muss erst klar sein, wie weit die Forschung bei KIs für das Spiel Pokémon Showdown bereits fortgeschritten ist. In diesem Kapitel werden bereits vorhandene Arbeiten zu dem Thema analysiert. Ziel der Analyse ist es, Kenntnis über den Forschungsstand zu erlangen, damit diese Arbeit nicht sprichwörtlich das Rad neu erfindet. Es muss klar sein, wo man am Forschungsstand weiter entwickeln kann und welche hilfreichen Erkenntnisse und Programme es bereits gibt.

5.1 KIs auf Github

Auf Github sind Quellcodes verschiedener KIs für Pokémon Showdown veröffentlicht. ([26, 46, 37, 1])

Die meisten dieser KIs bewerten Zustände von Kämpfen und versuchen, Zustände mit einer möglichst hohen Wertung zu erreichen. Dabei verwenden sie zur Bewertung meistens einfache Funktionen, die nur die verbleibenden KP der Pokémon verwenden. Da auch andere Faktoren auch eine Rolle spielen, aber größtenteils nicht berücksichtigt werden, haben diese hier klare Schwachpunkte und sind stark verbesserungswürdig.

Es gibt eine KI mit öffentlichem Quellcode, die auch andere Faktoren in die Statusbewertung einbezieht. Der Titel der KI ist „Percymon“ . Im Code ist die Kommunikation mit dem Server und die Ermittlung vom Kampfstatus von dem KI-Algorithmus abgegrenzt. Der KI-Algorithmus implementiert eine Funktion, die den aktuellen Kampfstatus überreicht bekommt und die Zugentscheidung zurückgibt.

Der Kampfstatus enthält nur die für die KI zugänglichen Informationen, da sie als Client mit dem Showdown-Server kommuniziert und somit keinen Zugang auf den kompletten Kampfstatus hat. Der Code von Percymon enthält den Code, mit dem serverseitig Kämpfe programmiert sind. Mit diesem Code können KIs zukünftige Runden simulieren. Die damit implementierten KI-Algorithmen haben einige Zugentscheidungen fest im Code verankert, folglich wird menschliches Strategiewissen verwendet. Es gibt „TODO“ -Kommentare, die zeigen, dass Teile noch nicht implementiert sind. Der Code ist komplett auf die Generationen 6 und 7 ausgerichtet. [37] Dieses Repository ist in einem Artikel dokumentiert, der nicht in einem wissenschaftlichen Journal veröffentlicht ist. [19]

Zu diesem Repository gibt es einen Fork mit dem Titel „Lance“ . In diesem wird ein MCTS implementiert, der einige Probleme aufweist. Die Herausforderung, dass das gegnerische Team unbekannt ist, ist teilweise gelöst. [1]

Zu einem Repository mit dem Titel Pokemon Showdown AI Client (PSAC) gibt es einen Artikel, der einen Wettbewerb sieben verschiedener KIs beschreibt, die alle das Problem haben, ausschließlich KP in der Wertung zu verwenden. Der Code beinhaltet ein Framework für KIs, die Pokémon Showdown spielen. Jedoch hält sich das Framework nicht an die Regeln, da es den KIs das gesamte Wissen über den aktuellen Status vermittelt einschließlich dem gegnerischen Team, das die KI den Regeln nach nicht kennen darf. ([26, 27])

Ein erwähnenswertes Repository ist ein Fork des PSAC. Hier sind drei zusätzliche KIs mittels des Frameworks implementiert worden, unter anderem eine KI, die MCTS implementiert. Die Brechung der Spielregel durch das Übergeben des gesamten Spielstatus ist in diesem Fork nicht korrigiert. Obwohl eine Implementierung von MCTS ein wichtiger Zwischenschritt zum Erreichen des Ziels ist (Siehe 3 und 4), hilft das Repository nicht bei der Entwicklung, da bei entscheidenden Problemen die Spielregeln anpasst werden, anstatt dass die Probleme gelöst werden. [46]

5.2 KI basierend auf neuronalen Netzen

Kush Khosla, Lucas Lin und Calvin Qi haben eine Arbeit mit dem Titel „Artificial Intelligence for Pokemon Showdown“ geschrieben. [23] Darin beschreiben sie die Entwicklung einer KI, die einen Spielstatus mit einem neuronalen Netz bewertet.

Als Trainingsdaten nutzen sie 22.000 Spielaufzeichnungen von Spielern mit hohem Spielrang. Diese KI hat aber laut Autoren auch Probleme, da sie beim Spielen nicht weiterlernt und grobe Fehler macht. Beispielsweise versucht die KI, mehr als eine Lage des Entry Hazards Tarnsteine zu legen, obwohl bereits eine Lage Tarnsteine auf dem gegnerischen Feld liegt und keine weitere Lage gelegt werden kann. [23]

Dennoch ist die Idee, die Statusbewertung durch ein neuronales Netz durchführen zu lassen, ein guter Ansatz, da hier alle Faktoren eines Status bewertet und ausgewogen einkalkuliert werden. Außerdem ist das Lernen aus Spielaufzeichnungen Teil des AlphaGo-Algorithmus, was die Autoren hier umgesetzt haben. Leider ist der Quellcode nicht auffindbar, weshalb der Ansatz neu implementiert werden müsste, um ihn zu verwenden, zumal die Implementierung Korrekturen bräuchte, damit die identifizierten Probleme gelöst sind.

5.3 Bachelorarbeit von Miquel Llobet Sanchez

2018 hat Miquel Llobet Sanchez eine Bachelorarbeit geschrieben mit dem Ziel, eine erfolgreiche KI für vereinfachte Pokémon-Kämpfe zu entwickeln. [28]

Der Ansatz, einen AlphaZero-Algorithmus zu implementieren, stammt ursprünglich von ihm. Er kommt allerdings zu dem Schluss, dass es nicht möglich sei, den AlphaZero-Algorithmus auf Pokémon Showdown anzuwenden. Die Unvollständigkeit an Informationen über den aktuellen Zustand, die Stochastik und die gleichzeitige Optionenauswahl würden es unmöglich machen, dass der Algorithmus funktionieren könne. [28]

5.4 Minimax-KI von Robert A. Mills

Mills veröffentlicht 2018 fast zeitgleich mit der Bachelorarbeit von Llobet (Siehe 5.3) ein Paper mit dem Titel „A Deep Learning Agent for Games with Hidden Information“. In diesem analysiert er die im Kapitel 4 genannten Schwierigkeiten und sucht nach Lösungen.

Er bezieht sich auf vorherige Arbeiten und bemängelt, dass diese zu wenige Informationen über ein Spiel verwenden und wichtige Daten über die Statuswerte und Movepools der Pokémon ignorieren. Um hier eine Verbesserung zu erzielen,

nutzt er den World2Vec-Algorithmus, um die verschiedenen Rollen der Pokémon zu identifizieren. Seine Idee basiert auf der Theorie, dass Pokémon basierend auf den Daten ihrer Spezies verschiedene Rollen in einem Team einnehmen. Word2Vec soll diese Rollen basierend auf öffentlichen Spielstatistiken identifizieren. Diesen Algorithmus zusammen mit Hill-Climbing verwendet Mills, um effektive Teams zusammenzustellen.

Darauf basierend entwickelt er eine KI, die mit dem Minimax-Algorithmus entscheidet, bei dem die Kampfzustände mit einer Form neuronaler Netze bewertet werden.

Seiner eigenen Einschätzung nach ist die Bewertung der Kampfzustände ein Erfolg, nicht aber die Zuordnung der Spezies zu ihren Rollen. Die KI lernt und verbessert seine Spielweise und er beobachtet, dass die KI bekannte Strategien neu entdeckt, diese aber noch nicht richtig umsetzt. Aufgrund seiner limitierten Resourcen und dem hohen Zeitaufwand, die KI ein Spiel spielen zu lassen, kann er die KI nicht lange genug spielend lernen lassen, damit sie ein professionelles Niveau erreicht. Als zukünftige Arbeit schlägt er die Implementierung von MCTS für Pokémon Showdown vor.[30]

5.5 Zusammenfassung

Aktuell gibt es keine KI, die auf professionellem Niveau spielt und sich an die Regeln hält. Insgesamt ist die Forschung also nicht besonders weit. Es gibt aber gute Ansätze, auf die aufgebaut werden kann.

Das Problem, dass das gegnerische Team nicht bekannt ist und somit auch nicht der aktuelle Zustand, ist in den angeführten Arbeiten ungelöst. Einige der KIs verstoßen sogar gegen die Regeln, um das Problem zu umgehen. ([26, 46]) In anderen Arbeiten wird das Problem komplett ignoriert. ([3, 27])

Der Ansatz, einen AlphaZero-Algorithmus anzuwenden, wird weiter verfolgt. Es gibt einige Probleme, AlphaZero anzuwenden, weshalb Miquel Llobet Sanchez davon ausgeht, dass es nicht möglich ist. [28] Wenn aber die Probleme gelöst werden, kann ein AlphaZero-Algorithmus für Pokémon Showdown implementiert werden. Dafür gibt es eine eigene Idee, die im folgenden Kapitel näher erklärt wird.

Kapitel 6

Eigener Implementierungsplan

In diesem Kapitel wird ein Plan entworfen, um mit der begrenzten Zeit dem Ziel aus Kapitel 4.1 möglichst nahe zu kommen. Dabei werden auch Lösungen für die Herausforderungen des Kapitels entwickelt. Dieser Plan wird dann in der Implementierungsphase der Arbeit umgesetzt, wobei Abweichungen vom Plan möglich sind.

6.1 Setting

Mit AlphaZero als Weg zum Ziel wird sich die Arbeit darauf fokusieren, diesen Algorithmus zu implementieren. Wie das vorherige Kapitel 5 zeigt, ist der Stand der Technik einer erfolgreichen Implementierung des Algorithmus weit entfernt. Deshalb werden einige Bestandteile der KI komplett neu implementiert. Einzelne Elemente und Codefragmente von bereits bestehenden KIs werden dann verwendet, wenn sie ein vorhandenes Problem bereits gelöst haben, beispielsweise die Kommunikation mit dem Server.

Die Arbeit ist auf drei Monate ausgelegt. Es wird nicht davon ausgegangen dass diese Zeit reicht, um das Ziel zu erreichen, AlphaZero komplett zu implementieren. Deshalb fokussiert sich die Arbeit darauf, bei der Implementierung dem Algorithmus möglichst nahe zu kommen, sodass kommende Arbeiten zu dem Thema darauf aufbauen können.

6.2 Verwendete Algorithmen

Zunächst wird definiert, welche erwägten Algorithmen aus Kapitel 3 implementiert werden sollen. Weil AlphaZero in anderen Spielen professionell spielen kann, wird eine Implementierung dieses Algorithmuses angestrebt. Für AlphaZero wird ein Verlaufbaum vorrausgesetzt, der auch bei MCTS verwendet wird. Darum wird als Zwischenetappe MCTS implementiert. AlphaZero beinhaltet neuronale Netze, darum werden diese ebenfalls verwendet.

Obwohl AlphaZero ein Nachfolger von AlphaGo und AlphaGo Zero ist, werden diese Algorithmen nicht verwendet. Bei ihnen werden die neuronalen Netze mit Spielaufnahmen menschlicher Spieler trainiert. Das ist technisch möglich, da es eine Datenbank von Aufzeichnungen gibt und in einer verwandten Arbeit diese Trainingsart bereits implentiert ist. (Siehe 5.2) Jedoch würde diese Funktion zusätzlichen Implementierungsaufwand bedeuten, der vermieden wird.

