

Author(en): Jan Aeberli & Jürg Schleutermann
Betreuer: Prof. Dr. Markus Stolze
Experte: Thomas Kälin
Gegenleser: Ivan Buetler

Hochschule für Technik Rapperswil
Abteilung Informatik
Herbstsemester 2018



Bachelorarbeit

Praktische Anwendung
spieltheoretischer Ansätze für den
Schieber Jass

Simulation trifft Tradition



Inhaltsverzeichnis

1	Abstract	3
2	Management Summary	4
3	Projektübersicht	5
3.1	Zweck und Ziel	6
3.2	Zeitliche Vorgaben	6
3.3	Lieferumfang	6
4	Anforderungen	7
4.1	Aktoren	7
4.2	Schieber Jass Bot	7
4.3	Benchmark Umgebung	10
5	Hintergrund und Analyse	14
5.1	Spielbeschreibung	14
5.2	Spieltheorie	20
5.3	Kombinatorik	28
5.4	Kartenbuchhaltung	31
6	Architektur	39
6.1	Packages	40
6.2	Cluster Deployment	41
7	Benchmarking	44
7.1	Testablauf	45
7.2	Unabhängigkeit	49
7.3	Parametrisierung	50
8	Lösungsansätze	52
8.1	Spielzustand	52
8.2	Counterfactual Regret Minimization	54
8.3	Monte Carlo Counterfactual Regret Minimization	59
8.4	Monte Carlo Tree Search	64
9	Resultate und Auswertungen	74
9.1	Bedeutung der Trumpfwahl	74
9.2	Counterfactual Regret Minimization	77
9.3	Monte Carlo Counterfactual Regret Minimization	82
9.4	Monte Carlo Tree Search	85
9.5	Hypothesentests	88
9.6	Erfahrungsbericht	90
10	Schlussfolgerungen und Ausblick	92
10.1	Resultate	92
10.2	Erweiterungsmöglichkeiten	92

Anhang	94
A Aufgabestellung	94
B Erfüllung der Anforderungen	97
C Lieferumfang	100
D Software und Libraries	101
E Rechtliche Hinweise	103
F Glossar	104
G Verzeichnisse	110
G.1 Abbildungsverzeichnis	110
G.2 Tabellenverzeichnis	111
G.3 Quellenverzeichnis	113

1 Abstract

Im Jahr 2017 initiierte die Firma Zühlke einen Wettbewerb, bei dem die Teilnehmer eigene Schieber Jass Bots entwickeln und gegeneinander antreten lassen konnten. Der Gewinner Bot dieses Wettbewerbs musste am Ende gegen ein Team von menschlichen Spielern bestehen. Da diese aber triumphierten, hat die Hochschule für Technik in Rapperswil beschlossen weiter an der Entwicklung von Jass Bots zu arbeiten. Dieser Forschungsarbeit entsprangen bisher zwei weitere Jass Bots, welche auf reinen Deep Learning Ansätzen basierten. Keinem der beiden Bots ist es aber gelungen den Gewinner Bot des Jass Wettbewerbs zu besiegen. Im Rahmen dieser Arbeit soll untersucht werden, inwiefern sich spieltheoretische Ansätze für den Schieber Jass eigenen und ob sich damit ein Bot entwickelt lässt, welcher den Gewinner Bot des Wettbewerbs schlagen kann.

In einer ersten Phase wurden vielversprechende Lösungsansätze aus der Spieltheorie identifiziert und der Schieber Jass im Detail analysiert. In einer zweiten Phase ist für jeden der Ansätze ein Jass Bot entwickelt worden. Insgesamt sind im Laufe der Arbeit drei Bots entstanden. Der erste Bot arbeitet mit Counter Factual Regret Minimization, kann aber aufgrund der Komplexität des Spiels nur in den beiden letzten Spielzügen eine Entscheidung treffen. Der zweite Bot stellt eine Erweiterung des ersten Ansatzes um eine Monte Carlo Komponente dar und ermöglicht dem Bot dadurch ein ganzes Spiel zu bestreiten. Der dritte Bot arbeitet mit einer Monte Carlo Tree Search, um den bestmöglichen Zug zu finden und kann ebenfalls in jeder Spielsituation Entscheidungen treffen. Damit die Bots verglichen werden können, wurde zudem eine Art virtueller Prüfstand entwickelt. Mit Hilfe dieses Prüfstands konnten in einer dritten Phase Experimente durchgeführt und die Resultate statistisch ausgewertet werden.

Die Resultate der Experimente zeigen, dass durch den Counter Factual Regret Minimization Algorithmus bereits eine kleine Verbesserung erreicht wurde. Die Monte Carlo Variante des Ansatzes konnte aber nicht die gewünschten Ergebnisse liefern und schnitt schlechter ab als der Gewinner Bot des Wettbewerbs. Die besten Resultate lieferte mit Abstand der Monte Carlo Tree Search Algorithmus. Mit Hilfe dieses Ansatzes konnte ein Bot entwickelt werden, welcher signifikant besser spielt. Ob der Bot allerdings auch gegen menschliche Spieler gewinnt, konnte im Rahmen dieser Arbeit nicht abschliessend geklärt werden. Stichproben haben jedoch gezeigt, dass Spiele gegen den Bot zumindest unterhaltsam sind.

2 Management Summary

Ausgangslage

Dank der Entwicklungen in der Computertechnik konnten in den letzten Jahren enorme Fortschritte im Bereich der künstlichen Intelligenz (KI) verzeichnet werden. Deepmind - ein Tochterunternehmen von Google - sorgte in letzter Zeit immer wieder für Schlagzeilen mit Veröffentlichungen ihrer KI Projekte. So ist es ihren selbstlernenden Systemen beispielsweise gelungen, die besten "Go" Spieler der Welt zu besiegen. Die KI meistert die Komplexität des Spiels durch Kombination von Ansätzen aus dem Bereich des maschinellen Lernens sowie der Spieltheorie. Der Schieber Jass erfreut sich in der Schweiz schon seit jeher grosser Beliebtheit. Diese Arbeit soll deshalb evaluieren, inwiefern sich spieltheoretische Ansätze für das Jass Spiel adaptieren lassen.

Vorgehen, Technologie

Im Rahmen dieser Arbeit wurden verschiedene spieltheoretische Ansätze identifiziert. Die vielversprechendsten Kandidaten wurden anschliessend genauer untersucht und selber umgesetzt. Für jeden Lösungsansatz wurde eine Jass KI in der Programmiersprache Python entwickelt. Zum Vergleich wurden die Jass KIs in unabhängigen Experimenten gegen eine regelbasierte Jass KI ins Rennen geschickt. Die Resultate sowie die gewonnen Erkenntnisse sind in der Dokumentation dieser Bachelorarbeit festgehalten.

Ergebnisse

Das Resultat dieser Bachelorarbeit umfasst drei KIs für den Schieber Jass sowie eine statistische Auswertung ihres Spielniveaus. Die Auswertung belegt, dass eine der drei programmierten KIs signifikant mehr Spiele gegen die regelbasierte Jass KI gewinnen konnte. Zur Qualität ihrer Entscheidungen bei der Interaktion mit menschlichen Spielern kann keine abschliessende Aussage getroffen werden. Es konnte gezeigt werden, dass sich spieltheoretische Ansätze auf das Jass Spiel adaptieren lassen und Forschungsarbeit in diese Richtung zu weiteren Erfolgen führen könnte.

Ausblick

Maschinelles Lernen und künstliche Intelligenz haben in letzter Zeit sowohl in der Forschung als auch in der Industrie enorm an Bedeutung gewonnen. Sie ermöglichen neue Lösungsansätze für allerlei Probleme des Alltags. Die Resultate dieser Arbeit zeigen anhand eines konkreten Beispiels, wie sich Ansätze aus der Spieltheorie auf komplexe Entscheidungs- und Planungsprobleme anwenden lassen. Zusätzliches Einbringen von Expertenwissen sowie Techniken aus dem Bereich des maschinellen Lernens sind vielversprechende Kandidaten hinsichtlich Erweiterung und Verbesserung der analysierten Ansätze.

3 Projektübersicht

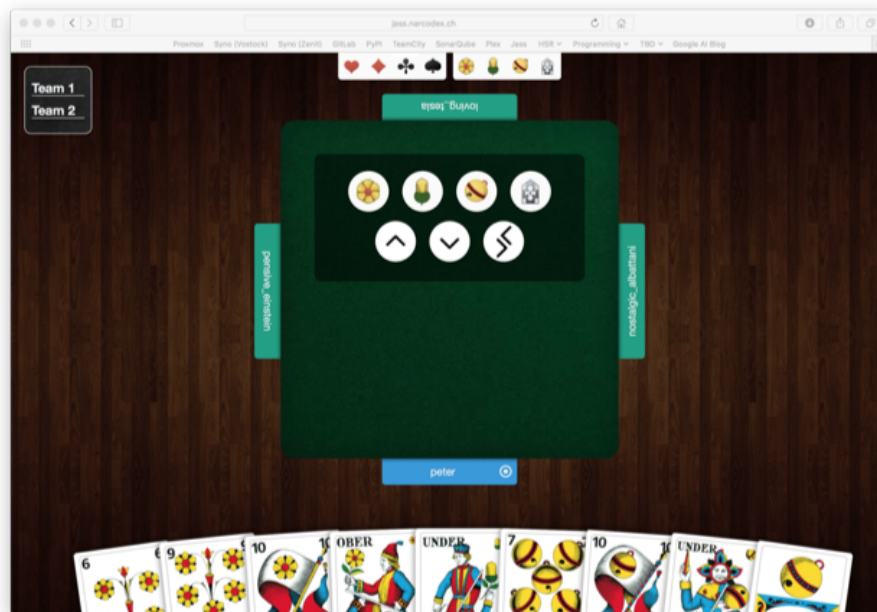


Abbildung 1: Screenshot der Jass Umgebung von Zühlke

Jassen ist das Schweizer Kartenspiel schlecht hin. Mit dem Voranschreiten der Digitalisierung wurde auch das Jass Spiel revolutioniert. Inzwischen gibt es eine Vielzahl von Online Plattformen, auf denen ambitionierte Jasser sich miteinander messen können. Eine davon ist die Plattform von Swisslos, auf der dieses Jahr die 2. Schweizer Schieber Jass Meisterschaft stattfindet [1]. Mit den neuen technischen Möglichkeiten drängt sich immer mehr die Frage auf, ob es für einen Computer möglich ist, besser zu jassen als menschliche Spieler. Dieser Frage wollte auch die Firma Zühlke Engineering AG weiter nachgehen.

Im Frühjahr 2017 starteten sie daher einen Wettbewerb, in welchem die Teilnehmer Bots für den Schieber Jass entwickeln und diese gegeneinander antreten lassen konnten. Dabei spielten jeweils zwei Instanzen des gleichen Bot zusammen. Als Abschluss musste sich der Siegerbot gegen zwei menschliche Spieler zur Wehr setzen. Die HSR Studenten Sandro Feller und Daniel Latzer haben sich mit ihrem Jass Bot “Mighty Hans” erfolgreich gegen die Konkurrenz durchgesetzt. Dem Bot war es aber noch nicht möglich gegen die menschlichen Kontrahenten zu gewinnen. Daher hat die HSR beschlossen, weiter Forschungsarbeit in diesem Bereich zu leisten. [2]

Im Rahmen einer Bachelorarbeit versuchten die Studenten Andreas Eder und Pascal Kistler mithilfe von Supervised Learning und Reinforcement Learning Ansätzen einen Prototyp zu entwickeln, welcher auf dem Niveau eines menschlichen Spielers jassen kann. Der entwickelte Prototyp spielte zwar etwa auf dem Niveau eines Gelegenheitsspielers, konnte sich aber nicht gegen den Gewinner der Jass Challenge (im Folgenden Challenge Bot genannt) durchsetzen. [3]

Der Masterstudent Samuel Kurath forschte weiter im Bereich des Deep Learning. Mithilfe von Reinforcement Learning Ansätzen schaffte er es einen Bot zu entwickeln, welcher sich erfolgreich gegen Random Bots und Greedy Bots durchsetzen konnte. Gegen den Challenge Bot konnte aber auch er bisher noch nicht gewinnen. Zudem ging aus seiner Arbeit die Jass Spielumgebung “Pyschieber” hervor, welche das Ausprobieren von neuen Lösungsansätzen erheblich erleichtert (mehr dazu in Kapitel 7). [4]

Aufgrund der gewonnen Erkenntnisse drängt sich die Frage auf, ob reine Machine Learning Ansätze überhaupt zu guten Resultaten führen können für den Schieber Jass.

3.1 Zweck und Ziel

Diese Bachelorarbeit ist ein Forschungsprojekt der Hochschule für Technik in Rapperswil. Im Rahmen des Projekts soll untersucht werden, inwiefern sich Konzepte der Spieltheorie für die Entwicklung eines Schieber Jass Bots eignen. Der Bericht richtet sich dabei an Ingenieure, welche sich vertieft mit diesen Themen auseinander setzen möchten. Ziel des Projekts ist es, vielversprechende Ansätze zu identifizieren und jeweils einen Prototypen dafür zu erstellen. Um zu überprüfen, wie gut sich die Prototypen im Spiel schlagen, sollen sie mit der bisher besten Implementation (Challenge Bot) verglichen werden.

Mehr Informationen zum Zweck und Ziel der Arbeit können in der offiziellen Aufgabenstellung in Anhang A nachgelesen werden.

3.2 Zeitliche Vorgaben

Die Studenten erhalten für die Bachelorarbeit jeweils 12 ECTS Punkte. Ein ECTS Punkt entspricht ca. 30 Stunden geleistetem Arbeitsaufwand. Somit stehen für die gesamte Bachelorarbeit **720 Stunden** zur Verfügung. Die verlangten Arbeitsstunden sind innerhalb der Zeitspanne vom 17.09.2018 bis zum 21.12.2018 zu leisten.

3.3 Lieferumfang

Der Lieferumfang des Projekts umfasst:

- Abstract und Management Summary
- Quellcode der Experimente
- Technischer Bericht
- Projektjournal

Detaillierte Angaben zum Lieferumfang sind in Anhang C aufgeführt.

4 Anforderungen

Die Anforderungen wurden im Laufe des Projekts erarbeitet und analysiert. Sie teilen sich grundsätzlich in **funktionale** und **nicht funktionale** Anforderungen auf. Da sich dieses Projekt hauptsächlich mit der Evaluation neuer Ansätze befasst, werden die funktionalen Anforderungen an die Implementation kurz gehalten und als Brief Use Cases [5, S. 49] formuliert. Nichtfunktionale Anforderungen wurden Charakteristiken für Softwarequalität nach ISO/IEC 25010:2011 [6] zugeordnet. Die Überprüfung hinsichtlich Abdeckung der funktionalen und nichtfunktionalen Anforderungen wird über ein einfaches Testprotokoll festgehalten (siehe Anhang B). Auf aufwendige Testszenarien wurde verzichtet, da es sich bei den Entwicklungen um Prototypen handelt.

4.1 Aktoren

ID	Aktor	Ziele
A-01	Jass Spieler	Der Jass Spieler möchte eine möglichst gute Jass Erfahrung. Er will gefordert werden und Spass am Spiel haben.
A-02	Entwickler	Der Entwickler möchte seine Ansätze validieren und verifizieren können, um sicherzustellen dass sein Ansatz funktioniert. Um Experimente durchführen zu können braucht er eine flexible und performante Umgebung.

Tabelle 1: Übersicht über die Aktoren und ihre Ziele

4.2 Schieber Jass Bot

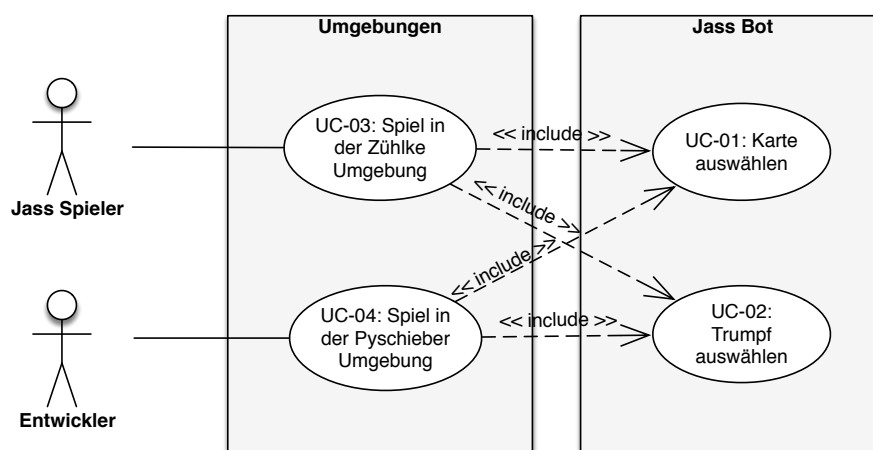


Abbildung 2: Use Cases für die Jass Bots

Kontext

Im Kontext eines Programmierwettbewerbs der Firma Zühlke [2] ist 2017 neben verschiedenen Jass Bots auch eine grafische Spielumgebung für den Browser [7] entstanden. Zusätzlich zu einem normalen Spielmodus gibt es die Möglichkeit, Bots beim Schieber Jass gegeneinander antreten zu lassen. Durch die Client Server Architektur der Umgebung wird zwar Flexibilität geboten, ihr Aufbau gestaltet sich jedoch komplexer. Die Kommunikation der Bots erfolgt über eine Netzwerkschnittstelle mit dem spieleleitenden Server. Daher ist diese Umgebung ungeeignet, wenn man eine Vielzahl von Spielsituationen simulieren will. Die grafische Oberfläche bietet sich allerdings für Demonstrationen an, oder um Spiele zwischen menschlichen Spielern und Bots auszutragen.

Der HSR Masterstudent Samuel Kurath hat für seine Projektarbeit eine alternative Spielumgebung in Python [8] - nachfolgend als Pyschieber bezeichnet - entwickelt. Die Umgebung bietet zwar keine grafische Oberfläche, macht es Entwicklern jedoch umso einfacher, eigene Jass Bots zu erstellen und gegeneinander antreten zu lassen. Die Umgebung wurde für die Entwicklung solcher Bots massgeschneidert. Spielsituationen können mit sehr wenig Einrichtungsaufwand effizient simuliert werden. Python wird als Programmiersprache im Bereich des maschinellen Lernens und künstlicher Intelligenz oft eingesetzt, somit gibt es auch eine Vielzahl von Bibliotheken, die einem mit entsprechender Funktionalität beim Lösen der eigenen Problemstellung unterstützen.

Ein Jass Bot sollte optimalerweise auf beiden Plattformen funktionieren und an einem Spiel teilnehmen können, um von den verschiedenen Vorzügen zu profitieren.

Funktionale Anforderungen

UC-01: Karte auswählen

Als Jass Spieler erwarte ich, dass der Jass Bot in jeder Situation eine Karte auswählen und ausspielen kann. Unerlaubte Karten sollen nie ausgewählt werden.

UC-02: Trumpf auswählen

Als Jass Spieler erwarte ich, dass der Jass Bot alle Trumpfmöglichkeiten kennt und eine davon auswählt.

UC-03: Spiel in der Zühlke Umgebung

Als Jass Spieler möchte ich den Jass Bot in der grafischen Spielumgebung der Firma Zühlke nutzen können. Dabei soll es keine Rolle spielen, ob er als mein Partner oder mein Gegner antritt.

UC-04: Spiel in der Pyschieber Umgebung

Als Entwickler möchte ich den Jass Bot in der Pyschieber Umgebung gegen andere Bots

antreten lassen und die Resultate dieser Spiele auswerten können.

Nichtfunktionale Anforderungen

ID	Charakteristik	Anforderung
NFR-01	Performance Efficiency: Time Behaviour, Resource Utilization	Der Bot soll bei Ausführung auf einem handelsüblichen Computer in einem Spiel nicht mehr als 10 Sekunden Zeit benötigen, bis er eine Entscheidung trifft. Ansonsten wird der Spielfluss zu stark beeinträchtigt. Dies soll mit der durchschnittlichen Rechenzeit pro Algorithmus gezeigt werden.
NFR-02	Functional Suitability: Functional Completeness, Functional Correctness, Functional Appropriateness	Der Bot soll das Spiel meistern und statistisch signifikant besser spielen als seine virtuellen Konkurrenten. Als Referenzimplementation gilt der bereits erwähnte Challenge Bot. Die Verbesserung soll mit einem Exact Binomial Test gezeigt werden.
NFR-03	Usability: Appropriateness Recognizability	Der Bot soll sich in Spielrunden mit mehrheitlich menschlichen Spielern integrieren können. Falls ein Spieler fehlt, nimmt der Bot als würdiger Vertreter seinen Platz ein, damit das Spiel trotzdem stattfinden kann. Der Mensch möchte vom Jass Bot unterhalten werden. Dies soll mittels Praxistest gezeigt werden.
NFR-04	Maintainability: Modularity, Reusability, Analysability, Modifiability, Testability	Der Quellcode soll leserlich geschrieben und wo nötig kommentiert sein. Der Coding Standard PEP8 für Python [9] soll eingehalten werden. Dies soll durch Code Reviews und statische Analysen überprüft werden.

Tabelle 2: Nichtfunktionale Anforderungen an die Jass Bots nach [6]

4.3 Benchmark Umgebung

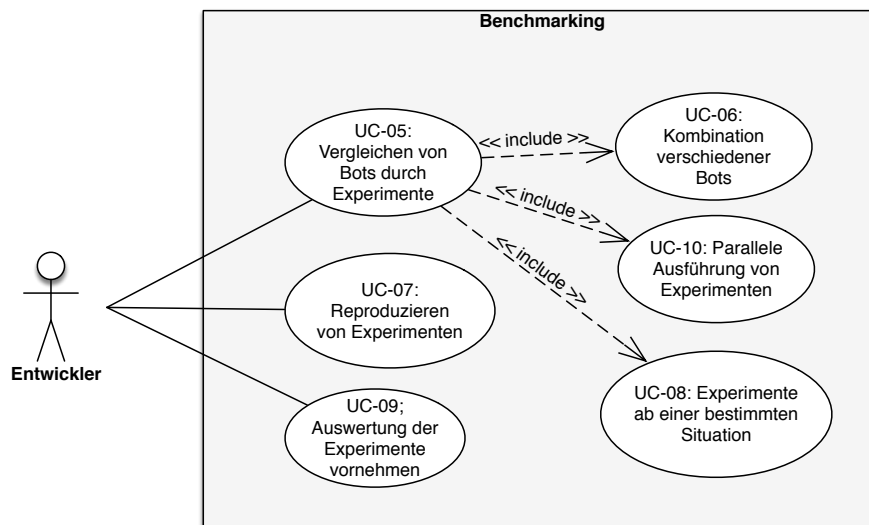


Abbildung 3: Use Cases für die Benchmark Umgebung

Kontext

Beim Entwickeln eines Computerprogramms stellt sich die Frage, wie man seinen Code testen kann. Einerseits braucht es eine Form der **Verifikation**, welche den Code auf Korrektheit prüft. Andererseits braucht es **Validierung** um zu prüfen, ob das entwickelte System das tut, was die Interesseneigner von ihm erwarten. In der Wirtschaft handelt es sich dabei typischerweise um Personen, die das Projekt mit finanziellen Mitteln versorgen.

Grundsätzlich handelt es sich bei einem Jass Bot ebenfalls um ein Programm, allerdings mit einer sehr speziellen Aufgabe. Gängige Mittel zur Verifikation wie beispielsweise automatisierte Unit Tests können dabei helfen, interne Hilfsfunktionen des Bots auf Korrektheit zu prüfen. Ein mögliches Beispiel dafür wäre eine Funktion, die der Bot nutzt um in einer bestimmten Spielsituation alle möglichen Kartenverteilungen aufzuzählen. Die wichtigsten Hilfsfunktionen wurden während dieser Bachelorarbeit mit Unit Tests getestet. Das gibt dem Entwickler nicht nur ein besseres Gefühl bei seiner Arbeit, sondern hilft auch bei der Fehlersuche, wenn ein Bot abnormales Verhalten an den Tag legt.

Hinsichtlich Validierung wird es in diesem Fall etwas komplizierter. Normalerweise würde man sich mit den Interesseneigner zusammensetzen, um das entstandene Produkt gemeinsam zu beurteilen. Doch wie soll das bei einem Jass Bot funktionieren? Im Laufe der Arbeit hat sich herausgestellt, dass es zwei unterschiedliche Erwartungshaltungen an den Bot gibt:

1. Konkurrenzfähigkeit (vgl. NFR-02 in Kapitel 4.2)
2. Attraktivität und Unterhaltung (vgl. NFR-03 in Kapitel 4.2)

Ziel der Benchmark Umgebung ist es, sich mit der ersten Erwartung auseinander zu setzen. Auf der Suche nach einer Möglichkeit, das Spielniveau eines Bots mit dem anderer Bots zu

vergleichen, ist eine umfangreiche Testumgebung entstanden. Sie ermöglicht die Validierung von Jass Bots über viele verschiedene Spielsituationen hinweg und gibt damit Antworten auf Fragen kompetitiver Natur.

Funktionale Anforderungen

UC-05: Vergleichen von Bots durch Experimente

Als Entwickler möchte ich Experimente mit Bots durchführen können. Die Resultate müssen vergleichbar sein, so dass eine Aussage darüber getroffen werden kann, wer besser gespielt hat.

UC-06: Kombination verschiedener Bots

Als Entwickler möchte ich in einem Experiment verschiedene Kombinationen von Bots spielen lassen.

UC-07: Reproduzieren von Experimenten

Als Entwickler möchte ich einmal durchgeführte Experimente reproduzieren können. Die wiederholten Experimente sollen also die gleiche Ausgangssituation besitzen.

UC-08: Experimente ab einer bestimmten Situation

Als Entwickler möchte ich ein Experiment ab einer bestimmten Situation im Spiel durchführen können. Die Bots sollen das Spiel ab dieser Situation bis zum Spielende fertig spielen können.

UC-09: Auswertung der Experimente vornehmen

Als Entwickler möchte ich die durchgeführten Experimente auswerten können. Dazu gehören Metriken wie z.B. die erreichte Punktzahl und die Anzahl der Siege. Diese geben Auskunft darüber, wie sich die jeweiligen Teams bzw. Bots im Spiel geschlagen haben. Die Resultate sollen als Excel Datei zur Verfügung gestellt werden.

UC-10: Parallele Ausführung von Experimenten

Als Entwickler möchte ich Experimente parallel ausführen können, um die Ausführungszeit zu reduzieren und somit innert nützlicher Frist Resultate zu erhalten.

Nichtfunktionale Anforderungen

ID	Charakteristik	Anforderung
NFR-05	Maintainability: Modularity, Reusability, Analysability, Modifiability, Testability	Der Quellcode soll leserlich geschrieben und wo nötig kommentiert sein. Der Coding Standard PEP8 für Python [9] soll eingehalten werden. Dies soll durch Code Reviews und statische Analysen überprüft werden.

Tabelle 3: Nichtfunktionale Anforderungen an die Benchmark Umgebung nach [6]

Abgrenzung

Hinweis: Spiel gegen menschliche Spieler

Es wäre auch wünschenswert, dass der Bot besser spielt als menschliche Spieler. Um einen Beweis erbringen zu können, müsste eine Benutzerstudie durchgeführt werden. Man müsste dazu über einen längeren Zeitraum hinweg, oder im Rahmen einer grösseren Veranstaltung mit Testspielern messen, wie menschliche Spieler das Zusammenspiel bzw. das Spiel gegen die Jass Bots empfinden und wie gut sie abschneiden. Für diese Bachelorarbeit war es leider nicht möglich, eine solche Studie aufzusetzen. Es wurden allerdings ein paar Testspiele “von Hand” durchgeführt. Die dazugehörigen Erfahrungsberichte sind in Kapitel 9 aufzufinden.

Die folgende Tabelle zeigt, welche Charakteristiken bewusst ausgeschlossen wurden:

Charakteristik	Begründung
Performance Efficiency: Capacity	Diese Eigenschaften sind von der Spielumgebung zu erfüllen, in der die Jass Bots eingesetzt werden.
Compatibility: Co-existence, Interoperability	Irrelevant, da die Jass Bots in genau zwei vorgegebenen Spielumgebungen eingesetzt werden.
Usability: Learnability, Operability, User Error Protection, User Interface Aesthetics, Accessibility	Diese Eigenschaften sind von der Spielumgebung zu erfüllen, in der die Jass Bots eingesetzt werden.
Reliability: Maturity, Availability, Fault Tolerance, Recoverability	Wird relevant, wenn die Jass Bots im grossen Stil zum Einsatz kommen, bspw. auf einer Online Plattform.
Security: Confidentiality, Integrity, Non-repudiation, Authenticity, Accountability	Diese Eigenschaften sind von der Spielumgebung zu erfüllen, in der die Jass Bots eingesetzt werden.

Portability: Adaptability, Installability, Re- placeability Wird primär durch die Spielumgebungen vorgegeben, in der die Jass Bots integriert werden.

Tabelle 4: Ausgeschlossene Charakteristiken nach [6]

5 Hintergrund und Analyse

5.1 Spielbeschreibung

Jassen ist ein traditionelles Schweizer Kartenspiel. Gespielt wird, abhängig von der Region, mit französischem oder mit deutschschweizerischem Kartenblatt. Im weiteren Verlauf dieses Berichts wird für Beispiele und Illustrationen Bezug auf das deutschschweizerische Kartenblatt genommen. Die Karten lassen sich allerdings sehr leicht auf das französische Blatt übersetzen, siehe Tabelle 5.

Deutschschweizer Blatt		Französisches Blatt
 Eichel	↔	 Schaufel / Pik
 Rose	↔	 Kreuz
 Schelle	↔	 Herz
 Schilte	↔	 Ecken / Karo

Tabelle 5: Übersetzung zwischen deutschen und französischen Karten (nach [10])

Das Kartendeck umfasst 36 Karten in 4 verschiedenen Farben. Von jeder Farbe gibt es 9 Spielkarten. Eine Übersicht aller Karten zeigt Tabelle 6.

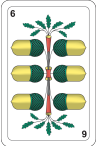
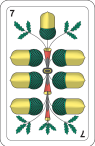
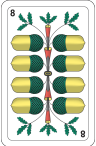
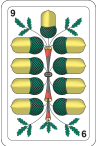
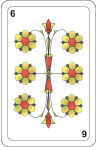
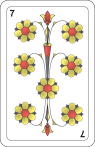
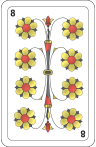
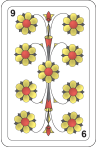
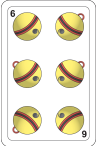
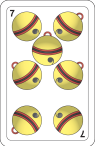
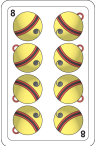
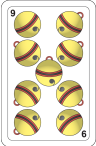
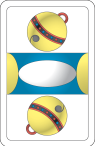
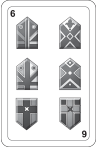
6er	7er	8er	9er	Banner	Under	Ober	König	Ass
								
								
								
								

Tabelle 6: Spielkarten für das Jass Spiel

Innerhalb der Schweiz findet man eine Vielzahl an Spielvarianten und Spezialregeln für das Jass Spiel. Diese Bachelorarbeit befasst sich ausschliesslich mit einer der populärsten Varianten, dem sogenannten **Schieber Jass**. Der Spielmechanismus wird in diesem Kapitel kurz zusammengefasst, um dem Leser das nötige Hintergrundwissen für nachfolgende Kapitel zu vermitteln. Es soll aber in keinsten Weise als vollständige Anleitung verstanden werden. Die Inhalte nehmen mehrheitlich Bezug auf den Schweizer Jassführer “Stöck Wys Stich” [10].

1 Aufstellung

Beim Schieber Jass sitzen 4 Spieler an einem Tisch. Sich gegenüber sitzende Personen bilden dabei ein Team, wie dargestellt in Abbildung 4. Es gewinnt zum Schluss nicht ein einzelner Spieler, sondern eines der beiden Teams. Man spricht deshalb auch von einem Partnerjass.

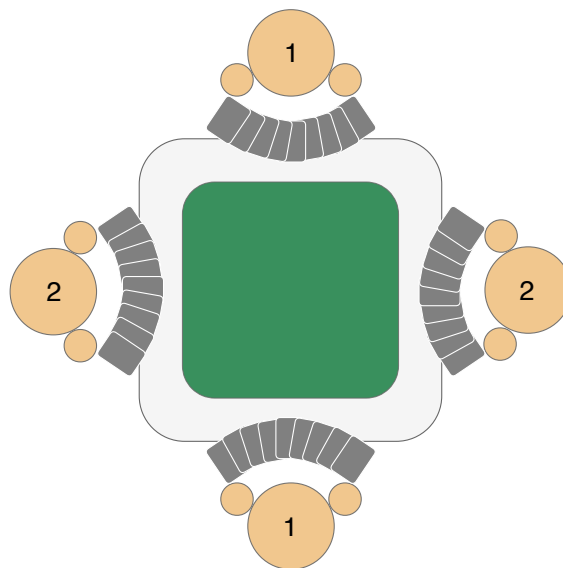


Abbildung 4: Spieleraufstellung beim Schieber Jass

Zu Beginn wird einer der Spieler als **Kartengeber** bestimmt. Das Kartendeck wird vom Kartengeber gemischt und anschliessend auf die 4 Spieler am Tisch aufgeteilt. Jeder der Spieler erhält somit 9 zufällige Handkarten. Die Spieler müssen ihre eigenen Karten vor den anderen Spielern verdeckt halten.

Hinweis: Zusammenspiel mit Spielpartner

Das Zusammenspiel mit dem Spielpartner im gleichen Team ist ein wichtiger Aspekt dieses Spiels. Grundsätzlich ist es den Spielern untersagt, auf direktem Wege Informationen zu den Karten auf ihrer Hand auszutauschen. Bei Gesprächen mit menschlichen Spielern hat sich allerdings gezeigt, dass eine solche Kommunikation oft auf indirektem Wege stattfindet. Die Spielpartner geben sich Signale, indem sie in bestimmten Situationen bestimmte Karten ausspielen. Damit möchten sie beispielsweise signalisieren, dass sie von einer bestimmten Farbe keine Karten mehr auf der Hand haben. Solche Spielmuster sind typischerweise anzutreffen bei Spielpartnern, die häufig zusammenspielen.

2 Start

Der **Startspieler** sitzt zur rechten Hand des Kartengebers. Ihm fällt eine der wichtigsten Entscheidungen im Spiel zu, die sogenannte **Trumpfwahl**. Damit beeinflusst er die Rangordnung und den Punktwert der einzelnen Karten. Tabelle 7 zeigt die Auswahlmöglichkeiten.

	Eichel schlägt alle anderen Farben ("Eichlä")
	Rose schlägt alle anderen Farben ("Rosä")
	Schelle schlägt alle anderen Farben ("Schälle")
	Schilte schlägt alle anderen Farben ("Schiltä")
	Hohe Karte schlägt tiefe Karte ("Obenabe")
	Tiefe Karte schlägt hohe Karte ("Undenufe")
	Spielpartner muss wählen ("Gschobe")

Tabelle 7: Auswahlmöglichkeiten bei der Trumpfwahl

Die Trumpfwahl wird basierend auf den eigenen Handkarten vorgenommen. Es gilt dabei den Nutzen der eigenen Karten zu maximieren. So wählt man beispielsweise eine Farbe als Trumpf, von der man besonders viele bzw. sehr wertvolle Karten auf der Hand hat. Besitzt man sehr tiefe Karten, dann entscheidet man sich eher für "Undenufe" als Trumpf. Erlaubt die eigene Hand keine eindeutige Trumpfwahl, dann sollte die Wahl dem Spielpartner überlassen werden. Natürlich in der Hoffnung, dass dieser bessere Karten auf der Hand hält.

Hinweis: Problematik der Trumpfwahl

Im Verlauf dieser Arbeit hat sich gezeigt, wie entscheidend die Trumpfwahl für den Ausgang eines Spiels sein kann. Aus zeitlichen Gründen war es leider nicht mehr möglich, eine konkrete Lösung dafür auszuarbeiten. Denkbar wäre ein regelbasierter Ansatz, wie er beispielsweise in [10, S. 119] beschrieben wird. Zudem gab es bereits Versuche, diese Entscheidung einem neuronalen Netz zu überlassen, nachdem dieses mittels Daten aus Spielaufzeichnungen trainiert wurde. [3]

Jeder Spieler hat beim Ausspielen seiner ersten Karte theoretisch die Möglichkeit, spezielle Kartenkombinationen in seiner Hand - wie beispielsweise aufeinanderfolgende Karten der gleichen Farbe - auszuweisen. Dieser Vorgang wird als **Weis** bezeichnet und gibt den Spielern die Chance, sich zusätzliche Punkte für das eigene Team zu sichern. Um die Problemstellung

einzugrenzen wurde dieser Aspekt des Spiels weggelassen, weshalb an der Stelle nicht genauer darauf eingegangen wird.

