Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Simulation und Graphik

Bakkalaureatsarbeit

Monte Carlo Simulation im Skat

Verfasser:

Jan Schäfer

18. Dezember 2005

Betreuer:

Dr. Michael Buro
University of Alberta
Department of Computing Science
Edmonton, Alberta
Canada

Dr. Knut Hartmann
Otto-von-Guericke-Universität Magdeburg
Institut für Simulation und Graphik
Magdeburg
Deutschland

Schäfer, Jan:
Monte Carlo Simulation im Skat
Bakkalaureatsarbeit
Otto-von-Guericke-Universität Magdeburg, 2005.

Danksagung

Mein Dank gilt allen Menschen, die mich bei der Erstellung dieser Arbeit unterstützt und in diesem Vorhaben bestärkt haben. Besonders herausheben möchte ich Michael Buro, der immer ein offenes Ohr für mich hatte. Trotz der großen räumlichen Entfernung hatte ich nie das Gefühl, alleine gelassen zu sein. Das Bereitstellen von Rechnerkapazitäten war außerdem eine große Hilfe. Gunther Gerhardt und Sebastian Kupferschmid verdienen ebenfalls ein Dankeschön für die Bereitstellung des Quellcodes ihrer Programme XSkat und Double Dummy Skat Solver. Ohne diese Quellen wäre die Arbeit nicht möglich gewesen. Knut Hartmann danke ich für die Betreuung der Arbeit an der Otto-von-Guericke-Universität und für die wertvollen Hinweise beim Durchsehen der Arbeit. Nicht zuletzt möchte ich meiner Familie, meinen Freunden und Arbeitskollegen für ihr Verständnis und die Unterstützung danken. Sie mussten in den letzten Wochen viel zu oft mein Gerede über Skat ertragen. Ina Schulze und Angela Schäfer danke ich speziell für die Verbesserung meiner Rechtschreibfehler.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Abkürzungen

DDSS	Double Dummy Skat Solver
DSkV	Deutscher Skatverband e.V.
GIB	Goren In a Box oder Ginsberg's Intelligent Bridgeplayer
IRC	Internet Relay Chat
ISkO	Internationale Skatordnung
ISPA-World	International Skat Players Association e.V.
KI	Künstliche Intelligenz
MC	Monte Carlo
MCU	Monte Carlo Simulation mit unvollständiger Information

Kapitel 1

Einleitung

Seit den Anfängen der Informatik beschäftigen sich Wissenschaftler auch mit Spielen. Schon in den 40er und 50er Jahren des letzten Jahrhunderts setzten sie sich mit der Frage auseinander, wie man Computern das Spielen beibringt.[RN04] In den 90er Jahren wurden für einige Spiele erstaunliche Erfolge erzielt. So spielen heute Computerprogramme für Schach, Othello (Reversi) und Dame auf Weltmeisterniveau oder sogar besser als jeder Mensch.[Sch01] Alle drei genannten Spiele gehören zur Gruppe der Spiele mit vollständiger Information. Das heißt, alle Spieler haben jederzeit einen kompletten Überblick über den Spielstand. Für diese Spiele werden Suchbäume mit möglichen Spielständen aufgebaut, in denen mit dem Minimax-Algorithmus, die optimale Spielstrategie berechnet werden kann.

Es gibt aber bis heute Spiele, bei denen die gängigen Ansätze nicht funktionieren. Das Brettspiel Go ist robust gegen die Alpha-Beta-Suche[1], weil es einen sehr hohen Verzweigungsfaktor im Suchbaum aufweist. Der Aufbau eines vollständigen Suchbaumes ist selbst mit den Kapazitäten heutiger Rechner nicht möglich. Demgegenüber stehen Spiele mit unvollständiger Information. Bei ihnen haben nicht alle Spieler das gleiche Wissen über den Spielstand. Die Kartenspiele Poker, Bridge und Skat sind klassische Vertreter für diese Gruppe von Spielen. Hier können die Suchalgorithmen nicht angewendet werden, weil nicht alle Informationen zum Aufbau des Suchbaumes verfügbar sind. In [BMvL96] wurde sogar gezeigt, dass die Berechnung einer optimalen Strategie für diese Gruppe von Spielen wegen der Größe des Suchbaumes $\mathcal{NP}$-hart ist.

Ein alternativer Ansatz ist das Überführen von Spielen mit unvollständiger Information auf Spiele mit vollständiger Information. Er wurde für das Kartenspiel Bridge in [Lev89] erstmals vorgeschlagen. Eine Umsetzung ist in [Gin99] beschrieben. Dabei werden für die unbekannten Informationen Werte erzeugt, die zum aktuellen Spielverlauf passen. Man spricht dabei vom Erzeugen möglicher Weltzustände oder einfach vom Er-

[1] Die Alpha-Beta-Suche ist eine Optimierung des Minimax-Algorithmus.

zeugen möglicher Welten.[FB98] Jetzt kann jede erzeugte Welt als Spiel mit vollständiger Information betrachtet werden. Durch mehrmaliges Erzeugen dieser Welten hofft man, eine gute Stichprobe aus der Menge aller möglichen Welten zu ziehen, um die gewonnenen Ergebnisse auch auf das Spiel mit unvollständiger Information anzuwenden. Für das Kartenspiel Bridge wurde mit diesem Ansatz erstmals ein Computerprogramm entwickelt, dessen Spielstärke mit einem durchschnittlichen menschlischen Spieler mithalten konnte und in einigen Situationen auch auf Expertenniveau spielen konnte.

In dieser Arbeit werden in Kapitel 2 zunächst die Spielregeln für das Skatspiel erklärt und die bisherigen Arbeiten zu den Kartenspielen Bridge und Skat vorgestellt. Der neue Ansatz zur Verwendung der Monte Carlo Simulation im Skat wird im Kapitel 3 beschrieben. Das Kapitel 4 ist der Turnierumgebung für Skatspiele gewidmet, die im Zuge dieser Arbeit entstanden ist. Das Kapitel 5 erläutert die Implementierung der neuen Monte Carlo Simulation mit unvollständiger Information. Die Ergebnisse der ersten Experimente und eine Untersuchung zur Optimierung der Monte Carlo Simulation werden ebenfalls behandelt. Im Kapitel 6 ist die Verfeinerung der Simulation durch Bewertungsfunktionen und deren Einfluss auf die Spielstärke beschrieben. Ein Spielstärkenvergleich mit dem Double Dummy Skat Solver aus [Kup03] rundet die Experimente im Kapitel 7 ab. Am Schluss liefert das Kapitel 8 eine Zusammenfassung der Arbeit und einen Ausblick auf zukünftige Arbeiten.

Kapitel 2

Grundlagen

Dieses Kapitel gibt eine kurze Einführung in die Spielregeln von Skat und stellt Vergleiche zum Kartenspiel Bridge her. Im zweiten Teil werden die bisherigen Arbeiten auf dem Gebiet der Monte Carlo Simulation im Bridge und Skat vorgestellt.

2.1 Einführung in das Skatspiel

Das Skatspiel hat sich seit dem 19. Jahrhundert aus verschiedenen alten Kartenspielen entwickelt. Der Name rührt vom italienischen Verb *scatare* (weglegen) her. Heute zählt es zu den beliebtesten Kartenspielen in Deutschland und wird auch weltweit von einer großen Spielergemeinde gespielt.

2.1.1 Spielregeln des Skatspieles

In diesem Abschnitt werden nur die für das Verständnis der Arbeit notwendigen Skatregeln erläutert. Alle weiterführenden Regeln können in der Internationalen Skatordnung (ISkO) [IW02] nachgelesen werden, die vom Deutschen Skatverband e.V. (DSkV) und der International Skat Players Association e.V. (ISPA-World) verabschiedet wurden.

Skat ist ein Kartenspiel für drei Spieler und wird mit 32 Karten gespielt. Das Kartenspiel ist in die vier Farben Kreuz (♣), Pik (♠), Herz (♡) und Karo (♢) unterteilt.[1] Jede Farbe hat die folgenden Karten: Sieben (7), Acht (8), Neun (9), Zehn (10), Bube (J), Dame (Q), König (K) und Ass (A).[2] Die Reihenfolge und Zählwerte der einzelnen Karten

[1] In dieser Arbeit werden nur die französischen Farben benutzt, da sie international gebräuchlicher sind als die deutschen Farben Eichel, Grün, Herz und Schell.

[2] Die in den Klammern angegebenen Abkürzungen orientieren sich an der früheren Literatur über Skat und Bridge und werden in dieser Arbeit ebenso verwendet. So steht zum Beispiel die Angabe ♣J für den Kreuz Buben.

sind in Tabelle 2.1 dargestellt. Der Zählwert einer Karte wird in Augen angegeben. Diese werden nach Beendigung eines Spieles zur Abrechnung benötigt. Alle Karten zusammen ergeben eine Gesamtaugenzahl von 120 Augen.

Karte	Zählwert
Bube	2 Augen
Ass	11 Augen
Zehn	10 Augen
König	4 Augen
Dame	3 Augen
Neun	0 Augen
Acht	0 Augen
Sieben	0 Augen

Tabelle 2.1: Zählwerte der verschiedenen Karten

Obwohl drei Spieler am Tisch sitzen, gibt es beim Skat nur zwei Spielerparteien. Der Alleinspieler versucht sein Spiel gegen die Gegenspieler zu gewinnen. Die Gegenspieler können sich zwar nicht in die Karten sehen, arbeiten aber trotzdem zusammen.

Geben

Beim Geben werden jeweils zehn Karten an die drei Spieler verteilt. Die zwei übrig gebliebenen Karten heißen Skat und werden vorerst zur Seite gelegt. Das Geben bei einem realen Skatspiel erfolgt nach festen Regeln. So werden zuerst drei Karten pro Spieler ausgeteilt, danach zwei Karten in den Skat gelegt und schließlich nacheinander jeweils vier und drei Karten an die Spieler verteilt. Beim Kartengeben im Rechner können die Karten zufällig den einzelnen Spielern zugeordnet werden, da hier das Mischen der Karten nicht durch einen der Spieler erfolgt. Der Spieler links neben dem Geber wird Vorhand genannt. Die weiteren Spieler heißen Mittelhand und Hinterhand.

Reizen

Das Reizen dient zur Bestimmung des Alleinspielers. Nach einer festen Abfolge fragt zunächst die Mittelhand bei der Vorhand Spielwerte von möglichen Spielen ab. Die Vorhand kann bei jeder Frage entscheiden, ob sie diesen oder einen höheren Spielwert bei einem Spiel erreichen kann und geht dementsprechend mit oder muss passen. Kann einer von beiden Spielern keinen höheren Wert mehr ansagen oder akzeptieren, kann die Hinterhand mit dem Gewinner aus der ersten Reizrunde weiterreizen. Der Sieger des Reizens ist der neue Alleinspieler. Die Vorhand für den ersten Stich bleibt trotzdem der

Spieler links neben dem Geber. Hat keiner der Spieler ein Gebot abgegeben, wird das Spiel eingegeben, die Karten neu gemischt und noch einmal gegeben.

Drücken

Der Alleinspieler hat nach dem Reizen die Möglichkeit, in den Skat zu sehen. Er darf bis zu zwei Karten aus seiner Hand gegen Karten aus dem Skat tauschen. Damit kann er mit guten Karten aus dem Skat sein Blatt verbessern oder schlechte Karten aus seiner Hand entfernen. Nach dem Drücken müssen wieder zwei Karten im Skat liegen. Hat der Alleinspieler nicht in den Skat gesehen, so ist das nächste Spiel ein Handspiel.

Grundregeln

Ein Spiel ist in zehn Stiche unterteilt. In jedem Stich dürfen die Spieler der Reihe nach eine Karte ausspielen. Der erste Ausspieler ist die Vorhand des Stiches. Danach spielen Mittel- und Hinterhand ihre Karten aus. Haben alle Spieler ein Karte gespielt, ist der Stich beendet. Der Stichgewinner erhält die Karten des Stiches und wird die Vorhand im nächsten Stich.

Beim Skat gibt es sogenannte Trumpfkarten. Dazu gehören die vier Buben und eine bei jedem Spiel neu festgelegte Farbe. Diese Karten stehen über allen anderen Karten, wobei die Buben die höchsten Trumpfkarten sind.

In jedem Stich herrscht Farb- bzw. Bedienzwang. Das bedeutet, dass die von der Vorhand ausgespielte Karte festlegt, welche Karten die anderen Spieler ausspielen dürfen. Wenn die Vorhand eine Farbe anspielt, so müssen Mittel- und Hinterhand immer die geforderte Farbe spielen, sofern sie diese auf der Hand halten. Das Gleiche gilt, wenn Trumpf angespielt wird. Kann ein Spieler die geforderte Farbe nicht bedienen, so kann er die Karte frei wählen. Wird eine andere Farbe als die geforderte gespielt und hat diese einen geringen Zählwert, so spricht man vom Abwerfen der Karte. Bei einer abgeworfenen Karte mit hohem Zählwert wird der Spielzug Buttern genannt. Wird statt der geforderten Farbe eine Trumpfkarte gespielt, so sticht der Spieler mit dieser Karte. Bedient ein Spieler falsch, so hat er Schwarz gespielt und das Spiel ist für seine Partei verloren.

Spielarten

Beim Skat gibt es drei verschiedene Spielarten: Farb-, Grand- und Nullspiel. Die ersten beiden unterscheiden sich nur durch die Anzahl der Trumpfkarten. Bei einem Farbspiel sind die Buben und alle Karten einer bestimmten Farbe Trumpfkarten. Das Grandspiel kennt nur die Buben als Trumpfkarten. Die Buben haben trotz ihrer Wertigkeit von nur zwei Augen eine herausragende Stellung. Sie stehen über allen Karten, da sie die

höchsten Trümpfe darstellen. Das Ziel des Alleinspielers beim Farb- und Grandspiel ist es, durch die Karten der gewonnenen Stiche mehr als 60 Augen zu erzielen. Sobald dieses Ziel erreicht ist, hat der Alleinspieler gewonnen.

Das Nullspiel steht im Gegensatz zu den Farb- und Grandspielen. Es gibt keine Trumpfkarten und die Buben reihen sich in die Farben ein. Des Weiteren hat auch die Zehn eine geringere Wertigkeit innerhalb der Farben. Die geänderte Reihenfolge der Karten einer Farbe beim Nullspiel ist in Tabelle 2.2 zu finden.

Karte	Wertigkeit
Ass	höchste Karte der Farbe
König	
Dame	
Bube	
Zehn	
Neun	
Acht	
Sieben	niedrigste Karte der Farbe

Tabelle 2.2: Wertigkeiten der Karten beim Nullspiel

Der Alleinspieler darf bei einem Nullspiel keinen Stich bekommen. Sobald er den ersten Stich erhält, ist das Spiel für ihn verloren. Dabei ist es egal, ob die Karten dieses Stiches Augenwerte haben oder nicht.

Alle drei Spielarten können mit oder ohne Skataufnahme gespielt werden. Hat der Alleinspieler nicht in den Skat gesehen, so zählt das Spiel als Handspiel. Beim Ouvertspiel deckt der Alleinspieler seine Karten vor dem Ausspielen der ersten Karte auf. Die Gegenspieler spielen weiterhin mit verdeckten Karten.

Spielbewertung

Bei den Farb- und Grandspielen ist für die Spielbewertung zunächst die Anzahl der Spitzen wichtig. Damit ist ausgehend vom ♣J die Zahl der Trumpfkarten gemeint, welche der Alleinspieler in ununterbrochener Reihenfolge inklusive der Karten im Skat besaß oder nicht besaß. Hatte der Alleinspieler den ♣J selbst auf der Hand, so spielt er mit Spitzen ansonsten ohne.

Bei besonderen Ereignissen wie Hand- oder Ouvertspielen werden zur Anzahl der Spitzen noch weitere Gewinnstufen addiert. Wenn der Alleinspieler 90 Augen oder mehr erreicht, haben die Gegenspieler Schneider gespielt. Schafft der Alleinspieler sogar 120 Augen, so haben die Gegenspieler Schwarz gespielt. In beiden Fällen wird jeweils eine weitere Gewinnstufen aufaddiert.

