# Untersuchung von Allgemeinen KI-Agenten für das Spiel Poker im General Board Games Framework

Masterarbeit zur Erlangung des Master-Grades
*Master of Science* im Studiengang Wirtschaftsinformatik
an der Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln


| | |
|---|---|
| vorgelegt von: | Tim Zeh |
| Matrikel-Nr.: | 11 097 189 |
| Adresse: | Von-Sparr-Straße 25<br>51063 Köln |


| | |
|---|---|
| eingereicht bei: | Prof. Dr. Wolfgang Konen |
| Zweitgutachter/in: | Prof. Dr. Heinrich Klocke |

| | |
|---|---|
| vorgelegt am: | 30. Juli 2021 |

# Abstract

Das Spiel Poker stellte über Jahrzehnte hinweg eine große Herausforderung für das Forschungsfeld Machine Learning dar. Diese Herausforderung ergab sich zum einen durch die Komplexität des Spiels an sich, aber vor allem durch die unvollständigen Informationen. Dennoch konnten inzwischen verschiedene KI-Pokerspieler das Spiel meistern und zumindest Teile des Spiels lösen. In dieser Arbeit wird untersucht, ob es, auch für nicht auf das Pokerspiel spezialisierte KI-Agenten, möglich ist, das Pokerspiel zu erlernen. Um diese Fragestellung zu beantworten, wurde das Spiel in das General Board Games Framework implementiert und untersucht, ob die verschiedenen Agenten des Frameworks in der Lage sind, das Spiel erfolgreich zu spielen. Im ersten Schritt wurde mit Kuhn Poker eine vereinfachte Variante des Spiels untersucht. Verschiedene Agenten waren in der Lage, zumindest für die Startposition, eine optimale Strategie zu identifizieren, als Folgespieler konnte jedoch keiner der Agenten die optimale Strategie finden. Anschließend wurde das Spiel Texas Hold'em als Zwei-Spieler-Variante betrachtet. Im ersten Schritt wurde ein Benchmark-Agent implementiert, welcher als Vergleichspunkt dienen sollte, da keine optimale Strategie, welche diesen Zweck erfüllen könnte, verfügbar war. Anschließend wurden das Spiel der verschiedenen Agenten gegen den Benchmark Agenten untersucht. Die Agenten konnten gegen den Benchmark-Agenten Gewinne erzielen, wobei die Temporal Difference Agenten bessere Ergebnisse erzielen konnten als die Monte Carlo Agenten. Dies deutet darauf hin, dass die Agenten, zumindest zu einem gewissen Grad, in der Lage waren, das Spiel zu erlernen. Insgesamt wird jedoch deutlich, dass es verschiedene Aspekte gibt, welche das Framework erweitern könnten, um bessere Ergebnisse zu ermöglichen. Erste Schritte, wie die Implementierung eines *Partial States*, welcher den Umgang mit unvollständigen Informationen ermöglicht oder das Training, beziehungsweise die Bewertung einzelner Spielrunden, wurden bereits gemacht. Jedoch gibt es weiteres Potential, wie zum Beispiel die Implementierung eines Counterfactual Regret Minimization Agenten, welcher vermutlich in der Lage gewesen wäre, bessere Ergebnisse zu erzielen.

# Inhaltsverzeichnis

**Abbildungsverzeichnis** **IV**

**Tabellenverzeichnis** **VI**

**Poker Begriffe** **VIII**

**1 Einleitung** **1**
1.1 Problemstellung . . . . . 2
1.2 Methodik . . . . . 2

**2 Poker** **3**
2.1 Allgemeines . . . . . 3
2.2 Kuhn Poker . . . . . 4
2.3 Texas Hold'em . . . . . 8

**3 Grundlagen** **10**
3.1 Reinforcement Learning . . . . . 10
3.2 General Board Games . . . . . 11
3.3 Counterfactual Regret Minimization . . . . . 25

**4 Aktueller Forschungsstand** **32**
4.1 Allgemein . . . . . 32
4.2 Cepheus . . . . . 34
4.3 DeepStack . . . . . 37
4.4 Pluribus . . . . . 40

**5 Kuhn Poker** **43**
5.1 Spiel gegen die optimale Strategie . . . . . 43
5.2 Random Agent . . . . . 47
5.3 MC-N . . . . . 52
5.4 MCTSE . . . . . 55
5.5 Expectimax . . . . . 57
5.6 Lernende Agenten . . . . . 59
5.7 Übersicht & Fazit . . . . . 61

**6 Texas Hold'em Poker** **62**
6.1 Episoden . . . . . 62
6.2 Benchmark-Agent . . . . . 64
6.3 Nicht-lernende Agenten . . . . . 65
6.4 Feature und Tupel . . . . . 66
6.5 Bewertung der Agenten . . . . . 71
6.6 Auswahl des Agenten . . . . . 76
6.7 Konstanz der Ergebnisse . . . . . 78
6.8 Training . . . . . 81
6.9 Übersicht der Ergebnisse . . . . . 83
6.10 Diskussion . . . . . 83

**7 Fazit und Ausblick** **85**

**Literaturverzeichnis** **90**

**Erklärung** **91**

# Abbildungsverzeichnis


2.1 Übersicht der möglichen Aktionen, und der Gewinne in Kuhn Poker . 5
2.2 Spielablauf einer Spielrunde bei Texas Hold'em . . . . . 9

3.1 Poker im General Board Game Framework . . . . . 12
3.2 Kuhn Poker im General Board Game Framework . . . . . 13
3.3 Beispieldarstellung des Baums, der vom Minimax Agenten erzeugt wird 14
3.4 Pseudocode für eine rekursive Implementierung des Minimax Algorithmus . . . . . 15
3.5 Beispielhafte Darstellung eines Expectimax Baums für Backgammon . 16
3.6 Eine Iteration der Monte Carlo Tree Search . . . . . 18
3.7 Beispielhafte Darstellung eines MCTSE Baums . . . . . 20
3.8 Pseudocode für TD($\lambda$) Algorithmus . . . . . 21
3.9 Pseudocode Counterfactual Regret Minimization (with chance sampling) . . . . . 28
3.10 CFR-Strategie zum Zeitpunkt T+1 . . . . . 29
3.11 Vergleich CFR und MCFR . . . . . 30
3.12 Exploitability von KIs für Heads-up limited Hold'em . . . . . 30

4.1 Zunehmende Größe der verarbeiteten Information Sets im Laufe der Zeit . . . . . 35
4.2 Exploitability der Näherungslösung mit zunehmender Rechenzeit für Cepheus . . . . . 36
4.3 Gewinnquote von Deepstack gegen Pokerspieler und Anzahl der gespielten Hände . . . . . 37
4.4 Aufbau von DeepStack . . . . . 38
4.5 Kumulative Gewinne und durchschnittlicher Gewinn in MBB/G von Pluribus im Spiel gegen 5 menschliche Spieler . . . . . 41

5.1 Spiel Random Agent gegen optimale Strategie in Kuhn Poker . . . . . 48
5.2 Spiel optimale Strategie gegen Random Agent in Kuhn Poker . . . . . 48
5.3 Strategiebildung MC-N mit Bube als Startspieler . . . . . 53
5.4 Strategiebildung MCTSE für Kuhnpoker mit Bube als Startspieler . . 55
5.5 Strategiefindung für Expectimax als Startspieler . . . . . 58

6.1 Simulierte Anzahl der Runden pro Spiel . . . . . 63

6.2 Potentiell überbewertete Hand in einem Spiel Texas Hold'em . . . . . 67
6.3 Beispiel Karten in einem Spiel Texas Hold'em . . . . . 69
6.4 Variationskoeffizient im Verlauf bei 10 mio. gespielten Runden . . . . 74
6.5 Vergleich Verteilung der durchschnittlichen Punktzahl bei 10 Mio. und 1 Mio. Spielrunden . . . . . 74
6.6 Vergleich Verteilung der durchschnittlichen Punktzahl bei 4 Mio. und 1 Mio. Spielen . . . . . 75
6.7 Vergleich fünf verschiedener SARSA-4 Agenten über 250.000 Trainingsrunden . . . . . 79
6.8 Vergleich fünf verschiedener Sarsa-4 Agenten über 1 Mio. Trainingsrunden . . . . . 80
6.9 Vergleich fünf verschiedener Sarsa-4 Agenten über 10 Mio. Trainingsrunden . . . . . 81
6.10 Durchschnittliche Abweichung der vier untersuchten Agenten von ihrem jeweils besten Ergebnis . . . . . 82

# Tabellenverzeichnis


2.1 Beschreibung der Aktionen im Spiel Kuhn Poker . . . . . 6
2.2 Optimale Strategie für Kuhn Poker . . . . . 6

3.1 Rewards im Spiel Stein-Schere-Papier . . . . . 25
3.2 Beispielhaftes Regret Matching bzw.Minimization am Beispiel von Stein, Schere, Papier . . . . . 26

5.1 Strategie gegen die optimale Strategie für Kuhn Poker . . . . . 47
5.3 Erster realisierter Gewinn im Spiel Random Agent gegen die optimale Strategie in Kuhn Poker . . . . . 49
5.4 Optimale Strategie für Kuhnpoker als Startspieler mit $\alpha = \frac{1}{5}$ . . . . . 49
5.5 Übersicht der Aktionen des Agenten, der der optimalen Strategie folgt 49
5.6 Fehlerhafte Verteilung der Handkarten . . . . . 50
5.7 Erwartete Werte einer normalverteilten Zufallsvariablen für Buben innerhalb des Streuintervalls . . . . . 50
5.8 Auszug Spielverlauf vom Random Agent gegen optimale Strategie . . 51
5.9 Korrekte Verteilung der Handkarten . . . . . 51
5.10 Vergleich erwarteter und realisierter Gewinn im Spiel Random Agent gegen die optimale Strategie in Kuhn Poker . . . . . 52
5.11 Strategie MC-N für Kuhnpoker . . . . . 54
5.13 Vergleich erwarteter und realisierter Gewinn für MC-N . . . . . 54
5.14 Strategie von MCTSE im Spiel Kuhn Poker . . . . . 56
5.16 Vergleich erwarteter und realisierter Gewinn MC-N . . . . . 56
5.17 Strategie Expectimax für Kuhn Poker . . . . . 57
5.19 Vergleich erwarteter und realisierter Gewinn für Expectimax . . . . . 59
5.20 Strategie TDS für Kuhn Poker . . . . . 60
5.22 Strategie TD-NTuple-3 für Kuhn Poker . . . . . 60
5.24 Strategie TD-NTuple-4 für Kuhn Poker . . . . . 60
5.26 Strategie Q-Learn-4 für Kuhn Poker . . . . . 60
5.28 Strategie Sarsa-4 für Kuhn Poker . . . . . 61
5.30 Durchschnittlicher Gewinn der Agenten im Spiel Kuhn Poker gegen die optimale Strategie . . . . . 61

6.1 Darstellung der Beispielkarten in einer Zusammenfasung . . . . . 69

6.2 Anzahl der möglichen Kombinationen der verschiedenen Pokerhände und deren Häufigkeit . . . . . 72
6.3 Anzahl der notwendigen Spielrunden (in Tausend) je Agent, damit die maximale Abweichung vom Gesamtdurchschnitt einem bestimmten Wert entspricht. . . . . 73
6.4 Standardabweichung und Größe des 99%-Konfidenzintervalls je Menge betrachteter Spiele . . . . . 75
6.5 Durchschnittlicher und maximaler Gewinn aller Agenten je Tupelklasse im Spiel gegen den Benchmark-Agenten . . . . . 77
6.6 Durchschnittlicher und maximaler Gewinn aller Tupelklassen je Agent 77
6.7 Durchschnittlicher Gewinn von Sarsa-4 mit 4 Mio. Trainingsspielen und 4 Mio. Vergleichsspielen . . . . . 78
6.8 Vergleich der besten Ergebnisse der vier Sarsa-4 Agenten mit erzielten Endergebnissen . . . . . 82
6.9 Durchschnittlicher Gewinn der Agenten bei 4 Mio. gespielten Spielrunden gegen den Benchmark-Agenten . . . . . 83

# Poker Begriffe

- **Ante**: Verpflichtender Grundeinsatz, der vor Beginn der Runde von allen Spielern erbracht werden muss.
- **Bet**: Im Rahmen der in der Arbeit genutzten Implementierung: Einen Einsatz machen in Höhe des Bigblinds. Ansonsten ist es auch möglich, Einsätze in unterschiedlicher Höhe zu machen.
- **Big Blind**: Verpflichtender Grundeinsatz in einer bestimmten Höhe für den Spieler nach dem Small Blind.
- **Call**: Einen Einsatz in Höhe der ausstehenden Beträge machen. Beispielsweise denselben Betrag setzen, den ein vorheriger Spieler bei einem Bet gemacht hat.
- **Check**: Passen und keinen Einsatz machen. Diese Aktion ist nur möglich, wenn bisher kein Spieler einen Einsatz gemacht hat.
- **Community Cards**: Die offenen Karten, die von allen Spielern beim Texas Hold'em genutzt werden können.
- **Flop**: Bezeichnet entweder das Aufdecken ersten drei Community Karten oder die Wettrunde, nachdem diese aufgedeckt wurden.
- **Fold**: Aufgeben der aktuellen Runde. Alle bisherigen Einsätze verbleiben im Pot und es wird nicht mehr an der Spielrunde teilgenommen.
- **Hole Cards**: Spielkarten, die nur dem jeweiligen Spieler zugänglich sind.
- **Pot**: Die bisherigen Einsätze aller Spieler, die in der Spielrunde gewonnen werden können.
- **Raise**: Eine Erhöhung der bisherigen Einsätze.
- **River**: Bezeichnet entweder das Aufdecken der letzten Community Karte oder die anschließende finale Wettrunde.
- **Small Blind**: Verpflichtender Grundeinsatz in Höhe eines halben Big Blinds. Der Small Blind muss in der nächsten Runde vom nachfolgenden Spieler erbracht werden.
- **Turn**: Bezeichnet entweder das Aufdecken der vierten Community Karte oder die anschließende Wettrunde.

# 1 Einleitung

Auf dem Gebiet der künstlichen Intelligenz wurden große Fortschritte erzielt, wodurch Computer in die Lage versetzt wurden, gegen menschliche Meister eines Spiels zu gewinnen. Einen der ersten großen und prominentesten Erfolge auf diesem Gebiet stellt der Sieg von *Deep Blue* gegen Garri Kasparov im Schach über sechs Partien unter Tunierbedingungen dar [Newborn, 1997]. *Alpha Go* ist in der Lage, das Spiel Go mit einer Gewinnquote von 99,8% erfolgreich zu spielen [Silver et al., 2016]. Im Jahr 2016 konnte Alpha Go vier von fünf Spielen gegen Lee Sedol, als einen der bekanntesten und erfolgreichsten Go-Spieler, für sich entscheiden. Dies ist besonders erstaunlich, da Alpha Go mit $3^{361}$ möglichen Spielzuständen eines der komplexesten Spiele darstellt. Auch im Bereich des Pokerspiels konnten durch Entwicklungen im Bereich der Künstlichen Intelligenz verschiedene Meilensteine erreicht werden. Der erste dieser Meilensteine war der Erfolg von Deepstack im heads-up no-limit Texas Hold'em, einer Zwei-Spieler-Variante, gegen eine Vielzahl professioneller Pokerspieler im Jahr 2017 [Moravčík et al., 2017]. Kurz darauf konnte mit Libratus eine weitere KI diesen Erfolg wiederholen [Gibney, 2017]. Auch der nächste große Meilenstein, ein Sieg in einem Mehrspieler-Spiel, konnte inzwischen durch Pluribus erreicht werden, als er in einem Sechs-Spieler Texas Hold'em den Sieg gegen professionelle Pokerspieler davongetragen hat [Brown und Sandholm, 2019]. Diese Errungenschaften zeigen, wie erfolgreich künstliche Intelligenzen sein können. Jedoch haben sie gemeinsam, dass es sich hierbei um hochspezialisierte Entwicklungen handelt, welche speziell für eine Herausforderung, zum Beispiel ein bestimmtes Spiel, trainiert wurden. Dies hat den Vorteil, dass die Regeln, der Spielablauf und alle Rahmenbedingungen vorab bekannt sind und in der Entwicklung beachtet werden können, um die KI auf diesen Einsatzfall hin zu optimieren. Der Nachteil bei diesem Vorgehen ist jedoch, dass der entwickelte Agent hierdurch nicht, oder zumindest nicht einfach, auf andere Herausforderungen, zum Beispiel andere Spiele, übertragbar ist. Dies führt dazu, dass die Entwicklung von KI-Agenten ein sehr aufwendiger Prozess sein kann. Im Gegensatz zu diesem Vorgehen gibt es auch andere Modelle, die Spiele und KI-Agenten abstrahieren. Auf diesem Weg ist es zum einen möglich, schnell verschiedene Agenten in einem neuen Spiel zu implementieren und zu analysieren, zum anderen aber auch einen neuen Agenten schnell in vielen Spielen testen zu können. Ein Beispiel hierfür ist General Game Playing (GGP), in dem Spiele in einer Game Description Language modeliert werden [Mańdziuk und Świechowski, 2012]. Ein weiteres Beispiel

ist das General Board Game (GBG) Framework, welches ähnliche Ziele verfolgt wie GPP, sich jedoch in manchen Details unterscheidet. So tauscht GBG die Flexibilität von GPP, die es erlaubt Spiele erst während der Laufzeit zu erlernen, gegen eine höhere Performance, die es erlaubt, Spiele schneller zu erlernen [Konen, 2019, S. 2]. Im Rahmen dieser Arbeit soll untersucht werden, ob GBG, als allgemeines Framework, in der Lage ist, ein komplexes Spiel wie Poker zu erlernen, oder ob es hierfür spezielle KI-Agenten braucht.

## 1.1 Problemstellung

Im Rahmen dieser Arbeit wird untersucht, ob die Agenten, die im General Board Games Framework implementiert sind, in der Lage sind, das Spiel Poker zu erlernen. Dabei werden zwei verschiedene Pokervarianten untersucht. Zum einen die vereinfachte Variante Kuhn Poker, zum anderen heads-up Texas Hold'em, also eine zwei-Spieler-Variante . Kuhn Poker wurde in die Betrachtung aufgenommen, damit in der Analyse der einfachen Variante Erkenntnisse gewonnen werden können, die möglicherweise auf die Gesamtheit des Pokerspiels übertragen werden können. Die Entscheidung, Texas Hold'em im ersten Schritt auf eine zwei-Spieler-Variante zu reduzieren, wurde getroffen, um die Komplexität zu reduzieren und eine Auswertung zu vereinfachen. Zielsetzung ist es, alle Agenten des Frameworks, die für diese Herausforderung geeignet sind zu evaluieren, um festzustellen, in welchem Umfang sie in der Lage waren, das jeweilige Spiel zu erlernen. Die Implementierung des Spiels Texas Hold'em in das Framework wird in dieser Arbeit nicht thematisiert, da diese bereits im Rahmen einer Projektarbeit erarbeitet wurde [Zeh, 2021]. Auch auf eine Beschreibung der Implementierung von Kuhn Poker wird verzichtet, da dies größtenteils eine Abwandlung des entwickelten Texas Hold'em Codes darstellt.

## 1.2 Methodik

Um die eingehende Fragestellung zu beantworten, wurde auf die, durch das General Board Games Framework bereitgestellten, Funktionalitäten zurückgegriffen, und für Aspekte, die nicht durch das Framework bereitgestellt wurden, ein darauf aufbauender Code entwickelt. Zur Ermittlung der Daten, welche für die Evaluierung der Agenten notwendig sind, wurden verschiedene Experimente durchgeführt, um unterschiedliche Aspekte des Vorgehens näher zu untersuchen. Zur detaillierteren Analyse der Daten, wurde neben dem Framework in Java und einfachen Tabellenkalkulationsprogrammen die Programmiersprache R verwendet, um verschiedene Analysen durchzuführen.

# 2 Poker

Bei Poker handelt es sich nicht um *ein* klar definiertes Spiel mit einem einzigen Regelwerk, sondern um eine Sammlung an Spielen, welche auf denselben grundsätzlichen Prinzipien aufbauen. [Zeh, 2021, S. 2–5] Zu Beginn dieser Arbeit steht daher eine Einführung in die Grundzüge des Pokerspiels. Dabei werden im ersten Schritt allgemeine Aspekte des Spiels betrachtet und im Anschluss die beiden in dieser Arbeit vorgestellten Pokervarianten im Detail vorgestellt.

## 2.1 Allgemeines

In diesem Abschnitt werden die allgemeinen Grundlagen des Pokerspiels vorgestellt. Dabei steht zu Beginn eine spieltheoretische Einordnung des Spiels, im Anschluss folgt eine kurze Übersicht der gemeinsamen Aspekte.

### 2.1.1 Einordnung

Bei Poker handelt es sich spieltheoretisch um ein dynamisches, nicht-deterministisches, nicht-kooperatives Mehrspieler-Nullsummenspiel mit unvollständigen Informationen.

*Dynamisch* bedeutet in diesem Kontext ein Spiel, bei dem die Spieler in einer vorgegebenen Reihenfolge agieren und sie somit die Möglichkeit haben, ihre Strategie auf Basis der beobachteten Aktionen der anderen Spieler anzupassen. [Winter, 2019, S. 2] Dies ist bei allen Poker Varianten durch die vorgegebene Zugreihenfolge gegeben.

Bei einem deterministischen Spiel lässt sich mit Sicherheit vorhersagen, wie der nächste Spielzustand aussieht, nachdem eine Aktion ausgeführt wurde. Bei Poker ist dies aufgrund der zufällig verteilten Spielkarten nicht gegeben, weshalb es sich um ein *nicht-deterministisches* Spiel handelt.

Ein Nullsummenspiel beschreibt ein Spiel, bei dem die Gewinne des einen Spielers die Verluste der anderen Spieler darstellen, und somit die Gesamtsumme 0 ist [Holler et al., 2019, S. 61].

Ein Spiel mit vollständigen oder auch perfekten Informationen liegt dann vor, wenn alle Spieler zu dem Zeitpunkt ihrer Entscheidungsfindung alle Informationen über

alle bisherigen Ereignisse besitzen [Osborne und Rubinstein, 1994, S. 89]. Bei Poker ist dies nicht der Fall, da in allen Varianten Karten verteilt werden, die nur dem jeweiligen Spieler bekannt sind.

### 2.1.2 Gemeinsame Aspekte

Auch wenn es zahlreiche verschiedene Pokervarianten gibt, die auf unterschiedliche Weise gespielt werden, teilen sie sich grundlegende Aspekte. Ziel des Spiels, in allen Varianten, ist es, im Rahmen von Wettrunden, die eingesetzten Chips seiner Mitspieler zu gewinnen. Jeder Spieler startet mit einer bestimmten Menge an Chips und das Spiel endet, sobald ein Spieler alle Chips seiner Mitspieler gewinnen konnte. Eine Spielrunde gewinnt dabei derjenige Spieler, der die beste Hand mit den ihm zugänglichen Karten bilden kann. Die Anzahl der ihm zugänglichen und bekannten Karten variiert je nach gespielter Variante, wobei typischerweise, bis auf wenige Ausnahmen, zu der auch das in dieser Arbeit vorgestellte Kuhn Poker zählt, mit einem Deck aus 52 Spielkarten gespielt wird. Ein weiterer gemeinsamer Aspekt ist ein erzwungener Einsatz zu Beginn einer Spielrunde, dem *Ante*, beziehungsweise den *Blinds*. Dieser obligatorische Einsatz soll einen Anreiz für die Spieler darstellen, aktiv an den Wettrunden teilzunehmen und nicht solange abzuwarten, bis sie über ein vielversprechendes Blatt verfügen.

## 2.2 Kuhn Poker

Bei Poker handelt es sich nicht um ein eindeutig definiertes Spiel, sondern es stellt einen Sammelbegriff dar, unter dem viele Spielvariationen mit unterschiedlichen Rahmenbedingungen zusammengefasst werden, die jedoch einer gemeinsamen Struktur folgen [Zeh, 2021, S. 2–5]. Die meisten Spielvariationen haben die Gemeinsamkeit, dass sie durch die Verwendung von Spielkarten, die zufällig verteilt werden, sehr schnell eine sehr große Komplexität erreichen. Diese Komplexität ist nach Kuhn zu groß, um eine vollständige Analyse durchzuführen. Jedoch stellt das Spiel Poker eine interessante Herausforderung dar, weshalb Kuhn eine vereinfachte Form vorgestellt hat [Kuhn, 1951, S. 97]. In dieser vereinfachten Form ist es möglich, alle optimalen Strategien für beide Spieler zu ermitteln, wobei trotz der Einschränkungen wesentliche Aspekte des Pokerspiels erhalten bleiben. Im folgenden wird diese Variante von Poker näher vorgestellt. Dabei werden zuerst die Regeln erläutert. Anschließend wird die optimale Strategie betrachtet, welche im besten Fall durch die Agenten erlernt werden sollte. Abschließend wird ein Vergleich mit den verbreiteteren Poker Varianten angestellt.

### 2.2.1 Regeln

Kuhn Poker wird nicht, wie die meisten anderen Poker-Varianten, mit 52 Karten gespielt, sondern mit einem Deck aus drei Karten mit verschiedenen Werten, typischerweise einem Buben, einer Dame und einem König, wobei die Wertigkeit der Karten dem üblichen Pokerspiel folgt:

$$Bube < Dame < König$$

Das Spiel beginnt mit einem obligatorischen Ante, also einem verpflichtenden Einsatz, den jeder Spieler setzen muss, in Höhe von einem Chip pro Person. Anschließend erhält jeder Spieler zufällig eine der drei Spielkarten, die letzte Karte bleibt beiden Spielern unbekannt. Nun hat der Startspieler zwei mögliche Aktionen: Er kann passen, was das Äquivalent zu einem Check ist, oder einen Einsatz in Höhe von einem Chip machen. Der zweite Spieler hat daraufhin ebenfalls zwei mögliche Aktionen, auch er kann entweder passen, oder einen Einsatz machen. Sollten entweder beide Spieler in Folge gepasst haben oder beide Spieler in Folge einen Einsatz gemacht haben, kommt es zum Showdown und der Spieler mit der besseren Handkarte gewinnt die Spielrunde und die gemachten Spieleinsätze. Sollte ein Spieler nach einem vorhergehenden Einsatz passen, gewinnt der Spieler, der den Einsatz gemacht hat ohne Showdown, was dem Fold in anderen Pokervarianten entspricht.

KUHN

| | First Round Player I | First Round Player II | Second Round Player I | Payoff |
|---|---|---|---|---|
| (1) | pass | pass | | 1 to holder of higher card |
| (2) | | bet | pass | 1 to player II |
| (3) | | | bet | 2 to holder of higher card |
| (4) | bet | pass | | 1 to player I |
| (5) | | bet | | 2 to holder of higher card |

Abbildung 2.1 Übersicht der möglichen Aktionen, und der Gewinne in Kuhn Poker [Kuhn, 1951]

Sollten am Ende einer Spielrunde beide Spieler noch Chips haben, kann eine neue Spielrunde begonnen werden. Das Spiel endet, sobald ein Spieler keine Chips mehr hat und damit nicht mehr am Spiel teilnehmen kann.

Tabelle 2.1 Beschreibung der Aktionen im Spiel Kuhn Poker

| Aktion | Beschreibung |
|---|---|
| PASS | Es wird kein Einsatz gemacht. Für den beginnenden Spieler im ersten Zug entspricht dies dem CHECK in den üblichen Pokervarianten. Ansonsten entspricht es dem FOLD, da das Spiel aufgegeben wird und der Gegner die Spielrunde gewinnt. |
| BET | Es wird ein Einsatz gemacht. Der Gegner hat daraufhin die Wahl, ob er ebenfalls mit BET einen Einsatz macht, was in den üblichen Spielen einem CALL entspricht, oder mit einem PASS die Runde aufgibt. |

### 2.2.2 Optimale Strategie

Für das Spiel Kuhn Poker gibt es insgesamt 30 verschiedene Spielverläufe, die sich aus den sechs möglichen Verteilungen der Starthände und den fünf möglichen Aktionsverläufen in Abbildung 2.1 ergeben. Auf Basis der verschiedenen Spielverläufe und den jeweiligen Gewinnen hat Kuhn die optimalen Strategien sowohl für Spieler 1, als auch Spieler 2 aufzeigen können, die in Tabelle 2.2 abgebildet sind.

Tabelle 2.2 Optimale Strategie für Kuhn Poker

(a) Als Startspieler

| **Hand** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | $\alpha$ BET<br>$(1-\alpha)$ CHECK | FOLD |
| Dame | CHECK | $\alpha + \frac{1}{3}$ CALL<br>$(1-(\alpha+\frac{1}{3}))$ FOLD |
| König | $3*\alpha$ BET<br>$(1-3*\alpha)$ CHECK | CALL |

(b) Als Folgespieler

| | Aktion Gegner | |
|---|---|---|
| **Hand** | **CHECK** | **BET** |
| Bube | $\frac{1}{3}$ BET<br>$\frac{2}{3}$ FOLD | FOLD |
| Dame | CHECK | $\frac{1}{3}$ CALL<br>$\frac{2}{3}$ FOLD |
| König | BET | CALL |

Sollte also der Startspieler der optimalen Strategie folgen und als Handkarte eine Dame besitzen, so wird er in jedem Fall die Aktion CHECK wählen. Angenommen sein Gegner folgt ebenfalls der optimalen Strategie und hat als Handkarte einen Buben, so wird er, da die Aktion seines Gegners ein CHECK war, mit einer Wahrscheinlichkeit von $\frac{1}{3}$ mit BET einen Einsatz machen und mit einer Wahrscheinlichkeit von $\frac{2}{3}$ mit FOLD die Spielrunde aufgeben. Für den Fall, dass der Gegner einen Einsatz macht, ist nun der Startspieler an der Reihe, auf diese Aktion zu reagieren. Mit einer Dame wird er mit der Wahrscheinlichkeit $\alpha + \frac{1}{3}$ mit CALL mit dem Einsatz mitziehen und mit einer Wahrscheinlichkeit von $(1-(\alpha+\frac{1}{3}))$ mit FOLD die Spielrunde aufgeben.

Bemerkenswert ist, dass sich für Spieler 2 genau eine Strategie ergibt, während für Spieler 1 unendlich viele optimale Strategien existieren, die von $\alpha$ abhängig sind,

wobei $\alpha \in [0; \frac{1}{3}]$. Wenn diese beiden Strategien in einem Spiel aufeinandertreffen, beträgt der erwartete Gewinn für Spieler 1 auf lange Sicht $-\frac{1}{18}$. Da es sich bei Kuhn Poker um ein Nullsummenspiel handelt, folgt daraus, dass der erwartete Gewinn für den 2. Spieler $+\frac{1}{18}$ beträgt.

*Optimale Strategie* bedeutet in diesem Zusammenhang nicht, dass diese Strategie gegen jeden Gegner den maximalen Gewinn erzielt, sondern dass ein Nash-Gleichgewicht vorliegt. Demnach würden beide Spieler nicht von der Strategie abweichen, da sie sonst ihren Gewinn verringern würden. Hierbei ist anzumerken, dass es sich bei den gefundenen Strategien um *gemischte* Strategien handelt, bei denen im Gegensatz zu den *puren* Strategien nicht immer in einer bestimmten Spielsituation eine bestimmte Aktion gewählt wird, sondern eine zufällige Entscheidung Teil der Auswahl der Aktionen ist [Holler et al., 2019].

Zu bedenken ist, dass das Ziel von Poker ist, nicht nur den eigenen Verlust zu minimieren, sondern den möglichen Gewinn zu maximieren. In der Praxis wird nicht jeder Gegner der optimalen Strategie folgen, sondern er könnte Fehlentscheidungen treffen, welche durch den anderen Spieler ausgenutzt werden könnten. Dies kann dazu führen, dass es Strategien gibt, die diese Fehlentscheidungen ausnutzen, um den eigenen Gewinn zu maximieren. So konnten Hoehn et al. zeigen, dass es vorteilhaft sein kann, von der pessimistischen Nash-Gleichgewicht Strategie abzuweichen, um zu lernen, die Schwächen in der Strategie des Gegners auszunutzen und die eigene Strategie dementsprechend anzupassen. [Hoehn et al., 2005]

### 2.2.3 Vergleich Poker

Durch die starke Vereinfachung wird ein großer Teil der Komplexität reduziert. Dennoch bleiben viele grundlegende Bestandteile des Pokerspiels erhalten. Der festgesetzte *Ante* ist mit den *Blinds* im verbreiteten Texas Hold'em vergleichbar. Beide Spieler in Kuhn Poker verfügen über eine Handkarte, die dem Gegner nicht bekannt ist, und es existiert eine weitere Karte als Deck, welche beiden Spielern unbekannt ist. Somit wurde zwar die Menge an Möglichkeiten reduziert, es bleibt jedoch das Kernproblem der unvollständigen Informationen erhalten, wodurch Spielzüge wie Bluffs ermöglicht werden. Auch verfügt der Spieler über eine bestimmte Menge an Startchips und er kann über die Spielrunden hinweg Chips gewinnen und verlieren, genau wie in anderen Varianten. Trotz der vielen Gemeinsamkeiten gibt es verschiedene Aspekte des Pokerspiels, die nicht in der Kuhn-Poker-Variante enthalten sind, die jedoch in vielen der anderen Varianten eine große Rolle spielen. Es gibt keine Community Karten, die beiden Spielern bekannt sind, die Wetteinsätze, die gemacht werden können, sind nicht flexibel und es gibt keine Möglichkeit den Pot, in Form eines Splitpots, aufzuteilen [Risk und Szafron, 2010, S. 163]. Ein weiterer relevanter

Unterschied ist, dass das ursprüngliche Kuhn Poker für zwei Spieler konzipiert ist, während andere Pokervarianten typischerweise auf mehrere Spieler ausgelegt sind. Jedoch gibt es neben dem einfachen Kuhn Poker eine weitere Variante mit drei Spielern, für die ebenfalls ausführliche Analysen gemacht wurden und verschiedene Strategien gefunden werden konnten [Billingham, 2017]. Da diese Variante jedoch deutlich umfangreicher und damit aufwendiger zu analysieren ist, habe ich mich im Rahmen dieser Ausarbeitung für die Zwei-Spieler-Variante entschieden.

## 2.3 Texas Hold'em

Bei Texas-Hold'em handelt es sich mittlerweile um eine der weitverbreitetsten Poker Varianten, wobei sie besonders von dem großen Poker Aufschwung profitiert hat [Zeh, 2021, S. 4]. Dabei handelt es sich um eine Hold'em Variante des Pokerspiels, bei dem ein Spieler sowohl eigene, nur ihm bekannte Handkarten (*Hole Cards*) besitzt, darüber hinaus aber auch auf mit anderen Spielern geteilte *Community Cards* zugreifen kann.

### 2.3.1 Spielablauf

Das Spiel besteht aus mehreren Spielrunden. Es werden solange weitere Spielrunden gespielt, bis ein Spieler alle Chips seiner Gegner gewinnen konnte. Eine Spielrunde besteht aus mehreren Wettrunden, wobei die Spieler zwischen den Wettrunden neue Informationen, in Form von weiteren Spielkarten, erhalten.