3 Stich

Nach der Trumpfwahl spielt der Startspieler seine erste Karte aus. Dies unabhängig davon, ob die Trumpfwahl durch ihn, oder durch seinen Spielpartner erfolgt ist. Gespielt wird im Gegenuhrzeigersinn, wobei die Spieler pro Runde jeweils nacheinander eine Karte auf den Tisch legen. Eine Runde wird im nachfolgenden auch als **Stich** bezeichnet.

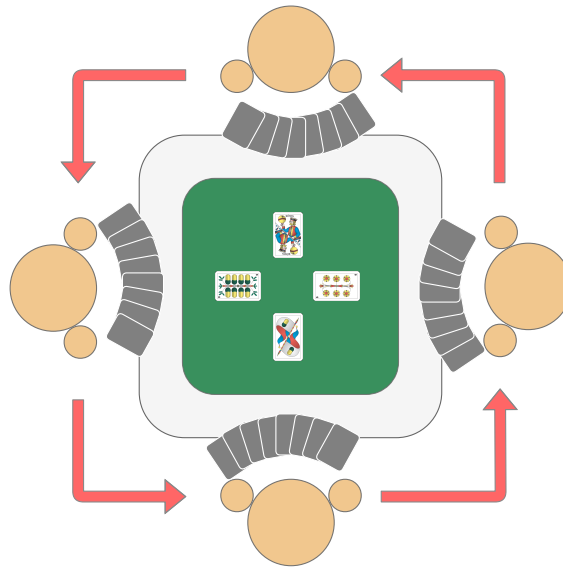


Abbildung 5: Ausführung eines Stiches beim Schieber Jass

Mit der ersten Karte auf dem Tisch wird gleichzeitig die vorgegebene Farbe für den aktuellen Stich festgelegt. Die nachfolgenden Spieler müssen eine Karte der gleichen Farbe ausspielen, sofern sie Karten der vorgegebenen Farbe auf der Hand haben. Anderenfalls dürfen sie eine beliebige Karte ausspielen. Sofern bei der Trumpfwahl eine Farbe gewählt wurde, darf jederzeit auch eine Karte in Trumpffarbe ausgespielt werden.

Ein Spezialfall bildet die Under Karte in Trumpffarbe. Selbst wenn Trumpffarbe in einem Stich durch die erste Karte vorgegeben wurde, muss die Under Karte von ihrem Besitzer nicht zwingend ausgespielt werden. Die Karte dient sozusagen als "Joker", den man bis zu einem günstigen Zeitpunkt auf der Hand lassen kann.



Tabelle 8: Under Karte in Trumpffarbe dient als Joker

Sobald 4 Karten auf dem Tisch liegen wird der Stich ausgewertet. Anhand der Rangordnung wird bestimmt, welches davon die stärkste Karte ist. Man spricht hierbei auch von **stechen**, wenn eine Karte durch eine Karte höherer Rangordnung im Stich geschlagen wird. Der Spieler der stärksten Karte gewinnt alle 4 Karten auf dem Tisch. Gewonnene Karten legen die Spieler verdeckt vor sich auf einen Stapel oder Haufen, sie dürfen bis zum Ende des Spiels nicht mehr betrachtet werden.

Bei der **Rangordnung** kann grundsätzlich zwischen zwei Trumpfarten unterschieden werden. Wird eine bestimmte Farbe als Trumpf gewählt, dann gilt diese Farbe als **Trumpffarbe** und die restlichen 3 Farben als **Nebenfarben**. Karten in Trumpffarbe stechen alle Karten in Nebenfarbe. Die Rangordnung innerhalb der Farben kann Tabelle 9 entnommen werden. Bei Trumpf “Obenabe” und “Undenufe” wird die **Ass Karte** bzw. die **6er Karte** zur stärksten Karte erklärt. Die Rangordnung innerhalb der Farben ist dann absteigend bzw. aufsteigend.

Trumpffarbe	Under > 9er > Ass > König > Ober > Banner > 8er > 7er > 6er
Nebenfarbe	Ass > König > Ober > Under > Banner > 9er > 8er > 7er > 6er
“Obenabe”	Ass > König > Ober > Under > Banner > 9er > 8er > 7er > 6er
“Undenufe”	6er > 7er > 8er > 9er > Banner > Under > Ober > König > Ass

Tabelle 9: Rangordnung der Karten in Abhängigkeit von der Trumpfwahl

Ein weiterer Spezialfall ist das sogenannte **Untertrumpfen**. Voraussetzung dafür ist, dass die erste Karte im Stich eine Nebenfarbe hat. Versucht nun ein Spieler durch Ausspielen einer Karte in Trumpffarbe den Stich für sich zu entscheiden, dann dürfen die nachfolgenden Spieler keine schwächere Karte in Trumpffarbe mehr ausspielen. Ausgenommen davon ist die Situation, in der einer der nachfolgenden Spieler nur noch Karten in Trumpffarbe auf der Hand hat.

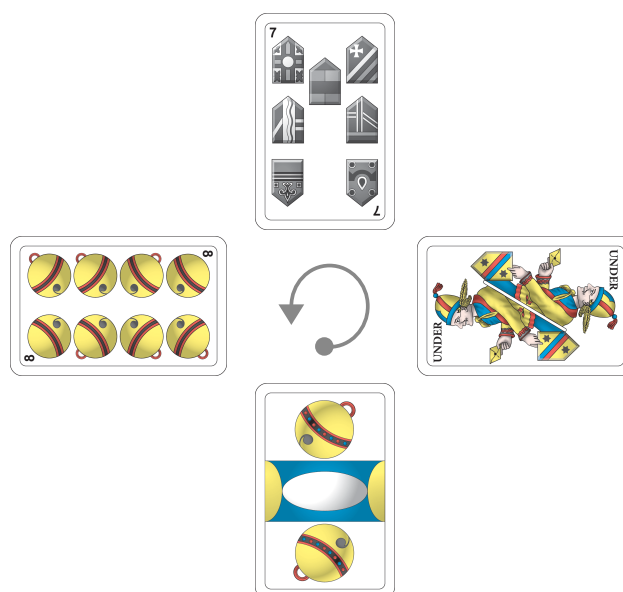


Abbildung 6: Beispielsituation für Untertrumpfen (Beispiel aus [11])

Abbildung 6 zeigt ein mögliches Beispiel für das Untertrumpfen, unter der Annahme, dass Schilte als Trumpffarbe gewählt wurde. Mit Ausspielen der Ass Karte wird Schelle als Farbe vorgegeben durch den Startspieler. Der zweite Spieler entscheidet diesen Stich klar für sich durch Ausspielen der Under Karte in Trumpffarbe. Der dritte Spieler untertrumpft, da er eine schwächere Karte in Trumpffarbe ausspielt.

Hinweis: Spezielle Karten in Trumpffarbe

Aufgrund ihrer speziellen Bedeutung werden die Under bzw. die 9er Karte in Trumpffarbe umgangssprachlich auch als “Buur” bzw. “Näll” bezeichnet.

Der Gewinner des aktuellen Sticks ist gleichzeitig der **Startspieler** für den darauffolgenden Stich. Dieser Schritt wird wiederholt, bis alle 36 Handkarten ausgespielt wurden. Insgesamt umfasst ein **Spiel** somit **9 Stiche**.

4 Auswertung

Bei Spielende zählt jeder der Spieler die Punkte der Karten, die er innerhalb der vergangenen 9 Stiche gewonnen hat. Unabhängig von der Trumpfwahl gibt es maximal 157 Punkte, die man in einem Spiel erzielen kann. 5 davon sind Bonuspunkte für den Spieler, welcher den **letzten Stich** im Spiel für sich entscheiden konnte.

$$152 \text{ Kartenpunkte} + 5 \text{ Punkte für letzten Stich} = \text{Total } 157 \text{ Punkte}$$

Analog zur Rangordnung ist die Punkteverteilung über die 36 Karten hinweg ebenfalls von der Trumpfwahl abhängig. Tabelle 10 zeigt die Punktwerte der einzelnen Karten.

	Ass	König	Ober	Under	Banner	9er	8er	7er	6er
Trumpffarbe	11	4	3	20	10	14	0	0	0
Nebenfarbe	11	4	3	2	10	0	0	0	0
“Obenabe”	11	4	3	2	10	0	8	0	0
“Undenufe”	0	4	3	2	10	0	8	0	11

Tabelle 10: Punktwerte der Karten in Abhängigkeit von der Trumpfwahl

Erzielte Punkte werden mit denen des jeweiligen Spielpartners zusammengezählt. Die Schritte 1 bis 4 werden nun so lange wiederholt, bis eines der Teams das zuvor ausgemachte Punktemaximum erreicht hat. Eine **Partie** wird wahlweise auf 1000, 2500 oder auf 5000 Punkte gespielt. Als zusätzliches strategisches Element kommt bei der Auswertung ein **Multiplikator** hinzu, welcher mit den erzielten Punkten verrechnet wird. Er gilt für jeden der Spieler am

Tisch gleichermassen. Die Höhe des Multiplikators ist abhängig von der Trumpfwahl und der maximalen Punktzahl, die es im Rahmen der Partie zu erzielen gilt.

						
	Eichel	Rose	Schelle	Schilte	“Obenabe”	“Undenufe”
Auf 1000 Punkte	1x	1x	1x	1x	1x	1x
Auf 2500 Punkte	1x	1x	2x	2x	3x	3x
Auf 5000 Punkte	7x	3x	2x	4x	5x	6x

Tabelle 11: Multiplikator für Punkte in Abhängigkeit von der Trumpfwahl

Tabelle 11 zeigt die möglichen Multiplikatoren auf. Beispielsweise zählen in einer Partie auf 2500 Punkte alle erzielten Punkte eines einzelnen Spiels doppelt, wenn Schelle in diesem Spiel als Trumpffarbe gewählt wurde.

Hinweis: Punkte, Multiplikator und Trumpfwahl

Der Multiplikator kann beispielsweise helfen, wenn man als Startspieler die Trumpfwahl hat und das eigene Team hinsichtlich erzielter Punkte im Rückstand ist. So kann es vorteilhaft sein, sich für einen Trumpf zu entscheiden, welcher einen hohen Multiplikator mit sich bringt. Mit etwas Kartenglück können selbst grössere Punktdifferenzen mit einem einzigen Spiel wieder ausgeglichen werden. Der Multiplikator bzw. der Punktestand kann somit neben den eigenen Handkarten als zusätzliche Entscheidungsgrundlage für die Trumpfwahl dienen. Mehr Risiko für mehr Punkte.

5.2 Spieltheorie

In den folgenden Kapiteln werden algorithmische Ansätze präsentiert, welche auf Basis der Spieltheorie versuchen optimale Züge in einem Schieber Jass zu finden. Daher ist es notwendig ein grundlegendes Wissen über die Spieltheorie zu vermitteln. Ziel dieses Kapitels ist es, dem Leser die nötigen Werkzeuge an die Hand zu geben, um die kommenden Algorithmen verstehen zu können.

Hinweis: Vorwissen

Keiner der Autoren hatte Vorwissen im Bereich der Spieltheorie. Alles vermittelte Wissen wurde im Laufe der Arbeit gewonnen und erhebt keinen Anspruch auf Vollständigkeit.

Die Spieltheorie befasst sich nicht nur mit Gesellschaftsspielen. Es geht dabei vielmehr um die Analyse von teilweise komplexen Entscheidungsproblemen. Meist ist das Ziel, eine optimale

Lösung für eine zu treffende Entscheidung zu finden. Die Spieltheorie geht also der Frage nach, welche Entscheidung man treffen soll bei einer Vielzahl von Optionen und zu berücksichtigenden Einflussfaktoren. Entscheidungen müssen überall getroffen werden, weshalb Spieltheorie unter anderem in Bereichen wie Wirtschaft oder Militär zur Anwendung kommt. [12, S. 5–11].

Hinweis: Begriffe in Englisch

Im Folgenden werden meist die englischen Fachbegriffe verwendet, da ein Grossteil der Literatur zur Spieltheorie in Englisch verfasst ist. Eine Übersetzung würde vermutlich nur wenig zum Verständnis beitragen.

Allgemeine Begriffe

Game (Spiel)

Ein Game im Sinne der Spieltheorie stellt eine Menge von Entscheidungsproblemen dar, welche durch die Entscheidungen der Teilnehmer am Schluss zu einem gewissen Ausgang führen.

Player (Spieler)

Ein Player ist ein Teilnehmer an einem Game. Er hat im Game die Möglichkeit Entscheidungen zu treffen und die anderen Player dadurch zu beeinflussen.

Action (Aktion)

Bei einer Action handelt es sich um eine Entscheidung, welche zu einem bestimmten Zeitpunkt des Games getroffen wird [12, S. 15]. Die möglichen Actions, welche einem Player zur Verfügung stehen, sind dabei oft vom Zustand des Games abhängig. Im Falle eines Kartenspiels würde eine Action z.B. dem Ausspielen einer Karte entsprechen.

Strategy (Strategie)

Im Laufe eines Games müssen immer wieder Entscheidungen (Actions) getroffen werden. Das Ziel eines Players ist es, für jede einzelne Situation in die er geraten kann, einen Plan zu besitzen, wie er agieren soll. Er trifft also Entscheidungen anhand seiner gewählten Strategy. Man unterscheidet grundsätzlich zwischen *Pure Strategies* und *Mixed Strategies*. Eine Pure Strategy bestimmt klar welche Aktion in welcher Situation zu wählen ist und ist daher auch immer deterministisch. Eine Mixed Strategy entspricht einer Wahrscheinlichkeitsverteilung über die möglichen Aktionen. Eine Mixed Strategy ist daher nicht deterministisch. [12, S. 15]

Strategy Profile (Strategie Profil)

Da es in einem Spiel nicht nur eine Entscheidung gibt, sondern unter Umständen ganz viele Entscheidungen getroffen werden müssen, braucht der Player für jede dieser Entscheidungen eine Strategy. Alle Strategies zusammengefasst nennt man ein Strategy Profile [13, S. 6–7].

Reward / Payoff / Utility

Die Auswahl von Actions durch Anwendung von Strategies in einem Game führt zu einem ganz bestimmten Ausgang. Ein solcher Ausgang hat für jeden Player Konsequenzen,

sei es positiver oder negativer Art. Die Quantifizierung eines Ausgangs wird als Reward bezeichnet. Je nach Literatur findet man auch die Begriffe Payoff oder Utility. [12, S. 15]

Complete Information

Bei einem Game mit **Complete Information** sind alle wichtigen Aspekte des Games allgemein bekannt. Folgende Eigenschaften müssen erfüllt sein, damit man von einem Spiel mit Complete Information sprechen kann: [14, S. 1–2]

- Jeder weiss, wer die anderen Player sind
- Jeder kennt seine möglichen Actions
- Jeder kennt die möglichen Rewards
- Jeder weiss, wie Reward und Actions zusammenhängen
- Jeder weiss, was die anderen Player über einen wissen und umgekehrt

Vereinfacht gesagt gibt es also für das Game klar definierte Regeln und diese Regeln sind allen Playern bekannt. Die meisten gängigen Games (wie z.B. Schach) sind Complete Information Games. **Incomplete Information** Games findet man z.B. bei der Analyse von ökonomischen Prozessen. [14, S. 1–2]

Perfect Information

Von einem **Perfect Information Game** spricht man, wenn es in einem Game keinerlei versteckte Informationen gibt, welche nur einem Teil der Player bekannt sind. Alle Player haben also genau den gleichen Informationsstand. Dadurch hat keiner der Player die Möglichkeit etwas vor den anderen zu verheimlichen. Somit können alle Player ihre Entscheidungen auf der gleichen Grundlage treffen. Beispiele für solche Spiele sind z.B. Schach oder Go. [13, S. 26]

Im Gegensatz dazu steht das **Imperfect Information Game**. Dort gibt es Informationen, welche nur einem oder einem Teil der Player bekannt sind. Dies ermöglicht weitere strategische Elemente wie z.B. das Bluffen. Entscheidungen werden hier aufgrund von Wahrscheinlichkeiten getroffen. Games mit Imperfect Information sind in der Regel komplexer, da der genaue Spielzustand nicht bekannt ist und es deshalb sehr viele Möglichkeiten gibt. Beispiele für Imperfect Information Games sind z.B. Poker oder Skat. [13, S. 26–27]

Zero-Sum Games

In einem **Zero-Sum Game** gleichen sich die Rewards der Player gegenseitig aus. Die Summe aller positiven und negativen Rewards der Player ergeben somit 0. Eines der einfachsten Beispiele ist sicherlich das Spiel “Schere, Stein, Papier”. An diesem Spiel sind zwei Spieler beteiligt. Beide Spieler entscheiden sich gleichzeitig für eines von drei Symbolen: Schere, Stein oder Papier. Jedes der drei Symbole verliert gegen ein anderes Symbol. Wählen beide das gleiche Symbol gibt es ein Unentschieden. Abbildung 7 zeigt, wie sich die Gewinne gegenseitig ausgleichen und die Summe der Rewards 0 ergibt.

Beschreibungsformen

In der Spieltheorie können Spiele in verschiedenen Formen beschrieben werden. Grundsätzlich wird zwischen **Normal Form Games** und **Extensive Form Games** unterschieden. [13]

Bei einem **Normal Form Game** wählen die Player anhand ihrer Strategy simultan eine Aktion aus. Dabei besitzen sie keine weiteren Informationen darüber, welche Actions die anderen Player wählen werden. Die Actions werden ausgeführt und die Player erhalten einen Reward abhängig von den gewählten Actions. Diese Art von Games wird üblicherweise in einer Matrix abgebildet. Ein Beispiel dafür ist das bereits beschriebene Spiel “Schere, Stein, Papier” (siehe Abbildung 7) [13, S. 8]

		Player 2		
		Schere	Stein	Papier
Player 1	Schere	0, 0	-1, +1	+1, -1
	Stein	+1, -1	0, 0	-1, +1
	Papier	-1, +1	+1, -1	0, 0

Abbildung 7: “Schere, Stein, Papier” abgebildet in einer Matrix

Bei einem **Extensive Form Game** werden die Actions der Player im Gegensatz zum Normal Form Game nicht gleichzeitig ausgeführt. Vielmehr werden solche Games sequenziell in Runden abgehandelt. In jeder Runde ist ein einzelner Player am Zug. Der Player wählt eine Action und verändert damit den Zustand (aktueller Player, Informationsstand der Player etc.) des Games. Dieser Zustand wird aber nicht nur durch die Actions der Player bestimmt, sondern kann auch durch zufällige Actions verändert werden (z.B. durch das zufällige Verteilen von Karten). Die Zustandsänderungen werden üblicherweise als Baumstruktur abgebildet (**Game Tree**). Je nach Komplexität des Games können solche Bäume enorm gross werden, wie Kapitel 5.3 versucht zu verdeutlichen.

Ein Übergang von einem Knoten (**Node**) zu einem anderen Knoten entspricht einer möglichen Action. Ist man am Ende des Baums angekommen, ist das Game zu Ende. Die Knoten des Baums können einer von drei Kategorien zugeordnet werden (siehe Abbildung 8) [13, S. 10–12]:

- **Chance Node:** Von diesem Knoten aus werden Actions zufällig oder nach einer definierten Wahrscheinlichkeitsverteilung ausgewählt.

- **Action Node:** Von diesem Knoten aus wählt der aktuelle Player jeweils eine Actions aus.
- **Terminal Node:** Vom letzten Knoten des Baums können keine weiteren Actions mehr ausgeführt werden. Ist man bei einem solchen Knoten angelangt, ist das Game zu Ende und der Reward kann berechnet werden.

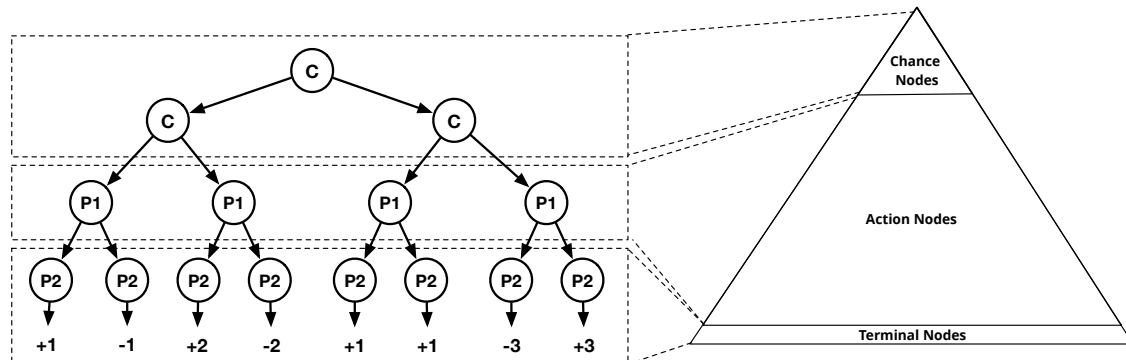


Abbildung 8: Beispiel eines Game Trees für ein fiktives Spiel

Jeder Knoten bzw. Zustand im Game Tree ist einzigartig und kann nur durch eine bestimmte Sequenz von Actions erreicht werden. Diese Sequenz wird im folgenden **History** genannt, da sie die bisherige Geschichte des Games abbildet. Sobald die Sequenz einen Terminal Node erreicht hat, spricht man von einer **Terminal History**. Im Falle eines Imperfect Information Games kann es sein, dass für einen Player nur ein Teil der Actions ersichtlich sind. Insbesondere bei Chance Nodes (zufälligen Actions) ist es möglich, dass nur ein Player die effektiv selektierte Action mitbekommt. Dies ist z.B. bei der Verteilung von verdeckten Handkarten der Fall. Player haben somit Zugriff auf öffentliche und private Informationen. Die privaten Informationen werden im Folgenden als **Private State** bezeichnet (z.B. eigene Handkarten) und die öffentlichen Informationen als **Public State** (z.B. ausgespielte Karten). Die Kombination von Private und Public State heisst **Information Set** und bildet die komplette Sicht eines Players auf das Spiel ab. Bei einem Perfect Information Game gibt es keinen Private State.

Am besten lässt sich das Konzept des Information Sets an einem einfachen Beispiel zeigen (siehe Abbildung 9). Gegeben sei ein einfaches Kartenspiel mit zwei Spielern und drei Karten A, B und C. Zuerst erhält Spieler 1 zufällig eine Karte. Anschliessend erhält Spieler 2 ebenfalls zufällig eine Karte. Es gilt nun die Annahme, dass Spieler 1 die Karte A erhalten hat. Spieler 1 kann nicht sagen, ob sein Kontrahent die Karte B oder die Karte C auf der Hand hält. Er befindet sich damit im Information Set A. Hätte Spieler 1 zu Beginn eine andere Karte erhalten, würde er sich jeweils in einem anderen Information Set befinden. Das Information Set bildet somit die Entscheidungsgrundlage, welche einem Spieler zur Verfügung steht bei der Wahl einer Action.

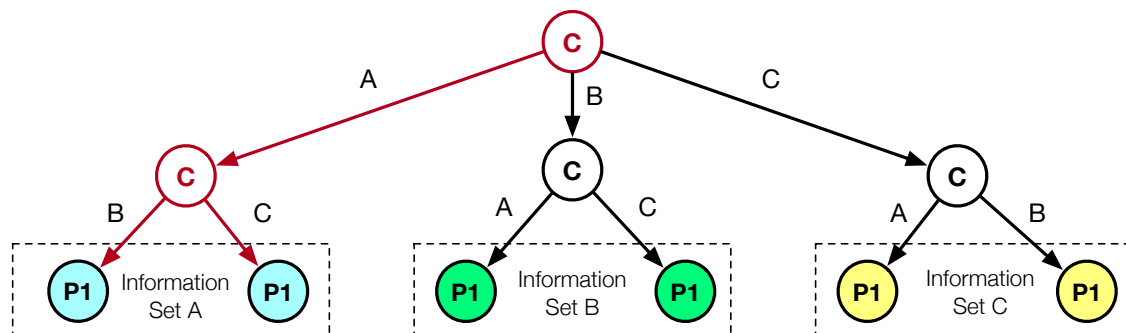


Abbildung 9: Illustration für History und Informations Sets

Nash Equilibrium

Equilibrium ist ein Begriff aus der Wissenschaft und steht für ein System, bei dem alle sich beeinflussenden Kräfte im Gleichgewicht stehen. Das Nash Equilibrium ist ein Set von Strategy Profiles (ein Profil pro Player), bei dem es sich für keinen der Player lohnt davon abzuweichen. Tut er es trotzdem, so verringerte sich der zu erwartende Reward für diesen Player.

		Gefangener 2				Player 2		
		Schweigen	Gestehen			Schere	Stein	Papier
Gefangener 1	Schweigen	1, 1	10, 0	Player 1	Schere	0, 0	-1, +1	+1, -1
	Gestehen	0, 10	5, 5		Stein	+1, -1	0, 0	-1, +1
					Papier	-1, +1	+1, -1	0, 0

Abbildung 10: Matrix Abbildung für "Prisoners Dilemma" und "Schere, Stein, Papier"

Ein Nash Equilibrium bringt aber dem Player nicht unbedingt den höchsten Reward. Es ist in einem kompetitiven Spiel vielmehr die perfekte Verteidigung gegen die Strategie des Gegners. Dies kann man sehr gut am Beispiel des "Prisoners Dilemma" zeigen.

Beim Prisoners Dilemma werden zwei Verbrecher inhaftiert. Die Ermittler haben aber nicht genug Beweise um den Gefangenen alle Straftaten nachzuweisen. Daher trennen sie die beiden voneinander und befragen sie einzeln. Möglichkeiten zur Absprache werden ausgeschlossen. Die Ermittler geben beiden Gefangenen die Möglichkeit zu gestehen, um so das Strafmass zu verringern. In Abbildung 10 sieht man, wie viele Jahre die Gefangenen jeweils absitzen müssten. Gesteht einer von ihnen, kommt er frei und sein Partner wandert für 10 Jahre ins Gefängnis. Gestehen beide, erhalten beide 5 Jahre. Würden beide schweigen, kämen beide mit nur einem Jahr davon. Dies bringt die Gefangenen in eine Zwickmühle. Am besten würden sie davonkommen, wenn beide schweigen. Da Gestehen einen höheren Reward verspricht, kann es aber gut sein, dass einer der Gefangenen gestehen wird. Somit müssen eigentlich beide davon

ausgehen, dass der jeweils andere gestehen wird. Die perfekte Gegenstrategie dagegen ist in diesem Fall, ebenfalls zu gestehen. Dies entspricht gleichzeitig dem Nash Equilibrium für dieses Spiel. Wie man an diesem Beispiel ebenfalls sehr gut sehen kann, hätte es mit Schweigen für beide eine noch bessere Lösung gegeben. [15]

Anhand “Schere, Stein, Papier” lässt sich das Nash Equilibrium ebenfalls sehr gut erklären. Hier entspricht es folgenden Mixed Strategies:

$$Strategy_{P1} = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{pmatrix} \quad Strategy_{P2} = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{pmatrix}$$

Die Player $P1$ und $P2$ führen jede ihrer Actions mit genau der gleichen Wahrscheinlichkeit aus. Wie man in Abbildung 10 sehen kann, ergibt es für keinen Player Sinn von seiner Strategy abzuweichen. Würde einer von ihnen z.B. mehr “Stein” spielen, könnte der andere seine Strategy anpassen und mehr “Papier” spielen, um einen höheren Reward herauszuholen.

Zusammengefasst gelten für ein **kompetitives Zero-Sum Game mit zwei Spielern** folgende Eigenschaften:

- Das Spielen nach Nash Equilibrium führt zum höchstmöglichen Gewinn, wenn der Gegner dasselbige tut.
- Weicht der Gegner vom Nash Equilibrium ab, fährt er schlechter, als wenn er sich daran gehalten hätte.

Über sehr viele Spiele hinweg ist gegen einen solchen Spieler somit maximal ein Unentschieden zu erreichen. [13]

Anwendung auf Schieber Jass

Nun geht es daran die Theorien aus diesem Kapitel auf den Schieber Jass anzuwenden. Abbildung 11 zeigt, wie die Konzepte aus der Spieltheorie mit dem Schieber Jass verknüpft werden können.

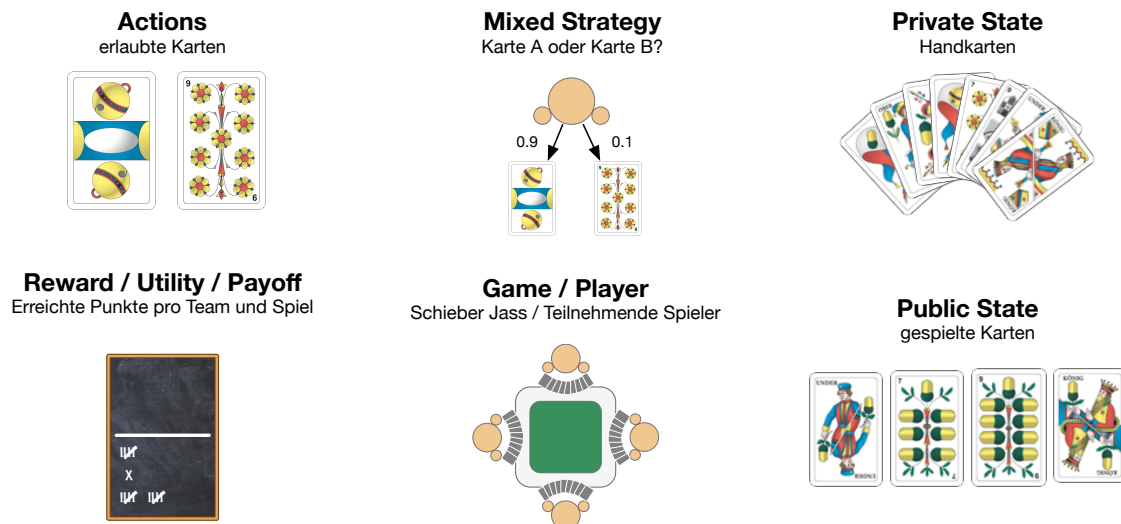


Abbildung 11: Konzepte der Spieltheorie angewendet auf den Schieber Jass

Der Schieber Jass ist ein Spiel mit vordefinierten Regeln, welche in Kapitel 5.1 nachzulesen sind. Mitspieler sowie der eigene Spielpartner sind für jeden bekannt. Es ist zu jedem Zeitpunkt klar, welche Karten ausgespielt werden dürfen, und welche Punktzahlen man damit erreichen kann. Somit handelt es sich beim Schieber Jass um ein **Complete Information Game**.

Die ausgespielten Karten entsprechen dem **Public State** im **Game**. Durch das zufällige Verteilen von verdeckten Handkarten zu Beginn des Spiels hat zudem jeder Spieler seinen eigenen **Private State**. Der Schieber Jass gehört damit zur Klasse der **Imperfect Information Games**.

Ein Schieber Jass wird in Runden abgehandelt. Pro Runde ist jeweils ein Spieler am Zug. In jeder Runde spielt jeder Spieler eine seiner Karten, bis keiner mehr eine Karte auf der Hand hält. Das Ausspielen der Karten kann man als Sequenz von **Actions** betrachten. Daher kann das Spiel als **Extensive Form Game** beschrieben und mittels **Game Tree** abgebildet werden.

Insgesamt gibt es in einem Schieber Jass Spiel 157 Punkte zu gewinnen (Multiplikatoren bewusst ausgeschlossen). Die Punkte werden pro Team gezählt. Die gewonnen Punkte eines Teams entsprechen dem Verlust dieser Punkte für das andere Team, somit handelt es sich beim Schieber Jass um ein **Zero-Sum Game**.

Hinweis: Mehr als zwei Spieler

Beim Schieber Jass sitzen grundsätzlich 4 Spieler am Tisch. Da es sich bei dieser Variante jedoch um einen Partnerjass handelt, spielen im Endeffekt zwei Teams gegeneinander, und nicht die einzelnen Spieler. Es wurde deshalb die Annahme getroffen, dass die Eigenschaften für Spiele mit zwei Spielern trotzdem gelten.

5.3 Kombinatorik

In diesem Kapitel geht es darum die Komplexität des Schieber Jass zu analysieren. Ziel ist abschätzen zu können, wie viele Zustände das Spiel besitzt und ob es überhaupt möglich ist, alle Zustände eines Spiels zu berechnen. Diese Information ist für die Auswahl möglicher Ansätze zur Realisierung eines Jass Bots essentiell.

Hinweis: Spielregeln

Bei den folgenden Berechnungen werden die Spielregeln des Schieber Jass aus Kapitel 5.1 **nicht** beachtet. Es wird angenommen, dass jeder Spieler pro Stich eine beliebige Handkarte auf den Tisch legen darf.

Mögliche Spielausgänge

Ein Spiel besteht aus 9 Stichen. In jedem Stich spielen die Spieler eine ihrer Handkarten aus. Am Spiel sind 4 Spieler beteiligt. Um also die Anzahl der möglichen Spielausgänge zu berechnen, muss man sich überlegen, wie viele mögliche Reihenfolgen es für das Ausspielen der 36 Karten im Spiel gibt. Dies entspricht der Anzahl Permutationen für 36 Elemente, welche mittels Fakultät berechnet wird.

n_C = Anzahl Karten im Spiel

n_T = Anzahl möglicher Spielausgänge (ohne Spielregeln)

$$n_T = n_C! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot 36 \approx 3.72 \cdot 10^{41}$$

Kartenverteilungen

Die 36 Handkarten werden zu Beginn des Spiels zufällig auf die 4 Spieler am Tisch verteilt. Der erste Spieler bekommt aus 36 Karten 9 zugewiesen. Der zweite Spieler bekommt aus den verbleibenden 27 Karten seine 9 Karten zugewiesen usw. Die Reihenfolge ist bei der Verteilung irrelevant. Zur Berechnung der Anzahl Möglichkeiten, aus einer Menge von n Elementen k auszuwählen verwendet man in der Kombinatorik den Binomialkoeffizienten:

n = Anzahl der zur Auswahl stehenden Karten

k = Anzahl der Handkarten pro Spieler

n_V = Anzahl möglicher Kartenverteilungen (4 Spieler)

$$n_V = \binom{36}{9} \cdot \binom{27}{9} \cdot \binom{18}{9} \cdot \binom{9}{9} \approx 2.15 \cdot 10^{19}$$

Da man als Spieler die eigenen Handkarten kennt, verbleiben für die anderen drei Spieler noch folgende mögliche Kartenverteilungen:

Stich	Handkarten	Berechnung	Möglichkeiten
1	9	$\binom{27}{9} \cdot \binom{18}{9} \cdot \binom{9}{9}$	227873431500
2	8	$\binom{24}{8} \cdot \binom{16}{8} \cdot \binom{8}{8}$	9465511770
3	7	$\binom{21}{7} \cdot \binom{14}{7} \cdot \binom{7}{7}$	399072960
4	6	$\binom{18}{6} \cdot \binom{12}{6} \cdot \binom{6}{6}$	17153136
5	5	$\binom{15}{5} \cdot \binom{10}{5} \cdot \binom{5}{5}$	756756
6	4	$\binom{12}{4} \cdot \binom{8}{4} \cdot \binom{4}{4}$	34650
7	3	$\binom{9}{3} \cdot \binom{6}{3} \cdot \binom{3}{3}$	1680
8	2	$\binom{6}{2} \cdot \binom{4}{2} \cdot \binom{2}{2}$	90
9	1	$\binom{3}{1} \cdot \binom{2}{1} \cdot \binom{1}{1}$	6

Tabelle 12: Mögliche Kartenverteilungen pro Stich aus Spielersicht

Dabei wird ignoriert, dass Spieler typischerweise im Verlauf des Spiels Hinweise hinsichtlich möglicher Handkarten der anderen Spieler erhalten. In späteren Stichen steigt die Wahrscheinlichkeit, dass es nach Ausschlussverfahren effektiv viel weniger Möglichkeiten gibt, als hier theoretisch berechnet wird. Die Zahlen widerspiegeln sozusagen den “Worst Case”.