Die Anzahl der Spitzen und die Gewinnstufen werden am Schluss noch um eins erhöht und mit dem Wert für das gespielte Farb- oder Grandspiel aus Tabelle 2.3 multipliziert. Ein ♠-Spiel mit einer Spitze ist demnach 22 Punkte wert. Ein Grandspiel mit vier Spitzen und Schneider wird mit 144 Punkten bewertet.

Spieltyp	Wertigkeit
♢	9
♡	10
♠	11
♣	12
Grand	24

Tabelle 2.3: Wertigkeiten der Farb- und Grandspiele

Bei den Nullspielen gibt es nur vier verschiedene Spielwerte. Ein einfaches Nullspiel wird mit 23 Punkten bewertet, ein Null Hand ist 35 Punkte wert, ein Null Ouvert zählt 46 Punkte und das Null Hand Ouvert wird mit 59 Punkten belohnt.

Die Spielwerte werden auch beim Reizen verwendet. Jeder Spieler entscheidet anhand der Spitzen und der Spielart, welches Spiel er gewinnen könnte und nimmt den Spielwert als Obergrenze für sein Reizen.

Bei einem verlorenen Spiel wird der Spielwert mit -2 multipliziert. Hat der Alleinspieler außerdem noch Schneider gespielt, das heißt er selbst hat nur 30 Augen oder weniger erreicht, so wird der Spielwert im Negativen noch einmal verdoppelt. Das Gleiche erfolgt, wenn der Alleinspieler Schwarz gespielt hat.

2.1.2 Spieltheoretische Einordnung von Skat

Skat gehört zu den Zwei-Personen-Nullsummen-Spielen mit unvollständiger Information. Obwohl drei Spieler am Spiel beteiligt sind, gibt es nur zwei Spielerparteien, den Alleinspieler und die Gegenspieler. Skat ist ein Nullsummenspiel, da der Gewinn einer Partei einen Verlust der anderen Partei bedeutet. So kann ein verlorenes Spiel sowohl mit negativen Punkten beim Alleinspieler als auch mit Pluspunkten bei den Gegenspielern angeschrieben werden. Die Spieler kennen zu Beginn des Spieles nur ihre eigenen Karten. Man spricht deshalb von einem Spiel mit unvollständiger Information, obwohl im Spielverlauf die Karten nach und nach aufgedeckt werden und am Schluss alle Spieler alle Karten kennen. Durch das Beobachten der Spielweise der Mitspieler und dem Wissen über den Farb- und Bedienzwang können einzelne Spieler schon vor dem letzten Stich die vollständige Information über die Kartenverteilung erlangen. Ab diesem Zeitpunkt ist das Spiel für sie ein Spiel mit vollständiger Information. Die beiden anderen Spieler haben aber nicht automatisch zum selben Zeitpunkt dieses vollständige Wissen.

2.2 Vergleich mit dem Kartenspiel Bridge

Bridge ist ein Kartenspiel für vier Spieler und 52 Karten. Es gibt verschiedene Formen beim Bridge. Die in der Literatur am meisten untersuchte Spielart ist das sogenannte Contract Bridge. Hier kommt es nicht auf die Anzahl der erreichten Augen an, sondern auf die Anzahl der gemachten Stiche. Auch beim Bridge gibt es zwei Spielerparteien. Die vier Spieler sind nach den Himmelsrichtungen Nord, West, Süd und Ost benannt, wobei immer Nord-Süd gegen Ost-West spielen. Wie beim Skat wird zu Beginn eines Spieles zunächst gereizt. Hier geht es aber darum, wieviele Stiche eine Partei machen kann. Der Gewinner des Reizens spielt mit allen Karten seiner Partei. Das heißt, sein Mitspieler muss seine Karten offen hinlegen und ist für den Rest des Spieles nur Beobachter.[3] Die Gegenpartei und der Alleinspieler spielen weiterhin mit verdeckten Karten. Beim Bridge kann es auch eine Trumpffarbe geben. Es herrscht wie beim Skat der Farb- bzw. Bedienzwang. Nach dem Spiel werden die gemachten Stiche gezählt und mit dem Reizgebot verglichen. Hat der Alleinspieler sein Gebot erreicht oder sogar übertroffen, so ist das Spiel gewonnen. Beim Contract Bridge gibt es nur einen Spieltyp, welcher am ehesten dem Farbspiel beim Skatspiel ähnelt. Für Grand- und Nullspiele gibt es keine entsprechenden Spielarten bei Bridge.

Trotz der Unterschiede zwischen Skat und Bridge lassen sich doch einige Gemeinsamkeiten der beiden Kartenspiele erkennen. Die größte Gemeinsamkeit ist das Vorhandensein von unvollständiger Information. Keiner der Spieler kennt alle Karten. Schon während des Spielens versuchen die einzelnen Spieler jedoch, die unvollständige Information zu beseitigen, indem sie Rückschlüsse auf die Kartenverteilung anhand der ausgespielten Karten ziehen. Nach dem Reizen, das auch beim Bridge dem Ermitteln des Alleinspielers dient, spielen zwei Spielerparteien gegeneinander. Bei jedem Stich wird ein Gewinner ermittelt und dieser wird der nächste Ausspieler. Am Ende werden alle gemachten Stiche oder Augen gezählt und der Spielgewinner ermittelt.

2.3 Monte Carlo Simulation im Bridge

In der Literatur lassen sich mehrere Arbeiten zur Verwendung von Monte Carlo Simulationen bei Bridge finden. Die bekanntesten sind sicherlich die Arbeiten von Ginsberg und Frank. Beide versuchen, das Spiel mit unvollständiger Information auf Welten mit vollständiger Information abzubilden. Beide erzielen mit diesem Ansatz gute Erfolge in der Spielstärke der Computer-Bridge-Spieler.

[3] Er wird auch als Dummy bezeichnet.

2.3.1 GIB: Goren In a Box

Zwischen 1994 und 1997 wurde das Bridge-Programm GIB[4] von Ginsberg entwickelt und unter anderem in [Gin99] ausführlich beschrieben. Es nutzt die Monte Carlo Simulation, um die unvollständige Information mit Werten zu füllen, welche dem bisherigen Spielverlauf entsprechen. In der so enstandenen Welt wird ein Suchbaum für die zukünftigen Spielzüge aufgebaut und mit einer Alpha-Beta-Suche die optimale Strategie für diese Welt gesucht. Dabei werden mehrere Optimierungen angewandt, um die Anzahl der zu untersuchenden Knoten im Suchbaum zu verringern. Durch mehrmaliges Simulieren und Mitteln der errechneten Strategien über alle Welten kann ein guter Spielzug für die meisten Welten gefunden werden.

GIB gelang es, mit 100 simulierten Welten in einige Situationen auf dem Niveau eines menschlichen Experten zu spielen. Dabei wurde darauf geachtet, dass die erzeugten Welten tatsächlich nach dem bisherigen Spielverlauf möglich sind. Dies wurde durch ein Sieben einer Menge von zufällig erzeugten Kartenverteilungen erreicht, wobei nicht passende Verteilungen über mehrere Prüfregeln aus der Menge der möglichen Kartenverteilungen entfernt wurden.

Das größte Hindernis für die Beurteilung der Spielstärke von GIB im Vergleich zu anderen Brigde-Programmen sah Ginsberg damals im Fehlen einer einheitlichen Schnittstelle für das Durchführen von Spielen. So mussten alle Spiele gegen kommerzielle Programme per Hand durchgeführt werden. Dadurch ist die Zahl der untersuchten Spiele natürlich gering, was zu Problemen bei der statistischen Auswertung führt. Des Weiteren gibt es beim Bridge keine Bewertungsmaßstäbe für die Spielstärke eines Spielers oder Programmes. Somit sind Vergleiche der Spielstärke bei Bridge schwieriger als sie zum Beispiel beim Schach möglich sind.

2.3.2 Kritik an GIB

In [FB98] wurde der Ansatz von GIB bereits eingehend untersucht. Ein Kritikpunkt an der Monte Carlo Simulation ist, dass durch das Überführen der unvollständigen Information auf Welten mit vollständiger Information es zur Vermischung von Strategien kommen kann. In jeder simulierten Welt herrscht vollständiges Wissen aller Spieler über den Spielstand. Die angewendete Alpha-Beta-Suche nimmt dies ebenfalls implizit an und betrachtet die Gegenspieler als optimal spielende Spieler. Durch das Mischen aller Strategien aus den einzelnen Simulationen wird unter Umständen nicht immer die optimale Strategie für alle Spiele ausgewählt.

Frank stellte in [FBM98] die Algorithmen Vector Minimaxing und Payoff-Reduction Minimaxing vor, welche der Monte Carlo Simulation überlegen sind. Dies wurde an der

[4]Eine andere Deutung des Namens GIB ist Ginsberg's Intelligent Bridgeplayer.

Anzahl gelöster Probleme aus der Official Encyclopedia of Bridge nachgewiesen, wobei das Payoff-Reduction Minimaxing eine deutliche Verbesserung darstellte.

Ein weiterer Kritikpunkt an der Vorgehensweise von GIB ist in [RN04] dargestellt. Da in den erzeugten Welten vollständige Information herrscht, werden nie Spielzüge ausgewählt, die zum Informationsgewinn dienen. Bei Bridge und Skat sind aber solche Spielzüge durchaus gebräuchlich und gehören zu den Strategien guter Spieler.

Trotz dieser Kritikpunkte schaffte es GIB auf hohem Niveau Bridge zu spielen und wird heute als eines der stärksten Computer-Bridge-Programme bezeichnet. Bis heute gibt es in der Literatur keine weiteren Ansätze, als mittels Monte Carlo Simulation die unvollständige Information im Bridge zu ersetzen und in den erzeugten Welten eine Strategie für den nächsten Zug zu berechnen. In den letzten Jahren wurde lediglich untersucht, wie die Suchbäume effektiv beschnitten werden können, um somit eine Optimierung der Berechnungszeit und der Spielstärke zu erreichen.

2.4 Monte Carlo Simulation im Skat

Auch für das Skatspiel wurde schon einmal der Einsatz der Monte Carlo Simulation untersucht. In der Diplomarbeit von Kupferschmid wurden die Ergebnisse und Methoden der Arbeiten von Ginsberg über Bridge auf das Skatspiel angewandt und erweitert.[Kup03]

Der in der Diplomarbeit entstandene Double Dummy Skat Solver (DDSS) ermittelt seine Spielstrategie durch die vollständige Berechnung von Spielbäumen für Skatspiele mit offenen Karten. Dies gelingt durch vielfältige Optimierungen in sehr kurzer Zeit.[5] Der DDSS ermittelt ebenso wie GIB die unbekannten Karten eines Spieles mittels Monte Carlo Simulation. Danach wird das Spiel als offenes Skatspiel betrachtet. Nach einer festen Anzahl von Simulationen wird die Karte ausgewählt, welche in den meisten Spielen die besten Werte erzielt hat.

[5] Eine komplette Berechnung eines Spielbaumes gelang mit der damals verwendeten Hardware innerhalb von 0,06 Sekunden.

Kapitel 3

Monte Carlo Simulation mit unvollständiger Information

In diesem Kapitel soll kurz der neue Ansatz für die Nutzung der Monte Carlo Simulation bei Spielen mit unvollständiger Information dargestellt werden.

3.1 Bisherige Ansätze

Laut [IW02] gibt es beim Skat mehr als $2{,}7 \cdot 10^{15}$ mögliche Kartenverteilungen. Diese Zahl reduziert sich während des Spielens, da jeder Spieler nach dem Geben seine Karten kennt und bei jedem Stich sich das Wissen über die tatsächliche Kartenverteilung erweitert. Es wird aber kaum machbar sein, alle möglichen Kartenverteilungen durchzurechnen. Bei den bisher vorgestellten Arbeiten über Monte Carlo Simulation wird versucht, durch die Untersuchung von zufällig erzeugten Stichproben aus der Menge aller möglichen Welten eine Abschätzung für die beste Strategie zu finden. Die Welten beim Skatspiel sind die möglichen Kartenverteilungen. Alle Verteilungen haben die gleiche Wahrscheinlichkeit, da die Karten zunächst in einer zufälligen Reihenfolge gemischt werden und dann unabhängig voneinander auf die Spieler verteilt werden. Nun muss in jeder Welt untersucht werden, welche Spielkarte die erfolgversprechendste ist. Da die simulierte Welt ein Spiel mit vollständiger Information darstellt, können alle Spielzustände in einem Suchbaum abgelegt werden. Ein vollständiger Suchbaum für Skat kann nach [Kup03] mehr als 300 Millionen Knoten beinhalten. Durch Beschneiden erreicht man eine Verkleinerung auf durchschnittlich 150.000 Knoten.

Wenn Spiele mit unvollständiger Information auf Spiele mit vollständiger Information abgebildet werden, können sich die ermittelten Strategien der Einzelwelten überlagern. Deshalb wird nicht immer die optimale Strategie gefunden. Der neue Ansatz versucht, dieses Manko zu beheben und behält deshalb die unvollständige Information bei.

3.2 Grundidee des neuen Ansatzes

Die Monte Carlo Simulation ist nicht die einzige Möglichkeit, Computerprogrammen das Spielen beizubringen. Es gibt auch noch den Ansatz, Heuristiken für das Spielen zu implementieren. Das sind feste Regeln, welche in bestimmten Spielsituationen angewendet werden. Für Skat könnten sie wie folgt aussehen:

- Wenn du Alleinspieler bist und das Spiel ein Grandspiel ist und alle Buben auf deiner Hand liegen, dann spiele den niedrigsten Buben.
- Wenn das Spiel ein Nullspiel ist dann spiele immer die kleinste Karte einer Farbe.

Für einfache Spiele mit wenigen Regeln oder wenigen Spielzuständen ist dieser Ansatz durchaus sinnvoll. Wenn das Regelwerk komplexer wird oder die Anzahl der möglichen Welten wächst, müssen immer mehr Heuristiken implementiert werden, um alle Situationen abzudecken. Ab einem gewissen Punkt ist es nicht mehr möglich, für alle Situationen die optimalen Strategien festzulegen. Solche Programme können auch nie ermitteln, ob es eine bessere Strategie gibt, da ein Programm mit Heuristiken davon ausgeht, dass es perfekt spielt. Bei den Spielen mit unvollständiger Information kommt noch dazu, dass die Regeln nicht auf das gesamte Wissen zugreifen können. Hier müssen die Heuristiken noch allgemeiner gehalten werden, um alle möglichen Zustände abdecken zu können. Deshalb sind diese Programme denen mit Monte Carlo Simulation unterlegen.

Die Grundidee des neuen Ansatzes für Spiele mit unvollständiger Information besteht in der Kombination der oben beschriebenen Methoden. Es kommen sowohl Monte Carlo Simulation als auch Heuristiken für das Skatspiel zum Einsatz. Ist ein Spieler am Zug, werden mittels Monte Carlo Simulation mögliche Welten erzeugt, die zum bisherigen Spielverlauf passen. In jeder dieser Welten werden die spielbaren Karten in einem jeweils eigenen Skatspiel ausgespielt und dann das Spiel mit Heuristiken zu Ende geführt. Am Schluss ist nur das Spielergebniss bekannt. Es wird weiterhin verdeckt, welcher Spieler welche Karten gespielt hat. Die Eigenschaft der unvollständigen Information bleibt bestehen. Das Aufbauen und Beschneiden eines Suchbaumes entfällt ebenfalls, da nur die Endergebnisse aller Spiele untersucht werden. Anhand der erreichten Spielergebnisse kann nun die beste Karte ermittelt und ausgespielt werden.

3.3 Verwendete Heuristiken

In dieser Arbeit wurden die Heuristiken von XSkat 4.0 verwendet. XSkat ist ein bekanntes Skatspiel aus der Unix-Welt und wird heute bei den meisten Linuxdistributionen mitgeliefert. Der Quellcode liegt offen und darf in anderen Projekten weiterverwendet werden.