Nähere Informationen zu den Algorithmen einschließlich Quellenangaben sind im Kapitel 3.

6.3 Teamanalysierer

Ein wichtiger Schritt, um das Problem des unbekannten, gegnerischen Teams zu umgehen, ist, die bekannten Informationen zu sammeln. Dafür wird ein Teilprogramm entwickelt. Dieses kennt den bisherigen Kampfverlauf und das Team der KI. Aufgabe des Teilprogramms ist es, alle Informationen über das gegnerische Team zu sammeln, die garantiert stimmen.

Wenn der Gegner beispielsweise in einer Runde ein Pokémon einwechselt und es vorher nicht bekannt ist, kann dieses Teilprogramm dadurch die Spezies und Form des Pokémon bestimmen. Das Programm berechnet auch Statuswerte und identifiziert Trageitems und Fähigkeiten. Diese können die Statuswerte eines Pokémon beeinflussen, ohne dass es der gegnerische Trainer merkt. Dadurch wird es besonders kompliziert, das Teilprogramm mit Berücksichtigung für Trageitems und Fähigkeiten zu implementieren.

Um die Implementierung einfacher zu machen, wird die Implementierung auf Generation eins ausgerichtet. In Generation eins gibt es keine Trageitems und keine Fähigkeiten. Es ist außerdem in den ersten beiden Generation möglich, alle

Statuswerte eines Pokémon auf den jeweiligen Maximalwert anzupassen, was auch empfohlen wird. Es gibt keinen Grund, die Statuswerte nicht zu maximieren, weshalb der Teamanalysierer davon ausgehen kann, dass der Gegner dies auch macht. Das vereinfacht die Implementierung noch mehr, da nur noch die Spezies und Attacken analysiert werden müssen.

Wenn die KI für Generation eins funktioniert und sie an eine andere Generation angepasst werden soll, muss auch dieses Teilprogramm angepasst werden. Diese Anpassung kann in zukünftigen Arbeiten über KI für Pokémon Showdown geschehen.

6.4 Teamsimulationen

Damit die gegnerischen Entscheidungen simuliert werden können, muss sein Team bekannt sein, was meistens trotz der Teilinformationen vom Teamanalysierer nicht der Fall ist.

Algorithm 1 SimulateTeam

```
1:  procedure SIMULATETEAM
2:      probability ← 1
3:
4:      for each member in opponents team do
5:          for each choices to make per team member do
6:              choosenOption ← TeamAnalyser.getChoosenOption(member, choice)
7:              optionOdds ← getOptionOdds(state)
8:
9:
10:             if choosenOption = null then
11:                 choosenOption ← getRandomOption(optionOdds)
12:             else
13:                 probability ← probability · optionOdds[choosenOption]
14:
15:             state ← state.addOption(member, choice, choosenOption)
16:
17:     probability ← probability · TeamAnalyser.getProbabilityDuringBattle()
```

Die Teamauswahl kann als eine Reihe von Entscheidungen aufgefasst werden. Ein Spieler oder ein Zufallsalgorithmus entscheidet, welche Spezies und Formen verwendet werden, welche Attacken die Pokémon erlernt haben, und wie er die

Statuswerte der Pokémon anpasst. Je nach Regelwerk werden auch weitere Entscheidungen getroffen, die in ihrer Weise nicht fundamental von anderen Entscheidungen abweichen.

Durch die Auffassung des gegnerischen Zustands als Reihe von Entscheidungen, die teils unbekannt sind, ergibt sich die Möglichkeit, diese im Sinne des MCTS zusätzlich zu den zu dem zukünftigen Kampfverlauf zu simulieren. Algorithmus 1 zeigt den Plan für die Teamsimulation. Jedes Pokémon des gegnerischen Teams wird durch eine Reihe von solchen Entscheidungen erstellt. In Generation eins wird die Spezies des Pokémon und seine vier Attacken gewählt, und die Statuswerte werden angepasst. In den folgenden Generationen kommen weitere Entscheidungen hinzu. In der Simulation wird für jedes Pokémon jede Entscheidung simuliert. Wenn eine Teamentscheidung durch das eben genannte Teilprogramm bereits bekannt ist, wird diese übernommen. Ansonsten wird sie gemäß des MCTS zufällig gewählt. Die Teamentscheidungen hängen miteinander zusammen, weshalb bei jeder Entscheidung der Status mit einbezogen wird, der durch die vorherigen Entscheidungen entsteht.

Dabei sollten auch Hinweise auf den Aufbau des gegnerischen Teams berücksichtigt werden, die im Laufe des Spiels entstehen. Wenn der Gegner in einer Runde eine Attacke seines Pokémons nicht wählt, kann das daran liegen, dass sein Pokémon diese Attacke nicht erlernt hat. Auch die Wahrscheinlichkeiten von zufälligen Ereignissen werden von der Teamauswahl beeinflusst. Idealerweise sollte die KI aus dem bisherigen Kampfverlauf Rückschlüsse auf das gegnerische Team ziehen. Durch die Simulation des Teamaufbaus sind solche Hinweise nicht berücksichtigt.

Um die KI in diesem Aspekt klüger zu machen, wird deshalb nicht nur die Zusammenstellung seines Teams simuliert, sondern auch berechnet, wie wahrscheinlich die bisherigen Ereignisse mit dem simulierten Team sind. Diese Berechnung findet in Zeile 17 statt. Das Team bekommt die Wahrscheinlichkeit der bisherigen Ereignisse als Wertung. Die Wertung V vom Team T wird mit dieser Formel berechnet:

$$V_T = \prod_{i=1}^{N} P_i(T) \tag{6.1}$$

Diese Formel iteriert durch alle N geschehenen Zufallsereignisse i mit bekanntem Ausgang und berechnet deren Wahrscheinlichkeit P_i mit der Annahme, dass der

Gegner das gewertete Team T gewählt hat. Die Zufallsereignisse umfassen sowohl gegnerische Entscheidungen, als auch stochastische Ereignisse im Kampf wie beispielsweise Volltreffer. Auch bekannte Entscheidungen in der Teamauswahl werden berücksichtigt, was in Zeile 13 von Algorithmus 1 geschieht. Zufallsereignisse, deren Ausgang simuliert wird, werden nicht mitberechnet. Ihre Wahrscheinlichkeit wird bereits dadurch miteinbezogen, dass wahrscheinliche Ausgänge häufiger simuliert werden.

Im Kampfverlauf kommen immer mehr Events hinzu. Durch die zusätzlichen Multiplikatoren werden die Wertungen der möglichen Teams immer kleiner. Da dies für die Wertungen aller Teams gilt und die Wertungen für einen relativen Vergleich untereinander verwendet werden, hat dies keinen Effekt. Dadurch, dass einige Teams in immer mehr werdenden Situationen höhere Chancen für den Eventausgang haben, werden die relativen Wertungsunterschiede immer größer. So kann über die Dauer eines Kampfes immer genauer bestimmt werden, welche Teams am realistischsten sind.

6.5 Verwaltung der simulierten Teams

Durch den Teamanalysierer und den Teamsimulator kann ein mögliches gegnerisches Team simuliert werden. Die KI sollte seine Züge aber nicht auf ein mögliches, gegnerisches Team auslegen, sondern möglichst viele und hauptsächlich auf die gegnerischen Teams mit einer hohen Übereinstimmungswahrscheinlichkeit. Darum sollten immer mehrere simulierte Teams mit ihren jeweiligen Wertungen im Algorithmus verwendet werden.

Der Algorithmus 2 zeigt, wie die KI die simulierten Teams verwendet. Die KI legt vor einem Kampf ein Array möglicher Teams an, die der Gegner haben könnte und speichert es in den Zeilen 2 bis 5. Dann beginnt der Kampf. Je nach Regelwerk und Generation gibt es ein Team Preview oder der Gegner schickt sofort ein Pokémon in den Kampf (Siehe 2.8). In beiden Fällen wird die Spezies und Form von mindestens einem gegnerischen Pokémon bekannt. Der Teamanalysierer erfasst diese Information in Zeile 9. Alle simulierten Teams, die der neuen Information widersprechen, sind nicht mehr möglich. Deshalb werden diese Teams in Zeile 11 identifiziert und in Zeile 12 mit neuen simulierten Teams ersetzt, die den gegebenen Informationen nicht widersprechen. In Zeile 13 wird bei jedem Team die Wertung aktualisiert.

Algorithm 2 ManageTeams

```
1: procedure ManageTeams
2:     teamAnalyser ← InitTeamAnalyser()
3:     simulatedTeams ← Array of simulatedTeamsAmount teams
4:     for each index in simulatedTeams do
5:         simulatedTeam[index] ← SimulateTeam()
6:
7:     while Battle is still going do
8:         round ← loadRound()
9:         teamAnalyser.update(round)
10:        for each team in simulatedTeams do
11:            if !teamAnalyser.isTeamPossible(team) then
12:                replace team in array with a new simulated Team
13:            team.updateProbability(round)
14:
15:        makeTurn()
16:    where simulatedTeamsAmount is an integer constant.
```

6.6 Implementierung des MCTS

Der nächste Schritt wird sein, den MCTS zu implementieren, auf den AlphaZero aufbauen kann. Eine Besonderheit dieser Implementierung wird die Simulation sein, die nicht nur auf zukünftigen, gegnerischen Entscheidungen basiert, sondern auch die Zufallsereignisse des Spiels und die vergangene Teamauswahl simuliert.

Durch die soeben gelisteten Komponenten gibt es eine Auswahl an Teams, die der Gegner haben könnte. Ein Team taucht mit der Häufigkeit auf, mit der es basierend auf den gesichterten Informationen zusammengestellt werden würde und hat eine Wertung entsprechend der Wahrscheinlichkeit, mit der der bisherige Spielverlauf basierend auf dem gegnerischen Team verläuft. Dadurch ist alles vorbereitet, damit die gegnerische Teamauswahl in einem Simulationsschritt vom MCTS simuliert werden kann.