Sicht eines Spielers

Jedem Spieler sind die eigenen Handkarten bekannt. Alle anderen Karten sind für ihn unbekannte Karten. Der Spieler muss im ersten Stich eine seiner 9 Handkarten ausspielen. Für jede dieser Entscheidungen können durch die anderen Spieler 3 aus 27 unbekannten Karten ausgespielt werden. Dabei ist die Reihenfolge der ausgespielten Karten relevant:

n_U = Anzahl dem Spieler unbekannte Karten

n_H = Anzahl Handkarten des Spielers

n_S = Anzahl möglicher Stichausgänge aus Sicht eines Spielers

$$n_S = \binom{n_H}{1} \cdot \binom{n_U}{3} \cdot 3!$$

Die folgende Tabelle zeigt die Berechnungen für mögliche Stichausgänge pro Stich aus der Sicht eines Spielers:

Stich	Handkarten	Unbekannte Karten	Berechnung	Möglichkeiten
1	9	27	$9 \cdot \binom{27}{3} \cdot 3!$	157950
2	8	24	$8 \cdot \binom{24}{3} \cdot 3!$	97152
3	7	21	$7 \cdot \binom{21}{3} \cdot 3!$	55860
4	6	18	$6 \cdot \binom{18}{3} \cdot 3!$	29376
5	5	15	$5 \cdot \binom{15}{3} \cdot 3!$	13650
6	4	12	$4 \cdot \binom{12}{3} \cdot 3!$	5280
7	3	9	$3 \cdot \binom{9}{3} \cdot 3!$	1512
8	2	6	$2 \cdot \binom{6}{3} \cdot 3!$	240
9	1	3	$1 \cdot \binom{3}{3} \cdot 3!$	6

Tabelle 13: Mögliche Ausgänge pro Stich aus Spielersicht

Um die Anzahl an möglichen Spielausgängen für einen Spieler zu berechnen müssen die Möglichkeiten pro Stich für die verbleibenden Stiche aufmultipliziert werden:

n = Anzahl verbleibende Stiche

k = Zähler für Stiche

$f_T(n)$ = Anzahl möglicher Spielausgänge für einen Spieler

$$f_T(n) = \prod_{k=1}^n k \cdot \binom{3 \cdot k}{3} \cdot 3!$$

Das gleiche Resultat erhält man bei Multiplikation der möglichen Reihenfolgen für das Ausspielen der eigenen 9 Handkarten mit den möglichen Reihenfolgen der restlichen 27, dem Spieler unbekannten Karten:

n_H = Anzahl Handkarten des Spielers

n_U = Anzahl dem Spieler unbekannte Karten

$$n_H! \cdot n_U!$$

Somit ergeben sich folgende mögliche Spielausgänge pro Stich aus Spielersicht:

Stich	Handkarten	Unbekannte Karten	Berechnung	Möglichkeiten
1	9	27	$9! \cdot 27!$	$3.95 \cdot 10^{33}$
2	8	24	$8! \cdot 24!$	$2.50 \cdot 10^{28}$
3	7	21	$7! \cdot 21!$	$2.58 \cdot 10^{23}$
4	6	18	$6! \cdot 18!$	$4.61 \cdot 10^{18}$
5	5	15	$5! \cdot 15!$	156920924160000
6	4	12	$4! \cdot 12!$	11496038400
7	3	9	$3! \cdot 9!$	2117280
8	2	6	$2! \cdot 6!$	1440
9	1	3	$1! \cdot 3!$	6

Tabelle 14: Mögliche Spielausgänge pro Stich aus Spieler-sicht

Schlussfolgerungen

Auch wenn die Spielregeln bei dieser Analyse nicht berücksichtigt wurden, wird schnell klar, dass die Anzahl der Möglichkeiten die Fähigkeiten eines handelsüblichen Computers überschreitet. Die Zustände im Spiel lassen sich als Ganzes weder abspeichern noch ist die Anwendung eines Algorithmus denkbar. Die Herausforderung wird darin bestehen einen Ansatz zu finden, der mit dieser Komplexität in irgendeiner Form umgehen kann. Es braucht "Mut zur Lücke".

5.4 Kartenbuchhaltung

Die Kartenbuchhaltung ist ein Konzept, welches in jedem der analysierten Lösungsansätze zum Tragen kommt. Man beschäftigt sich dabei mit zwei wichtigen Fragestellungen:

- Welche Karten sind überhaupt noch im Spiel?
- Welche Karten haben die einzelnen Spieler möglicherweise noch auf ihrer Hand?

Erfahrene Jass Spieler sind in der Lage, diese Fragen für sich durch aufmerksame Beobachtungen während eines Spiels zu beantworten. Basierend darauf werden sie Entscheidungen treffen, wenn es ums Ausspielen einer Karte geht. Je nach Spielsituation können sie dadurch mehrere Spielzüge im Voraus planen, was die Gewinnchancen für das eigene Team erhöht. Es stellt sich somit die Frage, wie man einem Computerprogramm diese Auffassungsgabe beibringen kann. Die möglichen Beobachtungen während eines Spiels können in zwei Kategorien unterteilt werden.

- Die Beobachtung gibt Hinweise auf eine Tatsache.

- Die Beobachtung gibt Hinweise auf eine Vermutung.

Tatsachen

Handelt es sich bei der Schlussfolgerung um eine **Tatsache**, dann kann man diese Information in binärer Form abbilden. Also ein Wert 0 oder 1, je nachdem ob die Beobachtung während eines Spiels bereits gemacht wurde oder nicht. Zu dieser Kategorie zählen Beobachtungen, die in direktem Zusammenhang mit den Spielregeln stehen. Die nachfolgend angesprochenen Spielregeln sind in Kapitel 5.1 genauer beschrieben.

Hinweis: Regelkonformität

Im Rahmen der Bachelorarbeit konnte die Annahme getroffen werden, dass jederzeit regelkonform gespielt wird. Die virtuelle Spielumgebung sorgt dafür, dass die Regeln bedingungslos umgesetzt werden. Schwieriger wird es, wenn diese Annahme nicht mehr zutrifft. Dies wäre beispielsweise der Fall, sobald ein Spiel durch menschliche Spieler geleitet wird und Regeln möglicherweise leicht anders oder sogar falsch angewendet werden. Betrug - wie das berühmte "Ass im Ärmel" - ist zwar aufgrund der geringen Grösse des Kartendecks eher unwahrscheinlich, aber trotzdem nicht ganz auszuschliessen.

Tatsache - Eigene Handkarten

Betrachtet man ein Spiel als Beobachter - quasi aus der Vogelperspektive - dann muss die Kartenbuchhaltung für alle vier Spieler gemacht werden. Aus Sicht eines einzelnen Spielers ist die Buchhaltung jedoch nur für die drei anderen Spieler nötig, da man die eigenen Handkarten kennt (Private State). Somit braucht man sich die hier beschriebenen Überlegungen nur noch für 27 der insgesamt 36 Karten im Deck zu machen, was das Problem hinsichtlich Kombinatorik und grossen Zahlen etwas entschärft.

Tatsache - Ausgespielte Karten

Das Kartendeck enthält jede der Karten genau einmal. Sobald eine Karte für alle Spieler ersichtlich auf dem Tisch liegt, kann man davon ausgehen, dass ab diesem Zeitpunkt keiner der Spieler diese Karte mehr auf der Hand haben kann (Public State).

Tatsache - Vorgegebene Farbe

Zu Beginn eines Stichs gibt der Startspieler mit seiner ausgespielten Karte eine Farbe vor. Verstösst einer der nachfolgenden Spieler gegen diese Vorgabe, dann kann man davon ausgehen, dass er keine Karten der vorgegebenen Farbe auf der Hand haben kann. Anderenfalls hätte er eine Karte in genau dieser Farbe ausspielen müssen.

Hinweis: Karten in Trumpffarbe

Davon ausgenommen sind Situationen, in denen ein nachfolgender Spieler mit einer Karte in Trumpffarbe gegen die Vorgabe verstösst. Trumpffarbe darf jederzeit ausgespielt werden, selbst wenn man noch Karten der vorgegebenen Farbe auf der Hand hat. Dies gilt aber nur, wenn bei der Trumpfwahl eine Farbe gewählt wurde.

Hinweis: Under in Trumpffarbe

Der Besitzer der Under Karte in Trumpffarbe muss die Karte nicht ausspielen, selbst wenn Trumpffarbe in einem Stich vorgegeben ist. Somit gibt es keine Möglichkeit, diese Karte für einzelne Spieler vorzeitig auszuschliessen. In Anbetracht ihrer Position in der Rangordnung und ihres Punktwerts stellt diese Karte durchaus eine Gefahr für die restlichen Spieler dar. Dies gilt aber nur, wenn bei der Trumpfwahl eine Farbe gewählt wurde.

Tatsache - Untertrumpfen

Untertrumpfen ist beim Schieber Jass grundsätzlich verboten, es sei denn ein Spieler hat tatsächlich nur noch Karten in Trumpffarbe auf der Hand. Diese Situation mag zwar eher selten auftreten, ist aus Sicht der Kartenbuchhaltung aber enorm wertvoll. Durch Untertrumpfen können somit alle anderen, im Spiel verbleibenden Karten in Nebenfarbe für den betroffenen Spieler ausgeschlossen werden.

Tatsache - Weisen

Beim Weis müssen Informationen zu den eigenen Handkarten öffentlich gemacht werden, um die angemeldeten Punkte zu bekommen. Es handelt sich hierbei also um eine sichere - und je nach Weis - ergiebige Quelle für Informationen zu möglichen Handkarten.

Tatsache - Anzahl Handkarten

Durch den Spielablauf ist zu jedem Zeitpunkt definiert, wie viele Karten die Spieler auf ihrer Hand haben dürfen. Zu Beginn eines Spiels kann aus Sicht eines Spielers jeder der anderen drei Spieler 9 von 27 möglichen Karten auf der Hand haben. Mit jedem Stich wird die Anzahl der Handkarten um 1 reduziert. Zudem schrumpft die Menge der möglichen Karten pro Spieler jeweils mindestens um die vier Karten, welche in einem Stich ausgespielt wurden. Je nach Beobachtung können es für einzelne Spieler auch mehr Karten sein, die wegfallen. Stimmen Anzahl der Handkarten sowie die Menge der möglichen Karten für einen Spieler überein, dann hat man seine Handkarten herausgefunden. Diese Karten können gleichzeitig für alle anderen Spieler ausgeschlossen werden.

Tatsache - Ausschlussverfahren

Manche Beobachtungen können abhängig vom Spielverlauf dazu führen, dass gewisse Karten laut Buchhaltung nur noch bei genau einem der anderen drei Spieler auf der Hand sein können. Beispielsweise kann sich herausstellen, dass eine Farbe nur noch bei einem der Spieler auffindbar ist, da alle anderen zuvor gegen eine entsprechende Vorgabe verstossen haben. Stimmen Anzahl der Handkarten sowie die Menge solch "exklusiver" Karten überein, dann hat man die Handkarten dieses Spielers herausgefunden. Alle anderen verbleibenden Karten im Spiel können für ihn somit ausgeschlossen werden.

Die Kartenbuchhaltung kann in Software relativ einfach in Form von Sets oder Listen abgebildet werden, wie Abbildung 12 illustriert. Sie wird nach jedem Spielzug basierend auf den dabei gemachten Beobachtungen und den dadurch möglichen Schlussfolgerungen aktualisiert.

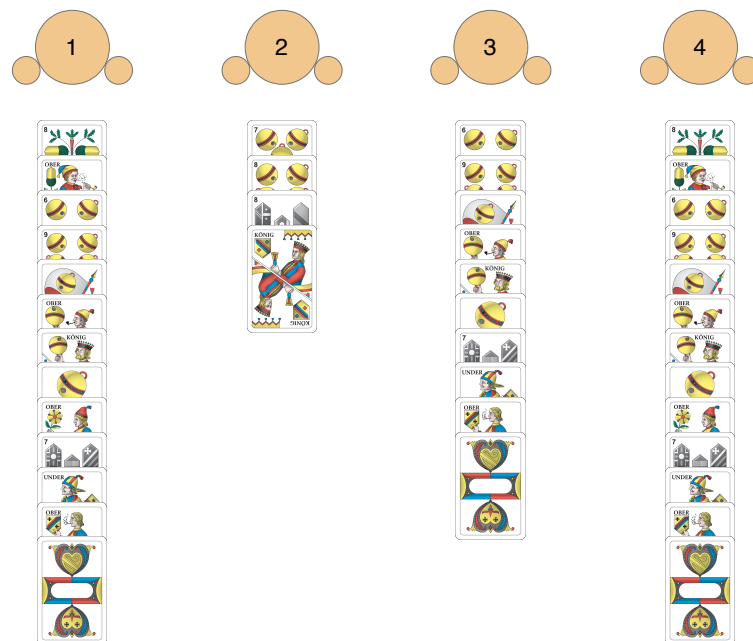


Abbildung 12: Buchhaltung über mögliche Handkarten pro Spieler

Es gilt zu beachten, dass dieser Vorgang iterativ ablaufen muss. Durch die Anwendung von Schlussfolgerungen auf die Buchhaltung kann es sein, dass in der aktualisierten Version weitere Schlussfolgerungen möglich werden. Die Aktualisierung ist erst abgeschlossen, wenn die Buchhaltung keine weiteren Schlussfolgerungen mehr zulässt. Man könnte an der Stelle vielleicht sogar von einer **Normalisierung** sprechen.

Hinweis: Komplexität

Auch wenn die vorgestellten Schlussfolgerungen teilweise trivial erscheinen, hat die saubere Umsetzung der Kartenbuchhaltung verhältnismässig viel Zeit in Anspruch genommen.

Die Korrektheit der Kartenbuchhaltung zu einem beliebigen Zeitpunkt innerhalb eines Spiels ist von essenzieller Bedeutung:

- Werden mögliche Beobachtungen verpasst, dann verschwendet man Rechenzeit für die Betrachtung von Kartenverteilungen, die aufgrund des bisherigen Spielverlaufs gar nicht mehr möglich sind.
- Werden falsche Schlussfolgerungen auf die Kartenbuchhaltung angewendet kann es sein, dass man beim Versuch, die Karten unter Berücksichtigung der Buchhaltung zufällig auf die Spieler zu verteilen, keine gültige Kartenverteilung finden wird.

Letzteres ist insofern kritisch, weil praktisch alle Lösungsansätze die Fähigkeit einer zufälligen Kartenverteilung für die verbleibenden Karten im Spiel voraussetzen. Dieses Aufgabe hat sich während der Umsetzung im Code als nicht trivial herausgestellt.

Hinweis: Zufällige Kartenverteilung

Das Problem kann auch allgemeiner beschrieben werden. Im Prinzip hat man Elemente, die auf Posten verteilt werden müssen, wobei am Schluss jeder Posten eine bestimmte Anzahl zugewiesener Elemente aufweisen muss. Zusätzlich gibt es eine Liste von Einschränkungen für Elemente, welche nur ganz bestimmten Posten zugeteilt werden dürfen. Für Probleme dieser Art gibt es keinen effizienten Algorithmus, um eine bzw. alle Lösungen zu finden. Es bleibt nur ein "Brute Force" Ansatz. Man kann zufällige Verteilungen produzieren und jeweils überprüfen, ob alle Einschränkungen damit erfüllt sind, was irgendwann zu einer gültigen Lösung führen wird. Die Gefahr eines solchen Algorithmus ist, dass er im schlimmsten Fall endlos laufen wird, wenn es für die Problemstellung gar keine Lösung gibt. Eine inkonsistente Kartenbuchhaltung kann zu einer solchen Situation führen.

Es braucht eine Möglichkeit um festzustellen, ob sich die Kartenbuchhaltung in einem konsistenten Zustand befindet, bevor man damit weiterarbeitet. So kann man auf kontrollierte Art und Weise mit solchen Situationen umgehen. Seien

$$K_1, K_2, K_3, K_4$$

Mengen mit den möglichen Handkarten pro Spieler als Inhalt und

$$n_1, n_2, n_3, n_4$$

die Anzahl an Handkarten, die Spieler zu einem bestimmten Zeitpunkt im Spiel noch auf der Hand haben müssen. Für alle Spieler soll nun überprüft werden, ob es mit einem gegebenem Stand der Kartenbuchhaltung überhaupt möglich ist, die verlangte Anzahl an Handkarten zuzuweisen. Dafür werden zunächst Tupel der Form

$$(n_1, K_1), (n_2, K_2), (n_3, K_3), (n_4, K_4)$$

gebildet. Alle möglichen Kombinationen dieser 4 Tupel müssen auf Konsistenz geprüft werden, was insgesamt

$$\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1} = 1 + 4 + 6 + 4 = 15$$

Überprüfungen entspricht. Für jede Kombination der Tupel wird mittels Bilden der Vereinigungsmenge überprüft, ob nach Zusammenführung der Einschränkungen hinsichtlich möglicher Handkarten noch ausreichend Karten für die entsprechenden Spieler zur Verfügung

stehen.

$$\sum n \leq |\bigcup K|$$

Fällt eine dieser Überprüfungen negativ aus, dann befindet sich die Kartenbuchhaltung in einem inkonsistenten Zustand und darf somit nicht weiterverwendet werden. Nachfolgend sind alle Überprüfungen der Vollständigkeit halber nochmals aufgelistet.

$$\begin{aligned} n_1 &\leq |K_1| \\ n_2 &\leq |K_2| \\ n_3 &\leq |K_3| \\ n_4 &\leq |K_4| \\ n_1 + n_2 &\leq |K_1 \cup K_2| \\ n_1 + n_3 &\leq |K_1 \cup K_3| \\ n_1 + n_4 &\leq |K_1 \cup K_4| \\ n_2 + n_3 &\leq |K_2 \cup K_3| \\ n_2 + n_4 &\leq |K_2 \cup K_4| \\ n_3 + n_4 &\leq |K_3 \cup K_4| \\ n_1 + n_2 + n_3 &\leq |K_1 \cup K_2 \cup K_3| \\ n_1 + n_2 + n_4 &\leq |K_1 \cup K_2 \cup K_4| \\ n_1 + n_3 + n_4 &\leq |K_1 \cup K_3 \cup K_4| \\ n_2 + n_3 + n_4 &\leq |K_2 \cup K_3 \cup K_4| \\ n_1 + n_2 + n_3 + n_4 &\leq |K_1 \cup K_2 \cup K_3 \cup K_4| \end{aligned}$$

Vermutungen

Zu Beginn dieses Kapitels wurden **Vermutungen** angesprochen, die man anhand gewisser Beobachtungen während eines Spiels machen kann. Schlussfolgerungen dieser Art können nicht mehr binär, sondern nur durch eine Wahrscheinlichkeit abgebildet werden. Also eine Zahl zwischen 0 und 1, die angibt, mit welcher Wahrscheinlichkeit eine Beobachtung zur daraus möglichen Schlussfolgerung führt.

Hinweis: Einschränkung

Im Rahmen dieser Bachelorarbeit wurden bei der Umsetzung der Kartenbuchhaltung in erster Linie **Tatsachen** berücksichtigt. Im Verlauf der Arbeit kamen diverse Ideen auf, wie man die Buchhaltung um **Vermutungen** erweitern könnte. Aus zeitlichen Gründen konnten diese Ideen nicht mehr ausprobiert werden. Trotzdem sollen sie hiermit für mögliche

Folgeprojekte festgehalten werden.

Eines haben ein Grossteil der Vermutungen gemeinsam. Sie setzen zu einem gewissen Grad voraus, dass sich der elektronische Spieler **typischem Spielverhalten menschlicher Spieler** anpassen kann. So wünscht man sich beispielsweise, dass der Spielpartner im gleichen Team auf gewisse Signale reagiert, die man ihm über allgemein gültige - wenn nicht sogar individuell vereinbarte - Spielmuster vermitteln will. Das Zusammenspiel ist ein wichtiger Aspekt hinsichtlich Verbesserung der Akzeptanz eines Bots bei menschlichen Spielern. Zudem ist mit einem eingespielten Team eine deutliche Verbesserung des Spielniveaus zu erwarten.

Gleichzeitig birgt die Berücksichtigung solcher Vermutungen immer ein gewisses **Risiko**. Die Erkennung bestimmter Spielmuster könnte von anderen Spielern ausgenutzt werden, um den Algorithmus bewusst in die Irre zu führen. Es gilt somit abzugwägen - oder vielleicht sogar durch Spielaufzeichnungen zu erlernen - wie stark solche Signale in die Entscheidungsfindung einfließen sollen. Nachfolgend wird eine Auswahl von möglichen Vermutungen aufgelistet, die man in die Kartenbuchhaltung einfließen lassen könnte. Es handelt sich hierbei um typisches Verhalten, niemand hält die Spieler jedoch davon ab, atypische Entscheidungen zu treffen um Verwirrung zu stiften.

Vermutung - Trumpfwahl

Bei der Trumpfwahl gibt ein Spieler normalerweise erste Informationen zu seinen Handkarten preis. So könnte man beispielsweise die Wahrscheinlichkeit für Karten in Trumpffarbe bei ihm höher ansetzen. Gleiches gilt für sehr tiefe Karten bzw. sehr hohe Karten, wenn die Wahl auf "Obenabe" bzw. "Undenufe" fällt.

Vermutung - Geschobene Trumpfwahl

Gibt ein Spieler die Trumpfwahl an seinen Spielpartner ab ("Gschobe"), dann kann mit einer gewissen Wahrscheinlichkeit auf die Qualität seiner Handkarten geschlossen werden. Je besser man die von ihm angewandten Kriterien für die Trumpfwahl kennt, desto präziser kann die Wahrscheinlichkeit gewisser Handkarten bei ihm gesetzt werden.

Vermutung - Verwerfen

Durch das Ausspielen schwacher Karten in einem Stich kann der Spielpartner signalisieren, dass er von einer bestimmten Farbe keine vernünftigen Karten auf der Hand hat. Damit sinkt die Wahrscheinlichkeit, dass gute Karten dieser Farbe beim ihm auf der Hand sind. Gleichzeitig steigt die Wahrscheinlichkeit bei den Spielern des gegnerischen Teams, und man kann seine Spielzüge entsprechend anpassen.

Vermutung - Anziehen

Im Prinzip das Gegenstück zum Verwerfen, der Spielpartner will signalisieren, dass er von einer bestimmten Farbe gute Karten auf der Hand hat. Damit steigt seine Wahrscheinlichkeit für gute Handkarten in dieser Farbe. Gleichzeitig sinkt die Wahrscheinlichkeit bei den Spielern des gegnerischen Teams. Dies erlaubt, dass man dem Spielpartner Vorlagen gibt, damit er möglichst viele Stiche für sich entscheiden kann.

Vermutung - Individuelle Vereinbarung

Gut eingespielte Spielpartner haben oft ganz spezielle Spielmuster, die sie für den Informationsaustausch anwenden. Selbst nach Erlernen gängiger Spielmuster müsste man einen Schritt weitergehen, um mit solchen Situationen umgehen zu können. Der Bot

müsste in der Lage sein, seinen Spielpartner wiederzuerkennen, um die entsprechenden Spielmuster zu aktivieren oder um sie über mehrere Spiele hinweg zu lernen.

6 Architektur

Bei Bots handelt es sich typischerweise nicht um eigenständige Applikationen, sondern im weitesten Sinne um Plugins, welche in eine dafür geschaffene Umgebung integriert werden. So auch in diesem Fall, die Bots sind im Prinzip einfach mögliche Spieler, die sich innerhalb der virtuellen Spielumgebung - dem Pyschieber [8] - an einen Tisch setzen können für ein Jass Spiel. Der Pyschieber bietet auch schon Funktionalitäten an, um sich mit der Zühlke Umgebung verbinden zu können. Somit werden Teile der Architektur vorgegeben durch die Schnittstelle des Pyschieber, über die Bots mit der Spielumgebung kommunizieren. Die Bot Schnittstelle vom Pyschieber ist sehr einfach gehalten. Eine Python Klasse ist ausreichend, um den eigenen Bot an virtuellen Jass Spielen teilnehmen zu lassen. Nachfolgendes Beispiel zeigt die wichtigsten Methoden, die man darin implementieren sollte.

```
from pyschieber.player.base_player import BasePlayer

class ExamplePlayer(BasePlayer):

    def __init__(self, name):
        super().__init__(name)
        # TODO: Initialize.

    def choose_trumpf(self, geschoben):
        # TODO: Return trumpf choice.

    def choose_card(self, state=None):
        # TODO: Return hand card choice.

    def move_made(self, player_id, card, state):
        super().move_made(player_id, card, state)
        # TODO: Track game state.
```

BasePlayer ist eine Basisklasse, die im Pyschieber Package zur Verfügung gestellt wird. Sie implementiert grundlegende Funktionalität wie beispielsweise das Speichern der eigenen Handkarten, oder die Überprüfung, welche der Handkarten in einer gegebenen Spielsituation ausgespielt werden dürfen. Eigene Bot Klassen müssen von dieser Basisklasse ableiten. *__init__* entspricht dem Konstruktor der Klasse und wird aufgerufen, wenn vom Bot eine Instanz erzeugt wird. Der Name des Bots wird als Parameter entgegen genommen. *choose_trumpf* wird vom Pyschieber aufgerufen, wenn ein Bot zu Beginn eines Spiels den Trumpf wählen muss. Der Parameter *geschoben* gibt Auskunft darüber, ob die Trumpfwahl durch den Spielpartner bereits an den eigenen Bot weitergegeben wurde. In einem solchen Fall muss der Bot natürlich eine Entscheidung treffen, Schieben fällt für ihn als mögliche Option weg. *choose_card* wird vom Pyschieber aufgerufen, wenn der Bot an der Reihe ist und eine seiner Handkarten ausspielen muss. Pyschieber wird die Karte ablehnen, falls sie gegen die Spielregeln des Schieber Jass verstösst. Der Parameter *state* gibt Auskunft über den aktuellen Spielzustand (z.B. Trumpf und bisherige Stiche). *move_made* dient der Beobachtung des Spiels, der Bot wird mittels dieser Funktion über ausgespielte Karten informiert. Die Funktion bietet sich an, um intern abgespeicherte Informationen zum Spiel und zu den anderen Spielern auf dem neusten Stand zu halten.

Alle im Rahmen dieser Bachelorarbeit umgesetzten Bots wurden nach diesem Schema implementiert. Die zugrundeliegenden Algorithmen wurden jeweils ausgelagert, um die Klasse übersichtlich zu halten. Die restliche Architektur basiert grösstenteils auf freien Funktionen, insbesondere für Funktionalität, die in verschiedenen Ansätzen wiederverwendet wurde.

6.1 Packages

Der Code ist nach Themengebiet in Python Packages unterteilt, wie dargestellt in Abbildung 13. Die Packages können analog zu anderen Programmiersprachen als Bibliothek eingebunden werden, um von deren Funktionalität Gebrauch zu machen.

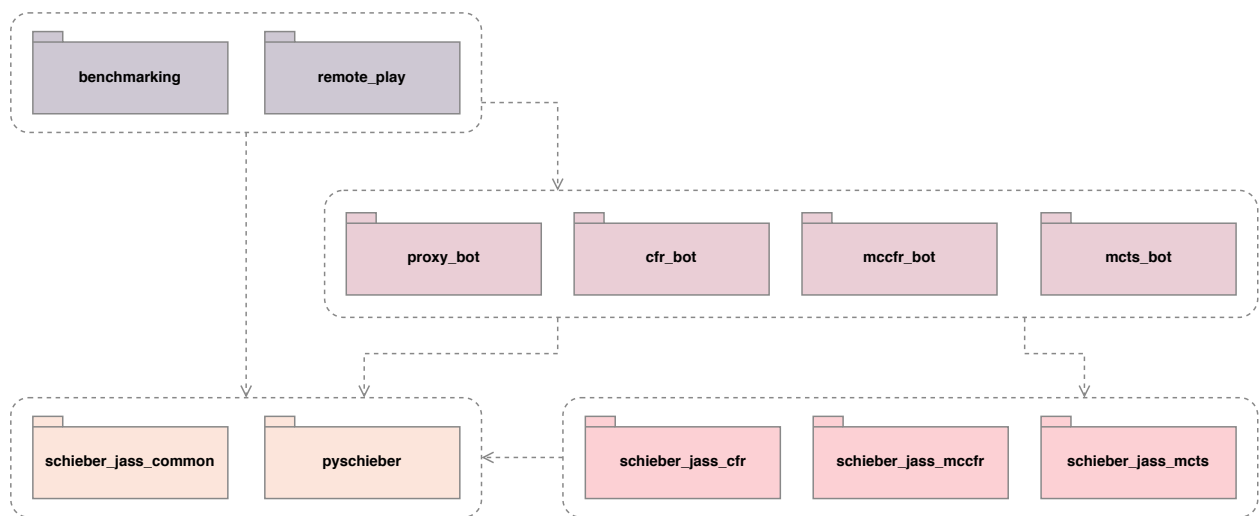


Abbildung 13: Übersicht der Python Packages und ihrer Abhängigkeiten

Package	Beschreibung
schieber_jass_common	Erweitert das Pyschieber Package um allgemeine Funktionalität, die von den implementierten Algorithmen benötigt wird. Dazu gehören Themen wie Kartenbuchhaltung, Kartenverteilung und die Analyse von aktuellen sowie zukünftigen Spielzuständen.
schieber_jass_cfr	Implementation des CFR Algorithmus. Mehr dazu in Kapitel 8.2.
schieber_jass_mccfr	Implementation des MCCFR Algorithmus. Mehr dazu in Kapitel 8.3.
schieber_jass_mcts	Implementation des MCTS Algorithmus. Mehr dazu in Kapitel 8.4.
cfr_bot	Pyschieber Bot mit CFR Algorithmus für die Wahl der nächsten auszuspielenden Karte.
mccfr_bot	Pyschieber Bot mit MCCFR Algorithmus für die Wahl der nächsten auszuspielenden Karte.
mcts_bot	Pyschieber Bot mit MCTS Algorithmus für die Wahl der nächsten auszuspielenden Karte.

proxy_bot	Spezieller Pyschieber Bot, welcher unter anderem für die Benchmark Umgebung entwickelt wurde.
benchmarking	Beinhaltet die Benchmark Umgebung, um Bots in konfigurierbaren Spielsituationen und Spieleraufstellungen im Schieber Jass gegeneinander antreten zu lassen. Mehr dazu in Kapitel 7.
remote_play	Integriert Bots in die grafische Spielumgebung von Zühlke [7], damit man mit menschlichen Spielern zusammenspielen bzw. gegen die Bots antreten kann.

Tabelle 15: Beschreibung der Python Packages

6.2 Cluster Deployment

Für rechenintensive Aufgaben - wie beispielsweise die Ausführung von Benchmarks - konnte auf den Cluster vom Institut für Software (IFS) an der HSR zurückgegriffen werden. Es handelt sich dabei um leistungsstarke Hardware, auf der man via Docker eigene Applikationen ausführen kann. Zur Verfügung standen 48 Prozessoren und rund 540 Gigabyte Arbeitsspeicher. Abbildung 14 zeigt einen der beiden Anwendungsfälle, wie dieses System während der Bachelorarbeit genutzt wurde.

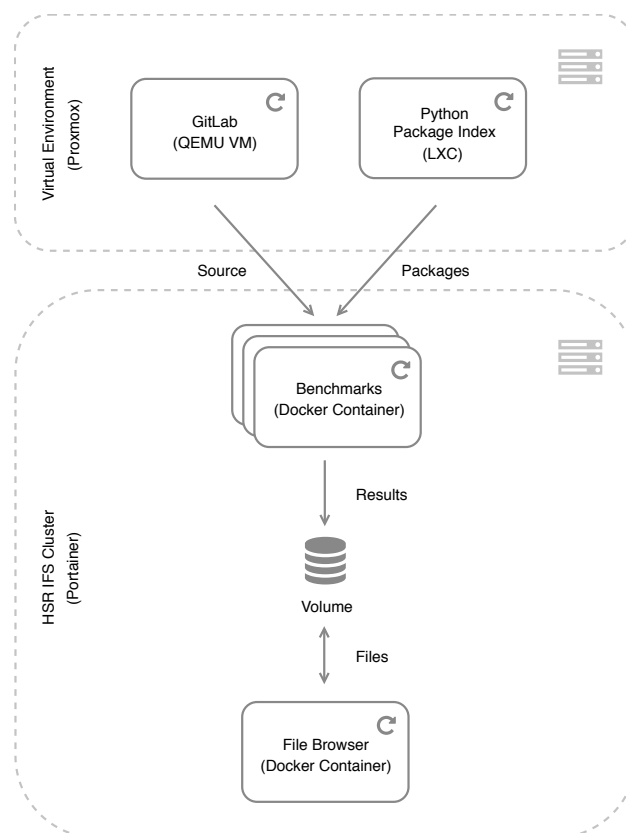


Abbildung 14: Ausführung von Benchmarks auf dem Cluster

Die virtuelle Umgebung mit Proxmox war schon gegeben, da sie vor Beginn des Studiums aufgesetzt wurde und währenddessen immer wieder zum Einsatz kam. Neu eingerichtet wurde allerdings ein privater Package Index, sozusagen eine Datenbank für freigegebene Versionen von Python Packages. Zusammen mit der eigenen GitLab Instanz konnte so der Zugang für den Cluster sowohl direkt auf den Source Code als auch auf Zwischenversionen der eigenen Packages geschaffen werden.

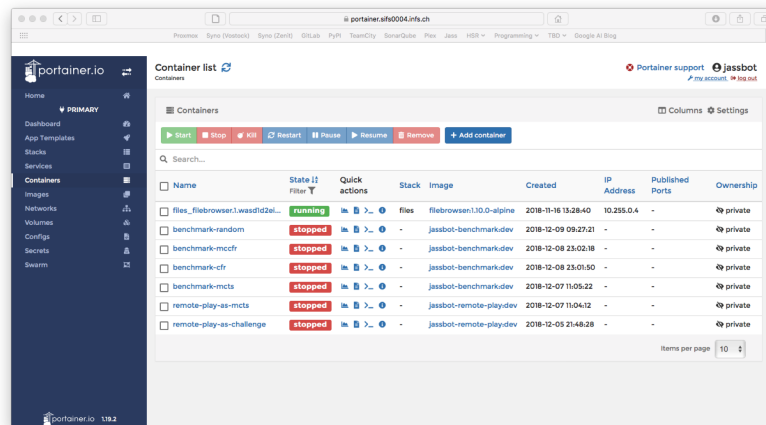


Abbildung 15: Verwaltung der Docker Container auf dem Cluster

Für die Ausführung von Benchmarks auf dem Cluster wurde ein Docker Image¹ erstellt, welches sich den neusten Stand der Benchmark Umgebung von GitLab herunterlädt, die darin referenzierten Packages über den Index bezieht, und sie anschliessend bei sich installiert. Hat bis dahin alles funktioniert wie erwartet, dann wird der Benchmark gestartet. Je nach Konfiguration kann dessen Ausführung mehrere Stunden in Anspruch nehmen. Zum Schluss werden die Resultate des Benchmarks auf ein Docker Volume geschrieben, damit sie nach Herunterfahren des Docker Containers nicht verloren gehen.

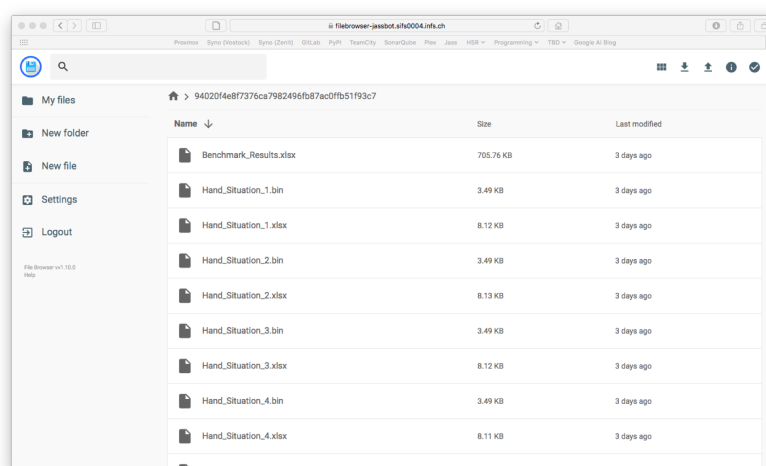


Abbildung 16: Grafische Dateiverwaltung auf dem Cluster

¹Docker nutzt Images als Vorlage zur Erzeugung von Containern für die Ausführung.