Über die Spielstärke von XSkat ist wenig bekannt. Eine Untersuchung, welche der Autor von XSkat in einer Skatrunde mit rund 850 Spielen gegen zwei menschliche Spieler durchgeführt hat, ergab, dass XSkat gegen Menschen nicht bestehen konnte. Bei dieser Betrachtung fließt aber auch die Stärke des Programmes beim Reizen mit ein. Reizt ein Spieler zurückhaltend, macht er weniger Spiele und bekommt deshalb auch weniger Punkte. XSkat reizt oft zurückhaltend, wenn die Spiele nicht eindeutig gewinnbar sind. In der Arbeit wurden aber nur die Heuristiken für das Spielen verwendet, so dass das schwache Reizen keinen Einfluss auf die Spielstärke hat.

Kapitel 4

Turnierumgebung

Im folgenden Kapitel wird die Turnierumgebung beschrieben, welche im Rahmen dieser Arbeit entwickelt wurde. Die Einbindung der einzelnen Spielertypen in die Turnierumgebung wird ebenfalls dargestellt.

4.1 Anforderungen an die Turnierumgebung

Damit man die Spielstärken verschiedener Skatspieler vergleichen kann, ist es notwendig, diese Spieler unter kontrollierten Bedingungen gegeneinander antreten zu lassen. Die Turnierumgebung sollte folgende Funktionalitäten besitzen:

- Feste Schnittstelle für Skatspielprogramme
- Freie Kombination der Spieler in einer Skatrunde
- Verwendung mehrerer Instanzen eines Spielertypes in einer Skatrunde
- Wiederholung von Spielen mit anderen Spielerkombinationen
- Gezieltes Spielen von vorgegebenen Kartenverteilungen oder seltenen Spielen
- Bereitstellen von Daten zur statistischen Auswertung der Spielstärken der einzelnen Spielertypen

4.1.1 Spielmöglichkeiten zum Spielstärkenvergleich

Das Spielen von normalen Skatrunden nach den Regeln der Internationalen Skatordnung sollen ebenso möglich sein wie spezielle Skatrunden. Letztere sollen Daten für den Vergleich der Spielstärke der einzelnen Spieler erzeugen.

Bei einer normalen Skatrunde wechseln nach jedem Spiel die Sitzpositionen. Die Mittelhand des alten Spieles wird die neue Vorhand, die Hinterhand wird Mittelhand und die Vorhand ist die neue Hinterhand. Es werden im Allgemeinen rund 50 bis 100 Spiele gespielt. Bei dieser Spielweise hängt der Erfolg eines Spielers auch davon ab, welche Karten er durch das Geben erhält. Wenn er oft schlechte Karten bekommt, kann er nicht so hoch reizen und damit keine Spiele machen, was am Ende zu weniger Punkten bei der Abrechnung führt. Zudem ist die geringe Anzahl von Spielen für statistische Auswertungen nicht geeignet.

Es gibt mehrere Möglichkeiten, diese Fehlerquellen auszuschalten. Zum Ersten können sehr viele Spiele durchgeführt werden. Bei den Untersuchungen im Rahmen dieser Arbeit wurden meist 1.000 Spiele durchgeführt. Damit wäre das Problem der geringen Anzahl von Spielen gelöst. Zum Zweiten muss es möglich sein, die einzelnen Spiele in der Turnierumgebung mit einer anderen Verteilung der Sitzpositionen noch einmal zu spielen. In einem Skatspiel hängen die gegebenen Karten fest mit den Sitzpositionen Vor-, Mittel- und Hinterhand zusammen. Jeder Spielertyp sollte somit die Chance bekommen, jede dieser Positionen gegen die anderen Spielertypen zu spielen. Je nach Anzahl der verschiedenen Spielertypen müssen deshalb die gleichen Kartenverteilungen ein-, drei- oder sechsmal gespielt werden. Diese Zahlen ergeben sich durch die möglichen Sitzverteilungen. Spielt nur ein Spielertyp auf allen drei Sitzpositionen, so muss das Spiel nicht noch einmal wiederholt werden. Bei zwei Spielertypen pro Runde werden zwei Sitzpositionen immer durch den selben Spielertyp besetzt. Diese Spiele müssen also nicht mit getauschten Positionen des doppelt vorhandenen Spielers noch einmal gespielt werden. Spielen drei verschiedene Spielertypen miteinander in einer Skatrunde, so müssen sechs Spiele mit der selben Kartenverteilung durchgeführt werden, um allen Spielern die Möglichkeit zu geben, in jeder Konstellation die gegebenen Karten zu spielen. Mit diesem Ansatz spielt die Kartenverteilung keine Rolle mehr.

4.1.2 Abrechnungsarten zum Spielstärkenvergleich

Bei einer Skatrunde wird nach jedem Spiel der Spielwert nach den Skatregeln ermittelt und für den Alleinspieler notiert. Nach dem Durchführen einer vorher festgelegten Anzahl von Spielen gewinnt der Spieler mit der höchsten Punktzahl die Skatrunde.

Bei Skatturnieren werden nur kurze Skatrunden mit 18 bis 48 Spielen durchgeführt. Bei dieser geringen Anzahl an Spielen, ist die normale Leistungsbewertung ungerecht, da ein Spieler mit einem sehr hohem Grandspiel einen unaufholbaren Vorsprung erringen würde. Um dem entgegen zu wirken, wird das Seeger-Fabian-System zur Leistungsbewertung eingesetzt. Der Alleinspieler erhält für jedes gewonnene Spiel noch zusätzlich 50 Punkte zum eigentlichen Spielwert gutgeschrieben. Verliert er das Spiel, so werden zusätzlich noch 50 Minuspunkte angeschrieben. Die beiden Gegenspieler erhalten im Falle

eines verlorenen Spieles für den Alleinspieler außerdem noch 40 Punkte gutgeschrieben. Somit wird auch das gute Spielen in der Gegenpartei belohnt, wenn es zum Verlust eines Spieles beim Alleinspieler führt.

4.2 Implementierung der Turnierumgebung

Die Turnierumgebung wurde in C++ mit dem objektorientierten Programmieransatz implementiert. Die Nutzung eines objektorientierten Designs hat mehrere Vorteile. Zunächst kann die Turnierumgebung alle Skatspieler über eine feste Schnittstelle ansprechen. Neue Skatspieler können leicht in die Turnierumgebung eingebunden werden, weil dafür nur diese Schnittstelle implementiert werden muss. Der Zugriff auf die Daten der Skatspieler erfolgt über Methoden. Somit kann der Skatspieler intern andere Kodierungen als die Turnierumgebung für die Spielzustände verwenden. Beim Abruf der Daten muss er diese in die Kodierungen der Turnierumgebung umwandeln. Die Turnierumgebung übergibt ihrer Daten ebenfalls durch den Aufruf von Methoden der Skatspieler. Das ermöglicht einerseits das Prüfen der übergebenen Daten auf Plausibilität und andererseits kann der Skatspieler auf die Veränderung der Daten mit einer Anpassung seiner internen Datenstrukturen reagieren.

Ein weiterer Grund für die Wahl von C++ als Programmiersprache war das Weiterverwenden der Heuristiken des Skatprogramms XSkat, welches in C implementiert ist. Ein Wechsel der Sprachfamilie hätte einen unvertretbaren Aufwand bedeutet.

Die Klassen der Turnierumgebung sind in zwei Gruppen aufgeteilt. Auf der einen Seite stehen die Klassen für die Skatspieler, auf der anderen Seite die Kontroll- und Hilfsklassen. Die Kontrollklassen führen die Spiele durch und steuern die Skatspieler. Die Methoden der Hilfsklassen werden von allen anderen Klassen für oft benutzte Funktionen verwendet. Das Klassendiagramm in Abbildung 4.1 zeigt die Abhängigkeiten der Klassen untereinander.

4.2.1 Repräsentation von Skatspielen, Skatrunden und Karten

Ein Skatspiel wird durch Objekte der Klasse `SkatGame` repräsentiert. Sie speichern die Daten der einzelnen Spiele wie Spielertypen, Spielart, die gemachten Stiche und das Spielergebnis. Die Kodierungen der einzelnen Attribute wurden größtenteils von XSkat übernommen. Ein Skatspiel-Objekt steuert den gesamten Ablauf eines Skatspieles vom Reizen, Drücken, Spielansage und Ausspielen bis zum Ermitteln des Gewinners und des Spielwertes. Beim Reizen, Drücken und der Spielansage werden die Heuristiken von XSkat verwendet. Das ist dadurch begründet, dass noch nicht alle KI-Spieler diese Funktionalität besitzen. Es ist aber geplant, diese Spielabschnitte in einer späteren Version

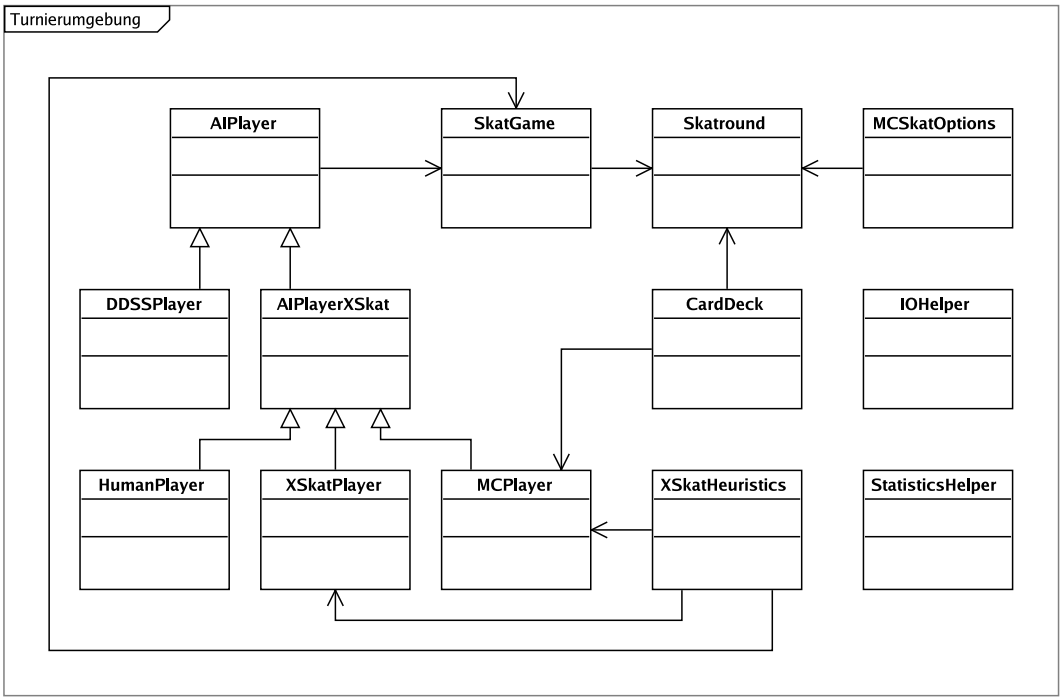

Abbildung 4.1: Klassendiagramm der Turnierumgebung

der Turnierumgebung von den KI-Spielern durchführen zu lassen, die das Reizen und Drücken beherrschen. Die Spielergebnisse wie Augenzahl, Spielwert und Gewinn oder Verlust können von außen über Methoden vom Skatspiel-Objekt abgefragt werden.

Die Klasse `SkatRound` fasst mehrere Skatspiele zusammen. Sie ist die Steuerklasse für alle Spielrunden. Die Skatrunde erzeugt vor dem Start eines Spieles eine neue Kartenverteilung oder liest eine gespeicherte Verteilung aus einer Textdatei. Danach werden die Instanzen der Spieler-Objekte erzeugt. Diese vier Objekte werden einem Skatspiel-Objekt übergeben und das Spiel gestartet. Unter Verwendung der XSkat-Heuristiken wird zunächst für alle Spieler gereizt, gedrückt und ein Spiel angesagt. Das Skatspiel-Objekt fragt nun der Reihe nach die einzelnen Spieler nach den Karten, die sie ausspielen möchten. Dabei wird geprüft, ob die zu spielende Karte überhaupt auf der Hand des Spieler liegen kann. Nach jedem Stich ermittelt das Skatspiel-Objekt den Stichgewinner und informiert alle Skatspieler darüber. Am Ende des Spieles werden die Ergebnisse durch das Skatrunden-Objekt abgefragt und dort gespeichert. Nach dem Ende aller Spiele werden Datendateien für die statistische Auswertung erzeugt und die gespielten Kartenverteilungen zur späteren Wiederverwendung gespeichert. Dabei erzeugt die Skatrunde mehrere Textdateien. Zunächst werden alle Kartenverteilungen in der Datei `all_games.txt` abgelegt. Für jede Spielart gibt es außerdem eine eigene Datei (`suit_games.txt`, `grand_games.txt` und `null_games.txt`). Die Kartenver-

♢A	♢10	♢K	♢Q	♢J	♢9	♢8	♢7	♡A	...	♡7	...	♣7
0	1	2	3	4	5	6	7	8	...	15	...	31

Abbildung 4.2: Kodierung der Karten in der Turnierumgebung

teilung von Farb- und Grandspielen, welche knapp verloren wurden,[1] landen in der Datei `close_games.txt`. Alle verlorenen Spiele speichert die Skatrunde in der Datei `lost_games.txt`. Die einzelnen Dateien können bei späteren Experimenten wiederverwendet werden, um die gleichen Kartenverteilungen mit anderen Spielerkombinationen oder mit mehr Simulationen pro Spielzug durchzuführen. Auch das gezielte Nachspielen von knapp verlorenen Spielen ist damit möglich.

Die Karten werden durch Integer-Werte dargestellt. Die Kodierung erfolgt wie bei XSkat. Alle Karten einer Farbe sind absteigend vom Ass bis zur Sieben sortiert und die Farben sind in der Reihenfolge ♢, ♡, ♠ und ♣ geordnet. Die Kodierung der Kartenwerte ist in Abbildung 4.2 aufgelistet. Die Farbe einer Karte lässt sich durch eine Integer-Division mit 8 ermitteln. Das Bild einer Karte erhält man durch eine Modulo-Operation mit der Basis 8.

Die Karten eines Kartenspieles werden in einem Objekt der Klasse `CardDeck` gespeichert. Die Zuordnung der Karten zu den einzelnen Spielern oder zum Skat ist ebenfalls in dem Kartenspiel-Objekt festgehalten. Unbekannte Kartenpositionen haben die Kodierung -1. Die Klasse `CardDeck` ermöglicht aber nicht nur das Speichern der Karten, sondern stellt auch Methoden zur Generierung von Kartenverteilungen bereit. Eine genauere Beschreibung dazu erfolgt im nächsten Kapitel.

4.2.2 Schnittstelle für KI-Spieler

Alle KI-Spieler sind von der Klasse `AIPlayer` abgeleitet. Diese Klasse definiert die Methoden eines KI-Spielers, welche von der Turnierumgebung genutzt werden, um die Spiele durchzuführen. Gegebenenfalls muss der abgeleitete KI-Spieler die Methoden auch überschreiben, wenn er intern eine andere Repräsentation der Karten und Spielstände besitzt. Die Methoden, welche durch die Klasse `AIPlayer` definiert werden, sind in Tabelle 4.1 aufgeführt.

Die Methode `makeNextMove()` ist in der Klasse `AIPlayer` nur definiert, aber nicht implementiert. Sie muss von jedem KI-Spieler selbst umgesetzt werden. In dieser Methode berechnet der KI-Spieler seine Strategie für das Ausspielen der nächsten Karten und liefert diese Karte als Rückgabewert.

[1] Das sind Spiele mit 55 bis 60 Punkten für den Alleinspieler.

Methode	Beschreibung
`newGame()`	Setzt den KI-Spieler in den Ausgangszustand zurück.
`takeCards()`	Dient zum Übergeben der Karten an den KI-Spieler.
`startGame()`	Informiert den KI-Spieler über seine Sitzposition, den Spieltyp, den Alleinspieler und die erste Vorhand.
`nextMove()`	Informiert den KI-Spieler über eine gespielte Karte der Gegenspieler.
`trickCompleted()`	Informiert den KI-Spieler am Ende eines Stiches über den Stichgewinner.
`makeNextMove()`	Fordert den KI-Spieler auf, eine Karte zu spielen.