Die Spielrunde beginnt mit den obligatorischen Einsätzen von zwei Spielern, dem *Small Blind* und dem *Big Blind*. Die Höhe der Blinds ist variabel, wobei der Small Blind der Hälfte des Big Blinds entspricht, und dieser an den vorgegebenen Mindesteinsatz gekoppelt ist. Die Blinds werden dabei abwechselnd von den verschiedenen Spielern gezahlt. Nach Zahlung der Blinds erhalten die Spieler ihre Handkarten und die erste Wettrunde beginnt. Eine Wettrunde ist dann beendet, wenn alle Spieler an der Reihe waren und die gleichen Einsätze getätigt wurden. Die Blinds gelten dabei in der ersten Wettrunde als Einsatz, weshalb alle Spieler mindestens einen Einsatz in Höhe des Big Blinds tätigen müssen, um am Spiel teilzunehmen.

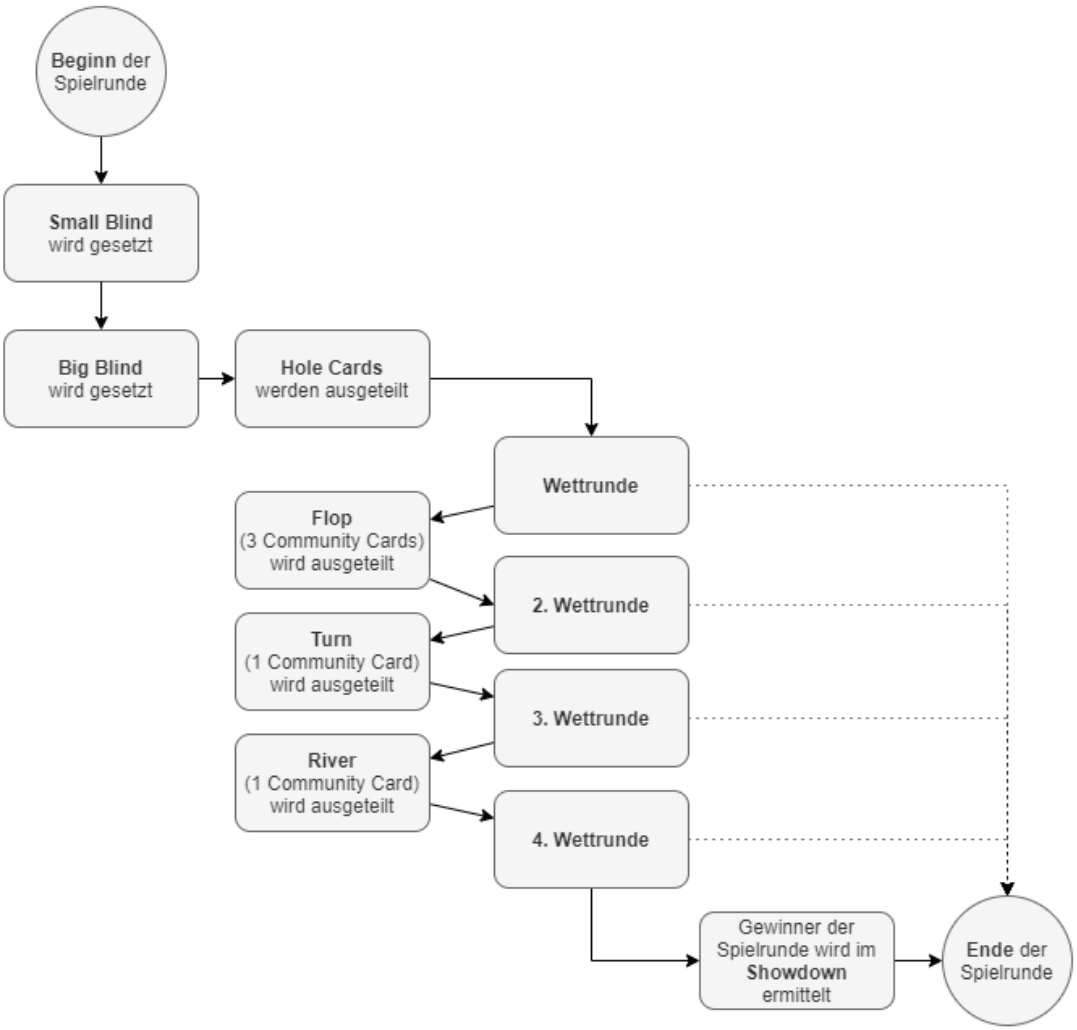


Abbildung 2.2 Spielablauf einer Spielrunde bei Texas Hold'em

Nachdem die erste Wettrunde abgeschlossen wurde, werden drei weitere Karten, die als *Flop* bezeichnet werden, aufgedeckt, die allen Spielern zur Verfügung stehen. Durch diese neuen Informationen ist es den Spielern möglich, die Stärke und die Wahrscheinlichkeit ihrer potentiellen Hand besser abzuschätzen. Anschließend folgen weitere Wettrunden, unterbrochen von zwei einzeln aufgedeckten neuen Community Karten, dem *Turn* und dem *River*. Nach Abschluss der 4. Wettrunde kommt es zum Showdown, in dem unter allen, noch an der Runde teilnehmenden Spielern, der Gewinner der Spielrunde bestimmt wird. Der Gewinner ist der Spieler mit der stärksten Hand. Er gewinnt die Spieleinsätze der anderen Spieler. Sollten mehrere Spieler gleichstarke Hände haben, oder kann ein Spieler nicht den kompletten erforderlichen Einsatz erbringen, ist es möglich, dass der angesammelte Pot anteilig auf die Spieler verteilt wird. [Zeh, 2021, S. 9].

# 3 Grundlagen

In diesem Kapitel werden die Grundlagen vermittelt, welche die Basis für diese Arbeit darstellen. Dabei wird zuerst eine Einleitung in das Forschungsfeld *Reinforcement Learning* gegeben. Im Anschluss wird das General Board Games Framework, auf dem diese Arbeit aufbaut, thematisiert. Dabei wird zum einen die Zielsetzung und der Aufbau des Frameworks vorgestellt, zum anderen aber auch die verschiedenen implementierten Agenten. Abschließend werden verschiedene KI-Agenten thematisiert, die bereits in der Lage waren, erhebliche Erfolge im Pokerspiel zu erringen.

## 3.1 Reinforcement Learning

Das Fachgebiet *Machine Learning* ist ein Teilgebiet des Themas künstliche Intelligenz und befasst sich mit dem Trainieren von Maschinen zur Lösung von Problemen. Machine Learning teilt sich in verschiedene Bereiche [Sutton und Barto, 1998, S. 2]:

- *Supervised Learning*: Im überwachten Lernen werden einem Algorithmus im ersten Schritt Trainingsdaten zur Verfügung gestellt. Ziel ist es, auf Basis der Trainingsdaten, Muster innerhalb der Daten zu erkennen und auf deren Basis Erkenntnisse zu gewinnen, die sich auf allgemeine Daten übertragen lassen. Ein Beispiel hierfür ist die Nutzung von Algorithmen zur Erkennung von Kreditkartenbetrug. Hierbei wird dem Algorithmus ein Trainingsdatenset aus nachweislichen Betrugsfällen in Kombination mit normalen Vorgängen vorgelegt. Der Algorithmus kann auf dieser Basis lernen, worin sich Betrugsfälle von typischem Verhalten unterscheiden, und lernt auf diesem Weg auffälliges Verhalten zu identifizieren.

- *Unsupervised Learning*: Ziel im nicht-überwachten Lernen ist es, Strukturen innerhalb gegebener Daten zu finden, ohne dass weitere Eingaben, zum Beispiel in Form von Labeln innerhalb der Daten, notwendig wären. Hierbei werden die Daten lediglich auf Basis gemeinsamer Eigenschaften gruppiert. Es wird jedoch keine Bewertung der identifizierten Gruppen durchgeführt. Ein Beispiel hierfür ist das Bilden von Clustern innerhalb von Daten zur Gruppierung in Form einer Kundenanalyse. Ein weiteres Beispiel ist die Entdeckung von Anomalien als möglicher Anwendungsfäll.

- *Reinforecement Learning*: Im bestärkenden Lernen interagiert ein lernender Agent mit seiner Umwelt, um ein Ziel zu erreichen. Hierbei wird dem Agenten nicht vorgegeben, was die beste Aktion ist, sondern es werden ihm Belohnungen gegeben, die sowohl positiv als auch negativ sein können. Durch diese Belohnungen wird gutes Verhalten verstärkt und schlechtes Verhalten bestraft. Ziel des Agenten ist es, die Belohnungen zu maximieren, wodurch er gleichzeitig eine Strategie erlernt, das gegebene Problem zu lösen.

Die in dieser Arbeit vorgestellten Methoden fallen in den Bereich des Reinforcement Learnings.

## 3.2 General Board Games

Es gibt eine große Anzahl Algorithmen, die in der Lage sind, verschiedenste Spiele zu erlernen. Jedoch ist der Prozess der Implementierung für jeden Algorithmus individuell, da unterschiedliche Informationen in unterschiedlicher Weise bereitgestellt werden müssen, was dazu führt, dass die Anwendung verschiedener Algorithmen in einem Spiel ein sehr zeitaufwendiger Prozess sein kann. Dies war, unter anderem, Motivation zur Entwicklung des *General Board Game* (GBG) Frameworks [Konen, 2019, S. 1].

In diesem Abschnitt werden zunächst das Framework und die Umsetzung des Pokerspiels im Rahmen des Frameworks vorgestellt. Abschließend werden die verschiedenen Agenten und ihre Funktionsweise erläutert.

### 3.2.1 GBG-Framework

Ziel des Frameworks ist es, sowohl das schnelle und einfache Testen verschiedener KI-Agenten in einem neu implementierten Spiel zu ermöglichen, als auch das Testen eines neuen KI-Agenten mit einer Vielzahl von Spielen, ohne dass viel Implementierungsaufwand notwendig ist. [Konen, 2020, S. 4]

Der Fokus liegt dabei auf *Board Games*, also Brettspielen, wie sie üblicherweise am Tisch gespielt werden. Hierbei führen eine bestimmte Menge an Spielern rundenweise Aktionen, in Form von Spielzügen, durch. Es können sowohl deterministische Spiele (z.B. Tic-Tac-Toe), als auch nicht-deterministische Spiele (z.B. 2048) abgebildet werden, sowie Spiele mit vollständigen Informationen (z.B. Othello) und Spiele mit unvollständigen Informationen (z.B. Poker).

Um das Ziel der einfachen Implementierung zu erreichen, stellt das Framework verschiedene Klassen und Schnittstellen bereit, welche die Implementierung erleichtern. Um ein neues Spiel im Framework zu etablieren, ist es lediglich notwendig eine

handvoll Klassen zu implementieren, welche die Spiellogik beschreiben und die Informationen den Agenten zur Verfügung stellen. Anschließend ist es möglich, alle allgemeinen Agenten des Frameworks im neu implementierten Spiel zu testen.

### 3.2.2 Umsetzung von Poker im GBG-Framework

Das Spiel Poker wurde im Rahmen einer Projektarbeit in das General Board Game Framework implementiert. [Zeh, 2021]. Hierbei wurden die verschiedenen notwendigen Klassen implementiert und ein Spiel gegen die Agenten wurde ermöglicht.

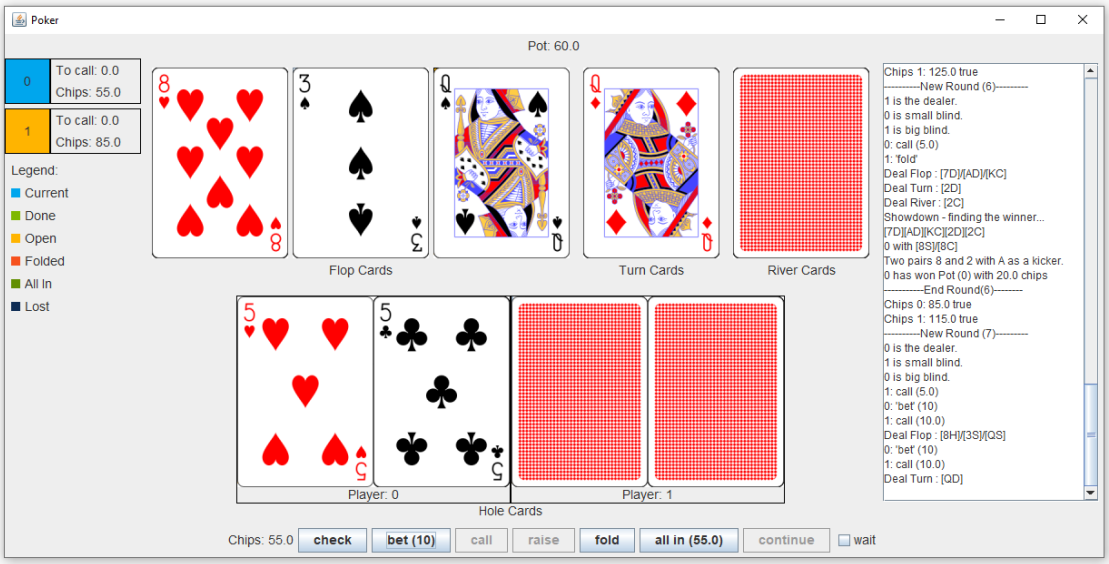


Abbildung 3.1 Poker im General Board Game Framework

Um allerdings eine Bewertung der Agenten durch einen Benchmark-Agenten zu ermöglichen, war es notwendig, einen Agenten zu implementieren. Dieser Agent implementiert die Schnittstelle *PlayAgent* und leitet sich von der abstrakten Klasse *AgentBase* ab. [Konen, 2020, S. 4] Jedoch hat er nicht die typischen Merkmale, welche die anderen Agenten haben, da er lediglich für das Pokerspiel konzipiert ist.

Im Rahmen der Arbeit wurde neben dem bereits implementierten Poker Spiel auch das Spiel Kuhn Poker untersucht, welches nicht in der Projektarbeit implementiert wurde. Eine vollständige Beschreibung der Implementierung würde den Rahmen dieser Arbeit sprengen. Aufgrund der vielen Ähnlichkeiten zwischen dem bereits implementierten Poker und Kuhn Poker, konnten viele Aspekte wiederverwendet werden, wobei es in den meisten Fällen die Komplexität reduziert hat. Beispiele hierfür sind die Ermittlung des Gewinners einer Spielrunde oder die Ausschüttung der Gewinne. Um den Gewinner einer Spielrunde zu ermitteln, müssen nur die Werte der Handkarten verglichen werden, nicht wie im normalen Pokerspiel erst der Wert

der Hand ermittelt werden, welcher im Anschluss verglichen werden kann. Auch steht der Gewinner eindeutig fest und es gibt nicht die Möglichkeit, dass der Pot aufgeteilt werden könnte. Eine weitere Vereinfachung ergibt sich dadurch, dass das implementierte Pokerspiel für eine beliebig Anzahl Spieler konzipiert war, während Kuhn Poker prinzipiell nur auf zwei Spieler ausgelegt ist.

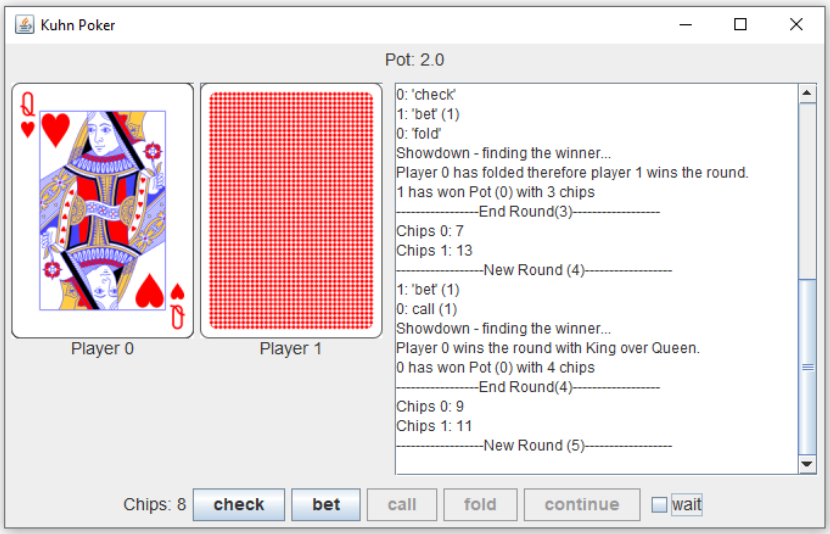


Abbildung 3.2 Kuhn Poker im General Board Game Framework

### 3.2.3 Agenten

In diesem Abschnitt werden die einzelnen Agenten, welche im Rahmen der praktischen Umsetzung dieser Arbeit genutzt werden, im Detail vorgestellt und notwendiges Hintergrundwissen vermittelt. Hierbei ist zu beachten, dass nicht jeder Agent, der im Framework nutzbar ist, auch für die gegebene Herausforderung geeignet ist. Aus diesem Grund wird hier der Max-N-Agent nicht betrachtet, da dieser ein deterministisches Spiel voraussetzt. Der Human-Agent ermöglicht das Spiel als Mensch gegen einen der KI-Agenten, was durchaus möglich und vollständig implementiert ist, jedoch eine sehr große Anzahl an Spielen voraussetzt, um verlässliche Aussagen über die Qualität eines Agenten zu treffen, weshalb dieses Vorgehen im Rahmen der Arbeit nicht verfolgt wurde.

#### Random

Der Random-Agent wählt seine "beste"nächste Aktion rein zufällig aus allen, in einem gegebenen Spielzustand möglichen, Aktionen aus. Dies hat den Vorteil, dass der Agent weder trainiert werden muss, noch viel Zeit für die Auswahl der besten nächsten Aktion benötigt wird. Dieses Charakteristikum trägt dazu bei, dass dieser Agent gut für erste Versuche bei einem neu implementierten Spiel geeignet ist, da er schnell einsatzfähig ist und das Spiel getestet werden kann. Ein weiterer denkbarer

Anwendungsfall des Agenten ist als Vergleichspunkt, an dem die Ergebnisse anderer Agenten gemessen werden können. Die erzielten Ergebnisse des Agenten selbst sind verständlicherweise nicht gut, da diese rein zufällig gewählt wurden.

### Expectimax

Der Max-N-Agent kann nicht im Pokerspiel angewandt werden, da dieser lediglich für deterministische Spiele geeignet ist. Der Expectimax-Agent stellt eine Erweiterung dieses Agenten dar, und ermöglicht Spiele mit Zufallselementen zu analysieren. Da Expectimax jedoch auf Max-N bzw. Minimax aufbaut, wird dieser zunächst beschrieben und im Anschluss die Erweiterung aufgezeigt. Ziel des Minimax-Algorithmus und Max-N als einer Erweiterung für $N$ Spieler ist es, die optimale Strategie für einen gegebenen Spielzustand zu ermitteln, wobei diese durch den *Minimax-Wert* eines jeweiligen Knotens, ermittelt werden kann, vorausgesetzt beide Spieler folgen der optimalen Strategie [Russell und Norvig, 2010, S. 164]. Der Minimax-Algorithmus versucht dabei, den maximalen Reward seines Gegners zu minimieren. Hierbei werden die Spielzustände als Baum dargestellt, wobei die Wurzel den aktuellen Spielzustand darstellt. Die Kindknoten des jeweiligen Spielzustands spiegeln den folgenden Spielzustand wieder, nachdem eine bestimmte Aktion ausgeführt wurde. Somit entspricht die Anzahl der Kindknoten der Anzahl der möglichen Aktionen. Bei den Knoten wechseln sich MAX-, und MIN-Knoten ab, wobei MAX-Knoten den maximalen Wert ihrer Kindknoten annehmen, MIN-Knoten den minimalen Wert ihrer Kindknoten. Sollte ein Knoten keine Kindknoten besitzen, handelt es sich um einen Endzustand, wodurch der Wert des Knotens durch den Spielzustand bestimmt wird.

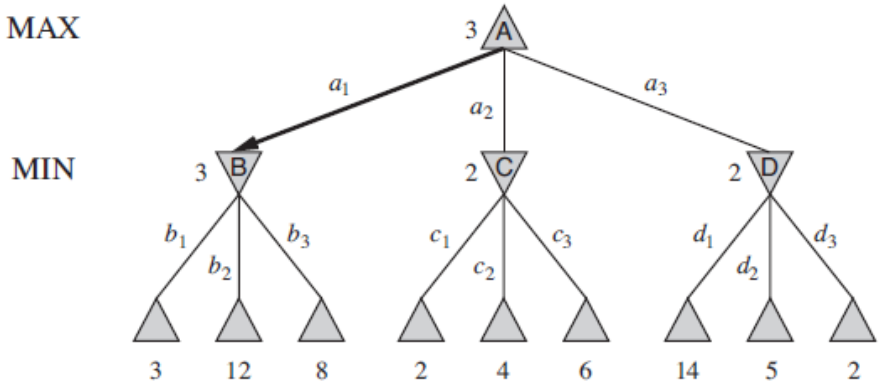


Abbildung 3.3 Beispieldarstellung des Baums, der vom Minimax Agenten erzeugt wird [Russell und Norvig, 2010, S. 164]

Eine beispielhafte rekursive Implementierung wie in Abbildung 3.4 zeigt, dass der Baum zuerst bis zu einem der Blattknoten durchsucht wird und somit eine Depth-First-Search gegeben ist. Der Aufwand ist dabei zum einen von der maximalen Tiefe des Baums $m$, zum anderen von der Anzahl der möglichen Aktionen $b$ abhängig.

Die Gesamtkomplexität des einfachen Minimax Algorithmus entspricht somit $O(b^m)$ [Russell und Norvig, 2010].

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Abbildung 3.4 Pseudocode für eine rekursive Implementierung des Minimax Algorithmus [Russell und Norvig, 2010, S. 166]

Es gibt verschiedene Wege diese Komplexität zu reduzieren, beispielsweise mit einer variablen oder dynamischen Suchtiefe, oder aber auch der Alpha-Beta-Suche, bei der nicht der komplette Baum durchsucht wird, sondern Zweige, die nicht erfolgversprechend sind, ausgelassen werden [Pollice et al., 2008, S. 183].

Eine Eigenschaft des Algorithmus ist, dass von einem deterministischen Übergang zwischen den verschiedenen Spielzuständen ausgegangen wird. In Spielen mit Zufallselementen, wie Poker, ist dies jedoch nicht der Fall, da der Spieler nicht weiß, über welche Karte der Gegner verfügt oder welche Karte als nächstes aufgedeckt wird. Um diesen Aspekt abzubilden, wird der Baum aus MAX- und MIN-Knoten um CHANCE-Knoten ergänzt, wie in Abbildung 3.5 gezeigt wird. Diese Knoten bündeln alle möglichen Kindknoten, die aus einer Aktion folgen und stellen den *erwarteten* Wert, also den nach Wahrscheinlichkeit gewichteten Durchschnittswert aller Kindknoten dar [Russell und Norvig, 2010, S. 178].

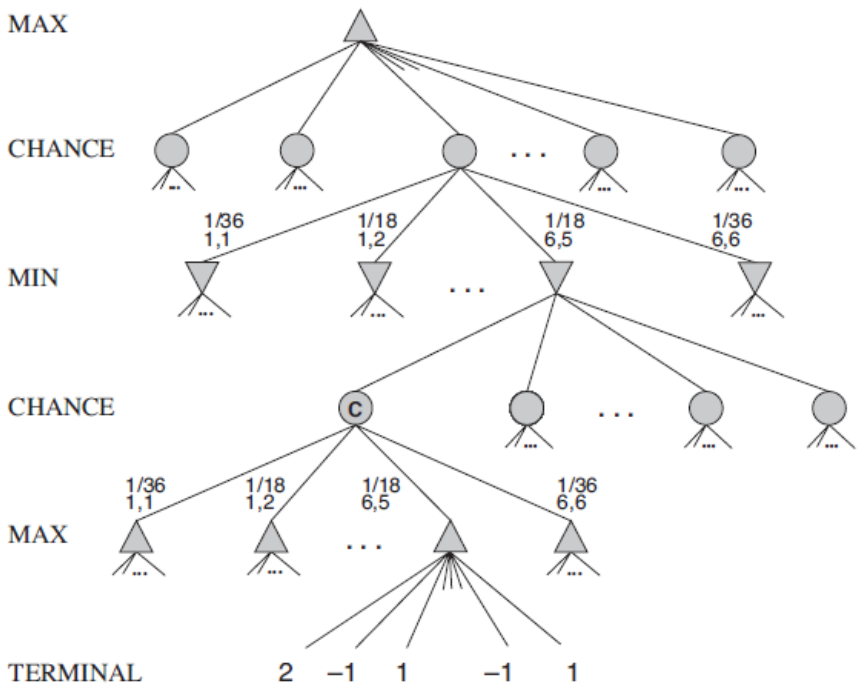


Abbildung 3.5 Beispielhafte Darstellung eines Expectimax Baums für Backgammon, bei dem die Zufallsaktion durch zwei Würfel gegeben ist [Russell und Norvig, 2010, S. 178]

Trotz der möglichen Verbesserungsmöglichkeiten wie Alpha-Beta-Suche, die auch für Expectimax genutzt werden kann, ist der Algorithmus aufgrund der Komplexität von Texas Hold'em hierfür nicht geeignet, da durch die vielen Zufallselemente sehr schnell ein sehr großer Baum entstehen würde. Kuhn Poker hingegen hat einen sehr kleinen Spielbaum und stellt daher einen möglichen Anwendungsfall dar.

### MC-N

In der Stochastik versteht man unter einer Monte-Carlo-Simulation ein Verfahren, das mit der Durchführung einer großen Anzahl von Zufallsexperimenten Problemstellungen auf statistischem Weg löst. Dies ist besonders dann ein attraktiver Weg, wenn die Aufgabe analytisch nicht oder nur schwer lösbar ist. Der Mont-Carlo-Agent (MC-Agent) verfolgt einen sehr ähnlichen Ansatz bei der Lösung von Reinforment Learning Problemen, wobei die erwarteten Ergebnisse dem Durchschnittswert der Ergebnisse einer Vielzahl zufälliger Spiele entspricht [Sutton und Barto, 1998, S. 113]. Das Vorgehen des Agenten kann dabei in zwei Phasen unterteilt werden. Auf der einen Seite die *Simulation*, in der eine Vielzahl Spielverläufe simuliert wird, und der *Evaluation*, in der die zusammengefassten Ergebnisse ausgewertet werden und die beste Aktion bestimmt wird. In der Simulation wird zuerst eine der, in der aktuellen Spielsituation verfügbaren Aktionen gewählt, und das Spiel in den neuen Spielzustand überführt. Anschließend wird das Spiel durch zufällige Aktionen solange gespielt, bis entweder die maximale Spieltiefe (Rolloutdepth) erreicht wurde, oder die aktuelle Episode beendet wurde. Der Agent simuliert so

lange zufällige Spielverläufe, bis eine maximale Anzahl an Iterationen erreicht wurde oder ein vorgegebenes Zeitkontingent aufgebraucht wurde. Anschließend werden die durchschnittlichen Ergebnisse je gewählter Aktion ermittelt. Jene Aktion mit dem höchsten durchschnittlichen Ergebnis wird als beste Aktion definiert. Dieses Vorgehen hat den Vorteil, dass der Agent keinerlei Vorkenntnisse über das zu erlernende Spiel, oder die zugrundeliegenden Mechaniken benötigt, da für ihn nur das aus den Zufallsspielen resultierende Ergebnis von Relevanz ist. Somit benötigt der Agent keine Zeit, um trainiert zu werden, da er seine Strategie erst während des eigentlichen Spiels formt. Dennoch ist der Agent grundsätzlich in der Lage, eine optimale Strategie zu finden. [Sutton und Barto, 1998, S. 113]. Der Nachteil hierbei ist jedoch, dass mit zunehmend komplexeren Spielen ebenfalls die Anzahl der benötigten Iterationen wächst und der Agent in jedem Zug neue Simulationen durchführen muss, um eine Entscheidung zu treffen, und nicht auf seine vorherigen Erkenntnisse zurückgreifen kann. Dies führt dazu, dass bei einer großen Anzahl von Spielen der Agent viel Zeit und Ressourcen benötigt.

### MCTSE

Die Monte-Carlo-Tree-Search-Methode (MCTS) vereint die Allgemeinheit von auf Zufallsexperimenten basierten Verfahren, wie der Monte Carlo Simulation, mit der Präzision von auf Suchbäumen basierten Verfahren [Browne et al., 2012]. Auf diese Weise können verschiedene Vorteile beider Ansätze kombiniert werden. Der Algorithmus benötigt keinerlei Vorkenntnisse über die Spielgegebenheiten und kann mit wenig Aufwand auf die meisten Problemstellungen angewandt werden. Die erzielten Ergebnisse stehen dabei in direktem Zusammenhang mit der verfügbaren Rechenleistung, da die Qualität der Ergebnisse mit der Anzahl der betrachteten Simulationen wächst. Hierbei ist jedoch anzumerken, dass der Agent sich vorrangig durch den hinterlegten Suchbaum auf die vielversprechendsten Aktionen konzentriert, um auf diesem Wege schneller eine gute Lösung zu erzielen. MCTS basiert auf zwei grundlegenden Konzepten:

> *...that the true value of an action may be approximated using random simulation; and that these values may be used efficiently to adjust the policy towards a best-first strategy.* [Browne et al., 2012, S. 5]

In der Praxis setzt der Agent diese Konzepte um, indem er Schritt für Schritt einen Spielbaum aufbaut, wobei die einzelnen Knoten die bisherigen Ergebnisse der Simulationen widerspiegeln. Dieser Baum wächst mit jeder Iteration und wird mit zunehmender Zeit immer genauer. Der Agent optimiert seine Ergebnisse, bis eine vorgegebene Grenze erreicht ist. Diese Grenze kann zum Beispiel eine maximale Anzahl an Iterationen oder eine maximale Zeitdauer sein. Dies bedeutet, dass der

Agent zu jeder Zeit in der Lage ist, eine zu dem Zeitpunkt voraussichtlich beste Aktion zu wählen.

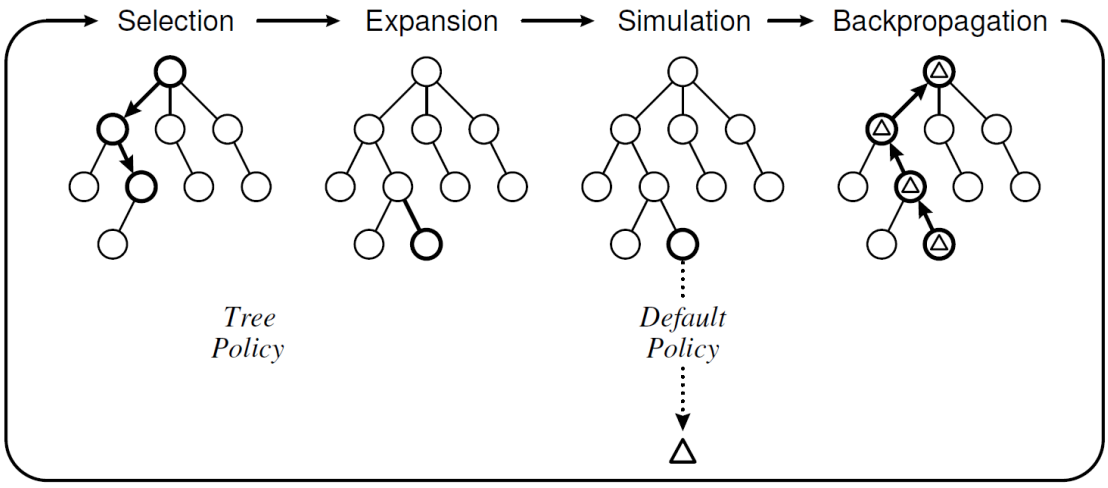


Abbildung 3.6 Eine Iteration der Monte Carlo Tree Search [Browne et al., 2012, S. 6]

Für die Optimierung der Ergebnisse durchläuft der Agent in jeder Iteration die vier in Abbildung 3.6 abgebildeten Phasen [Browne et al., 2012, S. 6]:

- *Auswahl*: Aus dem Baum wird auf Basis einer gegebenen *Tree Policy* der zum aktuellen Zeitpunkt relevanteste Knoten bestimmt, der noch über mindestens einen nicht betrachteten Kindknoten verfügt. Die Kindknoten entsprechen dabei Aktionen, die zu einem Folgezustand führen.

- *Expansion*: Der Baum wird um die in der Auswahl bestimmten Kindknoten erweitert.

- *Simulation*: Ausgehend vom ausgewählten Kindknoten wird der weitere Spielverlauf simuliert. Das in der Simulation genutzte Vorgehen wird durch die *Default Policy* definiert.

- *Backpropagation*: Im letzten Schritt werden die, in der Simulation erzielten Ergebnisse, im Kindknoten gespeichert und anschließend bis zur Wurzel des Baums propagiert.

Die während der Iteration gewonnenen Erkenntnisse, die in der Backpropogation Phase an die übergeordneten Knoten zurückgegeben werden, werden in jedem Knoten gespeichert. Dabei werden sowohl die Anzahl der Besuche des jeweiligen Knotens, als auch der gesamte erzielte Reward erfasst, wodurch es möglich wird, den durchschnittlichen Reward je Besuch zu ermitteln.

Im Rahmen der Auswahl des relevantesten Kindknoten durch die Tree Policy ist es wichtig, eine Balance zwischen dem Testen neuer Aktionen, also dem Erkunden neuer Zweige des Baums (*Exploration*) und der Auswahl der vielversprechenden

Aktionen, die gute Ergebnisse versprechen (*Exploitation*), zu finden. Auf der einen Seite besteht die Möglichkeit in unbekannten Bereichen des Baums eine noch bessere Strategie zu finden, auf der anderen Seite könnten mit einem Fokus auf die schon starken Aktionen, diese vielleicht weiter verbessert werden [Chaslot, 2010, S. 19f.].

Nachdem der Algorithmus seine vorgegebenen Ressourcen in Form von Zeit, Iterationen oder Ähnlichem aufgebraucht hat, muss der Algorithmus auf Basis des erstellten Spielbaums die beste mögliche Aktion auswählen. Für diese Auswahl gibt es unterschiedliche Herangehensweisen [Schadd, 2009, S. 33–34]:

- Max Child: Auswahl des Knotens mit dem besten Ergebnis.
- Robust Child: Auswahl des am häufigsten besuchten Knotens.
- Max-Robust Child: Auswahl des Knotens mit dem besten Ergebnis und den häufigsten Besuchen. Sollte kein Knoten diese Eigenschaft haben, folgen weitere Simulationen.
- Secure Child: Auswahl des Knotens, der die untere Konfidenzgrenze maximiert.

Experimente zeigten jedoch, dass die Auswahl einer bestimmten Methodik vorrangig bei einer geringen Anzahl an Iterationen einen nennenswerten Einfluss hat, da hier die Max Child Methode das Risiko mit sich bringt, suboptimale Aktionen auszuwählen [Chaslot, 2010, S. 25].

Für sich genommen ist der MCTS, ähnlich wie Minimax, nicht für den Einsatz in Spielen mit unvollständigen Informationen oder Zufallselementen geeignet. Es gibt jedoch mit dem MCTS Expectimax (MCTSE) eine Variante, die dieses Problem löst. Ähnlich wie schon bei Expectimax wird dabei der entstandene Baum um eine weitere Knotenart, den CHANCE-Knoten, erweitert. Hierbei werden die beiden Knotenarten alternierend verwendet. Dabei kann ein Tree Knoten einen oder mehrere CHANCE-Knoten als Kindknoten haben, wobei die Anzahl von den möglichen Zufallsereignissen abhängig ist. Ein CHANCE-Knoten kann ebenfalls mehrere Kindknoten haben, wobei die Anzahl den möglichen Aktionen im jeweiligen Spielzustand entspricht.

