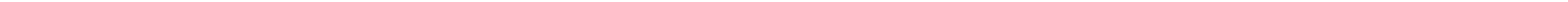
Ein Framework zur Entwicklung und Evaluation intelligenter Pokeragenten

Bachelorarbeit

Markus Zopf
Knowledge Engineering Group
Technische Universität Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT



Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht worden.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsinstanz vorgelegen.

Darmstadt, den 30.12.2010

Markus Zopf

Abstract

Poker bietet für die Forschung im Bereich der künstlichen Intelligenz zahlreiche neue Herausforderungen. Eine dieser Herausforderungen ist die objektive Einschätzung der Spielstärke von Pokerspielern und insbesondere automatisierten Pokeragenten. Um eine Plattform zu bieten, auf der sich verschiedene so genannte Bots messen können, wird seit 2006 die ‚Annual Computer Poker Competition (ACPC)' veranstaltet.

Ziel dieser Bachelorarbeit ist es ein Grundgerüst von Funktionalität in einem zum ACPC kompatiblen Framework bereit zu stellen um den Fokus bei der Bot Entwicklung möglichst schnell auf die wesentlichen und kreativen Aspekte wie z.B. die Gegnermodellierung oder ausgefeilte Strategien legen zu können. Hierbei wird insbesondere der Spielzustand leicht auswertbar modelliert, performante Algorithmen zur Analyse der aktuellen Situation implementiert und eine einfache Möglichkeit der Evaluierung entwickelter Bots geschaffen.

Inhaltsverzeichnis

1 Einleitung 1

1.1 Motivation 1

1.2 Die ‚Annual Computer Poker Competition' 3

1.3 Ziele der Bachelorarbeit 5

1.4 Aufbau der Arbeit 7

2 Architektur des Frameworks 9

2.1 Einleitung 9

2.2 Framework-Übersicht 9

2.3 package-Übersicht 10

2.3.1 pokertud.cards 10

2.3.2 pokertud.gamestate 10

2.3.3 pokertud.clients 10

2.3.4 pokertud.server 11

2.3.5 pokertud.message 11

2.3.6 pokertud.divat 12

2.3.7 pokertud.uct 12

2.3.8 pokertud.util 12

2.4 Ablauf Initialisierung & Reaktion auf Updates 12

3 Message-Subsystem 14

3.1 Einleitung 14

3.2 Nachrichten 14

3.3 Initialisierung 15

3.4 Nachrichtenübermittlung 18

4 Zustandsrepräsentierung 20

4.1 Einleitung 20

4.2 Der GameStateString 20

4.3 GameState 21

5 Effiziente Handevaluation 23

5.1 Einleitung 23

5.2 RankCode 24

5.3 HandStrengthCode 25

5.3.1 Einleitung 25

5.3.2 Codierung von ‚Steve Brecher' 26

5.3.3 Neue Codierung 26

5.4 Auswertung und Optimierung 28

6 Metriken 30

6.1 Einleitung 30

6.2 Immediate Handrank 30

6.3 Seven-Card Handrank 32
6.4 Effective Handrank 33
6.5 All-In Equity 34
6.6 Pot Equity 34
6.7 Roll-Out Equity 34
6.8 Zusammenfassung 35

7 UCT-Simulation mit Opponent Modeling 36

7.1 Einleitung 36
7.2 Funktionsweise 36
7.3 Implementierung und Beispiel 38

8 Benutzungsbeispiel 40

9 Fazit 44

9.1 Zusammenfassung 44
9.2 Ausblick 45

Abbildungsverzeichnis 48

Literaturverzeichnis 49

Glossar 50

1 Einleitung

1.1 Motivation

Seit einiger Zeit beschäftigt man sich nun schon in der Forschung rund um künstliche Intelligenz mit vollständig beobachtbaren und deterministischen Spielen wie ‚Vier Gewinnt', Schach oder Go. Diese Spiele bieten exzellente Vorrausetzungen, um mit traditionellen Ansätzen wie z.B. Breiten- oder Tiefensuche gute Gewinnstrategien berechnen zu können. So ist ‚Vier Gewinnt' bereits 1988 gelöst worden wie in [A88] beschrieben wird. Spiele wie Schach oder Go mit größerem Zustandsraum konnten bislang zwar noch nicht stark gelöst werden, jedoch existieren auch hier bereits Computerprogramme, die auf Weltklasseniveau spielen bzw. durch Menschen kaum noch besiegt werden können. Verantwortlich hierfür ist nicht zuletzt der Minimax-Algorithmus, der für die genannten Spiele mit Hilfe von einer ausreichend großen Rechenkapazität einen optimalen Zug (bei optimaler Spielweise des Gegners) bestimmen kann. Betrachtet man Spiele mit einer Wahrscheinlichkeitskomponente und vollständig beobachtbaren Zuständen erhöht sich die Anzahl der Pfade in einem Suchbaum in der Regel rapide. Backgammon liefert hierfür ein gutes Beispiel. Jedoch wurde auch dieses Spiel mit Hilfe der künstlichen Intelligenz und dem Einsatz von neuronalen Netzten in dem Programm ‚TD-Gammon'[1] durch Computer beherrschbar. Spielte TD-Gammon noch auf dem gleichen Niveau wie Experten in diesem Spiel, sind aktuelle Bots inzwischen besser als die weltbesten menschlichen Spieler. Ebenfalls ein Indiz für die Stärke von Agenten in dieser Art von Spielen ist die Tatsache, dass die Spielweisen von Bots wie TD-Gammon untersucht und die Analysefähigkeiten dieser Programme genutzt wurden, um die bestehenden Backgammon-Strategien zu erweitern.

Spiele der eben beschriebenen Art mit vollständig beobachtbaren Spielsituationen haben also etwas an Reiz für die aktuelle Forschung verloren, da man hier schon den vermeintlich interessantesten Gegner, den Menschen, besiegt hat. Die entwickelten Bots werden mit der Verbesserung der Computertechnik im Bereich der Geschwindigkeit und der Speichermöglichkeiten fast automatisch immer besser, da so immer größere Teile des Suchbaums ausgewertet werden können. So ist es nur eine Frage der Zeit bis bei stetig stärker werdenden Schachprogrammen der Zeitpunkt gekommen ist, an dem sie nahezu unbesiegbar werden. Also müssen neue Herausforderungen gefunden werden umso das Spektrum an abgedeckten Problemfeldern zu erweitern. Eine große Menge an völlig neuen Aspekten, die es so in den bereits erwähnten Spielen nie geben wird, bietet Poker insbesondere mit der Variante ‚Texas Hold'em'.

Eines der Hauptprobleme bei Poker im Unterschied zu beispielsweise Schach ist der nicht vollständig beobachtbare Spielzustand. Die beiden Holecards jedes Gegners sind für den Spieler verdeckt. Somit kann maximal geschätzt werden, wie der aktuelle Spielzustand genau aussieht. Ein weiter, nicht so offensichtlicher Aspekt der unvollständigen Information sind die Gegner selbst. Es ist nahezu unmöglich eine verlässliche Aussage darüber zu treffen, wie sich ein bestimmter Gegner verhalten wird. Spielt der Gegner sehr aggressiv? Blufft er oft? Welche Strategie sieht er in einer bestimmten Situation als optimal an? Alles Fragen, die von größter Bedeutung für die Auswahl eines guten Zuges sind, die aber wie gesagt nicht objektiv beantwortet werden können. Eine direkte Folge dieser fehlenden Möglichkeit bestimmte Aspekte genau und insbesondere objektiv zu bestimmen ist die Tatsache, dass ein Zug nur schwer eindeutig bewertet werden kann. Hat ein Spieler beispielsweise die Erfahrung gemacht, dass ein konkreter Gegner während einer bestimmten Situation schlechte Hände immer foldet, ein anderer Spieler aber der Meinung ist das dieser Gegner solche Gelegenheiten oft zum Bluffen benutzt, so folgen daraus möglicherweise 2 vollkommen verschiedene Ansichten über den nächsten zu tätigenden Zug. Meist können diese Argumente die für oder gegen einen gewählten Zug sprechen kaum formal beschrieben werden. Es bleibt also bei informeller, verbaler Argumentation, die

[1] http://en.wikipedia.org/wiki/TD-Gammon

sich schwer in Regeln fassen lässt und es so kaum ermöglicht von intelligenten Agenten genutzt zu werden.

Ein weiterer Grund, warum Poker einige neue zu betrachtende Facetten bietet, ist die Spieleranzahl. Backgammon, Schach und Go sind 2-Spieler Spiele. Die Anzahl an Gegnern ist also stets auf einen beschränkt, woraus folgt, dass man ca. die Hälfte aller Züge eines Spiels machen wird. Nicht umsonst spricht man beim Schach auch von einem Halbzug, wenn ein Spieler eine Figur bewegt hat. Das heißt man selbst bestimmt in einem potentiellen Spielbaum an ca. der Hälfte aller Knoten die einzuschlagende Richtung, indem man einen bestimmten Zug macht. Verringert wird dieser Anteil bereits in Backgammon durch das Vorhandensein von Zufallsereignissen. Es ist dort schon nicht mehr so einfach möglich einen bestimmten Teil des Baums als gewünschten Zielbereich zu erreichen. Bei Poker ist diese Möglichkeit, das Spiel teilweise zu lenken, allerdings noch viel stärker durch die größere Spieleranzahl eingeschränkt, da wesentlich weniger Knoten des Spielbaums unter eigener Kontrolle stehen. Lässt man Zufallsereignisse zunächst außer Acht, so bestimmt man den Pfad eines Spiels im Spielbaum bei einer Pokerrunde mit insgesamt 10 Spielern durch einen eigenen Zug nur noch zu 10% im Vergleich zu 50% in 2-Spieler Spielen. Als Beispiel kann man hier die oft auftretende Situation nennen, in der entschieden werden muss, ob sich ein Call in einer Runde lohnt um auf das Dealen von neuen Karten zu warten, um dann seine Lage neu zu evaluieren. Spielt man nur gegen einen Gegner kann man sich der Kosten für diesen Call absolut sicher sein, da man mit dem Call die aktuelle Spielrunde beendet und die nächste Gemeinschaftskarte ohne weitere Aktionsmöglichkeiten des Gegners aufgedeckt wird. Sind aber noch eine Vielzahl von Gegner in der aktuellen Runde beteiligt so kann jeder von denen, die noch nach Hero dran sind, den Einsatz weiter erhöhen und es wird teurer die nächste Karte zu sehen. Hieran sieht man, dass es viel leichter ist den Spielfluss eines 2-Spieler Spiels zu kontrollieren als es bei einem Mehrspieler Spiel der Fall ist.

Nur partiell beobachtbare Spielsituationen, schwer objektiv beurteilbare Spielzüge und mehrere Gegner begründen neben Zufallsereignissen die hohe Komplexität bei der Entwicklung von Pokeragenten. Jedoch ist es nicht so, dass diese neuen Probleme getrennt voneinander betrachtet werden und nach der Lösung der Teilprobleme zu einer Gesamtlösung nach dem ‚Separate and Conquer'-Verfahren zusammengesetzt werden können. Denn es ist so, dass das Gesamtproblem nicht nur die Summe der Teilprobleme ist, sondern sich die Teilprobleme vielmehr gegenseitig beeinflussen und deren Auswirkungen durch diese Wechselwirkung potenziert werden. So ist es nicht möglich das Problem des Mehrspieler-Spiels zu lösen ohne dabei auftretende Zufallsereignisse zu betrachten. Genauso wenig macht es Sinn zu versuchen die Gegner möglichst exakt zu beurteilen ohne gut abschätzen zu können in welchem Spielzustand sich das Spiel gerade befindet.

Neben der Schwierigkeit gute Pokeragenten zu entwickeln gibt es aber noch ein weiteres, nicht weniger bedeutendes Problem, welches sich nahtlos an die Entwicklung anschließt. Um festzustellen ob Fortschritte bei einer neuen Version eines bestehenden Pokerbots gemacht wurden oder um die Leistungsfähigkeit eines neuen Konzepts zu überprüfen müssen die entwickelten Agenten ausgiebig evaluiert werden. Diese Evaluation gestaltet sich allerdings nicht leicht. Gründe hierfür sind ähnlich wie bei der Entwicklung der Bots die hohe Komplexität und Vielfalt der Spielsituationen. Es lassen sich nur schwer Regeln finden, in beim Pokern immer gelten. Situationen lassen sich kaum zu Äquivalenzklassen zusammenfassen ohne dabei zu viel zu generalisieren. Ist in einer bestimmten Situation eine Spielweise offensichtlich korrekt kann die Änderung eines kleinen Details zu einer vollkommen anderen Schlussfolgerung führen. Es ist also nur schwer (automatisiert) zu beurteilen, ob eine Situation bestimmten Kriterien genügt und daraus ein eindeutig korrekter Zug abgeleitet werden kann.

Erschwerend kommt hinzu, dass wie bereits weiter oben beschrieben, selbst unter Experten oft Uneinigkeit über die korrekte Verhaltensweise in bestimmten Situationen herrscht. Genauso wie es schwer ist, einem Bot bei der Entwicklung beizubringen was ein guter Zug ist, ist es ebenfalls schwer

zu beurteilen ob ein konkreter Zug sinnvoll ist. Nun könnte man bei der Evaluierung auf die Idee kommen das Problem der unvollständigen Informationen damit zu lösen indem man einfach die komplette Situation betrachtet. Denn bei der nachträglichen Evaluation hat man, anders als während des Spiels, die Möglichkeit z.B. die Karten der anderen Spieler zu sehen. Diese Informationen lassen aber schnell einen sehr unrealistischen Eindruck des Bots, insbesondere in einem Spiel mit mehreren Gegnern, entstehen. Details hierzu sind in [BK06] nachzulesen, wo versucht wurde ein solches Bewertungssystem mit vollständiger Information für Heads-Up Poker (also das Spiel gegen nur genau einen Gegner) zu entwickeln. Schon bei einem 2-Spieler Spiel treten dabei einige Schwierigkeiten auf, die in einem Mehrspieler-Spiel kaum noch ohne eine sehr gute Gegnermodellierung gelöst werden können. Bei einem Bewertungssystem, welches nicht den vollständigen Spielzustand verwendet, entstehen also ähnliche Probleme wie bei der Entwicklung des Bots und eine Nutzung aller Informationen ist wie beschrieben problematisch. Es lässt sich also kein einfacher Ausweg aus den mangelnden objektiven Beurteilungsmöglichkeiten finden.

Ein weiteres Problem bei der Evaluierung lässt sich durch folgendes Beispiel erläutern: Spielt ein Bot A besser als ein Bot B muss dies nicht zur Folge haben, dass Bot A grundsätzlich besser ist als Bot B. Es ist durchaus möglich, dass Bot B eine Schwäche hat, die von Bot A zufälligerweise überdurchschnittlich stark ausgenutzt wird und A deshalb so stark wirkt. Es kann aber eine Vielzahl anderer Bots geben, die diese Schwäche nicht haben und gegen die Bot A unterlegen ist. Falls diese anderen Bots die Schwäche von B nicht ausnutzen können, weil sie entweder nicht darauf programmiert wurden oder weil sie einfach diese Schwäche noch nicht erkannt haben können sie im Vergleich zu Bot B schlechter sein. Es kann also keine absolute Größe für die Stärke eines Bots angegeben werden. Die Stärke eines Bots ist immer relativ zur Spielweise eines anderen Bots oder einer Menge von anderen Bots zu bewerten. Ein anderer Aspekt folgt aus der Tatsache, dass gegen mehrere Spieler gleichzeitig gespielt werden muss. Sitzen alle eben genannten Bots an einem Tisch und spielen gegeneinander nutzt A vielleicht seine Fähigkeit um B zu dominieren. Es kann aber durchaus vorkommen, dass A dadurch anderen Bots eine große Angriffsfläche bietet und die von B gewonnenen Chips sofort wieder an andere Gegner verspielt. Ein letzter Punkt zu diesem Thema betrifft die fehlende Transitivität der Botstärke, die auch schon in einem 2-Spieler Turnier zu erkennen ist. Schlägt Bot A einen Bot B, der wiederum Bot C dominiert, folgt daraus nicht das Bot A auch gegen C gewinnt.

1.2 Die ‚Annual Computer Poker Competition'

Um das Problem der schlechten Evaluierbarkeit, die in der Einleitung beschrieben wurde, abzuschwächen wird vom Department of Computer Science[2] der University of Alberta[3] die sogenannte Annual Computer Poker Competition veranstaltet. Dieser seit 2006 jährlich stattfindende Wettkampf hat laut der Veranstalter den Sinn „*to benefit artificial intelligence by promoting, aiding, and evaluating research in the challenging problems presented by the wide variety of poker games*“ [ACPA]. Es ist also insbesondere ein Ziel der Challenge eine bessere Evaluierung von entwickelten Pokeragenten zu ermöglichen. Erreicht wird dies durch die Durchführung von verschiedenen Turnieren, in denen die Bots gegeneinander antreten.

Es wird bei der Competition zunächst zwischen den beiden verschiedenen Texas Hold‘em Varianten ‚No Limit Texas Hold‘em‘ und ‚Fixed Limit Texas Hold‘em‘ unterschieden. Während die Variante No Limit nur als Heads-Up (also mit 2 Spielern) durchgeführt wird, wird Fixed Limit sowohl als Heads-Up als auch als 3-Spieler Version, bei der jeder Spieler 2 Gegner hat, gespielt. Bei den beiden Heads-Up Spielen werden pro Match 3000 Hände gespielt. Die maximale Zeit, die benutzt werden darf, entspricht der Anzahl Hände multipliziert mit sieben. D.h. im Schnitt dürfen 7 Sekunden pro Hand verwendet werden, die maximale Gesamtbedenkzeit eines Bots beträgt dann bei 3000 Händen 21000

[2] http://webdocs.cs.ualberta.ca/
[3] http://www.ualberta.ca/

Sekunden bzw. 350 Minuten. Nutzen beide Spieler ihre volle Bedenkzeit aus kann ein Match über 11,5 Stunden dauern. Zudem dürfen für eine einzelne Hand nicht mehr als 10 Minuten Rechenzeit beansprucht werden.

Wie man sieht entsteht bei nur 3000 Händen bereits ein potenziell sehr langes Turnier. Jedoch sind 3000 Hände aufgrund der hohen Varianz von Texas Hold'em Poker nicht annähernd ausreichend, um festzustellen ob ein Spieler wirklich besser spielt als sein Gegner. Einer der Spieler könnte auch einfach etwas Glück mit den Karten gehabt haben und ist vielleicht in wesentlich einfacher zu spielende Situationen gekommen. Um diese Varianz etwas abzuschwächen werden alle Spiele als ‚duplicate matches' gespielt. Das heißt im Fall von dem eben beschrieben Heads-Up Poker, dass sämtliche Matches doppelt gespielt werden. Alle Karten die gespielt werden, werden hierfür aufgezeichnet. Ist ein Match beendet wird der Speicher der Pokerbots resettet, die Positionen der beiden Bots werden getauscht und alle Hände werden noch einmal gespielt. Hierdurch wird die Varianz zwar nicht komplett eliminiert, jedoch werden die Schwankungen merklich verringert. Es werden also insgesamt 2 Matches à 3000 Hände gespielt. Hierdurch entsteht eine Maximalspieldauer von rund 23 Stunden.

Wie zu erkennen ist, beansprucht die Evaluation der Bots bereits im Heads-Up enorm viel Zeit. Dieser Zeitaufwand wird bei den 3-Spieler Spielen noch größter, da auch hier im ‚duplicate match'-Modus gespielt werden muss, um die Varianz ausreichend zu senken. Um den Zeitbedarf in annehmbaren Grenzen zu halten werden bei den 3-Spieler Spielen pro Match nur 1000 Hände gespielt. Die durchschnittliche Zeit pro Hand ist wieder mit 7 Sekunden begrenzt. Soll jeder Spieler auf jeder Position mit allen Permutationen an Gegnerpositionen gesessen haben sind 3! = 6 Matches nötig. Aus 1000 Händen pro Match, insgesamt 6 Matches und 7 Sekunden pro Hand ergibt sich eine Maximalspielzeit von 6 Matches * 1000 Hände * 7 Sekunden * 3 Spieler = 35 Stunden.

Die Gewinner der Turniere werden in 2 Kategorien bestimmt: Zunächst gibt es die Gewinnkategorie ‚Total Bankroll'. Hier werden die Bots nach ihren über alle Spiele im Turnier gewonnenen Chips platziert. Das heißt derjenige, der insgesamt die meisten Chips gewonnen hat, landet auf Platz 1, der Spieler mit den 2. meisten Chips findet sich auf dem zweiten Platz wieder und so weiter. Hierbei besteht aber die Möglichkeit dass ein Bot die Schwäche eines anderen Bots extrem gut ausnutzen konnte und deshalb enorm viele Chips gewonnen hat. Wenn dieser Bot gegen die Restlichen nur mittelmäßig spielt, wird er möglicherweise trotzdem an der Spitze des Rankings stehen. Um dieses ‚Ausnutzen' der Auswertungsregeln einzuschränken, gibt es noch ein 2. Bewertungsverfahren, das ‚Bankroll Instant Runoff'-Verfahren. Hierbei muss wieder zwischen Heads-Up und 3-Spieler Spielen unterschieden werden. Im Heads-Up werden nach allen zu spielenden 2-Spieler Spielen von allen Bots die gewonnen Chips summiert. Nun wird der Bot, der am Meisten verloren hat, von den weiter teilnehmenden Bots getrennt. Im nächsten Schritt werden nur noch die Spiele, in denen keine ausgeschiedenen Bots teilgenommen haben, betrachtet und die Bankrolls erneut berechnet. Dies wird solange wiederholt bis nur noch 2 Bots verblieben sind. Das Ergebnis dieses Spiels bestimmt den Spieler, die Reihenfolge der Ausschlüsse der anderen Spieler die verbliebenen Plätze. Bei den 3-Spieler Spielen wird zunächst ein Durchgang mit allen möglichen Kombinationen von 3-Spieler Spielen durchgeführt. Der Bot, der am meisten Chips verloren hat, wird aus der Menge der teilnehmenden Bots herausgenommen und die verblieben Spieler spielen wieder alle möglichen Kombinationen durch. Dies wird solange fortgesetzt bis nur noch 3 Spieler übrig sind. Das Ergebnis des Spiels dieser 3 verblieben Spieler bestimmt die Plätze 1, 2 und 3; die Reihenfolge der Ausschlüsse der anderen Agenten wieder die verbliebene Plätze. Mit dieser Art der Siegerbestimmung ist es nicht mehr möglich, dass ein Bot durch Ausnutzen der Schwäche eines einzelnen Bots zum Gesamtsieger wird.

1.3 Ziele der Bachelorarbeit

Seit dem Sommersemester 2008 findet am Fachbereich Informatik[4] der Technischen Universität Darmstadt[5] die ‚TUD Computer Poker Challenge‘ statt. Ziel der von der Knowledge Engineering Group veranstalteten Challenge ist es zunächst intelligente Pokeragenten zu entwickeln, um dann ggf. an der ACPC teilzunehmen.

Im Rückblick auf die TUD Computer Poker Challenge im Sommersemester 2009 sind einige Probleme und damit auch Verbesserungsmöglichkeiten, die den Nutzen und die Effizienz der Challenge erheblich steigern können, aufgefallen. Diese Probleme sollen im Rahmen dieser Bachelorarbeit adressiert und (teilweise gelöst) werden. Im Folgenden werden detailliert verschiedene Aspekte erläutert und angestrebte Lösungen vorgeschlagen.