Tabelle 4.1: Methoden eines KI-Spielers

Das aktuelle Wissen eines Skatspielers über den Spielstand ist ebenfalls im KI-Spieler-Objekt gekapselt. Somit weiß jeder Spieler welches Spiel gerade gespielt wird, in welchem Stich sich das Spiel gerade befindet oder wie die anderen Spieler gespielt haben.

4.2.3 Einbindung von XSkat

Zunächst wurden die Quellen der Version 4.0 [Ger04] von allen Bestandteilen bereinigt, die mit der graphischen Oberfläche von XSkat oder der Kommunikation per Internet Relay Chat (IRC) zu tun haben. Übrig geblieben sind die Heuristiken, die XSkat zum Spielen benutzt. Diese wurden in der Klasse `XSkatHeuristics` gekapselt. Mit Objekten dieser Klasse können jetzt komplette Skatspiele gespielt werden. Viel interessanter ist aber die Verwendung der Klasse für einzelne Spielzüge wie zum Beispiel Reizen, Drücken und das Ermitteln der spielbaren Karten. Dazu muss nur der aktuelle Spielzustand in einem Objekt der Klasse `XSkatHeuristics` hergestellt und die entsprechenden Methoden für die nächsten Spielzüge aufgerufen werden.

4.2.4 Schnittstelle für XSkat-basierte KI-Spieler

Einige in der Turnierumgebung implementierte KI-Spieler verwenden eine Instanz der Klasse `XSkatHeuristics` für die Berechnung ihrer Spielstrategien. Dazu ist es notwendig, dass der aktuelle Spielstand immer auch in dem XSkat-Objekt bekannt ist. Deshalb erweitert die Klasse `AIPlayerXSkat` die Klasse `AIPlayer` um einige Methoden und überschreibt die vorhandenen Methoden, damit bei jeder Änderung im Spiel zusätzlich die Daten im XSkat-Objekt automatisch angepasst werden.

4.2.5 KI-Spieler

In der Turnierumgebung sind derzeit vier verschiedene Spieler implementiert:

- Der `XSkatPlayer` spielt mit den Heuristiken von XSkat. Er wird für die späteren Spielstärkenvergleiche als Referenzspieler eingesetzt.
- In der Klasse `MCPlayer` ist der Monte Carlo Spieler mit dem in dieser Arbeit entwickelten Ansatz der Monte Carlo Simulation mit unvollständiger Information implementiert. Im weiteren Verlauf wird dieser Spieler MCU-Spieler genannt. Während der Simulationen nutzt er mehrere Instanzen der XSkat-Heuristiken.
- Der `DDSSPlayer` ist die Einbindung des Double Dummy Skat Solvers aus [Kup03]. Als einziger KI-Spieler nutzt der `DDSSPlayer` nicht die Heuristiken von XSkat und ist somit direkt von der Klasse `AIPlayer` abgeleitet.
- Der `HumanPlayer` ermöglicht es auch menschlichen Spielern, gegen die KI-Spieler anzutreten. Er ist kein KI-Spieler im eigentlichen Sinne, implementiert aber die Schnittstelle für die KI-Spieler. Die gespielten Karten werden von der Turnierumgebung während des Spieles abgefragt. Der `HumanPlayer` nutzt die XSkat-Heuristiken zur Ermittlung der spielbaren Karten. Das ermöglicht eine Prüfung der eingegebenen Karten.

4.2.6 Hilfsklassen

Die Klassen `MCSkatOptions`, `StatisticsHelper` und `IOHelper` sind Hilfsklassen für Funktionalitäten, die von anderen Klassen benötigt werden.

In `MCSkatOptions` sind alle Optionen gespeichert, die durch Kommandozeilenparameter beim Programmstart an die Turnierumgebung übergeben werden. Ein Options-Objekt wird beim Start einer Skatrunde übergeben und ausgewertet. Es enthält die Vorgaben für die Spielertypen, die Anzahl der zu spielenden Spiele oder den Dateinamen für eine Textdatei mit vorgegebenen Kartenverteilungen sowie die Art der Spieldurchführung. Damit ist es leicht möglich, viele Optionen zu übergeben oder neue Optionen einzuführen, ohne die Signaturen der Methoden der Skatrunden-Klasse ändern zu müssen.

Die Klasse `StatisticsHelper` berechnet statistische Maßzahlen wie zum Beispiel den Median für Zahlenwerte in einem Feld. Diese Maßzahlen werden bei den Simulationen für die Auswahl der besten Karte benutzt.

Der `IOHelper` stellt mehrere Methoden zur Ein- und Ausgabe von Kartenwerten zur Verfügung. Somit ist es möglich, die einzelnen Karten in menschenlesbarer Form

einzugeben und dann in die Kodierung der Turnierumgebung umzuwandeln oder die internen Kodierungen für Karten auf der Kommandozeile menschenlesbar auszugeben.

4.3 Ausgabedaten

Für eine spätere statistische Auswertung werden die Spielergebnisse nach dem Ende aller Spiele in eine Datendatei geschrieben. Der Dateiname wird aus der Anzahl der gemachten Spiele und den Signaturen[2] der einzelnen Skatspieler gebildet. In der Datendatei sind der Spieltyp und für jeden Spieler die erreichten Augen, der Spielausgang (gewonnen/verloren), der Spielwert und der Spielwert nach Turnierrechnung enthalten.

[2]Eine Signatur enthält den Spielertyp und Kodierungen für die Eigenschaften eines Spielers.

Kapitel 5

Implementierung der Monte Carlo Simulation

Dieses Kapitel beschreibt den MCU-Spieler, welcher den neuen, in Kapitel 3 beschriebenen Ansatz für Monte Carlo Simulation mit unvollständiger Information bei Skat implementiert. Die Algorithmen zur Erzeugung von möglichen Kartenverteilungen werden ebenfalls in diesem Kapitel behandelt.

5.1 Monte Carlo Spieler mit unvollständiger Information

Wenn der MCU-Spieler eine Karte ausspielen soll, ermittelt er zunächst alle spielbaren Karten. Wegen des Farb- und Bedienzwanges beim Skat kann unter Umständen nur ein Teil der Karten, welche ein Spieler auf der Hand hält, ausspielbar sein. Ist nur eine Karte erlaubt, so wird diese vom MCU-Spieler sofort ausgespielt. Sind mehrere Karten möglich, so ermittelt er seine Spielzüge mittels Monte Carlo Simulation mit unvollständiger Information nach einem Algorithmus, der in Abbildung 5.1 im Pseudocode angegeben ist.

Der MCU-Spieler lässt die Klasse `CardDeck` die Kartenverteilungen erzeugen. Danach instantiiert er für jede spielbare Karte ein Objekt der Klasse `XSkatHeuristics`, stellt den aktuellen Spielstand in dem XSkat-Objekt her und spielt die Karte aus. Nun wird das XSkat-Objekt aufgefordert, das Spiel zu Ende zu spielen. Zum Beenden eines Spieles werden also die Heuristiken von XSkat verwendet. Der MCU-Spieler fragt den Spielstand von dem XSkat-Objekt ab und speichert ihn. Diese fünf Schritte werden solange wiederholt, bis die vorgegebene Anzahl der Simulationen erreicht oder die festgelegte, maximale Bedenkzeit verstrichen ist. Nach Beendigung der Simulationen werden die Spielausgänge ausgewertet. Die Karte mit der besten Bewertung wird am Ende aus-

```
calculateBestCard(knownCards, playerCards) {

    cardDistribution = knownCards.simulateUnknownCards();
    playableCards = playerCards.getPlayableCards();

    if (playableCards.count() == 1) {

        bestCard = playableCards.getFirst();
    }
    else {

        forall (card in playableCards) {

            xskat = new XSkatHeuristics();
            xskat.setCards(cardDistribution);
            xskat.setGameStatus();
            xskat.playCard(card);
            xskat.finishSkatGame();

            results.add(xskat.getGameResults());
        }

        bestCard = playableCards.get(results.getBestCard());
    }

    return bestCard;
}
```

Abbildung 5.1: Grundalgorithmus zur Ermittlung der besten Karte

gespielt. Zur Bewertung kann zum Beispiel die Anzahl der gewonnenen und verlorenen Spiele genutzt werden. Die erreichte Augenzahl oder eine spezielle Bewertungsfunktion für den Spielausgang sind ebenfalls denkbare Ansätze. In diesem Kapitel wird zunächst nur die mittlere erreichte Augenzahl zur Bewertung der Karten verwendet.

Für Nullspiele musste ein anderer Ansatz zur Bewertung der Simulationsergebnisse gefunden werden, da bei dieser Spielart die Anzahl der erreichten Punkte nichts mit dem Spielausgang zu tun hat. Deshalb vergleicht der MCU-Spieler bei Nullspielen die Anzahl der gewonnenen Spiele für jede spielbare Karte. Die Karte mit der höchsten Gewinnaussicht wird gespielt. Erzeugen mehrere Karten die selbe Zahl an gewonnenen Spielen, so wird zufällig eine von ihnen ausgewählt.

Der Klasse `MCPlayer` können verschiedene Optionen im Konstruktor übergeben werden. Zunächst kann die Anzahl der durchzuführenden Simulationen pro Spielzug festgelegt werden. Das ermöglicht den Vergleich der Spielstärken von KI-Spielern, welche ebenfalls die Monte Carlo Simulation für die Berechnung ihrer Spielzüge verwenden. Die Spieler ermitteln durch die gleiche Anzahl an Simulationen gleich große Stichproben aus der Menge aller möglichen Welten und bewerten diese. Außerdem lässt sich mit der Begrenzung der Simulationen feststellen, ab welcher Stichprobengröße der Ansatz besser spielt als ein Programm mit festen Heuristiken.

Die maximale Bedenkzeit kann ebenfalls begrenzt werden. In diesem Falle werden solange neue Kartenverteilungen erzeugt bis die Bedenkzeit abgelaufen ist. Erst dann werden die Ergebnisse der einzelnen simulierten Spiele ausgewertet. Das lässt Untersuchungen der Spielstärke bei Spielen mit menschlichen Spielern zu, da beim Spielen mit realen Spielern nicht unbegrenzt Bedenkzeit zur Verfügung steht. Da schnellere Rechner mehr Simulationen in der gleichen Zeit durchführen können, wurde noch ein Faktor eingeführt, der von der CPU-Frequenz abhängig ist. Als Referenzfrequenz wurden 3 GHz festgelegt. Somit ist es möglich, die zeitlich begrenzten Simulationen auch auf Computern mit unterschiedlichen CPU-Frequenzen berechnen zu lassen.

5.2 Untersuchung der Verteilung der Augenzahlen

Bei den Farb- und Grandspielen muss der Alleinspieler mindestens 61 Punkte erreichen, um das Spiel zu gewinnen. Die Gegenspieler haben ebenso ein Interesse, so viele Punkte wie möglich zu erlangen. Es ist daher naheliegend, die erzielten Augen zur Bewertung der einzelnen Spiele heranzuziehen. Die Art der Verteilungen der erzielten Augenzahlen am Ende der Spiele ist bis jetzt noch unbekannt und soll deshalb an dieser Stelle näher beleuchtet werden.

Dazu wurden die erreichten Augenzahlen pro Karte und Stich bei einem MCU-Spieler mit 10.000 Simulationen pro Spielzug untersucht. Es konnten in unterschiedlichen Spielen vielfältige Verteilungsarten beobachtet werden, die von der Normalverteilung, schiefen Verteilungen und Verteilungen mit zwei Maxima bis zur Gleichverteilung reichten. In manchen Fällen verteilen sich die erzielten Augenzahlen sogar nur auf einige wenige Werte. In den Abbildungen 5.2, 5.3 und 5.4 sind die Verteilungen der Augen graphisch dargestellt. Es fällt auf, dass in den ersten Stichen eher Normalverteilungen und schiefe Verteilungen vorherrschen. Je weiter das Spiel fortschreitet, desto geringer wird im Allgemeinen die Streuung. Die Verwendung des Mittelwertes ist bei dieser Vielzahl an Verteilungen die einfachste Möglichkeit, eine Entscheidung für eine Karte zu treffen.

Die Standardabweichung der erreichten Augenzahlen wird ebenfalls bei jeder Simulation errechnet. Führen die Simulationen der Spielverläufe in der Mehrzahl zu gewon-

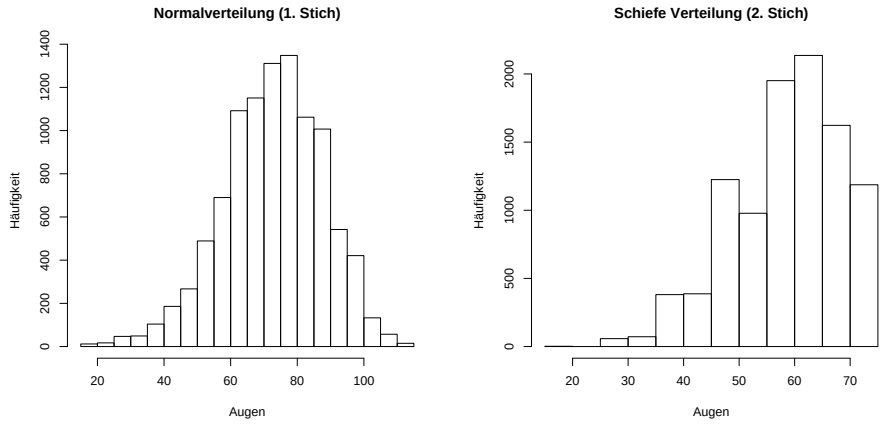

Abbildung 5.2: Normalverteilung und schiefe Verteilung der Augenzahlen

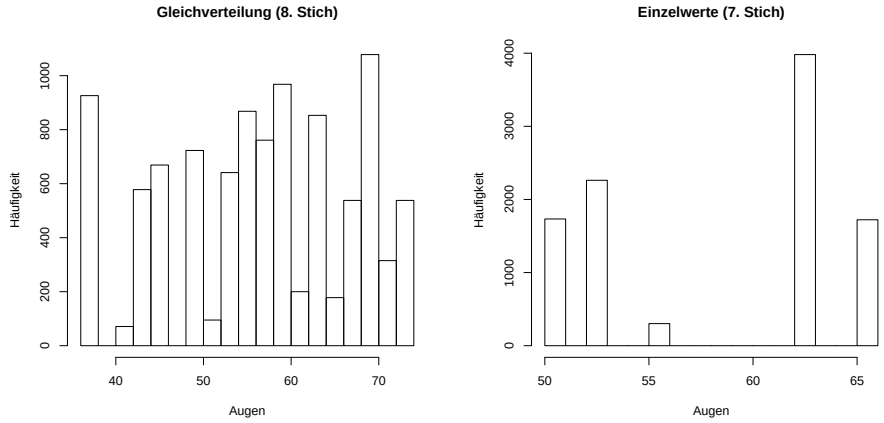

Abbildung 5.3: Gleichverteilung und Einzelwerte bei den Augenzahlen

nenen Spielen, dann sollte der Spieler eher auf Sicherheit spielen und eine Karte mit geringer Standardabweichung wählen. Hier streuen die Ergebnisse der einzelnen Simulationen nicht so stark. Steht der Spieler eher nicht so gut da, zum Beispiel wenn er sich bei den Spielen innerhalb der simulierten Welten in der Verlustzone[1] befindet, so kann er risikoreicher spielen und eine Karte mit höherer Standardabweichung wählen, vorausgesetzt der Mittelwert der erreichten Augen plus Standardabweichung verspricht bei einigen Spielen einen Gewinn.

[1]Für den Alleinspieler ist das der Bereich von 0 bis 60 Punkten bei Farb- und Grandspielen.