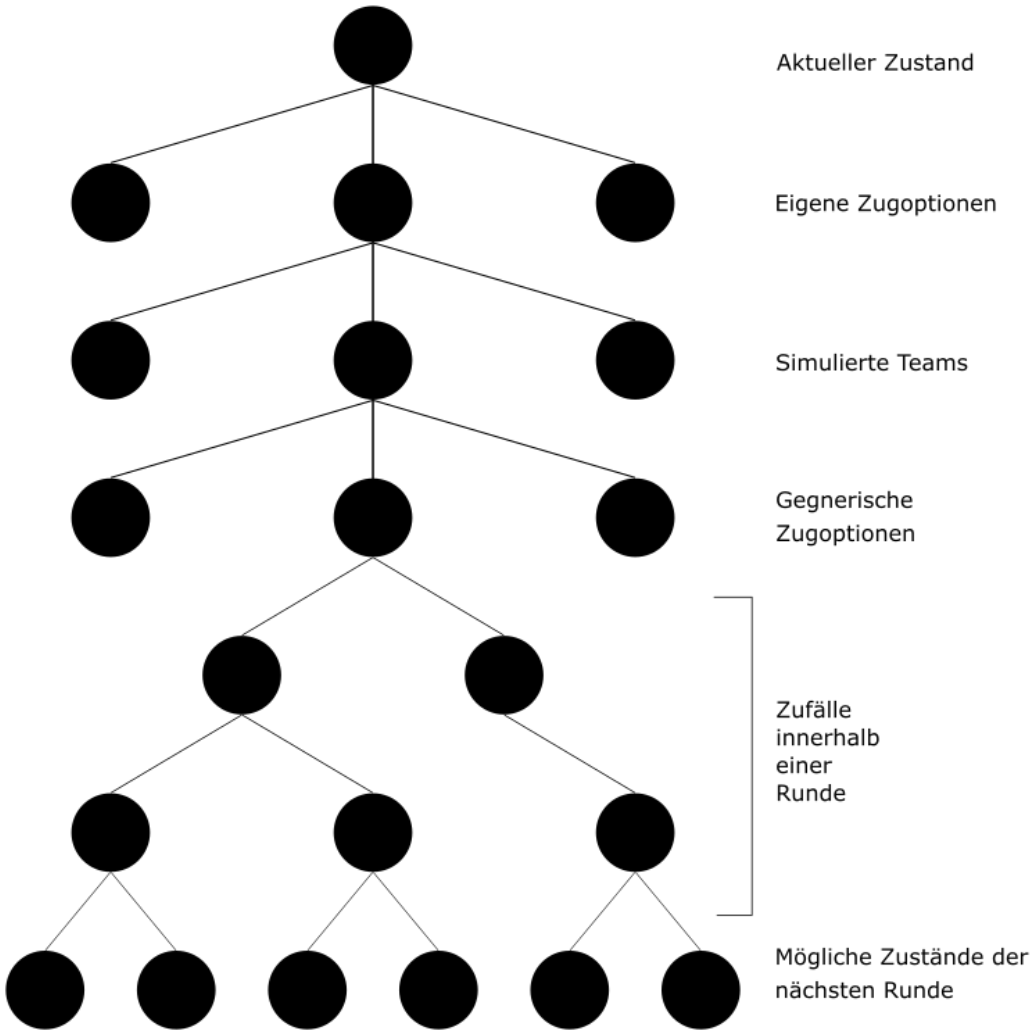

Abbildung 6.1: Diese Abbildung zeigt den Verlaufbaum, der von der MCTS-Implementierung verwendet wird. Es entspricht dem Baum aus Abbildung , da er gleich verwendet wird. Die erste Verzweigung entsteht durch die eigenen Zugmöglichkeiten. In der zweiten Verzweigung werden verschiedene mögliche Szenarien betrachtet, wie das gegnerische Team zusammengestellt sein könnte. Der Baum verläuft nach dieser Verzweigung unter der Annahme, dass der Gegner genau das entsprechende Team besitzt. Die dritte Verzweigung entsteht durch die aktuellen Zugmöglichkeiten des Gegners. Weil danach die Zugentscheidung beider Trainer feststeht, wird die Runde ausgeführt. Stochastische Ereignisse innerhalb der Runde führen zu weiteren Verzweigungen. Die Knoten nach der Runde befinden sich in der nächsten Runde. Ab der zweiten Runde geht der Verlaufbaum weiter mit Verzweigungen für den eigenen Zug, den gegnerischen Zug, und weiteren stochastischen Ereignissen im Spiel. Eine Verzweigung für mögliche Szenarien ist in zukünftigen Runden nicht nötig, da bereits ein Szenario als Annahme dient.

Die Implementierung orientiert sich an dem in Abbildung 6.1 skizzierten Verlaufbaum. Wenn der MCTS eine Simulation startet, wählt er zuerst eine Zugoption der KI. Als nächsten Schritt wird das gegnerische Team selektiert. Dafür wird aus dem

Array der simulierten Teams ein zufälliges Team gewählt. Die Wahrscheinlichkeit P, dass ein Team T gewählt wird, wird mit der folgenden Formel berechnet:

$$P_T = \frac{V_T}{\sum_{j=1}^{M} V_j} \tag{6.2}$$

Die Wahrscheinlichkeit P_T ist somit gleich der Wertung V_T relativ zu der Summe der Wertungen V_j aller M Teams, durch die mit j iteriert wird. Damit werden Teams, die den bisherigen Spielverlauf am besten begründen, am ehesten für Simulationen verwendet. Für den restlichen Teil des Verlaufbaums wird davon ausgegangen, dass der Gegner genau dieses Team verwendet.

Der dritte simulierte Verlaufsabzweigung ist die gegnerische Zugwahl in dieser Runde. Dieser Verzweigung muss nach der Bestimmung des gegnerischen Teams geschehen, da ohne ein genaues Team die Zugoptionen unbekannt sind.

Nach diesem Verlaufszweig sind beide Teams und beide Zugwahlen definiert. Basierend darauf, kann die KI mit dem öffentlichem Spielquellcode den kommenden Zug simulieren. Dabei wird die Spielstochastik berücksichtigt.

Nachdem die Runde simuliert ist, offenbart sich ein möglicher Zustand in der nächsten Runde. Von dem Punkt aus kann der MCTS weitersimulieren, indem für jede Runde die eigene Entscheidung, die gegnerische Entscheidung und der resultierende Rundenverlauf mit Spielstochastik simuliert wird.

Der Ausgang einer Simulation wird für die Zugoption in der aktuellen Runde gespeichert. Durch die vielen Verzweigungen zwischen zwei Runden ist es relativ unwahrscheinlich, dass eine Runde genau gemäß einer Simulation verläuft. Deshalb lohnt es sich nicht, den Ausgang für spätere Runden zwischenzuspeichern.

Eine Arbeit von Browne et. al. gibt eine Übersicht über Anpassungsmöglichkeiten des MCTS. Hier werden auch Wege aufgezeigt, wie eine Implementierung von MCTS für ein nicht-deterministisches Spiel funktionieren kann. Um mit unbekannten Informationen umzugehen, kann man über verschiedene mögliche Stati iterieren, bei denen die unbekannten Informationen mit Annahmen ersetzt werden. Dieser Ansatz findet sich in den simulierten Teams wieder. Eine Möglichkeit, mit zufälligen Ereignissen umzugehen ist, den Ausgang der Ereignisse bei dem Algorithmus deterministisch festzulegen, was hier nicht geschieht. [7]

6.7 Einsatz von neuronalen Netzen

Die erstrebten Algorithmen setzen neuronale Netze ein, um die Wahrscheinlichkeiten für Zugentscheidungen, und die Gewinnchance aus einem Zustand zu bestimmen.

Der Autor geht nicht davon aus, dass die begrenzten Ressourcen ausreichen, um den Algorithmus mit den nötigen neuronalen Netzen auszustatten. Deshalb werden zunächst keine neuronalen Netze verwendet. Die Komponente wird durch einen Prototypen ersetzt, der immer für jede Option die gleiche Wahrscheinlichkeit zurückgibt.

Damit die KI richtig funktioniert, muss der Prototyp schließlich mit einem neuronalen Netz ersetzt werden. Dieses muss auch trainiert werden, indem die KI aus seinen Kämpfen Trainingsdaten erhebt, mit denen das neuronale Netz trainiert werden kann.

Kapitel 7

Implementierung

In diesem Kapitel wird der Verlauf der Implementierung dokumentiert. Die Implementierung verläuft nach dem in Kapitel 6 dokumentierten Plan. Der dabei entstandene Prototyp wird im Folgenden TalonBot genannt.

7.1 Architektur und Codebasis

Es gibt KIs, die serverseitig implementiert sind. Diese haben Zugriff auf Informationen über das gegnerische Team, was nicht regelkonform ist. (Siehe 5) Deshalb sollte die KI auf der Client-Seite implementiert werden, damit kein Zugriff auf unerlaubte Informationen möglich ist.

Das Repository „Lance“ erfüllt diese Anforderungen. Zudem beinhaltet es bereits den Code, der das serverseitige Kampfobjekt berechnet, um zukünftige Runden zu simulieren. Das wird die Implementierung des MCTS vereinfachen, da zukünftige Runden simuliert werden müssen. [1]

Das Repository bietet insgesamt eine gute Codebasis für die Anforderungen. Die Kommunikation mit dem Server funktioniert und die eintreffenden Nachrichten über den Kampf werden verarbeitet. Die grobe Architektur von Lance ist in Abbildung 7.1 abgebildet.

Es werden auch schon Informationen über das gegnerische Team analysiert. Der Code ist auf Kämpfe in den Regelwerken RandomBattle für Generation sechs und sieben ausgelegt, kann aber auch andere mit anderen Regelwerken kämpfen, sofern eine Datei existiert, die ein Team für dieses Regelwerk beinhaltet. [1]

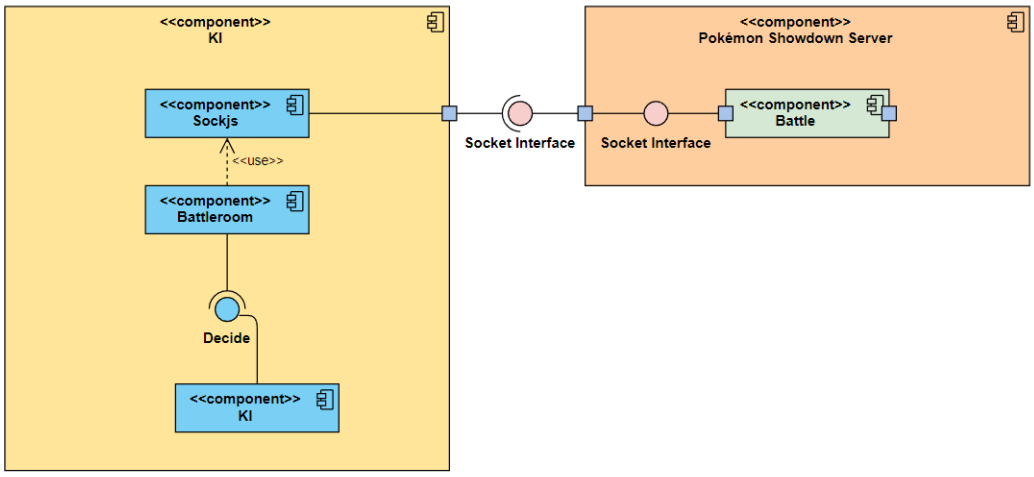

Abbildung 7.1: Das Komponentendiagramm zeigt die grobe Architektur von Lance. Lance kommuniziert über einen WebSocket mit dem Server. Dabei wird die Library sockjs verwendet. Die Nachrichten vom Server werden an die Klasse Battleroom weitergeleitet. Der Battleroom nutzt die darin enthaltenen Informationen, um das serverseitige Kampfobjekt soweit es geht zu replizieren. Wenn eine Zugentscheidung benötigt wird, ruft BattleRoom die „decide“-Methode einer KI auf. Es sind mehrere KIs implementiert, die alle diese Methode implementieren, weshalb diese als Interface dargestellt werden kann. Welche KI verwendet wird, wird durch eine Flag definiert. Die Entscheidung wird vom Battleroom zu einer vom Server verständlichen Nachricht übersetzt und über sockjs an den Server geschickt. (Abbildung: Eigens, Bezugsprogramm: [1])

Diese Codebasis enthält eine Implementierung des MCTS, dessen Code übernommen und weiterentwickelt wird. In dieser ist das Problem der unbekannten Information nur zum Teil gelöst. In allen RandomBattle-Regelwerken werden die Teams beider Trainer zufällig zusammengestellt, daher der Name. Der Movepool der verschiedenen Pokémon ist in RandomBattles stark reduziert. Die Wahrscheinlichkeiten, welche Attacken ein Pokémon hat, sind im Zufallsalgorithmus klar definiert. Somit kann der MCTS von Lance bei gegnerischen Pokémon mit bekannter Spezies genau ausrechnen, welche Attacke mit welcher Wahrscheinlichkeit verwendet wird. Aus dem gegnerischen Verhalten werden keine Rückschlüsse auf sein Team gezogen, wie es im Plan im vorherigen Kapitel 6 vorgesehen ist.