Teil der Computer Poker Challenge war bisher ein Praktikum, in dem konkrete Pokerbots implementiert werden sollten. Die Teilnehmer des Praktikums wurden in Gruppen von ca. 3 Personen aufgeteilt, die dann weitgehend unabhängig voneinander die Pokeragenten entwickelt haben. Diese Gruppen haben jeweils auf dem Framework der University of Alberta aufbauend ihre Entwicklung begonnen. Dieses Framework ist allerdings sehr spartanisch. Es stellt kaum mehr zur Verfügung als den Server, auf dem Bots gegeneinander spielen können und eine abstrakte Klasse, die als Wurzel für alle Pokeragenten, die zum Alberta-Server kompatibel sein sollen, dient. Mit Hilfe dieser Klasse ist es möglich einen einfachen String zu empfangen, der den Spielzustand repräsentiert, und die grundlegenden Aktionen ‚fold‘, ‚call‘ und ‚raise‘ an den Server zu senden. Hieraus resultierte für jede Gruppe zunächst die Aufgabe, diesen sogenannten ‚GameStateString‘, der in ‚4 Zustandsrepräsentierung‘ genauer beschrieben wird, zu parsen und auszuwerten, um damit eine benutzbare Repräsentation des aktuellen Spielzustands zu generieren. Diese grundlegende Aufgabe muss also zunächst von allen Teams gelöst werden bevor man mit der eigentlichen, interessanten Tätigkeit, nämlich dem Entwickeln des Agenten, beginnen konnte. Ein Ziel dieser Bachelorarbeit ist es, die Gruppen von dieser Routineaufgabe zu entbinden und den Spielzustand in einer einfach auswertbaren Form zur Verfügung zu stellen. Diese Repräsentation soll wesentlich detaillierter sein als die Informationen, die vom GameStateString direkt abgelesen werden können. So liefert der String nur sehr grundlegende Informationen wie z.B. die eigenen Holecards, die aktuelle Rundennummer oder die von den Spielern getätigten Aktionen. Hieraus lassen sich allerdings wesentlich komplexere Informationen ableiten. Hierzu zählen insbesondere auch einfach zu berechnende Informationen wie z.B. die Anzahl der verbliebenen Spieler, die aktuelle Potgröße oder die momentane Straße. Die Berechnung dieser einfachen, aber enorm hilfreichen Fakten über den zurzeit vorliegenden Spielzustand muss von jeder Gruppe des Praktikums selbst implementiert werden. Es ist also auch Ziel der Arbeit solche weitergehende Informationen automatisiert von dem Framework berechnen lassen zu können und den Entwicklern zur Verfügung zu stellen.

Eine weitere Hürde während der Challenge war das beschwerliche Starten der Auswertungsrunden. Aufgrund der Tatsache, dass das Alberta-Framework nur sehr grundlegende Funktionalität bereit stellte, mussten die einzelnen Auswertungsrunden extern per Skript gestartet und mit Hilfe von diversen Konfigurationsdateien eingestellt werden. Zudem war eine Zusammenfassung eines aktuell laufenden Spiels nicht möglich oder musste händisch berechnet werden. Es ist daher wünschenswert diesen Ablauf zu vereinfachen, umso wiederum den Fokus der Arbeit auf das Wesentliche legen zu können. Die Implementierung eines einfach zu bedienenden Servers mit der Möglichkeit Clients leicht mit diesem verbinden zu können ist also ein weiteres angestrebtes Ziel dieser Arbeit. Zudem soll es möglich sein Zwischenstände von Spielen anzeigen zu lassen, um schon früh Tendenzen während der Turniere erkennen zu können.

[4] http://www.informatik.tu-darmstadt.de/

[5] http://www.tu-darmstadt.de/

Zusätzlich zu der Problematik des umständlichen Durchführens von Turnieren kam die Unregelmäßigkeit, mit der Turniere durchgeführt wurden, an denen alle Teams mit ihren aktuell besten Pokeragenten teilnahmen. Während der Challenge wurden Termine vereinbart, an denen die besten Bots, die in den einzelnen Teams entwickelt wurden, eingeschickt werden sollten. Mit diesen Bots wurden dann Turniere durchgeführt und die Ergebnisse veröffentlicht. Problematisch hierbei war, dass diese Termine meist mehrerer Tage, manchmal sogar mehrere Wochen, auseinander lagen. Das heißt hatte man bedeutende Neuerungen in einem Bot implementiert, musste man eventuell mehrere Tage warten bis man diese gegen die Bots der anderen Teams testen konnte. Hierdurch entstand immer ein kleiner Leerlauf in der Entwicklung, da man natürlich zunächst den Erfolg einer Veränderung testen wollte, bevor man weitere Modifikationen vornahm. Zwar konnte man jederzeit lokale Turniere starten, jedoch macht es wesentlich mehr Sinn die entwickelten Bots gegen ‚fremde' Bots, deren Funktionsweise man nicht kennt, antreten zu lassen. Denn nur dadurch kann man ermitteln, ob die neue Version eines Bots gegen sehr unterschiedliche Botdesigns besser spielt als eine alte. Dieser Zeitverlust war insbesondere bei kleinen Veränderungen ärgerlich. Zudem war es nicht möglich verschiedene Werte für einfache Parameter des Bots einfach und schnell zu testen. So konnte man beispielsweise nicht einfach 3 Werte für einen fold-Schwellenwert durchtesten, da die Anzahl der einreichbaren Bots aufgrund von Ressourcenknappheit auf nur 2-3 Implementationen begrenzt war. Ebenfalls war die Auswahl der zugeteilten Gegner nicht immer zielführend. So konnte es vorkommen, dass man gegen einen bestimmten Bot einer gegnerischer Gruppe Neuerungen testen wollte, diesen aber nicht als Gegner zugelost bekam. Diese Auslosung von Konstellationen war nötig, da die Ressourcen der internen Auswertung begrenzt waren und so nicht alle Permutationen durchgespielt werden konnten. Hieraus resultiert das nächste Ziel der Arbeit: Es soll die Möglichkeit geschaffen werden neue Implementierungen von Pokeragenten auf eine einfache Art und Weise zu jederzeit gegen andere Bots antreten lassen zu können. Der bereits beschriebene, angestrebte Server soll diese Funktionalität anbieten. Er soll dauerhaft auf einem Rechner laufen und Gegner anbieten können, gegen die man die eigenen Bots testen kann. Somit können kleine Änderungen gegen unbekannte Bots kurzfristig evaluiert werden. Durch die Tatsache, dass der eigene Bot auf einem lokalen Rechner läuft, werden zudem die Ressourcen des Servers mit den angeschlossenen Bots weniger stark belastet.

Ebenfalls problematisch ist die effiziente Implementierung von verschiedenen Algorithmen, die nahezu bei jedem weiterentwickelten Pokerbot benötigt werden. So ist zum Beispiel die Verwendung der All-In Equity oder anderer Metriken wie des Immediate Hand Ranks, die in ‚6 Metriken' beschrieben werden, in vielen Spielsituationen ein guter Indikator, welche Aktion profitabel ist. Hat man beispielsweise eine hohe All-In Equity, so ist es oft ein Fehler zu folden. Eine hohe All-In Equity ist vielmehr ein starker Indikator dafür, dass ein Raise eine gute Aktion sein könnte. Das Problem bei diesen Metriken ist, dass sie sich meist nur annäherungsweise bestimmen lassen. Eine vollständige Auswertung aller möglichen Zustände ist wegen des großen Zustandsraums nicht möglich. Durch eine Monte-Carlo-Simulation lassen sich diese Werte jedoch meist ausreichend genau bestimmen. Diese Simulationen sind aber sehr rechenaufwendig, was eine effiziente Implementierung der Berechnungsalgorithmen enorm wichtig macht. Eine gute, schnelle und korrekte Implementierung ist allerdings nicht trivial. Ein weiteres Ziel ist es also die Berechnung einiger Metriken effizient zu implementieren, so dass sie von den Benutzern des Frameworks nur noch in die zu entwickelnden Pokeragenten eingebaut werden können.

Als letztes wichtiges Ziel ist die Erweiterbarkeit des zu entwickelnden Frameworks zu nennen. Da das Framework nicht alle Probleme bei der Entwicklung intelligenter Pokeragenten lösen kann soll es unbedingt leicht erweiterbar sein. Bisherige Ergebnisse der Challenge konnten, aufgrund der fehlenden Codebasis, nur schwer mit in die nächste Challenge übernommen werden. Es soll in Zukunft möglich sein Fortschritte, die von den einzelnen Gruppen in einer Challenge erzielt wurden, einfach an andere Gruppen weiterzugeben oder das Framework bei der nächsten Challenge in einer verbesserten Version anzubieten, indem die Neuerungen eingearbeitet werden. Es wäre zum Beispiel denkbar die TUD Computer Poker Challenge in 2 Phasen einzuteilen: Zunächst werden verschiedene Aspekte des Frameworks von den einzelnen Gruppen bearbeitet und somit das Framework verbessert. Einige Anregungen hierfür finden sich in der Sektion ‚9.2 Ausblick'. In der 2. Phase nutzen dann alle Gruppen

diese verbesserte Version und entwickeln aufbauend auf diesem ihre Pokeragenten. Um solche Gestaltungsmöglichkeiten zu unterstützen muss das Framework wie schon beschrieben leicht erweiterbar sein. Hieraus resultiert das Ziel an die Implementierung einen hohen Grad an Objektorientierung zu erreichen um die genannte Erweiterbarkeit zu gewährleisten.

Zusammenfassend ist es also das Ziel dieser Bachelorarbeit einen schnellen Einstieg in die Entwicklung intelligenter Pokeragenten zu ermöglichen. Hierfür ist es nötig den aktuellen Spielzustand leicht verwertbar zu repräsentieren, eine einfache und regemäßige Möglichkeit der Evaluation zu bieten, effiziente Algorithmen zur Berechnung von Bewertungskriterien der aktuellen Situation bereit zu halten und es insbesondere zu gewährleisten, dass gemachte Fortschritte leicht verteilt und wiederverwendet werden können.

1.4 Aufbau der Arbeit

Nachdem die Motivation für die Entwicklung eines Frameworks für Pokerbots erläutert, die Annual Computer Poker Competition vorgestellt und die Ziele der Bachelorarbeit genannt wurden, folgt nun ein Überblick über das Framework, bevor in weiteren Kapiteln auf einzelne Details eingegangen wird. Dieser Überblick gliedert sich in einen Teil, in dem zunächst eine grobe Übersicht über wichtige Komponenten des Frameworks gegeben wird, und einen zweiten Teil, der eine Aufstellung der angelegten Pakete und Klassen enthält. Abschließend wird noch kurz auf den Ablauf der Initialisierung eines Pokerbots eingegangen und wie der Bot über Aktualisierungen des Spielzustandes benachrichtigt wird.

Das 3. Kapitel widmet sich dem Nachrichten-Subsystem. Dieses System stellt die nötige Funktionalität bereit, um die Kommunikation zwischen den Klienten (den Pokerbots) und dem Pokerserver zu ermöglichen. Es wird hierbei als erstes auf die zu verschickenden Nachrichten eingegangen bevor die Initialisierung des Subsystems beschrieben wird. Ein Beispiel einer erfolgreichen Übermittlung einer Nachricht schließt dieses Kapitel ab.

Im 4. Teil steht die Zustandsrepräsentierung im Mittelpunkt, die eine sehr zentrale Rolle bei der Entwicklung eines Pokerbots spielt. Den nur anhand dieser kann der Agent erkennen, wie seine Umgebung (der Pokertisch) aussieht und welche Eigenschaften gegeben sind. Hierfür wird als erstes der so genannte ‚GameStateString' beschrieben, der aus Kompatibilitätsgründen zum Framework der University of Alberta immer noch die Grundlage für weitere Auswertungen bietet. Eine auf dieser sehr einfachen Repräsentation aufbauenden Zustandsbeschreibung wird danach vorgestellt. Hierbei handelt es sich um eine Neuentwicklung, um den Zustand für die Pokerbots leichter auswertbar zu machen.

Über die Bedeutung effizienter Algorithmen insbesondere zur Auswertung des Showdowns wurde ebenfalls bereits in der Einleitung berichtet. Dieser Gedanke wird im 5. Abschnitt weitergeführt. Es wird die Notwenigkeit der Effizienz näher erläutert, bevor eine Lösung dieses Problems durch eine Lookup Tabelle vorgeschlagen wird. Der hierfür benötigte Index und der zu einer Hand gehörende Wert, der die Stärke der Hand angibt, werden darauffolgend beschrieben. Abschließend wird der gemachte Ansatz analysiert und eine weitere Optimierung diskutiert, die aufgrund einer Designänderung des Systems ermöglicht wurde.

Ein weiterer wichtiger Teil des Frameworks wird in Kapitel 6 beschrieben. Es handelt sich hierbei um Metriken, mit denen insbesondere Aussagen über die Stärke der eigenen Hand in einem gegeben Zustand gemacht werden können. Es kann die aktuelle Situation bewertet und eine Einschätzung vorgenommen werden, wie die Entwicklungsmöglichkeiten im Verlauf der aktuellen Hand sind. Dies geschieht durch die Kombination verschiedener Maße, die zusammengenommen und intelligent ausgewertet eine brauchbare Einschätzung der Lage liefern können.

Eine weitere Möglichkeit einen guten Spielzug aus der aktuellen Situation abzuleiten ist neben der Analyse mit den vorgestellten Metriken eine Simulation der zukünftigen Ereignisse. Hierzu gehören die Karten, die noch als Gemeinschaftskarten aufgedeckt werden können genauso wie eine Einschätzung der gegnerischen Handkarten und ihrer möglichen, daraus resultierenden Aktionen. Eine Methode eine solche Simulation durchzuführen bietet die so genannte ‚UCT-Simulation', auf die im 7. Abschnitt eingegangen wird. Hierzu wird zunächst die Funktionsweise genauer erläutert und anschließend die Implementierung vorgestellt, deren Benutzung anhand eines Beispiels veranschaulicht wird.

Das 8. Kapitel stellt eine Anleitung da, wie die Oberfläche zum Starten eines Turniers benutzt werden kann und welche Bedeutung die verschiedenen Einstellmöglichkeiten haben. Insbesondere werden hier die beiden Betriebsmodi, die jeder Pokerbot im Framework besitzt, erklärt. Ebenso wird auf die Anzeige von Zwischenständen und der Auswertung der Resultate eingegangen.

Kapitel 9 fasst die Ergebnisse der Bachelorarbeit noch einmal zusammen und gibt einen Ausblick über verschiedene mögliche Erweiterungen und Verbesserungen des Frameworks, die in Zukunft bearbeitet werden könnten und von denen großer Nutzen ausgehen kann.

2 Architektur des Frameworks

2.1 Einleitung

Im folgenden Kapitel soll zunächst ein Überblick über das Framework gegeben werden, um einen Gesamteindruck des Aufbaus zu erhalten. Hierzu wird im Abschnitt ‚2.2 Framework-Übersicht' kurz erläutert welche Komponenten im Framework existieren und auf welche Art und Weise sie zusammen arbeiten. Es soll so ein vom konkreten Code abstrahierter Eindruck des Systems gewonnen werden können. Anschließend wird im Teil ‚2.3 package-Übersicht' der Fokus auf den Code gelegt, um direkt einen Einstieg in die Benutzung, Wartung und Weiterentwicklung zu ermöglichen. Hierbei werden auch die einzelnen Klassen, die bereits in der Übersicht angesprochen wurden, detaillierter erläutert. Den Abschluss bildet ein Sequenzdiagramm in ‚2.4 Ablauf Initialisierung & Reaktion auf Updates', welches zunächst das Starten eines Pokerbots aus Sicht des Frameworks beschreibt und dann veranschaulicht, wie das Aktualisieren des Bots bei einem Zustandswechsel abläuft und wie er dann darauf reagieren kann.

2.2 Framework-Übersicht

Um zunächst einen schnellen Überblick um die wichtigsten Komponenten und das allgemeine Design des Frameworks zu erhalten dient Abbildung 1.

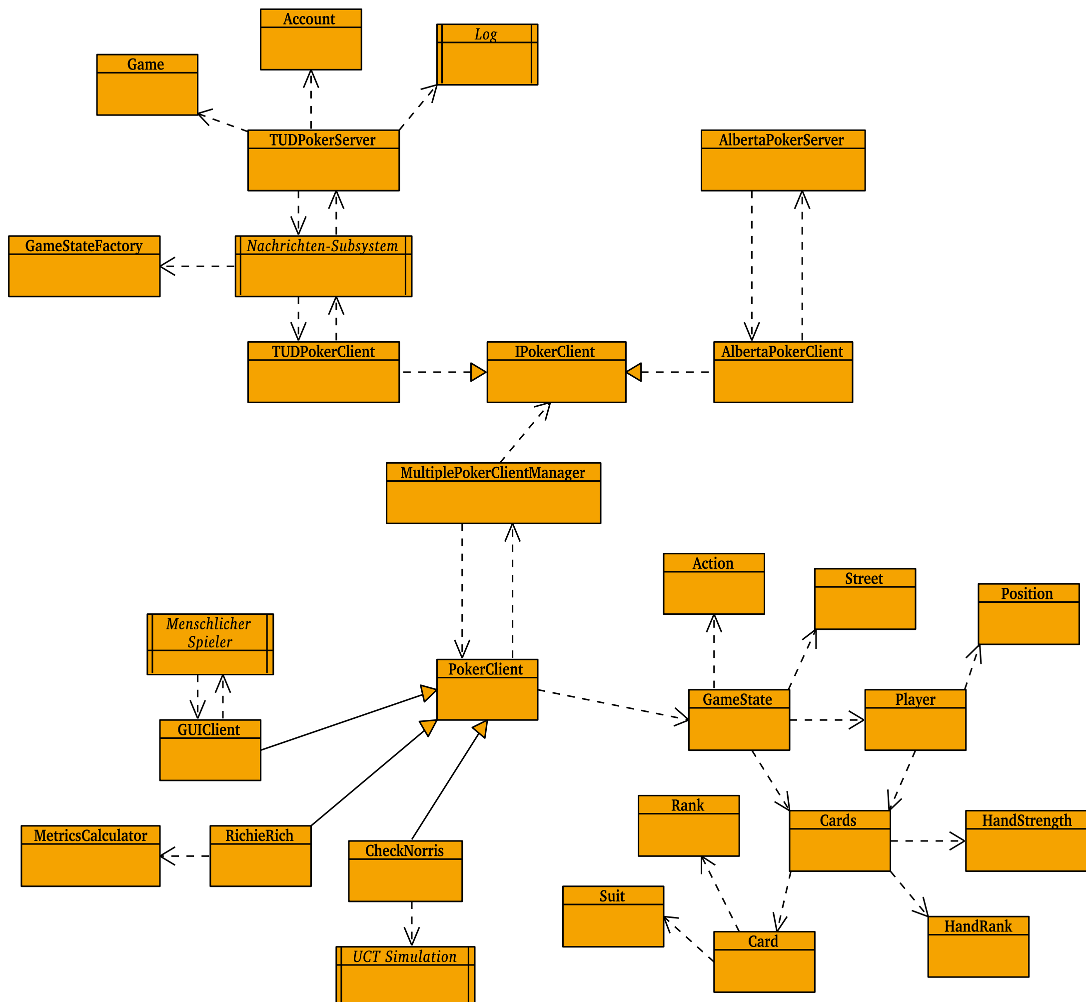


Abbildung 1: Übersicht über die wichtigsten Teile des Frameworks

Im Zentrum des Frameworks stehen die 3 wichtigsten Konzepte: ein Pokerserver, der durch TUDPokerServer modelliert wird, die Pokerclients mit dem abstrakten Interface PokerClient und dem GameState, mit welchem der Spielzustand eines Pokerspiels repräsentiert werden kann. Aus Kompatibilitätsgründen wird auch eine Schnittstelle zum Alberta-Pokerserver angeboten. Die Clients können die im Framework implementierten Metriken benutzen, die im Abschnitt ‚6 Metriken' beschrieben sind. In der Abbildung tut dies beispielsweise die konkrete Implementierung eines Bots mit dem Namen ‚RichieRich', während der Agent ‚CheckNorris' eine UCT Simulation einsetzt, um einen guten Zug zu ermitteln.

2.3 package-Übersicht

Die Package Übersicht stellt eine High-Level-Übersicht der Framework-Klassen da. Sie dient dazu einen groben Überblick über die Strukturierung des Quellcodes zu bekommen. Hierzu werden nun die einzelnen Packages und ihre Verantwortlichkeiten beschrieben. Zur besseren Lesbarkeit wird auf die vollqualifizierenden Klassennamen (Fully-Qualified Class Name, FQCN) verzichtet und nur der einfache Klassenname genannt. Für einen weitergehenden Blick auf die einzelnen Komponenten wird an dieser Stelle auf den Quellcode und die JavaDoc-Dokumentation verwiesen.

2.3.1 pokertud.cards

Das Package `pokertud.cards` beinhaltet Klassen, um grundlegende Objekte der Pokerdomäne darzustellen. Hierzu zählt u.a. die Klasse `Card`, die eine einzelne Spielkarte repräsentiert. Eine `Card` besitzt insbesondere einen Rang, der durch die Klasse `Rank` modelliert wird, sowie eine Farbe, die durch die Klasse `Suit` beschrieben wird. Eine Menge von Karten wird durch Objekte der Klasse `Cards` repräsentiert. Diese Klasse stellt auch die Funktionalität bereit, die zum Evaluieren einer Hand, also dem Bestimmen der Handstärke, benötigt wird. Die Handstärke selbst wird durch die Klassen `HandStrength` und `HandRank` modelliert.

2.3.2 pokertud.gamestate

Im `pokertud.gamestate` Paket wird eine weitere, zentrale Klasse modelliert: Objekte vom Typ `GameState` repräsentieren den Zustand einer Spielsituation. In der Klasse werden z.B. die Karten auf Flop, Turn und River, die aktuelle Potsize und der aktuell agierende Spieler gespeichert. Die Spieler selbst werden durch die Klasse `Player` beschrieben. Die Spieler speichern unter anderem ihren Namen, ihre relative Position zum Button und wie viel Chips bereits in den Pot investiert wurden. Mit den Enumerationen `Action`, `Street` und `Position` werden die im Spiel möglichen Aktionen, die gespielten Straßen sowie die Positionen, auf denen sich die Spieler befinden können, gespeichert. Die `GameStateFactory` ist dafür zuständig gültige `GameState`s aus einem `GameStateString` bzw. aus einem `ResultString` zu erzeugen.

2.3.3 pokertud.clients

Das Package `pokertud.clients` teilt sich in die Packages `pokertud.clients.framework`, `pokertud.clients.guiclient` und `pokertud.clients.swt` auf. Direkt im Package `pokertud.clients` sind die Klassen `BreakIvan`, `CheckNorris` und `RichieRich` enthalten, welche (sehr einfache) Beispielimplementierungen von Pokeragenten darstellen. Diese Agenten sind bereits voll funktionsfähig und können z.B. als Trainingsgegner eingesetzt oder als Startpunkt für die Entwicklung eigener Pokeragenten genutzt werden. Alle Pokeragenten, die mit dem Framework

kompatibel sind, erben von der Klasse `PokerClient` im Package `pokertud.clients.framework`. Die Klasse `PokerClient` speichert den aktuellen Zustand des Spiels und lässt die Pokeragenten auf Änderungen des Zustands reagieren. Dies geschieht, indem alle Pokeragenten die abstrakte Methode `handleGameStateChange` überschreiben und so durch Polymorphie in das Geschehen mit eingebunden werden. Ebenfalls von `PokerClient` erbt die Klasse `GUIClient`. Objekte dieser Klasse reagieren aber nicht selbstständig auf Änderungen des Spielzustandes, sondern benachrichtigen lediglich die bei ihr angemeldeten Observer. Diese Funktionalität wird in `PokerGUI` benutzt, um eine einfache Poker-GUI zu implementieren. Somit ist hier auch eine Schnittstelle für menschliche Spieler geschaffen, die sich so mit andern Spielern beziehungsweise Bots messen können. Dadurch bietet sich z.B. auch die Möglichkeit gegen Eigenentwicklungen zu spielen und eventuelle Schwachstellen der Bots ausfindig zu machen.

Die Verbindung zum Pokerserver wird über die Klasse `AlbertaPokerClient` bzw. `TUDPokerClient` hergestellt. Die Klasse `AlbertaPokerClient` stammt ursprünglich aus dem Computerpoker Framework der University of Alberta. Es wurde leicht modifiziert um sowohl mit dem Server aus Alberta als auch mit Clients, die auf dem TUD-Framework basieren, kompatibel zu sein. `TUDPokerClient` wurde erstellt, um mit dem TUD-Server in Verbindung zu treten. Beide Klassen implementieren `IPokerClient` und damit `sendAction (Action action)`, was wiederum das Senden einer Action an den jeweiligen Server darstellt.
Der `MultiplePokerClientManager` ist eine Factory-Klasse und erzeugt Instanzen von `AlbertaPokerClient` und `TUDPokerClient`. `MultiplePokerClientManager` abstrahiert aus Sicht des Clients von den unterschiedlichen Pokerservern, leitet die gemachten Aktionen weiter und benachrichtigt die Clients über Veränderungen des Spielzustands.