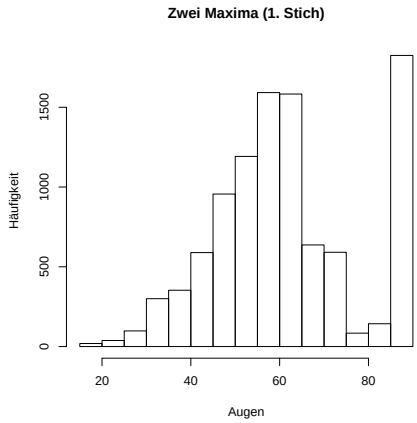

Abbildung 5.4: Verteilung der Augenzahlen mit zwei Maxima

5.3 Einfache Monte Carlo Simulation

Zunächst wurde ein sehr einfacher Algorithmus zur Erzeugung von möglichen Kartenverteilungen implementiert. Zur Simulation wird nur das Wissen über alle bekannten Karten verwendet. Das sind einerseits die Karten, welche der Spieler selbst auf der Hand hat und die ausgespielten Karten. Wenn der MCU-Spieler der Alleinspieler in einem Spiel ist und in den Skat gesehen hat, werden diese beiden Karten ebenfalls zu den bekannten Karten gezählt. Von Stich zu Stich gibt es immer weniger unbekannte Karten, die simuliert werden müssen. Die simulierten Kartenverteilungen weisen deshalb in den letzten Stichen weniger Varianzen auf als am Anfang.

5.3.1 Algorithmus

Der Algorithmus besteht aus drei einfachen Schritten und ist im Pseudocode in Abbildung 5.5 dargestellt. Er kann für die Erzeugung von möglichen Kartenverteilungen verwendet werden, da jede Kartenverteilung beim Skat bei gutem Mischen gleichwahrscheinlich ist. Die unbekannten Karten werden durch eine Differenz der Menge aller Karten und der Menge der bekannten Karten ermittelt. Die Elemente der erzeugten Menge werden in eine zufällige Ordnung gebracht und dann auf die noch unbekannten Kartenpositionen der einzelnen Spieler verteilt.

5.4 Ermittlung der Spielstärke

Damit sich alle Experimente in dieser Arbeit besser vergleichen lassen, wurden zufällig 1.000 Kartenverteilungen erzeugt und bei jedem neuen Experiment wiederverwendet.

```
getCardDistribution(knownCards) {

  constant allCards = {0, 1, 2, ... , 30, 31}

  unknownCards = getDifference(allCards, knownCards);
  shuffle(unknownCards);
  dealToPlayers(unknownCards);
}
```

Abbildung 5.5: Einfacher Algorithmus zur Erzeugung möglicher Kartenverteilungen

Durch die Verwendung der Heuristiken von XSkat für das Reizen und Drücken bei allen drei Spielern ist die Verteilung der einzelnen Spielarten in dieser Stichprobe immer gleich (siehe Tabelle 5.1). Die Nullspiele sind eher unterrepräsentiert, was sich mit dem Reizverhalten von XSkat erklären lässt. Es reizt nur auf ein Nullspiel, wenn es sicher ist. Deshalb werden die Nullspiele später noch einmal gesondert betrachtet.

Spielart	Anzahl
Nullspiele	7
Farbspiele	701
davon	
♢-*Spiele*	*144*
♡-*Spiele*	*158*
♠-*Spiele*	*192*
♣-*Spiele*	*207*
Grandspiele	292
Gesamt	1.000

Tabelle 5.1: Verteilung der Spielarten in den Experimenten

5.4.1 Experimente

Der implementierte MCU-Spieler mit einfacher Kartensimulation trat in den ersten Experimenten gegen XSkat an. Da auch das Zusammenspiel untersucht werden sollte, wurden zwei Skatrunden durchgeführt. Zunächst spielte der MCU-Spieler gegen zwei Instanzen von XSkat. In der zweiten Runde trat XSkat gegen zwei Instanzen des MCU-Spielers an. Die Kartenverteilungen wurden jeweils mit allen möglichen Sitzkombinationen wiederholt. Das führte zu 3.000 Spielen pro Skatrunde, wobei jeder Spielertyp die gleiche Chance hatte, auf jeder Sitzposition gegen jede andere Spielerkombination zu spielen.

Beim MCU-Spieler wurde die maximale Anzahl von Simulationen bei aufeinanderfolgenden Experimenten auf 5, 10, 25, 50, 100, 250, 500 und 1.000 Simulationen beschränkt. In einem zweiten Durchlauf war die Bedenkzeit mit 1, 2 und 3 Sekunden nach oben begrenzt. Längere Bedenkzeiten wurden nicht untersucht, da beim Skatspielen mit menschlichen Gegnern schon drei Sekunden eine lange Bedenkzeit bedeuten. Außerdem waren die Laufzeiten einer Skatrunde von mehr als 20 Stunden bei 3 Sekunden Bedenkzeit schon sehr lang.[2]

5.4.2 Vergleich der Spielstärken

Für den Vergleich der Spielstärken lassen sich vielfältige Maßzahlen finden. Die naheliegendste ist die Anzahl gewonnener und verlorener Spiele, da die Spiele wiederholt wurden und somit jeder Skatspieler die gleiche Chance hatte, alle Spiele mit den gegebenen Karten durchzuführen. Der Spieler mit den meisten gewonnenen Spielen ist auch der beste Spieler.

Eine weitere Maßzahl sind die erzielten Augen pro Spiel. Somit hat man für jedes Farb- und Grandspiel die selbe Wertigkeit. Leider passen die Nullspiele nicht in dieses Schema, da der Alleinspieler nur durch das Erzielen von null Augen das Spiel gewinnt.

Das Punktergebnis innerhalb der Skatrunde kann ebenfalls betrachtet werden. Der Spieler mit den höchsten Punkten gewinnt die Skatrunde. Leider sind die Spielwerte sehr unterschiedlich. Sie können von 18 Punkten für ein Karospiel mit einer Spitze bis zu 264 Punkten für ein Grand Ouvert mit vier Spitzen und darüber hinaus reichen. Dabei kann ein Spieler mit wenigen hohen Grandspielen, die durch eine glückliche Kartenverteilung zustande gekommen sind, viele kleine Farbspiele eines anderen Spielers überdecken. Da aber die Spiele bei den Experimenten wiederholt werden, hat jeder Spieler die gleiche Chance, jedes Spiel zu spielen. Diese Bewertung spiegelt aber nur die Spielstärke der Spieler als Alleinspieler wieder.

Um auch das Spielen in der Gegenpartei mit in die Bewertung einfließen zu lassen, bietet sich die Turnierbewertung nach dem Seeger-Fabian-System an. Hat ein Spieler mehr Punkte erhalten, so spielt er besser als die anderen.

Die Tabellen 5.2 und 5.3 zeigen die Ergebnisse der ersten Experimente. Die maximale Anzahl von simulierten Kartenverteilungen pro Spielzug sind mit S_x angegeben. Analog dazu sind die Spiele mit begrenzter Bedenkzeit mit T_x bezeichnet. Wenn von einem Spielertypen mehr als eine Instanz in der Skatrunde mitgespielt hat, so wurden die Ergebnisse aller Instanzen gemittelt und auf ganze Punkte abgerundet.

Schon ab 10 Simulationen pro Spielzug kann der MCU-Spieler ein höheres Ergebnis erzielen als XSkat. Dies ist nicht weiter verwunderlich, da der MCU-Spieler die Heuris-

[2]Diese Zeit bezieht sich auf ein Spiel XSkat vs. 2x MCU auf einem Athlon XP 3000+.

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$XSkat$	80.259	77.178	72.321	71.898	71.248	72.074	72.156	70.642
MCU_{einf}	78.719	86.444	90.115	89.869	93.109	91.261	91.772	92.046

Spieler	T_1	T_2	T_3
$XSkat$	71.316	71.109	70.887
MCU_{einf}	90.754	92.423	92.532

Tabelle 5.2: Spiele mit Turnierbewertung: 2x $XSkat$ vs. $MCU_{einfach}$

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$XSkat$	81.177	81.242	75.267	73.764	73.877	69.865	71.545	68.677
MCU_{einf}	79.867	82.395	83.004	84.056	82.775	84.980	83.146	85.234

Spieler	T_1	T_2	T_3
$XSkat$	69.383	67.747	67.211
MCU_{einf}	78.918	78.510	79.915

Tabelle 5.3: Spiele mit Turnierbewertung: $XSkat$ vs. 2x $MCU_{einfach}$

tiken von XSkat zwar nicht genau kennt, sie aber für das Finden seiner Spielstrategie benutzt. Ab 250 Simulationen pendelt sich die Punktezahl in der Runde mit nur einem MCU-Spieler bei rund 92.000 Punkten ein. Bei beschränkter Bedenkzeit gelingt es dem MCU-Spieler bei einer Sekunde mehr als 1.000 Simulationen pro Spielzug durchzuführen. Dies führt aber nur zu einer leichten Steigerung der Punktezahl. In der Skatrunde mit zwei MCU-Spielern erreichen diese nur eine Punktzahl von maximal 85.000 Punkten. XSkat spielt ungefähr auf dem Niveau der ersten Runde. Die beiden MCU-Spieler beeinflussen sich also sehr stark in ihrer Spielweise. Bei den zeitbegrenzten Spielen ist ein dramatischer Abfall der Spielstärke zu beobachten. Das ist ein Hinweis auf die unzureichende Qualität der simulierten Kartenverteilungen.

5.5 Optimierung der Monte Carlo Simulation

Der oben vorgestellte Algorithmus zur Erzeugung von möglichen Kartenverteilungen ermöglicht es dem MCU-Spieler, besser als XSkat zu spielen. Bisher wurden aber nur die Informationen über die ausgespielten Karten für die Simulationen verwendet. Der beim Skat herrschende Farb- oder Bedienzwang lässt aber noch weitere Schlüsse über die mögliche Kartenverteilung zu.

Wenn ein Spieler eine Farbe nicht bedienen kann, so kann man sicher sein, dass er diese Farbe nie hatte oder nicht mehr besitzt. Hätte der Spieler eine solche Karte zurückgehalten, würde er Schwarz spielen und das Spiel wäre verloren. Das Gleiche gilt für die Trumpfkarten. Man kann diese Informationen in die Simulation von Kartenverteilungen einfließen lassen, um nur noch Kartenverteilungen zu erzeugen, die tatsächlich noch möglich sind. Diese Optimierung reduziert die Anzahl der möglichen Kartenverteilungen. Außerdem gibt es keine falschen Verteilungen mehr, welche die Ergebnisse der simulierten Spiele verfälschen können.

5.5.1 Verbesserter Stichprobenalgorithmus

Zusätzlich zu den gespielten Karten merkt sich der MCU-Spieler, wieviele Karten einer Farbe noch nicht gespielt wurden. Bei Farb- und Grandspielen werden die Buben extra gezählt, da sie zum Trumpf zählen. Kann ein Spieler eine Farbe oder Trumpf nicht bedienen, so wird diese Information ebenfalls erfasst. Der MCU-Spieler übergibt dann für die Simulation einer Kartenverteilung durch ein CardDeck-Objekt nur sichere Informationen. Nur wenn sicher ist, dass ein Spieler eine Farbe oder Trumpf nicht mehr hat, fließt diese Information in die Simulation einer Kartenverteilung ein. Der optimierte Algorithmus benutzt dieses Wissen zur Erzeugung von Kartenverteilungen, die tatsächlich noch möglich sind. Er ist in Abbildung 5.6 im Pseudocode zu finden.

Zu Beginn der Simulation einer möglichen Kartenverteilung wird der Gegenspieler mit den meisten Einschränkungen ermittelt. Das ist derjenige Spieler, welcher die meisten Farben nicht mehr bedienen kann. Diese Information stammt aus dem Wissen über die bekannten Karten und der Spielweise der Gegenspieler. Der Gegenspieler mit den meisten Einschränkungen muss zuerst simuliert werden, damit nicht zufällig Karten, die bei beiden Gegenspielern möglich sind, alle auf den Spieler mit den geringeren Einschränkungen verteilt werden und am Ende für den Spieler mit den höheren Einschränkungen keine Karten mehr übrig bleiben.

Als Nächstes werden alle 32 Karten eines Skatblattes untersucht und in Klassen eingeteilt. Wenn eine Karte schon gespielt wurde oder im Besitz des simulierenden Spielers ist, so landet sie in der Menge der bekannten Karten. Trifft dies nicht zu, wird für beide Gegenspieler getestet, ob sie nach den bisher bekannten Informationen über die Kartenverteilung die Karte besitzen könnten oder nicht. Kann der aktuell untersuchte Gegenspieler die Karten nicht besitzen, wird sie zur Menge der bekannten Karten hinzugefügt. Kann keiner der beiden Gegenspieler die Karte besitzen, so gehört sie in den Skat. Wenn nur der zweite Gegenspieler die Karte nicht haben kann, so gehört diese Karte zu den zwingenden Karten, die unbedingt dem aktuellen Spieler zugeordnet werden müssen.

```
getOptimizedCardDistribution(player) {

  constant allCards = {0, 1, 2, ... , 30, 31}
  knownCards = skatCards = obligatoryCards = unknownCards = {};
  opponentPlayers = getDifference(allPlayers, player);
  mostRestrictedPlayer = getMostRestrictedPlayer(opponentPlayers);

  forall (opp in opponentPlayers startingwith mostRestrictedPlayer) {
    forall (card in allCards) {

      if (cardWasAlreadyPlayed(card) or
          cardOnPlayersHand(player, card)) {
        knownCards.add(card);
      }
      else {

        icp1 = isCardPossible(opp);
        icp2 = isCardPossible(getDifference(opponentPlayers, opp));

        if (!icp1) {
          knownCards.add(card);
        }
        if (!icp2) {
          if (!icp1) {
            skatCards.add(card);
          }
          else {
            obligatoryCards.add(card);
    } } } }

    if (skatCards.count() > 0) {
      putIntoSkat(skatCards);
      knownCards.add(skatCards);
    }
    if (obligatoryCards.count() > 0) {
      dealCardsToPlayer(opp, obligatoryCards);
      knownCards.add(obligatoryCards);
    }
    unknownCards = getDifference(allCards, knownCards);
    unknownCards.shuffle();
    dealToPlayer(unknownCards, opp);
  }
}
```

Abbildung 5.6: Verbesserter Algorithmus zur Erzeugung möglicher Kartenverteilungen

Nach der Untersuchung aller Karten werden diejenigen, welche definitiv zum Skat gehören, in den Skat gelegt und der Menge der bekannten Karten zugeordnet. Danach werden die zwingenden Karten dem aktuellen Spieler zugeordnet und ebenfalls in die Menge der bekannten Karten aufgenommen. Nun wird eine Differenz aus allen Karten und den bekannten Karten gebildet, um die noch unbekannten Karten zu ermitteln, die jeder der beiden Gegenspieler auf der Hand haben könnte. Die ermittelte Menge wird in eine zufällige Reihenfolge gebracht und die noch offenen Kartenpositionen für den aktuellen Gegenspieler aufgefüllt.

Diese Schritte werden für den Spieler mit den geringeren Einschränkungen nochmals wiederholt. Am Schluss erhält man immer eine Kartenverteilung, die mit den aktuellen Informationen über die tatsächliche Kartenverteilung konform ist. Es werden keine Kartenverteilungen erzeugt, die nach dem aktuellen Wissensstand unmöglich sind.

5.5.2 Experimente

Für die Untersuchung der Wirkungsweise des optimierten Algorithmus zur Erzeugung von möglichen Kartenverteilungen wurden wieder die oben beschriebenen 1.000 Kartenverteilungen verwendet. Es wurden erneut zwei Skatrunden gespielt, einmal mit zwei Instanzen des XSkat-Spielers und in der anderen Runde mit zwei MCU-Spielern. Wie beim einfachen Algorithmus wurde die Zahl der Simulationen auf 5, 10, 25, 50, 100, 250, 500 und 1.000 Simulationen pro Spielzug beschränkt. In einem zweiten Durchlauf war die Bedenkzeit wieder auf maximal 1, 2 und 3 Sekunden gesetzt. Die Ergebnisse der einzelnen Skatrunden sind in den Tabellen 5.4 und 5.5 dargestellt.

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$XSkat$	75.213	74.657	72.155	70.922	68.545	69.044	69.132	70.131
MCU_{opt}	83.451	85.133	88.592	90.866	92.308	93.219	94.280	92.277

Spieler	T_1	T_2	T_3
$XSkat$	69.796	69.362	69.866
MCU_{opt}	92.016	92.300	91.002

Tabelle 5.4: Spiele mit Turnierbewertung: 2x $XSkat$ vs. $MCU_{optimiert}$

Durch die optimierte Simulation der Kartenverteilung schafft es der MCU-Spieler schon bei 5 Simulationen pro Spielzug, besser als XSkat zu spielen. Bei 500 Simulationen ist schon das Maximum der Punktezahl erreicht. Danach pendeln sich die Punkte wieder bei rund 92.000 Punkten ein. Wenn zwei MCU-Spieler in einer Runde mitspielen, konnte wieder beobachtet werden, dass die beiden MCU-Spieler sich behindern und eine geringere Punktzahl erzielen.