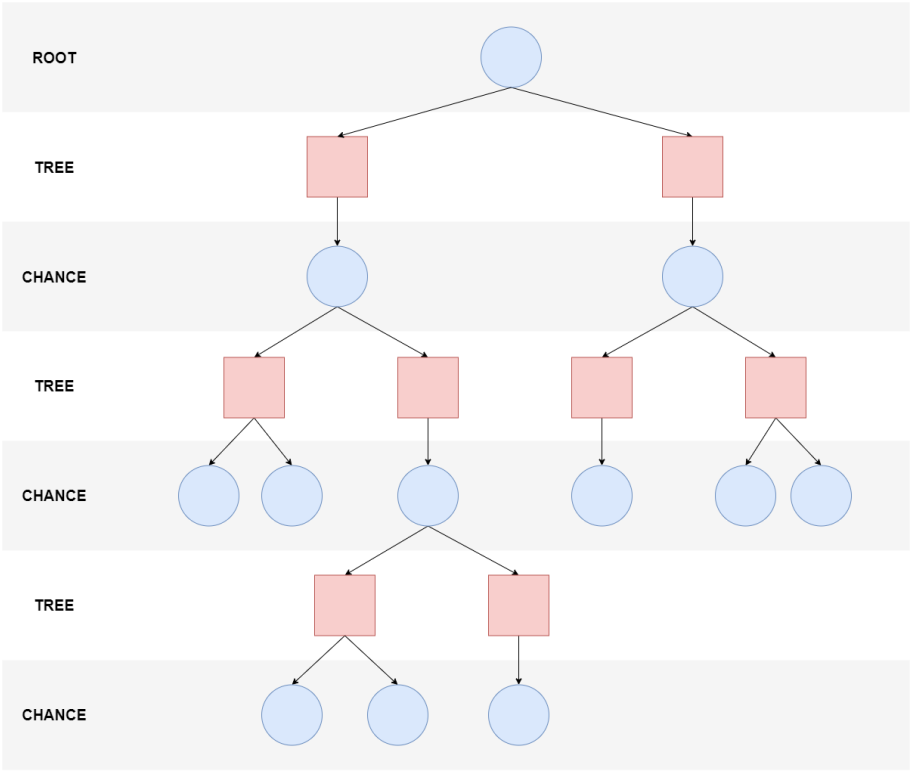


Abbildung 3.7 Beispielhafte Darstellung eines MCTSE Baums

### Temporal Difference Learning

Temporal Difference (TD) Learning vereint Ideen der Monte-Carlo-Methoden, mit denen vom Dynamic Programming. Wie die Monte-Carlo-Methoden benötigen die Temporal Difference Methoden kein Modell der Umwelt, sondern lediglich die Erfahrungen, die sie im Rahmen der Interaktion sammeln. Wie die Dynamic Programming Methoden aktualisieren die Temporal Difference Methoden ihre Erwartungswerte zumindest teilweise auf der Basis von anderen Erwartungswerten [Sutton und Barto, 1998, S. 119]. Während also die Monte-Carlo-Methoden das Endergebnis abwarten, um die erwarteten Ergebnisse zu aktualisieren, aktualisieren Temporal Difference Methoden bereits im nächsten Schritt. Für eine Monte Carlo Methode kann der Wert eines Spielzsutands aktualisiert werden mit:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Wobei $V$ eine Funktion zur Bestimmung des Wertes des jeweiligen Spielzsutands $S$ darstellt und $G$ den erhaltenen Reward am Spielende. Die Aktualisierung der Temporal Difference Methoden hingegen kann beschrieben werden als:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Hierbei sind $R_{t+1}$ der Reward und $V(S_{t+1})$ der Wert des Spielzustands im nächsten Zeitschritt [Sutton und Barto, 1998, S. 119 f.]. $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ hat an

dieser Stelle den Zweck, den Wert als Fehlersignal näher an den besten Nachfolgezustand heranzuführen [Konen und Bartz-Beielstein, 2008, S. 3]. Alle Spielzustände in Tabellenform darzustellen, um sie zu durchlaufen und für Schätzungen zu nutzen, ist oft nicht möglich, da der dadurch gebildete Raum zu groß ist [Konen und Bartz-Beielstein, 2008, S. 3]. Aus diesem Grund wird ein *Feature State* $g(s)$ des Spielzustands gebildet, welcher gemeinsam mit einem Vektor zur Gewichtung $w$ genutzt wird, um durch eine Funktion $f(w; g(s))$ eine Näherungslösung für $V(s)$ zu ermitteln. Der Algorithmus passt dabei Schritt für Schritt die Gewichte in $g$ an, um sich der korrekten Lösung anzunähern.

Input: player $p_0$ [=+1 ("X") or -1 ("O")] for the first move, initial state $s_0$, and a (partially trained) function $f(\boldsymbol{w}; \boldsymbol{g}(s_t))$ to calculate the game function $V(s_t)$.

1: $V_{\text{old}} := f(\boldsymbol{w}; \boldsymbol{g}(s_0))$ and $t := 0$ ▷ with player $-p_0$ in after state $s_0$
2: $\boldsymbol{e}_0 := \nabla_{\boldsymbol{w}} f(\boldsymbol{w}; \boldsymbol{g}(s_0))$
3: **for** $(p := p_0;\ 1;\ -p \rightarrow p,\ t++)$ **do** ▷ switch forever between players
4: select random number $r \in [0, 1]$
5: **if** $r < \epsilon$ **then** select randomly an after state $s_{t+1}$ ▷ explorative move
6: **else** select after state $s_{t+1}$ which maximizes $p \cdot f(\boldsymbol{w}; s_{t+1})$ ▷ greedy move
7: get response $V(s_{t+1}) := f(\boldsymbol{w}; \boldsymbol{g}(s_{t+1}))$ and reward $r_{t+1} := r(s_{t+1})$ from environment
8: calculate error signal $\delta_t := r_{t+1} + \gamma V(s_{t+1}) - V_{\text{old}}$
9: **if** $s_{t+1}$ is greedy move or $s_{t+1}$ is final state **then**
10: make learn step $\boldsymbol{w} := \boldsymbol{w} + \alpha \delta_t \boldsymbol{e}_t$
11: **end if**
12: **if** $s_{t+1}$ is final state **then** break ▷ exit for-loop
13: $V_{\text{old}} := y_{t+1} := f(\boldsymbol{w}; \boldsymbol{g}(s_{t+1}))$ ▷ because $\boldsymbol{w}$ has changed!
14: $\boldsymbol{e}_{t+1} := \gamma \lambda \boldsymbol{e}_t + \nabla_{\boldsymbol{w}} f(\boldsymbol{w}; \boldsymbol{g}(s_{t+1}))$ ▷ becomes $\boldsymbol{e}_t$ for the next iteration
15: **end for**

Abbildung 3.8 Pseudocode für TD($\lambda$) Algorithmus [Konen und Bartz-Beielstein, 2008, S. 3]

Im ersten Schritt ist es notwendig, den *Feature State* $g(s)$ zu definieren. Der Feature State ist eine Abstraktion des aktuellen Spielzustands, welcher die, für den Agenten relevanten Informationen als numerische Werte bereitstellt, welche die Basis für den Algorithmus darstellt. Die Herausforderung bei der Definition von einem Feature ist die Identifikation der relevanten Informationen. Für komplexe Spiele ist es unter Umständen nicht zielführend alle gegebenen Informationen ungefiltert aufzunehmen, da hierdurch das Risiko besteht, dass die Lernphase sehr viel Zeit in Anspruch nimmt und gegebenenfalls erfolglos bleibt. Die Ursache hierfür ist, dass sich eine sehr große Anzahl von Kombinationsmöglichkeiten ergibt, wodurch die Anpassung der Gewichte erschwert wird.

Zu Beginn des eigentlichen Trainings des Algorithmus werden die Gewichte in $w$ mit zufälligen Werten initialisiert. Diese Gewichte werden auf Basis der Fehlerfunktion mit jedem Durchlauf angepasst, wie in Abbildung 3.8 gezeigt wird. Hierdurch nähert

sich die Funktion immer weiter an die korrekten Werte an und ist im besten Fall in der Lage auch Spielzustände, welche nicht im Training aufgetaucht sind, erfolgreich zu bewerten.

Zur Steuerung des Lernverhaltens des Algorithmus gibt es verschiedene Parameter:

- Learning Parameter $\alpha$: Der Learning Parameter bestimmt, wie stark sich der Algorithmus in einer Iteration verändert, um den Zielwert zu erlernen, wie in Zeile 10 des Pseudocodes zu sehen ist. Zu kleine Werte führen dazu, dass sich der Wert nur langsam anpasst, was wiederum dazu führt, dass das Training mehr Zeit in Anspruch nimmt. Zu große Werte auf der anderen Seite führen dazu, dass das Risiko besteht, dass überkorrigiert wird. Aus diesem Grund handelt es sich bei $\alpha$ oft nicht um einen statischen Wert. Anstelle dessen werden zwei Werte definiert $\alpha_{init}$ und $\alpha_{final}$, wobei sich der initiale Wert schrittweise dem finalen annähert. So kann zu Beginn schnelles Lernen ermöglicht werden und durch eine Reduzierung des Werts ein Optimum gefunden werden.

- Exploration Parameter $\epsilon$: Der Explorations Parameter steuert, wie oft der Agent zufällige Züge wählt, um neue erfolgversprechende Züge zu entdecken. Zu sehen ist das Verhalten in Zeile 4 und 5 des Pseudocodes. Wenn der Wert zu niedrig ist besteht das Risiko, dass sich der Agent schnell auf die ersten erfolgreichen Züge festlegt und bessere Züge nicht entdeckt. Ist der Wert jedoch zu hoch, verschwendet der Agent viel Zeit damit, schlechte Spielzüge zu testen. Wie schon beim Learning Parameter kann es sinnvoll sein, nicht einen konstanten Wert zu verwenden, sondern den Wert im Laufe des Trainings zu reduzieren.

- Zerfallrate $\lambda$: Die Zerfallrate, als Wert $\lambda \in [0, 1]$ , steuert den Zerfall der *eligibility trace* [Sutton und Barto, 1998, S. 287 f.]. Die eligibility trace erfasst, welche Relevanz ein Zustand für den aktuellen Reward hatte und in welchem Maße er demzufolge angepasst werden sollte. Dabei wird der Wert eines Zustands bei einem Besuch auf 1 gesetzt und im Laufe des weiteren Verlaufs über die Zerfallrate reduziert. Bei einer Zerfallrate von 1 wird der Wert nicht reduziert und alle besuchten Zustände werden gleichermaßen angepasst, sollte die Zerfallsrate 0 entsprechen, werden die früheren Zustände nicht angepasst.

- Discount $\gamma$: Der Discount Parameter $\gamma$ ist ein Wert mit $0 \leq \gamma \leq 1$ und dient dazu, den Einfluss zukünftiger Rewards auf den Wert eines Zustands zu steuern. Ist der Wert 0, werden zukünftige Rewards nicht berücksichtigt, sondern nur der direkte. Je größer der Wert, desto mehr Gewicht liegt auf den zukünftigen Rewards und desto weitsichtiger agiert der Agent. Typischerweise werden Werte nahe 1 gewählt. [Konen und Bartz-Beielstein, 2008, S. 3]

### TD-N-Tuple Agents

Die Herausforderung im Temporal Difference Learnings, wie es im vorhergehenden Kapitel beschrieben wurde, ist die Definition eines geeigneten Feature-Vektors. Für Spiele mit einer großen Komplexität wie Schach, Go oder Othello, ist der gegebene Raum an Informationen sehr groß, wodurch die Erstellung eines geeigneten Feature-Vektors schwierig ist. Der $n$-Tupel Ansatz stellt einen Lösungsansatz für dieses Problem dar. Den Ursprung hat die Methodik bereits in den 1950er Jahren und wurde unter anderem zur maschinellen Zeichenerkennung genutzt [Bledsoe und Browning, 1959]. Der Kerngedanke des Verfahrens ist es, bestimmte Punkte aus dem gegebenen Raum an Informationen zu entnehmen, welche als Tupel gespeichert werden und als Basis für die weitere Verarbeitung dienen. Die Methode wurde erstmals von Lucas mit den TDL-Methoden kombiniert und auf Spiele angewandt [Lucas, 2008a]. Eine der ersten Anwendungen war das Spiel Othello, welches mit einem 8x8 großen Spielfeld eine geeignete Herausforderung darstellt. Hierbei konnte Lucas zeigen, dass nach nur 500 Spielen gegen sich selbst, der auf $n$-Tupel basierte Agent in der Lage war, den zu diesem Zeitpunkt stärksten Agenten, unter den gegebenen Bedingungen, in 70% der Spiele zu besiegen [Lucas, 2008b, S. 16]. Somit konnte der Agent eine bessere Leistung erzielen und das trotz einer kürzeren Lernphase.

Für den $n$-Tupel Agenten werden Informationen über den aktuellen Spielzustand in Tupeln der Größe $n$ zur Verfügung gestellt. Wenn jeder der im Tupel enthaltenen Informationen $m$ mögliche Ausprägungen hat, kann ein Informationspunkt als Zahl mit $n$ Stellen in der Basis $m$ dargestellt werden. Hierdurch ergibt sich als notwendige Größe des Arrays $n^m$ [Lucas, 2008b, S. 9]. Hierdurch wird deutlich, dass es wichtig ist, die Größe des Tupels mit Bedacht zu wählen. Im Beispiel von Othello kann man von drei möglichen Werten ausgehen, welche die einzelnen Positionen annehmen können: Weiß, Schwarz, leer. Falls ein Tupel der Größe 6 gewählt wird, ergibt sich ein Array der Größe: $6^3 = 216$. Sollte hingegen ein Tupel der Größe 10 gewählt, wird bereits ein Array mit 1.000 Positionen benötigt.

Ein weiterer wichtiger Aspekt ist die Wahl der Informationspunkte, die in einem Tupel zusammengefasst werden sollen. Hierbei gibt es unterschiedliche Methoden:

- *Fixed*: Die gewählten Informationspunkte, die in einem Tupel zusammengefasst werden, können fest definiert werden. Dies hat den Vorteil, dass relevante Bereiche des Spiels auf jeden Fall mit einbezogen werden, und dass die Möglichkeit besteht, Zusammenhänge zwischen verschiedenen Bereichen zu modellieren. Auf der einen Seite können auf diesem Weg erfolgreiche Strategien nachgebildet werden, indem die vermeintlich wichtigen Bereiche in einem Tupel zusammengefasst werden, auf der anderen Seite verliert der Agent die Chance, potentiell wichtigere Bereiche zu identifizieren. In manchen Spielen gibt es

jedoch auch pragmatische Gründe Beziehungen definiert vorzugeben, beispielsweise, wenn bestimmte Felder in einem relevanten Zusammenhang stehen, der ansonsten verloren gehen könnte.

- *Random Point*: Bei diesem Vorgehen werden $n$ Punkte zufällig aus dem gegebenen Informationszustand ausgewählt, ohne dass die Beziehungen zu anderen Punkten berücksichtigt werden. Dieses Vorgehen birgt das Risiko, dass lokale Muster unentdeckt bleiben können, hat allerdings den Vorteil, dass keine Beziehung zwischen den Punkten modelliert werden muss.

- *Random Walk*: Um die Punkte mit einem *Random Walk* auszuwählen, ist es relevant, die Nachbarschaftsbeziehung eines Punktes zu modellieren. Im ersten Schritt wird ein zufälliger Punkt gewählt. Anschließend wird ein zufälliger Nachbar des Punkts gewählt und dem Tupel hinzugefügt. Dieses Vorgehen wird solange wiederholt bis die gewünschte Tupelgröße erreicht wurde, wobei dennoch jeder Punkt nur einmal in einem Tupel enthalten sein kann.

### 3.2.4 Limitationen & Anpassungen

Das GBG-Framework bietet zahlreiche Möglichkeiten, welche die Entwicklung von neuen Spielen und Agenten erleichtert. Jedoch gibt es zahlreiche Spiele mit unterschiedlichsten Rahmenbedingungen, was dazu führt, dass nicht zwingenderweise alle notwendigen Aspekte eines Spiels im Framework bereits bedacht und implementiert sind. Das Framework wird jedoch kontinuierlich weiterentwickelt und neue Anforderungen werden in das bestehende Framework eingebaut. Im Verlaufe der Ausarbeitung dieser Arbeit wurden zwei unterschiedliche Aspekte in das Framework mit aufgenommen. In implementierten Spielen wie 2048, Tic-Tac-Toe oder Hex sind alle Informationen allen Spielern zu jeder Zeit bekannt. Ein Grundproblem des Pokerspiels ist jedoch das der unbekannten Information. Die Spieler kennen die Handkarten ihrer Gegner nicht. Umgesetzt wurde dieses Konzept mithilfe des *Partial States*. Im Partial State werden alle Informationen, die einem Spieler unbekannt sind, aus dem *Game State* entfernt und in einem terminalen Spielzustand einer Simulation durch zufällig gewählte Elemente ergänzt, um einen Gewinner des simulierten Spiels zu ermitteln. Ein zweiter Aspekt ist, dass die betrachteten Episoden in den bisher etablierten Spielen den kompletten Spielumfang umfassen. In einem Spiel wie Poker ist es jedoch sinnvoll, keine kompletten Spiele zu analysieren, sondern die Ergebnisse einzelner Spielrunden zu betrachten. Um dieses Konzept zu etablieren, wurden die verschiedenen Agenten angepasst, um einen frühzeitigen Abbruch zum Rundenende zu ermöglichen.

Neben den bereits etablierten Erweiterungen gibt es einen Aspekt, der nicht im Rahmen der Arbeit implementiert werden konnte, jedoch eine Limitierung des Fra-

meworks darstellt. Bisherige Agenten verfolgen nur pure Strategien. Sie bestimmten deterministisch für einen bestimmten Spielzustand eine beste Aktion, die sie in diesem Spielzustand in jedem Fall auswählen. Die einzige Ausnahme von diesem Vorgehen besteht, wenn ein Agent Aktionen gleich bewertet und daher zufällig eine von ihnen auswählt. Eine Nutzung von gemischten Strategien, also einer Strategie, die von einem Zufallsmechanismus bestimmt wird [Holler et al., 2019], ist bisher nicht vorgesehen.

## 3.3 Counterfactual Regret Minimization

Bei der Counterfactual Regret Minimization Methode handelt es sich um ein relativ neues Konzept, welches erst in den 2000er Jahren entwickelt wurde und einen großen Fortschritt für Spiele mit unvollständigen Informationen bedeutete [Neller, 2013, S. 1]. Zum Zeitpunkt der Erstellung dieser Arbeit ist kein Agent, der auf diesen Methoden aufbaut, im General Board Games Framework enthalten [Konen, 2020, S. 11]. Deshalb wurde diese Methode nicht in der praktischen Auswertung thematisiert. Diese Art von Methoden stellen jedoch eine wichtige Grundlage für viele der KI-Agenten dar, die in der Lage sind, erfolgreich Poker zu spielen, weshalb sie im weiteren Verlauf dieses Abschnitts vorgestellt werden. Begonnen wird dabei mit der grundlegenden Erklärung von *Regret* und *Regret Matching*. Anschließend wird das Konzept der *Counterfactual Regret Minimization* (CFR) vorgestellt und abschließend dessen Erweiterung CFR+ betrachtet, welche eine wichtige Voraussetzung für den Erfolg von Cepheus darstellte.

### 3.3.1 Regret Minimization

Der *Regret* (englisch für Bedauern oder Reue), eine Aktion nicht gewählt zu haben, wird definiert als die Differenz zwischen der nicht gewählten und der gewählten Aktion [Neller, 2013, S. 4]. Als Beispiel sei das Spiel *Stein-Schere-Papier* gegeben, wie es in Tabelle 3.1 abgebildet wird, gegeben, wobei die Funktion $u$ dem Wert der Aktionen entspricht.

Tabelle 3.1 Rewards im Spiel Stein(R)- Schere(S)-Papier(P)

| | R | P | S |
|---|---|---|---|
| R | 0 | -1 | 1 |
| P | 1 | 0 | -1 |
| S | -1 | 1 | 0 |

Sollte der Spieler die Aktion *Stein* wählen und sein Gegner Papier, dann ergibt sich als Regret für die Aktion *Schere*:

$$u(Schere, Papier) - u(Stein, Papier) = 1 - (-1) = 2$$

Und für die Aktion *Papier*:

$$u(Papier, Papier) - u(Stein, Papier) = 0 - (-1) = 1$$

Negative Werte können ignoriert werden.

Um aus dem Regret eine Strategie abzuleiten, kann aus dem bisherigen positiven Regret die nächste Aktion abgeleitet werden, indem der positive Regret der Aktion dem gesamten Regret gegenübergestellt wird [Neller, 2013, S. 4]. Im vorhergehenden Beispiel wären die Wahrscheinlichkeiten Stein zu wählen 0, die Wahrscheinlichkeit Papier zu wählen $\frac{1}{3}$ und die Wahrscheinlichkeit Schere zu wählen $\frac{2}{3}$.

Tabelle 3.2 Beispielhaftes Regret Matching bzw.Minimization am Beispiel von Stein, Schere, Papier

| | | | Regret | | | Kum.Regret | | | Wahrscheinlichkeit | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Spieler | Gegner | Reward | R | S | P | R | S | P | R | S | P |
| R | P | -1 | 0 | 2 | 1 | 0 | 2 | 1 | 0,0% | 66,7% | 33,3% |
| S | R | -1 | 1 | 0 | 2 | 1 | 2 | 3 | 16,7% | 33,3% | 50,0% |
| P | P | 0 | -1 | 1 | 0 | 0 | 3 | 3 | 0,0% | 50,0% | 50,0% |
| P | S | -2 | 2 | -1 | 0 | 2 | 2 | 3 | 28,6% | 28,6% | 42,9% |
| S | S | 0 | 1 | 0 | -1 | 3 | 2 | 2 | 42,9% | 28,6% | 28,6% |
| R | R | 0 | 0 | -1 | 1 | 3 | 1 | 3 | 42,9% | 14,3% | 42,9% |
| P | P | -1 | 1 | 2 | 0 | 4 | 3 | 3 | 40,0% | 30,0% | 30,0% |

Nehmen wir an, im nächsten Schritt wäre die wahrscheinlichste Aktion (Schere) gewählt worden, und der Gegner hätte Stein gewählt, dann würde sich als Regret für Stein 1 ergeben und für Papier 2. Auf diesem Weg kann Schritt für Schritt die Strategie weiter aufgebaut werden, wie in Tabelle 3.2 gezeigt wird.

Falls dies im praktischen Spiel angewandt wird, birgt es jedoch das Risiko, dass der Gegner diese schrittweise Annährung ausnutzen kann, um seinen Gewinn zu maximieren. Wenn der Spieler sich wiederum dieser Strategie angepasst hat, kann der Gegner eine andere Schwachstelle ausnutzen. Dieses Vorgehen bietet jedoch die Möglichkeit, im Spiel mit sich selbst eine Strategie zu bilden und ein Gleichgewicht zu finden. [Neller, 2013, S. 4]

### 3.3.2 Counterfactual Regret Minimization

Das bisher betrachtete Regret Matching bzw. Regret Minimization zeigt einen Lösungsweg, sollte nur eine einzige Aktion notwendig sein. Counterfactual Regret Minimization (CFR) erweitert dieses Vorgehen auf Spiele, bei denen mehrere Aktionen aufeinander folgen, um einen Endzustand im Spiel zu erreichen. Vorgestellt wurde es 2007 von Zinkevich et al. als eine Methode zur Lösung komplexer Spiele mit unvollständigen Informationen zur Findung des Nash-Gleichgewichts. [Zinkevich et al., 2007, S. 1729] Als Beispiel hierfür zeigten sie das Lösen einer vereinfachten Version von Poker mit $10^{12}$ Zuständen, was einem deutlich größeren Umfang im Vergleich zu der bis dahin verwendeten Methoden entspricht.

Bei der Methode wird das Vorgehen, wie in anderen Methoden auch, als Baum dargestellt, der aus verschiedenen Arten von Knoten besteht. Es können entweder Knoten sein, in denen ein Spieler eine Entscheidung trifft, Zufallsknoten, in denen Aktionen Zufallselemente darstellen und Blattknoten, die das Ende eines Spiels und damit einen konkreten Wert darstellen. Anders jedoch als in anderen Methoden wird neben dem aktuellen Spielzustand auch der Informationszustand (*Information Set*) verwendet, welcher verschiedene Spielzustände beschreibt, die für den Spieler nicht unterscheidbar sind [Zinkevich et al., 2007, S. 1730]. So ist im Beispiel Kuhn Poker dem Spieler mit einer Dame nicht bekannt, ob sein Gegner einen Buben oder einen König hat, wodurch es zwei unterschiedliche Spielzustände darstellt, dem Spieler jedoch nicht klar ist, in welchem er sich befindet, es also nur einen einzigen Informationszustand darstellt.

Im Laufe einer Iteration wird der komplette Spielbaum durchlaufen, wobei zwei Aspekte aktualisiert werden. Zum einen werden die Ergebnisse der Blattknoten an die Elternknoten propagiert, zum anderen wird die Wahrscheinlichkeit aktualisiert, mit der ein Knoten erreicht wird.

```
Initialize cumulative regret tables: ∀I, r_I[a] ← 0.
Initialize cumulative strategy tables: ∀I, s_I[a] ← 0.
Initialize initial profile: σ^1(I, a) ← 1/|A(I)|

function CFR(h, i, t, π_1, π_2):
if h is terminal then
    return u_i(h)
else if h is a chance node then
    Sample a single outcome a ~ σ_c(h, a)
    return CFR(ha, i, t, π_1, π_2)
end if
Let I be the information set containing h.
v_σ ← 0
v_{σ_{I→a}}[a] ← 0 for all a ∈ A(I)
for a ∈ A(I) do
    if P(h) = 1 then
        v_{σ_{I→a}}[a] ← CFR(ha, i, t, σ^t(I, a) · π_1, π_2)
    else if P(h) = 2 then
        v_{σ_{I→a}}[a] ← CFR(ha, i, t, π_1, σ^t(I, a) · π_2)
    end if
    v_σ ← v_σ + σ^t(I, a) · v_{σ_{I→a}}[a]
end for
if P(h) = i then
    for a ∈ A(I) do
        r_I[a] ← r_I[a] + π_{-i} · (v_{σ_{I→a}}[a] − v_σ)
        s_I[a] ← s_I[a] + π_i · σ^t(I, a)
    end for
    σ^{t+1}(I) ← regret-matching values computed using Equation 5 and regret table r_I
end if
return v_σ

function Solve():
for t = {1, 2, 3, ..., T} do
    for i ∈ {1, 2} do
        CFR(∅, i, t, 1, 1)
    end for
end for
```

Abbildung 3.9 Pseudocode Counterfactual Regret Minimization (with chance sampling) [Neller, 2013, S. 12]

In Abbildung 3.9 wird der Ablauf des Vorgehens dargestellt. Dabei sind die einzelnen Komponenten definiert als:

- $I$: Der gegebene Informationszustand
- $h$: Historie der gewählten Aktionen inklusive der Zufallsaktionen
- $r$: Der Regret, eine Aktion nicht gewählt zu haben
- $\sigma$: Die Strategie beider Spieler. $\sigma^t$ entspricht der Strategie zum Zeitpunkt $t$.
- *P(h)*: Der aktive Spieler nach der Historie $h$

- $\sigma_{I \to a}$: Dieselbe Strategie wie $\sigma$, jedoch wird im Informationszustand $I$ die Aktion $a$ gewählt.
- $v$: stellt den *counterfactual* Wert dar.

Anhand des Codes wird deutlich, dass die Strategie den Baum mehrfach in verschiedenen Iterationen durchläuft (siehe Zeile 33) und im Spiel gegen sich selbst, zuerst die Strategie des einen Spielers und dann die des anderen ermittelt (Zeile 34). Im Rahmen einer Iteration wird zuerst der oberste Knoten aufgerufen, der wiederum alle möglichen Aktionen in der aktuellen Situation durchläuft und rekursiv die Funktion erneut aufruft, wobei zum einen die Historie um die gewählte Aktion erweitert wird, zum anderen die Wahrscheinlichkeit angepasst wird. Nachdem alle Kindknoten durchlaufen wurden, kann der Regret der einzelnen Aktionen ermittelt werden, womit die gesamte Strategie aktualisiert wird. Das Vorgehen hierbei ähnelt dem bisherigen Vorgehen, in dem der Regret der einzelnen Aktion ins Verhältnis zum kumulativen Regret gesetzt wird. [Neller, 2013, S. 11] Die formale Ableitung der Strategie der nächsten Iteration zeigt Abbildung 3.10, wobei $R_i^{T,+}(I,a)$ den gesamten Regret einer bestimmten Aktion im Informationszustand $I$ entspricht.

$$\sigma_i^{T+1}(I)(a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a \in A(I)} R_i^{T,+}(I,a)} & \text{if } \sum_{a \in A(I)} R_i^{T,+}(I,a) > 0 \\ \frac{1}{|A(I)|} & \text{otherwise.} \end{cases}$$

Abbildung 3.10 CFR-Strategie zum Zeitpunkt T+1 [Zinkevich et al., 2007, S. 1733]

Insgesamt wird deutlich, dass es sich bei dieser Methode um ein zeitaufwendiges Vorgehen handelt, da mit jeder Iteration der komplette Spielbaum durchlaufen werden muss. Eine Möglichkeit der Optimierung der Laufzeit ist eine Parallelisierung des Vorgehens, wobei unterschiedliche Informationszustände auf unterschiedliche Knoten verteilt werden. Eine andere Möglichkeit der Optimierung, neben der Nutzung von mehr Hardware, ist die Erweiterung des Algorithmus. Ein Beispiel dafür ist Lazy CFR, bei dem ein Informationszustand in verschiedene Segmente aufgeteilt wird und die Anpassungen der Strategie nur zu Beginn eines Segments, nicht kontinuierlich erfolgt, wodurch ein Großteil der benötigten Ressourcen gespart wird [Zhou et al., 2018, S. 1]. Ein weiteres Beispiel ist die Monte Carlo Counterfactural Regret Minimization (MCCFR). Bei dieser Methode werden die verschiedenen Aktionsabläufe, die in einem Spielende resultieren, in Blöcke unterteilt. Aus diesen Blöcken können auf verschiedene Arten Stichproben genommen werden, wobei immer nur ein Teil der Knoten aktualisiert wird [Chen und Kaneko, 2019, S. 2]. Dies stellt einen deutlichen Vorteil in der Menge der betrachteten und aktualisierten Knoten dar, wie in Abbildung 3.11 deutlich wird.

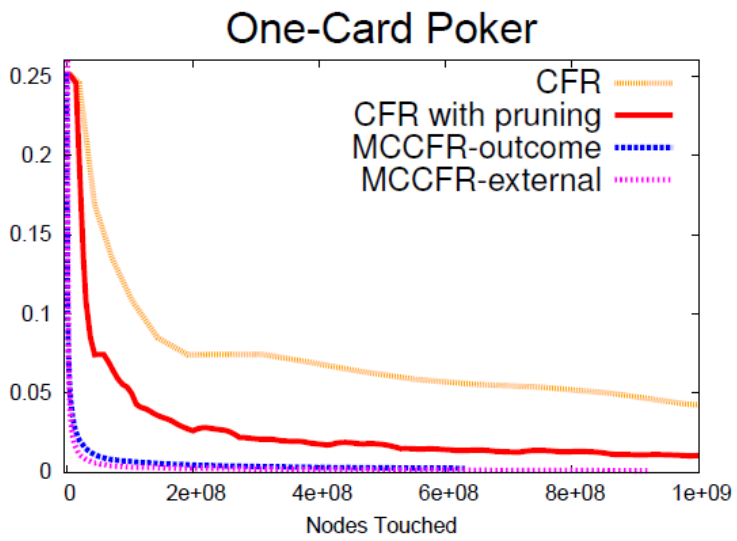


Abbildung 3.11 Vergleich CFR und MCFR [Lanctot et al., 2009, S. 1086]

### 3.3.3 CFR+

CFR+ stellt eine Erweiterung von CFR dar, welche es ermöglicht hat, eine Lösung für Heads-up limited Texas Hold' zu finden, ohne dass Vereinfachungen notwendig gewesen sind [Bowling et al., 2015b, S. 5]. Hierdurch ist ein großer Sprung in der Qualität von Poker Agenten möglich gewesen, wie in Abbildung 3.12 deutlich wird.

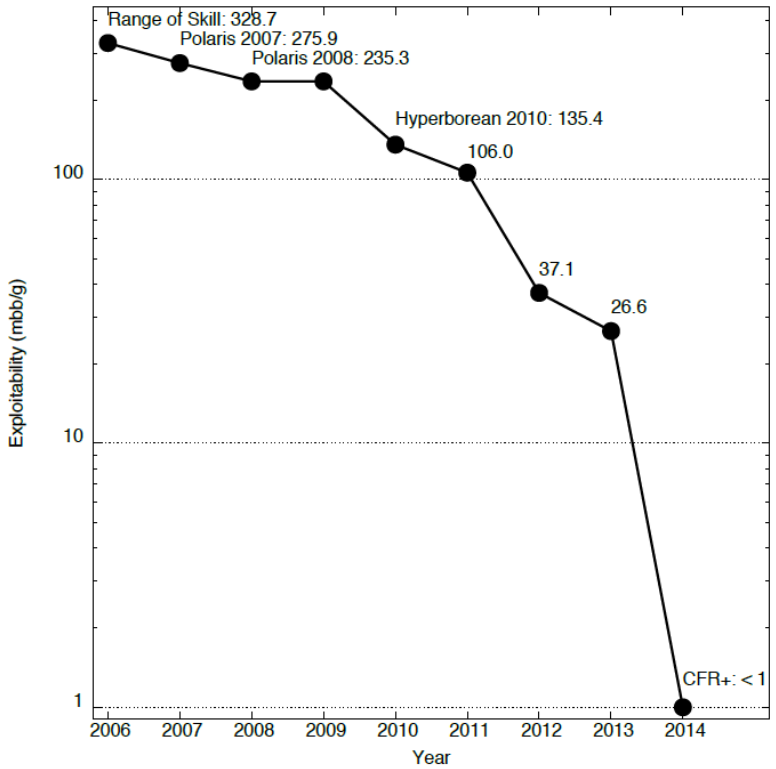


Abbildung 3.12 Exploitability von KIs für Heads-up limited Hold'em [Bowling et al., 2015b, S. 6]

Der Unterschied zwischen CFR und CFR+ besteht im *Regret Matching* [Bowling et al., 2015b, S. 9]. Für CFR ist es möglich, dass große Mengen negtive Regrets angesammelt werden. Dies hat zur Folge, dass, auch wenn eine Aktion einen hohen Regret erhält, erst einmal der negative Regret abgebaut werden muss. So entsteht das Risiko, dass vielversprechende Aktionen nicht direkt weiter verfolgt werden. Aus diesem Grund werden im Rahmen von CFR + die Counterfactual Regret Values definiert als:

$$Q^0(I,a) = 0$$

$$Q^t(I,a) = (Q^{t-1}(I,a) + R^t(I,a) - R^{t-1}(I,a))^+$$

Woraus sich für das Matching:

$$\sigma_i^{t+1}(a|I) = \begin{cases} \frac{Q_i^t(I,a)}{\sum_{a' \in A} Q_i^t(I,a')} & \text{if } \sum_{a' \in A} Q_i^t(I,a') > 0 \\ \frac{1}{|A|} & \text{otherwise} \end{cases}$$

ergibt. Auf diesem Weg können Aktionen, auch wenn sie in der Vergangenheit schlecht erschienen und damit einen negativen Regret besitzen würden, sofort erneut verfolgt werden, sobald sie gute Ergebnisse erzielt haben [Bowling et al., 2015a, S. 147].