Für Verwaltung und Zugriff auf die mittels Docker Volume persistierten Benchmark Resultate wurde ein weiteres Docker Image erstellt, welches eine Webapplikation zur Dateiverwaltung [16] ausführt. Damit können Dateien aller Art auf dem Docker Volume direkt im Browser verwaltet und heruntergeladen werden. Dieser Lösungsansatz zur Dateiverwaltung in einer Docker Umgebung hat sich während der Bachelorarbeit eigentlich ganz gut bewährt, auch wenn es dafür vermutlich elegantere Ansätze gegeben hätte.

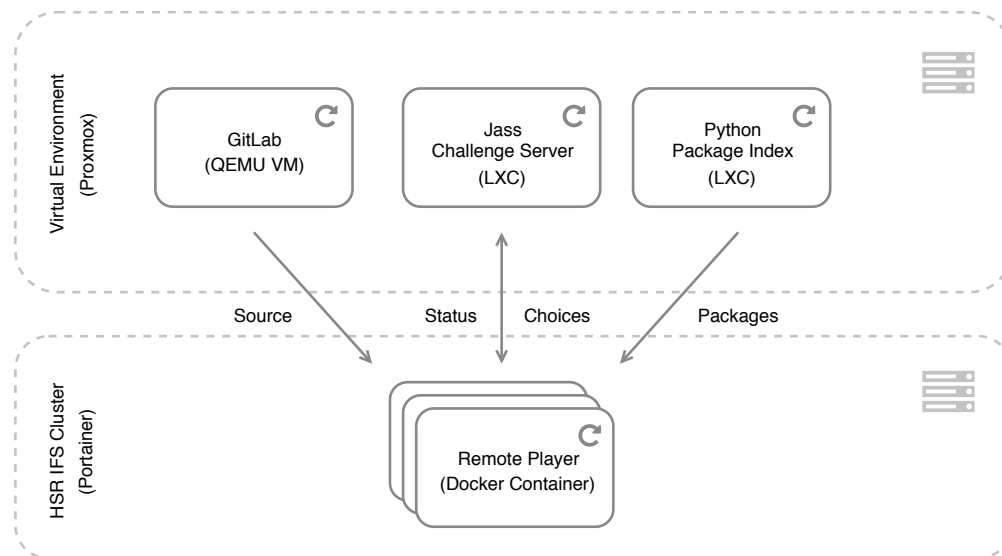


Abbildung 17: Integration der Bots in die grafische Spielumgebung via Cluster

Abbildung 17 zeigt den zweiten Anwendungsfall für den Cluster. Bei der Implementierung der verschiedenen Ansätze wurde schnell klar, dass die Ausführung der Bots generell sehr rechenintensiv ist. Grundidee war deshalb, dass man die Bots auf dem Cluster ausführt, wenn man damit menschlichen Spielern in einer grafischen Spielumgebung entgegentreten möchte. Das Pyschieber Package bringt für die Integration bereits einen entsprechenden Adapter mit. Dieser verbindet den Pyschieber Bot via Websockets mit einer beliebigen Challenge Server Instanz, dem Backend für die grafische Spielumgebung von Zühlke [7]. Menschliche Spieler können über ihren Browser einem Spiel mit Bots beitreten. Als Entwickler muss man die Bots dadurch nicht mehr auf dem eigenen Rechner ausführen, wenn man sie jemandem demonstrieren will. Durch Parallelisierung ihrer Berechnungen könnten die Bots das volle Potenzial des Clusters ausschöpfen. Mehr Berechnungen sollten - zumindest bei richtiger Umsetzung der zugrundeliegenden Algorithmen - auch zu besseren Entscheidungen der Bots führen. Dies war zwar angedacht, aus zeitlichen Gründen jedoch nicht mehr realisierbar.

Hinweis: Möglichkeit der Parallelisierung

Ob und wie sich die eingesetzten Algorithmen parallelisieren lassen stellt nochmals eine ganz eigene Problemstellung dar. Es wäre auf jeden Fall spannend herauszufinden, ob sich das Spielniveau der Bots dadurch massgeblich steigern lässt. Nachteil davon ist, dass die Bots zu einem gewissen Grad an den Cluster gebunden werden, da sie auf einem handelsüblichen Computer das erwartete Spielniveau nicht mehr erreichen, ausser man ist dazu bereit, längere "Bedenkzeiten" der Bots in Kauf zu nehmen.

7 Benchmarking

Basierend auf den Anforderungen aus Kapitel 4.3 wurde ein virtueller Prüfstand entwickelt, auf dem verschiedene Bots gegeneinander im Schieber Jass antreten können. Abbildung 18 gibt einen Überblick des Testablaufs.

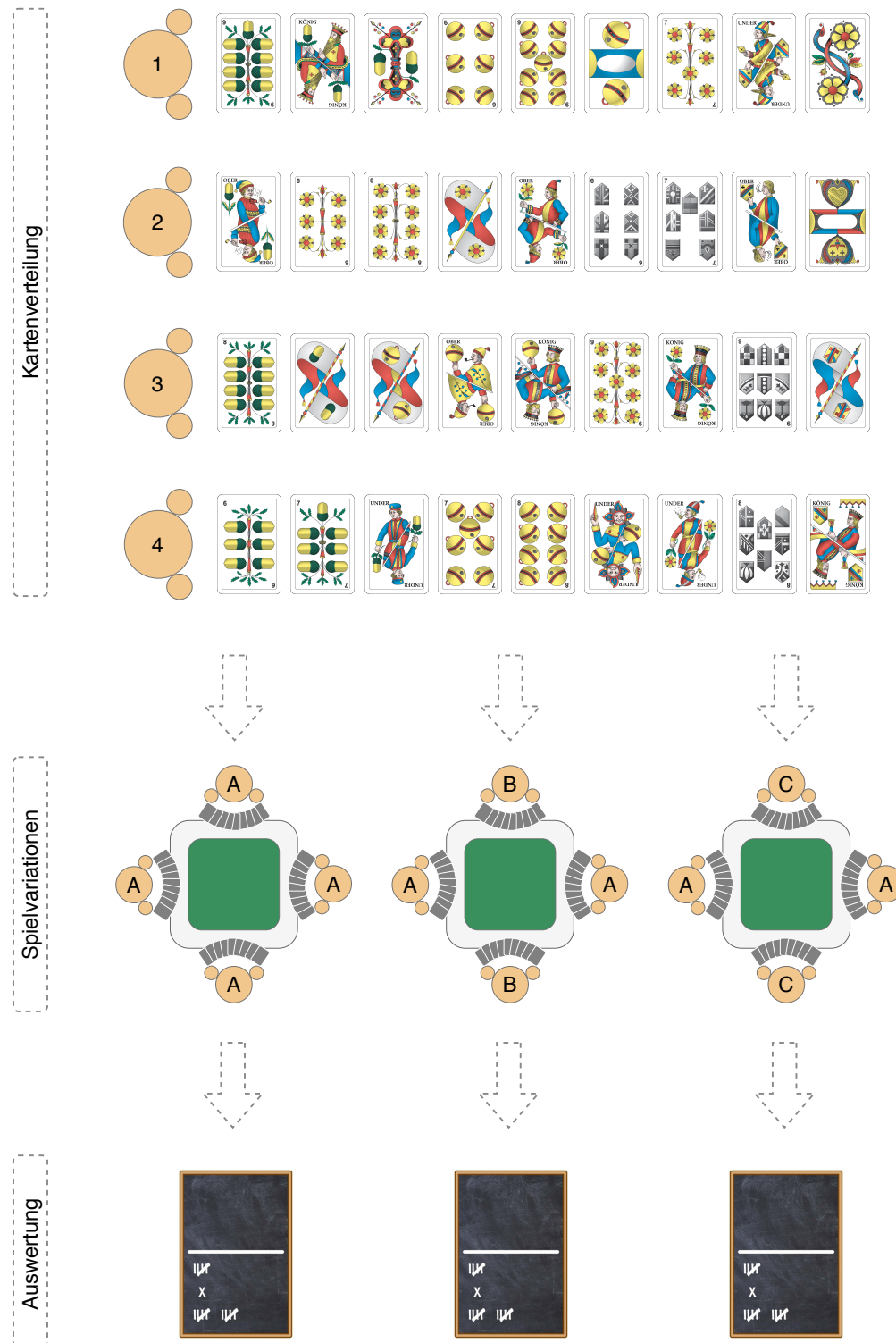


Abbildung 18: Übersicht des Testablaufs beim Benchmarking

7.1 Testablauf

1 Generierung von Kartenverteilungen

Als Ausgangslage für einen Testdurchlauf dient eine zufällige Kartenverteilung, nachfolgend auch als **Handsituation** bezeichnet. Im Prinzip kann man sich darunter 4 Stapel mit 9 zufällig verteilten Karten vorstellen. Die Anzahl der generierten Handsituationen entspricht der Anzahl verschiedener Spielsituationen, die man mit dem Benchmark abdecken möchte. Mit der Anzahl der betrachteten Spielsituationen sollte auch die Aussagekraft der Benchmark Resultate zunehmen, da der Effekt des “Kartenglücks” schwächer wird.

Hinweis: Randomisierung

Der Seed für die zufällige Generierung von Handsituationen ist konfigurierbar. Ein einzelner Seed führt immer zur gleichen Sequenz von zufällig generierten Handsituationen. Siehe dazu auch die Parameterübersicht in Tabelle 18.

2 Vorbereitung des Spiels

Nun werden die Spieler bzw. die Bots an den Tisch gesetzt, sie bekommen damit ihre Handkarten auf Basis der generierten Handsituation zugewiesen. Um welche konkreten Bots es sich dabei handelt, ist frei konfigurierbar. Für die in Kapitel 9 vorgestellten Resultate wurde die Aufstellung bzw. “Sitzordnung” aus Tabelle 16 verwendet.

Sitzposition	Team	Spieler
1	1	Proxy Bot
2	2	Challenge Bot
3	1	Proxy Bot
4	2	Challenge Bot

Tabelle 16: Mögliche Spieleraufstellung für Benchmarking

Team 2 setzt sich aus zwei regelbasierten Bots zusammen, welche für diese Bachelorarbeit als Referenzimplementation dienen. Als bisher ungeschlagene virtuelle Jass Spieler setzen sie die Messlatte, welche es mit eigenen Bots zu knacken gilt. Bei ihren Gegnern in Team 1 handelt es sich um zwei Proxy Bots.

Das Konzept für den Proxy Bot wurde während der Entwicklung dieser Benchmark Umgebung erarbeitet. Da man verschiedene Bots miteinander vergleichen will, braucht es eine Möglichkeit, die Bots in einem “fliegenden Wechsel” innerhalb eines vorgegebene Spiels auszutauschen. Man möchte damit herausfinden, wie unterschiedliche Bots eine vorgegebene Spielsituation handhaben. Vergleichbarkeit wird dadurch gewährleistet, dass die Bots bei ihrem

Einsatz exakt die gleiche Ausgangssituation vorfinden. Ein solcher Mechanismus wird durch den Proxy Bot ermöglicht. Er agiert quasi als transparente Instanz mit effektiven Bots im Hintergrund. Er leitet alle Aufrufe und Anfragen der Pyschieber Umgebung an die Bots im Hintergrund weiter.

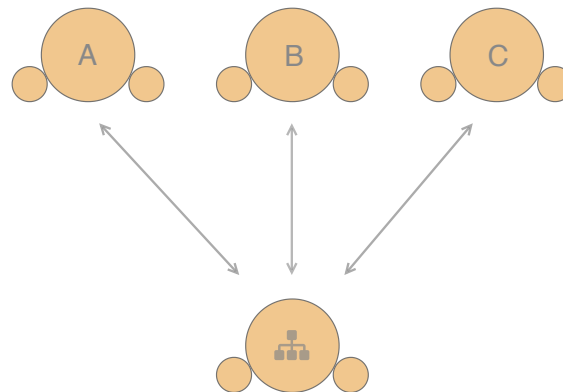


Abbildung 19: Schematische Darstellung des Proxy Bots

Die Bots im Hintergrund bleiben jederzeit durch den Proxy Bot über den aktuellen Spielstand informiert. Wenn es um eine Entscheidung geht - hier die Trumpfwahl oder das Ausspielen einer Handkarte - wird die Anfrage vom Proxy Bot basierend auf einer konfigurierbaren Strategie an einen der Bots im Hintergrund weitergeleitet. Dafür wurde das Strategy Pattern eingesetzt [17]. So könnte man beispielsweise als Strategie festlegen, dass die Trumpfwahl durch Bot "A" stattfindet und das Ausspielen der 9 Handkarten durch Bot "B" (siehe Abbildung 19). Gleichermassen kann im Rahmen eines Benchmarks die gleiche Spielsituation mehrfach zu Ende gespielt werden, einmal mit Bot "A", einmal mit Bot "B" und einmal mit "C", rein durch Ändern der internen Strategie vom Proxy Bot. Die Bots im Hintergrund des Proxy entsprechen somit den Bots, die durch den Benchmark miteinander verglichen werden sollen. Für die in Kapitel 9 vorgestellten Resultate wurde das Schema aus Tabelle 17 verwendet.

Bot hinter Proxy	Begründung
Challenge Bot	Zum Vergleich, inwiefern sich die getesteten Bot von der Referenzimplementierung abheben. Dient gleichzeitig der Kontrolle des Testablaufs. Erwartet wird, dass die Teams über mehrere Spielsituationen hinweg ein Unentschieden erreichen, da in diesem Fall 4 identische Bots gegeneinander antreten.
Random Bot	Zum Vergleich, inwiefern sich die getesteten Bots von zufälligem Spielverhalten abheben.
X	"X" steht in diesem Fall für einen Platzhalter. Hier wurde jeweils einer der Bots eingesetzt, welche während dieser Bachelorarbeit entwickelt wurden.

Tabelle 17: Schema für die Bots im Hintergrund des Proxy Bots

Hinweis: Flexibilität des Proxy Bots

Der Proxy Bot hat sich nicht nur im Kontext des Benchmarking bewährt. Wie in Kapitel 5.1 bereits angedeutet konnte für die Problematik der Trumpfwahl keine Lösung realisiert werden. Die entwickelten Bots sind also nicht in der Lage, die Trumpfwahl im Spiel vorzunehmen. Dank dem Proxy Bot kann man sie allerdings trotzdem ein komplettes Spiel bestreiten lassen, indem man für die Trumpfwahl via Strategie des Proxy Bots einen anderen Bot - beispielsweise den Challenge Bot - aktiviert. Die Fähigkeiten verschiedener Bots lassen sich damit beliebig kombinieren.

Im letzten Vorbereitungsschritt wird das Spiel nach Vorgabe der Benchmark Parameter in die gewünschte Ausgangssituation gebracht. Dafür wird bis zu einer bestimmten Anzahl Karten gespielt, bevor die eigentliche Messung beginnt. Die zu vergleichenden Bots nehmen passiv am Spiel teil, kommen aber erst bei der Messung wirklich zum Einsatz. Bis die Ausgangssituation erreicht ist dürfen sie das Spiel nur beobachten. Die Anzahl der zu spielenden Karten während der Messung ist frei wählbar, wobei 36 Karten der Messung eines kompletten Spiels entsprechen. 8 Karten würde einer Messung der letzten beide Stiche im Spiel entsprechen.

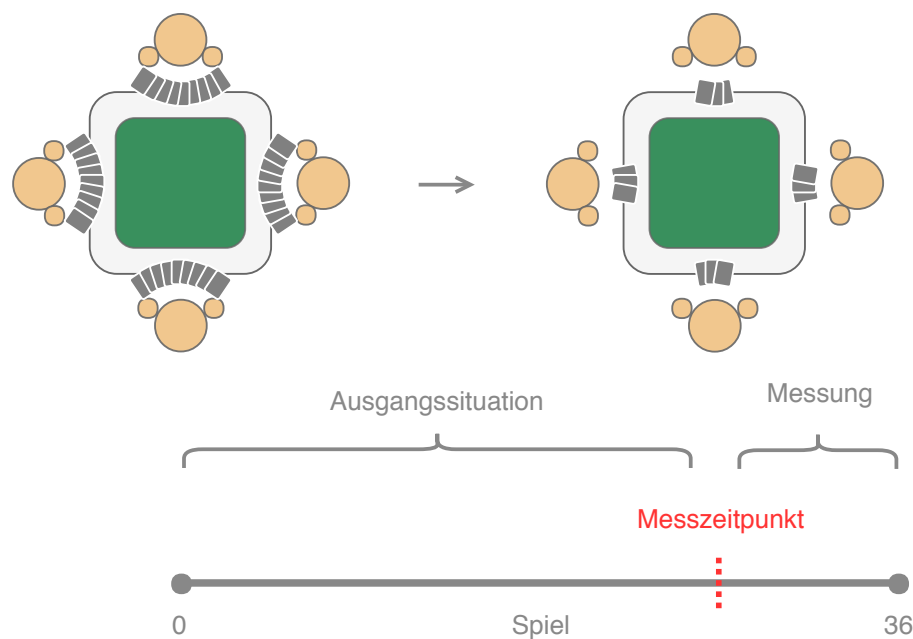


Abbildung 20: Das Spiel wird vor der Messung in die Ausgangssituation gebracht

Typischerweise will man das Spielniveau der Bots während eines kompletten Spiels messen, womit dieser letzte Vorbereitungsschritt eigentlich wegfällt. Es gibt jedoch Ausnahmen. Der in Kapitel 8.2 beschriebene Ansatz ist derart rechenintensiv, dass er sich nur für die letzten beiden Stiche eines Jass Spiels anwenden lässt. Die Messung des dazugehörigen Bots kann somit nur in den letzten zwei Stichen gemacht werden. Dafür braucht es eine Ausgangssituation, in der bereits 28 Karten ausgespielt wurden. Erst ab diesem Zeitpunkt kann der Bot eingewechselt werden um herauszufinden, wie gut er den verbleibenden Rest des Spiels bestreitet.

3 Austragung des Spiels

Diese Phase entspricht der eigentlichen Messung, und besteht grundsätzlich aus der Ausführung einer oder mehrerer **Spielvariationen**. Pro Spielvariation wird ein Bot eingewechselt, um das vorbereitete Spiel fertig zu spielen. Das Einwechseln der Bots ist über die interne Strategie der am Tisch sitzenden Proxy Bots gelöst. Eine Spielvariation gibt somit vor:

- wie die Strategie der Proxy Bots am Tisch für die Trumpfwahl aussieht.
- wie die Strategie der Proxy Bots am Tisch für das Ausspielen von Handkarten aussieht.

Hinweis: Wiederholung der Spielvariation

Bei Bedarf kann jede Spielvariation mehrfach wiederholt werden. Damit lässt sich insbesondere feststellen, wie stabil das Spielverhalten eines Bots in einer bestimmten Spielsituation ist. Die meisten der Bots spielen nicht - oder zumindest nicht bei jedem Spielzug - deterministisch. Bei nur einer Ausführung hat man somit auch nur eine mögliche Momentaufnahme, wie sich der Bot entschieden haben könnte. Siehe dazu auch die Parameterübersicht in Tabelle 18.

4 Auswertung

Für jede ausgeführte Spielvariation wird es einen möglichen Spielausgang geben. Der Ausgang gilt für ein Jass Spiel auf 157 Punkte und umfasst:

- Zeitdauer der Ausführung
- Trumpfwahl
- Gespielte Karten bei Ausgangssituation
- Gespielte Karten bei Spielende
- Punktzahl pro Team bei Ausgangssituation
- Punktzahl pro Team bei Spielende

Ein Spielausgang wird zusammen mit den Rahmenbedingungen, unter denen der Ausgang entstanden ist, zusammengefasst zu einem Resultatwert des Benchmarks. Pro Handsituation und Spielvariation wird ein solcher Resultatwert generiert, siehe Abbildung 21.

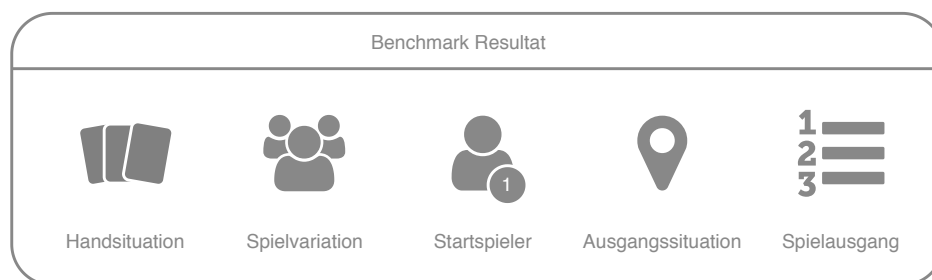


Abbildung 21: Zusammensetzung eines Benchmark Resultats

Hinweis: Wiederholung der Spielvariation

Bei mehrfacher Wiederholung der gleichen Spielvariation (siehe Hinweis bei Schritt 3) werden die Durchschnittswerte der dadurch entstandenen Spielausgänge für die Zusammensetzung des Resultatwerts verwendet.

Im letzten Schritt werden die Resultatwerte gruppiert nach Handsituation. Dies ermöglicht den Vergleich der getesteten Bots in verschiedenen Spielsituationen anhand des jeweiligen Spielausgangs. Die Resultate werden nach Ausführung des Benchmarks als Excel Datei abgespeichert. Pro Handsituation wird zusätzlich jeweils eine separate Excel Datei mit einer genauen Beschreibung erstellt. Dazu gehört eine gleichnamige Binärdatei, welche die Ausgangssituation in Form persistierter Python Objekte enthält. Damit lassen sich einzelne Handsituationen des Benchmarks durch Laden dieser Binärdatei bei Bedarf wiederholen bzw. das Verhalten der Bots in dieser Situation genauer untersuchen.

Name	Date Modified	Size
 Benchmark_Results.xlsx	9 Dec 2018 at 15:02	1 MB
 Hand_Situation_1.bin	9 Dec 2018 at 12:06	4 KB
 Hand_Situation_1.xlsx	9 Dec 2018 at 12:06	8 KB
 Hand_Situation_2.bin	9 Dec 2018 at 12:06	4 KB
 Hand_Situation_2.xlsx	9 Dec 2018 at 12:06	8 KB
 Hand_Situation_3.bin	9 Dec 2018 at 12:06	4 KB
 Hand_Situation_3.xlsx	9 Dec 2018 at 12:06	8 KB
 Hand_Situation_4.bin	9 Dec 2018 at 12:06	4 KB

Abbildung 22: Generierte Dateien mit den Resultaten eines Benchmarks

7.2 Unabhängigkeit

Für eine statistische Auswertung des Spielniveaus (i.e. “Bot A spielt besser als Bot B”) werden unabhängige Stichproben - in diesem Fall Spielausgänge - benötigt. Somit müssen die beim Benchmark durchgeführten Experimente voneinander unabhängig sein. Es wurde deshalb versucht, die Experimente so zufällig wie möglich zu gestalten, um dieser Eigenschaft gerecht zu werden:

- Zufällige Generierung der Kartenverteilungen
- Zufällige Wahl des Startspielers

Bei der Messung basierend auf einem bereits begonnen Spiel - wie es beispielsweise für den in Kapitel 8.2 beschriebenen Ansatz gemacht wird - müsste man konsequenterweise sicherstellen, dass die dafür simulierten Spielzüge ebenfalls zufälliger Natur sind. Die Benchmark Umgebung würde solches Verhalten grundsätzlich unterstützen. So könnte man pro Handsituation für alle vier Spieler am Tisch Random Bots einsetzen, das Spiel bis zur gewünschten Anzahl Karten vorspielen, und danach mit der eigentlichen Messung der Bots beginnen. **Voraussetzung dafür ist, dass die zu vergleichenden Bots jederzeit in der Lage sind, ein bereits begonnenes Spiel fertig zu spielen.**

Hinweis: Vorbelastete Ausgangssituationen

Leider führt eine technische Limitation im Challenge Bot dazu, dass dieser nicht in der Lage ist, ein Spiel ab einem beliebigen Zeitpunkt fertig zu spielen. Aufgrund von Seiteneffekten beim Auswählen einer Karte muss der Challenge Bot zwingend bei allen 9 Stichen eines Spiels die Entscheidung treffen, welche Karte ausgespielt werden soll. Ansonsten ist es mit der aktuellen Implementierung (Stand Dezember 2018) dieses Bots im Pyschieber [8] nicht möglich, Messungen für Teilbereiche eines Spiels vorzunehmen. Im Endeffekt führt diese Limitation dazu, dass der Challenge Bot eingesetzt werden muss, um Spiele in eine bestimmte Ausgangssituation zu bringen. Zumindest, wenn man für das restliche Spiel einen Vergleich des Spielausgangs zwischen anderen Bots und dem Challenge Bot anstellen will. Es besteht deshalb Grund zur Annahme, dass die dadurch generierten Ausgangssituationen zu einem gewissen Grad “vorbelastet” sind.

Hinweis: Realitätsnahe Ausgangssituationen

Während der Bachelorarbeit kam mehrfach die Diskussion auf, ob man die Ausgangssituationen nicht aus realen Spielaufzeichnungen extrahieren könnte, statt sie mit Bots zu generieren. Damit wäre gewährleistet, dass es sich dabei um mehr oder weniger realitätsnahe Situationen handelt. Natürlich unter der Annahme, dass die menschlichen Spieler bei der Aufzeichnung nicht völlig sinnfrei gespielt haben. Spielaufzeichnungen standen zwar zur Verfügung, der Ansatz konnte aus zeitlichen Gründen jedoch nicht mehr ausprobiert werden.

Zufällige Vorgänge wurden mit Hilfe des in Python standardmässig verfügbaren Generators für pseudozufällige Zahlen [18] simuliert.

7.3 Parametrisierung

Die Parametrisierung des Benchmarks muss direkt im Code vorgenommen werden. Tabelle 18 beschreibt die zur Verfügung stehenden Parameter und ihre Bedeutung. Für mehr Informationen soll an dieser Stelle auf den Source Code im *benchmarking* Package verwiesen werden.

Name	Wert	Beschreibung
card_shuffle_seed	Zahlenfolge / Zeichenfolge	Seed für die Generierung zufälliger Handsituationen. Der gleiche Seed wird immer zur gleichen Sequenz von Handsituationen führen.
teams_setup	4 Bots in 2 Teams	Zusammensetzung aller am Benchmark beteiligten Bots, aufgeteilt in zwei Teams. Mittels Proxy Bot können Bots in Spielvariationen ausgewechselt werden, um sie miteinander zu vergleichen.

nr_of_hand_card_situations	Zahl grösser 0	Steuert die Anzahl der Spielsituationen, die durch den Benchmark simuliert werden.
nr_of_cards_to_play	Zahl zwischen 1 und 36	Definiert die Anzahl der Karten, die während der Messung durch die getesteten Bots ausgespielt werden sollen. 36 entspricht der Messung eines kompletten Spiels. Bei einer Zahl kleiner 36 wird vor der Messung eine entsprechende Ausgangssituation generiert.
game_finish_variations	Spielvariationen	Eine Spielvariation entspricht im Prinzip einer Änderung der internen Strategie für die am Benchmark beteiligten Proxy Bots. Für eine generierte Handsituation wird jede Spielvariation mindestens einmal ausgeführt bzw. ein möglicher Spielausgang dafür produziert.
nr_of_samples_per_variation	Zahl grösser 0	Anzahl Wiederholungen, die beim Ausführen einer Spielvariation gemacht werden sollen. Empfohlen wird ein Wert von 1, um unabhängige Experimente durchzuführen (Wiederholungen entsprechen abhängigen Experimenten). Wiederholungen sind primär hilfreich, um die Stabilität der durch die Bots getroffenen Entscheidungen zu messen.
nr_of_cpu_cores_to_use	Zahl grösser 0	Gibt die Anzahl der Prozesse vor, die der Benchmark zur Abarbeitung der generierten Handsituationen erstellen wird.
output_directory_path	Dateipfad	Pfad zu einem Ordner auf dem Dateisystem, in dem die Benchmark Resultate abgelegt werden sollen.

Tabelle 18: Parameter für die Benchmark Umgebung

8 Lösungsansätze

8.1 Spielzustand

Für die nachfolgend vorgestellten Ansätze musste eine gemeinsame Grundlage geschaffen werden. Alle der untersuchten Lösungsansätze benötigen in irgendeiner Form Informationen zum aktuellen **Spielzustand**. Im Kontext des Schieber Jass kann der Spielzustand durch folgende Attribute beschrieben werden:

Spielerliste

Natürlich handelt es sich hierbei nicht um die effektiven Spieler am Tisch, ansonsten könnte man ihnen direkt in die Karten schauen. Es handelt sich dabei vielmehr um korrekt am Tisch platzierte Stellvertreter mit einer eindeutigen Kennung. Diese Kennung wird genutzt, um Aktionen im Spiel einem der Spieler zuzuordnen.

Trumpfwahl

Die Trumpfwahl wird zu Beginn des Spiels durch den Startspieler vorgenommen. Dazu gehört auch ein Vermerk, ob die Wahl an den Spielpartner weitergegeben wurde, also ob “geschoben” wurde.

Gespielte Karten

Die Liste der gespielten Karten gehört zu den grundlegendsten Informationen eines Spiels. Sie widerspiegelt den bisherigen Spielverlauf. Neben der eigentlichen Karte wird dabei auch ein Verweis auf den Spieler hinterlegt, welcher die Karte ausgespielt hat. Es lassen sich daraus unter anderem Hinweise für die Kartenbuchhaltung gewinnen.

Mögliche Handkarten

Die Kartenbuchhaltung dient als Grundlage zur Generierung möglicher Kartenverteilungen. Ihr Informationsgehalt hängt stark vom bisherigen Spielverlauf ab, also welche Rückschlüsse hinsichtlich möglicher Handkarten die bisherigen Aktionen der Spieler zulassen. Näheres dazu in Kapitel 5.4.

Aktueller Spieler

Diese Information kann theoretisch relativ einfach über die Liste der gespielten Karten berechnet werden. Die Speicherung der Information stellt jedoch sicher, dass sie nicht permanent neu berechnet werden muss.

Um den Spielstand jederzeit griffbereit zu haben für die zu implementierenden Algorithmen wurden zwei Massnamen ergriffen. Erstens wurden Hilfsfunktionen erstellt, welche die aufgelisteten Attribute aus den Daten der Pyschieber Umgebung extrahieren und aufbereiten. Zweitens wurde für die Implementation eigener Pyschieber Bots eine zusätzliche Abstraktionsebene eingeführt. Diese besteht aus einer erweiterten Basisklasse für Bots mit dem Zweck, den Spielstand abzuspeichern und auf dem neusten Stand zu halten. Die Basisklasse nutzt dafür die Bot Schnittstelle vom Pyschieber in Kombination mit den bereits erwähnten Hilfsfunktionen. Aufgrund dieser Massnamen entstand schlussendlich das *schieber_jass_common* Package, welches bereits in der Beschreibung der Architektur (Kapitel 6) angesprochen wurde. Es diente bei der Umsetzung der vorgestellten Ansätze als gemeinsame Grundlage.

Die aus den Ansätzen resultierenden Algorithmen konnten nun mit wenig Aufwand in die Pyschieber Umgebung als Bot integriert werden. Die Implementation dieser Bots folgt immer dem gleichen Schema:

```
from schieber_jass_common.bookkeeping.player import BookkeepingPlayer

class MctsPlayer(BookkeepingPlayer):

    # Access to current game state is provided with self.game_state

    def __init__(self, name):
        super().__init__(name)
        # TODO: Initialize algorithm.

    def choose_card(self, state=None):
        # TODO: Run algorithm and pick a card based on the results.
```

Hinweis: Fehlende Trumpfwahl

Die Trumpfwahl wurde hier bewusst weggelassen, weil dafür im Rahmen der Bachelorarbeit keine Lösung ausgearbeitet werden konnte. Mittels Verwendung des in Kapitel 7 beschriebenen Proxy Bots können die eigenen Bots trotz fehlender Fähigkeit zur Trumpfwahl an Spielen in der Pyschieber Umgebung teilnehmen. Die Trumpfwahl wird dabei dem Challenge Bot überlassen. Dieser trifft eine Entscheidung basierend auf gängigen Regeln, welche auch bei menschlichen Spielern oft zur Anwendung kommen.

8.2 Counterfactual Regret Minimization

Motivation

Poker gilt als klassisches Beispiel für ein Imperfect Information Game. Daher gab es auch eine ganze Reihe von wissenschaftlichen Arbeiten zu diesem Spiel. Als Ausgangspunkt diente das Paper von “Deep Stack”. Deep Stack kombinierte klassische spieltheoretische Ansätze mit Deep Learning, um einen Algorithmus für Heads-Up No-Limit Poker umzusetzen. Der Algorithmus von Deep Stack schaffte es professionelle Pokerspieler im direkten Vergleich statistisch signifikant zu schlagen. Beim Heads-Up No-Limit Poker handelt es sich um ein Two-Player Zero-Sum Extensive Form Game mit Imperfect Information. Wie bereits in Kapitel 5.2 gezeigt wurde, kann der Schieber Jass (aus Sicht des Teams) in diese Problemklasse eingeordnet werden. Daher ist es naheliegend diesen Ansatz weiter zu verfolgen. [19]

Es wurde schnell klar, dass eine vollständige Implementation des Ansatzes im Rahmen dieser Arbeit kaum möglich sein würde. Deep Stack verwendet für einen Teil ihres Lösungsansatzes eine Form von Counterfactual Regret Minimization, um neue Strategien für den aktuellen Zustand des Spieles zu berechnen. Wie sich herausgestellt hat, bauen fast alle modernen Poker Programme in irgendeiner Form auf diesem Algorithmus auf. Daher schien es sinnvoll, sich weiter mit diesem Ansatz auseinanderzusetzen. Im Folgenden wird der Algorithmus in seiner Reinform (“Vanilla” CFR) erklärt und aufgezeigt, wie dieser auf den Schieber Jass angewendet werden kann.

Ansatz

Ziel des Algorithmus ist es, ein Nash Equilibrium für das Spiel zu approximieren. Also ein Strategie Profil pro Spieler zu berechnen, welches für jede mögliche Spielsituation eine möglichst erfolgreiche Strategie besitzt. Im Gegensatz zu einem Perfect Information Game kann bei einem Imperfect Information Game nicht jeder Subbaum einzeln analysiert werden, da der Spieler der am Zug ist, ja nicht weiss, welche Züge sein Gegner tätigen kann. Es ist daher nötig den ganzen Spielbaum mehrfach zu durchlaufen, um die beste Gegenstrategien zu finden. [20]

Problem: Zu grosse Komplexität des Schieber Jass

Wie in Kapitel 5.3 bereits gezeigt wurde, ist die Komplexität eines Schieber Jass viel zu hoch um überhaupt nur den ganzen Baum abspeichern zu können. Aus diesem Grund wird das Problem vereinfacht, indem nur für die letzten beiden Stiche des Spiels Lösungen berechnet werden. Ohne Berücksichtigung der Spielregeln und somit der Kartenbuchhaltung, kann das Spiel so noch auf 1440 verschiedene Arten enden.