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$XSkat$	80.860	74.921	72.390	67.598	67.060	64.781	68.368	68.791
MCU_{opt}	81.365	81.832	83.449	84.807	83.330	84.363	84.314	83.761

Spieler	T_1	T_2	T_3
$XSkat$	67.948	66.565	66.488
MCU_{opt}	80.450	79.846	82.015

Tabelle 5.5: Spiele mit Turnierbewertung: $XSkat$ vs. 2x $MCU_{optimiert}$

5.6 Nullspiele

Bei den Nullspielen spielt die erreichte Augenzahl keine Rolle. Deshalb wird bei den Nullspielen innerhalb der Simulationen nicht die erreichten Augen zur Bewertung einer Karte hinzugezogen, sondern die Anzahl der gewonnenen Spiele. Die Karte mit der höchsten Anzahl wird für das Ausspiel ausgewählt. Sollten mehrere Karten die gleiche Anzahl gewonnener Spiele erzielen, so wird zufällig eine dieser Karten ausgewählt.

Wie bereits erwähnt, waren die Nullspiele in den bisherigen Experimenten sehr unterrepräsentiert. Für die Untersuchung wurden deshalb 1.000 Kartenverteilungen für Spiele generiert, bei denen XSkat auf Null reizt. Es wurden wiederum zwei Skatrunden untersucht, wobei beim MCU-Spieler gleich die optimierte Simulation von Kartenverteilungen zum Einsatz kam.

Die Ergebnisse sind in den Tabellen 5.6 und 5.7 dargestellt. Bei den mit „n.d.“ gekennzeichneten Skatrunden handelt es sich um Experimente, die bis zur Abgabe der Arbeit nicht zu Ende gerechnet werden konnten. Ab 25 Simulationen pro Spielzug gelingt es dem MCU-Spieler ein höheres Endergebnis als XSkat zu erreichen. Bei 1.000 Simulationen verliert er sogar nur 6 der 1.000 gespielten Nullspiele. Auch hier pendeln sich die erzielten Punkte ab 250 Simulationen auf einen Wert von rund 97.000 Punkten ein. Mehr Simulationen führen ab diesem Punkt kaum noch zu einer Steigerung der Spielstärke

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$XSkat$	89.345	88.582	87.220	87.292	85.885	85.693	86.097	85.645
MCU_{opt}	84.505	87.948	95.340	95.340	96.044	96.659	96.869	97.274

Spieler	T_1	T_2	T_3
$XSkat$	85.272	85.401	85.120
MCU_{opt}	97.269	97.189	97.679

Tabelle 5.6: Nullspiele mit Turnierbewertung: 2x $XSkat$ vs. $MCU_{optimiert}$

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$XSkat$	88.759	88.762	84.825	84.926	85.096	83.053	83.141	83.383
MCU_{opt}	85.271	89.439	92.015	92.815	93.086	94.206	93.696	94.327

Spieler	T_1	T_2	T_3
$XSkat$	82.973	83.812	n.d.
MCU_{opt}	94.479	94.122	n.d.

Tabelle 5.7: Nullspiele mit Turnierbewertung: $XSkat$ vs. 2x $MCU_{optimiert}$

Kapitel 6

Bewertungsfunktionen für die Simulationen

Bei den Simulationen entstehen vielfältige Daten, welche sich zur Verwendung in einer Bewertungsfunktion eignen. Die Verwendung einer Bewertungsfunktion für den Spielausgang hat den Vorteil, dass sie auf alle Spielarten anwendbar ist. Es muss also nicht zwischen Farb- und Grandspielen auf der einen und Nullspielen auf der anderen Seite unterschieden werden. Deshalb darf die Bewertungsfunktion sich nicht an der erzielten Augenzahl orientieren, sondern es muss ein anderes Bewertungskriterium gefunden werden.

6.1 Treppenfunktion

Eine von der Spielart und der erreichten Augenzahl unabhängige Größe beim Skat ist die erreichte Gewinnstufe für die Spielabrechnung. In Tabelle 6.1 sind die einzelnen erreichbaren Gewinnstufen mit dem Faktor für die Spielabrechnung aufgelistet.

Spielausgang	Gewinnstufe
Verloren + Schwarz	-8
Verloren + Schneider	-4
Verloren	-2
Gewonnen	1
Gewonnen + Schneider	2
Gewonnen + Schwarz	3

Tabelle 6.1: Spielbewertungsfunktion

Der Ausgang jedes Spieles kann durch diese Bewertungsfunktion auf den Zahlenbereich von -8 bis 3 abgebildet werden. Bei den Nullspielen gibt es nur die beiden Spielausgänge Gewonnen (Wert 1) und Verloren (Wert -2).

Die resultierende Treppenfunktion, die sich für Farb- und Grandspiele ergibt, hat jeweils bei 0, 30, 61 (60 bei den Gegenspielern), 90 und 120 Augen ihre Stufen. Der Unterschied der mittleren Stufe zwischen den beiden Spielerparteien ist darin begründet, dass die Gegenspieler schon bei 60 Augen das Spiel gewinnen, wohingegen der Alleinspieler 61 Augen benötigt, um das Spiel zu gewinnen. In Abbildung 6.1 ist die Treppenfunktion für einen Alleinspieler dargestellt.

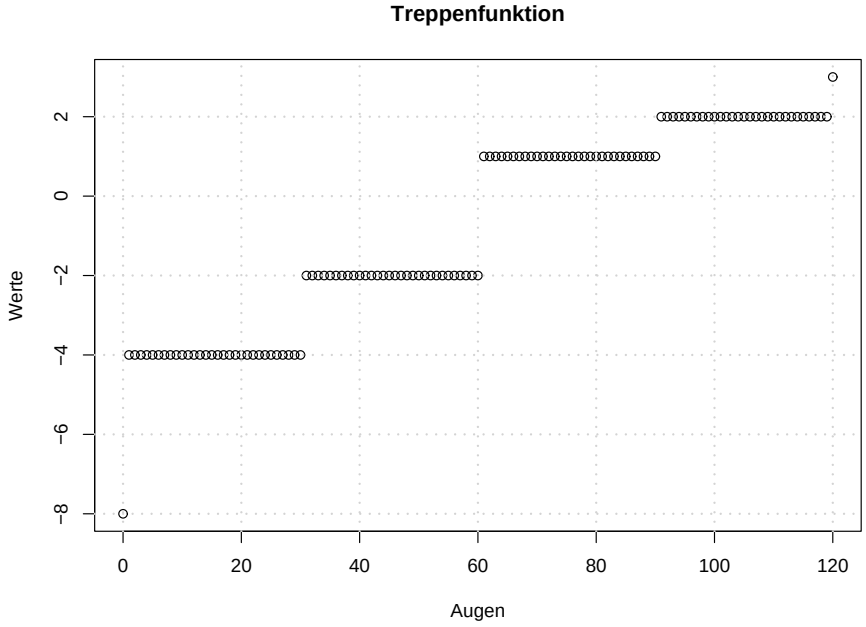

Abbildung 6.1: Treppenfunktion

Mit der Treppenfunktion als Bewertungsfunktion für die Spielausgänge innerhalb der Simulationen wurden wieder die bekannten 1.000 Kartenverteilungen gespielt. Um die Auswirkung der Bewertungsfunktion darstellen zu können, werden die folgenden Skatrunden immer mit dem MCU-Spieler mit optimierter Kartensimulation und der mittleren Augenzahl als Bewertungskriterium durchgeführt. Dieser MCU-Spieler hatte sich ja bereits als bester Spieler bei den Vergleichen mit XSkat herausgestellt.

Die Ergebnisse zeigen deutlich, dass die Treppenfunktion nicht gegen die Auswertung der mittleren erreichten Augenzahl bestehen kann. Auch mehr Simulationen pro Spielzug führen nicht zu einer Steigerung der Spielstärke.

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
MCU_{opt}	87.020	86.095	83.636	81.401	81.713	79.554	80.404	79.507
MCU_{opt}^{Treppe}	68.976	67.212	70.398	66.265	70.182	69.873	70.129	68.781

Spieler	T_1	T_2	T_3
MCU_{opt}	80.422	80.865	79.718
MCU_{opt}^{Treppe}	70.274	70.024	72.334

Tabelle 6.2: Spiele mit Turnierbewertung: 2x $MCU_{optimiert}$ vs. $MCU_{optimiert}^{Treppe}$

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
MCU_{opt}	95.145	92.584	89.134	79.804	84.589	79.279	82.197	81.055
MCU_{opt}^{Treppe}	71.390	71.738	70.604	76.912	72.566	71.163	71.045	69.655

Spieler	T_1	T_2	T_3
MCU_{opt}	79.916	77.234	82.271
MCU_{opt}^{Treppe}	71.415	70.217	68.328

Tabelle 6.3: Spiele mit Turnierbewertung: $MCU_{optimiert}$ vs. 2x $MCU_{optimiert}^{Treppe}$

6.2 Bewertungsfunktion mit Schrägen Typ I

Die langen Stufen in der Treppenfunktion eignen sich nicht zur Unterscheidung, ob die erzielte Punktzahl am Anfang oder am Ende einer Stufe liegt. Somit werden Karten, welche einen Spielwert knapp vor der nächsten Stufe erreichen, genauso bewertet, wie Karten, die diese Stufe gerade so erreicht haben. Durch Einführung von Schrägen an den Treppenstufen soll nun untersucht werden, ob dies zu einer besseren Bewertung von Karten und dadurch zu einer besseren Spielstärke des MCU-Spielers führt.

Bei Typ I werden die Schrägen zunächst einige Augen vor der nächsten Stufe (5, 10, 15 oder 20 Augen) mit einem linearen Anstieg bis zur nächsten Stufe angelegt. In Abbildung 6.2 ist die Funktion graphisch dargestellt. Die Schrägen beginnen hier 10 Augen vor der nächsten Gewinnstufe. Bei Nullspielen gibt es wieder nur die Werte -2 und 1 für verlorene und gewonnene Spiele

Auch für die Bewertungsfunktion mit Schrägen vom Typ I wurden die Vergleiche der Spielstärke mit dem bisher besten MCU-Spieler durchgeführt. Ihm wurde der MCU-Spieler mit verschiedenen Ausprägungen der Bewertungsfunktion mit Schrägen vom Typ I (5, 10, 15, 20 Augen vor einer Stufe) gegenübergestellt.

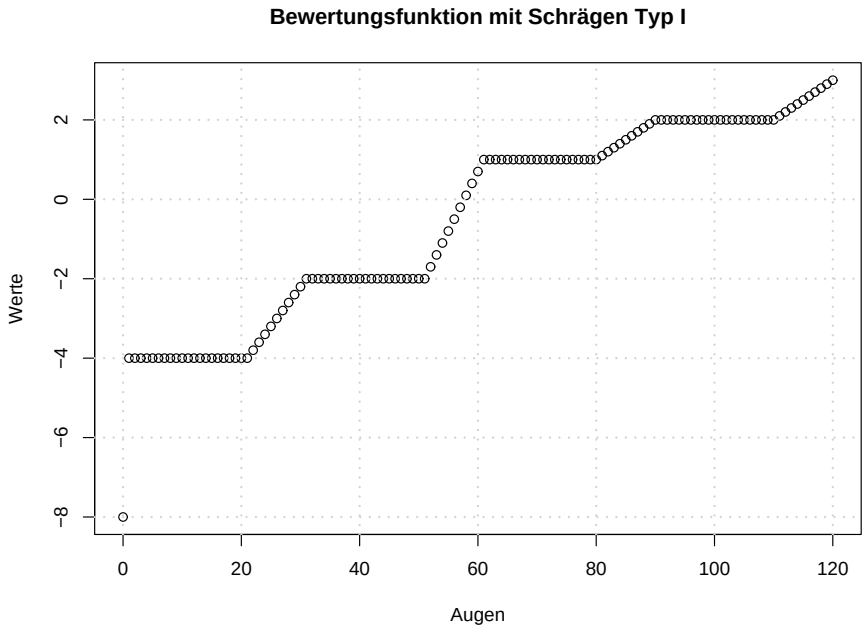

Abbildung 6.2: Bewertungsfunktion mit Schrägen Typ I

Die Ergebnisse in den Tabellen 6.4 und 6.5 zeigen für die untersuchten Bewertungsfunktionen vom Typ I kein einheitliches Ergebnis. Es kommt beim Erhöhen der Anzahl der simulierten Kartenverteilungen immer wieder ein Vorteil für die Verwendung der mittleren erreichten Augenzahl oder für die Bewertungsfunktion heraus. Nur bei der letzten Bewertungsfunktion vom Typ I mit Schrägen, die 20 Punkte vor der nächsten Stufe beginnen, liegt der Vorteil öfter bei der Funktion als bei der mittleren Augenzahl.

6.3 Bewertungsfunktion mit Schrägen Typ II

Beim Typ II werden die Schrägen linear ansteigend, beginnend vom Anfang der Stufe bis zu einer gewissen Höhe der nächsten Stufe, angelegt. Die Schräge auf der mittleren Stufe steigt aber nicht über den Wert 0 hinaus. Damit soll der Sprung beim Übergang vom Verlust zum Gewinn des Spieles nicht zu sehr verwischt werden. Die Schrägen der einzelnen Funktionen erreichen 25 %, 50 % und 75% der Stufenhöhe. In Abbildung 6.3 wird die Funktion mit Schrägen bis 50 % der nächsten Stufenhöhe gezeigt.

Bei den Experimenten mit den drei verschiedenen Funktionen vom Typ II wurden die Ergebnisse in den Tabellen 6.6 und 6.7 erzielt. Zumindest bei den Spielen mit zwei MCU-Spielern, welche die Bewertungsfunktion vom Typ II nutzen, konnte diese öfter gegen den normalen MCU-Spieler mit Verwendung der mittleren Augenzahl bestehen.

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$MCU_{optimiert}$	86.503	83.359	82.056	81.207	78.064	80.261	78.218	79.867
$MCU_{optimiert}^{TypI(5)}$	72.978	77.333	75.585	74.579	79.429	74.097	74.402	75.098
$MCU_{optimiert}$	83.129	83.173	81.849	80.592	79.704	77.989	79.642	78.814
$MCU_{optimiert}^{TypI(10)}$	78.208	77.870	79.239	77.334	78.206	78.508	79.201	79.311
$MCU_{optimiert}$	82.438	81.882	82.937	77.987	78.595	78.957	76.234	77.202
$MCU_{optimiert}^{TypI(15)}$	81.259	81.159	79.555	82.230	82.000	78.515	83.133	79.288
$MCU_{optimiert}$	84.178	81.500	78.224	78.172	79.746	79.556	78.522	78.351
$MCU_{optimiert}^{TypI(20)}$	75.938	82.467	85.189	81.550	77.473	77.922	81.474	80.451

Spieler	T_1	T_2	T_3
$MCU_{optimiert}$	79.171	79.356	79.629
$MCU_{optimiert}^{TypI(5)}$	72.669	73.989	74.358
$MCU_{optimiert}$	80.407	77.459	77.744
$MCU_{optimiert}^{TypI(10)}$	77.629	79.387	79.316
$MCU_{optimiert}$	78.310	77.444	78.050
$MCU_{optimiert}^{TypI(15)}$	81.577	81.497	80.826
$MCU_{optimiert}$	77.586	78.559	77.929
$MCU_{optimiert}^{TypI(20)}$	81.443	77.197	77.756

Tabelle 6.4: Spiele mit Turnierbewertung: 2x $MCU_{optimiert}$ vs. $MCU_{optimiert}^{TypI}$

Dabei zeigte sich, dass der lineare Anstieg bis 50 % der nächsten Stufenhöhe die besten Ergebnisse erzielte. Der normale MCU-Spieler erreichte deutlich weniger Punkte.