# 4 Aktueller Forschungsstand

Das Spiel Poker stellte über einen langen Zeitraum eine große Herausforderung für den Bereich der Künstlichen Intelligenz dar. Das Spiel umfasst viele interessante Aspekte. Vor allem die unvollständigen Informationen, die bestimmte Spielzüge, wie das Bluffen, erst ermöglichen und damit gleichzeitig die Komplexität des Spiels steigern, können im Pokerspiel gut untersucht werden.

Im Laufe der letzten Jahrzehnte wurden jedoch immer weitere Fortschritte und damit Erfolge erzielt, wodurch das Spiel immer weiter gelöst wurde und somit immer weniger Vereinfachungen notwendig wurden. Polaris (2007), der als einer der ersten Bots gegen professionelle Pokerspieler angetreten ist und dabei knapp verlor, konnte nach einigen Verbesserungen im Jahre 2008 einen Sieg davontragen. Der nächste große Durchbruch konnte von Cepheus (2015) erzielt werden, der es geschafft hat, eine optimale Strategie für heads-up limit Texas Hold'em zu ermitteln und somit das Spiel gelöst hat. Im nächsten Schritt konnte Libratus (2017) im Heads up *no-limit* Texas Hold'em gegen verschiedene Pokerspieler gewinnen. Der letzte Durchbruch konnte von Pluribus (2019) erzielt werden, der No-Limit Texas Hold'em mit sechs Spielern spielen und im Spiel gegen professionelle Spieler gewinnen konnte, was laut dem verantwortlichen Forscherteam die letzte große Herausforderung darstellte.

Im folgenden Kapitel wird eine Auswahl der oben genannten Bots betrachtet. Hierbei wird zum einen ihr Aufbau und damit die Funktionsweise beschrieben, aber auch die von ihnen erreichten Erfolge skizziert. Zu Beginn findet sich jedoch eine Einleitung mit wiederkehrenden Elementen, die von den verschiedenen Bots genutzt wurden.

## 4.1 Allgemein

Auch wenn der Aufbau der Bots unterschiedlich ist, greifen sie auf gemeinsame Vorgehensweise beziehungsweise Metriken zurück. Zwei dieser Aspekte sind die *Exploitability* und *AIVAT*, die in diesem Abschnitt kurz vorgestellt werden.

### 4.1.1 Epxloitability

Die *Exploitability* bezeichnet die Abweichung vom ermittelten Spielwert der Strategie im Spiel gegen den schlimmsten anzunehmenden Gegner. Dieser Wert ist jedoch

nicht unabhängig, sondern steht im Verhältnis zu den Rahmenbedingungen des gegebenen Spiels. Wenn zum Beispiel in einem Spiel die Spieler mit 100.000 Chips beginnen und die Blinds eine Höhe von 500 und 1000 haben, dann ist die Situation direkt vergleichbar mit einem Spiel, bei dem die Spieler lediglich 1000 Chips erhalten, die Blinds jedoch 5 und 10 betragen. Wenn die Exploitability an einem absoluten Wert gemessen werden würde, dann wäre der Wert mit gleicher Strategie im ersten Spiel um den Faktor 100 größer. Um diesen Effekt zu vermeiden, wird der Wert üblicherweise in Relation zum gegebenen Big Blind ermittelt. In der aktuellen Forschung wird inzwischen typischerweise die Einheit milli-big-blinds pro Spiel ($mbb/g$) verwendet, wobei dies dem Wert von $\frac{1}{1000}$ des gegebenen Big Blinds entspricht. [Bowling et al., 2015b, S. 3]

Wenn beispielsweise die gegebene Strategie vorsieht in jedem Fall zu passen, beträgt die Exploitability 750mbb/g. Der Agent verliert gegen jede Strategie, damit auch gegen die schlimmsten anzunehmenden Gegner, abwechselnd den Big Blind (1000 mbb) und den Small Blind (500 mbb).

Normalerweise ist die Berechnung dieses Werts deutlich aufwendiger, da im ersten Schritt die stärkste Gegenstrategie identifiziert werden muss.

Als Faustregel für professionelle menschliche Pokerspieler sollte ein Gewinn von mindestens 50 mbb/g angestrebt werden [Bowling et al., 2015b, S. 3].

### 4.1.2 AIVAT

Die Auswertung der Ergebnisse eines Pokerspiels kann eine große Herausforderung darstellen. Dies resultiert unter anderem aus der großen Menge an zufälligen Möglichkeiten durch die Verteilung der Handkarten und der Community Karten. Es kann sein, dass der stärkste Spieler viele schwache Hände in Folge erhält und daher verliert oder einen Bluff in einer ungünstigen Situation wählt, der zu einem Verlust führt, der in den meisten anderen Situationen einen Sieg bedeutet hätte [Moravčík et al., 2017, S. 9]. Daraus folgt, dass es selbst nach 80.000 gespielten Runden sein kann, dass nicht mit Sicherheit statistisch bestimmt werden kann, dass ein Spieler besser ist, auch wenn sich die Qualität der Spieler deutlich unterscheidet. Oft ist es nicht möglich beliebig viele Runden zu spielen, um die Ungenauigkeit auf diesem Weg zu minimieren. Die im weiteren Verlauf der Arbeit vorgestellten Bots nutzen das *action-informed value assessment tool* (AIVAT). Das Tool verwendet Methoden, um den Einfluss des Zufalls zu reduzieren. Wenn ein Pokerspieler eine gute Hand bekommt ist die Wahrscheinlichkeit hoch, dass er auch ein gutes Ergebnis erzielt, bei einer schlechten Hand ist die Wahrscheinlichkeit hoch, dass das Ergebnis ebenfalls schlecht ist. Aus diesem Grund wird ein gewisser Betrag des Ergebnisses reduziert, wenn der Spieler eine gute Hand hatte, und erhöht sollte er eine schlech-

te Hand gehabt haben. Darüber hinaus verwendet es *Imaginary Observations*, das heißt, dass, wenn die Strategie des Spielers bekannt, ist mögliche Alternativzüge ebenfalls betrachtet und gewertet werden. [Burch et al., 2016, S. 2]

## 4.2 Cepheus

Cepheus ist ein, von der Computer Research Group der Universität von Alberta entwickelter Bot, der das Spiel heads-up limit Texas Hold'em gemeistert hat. Das involvierte Team bestand aus Informatikern der Universität rund um Michael Bowling mit Unterstützung durch Oskari Tammelin als Software Entwickler. [Ball, 2015]

### 4.2.1 Erfolge

Der Bot war der erste, der in der Lage war, eine optimale Strategie zu identifizieren oder zumindest eine nahezu optimale Strategie. Eine Strategie ist optimal, wenn sie eine Nash-Gleichgewicht Strategie ist, welche dazu führen würde, dass die Exploitability einem Wert von 0 mbb/g entspricht. Einen Wert von 0 kann der Bot nicht erreichen, jedoch einen Wert von 0,986 mbb/g.

Für das Lösen von Spielen gibt es unterschiedliche Definitionen, je nach Grad der Vollständigkeit der Lösung. Eine davon:

> weakly solved: For the initial position(s), a strategy has been determined to obtain at least the game-theoretic value of the game, for both players, under reasonable resources. [Allis u a., 1994, S. 8]

Selbst wenn eine Strategie theoretisch ausgenutzt werden könnte und somit eine *Exploitability* > 0 hat, ist dies keine Garantie dafür, dass ein Gegner, welcher der besten Gegenstrategie folgt, am Ende einer begrenzten Anzahl betrachteter Spiele gewinnt. [Bowling et al., 2015a, S. 147] Die Begründung hierfür ist, dass das Spiel eine große Varianz durch die gegebenen Zufallselemente aufweist. Sollten beide Spieler ähnlich stark sein, also die Exploitability nahe 0 sein, wird es eine große Anzahl gespielter Runden benötigen, um mit statistischer Sicherheit festzustellen, dass einer der Spieler überlegen ist.

Das Forscherteam für Cepheus interpretiert das Spiel als gelöst, wenn eine Person in ihrer Lebenszeit nicht mit statistischer Sicherheit in einem Konfidenzintervall von 95% sagen kann, dass eine Strategie überlegen ist. Hierbei wird von 200 Pokerspielen pro Stunde für 12 Stunden am Tag über 70 Jahre ausgegangen und einer Standard-

abweichung von 5 bb/g. Hierdurch ergibt sich:

$$\frac{1.64 * 5}{\sqrt{200 * 12 * 365 * 70}} \approx 0,00105$$

Und somit der Grenzwert von etwa 1 mbb/g [Bowling et al., 2015a, S. 147]

Wenn man diese Definition als Maßstab akzeptiert, dann kann man das Spiel als gelöst betrachten, da die erzielte Exploitability mit 0,976 mbb/g unter dem definierten Grenzwert liegt.

### 4.2.2 Aufbau

Lange Zeit stellte die große Menge an Information Sets für Heads-Up limited Texas Hold'em eine große Herausforderung dar. Es wurden verschiedene Vereinfachungen angenommen, um diese Anzahl zu verringern, wodurch Erfolge für bis zu $3.8 * 10^{10}$ Information Sets erzielt werden konnten, was zwar deutliche Fortschritte darstellt, jedoch weiterhin deutlich unter den erforderlichen $1.38 * 10^{13}$ bleibt. Ursache hierfür ist, dass mit den bisherigen Methoden sehr große Mengen an Speicher notwendig sind, um die erforderlichen Berechnungen durchzuführen.

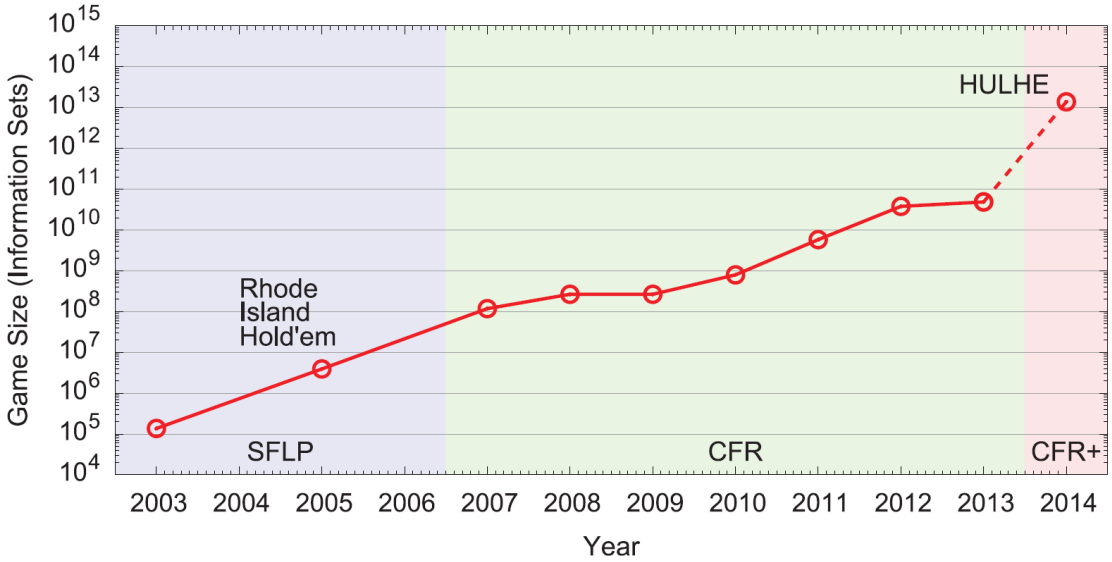


Abbildung 4.1 Zunehmende Größe der verarbeiteten Information Sets im Laufe der Zeit [Bowling et al., 2015a, S. 146]

Aus dieser Problematik resultierten nach Bowling zwei Herausforderungen:

1. Für jedes Information Set müssen die Werte gespeichert werden. Durch die große Anzahl an Information Sets führt das dazu, dass eine sehr große Menge an Speicher benötigt wird. Selbst bei einfacher Genauigkeit, also je nach

Rechnerarchitektur zwischen 7 und 8 Stellen, werden etwa 262 TB Speicher benötigt.

2. Erfahrungen zeigen, dass eine Vergrößerung der Information Sets um drei Größenordnungen, ebenfalls eine Erhöhung der notwendigen Rechenleistung um mindestens drei Größenordnungen mit sich bringt.

Um das Speicherproblem zu lösen, wurden die ermittelten Werte skaliert, in Integer umgewandelt, geordnet und anschließend komprimiert. Auf diesem Weg war es möglich, den gesamten Speicherbedarf auf etwa 17TB zu reduzieren. Die Daten wurden anschließend auf ein Computercluster verteilt, in dem jedem Rechner bestimmte Sub-Spiele und damit Information Sets zugeordnet wurden. Durch die notwendige Komprimierung und Dekomprimierung der Daten entsteht ein gewisser administrativer Aufwand, der jedoch durch Cachingverfahren auf etwa 5% der Gesamtlaufzeit reduziert werden konnte. [Bowling et al., 2015a, S. 147]

Das Problem der Rechenleistung wurde durch die Anwendung einer neuen Methode, der CFR+, gelöst. Das Forscherteam konnte empirisch zeigen, dass weniger Rechenleistung benötigt und dennoch eine massive Parallelisierung ermöglicht wurde.

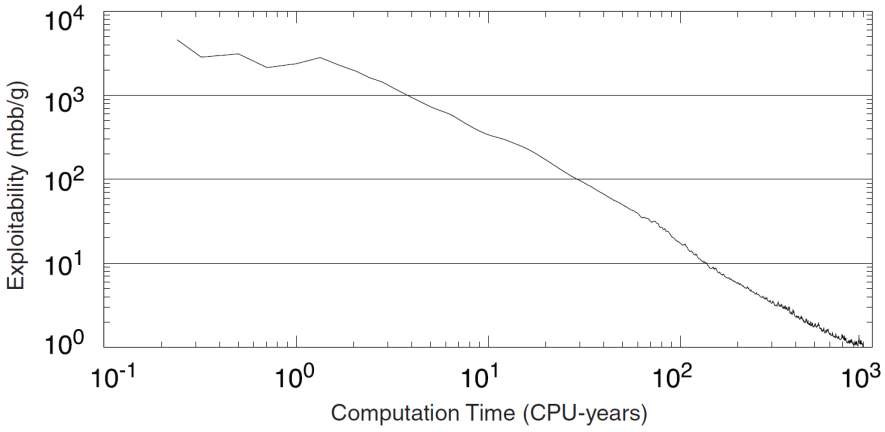


Abbildung 4.2 Exploitability der Näherungslösung mit zunehmender Rechenzeit [Bowling et al., 2015a, S. 147]

Wie Abbildung 4.2 zeigt ist trotz dieser Verbesserungen für die Ermittlung der Strategie ein sehr hoher Aufwand notwendig. Das Team nutzte ein Computercluster mit 200 Rechnern mit jeweils 24 Kern-2,1 GHz Prozessoren, 32 GB RAM und 1TB HDD, welches 68,5 Tage an der Berechnung arbeitete [Bowling et al., 2015a, S. 147]. Hieraus ergibt sich also eine Berechnungszeit von über 900 "*Core Years*"$(200 * 24 * 68,5$ Tage).

## 4.3 DeepStack

DeepStack wurde von Informatikern der Universität von Alberta in Zusammenarbeit mit Mitarbeitern der Charles University und der Tschechischen Technischen Universität in Prag entwickelt [Gibney, 2017]. DeepStack spielt im Gegensatz zu Cepheus Heads-Up ***no*** limit Texas Hold'em, was die Komplexität des Spiels noch einmal erhöht Während in der Limit Variante mit einem festen Wetteinsatz etwa $10^{14}$ Entscheidungspunkte existieren, wächst die Anzahl durch den Wegfall des Limits auf etwa $10^{160}$ Entscheidungspunkte und ist somit vergleichbar mit Go ($10^{170}$ Entscheidungspunkte) [Moravčík et al., 2017, S. 509].

### 4.3.1 Erfolge

Um DeepStacks Qualität zu messen, wurde 2016 eine Studie durchgeführt. In der Studie ist DeepStack gegen 33 Pokerspieler angetreten, wobei den Pokerspielern ein monetärer Anreiz in Form eines Preisgelds für die top platzierten Spieler gegeben wurde. Ursprüngliches Ziel war es, gegen jeden der Spieler 3.000 Hände zu spielen, jedoch haben nur 11 Spieler dieses Ziel erreicht. Insgesamt wurden 44.852 Spielrunden gespielt.

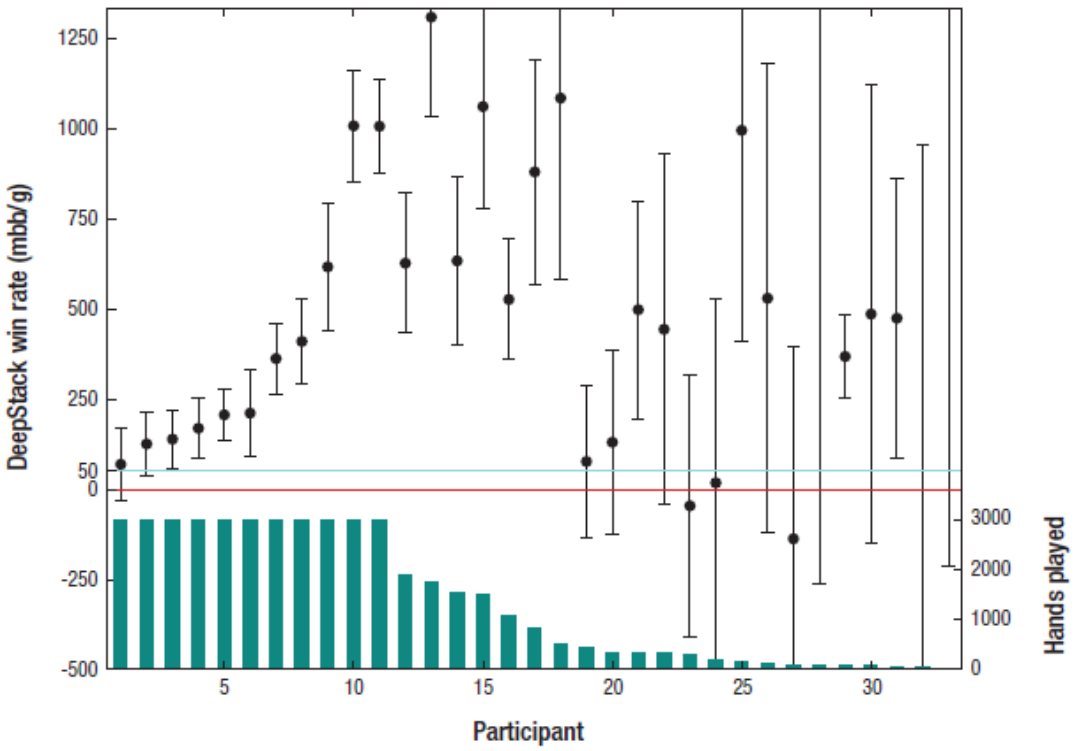


Abbildung 4.3 Gewinnquote von Deepstack gegen Pokerspieler und Anzahl der gespielten Hände [Moravčík et al., 2017, S. 10].

Im Durchschnitt konnte DeepStack in dieser Studie durchschnittlich 492mbb/g gewinnen. [Moravčík et al., 2017, S. 10]. Wenn nur die gespielten Hände der Spieler, die ihre 3.000 Runden beendet haben, betrachtet werden, sinkt das Ergebnis auf etwa 394mbb/g. Im Spiel gegen den besten menschlichen Spieler konnte ein Ergebnis von etwa 70mbb/g erzielt werden.

Insgesamt konnte DeepStack somit einen großen Erfolg erzielen und die professionellen Pokerspieler besiegen, wobei die Ergebnisse statistisch signifikant sind.

Das Vorgehen dabei ist nicht nur auf Poker begrenzt, sondern stellt im Allgemeinen einen Erfolg für die KI Entwicklung dar:

> The implications go beyond being a milestone for artificial intelligence. DeepStack represents a paradigm shift in approximating solutions to large, sequential imperfect information games. [Moravčík et al., 2017, S. 11]

### 4.3.2 Aufbau

Bei DeepStack handelt es sich um einen allgemeinen Algorithmus, der nicht allein auf Poker beschränkt ist, sondern auf alle sequentiellen Spiele mit unvollständigen Informationen angewendet werden kann [Moravčík et al., 2017, S. 3]. DeepStack verfolgt das Ziel, eine Strategie zu finden und zu spielen, die eine möglichst geringe Exploitability innehat, was einer Annäherung an das Nash-Gleichgewicht entspricht. Trotz desselben Ziels ist der Aufbau grundverschieden zu dem anderer Poker Bots. Während viele der bisherigen Ansätze auf Counterfactual Regret Minimization aufbauen, nutzt DeepStack eine Kombination aus verschiedenen Methoden. Ein weiterer Unterschied, der hieraus resultiert, ist, dass die Strategie nicht im Vorfeld komplett ermittelt wird, sondern erst dann, wenn die Situation tatsächlich auftritt.

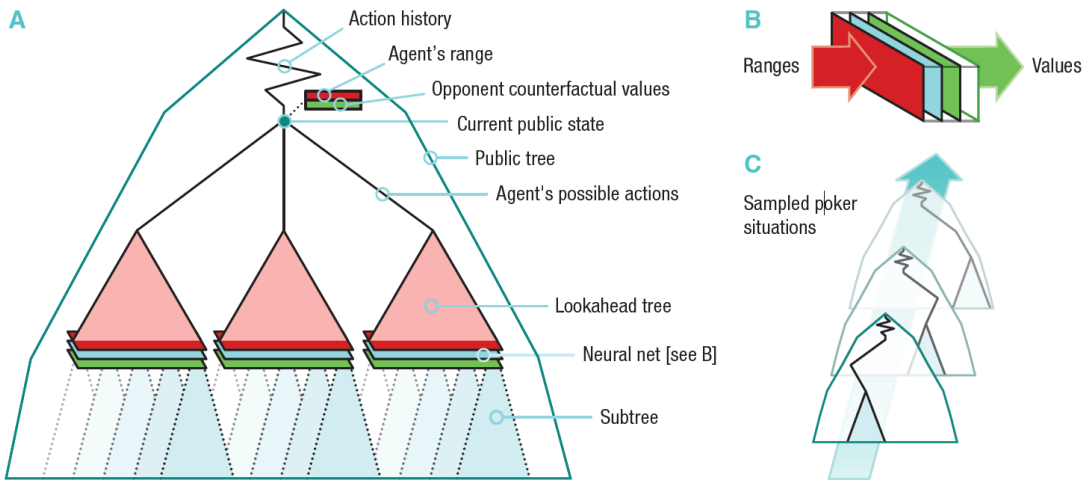


Abbildung 4.4 Aufbau von DeepStack [Moravčík et al., 2017, S. 7]

Kern der Strategiefindung ist das *continual re-solving* in Verbindung mit einem tiefen neuronalen Netz. Die Strategie wird dabei nicht für das komplette Spiel definiert, sondern es wird immer, wenn eine Aktion fällig wird, eine neue Strategie berechnet. Hierbei werden zwei Aspekte über die Strategien hinweg gespeichert, zum einen die *Range* der eigenen Hand, zum anderen ein Vektor mit den *counterfactual values* des Gegners. Die *Range* stellt dabei eine Wahrscheinlichkeitsverteilung der möglichen Hände des Spielers dar, die in dem aktuellen Zustand erreicht werden könnten. Die Werte, die über die gegnerische Hand gespeichert werden, beschreiben Moravčík et al. als:

> Each value in the opponent's vector is a counterfactual value, a conditional "what-if" value that gives the expected value if the opponent reaches the public state with a particular hand. [Moravčík et al., 2017, S. 5]

Nach jeder Aktion wird demnach eine neue Strategie gebildet. Bevor dies jedoch geschieht, werden sowohl die Werte der eigenen Range aktualisiert, als auch die Werte des Gegners. Hierbei spielt es keine Rolle, ob die Aktion durch den Spieler geschieht oder durch das Aufdecken einer Karte.

Um die eigentliche Strategie zu bilden wird versucht den Teil des Baums nach einer gewählten Aktion zu lösen (*lookahead tree*). Da es jedoch nicht möglich ist den kompletten Baum zu durchsuchen, wird er nur bis zu einer bestimmten Tiefe durchsucht. Wenn diese Tiefe erreicht wurde, wird der Wert des Knotens nicht wie bei anderen Verfahren durch eine Heuristik oder einen vorgegebenen Wert bestimmt, sondern durch ein tiefes neuronales Netz. Dieses neuronale Netz erhält die Größe des Pots im Verhältnis zu den vorhandenen Chips und Informationen zu den Ranges des Spielers.

Genau genommen werden dabei verschiedene neuronale Netze verwendet, eines, welches zu Beginn des Spiels genutzt wird (Flop Network) und ein weiteres, welches ab der 4. Community Karte verwendet wird (Turn network). Bei den Netzen handelt es sich um mehrschichtige feedforward Netze mit sieben verborgenen Schichten, die jeweils über 500 vollständig verbundene Neuronen verfügen. Für das Training der Netze wurden 10 Mio. zufällige generierte Pokerspiele gespielt.

## 4.4 Pluribus

Die meisten Erfolge, die im Pokerspiel errungen werden konnten, gab es lange Zeit in einem Zwei-Spieler-Heads-Up Spiel, da dies die Komplexität des Spiels deutlich reduziert. Pluribus ist jedoch der erste Poker Bot dem es gelungen ist, im Spiel mit mehr als zwei Spielern erfolgreich zu sein [Heaven, 2019, S. 307]. Hierbei trat der Bot in einem Sechs-Spieler No-Limit Texas Holdem Spiel gegen professionelle Spieler an und konnte gegen diese gewinnen.

Noam Brown, einer der Forscher hinter Pluribus sagt über diesen Erfolg:

> *This was the last remaining challenge in poker. [Heaven, 2019, S. 308]*

Ziel der bisher erfolgreichen Pokerbots war es, eine Strategie zu finden, die möglichst dem Nash-Gleichgewicht entspricht, da auf diesem Weg die eigene *Exploitability* minimiert wird. Diese Tatsache beruht darauf, dass es sich hierbei um ein zwei-Spieler Nullsummenspiel handelt. In Spielen mit mehr als drei Spielern existiert, zumindest theoretisch, ebenfalls ein Gleichgewicht und dementsprechend Gleichgewichtsstrategien. Zum aktuellen Zeitpunkt ist jedoch kein Vorgehen bekannt, welches effizient in der Lage ist, ein solches Gleichgewicht zu bestimmen. Auch Annäherungen an ein Gleichgewicht sind aufwendig, selbst in Spielen mit wenigen möglichen Strategien. Und selbst wenn eine Gleichgewichtsstrategie bekannt ist, bedeutet dies in einem Spiel mit mehr als zwei Spielern nicht, dass es optimal sein muss, dieser Strategie zu folgen. [Brown und Sandholm, 2019, S. 887] Aus diesem Grund hat Pluribus nicht das Ziel eine Gleichgewichtsstrategie zu ermitteln, sondern das Spiel gegen menschliche (professionelle) Spieler zu optimieren, um auf diesem Weg empirisch die Qualität des Bots zu demonstrieren.

### 4.4.1 Erfolge

Pluribus wurde in zwei Formaten gegen professionelle Pokerspieler getestet. Bei den Pokerspielern handelt es sich um renommierte Spieler, von denen jeder bereits in Turnieren mehr als 1 Mio. Dollar gewonnen hat. Beim ersten Format handelt es sich um ein Spiel mit fünf menschlichen Spielern und einer Kopie von Pluribus, im zweiten Format traten fünf Kopien von Plurbius und ein menschlicher Spieler gegeneinander an.

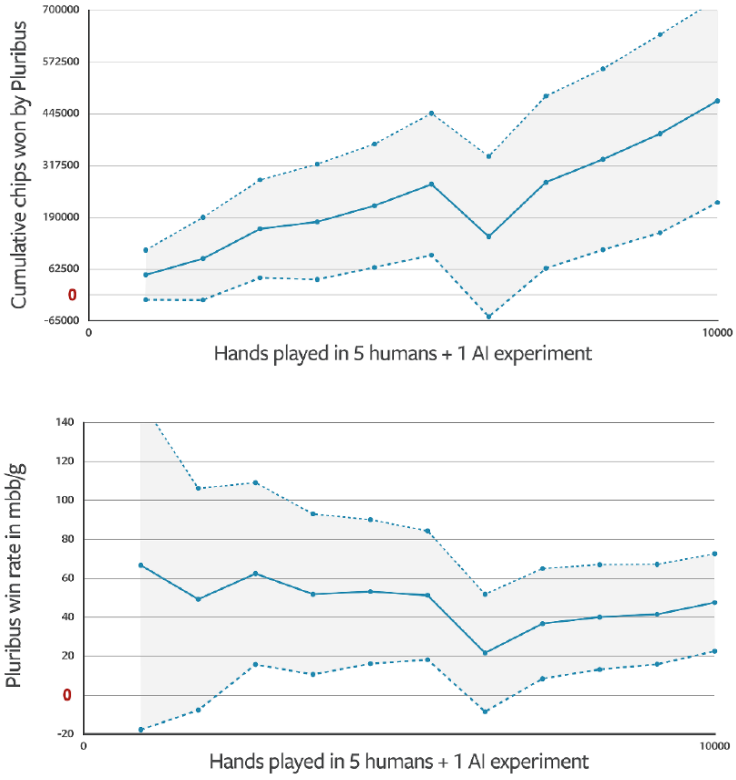


Abbildung 4.5 Kumulative Gewinne und durchschnittlicher Gewinn in MBB/G von Pluribus im Spiel gegen 5 menschliche Spieler [Brown und Sandholm, 2019]

Im Spiel mit fünf menschlichen Spielern wurden insgesamt 10.000 Hände gespielt, wobei Pluribus im Schnitt einen Gewinn von 48mbb/Spiel mit einem Standardfehler von etwa 25 mbb/Spiel erzielte. Verglichen mit professionellen Pokerspielern stellt dies eine sehr gute Gewinnquote dar, insbesondere im Spiel gegen starke Gegner [Brown und Sandholm, 2019, S. 889]. Im Spiel mit nur einem menschlichen Spieler und fünf Kopien von Pluribus wurden ebenfalls 10.000 Hände gespielt. Den Spielern war hierbei nicht die Identität der Gegner bekannt und somit war es Pluribus nicht möglich, sich gegen den menschlichen Spieler zu verbünden [Brown und Sandholm, 2019, S. 889]. In diesem Spiel gelang es Pluribus einen Gewinn von durchschnittlich 32mbb/Spiel mit einem Standardfehler von 15 mbb/g zu erzielen.

Insgesamt konnte Pluribus somit in beiden Szenarien signifikante Gewinne erzielen, was wiederum darauf hindeutet, dass der Bot menschlichen Spielern überlegen ist.

### 4.4.2 Aufbau

Pluribus ist im Kern ein Bot, der im Spiel gegen sich selbst das Spiel erlernt, also nicht auf vorherige Daten über Spieler oder Strategien angewiesen ist.

Hierbei wird auf verschiedene Varianten des Counterfactual Regret zurückgegriffen, wobei zwischen zwei unterschiedlichen Phasen unterschieden wird. Zu Beginn

wurde eine *Blueprint*-Strategie entwickelt, welche Pluribus als Standard-Strategie dient. Um die Komplexität zu reduzieren, wurden dabei verschiedene Abstraktionen vorgenommen. Hierdurch wird die Komplexität reduziert und eine effizientere Strategiefindung ermöglicht. Im ersten Schritt wurden die möglichen Aktionen in verschiedene Segmente unterteilt. Im No-limit Poker ist es grundsätzlich möglich beliebige Einsätze, die größer als der Big Blind sind, zu machen. Hierdurch entsteht eine große Menge an Aktionen, wobei es in der Praxis nur einen geringfügigen Unterschied zwischen einem Einsatz in Höhe von 100$ und 101$ gibt. Ein ähnliches Vorgehen wird bei Abstraktion der Informationen verfolgt. Hierbei werden ähnliche Hände ebenfalls zusammengefasst, da es auch hier praktisch nur geringfügige Unterschiede in einer Straße, die auf eine 10 endet, und einer Straße, die auf einer 9 endet, gibt. Durch diese Vereinfachungen wird zwar auf der einen Seite die Komplexität reduziert und damit die Strategiefindung vereinfacht, jedoch werden gleichzeitig wichtige Informationen eingebüßt. Aus diesem Grund werden diese Abstraktionen nur zur Abschätzung zukünftiger Wettrunden und während des initialen Spiels mit sich selbst verwendet, nicht im eigentlichen Spiel. [Brown und Sandholm, 2019, S. 888] Dieser Blueprint-Strategie wird jedoch nur während der ersten Wettrunde gefolgt. In den darauffolgenden Wettrunden wird eine erneute Suche nach einer optimalen Strategie durchgeführt, wobei je nach Fortschritt des Spiels, beziehungsweise der Größe des Spielbaums, entweder eine *Monte Carlo Linear CFR* oder eine vektorbasierte Form der *Linear CFR* genutzt wird [Brown und Sandholm, 2019, S. 890]. Für die Lösung des Spielbaums werden dabei vier verschiedene mögliche Strategien der Gegner angenommen, aus der eine zufällig ausgewählt wird. Bei den Strategien handelt es sich zum einen um die reine Blueprint Strategie, und zum anderen um die Blueprint-Strategie mit einem Bias zu einer bestimmen Aktion (BET, RAISE, FOLD).

Trotz der aufwendigen Berechnung und des komplexen Spiels benötigt Pluribus vergleichsweise wenig Rechenleistung. Zur Ermittlung der Blueprint Strategie wurden weniger als 1.5 Core Years benötigt (8 Tage auf einem 64-Core Rechner) mit weniger als 512GB Arbeitsspeicher. Während der eigentlichen Laufzeit benötigt Pluribus nur einen Rechner mit etwa 128GB Arbeitsspeicher.

# 5 Kuhn Poker

Dieses Kapitel stellt die Ergebnisse der Betrachtung des Spiels *Kuhn Poker* vor. Bei Kuhn Poker handelt es sich um eine vereinfachte Variante des Pokerspiels, die lediglich mit drei Spielkarten gespielt wird [Kuhn, 1951, S. 97–103], die in Abschnitt 2.2 beschrieben wird. Trotz der verringerten Komplexität enthält das Spiel wichtige Aspekte des allgemeinen Pokerspiels: Es existieren versteckte Informationen, es werden rundenweise Einsätze getätigt und der Gewinner wird durch die beste Hand bestimmt. Dieser Abschnitt prüft, ob die im GBG-Framework implementierten Agenten in der Lage sind, das Spiel zu lernen, und falls ja, zu prüfen, wie gut sie das Spiel beherrschen. Als Kriterium für das erfolgreiche Spielen wird der *durchschnittliche Gewinn pro Spielrunde* auf der jeweiligen Spielposition gewählt. Die untere Grenze des Spielens wird durch den zufällig agierenden Agenten definiert. Sobald ein Agent besser als dieser ist, hat er das Spiel zumindest zu einem gewissen Grad erlernen können. Als obere Grenze wird die optimale Strategie gewählt. In den folgenden Abschnitten wird zuerst das Spiel gegen die optimale Strategie analysiert und anschließend werden die Ergebnisse der einzelnen Agenten vorgestellt.