Begriffe

Der **Regret** ist ein numerischer Wert der angibt, wie stark ein Spieler es bereut, eine gewisse Action ausgewählt zu haben. Ein Regret kann entweder: [20]

- **Positiv** sein: In diesem Fall hätte der Spieler lieber eine andere Action (Karte) ausgewählt. Er bereut also seine Entscheidung. [20]
- **Negativ** sein: In diesem Fall kann der Spieler das nächste Mal wieder diese Action (Karte) auswählen. Er ist also mit seiner Entscheidung zufrieden. [20]
- **Gleich** gross sein: In diesem Fall spielt es keine Rolle, ob der Spieler diese oder eine andere Action auswählt. [20]

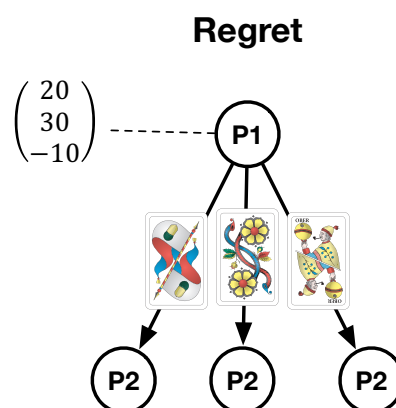


Abbildung 23: Darstellung des Regrets für den Schieber Jass

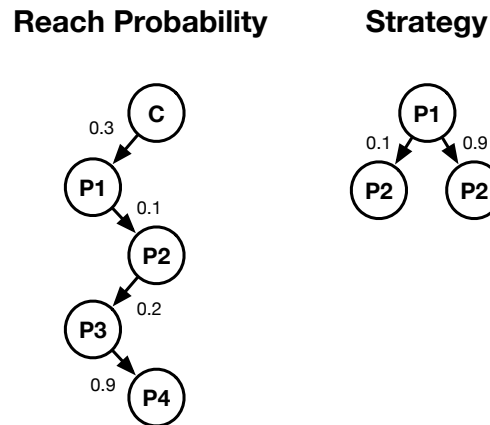


Abbildung 24: Reach Probability und Strategy

Die **Reach Probability** entspricht der Wahrscheinlichkeit, dass ein bestimmter Knoten im Baum überhaupt erreicht wird. Diese Wahrscheinlichkeit kann in einzelne Wahrscheinlichkeitsbeiträge der einzelnen Spieler aufgeteilt werden. Ebenfalls zu den Spielern gehört der so genannte **Chance Player**. Dieser ist für alle zufälligen Aktionen im Spiel zuständig (z.B. das zufällige Verteilen von Karten). Die aufmultiplizierten Wahrscheinlichkeitsbeiträge des Chance Player nennen wir **Chance Probability**. Die Wahrscheinlichkeitsbeiträge der Spieler werden aus der Strategie berechnet, indem man die einzelnen Wahrscheinlichkeiten aus den Strategien der Spieler aufmultipliziert [20]. Wie man zu den Strategiewerten kommt, wird später erklärt. In der Abbildung 24 wäre also die Wahrscheinlichkeit den letzten Knoten von Spieler 4 zu erreichen:

$$0.3 \cdot 0.1 \cdot 0.2 \cdot 0.9 = 0.0054$$

Die einzelnen Beträge wären dabei:

- Chance Player: 0.3
- Player 1: 0.1
- Player 2: 0.2
- Player 3: 0.9

Counterfactual Regret ist eine Art von gewichtetem Regret. Der Regret wird dabei mit den Wahrscheinlichkeitsbeiträgen der *anderen* Spieler gewichtet und jeweils auf dem Information Set eines Spielers aufsummiert. Der Counterfactual Regret zeigt also, wie sehr man es bereut eine Aktion ausgewählt zu haben und wird gewichtet mit dem Betrag, den die anderen Spieler dazu leisten, dass dieses Information Set erreicht wird. Zur Erinnerung: Ein Information Set entspricht der individuellen Sicht eines Spielers auf den aktuellen Zustand des Spiels. [20]

Regret Matching beschreibt den Prozess in welchem die Strategien für die Iterationen neu berechnet werden. Regret Matching normalisiert die positiven Anteile des aufsummierten Counterfactual Regrets und errechnet daraus eine neue Strategie. Falls es nur negative oder gleich grosse Regrets gibt, produziert Regret Matching eine gleichverteilte Strategie [20]. Also z.B.

$$\begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$$

Die **Average Strategy** ist die Strategie, welche über alle Iterationen des Algorithmus hinweg durchschnittlich gespielt wurde. Die durchschnittlichen Strategien werden jeweils pro Information Set berechnet. [20]

Algorithmus

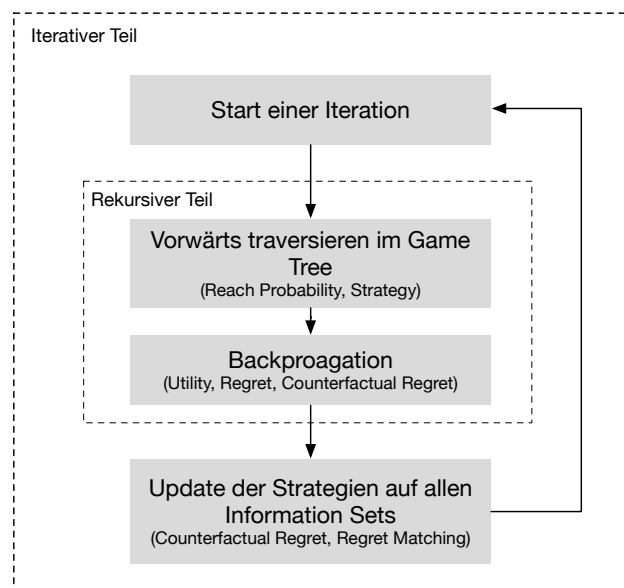


Abbildung 25: Übersicht über den Ablauf des “Vanilla” CFR

Der Algorithmus selbst besteht aus einem iterativen und einem rekursiven Teil. Einfach gesagt, spielt der Algorithmus ganz viele Spiele gegen sich selbst. Er passt dabei aber jedes mal die Strategien der einzelnen Spieler an. Je mehr Iterationen der Algorithmus durchläuft, desto genauer wird er das Nash Equilibrium approximieren.

In jeder Iteration wird der Spielbaum einmal rekursiv durchlaufen. In der Vorwärtsbewegung wird die History und somit der Baum schrittweise aufgebaut. Im gleichen Zug werden die Reach Probabilities mit Hilfe der aktuellen Strategien der Spieler berechnet. Sobald ein Terminal Node im Game Tree erreicht wurde, kann der Reward berechnet werden. Im Fall des Schieber Jass wäre dies die erreichte Punktzahl pro Team. In der Rückwärtsbewegung (Backpropagation) kann nun der Regret und somit auch der Counterfactual Regret berechnet werden.

Nach vollständiger Traversierung des Game Trees, werden auf allen Information Sets die bisherigen Strategien verworfen und neue Strategien aufgrund des aufsummierten Counterfactual Regrets berechnet (Regret Matching). Da die Strategien jeweils pro Iteration angepasst werden, wird der Counterfactual Regret immer kleiner und somit die Strategien immer besser

(führen zu einem besseren Reward). Das Nash Equilibrium entspricht am Schluss der Average Strategy über alle Iterationen hinweg. [13]

Resultat

Aus diesem Ansatz ist ein Bot entstanden, der in den letzten zwei Stichen eines Schieber Jass Spiels die Entscheidung zum Ausspielen einer seiner Handkarten treffen kann. Die Implementierung folgt dem Schema aus Kapitel 8.1, um immer den aktuellen Zustand des Spiels zur Verfügung zu haben. Die Funktionalität des Algorithmus wurde in ein separates Python Package namens *schieber_jass_cfr* ausgelagert. Die Implementation des Bots befindet sich im Package *cfr_bot*. Siehe Kapitel 6 für eine Übersicht aller Packages.

Name	Wert	Beschreibung
iterations	Zahl grösser 0	Steuert die Anzahl an Iterationen, die vom CFR Algorithmus durchlaufen werden, bevor der Bot eine Entscheidung trifft. Mehr Iterationen führen zu längeren “Bedenkzeiten” des Bots.

Tabelle 19: Mögliche Parameter für die Ausführung des CFR Bots

8.3 Monte Carlo Counterfactual Regret Minimization

Motivation

Mit dem Vanilla CFR Ansatz (Siehe Kapitel 8.2) war es bisher nicht möglich, mehr als die letzten beiden Stiche zu spielen. Damit ein potentieller Bot aber ein ganzes Spiel bestreiten kann, musste ein Ansatz gefunden werden, welcher mit der Komplexität des Game Trees umgehen kann. Hierfür gibt es in der Literatur mehrere Ansätze. Einer dieser Ansätze stellt eine Erweiterung des bereits vorgestellten CFR Algorithmus um eine Monte Carlo Komponente dar. Der Ansatz bietet sich an, da bereits Erfahrungen mit dem “Vanilla” Algorithmus gesammelt wurden. Als Grundlage wurde das Paper von Marc Lanctot “Monte Carlo Sampling and Regret Minimization for equilibrium computation and decision-making in large extensive form games” verwendet [13].

Ansatz

Beim Monte Carlo Counterfactual Regret Minimization (MCCFR) Ansatz wird nicht mehr der ganze Baum pro Iteration traversiert. Stattdessen wird pro Iteration nur ein Teil des ganzen Game Trees berücksichtigt. Die Berechnungen finden also nur auf einer Stichprobe (Sample) des Game Trees statt. Dies hat aber zu Folge, dass sich der Counterfactual Regret nicht mehr so einfach berechnen lässt. Der Algorithmus löst dieses Problem, indem er die gleichen Techniken wie im “Vanilla” CFR anwendet. Jedoch wird für den Counterfactual Regret ein Schätzer verwendet. [13, S. 42]

Begriffe

Es gibt grundsätzlich drei verschiedene Möglichkeiten den Baum in einzelne Samples aufzuteilen (Siehe Abbildung 26).

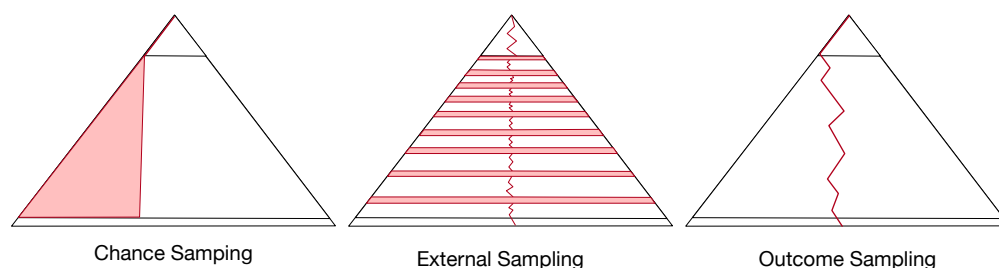


Abbildung 26: Sampling Varianten für den MCCFR

Chance Sampling

Beim Chance Sampling wird eine zufällige Kartenverteilung angenommen. Im Gegensatz zum “Vanilla” CFR müssen also nicht mehr alle Verteilungen berücksichtigt werden sondern nur noch eine. Für diese eine Verteilung muss aber trotzdem der ganze verbleibende Baum traversiert werden. Dies entspricht, wie ja bereits in Kapitel 5.3 ausgeführt

wurde (ohne die Regeln zu berücksichtigen), theoretisch etwa $3.9 \cdot 10^{33}$ möglichen Spiel-
ausgängen.

External Sampling

Beim External Sampling werden nicht alle externen Actions durchgerechnet. Externe Actions sind in diesem Fall alle Actions der anderen Spieler (inklusive dem Chance Player). Der Algorithmus probiert also nur die eigenen möglichen Actions vollständig durch. Bei den anderen Spielern wird jeweils nur eine Action ausgewählt.

Outcome Sampling

Beim Outcome Sampling wird jeweils genau eine Kartenverteilung und ein Spielausgang pro Iteration durchgerechnet. Es werden also genau 36 Nodes durchlaufen. Zudem wird so genau eine gültige "Terminal History" produziert. Gerade wenn der Algorithmus online (die Berechnung kann nicht im Vorherein durchgeführt werden) ausgeführt werden muss, eignet sich dieser Algorithmus besonders gut.

Aufgrund der kleineren Sample Grösse und der damit verbundenen Möglichkeit den Algorithmus auch kürzer laufen zu lassen, wurde das Outcome Sampling weiter verfolgt. Chance und External Sampling kommen aufgrund der grossen Menge an zu traversierenden Knoten pro Iteration nicht in Frage.

Um die Regrets berechnen zu können, brauchte man im CFR jeweils den Reward für jede einzelne Entscheidungsmöglichkeit. Da aber beim Monte Carlo CFR mit Outcome Sampling nicht mehr der ganze Baum durchlaufen wird, ist dies nicht mehr so einfach möglich. Stattdessen erhält man pro Iteration jeweils einen einzigen Reward. Es wird also so zu sagen nur ein Spiel komplett fertig gespielt. Somit lässt sich der Regret aber auch nicht mehr einfach berechnen. Stattdessen definiert der Algorithmus einen Schätzer für den Counterfactual Regret [13].

Für diesen Schätzer wird ein zusätzlicher Wahrscheinlichkeitswert benötigt. Dieser Wert gibt an wie wahrscheinlich es ist, dass ein Knoten in dieser Iteration im Sample enthalten ist. Er wird im Folgenden **Sample Probability** genannt. Der Schätzer funktioniert so, dass jeder einzelne Regret auf dem Information Set durch die Sample Probability geteilt wird. Die genaue Definition des Schätzers für den Counterfactual Regret kann im Paper von Marc Lanctot nachgelesen werden [13].

Um die Sample Probability berechnen zu können, muss man sich überlegen, wie die Stichprobe im Baum ausgewählt wird. Dabei ist einerseits wichtig, dass Information Sets, welche durch die Strategien der Spieler oft ausgewählt werden, auch im Sample enthalten sind. Andererseits ist es aber auch wichtig, dass neue - noch nie besuchte - Information Sets auch untersucht werden. Würde dies nicht geschehen, so würde man sich zu schnell auf die erste vielversprechende Möglichkeit stürzen. Die Berechnung im Algorithmus erfolgt dabei analog zur Reach Probability. Daher wird eine Strategie benötigt, welche sowohl dafür sorgt, dass neue Knoten im Baum untersucht werden und gleichzeitig aber auch die Strategien der Spieler berücksichtigt wird [13].

Ein **Sampling Profile** ist eine Set von Strategien, welche im Grunde beschreiben, wie das Sampling im Baum vorgenommen wird. Beim MCCFR wird nach einer so genannten **epsilon-on-policy** vorgegangen. Dabei wird auf jedem Knoten - mit Hilfe der aktuellen Strategy - eine Sampling Strategy berechnet. Diese wird anschliessend für die jeweilige Auswahl der Action

verwendet. Die Sampling Strategy wählt mit der Wahrscheinlichkeit ϵ zufällig eine Action und mit der Wahrscheinlichkeit $1 - \epsilon$ nach der aktuellen Strategie aus. Die folgende Formel zeigt, wie eine solche Strategie berechnet werden kann [13]:

$$\sigma' = \epsilon \cdot Unif + (1 - \epsilon) \cdot \sigma^t$$

wobei:

ϵ = Wahrscheinlichkeit, dass zufällig ausgewählt wird

σ^t = Mixed Strategy für die Iteration t (aktuelle Iteration)

σ' = Sampling Strategy

$Unif$ = Gleichverteilte Strategy (zufällige Auswahl)

Das folgende Beispiel soll zeigen, wie eine solche Berechnung funktioniert. In diesem Beispiel befindet sich der Algorithmus gerade in einem Information Set mit zwei Auswahlmöglichkeiten. Dabei ist $\epsilon = 0.6$ und die aktuelle Strategie σ^t ist

$$\sigma^t = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}$$

Somit ergibt sich folgende Strategie für das Sampling:

$$0.6 \cdot \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} + (1 - 0.6) \cdot \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix} = \begin{pmatrix} 0.58 \\ 0.42 \end{pmatrix}$$

Die Sampling Strategy auf dem Information Set, wählt also in **58%** der Fälle die erste und in **42%** der Fälle die zweite Action aus. Auf diese Weise ist sichergestellt, dass die berechneten Strategien berücksichtigt werden und zugleich aber auch neue Wege im Baum ergründet werden.

Algorithmus

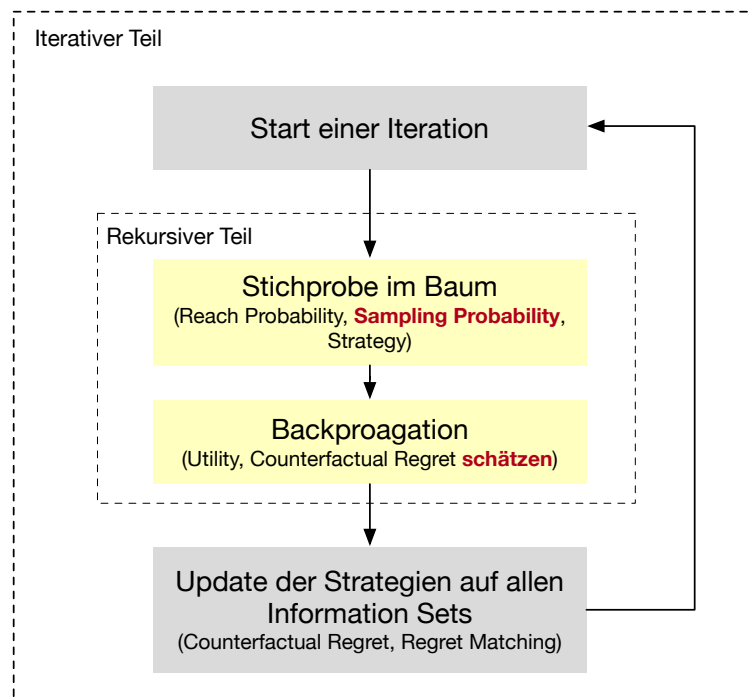


Abbildung 27: Übersicht über den Ablauf des MCCFR

Der Algorithmus hat - gleich wie die Vanilla Variante - einen iterativen und einen rekursiven Teil. In jeder Iteration wird aber genau *Kartenverteilung* untersucht. In der Vorwärtsbewegung im Baum wird nun nach dem Sampling Profile (epsilon-on-policy) vorgegangen. Im gleichen Zug werden die Reach Probabilities und die Sampling Probability berechnet. In der Rückwärts Bewegung werden die Counterfactual Regrets mit dem Schätzer bestimmt und auf dem Information Set aufsummiert. Im letzten Schritt werden auf allen besuchten Information Sets die Strategien mittels Regret Matching aktualisiert und eine neue Iteration kann beginnen (Siehe Abbildung 27).

Resultat

Aus diesem Ansatz ist ein Bot entstanden, der in jedem Stich eines Schieber Jass Spiels die Entscheidung zum Ausspielen einer seiner Handkarten treffen kann. Die Implementierung folgt dem Schema aus Kapitel 8.1, um immer den aktuellen Zustand des Spiels zur Verfügung zu haben. Die Funktionalität des Algorithmus wurde in ein separates Python Package namens *schieber_jass_mccfr* ausgelagert. Die Implementation des Bots befindet sich im Package *mccfr_bot*. Siehe Kapitel 6 für eine Übersicht aller Packages.

Name	Wert	Beschreibung
------	------	--------------

iterations	Zahl grösser 0	Steuert die Anzahl an Iterationen, die vom MCCFR Algorithmus durchlaufen werden, bevor der Bot eine Entscheidung trifft. Mehr Iterationen führen zu längeren “Bedenkzeiten” des Bots.
exploration_factor	Zahl grösser 0	Entspricht der Wahrscheinlichkeit ϵ im MCCFR Algorithmus.

Tabelle 20: Mögliche Parameter für die Ausführung des MCCFR Bots

8.4 Monte Carlo Tree Search

Motivation

Nach Umsetzung des “Vanilla” CFR Ansatzes im ersten Experiment aus Kapitel 8.2 liessen die Benchmark Resultate darauf hoffen, dass sich dieser Ansatz für den Schieber Jass tatsächlich bewähren könnte. Dies war zu Beginn nicht selbstverständlich, da sich die dafür auffindbare Literatur hauptsächlich mit dem Poker Spiel beschäftigt. Mit dem in Kapitel 8.3 beschriebenen Folgeexperiment wurde versucht, den Ansatz dahingehend zu erweitern, dass er auch für ein komplettes Spiel funktioniert, und nicht nur für das Ausspielen der letzten zwei Karten.

Parallel dazu sollte noch ein völliger neuer Ansatz ausprobiert werden. Anstoss dafür waren wissenschaftliche Arbeiten, die ihm Rahmen der Recherchearbeiten entdeckt wurden. Diese Arbeiten setzen sich mit Lösungen für stichbasierte Kartenspiele wie Skat [21] oder Doppelkopf [22] auseinander. Auch der Schieber Jass ist stichbasiert, wie in der Spielbeschreibung in Kapitel 5.1 bereits gezeigt wurde. Aufgrund dieser Ähnlichkeit schien es nur vernünftig, sich die in den Arbeiten vorgestellten Lösungsansätze genauer anzuschauen. Dabei haben sich Lösungen basierend auf Monte Carlo Tree Search (dt. Monte Carlo Baumsuche) Verfahren als besonders vielversprechend herausgestellt.

Dieses Kapitel soll eine kurze Einführung in die Funktionsweise von MCTS (Monte Carlo Tree Search) Algorithmen geben. Des Weiteren wird ein konkreter Lösungsansatz für den Schieber Jass vorgestellt, welcher im Rahmen dieses Experiments ausgearbeitet wurde. Dabei wird auch die dafür angewandte UCT (Upper Confidence Bound for Trees) Erweiterung für MCTS vorgestellt. Als Informationsquelle dienten die IEEE Publikation “A Survey of Monte Carlo Tree Search Methods” [23] sowie ein Blog Eintrag zum Thema von Kamil Czarnogórski [24].

Ansatz

Wie der Name bereits vermuten lässt wird das Spiel auch beim MCTS Ansatz in Form eines Baums (Game Tree) abgebildet. Die Knoten dieses Baums entsprechen dabei den möglichen Spielzuständen (Action Nodes). In Kapitel 8.1 wird näher darauf eingegangen, wie der Spielzustand für den Schieber Jass beschrieben werden kann. Die Übergänge zwischen den Knoten repräsentieren mögliche Spielzüge (Actions), welche vom Spielzustand im Ausgangsknoten zum Spielzustand im Zielknoten führen. Im Kontext des Schieber Jass entsprechen sie den erlaubten Handkarten, welche der aktuelle Spieler im Ausgangszustand ausspielen darf. Spielzustände und mögliche Spielzüge werden im Baum somit sequenziell abgebildet. Ein Beispiel dazu zeigt Abbildung 28.

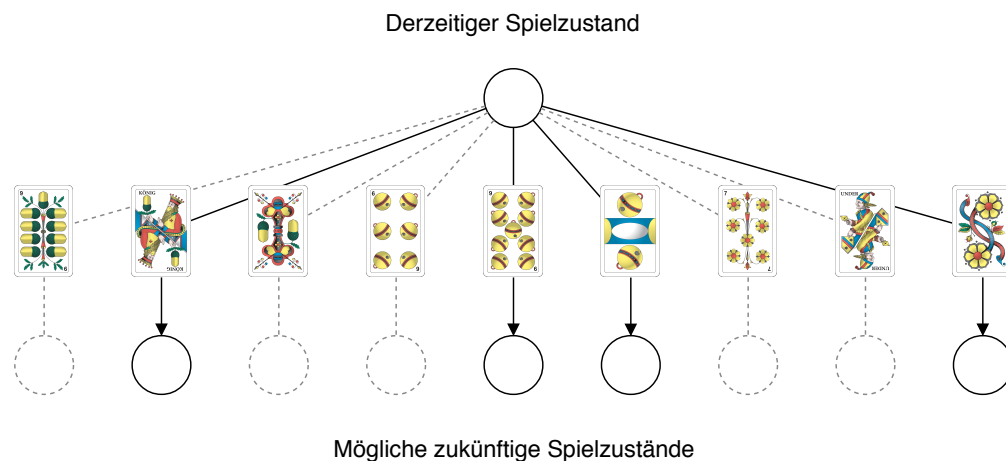


Abbildung 28: Knoten und Übergänge im Baum des MCTS

Die Berechnungen aus Kapitel 5.3 zeigen, dass es in der Praxis nicht möglich ist, ein komplettes Schieber Jass Spiel in einer solchen Baumstruktur auf einem Computer abzubilden. Die Anzahl der möglichen Spielzustände ist einfach zu gross in frühen Stadien des Spiels. Für Entscheidungsprobleme dieser Art sieht der MCTS Algorithmus einen iterativen Aufbau des Baums vor. Um dies zu ermöglichen werden einem einzelnen Knoten im Baum neben dem Spielzustand weitere Attribute zugeordnet.

Start

Da es sich beim Baum um einen gerichteten, azyklischen Graphen (sequenzieller Spielablauf) handelt gibt es dafür genau einen Einstiegspunkt. Dieser entspricht beim Schieber Jass dem Knoten für den Spielzustand, aus dem der MCTS Algorithmus gestartet wird. Der Knoten zeichnet sich dadurch aus, dass er keine Vorgänger besitzt, es gibt also keine auf ihn gerichteten Übergänge anderer Knoten.

Ende

Wurde das Spielende in einem Spielzustand erreicht, dann muss der dazugehörige Knoten entsprechend markiert werden. An dieser Stelle kann der Baum durch den MCTS Algorithmus nicht mehr weiter expandiert werden. Solche Knoten zeichnen sich dadurch aus, dass sie keine Nachfolger besitzen (Terminal Node), es gibt also keine davon ausgehenden Übergänge auf andere Knoten.

Anzahl Besuche

Im Prinzip ein einfacher Zähler der angibt, wie oft der MCTS Algorithmus bei diesem Knoten vorbeigekommen ist. Der Zähler wird jeweils um 1 erhöht, wenn der Knoten zu einer Simulation geführt hat.

Ausbaustadium

Ein Knoten kann entweder neu, teilweise ausgebaut oder vollständig ausgebaut sein. Der Ausbau bezieht sich auf Übergänge zu nachfolgenden Knoten, in diesem Fall also mögliche Karten, die im Spielzustand des Knotens vom aktuellen Spieler ausgespielt werden könnten. Für neue Knoten muss zuerst evaluiert werden, um welche Karten es sich dabei handelt. Ein Knoten gilt als vollständig ausgebaut, wenn alle seiner Übergänge bekannt

sind und die dadurch erreichbaren Knoten vom MCTS Algorithmus mindestens einmal besucht worden sind. Abbildung 29 illustriert die verschiedenen Stadien.

Bewertung

Spielzustände lassen sich aus der Perspektive einzelner Spieler bewerten. Indirekt bewertet man damit eigentlich die Spielzüge, welche schlussendlich zu einem bestimmten Zustand des Spiels geführt haben. Beim Schieber Jass kann diese Bewertungs jeweils aus der Sicht eines Team vorgenommen und interpretiert werden. Als Bewertungsgrundlage dienen erzielte Punkte im Spiel (Reward).

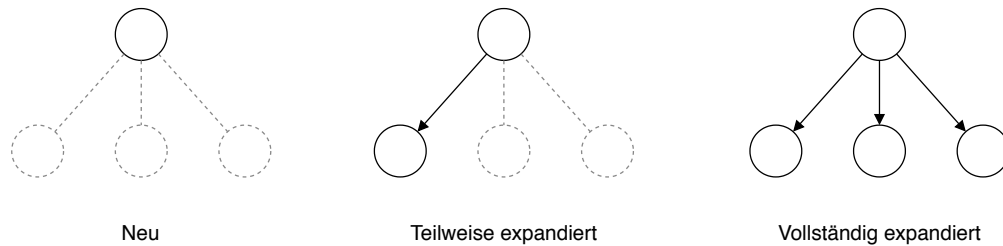


Abbildung 29: Ausbaustadien eines einzelnen Knotens bei MCTS

Basierend auf dieser Datenstruktur werden zukünftige Spielzustände und Spielzüge nun schrittweise durch den MCTS Algorithmus ausgebaut, analysiert und bewertet. So entsteht eine Abschätzung, welche Handkarten eines Spielers zum Erfolg des eigenen Teams führen könnten. Mehr Iterationen des Algorithmus bzw. Rechenzeit werden diese Abschätzung näher an die Realität bringen.

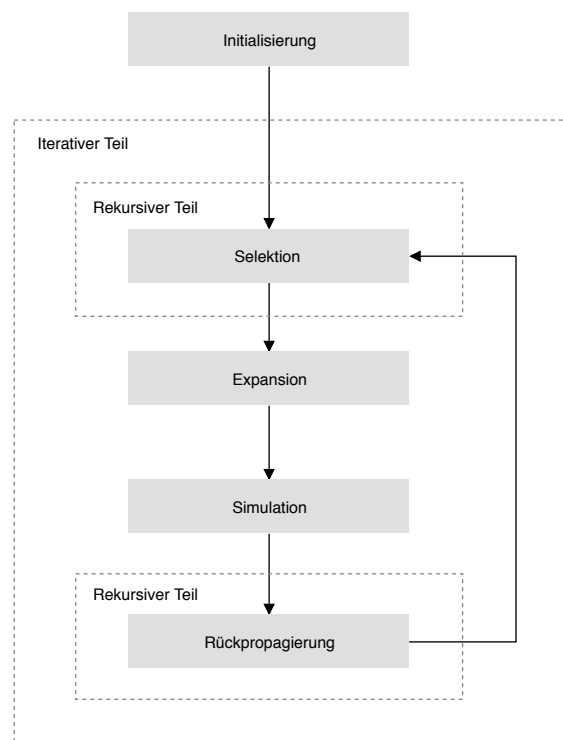


Abbildung 30: Ablauf des MCTS Algorithmus

Algorithmus

Abbildung 30 zeigt eine schematische Darstellung des Ablaufs vom MCTS Algorithmus. Die einzelnen Schritte sollen nachfolgend genauer erläutert werden mit Bezug auf das Schieber Jass Spiel.

1 Initialisierung

Der MCTS Algorithmus wird mit dem aktuellen Spielzustand initialisiert. Beim Schieber Jass geschieht dies jeweils zum Zeitpunkt, wenn der Spieler an der Reihe ist, und eine seiner Handkarten ausspielen muss. Die erlaubten Handkarten - unter Berücksichtigung der bereits auf dem Tisch liegenden Spielkarten - entsprechen den nächstmöglichen Spielzügen im Spiel. Sie sind der Einstiegspunkt zur Abschätzung möglicher Spielausgänge.

Die Kartenbuchhaltung aus Kapitel 5.4 spielt auch bei diesem Algorithmus eine wichtige Rolle. Sie muss während der gesamten Ausführung auf dem neusten Stand gehalten werden, da sie als Hauptinformationsquelle für den Aufbau des Baums dient.

2 Selektion

Dies ist der erste Schritt einer Iteration des MCTS Algorithmus. Ausgehend vom derzeitigen Spielzustand wird der Baum so lange traversiert, bis der Algorithmus auf einen Knoten trifft, welcher noch nicht vollständig expandiert wurde. Das Traversieren im Baum entspricht dabei der Annahme von theoretisch möglichen Spielzügen der Spieler in der Zukunft. Es wird im späteren Verlauf dieses Kapitels nochmals genauer darauf eingegangen, nach welchen Kriterien die Traversierung stattfindet.

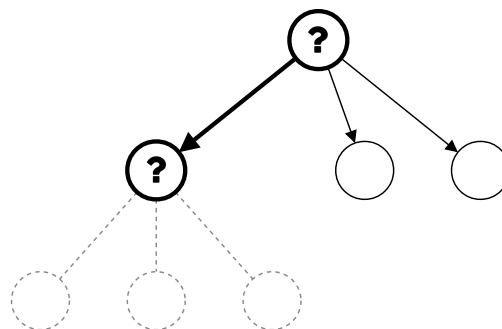


Abbildung 31: Selektion im Baum durch MCTS Algorithmus

Hinweis: Annahme möglicher Spielzüge

Annahmen der Form “Unter der Bedingung, dass Spieler 1 die Schellen 6 Karte spielen wird, Spieler 2 die Schellen König Karte, Spieler 3 die Eichel 9 Karte, Spieler 4 die Schellen Ass Karte, ...” führen zu einem möglichen zukünftigen Zustand im Spiel.

Abbildung 31 soll die Selektion vereinfacht darstellen. In der Praxis findet sie allerdings über mehrere Ebenen des Baums hinweg statt. Es wird quasi ein möglicher Pfad im Baum gewählt.

Dabei landet man irgendwann in einem Spielzustand, für welchen die daraus möglichen Spielzüge noch nicht vollständig ausgelotet worden sind, oder bei dem das Spiel ein Ende gefunden hat. Vom dazugehörigen Knoten aus geht es weiter mit Schritt 3.

3 Expansion

Der Algorithmus wechselt während seiner Ausführung ständig die Spielerperspektive, aus welcher er weitere mögliche Spielzüge betrachten muss. Die theoretisch möglichen Spielzüge jedes Spielers am Tisch können von der permanent mitzuführenden Kartenbuchhaltung abgeleitet werden. Der Perspektivenwechsel ergibt sich durch den vorgeschriebenen Spielablauf, siehe dazu auch die Spielregeln in Kapitel 5.1.

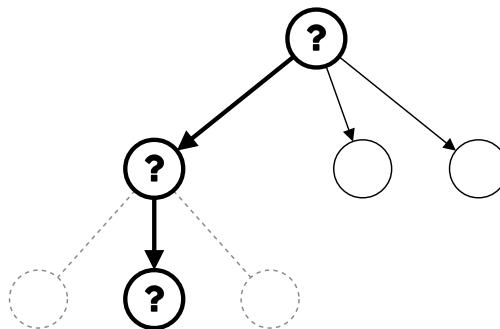


Abbildung 32: Expansion des Baums durch MCTS Algorithmus

Der selektierte Knoten aus Schritt 2 wird nun expandiert, indem **zufällig** für einen der nächstmöglichen Spielzüge ein Übergang zum daraus resultierenden Spielzustand erstellt wird. Der Baum wird somit um einen Knoten erweitert. Dieser Schritt fällt weg, wenn das Spielende im Spielzustand des selektierten Knotens bereits erreicht wurde, da es in einem solchen Fall keine weiteren Spielzüge mehr gibt.

4 Simulation

Nach Erstellung des neuen Knotens in Schritt 3 soll nun der dazugehörige Spielzustand bewertet werden. Dafür wird das Spiel ausgehend vom neuen Zustand bis zum Ende simuliert. In der Simulation werden die Spieler ihre Handkarten **zufällig** ausspielen. Sie bekommen ihre verbleibenden Handkarten unter Berücksichtigung der Kartenbuchhaltung zu Beginn der Simulation **zufällig** zugewiesen. Abbildung 33 visualisiert diesen Vorgang.

Hinweis: Realitätsnähe der Simulation

Aufgrund der zufälligen Spielzüge kann die Realitätsnähe der Simulation durchaus hinterfragt werden. Schliesslich werden die echten Spieler nicht einfach zufällig ihre Karten auf den Tisch legen für das restliche Spiel. Es handelt sich hierbei sozusagen um die domänenunabhängige Standardausführung der Simulation. In der Praxis wird die Simulation typischerweise um domänenspezifisches Expertenwissen – in diesem Fall wären das gän-

gige Spielmuster des Schieber Jass - erweitert, um die Resultate des MCTS Algorithmus weiter zu verbessern.

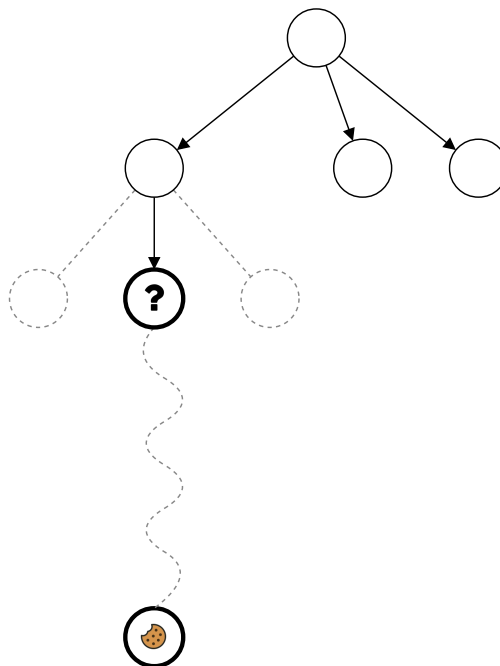


Abbildung 33: Simulation zur Bewertung von Knoten durch MCTS Algorithmus

Am Ende der Simulation gibt es pro Team eine erreichte Punktzahl, welche im Rahmen dieser Anwendung des MCTS Algorithmus für die Bewertung von Spielzuständen verwendet wird. Die Bewertung des Spielzustands gibt Auskunft darüber, ob die zuvor gemachten Spielzüge dem eigenen Team dienlich waren oder nicht. Man könnte an dieser Stelle also auch von einer Belohnung sprechen, was in Abbildung 33 durch einen Keks symbolisiert wird.

Die Simulation ist mit Abstand der zeitaufwendigste Schritt des MCTS Algorithmus. Da mehr Iterationen bei korrekter Implementierung zu besseren Entscheidungen führen sollten, könnte sich eine Optimierung der Simulation durchaus lohnen.