6.4 Bewertungsfunktion mit Schrägen Typ III

Der Typ III ist eine Kombination aus den Typen I und II. Die Schrägen steigen linear vom Beginn der Stufe bis zur nächsten Stufe an. Die Funktion ist in Abbildung 6.4 zu sehen.

Die Ergebnisse der Experimenten mit der Funktion vom Typ III sind in den Tabellen 6.8 und 6.9 dargestellt. Auch bei Verwendung der Bewertungsfunktion vom Typ III konnte ein höheres Punkteergebnis erzielt werden als mit der Verwendung der mittleren Augenzahl als Bewertungskriterium. Die Punkte des normalen MCU-Spielers konnten aber nicht so deutlich gedrückt werden, wie es bei der Bewertungsfunktion vom Typ II gezeigt werden konnte.

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$MCU_{optimiert}$	89.935	88.558	82.094	81.604	79.242	79.390	80.469	79.496
$MCU_{optimiert}^{TypI(5)}$	76.247	76.629	74.734	75.578	75.995	74.832	73.404	74.146
$MCU_{optimiert}$	81.559	85.404	80.419	81.174	78.883	79.337	77.734	79.424
$MCU_{optimiert}^{TypI(10)}$	80.184	78.774	79.390	79.341	80.049	76.445	76.522	78.932
$MCU_{optimiert}$	85.369	82.131	81.716	82.389	81.190	78.921	77.962	76.770
$MCU_{optimiert}^{TypI(15)}$	81.348	82.443	81.879	78.877	77.974	79.001	78.971	80.128
$MCU_{optimiert}$	83.614	81.795	79.056	77.307	75.858	77.485	75.200	77.602
$MCU_{optimiert}^{TypI(20)}$	80.751	81.983	80.133	80.909	80.497	79.994	81.395	78.407

Spieler	T_1	T_2	T_3
$MCU_{optimiert}$	80.596	80.244	77.764
$MCU_{optimiert}^{TypI(5)}$	73.034	73.764	76.004
$MCU_{optimiert}$	77.872	78.686	77.600
$MCU_{optimiert}^{TypI(10)}$	78.554	79.294	79.115
$MCU_{optimiert}$	76.977	77.111	76.282
$MCU_{optimiert}^{TypI(15)}$	79.006	79.243	80.159
$MCU_{optimiert}$	77.100	78.661	78.054
$MCU_{optimiert}^{TypI(20)}$	77.839	79.561	78.212

Tabelle 6.5: Spiele mit Turnierbewertung: $MCU_{optimiert}$ vs. 2x $MCU_{optimiert}^{TypI}$

6.5 Nullspiele

Bei Nullspielen gibt es bei jeder vorgestellten Bewertungsfunktion nur die Funktionswerte -2 und 1 für verlorene und gewonnene Spiele. Weitere Gewinnstufen gibt es bei Nullspielen nicht. Deshalb sind in den Tabellen 6.10 und 6.11 nur stellvertretend die Ergebnisse für die beiden Skatrunden des normalen MCU-Spielers gegen den MCU-Spieler mit der Bewertungsfunktion vom Typ II angegeben.

Bei den Nullspielen konnte keine Überlegenheit der Bewertungsfunktion gegenüber der Ermittlung der besten Karte aus der Anzahl der gewonnenen Spiele gezeigt werden. Bei den durchgeführten Skatrunden können beide ähnliche Endergebnisse erzielen. Ab 100 Simulationen ändern sich diese kaum noch und liegen im Bereich von rund 90.000 Punkten. Die Gleichwertigkeit lässt sich erklären, wenn man die Bewertung nach der Anzahl der gewonnenen Spiel auch als Bewertungsfunktion mit den Ausprägungen 0 und 1 betrachtet. Bei der Funktion vom Typ III wird der Mittelwert über alle Funktionswerte gebildet, anstatt die Werte nur zu addieren. Dies scheint aber keinen Einfluss auf die Bewertung der Karte zu haben.

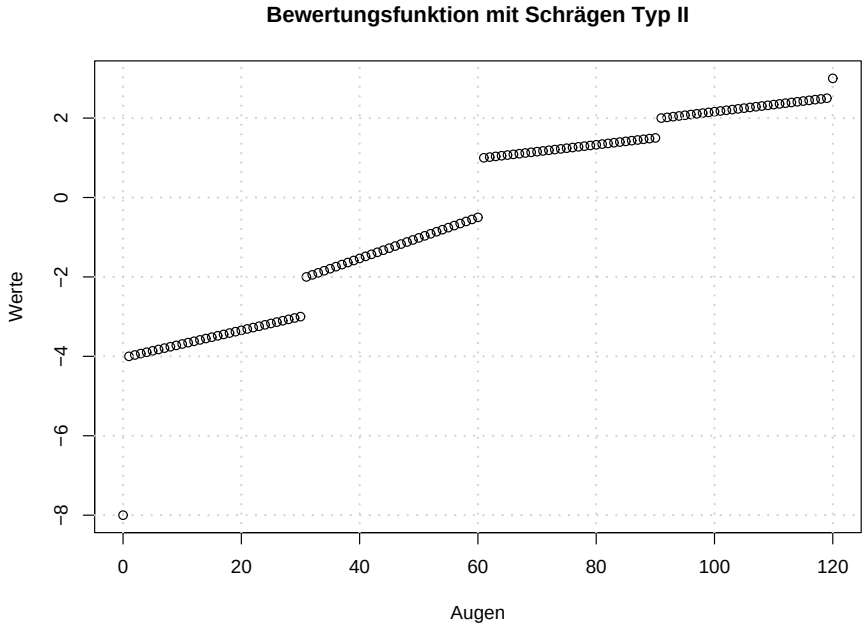

Abbildung 6.3: Bewertungsfunktion mit Schrägen Typ II

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$MCU_{optimiert}$	82.163	83.122	79.311	79.369	78.674	79.322	76.011	78.674
$MCU_{optimiert}^{TypII(25)}$	81.707	80.111	83.035	80.395	81.168	77.858	79.939	81.168
$MCU_{optimiert}$	83.797	80.897	81.073	79.152	78.996	78.252	78.373	76.771
$MCU_{optimiert}^{TypII(50)}$	82.314	82.131	80.291	79.516	81.619	78.716	77.905	84.219
$MCU_{optimiert}$	81.626	81.731	81.737	80.382	78.489	78.558	77.619	78.797
$MCU_{optimiert}^{TypII(75)}$	82.034	82.234	80.155	78.595	80.954	80.818	78.986	78.019

Spieler	T_1	T_2	T_3
$MCU_{optimiert}$	78.939	78.049	78.051
$MCU_{optimiert}^{TypII(25)}$	76.385	78.745	78.826
$MCU_{optimiert}$	78.752	77.365	76.438
$MCU_{optimiert}^{TypII(50)}$	78.892	81.431	81.156
$MCU_{optimiert}$	78.104	78.679	77.676
$MCU_{optimiert}^{TypII(75)}$	81.504	80.522	81.133

Tabelle 6.6: Spiele mit Turnierbewertung: 2x $MCU_{optimiert}$ vs. $MCU_{optimiert}^{TypII}$

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$MCU_{optimiert}$	83.890	85.375	82.189	80.259	78.465	72.903	77.314	80.810
$MCU_{optimiert}^{TypII(25)}$	80.651	79.791	79.483	78.392	79.220	80.090	78.232	76.316
$MCU_{optimiert}$	80.060	80.025	78.737	78.868	80.327	76.480	75.083	74.999
$MCU_{optimiert}^{TypII(50)}$	82.030	80.945	80.282	79.729	78.973	78.662	78.565	79.709
$MCU_{optimiert}$	81.601	81.292	75.118	76.885	78.202	77.961	77.903	75.590
$MCU_{optimiert}^{TypII(75)}$	82.056	81.000	81.284	80.711	79.778	77.784	78.571	79.047

Spieler	T_1	T_2	T_3
$MCU_{optimiert}$	74.085	76.762	76.203
$MCU_{optimiert}^{TypII(25)}$	78.211	77.740	76.853
$MCU_{optimiert}$	75.461	74.190	76.655
$MCU_{optimiert}^{TypII(50)}$	79.934	79.571	77.930
$MCU_{optimiert}$	79.240	75.459	77.573
$MCU_{optimiert}^{TypII(75)}$	78.468	79.942	77.867

Tabelle 6.7: Spiele mit Turnierbewertung: $MCU_{optimiert}$ vs. 2x $MCU_{optimiert}^{TypII}$

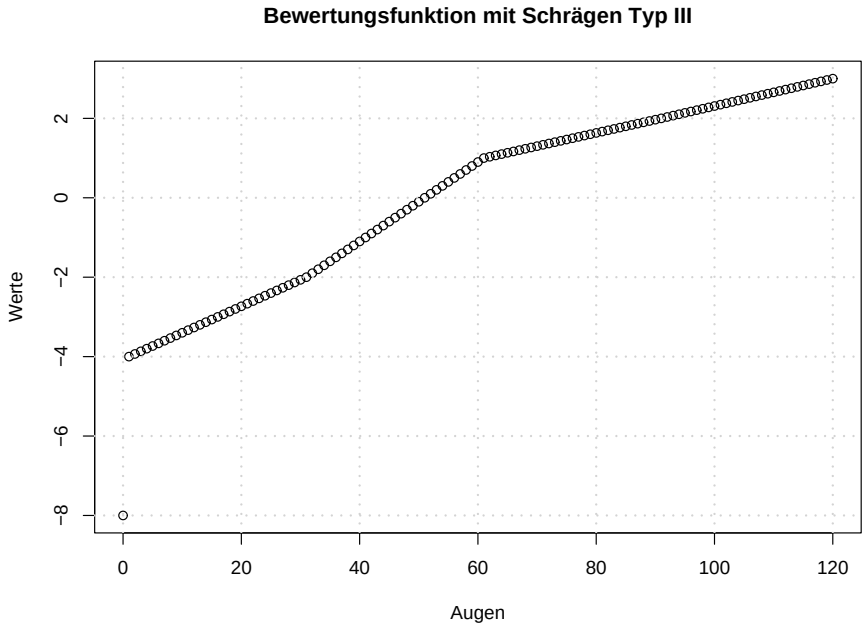

Abbildung 6.4: Bewertungsfunktion mit Schrägen Typ III

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$MCU_{optimiert}$	82.300	82.048	79.229	79.197	79.482	77.644	78.388	78.781
$MCU^{TypIII}_{optimiert}$	80.974	82.094	84.512	82.102	79.907	80.367	79.639	79.474

Spieler	T_1	T_2	T_3
$MCU_{optimiert}$	78.266	78.726	78.110
$MCU^{TypIII}_{optimiert}$	80.263	78.074	77.149

Tabelle 6.8: Spiele mit Turnierbewertung: 2x $MCU_{optimiert}$ vs. $MCU^{TypIII}_{optimiert}$

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$MCU_{optimiert}$	82.550	80.302	82.439	78.262	76.953	76.937	76.979	77.830
$MCU^{TypIII}_{optimiert}$	83.482	81.728	80.708	79.611	81.483	79.036	78.204	78.102

Spieler	T_1	T_2	T_3
$MCU_{optimiert}$	78.489	77.090	77.605
$MCU^{TypIII}_{optimiert}$	78.175	78.904	78.490

Tabelle 6.9: Spiele mit Turnierbewertung: $MCU_{optimiert}$ vs. 2x $MCU^{TypIII}_{optimiert}$

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$MCU_{optimiert}$	86.425	87.499	87.988	90.854	90.291	90.533	90.691	90.237
$MCU^{TypII}_{optimiert}$	84.710	85.388	88.500	88.296	90.796	91.816	90.410	91.462

Spieler	T_1	T_2	T_3
$MCU_{optimiert}$	91.187	90.931	90.070
$MCU^{TypII}_{optimiert}$	91.115	89.036	89.436

Tabelle 6.10: Nullspiele mit Turnierbewertung: 2x $MCU_{optimiert}$ vs. $MCU^{TypII}_{optimiert}$

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$MCU_{optimiert}$	84.463	86.983	90.463	89.194	91.962	90.173	88.607	91.057
$MCU^{TypII}_{optimiert}$	84.463	87.451	87.393	89.403	90.737	90.540	91.028	90.797

Spieler	T_1	T_2	T_3
$MCU_{optimiert}$	90.445	91.159	n.d.
$MCU^{TypII}_{optimiert}$	90.200	89.718	n.d.

Tabelle 6.11: Nullspiele mit Turnierbewertung: $MCU_{optimiert}$ vs. 2x $MCU^{TypII}_{optimiert}$

Kapitel 7

Vergleich zur Suche mit vollständiger Information

Dieses Kapitel vergleicht die Spielstärke des MCU-Spielers mit dem DDSS aus [Kup03]. Es dient zum Nachweis, dass der MCU-Spieler auch gegen Gegner besteht, deren Strategie er nicht perfekt kennt.

7.1 Beschreibung des Double Dummy Skat Solver

Der DDSS implementiert die bekannte Vorgehensweise, Spiele mit unvollständiger Information auf Spiele mit vollständiger Information abzubilden. Er erzeugt mittels Monte Carlo Simulation eine Kartenverteilung, die mit dem bisherigen Spielverlauf vereinbar ist. Dieses Spiel wird dann als Skatspiel mit vollständiger Information betrachtet. Dazu wird ein optimierter Suchalgorithmus verwendet der den Suchbaum sehr stark beschneiden kann. Die Simulation wird mehrfach durchgeführt und dann die Karte ausgespielt, durch deren Ausspiel die meisten Spiele gewonnen wurden.

Die Quellen des DDSS wurden für den Rahmen dieser Arbeit freundlicherweise vom Autor zur Verfügung gestellt. Zunächst musste die Schnittstelle für die KI-Spieler implementiert werden. Der DDSS wurde nur von der Klasse `AIPlayer` abgeleitet, da er alle Funktionen eines Skatspielers selbst schon umgesetzt hat. Trotzdem übernimmt die Turnierumgebung das Reizen, Drücken und die Spielansagen. Der Autor des DDSS beurteilt die Fähigkeiten des DDSS beim Reizen und Drücken auch noch nicht als sehr ausgereift. Neben der Anzahl der Simulationen kann auch beim DDSS eine Schranke für die maximale Bedenkzeit angegeben werden.

7.2 Experimente

In [Kup03] sind sechs Spiele XSkat vs. DDSS aufgelistet. Der DDSS gewinnt zwar alle Spiele als Alleinspieler, eine Aussage über die Spielstärke ist bei so wenigen Spieler aber nicht möglich. Deshalb wurden zunächst die 1.000 Kartenverteilungen in zwei Skatrunden mit XSkat als Gegner durchgeführt. Die Tabelle 7.1 zeigt die Ergebnisse. Pro Skatrunde kann zurzeit nur eine Instanz des DDSS mitspielen. Die interne Struktur des DDSS ist noch nicht dafür ausgelegt.

Spieler	S_5	S_{10}	S_{25}	S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
$XSkat$	92.264	85.477	78.827	73.901	72.270	70.057	n.d.	69.544
$DDSS$	52.986	66.976	76.835	80.853	85.939	85.271	n.d.	87.530

Spieler	T_1	T_2	T_3
$XSkat$	73.300	72.740	71.100
$DDSS$	81.834	82.524	85.666

Tabelle 7.1: Spiele mit Turnierbewertung: 2x $XSkat$ vs. $DDSS$

Die Ergebnisse zeigen, dass der DDSS erst ab 50 Simulationen besser spielt als XSkat. Außerdem schafft der DDSS es nicht, während kurzer Bedenkzeiten genügend Simulationen durchzuführen. Bei einer Sekunde Bedenkzeit kann der DDSS maximal 200 Simulationen betrachten. Im Mittel untersuchte er 100 Simulationen pro Spielzug, aber gerade in den ersten Stichen wurde diese Anzahl deutlich unterschritten. Dies führt zu der schlechteren Spielstärke bei den zeitbegrenzten Skatrunden, weil in den ersten Stichen meist auch der Grundstein für die Spielstrategie des gesamten Spieles gelegt wird. Insgesamt erreichte der DDSS im Spiel gegen XSkat nicht die Punkte des MCU-Spielers, als dieser gegen XSkat spielte.