7.2 Verwendung von Servercode

Lance beinhaltet einen Ordner „battle-engine“, in dem Codefragmente des Servers von Pokémon Showdown enthalten sind. Der Code ist an die Anforderungen des Projekts angepasst. Das ist problematisch, da kontinuierliche Änderungen im

öffentlichen Servercode schwerer auf den angepassten Code übertragbar sind, als wenn der Code ohne Änderungen übernommen wird.

Hinzu kommt, dass die abgewandelte Variante von Lance zulässt, dass mehr als vier möglich beherrschte Attacken pro Pokémon gespeichert werden. Für jede Attacke gibt es eine im Quellcode definierte Wahrscheinlichkeit, dass das Pokémon diese beherrscht. So kann die KI die verschiedenen Angriffe, die der Gegner haben kann, miteinbeziehen. [1]

Für den eigenen Implementierungsplan wird das Feature nicht benötigt, da die unbekannten Attacken in den Teams gespeichert werden. Jedoch wird der serverseitige Code benötigt, um den aktuellen Kampfstatus zu beschreiben, zukünftige Runden zu simulieren und legale Teamoptionen zu überprüfen. Deshalb wird der Ordner „battle-engine“ gelöscht und der komplette Servercode aus dem Repository https://github.com/Zarel/Pokemon-Showdown geklont und der restliche Code an die Änderung angepasst.

7.3 Replizierung des Kampfstatus

Lance kommuniziert als Client mit dem Server und nutzt die Informationen, die es vom Server erhält, um den aktuellen Kampfstatus zu replizieren. Dafür wird ein Objekt erstellt, welches der Klasse „Battle“ entspricht. Die Klasse Battle ist Teil des in Abschnitt 7.2 beschriebenen Servercodes, sowohl in der von Lance angepassten Version, als auch in der übernommenen Version ohne Änderungen.

Ziel der Implementierung ist es, dass dieses Objekt zu jedem Zeitpunkt eines Kampfes dem Kampfobjekt des Servers so genau wie möglich entspricht, ohne direkten Zugriff auf das serverseitige Objekt zu haben. Bei der Kommunikation mit dem Server erhält die KI Informationen über Änderungen im Kampfstatus. Wenn ein Pokémon in einem Kampf eine Attacke einsetzt, wird die Information darüber vom Server geschickt. Die KI speichert dann im Battle-Objekt, dass das Pokémon die Attacke beherrscht und repliziert die Wirkung der Attacke. [1]

Bei der Weiterentwicklung wird der Verarbeitungsprozess der BattleLogs verbessert. Wenn eine Zeile den Einsatz einer Attacke beschreibt, wird die Methode „runMove“ des Battle-Objekts aufgerufen, die eine Attacke ausführt. Da die gleiche Methode auf Serverseite sämtliche Effekte der Attacke hervorruft, ist dies eine effektive Methode, sämtliche Effekte einer Attacke zu replizieren. Da der Ausgang

einer Attacke stochastische Teile hat, werden die entsprechenden Methoden so überschrieben, dass der Ausgang dem des Servers entspricht.

Es werden auch weitere Änderungen unternommen, die insgesamt die Replizierung des Kampfstatus verbessern, beispielsweise durch die Behebung von Bugs.

7.4 Teamsimulator und MCTS

Teamsimulationen werden gemäß den Kapiteln 6.4, und 6.4 implementiert. Eine kleine Abweichung vom Plan ist, dass Zufallsereignisse während dem Kampf nicht in der Wertung eines Teams berücksichtigt werden, sondern nur die gegnerischen Entscheidungen. Diese Abweichung ist eine Vereinfachung, die für die Funktionsweise des Prototypen nicht von Bedeutung ist. In Generation eins werden die Zufallsereignisse nicht von den unbekannten Komponenten eines Teams beeinflusst, da Trageitems und Fähigkeiten erst in späteren Generationen eingeführt werden. Einzige Ausnahme ist die Volltrefferwahrscheinlichkeit von Attacken, die von dem Initative-Statuswert des Anwenders beeinflusst wird. Da der Prototyp von einem maximierten Initiative-Statuswert ausgeht (Siehe 6.4), ist auch dies unbedeutend. Weil die unbekannten Komponenten keinen Einfluss auf die Wahrscheinlichkeiten von Zufallsereignissen haben, ist der Wahrscheinlichkeitsfaktor, der von den Zufallsereignissen ausgeht für alle simulierten Teams gleich, folglich bleiben durch das Weglassen dieses Faktors die relativen Wahrscheinlichkeiten gleich.

Lance's Implementierung des MCTS wird so umgeschrieben, dass der Teamsimulator verwendet wird. Der Verlaufbaum aus Abbildung 6.1 dient hier als Vorbild. Um den weiteren Kampfverlauf zu simulieren, reicht es, das Battle-Objekt zu kopieren, das simulierte gegnerische Team einzufügen und dann Runde für Runde die performMove-Methode des Objekts aufzurufen, mit der die künftigen Entscheidungen der Trainer festgelegt werden.

7.5 Wahrscheinlichkeitsberechnung

Bei der Simulation zukünftiger Runden entscheidet das Battle-Objekt selbst, welchen Ausgang Zufallsereignisse haben. Da der Code dem des Servers entspricht und die Wahrscheinlichkeiten gleich bleiben und auch für Teamsimulationen die

Wahrscheinlichkeiten zunächst nicht bekannt sein müssen (Siehe 7.4), müssen die Wahrscheinlichkeiten nicht mit zusätzlichem Code berechnet werden.

AlphaZero sieht vor, dass für simulierte Spielentscheidungen neuronale Netze verwendet werden, um die Wahrscheinlichkeiten für die möglichen Ausgänge der Entscheidungen zu berechnen. (Siehe 3.5) Bei Pokémon Showdown gibt es Entscheidungen während des Kampfes und Entscheidungen über das Team, die vor einem Kampf getroffen werden.

Zunächst werden drei neuronale Netze implementiert, die für Teamentscheidungen verwendet werden. Ein Netz ist für die Levelbestimmung eines Pokémon, das zweite für die Attackenwahl und das dritte für die Spezieswahl. Als Input bekommen die Netze ein unfertiges Team und diese geben als Output die Wahrscheinlichkeiten für die Entscheidungsoptionen aus. Es wird angenommen, dass die Entscheidungen im Teambauprozess immer in der gleichen Reihenfolge stattfinden. Alle Entscheidungen, die gemäß der Reihenfolge vor der aktuellen Entscheidung stattfinden, werden im Input verwendet. Beim Output wird für jede Entscheidungsmöglichkeit ein Neuron verwendet.

Die Regelwerkauswahl wird weiter beschränkt, sodass nur noch „gen1random“ unterstützt wird. Bei Regelwerken mit dem Suffix „random“ werden die Teams der Trainer automatisch vom Server erstellt und der Quellcode der Erstellung ist zusammen mit dem restlichen Servercode in der KI verbaut. Um Trainingsdaten für die ersten neuronalen Netze zu erstellen, muss nur eine Methode mit öffentlichen Quellcode aufgerufen werden, was das Trainieren dieser Netze besonders einfach macht.

Zwei weitere neuronale Netze werden vom Algorithmus AlphaZero gefordert. Eines gibt die Wahrscheinlichkeiten aus, mit denen Zugmöglichkeiten gewählt werden, das andere die Gewinnwahrscheinlichkeit für die KI. Als Input bekommen die Netze den jeweiligen Kampfstatus. Dieser wird erst in einen Vektor übersetzt. Der Zustand besteht aus den jeweils sechs Pokémon der beiden Spieler. In zukünftigen Generationen kommen weitere Eigenschaften wie Feldeffekte und Wettereffekte hinzu, die können aber zunächst ausgeblendet werden. Die Werte der zwölf Pokémon werden einzeln zu Vektoren übersetzt.

Der Vektor des Zustands ist also die Zusammensetzung von zwölf Vektoren, die jeweils ein Pokémon beschreiben. Dabei wird eine bestimmte Reihenfolge eingehalten, da die Position der Pokémon eine Rolle spielt, sowohl im Spiel als auch im

Vektor. Beim Netz der Gewinnchancenbestimmung kommen die Pokémon der KI zuerst, dann die Pokémon des Gegners. Beim Netz für die Zugmöglichkeiten kommt das Team des Akteurs zuerst. Bei beiden Teams wird jeweils das aktive Pokémon an erste Stelle gesetzt.

Die einzelnen Pokémon werden durch ihre Spezies, permanente Attacken, Level, KP, permanente Statuswerte und primäre Statusprobleme beschrieben. Aktive Pokémon werden zusätzlich durch aktuell verfügbare Attacken, Statuswerte mit Modifikationen und sekundären Statusproblemen beschrieben. Die Spezies ist eine Enumeration, also hat der Vektor des Pokémon einen Index je mögliche Spezies. Der Wert dieses Teilektors ist 1 bei dem Index der Spezies und 0 bei allen anderen. In zukünftigen Generationen müssen hier auch Formen der Spezies berücksichtigt werden. Bei numerischen Eigenschaften wie Level und KP wird ebenfalls ein solcher Teilvektor erstellt für alle möglichen Werte mit einer 1 beim realen Wert. Bei den KP werden auch alle Werte auf 1 gesetzt, die niedriger als der reale Wert sind, da sich ähnlich hohe KP-Werte nicht wesentlich unterscheiden sollen. Die verschiedenen primären und sekundären Statusprobleme werden ebenfalls durch solche Teilvektoren dargestellt mit einer 1 beim zustimmenden Statusproblem. Einige Statusprobleme haben auch eigene numerische Eigenschaften, zum Beispiel die Dauer in Runden vom Statusproblem Schlaf. Diese werden in einem zusätzlichen Index übergeben welches beim Nicht-Existieren des Problems 0 ist und beim Existieren den entsprechenden Wert so wie er ist übernimmt. Statuswerte werden auf die gleiche Weise im Vektor gespeichert. Attacken sind wie die Spezies eine Enumeration, jedoch mit mehreren wahren Werten. Außerdem haben Attacken eigene Eigenschaften, ob sie genutzt wurden und somit dem Gegner offenbart sind und auch die Angriffspunkte (AP) sind eine Eigenschaft. Die Attacken werden wie Spezies in einem Teilvektor gespeichert mit dem Unterschied, dass es je mögliche Attacke drei Indexe gibt. Einer wird 1 gesetzt, wenn das Pokémon die Attacke beherrscht, ein weiteres gibt die AP an und ein weiteres ist eins wenn die Attacke dem Gegner bekannt ist. Dies geschieht je Pokémon ein mal mit den permantent erlernten Angriffen und bei aktiven Pokémon ein mal mit den aktuell verfügbaren Attacken, da diese temporär geändert werden können. Aufgrund eines Bugs, der die AP auf unrealistische Werte setzt werden AP schließlich auf 0 gesetzt. Da diese nur selten wichtig sind, ist dies eine akzeptable Vereinfachung.