2.3.4 pokertud.server

In `pokertud.server` ist der Pokerserver des Frameworks zu finden. Die Klasse `Server` startet und beendet Spiele und reagiert auf die Aktionen der Spieler. Die Spieler werden serverseitig durch die Klasse `Account` modelliert. `Game` repräsentiert einen kompletten 3-Spieler Durchgang eines Spiels mit den 6 Iterationen im duplicate-Match Modus. Diese Klasse beinhaltet auch die `GameState`s aus Sicht der jeweiligen Spieler.

2.3.5 pokertud.message

Das Message-Subsystem befindet sich im Package `pokertud.message`. In diesem Package befindet sich die abstrakte Klasse Message, die als Superklasse für alle Nachrichten, die vom Message-Subsystem verarbeitet werden können, dient. Im Package `pokertud.message.client` befinden sich Nachrichten, die der Pokerclient verschickt, `pokertud.message.server` beinhaltet die Nachrichten, die vom Pokerserver verschickt werden und in `pokertud.message.intern` liegen interne Messages, die für Verwaltungszwecke vom Message-System benutzt werden. Im Package `pokertud.message.uidandtime` sind Klassen vorhanden, die es ermöglichen systemweit eindeutige Nachrichten-IDs zu vergeben und eine systemweite einheitliche Zeitmessung ermöglichen. Klassen im Package `pokertud.message.serverclient` sind für das Versenden und Empfangen von Nachrichten verantwortlich. Hierfür gibt es einen `MessageServer`, der einen `MessageServerSocket` besitzt. Die Schnittstelle für den Pokerserver zum Messaging-System stellt die Klasse `ServerMessageClient` da, während die Clients über den `ClientMessageClient` kommunizieren. Beide erben von der abstrakten Klasse `MessageClient`, deren abstrakte `eval (Message message)` Methode von den konkreten Klassen überschrieben werden muss. `MessageClient` stellt die grundlegende Funktionalität zum Senden und Empfangen von Nachrichten

bereits zur Verfügung. Die Klasse `MessageLoggingServer` implementiert eine einfach Möglichkeit für ein Logging der vom System verarbeiteten Nachrichten.

2.3.6 pokertud.divat

Im Package `pokertud.divat` wird Code zur Berechnung grundlegender Metriken zur Verfügung gestellt. Mit der Klasse `MetricsCalculator` lassen sich z.B. der Immediate Handrank und die All-In Equity berechnen. Die Klasse `Divat2Max` führt eine DIVAT-Analyse, wie sie in [BK06] beschrieben wird durch. `GeneratePreflopIHRArrays` implementiert einen Algorithmus für die aufwendige Berechnung der preflop Immediate Handranks.

2.3.7 pokertud.uct

Das Grundgerüst einer UCT-Simulation wird im Paket `pokertud.uct` implementiert. `UCTSimulation` stellt dabei den Startpunkt einer Simulation da. In der Klasse enthalten ist ein Objekt der Klasse `RootNode`. Sie erbt, wie alle Knoten, von der abstrakten Klasse `Node`. Ist ein Gegner am Zug, so befindet sich im Spielbaum an der aktuellen Stelle ein Objekt vom Typ `OpponentNode`. Handelt es sich im ein Zufallsereignis (es wird z.B. der Flop aufgedeckt) so handelt es sich um eine `ChanceNode`. `HeroNode`s beschreiben Knoten im Baum an denen der Spieler, aus dessen Sicht die Evaluation durchgeführt wird, am Zug ist. `LeafNode`s sind Blätter des Baums, die verwendet werden, wenn es zum Showdown kommt oder wenn alle Spieler bis auf einen gefoldet haben. Jeder Spieler im `GameState` kann zudem ein Objekt vom Typ `DecisionMaker` besitzen. Damit ist es möglich Spielern während der Evaluation des Baums ein spezielles Gegnermodell zuzuweisen um die Simulation realistischer zu machen.

2.3.8 pokertud.util

In `pokertud.util` befinden sich allgemeine Hilfklassen. `DeepObjectCopy` fertig tiefe Objektkopien an. `Util` stellt eine Möglichkeit bereit, um Arrays formatiert auszugeben bereit.

2.4 Ablauf Initialisierung & Reaktion auf Updates

In diesem Abschnitt wird auf das Sequenzdiagramm in Abbildung 2 eingegangen. Das Diagramm beschreibt im oberen Teil den Ablauf beim Starten eines Pokerbots und im unteren Teil wie der Pokerbot über die Aktualisierung des Spielzustandes informiert wird und wie der darauf reagiert.

Das Initialisieren eines Pokeragenten im Framework beginnt damit, dass der Benutzer eine konkrete Implementierung eines Bots startet. Genauer gesagt wird nicht der Bot gestartet, sondern der ClientRunner wird dazu aufgefordert eine Instanz des Bots auszuführen. Hierfür werden zunächst Bot und ClientRunner instanziiert und dann dem ClientRunner die Instanz des Pokerbots übergeben. Der ClientRunner instanziiert daraufhin den MultiplePokerClientManager. Danach weist er diesen an, einen TUDPokerClient zu erstellen, um mit dem Pokerserver kommunizieren zu können. Der MultiplePokerClientManager fügt sich nach dem Erstellen bei dem TUDPokerClient als Observer hinzu, um so über mögliche Aktualisierungen informiert zu werden. Diesen Client lässt sich der ClientRunner zurückgeben, um dem TUDPokerClient direkt die Anweisung zum Aufbau einer Verbindung zum Pokerserver zu geben. Nach dem der Verbindungsaufbau beendet ist fügt der ClientRunner den Pokerbot als Observer des MultiplePokerClientManager hinzu. Hierdurch entsteht eine

Benachrichtigungskette, in der alle Updates bis an das letzte Glied, nämlich den Pokerbot, weitergegeben werden. Die Initialisierung ist damit abgeschlossen.

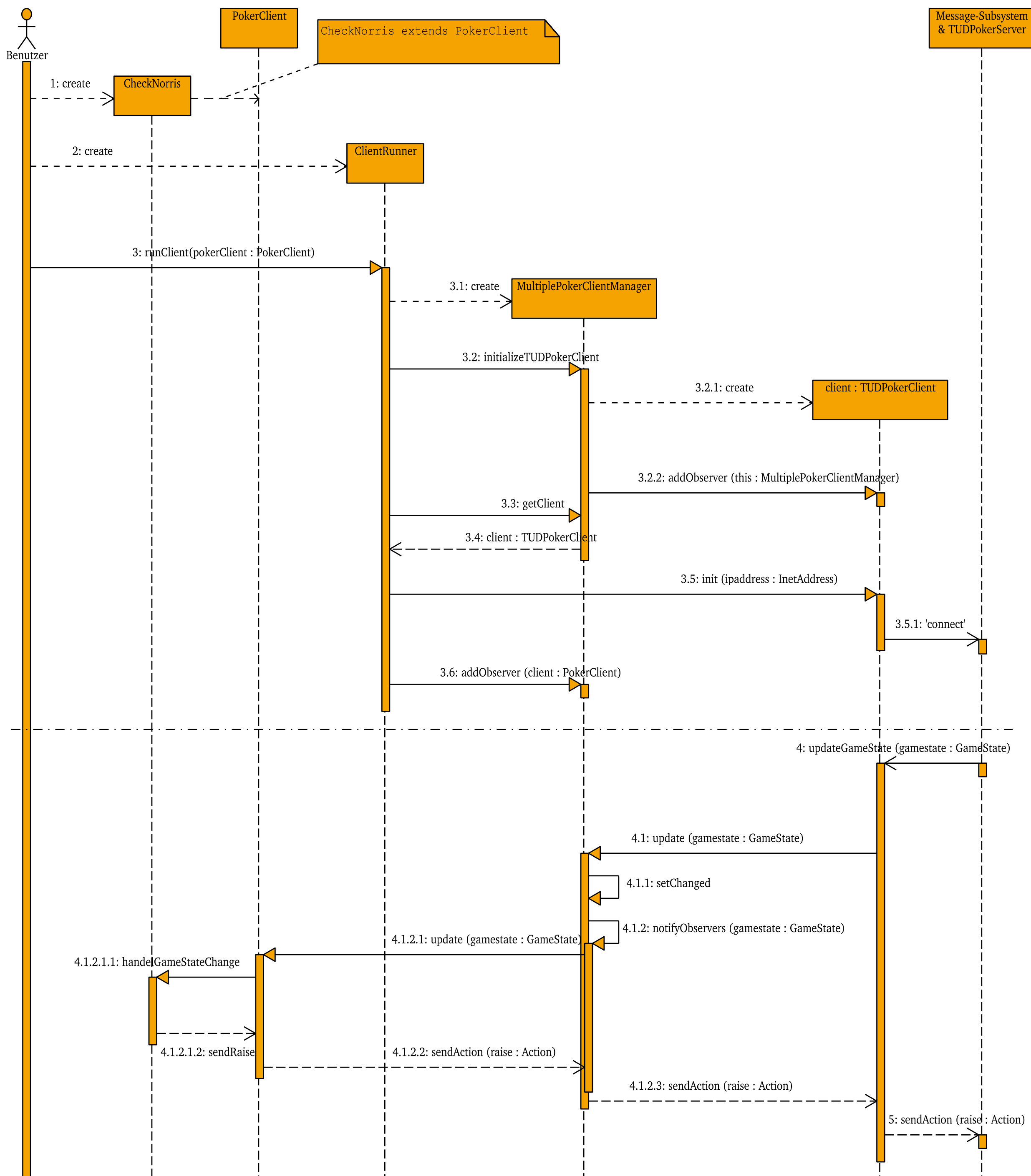


Abbildung 2: Initialisierung des Pokerbots 'CheckNorris', Aktualisierung des GameStates und Reaktion auf diese

Soll nun der Spielzustand geupdatet werden, so sendet der Pokerserver über das Nachrichten-Subsystem eine Nachricht mit dem neusten GameState an den zugehörigen Client. Hat der MultiplePokerClientManager das Update erhalten setzt er seinen Status auf ‚changed' und informiert seine Observer. Dies ist der Pokerbot, der Aufgrund der Vererbung von PokerClient, den neuen GameState speichert und die Methode zum Behandeln des neuen Zustands aufruft. Nun Beginnt für den Bot die eigentlichen Arbeit, falls er am Zug ist: der zu tätigende Zug muss ausgewählt werden. Ist dies geschehen wird auf dem gleichen Weg die Aktion zurück gesendet.

3 Message-Subsystem

3.1 Einleitung

Das Message-Subsystem stellt die zentrale Funktionalität zum Senden und Empfangen von Nachrichten im Framework zur Verfügung. Es wurde bei der Entwicklung auf eine möglichst einfache, leicht erweiterbare und robuste Möglichkeit der Nachrichtenübermittlung wert gelegt. Ein wesentlicher Vorteil des Systems gegenüber dem Versenden einzelner ASCII-kodierter Zeichen, wie es in dem Alberta-Framework benutzt wird, ist die Möglichkeit, dass ganze Objekte versendet werden können. Dies bietet eine Vielzahl von Vorteilen. Beispielsweise können ganze Listen, wie die Liste der auf dem Server zur Verfügung stehenden Gegner, sehr einfach über das Nachrichtensystem geschickt werden. Ebenfalls sind eine Implementierung von neuen Klassen und die Möglichkeit zum Versenden dieser so sehr einfach. Das Nachrichtensystem unterstützt auf diese Weise die einfache Erweiterbarkeit des Frameworks.

In diesem Kapitel werden nun zur genaueren Betrachtung des Nachrichtensystems zunächst die Nachrichten erläutert, die in dem System implementiert wurden. Anschließend wird auf die Initialisierung eingegangen bevor gezeigt wird, wie das Versenden einer konkreten Nachricht abläuft.

3.2 Nachrichten

Alle Nachrichten die versendet oder empfangen werden können sind Instanzen von den jeweilig zugehörigen Klassen. So ist beispielsweise eine Nachricht zum Starten eines Spiels eine Instanz der Klasse StartGameMessage und eine Nachricht zum Updaten eines Clients eine Instanziierung von UpdateClientMessage. Alle Nachrichtenklassen erben von der Klasse Message. Diese Klasse ist abstrakt, eine Instanziierung also ausgeschlossen, da das Versenden einer semantisch bedeutungslosen Nachricht keinen Nutzen bringt. Jedoch stellt die Klasse Message einige grundlegende Funktionalitäten bereit, die für jede Nachricht nützlich bzw. sogar notwendig ist und speichert einige unverzichtbare Attribute. Zu den Funktionalitäten gehört zunächst das Setzen eines Erstellungsdatums (creationDate) und einer eindeutigen Nachrichtenidentifikationsnummer (messageID). Zusätzlich werden für alle Nachrichten die SenderID und die EmpfängerID gespeichert. Hiermit kann das System feststellen, zu welchem Nachrichtenclient die Nachricht gesendet werden muss und von wem die Nachricht versendet wurde.

Alle Nachrichten, die im System verwendet werden, lassen sich in 3 Bereiche unterteilen. Zunächst bilden die Nachrichten, die von einem der Pokerclients gesendet werden können, die größte Menge. Sie befinden sich im Package message.client. Nachrichten, welche ausschließlich vom Server versendet werden, sind analog zu den Clientmessages im Packet message.server zu finden. 2 Nachrichten, die nur intern im Message-Subsystem verwedent werden und weder vom Server noch von einem der Client verschickt wird, sind im Package message.intern untergebracht. Abbildung 3 veranschaulicht diese Unterteilung.

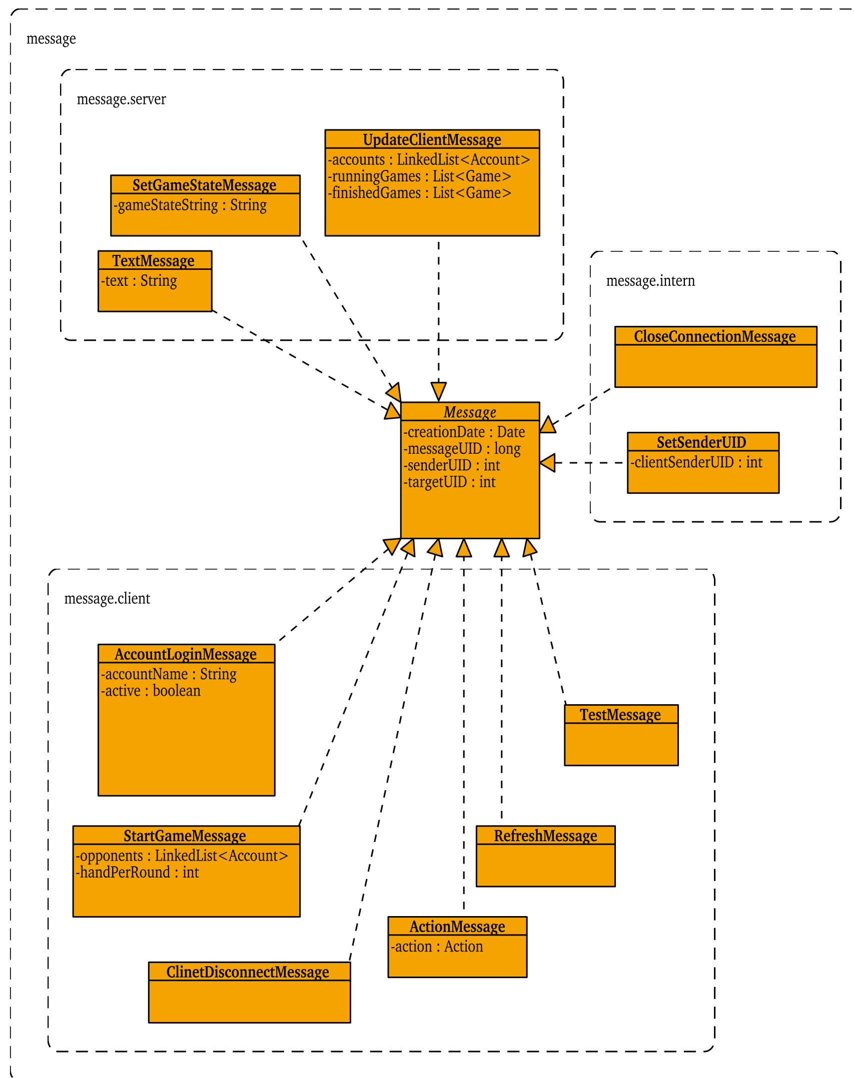


Abbildung 3: Übersicht über alle Nachrichten unterteilt in die 3 Bereiche Client, Server und Intern

3.3 Initialisierung

Das Nachrichten-Subsystem ist seinerseits als eine Client-Server-Architektur aufgebaut. Sowohl der Pokerserver als auch die einzelnen Pokerclients sind selbst wieder Clients im Nachrichtensystem. Um die Funktionsweise der Nachrichtenübermittlung besser nachvollziehen zu können wird nun den Ablauf der Initialisierung und das Versenden einer Refresh-Nachricht von einem Pokerclient zum Pokerserver erläutert. In Abbildung 4 ist dieser Ablauf als Sequenzdiagramm veranschaulicht.

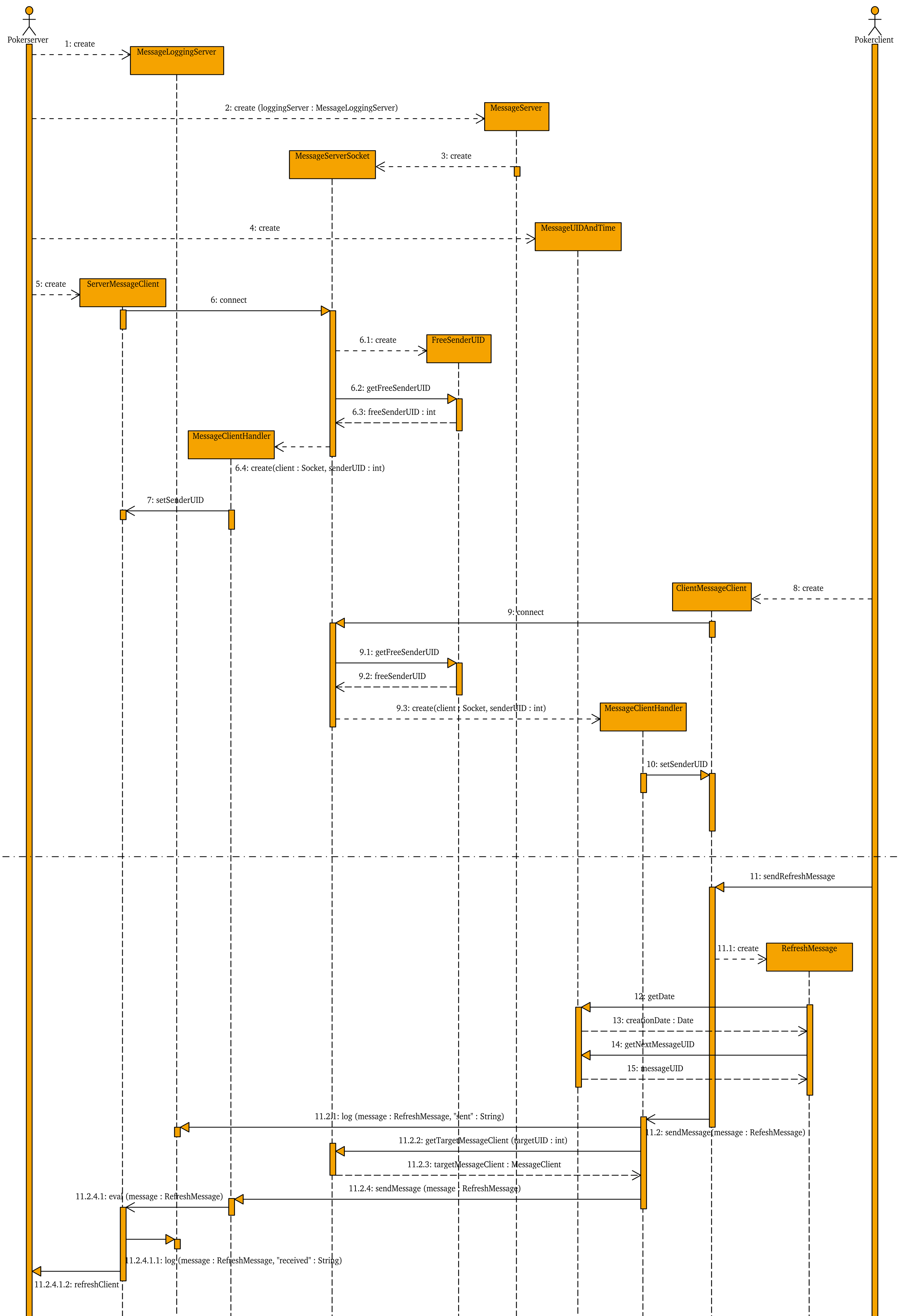


Abbildung 4: Initialisierung des Nachrichten-Subsystems und versenden einer Refresh-Nachricht

Zunächst wird immer der Pokerserver gestartet. Dieser erzeugt zunächst einen MessageLoggingServer. Der Logger übernimmt die Verantwortlichkeit Meldungen des Message-Subsystems aufzuzeichnen bzw. auszugeben. In der implementierten Version werden geloggte Meldungen einfach in der Konsole des Servers ausgegeben. Es ist aber auch ein wesentlich umfangreicheres Logging wie z.B. das Speichern der Nachrichten in einer Datenbank oder die Übergabe an ein komplexes Loggingframework wie log4j[6] denkbar. Die aktuelle Implementierung des Loggers soll also nur als Dummy verstanden werden um, gegebenenfalls eine Erweiterung so einfach wie möglich zu gestalten.

Nachdem der Loggingserver erzeugt wurde, wird der eigentliche MessageServer instanziiert und diesem eine Referenz zum Loggingserver übergeben. Der MessageServer öffnet daraufhin den MessageServerSocket. Dieser Socket stellt den Bezugspunkt für alle Verbindungsanfragen an den Server da. Ansprechbar ist der Socket per Default über den Port 35676. Wurde der Socket fehlerfrei erzeugt (gelingt dies nicht wird eine IOException geworfen) wartet der Socket auf eine Verbindungsanfrage in einem vom Pokerserver getrennten Thread. Nachdem der MessageUIDAndTimeServer instanziiert wurde, auf den später beim Versenden einer Nachricht noch genauer eingegangen wird, erzeugt der Pokerserver einen sogenannten ServerMessageClient. Es handelt sich hierbei um einen Messageclient im Nachrichtensystem. Jeder Teilnehmer in diesem System muss einen solchen Client erzeugen, um Nachrichten verschicken bzw. empfangen zu können. Der Zusatz ‚Server' in ServerMessageClient bedeutet, dass es sich hierbei um einen Client im Nachrichtensystem geht, der an einen Pokerserver gebunden ist. Es ermöglicht dem Pokerserver so alle Nachrichten zu versenden, die der Pokerserver versenden können soll. Dies sind genau die Nachrichten, die in Abbildung 3 im Bereich ‚message.server' zu finden sind. Der Nachrichtenclient verbindet sich ebenfalls wieder per Default zur IP-Adresse ‚127.0.0.1', also in der Regel dem gleichen System auf dem auch der Pokerserver gestartet wurde. Diese Adresse kann natürlich gegebenenfalls durch die Adresse eines im Internet erreichbaren Pokerservers ersetzt oder zur Laufzeit manuell eingegeben werden, um auch nicht lokale Server erreichbar zu machen. Der in der abstrakten Klasse angegeben Port (hier wieder 35676) muss dem Port entsprechen, mit dem der Messageserver bzw. der Socket des Servers gestartet wurde, um eine Verbindung zu ermöglichen. Dieser Verbindungsvorgang ist in Abbildung 4 in Nachricht ‚6: connect' abgebildet. Verbindet sich ein Client mit dem Socket des Servers muss zunächst eine noch freie, eindeutige Teilnehmernummer für den Client des Nachrichtensystem generiert werden. Hierfür ist der FreeSenderUIDHandler zuständig. Da der erste sich verbindende Client immer der Server ist, bekommt dieser eine im Code fest definierte ID, nämlich die 1. Mit dieser ID wird ein MessageClientHandler erzeugt, das Gegenstück zum ServerMessageClient auf der Pokerserverseite. Dieser Clienthandler bekommt vom Socket den Socket des Clients und die ihm zugeteilte ID übergeben. Den Socket nutzt der Handler, um dem ServerMessageClient eine erste Nachricht zu schicken, die dem Client die ab jetzt für ihn gültige Sender ID mitteilt. Der MessageClientHandler läuft in einem vom Socket getrennten Thread. Dies ist nötig, damit der Serversocket schnellstmöglich wieder neue Verbindungsanfragen bearbeiten kann. Für jeden Client im Nachrichtensystem wird also ein ClientMessageHandler erzeugt, über den mit dem Client kommuniziert werden kann. Ein extra Thread für jeden ClientMessageHandler ist nötig, um den MessageServerSocket nicht zu blockieren. Eine Veranschaulichung einer möglichen Ausprägung eines Nachrichtensystems findet sich in Abbildung 4. Hat der ServerMessageClient die Nachricht mit seiner senderUID erhalten ist er voll initialisiert und kann ab sofort Nachrichten an Clients im System versenden.