## 5.1 Spiel gegen die optimale Strategie

Im folgenden Abschnitt wird das Spiel gegen die optimale Strategie näher betrachtet um herauszufinden, welche Züge sinnvoll und somit erstrebenswert für die Agenten zu lernen sind. Zuerst wird das Spiel als Startspieler betrachtet, anschließend als Folgespieler und abschließend wird die gefundene Strategie vorgestellt und ein Fazit gezogen.

### 5.1.1 Startspieler

#### Bube

Falls der Spieler mit einem Buben CHECK wählt, ist der Gewinn -1, sollte der Gegner ebenfalls CHECK wählen, so gewinnt der Gegner, sollte der Gegner BET wählen, muss der Spieler FOLD wählen, um weitere Verluste zu vermeiden. Falls der Spieler mit einem Buben BET wählt, wird der Gegner bei einem König in jedem Fall CALL wählen, bei einer Dame wird er mit einer Wahrscheinlichkeit von $\frac{1}{3}$ CALL

wählen und ansonsten FOLD. Somit ist:

$$Gewinn(König, BET) = -2$$

$$Gewinn(Dame, BET) = \frac{1}{3} * (-2) + \frac{2}{3} * 1$$

$$Gewinn(BET) = \frac{Gewinn(König,BET)+Gewinn(Dame,BET)}{2} = -1$$

Für den Buben macht es demnach keinen Unterschied, ob CHECK oder BET gewählt wird, wenn gegen die optimale Strategie gespielt wird, da der erwartete Gewinn in beiden Fällen -1 ist.

### Dame

Falls der Spieler mit einer Dame CHECK wählt und der Gegner einen König hat, wird dieser in jedem Fall BET wählen, sollte er einen Buben haben, wird er mit einer Wahrscheinlichkeit von $\frac{2}{3}$ CHECK wählen, ansonsten BET. Falls der Spieler auf ein BET mit einem CALL reagiert, ergibt sich:

$$Gewinn(Bube, CHECK) = \frac{2}{3} * 1 + \frac{1}{3} * 2$$

$$Gewinn(König, CHECK) = -2$$

$$Gewinn(CHECK) = \frac{Gewinn(Bube,CHECK)+Gewinn(König,CHECK)}{2} = -\frac{1}{3}$$

Falls der Spieler auf ein BET mit einem FOLD reagiert ergibt sich:

$$Gewinn(Bube, CHECK) = \frac{2}{3} * 1 + \frac{1}{3} * -1$$

$$Gewinn(König, CHECK) = -1$$

$$Gewinn(CHECK) = \frac{Gewinn(Bube,CHECK)+Gewinn(König,CHECK)}{2} = -\frac{1}{3}$$

Es macht demnach keinen Unterschied, ob mit CALL oder FOLD reagiert wird, da der erwartete Gewinn in beiden Fällen $-\frac{1}{3}$ ist. Falls der Spieler mit einer Dame BET wählt und der Gegner einen König hat, wird dieser in jedem Fall CALL wählen, sollte er einen Buben haben in jedem Fall FOLD. Somit ergibt sich der erwartete Gewinn von:

$$Gewinn(Bube, BET) = 1$$

$$Gewinn(König, BET) = -2$$

$$Gewinn(BET) = \frac{Gewinn(Bube,BET)+Gewinn(König,BET)}{2} = -0,5$$

Mit der Dame ist demnach CHECK zu bevorzugen, da hier nur ein Verlust von $\frac{1}{3}$ erwartet wird.

#### König

Falls der Spieler mit einem König CHECK wählt und der Gegner eine Dame hat, wird dieser auf jeden Fall CHECK wählen, bei einem Buben mit einer Wahrscheinlichkeit von $\frac{1}{3}$ BET und ansonsten FOLD. Somit ergibt sich als erwarteter Gewinn für ein CHECK:

$$Gewinn(Bube, CHECK) = \frac{1}{3} * 2 + \frac{2}{3} * 1 = \frac{4}{3}$$

$$Gewinn(Dame, CHECK) = 1$$

$$Gewinn(CHECK) = \frac{Gewinn(Bube,CHECK)+Gewinn(Dame,CHECK)}{2} = \frac{7}{6}$$

Falls der Spieler mit einem König BET wählt und der Gegner einen Buben hat, wird dieser in jedem Fall FOLD wählen. Sollte der Gegner eine Dame haben, wird er mit einer Wahrscheinlichkeit von $\frac{1}{3}$ CALL wählen, ansonsten FOLD. Somit ergibt sich der erwartete Gewinn:

$$Gewinn(Bube, BET) = 1$$

$$Gewinn(Dame, BET) = \frac{1}{3} * 2 + \frac{2}{3} * 1 = \frac{4}{3}$$

$$Gewinn(BET) = \frac{Gewinn(Bube,BET)+Gewinn(Dame,BET)}{2} = \frac{7}{6}$$

Somit macht es auch für den König keinen Unterschied, ob CHECK oder BET gewählt wird, wenn gegen die optimale Strategie gespielt wird, da der erwartete Gewinn in beiden Fällen $\frac{7}{6}$ ist.

### 5.1.2 Folgespieler

Für den Folgespieler lassen sich viele Entscheidungen auch ohne Berechnung ermitteln:

- Ein Bube sollte auf ein BET mit FOLD reagieren, da er in jedem Fall verliert.
- Eine Dame sollte auf ein CHECK mit CHECK reagieren, gegen einen Buben erhält man denselben Gewinn, gegen einen König verliert man nur das Minimum.
- Ein König sollte auf ein BET mit CALL reagieren, da er in jedem Fall gewinnt.
- Ein König sollte auf ein CHECK mit BET reagieren, da er, selbst wenn der Gegner FOLD wählt, den selben Gewinn erhält.

Somit müssen nur die Reaktionen auf ein BET mit einer Dame und ein CHECK mit einem Buben ermittelt werden.

#### Reaktion von Bube auf CHECK

Sollte ein Bube auf ein CHECK mit einem CHECK reagieren, so ist der erwartete Gewinn -1. Wird mit einem BET reagiert und der Gegner hat einen König, wird er in jedem Fall mit einem CALL reagieren, sollte er eine Dame haben wird er mit der Wahrscheinlichkeit $\frac{1}{3} + \alpha$ ebenfalls mit CALL reagieren und ansonsten FOLD wählen. Somit ergibt sich:

$$Gewinn(Dame, CHECK, BET) = (\tfrac{1}{3} + \alpha) * -2 + (1 - (\tfrac{1}{3} + \alpha)) * 1$$

$$Gewinn(Dame, CHECK, BET) = -3 * \alpha$$

$$Gewinn(König, CHECK, BET) = -2$$

Falls ein CHECK vom Gegner gewählt wird, tut er dies in jedem Fall mit einer Dame und in $1 - 3 * \alpha$ der Fälle mit einem König. Daraus ergibt sich:

$$Gewinn(CHECK, BET) = -3 * \alpha * \frac{1}{1+1-3\alpha} + (-2)\frac{1-3\alpha}{1+1-3\alpha}$$

$$Gewinn(CHECK, BET) = \frac{-3*\alpha}{2-3\alpha} - \frac{2+6\alpha}{2-3\alpha} = \frac{3*\alpha-2}{2-3\alpha} = -1$$

Somit ist es für den Folgespieler egal, ob er mit einem Buben auf ein CHECK mit einem BET oder einem CHECK reagiert, da in beiden Fällen der erwartete Gewinn gegen die optimale Strategie -1 ist.

#### Reaktion von Dame auf BET

Sollte eine Dame auf ein BET mit einem FOLD reagieren, so ist der erwartete Gewinn -1. Wird mit einem CALL reagiert, setzt sich der Gewinn zusammen aus $+2$ im Spiel gegen einen Buben und -2 im Spiel gegen einen König. Der Gegner wählt BET mit einem Buben mit einer Wahrscheinlichkeit von $\alpha$ und mit einem König mit einer Wahrscheinlichkeit von $3\alpha$. Somit ergibt sich, dass der Gegner dreimal häufiger mit einem König BET wählt, als mit einem Buben. Daraus ergibt sich, dass beispielsweise von vier Spielen drei mit -2 verloren werden und eins mit 2 gewonnen wird. Hieraus folgt, dass der durchschnittlicher Gewinn -1 beträgt. Somit ist es für den Folgespieler ebenfalls unerheblich, ob er mit einer Dame auf ein BET mit einem CALL oder einem CHECK reagiert, da in beiden Fällen der erwartete Gewinn gegen die optimale Strategie -1 ist.

### 5.1.3 Fazit

Bei der optimalen Strategie handelt es sich um ein Nash-Gleichgewicht. Dieses Gleichgewicht zeichnet sich dadurch aus, dass es für beide Spieler nicht sinnvoll ist von der Strategie abzuweichen, solange der Gegner nicht ebenfalls Änderungen

vornimmt. Somit zeichnet sich die Strategie dadurch aus, dass in den Zügen, in denen es keine offensichtlich richtige Entscheidung gibt, die Auswahl für den Gewinn auf lange Sicht irrelevant ist, da beide Entscheidungen zum selben Ergebnis führen.

Tabelle 5.1 Strategie gegen die optimale Strategie für Kuhn Poker

(a) Als Startspieler

| **Hand** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | BET\|CHECK | FOLD |
| Dame | **CHECK** | CALL\|FOLD |
| König | BET\|CHECK | CALL |

(b) Als Folgespieler

| | **Aktion Gegner** | |
|---|---|---|
| **Hand** | **CHECK** | **BET** |
| Bube | BET\|CHECK | FOLD |
| Dame | CHECK | FOLD\|CALL |
| König | BET | CALL |

Für den Startspieler mit einer Dame ist es wichtig, mit einem CHECK zu eröffnen und offensichtliche Fehler, wie ein CALL mit einem Buben oder einem FOLD mit einem König zu vermeiden. Viele der anderen Entscheidungen haben keinen Einfluss und es ist beispielsweise egal, ob mit einem Buben BET oder CHECK gewählt wird. Als Folgespieler gibt es jedoch mehr richtige Entscheidungen, die für den Menschen zu erkennen sind und durch die Agenten gelernt werden können.

## 5.2 Random Agent

Der Random Agent agiert rein zufällig, indem er aus allen verfügbaren Aktionen in einem Spielzustand eine Aktion zufällig auswählt. Dies bedeutet, dass, sollten einem Agenten vier Aktionen zur Verfügung stehen, diese jeweils mit einer Wahrscheinlichkeit von 25% ausgewählt wird.

### 5.2.1 Erwartete Ergebnisse

Sowohl die Strategien des Random Agent, als auch die jeweiligen Strategien des Kuhn Agenten, sind in den einzelnen Spielpositionen bekannt. Die Aktionen des Random Agent werden sich bei einer ausreichend großen Menge gleichmäßig verteilen. Aus diesem Grund ist es möglich, den Spielverlauf zu modellieren und die erwarteten Ergebnisse im Vorfeld zu berechnen.

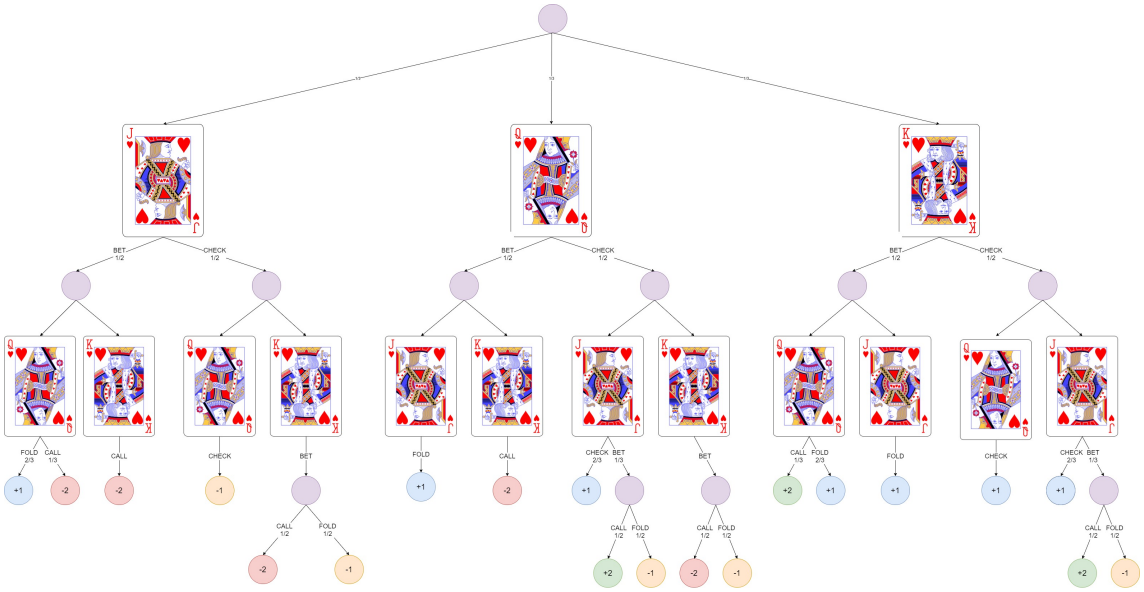

Abbildung 5.1 Spiel Random Agent gegen optimale Strategie in Kuhn Poker

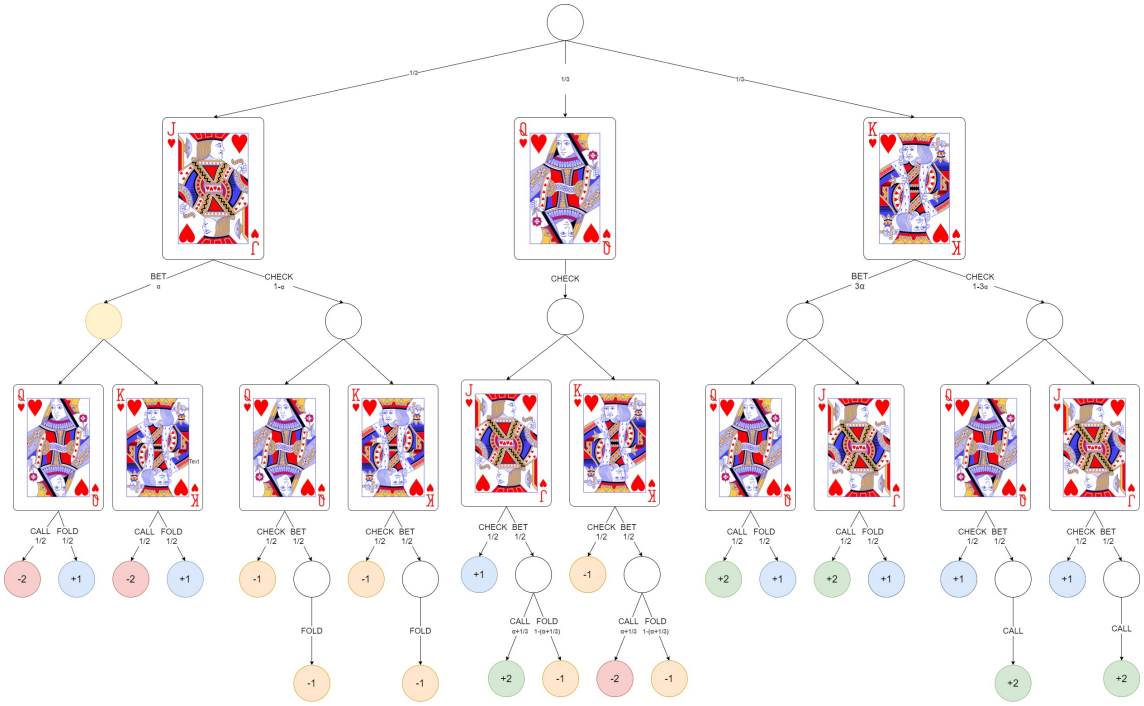

Abbildung 5.2 Spiel optimale Strategie gegen Random Agent in Kuhn Poker

Hierbei zeigt sich, dass der erwartete Gewinn für einen Agent mit optimaler Strategie als erster Spieler vom Parameter $\alpha$ abhängig ist, wobei der erwartete Gewinn für $\alpha = \frac{1}{3}$ mit etwa $\frac{1}{6}$ Chips pro Spielrunde auf lange Sicht am höchsten ist. Der Grund dafür ist, dass der Parameter $\alpha$ bestimmt, ob der Agent mit einem Buben bzw. König als Handkarte einen Einsatz macht oder lediglich checkt. Im Fall des Königs macht dies keinen Unterschied, der erwartete Gewinn ist für beide Aktionen 1,5 Chips. Im Fall des Buben beträgt der erwartete Gewinn bei einem Wetteinsatz -0,5 und ist somit dem Check, der einen sicheren Gewinn von -1 darstellt, vorzuziehen.

Dies liegt daran, dass der zufällig agierende Agent auch mit einem garantierten Sieg mit einem König mit einer Wahrscheinlichkeit von 50% passen wird.
Der erwartete Gewinn für einen zufällig agierenden Agenten als erster Spieler ist auf lange Sicht ein Verlust in Höhe von ungefähr $\frac{1}{6}$ Chips pro Spielrunde.

### 5.2.2 Realisierte Ergebnisse

Im nächsten Schritt wurde ein Experiment durchgeführt, bei dem je 1.000.000 Spielrunden pro Position gespielt wurden mit $\alpha = \frac{1}{5}$. Bei diesem Testergebnis ergaben sich die Werte in Tabelle 5.3

Tabelle 5.3 Erster realisierter Gewinn im Spiel Random Agent gegen die optimale Strategie in Kuhn Poker

| **Startspieler** | **Erwarteter Gewinn für Startspieler** | **Realisierter Gewinn für Startspieler** |
|---|---|---|
| Optimaler Agent | 0,122 | 0,159 |
| Random Agent | -0,167 | -0,108 |

Die hierbei deutlich werdenden Abweichungen sind sehr groß und sind bei der Menge gespielter Runden nicht durch Ausreißer zu erklären. Dies deutet darauf hin, dass es entweder einen Fehler in der Berechnung des erwarteten Gewinns gibt, die optimale Strategie nicht korrekt umgesetzt wurde oder ein Fehler im Programmcode existiert.

Tabelle 5.4 Optimale Strategie für Kuhnpoker als Startspieler mit $\alpha = \frac{1}{5}$

| **Handkarte** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | BET zu 20,0%<br>Ansonsten CHECK | Falls Gegner BET dann FOLD |
| Dame | CHECK | CALL zu 53,3%<br>Ansonsten FOLD |
| König | BET zu 60,0%<br>Ansonsten CHECK | Falls Gegner BET dann CALL. |

Bei der Auswertung der Ergebnisse in Tabelle 5.5 wird deutlich, dass der optimalen Strategie gefolgt wird und die korrekten Aktionen bei gegebenen Handkarten gewählt werden. Allerdings fällt bei der Verteilung der Handkarten auf, dass diese nicht gleichmäßig verteilt sind, wie in Tabelle 5.6 deutlich wird.

Tabelle 5.5 Übersicht der Aktionen des Agenten, der der optimalen Strategie folgt

| **Handkarte** | **Aktion** | **Anzahl** | **P(Aktion)** |
|---|---|---|---|
| Bube | CHECK<br>BET | 2626864<br>657016 | 79,993%<br>20,007% |
| Dame | CHECK | 3159896 | 100% |
| König | CHECK<br>BET | 1420908<br>2135316 | 39,956%<br>60,044% |

Tabelle 5.6 Fehlerhafte Verteilung der Handkarten

| **Handkarte** | **Anzahl** | **P(Handkarte)** |
|---|---|---|
| Bube | 3283880 | 32,839% |
| Dame | 3159896 | 31,599% |
| König | 3556224 | 35,562% |

Bei einer statistischen Betrachtung der Ergebnisse mit $n = 1.000.000$ und $p(Bube) = \frac{1}{3}$ ergibt sich:

$$E(Bube) = p(Bube) * n = 333.333,333$$

$$\sigma = \sqrt{V(Bube)} = \sqrt{n * p(Bube) * (1 - p(Bube))} = 471,405$$

Tabelle 5.7 Erwartete Werte einer normalverteilten Zufallsvariablen für Buben innerhalb des Streuintervalls

| **Sigma** | **Prozent inner-halb** | **Untere Grenze** | **Obere Grenze** |
|---|---|---|---|
| 1 $\sigma$ | 68,268% | 33.2861,928 | 333.804,737 |
| 3 $\sigma$ | 99,730% | 331.919,119 | 334.747,546 |
| 4 $\sigma$ | 99,994% | 331.447,715 | 335.218,951 |
| 6 $\sigma$ | 99,999% | 330.504,906 | 336.161,760 |

Grundsätzlich wäre somit zu erwarten, dass 99,999% Werte zwischen 330.504 und 336.161 liegen sollten. Dies ist nicht der Fall. Hierdurch werden die Ergebnisse verzerrt und es erklärt den höheren realisierten Gewinn für den ersten Spieler durch die überproportionale Menge an erhaltenen Königen. Bei näherer Betrachtung der Spielverläufe wird deutlich, dass sich die Verteilung der Handkarten auffällig häufig wiederholt.

Tabelle 5.8 Auszug Spielverlauf vom Random Agent gegen optimale Strategie

| **Spieler 1** | **Karte: Spieler1** | **Karte: Spieler 2** | **Spielzüge** | **Reward** |
|---|---|---|---|---|
| Kuhn | Bube | Dame | CHECK-BET-FOLD | -1.0 |
| Kuhn | Bube | König | CHECK-BET-FOLD | -1.0 |
| Kuhn | Bube | König | CHECK-BET-FOLD | -1.0 |
| Kuhn | Dame | König | CHECK-BET-CALL | -2.0 |
| Kuhn | Bube | Dame | CHECK-BET-FOLD | -1.0 |
| Kuhn | Bube | Dame | CHECK-BET-FOLD | -1.0 |
| Kuhn | Dame | König | CHECK-BET-CALL | -2.0 |
| Kuhn | Dame | König | CHECK-BET-CALL | -2.0 |
| Kuhn | Dame | König | CHECK-BET-FOLD | -1.0 |
| Kuhn | Dame | Bube | CHECK-BET-FOLD | -1.0 |
| Kuhn | König | Dame | BET-CALL | 2.0 |
| Kuhn | König | Dame | CHECK-BET-CALL | 2.0 |
| Kuhn | Dame | König | CHECK-BET-CALL | -2.0 |
| Kuhn | Dame | König | CHECK-BET-FOLD | -1.0 |
| Kuhn | Dame | König | CHECK-BET-CALL | -2.0 |

Hieraus lässt sich vermuten, dass die Generierung zufälliger Handkarten zu Beginn einer Spielrunde nicht komplett zufällig ist. Für die zufällige Verteilung der Handkarten wurde java.util.Random als Zufallsgenerator verwendet. Diese Klasse wird, sollte kein anderer Seed definiert werden, mit der aktuellen Systemzeit in Millisekunden als Seed initialisiert. Dies hat zur Folge, dass für zwei Spiele, die in derselben Millisekunde initialisiert werden, die gleichen Zufallszahlen-Abfolgen genutzt und damit die gleichen Handkarten gewählt werden. Insbesondere beim Spiel des zufälligen Agenten werden Aktionen sehr schnell getroffen, wodurch die Spielrunden nur wenig Zeit benötigen und neue Spielrunden schnell beginnen. Dies wiederum hat zur Folge, dass das Risiko, dieselbe Folge an Zufallszahlen zu nutzen, relativ hoch ist. Gelöst werden konnte das Problem, indem auf eine andere Java Klasse (ThreadLocalRandom) zurückgegriffen wurde, die eine Folge von Zufallszahlen isoliert für den aktuellen Thread erzeugt. Hierdurch konnten deutlich bessere Verteilungen der Handkarten erzielt werden, wie Tabelle 5.9 zeigt.

Tabelle 5.9 Korrekte Verteilung der Handkarten

| **Handkarte** | **Spieler1** | **Spieler 2** |
|---|---|---|
| Bube | 33,3496% | 33,2502% |
| Dame | 33,3338% | 33,4622% |
| König | 33,3166% | 33,2876% |

### 5.2.3 Korrigierte Ergebnisse

In einem weiteren Experiment nach Anpassung der Zufallszahlengenerierung mit je 1.000.000 gespielten Runden pro Position und $\alpha = \frac{1}{5}$ ergaben sich ungefähr die erwarteten Werte, wie Tabelle 5.10 zeigt. Der zufällig agierende Agent verliert über einen langen Zeitraum 0,168 Chips pro Spielrunde, wenn er beginnt, und verliert 0,121 Chips, wenn er als zweites am Zug ist.

Tabelle 5.10 Vergleich erwarteter und realisierter Gewinn im Spiel Random Agent gegen die optimale Strategie in Kuhn Poker

| **Startspieler** | **Erwarteter Gewinn für Startspieler** | **Realisierter Gewinn für Startspieler** |
|---|---|---|
| Optimaler Agent | 0,122 | 0,121 |
| Random Agent | -0,167 | -0,168 |

### 5.2.4 Bewertung

Die allgemeine optimale Strategie erzielt nicht die bestmöglichen Ergebnisse gegen einen zufällig agierenden Agenten. Die größten Gewinne gegen den zufällig agierenden Agenten erspielt der MC-N Agent, da dieser bei der Ermittlung der besten Aktion von einem zufällig agierenden Agenten ausgeht. Hierbei zeigt sich, dass eine sehr aggressive Strategie die beste ist, bei der fast immer gesetzt werden sollte, außer man gerät in die Situation mit einem Buben mitgehen zu müssen. In diesem Fall sollte gepasst werden. Mit dieser Strategie ist es möglich, als Startspieler 0,5 Chips pro Spielrunde und als zweiter Spieler 0,417 Chips pro Spielrunde zu gewinnen. Auffällig ist dabei, dass der beginnende Spieler in einer besseren Situation ist. Dies ist dadurch zu erklären, dass der sonst hilfreiche Informationsvorsprung des zweiten Spielers, gegen einen zufällig agierenden Spieler wertlos ist und man als erster Spieler dem Gegner mehr Raum für Fehler bietet.

## 5.3 MC-N

Der MC-N Agent wählt seine Aktionen basierend auf einer Monte-Carlo-Simulation von einem Spiel gegen einen zufällig agierenden Agenten. Bei einer hinreichend großen Menge an betrachteten Spielen lässt sich also die, vom Agenten gefundene Strategie im Vorfeld ermitteln. Daher wird zuerst die erwartete Strategie ermittelt, die im Anschluss mit den realisierten Ergebnissen verglichen wird. Abschließend wird ein Fazit zu dem Agenten gezogen.

### 5.3.1 Erwartete Ergebnisse

Der MC-N Agent simuliert für die Findung seines nächsten Zugs den weiteren Spielverlauf der aktuellen Spielsituation zwischen zwei zufällig agierenden Agenten, um damit den durchschnittlichen Gewinn für die gewählte Aktion zu ermitteln. Wenn dies für alle möglichen Spielzustände betrachtet wird, lässt sich damit die Strategie, welche vom MC-Agenten verfolgt wird, ermitteln. Abbildung 5.3 zeigt beispielhaft alle möglichen Spielverläufe für MC-N als Startspieler mit einem Buben als Handkarte. Die durchschnittlichen Gewinne je Aktion werden dabei in den einzelnen Knoten dargestellt.

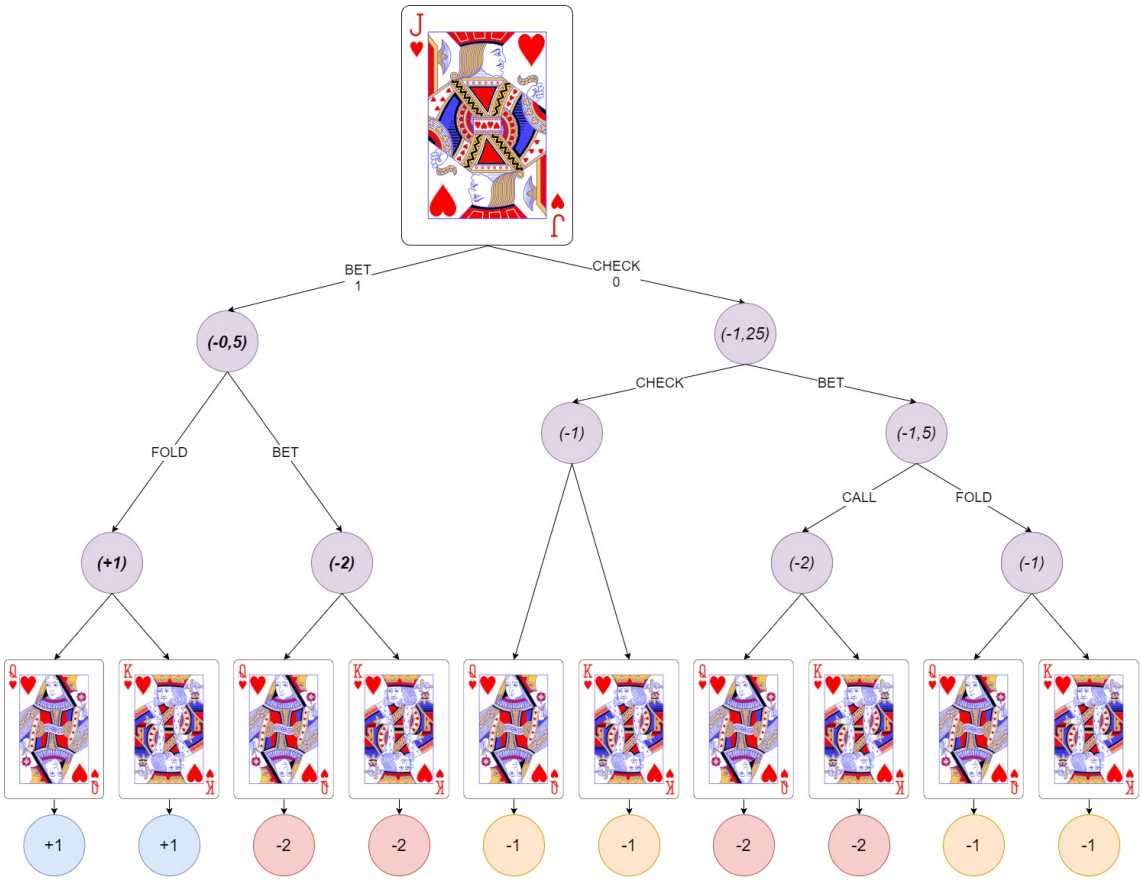


Abbildung 5.3 Strategiebildung MC-N mit Bube als Startspieler

Hierbei zeigt sich, dass der Agent in dieser Situation immer die Aktion BET bevorzugen wird, da er einen durchschnittlichen Gewinn von $-0,5$ erwartet, während er für die Aktion CHECK einen durchschnittlichen Gewinn von $-1,25$ erwartet. Nach Betrachtung aller Spielsituation, ergibt sich die Strategie, welche in Tabelle 5.11 gezeigt wird.

Tabelle 5.11 Strategie MC-N für Kuhnpoker. Grün hinterlegte Aktionen stimmen mit der optimalen Strategie überein, rot hinterlegte nicht.

(a) MC-N als Startspieler

| **Handkarte** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | BET | - |
| Dame | BET | - |
| König | BET | - |

(b) MC-N als Folgespieler

| | **Aktion Gegner** | |
|---|---|---|
| **Handkarte** | ***CHECK*** | ***BET*** |
| Bube | BET | FOLD |
| Dame | BET | CALL |
| König | BET | CALL |

Wenn die Strategien von sowohl der optimalen Strategie, als auch von MC-N bekannt sind, lässt sich damit der erwartete Gewinn der Agenten ermitteln, indem alle Spielverläufe betrachtet werden. Hieraus ergibt sich ein erwarteter Gewinn für MC-N als Startspieler von $-0,111$ und ein Gewinn von $-0,011$ als Folgespieler.

### 5.3.2 Realisierte Ergebnisse

Um die tatsächlichen Gewinne zu ermitteln, wurde ein Experiment durchgeführt mit 1.000.000 gespielten Runden zwischen MC-N und der optimalen Strategie. Hierbei wurden die erwarteten Ergebnisse bestätigt, wie in Tabelle 5.13 gezeigt wird.

Tabelle 5.13 Vergleich erwarteter und realisierter Gewinn für MC-N

| **Startspieler** | **Erwarteter Gewinn für Startspieler** | **Realisierter Gewinn für Startspieler** |
|---|---|---|
| MC-N | -0,111 | -0,1107 |
| Optimaler Agent | 0,011 | 0,0131 |

### 5.3.3 Fazit

Der MC-N Agent ist in der Lage Kuhn Poker zu spielen, jedoch erreicht er nicht die Qualität der optimalen Strategie. Die gefundene Strategie vermeidet offensichtliche Fehler, wie ein CALL mit einem Buben, jedoch unterschätzt sie den Gegner, weshalb nicht immer die beste Aktion gewählt wird. Deutlich wird dies in Abbildung 5.3. Durch die Simulation des Spiels mit zufälligen Agenten wird davon ausgegangen, dass ein Spieler mit einem König in 50% der Fälle auf ein BET mit einem FOLD reagiert. Ein vernünftig agierender Agent würde dies nicht tun, da ein sicherer Sieg verschenkt würde. Ein weiterer Nachteil der Strategie ist, dass sie ausgenutzt werden kann, sollte sie dem Gegner bekannt werden. Der MC-N Agent als Folgespieler reagiert mit einem Buben auf ein BET in jedem Fall mit einem FOLD, während er auf ein CHECK mit einem BET reagieren würde. Aus diesem Grund ist es nicht sinnvoll mit einem BET zu eröffnen, sondern ein CHECK zu wählen, da der Gegner mit einem BET reagieren wird. Hierdurch kann der erwartete Gewinn gegen den Agenten weiter erhöht werden.

## 5.4 MCTSE

Der MCTSE (Monte Carlo Tree Search Expectimax) Agent wählt seine nächste Aktion in einem gegebenen Spielzustand, ähnlich wie der MC-N Agent, basierend auf der Simulation einer Vielzahl von Spielen, wie in Abschnitt 3.2.3 beschrieben wird. Im Gegensatz zum MC-N simuliert er jedoch nicht nur wahllos zufällige Spielverläufe, sondern baut im Laufe der Simulation einen Baum aus den möglichen Spielzuständen und Aktionen auf, und wählt dabei die attraktivste Aktion für den jeweiligen Spieler.

### 5.4.1 Erwarteter Gewinn

Abbildung 5.4 zeigt beispielhaft die Auswahl der nächsten besten Aktion für den Agenten als Startspieler mit einem Buben als Handkarte. Der dargestellte Baum zeigt den erwarteten *Value* als normalisierten Reward für den jeweilig agierenden Spieler. In der Reaktion des Folgespielers (Player 0) wird deutlich, dass vielversprechendere Knoten häufiger besucht werden, da hier der CALL sechsmal häufiger besucht wird als die Aktion FOLD, die in jedem Fall einen Verlust für den Spieler darstellt. Insgesamt wird hier die Aktion CHECK mit einem erwarteten Wert von 0,439 der Aktion BET mit einem erwarteten Wert von 0,423 vorgezogen.