Da beide Netze Wahrscheinlichkeiten bestimmen und diese immer zwischen 0 und 1 liegen, wird eine sigmoide Transferfunktion verwendet. Diese beschreibt Hagan et. al mit der folgenden Formel.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7.1}$$

Die Gewinnwahrscheinlichkeit wird mit einem Ausgabeneuron ausgegeben. Das andere neuronale Netz hat ein Ausgabearray. Für jeden Angriff im Spiel, sowie für jede Spezies wird ein Feld in dem Ausgabearray angelegt. Die Wahrscheinlichkeit für eine Attackenwahl wird mit dem entsprechenden Ausgabeneuron repräsentiert. Ein Wechsel in ein anderes Pokémon wird mit dem Neuron seiner Spezies dargestellt.

Um die neuronalen Netze zu trainieren, wird eine Javascript-Datei geschrieben, die die KI gegen sich selber spielen lässt. Dabei wird lokal ein Battle-Objekt erstellt und seine send-Methode, die mit den Clients kommuniziert, wird so überschrieben, dass die Nachrichten direkt den entsprechenden Methoden der KI-Instanzen übergeben werden. Dabei entsteht kein Netzverkehr. Zusätzlich werden alle Zwischenstände gespeichert. Sobald ein Kampf zu Ende gespielt ist, werden die Netze mit gespeicherten Zwischenständen trainiert.

Das Netz für die Gewinnwahrscheinlichkeit bekommt als Trainingsdaten Zwischenstände eines Kampfes sowie die Sicht einer der beiden Spielerseiten. Als Zielwert wird 1 erwartet, wenn die gewählte KI-Instanz gewinnt und 0, wenn sie verliert. Da sich die Zwischenstände eines Kampfes durch die gleichen Teams ähneln, besteht die Gefahr, dass bei der Verwendung aller Zustände das Netz das Verliererteam dem Verlieren zuordnet und das Gewinnerteam dem Gewinnen, ohne wichtige Details zu berücksichtigen, die sich während einem Kampf ändern, beispielsweise die verbleibenden KP der Pokémon. Um der Gefahr zu entgehen, werden nur je KI-Instanz drei zufällige Zwischenstände als Trainingsdaten verwendet. Dieser Ansatz wird von anderen Arbeiten übernommen. [42, 40]

Das Netz, welches Zugentscheidungen vorhersagt, bekommt bei jedem Durchlauf die gleichen Zwischenstände. Dieses soll nacheinander die Zugentscheidungen beider Seiten vorhersagen mit einer eins bei der tatsächlichen Entscheidung und nullen bei allen anderen.

Kapitel 8

Evaluation

Nach der Implementierung der KI wird die KI überprüft, ob und wie weit mit der KI das in Kapitel 4 definierte Ziel erreicht wird. Ein Teil vom Ziel ist es, dass ein lauffähiger Prototyp entsteht. Das Wichtigste ist eine möglichst hohes Spielerstärke der KI, die sich dadurch definiert, wie häufig die KI gegen andere KIs und menschliche Spieler gewinnt. Die Implementierung von MCTS und AlphaZero sind lediglich Meilensteine, um dieses Ziel zu erreichen (Siehe Kapitel 4). Außerdem soll die KI mit möglichst wenigen Codeanpassungen weitere Regelwerke beherrschen können. Zunächst werden die trainierten neuronalen Netze auf erwartungsgemäße Funktionsweise überprüft, da darauf basierend entschieden wird, ob sie für den Rest der Evaluation verwenden werden oder mit einfachen Funktionen ersetzt werden.

8.1 Ealuation der neuronalen Netze

Bevor die Tests gegen andere Spieler beginnen, werden erst die neuronalen Netze trainiert und evaluiert. Die Funktionsweise der neuronalen Netze wird im Abschnitt erklärt.

Beim Training verwenden die KIs zunächst eine einfache Funktion verwendet, die nur die relativen KP der Pokémon zu ihren maximalen KP berücksichtigt, ähnlich wie es auch von Lee et al. [26] gelöst wird. Bevor das neuronale Netz zur Bestimmung der Gewinnwahrscheinlichkeit trainiert ist, ist dies eine vorübergehende Lösung, da die KP ein wichtiger Faktor bei der Bestimmung der Gewinnwahrscheinlichkeit sind.

Nach drei Tagen, in denen die neuronalen Netze trainieren, ist bei Stichproben kein Erfolg bei der Bestimmung der Gewinnwahrscheinlichkeit erkennbar. Das Netz gibt teilweise 1 aus, was 100% Gewinnwahrscheinlichkeit entspricht, obwohl die entsprechende KI den Kampf schließlich verliert. Da dieser Zustand auch nach Ablauf selbst gesetzter Fristen besteht, wird die Übergangslösung beibehalten, welche die vereinfachte Funktion ist, die nur KP berücksichtigt.

Das Netz, welches Züge vorhersieht, hat bei Stichproben gezeigt, dass auch dieses nicht perfekt ist. Zwar erhalten die Angriffe Hyperstrahl, Surfer und Bodyslam erwartungsgemäß hohe Wertungen. Allerdings werden manchmal Zugoptionen mit einer unrealistisch hohen Wahrscheinlichkeit von über 99% angezeigt, was für einen Misserfolg des Trainings spricht. Da dieses neuronale Netz zum Teil wie erwartet funktioniert, wird es beibehalten.

Welche Gründe der Misserfolg hat, kann nur mit Hypothesen analysiert werden, da für ihre Überprüfung in der Arbeit die Zeit fehlt. Eine Möglichkeit ist ein Potenzieller Fehler bei der Überführung des Kampfzustands in den Eingabevektor. Wenn an einer Stelle der Funktion ein Index zu wenig oder zu viel gesetzt wird, verschieben sich alle folgenden Eingabewerte und bekommen neue, falsche Bedeutungen. Ebenfalls ist möglich, dass das Training schlicht nicht ausreicht. Auch eine Theorie ist, dass die Erwartungswerte falsch gesetzt werden.

8.2 Evaluation der Lauffähigkeit

Der auf das Git-Repository https://github.com/1522784/Talonbot veröffentlichte Code des Talonbots ist mit Javascript geschrieben und somit mit node ausführbar. Mit „node bot.js“ verbindet sich der Prototyp mit einem Server von Pokémon Showdown. Sobald der Prototyp herausgefordert wird in einem Regelwerk, dass es beherrscht, nimmt es an und spielt den folgenden Kampf. Diese Funktionsweise unterscheidet sich wenig von Lance. [1]

Damit ist die KI lauffähig, allerdings können Programmierfehler nicht ausgeschlossen werden. Um zu testen wie häufig Talonbot Fehler ausgiebt, wird ein Programm geschrieben, welches den Talonbot hundert mal gegen eine Random-KI antreten lässt. Der Quellcode ist dem Trainingscode in Kapitel 8.1 nachempfunden. Das heißt, dass der Kampf lokal läuft ohne eine Netzwerkverbindung zwischen Server und Client. Somit können externe Fehler durch die Verbindung ausgeschlossen

werden. Die Random-KI wählt in jedem Zug eine zufällige Zugentscheidung, wobei jede Möglichkeit eine gleiche Wahrscheinlichkeit hat. Das ist ein sehr einfacher Bot mit wenig Quellcode, wodurch die Wahrscheinlichkeit auf Fehler seitens der Gegner-KI minimiert wird. Als Regelwerk dient „gen1random“.

Wenn die KI in einem Kampf eine Ausnahmebehandlung auslöst, wird der Kampf abgebrochen und das Programm merkt sich den Fehler. Wenn ein Kampf ohne Fehler zuende gespielt wird, wird das ebenfalls gespeichert. So soll eine Fehlerquote bestimmt werden.

In dem Versuch verlaufen 100 der 100 Kämpfe ohne Fehler.

Das Ergebnis zeigt, dass der Prototyp fehlerfrei oder mit wenig Fehlern läuft, da keiner der 100 Kämpfe eine Ausnahmebehandlung auslöst. Allerdings sollte das Ergebnis vorsichtig bewertet werden. Zum einen ist die Anzahl der Testspiele mit hundert sehr gering. Außerdem spielt der Randombot anders als andere Spieler, wodurch andere Situationen im Kampf entstehen. Bei anderen Gegnern können eher Situationen auftreten, dessen Verarbeitung auf einen Fehler stoßen.

Außerdem ist es auch möglich, dass ein Programmierfehler keine Fehlerausgabe zur Folge hat, sondern nur das Abbild des Kampfes verzerrt. Dann würde die KI Entscheidungen basierend auf einer falschen Situation treffen, was wiederum zu unvorteilhaften Entscheidungen führt, ohne dass eine Ausnahmebehandlung stattfindet. Solche Fehler sind durch den Versuch nicht feststellbar, da nur Ausnahmebehandlungen gezählt werden können. Sie beeinträchtigen dafür die Lauffähigkeit nicht, sondern nur die Spielerstärke der KI.

8.3 Evaluation der Spielerstärke

Um die momentane Spielerstärke zu evaluieren, wird die KI in Testkämpfen getestet. Zunächst spielt sie gegen andere KIs, anschließend gegen menschliche Spieler. Als Regelwerk für die Kämpfe wird weiter „gen1random“ verwendet.