[6] http://logging.apache.org/log4j/index.html

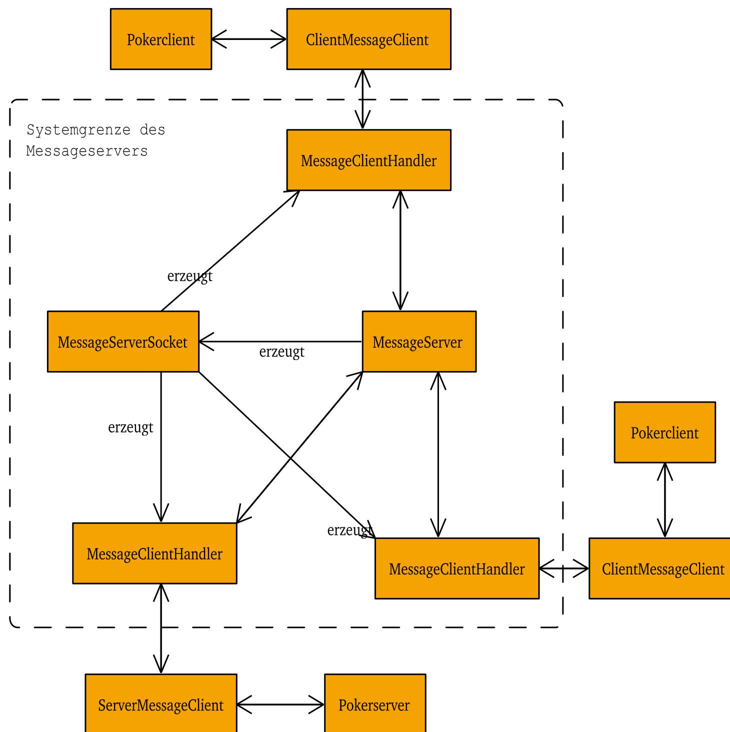


Abbildung 5: Beispielhafte Ausprägung des Nachrichten-Subsystems mit angeschlossenen Clients

Der Verbindungsaufbau eines Pokerbots zum Nachrichtensystem funktioniert nun ähnlich zum Verbindungsaufbau des Pokerserver. Im Gegensatz zum Pokerserver, der einen ServerMessageClient erzeugt, erzeugen die Pokerclients jeweils einen ClientMessageClient. Dies ermöglicht den Pokerclients nur genau die Nachrichten zu versenden, die sie versenden dürfen bzw. müssen. Zu finden sind diese Nachrichten wieder in Abbildung 3, diesmal jedoch im Bereich ‚message.client'. Der ClientMessageClient stellt eine Verbindung zum MessageServerSocket her, der aufgrund dieser Anfrage wiederum einen extra MessageClientHandler erzeugt, nachdem eine neue senderUID generiert wurde. Das Zuweisen der ID an den Client beendet den Initialisierungsvorgang und der Pokerclient kann mit den anderen Teilnehmern im Kommunikationsnetzwerk (und damit insbesondere dem Pokerserver) Nachrichten austauschen.

3.4 Nachrichtenübermittlung

Im Folgenden soll nun noch die Übermittlung einer Nachricht erläutert werden nachdem der Initialisierungsvorgang abgeschlossen ist und zumindest der Pokerserver und ein Client mit dem Nachrichtensystem verbunden sind. Dies wird auch im 2. Abschnitt in Abbildung 4 illustriert. Will der Client, wie hier im Beispiel gezeigt, einen aktualisierten Stand der Spiele auf dem Server und der zur Verfügung stehenden Spieler anfordern, so sendet er eine Aktualisierungsnachricht an den Server. Um Nachrichtensubsystem und Pokerclient bzw. Pokerserver zu entkoppeln erzeugt der Client nicht selbst die Nachricht, die über das Netzwerk verschickt wird. Stattdessen teilt er dem ClientMessageClient den Wusch der Aktualisierung mit. Dieser erzeugt dann die eigentliche Nachricht. Beim Initialisieren der RefreshMessage fordert die Nachricht vom MessageUIDAndTimeClient zunächst ein Erstellungsdatum an. Dieser hat bereits eine Verbindung zum zentralen MessageUIDAndTimeServer aufgebaut und erhält seinerseits von diesem ein Erstellungsdatum. Auf die gleiche Weise wird der neuen Nachricht eine Nachrichten-ID zugewiesen. Diese Aktionen werden zentral ausgeführt, um zu gewährleisten, dass keine Duplikate bei der Erstellung der Nachrichten-ID auftreten können und die Erstellungszeit stets anhand der gleichen Uhr festgesetzt wird. Ist die Nachricht fertig erstellt wird sie vom Messageclient

an den zuständigen MessageClientHandler gesendet. Dieser überprüft die in der Nachricht enthaltene targetUID, welche die ID des Empfängers darstellt. Der MessageServer besitzt eine Zuordnung von targetUID zu zuständigem MessageClientHandler. Mit dieser Zuordnung ist es möglich, den richtigen MessageClientHandler zu der gewünschten targetUID herauszufinden und die Nachricht an diesen zu übergeben. Dieser stellt nun fest, dass die Nachricht an seinen eigenen MessageClient gesendet werden soll und keine weitere Weiterleitung nötig ist. Also sendet dieser die Nachricht an den angebunden ServerMessageClient weiter, der wiederum auf die Nachricht reagiert und dem Server mitteilt, dass einer der Clients ein Update verlangt.

4 Zustandsrepräsentierung

4.1 Einleitung

Wie schon in der Einleitung beschrieben ist das Framework der University of Alberta sehr spartanisch ausgelegt. Es bietet unter anderem keine einfach auszuwertende Zustandsrepräsentation. Die Pokeragenten bekommen lediglich einen einfachen String übermittelt, aus dem dann alle weiteren Daten berechnet werden sollen. Aus diesem Grund wurde eine neue, verbesserte Zustandsrepräsentation für dieses Framework entwickelt. Aufbau, Auswertung und Benutzung dieser Repräsentation wird im Folgenden erläutert.

4.2 Der GameStateString

Wird ein AlbertaPokerClient gestartet, so wartet dieser nach der Initialisierung auf eingehende Nachrichten vom Pokerserver. Diese Nachrichten sind im Fall des Alberta-Pokerservers Strings, die den Zustand in dem sich der Pokerclient befindet, repräsentieren. Sobald sich etwas an der Situation ändert, wird den Clients jeweils ein neuer String gesendet. Eine Änderung kann beispielsweise bedeuten, dass ein Spieler eine Aktion getätigt hat, eine neue Runde begonnen hat oder das die nächste Straße ausgeteilt wurde. Es ist also so, dass eine Änderung des Zustandes nicht unbedingt bedeutet, dass der Bot, der benachrichtigt wurde, auch am Zug ist. Diese Information musste ebenfalls selbständig aus dem String gewonnen werden.

Der String kann 2 unterschiedliche Formate besitzen: Das Format ‚MATCHSTATE' und das Format ‚RESULTSTRING'. Während eines Spiels wird zunächst das Format ‚MATCHSTATE' verwendet, um die aktuelle Situation zu beschreiben. Der Matchstate beinhaltet Informationen über die aktuelle, absolute Position von Hero (also auf welchem Platz am Pokertisch er sitzt), den Index der aktuellen Runde und die bisher getätigten Aktionen der Spieler, unterteilt nach den gespielten Straßen. Darauf folgt eine Liste von bekannten Holecards aller Spieler. Während des Spiels hat diese Liste immer die Länge 1, da nur die eigenen Holecards bekannt sind. Zuletzt werden noch die bereits aufgedeckten Gemeinschaftskarten in dem String, durch einen ‚/' nach Straßen unterteilt, notiert:

$$MATCHSTATE:\{Sitz\}:\{Rundenindex\}:\{Aktionen\}:\{Holecards\}/\{Gemeinschaftskarten\}$$

Beispielhaft kann eine konkrete Ausprägung eines Matchstate-Strings die folgende Form haben:

$$MATCHSTATE:1:7:crcc/fcc:|9d7s|/AhKdJd$$

Die direkt abzulesenden Informationen sind:

- Hero befindet sich auf Sitz 1
- es wird die 7. Hand gespielt
- Preflop gab es die Aktionen check, raise, call, call
- auf dem Flop wurde fold, check, check gespielt
- Hero besitz die Holecards [9d7s]
- auf dem Flop liegen [AhKdJd]

Man sieht also, dass die Informationen, die dem Pokeragenten direkt zur Verfügung stehen, sehr eingeschränkt sind und in dieser Rohform kaum sinnvoll verarbeitet werden können. Zudem ist zu beachten, dass dieser String für jeden Spieler, der an einem Spiel teilnimmt, einen anderen Inhalt hat. Diese Tatsache modelliert die Eigenschaft der unvollständigen Information über den Spielzustand der Spieler.

Das 2. Format, der ‚RESULTSTRING', wird verwendet um bereits beendete Spiele in einem Log zu dokumentieren. Der Result-String besitzt nicht wie der Matchstate-String nur die unvollständigen Informationen wie sie ein Spieler am Tisch sieht, sondern den kompletten Zustand mit allen Holecards und Spielernamen. Er ist also nicht die Sicht eines Spielers auf das Geschehen, sondern stellt einen Überblick mit vollständiger Information da. Allgemein hat der String das Format

$$\{Spieler\}:\{Rundenindex\}:\{Aktionen\}:\{Holecards\}/\{Gemeinschaftskarten\} : \{Gewinne\}$$

Ein Beispiel für einen solchen Result-String könnte

$$BreakIvan\ (0)|CheckNorris\ (4)|RichieRich\ (3): \hookleftarrow$$
$$47: rfc/crc/cc/crc: 7h2s|Js6h|Qs2d/9s5c2c/5s/9c: -0.5| -5.0|5.5$$

sein.

Hieraus können zunächst folgende Informationen abgelesen werden:

- es spielten BreakIvan, CheckNorris und RichieRich gegeneinander
- es handelt sich um die 47. Hand
- die Preflopaktionen waren: raise, fold, call
- auf dem Flop/Turn/River wurde crc/cc/crc gespielt
- BreakIvan hatte [7h2s], CheckNorris [Js6h] und RichieRich [Qs2d]
- der Flop war [9s5c2c], der Turn war [5s] und auf dem River kam die [9c]
- BreakIvan hat 0,5 BB verloren, CheckNorris hat 5 BB verloren und RichieRich hat einen Gewinn von 5,5 BB gemacht

Auch hier sind die angebotenen Informationen, die direkt nach dem Parsen des Strings zur Verfügung stehen, sehr stark eingeschränkt.

4.3 GameState

Um diese, sehr einfache und damit schwer nutzbare Darstellung zu verbessern, wurden die Klassen GameState und GameStateFactory entwickelt. Die GameStateFactory parst einen Matchstate bzw. einen Result-String. Beim Parsen werden die oben aufgelisteten Informationen aus dem String extrahiert und in zugehörigen Objekten und Feldern gespeichert. Diese werden dann genutzt, um ein Objekt der Klasse GameState zu erzeugen, welche im neuen Framework die Aufgabe der Zustandsrepräsentation übernimmt. Die Informationen aus dem String werden dann einfach zu verwendend im Objekt gespeichert. Zudem werden weitergehende Informationen automatisch berechnet. Hierzu gehören beispielsweise die aktuelle Potgröße und die Anzahl der verbliebenen Spieler in der aktuellen Hand.

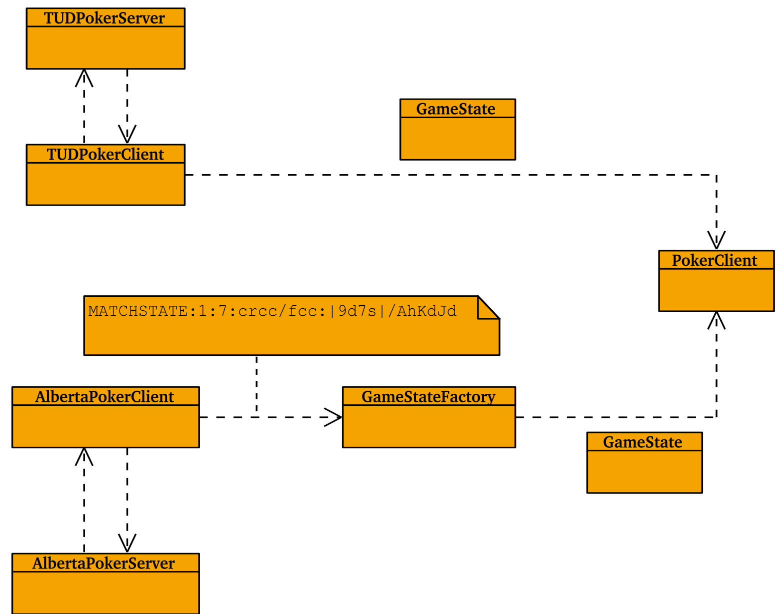


Abbildung 6: Datenfluss zwischen Server und Client

Diese Konvertierung von einem String in ein Objekt vom Typ GameState erfolgt bei der Benutzung eines AlbertaPokerClients automatisch, bei der Verwendung des neuen Servers wird direkt vom Server der GameState erzeugt. Die Pokeragenten können also unabhängig vom verwendeten Server immer auf die Darstellung des Spielzustandes in der Form eines GameStates zurückgreifen. In Abbildung 6 wird dies noch einmal dargestellt.

5.1 Einleitung

Unter Handevaluation versteht man das Auswerten der aktuellen Handstärke einer Pokerhand. Hierbei muss im Falle eines Showdowns ermittelt werden, welche 5-Karten Kombination (aus den 7 am Showdown zur Verfügung stehenden Karten) welches Spielers die stärkste Kombination darstellt. Es muss also der oder die Gewinner einer Runde berechnet werden. Eine effiziente Auswertung, ob es sich bei einer Hand beispielsweise um einen Flush oder eventuell nur ein Two Pair handelt, ist enorm wichtig, um z.B. Simulationen in akzeptabler Zeit durchführen zu können. Eine Berechnung der All-In Equity, bei der Zehntausende von Händen innerhalb kurzer Zeit ausgewertet werden müssen, wäre ohne eine schnelle Handevaluation nicht möglich. Sie ist der Flaschenhals von fast jeder simulativen Berechnung bei Poker, in der die eigene Handstärke in Bezug auf zufällige Gegnerhände evaluiert werden soll. Daher wurde beim Framework in diesem Bereich zunächst ausschließlich auf Performance geachtet und der Speicherverbrauch als zweitrangig eingestuft.

Da die Evaluation von Händen eine so zentrale Rolle spielt, wurde bereits viel in diesem Gebiet geforscht. Es gibt bereits einige gute und schnelle Handevaluatoren. Eines der meist verbreiteten Programme um die All-In Equity zu berechnen ist PokerStove[7]. Hier wird beispielsweise ein sehr effizienter Algorithmus eingesetzt, um Hände möglichst schnell auszuwerten. Leider ist die Software nicht Open Source und kann demnach nicht für eine genaue Analyse des Algorithmus eingesetzt werden. Es gibt aber auch Bibliotheken mit Handevaluatoren, die frei verfügbar sind. Zu nennen sind hier z.B. poker-eval[8], einem sehr effizienten Evaluierungssystem in C. Hold'Em Showdown[9] von Steve Brecher ist ein in Java geschriebener Handevaluator. Er wurde im Framework als Referenz eingesetzt, um die Korrektheit der neuen Implementierung zu überprüfen.

Die genannten Systeme beziehen ihre Effizienz aus einem sehr stark optimierten Algorithmus, um die Handstärke zu berechnen. Die schnellste Möglichkeit, um den Wert einer Pokerhand zu bestimmen, ist allerdings die Nutzung einer einfachen, aber dafür potentiell sehr großen, Lookup-Tabelle in der die Handstärke für jede mögliche Hand im Voraus berechnet und gespeichert wurde. Während der Ermittlung der Handstärke kann dann einfach in der Lookup-Tabelle die Stärke nachgeschlagen werden. Eine aufwendige Berechnung entfällt somit. Eine solche Lookup-Tabelle stellt meistens eine Sammlung von Tupeln der Form (Schlüssel, Wert) da. Im konkreten Fall des Poker Frameworks besitzt die Tabelle Tupel der Form (Poker Hand, Stärke der Poker Hand). Java bietet für solche Zwecke die Möglichkeit des Einsatzes von Hashmaps. Hierbei wird direkt von Objekten auf die zugeordneten Werte gemappt. Jedoch kommt es (wie in Hashmaps üblich) zu Kollisionen während des Lookups, der den Zugriff auf die Daten verlangsamt. Da der Zugriff auf den Wert aber möglichst schnell erfolgen soll, wurde zunächst getestet, ob man durch den Einsatz eines Arrays mit einem manuell berechneten Index einen Performancegewinn im Vergleich zu einer Hashmap erreichen kann. Der hierfür benötigte Index, der den Schlüssel in der Lookup Tabelle darstellt, wird im Abschnitt ‚5.2 RankCode' beschrieben. Der Wert, der im Array unter einem bestimmten RankCode zu finden ist, ist die Stärke der Hand, die zu dem jeweiligen RankCode gehört. Diese wird durch den HandStrengthCode modelliert, auf den in ‚5.3 HandStrengthCode' detaillierter eingegangen wird.

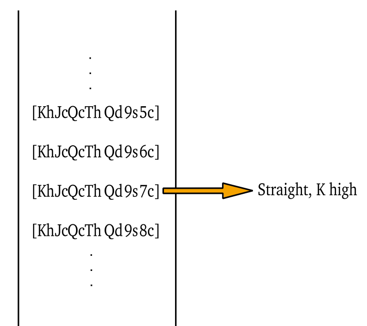


Abbildung 7: Zuordnung Hand zu Handstärke

[7] http://www.pokerstove.com/

[8] https://gna.org/projects/pokersource

[9] http://stevebrecher.com/Software/software.html

5.2 RankCode

Der RankCode dient bei der Handevaluation wie beschrieben als Schlüssel für den Lookup im Array, in dem die Zuordnung von Pokerhand (bzw. ihrem RankCode) zur Handstärke gespeichert ist. Da der RankCode direkt als Index für das Array benutzt werden soll, darf es zu keinen Kollisionen bei der Abbildung von Pokerhand zu RankCode kommen. Eine Kollision tritt auf, wenn 2 beliebige, verschiedene Pokerhände auf denselben RankCode abgebildet werden. Würde beispielsweise die Hand [KcJcQcTh9c] denselben RankCode besitzen wie die Hand [KcJcQcTc9c] würde in der Lookup-Tabelle für beide Hände an der gleichen Position die Handstärke gespeichert und nachgeschlagen werden. Das heißt es könnte nur eine gemeinsame Handstärke gespeichert werden, obwohl es sich einmal um eine Straße und einmal um einen Straight Flush handelt. Die Berechnung des RankCodes muss also zwingend injektiv sein, um solche Kollisionen zu vermeiden.

Eine weitere, wünschenswerte Eigenschaft an einen solchen Index ist die Surjektivität. In einem Array wird grundsätzlich für alle Indizes zwischen 0 und einem maximalen Index Speicherplatz für die zugehörigen Werte reserviert. Optimal lässt sich dieser Speicherplatz nur ausnutzen, wenn die Berechnung des Index nicht nur injektiv sondern auch surjektiv ist. Aus der Bijektivität der Abbildung folgt dann, dass jede Pokerhand einen gültigen Index besitzt und jedes Feld des Arrays auch mit Werten belegt wird. Die Berechnung einer bijektiven Abbildung beansprucht in diesem Fall aber zu viel Zeit. Daher wird hier wieder zugunsten der Performance und entgegen eines geringen Platzverbrauchs entwickelt und keine surjektive Abbildung eingesetzt. Es reicht aus, wenn die meisten der reservierten Speicherstellen auch mit Werten belegt werden. Es werden also Lücken in der Belegung des Array akzeptiert, um eine geringere Zugriffszeit zu erreichen.

Die zu betrachtenden relevanten Hände bei der Handevaluation bestehen aus 5 bis 7 Karten, denn es sind minimal 5 Karten nötig, um eine gültige Pokerhand zusammen zu setzen, und maximal kann man am Showdown eine Pokerhand aus 7 Karten bilden. Jedoch ist es so, dass bereits aus 5 Karten 311.875.200 mögliche Permutationen gebildet werden können. Würde man für alle möglichen Schlüssel die weiter unten beschriebenen 16 Bit Speicherplatz für den zugehörigen Wert benötigen, würde ein Platzbedarf von ca. 595 Megabyte entstehen. Bei 7 Karten würde Platz von mehr als einem Terabyte beansprucht werden. Da das Array immer komplett im Arbeitsspeicher gehalten werden muss, um die beabsichtigte Performancesteigerung zu erreichen, ist es nicht möglich das Array auf diese Weise anzulegen.

Es ist jedoch so, dass die Farben der jeweiligen Karten nur im Falle eines Flushs bzw. eines Straight Flushs einen Einfluss auf die Stärke der Hand haben. Die Wahrscheinlichkeit für einen Flush bei einer 5-Karten Hand beträgt nur ca. 0,2%, bei 7 Karten etwa 3%. Das heißt in der Regel handelt es sich bei

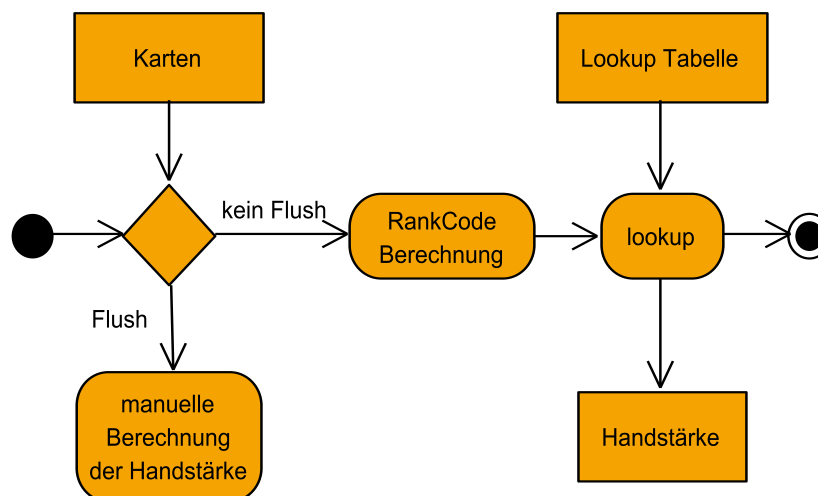


Abbildung 8: Evaluierung der Handstärke einer Hand

einer Pokerhand nicht um einen Flush, sondern um eine Hand bei der die Farben der Karten nicht von Bedeutung sind. Es bringt also immer noch eine gute Effizienzsteigerung, wenn man für alle ‚nicht-Flush' Hände einen Lookup durchführen kann und den kleinen verbleibenden Rest der Hände algorithmisch berechnet, wie es in Abbildung 8 dargestellt ist.
Der Index der Lookup-Tabelle berechnet sich also nur aus den Rängen der Karten und beachtet die Farbe der Karten nicht - daher auch der Name des Index: ‚RankCode'. Die Berechnung des RankCodes erfolgt durch die Formel

$$rankCode = \sum_{i=0}^{n-1} 13^i * Rang(Karte_i)$$

wobei n gleich der Anzahl der Karten ist und Rang(Karte) den Index des Rangs einer Karte liefert. So hat der Rang ‚TWO' beispielsweise den Index 0 und ‚ACE' den Index 12. Der schnell zu berechnende RankCode ist damit eine injektive Abbildung, wie weiter oben gefordert wurde. Der Index ist zudem fortlaufend, so dass keine durch die Berechnung verursachten Lücken im Array entstehen. Bei einer 5-Karten Hand exisiteren allerdings 13 ungültige Kombinationen (nämlich [22222] – [AAAAA]), die nie gebildet werden können, da die Anzahl der Karten eines Rangs in einem Deck auf 4 begrenzt ist. Es sind also in den 371.292 möglichen Plätzen des Arrays 13 enthalten, die nie erreicht werden können. Dies entspricht jedoch nur ca. 0,004% aller Plätze. Es wird also kaum Platz zugunsten einer schnellen Berechnung verschwendet. Bei 6 bzw. 7-Karten Händen ergibt sich ein ähnliches Verhältnis.