Hinweis: Alternativen zur Simulation

Im Endeffekt ist die Simulation nichts anderes als ein Schätzer für die Bewertung eines Spielzustands sowie der Spielzüge, die in diesen Zustand geführt haben. Somit wäre es durchaus denkbar, einen solchen Schätzer mit Werkzeugen aus dem Bereich des maschinellen Lernens zu bauen, wie auf beeindruckende Art und Weise beim Alpha Go Projekt [25] von Deepmind gezeigt wurde. Dies könnte die Laufzeit des MCTS drastisch verkürzen und würde gleichzeitig erlauben, Expertenwissen anhand von Spielaufzeichnungen in die Bewertung einfließen zu lassen.

5 Rückpropagierung

Im letzten Schritt der Iteration werden die Resultate der Simulation bzw. die Bewertung aus Schritt 4 an alle Knoten weitergereicht, die zuvor zu dieser Simulation geführt hatten. Die Weitergabe läuft rückwärts über den bei Schritt 2 genommenen Pfad durch den Baum, hoch bis zur Wurzel der Baumstruktur. Siehe dazu auch Abbildung 34.

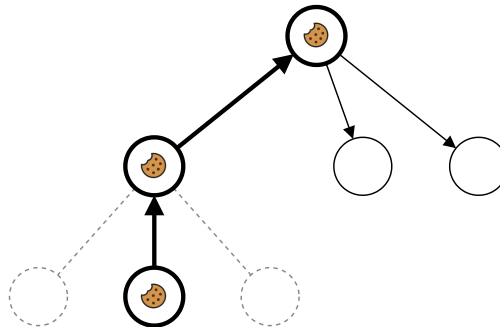


Abbildung 34: Rückpropagierung des Simulationsausgangs durch MCTS Algorithmus

Bei der nächsten Iteration wird der MCTS Algorithmus in Schritt 2 möglicherweise einen anderen Pfad nehmen, wenn der simulierte Spielausgang aus Sicht des entsprechenden Teams schlecht war. Es könnte aber auch sein, dass sich der Pfad als besonders vielversprechend erweist, wenn das Team in der Simulation viele Punkte erzielen konnte.

Über viele Iterationen hinweg entsteht dadurch eine ungefähre Abschätzung, zu welchem Spielausgang ein einzelner Spielzug führen könnte. Dementsprechend gibt es in jedem Spielzustand eine Entscheidungsgrundlage, welche seiner Handkarten der aktuelle Spieler als nächstes ausspielen sollte, um sein Team dem Gewinn des Spiels näher zu bringen.

Hinweis: Anzahl Iterationen

Der MCTS Algorithmus kann nach einer beliebigen Anzahl Iterationen abgebrochen werden. Das Resultat wird mit jeder zusätzlichen Iteration besser. Normalerweise gibt man dem Algorithmus eine fixe Zeitdauer zur Berechnung, und bricht ihn bei Erreichung dieses Zeitlimits ab. Man muss somit die Wartezeit gegen die Qualität des Resultats abwägen.

Erweiterung um UCT

Bei der Beschreibung der Selektionsphase im MCTS Algorithmus wurde bewusst offen gelassen, nach welchen Kriterien die Traversierung im Baum stattfinden soll. Grundsätzlich gibt es für die Suche zwei mögliche Strategien:

- Tiefensuche im Baum (einzelne Spielzüge genauer analysieren)
- Breitensuche im Baum (möglichst viele Spielzüge analysieren)

Sucht man nur in die Tiefe, dann schiesst sich der Algorithmus zu schnell auf einzelne - aus seiner Sicht vielversprechende - Spielzüge ein und ignoriert dabei eine Vielzahl anderer Spielzüge. Dies betrifft natürlich nicht nur die eigenen Entscheidungen, sondern auch die mögliche Entscheidungen des gegnerischen Teams.

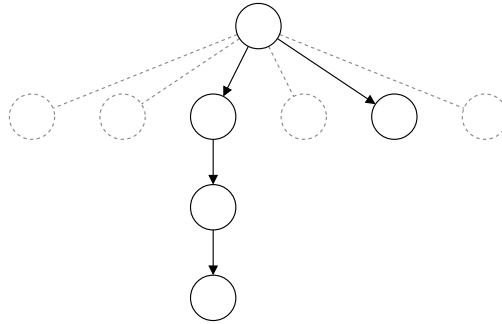


Abbildung 35: Tiefensuche im Baum durch MCTS Algorithmus

Sucht man nur in die Breite, dann berücksichtigt der Algorithmus zwar viele mögliche Spielzüge, seine Bewertungen dazu sind allerdings sehr ungenau, wenn nicht sogar vollkommen falsch. Eine einzelne Simulation zur Bewertung eines Spielzustands kann aufgrund der zufällig angenommenen Spielzüge der Spieler im Prinzip alle möglichen Resultate liefern. Die Simulation muss mehrfach wiederholt werden, bis sich ihr Resultat dem realen Erwartungswert nähert.

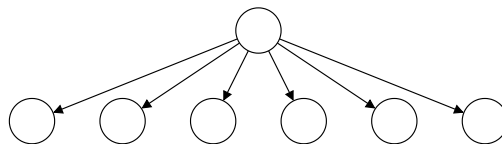


Abbildung 36: Breitensuche im Baum durch MCTS Algorithmus

Wie so oft gilt es ein gesundes Mittelmaß zu finden. Genau das versucht der Upper Confidence Bound for Trees (UCT) Ansatz zu ermöglichen. UCT kann als Erweiterung des MCTS Algorithmus betrachtet werden. In der Selektionsphase des MCTS Algorithmus wird das Konzept eines Vertrauensintervall für die Bewertung von Spielzuständen eingeführt. Bei wenig Simulationen wird die Bewertung eines Knotens im Baum absichtlich überschätzt (obere Grenze des Vertrauensintervalls), um eine erneute Selektion dieses Knotens durch den MCTS Algorithmus herbeizuführen. Mit jeder weiteren Simulation verkleinert sich das Vertrauensintervall und die Bewertung des Knotens nähert sich dem realen Erwartungswert. Der Algorithmus baut damit gewissermassen Vertrauen in seine eigene Bewertung von Spielzügen auf. Die Bewertung eines Knotens im Baum mit UCT wurde für den Schieber Jass wie folgt umgesetzt:

v = Knoten für Ausgangszustand

i = Möglicher Spielzug in Ausgangszustand

v_i = Knoten für Zielzustand nach Spielzug i

$N(v)$ = Anzahl der Simulationen für Ausgangszustand

$N(v_i)$ = Anzahl der Simulationen für Zielzustand nach Spielzug i

$Q(v_i)$ = Total erzielte Punkte in Simulationen für Zielzustand nach Spielzug i

c = Erkundungsfaktor

$$UCT(v_i, v) = \frac{Q(v_i)}{N(v_i) \cdot 157} + c \cdot \sqrt{\frac{\ln(N(v))}{N(v_i)}}$$

Die Berechnung findet aus der Sicht von einem der beiden Teams statt. Dementsprechend sind $Q(v_i)$ die total erzielten Punkte des betrachteten Teams unter der Annahme von Spielzug i . Der vordere Term $\frac{Q(v_i)}{N(v_i) \cdot 157}$ entspricht dem Anteil der maximal möglichen Punkte, welchen das Team in $N(v_i)$ Simulationen erzielen konnte. Der hintere Term $\sqrt{\frac{\ln(N(v))}{N(v_i)}}$ entspricht einem Bonus. Wird ein Zielzustand v_i im Vergleich zu seinem Ausgangszustand v sehr selten simuliert, steigt sein Bonus und damit die Wahrscheinlichkeit, dass er bei der nächsten Iteration selektiert wird. Gewichtet wird dieser Bonus mit dem frei wählbaren Erkundungsfaktor c . Über diesen Faktor kann man schlussendlich steuern, wie breit der MCTS Algorithmus im Baum suchen soll. Abbildung 37 zeigt, wie stark der Bonus bei der Bewertung eines Knotens in Abhängigkeit der dafür durchgeführten Simulationen ausgeprägt ist.

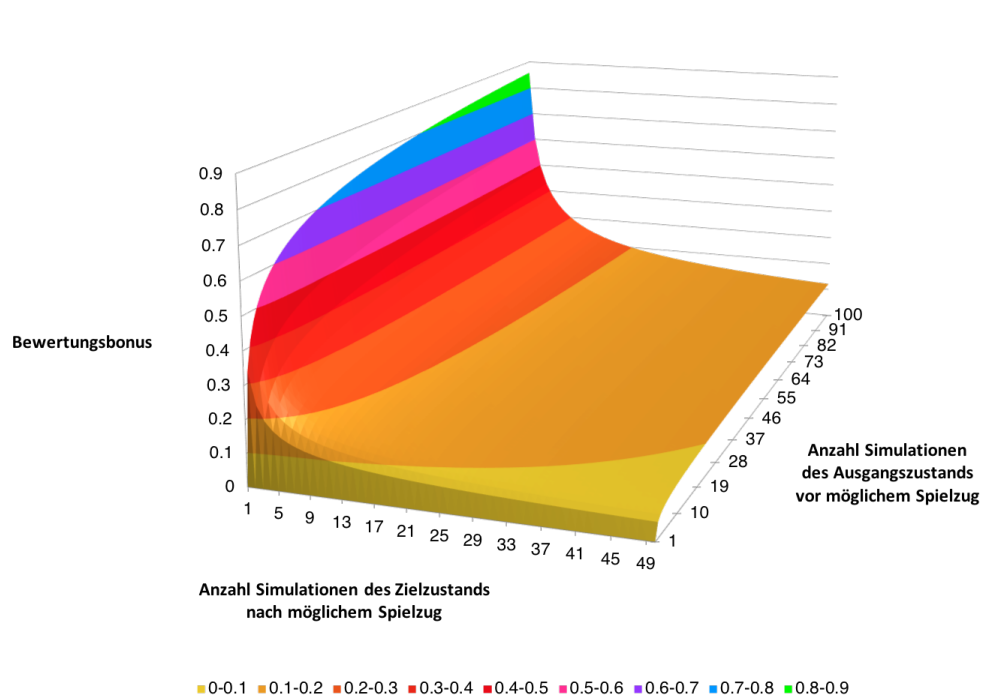


Abbildung 37: UCT Bewertungsbonus mit einem Erkundungsfaktor von 0.4

Hinweis: Erkundungsfaktor

Der Erkundungsfaktor c muss experimentell bestimmt werden. Im Rahmen dieser Bachelorarbeit wurde ein Faktor von 0.4 gewählt. Damit bewegt sich der Bonus zwischen 0 und 1, was analytisch betrachtet aufgrund der Addition mit dem Punkteanteil des Teams vernünftig scheint. Es ist jedoch nicht auszuschliessen, dass es für diesen Faktor eine bessere Wahl als 0.4 gibt.

Resultat

Aus diesem Ansatz ist ein Bot entstanden, der in jedem Stich eines Schieber Jass Spiels die Entscheidung zum Ausspielen einer seiner Handkarten treffen kann. Die Implementierung folgt dem Schema aus Kapitel 8.1, um immer den aktuellen Zustand des Spiels zur Verfügung zu haben. Die Funktionalität des Algorithmus wurde in ein separates Python Package namens *schieber_jass_mcts* ausgelagert. Die Implementation des Bots befindet sich im Package *mcts_bot*. Siehe Kapitel 6 für eine Übersicht aller Packages.

Name	Wert	Beschreibung
iterations	Zahl grösser 0	Steuert die Anzahl an Iterationen, die vom MCTS Algorithmus durchlaufen werden, bevor der Bot eine Entscheidung trifft. Mehr Iterationen führen zu längeren “Bedenkzeiten” des Bots.
exploration_factor	Zahl grösser 0	Entspricht dem Erkundungsfaktor c in der UCT Erweiterung des MCTS Algorithmus.

Tabelle 21: Mögliche Parameter für die Ausführung des MCTS Bots

9 Resultate und Auswertungen

Dieses Kapitel befasst sich mit der Validierung der in Kapitel 8.2 bis 8.4 behandelten Ansätze. Der Challenge Bot gilt dabei als Referenzimplementation. Mit Hilfe der in Kapitel 7 vorgestellten Benchmark Umgebung wurden die entwickelten Ansätze mit dem Challenge Bot verglichen. Pro implementiertem Ansatz wurde ein Experiment aufgesetzt. Im Folgenden wird jeweils der Aufbau des Experiments erklärt sowie die dazugehörigen Resultate tabellarisch zusammengefasst und visualisiert. Für alle Experimente wurde der gleiche Seed zur Generierung von Handsituationen verwendet, um gleiche Voraussetzungen zu schaffen.

Hinweis: Definition von Gewinn

Im Rahmen der Benchmarks werden nur einzelne Spiele ausgetragen, keine ganzen Jass Partie. Wird von “Gewinn” gesprochen, dann bezieht sich das lediglich auf den Ausgang eines einzelnen Spiels (Reward), also die Austragung von 9 Stichen inklusive Trumpfwahl.

9.1 Bedeutung der Trumpfwahl

Testaufbau und Ziel

In einem ersten Experiment soll untersucht werden, inwiefern sich die Qualität der Trumpfwahl auf das Spielergebnis auswirkt. Details zur Trumpfwahl sind in Kapitel 5.1 beschrieben. Abbildung 38 zeigt die Spielvariationen für den Benchmark. Der Random Bot spielt komplett zufällig. Der Semi Random Bot entspricht einer Kombination aus Challenge Bot für die Trumpfwahl, und Random Bot für das Ausspielen der Handkarten. Auf diese Weise kann untersucht werden, wie stark sich die Resultate von einer zufälligen Trumpfwahl unterscheiden. Tabelle 22 zeigt, welche Parameter für die Durchführung dieses Experiments in der Benchmark Umgebung verwendet wurden.

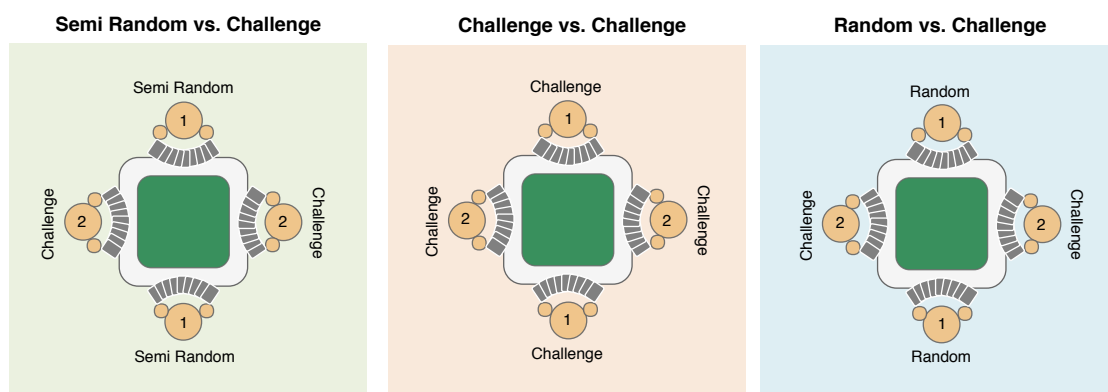


Abbildung 38: Benchmark Spielvariationen (Semi Random)

Name

Durchlauf 1

Durchlauf 2

nr_of_hand_card_situations	1000	10000
nr_of_cards_to_play	36	36
nr_of_samples_per_variation	1	1

Tabelle 22: Benchmark Parameter (Semi Random)

Resultate

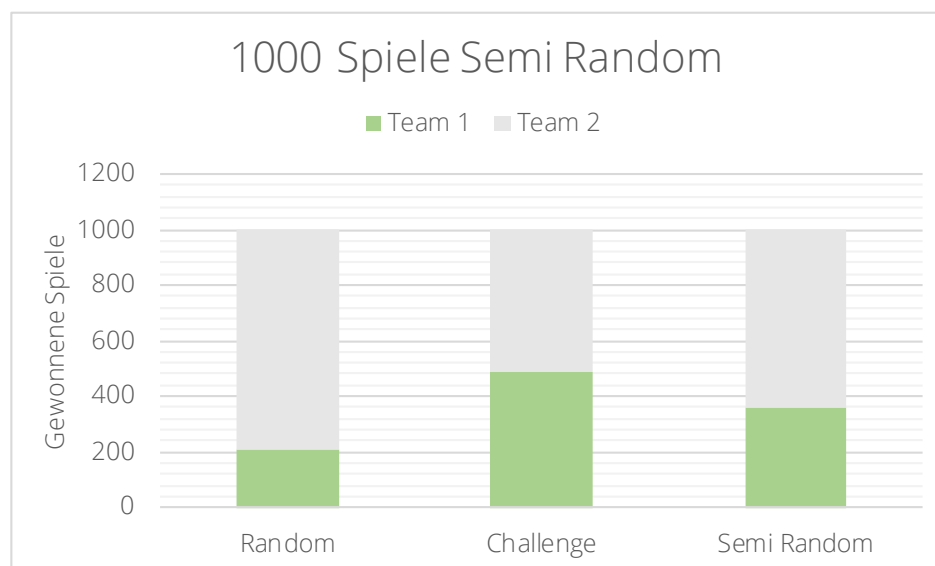


Abbildung 39: Anteil gewonnener Spiele bei 1000 Handsituationen (Semi Random)

	Semi Random	Challenge	Challenge	Challenge	Random	Challenge
Gewonnene Spiele	363	637	491	509	207	793
Gewinnrate	0.363	0.637	0.491	0.509	0.207	0.793
Erspielte Punkte	62662	94338	77337	79663	51189	105811
Durchschnitt Punkte	62.662	94.338	77.337	79.663	51.189	105.811

Tabelle 23: Benchmark Resultate bei 1000 Handsituationen (Semi Random)

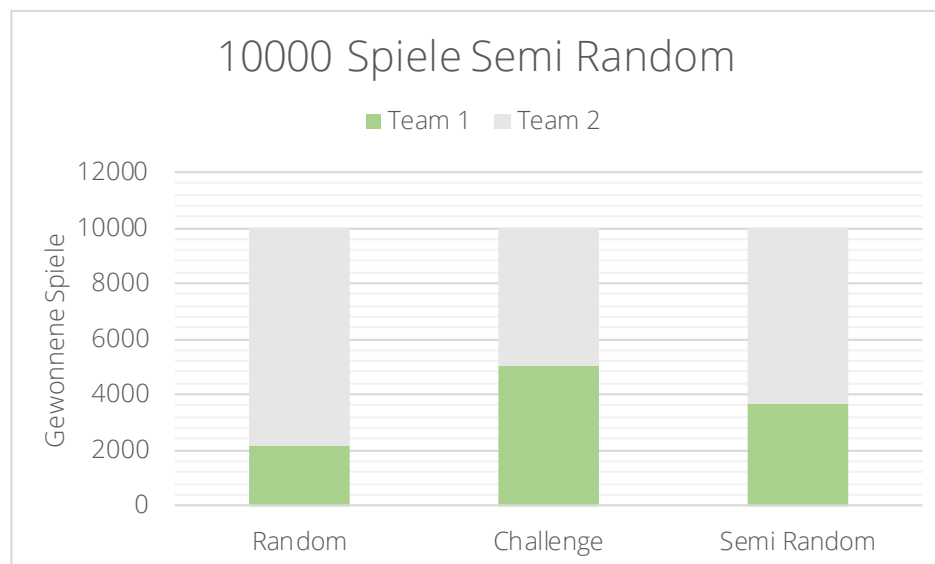


Abbildung 40: Anteil gewonnener Spiele bei 10000 Handsituationen (Semi Random)

	Semi Random	Challenge	Challenge	Challenge	Random	Challenge
Gewonnene Spiele	3692	6308	5009	4991	2196	7804
Gewinnrate	0.3692	0.6308	0.5009	0.4991	0.2196	0.7804
Erspielte Punkte	641572	928428	785716	784284	520464	1049536
Durchschnitt Punkte	64.1572	92.8428	78.5716	78.4284	52.0464	104.9536

Tabelle 24: Benchmark Resultate bei 10000 Handsituationen (Semi Random)

Wie man in den Tabellen 23 und 24 erkennen kann, gewinnt der Semi Random Bot mit ca. 36% einen überraschend hohen Anteil der Spiele, obwohl er seine Handkarten zufällig auf den Tisch legt. Im Vergleich dazu erreicht der "reine" Random Bot gerade mal 20%. Wenn man die erspielten Punkte betrachtet zeigt sich auch dort eine Verbesserung. Mit ca. 62 durchschnittlich erspielten Punkten erreicht der Semi Random Bot ca. 39% der möglichen Punkte im Spiel. Der Random Bot hingegen erzielt durchschnittlich ca. 51 Punkte, was einem Anteil von rund 32% entspricht.

Die Qualität der Trumpfwahl scheint somit einen nicht zu unterschätzenden Einfluss auf den Spielausgang zu haben. Um das absolute Maximum aus einem Bot herauszuholen, müsste man sich mit dieser Problemstellung auf jeden Fall noch auseinandersetzen.

9.2 Counterfactual Regret Minimization

Testaufbau und Ziel

In diesem Experiment geht es darum zu zeigen, wie sich der Counterfactual Regret Minimization Ansatz im Vergleich zum Challenge Bot schlägt. Abbildung 41 zeigt die Spielvariationen für den Benchmark. Der CFR Bot nutzt den Algorithmus des in Kapitel 8.2 beschriebenen Ansatzes zur Auswahl der nächsten auszuspielenden Handkarte. Die Parametrisierung des CFR Bots kann Tabelle 25 entnommen werden. Tabelle 26 zeigt zudem, welche Parameter für die Durchführung dieses Experiments in der Benchmark Umgebung verwendet wurden.

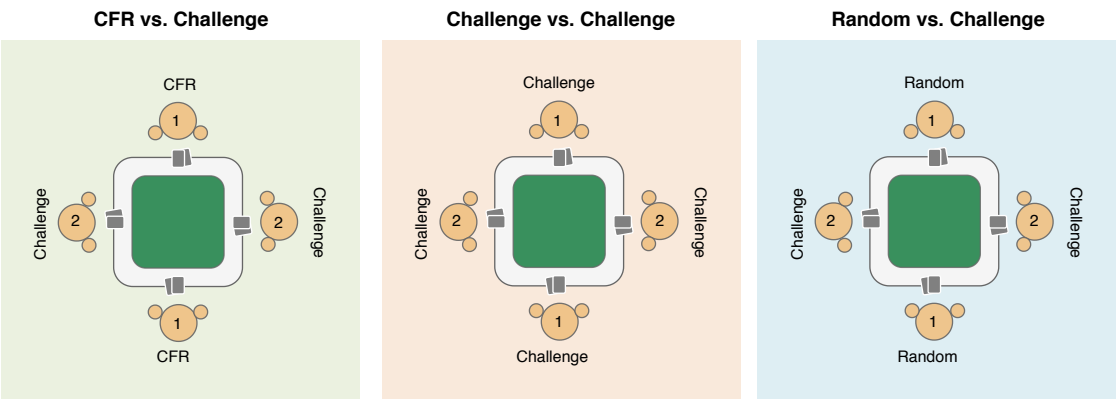


Abbildung 41: Benchmark Spielvariationen (CFR)

Name	Durchlauf 1	Durchlauf 2	Durchlauf 3
iterations	5	5	5

Tabelle 25: Bot Parameter (CFR)

Name	Durchlauf 1	Durchlauf 2	Durchlauf 3
nr_of_hand_card_situations	1000	10000	50000
nr_of_cards_to_play	8	8	8
nr_of_samples_per_variation	1	1	1

Tabelle 26: Benchmark Parameter (CFR)

Hinweis: Testaufbau CFR

Die nachfolgend präsentierten Resultate für den CFR Ansatz beziehen sich nur auf die letzten zwei Stiche eines Spiels. Es geht dabei nicht nur um die Frage, ob der Ansatz ei-

ne wahrnehmbare Verbesserung bringt. Es soll auch untersucht werden, wie viel Einfluss man noch durch Ändern der Spielweise in den letzten beiden Stichen auf den Spielausgang nehmen kann. Für jede der Spielvariationen aus Abbildung 41 gilt, dass die ersten sieben Handkarten durch den Challenge Bot ausgespielt wurden. Erst für die letzten zwei Handkarten kamen die zu vergleichenden Bots zum Einsatz.

Resultate



Abbildung 42: Erspielte Punkte in den letzten zwei Stichen bei 1000 Handsituationen (CFR)

	CFR	Challenge	Challenge	Challenge	Random	Challenge
Gewonnene Spiele	553	447	548	452	524	476
Gewinnrate	0.553	0.447	0.548	0.452	0.524	0.476
Erspielte Punkte	17911	16135	17620	16426	15595	18451
Durchschnitt Punkte	17.911	16.135	17.62	16.426	15.595	18.451

Tabelle 27: Benchmark Resultate bei 1000 Handsituationen (CFR)

Die erspielten Punkte in Tabelle 27 geben erste Hinweise darauf, dass sich der CFR Bot in den letzten zwei Stichen eines Spiels ungefähr auf gleichem Niveau wie der Challenge Bot bewegt. Obwohl es in diesem Fall nur noch um die Entscheidung zwischen zwei möglichen Handkarten geht, scheint man mit zufälligem Spielverhalten einiges schlechter zu fahren, wie die erspielten Punkte des Random Bots verdeutlichen. Die Auswirkungen auf die gewonnenen Spiele sind

im Vergleich dazu geringfügiger, was darauf hinweist, dass man in den letzten zwei Stichen eines Spiels tatsächlich nicht mehr viel am Spielausgang ändern kann.

Hinweis: Punktunterschiede bei vier Challenge Bots

Die Punktunterschiede der Spielvariation “Challenge vs. Challenge” in Tabelle 27 sind auf die geringe Anzahl an simulierten Spielsituationen zurückzuführen. Erwartungsgemäss müssten Teams mit Bots vom gleichen Typ in den letzten zwei Stichen über viele Spielsituationen hinweg etwa gleich viele Punkte holen. Die Zahlen deuten auf mehr “Kartenglück” für Team 1 hin.



Abbildung 43: Erspielte Punkte in den letzten zwei Stichen bei 10000 Handsituationen (CFR)

	CFR	Challenge	Challenge	Challenge	Random	Challenge
Gewonnene Spiele	5125	4875	4985	5015	4854	5146
Gewinnrate	0.5125	0.4875	0.4985	0.5015	0.4854	0.5146
Erspielte Punkte	179658	156377	168254	167781	152982	183053
Durchschnitt Punkte	17.9658	15.6377	16.8254	16.7781	15.2982	18.3053

Tabelle 28: Benchmark Resultate bei 10000 Handsituationen (CFR)

Bei 10000 simulierten Spielsituationen wird das Bild für die letzten zwei Stiche eines Spiels schon klarer, wie der Blick auf Tabelle 28 zeigt. Hinsichtlich erspielter Punkte erzielt der CFR Bot nun einen wahrnehmbaren Vorsprung zum Challenge Bot. Die zuvor beobachteten Ausreisser der Spielvariation “Challenge vs. Challenge” haben sich gelegt, die erspielten Punkte beider Teams sind nun erwartungsgemäss praktisch identisch. Zufälliges Spielverhalten

scheint nach wie vor keine gute Idee zu sein, die Ergebnisse des Random Bots bleiben etwa gleich.



Abbildung 44: Erspielte Punkte in den letzten zwei Stichen bei 50000 Handsituationen (CFR)

	CFR	Challenge	Challenge	Challenge	Random	Challenge
Gewonnene Spiele	25570	24430	25033	24967	24150	25850
Gewinnrate	0.5114	0.4886	0.50066	0.49934	0.483	0.517
Erspielte Punkte	892827	793248	836143	849932	756249	929826
Durchschnitt Punkte	17.85654	15.86496	16.72286	16.99864	15.12498	18.59652

Tabelle 29: Benchmark Resultate bei 50000 Handsituationen (CFR)

Hier zeigt sich ein sehr ähnliches Bild wie bei 10000 Handsituationen. Der CFR Bot scheint in den letzten zwei Stichen etwas besser zu spielen als der Challenge Bot. Eine abschliessende Aussage zu treffen ist allerdings schwierig.

Hinweis: Abhängigkeit in Ausgangssituationen

Die Stichproben dieses Experiments sind nicht wirklich unabhängig. Die Ausgangssituationen zur Messung der letzten zwei Stiche sind stark vom Spielverhalten des Challenge Bots geprägt. Dies fällt beispielsweise bei Betrachtung der verbleibenden zwei Handkarten auf, welche die Bots im Rahmen dieses Experiments jeweils ausspielen müssen. In vielen Fällen sind keinerlei Karten in Trumpffarbe mehr im Spiel. Diese Problematik so-

wie eine mögliche Lösung dafür werden in Kapitel 7.2 genauer diskutiert.

Aus Sicht der Autoren deuten die Resultate jedoch schon darauf hin, dass durch den CFR Ansatz in gewissem Masse eine Verbesserung des Spielniveaus - verglichen zum Challenge Bot - möglich sein sollte.

9.3 Monte Carlo Counterfactual Regret Minimization

Testaufbau und Ziel

In diesem Experiment geht es darum zu zeigen, wie sich der Monte Carlo CFR Ansatz im Vergleich zum Challenge Bot schlägt. Abbildung 45 zeigt die Spielvariationen für den Benchmark. Der MCCFR Bot nutzt den Algorithmus des in Kapitel 8.3 beschriebenen Ansatzes zur Auswahl der nächsten auszuspielenden Handkarte. Die Parametrisierung des MCCFR Bots kann Tabelle 30 entnommen werden. Tabelle 31 zeigt, welche Parameter für die Durchführung dieses Experiments in der Benchmark Umgebung verwendet wurden.

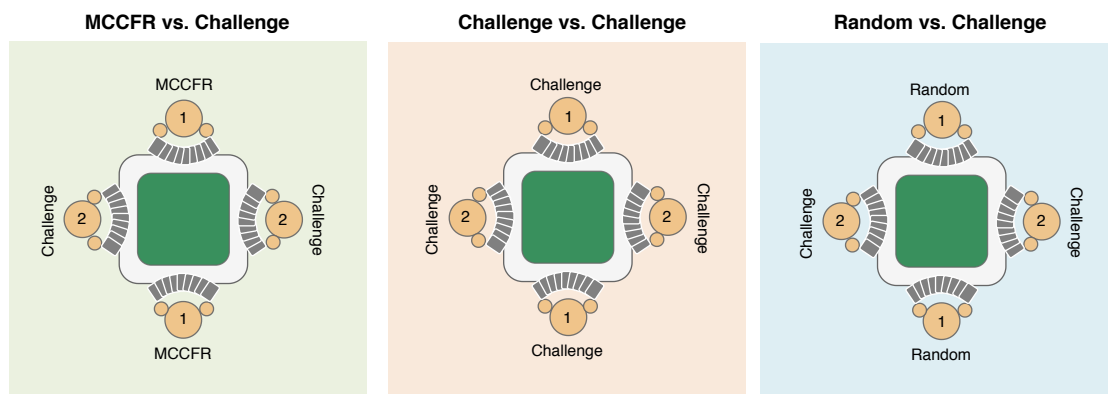


Abbildung 45: Benchmark Spielvariationen (MCCFR)

Name	Durchlauf 1	Durchlauf 2
exploration_factor	0.6	0.6
iterations	1000	1000

Tabelle 30: Bot Parameter (MCCFR)

Name	Durchlauf 1	Durchlauf 2
nr_of_hand_card_situations	1000	10000
nr_of_cards_to_play	36	36
nr_of_samples_per_variation	1	1

Tabelle 31: Benchmark Parameter (MCCFR)

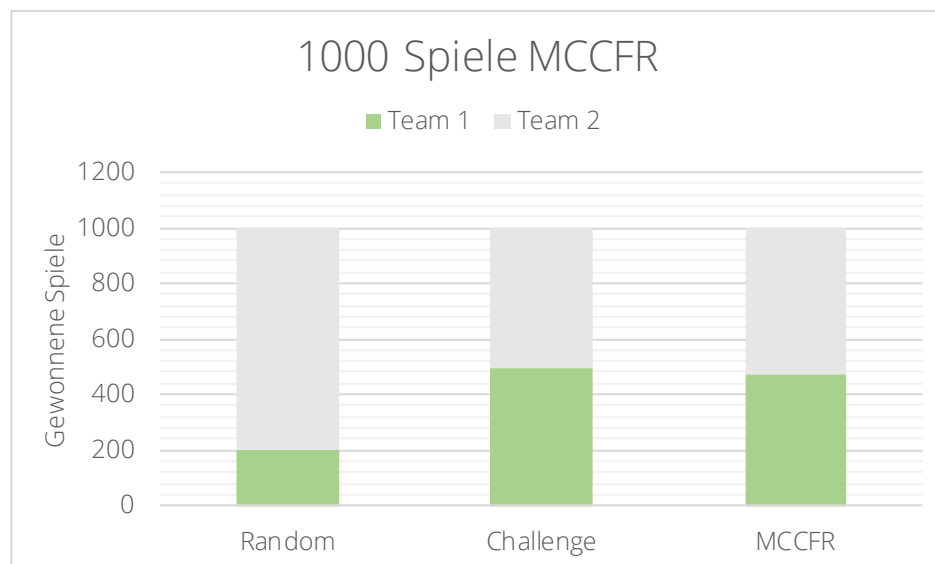


Abbildung 46: Anteil gewonnener Spiele bei 1000 Handsituationen (MCCFR)

	MCCFR	Challenge	Challenge	Challenge	Random	Challenge
Gewonnene Spiele	470	530	497	503	201	799
Gewinnrate	0.47	0.53	0.497	0.503	0.201	0.799
Erspielte Punkte	74809	82191	78744	78256	51456	105544
Durchschnitt Punkte	74.809	82.191	78.744	78.256	51.456	105.544

Tabelle 32: Benchmark Resultate bei 1000 Handsituationen (MCCFR)

Wie man in Tabelle 32 erkennen kann, gewinnt der MCCFR Bot mit 47% knapp die Hälfte der Spiele gegen den Challenge Bot. Die Bots bewegen sich auf sehr ähnlichem Spielniveau, wenn auch der MCCFR Bot bei genauerer Betrachtung leicht schlechter abschneidet. Im Vergleich dazu schafft es der Random Bot gerade mal auf 20%, was einen sehr deutlichen Unterschied zur zufälligen Spielweise aufzeigt. Die erspielten Punkte bestätigen den Eindruck hinsichtlich der gewonnenen Spiele.

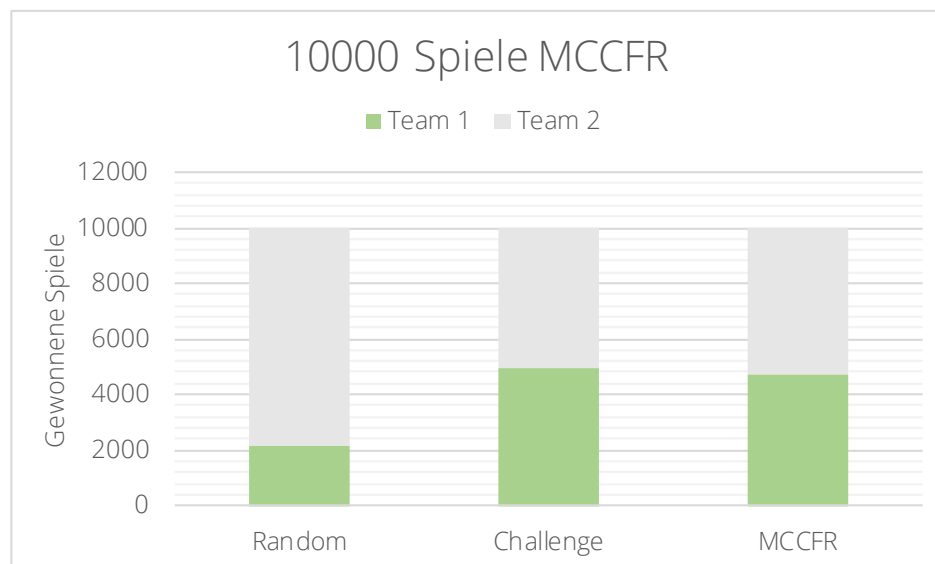


Abbildung 47: Anteil gewonnener Spiele bei 10000 Handsituationen (MCCFR)

	MCCFR	Challenge	Challenge	Challenge	Random	Challenge
Gewonnene Spiele	4757	5243	4943	5057	2194	7806
Gewinnrate	0.4757	0.5243	0.4943	0.5057	0.2194	0.7806
Erspielte Punkte	745319	824681	784019	785981	514735	1055265
Durchschnitt Punkte	74.5319	82.4681	78.4019	78.5981	51.4735	105.5265

Tabelle 33: Benchmark Resultate bei 10000 Handsituationen (MCCFR)

Die Ausweitung des Experiments auf 10000 Handsituationen scheint keine neuen Erkenntnisse zu bringen. Die Resultate in den Tabellen 32 und 33 weisen keine nennenswerten Unterschiede auf. Wenn auch deutlich besser als zufällig, spielte der MCCFR Bot im Rahmen dieses Experiments ein wenig schlechter als der Challenge Bot.