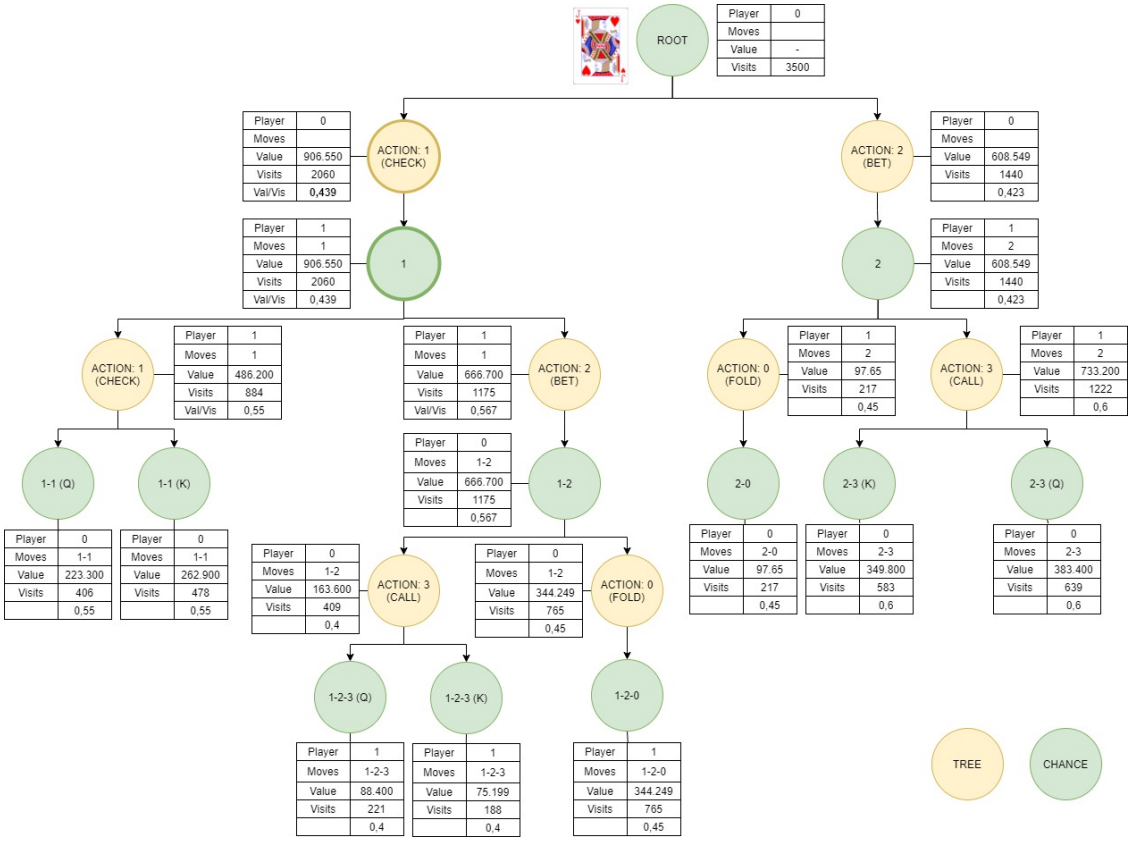


Abbildung 5.4 Strategiebildung MCTSE für Kuhnpoker mit Bube als Startspieler

Dieses Vorgehen kann auf alle Spielsituationen angewandt werden, wodurch sich die Strategie, welche in Tabelle 5.14 gezeigt wird, ergibt.

Tabelle 5.14 Strategie von MCTSE im Spiel Kuhn Poker. Grün hinterlegte Aktionen stimmen mit der optimalen Strategie überein, rot hinterlegte nicht.

(a) MCTS als Startspieler

| **Handkarte** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | CHECK | FOLD |
| Dame | BET | - |
| König | BET | - |

(b) MCTSE als Folgespieler

| | **Aktion Gegner** | |
|---|---|---|
| **Handkarte** | ***CHECK*** | ***BET*** |
| Bube | CHECK | FOLD |
| Dame | BET | CALL |
| König | BET | CALL |

Wenn die Strategien von sowohl der optimalen Strategie, als auch von MCTSE bekannt sind, lässt sich damit der erwartete Gewinn der Agenten ermitteln, indem alle Spielverläufe betrachtet werden. Hieraus ergibt sich ein erwarteter Gewinn für MCTSE als Startspieler von $-0,111$ und ein Gewinn von $-0,011$ als Folgespieler.

### 5.4.2 Realisierte Ergebnisse

Um die tatsächlichen Gewinne zu ermitteln, wurde ein Experiment durchgeführt mit 1.000.000 gespielten Runden zwischen MCTSE und der optimalen Strategie. Hierbei wurden die erwarteten Ergebnisse bestätigt, wie in Tabelle 5.16 gezeigt wird.

Tabelle 5.16 Vergleich erwarteter und realisierter Gewinn MC-N

| **Position** | **Erwarteter Gewinn** | **Realisierter Gewinn** |
|---|---|---|
| Startspieler | -0,111 | -0,1105 |
| Folgespieler | -0,011 | -0,0101 |

### 5.4.3 Fazit

Der MCTSE-Agent erzielt mit MC-N vergleichbare Ergebnisse. Es werden offensichtliche Fehler, wie ein CALL mit einem Buben vermieden, jedoch lässt sich die gefundene Strategie ausnutzen, sobald sie dem Gegner bekannt ist. Sollte der MCTSE Agent mit einem BET eröffnen, kann man durch Ausschlussverfahren feststellen, ob MCTSE eine Dame oder einen König hat, wodurch der durchschnittliche Gewinn weiter erhöht werden kann. Auffällig ist, dass der Agent den Buben nicht nutzt und lediglich CHECK beziehungsweise FOLD wählt, um den möglichen Verlust zu minimieren. Somit stellt ein Bube einen garantierten Verlust von -1 dar, und die Chance zu bluffen wird ignoriert.

## 5.5 Expectimax

In diesem Abschnitt werden die Ergebnisse des Expectimax Agenten, der in Abschnitt 3.2.3 beschrieben wurde, vorgestellt. Zuerst wird die erwartete Strategie und der daraus resultierende erwartete Gewinn ermittelt und anschließend mit dem in einem Experiment ermittelten tatsächlichen Gewinn verglichen.

### 5.5.1 Erwarteter Gewinn

Um den erwarteten Gewinn des Expectimax Agenten zu bestimmen, wird im ersten Schritt die Strategie des Agenten ermittelt. Dafür ist es notwendig die unterschiedlichen Spielsituationen zu betrachten und den durchschnittlichen Gewinn für jede mögliche Aktion zu bestimmen. Abbildung 5.5 zeigt beispielhaft diese Bestimmung für den Agenten als Startspieler. Auf diesem Weg kann für jede Spielsituation die, laut Agent, optimale Aktion identifiziert werden. Hieraus ergibt sich die in Tabelle 5.17 gezeigte Strategie.

Tabelle 5.17 Strategie Expectimax für Kuhn Poker. Grün hinterlegte Aktionen stimmen mit der optimalen Strategie überein, rot hinterlegte nicht.

(a) Expectimax als Startspieler

| **Hand** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | CHECK | FOLD |
| Dame | 50% CHECK<br>50% BET | CALL |
| König | 50% CHECK<br>50% BET | CALL |

(b) Expectimax als Folgespieler

| | **Aktion Gegner** | |
|---|---|---|
| **Hand** | ***CHECK*** | ***BET*** |
| Bube | CHECK | FOLD |
| Dame | 50% CHECK<br>50% BET | CALL |
| König | 50% CHECK<br>50% BET | CALL |

Da die Strategien von sowohl der optimalen Strategie, als auch von Expectimax bekannt sind, lässt sich der erwartete Gewinn der Agenten ermitteln, indem alle Spielverläufe betrachtet werden. Hieraus ergibt sich ein erwarteter Gewinn für Expectimax als Startspieler von $-0.083$ und ein Gewinn von$-0,022$ als Folgespieler.

### 5.5.2 Realisierter Gewinn

Um die tatsächlichen Gewinne zu ermitteln, wurde ein Experiment durchgeführt mit 1.000.000 gespielten Runden zwischen Expectimax und der optimalen Strategie. Hierbei wurden die erwarteten Ergebnisse bestätigt, wie in Tabelle 5.19 gezeigt wird.

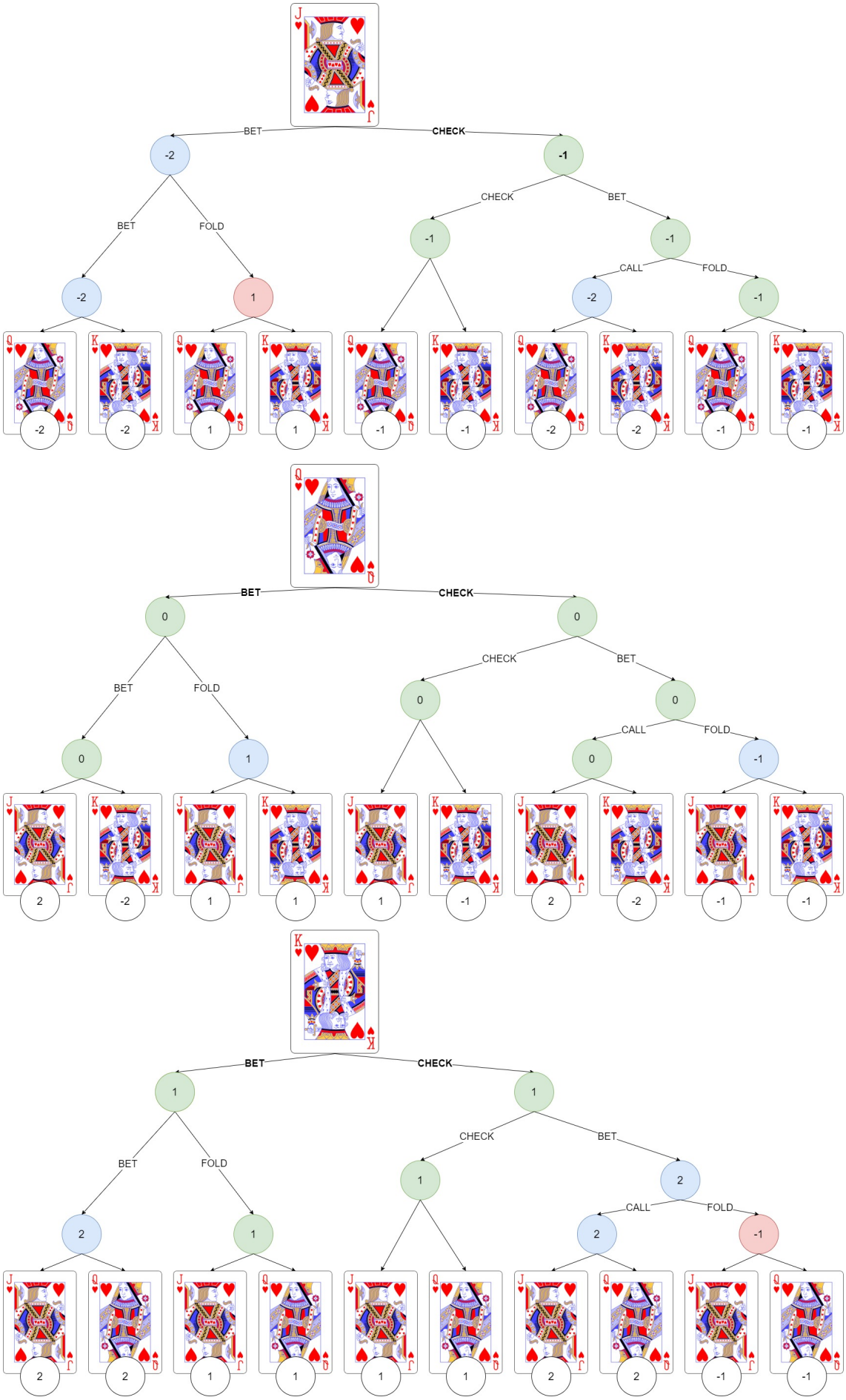


Abbildung 5.5 Strategiefindung für Expectimax als Startspieler

Tabelle 5.19 Vergleich erwarteter und realisierter Gewinn für Expectimax

| **Startspieler** | **Erwarteter Gewinn für Startspieler** | **Realisierter Gewinn für Startspieler** |
|---|---|---|
| Expectimax | -0.083 | -0,0838 |
| Optimale Strategie | 0,022 | 0,0215 |

### 5.5.3 Fazit

Der Expectimax Agent erzielt bessere Ergebnisse als MC-N und MCTSE als Startspieler, jedoch schlechtere Ergebnisse als Folgespieler. Offensichtliche Fehler wie ein CALL mit einem Buben werden vermieden, jedoch wird der Bube nicht genutzt, da davon ausgegangen wird, dass der Gegner keinen Fehler macht und die Verluste in jeder Runde mit der Strategie minimiert werden sollen. Das Spiel gegen diese Strategie lässt sich beispielsweise dadurch optimieren, dass es bei einem BET einfach ist, die Karte des Gegners zu bestimmen, da er nur mit einer Dame und einem König BET wählt. Sollte man also einen König haben ist es sinnvoll, mit einem eigenen BET zu reagieren, ansonsten sollte FOLD gewählt werden, da der Gegner eine bessere Karte hat.

## 5.6 Lernende Agenten

Im Gegensatz zu den bisher beschriebenen Agenten ist es nicht ohne weiteres möglich, im Vorfeld die erwarteten Strategien der verschiedenen lernenden Agenten zu ermitteln. Aus diesem Grund ist es auch nicht möglich, den erwarteten Gewinn im Vorfeld zu bestimmen. Im Folgenden werden daher die einzelnen Strategien der Agenten vorgestellt und mit der ermittelten Strategie in Tabelle 5.1 verglichen. Dabei wurden sowohl für die N-Tuple Repräsentation, als auch das erstellte Feature für TDS die Handkarte des Spielers, sowie die bisherigen Aktionen als Informationen für die Agenten bereitgestellt. Als *learn step size* wurde 0.001 verwendet, als *initial random move rate* 0.3 und als *final random move rate* 0.

### 5.6.1 Strategien

Im Folgenden werden die gefundenen Strategien der einzelnen Agenten vorgestellt. Eine Abweichung von einer optimalen Strategie ist rot hinterlegt, eine Übereinstimmung grün.

Tabelle 5.20 Strategie TDS für Kuhn Poker. Grün hinterlegte Aktionen stimmen mit der optimalen Strategie überein, rot hinterlegte nicht.

(a) TDS als Startspieler

| **Hand** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | CHECK | FOLD |
| Dame | CHECK | 50% CALL<br>50% FOLD |
| König | CHECK | CALL |

(b) TDS als Folgespieler

| | **Aktion Gegner** | |
|---|---|---|
| **Hand** | ***CHECK*** | ***BET*** |
| Bube | CHECK | FOLD |
| Dame | CHECK | 50% CALL<br>50% FOLD |
| König | CHECK | CALL |

Tabelle 5.22 Strategie TD-NTuple-3 für Kuhn Poker. Grün hinterlegte Aktionen stimmen mit der optimalen Strategie überein, rot hinterlegte nicht.

(a) TD-NTuple-3 als Startspieler

| **Hand** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | 50% CALL<br>50% FOLD | FOLD |
| Dame | 50% CALL<br>50% FOLD | 50% CALL<br>50% FOLD |
| König | 50% CALL<br>50% FOLD | CALL |

(b) TD-NTuple-3 als Folgespieler

| | **Aktion Gegner** | |
|---|---|---|
| **Hand** | ***CHECK*** | ***BET*** |
| Bube | BET | FOLD |
| Dame | 50% CALL<br>50% FOLD | 50% CALL<br>50% FOLD |
| König | CHECK | CALL |

Tabelle 5.24 Strategie TD-NTuple-4 für Kuhn Poker. Grün hinterlegte Aktionen stimmen mit der optimalen Strategie überein, rot hinterlegte nicht.

(a) TD-NTuple-4 als Startspieler

| **Hand** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | 50% BET<br>50% FOLD | 50% CALL<br>50% FOLD |
| Dame | 50% BET<br>50% FOLD | 50% CALL<br>50% FOLD |
| König | 50% BET<br>50% FOLD | CALL |

(b) TD-NTuple-4 als Folgespieler

| | **Aktion Gegner** | |
|---|---|---|
| **Hand** | ***CHECK*** | ***BET*** |
| Bube | BET | FOLD |
| Dame | 50% CALL<br>50% FOLD | 50% CALL<br>50% FOLD |
| König | CHECK | CALL |

Tabelle 5.26 Strategie Q-Learn-4 für Kuhn Poker. Grün hinterlegte Aktionen stimmen mit der optimalen Strategie überein, rot hinterlegte nicht.

(a) Q-Learn-4 als Startspieler

| **Hand** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | BET | - |
| Dame | CHECK | CALL |
| König | BET | - |

(b) Q-Learn-4 als Folgespieler

| | **Aktion Gegner** | |
|---|---|---|
| **Hand** | ***CHECK*** | ***BET*** |
| Bube | CHECK | FOLD |
| Dame | CHECK | CALL |
| König | CHECK | CALL |

Tabelle 5.28 Strategie Sarsa-4 für Kuhn Poker. Grün hinterlegte Aktionen stimmen mit der optimalen Strategie überein, rot hinterlegte nicht.

(a) Sarsa-4 als Startspieler

| **Hand** | **Aktion** | **Reaktion** |
|---|---|---|
| Bube | BET | - |
| Dame | CHECK | CALL |
| König | BET | - |

(b) Sarsa-4 als Folgespieler

| | **Aktion Gegner** | |
|---|---|---|
| **Hand** | ***CHECK*** | ***BET*** |
| Bube | CHECK | FOLD |
| Dame | CHECK | CALL |
| König | CHECK | CALL |

### 5.6.2 Fazit

Es zeigt sich, dass verschiedene lernende Agenten in der Lage sind, eine optimale Strategie als Startspieler zu finden, was durch die erzielten durchschnittlichen Gewinne in Tabelle 5.30 bestätigt wird. Für das Spiel als Folgespieler, der grundsätzlich in der Lage ist, Gewinne gegen die optimale Strategie zu erzielen, ist es jedoch keinem der Agenten gelungen, eine optimale Strategie zu finden.

## 5.7 Übersicht & Fazit

Verschiedene Agenten sind in der Lage Kuhn Poker zu spielen, jedoch erreicht kein Agent sowohl als Startspieler, als auch als Folgespieler die Qualität der optimalen Strategie. Als Startspieler finden *TDS*, *Q-Learn-4* und *Sarsa-4* eine Strategie, die mit den Ergebnissen der optimalen Strategie vergleichbar ist. Als Folgespieler ist jedoch keiner der Agenten in der Lage, einen Gewinn zu erzielen.

Tabelle 5.30 Durchschnittlicher Gewinn der Agenten im Spiel Kuhn Poker gegen die optimale Strategie

| **Agent** | **Gewinn Startspieler** | **Gewinn Folgespieler** |
|---|---|---|
| Random | -0.1664 | -0.1230 |
| MCN | -0.1107 | **-0.0131** |
| MCTSE | -0.1105 | **-0.0101** |
| Expectimax | -0.0838 | -0.0215 |
| TDS | **-0.0549** | -0.0327 |
| QLearn-4 | **-0.0554** | -0.0334 |
| Sarsa-4 | **-0.0548** | -0.0321 |
| TD-NTuple-3 | -0.0832 | -0.0674 |
| TD-NTuple-4 | -0.1239 | -0.0637 |
| Optimale Strategie | **-0.0530** | 0.0545 |

Die eingehende Fragestellung, ob die Agenten im GBG-Framework in der Lage sind Kuhn Poker, trotz der Herausforderungen durch zum Beispiel versteckte Informationen zu erlernen, lässt sich somit klar bejahen.

# 6 Texas Hold'em Poker

In diesem Kapitel wird die Frage betrachtet, ob die im GBG-Framework implementierten Agenten das Spiel Texas Hold'em Poker in einer zwei-Spieler-Variante erlernen können.

Der Aufbau des Kapitels orientiert sich am Vorgehen während der Auswertung, welche im Folgenden näher beschrieben wird. Im ersten Schritt für die Auswertung wird die Art des Vergleichs, beziehungsweise die betrachteten *Episoden* definiert. Sowohl ein komplettes Spiel, aber auch einzelne Spielrunden, stellen dabei denkbare Betrachtungen dar. Die einzelnen Vor- und Nachteile werden in Abschnitt 6.1 beschrieben. Anschließend wird in Abschnitt 6.2 näher erläutert, nach welchen Kriterien die Agenten bewertet und im Anschluss verglichen werden. Nachdem so definiert wurde, *was* verglichen werden soll, muss darüber hinaus geprüft werden, *wie* der Vergleich stattfinden soll. Um eine perfekte Aussage über die Stärke eines Agenten gegen einen anderen Agenten treffen zu können, müsste jede Spielsituation betrachtet werden, wie es beispielsweise bei den erwarteten Gewinnen für die Agenten bei Kuhn Poker gemacht wurde. Im Gegensatz zu einem einfachen Spiel wie Kuhn-Poker ist der Raum der möglichen Spielzustände für Texas Hold'em allerdings zu groß, um ihn vollständig, im Rahmen des GBG-Frameworks, zu betrachten. Aus diesem Grund wird im ersten Schritt die Frage beleuchtet, wie viele Episoden gespielt werden müssen, damit die Ergebnisse belastbar sind. Anschließend wird durch erste Auswertungen der vielversprechendste Agent identifiziert, der im Anschluss optimiert werden soll. Die Ergebnisse dieser Auswertung werden in Abschnitt 6.6 vorgestellt. Im weiteren Verlauf wird der Fokus auf den vielversprechendsten Agenten gelegt, und dieser exemplarisch näher behandelt. Es wird zuerst die Varianz der Ergebnisse in Abschnitt 6.7 untersucht und danach betrachtet, wie viele Trainingsepisoden notwendig sind, um gute Ergebnisse zu erziele. Abschließend wird eine Übersicht der Ergebnisse aller Agenten in Abschnitt 6.9 gegeben und ein Fazit in Abschnitt 6.10 gezogen.

## 6.1 Episoden

Eine grundlegende Entscheidung für die Bewertung der Agenten ist die Frage, was genau analysiert werden soll. In vielen Spielen, wie beispielsweise Tic-Tac-Toe oder

Schach, ist diese Frage einfach zu beantworten: ein Spiel hat einen Anfang und ein Ende, die eine Episode definieren. Ein Reward wird dabei im Normalfall am Ende der Episode an den Agenten ausgeschüttet und die Bewertung der Agenten findet auf der Basis vollständiger Episoden statt. Wenn dieselbe Vorgehensweise auf Poker angewandt wird, entstehen dadurch jedoch unterschiedliche Probleme. Als erstes fällt dabei auf, dass durch die Betrachtung vollständiger Spiele sehr viel Zeit in Anspruch genommen wird. Dies resultiert daraus, dass es durch den rundenbasierten Aufbau des Spiels sehr viel vor und zurück geben kann, bis ein Sieger feststeht. Dieser Effekt wird umso deutlicher, je vergleichbarer die Stärke der Agenten ist. Um dies deutlich zu machen wurde ein Spiel zwischen zwei Agenten simuliert, wobei als Vereinfachung angenommen wurde, dass ein Spieler lediglich 10 Chips gewinnen, beziehungsweise verlieren kann.

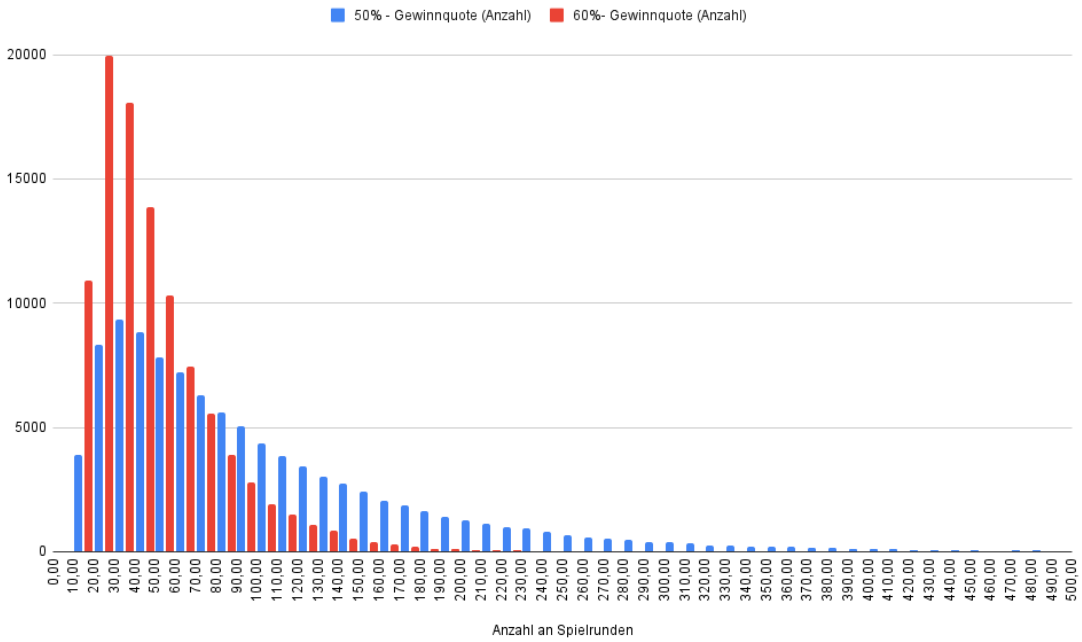


Abbildung 6.1 Simulierte Anzahl der Runden pro Spiel

Angenommen ein Agent hat eine Gewinnquote von 60%, dann ergibt sich eine durchschnittliche Anzahl von etwa 48,3 Spielrunden mit einer Standardabweichung 31,5 Spielrunden. Sollten zwei gleichstarke Agenten, also mit einer Gewinnquote von 50% gegeneinander antreten, so verdoppelt sich die Anzahl der durchschnittlichen Spielrunden auf 100,31 mit einer Standardabweichung von 81,27 Spielrunden. Dies führt dazu, dass die Analyse kompletter Spiele als betrachtete Episode sehr viel Zeit in Anspruch nimmt. Aufgrund der großen Menge an Spielzügen in einem kompletten Spielverlauf, die für den Agenten nur schwer in gute und schlechte Züge unterteilbar sind, ist anzunehmen, dass der Agent größere Schwierigkeiten haben wird, erfolgrei-

che Strategien zu erlernen.

Aufgrund dieser Schwierigkeiten wurde im Rahmen dieser Ausarbeitung entschieden nicht komplette Spiele, sondern Spielrunden in den Fokus zu stellen. Dies hat den Vorteil, dass sie schnell zu simulieren sind und das Ergebnis in direktem Zusammenhang mit den gewählten Aktionen steht. Man verzichtetet dadurch jedoch auch auf manche Aspekte des Pokerspiels, die gelernt werden könnten. Ein Beispiel hierfür wäre ein unterschiedliches Verhalten in Abhängigkeit der vorhandenen Chips, da ein größerer Vorsprung gegenüber den Mitspielern oftmals eine aggressivere Strategie ermöglicht, während ein Rückstand oft eine defensive Spielweise nahelegt.

## 6.2 Benchmark-Agent

Um die Qualität der Agenten zu messen ist es notwendig, einen Maßstab zu definieren, anhand dessen Agenten gemessen werden können. Im besten Fall wäre es möglich das Minimum und das Maximum zu definieren. Als Minimum für die Auswertung kann der zufällig agierende Agent verwendet werden. Dieser Agent kann zwar das Spiel spielen, trifft aber offensichtliche Fehlentscheidungen. Ein Agent, der konstant besser abschneidet als der zufällig agierende Agent, konnte das Spiel demnach zu einem gewissen Grad erlernen. Das Maximum zu definieren ist für Texas Hold'em nicht einfach möglich. Im Gegensatz zum relativ einfachen Spiel Kuhn Poker gibt es für das Spiel Texas Hold'em keine optimale Strategie, welche als Maßstab herangezogen werden könnte und einfach im Framework zu implementieren wäre. Aus diesem Grund wird für die Auswertung ein Agent implementiert, welcher als Benchmark genutzt wird. Dieser Agent folgt fest vorgegeben Regeln und stellt keinen besonders starken Spieler dar, jedoch vermeidet er offensichtliche Fehler und stellt damit einen sinnvolleren Vergleich als der Random Agent dar. Die Reproduzierbarkeit der Ergebnisse ist ein weiterer Vorteil. Der Agent greift zu Beginn der Spielrunde auf eine Lookup-Tabelle zurück, um die Stärke seiner Starthand einzuschätzen. Diese Lookup-Tabelle stellt eine Bewertung jeder Handkartenkombination dar und wurde in Form eines mehrdimensionalen Arrays implementiert, wobei zwischen gleichen und unterschiedlichen Farben differenziert wurde. In den weiteren Zügen wird die Stärke der Hand basierend auf der Anzahl an vorhandenen Paaren, beziehungsweise anderen Blättern bewertet, wobei er, je stärker die Hand eingeschätzt wird, umso aggressiver setzt. Dies bedeutet zum Beispiel, dass sollte der Bot ein Paar besitzen, er voraussichtlich bei einem BET des Gegners mitgeht, sollte er hingegen einen Drilling haben, wird er mindestens ein RAISE wählen und zu einer Wahrscheinlichkeit von 10% ein ALL-IN wählen. Im Spiel gegen einen zufällig agierenden Agenten gewinnt der Benchmark Agent im Schnitt als Startspieler 4.32 Chips pro Runde und als Folgespieler 2.866 Chips pro Runde. Im Spiel gegen sich selbst gewinnt der Benchmark

Agent als Startspieler etwa 0.958 Chips pro Runde, und verliert entsprechend etwa 0.958 Chips pro Runde als Folgespieler. Dieser Agent wird in den folgenden Auswertungen als Gegenspieler für alle anderen Agenten verwendet und dient somit als Benchmark.

## 6.3 Nicht-lernende Agenten

Im Gegensatz zu den lernenden Agenten gibt es verschiedene Agenten im GBG-Framework, die das Spiel nicht *erlernen* und somit nicht trainiert werden müssen, sondern versuchen, die beste Aktion in einer gegebenen Spielsituation zu finden, indem weitere Spiele simuliert werden. Diese Agenten könnten als ein Vergleichskriterium für die lernenden Agenten genutzt werden, da sie voraussichtlich keine unmittelbaren Fehlentscheidungen treffen sollten. Im Folgenden werden diese Agenten vorgestellt und untersucht.

### 6.3.1 Expectimax

Im Spiel Kuhn Poker konnte der Expectimax Agent relativ gute Ergebnisse erzielen, jedoch ist er praktisch nicht für Texas Hold'em anwendbar. Der Grund dafür ist, dass der Expectimax Agent versucht, den höchsten erwarteten Gewinn der verfügbaren Aktionen zu bestimmen, wobei jede mögliche Spielsituation betrachtet werden muss. Für Texas Hold'em sind die möglichen Spielsituationen durch die vielen zufälligen Karten zahlreich, wodurch die notwendige Zeit, die der Agent bräuchte um eine Entscheidung zu finden, extrem hoch ist. Aus diesem Grund wird er für die Analyse nicht in Betracht gezogen.

### 6.3.2 MC-N

Der MC-N Agent wählt seine Aktion, indem er für jede Aktion eine möglichst große Menge an zufälligen Spielverläufen zwischen zufällig agierenden Spielern simuliert. Je größer die Anzahl der Simulationen ist, desto größer ist die voraussichtliche Spielstärke des Agenten. Bei den Simulationen sind zwei Aspekte zu beachten. Einerseits werden neue Informationen durch die neu aufgedeckten Karten gewonnen, wobei durch eine große Menge an Simulationen eine gute Einschätzung der Stärke der eigenen Hand ermöglicht werden sollte. Andererseits besteht das Problem, dass ein zufällig agierender Agent offensichtliche Fehlentscheidungen treffen wird, wie ein FOLD bei einem starken Blatt, wodurch die Ergebnisse verzerrt werden. Jedoch sollte auch hier eine möglichst große Menge an Simulationen helfen, diesen Effekt zumindest gleichverteilt bei allen Aktionen zu haben. Das Problem, das hierbei entsteht, ist, dass ein Ausgleich zwischen den simulierten Iterationen des MC-N Agenten und den gespielten Runden, die für die Analyse gebraucht werden, gefunden werden

muss. Für jede simulierte Iteration pro Spiel des MC-N Agents werden im Schnitt 0.0117ms benötigt. Daraus folgt, dass für einen Vergleich von 4 Mio. Spielrunden mit jeweils nur einer einzigen Iteration bereits etwa 47 Sekunden benötigt werden. Damit wird deutlich, dass nicht dieselbe Menge an Spielen simuliert werden kann, wie für die lernenden Agenten, da selbst mit nur 10.000 simulierten Iterationen bereits mehr als 5 Tage Rechenaufwand benötigt würden. Um vergleichbare Zahlen zu ermitteln, habe ich mich auf 100.000 simulierte Iterationen festgelegt und etwa 150.000 Spielrunden analysiert, wobei ein durchschnittlicher Gewinn von 1,056 Chips als Startspieler und 1,910 Chips als Folgespieler erzielt wurden.

### 6.3.3 MCTSE

Der MCTSE-Agent hat ähnliche Voraussetzungen, wie der MC-N Agent, wobei der Agent nicht, wie der MC-N Agent, lediglich die Aktionen betrachtet, sondern intern einen Baum der möglichen Aktionen und Spielzustände aufbaut. Im Gegensatz zum Expectimax-Agent wird jedoch nicht der komplette Spielbaum betrachtet, sondern lediglich so viel, wie der Agent im Rahmen der vorgegebenen Iterationen erkunden kann. Der Vorteil, den dieses Vorgehen bietet, ist, dass der Agent die Möglichkeit hat, sich auf die vielversprechendsten Bereiche des Baums und damit die Aktionen mit dem vermutlich größten Potential zu konzentrieren. Der Nachteil, der dadurch entsteht, ist, dass der Agent mehr Zeit und Ressourcen braucht um die Simulationen durchzuführen und die Werte entsprechend anzupassen. Dies führt dazu, dass der Agent etwa 0.021ms pro Spiel und Iteration benötigt. Wie schon beim MC-N Agent macht es der Zeit- und Ressourcenaufwand nicht sinnvoll 4 Mio. Spiele, wie für den lernenden Agenten, zu betrachten. Stattdessen habe ich mich auch hier auf 100.000 simulierte Iterationen festgelegt und etwa 100.000 Spielrunden analysiert. Hierbei erreichte der Agent mit etwa 0.160 Chips als Startspieler und 0.865 Chips als Folgespieler deutlich schlechtere Ergebnisse als der MC-N Agent. Der Grund hierfür könnte sein, dass der Agent viele Simulationen und Zeit in augenscheinlich vielversprechende Aktionen investiert, welche jedoch nur durch glückliche Ausgänge der ersten Simulationen vielversprechend wirkten.