Es wurde also mit dem RankCode eine Möglichkeit gefunden einen Index, der keine große Platzverschwendung herbeiführt, für eine Lookup-Tabelle effizient zu berechnen. Durch die Abstraktion von den Farben der Karten konnte die Anzahl der zu speichernden Kombinationen erheblich reduziert werden, so dass eine Speicherung aller verbliebenen Möglichkeiten im Arbeitsspeicher möglich geworden ist.

5.3 HandStrengthCode

5.3.1 Einleitung

Der HandStrengthCode einer Pokerhand ist ein Wert, der die Stärke einer Pokerhand absolut angibt. Mit dieser absoluten Angabe kann sofort aus einer Menge von Pokerhänden und ihren zugehörigen HandStrengthCodes bestimmt werden, welche der Hände die Stärkste nach den Auswertungsregeln von Hold'em Poker darstellt. Die beiden im Nachhinein vorgestellten Methoden erlauben es dann, die Ermittlung der Gewinnerhand auf einen einfachen und schnellen Vergleich von natürlichen Zahlen zurückzuführen. Die Aufgabe, aus einer Menge von Händen den Gewinner zu bestimmen, tritt wie in ‚5 Effiziente Handevaluation' beschrieben, häufig auf und muss daher äußerst schnell möglich sein. Da jedoch alle möglichen Handstärken vorausberechnet und persistent gespeichert werden sollen, spielt auch der Speicherverbrauch eine wichtige Rolle. Wäre dieser zu groß, könnte die Hashmap mit der Zuordnung von Hand zu Handstärke möglicherweise nicht komplett in den Arbeitsspeicher geladen werden. Es müsste also ständig von einem langsamen Massenspeicher nachgeladen werden, was den angestrebten Performancegewinn zunichtemachen würde. Selbst wenn die Tabelle komplett in den Arbeitsspeicher passt, wäre es immer noch wünschenswert möglichst wenig des Speichers zu verbrauchen, um die Ressourcen nicht für andere Aufgaben zu blockieren.

Es wird nun zunächst eine Codierung der Handstärke von Steve Brecher vorgestellt. Diese Methode sieht vor, die Handstärke in einem Integer mit 32 Bit zu codieren. Danach wird eine für das Framework neu entwickelte Codierung beschrieben, die es erlaubt den Speicherbedarf zu halbieren, in dem die Stärke einer Hand in nur 16 Bit gespeichert wird.

5.3.2 Codierung von ‚Steve Brecher'

Steve Brecher teilt die 32 Bit, die ihm im Integer zur Verfügung stehen, in 4 Blöcke zu je 4 Bit und einem Block mit 16 Bit auf. Der erste Block im Code ist immer 0 und hat daher keine weitere Bedeutung für die Handstärke. Im 2. Block (Block ‚V') wird die von Brecher sogenannte ‚HandCategory' gespeichert. Diese entspricht dem HandRank in dem neuen Framework und stellt damit die grobe Einteilung der Hände von High Card bis Straight Flush da. Der 3. Block repräsentiert im Falle eines Two Pairs den Rang des 1. Pärchens. Der Rang wird in diesem Block, genau wie im darauffolgenden Block ‚B', als natürliche Zahl gespeichert. Die Bitkombination ‚0101' hat in diesem Block beispielsweise die Semantik ‚5'. Handelt es sich nicht um ein Two Pair so ist dieser Block gleich 0. Block ‚B' (der 3. Block) beschreibt wieder wie Block T abhängig von der HandCategory einen bestimmen Rang. Im Falle von einem Vierling beschreibt er den Rang des Vierlings, handelt es sich um ein Full House so wird der Wert des darin enthaltenen Drillings gespeichert. Bei einer Straight oder einem Straight Flush wird der Rang der höchsten Karte in der Straße gespeichert (bei ‚KhTd9c8d7s6d2s' z.B. ‚1010' für ‚10'). Im Fall eines Two Pairs wird das niedrigere Paar (analog zum höheren Paar im Block T) gespeichert. Da ein Ass als ‚1' codiert wird, kann dieser Block niemals diesen Wert annehmen, da das 2. Pärchen eines Two Pairs niemals 2 Asse sein können. Der Wertebereich ist hier also 2-14. Handelt es sich um ein einfaches Paar wird der Rang des Paares gespeichert, ansonsten ist der Block gleich 0.

0000	VVVV	TTTT	BBBB	KKKKKKKKKKKKKKKK

Abbildung 9: Blockdarstellung des aufgeteilten Integers

Im letzten, 16-Bit großen Block, werden Ränge nicht mehr als natürliche Zahlen gespeichert. Stattdessen handelt es sich hier um ein Bitfeld bei dem jedes Bit jeweils einen Rang in der Hand beschreibt. Ist es ein Flush so werden die 5 Bits, die den Rängen der Karten im Flush entsprechen, gesetzt. Im Falle eines Pairs werden die 3 Kicker gespeichert, bei einem Drilling die beiden Kicker, bei einem Full House der Wert des darin enthaltenen Pärchens und bei einem Four Of A Kind oder einem Two Pair der Kicker. Tritt keiner dieser Fälle ein so ist der Block 0.

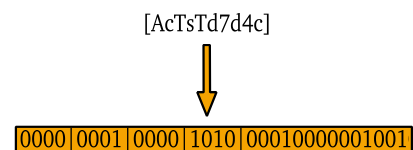


Abbildung 10: Beispiel einer Auswertung. Im 2. Block sieht man die Codierung von 'Pair', im 4. den Wert '10' für die Wertigkeit des Paares und im letzten Block die Bitmap der Kicker

Wie in dem Beispiel in Abbildung 10 zu erkennen ist, wird bei der Codierung, die im Framework von Steve Brecher genutzt wird, viel Platz verschwendet. So ist der 1. Block immer konstant 0 und z.B. im Falle eines Pärchens bleibt der 3.Block ebenfalls immer ungenutzt. Mit einer geschickteren Codierung lässt sich dieser Platzbedarf halbieren und dennoch weiterhin eine effiziente Auswertung realisieren.

5.3.3 Neue Codierung

In der neu entwickelten Codierung wird die Handstärke einer Hand in Feldern vom Typ ‚char' gespeichert. Dieser stellt 16 Bit zur Verfügung, die je nach Handrang in 4-Bit Blöcke eingeteilt werden um Wertigkeiten von ausgezeichneten Karten festzulegen oder als Bitmap genutzt werden, falls die Wertigkeiten vieler Karten gespeichert werden müssen. Die höherwertigen Bits des Datentyps werden genutzt, um den Handrang zu speichern. So wird sichergestellt, dass z.B. ein Straight immer einen niedrigeren Zahlenwert besitzt als beispielsweise ein Full House. Sollte der Handrang zweier Hände übereinstimmen, so ist durch die Anordnung der niederwertigen Bits sichergestellt, dass die bessere Hand ebenfalls wieder einen insgesamt höheren Zahlenwert erreicht. So kann durch einen einfachen Vergleich, der von den HandStrengthCodes repräsentierten Hände, bestimmt werden, welche die beste Hand ist, ohne den genauen Aufbau des Codes analysieren zu müssen. Abbildung 11 gibt einen Überblick über den neu entwickelten HandStrengthCode.

HandRank	3. Block				2. Block				1. Block				0. Block			
High Card	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X
Pair	Paar					3 Kicker										
Two Pair	1	0	0	1	höheres Paar				niedrigeres Paar				Kicker			
3 Of A Kind	1	0	1	0	Drilling				1.Kicker				2.Kicker			
Straight	1	0	1	1	High Card				0	0	0	0	0	0	0	0
Flush	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X
Full House	1	1	1	1	1	1	0	1	Drilling				Paar			
4 Of A Kind	1	1	1	1	1	1	1	0	Vierling				Kicker			
Straight Flush	1	1	1	1	1	1	1	1	High Card				0	0	0	0

Abbildung 11: Übersicht über die Codierung der Handstärke

Der 3. und 2. Block werden hierbei teilweise komplett für die Codierung des HandRanks benutzt, wie z.B. beim Full House der Code ‚1111 1101'. Bei andern Kategorien wie z.B. ‚High Card' wird nur ein Teil des 3. Blocks verwendet, um den zugehörigen Code ‚000' zu speichern. Werden Blöcke als Ganzes benutzt, um beispielsweise keinen Kicker oder ein Paar zu speichern, so wird in dem jeweiligen Block der Wert der beschrieben Karte als natürliche Zahl gespeichert. Steht beim Full House im 1. Block die Bitfolge ‚1011' so bedeutet das, dass es sich bei dem im Full House enthaltenen Drilling um einen Buben-Drilling handelt. Bei den HandRanks ‚High Card' und ‚Flush' müssen die Werte aller 5 Karten gespeichert werden, um die Handstärke eindeutig zu definieren. Da hierfür kein Platz in einem 16-Bit Feld ist, wenn man die Werte als Zahl speichern wollte, wird anstatt der Blockunterteilung eine Bitmap eingesetzt. Hier werden alle Bits der korrespondierenden Karten gesetzt, die in der Hand enthalten sind. Alle anderen Bits werden auf 0 gesetzt. Ist bei einem Flush z.B. ein Ass enthalten (also ein ace-high Flush) würde der 3. Block den Inhalt ‚1101' haben, die ‚110' für den Flush und die folgende 1 als Markierung für das enthaltene Ass.

Problematisch wird die Speicherung eines einfachen Paares. Hier gibt es ein ausgezeichneten Wert (nämlich der Wert des Paares) und 3 weitere Werte, die Kicker. Da der Wert des Paares unterscheidbar von den Kickern gespeichert werden muss ist es nicht möglich, diese Information in einem Bitfeld zu codieren. Die Verwendung der blockorientierten Speicherung ist allerdings auch nicht möglich, da hierfür 4 komplette Blöcke notwendig wären, aber ein Teil der Blöcke immer für den HandRank verbraucht wird. Es muss also eine wesentlich kompliziertere Speicherung angewendet werden. Ausgenutzt wird hierbei, dass die Wertigkeit der Karten nicht den Wertebereich eines kompletten Blocks verbrauchen. Gültige Werte für Kartenränge gehen nur von 2-A, also von 0-12. Ein 4-Bit Block bietet aber die Möglichkeit natürliche Zahlen von 0-15 zu speichern. Zudem ist es so, dass ein Wert, der als Wert des Pärchens bereits gespeichert wurde, nicht mehr als Kicker vorkommen kann. Handelt es sich um ein Damen-Pärchen, so kann keine Dame als Kicker vorkommen, da es ansonsten kein Pärchen sondern ein Drilling wäre. Ebenfalls kann der Wert eines Kickers nicht doppelt vorkommen. War der 1. Kicker ein König, so können 2. und 3. Kicker aus demselben Grund kein König mehr sein. Der Wertebereich schrumpft also noch weiter. Es muss allerdings darauf geachtet werden, dass jedes Paar einen insgesamt höheren Wert besitzt als jede Hand mit dem Rang ‚High Card' und einen niedrigeren Wert als jedes Two Pair. Dies wird mit der Codierung des Pärchens erreicht, welches mit ‚00100'-‚10000' im 3. Block und mit dem höchstwertigen Bit des 2. Blocks gespeichert wird. Es stehen dann noch 11 Bit ohne Beschränkungen zur Verfügung die genutzt werden, um die 3 Kicker zu speichern. In 11 Bits kann eine Zahl zwischen 0 und 2047 gespeichert werden. Das Polynom, welches die Kicker repräsentiert, darf diese Grenzen also nicht überschreiten, da sonst nicht ausreichend viel

Speicherplatz zur Verfügung steht. Mit der Annahme, dass der Wert des Paares nicht als Kicker vorkommen kann, leistet dies die Formel

$$1.Kicker * 13^2 + 2.Kicker * 13 + 3.Kicker$$

wobei der Wertebereich für ‚1. Kicker' von 0-11 geht. Es werden dabei die Karten wie gewöhnlich durchnummeriert. Der Wert, der bereits beim Paar codiert ist, wird dabei allerdings übersprungen. So bedeutet z.B. der Wert ‚9' beim 1. Kicker, dass es sich um eine 9 handelt, wenn der Wert des Paares größer war als 9. War der Wert kleiner handelt es sich beim 1. Kicker um eine zehn. Somit ist es auch möglich den komplexen Fall eines Paares in einem 16-Bit Datenfeld zu speichern.

5.4 Auswertung und Optimierung

Im Verlauf der Entwicklung des Frameworks hat sich gezeigt, dass die Nutzung von geordneten Listen im Zusammenhang mit Mengen von Karten einige Performanceprobleme mit sich bringt. Bei allen Operationen auf Kartensammlungen mussten immer eine schwergewichtige Liste bearbeitet werden. Z.B. beim Hinzufügen oder Entfernen von Karten aus einem Deck oder der Überprüfung, ob eine Karte bereits in einem bestimmten Stapel vorhanden ist oder nicht. Zudem mussten die Listen bei speziellen Aufgaben (z.B. beim Evaluieren der Handstärke) zunächst sortiert werden, um dann anschließend die eigentliche Aufgabe effizient durchführen zu können.

Daher wurde die interne Repräsentation der einem Stapel zugeordneten Karten geändert. Statt einer geordneten Liste wird ein Array mit booleschen Werten genutzt. Das Array ermöglicht es z.B. in konstanter Zeit zu prüfen, ob eine bestimmte Karte einem Deck bereits zugeordnet ist. Es müssen hierzu keine aufwendigen Vergleiche von Objekten durchgeführt werden sondern nur der boolesche Wert an der zur Karte zugehörigen Stelle geprüft werden. Auch das Sortieren der Sammlung entfällt, da die Bitmap bereits immer nach Farbe und Rang der Karten sortiert ist.

Aufgrund dieser grundlegenden Designänderung muss auch das Konzept des HandRanks neu überdacht werden, da der HandRank für geordnete Listen von Karten konzipiert war.

Mit geordneten Listen wird nur noch eine ungeordnete Menge der assoziierten Karten gespeichert. Daher erhöht sich der Anteil unerreichbarer Felder in der Lookup-Tabelle enorm. Dies ist darauf zurück zu führen, dass Kombinationen wie z.B. [2K5T7], [25KT7] oder [7K5T2] nicht mehr möglich sind. Alle diese Kartenkombinationen (welche eine ‚2', einen ‚K', eine ‚5', eine ‚T' und eine ‚7' enthalten) werden in der Äquivalenzklasse [257TK] zusammengefasst. Es wird also keine Unterscheidung mehr zwischen den genannten Kombinationen getroffen. Daraus resultiert, dass alle Kombinationen einer Äquivalenzklasse denselben HandRank besitzen und damit wesentlich weniger gültige HandRanks existieren. In Abbildung 12 ist anhand des genannten Beispiels konkret illustriert, welche HandRanks mit der neuen Darstellung der Karten als Bitmap nicht mehr gültig sind.

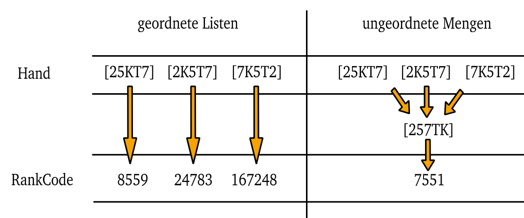


Abbildung 12: Reduzierung der möglichen HandRanks durch den Wechsel von Listen auf ungeordnete Mengen

Aus dieser Bildung von Äquivalenzklassen und der Reduktion gültiger HandRanks entstehen große Lücken in den Lookup-Tabellen. Waren wie oben beschrieben bei einer 5-Karten Hand mit geordneten Listen nur 0,003% der Felder nicht erreichbar so sind es mit der neuen Darstellung 365.118 - was über 98% der Gesamtmenge an Feldern entspricht. Es sollte daher ein neues System für die Indizierung der Kartenmengen evaluiert werden, die die Kartenmengen ohne große Lücken indiziert, aber dennoch von den Farben der Karten abstrahiert.

6 Metriken

6.1 Einleitung

Metriken sind Mittel zur Bewertung einer gegebenen Situation. Im Pokerkontext werden Metriken insbesondere dafür eingesetzt, um abzuschätzen, wie stark eine Pokerhand im Vergleich zu den Gegnern ist, um daraus profitable Aktionen abzuleiten.

Wie in der Einleitung beschrieben wurde, soll das Framework einige Metriken zur Verfügung stellen, die dann nicht mehr von den einzelnen Pokerbot-Entwicklern selbst implementiert werden müssen. Die im Framework implementierten Metriken werden im Folgenden genauer erläutert. Hierzu wird für jede Metrik zunächst ihre Semantik beschrieben. Es wird darauf eingegangen, was die berechnete Metrik bedeutet, welche Eigenschaften sie besitzt und wie sie bei der Ermittlung eines guten Spielzugs benutzt werden kann.

Zu beachten ist, dass alle vorgestellten Metriken (außer einer Varianten der All-In Equity) zunächst objektive Einschätzungen sind. Das heißt, es werden keinerlei Vermutungen über den Spielzustand vorgenommen, die aufgrund der Eigenschaft der unvollständigen Informationen für den Spieler nicht sichtbar sind. Hierzu gehören beispielsweise die Karten der Gegner, die der Spieler grundsätzlich nur aufgrund von verschiedenen Erfahrungen und Spielbeobachtungen abschätzen kann. Es werden in solch einem Fall immer alle möglichen Kombinationen an Holecards der Gegner, die theoretisch möglich wären, angenommen. Diese Annahme führt zwar zu objektiven Schlüssen, diese sind aber meistens recht weit entfernt von der Realität. Auch ohne zu viel der Objektivität zu verlieren kann man beispielsweise annehmen, dass ein Spieler der vor dem Flop mehrfach geraist hat, in der Regel nicht [7s2c] halten wird. Es bietet sich daher an, die beschriebenen Metriken mit einer einfachen und immer noch recht allgemein gehaltenen Gegnermodellierung zu verbinden. Es sollten hierbei allerding keine zu starken Annahmen getroffen werden, da sonst aus einer Metrik schnell eine subjektive Einschätzung der Situation entsteht für die die Pokeragenten zuständig sind. So wäre es beispielsweise möglich die Metriken so zu erweitern, dass für die Gegner nicht mehr zufällige Karten angenommen werden sondern abhängig von den gemachten Aktionen nur noch ein Bereich an möglichen Karten in Betracht gezogen wird. Man könnte annehmen, dass ein Gegner der vor dem Flop erhöht hat, keine der schwächsten 50% aller möglichen Karten halten wird. So eine Abschätzung ist, wenn der Bereich der Karten nicht zu eng gewählt wird, nur sehr selten falsch und liefert dafür aber eine wesentlich höhere Aussagekraft der berechneten Metrik.

6.2 Immediate Handrank

Die erste hier beschriebene Metrik ist der so genannte ‚Immediate Handrank'. Der Immediate Handrank stellt eine Aussage über die aktuelle Handstärke da, das heißt ohne Betrachtung von zukünftigen Ereignissen, wie dem Austeilen weiterer Straßen. In Relation gestellt werden die eigenen Handkarten zu allen möglichen Karten der Gegner. Hierzu werden das aktuelle Board und die Holecards von Hero genommen und gegen alle Kombinationen möglicher Gegnerhände ausgewertet. Gewinnt Hero eine dieser Auswertungen bekommt er einen Punkt, verliert er bekommt er keinen Punkt. Gibt es ein Unentschieden, an dem Hero beteiligt ist, bekommt er, je nach Anzahl der sich den Pot teilenden Spieler, anteilig Teilpunkte für die Hand. Die Metrik berechnet sich im Anschluss dieser Simulation durch

$$IHR = \frac{Anzahl\ erreichte\ Punkte}{Anzahl\ gemachte\ Spiele}$$

Der Immediate Handrank bewegt sich folglich zwischen 0 und 1, wobei 0 bedeutet dass gegen sämtliche mögliche Gegnerhände jede Auswertung verloren wurde. Der Wert 1 wird erreicht, wenn

alle Spiele gewonnen wurden. Der IHR gibt damit also keinerlei Auskunft über das Handpotential. So würde beispielsweise die Hand [Ks2s] bei einem Board wie [7s6cAs] eine relativ schlechte Bewertung bekommen, da man zurzeit nichts auf dem Board getroffen hat. Die Möglichkeit, die sich durch den Nutflush-Draw ergibt, sich zur bestmöglichen Hand zu verbessern, wird komplett außer Acht gelassen. Der IHR stellt somit keine Möglichkeit zur Verfügung, um das Potential einer Hand zu bewerten, sondern er bewertet ausschließlich die momentane Handstärke.