8.3.1 Evaluation gegen andere KIs

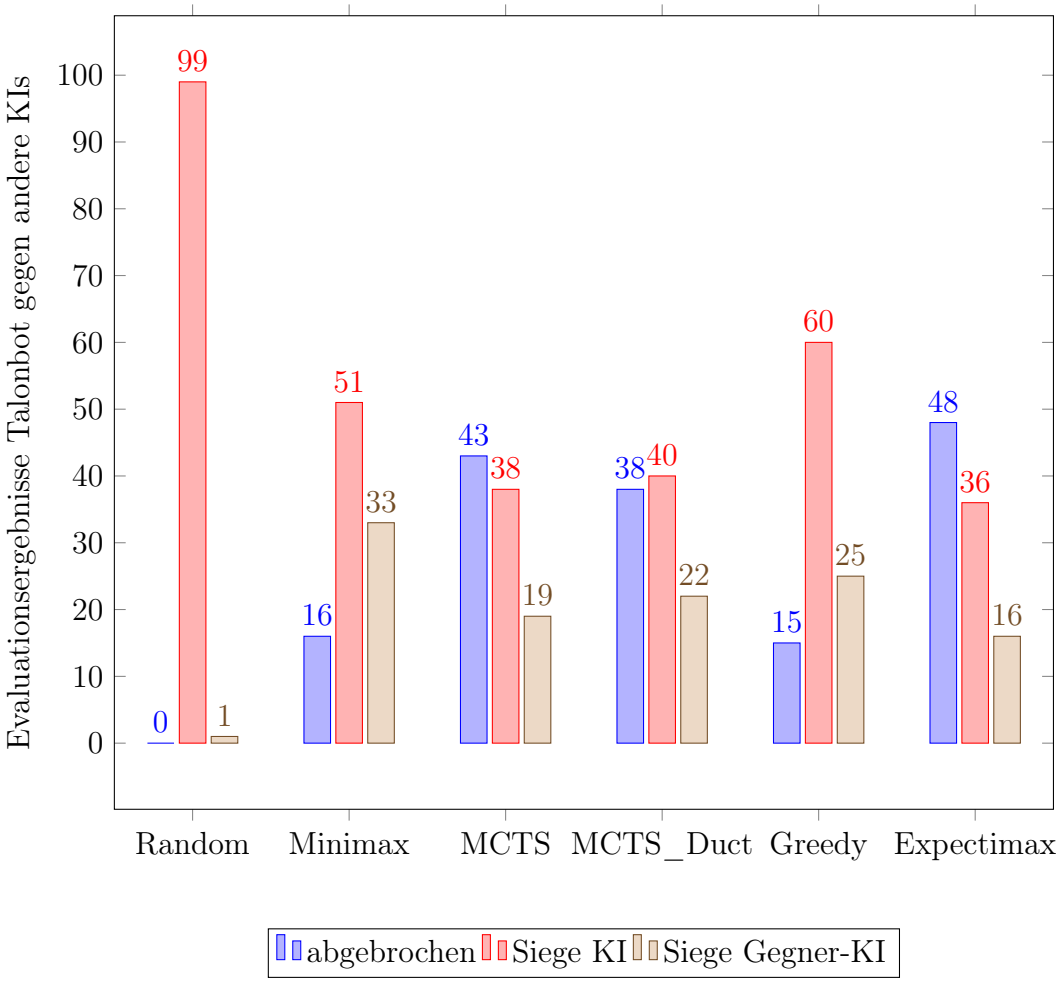

Abbildung 8.1: In diesem Diagramm werden Versuchsergebnisse dokumentiert. Bei dem Versuch spielt der programmierte Talonbot gegen verschiedene KIs von LANCE jeweils 100 mal. Die Balken „Siege KI“ zählt die Siege des Talonbots, während in den Balken „Siege Gegner-KI“ die Siege der jeweiligen gegnerischen KI doumentiert sind. Wenn ein Spiel in dem Balken „abgebrochen„ gezählt wird, dann heißt das, das bei dem Spiel eine Ausnahmebehandlung stattfindet. Dabei wird nicht unterschieden, wie die Ausnahmen zustande kommen. Sie können von dem Versuchsprogramm kommen oder von einer der KIs. Ein abgebrochenes Spiel hat keinen Gewinner, weshalb diese Spiele in keiner der beiden „Siege“-Balken gelistet werden. Da ein Spiel entweder in einem Sieg für genau eine KI enden kann oder abgebrochen wird, ergeben immer alle Werte einer Gruppierung summiert 100.

In dem Lauffähigkeitstest des vorangegangenen Abschnitts 8.2 werden auch die Gewinner der Kämpfe ausgegeben. In dem Test gewinnt der Talonbot 99 der 100 Spiele und verliert eines. Die Ursache für die Niederlage ist nicht identifizierbar, da nur die Ergebnisse der Kämpfe gespeichert werden, nicht die Kampfverläufe.

Der Talonbot wird auch gegen weitere KIs getestet, die bereits in Lance implementiert sind. Die Methode ist die gleiche. Der Talonbot und die Vegleichs-KI spielen 100 mal gegeneinander und die Ergebnisse werden gezählt. Mögliche Ergebnisse sind der Abbruch des Spiels, der Sieg des Talonbots oder einem Sieg der gegnerischen KI. Da alle Ergebnisse mit einer gewissen Wahrscheinlichkeit eintreten können, ist dies ein *Zufallsexperiment*. Wenn abgebrochene Kämpfe ausgeblendet werden, haben die übrigen Kämpfe zwei mögliche Ausgänge mit unbekannter Wahrscheinlichkeit. Wenn eine KI besser spielt als die gegnerische KI dann ist seine Gewinnchance höher als 50% bei den nicht abgebrochenen Kämpfen. Wenn beide KIs gleich gut spielen, ist die Gewinnwahrscheinlichkeit beider Seiten jeweils 50%. Mit dem Experiment werden die Wahrscheinlichkeiten ermittelt, um zu analysieren, ob Talonbot besser spielt als die anderen KIs oder schlechter.

Die in Lance programmierten KIs werden mit möglichst wenig Anpassungen übernommen, sodass sie möglichst dem Stand vor der Arbeit entsprechen. Einige Änderungen sind notwendig, da Lance auf eine andere Generation ausgelegt ist und der Unterschied Ausnahmebehandlungen auslösen kann. (Siehe 7.1 und 2.5) Außerdem kommt es manchmal zu Programmabbrüchen durch den zu hohen summierten Speicheranspruch zweier KIs. Bei den Änpassungen wird stets darauf geachtet, dass die Funktionsweisen der Algorithmen sich nicht grundlegend verändern.

Die Ergebnisse der Testläufe sind im Diagramm 8.1 dokumentiert. Wenn während einem Kampf eine Ausnahmebehandlung stattfindet, wird der Kampf abgebrochen.

Richard Mohr definiert den Schätzwert einer Wahrscheinlichkeit als die Häufigkeit des entsprechenden Ereignisses innerhalb eines Zufallexperiments in Relation zu der Anzahl der Versuche des Zufallsexperiments. [31] Da bei abgebrochenen Spielen nicht bestimmt werden kann, wer ohne den Abbruch gewonnen hätte, werden im Folgenden nur nicht abgebrochene Spiele als Teil des Zufallsexperiments wahrgenommen. Die Schätzwerte für die Gewinnchance des Talonbots bei einem nicht abgebrochenen Spiel lauten somit auf den Prozent gerundet 99% gegen die Random-KI, 61% gegen Minimax, 67% gegen die MCTS-Implementierung von Lance, 65%

gegen eine MCTS-Implementierung, die ein vereinfachtes Server-Objekt verwendet, 71% gegen einen Greedy-Bot und 69% gegen eine Expectimax-KI.

Gegen alle KIs gewinnt der Talonbot die Mehrheit der nicht abgerochenen Testkämpfe. Das deutet darauf hin, dass die KI im Regelwerk „gen1random“ besser als die vorherigen KIs ist, bewiesen ist dies mit dem Experiment aber nicht.

Bei der Bewertung der Ergebnisse sollte berücksichtigt werden, dass die in LANCE bereitgestellten KIs nicht auf Kämpfe der ersten Generation ausgelegt sind im Gegensatz zum Talonbot. Der Battleroom von LANCE füllt die Wissenslücken über das gegnerische Team mit Daten über die Attacken und Spezies, die bei der Teamerstellung verwendet werden können. Diese Daten sind auf Generation 6 und 7 ausgelegt. Somit erwarten diese KIs manchmal Attacken, die es in Generation 1 nicht gibt. Das verschafft dem Talonbot einen Vorteil.

8.3.2 Evaluation gegen menschliche Spieler

Bei diesem Versuch spielt die KI gegen Freiwillige, um seine Spielerstärke in den Kontext menschlichen Spielens zu setzen. Jeder Teilnehmer spielt in bis zu zehn Kämpfen gegen die KI, deren Ausgänge dokumentiert werden. Ein Spiel kann mit einem Sieg für die KI einem Sieg für den Spieler, einem Abbruch durch einen Programmabsturz der KI oder einem Abbruch durch externe Fehler, beispielsweise einen Verbindungsabbruch, ausgehen. Das Experiment ist somit ein ähnliches Experiment wie das des vorherigen Abschnitts 8.3 und ist auch ein Zufallsexperiment.

Um eine möglichst hohe Variation der Gegner zu haben, gibt es für die Teilnahme bei dem Test keine zu erfüllende Bedingung. Folglich haben die Teilnehmer eine unterschiedliche Spielerstärke, was sie schwer vergleichbar macht.

Um die Teilnehmer besser einzuordnen, wird ihre Spielerstärke eingeordnet. Die Einordnung geschieht einheitlich auf einer Skala von eins bis zehn und wird vom Versuchsleiter unternommen. Der Grund, dass die Teilnehmer sich nicht selbst einordnen, ist, weil Einordnungen, die von verschiedenen Personen durchgeführt werden, weniger einheitlich ist, als wenn alle Einordnungen von derselben Person stammen. So werden die Einordnungen einheitlicher und vergleichbarer. Da es Schätzungen sind, können Fehler bei der Einordnung nicht ausgeschlossen werden. Es werden auch Versuchspersonen angenommen, die noch nie Pokémon Showdown gespielt haben.

Spielerstärke Mensch	Spiele gesamt	Fehler KI	Abbruch Extern	Siege KI	Siege Mensch	Hätte KI erkannt
3	10	2	0	4	4	Nein
4	10	1	1	4	4	Ja
4	4	0	0	2	2	Nein
5	10	0	0	2	8	Ja
5	3	0	0	0	3	Nein
6	10	2	0	5	3	Nein
7	10	0	0	4	6	Nein
8	10	2	0	0	8	Ja
9	10	1	0	2	7	Ja

Tabelle 8.1: In der Tabelle sind Ergebnisse eines Tests dokumentiert, der im Rahmen dieser Arbeit durchgeführt wird. Bei diesem spielen freiwillige Versuchsteilnehmer bis zu zehn mal gegen die KI Talonbot. Jede Zeile der Tabelle steht für einen Versuch mit einem Versuchsteilnehmer. Die zehn Spiele können jeweils ausgehen mit einem Sieg des menschlichen Versuchsteilnehmers, dem Sieg der KI, einen Abbruch durch einen Fehler der KI oder einen Abbruch, dessen Grund nicht die KI ist. Wie häufig die Ausgänge bei einem Versuch auftreten, ist in der jeweiligen Spalte gelistet. Nach einem Versuch erhält jeder Versuchteilnehmer einen Fragebogen, bei dem unter anderem die Ja-Nein-Frage "Hättest du anhand der Spielweise vermutet oder erkannt, dass der Mitspieler kein Mensch ist, sondern eine KI enthalten ist. Die Antwort der Teilnehmer auf die Frage wird in der letzten Spalte dokumentiert. Bei jedem Versuch beobachtet ein Versuchsleiter die Spiele zwischen Mensch und KI, welcher die Ergebnisse dokumentiert und die Spielweise des Menschen subjektiv auf einer Skala von 1 bis 10 bewertet. Je besser seiner Ansicht nach der Spieler spielt, desto höher die Wertung. Diese Wertung ist in der ersten Spalte dokumentiert.

Bei ihnen ist es trotzdem möglich, dass sie durch Spiele der Hauptreihe Wissen über das Kampfsystem haben, was sich positiv auf ihre Spielerstärke auswirken sollte.

Jede Versuchsperson wird anschließend gefragt, ob sie erkannt oder vermutet hätte, dass ihr Gegner eine KI ist. Die Frage ist mit den Optionen „Ja“, „Nein“ und „Weiß nicht“ beantwortbar. Mit einer Auswertung kann so gedeutet werden, wie menschenähnlich die KI spielt.

Tabelle 8.1 fasst die Ergbnisse des Tests zusammen. Von den 77 gespielten Spielen werden 9 abgebrochen. Die übrigen 68 Spiele enden 23 mal mit einem Sieg für die KI und 45 mal mit einem Sieg für den jeweiligen Versuchsteilnehmer. Ausgenommen abgebrochener Spiele ist der Schätzwert für die durchschnittliche Gewinnquote für die KI etwa 34%. (Siehe Kapitel 12.1 in [31])

Aufgrund der niedrigen Anzahl an Spielen und der hohen Stochastik des Spiels lässt sich durch den Test kein genauer Wert für die Gewinnchance nachweisen. Um aussagekräftigere Ergebnisse zu erzielen, bräuchte man mehr Versuche, was im Rahmen dieser Arbeit aber nicht möglich ist.