## 6.4 Feature und Tupel

Damit die lernenden Agenten im GBG-Framework, also TDS und auf $n$-Tupel basierende Agenten, ein Spiel erlernen können, müssen die relevanten Informationen, die den Spielzustand beschreiben, den Agenten zur Verfügung gestellt werden. Hierbei wird zwischen dem TDS Agenten, der die Informationen als Feature erhält, und den $n$-Tupel Agenten unterschieden. Grundsätzlich müssen jedoch in beiden Varianten die für den Agenten relevanten Aspekte des Spiels identifiziert werden. Die wichtigs-

ten Informationen sind voraussichtlich die ausgeteilten Karten, bestehend aus den beiden Handkarten, drei Flop-Karten, einer Turn-Karte und der River-Karte. Hierbei zu beachten ist, dass nicht alle Karten gleich zu Beginn bekannt sind, sondern dass diese im Laufe der Zeit aufgedeckt werden. Auch sind nicht alle sieben Karten direkt für die eigene gebildete Hand relevant, da diese maximal aus fünf Karten besteht. Es gibt jedoch gibt es Situationen, in denen auch die vom Spieler nicht genutzten Karten Aufschluss über die potentielle Handstärke des Gegners geben kann. Ein Beispiel hierfür zeigt Abbildung 6.2. Der Spieler hat mit einem Drilling eine starke Hand, jedoch stellen die aufgedeckten restlichen Karten eine sehr hohe Chance auf einen Straight, beziehungsweise einen Flush dar, die beide höher bewertet sind als der Drilling des Spielers.

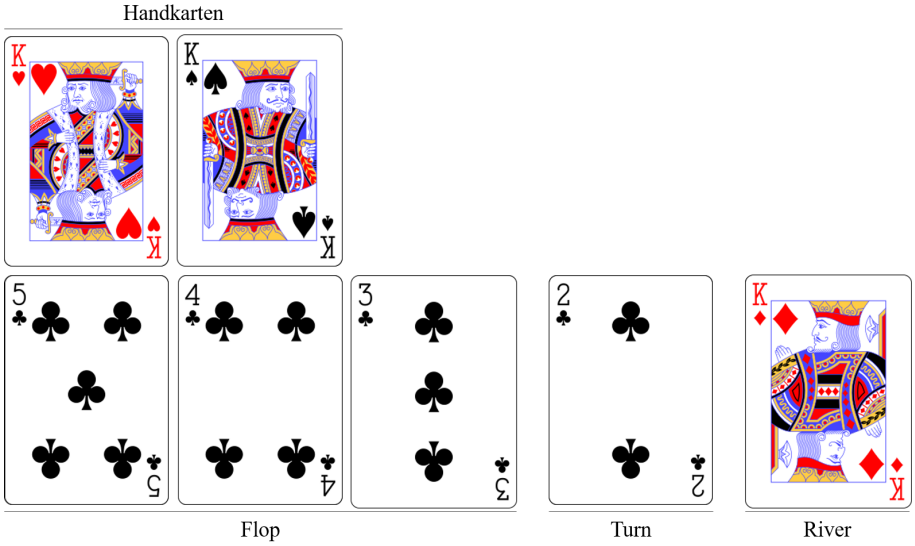


Abbildung 6.2 Potentiell überbewertete Hand in einem Spiel Texas Hold'em

Neben den Handkarten gibt es weitere, potentiell für den Agenten relevanten Informationen über das Spiel:

- Offener Betrag, der gesetzt werden muss.
- Die Menge der vorhandenen Chips der einzelnen Spieler, sowie die bereits gesetzten Chips, die den potentiellen Gewinn darstellen.
- Die Position in der Zugfolge, welche durch den rotierenden Smallblind, Bigblind und Dealer Button bestimmt wird.
- Die Aktionen der vorherigen Spieler.

Darüber hinaus könnte, sollte über mehrere Spielrunden gespielt werden, ebenfalls versucht werden, die Strategie des Gegners zu ermitteln und so das Spiel gegen einen aggressiven Spieler anders zu gestalten, als das gegen einen defensiven. Im Rahmen dieser Arbeit werden lediglich Spielrunden und keine kompletten Spiele betrachtet.

Aus diesem Grund könnten zwar Aspekte wie vorhandene Chips aufgenommen werden, jedoch wird es vermutlich kaum einen Einfluss auf diese Ergebnisse haben, da diese am Anfang jeder Spielrunde zurückgesetzt werden. Auch die Position wird voraussichtlich eine untergeordnete Rolle haben, da es sich hierbei lediglich um ein Spiel mit zwei Spielern handelt. In den folgenden Abschnitten werden sowohl die Ansätze zur Findung eines geeigneten $n$-Tupels, als auch die eines geeigneten Features beschrieben.

### 6.4.1 Feature

Das für das Poker Spiel genutzte Featurevektor besteht aus einem Vektor der Länge $10 + AnzahlSpieler - 1$, wobei die folgenden Aspekte des Spiels abgebildet sind:

0 : Offener Betrag, der gesetzt werden muss

1 : Anzahl der vorhandenen Chips des Agenten

2 : Anzahl der vorhandenen Chips im Pot

3-4 : ID der Handkarten [1]

5-7 : ID der drei Flop-Karten

8 : ID der Turn-Karte

9 : ID der River-Karte

10 : Spielzustand[2]

Dieses Feature beinhaltet auch Informationen, die voraussichtlich nicht benötigt werden, jedoch erzielt es solide Ergebnisse und kann auch in anderen Varianten des Pokerspiels mit mehr Spielern eingesetzt werden.

### 6.4.2 XN-Tuple

Für die Definition der $n$-Tupel muss im ersten Schritt bestimmt werden, welche Informationen voraussichtlich für einen Agenten zum Lernen des Spiels relevant sein könnten. Der einfachste Ansatz wäre es, dieselben Informationen wie im Feature zu übergeben. Die Informationen werden innerhalb der $n$-Tupel in einem Array abgebildet. Die Größe des Arrays setzt sich dabei zum einen aus den betrachteten Attributen, zum anderen aus den möglichen Ausprägungen dieser Attribute zusammen [Lucas, 2008a]. Dies führt allerdings dazu, dass nicht einfach alle Informationen

[1] also z.B. ♦2 = 1, ♦3 = 2, ...

[2] also 'muss noch einen Einsatz machen', 'hat gepasst' ...

in ein Tupel gegeben werden können. Würden alle Spielkarten in einem Tupel abgebildet werden, so ergibt sich daraus, auch ohne weitere Infomationen, ein Array der Größe:

$$52^{2+3+1+1} = 52^{7} = 1.028.071.702.528$$

Demnach würde allein dafür der Speicher für 1.028.071.702.528 (1E12) Double Werte allokiert werden. Ein Double Wert in Java benötigt 8 Byte. Somit werden allein dafür 8E12 Byte benötigt das entspricht also 8 TB. Somit ist diese Abbildung nicht umsetzbar. Ein weiterer möglicher Ansatz wäre es, die einzelnen Spielkarten als Zelle im Spiel zu betrachten, wobei die einzelnen Zellen über zwei Zustände verfügen würden, indem sie entweder Teil der Hand sind oder nicht. Dieses Vorgehen würde jedoch einen notwendigen Speicher von:

$$2^{52} = 4.503.599.627.370.496$$

benötigen und stellt somit eine schlechtere Alternative dar. Ein anderer Ansatz wäre es, nicht die konkreten einzelnen Karten zu verwenden, sondern lediglich eine Zusammenfassung der Werte bereitzustellen, indem nur die Anzahl der Werte und die Anzahl der Farben bereitgestellt werden.

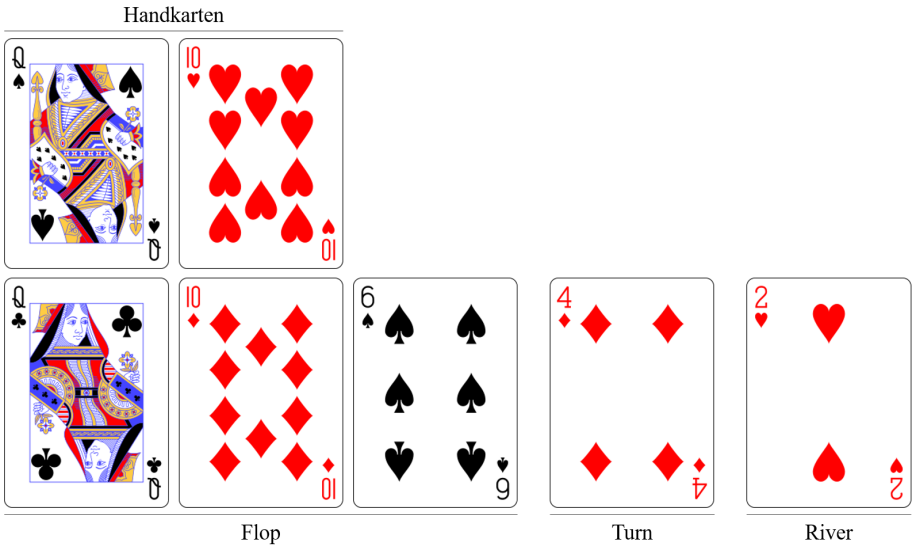


Abbildung 6.3 Beispiel Karten in einem Spiel Texas Hold'em

Die Hand in Abbildung 6.3 kann somit dargestellt werden als zwei Damen, zwei Zehner, eine Sechs eine 4 und eine 2, wobei jede Farbe zweimal vertreten ist bis auf Kreuz, das einmal vorkommt. Diese Darstellung findet sich beispielhaft in Tabelle 6.1

Tabelle 6.1 Darstellung der Beispielkarten in einer Zusammenfasung

| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | J | Q | K | A | ♠ | ♥ | ♦ | ♣ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 2 | 2 | 2 | 1 |

Mit dieser Darstellung wird ein Array der Größe

$$5^{13} * 6^{4} = 1.220.704.421$$

benötigt. Dies stellt zwar eine Verbesserung um den Faktor 1.000 dar, da nur noch ein Array der Größe 1E09 anstelle von 1E12 benötigt wird, im Gegenzug verzichtet man mit dieser Darstellung auf Informationen, da beispielsweise nicht eindeutig ist wenn man einen Flush hat, wie stark dieser ist. Doch wie wird dieses Problem in anderen Spielen gelöst? In einem Spiel wie Othello ergibt sich die Definition der $n$-Tupel aus dem Aufbau des Spiels. Es besteht aus einem 8*8 Spielfeld, wobei jede Zelle auf dem Spielfeld entweder von Weiß beziehungsweise Schwarz belegt ist oder sie leer ist. Somit müsste ein Array, welches die kompletten Informationen abdeckt, eine Größe haben von:

$$3^{8*8} = 3.433.683.820.292.512.484.657.849.089.281 \approx 3E30$$

Bei der Bildung der Tupel ist es jedoch nicht notwendig, alle Informationen gleichzeitig bereitzustellen, sondern es ist möglich, sich auf einzelne Bereiche zu beschränken. So ist es möglich, die voraussichtlich wichtigen Bereiche, wie die Ecken bei Othello, manuell zu definieren, es ist aber auch möglich zufällige Tupel beispielsweise durch Random Walks zu bilden. Die Frage ist, welche Tupel sinnvoll für Poker gebildet werden können. Es wurden unterschiedliche Versuche durchgeführt. Hierbei wurden unterschiedliche Tupel gebildet, um unterschiedliche Kombinationen abzubilden. Leider war dieses Vorgehen nur bedingt erfolgreich. Im nächsten Schritt wurden Tupel auf Basis der zusammengefassten Daten gebildet, indem zum Beispiel hohe und niedrige Bereiche definiert wurden, die anhand der Tupel abgebildet wurden. Dieses Vorgehen war zwar erfolgreicher als der erste Schritt, jedoch weiterhin noch nicht zufriedenstellend. Im letzten Schritt wurde die Idee des Zusammenfassens weiter verfolgt und die Informationen der Hand weiter kondensiert, indem bereits angegeben wurde, wie viele Karten in direkter Reihenfolge sind, wie viele Paare, Drillinge und so weiter vorhanden sind. Durch dieses Vorgehen wurden weitere Informationen eingebüßt, so ist es zum Beispiel für den Agenten nicht mehr möglich zu erkennen wie stark die eigene Straße ist. Ziel der Auswertung ist es jedoch auch nicht, einen perfekt spielenden Agenten zu trainieren, sondern es soll lediglich die Frage beantwortet werden, ob es grundsätzlich möglich ist. Da die Agenten mit diesem Tupel die besten Ergebnisse erzielt haben, wurde es für den weiteren Verlauf als Standard gewählt.

## 6.5 Bewertung der Agenten

Der folgende Abschnitt untersucht die Fragestellung, wie viele Spielrunden betrachtet werden müssen, um eine verlässliche Aussage über die Qualität eines Agenten geben zu können. Um die optimale Antwort auf diese Fragestellung zu bekommen, müsste jede potentielle Spielsituation betrachtet werden und für den Fall, dass einer der Agenten nicht deterministisch handelt, sogar mehrfach. Aus praktischer Sicht ist dies für ein Spiel wie Poker nicht möglich. Auch ist fraglich, ob die Simulation von einer Milliarde Spielrunden deutlich mehr Aussagekraft bietet, als die Bewertung von 10 Millionen Spielen. Grundsätzlich werden die Ergebnisse mit zunehmender Rundenzahl genauer, jedoch kostet jede Spielrunde Ressourcen, weshalb die Menge an Spielrunden so klein wie möglich gehalten werden sollte. Im Folgenden wird untersucht, wie viele Spielrunden benötigt werden, um ausreichend gute Ergebnisse zu erzielen, und wie groß die Spannweite dieser Ergebnisse ist. Die Analyse erfolgt unter der Annahme, dass es für die Ermittlung der Ergebnisse keine besondere Rolle spielt, welche Strategie im Detail verfolgt wird, da lediglich ermittelt werden soll, wie viele Spiele benötigt werden, um den durchschnittlichen Gewinn zu ermitteln. Ob dieser Gewinn nun bei einer starken Strategie sehr hoch oder bei einer schwachen Strategie niedrig ausfällt, sollte nur eine untergeordnete Rolle spielen. Der relevante Aspekt ist die Frage, wie viele Spielsituationen betrachtet werden müssen, um eine verlässliche Aussage über die Qualität geben zu können.

### 6.5.1 Pokerhände

Ein erster Indikator für die benötigte Anzahl von Spielrunden ist die Anzahl an verschiedenen Handkombinationen und deren Häufigkeit. Der grundlegende Gedanke ist, dass mit der Anzahl der betrachteten Spielrunden mit hoher Wahrscheinlichkeit alle Handkombinationen auftauchen und somit Bestandteil der Auswertung sind. Um die Wahrscheinlichkeiten der einzelnen Blätter zu ermitteln, können die einzelnen möglichen Kombinationen, die das Blatt bilden, betrachtet werden und mit allen möglichen Kombinationen verglichen werden. Die Gesamtmenge an möglichen Kombinationen ergibt sich aus:

$$\binom{52}{7} = 133.784.560$$

Um zu ermitteln, wie viele Kombinationen ein Blatt bilden können, muss betrachtet werden, wie sich dieses Blatt zusammensetzt. Ein Vierling setzt sich beispielsweise aus vier Karten eines bestimmten Werts zusammen, wobei es 13 mögliche Werte gibt. Die anderen drei Karten spielen für die Bildung des Vierlings keine Rolle und

können beliebige aus den verbleibenden 48 Karten sein. Somit ergeben sich:

$$\binom{48}{3} * 13 = 224.848$$

verschiedene Kombinationen für Vierlinge. Wenn nun die Anzahl möglicher Kombinationen ins Verhältnis zu allen möglichen Kombinationen gesetzt wird, ergibt sich die Wahrscheinlichkeit, für einen Vierling:

$$\frac{224848}{133784560} = 0,168\%$$

Dieses Vorgehen kann mit jedem möglichen Blatt durchgeführt werden, die Ergebnisse hiervon finden sich in Tabelle 6.2

Tabelle 6.2 Anzahl der möglichen Kombinationen der verschiedenen Pokerhände und deren Häufigkeit

| **Blatt** | **Kombinationen** | **Wahrscheinlichkeit** |
|---|---|---|
| Royal Flush | 4.324 | 0,003% |
| Straight Flush | 37.260 | 0,028% |
| Vierling | 224.848 | 0,168% |
| Full House | 3.473.184 | 2,596% |
| Flush | 4.047.644 | 3,025% |
| Straight | 6.180.020 | 4,619% |
| Drilling | 6.461.620 | 4,830% |
| Zwei Paare | 31.433.400 | 23,496% |
| Ein Paar | 58.627.800 | 43,823% |
| Höchste Karte | 23.294.460 | 17,412% |
| Total | 133.784.560 | |

Wenn also eine Repräsentation der möglichen Handkombinationen im Vordergrund stehen würde, sollte eine Menge von 100.000 Spielen ausreichen, um eine geeignete Stichprobe zu bilden. Diese Betrachtungsweise lässt jedoch zwei wichtige Aspekte außer Acht: Erstens sind nicht alle Spielkarten, die die Hand bilden, von Anfang an bekannt, sondern sie werden erst im Verlauf des Spiels bekannt. So gibt es

$$\binom{7}{2} * \binom{5}{3} * \binom{2}{1} * \binom{1}{1} = 420$$

verschiedene Kombinationen, in denen die Karten aufgedeckt werden können, und es macht einen großen Unterschied, ob der Spieler bereits mit der ersten Karte ein starkes Blatt hat, oder erst mit der letzten Karte. Dies kann dazu führen, dass Agenten bereits zu früh das Spiel verlassen. Ein weiterer Aspekt, der in dieser Betrachtung ausgelassen wird, sind die Handkarten des Gegners, die ebenfalls für deutlich mehr

Variation sorgen. Insgesamt hilft diese Betrachtungsweise daher nur bedingt für die Abschätzung der notwendingen Spielrundenanzahl.

### 6.5.2 Annäherung

Im ersten Schritt in den Auswertungen wurden 10 Millionen Spielrunden mit den verschiedenen Agenten simuliert und die durchschnittlichen Ergebnisse über die Spielrunden hinweg ermittelt. Je weniger sich die durchschnittlichen Ergebnisse ändern, desto näher sind sie der korrekten Bewertung. Als Vergleichskriterium wurde der maximale Variationskoeffizient der verschiedenen Positionen gewählt, damit die Agenten vergleichbar und unabhängig vom erzielten Gewinn sind. Die Ergebnisse der Auswertung finden sich in Tabelle 6.3 und sie zeigen, dass in der Untersuchung die maximale Abweichung der Ergebnisse im TD-N-Tuple4 Agent ab 1.655.000 Spielerunden maximal 10% vom Endergebnis betrug, wobei dies bei anderen Agenten, wie beim Q-Learn-4, deutlich früher bei 455.000 Spielrunden der Fall war.

Tabelle 6.3 Anzahl der notwendigen Spielrunden (in Tausend) je Agent, damit die maximale Abweichung vom Gesamtdurchschnitt einem bestimmten Wert entspricht.

| Abweichung vom Durchschnitt | TD-N-Tuple-4 | SARSA-4 | TD-N-Tuple-3 | Q-Learn-4 | TDS |
|---|---|---|---|---|---|
| 20% | 165 | 80 | 85 | 80 | 75 |
| 15% | 1,200 | 135 | 85 | 80 | 110 |
| 10% | 1,655 | 150 | 355 | 455 | 210 |
| 9% | 1,890 | 165 | 410 | 1,480 | 210 |
| 8% | 2,145 | 170 | 530 | 2,130 | 355 |
| 7% | 2,210 | 170 | 530 | 2,190 | 355 |
| 6% | 2,275 | 175 | 1,785 | 3,060 | 365 |
| 5% | 2,455 | 175 | 1,955 | 3,415 | 610 |
| 4% | 2,865 | 235 | 2,085 | 3,635 | 640 |
| 3% | 3,760 | 910 | 5,410 | 5,185 | 4,110 |
| 2% | 7,825 | 1,025 | 8 | 7,430 | 5,175 |
| 1% | 8,920 | 7,850 | 9,555 | 8,500 | 9,305 |

Im, in Abbildung 6.4 gezeigten Verlauf, wird deutlich, dass es zu Beginn noch starke Schwankungen gibt, diese aber im Verlauf deutlich zuückgehen und ab etwa 4 Mio. gespielten Spielrunden konstant unter 0.05 bleibt.

### 6.5.3 Streuung

Als zweiter Annäherungsversuch wurde die Größe der Streuung der Ergebnisse bei der Auswertung von Spielrunden betrachtet. Hierbei wurde eine bestimmte Menge an Spielrunden simuliert und die durchschnittlichen Ergebnisse untereinander verglichen. Da in der vorherigen Auswertung der TD-N-Tuple-4 Agent die meisten

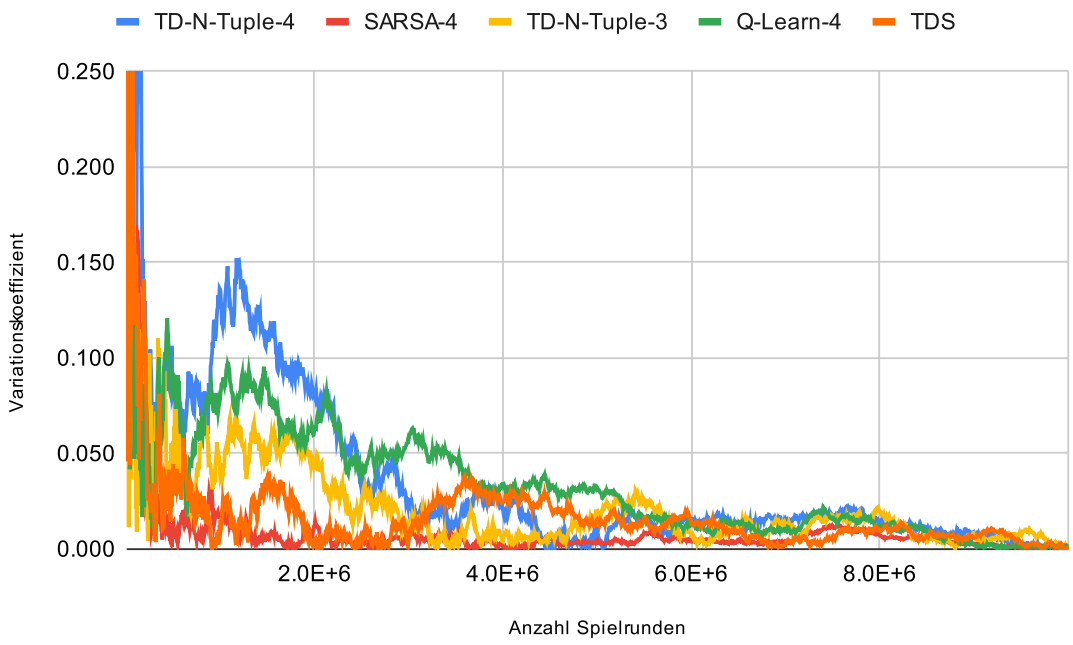


Abbildung 6.4 Variationskoeffizient im Verlauf bei 10 mio. gespielten Runden

Spiele benötigt hat, um nur noch eine geringe Abweichung vom Endergebnis zu haben, wurde dieser als Worst Case im folgenden näher betrachtet. In der Betrachtung wurden die durchschnittlichen Ergebnisse der beiden Positionen zwischen 1.000.000 und 10.000.000 Spielrunden verglichen, wobei für jede Spielrundenanzahl 1.000 Vergleichsspiele betrachtet wurden. Es wird hierbei schnell deutlich, dass die Varianz der durchschnittlichen Punktzahl mit größerer Spielrundenzahl abnimmt. Deutlich wird dies in Abbildung 6.5, in der die Ergebnisse von 1.000.000 Spielrunden mit den Ergebnissen aus 10.000.000 gespielten Runden verglichen werden.

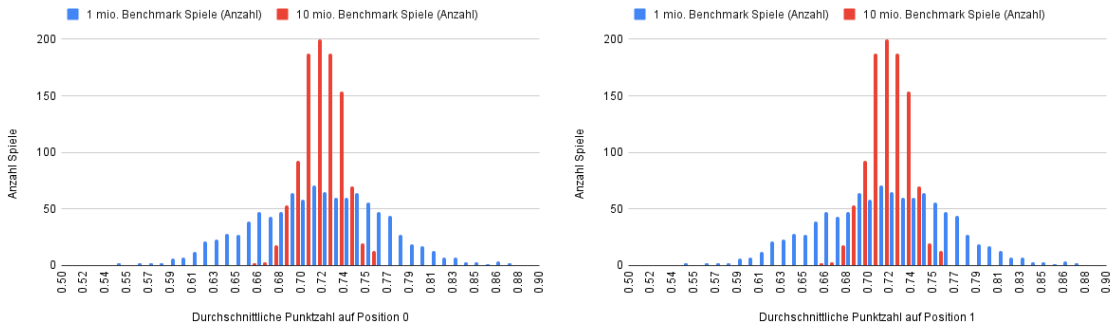


Abbildung 6.5 Vergleich Verteilung der durchschnittlichen Punktzahl bei 10 Mio. und 1 Mio. Spielrunden

Es wird deutlich, dass die Streuung der durchschnittlichen Punktzahl für 1 Mio. Spielrunden relativ hoch ist, mit einer Standardabweichung von etwa 0.054, was etwa 7,565% des Durchschnittswerts entspricht. Wenn davon ausgegangen wird, dass es sich bei der Verteilung der Ergebnisse um eine Normalverteilung handelt, was durch

die entstandenen Histogramme unterstützt wird, lägen etwa 99% aller Messwerte im Bereich $\sigma * 2,576 * 7,565\% = 19,419\%$ also etwa +/-20% des Durchschnittswerts. Im Vergleich dazu beträgt die Standardabweichung bei 10 Mio. gespielten Spielrunden lediglich 0,017, womit 99% der Messwerte nur eine Abweichung von etwa +/-6% um den Durchschnittswert aufweisen.

| Benchmark Spiele | $s_{p0}$ | $s_{p1}$ | $2,576 * s_{p0}$ | $2,576 * s_{p1}$ | $\frac{2,576 * s_{p0}}{\overline{p0}}$ | $\frac{2,576 * s_{p1}}{\overline{p1}}$ |
|---|---|---|---|---|---|---|
| 1.000.000 | 0,054 | 0,076 | 0,140 | 0,195 | 19,5% | 20,5% |
| 2.000.000 | 0,040 | 0,056 | 0,103 | 0,144 | 14,4% | 15,0% |
| 3.000.000 | 0,032 | 0,045 | 0,083 | 0,115 | 11,5% | 12,1% |
| 4.000.000 | 0,028 | 0,039 | 0,072 | 0,100 | 10,0% | 10,5% |
| 5.000.000 | 0,024 | 0,033 | 0,061 | 0,086 | 8,6% | 9,0% |
| 6.000.000 | 0,022 | 0,031 | 0,057 | 0,079 | 7,9% | 8,3% |
| 7.000.000 | 0,020 | 0,027 | 0,051 | 0,071 | 7,1% | 7,4% |
| 8.000.000 | 0,019 | 0,026 | 0,049 | 0,068 | 6,8% | 7,1% |
| 9.000.000 | 0,017 | 0,023 | 0,043 | 0,060 | 6,0% | 6,3% |
| 10.000.000 | 0,017 | 0,024 | 0,043 | 0,061 | 6,1% | 6,3% |

Tabelle 6.4 Standardabweichung und Größe des 99%-Konfidenzintervalls je Menge betrachteter Spiele

Aus den Daten wird deutlich, dass die ursprüngliche Vermutung, dass mehr betrachtete Spiele zu besseren Ergebnissen führen, zutrifft. Für das weitere Vorgehen muss jedoch ein Grenzwert für die betrachteten Spiele festgelegt werden, damit die Ergebnisse möglichst vergleichbar sind. Aus diesem Grund wurde festgelegt, dass ein Konfidenzintervall von etwa +/-10% ausreichend ist. Im konkreten Beispiel, welches in Abbildung 6.6 gezeigt wird, bedeutet dies, dass 99% der Werte bei 4 Mio. Spielrunden zwischen 0,676 und 0,815 liegen. Falls hingegen nur 1 Mio. Spielrunden betrachtet werden, liegen 99% der Werte zwischen 0,522 und 0,912, was einer fast dreimal so großen Spanne an Werten entspricht.

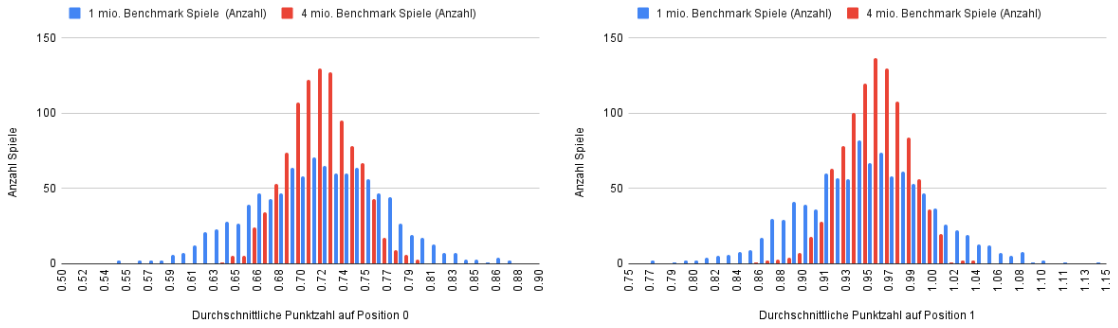


Abbildung 6.6 Vergleich Verteilung der durchschnittlichen Punktzahl bei 4 Mio. und 1 Mio. Spielen

## 6.6 Auswahl des Agenten

Nachdem im vorherigen Schritt ermittelt wurde, wie viele Spielrunden analysiert werden müssen, um vergleichbare Ergebnisse zu erhalten soll nun ein erster Vergleich der Agenten erfolgen. Zielsetzung dieses Vergleichs ist es, den vielversprechendsten der lernenden Agenten zu identifizieren, damit dieser im nächsten Schritt optimiert werden kann. Hierbei wurden auf der einen Seite die unterschiedlichen Agenten, aber auch die unterschiedlichen Tupelklassen miteinander verglichen. Dabei wurden folgende Tupelklassen betrachtet:

- XNTupleFuncsPokerSimple: Bei diesem Ansatz wurden die Karten ähnlich wie beim Feature-Vektor für TDS als ID der Karte abgebildet. So wird zum Beispiel: [♣7, ♠10, ♣8, ♣3, ♥2, ♠$K$, ♦2] dargestellt als: [45, 22, 46, 41, 1, 25, 27]. Da das komplette Array zu groß ist, um als ein Tupel abgebildet zu werden, wurden verschiedene Tupel gebildet.

- XNTupleFuncsPokerAbstract: Bei diesem Ansatz wird das Vorkommen der einzelnen Kartenwerte und die maximale Anzahl Karten in gleicher Farbe betrachtet. So wird zum Beispiel: [♣7, ♠10, ♣8, ♣3, ♥2, ♠$K$, ♦2] dargestellt als: $[2, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 3]$, wobei die ersten 13 Plätze die Anzahl des jeweiligen Werts angeben, und der letzte Wert die Anzahl Karten gleicher Farbe. Da das komplette Array zu groß ist um als ein Tupel abgebildet zu werden, wurden verschiedene Tupel gebildet.

- XNTupleFuncsPokerSimpleRanks: Dieser Ansatz ist ähnlich wie XNTupleFuncsPokerSimple, da auch hier jede Karte abgebildet wird, um jedoch die Komplexität zu verringern, wurde lediglich der Rang der Karte genutzt. So wird zum Beispiel: [♣7, ♠10, ♣8, ♣3, ♥2, ♠$K$, ♦2] dargestellt als: [5, 8, 6, 1, 0, 11, 0]. Trotz der verringerten Komplexität musste auch hier das Array in mehrere Tupel unterteilt werden, wobei jeder Bereich mindestens fünf aufeinanderfolgende Karten abdeckt, damit die Chance besteht einen Straight korrekt zu erkennen.

- XNTupleFuncsPokerSummary: Bei diesem Ansatz wurden die meisten Informationen vorverarbeitet und zusammengefasst. Dabei hat das gebildete Array die Größe: $6 + AnzahlSpieler - 1$. Auf den ersten beiden Positionen werden die Anzahl der Karten gleicher Farben und Anzahl der aufeinanderfolgenden Karten gespeichert, damit hier ein Flush bzw. Straight einfach erkannt werden kann. Position drei gibt den Wert der höchsten Karte für High Card an, Position vier, fünf und sechs den Wert für Paare, Drillinge und Vierlinge. Dabei ist anzumerken, dass nicht der eigentliche Wert des Paares verwendet wird. Die Karten werden in Segemente unterteilt. Diese Segmente erhalten einen Wert

von 1-5, welcher die Stärke darstellt. So wird zum Beispiel ein Paar mit einer 2,3 oder 4 mit der Stärke 1 repräsentiert, und ein Paar 5,6 und 7 mit der Stärke 2. Auf diesem Weg konnten die Informationen so weit heruntergebrochen werden, dass diese in einem einzigen Tupel abgebildet werden können.

Wie in Tabelle 6.5 deutlich wird, schneiden die Agenten mit Tupeln, die den Agenten möglichst viele Informationen in einfacher Form bereitstellen können, besser ab. Eine Vermutung ist, dass durch die seltene Wiederholung von Spielzuständen die Situationen nicht oft genug auftauchen, damit sie gelernt werden können. Hinzukommt, dass durch den Agenten eine Transferleistung notwendig ist. Wenn wir beispielsweise die Tupelklasse XNTupleFuncsPokerSimple betrachten, so zeigen sowohl: [5,17, ...] und [14,40, ...] ein Paar mit Zweien, wobei das eine Paar ♠2, ♥2 ist, und das andere ♣2, ♦2. Dies jedoch aus den gesamten Informationen zu erkennen, ist deutlich schwieriger, verglichen mit dem XNTupleFuncsPokerSummary, der dem Agenten diese Informationen bereits zur Verfügung stellt.

Tabelle 6.5 Durchschnittlicher und maximaler Gewinn aller Agenten je Tupelklasse im Spiel gegen den Benchmark-Agenten

| **Tupelklasse** | ∅ $Gewinn_{P0}$ | ∅ $Gewinn_{P1}$ | Maximum $Gewinn_{P0}$ | Maximum $Gewinn_{P1}$ |
|---|---|---|---|---|
| XNTupleFuncsPokerAbstract | -0.996 | 0.958 | -0.263 | 3.721 |
| XNTupleFuncsPokerSimple | -4.037 | 0.386 | -3.234 | 1.930 |
| XNTupleFuncsPokerSimpleRanks | -3.859 | 1.332 | -1.517 | 2.319 |
| XNTupleFuncsPokerSummary | 0.665 | 1.744 | 1.588 | 3.482 |

Die Ergebnisse zeigen, dass mit XNTupleFuncsPokerSummary sowohl die besten durchschnittlichen Ergebnisse der Agenten, als auch die besten maximalen Ergebnisse erzielt wurden. Deshalb wurde diese Tupelklasse für den weiteren Verlauf als Fokus gewählt.
Neben dem Vergleich der einzelnen Tupelklassen wurde ebenfalls ein Vergleich der unterschiedlichen mit Tupeln lernenden Agenten gemacht.