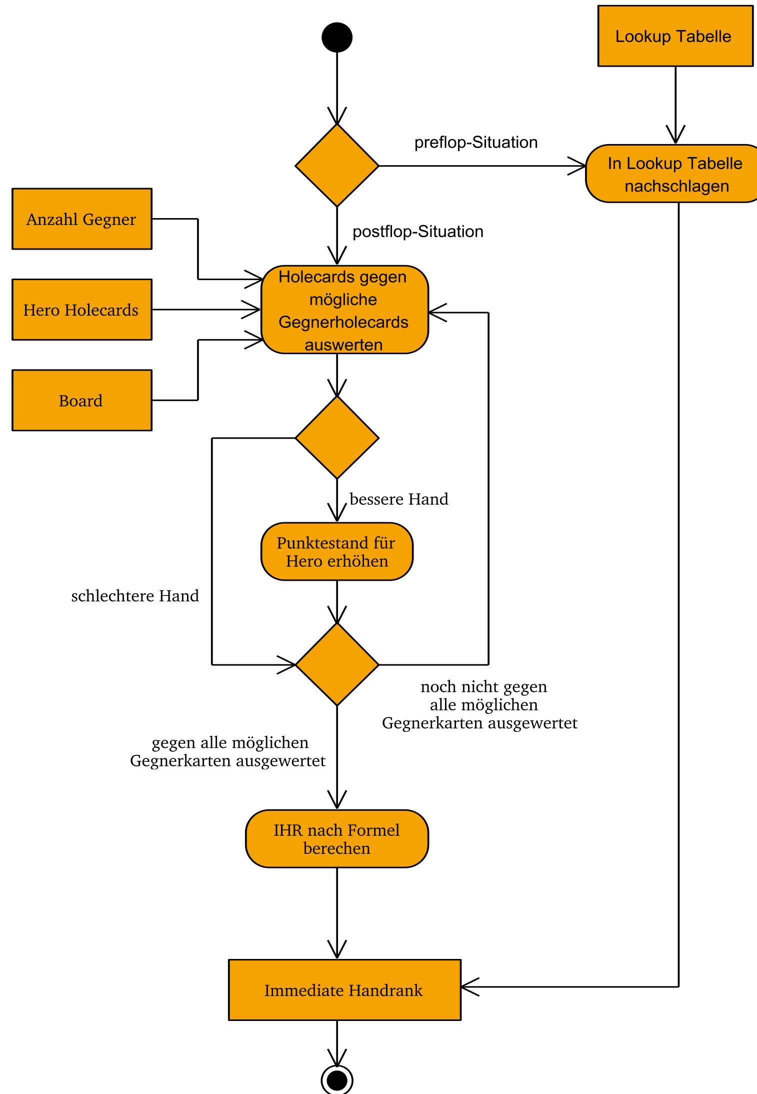


Abbildung 13: Algorithmus zum Berechnen des Immediate Handranks

Zudem stellt sich die Frage, wie die Metrik vor dem Flop berechnet werden soll. Dass Karten wie [AsAc] gegen beispielsweise [9c3s] einen Punkt bekommen, weil sie besser sind als die anderen, lässt sich noch recht leicht argumentieren. Aber bei Holecards wie [8s5c] und [9d4h] lässt sich dies nicht mehr so eindeutig sagen. Hierfür wurde eine etwas andere Definition des Immediate Handranks verwendet. Bei preflop-Situationen werden nicht nur für alle Gegner alle möglichen Kartenkombinationen angenommen, sondern auch alle möglichen Kombinationen für die Gemeinschaftskarten durchiteriert. Gewinnen die Holecards A von Hero gegen die Holecards B mehr als 55% aller Spiele mit zufälligen Boardcards, so werden die Holecards A als stärker eingeschätzt als die Karten B und bekommen damit einen Punkt. Gewinnen B mehr als 55% werden den Karten A keine Punkte angerechnet. Tritt keiner der beiden Fälle ein, wird der Durchlauf als Unentschieden gewertet und die Karten von Hero bekommen wieder anteilig Punkte. Diese Punkte werden dann, wie oben in der Formel zur Berechnung des IHR beschrieben, umgerechnet. Analog erfolgt die Berechnung gegen 2

Gegner. Der einzige Unterschied ist, dass sich die Karten von Hero nun gegen 2 Gegner behaupten müssen um Punkte für die Metrik zu sammeln.

Da der Vorgang zur Ermittlung der preflop-Immediate Handranks durch die Vielzahl der auszuwertenden Kombinationen sehr viel Zeit kostet wurden diese im Vorfeld vorausberechnet und in Lookup-Tabellen gespeichert, so dass während der Laufzeit des Pokerbots nur noch ein schneller Lookup und keine aufwendige Berechnung mehr erfolgen muss.

6.3 Seven-Card Handrank

Die 2. implementierte Metrik ist der ‚Seven-Card Handrank'. Der Seven-Card Handrank gibt im Gegensatz zum Immediate Handrank eine Auskunft über das Potential einer Hand. Der Handrank bewertet also nicht unbedingt die aktuelle Stärke der Karten, sondern über die mögliche Stärke der Karten wenn die Hand bis zum Showdown gespielt wird. Hierfür werden wieder, ähnlich wie beim IHR, die Holecards von Hero, die bisher aufgedeckten Gemeinschaftskarten und die Anzahl der Gegner mit in die Berechnung einbezogen. Zunächst werden die Boardkarten bis zum Showdown ‚aufgefüllt'. Das heißt falls beispielsweise für eine Situation am Flop der Seven-Card Handrank berechnet werden soll, werden den 3 bisher bekannten Boardkarten 2 weitere hinzugefügt, die als potentieller Turn und River möglich sind. Das Board besteht dann aus 5 Karten. Mit diesen teilweise zufällig gewählten Gemeinschaftskarten wird für die auszuwertende Situation der Immediate Handrank berechnet. Der so berechnete Wert wird zu einer Summe hinzugezählt. Dieses zufällige Auffüllen und Auswerten wird mehrfach wiederholt. Anschließend wird die Summe durch die Anzahl der Durchläufe geteilt und man erhält den Seven-Card Handrank. Die zugehörige Formel lautet

$$SCHR = \frac{\sum Immediate\ Handranks}{Anzahl\ der\ Durchläufe}$$

Der Seven-Card Handrank kann also bei der Pokerbot-Entwicklung dafür eingesetzt werden, um die zukünftige Chance eine gute Hand zu halten, einzuschätzen. Der Wertebereich der Metrik ist aufgrund des Wertebereichs des IHRs wieder zwischen 0 und 1. Den Wert 0 bekommen die Holecards, die alle Showdowns verlieren würden, der Wert 1 wird den Holecards zugewiesen, die unabhängig von den zukünftigen Boardkarten gegen alle anderen Holecards gewinnen. Dies ist der Fall, wenn man uneinholbar die bestmögliche Hand hält. Jedoch sind die Randwerte 0 und 1 kaum erreichbar, da es extrem unwahrscheinlich ist in eine Situation zu kommen, in der man von keiner Hand überholt werden kann oder in der man gegen sämtliche Karten chancenlos ist.

Der Seven-Card Handrank kombiniert also durch die Simulation der möglichen zukünftigen Gemeinschaftskarten die positiven Entwicklungsmöglichkeiten einer Hand (man liegt zurzeit hinten und hat die Chance sich zu verbessern) mit dem negativem Potenzial der Holecards (man führt zurzeit mit der Hand aber Gegnerhände haben eine gute Chance die Hand noch zu überholen). Diese gleichgewichtete Kombination ist allerdings nicht sehr realistisch, da davon ausgegangen wird, dass Hände mit großem positivem und großem negativem Potenzial immer bis zum Showdown gespielt werden. Dies ist aber keine gute Annahme, da insbesondere schlechte Hände ohne positive Entwicklungsmöglichkeiten nur sehr selten bis zum Showdown gespielt werden. Im Vergleich dazu werden beispielsweise auch recht schwache Hände, die allerdings ein großes Potenzial haben sich noch im Laufe des Spiels zu einer starken Hand zu verbessern, sehr oft zumindest bis zum River gespielt. Man muss also bei der Benutzung des Seven-Card Handranks beachten, dass dieser Hände mit schlechtem Potenzial, die normalerweise schon früh gefoldet werden würden, überschätzt. Auf der anderen Seite werden starke Hände mit großem negativem Potenzial unterschätzt, da auch hier immer davon ausgegangen wird, dass die Hand bis zum Showdown gespielt wird. In der Regel ist es aber so, dass Gegner mit schlechteren Händen durch aggressives Setzen zum folden gebracht werden können, so dass sich das negative Potenzial der Hand nicht auswirkt.

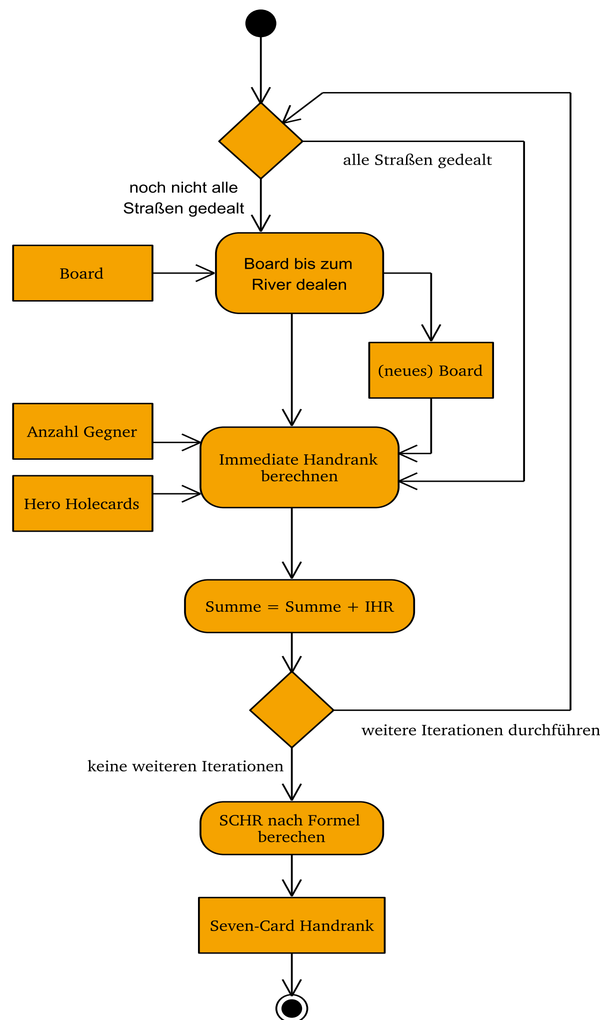


Abbildung 14: Algorithmus zum Berechnen des Seven-Card Handranks

6.4 Effective Handrank

Um den eben beschriebenen Effekt des Seven-Card Handranks, gewisse Karten zu über- bzw. zu unterschätzen, zu vermindern kann die 3. Metrik, der ‚Effective Handrank', genutzt werden. Dieser kombiniert die Eigenschaften des Immediate Handranks (dem Einschätzen der aktuellen Stärke der Hand) und dem Seven-Card Handrank, der in der Lage ist positives und negatives Potenzial einer Hand zu bewerten. Der Handrank wird relativ einfach berechnet, in dem das Maximum der beiden ersten Metriken gebildet wird:

$$EHR = Maximum(SCHR, EHR)$$

Der Effective Handrank kann also gut dafür eingesetzt werden, um eine Einschätzung über die Stärke der eigenen Hand zu gewinnen. Mit dem weiter oben beschriebenen Einsatz einer einfachen Gegnermodellierung lassen sich die Metriken in ihrer Aussagekraft jedoch noch wesentlich verbessern.

Weitere Details zu den Metriken ‚Immediate Handrank', ‚Seven-Card Handrank' und dem ‚Effective Handrank' finden sich in [BK06, Seite 6f.].

6.5 All-In Equity

Die 4. Metrik, die häufig Verwendung findet, ist die ‚All-In Equity'. Die All-In Equity wird berechnet indem die Holecards von Hero und die Holecards der Gegner verglichen werden. Hierbei werden die Boardkarten, ähnlich wie beim Seven-Card Handrank, wieder bis zum Showdown in mehreren Durchläufen simuliert. Die Anzahl der Siege wird dann durch die Anzahl der Iterationen geteilt und man erhält die All-In Equity. Diese beschreibt somit in jeder Iteration die Situation, die auftreten würden, wenn alle Spieler ‚All-In' gehen würden. Da diese Auswertung über alle möglichen zukünftigen Boardkarten gemacht wird entspricht die All-In Equity dem Prozentsatz der im Durchschnitt zu gewinnenden Spiele. Das heißt eine All-In Equity von 60% zeigt an, dass man den Pot im Durchschnitt über alle möglichen zukünftigen Karten auf Flop, Turn oder River mit 60%iger Wahrscheinlichkeit gewinnt. Berechnet wird die Metrik also durch

$$AIE = \frac{Anzahl\ Gewonnene\ Spiele\ am\ Showdown}{Anzahl\ der\ Durchläufe}$$

Nimmt man beispielsweise an, dass der Gegner in einer Heads-Up Situation ein mittleres Pärchen wie [TdTh] auf der Hand hat und Hero hält [AcKd], so besitzt er damit eine Wahrscheinlichkeit von 43,1% das Spiel zu gewinnen falls bis zum Showdown gespielt wird und keiner der beiden Spieler foldet. Im Unterschied zu den voran gegangenen Metriken werden hier konkrete Gegnerhände abgeschätzt bevor man die Berechnung der Metrik beginnt. Verwenden kann man die All-In Equity also, falls man bereits einen Eindruck über die möglichen Hände der Gegner hat. Eine Erweiterung der Metrik ermöglicht aber auch mit zufälligen Gegnerhänden zu simulieren, falls man die Gegnerhände nicht exakt genug abschätzen kann.

6.6 Pot Equity

Multipliziert man die All-In Equity noch mit der aktuellen Potgröße, so bekommt man den Anteil des Pots, der Hero im Durchschnitt zustehen würde. Diese Größe wird die ‚Pot Equity' genannt. Hat man in einer Hand mehr Chips gewonnen als einem laut Pot Equity zugestanden hätten, so hat man die Hand besser gespielt, als man dies im Durchschnitt hätte erwarten können. Jedoch werden bei dieser Betrachtung viele weitere wichtige Details außer Acht gelassen. So spielt es beispielsweise auch eine wichtige Rolle, wie viel Chips im Verlauf der Hand noch in den Pot investiert werden. Hat man eine All-In Equity von 80% und keine Chance den Gegner auf Grund seiner schwachen Hand dazu zu bringen weitere Chips zu investieren, so kann dies eine schlechtere Situation sein als eine in der man nur 60% Equity besitzt, der Gegner aber noch viele Chips im Verlauf der Hand investieren wird.

$$PE = All - In\ Equity * aktuelle\ Potgröße$$

6.7 Roll-Out Equity

Ein möglicher Ansatz das eben beschriebene Problem zu lösen ist die Verwendung der so genannten ‚Roll-Out Equity'. Hierbei wird versucht zu bestimmen, wie viele Chips bis zum Showdown noch in den Pot wandern werden. Es kann beispielsweise vereinfachend angenommen werden, dass pro Spieler und Setzrunde noch ein Chip zu dem Pot hinzugefügt wird. Mit einer ausgereifteren Gegnermodellierung kann diese Annahme noch weiter verfeinert werden, um die tatsächliche Potgröße zum Zeitpunkt des Showdowns besser abzuschätzen.

$$ROE = All - In\ Equity * geschätze\ Potgröße\ am\ Showdown$$

6.8 Zusammenfassung

Der ‚Immediate Handrank', der eine Einschätzung über die unmittelbare Handstärke liefert, und der ‚Seven-Card Handrank', der positives und negatives Potential einer Hand bewerten kann, bilden eine gute Kombination, ausgedrückt durch den ‚Effective Handrank', um eine Aussage über die Stärke einer Pokerhand zu geben. Die ‚All-In Equity' hat eine dem SCHR ähnliche Semantik. Durch die zusätzliche Betrachtung der Potgröße gewinnt man einen Eindruck darüber, wie viele Chips einem Spieler durchschnittlich zustehen. Dies wird durch die ‚Pot Equity' erfasst. Die ‚Roll-Out Equity' verbessert dieses Konzept, indem nicht der aktuelle Pot betrachtet wird, sondern eine geschätzte Potgröße zum Zeitpunkt des Showdowns.

Zusammenfassend kann gesagt werden, dass keine der Metriken allein geeignet ist um die Lage, in der sich Hero befindet, exakt zu beschreiben. Die Kombination mehrerer dieser Metriken kann allerdings schon gute Rückschlüsse auf die zu tätigenden Aktionen geben. Ein gutes Opponent-Modeling kann die Aussagekraft der Metriken zudem noch erheblich verbessern.

7 UCT-Simulation mit Opponent Modeling

7.1 Einleitung

Während der TUD Computer Poker Challenge entstand bei verschiedenen Gruppen die Idee, die Entscheidungsfindung (also welcher Zug am profitabelsten ist) mit Hilfe eines Suchbaums zu treffen. Hierbei wird der Spielverlauf durch einen Baum repräsentiert. Die Knoten des Baums können bei dem Spiel Poker verschiedener Natur sein. Zunächst gibt es die ‚normalen' Knoten, die auch in anderen Spielen wie z.B. Schach zu finden sind. Bei diesen Knoten handelt es sich um Situationen im Spiel, bei denen ein Spieler einen Zug tätigen muss. Ein Spieler hat meist die Wahl zwischen fold, call und raise. Ein Spielerknoten hat daher in der Regel 3 Kinder. Die einzige Ausnahme bilden hier die Situationen, in denen ein raise aufgrund des Caps nicht mehr möglich ist. Eine weitere Knotenart wird durch das zufällige Austeilen einer Gemeinschaftskarte nötig. Hierbei können die Spieler nicht den im Baum eingeschlagenen Pfad bestimmen, sondern es kann lediglich berechnet werden mit welcher Wahrscheinlichkeit ein Pfad eingeschlagen werden wird. Beim Pokern sind die Kinder eines solchen Knotens alle noch nicht ausgeteilten Karten, die alle die gleiche Wahrscheinlichkeit haben ausgewählt zu werden. Die letzte Knotenart sind die Blattknoten des Baums. Sie werden eingesetzt um das Ende eines Spielpfades zu markieren. Diese Situation tritt ein, wenn entweder alle Spieler bis auf den letzten verbliebenen gefoldet haben oder es zum Showdown kommt.

Aufgrund des äußerst großen Zustandsraums kann jedoch nicht der komplette Baum evaluiert werden. Es ist sogar so, dass nur ein relativ kleiner Teil ausgewertet werden kann. Nun stellt sich die Frage, welchen Teil man auswertet, um möglichst gute Vorhersagen über den zukünftigen Spielverlauf treffen zu können. Hierbei kann der UCT-Algorithmus genutzt werden, der im Folgenden kurz erläutert wird. Dieser Algorithmus löst dabei das so genannte ‚Exploration - Exploitation'-Dilemma [KS07], bei dem es zum Trade-Off zwischen Nutzung von bereits bekannten Wissen und Generierung von neuem Wissen kommt. Konkret heißt das im Falle von Poker, ob ein vielversprechender Zug genauer untersucht werden soll oder ob man besser andere Züge genauer betrachten soll, da diese eventuell noch besser sein könnten.

7.2 Funktionsweise

Der UCT-Algorithmus ist, wie bereits erwähnt, eine Strategie zum Aufbauen und Auswerten eines Spielbaums. Diese Strategie gliedert sich in 4 Phasen: der Selektion eines (zu expandierenden) Knotens, der Expansion dieses Knotens, der Simulation der entstanden Kinder und der Backpropagation der neuen Erkenntnisse.

Während der Selektionsphase wird der Baum zunächst bis zu einem Blattknoten durchlaufen. Die Auswahl des nächsten Kindknotens auf dem Pfad zu dem Blattknoten wird dabei mit Hilfe des so genannten ‚Upper Confidence Bound' bestimmt. Für jeden Knoten berechnet sich dieser Wert nach der Formel

$$ucb = value + C * \sqrt{\frac{\ln(parent_passes)}{passes}}$$

Hierbei ist ‚value' der aktuelle Wert eines Knotens. Im Fall von Poker also die Anzahl der in dieser Situation zu gewinnenden Chips aus der Sicht des eigenen Spielers. ‚parent_passes' ist die Anzahl der Besuche des Elternknotens, während ‚passes' die Anzahl der Besuche des momentanen Knotens darstellt. ‚C' ist eine Konstante, die die Explorationsrate beeinflusst.

Der Upper Confidence Bound errechnet sich also durch den aktuellen Wert eines Knotens und ist zusätzlich davon abhängig wie oft der Knoten bisher besucht wurde. Der aktuelle Wert wird genutzt um den Anteil der Exploitation in die Entscheidung mit einzubeziehen. Ist ein Knoten für den Spieler sehr wertvoll sollte dieser auch oft besucht werden, um einen guten Nutzen aus dieser Information zu gewinnen. Die Anzahl der Besuche bestimmt wie sicher die Information über den Wert ist. Wurde der Knoten im Verhältnis zu seinem Vater selten besucht, kann die Information über den Wert noch sehr vage sein. Es lohnt sich also den Knoten weiter zu untersuchen. Dieser Anteil des UCB-Wertes spiegelt also den Explorationsnutzen wieder. Die Konstante C gewichtet den Explorationsnutzen im Verhältnis zum Exploitationsnutzen und bestimmt damit wie ‚neugierig' der Algorithmus ist. Bei großem C wird ein größerer Wert auf eine breite Untersuchung des Baums gelegt während bei kleinem C der Baum eher in die Tiefe, entlang der schon als gut beurteilten Knoten, ausgewertet wird.

Diese Selektion wird nun so lange durchgeführt bis entweder ein noch nicht expandierter Knoten oder ein Knoten an dem das Spiel beendet ist erreicht wird. Wird ein Blattknoten erreicht, der nicht mehr weiter ausgewertet werden kann (weil das Spielende erreicht ist) wird der nächste Durchlauf des Algorithmus gestartet. Kann der Knoten expandiert werden beginnt die 2. Phase.

Während der Expansionsphase werden nun Kindknoten des zu expandierenden Knotens erstellt. Hierbei kann es möglich sein nur einen oder eine begrenzte Anzahl an Kindern zu generieren. Bei einem Zufallsknoten, der einige dutzend Kinder (anhängig von der Anzahl der verbliebenen Karten) besitzen kann, wäre eine solche Strategie denkbar. Es ist jedoch auch möglich direkt alle Kinder zu generieren, was die Implementierung des Algorithmus wesentlich vereinfacht.

Wurden Kinder erzeugt, muss nun der Wert dieser Kinder möglichst effizient geschätzt werden. Hierbei bietet es sich an, das Spiel mehrfach bis zum Ende mittels einer Monte-Carlo-Simulation zu simulieren. Der Wert des expandierten Knotens muss dann, abhängig von der Art des Knotens und der durch Simulation ermittelten Werte der Kinder, errechnet werden.

Bei der Backpropagation, der 4. Phase des UCT-Algorithmus, wird dann der aktualisierte Wert des Knotens bis zur Wurzel des Baums nach oben propagiert. Verändert sich der Wert eines Kindes muss der Wert des Vaters neu berechnet werden. Hierbei gibt es nun verschiedene Möglichkeiten wie dies geschehen muss. Handelt es sich bei dem Vater um ein Zufallsereignisknoten, bei denen alle Kinder gleichwahrscheinlich sind, kann einfach der Durchschnitt von den Werten aller Kinder gebildet werden, um den neuen Wert des Vaterknotens zu bestimmen. Ist der zu aktualisierende Knoten ein Gegnerknoten, bei dem ein Gegner einen Zug machen kann, ist es sinnvoll den neuen Wert des Knotens, ähnlich wie bei dem Zufallsknoten, nach Wahrscheinlichkeiten gewichtet zu bestimmen. Jedoch sind hier die 3 möglichen Aktionen des Gegners nicht gleich wahrscheinlich, sondern müssen aufgrund des aktuellen Spielzustands und dem Opponent Modeling des Gegners für jede Situation neu evaluiert werden. Bei Knoten, in denen der eigene Agent entscheiden kann welcher Pfad eingeschlagen wird, kann das Maximum aus den 3 Kindknoten ausgewählt werden. Denn hier würde der Bot mit Sicherheit den Pfad einschlagen, der den maximalen Profit bzw. den minimalen Verlust bringt. Weitere Details hierzu finden sich in [W08].

Da während der Challenge deutlich geworden ist, dass der Erfolg einer Simulation des Spiels sehr stark von einem guten Opponent Modeling abhängt, wurde bei der Implementierung des UCT-Verfahrens hierauf besonderen Wert gelegt. So macht es nur Sinn einen Baum bis zu den Blättern zu evaluieren, wenn man bei der Einschätzung der Gegneraktionen einigermaßen genau ist. Sollte beispielsweise das Opponent Model aussagen, dass der Gegner höchstwahrscheinlich auf einen Raise folden wird, er das aber im Spiel dann nicht auch wirklich zu einer hohen Wahrscheinlichkeit macht, kann die Simulation schnell einen falschen Eindruck der Situation liefern.

7.3 Implementierung und Beispiel

Die Implementierung der UCT-Simulation befindet sich wie bereits kurz in ‚2.3.7 pokertud.uct' beschrieben im Paket pokertud.uct. Grundlage für eine Simulation ist ein Spielzustand, von dem ausgehend die Simulation gestartet wird. Dieser Spielzustand ist also der Zustand der im Wurzelknoten des Baumes zu finden ist. In diesem Beispiel soll dieser Zustand durch den Matchstate-String

$$MATCHSTATE: 1: 1: crrcrcc/r: AcKh|Qc9s|2d2h/Ad4hJd$$

beschrieben sein. Hero befindet sich demnach auf der 1. Position und besitzt die Karten [Qc9s]. Als Gemeinschaftskarten bereits aufgedeckt sind [Ad4hJd]. Wir befinden uns also auf dem Flop und vor Hero hat ein Gegner bereits gesetzt. Dieser initiale Spielzustand wird der Klasse UCTSimulation beim Erstellen als Parameter übergeben. Wenn diese Klasse ausgeführt wird, wird eine Instanz der Klasse RootNode erstellt. Dieser Knoten bildet die Wurzel des Baums und bekommt ebenfalls den Spielzustand übergeben. Die RootNode erzeugt daraufhin sofort einen Kindknoten vom Typ HeroNode, da in dem aktuellen Zustand Hero am Zug ist.