Wenn der Schätzwert von 34% für die Gewinnquote ungefähr stimmt, dann spielt die KI unter dem durchschnittlichen Niveau eines Versuchsteilnehmers. Das heißt, dass die KI nicht auf professionellem Niveau spielt und Verbesserungsbedarf besteht.

Das gleiche gilt für die Erkennbarkeit. Vier der neun Teilnehmer behaupten, sie hätten eine KI vermutet oder erkannt. Der Schätzwert dafür ist somit 44%. Die Spielweise der KI ist offenbar menschenähnlich genug, sodass sie nicht von jedem Spieler erkannt wird. Jedoch gelingt es einigen Spielern, die KI als solche zu erkennen, was zeigt, dass die KI auch KI-typisches Verhalten aufweist.

8.4 Auswertung des Fragebogens

Im vorherigen Kapitel 8.3 spielen Versuchsteilnehmer gegen die KI. Nachdem eine Versuchsperson die Kämpfe gegen die KI bestritten hat, werden ihr folgende Fragen gestellt:

„Hättest du anhand der Spielweise vermutet oder erkannt, dass der Mitspieler kein Mensch ist, sondern eine KI?„

„Welche besonderen Schwächen in der Spielweise der KI sind dir aufgefallen?“

„Welche besonderen Stärken in der Spielweise der KI sind dir aufgefallen?“

Die erste Frage wird im vorherigen Abschnitt ausgewertet. Die anderen Fragen können mit Fließtexten beantwortet werden. Durch sie werden Meinungen und Tipps offenbart, die einer Weiterentwicklung des Prototypen helfen können. Im folgenden werden die Fragebögen ausgewertet und die Antworten analysiert. Die Angaben geschehen im Fließtext, jedoch lassen sich ähnliche Aussagen verschiedener Teilnehmer zusammenfassen.

So sagen drei Teilnehmer, dass die KI „kleine“ oder „keine“ Schwächen habe. Das ist anzuzweifeln, schließlich verliert die KI in dem Test die Mehrheit der Spiele.

Ein Teilnehmer kritisiert die technischen Fehler, die für insgesamt acht Abbrüche von Testspielen verantwortlich sind. Ein weiterer Teilnehmer bemängelt, dass die KI zu lange mit der Zugauswahl benötigt. Zwei Spieler kritisieren auch die Abstürze, die nachweislich da sind. Das sind zwar keine Schwächen im Spiel, die die Gewinnrate verringern, aber dennoch wichtige Kritikpunkte an der KI.

An der Spielweise kritisieren vier Teilnehmer, dass die KI zu selten das Pokémon auswechselt. Damit bleibt das Pokémon der KI auf dem Feld, auch wenn es gegen das aktuelle Pokémon des Teilnehmers einen Nachteil hat. Die Verhaltensweise hat auch einen Vorteil, den ein Teilnehmer anführt. Dadurch „präsentiert [die KI] möglichst wenige der eigenen Pokemon um sich nicht dem Gegner zu offenbaren“.

Zwei Teilnehmer kritisieren, dass die KI sogar manchmal in ein falsches Pokémon auswechselt, welches gegen das aktuelle Pokémon des Teilnehmers einen Nachteil hat. Andererseits lobt ein Spieler die „präzise[n] Wechsel aufgrund von Attackenvermutungen“ und ein weiterer Teilnehmer lobt die Auswechslungen ohne nähere Begründung. Dass die Teilnehmer widersprüchliche Meinungen haben, kann an den unterschiedlichen Testspielen liegen. Wenn die KI in den Spielen gegen einen Spieler besonders gute Wechsel ausführt und bei den anderen schlechte Wechsel, kann es zu einem solchen Meinungsbild kommen, weshalb davon auszugehen ist.

Zwei Spieler kritisieren, dass die KI zu anfällig ist gegen ihre gleichartige Strategie, mit Statusattacken die Statuswerte des eigenen Pokémon zu erhöhen, um dann mit dem gestärkten Pokémon die ungestärkten Pokémon der KI zu besiegen. Drei Teilnehmer geben an, dass die KI selbst zu selten Statusattacken einsetzt, sondern fast nur Angriffe einsetzt. Die Verhaltensweise ist durchaus problematisch. Statusattacken bieten Vorteile, die sich nicht in den KP der Pokémon zeigen. Da die KI wie als Bewertungsfunktion nur KP verwendet, kann sie diese Vorteile nicht erkennen, was sie insgesamt schwächt. Ein Teilnehmer fasst diesen Kritikpunkt gut zusammen. Er meint, „[Die KI] nutzt keine eigene Spielstrategie, sondern will den aktuellen Gegner auf schnellsten Wege besiegen“. Es gibt aber auch eine abweichende Meinung. Im Gegensatz dazu lobt ein Teilnehmer, dass „sie äußerst taktisch vorgegangen ist“.

Auf die Frage nach den Stärken kommt am häufigsten die Antwort, dass die Angriffswahl der KI stets den höchsten Schaden ausrichtet. Im Gegensatz dazu lobt ein Teilnehmer, dass die KI die Attacken abwechslungsreich einsetzt und damit unvorhersehbar ist. Die Verhaltensweise ist mit der KP-basierenden Bewertungsfunktion begründbar. Dass die KI abwechslungsreiche Attacken einsetzt, kann an den stochastischen Simulationen liegen. Da bei den Simulationen ebenfalls Volltreffer und Attackenverfehlungen einsetzten können, ist auch die Attackenwahl der KI stochastisch. Zwei Teilnehmer loben ebenfalls, dass die KI Statusattacken auswählt, die die KP der eigenen Pokémon regenerieren. Auch diese Verhaltensweise ist mit der Bewertungsfunktion zu erklären.

Zwei Teilnehmer nennen als Stärke, dass die KI die Initiativwerte der Pokémon kennt und folglich immer weiß, welches Pokémon zuerst agiert. Einer der beiden Teilnehmer lobt auch, dass die KI den Schaden genau kennt, den Attacken ausrichten. Da die KI durch die Nutzung des gesamten Servercodes quasi perfektes Wissen über die Spielregeln hat, einschließlich den Initiativ-Werten, ist eine solche Aussage wenig verwunderlich.

Zwei Aussagen sind für den Autor schwer einzuordnen. Ein Teilnehmer sagt über die Stärken, dass die KI „exzellent situativ reagiert hat“. Ein weiterer Teilnehmer gibt als Stärke nur an, die KI sei „insgesamt gut optimiert‘.

Kapitel 9

Ergebnisse und Ausblick

In der Arbeit wird ein Plan entworfen, eine professionelle KI für das rundenbasierte Spiel Pokémon Showdown zu entwickeln. Der Plan ist eine Abwandlung des Algorithmus AlphaZero und wird teilweise verwirklicht mit der Entwicklung eines lauffähigen Prototypen. Dieser Prototyp ist eine Weiterentwicklung des MCTS Lance und hat die Besonderheit, dass in einem Knotenpunkt des Verlaufbaums Annahmen bezüglich unbekannter Informationen getroffen werden. (Siehe Kapitel 6 und 7)

Ein Test, bei dem die KI gegen andere KIs spielt deutet an, dass die KI in dem ausgewählten Regelwerk besser spielt als alle verglichenen KIs. Da dieser Schluss aus dem Ergebnis eines Zufallsexperiment stammt und auch mit einer besseren Angepasstheit ans Regelwerk begründet werden kann, kann derzeit nicht gesagt werden, ob der Prototyp allgemein besser spielt. Ein weiterer Test gegen menschliche Spieler zeigt, dass die KI zwar menschliche Spieler besiegen kann, jedoch die meisten Spiele verliert. Im Experiment hat der Prototyp eine Gewinnquote von 38%, ausgenommen abgebrochener Spiele. Die KI spielt also nicht auf professionellem Niveau, trifft aber Entscheidungen, die gut genug sind, um Gewinnchancen gegen menschliche Gegner zu haben.

Ein möglicher Grund, warum die KI nicht auf professionellem Niveau spielen kann, ist die Funktion, mit der Zustände bewertet werden. AlphaZero sieht vor, dass für die Bewertung von Spielzuständen ein neuronales Netz verwendet wird. Der Prototyp verwendet stattdessen eine vereinfachte Funktion, wegen der sie sich zu sehr auf KP fokussiert und andere wichtige Faktoren für die Gewinnchance von einem Zustand aus vernachlässigt. Ein weiterer möglicher Grund ist die Tatsache, dass das neuronale Netz, mit dem Zugentscheidungen vorhergesagt werden, nur teilweise funktioniert. Für den Grund des Nicht-Funktionierens des Netzes gibt es eigene

Hypothesen. Eine fehlerhafte Erstellung des Eingabevektors, falsche Erwartungswerte und zu wenig Training sind hier mögliche Gründe. (Siehe 8.1)

Die These von Miuel Llobet Sanchez, dass AlphaZero für Pokémonkämpfe nicht verwendet werden kann, ist durch den lauffähigen Prototypen widerlegt. (Siehe sec:llobet) Dieser verwendet nämlich AlphaZero für Pokémonkämpfe ohne Regeln zu brechen. Nur ist damit nicht bewiesen, dass AlphaZero Pokémonkämpfe auf dem Niveau professioneller Spieler spielen kann. Zukünftige Arbeiten können den Prototypen verbessern. Die Theorie, dass auch eine professionelle KI für Pokémonkämpfe mit dem AlphaZero-Algorithmus möglich ist, ist erst bewiesen, wenn ein Nachfolger nachweislich professionelle Spieler in über 50% der Kämpfe besiegen kann.

Zukünftige Arbeiten können den Prototypen weiterentwickeln. Eine Möglichkeit der Verbesserung sind die neuronalen Netze für die Vorhersage von Zugentscheidungen und die Bewertung von Zuständen. Wenn neuronale Netze installiert werden, dessen Ergebnisse menschlichen Erwartungen entsprechen, kann auch die These überprüft werden, dass diese die Quelle der größten Schwäche der KI ist. Wenn die KI dann nochmal getestet wird und die Gewinnrate höher ist, ist die These weitgehend belegt.

Wenn die Gewinnrate etwa gleich bleibt oder sogar sinkt, ist die These weitgehend widerlegt. Da die Faktenbasis, auf der die Thesen überprüft werden, Zufallsexperimente sind, kann man keine Beweise schaffen. Man kann aber mit größer angelegten Tests einige Thesen als unwahrscheinlich und wahrscheinlich einstufen. Auch dies ist eine Möglichkeit für zukünftige Arbeiten.

Auch kann in Zukunft der Prototyp so weiterentwickelt werden, dass er auch andere Regelwerke spielen kann. Bei Lance gibt es bereits die Möglichkeit, Regelwerke mit selbst definierbaren Teams zu spielen, indem sie ein vorgefertigtes Team aus einer Datei liest. Talonbot hat somit auch diese Option, es fehlt jedoch ein eigenständer Teambauprozess der KI.

Um die KI auch auf Regelwerke anderer Generationen vorzubereiten, sind Erweiterungen der Implementierung notwendig. Der Prototyp beinhaltet Vereinfachungen durch die Beschränkung des Kampfsystems auf die erste Spielgeneration. Diese betreffen besonders den Teamsimulator. (Siehe Kapitel 7). Aus der Tabelle 2.1 ergeben sich folgende Änderungen, die der Prototyp für weitere Generationen benötigt.