Tabelle 6.6 Durchschnittlicher und maximaler Gewinn aller Tupelklassen je Agent

| **Tupelklasse** | ∅ $Gewinn_{P0}$ | ∅ $Gewinn_{P1}$ | ∅ $Gewinn$ | Maximum $Gewinn_{P0}$ | Maximum $Gewinn_{P1}$ | ∅-Maximum $Gewinn$ |
|---|---|---|---|---|---|---|
| Qlearn-4 | -2,921 | 1,826 | -0.548 | 0,126 | 3,721 | 1.924 |
| Sarsa-4 | -1,565 | 1,294 | -0.135 | 1,588 | 2,319 | **1.954** |
| TD-Ntuple-3 | -1,955 | 0,632 | -0.662 | 0,503 | 0,778 | 0.641 |
| TD-Ntuple-4 | -1,785 | 0,668 | -0.559 | 0,443 | 0,790 | 0.617 |

Es zeigt sich, dass Sarsa-4 die sowohl durchschnittlich besten Ergebnisse der Agenten mit den gegebenen Tupelklassen erzielt, als auch die besten maximalen Ergebnisse. Aus diesem Grund wurde für die weitere Analyse der Fokus auf Sarsa-4 gelegt.

## 6.7 Konstanz der Ergebnisse

Nachdem mit Sarsa-4 ein vielversprechender Agent identifiziert wurde, konnte im nächsten Schritt mit der Analyse der Auswirkung des Trainings auf den Agenten begonnen werden. Hierbei zeigte sich jedoch eine Auffälligkeit bei der Auswertung der Ergebnisse, die in Tabelle 6.7 deutlich wird. Die Ergebnisse des betrachteten Sarsa-4 Agenten schwankten sehr stark, in den betrachteten Spielen zwischen im Durchschnitt -1,182 und 1,253. Diese Abweichung ist nicht durch zufälle Streuung der Ergebnisse zu erklären, wie in Unterabschnitt 6.5.3 gezeigt wurde. Demnach müssen die erlernten Strategien unterschiedlich stark sein, um die Ergebnisse erklären zu können.

Tabelle 6.7 Durchschnittlicher Gewinn von Sarsa-4 mit 4 Mio. Trainingsspielen und 4 Mio. Vergleichsspielen

| **Agent** | $\varnothing - Gewinn_{P0}$ | $\varnothing - Gewinn_{P0}$ | $\varnothing - Gewinn$ |
|---|---|---|---|
| Sarsa-4 | -0,853 | -1,510 | **-1.182** |
| Sarsa-4 | -0,983 | 2,062 | 0.539 |
| Sarsa-4 | -0,467 | 1,864 | 0.698 |
| Sarsa-4 | 0,418 | 1,693 | 1.055 |
| Sarsa-4 | 0,167 | 1,814 | 0.990 |
| Sarsa-4 | -0,473 | -0,843 | -0.658 |
| Sarsa-4 | 0,584 | 1,921 | **1.253** |
| Sarsa-4 | -0,982 | 1,526 | 0.272 |
| Sarsa-4 | -0,613 | 0,905 | 0.146 |
| Sarsa-4 | -1,416 | 1,832 | 0.208 |

Um die unterschiedliche Qualität der Agenten zu untersuchen, wurde im nächsten Schritt analysiert, wie sich die Spielstärke der Agenten über den Trainingszeitraum entwickelt. Hierbei wurde der Agent abwechselnd erst mit einer gewissen Rundenanzahl trainiert und anschließend gegen den Benchmark Agenten evaluiert. Als Parametereinstellungen wurden dieselben Einstellungen beim Training aller Agenten verwendet:

$\alpha_{init} = \alpha_{final} = 0.001$ $\epsilon_{init} = 0.3$ $\epsilon_{final} = 0$ $\gamma = 1$ $\lambda = 0$

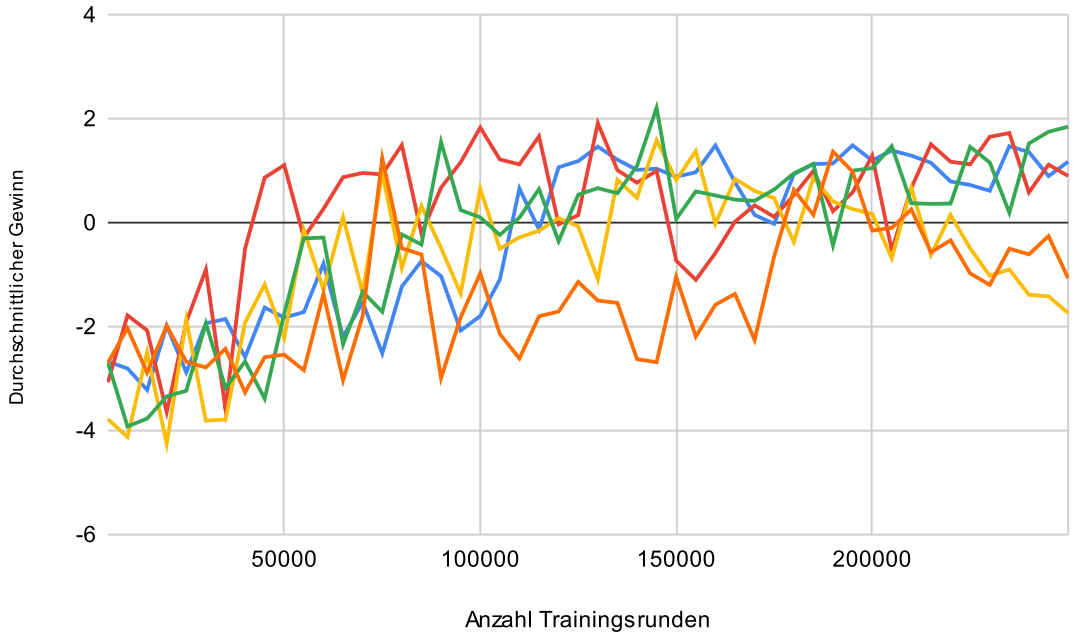


Abbildung 6.7 Vergleich fünf verschiedener SARSA-4 Agenten über 250.000 Trainingsrunden

Abbildung 6.7 zeigt die Qualität der Ergebnisse fünf verschiedener Sarsa-4 Agenten im Verlauf von 250.000 Trainingsrunden. Hierbei zeigt sich, dass die Spielstärke der Agenten stark schwankt. Vergleicht man den *roten* Agenten nach 100.000 Trainingsrunden mit dem *blauen* Agenten ist der *rote* Agent deutlich überlegen mit +1.831 durchschnittlichem Gewinn gegenüber -1.797 beim *blauen* Agenten. Nach etwa 150.000 Trainingsrunden hat sich der *rote* Agent jedoch deutlich verschlechtert und erzielt nur noch einen durchschnittlichen Gewinn von etwa -1.101, während sich der *blaue* Agent deutlich verbessert hat und einen Gewinn von durchschnittlich 0.969 erzielen konnte. Nach den 250.000 Trainingsrunden zeigt sich, dass drei der fünf Agenten gute Ergebnisse erzielen, während die verbleibenden zwei Agenten keinen durchschnittlich positiven Gewinn erreichen.

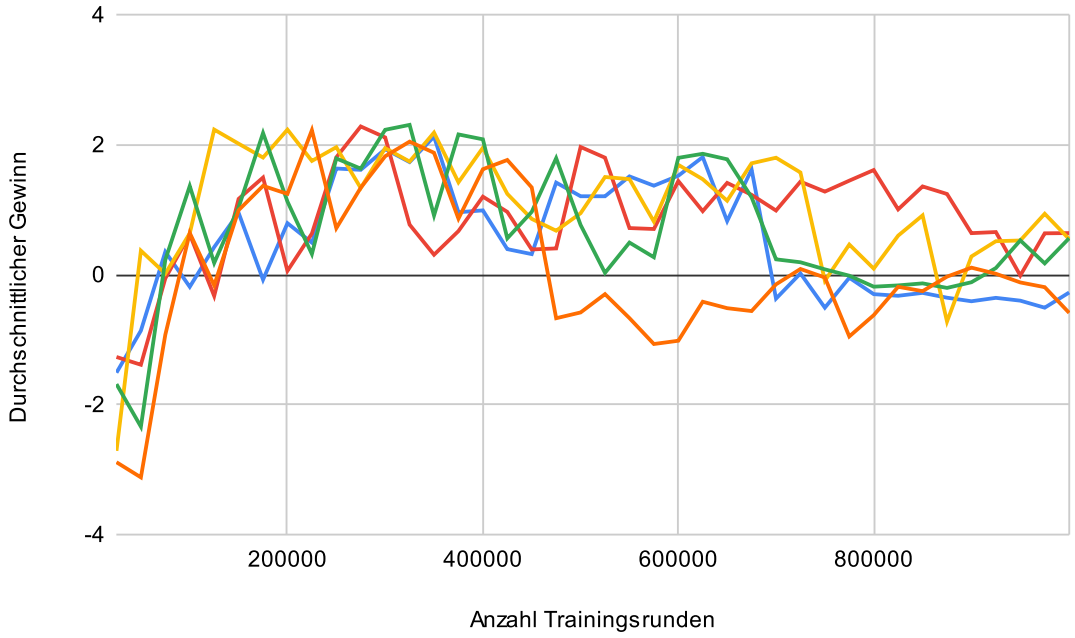


Abbildung 6.8 Vergleich fünf verschiedener Sarsa-4 Agenten über 1 Mio. Trainingsrunden

Abbildung 6.8 zeigt die Qualität der Ergebnisse weiterer fünf Sarsa-4 Agenten im Verlauf von 1 Mio. Trainingsrunden. Die Frage, die hiermit beantwortet werden sollte, ist, ob eine Erhöhung der Anzahl an Trainingsrunden zu konstanteren Ergebnissen führt. Es zeigt sich auch hier, dass sich die Agenten zu Beginn ähnlich entwickeln, bis etwa 300.000 gespielten Runden, anschließend jedoch eine größere Diskrepanz zwischen den Ergebnissen der verschiedenen Agenten entsteht. Besonders auffällig wird diese durch den *orangenen* Agenten, der ab etwa 375.000 gespielten Spielrunden negative Ergebnisse erzielt. Auch nach den 1 Mio. Trainingsrunden konnten nur drei der fünf Agenten positive Ergebnisse erzielen. Im nächsten Schritt wurde die Anzahl der Trainingsrunden weiter erhöht und der Verlauf über 10 Mio. Runden betrachtet, welcher in Abbildung 6.9 gezeigt wird. Auch hier zeigt sich ein ähnliches Verhalten wie bei vorherigen Beobachtungen. Es gibt zum Teil Abschnitte, in denen sich die Agenten ähnlich entwickeln, etwa bei 2 Mio. gespielten Runden erzielen alle Agenten vergleichbare Ergebnisse, jedoch läuft die Stärke der Agenten deutlich auseinander. Auch nach 10 Mio. gespielten Runden sind nur drei der fünf Agenten in der Lage, positive Ergebnisse zu erzielen, wobei der *rote* Agent nach den 10 Mio. Runden das beste Ergebnis erzielen konnte.

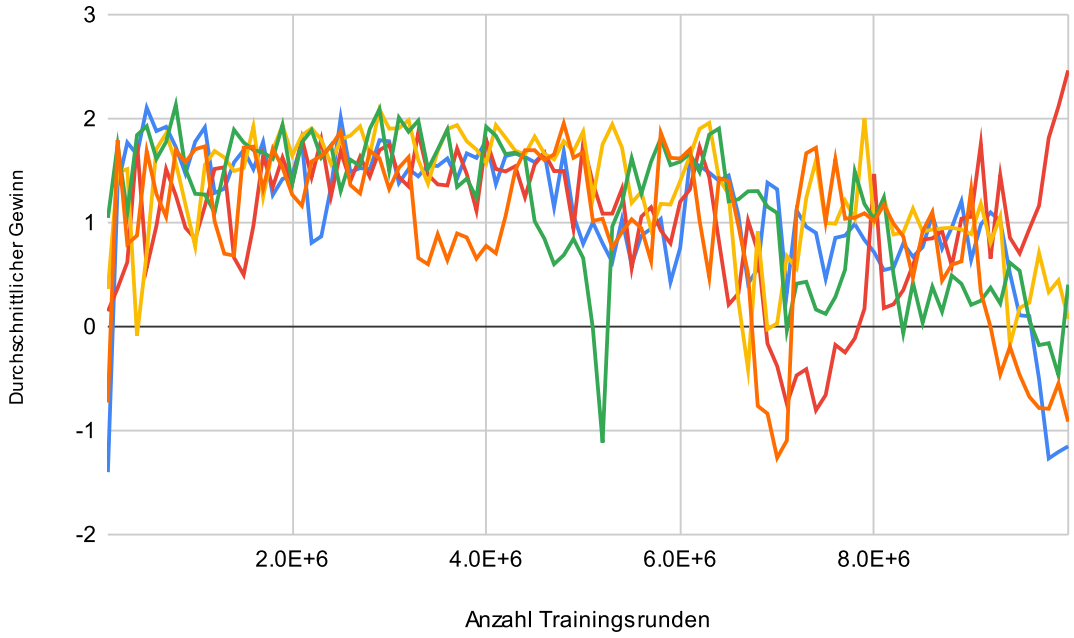


Abbildung 6.9 Vergleich fünf verschiedener Sarsa-4 Agenten über 10 Mio. Trainingsrunden

Insgesamt wird deutlich, dass die Agenten trotz einer großen Anzahl an Trainingsrunden stark unterschiedliche Ergebnisse erzielen. Dies deutet darauf hin, dass die Agenten nicht zu einer bestimmen Strategie konvergieren. Die Herausforderung den Prozess mit einer großen Menge von Agenten zu prüfen und zu untersuchen ist, dass dies ein zeitaufwendiger Prozess ist. Zum einen nimmt das Training der Agenten Zeit in Anspruch, zum anderen benötigt auch die Evaluation der Qualität des Agenten viel Zeit, da viele Runden gespielt werden müssen, um vergleichbare Ergebnisse erzielen zu können.

## 6.8 Training

Nachdem untersucht wurde, wie konstant der Lernerfolg der Agenten ist, soll die Frage beantwortet werden, wie viel Training sinnvoll für einen Agenten ist. Die ursprüngliche Annahme war, dass mehr Training zu besseren Ergebnissen führt. Jedoch zeigten die bisherigen Ergebnisse, dass die Agenten oft ihre beste Leistung bereits frühzeitig erreichen und sich nicht weiter verbessern. Um dies zu untersuchen wurden die Ergebnisse von vier Agenten im Verlauf von 5 Mio. Trainingsrunden betrachtet. Es wird deutlich, dass alle Agenten ihre besten Ergebnisse relativ früh erzielen und sich anschließend verschlechtern, wie in Tabelle 6.8 gezeigt wird.

Tabelle 6.8 Vergleich der besten Ergebnisse der vier Sarsa-4 Agenten mit erzielten Endergebnissen

| Bestes Ergebnis | Trainingsrunden für bestes Ergebnis | Endergebnis |
|---|---|---|
| 2.522 | 375.000 | 1.091 |
| 2.337 | 470.000 | -0.468 |
| 2.261 | 1.210.000 | -0.480 |
| 2.170 | 585.000 | 0.480 |

Noch deutlicher wird diese Beobachtung in Abbildung 6.10. In dem Graphen wird der Mittelwert der Abweichung der vier Agenten von ihrem jeweiligen besten Ergebnis dargestellt. Der Mittelwert wird zu keinem Zeitpunkt 0, da die vier Agenten ihre besten Werte zu unterschiedlichen Zeitpunkten erzielen. Jedoch zeigt sich, dass das durchschnittlich beste Ergebnis nach 625.000 Trainingsrunden erzielt werden konnte. Anschließend nimmt die Abweichung zu und die durchschnittliche Qualität der Agenten sinkt.

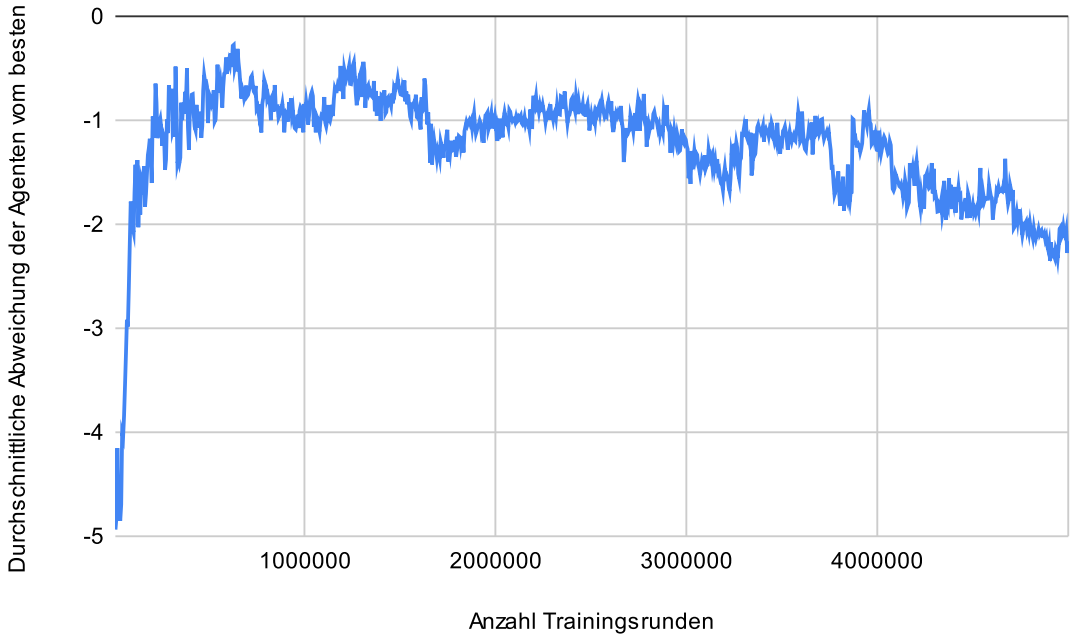


Abbildung 6.10 Durchschnittliche Abweichung der vier untersuchten Agenten von ihrem jeweils besten Ergebnis

Diese Erkenntnis widerspricht der eingehenden Vermutung, dass mehr Training zu besseren Ergebnissen führt. Im Gegenteil fürt es dazu, dass die Ergebnisse schlechter werden. Demnach wäre augenscheinlich das beste Vorgehen, um einen starken Agenten zu erzeugen, viele Agenten mit weniger Trainingsrunden zu trainieren und den besten daraus entstandenen Agenten zu nutzen.

## 6.9 Übersicht der Ergebnisse

Die Tabelle 6.9 zeigt den durchschnittlichen Gewinn der einzelnen Agenten im Spiel gegen den Benchmark Agent bei 4 Mio. betrachteten Spielrunden. Als erstes fällt positiv auf, dass alle Agenten besser, als der zufällig agierende Agent abschneiden. Fast alle Agenten sind in der Lage, einen Gewinn gegen den Benchmark-Agenten zu erzielen. Einzig der TDS-Agent als Startspieler verliert, wobei der Agent den Verlust auf der einen Position durch gute Ergebnisse auf der anderen Position ausgleicht.

Tabelle 6.9 Durchschnittlicher Gewinn der Agenten bei 4 Mio. gespielten Spielrunden gegen den Benchmark-Agenten

| Agent | $\varnothing Gewinn_{Spieler1}$ | $\varnothing Gewinn_{Spieler2}$ | $\varnothing Gewinn$ |
|---|---|---|---|
| Sarsa-4 | **1,588** | 2,319 | **1.954** |
| Qlearn-4 | 0,126 | **3,721** | 1.924 |
| TD-Ntuple-4 | 0,443 | 0,790 | 0.617 |
| TD-Ntuple-3 | 0,503 | 0,778 | 0.641 |
| TDS | -1,037 | 3,240 | 1.102 |
| MC-N | 1,056 | 1,910 | 1.483 |
| MCTSE | 0,160 | 0,865 | 0,513 |
| Random | -4,320 | -2,866 | -3.593 |

Positiv fällt weiter auf, dass die lernenden Agenten zum Teil bessere Ergebnisse als MC-N und MCTSE erzielen. Wobei hierbei bedacht werden muss, dass aufgrund der notwendigen Ressourcen und der zeitintensiven Berechnung der Ergebnisse von MCTSE und MC-N lediglich 100.000 Iterationen verwendet wurden und dennoch nur etwa 200.000 Benchmark Spielrunden betrachtet werden konnten. Überraschend sind die großen Unterschiede zwischen TD-NTuple-4 und TD-NTuple-3 und Sarsa-4 beziehungsweise QLearn-4.

## 6.10 Diskussion

Im Rahmen dieses Kapitels sollte die Frage beantwortet werden, ob die im GBG-Framework implementierten Agenten neben der vereinfachten Variante Kuhn Poker, auch die komplexere Variante Texas Hold'em erlenen können. Für Kuhn Poker existiert eine bekannte optimale Strategie, mit der die Agenten verglichen werden konnten. Außerdem war es möglich, einzelne Spielsituationen zu betrachten und die gefundenen Strategien der einzelnen Agenten zu analysieren und zu bewerten. Leider ist dies, trotz der Einschränkung auf zwei Spieler für Texas Hold'em, aufgrund der deutlich größeren Komplexität nicht möglich. Aus diesem Grund beschränkt sich diese Bewertung vorrangig auf die durchschnittlich erzielte Punktzahl bei einer großen Menge an Spielen gegen den in Abschnitt 6.2 beschriebenen Benchmark Agenten. Als Mindestmaß für das erfolgreiche Spielen wurde ein höhere Spielstärke als die

des zufällig agierenden Agentens festgelegt. Diese Hürde konnte von allen Agenten gemeistert werden und dies mit einem deutlichen Abstand von durchschnittlich 4 Chips pro Spielrunde. Die nächste Stufe in der Spielstärke kann durch das Spiel der hauptsächlich auf statistischen Auswertungen basierenden Agenten wie dem MC-N definiert werden. Dieser Agent hat das Spiel nicht *gelernt*, sondern verlässt sich auf seine Simulationen. Diese Hürde konnte nicht von allen Agenten gemeistert werden, jedoch haben zwei der lernenden Agenten *Sarsa-4* und *QLearn-4* den MC-N Agenten deutlich übertroffen. Auffällig ist dabei, dass die Agenten auf den unterschiedlichen Positionen unterschiedlich stark sind, wobei *QLearn-4* deutlich stärker als 2. Spieler ist. Hervorzuheben ist, dass die TD-N-Tuple Agenten nicht in der Lage waren das Spiel auf demselben Niveau wie Sarsa-4 und QLearn-4, oder zumindest auf dem Niveau von MC-N zu erlernen. Ähnliche Beobachtungen konnten auch von Meißner im Spiel Blackjack während einer Umsetzung im GBG-Framework gemacht werden. Hierbei steht die Vermutung im Raum, dass die Ursache für den ausbleibenden Lernerfolg mit der Episodenlänge und der Mittlung der erwarteten Rewards zusammenhängt. Empirisch konnten Meißner und Konen demonstrieren, dass der TD-N-Tuple 4 Agent nicht in der Lage ist ein vereinfachtes, dem Black Jack Spiel ähnliches, Spiel zu erlernen [Simon Meißner, 2021, S. 38]. Es ist denkbar, dass dies auch die schlechteren Ergebnisse für Texas Hold'em erklärt, jedoch müsste dies weiter untersucht werden.

Alles in allem konnte jedoch gezeigt werden, dass die Agenten im GBG-Framework trotz ihrer allgemeinen Gültigkeit in der Lage sind, das Spiel Texas Hold'em als Zwei-Spieler-Variante zumindest zu einem gewissen Grad zu erlernen.

# 7 Fazit und Ausblick

Zielsetzung der Arbeit war eine Untersuchung der Frage, ob die nicht-spezialisierten Agenten des General Board Games Framework in der Lage sind, das Spiel Poker zu erlernen. Um diese Fragestellung zu beantworten, wurde im ersten Schritt mit Kuhn Poker eine vereinfachte Form des Pokerspiels untersucht. Die Agenten waren nicht in der Lage, die optimale Strategie zu finden, jedoch haben sie gute Ergebnisse erzielen können, welche nur in wenigen Aktionen von der optimalen Strategie abwichen. Anschließend wurde das Spiel Texas Hold'em in einer zwei-Spieler-Variante untersucht. Die Ergebnisse zeigten, dass die lernenden Agenten zumindest teilweise in der Lage waren, das Spiel zu erlernen und bessere Ergebnisse als beispielsweise der MC-N Agent zu erzielen. Da beide untersuchten Varianten von Poker erfolgreich erlernt werden konnten, kann die zugrundeliegende Frage mit einem "ja"beantwortet werden. Jedoch liegt die Spielstärke der Agenten mit Sicherheit unter denen der spezialisierten Agenten. Spezialisierte Agenten bieten den Vorteil, dass diese an die Gegebenheiten des Spiels angepasst und dahingehend optimiert werden können. Darüber hinaus ist es möglich, unterschiedliche Verfahren zu kombinieren und somit die besten Ergebnisse zu erzielen. Teile davon ließen sich vermutlich auch im Rahmen des GBG-Frameworks umsetzen, jedoch würde eine starke Spezialisierung auf ein bestimmtes Spiel dem Grundgedanken des Frameworks widersprechen.

Es konnten bereits verschiedene Erweiterungen des Frameworks im Laufe der Ausarbeitung, in Zusammenarbeit mit Konen und Meißner, umgesetzt werden. Die prägendste Umstellung ist die Etablierung des *Partial States*, der es ermöglicht, unvollständige Informationen im Rahmen des Frameworks abzubilden. Hierbei werden alle, dem Agenten nicht zugänglichen, Informationen aus dem *Game State* entfernt und in einem jeweiligen Endzustand eines simulierten Spiels durch zufällige Werte ergänzt. Eine zusätzliche Erweiterung war die Möglichkeit, Agenten nicht nur auf Basis kompletter Spiele, sondern auch auf Basis einzelner Spielrunden zu bewerten und auch zu trainieren. Besonders im Rahmen von Poker ist dies relevant, da die Simulation kompletter Spiele viel Zeit in Anspruch nimmt und die Komplexität für den lernenden Agenten deutlich erhöht, da sich wesentlich längere Aktionsabläufe ergeben. Diese Erweiterung lässt sich jedoch auf alle Spiele übertragen, deren Spielablauf sich aus mehreren einzelnen Spielrunden zusammensetzt. Eine andere Erweiterung, welche konzeptionell getestet, jedoch nicht vollständig implementiert

wurde, ist die Umsetzung nicht-deterministisch agierender Agenten, beziehungsweise einer Erweiterung der bisherigen Agenten, nicht deterministisch zu handeln. Der Grundgedanke dahinter ist, dass die optimale Strategie in Kuhn Poker vorsieht, dass eine Aktion mit einer bestimmten Wahrscheinlichkeit ausgeführt wird. Dies hat den Vorteil, dass der Gegner größere Schwierigkeiten hat, die eigene Strategie zu ermitteln und es damit nicht möglich ist, die eigene Strategie auszunutzen. Die bisherigen Agenten verfolgen jedoch größtenteils deterministische Strategien, wenn von den Sonderfällen abgesehen wird, in denen mehrere Aktionen gleich gut bewertet sind.

Eine weitere potentielle Erweiterung des Frameworks wäre sicherlich die Implementierung eines Agenten, der auf der Counterfactual Regret Minimization Methodik aufbaut. Hintergrund hierfür ist, dass ein solcher Agent voraussichtlich besser in der Lage sein wird, Spiele, die auf unvollständigen Informationen basieren, zu erlernen. Hierdurch würde man auch einen Agenten implementieren, der nicht deterministisch handelt.

Insgesamt konnte das Framework jedoch sehr gut durch die gegebenen Funktionen unterstützen und hat es ermöglicht, schnell verschiedene Agenten zu analysieren und zu vergleichen.

# Literaturverzeichnis


[Allis u a., 1994] Allis, L. V. u a. (1994). *Searching for solutions in games and artificial intelligence*. Dissertation, Maastricht University.

[Ball, 2015] Ball, P. (2015). Game theorists crack poker. `https://doi.org/10.1038/nature.2015.16683`.

[Billingham, 2017] Billingham, J. (2017). Simplified three player Kuhn poker. `https://arxiv.org/abs/1704.08124`.

[Bledsoe und Browning, 1959] Bledsoe, W. W. und Browning, I. (1959). Pattern recognition and reading by machine. In Heart, F. E., Herausgeber, *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference on - IRE-AIEE-ACM '59 (Eastern)*, Seiten 225–232, New York, New York, USA. ACM Press.

[Bowling et al., 2015a] Bowling, M., Burch, N., Johanson, M., und Tammelin, O. (2015a). Heads-up limit hold'em poker is solved. *Science (New York, N.Y.)*, 347(6218):145–149.

[Bowling et al., 2015b] Bowling, M., Burch, N., Johanson, M., und Tammelin, O. (2015b). Heads-up limit hold'em poker is solved - Supplemantary Materials. `https://science.sciencemag.org/content/suppl/2015/01/07/347.6218.145.DC1`.

[Brown und Sandholm, 2019] Brown, N. und Sandholm, T. (2019). Superhuman AI for multiplayer poker. *Science (New York, N.Y.)*, 365(6456):885–890.

[Browne et al., 2012] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., und Colton, S. (2012). A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

[Burch et al., 2016] Burch, N., Schmid, M., Moravčík, M., und Bowling, M. (2016). AIVAT: A New Variance Reduction Technique for Agent Evaluation in Imperfect Information Games. `https://arxiv.org/abs/1612.06915`.

[Chaslot, 2010] Chaslot, G. M. J.-B. (2010). *Monte-Carlo Tree Search*. Dissertation, Maastricht University.

[Chen und Kaneko, 2019] Chen, C. und Kaneko, T. (112019). Acquiring Strategies for the Board Game Geister by Regret Minimization. In *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, Seiten 1–6. IEEE.

[Gibney, 2017] Gibney, E. (2017). How rival bots battled their way to poker supremacy. *Nature*, 543:160–161.

[Heaven, 2019] Heaven, D. (2019). No limit: AI poker bot is first to beat professionals at multiplayer game. *Nature*, 571:307–308.

[Hoehn et al., 2005] Hoehn, B., Southey, F., Holte, R. C., und Bulitko, V. (2005). Effective short-term opponent exploitation in simplified poker. In Jacobstein, N. und Porter, B., Herausgeber, *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, Seiten 783–788. AAAI, Menko Park.

[Holler et al., 2019] Holler, M. J., Illing, G., und Napel, S. (2019). *Einführung in die Spieltheorie*. Lehrbuch. Springer Gabler, Berlin, 8. Auflage.

[Konen, 2019] Konen, W. (2019). General Board Game Playing for Education and Research in Generic AI Game Learning. `https://arxiv.org/pdf/1907.06508`.

[Konen, 2020] Konen, W. (2020). The GBG Class Interface Tutorial V2.2: General Board Game Playing and Learning. `http://www.gm.fh-koeln.de/ciopwebpub/Konen20d.d/TR-GBG.pdf`.

[Konen und Bartz-Beielstein, 2008] Konen, W. und Bartz-Beielstein, T. (2008). Reinforcement Learning: Insights from Interesting Failures in Parameter Selection. In Rudolph, G., Jansen, T., Beume, N., Lucas, S., und Poloni, C., Herausgeber, *Parallel Problem Solving from Nature – PPSN X*, Seiten 478–487, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Kuhn, 1951] Kuhn, H. W. (1951). *Contributions to the Theory of Games (AM-24)*, Band 1 in *Annals of Mathematics Studies*. Princeton University Press, Princeton, United States.

[Lanctot et al., 2009] Lanctot, M., Waugh, K., Zinkevich, M., und Bowling, M. (2009). Monte Carlo Sampling for Regret Minimization in Extensive Games. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, NIPS'09, Seiten 1078–1086, Red Hook, NY, USA. Curran Associates Inc.

[Lucas, 2008a] Lucas, S. M. (2008a). Computational intelligence and games: Challenges and opportunities. *International Journal of Automation and Computing*, 5(1):45–57.

[Lucas, 2008b] Lucas, S. M. (2008b). Learning to Play Othello with N-Tuple Systems. *Australian Journal of Intelligent Information Processing*, 4(4):1–20.

[Mańdziuk und Świechowski, 2012] Mańdziuk, J. und Świechowski, M. (2012). Generic Heuristic Approach to General Game Playing. In Bieliková, M., Herausgeber, *SOFSEM 2012: theory and practice of computer science*, Lecture notes in computer science, Seiten 649–660. Springer, Heidelberg.

[Moravčík et al., 2017] Moravčík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., und Bowling, M. (2017). DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker. *Science (New York, N.Y.)*, 356(6337):508–513.

[Neller, 2013] Neller, Todd W.and Lanctot, M. (2013). An introduction to counterfactual regret minimization. `http://cs.gettysburg.edu/~tneller/modelai/2013/cfr/cfr.pdf`.

[Newborn, 1997] Newborn, M. (1997). *Kasparov versus Deep Blue: Computer chess comes of age*. Springer, New York and Berlin and Heidelberg and Barcelona and Budapest and Hong Kong and London and Milan and Paris and Santa Clara and Singapore and Tokyo.

[Osborne und Rubinstein, 1994] Osborne, M. J. und Rubinstein, A. (1994). *A course in game theory*. MIT Press, Cambridge, Massachusetts.

[Pollice et al., 2008] Pollice, G., Heineman, G. T., und Selkow, S. (2008). *Algorithms in a nutshell*. In a nutshell. O'Reilly Media, Sebastopol, Calif.

[Risk und Szafron, 2010] Risk, N. A. und Szafron, D. (2010). Using Counterfactual Regret Minimization to Create Competitive Multiplayer Poker Agents. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, Seiten 159–166, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Russell und Norvig, 2010] Russell, S. J. und Norvig, P. (2010). *Artificial intelligence: A modern approach*. Prentice-Hall series in artificial intelligence. Prentice-Hall, Upper Saddle River, NJ, 3. ed.. Auflage.

[Schadd, 2009] Schadd, F. C. (2009). *Monte-Carlo Search Techniques in the Modern Board Game Thurn and Taxis*. Master thesis, Maastricht University, Maastricht.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., und Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489.

[Simon Meißner, 2021] Simon Meißner (2021). *Untersuchung des Spiel- und Lernerfolgs künstlicher Intelligenzen für ein nichtdeterministisches Spiel mit imperfekten Informationen*. Bachelor thesis, Technische Hochschule Köln, Köln.

[Sutton und Barto, 1998] Sutton, R. S. und Barto, A. G. (1998). *Reinforcement learning: An introduction*. A Bradford book. MIT Press, Cambridge, Mass. [u.a.], 5. Auflage.

[Winter, 2019] Winter, S. (2019). *Grundzüge der Spieltheorie*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Zeh, 2021] Zeh, T. (2021). *Implementierung von Poker im General Board Game Framework*. Projektarbeit, Technische Hochschule Köln, Köln.

[Zhou et al., 2018] Zhou, Y., Ren, T., Li, J., Yan, D., und Zhu, J. (2018). Lazy-CFR: fast and near optimal regret minimization for extensive games with imperfect information. `http://arxiv.org/pdf/1810.04433v3`.

[Zinkevich et al., 2007] Zinkevich, M., Johanson, M., Bowling, M., und Piccione, C. (2007). Regret Minimization in Games with Incomplete Information. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, Seiten 1729–1736, Red Hook, NY, USA. Curran Associates Inc.

# Erklärung

Ich versichere an Eides Statt, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt beziehungsweise in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Köln, 30. Juli 2021

Ort, Datum

Tim Zeh

Rechtsverbindliche Unterschrift