Die Autoren sehen dafür drei mögliche Ursachen. Zum einen konnte die Parametrisierung des Bots nicht optimiert werden, es ist durchaus möglich, dass beispielsweise eine Erhöhung der Iterationen zu besseren Resultaten führen könnte. Zum anderen ist nicht auszuschliessen, dass die Implementierung des Algorithmus noch Fehler aufweist. Schon bei der Umsetzung des CFR Ansatzes haben schwer auffindbare Programmierfehler zu wirren Entscheidungen geführt. Aufgrund der erhöhten Komplexität des MCCFR Ansatzes im Vergleich zu "Vanilla" CFR ist diese Vermutung durchaus begründet. Vielleicht sind die Resultate aber auch ein Indiz dafür, dass sich der MCCFR Ansatz nicht eignet zum Ausspielen von Handkarten beim Schieber Jass.

9.4 Monte Carlo Tree Search

Testaufbau und Ziel

In diesem Experiment geht es darum zu zeigen, wie sich der Monte Carlo Tree Search Ansatz im Vergleich zum Challenge Bot schlägt. Abbildung 48 zeigt die Spielvariationen für den Benchmark. Der MCTS Bot nutzt den Algorithmus des in Kapitel 8.4 beschriebenen Ansatzes zur Auswahl der nächsten auszuspielenden Handkarte. Die Parametrisierung des MCTS Bots kann Tabelle 35 entnommen werden. Tabelle 34 zeigt, welche Parameter für die Durchführung dieses Experiments in der Benchmark Umgebung verwendet wurden.

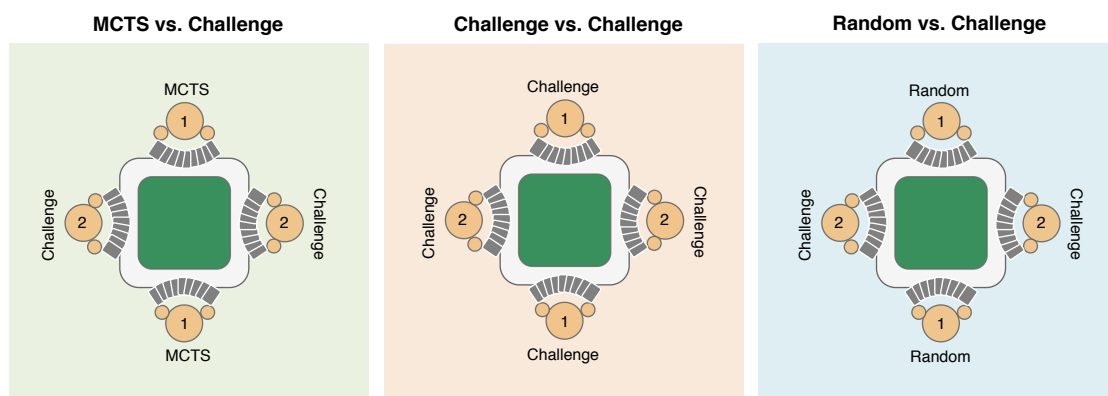


Abbildung 48: Benchmark Spielvariationen (MCTS)

Name	Durchlauf 1	Durchlauf 2
nr_of_hand_card_situations	1000	10000
nr_of_cards_to_play	36	36
nr_of_samples_per_variation	1	1

Tabelle 34: Benchmark Parameter (MCTS)

Name	Durchlauf 1	Durchlauf 2
exploration_factor	0.4	0.4
iterations	1000	1000

Tabelle 35: Bot Parameter (MCTS)

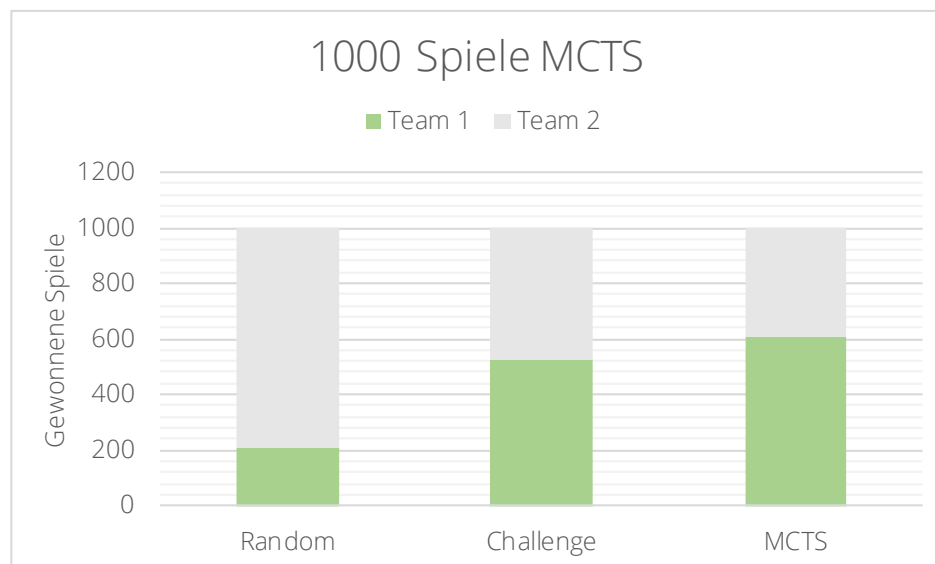


Abbildung 49: Anteil gewonnener Spiele bei 1000 Handsituationen (MCTS)

	MCTS	Challenge	Challenge	Challenge	Random	Challenge
Gewonnene Spiele	608	392	524	476	207	793
Gewinnrate	0.608	0.392	0.524	0.476	0.207	0.793
Erspielte Punkte	85762	71238	78834	78166	51031	105969
Durchschnitt Punkte	85.762	71.238	78.834	78.166	51.031	105.969

Tabelle 36: Benchmark Resultate bei 1000 Handsituationen (MCTS)

Schon nach 1000 Handsituationen zeigen die gewonnenen Spiele in Tabelle 36 eine Verbesserung des Spielniveaus durch den MCTS Bot. Er kann rund 60% der Spiele gegen den Challenge Bot für sich entscheiden. Der Random zeigt das übliche Bild, welches auch in den anderen Experimenten schon beobachtet werden konnte. Interessant ist hier insbesondere der Vergleich, denn sowohl MCTS als auch MCCFR basieren eigentlich auf Zufall, wenn auch angereichert mit mehr oder weniger komplexen Mechanismen.

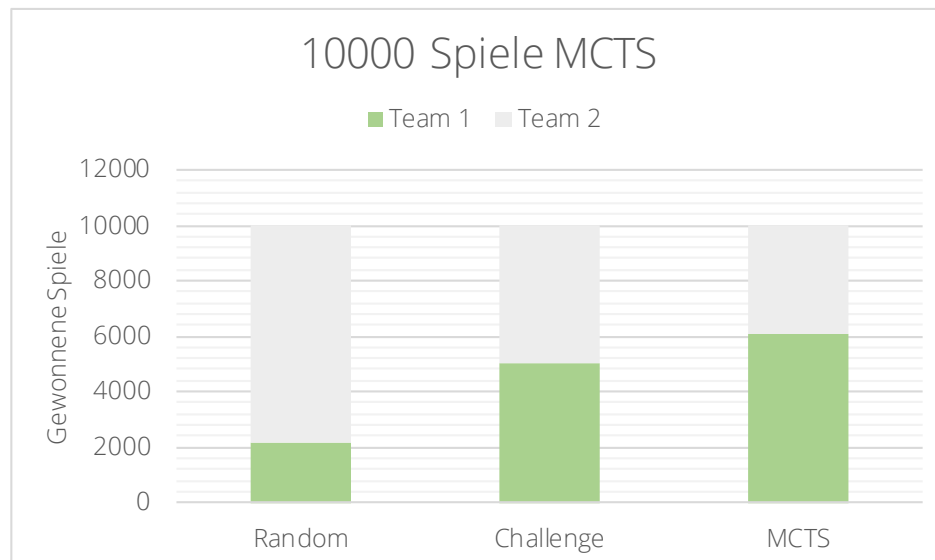


Abbildung 50: Anteil gewonnener Spiele bei 10000 Handsituationen (MCTS)

	MCTS	Challenge	Challenge	Challenge	Random	Challenge
Gewonnene Spiele	6062	3938	4991	5009	2185	7815
Gewinnrate	0.6062	0.3938	0.4991	0.5009	0.2185	0.7815
Erspielte Punkte	862970	707030	787030	782970	515307	1054693
Durchschnitt Punkte	86.297	70.703	78.703	78.297	51.5307	105.4693

Tabelle 37: Benchmark Resultate bei 10000 Handsituationen (MCTS)

Die Ausweitung des Experiments auf 10000 Handsituationen bestätigt das Bild, die Gewinnrate für den Challenge Bot scheint sich bei etwa 60% einzupendeln. Zudem holt der MCTS Bot rund 55% der möglichen Punkte im Schnitt, nicht ganz so deutlich wie bei den gewonnenen Spielen, aber trotzdem eine Verbesserung.

Es gibt daher Grund zur Annahme, dass der MCTS Bot besser spielt als der Challenge Bot. Durch Optimierung der Parameter wäre möglicherweise noch mehr herauszuholen.

9.5 Hypothesentests

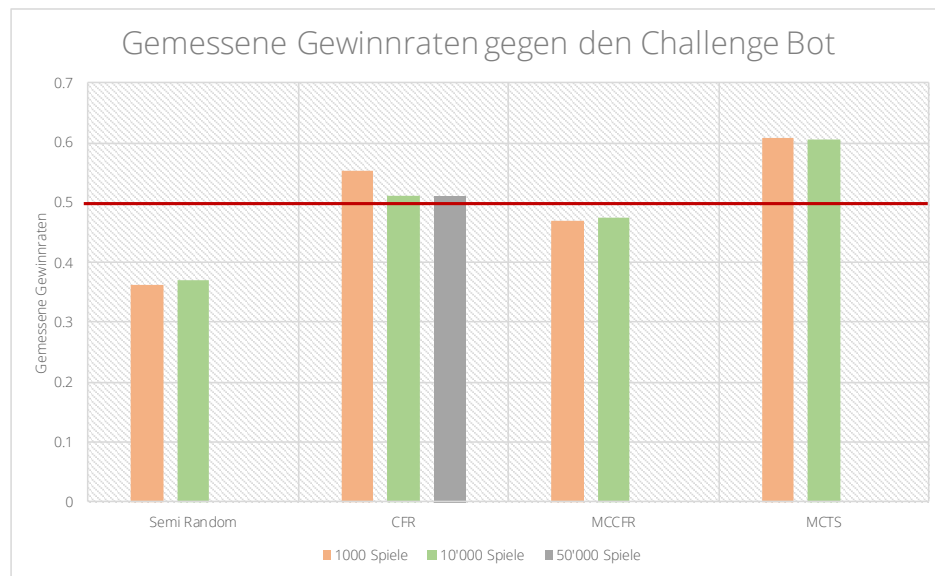


Abbildung 51: Überblick über alle gemessenen Gewinnraten

In den vorangegangenen Kapiteln wurden Experimente für die evaluierten Ansätze durchgeführt. Nun geht es daran, den vielversprechendsten Ansatz statistisch zu untersuchen, um eine Aussage darüber treffen zu können, ob der Bot auch wirklich signifikant besser spielt. Der bisher vielversprechendste Ansatz scheint der MCTS Bot zu sein. Mit 1000 Iterationen hat er es geschafft ca. 60% der Spiele gegen den Challenge Bot zu gewinnen. In diesem Kapitel wird daher versucht, mit einem Exact Binomial Test zu zeigen, dass durch den Monte Carlo Tree Search Lösungsansatz eine signifikante Verbesserung erreicht wurde.

In Abbildung 51 sieht man eine Zusammenfassung der gemessenen Gewinnraten in den Experimenten. Wenn zwei Bots gegeneinander antreten und gleich stark spielen, würde man erwarten, dass sie über viele Handsituationen hinweg etwa gleich viele Punkte erzielen. Dementsprechend müsste die Gewinnrate bei 50% liegen, dargestellt als horizontale rote Linie in Abbildung 51. In jedem Experiment wurde daher eine Spielvariation durchgeführt, in der ein Team von Challenge Bots gegen ein Team von Challenge Bot antritt, um dieses Phänomen zu bestätigen.

In einem ersten Schritt soll untermauert werden, dass in einem Spiel mit ausschliesslich Challenge Bots keines der Teams im Vorteil ist. Als Wahrscheinlichkeit für einen Fehler 1. Art wird für α ein Wert von 0.01 gewählt.

In einem zweiten Schritt soll gezeigt werden, dass ein Team von MCTS Bots signifikant besser spielt als ein Team von Challenge Bots. Die Gewinnrate sollte sich somit signifikant von 0.5 unterscheiden. Auch hier wird für α ein Wert von 0.01 gewählt.

Spielniveau des Challenge Bot

Nullhypothese H_0 : Die beiden Challenge Bot Teams spielen gleich gut. Ihre Gewinnwahrscheinlichkeit liegt daher bei $\frac{1}{2}$.

Für die Durchführung des Tests wird die Programmiersprache R verwendet:

```
binom.test(4991,10000,p=0.5, alternative="two.sided", conf.level=0.99)
```

Die Funktion liefert folgende Resultate:

```
data: 4991 and 10000
number of successes = 4991, number of trials = 10000, p-value = 0.865
alternative hypothesis: true probability of success is not equal to 0.5
99 percent confidence interval:
 0.4861738 0.5120271
sample estimates:
probability of success
      0.4991
```

Da der p-value mit 0.865 grösser ist als das gewählte $\alpha = 0.01$, gibt es keinen Grund an der Nullhypothese H_0 zu zweifeln. Man kann nicht sagen, dass eines der Challenge Bot Teams signifikant besser oder schlechter spielte als das andere bei der Durchführung der Experimente.

Spielniveau des MCTS Bot

In diesem Fall wird ein einseitiger Test gemacht, da nur untersucht werden soll, ob der MCTS Ansatz signifikant besser funktioniert als der regelbasierte Ansatz des Challenge Bots.

Nullhypothese H_0 : Das MCTS Bot Team spielt gleich gut oder schlechter als das Challenge Bot Team. Ihre Gewinnwahrscheinlichkeit liegt daher bei $\leq \frac{1}{2}$.

Nun gilt es die Nullhypothese zu widerlegen.

```
binom.test(6062,10000,p=0.5, alternative="greater", conf.level=0.99)
```

Die Funktion liefert folgende Resultate:

```
data: 6062 and 10000
number of successes = 6062, number of trials = 10000, p-value < 2.2e-16
alternative hypothesis: true probability of success is greater than 0.5
99 percent confidence interval:
 0.5947434 1.0000000
sample estimates:
probability of success
      0.6062
```

Da der p-value kleiner ist als das gewählte $\alpha = 0.01$, gibt es Anlass dazu die Nullhypothese H_0 zu verwerfen. Es konnte in den Experimenten gezeigt werden, dass das Team mit MCTS Bots signifikant besser spielt als das Challenge Bot Team.

9.6 Erfahrungsbericht

Um validieren zu können, ob der MCTS Ansatz auch gut mit bzw. gegen Menschen spielt, müssten eigentlich eine Vielzahl von Testspielen durchgeführt werden und das variierende Spielniveau der Spieler berücksichtigt werden. Dies war im Rahmen dieser Arbeit leider nicht mehr möglich.

Es wurden aber einige Testspiele in der Zühlke Umgebung durchgeführt, in denen zwei menschliche Spieler gegen ein Team von MCTS Bots angetreten sind. Die daraus gewonnen Erkenntnisse lassen zwar keine statistischen Schlüsse zu, sollen dem Leser aber ein Gefühl dafür vermitteln, wie es sich anfühlt gegen den MCTS Bot anzutreten. Die beiden menschlichen Spieler sind sicherlich keine Expertenspieler. Sie rangieren aber leicht über dem Niveau eines Gelegenheitsspielers. Die Spiele wurden als Video aufgezeichnet und anschliessend analysiert. Die daraus gewonnen Erkenntnisse werden im Folgenden festgehalten. Die Erkenntnisse sind natürlich subjektiv und entstammen einer sehr kleinen Stichprobe, wodurch die Aussagekraft relativiert werden muss.

Insgesamt wurden 5 Jass Partien auf 2500 Punkte gespielt. Die ersten drei Spiele wurden mit jeweils 1000 MCTS Iterationen durchgeführt. Anschliessend wurden die Iterationen auf 2000 erhöht um zu überprüfen, ob eine Verbesserung spürbar ist. Die Ergebnisse sind in Tabelle 38 ersichtlich. Von den 5 Jass Partien hat der MCTS Bot 2 gewonnen.

Nr	Menschliche Spieler	MCTS Bots	MCTS Iterationen
1	2689	1750	1000
2	2285	2582	1000
3	2785	1911	1000
4	2369	2655	2000
5	2619	2405	2000

Tabelle 38: Erzielte Punkte in 5 Jass Partien (Mensch gegen MCTS Bot)

Besonders auffällig im Spiel war das konsequente Abstecken sobald in einem Stich viele Punkte zu holen waren. Insbesondere wenn eine Ass oder eine Banner Karte auf dem Tisch lag, stachen die MCTS Bots die Karten ab, um sich die Punkte zu sichern. Andersherum spielten die Bots aber praktisch nie Karten mit hohem Punktwert aus, wenn das gegnerische Team den Stich bereits gesichert hatte. Überdies gelang es den Bots immer wieder wichtige Punkte zu sichern, indem sie die Punkte ihrem Partner "schmierten". Ansonsten gab es aber kaum Anzeichen für eine Interaktion zwischen Bots.

Hinweis: Schmieren

"Schmieren" kann man dann, wenn der Spielpartner den Stiche sicher für sich entscheidet. In diesem Fall bietet es sich an eine Karte mit hohem Punktwert auszuspielen (z.B.

eine Banner Karte). Auf diese Weise können wichtige Punkte für das eigene Team gesichert werden.

Hingegen ist aufgefallen, dass die Bots - wenn sie in einem Spiel die Trumpfwahl hatten - praktisch nie als erstes eine Karte in Trumpffarbe gespielt haben. Stattdessen sparten sie sich ihre Trumpfkarten auf, um sie später einzusetzen. Dies hatte zur Folge, dass das gegnerische Team mit kleinen Trumpfkarten viele Punkte stehlen konnte. Profispieler eröffnen deshalb meist (insbesondere wenn sie Trumpf ansagen ohne zu "Schieben") mit starken Trumpfkarten. Aufgrund der Regel, dass die nachfolgenden Spieler der Farbe folgen müssen, können die Trumpfkarten so gesichert und den anderen Spielern "gezogen" werden. Insgesamt hatte man das Gefühl, dass die Bots dadurch schlechter spielten und somit weniger Punkte erreichten. Bei "Undenufe" oder "Obenabe" spielten die Bots hingegen sehr souverän.

Subjektiv hatte man auch das Gefühl, dass die Steigerung auf 2000 MCTS Iterationen noch einmal eine Verbesserung des Spielniveaus gebracht hat. Leider dauert dann eine Entscheidung zwischen 5 bis 10 Sekunden pro Bot, was den Spielfluss spürbar ausbremst im Vergleich zum Spiel mit anderen Bots. Viele menschliche Spieler brauchen jedoch ähnlich viel Zeit für eine Entscheidung.

Eins kann man aber sicherlich sagen: Das Spiel gegen den MCTS Bot bereitet Spass! Für Gelegenheitsspieler ist nur selten zu erkennen, dass es sich beim Bot nicht um einen Menschen handelt. Profispieler würden den Unterschied aber vermutlich schnell bemerken. Im Spiel hat sich der MCTS Bot gut geschlagen, auch wenn er zwischendurch ungewohntes Verhalten zeigt.

10 Schlussfolgerungen und Ausblick

10.1 Resultate

Im Rahmen dieser Bachelorarbeit konnten insgesamt drei Prototypen (vgl. Kapitel 8) in Python entwickelt werden. Durch die Entwicklung einer eigenen Benchmark Umgebung (vgl. Kapitel 7) konnte das Spielniveau der verschiedenen Bots mit bestehenden Lösungen verglichen werden. Die Umgebung erleichtert die Validierung neuer Lösungsansätze enorm und könnte sicherlich auch für zukünftige Projekte zusammen mit dem *Pyschieber* eingesetzt werden.

Counterfactual Regret Minimization ist ein vielversprechender Ansatz. Leider lässt er sich aufgrund der Komplexität des Spiels nur sehr eingeschränkt anwenden. Die Erweiterung um eine Monte Carlo Komponente hat leider nicht den gewünschten Erfolg gebracht. Dies könnte mehrere Gründe haben. Einerseits ist es möglich, dass die vorgegebene Rechenzeit einfach zu kurz gewählt ist für bessere Resultate. Andererseits sind Fehler in der eigenen Umsetzung nicht auszuschliessen. Weitere Testserien könnten hier Aufschluss geben. Optimierungspotenzial bietet auch die Parametrisierung. Die Iterationen können nur in beschränktem Mass erhöht werden. Je mehr Iterationen durchgeführt werden, desto länger werden die Mitspieler im Spiel auf eine Entscheidung warten müssen. Vermutlich wäre eine performantere Implementierung des Ansatzes in einer anderen Programmiersprache hilfreich.

Die Monte Carlo Tree Search hat mit Abstand die besten Resultate geliefert. Es konnte gezeigt werden, dass damit eine signifikante Verbesserung gegenüber dem Challenge Bot erreicht wurde. Aus diesem Grund wäre es vermutlich sinnvoller, diesen Ansatz weiterzuverfolgen. Mögliche Ideen zur Erweiterung und Verbesserung des Ansatzes gibt es viele. Offen bleibt die Frage, wie gut sich der MCTS Bot tatsächlich gegen menschliche Spieler schlägt. Während dieser Arbeit war es leider aus zeitlichen Gründen nicht möglich, eine gross angelegte Testserie mit menschlichen Spielern durchzuführen. Um eine abschliessende Aussage treffen zu können, müsste dies weiter untersucht werden.

Das Spiel gegen den MCTS Bot ist auf jeden Fall unterhaltsam. Als durchschnittlicher Spieler könnte man denken, dass es sich dabei um menschliche Spieler handelt. Im Praxistest (siehe Kapitel 9.6) hat sich gezeigt, dass der MCTS Bot teilweise unerwartete Spielzüge durchführt (z.B. das Spiel nicht mit Trumpfkarten zu beginnen). Hier stellt sich die Frage, inwiefern dieses Verhalten durch geübte Spieler ausgenutzt werden kann. Es besteht Grund zur Annahme, dass die Einbringung von Expertenwissen im MCTS Algorithmus solche Schwachpunkte ausmerzen könnte.

10.2 Erweiterungsmöglichkeiten

Trumpfwahl

Wie in Kapitel 9.1 bereits angesprochen wurde, hat die Trumpfwahl einen relativ grossen Einfluss auf den Ausgang des Spiels. Bisher nutzen alle drei der implementierten Bots die Trumpfauswahl des Challenge Bots. Wie gut diese Auswahl tatsächlich ist, wurde nicht genauer untersucht. Eine Möglichkeit wäre z.B. die Trumpfauswahl von Pascal Kistler und Andreas

Eder in die existierenden Bots einzubauen, um zu schauen, ob sich damit schon eine Verbesserung erreichen lässt [3]. Eine weitere Möglichkeit bestünde darin, den MCTS Algorithmus dahingehend zu erweitern, dass er die Trumpfwahl übernehmen kann.

Kombination der Ansätze

Auch eine Kombination der bisher implementierten Ansätze wäre denkbar. So könnte z.B. der MCTS Algorithmus die ersten 7 Handkarten im Spiel wählen und der CFR Algorithmus die restlichen 2 Handkarten. Dank des entwickelten Proxy Bot (vgl. Kapitel 7) wäre dies relativ einfach zu realisieren. Im Rahmen dieser Arbeit blieb aber leider nicht genug Zeit, um solche Aufbauten auszuprobieren.

Simulationen mit anderen Bots

Statt rein zufällig Spiele zu simulieren für die Bewertung von Spielzuständen, wäre es auch denkbar, dass die Simulationen durch andere Bots fertiggespielt werden. Hier könnte z.B. der Challenge Bot verwendet werden. Man müsste jedoch sicherstellen, dass nicht nur der Challenge Bot Entscheidungen fällt. Es besteht sonst die Gefahr, dass der Algorithmus seine Entscheidungen nur auf das Spiel gegen den Challenge Bot optimiert. Vorstellbar wäre ein ähnlicher Ansatz wie beim MCCFR, so dass mit einer definierten Wahrscheinlichkeit zufällig gespielt wird und ansonsten der Challenge Bot eine Entscheidung trifft.

Einbringung von Expertenwissen

Eine weitere Möglichkeit die Entscheidungen des MCTS Algorithmus zu verbessern, wäre der Einbau von Expertenwissen. So könnte man z.B. nach eigener Trumpfwahl allen Trumpfkarten der eigenen Hand einen zusätzlichen Bonus für die Selektionsphase des Algorithmus geben. Auf diese Weise würden für Trumpfkarten mehr Simulationen durchgeführt werden, was automatisch zu einer realistischeren Bewertung für die entsprechenden Spielzüge führt.

Erweiterung mit Deep Learning

“AlphaGo”[26] und “DeepStack”[19] haben es vorgemacht. Beide Ansätze kombinierten klassische spieltheoretische Ansätze mit Deep Learning. Auch für den Schieber Jass wäre eine solche Lösung denkbar. Daher wäre es sicherlich interessant diese Idee weiterzuverfolgen. Möglich wäre z.B. dass man versucht die Simulationen im MCTS Algorithmus durch ein neuronales Netz zu ersetzen. Auf diese Weise könnte der Algorithmus viel mehr Iterationen durchlaufen und somit auch bessere Entscheidungen treffen.

Anhang

A Aufgabestellung



Abteilung Informatik HS 2018
Aufgabenstellung Bachelorarbeit Jan Aeberli, Jürg Schleutermann

Seite

1/3

Aufgabenstellung Bachelorarbeit Abteilung I, HS 2018 Jan Aeberli, Jürg Schleutermann

Praktische Anwendung spieltheoretischer Ansätze für den Schieber-Jass

1. Betreuer

Betreuer dieser Arbeit ist

Prof. Dr. Markus Stolze mstolze@hsr.ch

Co-Referent dieser Arbeit ist

Ivan Büttler

Externer Experte dieser Arbeit ist

Thomas Kälin, Noser

2. Ausgangslage

Anfang 2017 wurde von der Firma Zühlke ein Wettbewerb im Kontext des traditionellen Schweizer Kartenspiels Schieber-Jass durchgeführt. Dabei hatten die Teilnehmer die Aufgabe ein Programm (nachfolgend als Bot bezeichnet) zu entwickeln, welches gegen andere Bots im Schieber-Jass gewinnt. Der Wettbewerb war so aufgebaut, dass jeweils zwei Bot-Instanzen des einen Teams gegen zwei Bot-Instanzen eines anderen Teams spielten. Eine direkte Kommunikation zwischen den Bots war dabei verboten. Der Source-Code des gewinnenden Bots mit dem Namen "ChallengeBot" wurde der HSR von den Autoren für aufbauende Arbeiten zur Verfügung gestellt.

In einer ersten Arbeit untersuchten die Studierenden A. Eder und P. Kistler in ihrer Bachelorarbeit im HS17, ob sich ein verbesserter Bot mit Hilfe von "Deep Learning" Algorithmen (i.e. neuronale Netze) entwickeln lässt. Sie evaluierten hierbei zum einen den Ansatz des "Supervised Learning", unter Nutzung eines grossen Datensets von Spielen, welche von Swisslos bereitgestellt wurden. Zum anderen analysierten sie den Ansatz des "Reinforcement Learning", für welchen sie zufällige Partien mit Hilfe des "ChallengeBot" simulierten. Die Experimente zeigten, dass es mittels der gewählten Ansätze nicht möglich war, einen auf dem Niveau des "ChallengeBot" spielenden Bot zu entwickeln.

Auch weiterführende Experimente zum "Reinforcement Learning", welche der Master-Student S. Kurath im Rahmen seines Masterprojekts durchgeführt hatte, konnten keine erfolgreichen Lernstrategien für den Schieber-Jass identifizieren. In der Arbeit wurden auch kombinatorischen Eigenschaften des Spiels analysiert. Hierbei wurde der Bezug von Jass zu anderen Spielen der Klasse sogenannter "Imperfect Information Games" etabliert. Es konnte zudem gezeigt werden, dass die Komplexität von Jass insbesondere durch die grosse Anzahl von unbekannten Kartenzuteilungen am Anfang des Spiels bedingt ist. Die Schlussfolgerung war, dass reine "Machine Learning" Ansätze aufgrund der Komplexität des Spiels nicht erfolgreich sind.

Markus Stolze

Datei: Aufgabenstellung-2018-HS-JassBot3-V2-AeberliSchleutermann-
PraktischeAnwendungSpieltheoretischerAnsätze.docx



3. Ziele der Arbeit

Ziel dieser Arbeit ist es zu analysieren, inwieweit sich publizierte Ansätze für "Imperfect Information Games" für die Implementierung eines verbesserten Bots im Schieber-Jass nutzen lassen.

Startpunkt für diese Arbeit ist hierbei das Paper "DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker" (Science, March 02, 2017), welches die Kombination von "Machine Learning" Techniken und Algorithmen aus der Spieltheorie für die Berechnung geeigneter Aktionen beim Pokerspiel beschreibt.

In einem ersten Schritt soll ausgehend von diesem Paper eine Literatur-Recherche zu geeigneten algorithmischen Ansätzen zur Bestimmung von Spielzügen in stichbasierten Spielen gemacht werden. Vielversprechende Ansätze sollen dann auf Ihre Anwendbarkeit für das Spielen von Schieber-Jass analysiert werden. Die entsprechend durchgeführten Analysen und Experimente sind so zu dokumentieren, dass deutlich wird in welcher Reihenfolge und mit welchem Ziel diese gestartet wurden. Zudem sind daraus gewonnene Resultate und Schlussfolgerungen festzuhalten.

Bewertet wird die Stichhaltigkeit und Nachvollziehbarkeit des Vorgehens sowie die Weiterverwendbarkeit der Resultate (inkl. Source-Code).

4. Vereinbarte Rahmenbedingungen

Die Studierenden können sich Hilfe von Samuel Kurath (Mitarbeiter IFS) holen.

5. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die Dokumentation (inkl. Source-Code) ist vollständig entsprechend den Instruktionen des Studiengangs abzugeben. Zudem sind ein Download-Link für Prof. Dr. Markus Stolze sowie weitere Exemplare - nach Absprache mit dem Co-Referenten und dem Experten – bereitzustellen. Des Weiteren ist eine kurze Projektergebnisdokumentation im öffentlichen Wiki von Prof. Dr. Markus Stolze zu erstellen.

6. Weitere Regeln und Termine

Darüber hinaus gelten die allgemeinen Regeln zu Bachelor und Studienarbeiten
„Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik“ (HSR Intranet):
<https://www.hsr.ch/Ablaeufe-und-Regelungen-Studie.7479.0.html>

Der Terminplan ist hier ersichtlich (HSR Intranet):
<https://www.hsr.ch/Termine-Bachelor-und-Studien.5142.0.html>

6. Rechte

Die resultierende Software und Dokumentation soll möglichst als Open-Source Software publiziert werden, so dass sie ohne Einschränkungen sowohl von den Studenten, wie auch von der HSR weiter genutzt und erweitert werden kann. Bei der Programmierung soll darauf geachtet werden, dass möglichst keine Libraries mit viraler Open-Source Lizenz (z.B. GNU) genutzt werden.



7. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Entsprechend sollten ca. 350h Arbeit für die Bachelorarbeit aufgewendet werden. Dies entspricht ungefähr 25h pro Woche (auf 14 Wochen) und damit ca. 3 Tage Arbeit pro Woche pro Student.

Für die Beurteilung ist der HSR-Betreuer verantwortlich. Die Bewertung der Arbeit erfolgt entsprechend der verteilten Kriterienliste.

Diese definitive Aufgabenstellung wurde am Montag 29.10 beschlossen.

Rapperswil, Montag 29.10
Prof. Dr. Markus Stolze, Institut für Software, Hochschule für Technik Rapperswil

B Erfüllung der Anforderungen

CFR

Anforderung	Erfüllt?	Begründung
UC-01	Teilweise erfüllt	Der Bot kann zwar Karten ausspielen, jedoch nur für die letzten zwei Stiche (siehe Kapitel 8.2). Für alle Entscheidungen vor dem 2. letzten Stich, kann über den Proxy Bot die Kartenwahl des Challenge Bots verwendet werden.
UC-02	Teilweise erfüllt	Der Bot selbst führt die Trumpfwahl nicht durch. Mit dem Proxy Bot kann die Trumpfwahl aber dem Challenge Bot übergeben werden.
UC-03	Erfüllt	Der Bot kann über den <i>Server Player</i> in die Zühlke Umgebung eingebunden werden.
UC-04	Erfüllt	Der Bot kann als Player im <i>Pyschieber</i> eingebunden werden.
NFR-01	Erfüllt	Der Bot spielt eine Karte in weniger als 10 Sekunden.
NFR-02	Nicht erfüllt	Der Test wurde nicht durchgeführt, da die Resultate unter Umständen nicht realitätsgetreu sind (siehe Kapitel 7).
NFR-03	Nicht erfüllt	Praxistest konnte aus Zeitgründen nicht durchgeführt werden.
NFR-04	Erfüllt	Wurde durch Code Reviews und SonarQube Auswertungen belegt.

Tabelle 39: Erfüllung der Anforderungen für den CFR Bot

MCCFR

Anforderung	Erfüllt?	Begründung
UC-01	Erfüllt	Der Bot kann in jedem Stich eine Karte ausspielen.
UC-02	Teilweise erfüllt	Der Bot selbst führt die Trumpfwahl nicht durch. Über den Proxy Bot kann die Trumpfwahl aber dem Challenge Bot übergeben werden.
UC-03	Erfüllt	Der Bot kann über den <i>Server Player</i> in die Zühlke Umgebung eingebunden werden.
UC-04	Erfüllt	Der Bot kann als Player im <i>Pyschieber</i> eingebunden werden.
NFR-01	Erfüllt	Der Bot spielt eine Karte in weniger als 10 Sekunden.

NFR-02	Nicht erfüllt	Exact Binomial Test wurde nicht durchgeführt, da die Resultate bereits zeigen, dass der Bot schlechter spielt als der Challenge Bot.
NFR-03	Nicht erfüllt	Praxistest konnte aus Zeitgründen nicht durchgeführt werden.
NFR-04	Erfüllt	Wurde durch Code Reviews und SonarQube Auswertungen belegt.

Tabelle 40: Erfüllung der Anforderungen für den MCCFR Bot

MCTS

Anforderung	Erfüllt?	Begründung
UC-01	Erfüllt	Der Bot kann in jedem Stich eine Karte ausspielen.
UC-02	Teilweise erfüllt	Der Bot selbst führt die Trumpfwahl nicht durch. Über den Proxy Bot kann die Trumpfwahl aber dem Challenge Bot übergeben werden.
UC-03	Erfüllt	Der Bot kann über den Server Player in die Zühlke Umgebung eingebunden werden.
UC-04	Erfüllt	Der Bot kann als Player im Pyschieber eingebunden werden.
NFR-01	Erfüllt	Der Bot spielt eine Karte in weniger als 10 Sekunden.
NFR-02	Erfüllt	Siehe Kapitel 9.5
NFR-03	Erfüllt	Siehe Kapitel 9.6
NFR-04	Erfüllt	Wurde durch Code Reviews und SonarQube Auswertungen belegt.