In Generation zwei gibt es zusätzliche Anpassungsmöglichkeiten für die Pokémon. Neben Spezies, Level und Attacken kann auch das Geschlecht ausgewählt werden, es gibt die Option eines schillernden Pokémon und die Pokémon können eines von verschiedenen Trageitems tragen. Trageitems können die Chancen auf Zufallsereignisse beeinflussen. Um besser analysieren zu können, welches Trageitem ein gegnerisches Pokémon hat, ist es ab Generation zwei nötig, auch die Chancen auf geschehene Zufallsereignisse bei den Wertungen der simulierten Teams zu berücksichtigen. Das ist im aktuellen Prototyp nicht der Fall. (Siehe Abschnitt 6.4) Trageitems können auch den Schaden von Angriffen beeinflussen. Es ist somit möglich, aus dem Schaden von Attacken Rückschlüsse auf die getragenen Items des gegnerischen Pokémon zu ziehen. Auch diese Funktion, die die KI für die nachfolgenden Generationen verbessern kann, ist nicht implementiert. (Für Informationen zu Generation zwei siehe Quelle [11])

In Generation drei gibt es eine grundlegende Veränderung, wie die Statuswerte der Pokémon angepasst werden können, die in nachfolgenden Generationen beibehalten wird. In den betroffenen Generationen ist es nicht möglich, alle Statuswerte eines Pokémon auf das Maximum zu heben, im Gegensatz zu den Generationen eins und zwei. Da der Prototyp davon ausgeht, dass alle Statuswerte auf dem Maximum stehen, werden wohl Codeänderungen nötig sein, damit die KI auch in diesen Generationen spielen kann. Auch Trageitems und die in Generation drei eingeführten Fähigkeiten können Statuswerte beeinflussen. Eine Herausforderung ist, die Höhe der Statuswerte anhand ihrer Einflüsse auf den bisherigen Kampf zu erkennen und die Anpassungen der Statuswerte ihrer Ursache zuzuordnen. Wenn ein Pokémon einen höheren Statuswert als den Basiswert hat und dies durch den Kampfverlauf belegt ist, kann die Erhöhung des Wertes an den IV, den EV, dem Wesen, dem getragenen Item oder der Fähigkeit des Pokémon liegen. Der Teamsimulator sollte im Idealfall nur noch Teams erstellen können, bei denen das Pokémon durch die Kombination der Faktoren Statuswerte hat, die nicht durch den bisherigen Kampfverlauf widerlegt werden. (Für Informationen zu Generation drei siehe Quelle [12])

Zukünftige Generationen können mit ihren Erneuerungen auch weitere unvorhergesehene Implementierungsmaßnahmen fordern, die hier nicht näher analysiert werden. Durch die Weiterentwicklung kann gezeigt werden, ob der Algorithmus wie in Kapitel 6 dokumentiert, auch in Regelwerken späterer Generationen funktioniert.

Abkürzungsverzeichnis

KP Kraftpunkte

PvP Spieler-gegen-Spieler

Stats Statuswerte

KI Künstliche Intelligenz

MCTS Monte-Carlo Tree Search

FSP Ficticious Self-Play

NFSP Neural Ficticious Self-Play

PSAC Pokemon Showdown AI Client

RPG Rollenspiel

AP Angriffspunkte

VGC Video Game Championships

Tabellenverzeichnis

2.1 Generationenübersicht 7

8.1 Ergebnisse Talonbot gegen menschliche Spieler 48

Abbildungsverzeichnis

2.1 Ablaufdiagramm Pokémon Showdown 9

3.1 Minimax am Beispiel Tic-Tac-Toe 12
3.2 MCTS 13
3.3 Mehrschichtiges neuronales Netzwerk 15

6.1 Von Prototyp verwendeter Verlaufbaum 32

7.1 Komponentendiagramm für Lance 36

8.1 Evaluationsergebnisse Talonbot gegen andere KIs 45

Literatur

[7] Cameron B Browne u. a. „A survey of monte carlo tree search methods“. In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), S. 1–43.

[9] Michael C Fu. „AlphaGo and Monte Carlo tree search: the simulation optimization perspective“. In: *Proceedings of the 2016 Winter Simulation Conference*. IEEE Press. 2016, S. 659–670.

[18] Martin T Hagan u. a. *Neural network design*. Bd. 20. Pws Pub. Boston, 1996.

[20] Johannes Heinrich und David Silver. „Deep reinforcement learning from self-play in imperfect-information games“. In: *arXiv preprint arXiv:1603.01121* (2016).

[21] Kurt Hornik, Maxwell Stinchcombe und Halbert White. „Multilayer feedforward networks are universal approximators“. In: *Neural networks* 2.5 (1989), S. 359–366.

[22] Leslie Pack Kaelbling, Michael L Littman und Andrew W Moore. „Reinforcement learning: A survey“. In: *Journal of artificial intelligence research* 4 (1996), S. 237–285.

[25] Yann LeCun u. a. „Backpropagation applied to handwritten zip code recognition“. In: *Neural computation* 1.4 (1989), S. 541–551.

[27] Scott Lee und Julian Togelius. „Showdown AI competition“. In: *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE. 2017, S. 191–198.

[28] Miquel Llobet Sanchez. „Learning complex games through self play-Pokémon battles“. B.S. thesis. Universitat Politècnica de Catalunya, 2018.

[31] Richard Mohr. *Statistik für Ingenieure und Naturwissenschaftler: Grundlagen und Anwendung statistischer Verfahren*. Bd. 557. expert verlag, 2008.

[32] Chetprayoon Panumate, Shuo Xiong und Hiroyuki Iida. „An Approach to Quantifying Pokemon's Entertainment Impact with Focus on Battle". In: *Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence (ACIT-CSI), 2015 3rd International Conference on*. IEEE. 2015, S. 60–66.

[39] David Silver u. a. „Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815* (2017).

[40] David Silver u. a. „Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), S. 484.

[41] David Silver u. a. „Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017), S. 354.

[42] Marius Stanescu u. a. „Evaluating real-time strategy game states using convolutional neural networks". In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2016, S. 1–7.

[47] Mark HM Winands. „Monte-Carlo Tree Search". In: *Encyclopedia of Computer Graphics and Games* (2015), S. 1–6.

Online Literatur

[1] Richard Chen und A. Melone. *Lance: A Pokemon Showdown AI*. https://github.com/A-Malone/showdownbot. Accessed: 2019-10-29. 2017.

[2] Fullstack Academy. *Monte Carlo Tree Search (MCTS) Tutorial*. Accessed: 2019-10-29. Youtube. 2017. URL: https://www.youtube.com/watch?v=Fbs4lnGLS8M.

[3] Akshay Kalose, Kris Kaya, and Alvin Kim. *Optimal Battle Strategy in Pokemon using Reinforcement Learning*. Standford University, 21. Dez. 2018.

[4] *AlphaGo: The story so far*. Accessed: 2019-10-29. URL: https://deepmind.com/research/case-studies/alphago-the-story-so-far.

[5] *An Introduction to the Video Game Championships*. Accessed: 2019-10-29. URL: https://www.pokemon.com/us/strategy/an-introduction-to-the-video-game-championships/.

[6] *Attacke*. https://www.pokewiki.de/Attacke. Accessed: 2019-01-30.

[8] The Artificial Intelligence Channel. *Deepmind AlphaZero - Mastering Games Without Human Knowledge*. Accessed: 2019-10-29. Youtube. 2018. URL: https://www.youtube.com/watch?v=Wujy7OzvdJk.

[10] *Generation 1*. https://bulbapedia.bulbagarden.net/wiki/Generation_I. Accessed: 2019-02-28.

[11] *Generation 2*. https://bulbapedia.bulbagarden.net/wiki/Generation_II. Accessed: 2019-02-28.

[12] *Generation 3*. https://bulbapedia.bulbagarden.net/wiki/Generation_III. Accessed: 2019-02-28.

[13] *Generation 4*. https://bulbapedia.bulbagarden.net/wiki/Generation_IV. Accessed: 2019-02-28.

[14] *Generation 5*. https://bulbapedia.bulbagarden.net/wiki/Generation_V. Accessed: 2019-02-28.

[15] *Generation 6*. https://bulbapedia.bulbagarden.net/wiki/Generation_VI. Accessed: 2019-02-28.

[16] *Generation 7*. https://bulbapedia.bulbagarden.net/wiki/Generation_VII. Accessed: 2019-02-28.

[17] *Generation 8*.
https://bulbapedia.bulbagarden.net/wiki/Generation_VIII. Accessed: 2019-02-28.

[19] Varun Ramesh Harrison Ho. „Percymon: A Pokemon Showdown Artifical Intelligence“. Accessed: 2019-10-29. 2014. URL: https://varunramesh.net/content/documents/cs221-final-report.pdf.

[23] Kush Khosla, Lucas Lin und Calvin Qi. „Artificial Intelligence for Pokemon Showdown“. https://docplayer.net/63514819-Artificial-intelligence-for-pokemon-showdown.html. Accessed: 2019-10-29.

[24] *Kraftpunkte*. https://www.pokewiki.de/Kraftpunkte. Accessed: 2019-01-30.

[26] Scott Lee und Julian Togelius. *Pokèmon Showdown AI Client*.
https://github.com/scotchkorean27/showdownaiclient. Accessed: 2019-10-29. 2017.

[29] Guangcong Luo. *Pokèmon Showdown*.
https://github.com/Zarel/Pokemon-Showdown. Accessed: 2019-10-31.

[30] Robert A Mills. „A Deep Learning Agent for Games with Hidden Information“.
https://digitalcommons.bard.edu/cgi/viewcontent.cgi?article=1272&context=senproj_s2018. Accessed: 2019-10-29. 2018.

[33] *Pokèmon Games*.
https://bulbapedia.bulbagarden.net/wiki/Pok%C3%A9mon_games. Accessed: 2019-02-28.

[34] *Pokèmon Showdown*. https://pokemonshowdown.com/. Accessed: 2019-02-03.

[35] *Pokèmon(Spezies)*.
https://www.pokewiki.de/Pok%C3%A9mon_(Spezies). Accessed: 2019-01-29.

[36] *Pokémon-Team*. https://www.pokewiki.de/Pok%C3%A9mon-Team. Accessed: 2019-02-03.

[37] Varun Ramesh. *Percymon: A Pokemon Showdown AI*.
https://github.com/rameshvarun/showdownbot. 2017.

[38] Vamsi Sangam. *MiniMax Algorithm*. http://theoryofprogramming.com/2017/12/12/minimax-algorithm/. Accessed: 2019-10-29. 2017.

[43] *Statuswerte*. https://www.pokewiki.de/Statuswerte. Accessed: 2019-01-30.

[44] *Taking Advantage of Team Preview*. https://www.smogon.com/tiers/ou/advantage_team_preview. Accessed: 2019-02-03.

[45] *The AI in Pokemon is a Cheating Bastard*. Accessed: 2019-10-29. URL: https://www.kirsle.net/the-ai-in-pokemon-is-a-cheating-bastard.

[46] Stanislaw Wilczynski. *Pokèmon Showdown AI Client*. https://github.com/sjwilczynski/showdownaiclient. Accessed: 2019-10-29. 2018.