In weiteren Experimenten trat der MCU-Spieler mit optimierter Simulation der Kartenverteilung gegen den DDSS an. Bei beiden Spielern wurden 100 simulierten Welten pro Spielzug generiert. Dabei zeigte sich, dass der MCU-Spieler bei 100 Simulationen nicht gegen den DDSS bestehen konnte. Dies lässt sich durch die geringere Datenbasis erklären, welche dem MCU-Spieler zur Verfügung steht. Der DDSS kann neben dem Endergebnis der Spiele auch in die Karten der Gegner sehen und die Spielzüge der Gegner analysieren. Bei Verdoppelung der Simulationen für den MCU-Spieler konnte dieser mit dem DDSS mithalten.

7.3 Weitere Verbesserung des Monte Carlo Spielers

Die Erhöhung der Simulationen beim MCU-Spieler ist gegenüber dem DDSS unfair, weil der MCU-Spieler damit eine größere Stichprobe aus der Menge der möglichen Welten zieht. Deshalb wurde versucht, die Spielstärke des MCU-Spielers bei gleicher Anzahl der Simulationen zu erhöhen.

Die Idee ist das Wiederholen der Spiele für eine Karte innerhalb einer Simulation. Bisher wurde für jede Karte in jeder Simulation nur ein Spiel durchgeführt. Da die Heuristiken von XSkat an einigen Stellen auch Zufallszahlen zur Auswahl einer Karte verwenden, kann es passieren, dass XSkat gerade in dem einen untersuchten Spiel nicht die optimale Karte spielt. Deshalb wurde die Simulation für den MCU-Spieler erweitert, damit in jeder Simulation pro Karte mehrere Spiele durchgeführt werden können. Die zusätzlichen Spiele wurden nicht gesondert bewertet. Für die Auswertung der in den Spielen erreichten Augenzahlen wurde die Bewertungsfunktion vom Typ II mit einem Anstieg auf 50 % der Stufenhöhe verwendet. In der Tabelle 7.2 sind die Ergebnisse der Skatrunden aufgelistet.

Spieler	S_{100}
$MCU^{TypII(50)}_{opt,wiederh(50)}$	77.415
$DDSS$	78.155

Tabelle 7.2: Spiele mit Turnierbewertung: 2x $MCU^{TypII(50)}_{opt,wiederh(50)}$ vs. $DDSS$

Nach der Turnierbewertung ist ein kleiner Vorteil für den DDSS auszumachen. Betrachtet man aber die erreichten Punktzahlen für die Spieler einzeln, ergibt sich ein anderes Bild. Die Punkte der beiden MCU-Spieler wurden gemittelt. Die Einzelergebnisse sind 79.803 und 75.027 Punkte. Auch bei den Punkten nach der normalen Abrechnung ohne Zusatzpunkte für gewonnene Spiele liegen beide MCU-Spieler vorn (32083 und 30247 vs. 29875). Der DDSS konnte insgesamt mehr Spiele gewinnen als die beiden MCU-Spieler (852 und 831 vs. 856). Er hat also mehr kleine Farbspiele gewonnen. Diesen Verlust konnten die MCU-Spieler durch gewonnene Grandspiele und durch das Erreichen höherer Gewinnstufen wieder wett machen. Interessant wäre nun noch ein Vergleich mit einer Skatrunde 2x DDSS vs. MCU-Spieler. Zurzeit kann der DDSS wegen seiner internen Struktur nur einmal pro Skatrunde vorkommen. Deshalb konnte diese Skatrunde nicht durchgeführt werden.

Kapitel 8

Zusammenfassung und Ausblick

Im letzten Kapitel werden noch einmal alle Ergebnisse zusammengefasst. Im Rahmen dieser Arbeit konnten nicht alle Aspekte des Skatspiels beleuchtet werden. Es wurde sich nur auf das eigentliche Spielen beschränkt. Das Reizen und Drücken bietet aber noch weitere Felder, in denen die Monte Carlo Simulation eingesetzt werden kann. Der Ausblick soll einige Ideen für die Zukunft darstellen.

8.1 Bewertung der Ergebnisse

Die Auswertungen der Skatrunden zeigen, dass durch Monte Carlo Simulation mit unvollständiger Information die Spielstärke von Skatheuristiken erhöht werden kann. Der Einfluss von Zufallsentscheidungen in die Heuristik wurde durch das Wiederholen der Spiele für eine Karte innerhalb einer Simulation herausgerechnet.

Im Vergleich mit der Monte Carlo Simulation mit vollständiger Information konnte eine gleichwertige Spielstärke erreicht werden. Ein Vorteil der Verwendung von Heuristiken ist ihre Schnelligkeit. Der Aufbau und die Beschneidung von Suchbäumen sowie die anschließenden Suche nach den besten Spielzügen dauert im Durchschnitt zehnmal länger. Das lässt Spielraum für weitere Optimierungen oder mehr Simulationen bei gleicher Bedenkzeit.

Die meisten der untersuchten Bewertungsfunktionen haben kaum zur Steigerung der Spielstärke beigetragen. Nur die Bewertungsfunktion vom Typ II konnte eine spürbare Verbesserung der Spielstärke erreichen.

8.2 Weitere Untersuchungen

Es gibt sowohl Ideen für die Optimierung der Berechnungsgeschwindigkeit als auch für die Verbesserung der Spielstärke. Die Spielfunktionen Reizen und Drücken können ebenfalls mit Monte Carlo Simulation mit unvollständiger Information umgesetzt werden.

8.2.1 Verbesserungen der Geschwindigkeit und Spielstärke

Die beschränkte Zeit für einen Spielzug könnte besser genutzt werden, wenn man zunächst mit einigen wenigen Simulationen die Karten ermittelt, bei denen sich eine genauere Betrachtung lohnt. Karten, die hier jedesmal zu einem Spielverlust führen, brauchen nicht eingehender untersucht werden. Bei weiteren Simulationen werden nur noch die vielversprechendsten Karten analysiert.

Der mehrstufige Einsatz von Monte Carlo Simulation mit unvollständiger Information ist eine weitere Idee. Bisher wurde ja ein Spiel immer mit den Heuristiken zu Ende gespielt. Wenn man mehrere Spielzüge in einem Spiel durch Monte Carlo Simulation berechnet, könnte die Unzulänglichkeiten der Heuristiken noch weiter beseitigen werden. Die Ersatz der Heuristiken von XSkat durch stärker spielende Heuristiken führt auch zu einer Verbesserung der Spielstärke. Ein Skatspieler könnte durch Spielen in der Turnierumgebung Heuristiken für Skat lernen und diese dann in den Monte Carlo Simulationen verwenden.

Die Simulation der Kartenverteilung könnte in Zukunft auch auf unsichere Informationen zurückgreifen. So kann man zum Beispiel mit ziemlicher Sicherheit sagen, dass ein Spieler der als Hinterhand einen König abwirft, das Ass oder die Zehn der gleichen Farbe nicht mehr hat. Je mehr Informationen oder Vermutungen für die Simulation der Kartenverteilung verwendet werden, desto geringer ist die Varianz in den Simulationen. Damit benötigt man auch weniger Simulationen, um eine gute Stichprobe aller möglichen Welten zu erhalten.

Schließlich wäre zu klären, ob es andere Bewertungsfunktionen gibt, welche bessere Ergebnisse liefern als die hier vorgestellten. Vielleicht gibt es für die verschiedenen Verteilungen der Augenzahlen spezielle Funktionen, die eine bessere Aussage über das Endergebnis eines simulierten Spieles zulassen.

8.2.2 Monte Carlo Simulation beim Reizen

Beim Reizen muss der Spieler anhand seiner Karten die möglichen Spiele ermitteln, die er gewinnen kann. Mit Monte Carlo Simulation könnte man alle möglichen Spiele für verschiedene Welten durchspielen und ermitteln, ob das Spiel gewonnen werden kann

oder nicht. Zur Bewertung kann einerseits die Anzahl der gewonnenen Spiele als auch wieder die erreichte mittlere Augenzahl mit ihrer Standardabweichung verwendet werden.

Mit diesem Ansatz könnte man besser als XSkat reizen. Damit macht der Spieler mehr Spiele und kann ein höheres Endergebnis erzielen, falls er die gemachten Spiele auch gewinnt.

8.2.3 Monte Carlo Simulation beim Drücken

Beim Drücken stellt sich zunächst die Frage, ob der Spieler überhaupt in den Skat sehen soll. Wenn ein Handspiel gewonnen werden kann, sollte es auch gespielt werden, weil es mehr Punkte einbringt. Aber auch wenn der Spieler den Skat aufnimmt, kann durch Monte Carlo Simulation ermittelt werden, welche Karten in den verschiedenen Welten am ehesten gedrückt werden sollten. Dazu würde man für das Handspiel und alle Kombinationen von gedrückten Karten alle möglichen Spiele ausprobieren und die Ergebnisse vergleichen.

Anhang A

Ein Herzspiel

In diesem Anhang soll exemplarisch ein Herzspiel vorgestellt werden, das mit allen drei KI-Spielern durchgeführt wurde. Anhand dieses Beispiels können sehr schön die verschiedenen Strategien und Fehler der Spieler aufgezeigt und der Einfluss auf des Spielergebnis dargestellt werden. Die Kartenverteilung nach dem Reizen und Drücken sah folgendermaßen aus:

Spieler	Karten			
Spieler 0	♢AK7	♡K7	♠AK9	♣KQ
Spieler 1	♢TQ8	♡ATJ98	♠J	♣8
Spieler 2	♢J9	♡Q	♠TQ7	♣ATJ7
Skat	♠8	♣9		

Die Mittelhand (Spieler 1) gewann das Reizen mit einem Gebot von 18 Punkten. Die MCU-Spieler und der DDSS-Spieler durften 100 mögliche Kartenverteilungen simulieren. Der MCU-Spieler spielte in jeder Welt bei jeder Karte zusätzlich 50 Einzelspiele. Als Bewertungsfunktion nutzte der MCU-Spieler die Bewertungsfunktion vom Typ II mit einem Anstieg auf 50 % der nächsten Stufe.

Im Folgenden sind drei Spielverläufe aufgelistet. In jedem Spiel ist der Spieler 1 der Alleinspieler. Die unterstrichene Karte ist die Gewinnerkarte. Beim MCU-Spieler wird an einigen Stellen zusätzlich eine Karte in Klammern angegeben. An diesen Stellen hat der MCU-Spieler durch seine Simulationen eine bessere Karte als XSkat gefunden. Die Karte in den Klammern ist diejenige, welche XSkat in dieser Situation gespielt hätte.

Stich	Spieler 0 ($XSkat$)	Spieler 1 ($DDSS$)	Spieler 2 ($MCU^{TypII(50)}_{opt/rep(50)}$)
1	♢A	♢8	♢9
2	♠A	♡A	♠7
3	♡7	♡J	♡Q (♣J)
4	♡K	♡9	♢J
5	♠9	♣8	♠Q (♣A)
6	♣Q	♠J	♣A
7	♢K	♡T	♣J
8	♣K	♡8	♣T (♠T)
9	♢7	♢T	♣7
10	♠K	♢Q	♠T

Der DDSS gewinnt das Spiel in mit 84 Augen. Er versucht zeitig Trumpf zu ziehen, wobei er mit hohen Trumpfkarten beginnt. Im 7. Stich geht die ♡T verloren, was den DDSS wahrscheinlich die Gewinnstufe Schneider kostet. Hätte er an dieser Stelle die ♡8 gespielt, wäre der MCU-Spieler gezwungen gewesen, seinen ♣J zu spielen. Der MCU-Spieler verzichtet im 3. Stich auf den Stichgewinn und wirft die ♡Q ab, anstatt wie XSkat vorschlägt, den ♣J zu spielen. Damit gelingt es ihm die ♡T im 7. Stich zu stechen.

Stich	Spieler 0 ($XSkat$)	Spieler 1 ($MCU^{TypII(50)}_{opt/rep(50)}$)	Spieler 2 ($DDSS$)
1	♢A	♢8	♢9
2	♠A	♡8	♠T
3	♡K	♡9 (♡J)	♡Q
4	♠9	♣8	♠Q
5	♣Q	♡A	♣A
6	♡7	♠J	♣J
7	♢7	♡J	♢J
8	♠K	♡T (♢T)	♣T
9	♢K	♢T	♣7
10	♣K	♢Q	♠7

Der MCU-Spieler gewinnt das Spiel mit 95 Augen. Er zieht auch so schnell wie möglich Trumpfkarten, nimmt dazu aber kleine Karten. Im Mittel- und Endspiel zahlt sich diese Taktik aus. Ab dem 7. Stich gewinnt er alle restlichen Stiche. Alle hohen Trumpfkarten gewinnen einen Stich. Der MCU-Spieler kann die beiden anderen Spieler Schneider spielen.

Stich	Spieler 0 ($MCU^{TypII(50)}_{opt/rep(50)}$)	Spieler 1 ($XSkat$)	Spieler 2 ($DDSS$)
1	♠A (♢A)	♡A	♠Q
2	♡K	♡J	♣J
3	♣K (♣Q)	♣8	♣A
4	♣Q	♡T	♣T
5	♡7	♠J	♢J
6	♢7 (♢A)	♡9	♡Q
7	♠9 (♠K)	♡8	♣7
8	♢K (♢A)	♢8	♢9
9	♠K (♢A)	♢Q	♠T
10	♢A	♢T	♠7

Der XSkat-Spieler verliert das Spiel mit 52 Augen. Auch XSkat zieht von Beginn an Trumpf mit hohen Trumpfkarten. Der MCU-Spieler hält auffallend lange das ♢A zurück. Er kann es im letzten Stich buttern.

Literaturverzeichnis

[BMvL96] Blair, J. R. S.; Mutchler, D.; Lent, M. v.: Perfect Recall and Pruning in Games with Imperfect Information. *Computational Intelligence*, Band 12, Nr. 4, S. 131–154, 1996.

[FB98] Frank, I.; Basin, D.: Search in Games with Incomplete Information: A Case Study using Bridge Card Play. *Artificial Intelligence*, Band 100, S. 87–123, 1998.

[FBM98] Frank, I.; Basin, D.; Matsubara, H.: Finding Optimal Strategies for Imperfect Information Games. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence, (AAAI-98)*, S. 500–507, 1998.

[Ger04] Gerhardt, G. `http://www.xskat.de/xskat-latest-de.html`, 2004.

[Gin99] Ginsberg, M. L.: GIB: Steps Toward an Expert-Level Bridge-Playing Program. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, S. 584–589, 1999.

[IW02] ISPA-World. `http://www.skat.com/skatordnung.html`, 2002.

[Kup03] Kupferschmid, S.: Entwicklung eines Double Dummy Skat Solvers mit einer Anwendung für verdeckte Spiele. Diplomarbeit, Albert-Ludwigs-Universität Freiburg, Fakultät für angewandte Wissenschaften, Institut für Informatik, 2003.

[Lev89] Levy, D.: The Million Pound Bridge Program. In *Heuristic Programming in Artificial Intelligence*, S. 95–103. Ellis Horwood, 1989.

[RN04] Russell, S.; Norvig, P.: *Künstliche Intelligenz – Ein moderner Ansatz*. Prentice Hall, 2. Auflage, 2004.

[Sch01] Schaeffer, J.: A Gamut of Games. *AI Magazine*, Band 22, Nr. 3, S. 29–46, 2001.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Groitzsch, den 24. Juli 2007

Jan Schäfer

Thesen

1. Mit Monte Carlo Simulation kann die Spielstärke von Heuristiken erhöht werden.
2. Mit Heuristiken kann ein Spiel mit unvollständiger Information mittels Monte Carlo Simulation betrachtet werden, ohne die Eigenschaft der unvollständigen Information zu verlieren.
3. Programme, die Monte Carlo Simulation mit unvollständiger Simulation zur Berechnung ihrer Strategie verwenden, spielen auf dem gleichen Niveau wie Programme, welche die Spiele mit unvollständiger Information auf Spiele mit vollständiger Information abbilden.
4. Die Monte Carlo Simulation mit unvollständiger Information sind weniger rechenaufwändig.