Tabelle 41: Erfüllung der Anforderungen für den MCTS Bot

Benchmarking

Anforderung	Erfüllt?	Begründung
UC-05	Erfüllt	Die Vergleiche in Kapitel 9 wurden mit der Benchmark Umgebung durchgeführt. Daher ist dieser Punkt erfüllt.
UC-06	Erfüllt	Die Vergleiche in Kapitel 9 wurden mit der Benchmark Umgebung durchgeführt. Daher ist dieser Punkt erfüllt.

UC-07	Erfüllt	Experimente können mit den gleichen Voraussetzungen (Parameter, Kartenverteilung etc.) neu gestartet werden.
UC-08	Erfüllt	Die Vergleiche in Kapitel 9 wurden mit der Benchmark Umgebung durchgeführt. Diese Funktionalität wurde für die Messungen mit dem CFR Bot benötigt. Die Anforderung ist also erfüllt.
UC-09	Erfüllt	Die Vergleiche in Kapitel 9 wurden mit der Benchmark Umgebung durchgeführt. Daher ist dieser Punkt erfüllt.
UC-10	Erfüllt	Alle Experimente in Kapitel 9 wurden mittels Parallelisierung auf dem HSR Cluster ausgeführt.
NFR-05	Erfüllt	Wurde durch Code Reviews und SonarQube Auswertungen belegt.

Tabelle 42: Erfüllung der Anforderungen für die Benchmark Umgebung

C Lieferumfang

Alle Inhalte, welche während dieser Bachelorarbeit produziert wurden, befinden sich auf einem privaten GitLab Server. Der Zugriff erfolgt über folgende URL:

<https://git.narcodex.ch/bachelor-thesis/bachelor-thesis-project.git>

Die Ordnerstruktur der Arbeit ist dabei wie folgt aufgebaut:

Ordner	Inhalt
📁 admin	Administrative Dokumente (z.B. Eigenständigkeitserklärung, Issue Export, Open Issues)
📁 cluster	Docker Konfigurationsdateien für die Ausführung auf dem Cluster.
📁 infrastructure	Hinweise zur Einrichtung der verwendeten Infrastruktur.
📁 journal	$\text{\LaTeX}$ Quelldateien, Illustrationen und Quellenverweise des Projekt-journals.
📁 notes	Zusatzmaterial zur Thematik.
📁 presentation	Dateien für die Zwischenpräsentation.
📁 report	$\text{\LaTeX}$ Quelldateien, Illustrationen und Quellenverweise des technischen Berichts.
📁 src	Quellcode für Ansätze und Experimente.

Auf eine Installationsanleitung wurde im Rahmen dieses Berichts verzichtet. Es gibt aber zu jedem Bot, zu den Docker Container und zur Benchmark Umgebung README Dateien, in welchen die Installationsschritte beschrieben sind. Die README Dateien befinden sich hier:

- 📁 src ▶ bots ▶ cfr_bot ▶ README.md
- 📁 src ▶ bots ▶ mccfr_bot ▶ README.md
- 📁 src ▶ bots ▶ mcts_bot ▶ README.md
- 📁 src ▶ bots ▶ benchmarking ▶ README.md
- 📁 src ▶ cluster ▶ README.md
- 📁 src ▶ cluster ▶ Benchmark ▶ README.md
- 📁 src ▶ cluster ▶ FileBrowser ▶ README.md
- 📁 src ▶ cluster ▶ RemotePlay ▶ README.md

D Software und Libraries

Im Folgenden werden alle Libraries und Software Komponenten beschrieben, welche für die Lauffähigkeit des Projektes benötigt werden.

Da im Rahmen dieser Arbeit in erster Linie Prototypen entwickelt wurden, war es nicht sinnvoll eine aufwendige Nutzwertanalyse für jede Library durchzuführen. Ziel war es möglichst schnell ein lauffähiges Programm zu entwickeln. Daher wurden die Libraries vor allem aufgrund von ihrem Funktionsumfang und der Aktivität der Contributors ausgewählt. Zudem wurde darauf geachtet, dass keine der eingesetzten Python Libraries eine invasive OpenSource Lizenz verwendet, um eine problemlose Weiterentwicklung zu ermöglichen.

Die folgende Tabelle zeigt, welche Software im Projekt verwendet wurde:

Name	Version	Lizenz	Verwendung
Docker CE	18.06.1-ce	Apache 2.0 license	Wurde verwendet, um Benchmarks auf dem Cluster ausführen zu können und um die Bots in die Umgebung von Zühlke einzuklinken.
Python	3.7.1	Python Software Foundation License (PSFL)	Wurde verwendet, um die Bots zu implementieren.
TexLive	2018	https://www.tug.org/texlive/copying.html	Wurde verwendet, um die Dokumentation zu erstellen.
Biber	2.11	Artistic License 2.0	Wurde für die Verwaltung der Quellen in den Dokumenten verwendet.
Git	2.8.0	GPL-2.0	Wurde für die Versionierung und die Verwaltung der Projektdaten verwendet.
GNU Make	3.81	Hat die gleiche Lizenz, wie das System, mit dem Make ausgeliefert wird.	Wurde verwendet, um die Dokumente auf dem Build System regelmässig zu builden.

Tabelle 44: Verwendete Software

Die folgende Tabelle zeigt alle Libraries, welche in den Python Projekten verwendet wurden:

Name	Version	Lizenz	Verwendung
numpy	1.15.4	OSI Approved (BSD)	Wird verwendet, um Array Operationen in den Algorithmen effizient umzusetzen.

scipy	1.1.0	BSD License (BSD)	Wird für komplexere mathematische Operationen verwendet.
graphviz	0.10.1	MIT License	Wird verwendet, um die Graphen der Game Trees darzustellen.
twine	1.12.1	BSD License (BSD)	Wird verwendet, um die erstellten Python Pakete auf den eigenen Package Index Server hochzuladen.
setuptools	40.6.3	MIT License	Wird für das Python Package Management verwendet (build, install, upgrade etc.).
websockets	7.0	BSD License (BSD)	Wird vom Remote Player verwendet, um eine Websocketverbindung zur Zühlke Umgebung aufzubauen.
pyschieber	1.3.2	MIT	Wird verwendet, um Schieber Jass Spiele simulieren zu können.
openpyxl	2.5.12	MIT License (MIT/Expat)	Wird verwendet, um die Resultate des Benchmarking in Excel Dateien zu schreiben.
joblib	0.13.0	MIT License	Wird verwendet, um Benchmarks parallel ausführen zu können.
arrow	0.12.1	Apache Software License (Apache 2.0)	Wird in der Benchmark Umgebung verwendet, um Zeitberechnungen durchzuführen.

Tabelle 45: List aller verwendeten Python Libraries

E Rechtliche Hinweise

Alle Abbildungen von Jasskarten und Trumpfsymbolen in diesem Bericht sind Eigentum der Firma Digipress, Schulgasse 5, 8400 Winterthur. Jürg Schleutermann hat die Nutzungsrechte für diese Grafiken erworben. Ausserhalb dieser Arbeit dürfen sie nicht weiterverwendet werden.

F Glossar

Action Action ist ein Begriff aus der Spieltheorie und bezeichnet eine einzelne Entscheidung. Siehe Kapitel 5.2. 21–24, 26, 55, 60, 61, 64

Action Node Action Node ist ein Begriff aus der Spieltheorie. Ein Action Node ist ein Knoten (Node) im Game Tree, von dem aus ein Spieler eine Action wählen kann. Siehe Kapitel 5.2. 64

Banner Die Karte 10 heisst im Jassen auch Banner. 90, 91

Benchmark Umgebung Die Benchmark Umgebung steht in diesem Kontext für ein Python Softwareprojekt, welches die Aufgabe hat Pyschieber Jass Bots miteinander zu vergleichen. 10, 12, 41, 42, 45, 49, 51, 74, 77, 82, 85, 92, 98–100, 102, 110, 112, 113

bluffen Beim bluffen versucht man Andere mit seinem Verhalten so zu beeinflussen, dass man selber einen Vorteil daraus zieht. 22

Browser Typischerweise meint man mit einem Browser einen Webbrowser. Dabei handelt es sich um ein Programm, mit welchem Internetseiten aufgerufen werden können. 8, 43

CFR CFR steht für Counterfactual Regret Minimization. Siehe Kapitel 8.2. 40, 54, 57, 59, 60, 64, 77, 80, 81, 84, 93, 97, 99, 110, 113

Challenge Bot Challenge Bot ist die Bezeichnung für den bisher erfolgreichsten verfügbaren Jass Bot. Der Bot arbeitet rein regelbasiert. 6, 9, 45–47, 50, 53, 74, 77–89, 92, 93, 97, 98

Chance Node Chance Node ist ein Begriff aus der Spieltheorie. Ein Chance Node ist Knoten (Node) im Game Tree, bei dem eine zufällige Auswahl getroffen wird. Siehe Kapitel 5.2. 24

Cluster Siehe Kapitel 6.2. 41–43, 99–101, 110

Complete Information Complete Information ist ein Begriff aus der Spieltheorie und bezeichnet eine Charakteristik für ein Spiel, bei dem es klar definierte Regeln gibt. Diese Regeln müssen zudem allen Spielern bekannt sein. Siehe Kapitel 5.2. 22, 27

Counterfactual Regret Der Counterfactual Regret ist ein mit der Reach Probability gewichteter Regret. Siehe Kapitel 8.2. 56, 57, 59, 60, 62

Deep Learning Gehört zu den Machine Learning Methoden. Deep Learning Algorithmen arbeiten meist mit so genannten Neuronalen Netzen, welche versuchen Muster in Daten zu erkennen. 3, 6

deterministisch In der Informatik ist ein Programm deterministisch, wenn durch den gleichen Input immer wieder der gleiche Output produziert wird. Das Programm hat also kein zufälliges Verhalten. 21, 48

Docker Bei Docker handelt es sich um eine Umgebung zur Virtualisierung von Betriebssystemen mittels so genannten Containern. 41–43, 110

Docker Volume Bei Docker Volumes handelt es sich um eine Abstraktion von Speicher. Docker Container können Volumes verwenden um Daten zu persistieren. 42, 43

Doppelkopf Doppelkopf ist ein vorallem in Deutschland verbreitetes 4 Personen Kartenspiel. Doppelkopf zeigt viele Parallelen zum Schieber Jass. 64

ECTS ECTS steht für European Credit Transfer and Accumulation System. In diesem System erhält man Punkte (Credits) für den Fortschritt in der Ausbildung. 6

Erwartungswert Der Begriff Erwartungswert stammt aus der Statistik und beschreibt, welchen Wert eine Zufallsvariable (z.B. Messungen) theoretisch im Mittel erreichen sollte. 71

Exact Binomial Test Statistischer Hypothesentest für ein Bernoulli-Experiment (Experiment mit zwei Versuchsausgängen). 9, 88, 98

Excel Excel ist ein Tabellenkalkulationsprogramm von Microsoft. 11, 49, 102

Extensive Form Game Extensive Form Game ist ein Begriff aus der Spieltheorie. Ein Spiel in Extensive Form wird in Runden abgehandelt. In jeder Runde ist nur ein Spieler am Zug. Siehe Kapitel 5.2. 23, 27, 54

Game Tree Ein Game Tree ist ein Begriff aus der Spieltheorie. Dabei handelt es sich um eine Repräsentation eines Spiels in einer Baumstruktur. Siehe Kapitel 5.2. 23, 24, 27, 57, 59, 64, 102, 110

GitLab GitLab ist ein Projektplanung und Source Code Management Tool. Zudem bietet es Funktionalitäten für Continuous Integration und Continuous Deployment. 42

Go Go ist ein strategisches Brettspiel für zwei Personen, bei dem es darum geht, möglichst viele Territorien des Gegners zu erobern. 22

Greedy Bot Ein Greedy Bot ist ein Bot für die Pyschieber Umgebung. Er wählt immer die höchste Karte gemäss den Jass Regeln aus. 6

Heads-Up No-Limit Poker Head-Up No-Limit Poker ist eine zweispieler Variante von Poker. 54

History Eine History im Sinne der Spieltheorie entspricht einer Sequenz von Actions. Siehe Kapitel 5.2. 24

Imperfect Information Imperfect Information ist ein Begriff aus der Spieltheorie und bezeichnet eine Charakteristik für ein Spiel, bei dem es versteckte Informationen gibt. Siehe Kapitel 5.2. 22, 24, 27, 54, 55

Information Set Ein Information Set ist ein Begriff aus der Spieltheorie. Es umspannt mehrere Knoten (Nodes) in einem Game Tree. Diese Knoten bilden die Informationen ab, welche ein Spieler über ein Spiel besitzt. Siehe Kapitel 5.2. 24, 56, 57, 60–62

Jass Bot Ein Jass Bot ist ein Computerprogramm, welches die Entscheidungen in einem Jass Spiel übernehmen kann. 3, 5, 7–13, 28, 110, 111

Jass Partie In einer Jass Partie werden mehrere Spiele gespielt und die Punkte pro Team aufsummiert. Wer zuerst die vorgegebene Punktzahl erreicht hat, gewinnt die Partie. Üblicherweise wird beim Schieber Jass auf 1000 oder 2500 Punkte gespielt.. 74, 90, 113

Machine Learning Machine Learning ist eine Methode, mit welcher durch Datenanalyse von Daten gelernt wird. 6

MCCFR MCCFR steht für Monte Carlo Counterfactual Regret Minimization. Siehe Kapitel 8.3. 40, 59, 82–84, 93

MCTS MCTS steht für Monte Carlo Tree Search. Siehe Kapitel 8.4. 40, 64–67, 69–71, 73, 85–93, 98, 111, 113

Nash Equilibrium Nash Equilibrium ist ein Begriff aus der Spieltheorie. Es handelt sich dabei um eine Menge von Strategie Profilen (eins pro Spieler), bei denen es sich für keinen der Spieler lohnt von ihren Strategien abzuweichen. Siehe Kapitel 5.2. 25, 26, 55, 57, 58

Normal Form Game Normal Form Game ist ein Begriff aus der Spieltheorie. Bei einem Spiel in Normal Form wählen alle Spieler ihre Actions gleichzeitig aus. Siehe Kapitel 5.2. 23

Obenabe “Obenabe” ist ein Begriff aus dem Jassen. Es handelt sich dabei um eine Trumpfwahl, bei der die Stichreihenfolge vom Ass bis zur 6 geht. Siehe Kapitel 5.1. 16, 18–20, 91

OpenSource Open Source ist ein spezieller Typ von Computersoftware, bei dem der Entwickler den Code unter einer offenen Lizenz publiziert. Diese Lizenz erlaubt die freie Verwendung der Software (in gewissen Abstufungen). 101

Partnerjass Ein Partnerjass ist ein Jass, in dem die Teilnehmer jeweils mit einem Partner zusammenspielen. 15

PEP8 PEP8 ist ein Coding Standard für Python. 9, 12

Perfect Information Perfect Information ist ein Begriff aus der Spieltheorie und ist eine Charakteristik für ein Spiel, bei dem es keine versteckten Informationen gibt. Siehe Kapitel 5.2. 22, 24, 55

Poker Poker ist eine Gruppe von Spielen in denen vorallem das Bluffen, strategisches Spiel und Geldeinsätze eine wichtige Rolle spielen. 22, 54, 64

Prisoners Dilemma Das Prisoners Dilemma ist ein Standardbeispiel aus der Spieltheorie, an welchem sich viele Konzepte einfach demonstrieren lassen. Siehe Kapitel 5.2. 25, 110

Private State Private State ist ein Begriff aus der Spieltheorie. Der Private State eines Spieles umfasst alle Informationen, welche nur einem teil der Spieler zur Verfügung stehen. Siehe Kapitel 5.2. 24, 32

Prototyp Ein Prototyp ist in der Softwareentwicklung eine Art Vorabversion des eigentlichen Programms. 5–7, 92, 101

Proxmox Proxmox ist ein Tool zur Server Virtualisierung. 42

Proxy Bot Ein Proxy Bot ist ein Bot für die Pyschieber Umgebung, und leitet alle Funktionsaufrufe an dahinterliegende Bots weiter. 45–48, 50, 51, 53, 93

Public State Public State ist ein Begriff aus der Spieltheorie. Der Public State eines Spieles umfasst alle Informationen, welche allen Spielern zur Verfügung stehen. Siehe Kapitel 5.2. 24, 32

Pyschieber Beim Pyschieber handelt es sich um ein Softwareprojekt von Samuel Kurath, in welchem Schieber Jass Partien simuliert werden können. 6, 8, 39–41, 43, 46, 50, 52, 53, 92, 97, 98

Python Python ist eine Programmiersprache, welche nach dem OpenSource Prinzip funktioniert. Sie gilt als leicht zu erlernen und soll schnell sein. 4, 8, 9, 12, 49, 50, 58, 62, 73, 92, 101, 102, 113

R R ist eine Programmiersprache, welche vorallem bei statischen Berechnungen zum Einsatz kommt. 89

Random Bot Ein Random Bot ist ein Bot für die Pyschieber Umgebung und trifft alle Entscheidungen zufällig. 6, 46, 74

Reach Probability Die Reach Probability ist ein numerischer Wahrscheinlichkeitswert, welcher angibt, wie wahrscheinlich es ist, dass ein gewisser Zustand in einem erreicht wird. Siehe Kapitel 8.2. 57

Regret Regret ist ein numerischer Wert, welcher angibt, wie stark man eine Entscheidung bereut. Der Begriff stammt aus der Spieltheorie. Siehe Kapitel 8.2. 55–57, 60, 110

Regret Matching Regret Matching beschreibt den Prozess, mit dem im CFR Algorithmus neue Strategien berechnet werden. Siehe Kapitel 8.2. 56, 57, 62

Reinforcement Learning Reinforcement Learning gehört zu den Methoden des Machine Learnings. Beim Reinforcement Learning wird versucht Modelle durch die Interaktion mit der Umgebung zu trainieren. Das Modell soll also quasi selbst lernen. 5, 6

Reward Reward ist ein Begriff aus der Spieltheorie und bezeichnet den quantifizierten Gewinn, den man Ende von einem Spiel erhält. Siehe Kapitel 5.2. 22–26, 57, 58, 60, 66, 74

Schach Schach ist ein strategisches Brettspiel für zwei Personen, bei dem es darum geht, als erster den König des anderen Spielers Bewegungsunfähig zu machen. 22

Schere, Stein, Papier “Schere, Stein, Papier” ist ein einfaches Zweispielerspiel bei dem beide Spieler gleichzeitig eine Action auswählen (Schere, Stein oder Papier). 22, 23, 25, 26, 110

Schieben Schieben ist ein Begriff aus dem Schieber Jass. Wenn man eine Trumfwahl nicht selber treffen möchte, kann man die Wahl an seinen Partner übergeben. Siehe Kapitel 5.1. 39, 91

Schieber Jass Jassen ist ein schweizer Kartenspiel, bei dem verschiedene Spieler in Runden gegeneinander antreten. Mehr zum Schieber Jass in Kapitel 5.1. 3–6, 8, 15, 17, 20, 26–28, 33, 39, 41, 44, 52, 54, 55, 57, 64–67, 69, 71, 73, 84, 93, 102, 110

Skat Skat ist ein deutsches Stichtkartenspiel ähnlich wie Jassen. 22, 64

SonarQube SonarQube ist ein Tool welches mittels statischer Code Analyse die Qualität der Software einschätzt. 97–99

Spieltheorie Im Englischen auch Game Theory genannt. Die Spieltheorie befasst sich mit der Theorie der Entscheidungsfindung. Siehe Kapitel 5.2. 3, 4, 6

Stich Stich ist ein Jass Begriff. Beim Stich handelt es sich um den Prozess der Auswertung einer Runde im Jassen. Nachdem alle Spieler eine Karte ausgespielt haben, wird die Runde nach den Spielregeln ausgewertet. Siehe Kapitel 5.1. 17–19, 29, 30, 32, 33, 37, 39, 47, 50, 55, 58, 59, 62, 73, 74, 77–80, 90, 97, 98, 110, 111

Strategy Profile Strategy Profile ist ein Begriff aus der Spieltheorie und bezeichnet den Zusammenschluss mehrerer Strategien. Siehe Kapitel 5.2. 25

Supervised Learning Supervised Learning gehört zu den Methoden des Machine Learnings. Die daraus entstehenden Modelle werden mit existierenden Daten trainiert und getestet. 5

Terminal History Terminal History ist ein Begriff aus der Spieltheorie. Eine Terminal History ist eine Sequenz von Actions, welche zu einem Terminal Node führen. Sie bildet somit einen kompletten Spielverlauf ab. Siehe Kapitel 5.2. 24, 60

Terminal Node Terminal Node ist ein Begriff aus der Spieltheorie. Es handelt sich dabei um einen der letzten Knoten (Blattknoten) in einem Game Tree (Ende des Spieles). Siehe Kapitel 5.2. 24, 57, 65

Trumpf Trumpf ist ein Begriff aus dem Jassen. Der Trumpf gibt den Spielmodus vor und steuert somit die Punktwertung und Stichreihenfolge der Karten. Siehe 5.1. 8, 16–20, 32, 33, 37, 39, 46–48, 52, 53, 91–93, 97, 98, 112

Tupel Ein Tupel ist eine Zusammenfassung von Werte, bei denen die Reihenfolge eine Rolle spielt. 35

UCT UCT steht für Upper Confidence Bound applied to trees. Siehe Kapitel 8.3. 64, 71

Undenufe “Undenufe” ist ein Begriff aus dem Jassen. Es handelt sich dabei um eine Trumpfwahl, bei der die Stichreihenfolge von der 6 bis zum Ass geht. Siehe Kapitel 5.1. 16, 18–20, 91

Untertrumpfen Untertrumpfen ist ein Begriff aus dem Jassen. Von Untertrumpfen spricht man, wenn die angegebene Farbe nicht die Trumpffarbe ist und man eine tiefere Trumpfkarte auf eine höhere legt. Siehe Kapitel 5.1. 18, 19, 33, 110

Validierung Mit der Validierung wird geprüft, ob etwas den Anforderungen der Stakeholder entspricht. 10, 11, 74, 92

Verifikation Mit der Verifikation wird etwas auf seine formale Spezifikation hin geprüft (Korrektheit). 10

Vertrauensintervall Ein Vertrauensintervall ist ein Begriff aus der Statistik. Der Intervall ist ein Mass für die Präzision einer Schätzung. 71

WebSocket Über Websockets kann eine bidirektionale TCP Verbindung zwischen Client und Server aufgebaut werden. 43, 102

Zero-Sum Game Zero-Sum Game ist ein Begriff aus der Spieltheorie und bezeichnet ein Spiel, bei dem sich Gewinne und Verluste aller Spieler gegenseitig ausgleichen. Siehe Kapitel 5.2. 22, 26, 27

Zühlke Die Zühlke Engineering AG ist eine Business Management Consultant Firma mit Sitz in Schlieren. 3, 5, 8, 41, 43, 90, 97, 98, 101, 102

G Verzeichnisse

G.1 Abbildungsverzeichnis

1	Screenshot der Jass Umgebung von Zühlke	5
2	Use Cases für die Jass Bots	7
3	Use Cases für die Benchmark Umgebung	10
4	Spieleraufstellung beim Schieber Jass	15
5	Ausführung eines Sticks beim Schieber Jass	17
6	Beispielsituation für Untertrumpfen (Beispiel aus [11])	18
7	“Schere, Stein, Papier” abgebildet in einer Matrix	23
8	Beispiel eines Game Trees für ein fiktives Spiel	24
9	Illustration für History und Informations Sets	25
10	Matrix Abbildung für “Prisoners Dilemma” und “Schere, Stein, Papier”	25
11	Konzepte der Spieltheorie angewendet auf den Schieber Jass	27
12	Buchhaltung über mögliche Handkarten pro Spieler	34
13	Übersicht der Python Packages und ihrer Abhängigkeiten	40
14	Ausführung von Benchmarks auf dem Cluster	41
15	Verwaltung der Docker Container auf dem Cluster	42
16	Grafische Dateiverwaltung auf dem Cluster	42
17	Integration der Bots in die grafische Spielumgebung via Cluster	43
18	Übersicht des Testablaufs beim Benchmarking	44
19	Schematische Darstellung des Proxy Bots	46
20	Das Spiel wird vor der Messung in die Ausgangsposition gebracht	47
21	Zusammensetzung eines Benchmark Resultats	48
22	Generierte Dateien mit den Resultaten eines Benchmarks	49
23	Darstellung des Regrets für den Schieber Jass	55
24	Reach Probability und Strategy	56
25	Übersicht über den Ablauf des “Vanilla” CFR	57
26	Sampling Varianten für den MCCFR	59

27	Übersicht über den Ablauf des MCCFR	62
28	Knoten und Übergänge im Baum des MCTS	65
29	Ausbaustadien eines einzelnen Knotens bei MCTS	66
30	Ablauf des MCTS Algorithmus	66
31	Selektion im Baum durch MCTS Algorithmus	67
32	Expansion des Baums durch MCTS Algorithmus	68
33	Simulation zur Bewertung von Knoten durch MCTS Algorithmus	69
34	Rückpropagierung des Simulationsausgangs durch MCTS Algorithmus	70
35	Tiefensuche im Baum durch MCTS Algorithmus	71
36	Breitensuche im Baum durch MCTS Algorithmus	71
37	UCT Bewertungsbonus mit einem Erkundungsfaktor von 0.4	72
38	Benchmark Spielvariationen (Semi Random)	74
39	Anteil gewonnener Spiele bei 1000 Handsituationen (Semi Random)	75
40	Anteil gewonnener Spiele bei 10000 Handsituationen (Semi Random)	76
41	Benchmark Spielvariationen (CFR)	77
42	Erspielte Punkte in den letzten zwei Stichen bei 1000 Handsituationen (CFR)	78
43	Erspielte Punkte in den letzten zwei Stichen bei 10000 Handsituationen (CFR)	79
44	Erspielte Punkte in den letzten zwei Stichen bei 50000 Handsituationen (CFR)	80
45	Benchmark Spielvariationen (MCCFR)	82
46	Anteil gewonnener Spiele bei 1000 Handsituationen (MCCFR)	83
47	Anteil gewonnener Spiele bei 10000 Handsituationen (MCCFR)	84
48	Benchmark Spielvariationen (MCTS)	85
49	Anteil gewonnener Spiele bei 1000 Handsituationen (MCTS)	86
50	Anteil gewonnener Spiele bei 10000 Handsituationen (MCTS)	87
51	Überblick über alle gemessenen Gewinnraten	88

G.2 Tabellenverzeichnis

1	Übersicht über die Akteure und ihre Ziele	7
2	Nichtfunktionale Anforderungen an die Jass Bots nach [6]	9

3	Nichtfunktionale Anforderungen an die Benchmark Umgebung nach [6]	12
4	Ausgeschlossene Charakteristiken nach [6]	13
5	Übersetzung zwischen deutschen und französischen Karten (nach [10])	14
6	Spielkarten für das Jass Spiel	14
7	Auswahlmöglichkeiten bei der Trumpfwahl	16
8	Under Karte in Trumpffarbe dient als Joker	17
9	Rangordnung der Karten in Abhängigkeit von der Trumpfwahl	18
10	Punktwerte der Karten in Abhängigkeit von der Trumpfwahl	19
11	Multiplikator für Punkte in Abhängigkeit von der Trumpfwahl	20
12	Mögliche Kartenverteilungen pro Stich aus Spielersicht	29
13	Mögliche Ausgänge pro Stich aus Spielersicht	30
14	Mögliche Spielausgänge pro Stich aus Spielersicht	31
15	Beschreibung der Python Packages	41
16	Mögliche Spieleraufstellung für Benchmarking	45
17	Schema für die Bots im Hintergrund des Proxy Bots	46
18	Parameter für die Benchmark Umgebung	51
19	Mögliche Parameter für die Ausführung des CFR Bots	58
20	Mögliche Parameter für die Ausführung des MCCFR Bots	63
21	Mögliche Parameter für die Ausführung des MCTS Bots	73
22	Benchmark Parameter (Semi Random)	75
23	Benchmark Resultate bei 1000 Handsituationen (Semi Random)	75
24	Benchmark Resultate bei 10000 Handsituationen (Semi Random)	76
25	Bot Parameter (CFR)	77
26	Benchmark Parameter (CFR)	77
27	Benchmark Resultate bei 1000 Handsituationen (CFR)	78
28	Benchmark Resultate bei 10000 Handsituationen (CFR)	79
29	Benchmark Resultate bei 50000 Handsituationen (CFR)	80
30	Bot Parameter (MCCFR)	82
31	Benchmark Parameter (MCCFR)	82

32	Benchmark Resultate bei 1000 Handsituationen (MCCFR)	83
33	Benchmark Resultate bei 10000 Handsituationen (MCCFR)	84
34	Benchmark Parameter (MCTS)	85
35	Bot Parameter (MCTS)	85
36	Benchmark Resultate bei 1000 Handsituationen (MCTS)	86
37	Benchmark Resultate bei 10000 Handsituationen (MCTS)	87
38	Erzielte Punkte in 5 Jass Partien (Mensch gegen MCTS Bot)	90
39	Erfüllung der Anforderungen für den CFR Bot	97
40	Erfüllung der Anforderungen für den MCCFR Bot	98
41	Erfüllung der Anforderungen für den MCTS Bot	98
42	Erfüllung der Anforderungen für die Benchmark Umgebung	99
44	Verwendete Software	101
45	List aller verwendeten Python Libraries	102

G.3 Quellenverzeichnis

- [1] (3. Okt. 2018). Jassen: Startschuss zur 2. Schweizer Schieber-Meisterschaft, Blick, Adresse: <https://www.blick.ch/brand-studio/startschuss-zur-2-schweizer-schieber-meisterschaft-sticht-jasskoenig-ruedi-wieder-alle-aus-id8932344.html> (besucht am 19. 12. 2018).
- [2] S. Stefanelli. (2017). Wenn Mighty Hans auf Monika Fasnacht trifft, Adresse: <https://www.zuehlke.com/blog/jass-challenge-wenn-mighty-hans-auf-monika-fasnacht-trifft/> (besucht am 02. 12. 2018).
- [3] A. Eder und P. Kistler, „JassBot mit Machine Learning“, en, bachelor, HSR Hochschule für Technik Rapperswil, Dez. 2017. Adresse: <https://eprints.hsr.ch/617/> (besucht am 27. 11. 2018).
- [4] Samuel Kurath, „Schieber Jass Bot - Creating an autonomous player by applying reinforcement learning“, English, Master Team Project, HSR Hochschule für Technik Rapperswil, Rapperswil, 2018.
- [5] C. Larman, *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process*, en. Prentice Hall Professional, 2002, ISBN: 978-0-13-092569-5.
- [6] International Organization for Standardization (ISO), *ISO/IEC 25010:2011*, en. Adresse: <http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/57/35733.html> (besucht am 17. 12. 2018).

- [7] *JavaScript Challenge Server to play the Swiss card game Jass (Schieber): webplatform-z/challenge*, original-date: 2014-12-20T09:51:09Z, 27. Nov. 2018. Adresse: <https://github.com/webplatformz/challenge> (besucht am 10. 12. 2018).
- [8] S. Kurath, *pyschieber is a playground for popular Swiss Schieber Jass game, it provides an API to the game as well as a CLI: Murthy10/pyschieber*, original-date: 2017-12-29T17:10:40Z, 27. Nov. 2018. Adresse: <https://github.com/Murthy10/pyschieber> (besucht am 10. 12. 2018).
- [9] (). PEP 8 – style guide for python code, Python.org, Adresse: <https://www.python.org/dev/peps/pep-0008/> (besucht am 13. 12. 2018).
- [10] D. Müller, *Stöck - Wys - Stich - Der neue Schweizer Jassführer*, 1. Aufl. Zürich: Fona Verlag AG, 2016, ISBN: 978-3-037-81091-0.
- [11] A. Muff. (2018). Jassregel: Ist Untertrumpfen erlaubt?, Adresse: <https://www.jassverzeichnis.ch/index.php/blog/163-jassregel-ist-untertrumpfen-erlaubt> (besucht am 03. 12. 2017).
- [12] S. Winter, *Grundzüge der Spieltheorie: Ein Lehr- und Arbeitsbuch für das (Selbst-)Studium*, en. Gabler Verlag, 2015, ISBN: 978-3-662-44422-1. Adresse: <http://www.springer.com/de/book/9783662444221> (besucht am 28. 11. 2018).
- [13] M. Lanctot, *Monte Carlo Sampling and Regret Minimization for Equilibrium Computation and Decision-Making in Large Extensive Form Games*, en, Juni 2013. DOI: 10.7939/R3K085. Adresse: <https://era.library.ualberta.ca/items/e1a34eb8-e7e9-4397-a534-a1e45a73e6e9> (besucht am 27. 11. 2018).
- [14] J. Levin, „Games of Incomplete Information“, en, S. 10, 2002.
- [15] S. Kuhn, „Prisoner’s Dilemma“, in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Hrsg., Spring 2017, Metaphysics Research Lab, Stanford University, 2017. Adresse: <https://plato.stanford.edu/archives/spr2017/entries/prisoner-dilemma/> (besucht am 04. 12. 2018).
- [16] H. Dias, *Web File Browser which can be used as a middleware or standalone app: filebrowser/filebrowser*, original-date: 2015-09-11T21:21:58Z, 11. Dez. 2018. Adresse: <https://github.com/filebrowser/filebrowser> (besucht am 12. 12. 2018).
- [17] E. Gamma, R. Helm, R. Johnson und J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software (Adobe Reader)*, en. Pearson Education, Okt. 1994, Google-Books-ID: 6oHuKQe3TjQC, ISBN: 978-0-321-70069-8.
- [18] (). random – Generate pseudo-random numbers – Python 3.7.2rc1 documentation, Adresse: <https://docs.python.org/3.7/library/random.html> (besucht am 14. 12. 2018).
- [19] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson und M. Bowling, „DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker“, *Science*, Jg. 356, Nr. 6337, S. 508–513, Mai 2017, arXiv: 1701.01724, ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aam6960. Adresse: <http://arxiv.org/abs/1701.01724> (besucht am 27. 11. 2018).
- [20] Justin Sermeno, *Vanilla Counterfactual Regret Minimization for Engineers*, Blog. Adresse: <https://justinsermeno.com/posts/cfr/> (besucht am 27. 11. 2018).

- [21] J. Schäfer, „The UCT algorithm applied to games with imperfect information“, Diploma Thesis, Otto-von-Guericke-Universität, Magdeburg, 2007. Adresse: http://www.b0n541.net/study/jan_schaefer_uct_algorithm_applied_to_games_with_imperfect_information.pdf (besucht am 15. 12. 2018).
- [22] S. Sievers und M. Helmert, „A Doppelkopf Player Based on UCT“, in *Proceedings of the 38th Annual German Conference on Artificial Intelligence*, S. Holldobler, M. Krötzsch, R. Penaloza und S. Rudolph, Hrsg., Cham: Springer, 2015, S. 151–165, ISBN: 978-3-319-24488-4. DOI: [info:doi/10.1007/978-3-319-24489-1_12](https://doi.org/10.1007/978-3-319-24489-1_12). Adresse: <http://edoc.unibas.ch/43154/> (besucht am 15. 12. 2018).
- [23] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis und S. Colton, „A Survey of Monte Carlo Tree Search Methods“, *IEEE Transactions on Computational Intelligence and AI in Games*, Jg. 4, Nr. 1, S. 1–43, März 2012, ISSN: 1943-068X. DOI: [10.1109/TCIAIG.2012.2186810](https://doi.org/10.1109/TCIAIG.2012.2186810).
- [24] int8, *Monte Carlo Tree Search - beginners guide*, en-GB, März 2018. Adresse: <https://int8.io/monte-carlo-tree-search-beginners-guide/> (besucht am 27. 11. 2018).
- [25] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v. d. Driessche, T. Graepel und D. Hassabis, „Mastering the game of go without human knowledge“, *Nature*, Jg. 550, Nr. 7676, S. 354, Okt. 2017, ISSN: 1476-4687. DOI: [10.1038/nature24270](https://doi.org/10.1038/nature24270). Adresse: <https://www.nature.com/articles/nature24270> (besucht am 17. 12. 2018).
- [26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel und D. Hassabis, „Mastering the game of Go with deep neural networks and tree search“, en, *Nature*, Jg. 529, Nr. 7587, S. 484–489, Jan. 2016, ISSN: 0028-0836, 1476-4687. DOI: [10.1038/nature16961](https://doi.org/10.1038/nature16961). Adresse: <http://www.nature.com/articles/nature16961> (besucht am 17. 12. 2018).