Nach dieser Initialisierung startet die 1. Iteration des UCT-Algorithmus. In der 1. Phase wird ein Knoten selektiert, der sich am unteren Ende des Baums befindet. Da bisher nur der Wurzelknoten und die darunterliegende HeroNode existiert, wird diese ausgewählt und in der 2. Phase expandiert. Hero hat die 3 Möglichkeiten fold, call und raise, also wird die HeroNode um 3 Kindknoten expandiert. Falls

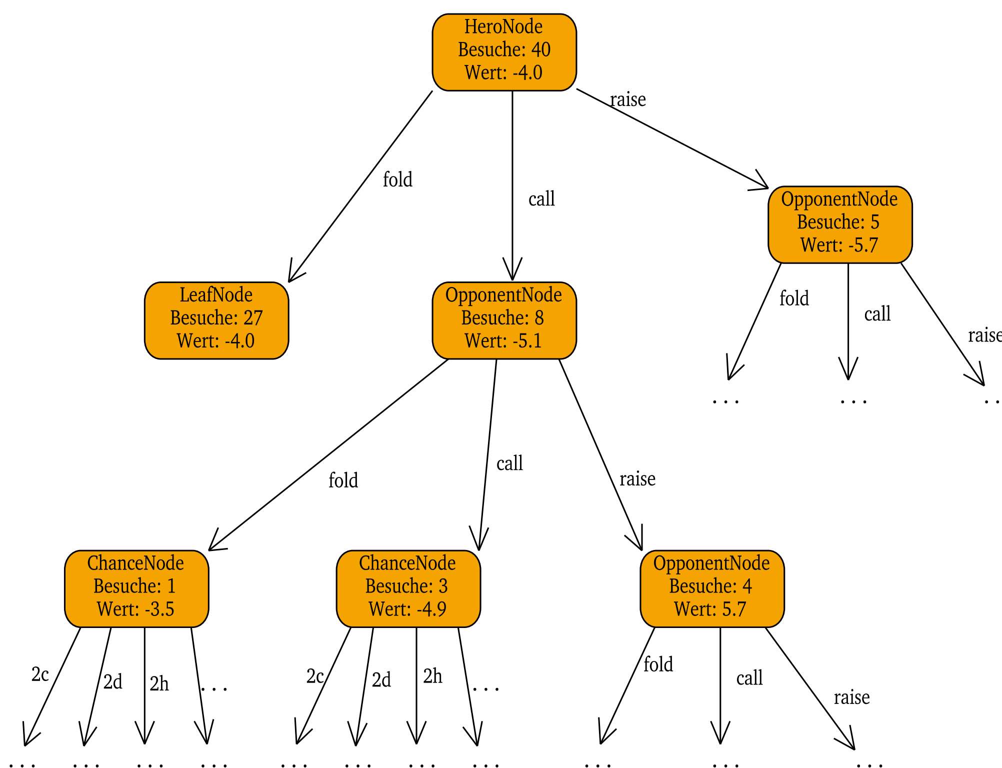


Abbildung 15: Beispiel eines Suchbaums der mit Hilfe des UCT-Verfahrens erstellt wurde

Hero foldet ist der Ausgang des Spiels aus Sicht von Hero bereits eindeutig: Er verliert seinen getätigten Einsatz. Der Pfad ist also am Spielende angekommen und es kann deshalb eine LeafNode erzeugt werden. Callt oder raist Hero jedoch ist als nächstes ein Gegner am Zug, deshalb sind diese Kinder Instanzen vom Typ OpponentNode.

Eine OpponentNode besitzt als Attribut immer ein so genanntes DecisionTriple, welches angibt wie wahrscheinlich es ist, dass der Gegner in der aktuellen Situation foldet, callt oder raiset. Diese Information wird benötigt um die Gewichtung der einzelnen Kindknoten bei der Berechnung des Wertes des Knotens zu bestimmen. Hierbei kommt das Opponent Model ins Spiel. Denn aufgrund der aktuellen Spielsituation muss bestimmt werden, wie Wahrscheinlich die genannten Handlungen eines bestimmen Gegners sind, um verwertbare Resultate aus der Simulation abzuleiten.

In der 3. Phase werden die Werte der Kindknoten dann durch eine Monte-Carlo-Simulation geschätzt. Die hierfür nötige Methode simulateToEnd() befindet sich in der abstrakten Klasse Node und simuliert, ausgehend vom aktuellen Zustand, das Spiel mehrfach bis zum Ende. Da auch hier zumindest ein annäherungsweise korrektes Gegnermodell benötigt wird, wird analog zur OpponentNode ein DecisionTriple der Gegner benötigt. Da hier allerdings nur grobe Angaben über das Verhalten des Gegners benötigt werden, kann ein vereinfachtes Model, welches den Gegner nur sehr primitiv und schnell approximiert, eingesetzt werden.

Die 4. Phase leitet die ermittelten Ergebnisse bis zum Wurzelknoten weiter. Die Neuberechnung der Werte erfolgt nach dem oben angegeben Schema.

Diese 4 Phasen werden dann wie beschrieben in vielen Iterationen wiederholt, bis ein verlässliches Ergebnis abgelesen werden kann. So ermittelte eine Simulation der oben angegeben Situation Besuchszahlen für den Knoten ‚fold' von 27 Besuchen, der Knoten ‚call' wurde 8 mal und der Knoten ‚raise' 5 mal besucht. Hieraus ergibt sich, dass folden in der aktuellen Situation von der Simulation als die beste Wahl für den nächsten Zug ausgewählt wurde. Konkret wurden die Erwartungswerte für einen Fold auf -4 Chips, für einen Call auf -5,1 Chips und für einen Raise auf -5,7 Chips festgelegt. Wenn man die Situation manuell beurteilt, würde man postflop mit großer Wahrscheinlichkeit auch feststellen, dass ein Fold hier eine gute Option für den nächsten Zug darstellt. Die Simulation hat also den vermeintlich korrekten Zug bestimmt.

8 Benutzungsbeispiel

Im folgenden Abschnitt soll nun erläutert werden wie man den Server startet, wie Client gestartet und beim Server angemeldet werden, wie Spiele begonnen und die Ergebnisse angezeigt werden können.

Begonnen wird mit dem Starten des Servers, da ohne ihn die Clients keinen Bezugspunkt haben zu dem sie sich verbinden können. Für den TUD-Server gibt es nur einen Betriebsmodus, das heißt er wird immer auf die gleiche Weise ohne weitere Parameter gestartet. Hierzu wird einfach die Klasse Server im Package pokertud.server oder alternativ die runServer.bat bzw. runServer.sh im Quellverzeichnis ausgeführt. Der Server gibt, falls er korrekt gestartet wurde, die Meldung „Server started!“ auf der Konsole aus. Sollte bereits ein anderer Server auf dem System laufen wird eine IOException geworfen und das Programm mit einer Fehlermeldung beendet.

Wurde der Server gestartet können Client gestartet werden, die sich mit dem Server verbinden können. Jeder Bot, der von PokerClient erbt, kann grundsätzlich in 3 verschiedenen Varianten ausgeführt werden. Welche Variante dies ist, hängt von den dem Pokerbot übergebenen Parametern ab. Werden 2 Parameter übergeben, nämlich die IP-Adresse gefolgt von dem Port des Servers, wird der Bot so gestartet, dass er sich zu dem unter der angegeben Adresse befindlichen Alberta-Pokerserver verbindet. Dieser Modus muss also immer dann verwendet werden, wenn der Alberta-Pokerserver eingesetzt werden soll. Wird der TUD-Server eingesetzt, so hat man die Wahlmöglichkeit zwischen 2 weiteren Betriebsmodi. Wird nur ein Parameter übergeben (die IP-Adresse des TUD-Pokerservers) wird der Client ohne GUI gestartet. Ein auf diese Weise gestarteter Client wird immer im Modus ‚passiv‘ ausgeführt. Das heißt der Bot kann keine eigenen Spiele starten und steht auf dem Pokerserver anderen aktiven Client als Gegner zur Verfügung. Mit dem 2. Betriebsmodus wird eine GUI gestartet, so wie sie in Abbildung 16 dargestellt ist.

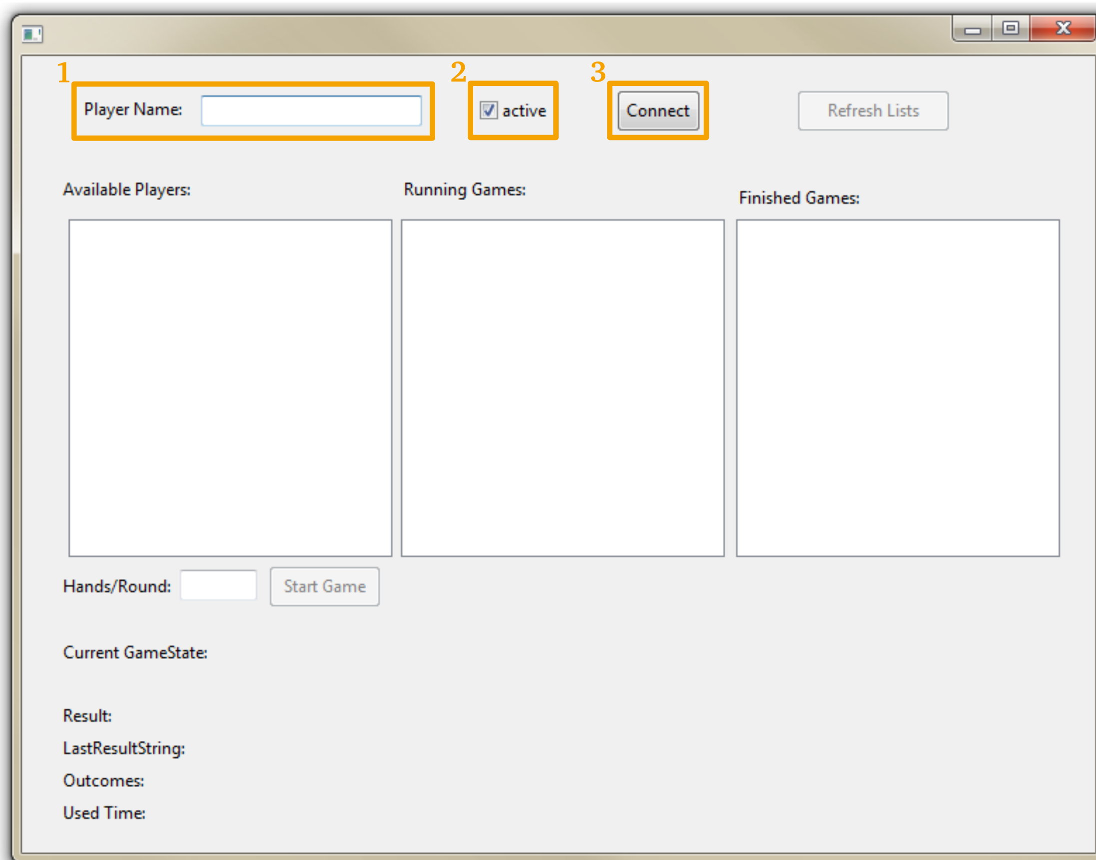


Abbildung 16: GUI nach dem Starten eines Pokerbots

In Feld 1 kann der Name des Bots angegeben werden unter dem er dann in den Listen im mittleren Bereich der GUI aufgelistet ist bzw. unter welchem Namen er in der Detailansicht ganz unten aufgeführt wird. Zu beachten ist hierbei, dass Namen mehrfach vergeben werden können. Interessiert man sich nicht für einen konkreten Bot, sondern nur dessen Spielweise, wie zum Beispiel im Falle des Beispielbots ‚RichieRich', so kann man einfach mehrere Instanzen des Bots ‚RichieRich' nennen. Mit der Checkbox ‚active' (Feld 2) kann ausgewählt werden, ob der Client als passiver Client, wie in der Konsolenvariante, gestartet werden soll oder ob es sich um einen aktiven Pokerbot handelt. Während wie bereits erwähnt es passiven Agenten nicht möglich ist Spiele zu starten verfügen aktive Client über diese Möglichkeit. Wie dies genau funktioniert wird weiter unten erläutert. Hat man die Auswahl über die Aktivität des Klienten getroffen verbindet man sich mit einem Klick auf den Button ‚Connect' in Feld 3 mit dem Server, sofern dieser korrekt initialisiert wurde und erreichbar ist. In dieser Version der GUI ist es noch nicht möglich eine IP-Adresse für den Server anzugeben. Standardmäßig wird hier die ‚127.0.0.1' als Zieladresse benutzt, was bedeutet, dass der TUD-Pokerserver unter der gleichen IP-Adresse erreichbar sein muss wie der Client.

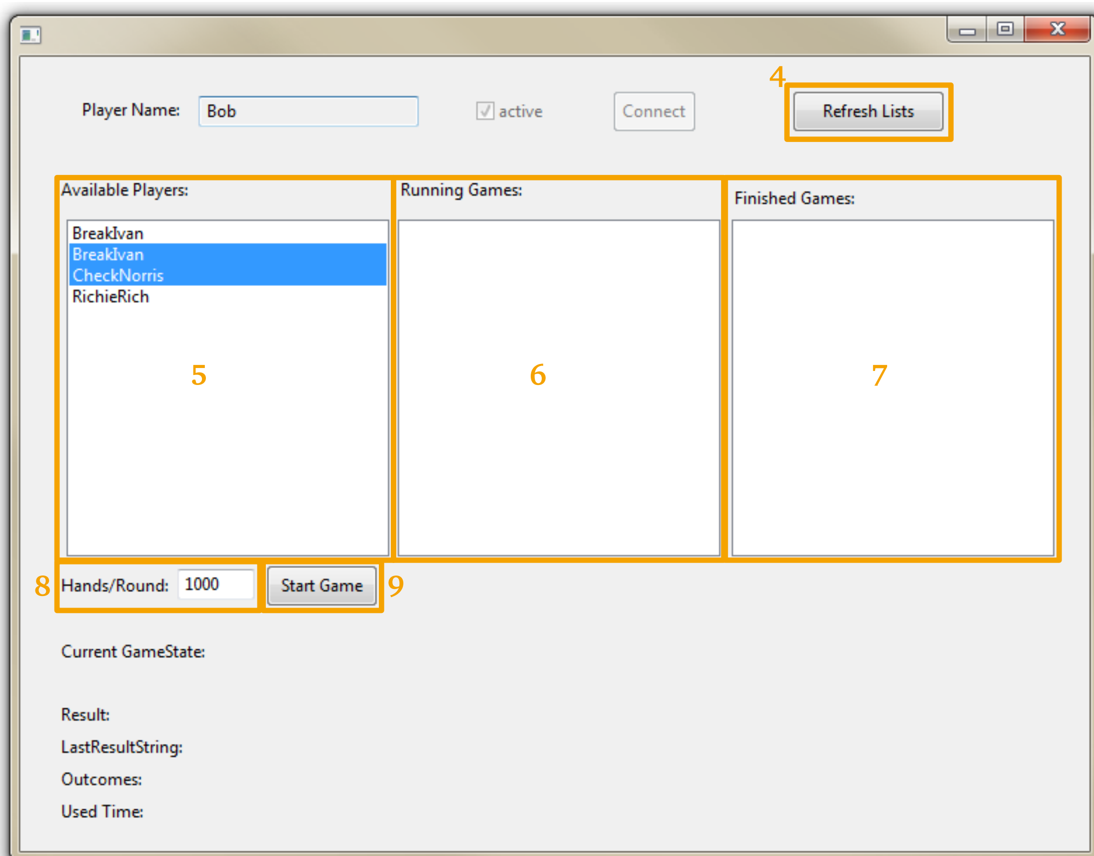


Abbildung 17: GUI nach dem Verbinden zum Server

Sobald der Pokerclient mit dem Server verbunden ist, werden die Listen ‚Available Players' (Feld 5), ‚Running Games' (Feld 6) und ‚Finished Games' (Feld 7) geladen, wie in Abbildung 17 zu erkennen ist. In ‚Available Players' sind alle die Spieler aufgelistet, die zurzeit auf dem Server als Gegner zur Verfügung stehen, sich also passive angemeldet haben und aktuell in keinem Spiel beteiligt sind. In der Liste ‚Running Games' sind alle laufenden Spiele aufgeführt. ‚Finished Games' beinhaltet alle beendeten Spiele. Zudem ist es jederzeit möglich mit dem ‚Refresh Lists'-Button in Feld 4 ein Update der Listen beim Server anfordern, um beispielsweise einen Zwischenstand des aktuell laufenden Spiels zu betrachten oder zu überprüfen wie weit fortgeschritten andere Spiele auf dem Server sind.

Um ein Spiel zu starten wählt man sich aus der Liste der verfügbaren Spieler genau 2 Spieler aus, gegen die der eigenen Bot spielen soll. Hierzu ist es, wie bei Listen üblich, mit der ‚Strg'-Taste einzelne

Spieler oder mit der ‚Shift'-Taste eine Liste von Spielern auszuwählen. In Feld 8 wird danach eingetragen, wie viele Hände pro Runde gespielt werden sollen. Hierbei ist mit Hände pro Runde die Anzahl Hände pro Iteration in dem ‚duplicate match'-Modus, wie er im Abschnitt ‚1.2 Die ‚Annual Computer Poker Competition" erläutert wurde, gemeint. Wurden die Gegner ausgewählt und die gewünschte Handanzahl eingestellt startet man das Spiel mit dem Button ‚Start Game' (Feld 9).

Der Server wird dadurch angewiesen das Spiel zu initialisieren, die Durchführung anzustoßen und die GUI upzudaten. Durch dieses Update verschwinden die beiden ausgewählten Spieler aus der Liste der zur Verfügung stehenden Spieler und das gestartete Spiel wird in der Liste der momentan laufenden Spiele eingetragen.

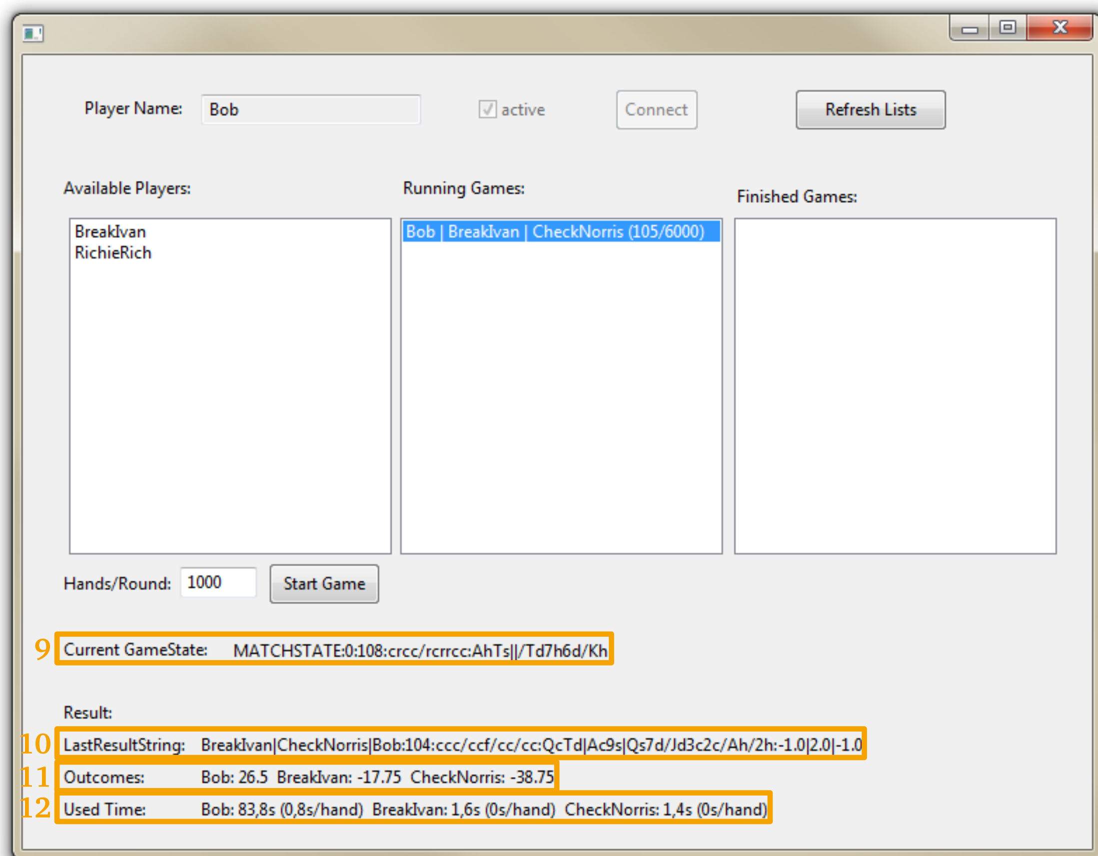


Abbildung 18: Detailansicht eines laufenden Spiels

Wurde ein Spiel gestartet können wie bereits erläutert durch Nutzung des Refresh-Buttons die Listen aktualisiert werden. Laufende und abgeschlossene Spiele können dann in den Listen ausgewählt werden, um eine Detailansicht zu den Spielen zu erhalten, wie in Abbildung 18 gezeigt wird.

Zunächst wird bei laufenden Spielen der aktuelle Spielzustand des Client in Feld 9 unter der Bezeichnung ‚Current GameState' angegeben. Das letzte komplett gespielte Spiel wird bei ‚LastResultString' (Feld 10) angezeigt. ‚Outcomes' (Feld 11) bietet einen Überblick über den Zwischenstand der Gewinne/Verluste der einzelnen Pokeragenten. Im Feld 12 ‚Used Time' wird die bisher benötigte Rechenzeit der Clients aufsummiert (im Fall von Bob z.B. 83,8 Sekunden) und die durchschnittlich benötigte Zeit pro Hand, die am Ende 7 Sekunden, wie in Abschnitt ‚1.2 Die ‚Annual Computer Poker Competition" beschrieben, nicht überschritten werden darf, berechnet. Bob hat beispielsweise einen durchschnittlichen Verbrauch von 0,8 Sekunden pro Hand in den ersten 104 Händen gehabt. Die durchschnittlich benötigte Zeit von BreakIvan liegt unter 0,1 Sekunden. Da an

dieser Stelle auf nur eine Nachkommastelle gerundet wird, werden hier 0 Sekunden als durchschnittlicher Verbrauch angegeben.

Sobald das Spiel beendet ist, wird es in der Liste der beendeten Spiele eingetragen und aus der Liste der laufenden Spiele entfernt. Zudem werden die beiden beteiligten passiven Bots wieder in die Liste der verfügbaren Spieler eingereiht und stehen ab diesem Zeitpunkt wieder für neue Spiele zur Verfügung.

Alternativ ist es aber auch möglich anstatt eines Pokeragenten selbst auf dem Pokerserver zu spielen umso eventuelle Schwächen eines Bots ausfindig zu machen. Hierzu kann mithilfe der Klasse PokerGUI eine spartanische aber dennoch voll funktionsfähige GUI erstellt werden. Diese meldet sich beim Start sofort, genau wie die Konsolenvariante der Bots, passiv beim Server unter dem Namen ‚Poker GUI' an und steht dann aktiven Bots als möglicher Gegner in der Liste der verfügbaren Agenten bereit. Zur Verfügung stehen in der GUI die Holecards des Spielers, seine zum Button relative Position, die aktuelle Potgröße sowie die bereits ausgeteilten Gemeinschaftskarten. Zusätzlich wird noch der momentane GameState als String in der GUI angezeigt, um einen Überblick über die bereits getätigten Aktionen zu bekommen.

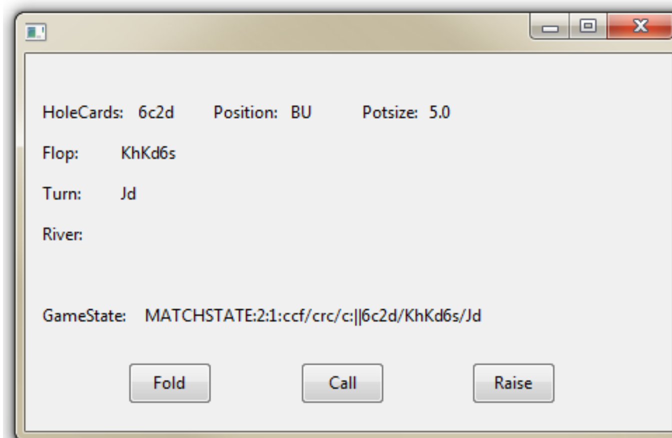


Abbildung 19: Interface für einen menschlichen Spieler um selbst gegen Bots spielen zu können

9 Fazit

9.1 Zusammenfassung

Ausgehend von Problemen während der TUD Computer Poker Challenge entstand die Idee ein Framework zu schaffen, welches eine Entwicklungsumgebung für in Zukunft zu entwickelnde Pokerbots darstellen kann. Es sollte eine Umgebung geschaffen werden, die diese Entwicklung vereinfacht indem vorgefertigte Werkzeuge einfach eingesetzt werden können und so von allerlei Aufgaben, die rund um die Entwicklung eines intelligenten Pokeragenten anfallen, entbindet.

Das Potenzial und der Umfang eines solchen Frameworks sind enorm. Daher war es im Rahmen dieser Bachelorarbeit nicht möglich alle Aspekte detailliert zu verfeinern und zu implementieren. Aufgrund des objektorientierten Designs der Software bietet sich allerdings die Möglichkeit das Framework leicht zu erweitern und der so geschaffene Mehrwert bei zukünftigen Bot-Entwicklungen wieder zu verwenden. Einige Ansätze hierfür werden im Abschnitt ‚9.2 Ausblick' genannt. Es werden aber sicher im Laufe der Entwicklung von Pokeragenten weitere Anforderungen entstehen und neue Strategien entdeckt werden, die bisher nicht bedacht wurden. Das Framework bietet hierfür eine Grundlage, die stetig erweitert werden kann.

Es wird nun noch einmal auf die in der Einleitung genannten Ziele eingegangen und beschrieben auf welche Art und Weise diese durch das Framework adressiert werden.

Mit dem ersten Ziel wurde die Forderung nach einer leicht auszuwertenden Zustandsrepräsentierung formuliert. Die Entwickler sollten vom Parsen des GameStateStrings entlastet und die gewonnenen Informationen leicht zugänglich gemacht werden. Dieses Ziel wurde durch die Einführung der Klasse ‚GameState' und der zugehörigen ‚GameStateFactory' erreicht. Eigenschaften über den Spielzustand sind in Zukunft mit dem Framework sehr leicht abrufbar, da der GameStateString automatisch geparst wird und die gewonnenen Informationen an einem Ort gesammelt und bereitgehalten werden.

Die 2 weiteren Ziele beschäftigten sich mit dem Durchführen der zur Evaluation der Pokerbots benötigten Turniere. Es sollte ermöglicht werden lokale Turniere einfach zu starten und die Ergebnisse leicht auszuwerten. Dieses Ziel wurde durch Einführung eines eigenen Pokerservers erreicht. Dieser lässt sich leicht und nahezu ohne Kenntnis des Frameworks starten und das Setup für ein Turnier ist schnell aufgestellt. Ebenfalls das Durchführen von den so genannten internen Auswertungsturnieren wird hierdurch vereinfacht. Es besteht allerdings noch Verbesserungsbedarf was Robustheit, Internetfähigkeit und Komfort des Servers betrifft.

Die effiziente Implementierung von Algorithmen wurde ebenfalls bei den Zielen der Arbeit aufgeführt. Hier wurde durch die Implementierung von Lookup-Tabellen Performance geschaffen. Dennoch besteht in diesem Bereich ebenfalls noch Potenzial, um die Abläufe der Auswertung einer Hand noch weiter zu beschleunigen. Da es hier fast ausschließlich auf Performance ankommt sollte überlegt werden, ob man eingeführte Abstraktionen wieder etwas auflöst umso, die Effizienz und den durch die Objektorientierung erzeugten Overhead verringert.

Das letzte genannte Ziel forderte die Möglichkeit der leichten Erweiterbarkeit und insbesondere die Wiederverwendbarkeit von implementierten Funktionalitäten. Auch dieses Ziel wurde mittels eines durchdachten Designs und den gewählten Entwicklungsparametern (wie z.B. der Programmiersprache Java) erreicht.

Es lässt sich also zusammenfassen, dass grundlegende Ziele dieser Bachelorarbeit erreicht wurden. Jedoch bietet der Themenbereich noch viele weitere interessante Möglichkeiten, um das entwickelte Fundament zu erweitern.

9.2 Ausblick

Im Folgenden werden nun noch abschließend einige mögliche Erweiterungen und Optimierungen des Frameworks vorgestellt, die nicht mehr im Rahmen dieser Bachelorarbeit verwirklicht werden konnten, aber von denen dennoch ein großer Nutzen ausgehen kann. Es sollen hierbei auch Ansatzpunkte genannt werden, die einen Einstieg für weitergehend Interessierte erleichtern.

Eine erste, sehr nützliche Verbesserung wäre eine Erweiterung der Zustandsrepräsentation. In der Klasse GameState, die für diese Repräsentation verantwortlich ist, sind bisher nur einige grundlegende Eigenschaften des aktuellen Zustands beschrieben. Hierzu zählen die von den Spielern getätigten Aktionen, bereits ausgeteilte Gemeinschaftskarten unterteilt nach den Straßen und Informationen wie die aktuelle Potgröße oder die Anzahl verbliebener Spieler. Aus dem GameState-String lassen sich jedoch noch wesentlich mehr nützliche Informationen gewinnen. Beispielsweise könnte automatisiert ermittelt werden, welcher Spieler zurzeit die so genannte Initiative besitzt. Diese beim Spielen sehr wichtige Information könnte direkt beim Parsen des Strings bestimmt und im GameState gespeichert werden. Ebenso könnte die Information, welcher Spieler die letzte Aktion in der aktuellen Runde machen kann oder wie viele Chips noch maximal eingesetzt werden müssen, um den Showdown zu sehen, bestimmt werden. Neben den eben vorgeschlagenen Aspekten einer konkreten Spielsituation gibt es noch unzählige weitere mögliche Werte, die bestimmt werden können. Um den Spielzustand möglichst detailliert zu beschreiben würde es sich anbieten eine Vielzahl dieser zusätzlichen Werte zu speichern. Dabei sollte allerdings darauf geachtet werden, dass in der Zustandsrepräsentation nur objektiv bestimmbare Werte gespeichert werden sollten. Subjektiv ermittelte Eigenschaften, wie beispielsweise eine Einschätzung welche Karten ein Gegner halten könnte, sollten nicht in der Klasse GameState gespeichert werden. Da jedoch der Spielzustand z.B. für Simulationen ggf. auch mit solchen Parametern ausgerüstet werden muss, muss hier ein Kompromiss zwischen objektiven Fakten eines Zustands und Annahmen über den Zustand, die eventuell falsch sein können, getroffen werden. Generell sollte allerdings die Klasse GameState möglichst nur korrekte Fakten beinhalten und gemachte Annahmen bei einer Weiterentwicklung der Klasse explizit gekennzeichnet werden, um den Benutzer darauf aufmerksam zu machen, dass er es eventuell mit potentiell falschen Werten zu tun hat.

Die eben beschriebenen subjektiven Annahmen über den Spielzustand sind überwiegend keine Annahmen über den gesamten Zustand, sondern meist Annahmen über einzelne Gegner. Diese Annahmen über konkrete Gegner wird als ‚Opponent Modeling' bezeichnet. Das Framework deckt diesen Teilbereich der Zustandsmodellierung nur in Ansätzen ab. Ein Grund für die nicht-Implementierung ist die Tatsache, dass es sich bei der Modellierung der Gegner um einen äußerst weitreichenden Themenkomplex handelt, der den Rahmen dieser Arbeit bei Weitem gesprengt hätte. Ein weiterer Grund ist, dass diese Beurteilung bzw. Einschätzung der Gegner wie beschrieben sehr stark vom subjektiven Empfinden und Erfahrungen der Entwickler abhängt. Ziel des Frameworks war jedoch zunächst eine möglichst objektive Beurteilung der Spielsituation zu geben und die Einschätzung über die Gegner den Benutzern des Frameworks zu überlassen. Da jedoch gutes Opponent Modeling ein wesentlicher Grundbaustein eines guten Pokerbots zu sein scheint, wurde bei der Entwicklung des Frameworks darauf geachtet, dass diese leicht zu implementieren ist. So gibt es beispielsweise bei jedem Player in einem GameState einen ‚DecisionMaker'. Dieser DecisionMaker kann zu einem komplexen Opponent Modeling erweitert werden, um die Spielweise verschiedener Spieler vorherzusagen. Zu dieser Modellierung der Gegner würde zum Beispiel zählen die Holecards möglichst präzise einzuschätzen oder zu bestimmen mit welcher Wahrscheinlichkeit der Gegner bei einem Raise

folden wird. Aufbauend auf diesen Annahmen könnte dann eine optimale Spielweise gegen diesen Gegner oder auch sein Verhalten in Mehrspielersituationen evaluiert werden.

Ebenfalls in die Richtung Opponent Modeling gehend sind verschiedene Statistiken über den Gegner, die bisher von dem Framework nicht gesammelt werden. So könnte man beispielsweise bestimmen wie oft ein Gegner grundsätzlich foldet oder raist, wie viel Prozent seiner Starthände er spielt, wie oft er bis zum Showdown geht und wie stark seine Hände am Showdown sind. Zudem könnten noch wesentlich spezifischere Informationen gesammelt werden. Hierzu gehört zum Beispiel die Spielweise eines Gegners in bestimmten, immer wieder vorkommenden Situationen. Spielt ein Gegner einen Flushdraw am Flop aggressiv oder passiv? Wie spielt er diesen am Turn weiter? Oder neigt ein Spieler dazu sehr starke Hände eher langsam zu spielen oder spielt er diese sehr aggressiv und offen? Alle diese Informationen könnten vom Framework automatisiert gesammelt und zur Verfügung gestellt werden. Ansatzpunkt für die Implementierung wäre hierbei die Klasse Player, die man um diese Felder erweitern könnte. Genutzt werden könnte das neue Wissen, um ein besseres Gegnermodell zu erstellen und Simulationen mit dem Gegner präziser durchzuführen. Zusätzlich kann eventuell auch aus der verwendeten Zeit eines Gegners für eine Entscheidung Rückschlüsse auf die Stärke der Hand gezogen werden. Reagiert ein Spieler beispielsweise sehr schnell könnte das ein Indiz für eine sehr starke oder sehr schwache Hand sein.

Eine detailliertere Auswertungsmöglichkeit der Turniere wäre ebenso eine weitere, interessante Erweiterung des Frameworks. Man könnte gemachte Spiele graphisch auswerten, um z.B. den Moneyflow der Spieler darzustellen. Diese Information könnte insbesondere bei Turnieren mit mehr als 3 Spielern sehr interessant sein. Auch der Zeitverbrauch der Bots könnte dargestellt werden. Hierfür müssten gegeben falls weitere Informationen in die Logdateien geschrieben werden. Interessant wäre auch sicher den Gewinn/Verlust der Pokeragenten über den Verlauf der Matches zu visualisieren um festzustellen, ob lernende Bots während der Spiele wirklich an Stärke zunehmen, das gelernte Wissen zu keiner signifikanten Veränderung des Spielstils beiträgt oder der Bot vielleicht sogar mit der Zeit schwächer wird. Weitere statistische Auswertungen von Turnieren und insbesondere eine einfache Darstellung würden bei der Analyse von neu entwickelten Agenten möglicherweise eine große Hilfe sein. Eine Erweiterung des Frameworks in diese Richtung sollte also auch in Betracht gezogen werden.

Wie während der Arbeit beschrieben, ist eine effiziente Evaluierung einer Showdown-Situation äußerst wichtig, um diverse Metriken in ausreichend kurzer Zeit berechnen zu können. Weitere Performancesteigerungen können für die Benutzer des Frameworks genau wie die bisher beschrieben Verbesserungen sehr nützlich sein. Denn die Aussagekraft von Simulationen wie beispielsweise die UCT-Simulation, wie im Abschnitt ‚7 UCT-Simulation mit Opponent Modeling' beschrieben, hängt stark davon ab ausreichend viele Iterationen machen zu können. Deshalb können die verwendeten Algorithmen nie performant genug sein, eine Verbesserung in der Laufzeit bringt auch immer die Möglichkeit genauer zu simulieren. Deshalb ist die Optimierung dieser grundlegenden Auswertungen weiterhin wünschenswert und bietet somit eine weitere gute Möglichkeit das bestehende Framework zu verbessern. Zu einer Performancesteigerung kann es beispielsweise durch geschicktere Nutzung von Lookup-Tabellen oder Nutzung von Bibliotheken in anderen Programmiersprachen (Stichwort: JNI) kommen. Ebenso ein Abbau der Objektorientierung kann weitere Geschwindigkeitsvorteile bringen, denn ‚schönes' Softwaredesign geht nahezu immer auf Kosten der Performance. Jedoch sind auch Änderungen am Framework denkbar, die den Grad der Abstraktion hoch halten. So könnte beispielsweise das Flyweight-Pattern [G07, Seite 195-206] bei der Klasse Card angewendet werde, um die Anzahl der Objekte stark zu reduzieren. Weitere Optimierungen in diesem Bereich sind denkbar.

Ebenfalls noch verbesserungsfähig ist die Funktionalität des Frameworks in Richtung durchführen von Turnieren. Es wäre wünschenswert wenn hier noch einiges an Komfort geboten wird, um z.B. auch größere Turniere automatisiert durchführen zu können. So werden beispielsweise für eine umfangreiche interne Auswertung während der Challenge nicht nur ein Spiel gespielt, sondern ein

komplettes Turnier. Daraus werden noch mit den beiden vorgestellten Verfahren ‚Total Bankroll' und ‚Bankroll Instant Runoff' auf verschiedene Arten Gewinner festgestellt. Die Möglichkeit, einige Bots aus einer Liste von am Server angemeldeten Agenten auszuwählen, um diese dann ein komplettes Turnier spielen zu lassen und den Gewinner nach den beiden Verfahren zu bestimmen wäre ebenfalls ein gut zu gebrauchendes Feature. Hierbei könnten dann Parameter wie z.B. die Handanzahl pro Match eingestellt werden.

Eine weitere Verbesserungsmöglichkeit bietet der Server. Bisher sind nur rudimentäre Funktionen wie das Verbinden bzw. Trennen von Client zum Server oder Starten von Spielen implementiert. Auch die Internetfähigkeit ist nur sehr eingeschränkt und noch nicht voll ausgereift entwickelt und getestet. Der Server könnte robuster gemacht werden, wenn unerwartete Situationen auftreten. Ausnahmen wie z.B. eine volle Festplatte beenden den Server zurzeit einfach ohne beispielsweise die Möglichkeit zu haben begonnene Spiele später fortzusetzen. Auch das Pausieren und vorzeitige Beenden von Spielen könnte eine wünschenswerte Funktionalität darstellen. Zudem könnten neue Instanzen von verfügbaren Pokeragenten automatisch gestartet werden, falls keine weiteren Instanzen eines bestimmen Bots auf dem Server als potentieller Gegner für aktive Clients zur Verfügung stehen. Somit wäre sichergestellt, dass immer ausreichend Konkurrenz auf dem Server ist, um neue Bot-Implementierungen zu testen.

Wie man sieht gibt es noch zahlreiche Punkte an denen das Framework weiter entwickelt werden sollte. Verbesserte Zustandsrepräsentation, erweiterte Statistiken, Opponent Modeling und performantere Algorithmen sind neben verbesserter Serverfunktionalität und Turniermanagement nur einige Ansatzpunkte. Es bleibt also festzuhalten, dass dieses Framework nur als Grundstein für eine umfassende Entwicklungs- und Evaluationsumgebung für intelligente Pokeragenten zu verstehen ist. Erst durch weitere Verbesserungen wird es möglich sein das volle Potential eines solchen Frameworks auszunutzen.

Abbildungsverzeichnis

Abbildung 1: Übersicht über die wichtigsten Teile des Frameworks 9
Abbildung 2: Initialisierung des Pokerbots 'CheckNorris', Aktualisierung des GameStates und Reaktion auf diese..13
Abbildung 3: Übersicht über alle Nachrichten unterteilt in die 3 Bereiche Client, Server und Intern ...15
Abbildung 4: Initialisierung des Nachrichten-Subsystems und versenden einer Refresh-Nachricht 16
Abbildung 4: Beispielhafte Ausprägung des Nachrichten-Subsystems mit angeschlossenen Clients......18
Abbildung 6: Datenfluss zwischen Server und Client 22
Abbildung 7: Zuordnung Hand zu Handstärke...23
Abbildung 8: Evaluierung der Handstärke einer Hand 24
Abbildung 9: Blockdarstellung des aufgeteilten Integers 26
Abbildung 10: Beispiel einer Auswertung. Im 2. Block sieht man die Codierung von 'Pair', im 4. den Wert '10' für die Wertigkeit des Paares und im letzten Block die Bitmap der Kicker 26
Abbildung 11: Übersicht über die Codierung der Handstärke...27
Abbildung 12: Reduzierung der möglichen HandRanks durch den Wechsel von Listen auf ungeordnete Mengen 28
Abbildung 13: Algorithmus zum Berechnen des Immediate Handranks 31
Abbildung 14: Algorithmus zum Berechnen des Seven-Card Handranks 33
Abbildung 15: Beispiel eines Suchbaums der mit Hilfe des UCT-Verfahrens erstellt wurde..................38
Abbildung 16: GUI nach dem Starten eines Pokerbots 40
Abbildung 17: GUI nach dem Verbinden zum Server 41
Abbildung 18: Detailansicht eines laufenden Spiels 42
Abbildung 19: Interface für einen menschlichen Spieler um selbst gegen Bots spielen zu können 43

Literaturverzeichnis

[A88] **Victor Allis:** *A Knowledge-based Approach of Connect-Four*. Report IR-163 by the Faculty of Mathematics and Computer Science at the Vrije Universiteit Amsterdam, The Netherlands, 1988.

[ACPC10] **Department of Computer Science, University of Alberta:** http://www.computerpokercompetition.org, abgerufen am 8. November 2010.

[BK06] **Darse Billings and Morgan Kan:** *A Tool for the Direct Assessment of Poker Decisions*. The International Association of Computer Games Journal, Oktober 2006

[BPSS02] **Darse Billings, Lourdes Pena, Jonathan Schaeffer, Duane Szafron:** *The challenge of poker*. Artificial Intelligence, Januar 2002.

[G07] **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides:** *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, März 2007.

[KS07] **Levente Kocsis, Csaba Szepesvari:** *Bandit based Monte-Carlo Planning*. Computer and Automation Research Institute of the Hungarian Academy of Sciences, Kende u. 13-17, 1111 Budapest, Hungary.

[MSM05] **Ed Miller, David Sklansky and Mason Malmuth:** *Small Stakes Hold'em: Winning Big With Expert Play*. Two Plus Two Publishing LLC, Dezember 2005

[RN04] **Stuart Russell, Peter Norvig:** *Künstliche Intelligenz: Ein Moderner Ansatz*. 2.Auflage, Pearson-Studium, 2004

[W08] **Michael Wächter:** *UCT: Selektive Monte-Carlo-Simulation in Spielbäumen*. TUD Computer Poker Challenge Ausarbeitung, Knowledge Engineering Group, Technische Universität Darmstadt, 2008.

[Y05] **King Yao:** *Weighting The Odds In Hold'em Poker*. Pi Yee Press, 2005

Glossar

B

Bankroll
Die Menge an Chips, die einem Spieler zur Verfügung stehen wird als Bankroll bezeichnet.

Board
→ Gemeinschaftskarten

Bot
Ein Bot ist ein automatisierter Agent in einem Spiel.

Button
Der Button oder auch Dealer-Button markiert einen Spieler während einer Spielrunde und stellt damit eine Spielposition da.

C

Chips
Chips stellen den Einsatz der Spieler während einer Spielrunde da.

F

Flop
Als Flop bezeichnet man sowohl die ersten 3 Gemeinschaftskarten als auch die 2. Spielrunde während eines Pokerspiels.

Fully-Qualified Class Name
Der Fully-Qualified Class Name (FQCN) ist ein eindeutiger Name für eine Klasse. Dieser besteht aus dem kompletten Pfad der Klasse und dem Klassennamen.

G

Gemeinschaftskarten
Die Gemeinschaftskarten bestehen aus den Karten, die auf dem Flop, Turn und River für alle Spieler sichtbar ausgeteilt werden.

H

Hand
Der Begriff Hand besitzt 2 Bedeutungen: Zum einen wird mit Hand eine Spielrunde vom Austeilen der Holecards bis zum Ende der Runde bezeichnet. Hand wird aber auch abkürzend für Pokerhand benutzt und beschreibt die Karten, die ein Spieler mit seinen Holecards und dem Board bilden kann.

Handkarten
Als Handkarten bezeichnet man die beiden Karten, die jeder Spieler zu Beginn einer Hand ausgeteilt bekommt. Diese beiden Karten bleiben bis zum Showdown für die Gegner verdeckt und bilden somit eine Situation mit verdeckter Information.

Handstärke
Die Handstärke beschreibt den absoluten Wert einer Hand. Sie besteht aus dem Handrang und evtl. noch weiteren Attributen um Hände mit gleichem Handrang bewerten zu können. So wird z.B. zum Handrang „Full House“ noch der Rang des Drillings sowie des Pärchens gespeichert.

Handrang
Der Handrang ist eine Kategorisierung einer Hand, die eine Aussage über ihre Stärke macht. Es wird dabei zwischen den folgenden Kategorien unterschieden:

- High Card
- Pair
- Two Pair
- Three Of A Kind
- Straight
- Flush
- Full House
- Four Of A Kind
- Straight Flush

Holcards
→ Handkarten

I

Initiative
Ein Spieler besitzt während einem Spiel die Initiative, wenn er der letzte war, der die Setzhöhe erhöht hat. Dies kann entweder durch einen Raise oder eine Bet geschehen.

M

Moneyflow
Der Moneyflow beschreibt, wie Chips während eines Spiels oder Turniers geflossen ist. Hat beispielsweise ein Spieler A an Spieler B Chips verloren, so gibt es einen Moneyflow von A nach B.

P

Position
Die Position eines Spielers gibt seine Position relativ zum Button an. Es werden je nach Spieleranzahl zwischen folgenden Positionen unterschieden:

- Small Blind (SB)
- Big Blind (BB)

- Under the gun (UTG1)
- Under the gun 2 (UTG2)
- Under the gun 3 (UTG3)
- Middle Position 1 (MP1)
- Middle Position 2 (MP2)
- Middle Position 3 (MP3)
- Cut Off (CO)
- Button (BU)

Hierbei sitzt beispielsweise der Button links neben dem Cut Off und rechts neben dem Small Blind.

R

River
Als River bezeichnet man sowohl die 5. Gemeinschaftskarte als auch die 4. und damit letzte Spielrunde während eines Pokerspiels.

S

Straße
Als Straßen bezeichnet man beim Pokern die verschiedenen Setzrunden. Es wird zwischen den 4 Straßen

- Preflop
- Flop
- Turn
- River

unterschieden.

T

TD-Gammon
TD-Gammon ist ein Backgammon-Bot, welches 1992 von Gerald Tesauro entwickelt wurde. Der Name stammt von dem im Programm benutzten neuronalen Netz, dass mit einer Form des ‚temporal-difference learning' trainiert wurde.

Texas Hold'em
Texas Hold'em ist eine Pokervariante, die in letzter Zeit stark an Beliebtheit zugenommen hat. Es werden hierbei bis zu 5 offene Gemeinschaftskarten ausgeteilt, mit denen, in Verbindung mit 2 eigenen, für die Gegner verdeckten Handkarten, eine Pokerhand gebildet wird.

Turn
Als Turn bezeichnet man sowohl die 4. Gemeinschaftskarte als auch die 3. Spielrunde während eines Pokerspiels.