Modulare Erweiterung des TUD-Pokerframeworks

Bachelor-Thesis von Julian Prommer aus Mannheim, Neckarau

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering

Modulare Erweiterung des TUD-Pokerframeworks

Vorgelegte Bachelor-Thesis von Julian Prommer aus Mannheim, Neckarau

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza Mencía

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis 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. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 31. Juli 2015

(Julian Prommer)

Abstract

Ziel dieser Arbeit war die Erneuerung des TUD-Pokerframeworks, welches vor allem für Tests vor der Teilnahme an der Annual Computer Poker Competition (ACPC) Computer-Poker-Competition (2014c), genutzt wird und als Plattform für Poker-Agenten-Entwicklung dient. Das Framework sollte folgende Verbesserungen erhalten: Die Pokerspieltypen waren Limited-Texas Hold Em, Nolimited Texas Hold Em und Kuhn-Poker mit jeweils mit einem Heads-Up und einem Ring-Play Modus zu einem gemeinsamen Framework zusammenzuführen. Der Speicherverbrauch war zu senken. Die Berechnungsgeschwindigkeit und die Programmstabilität wurden verbessert. Die Bedienbarkeit des Spiel-Servers war durch eine Befehlstruktur zu verbessern. Die bereits enthaltene GUI wurde erweitert. Die Module des Frameworks sollten für Poker-Agenten-Entwickler intuitiver verwendbar gemacht werden.
Die Ausarbeitung berichtet über den Stand der Umsetzungen der genannten Aspekte und soll als Anleitung dienen, um das Framework fortzuentwickeln und zu benutzen.

Inhaltsverzeichnis

1 Einleitung

Poker ist ein beliebtes Spiel, welches sich zugleich als lohnenswertes Forschungsobjekt darstellt. Die meisten Pokervarianten benötigen nur relativ wenige Regeln und sind dadurch schnell erlernbar. Wegen des einfachen Spielprinzips bietet es sich es an, einem Computer durch Agenten (künstliche Intelligenz) spielen zu lassen. Die Strategien für das dauerhaft erfolgreiche Pokerspielen sind für Mensch und Maschine schwierig zu erlernen. Die Strategien sollen den Glücksfaktor, der durch die ausgeteilten Karten entsteht, möglichst minimieren. Der Entwurf erfolgreicher Strategien ist bereits eine fordernde Aufgabe. Diese Arbeit soll dem Agenten-Entwickler beim Erstellen seiner Poker-Agenten, den sogenannten Pokerbots, unterstützen.

1.1 Motivation für das Erforschen des Pokerns

Spiele sind sehr geeignete Studienobjekte für die Untersuchung von algorithmischen Methoden. Bestimmte Spiele wie Poker besitzen grundsätzliche Eigenschaften, welche die Untersuchung der Algorithmen begünstigen. Sie haben eine feste Menge von Aktionen, die in einem Spielzug möglich sind. Zudem gibt es üblicherweise nur abzählbar viele oder eine feste Anzahl von Spielobjekten, wie Spieler, Karten und Aktionen mit ihren Attributen. Der Regelsatz für erlaubte Aktionen ist festgelegt und meistens relativ einfach. Der Erfolg eines Spielers ist wenigstens durch Sieg bewertbar, aber normalerweise sind weitere Performance-Maße identifizierbar wie Punkte, Anzahl der Züge oder Spieldauer und ähnliches. Für die einzelnen Spielzüge lassen sich Maße für die Optimalität ableiten, welche den Zustandsbaum beziehungsweise den Suchbaum eines Spiels einschränken. Für Spiele niedriger Komplexität lassen sich sogenannte starke Lösungen angeben, das heißt, im Spielbaum wird immer ein Ast gewählt, der trotz optimaler Spielzüge des Gegners, zum Sieg führen wird. Ist die Komplexität zu groß, um direkt berechnet werden zu können, werden Algorithmen wie MiniMax, Heuristiken oder Maschinelle-Lerner eingesetzt, um sich dem optimalen Spielzug anzunähern; solche Lösungen werden schwache Lösungen genannt.

Zufallskomponenten wie Zahlen oder verdeckt ausgeteilte Spielkarten erschweren die Bestimmung des optimalen Spielzuges erheblich, insbesondere bei mehrstufigen Spielen. Auch erschweren nur partiell beobachtbare Spielzustände eine profitable Entscheidungsfindung. Typische nicht beobachtbare Objekte sind verdeckte Spielsteine, Handkarte oder verbleibendes Kartendeck. Es können eigene Entscheidungen nur mit einer gewissen Unsicherheit getroffen werden und ebenso nur unter Unsicherheit Spielzüge des Gegners vorgesagt werden. Durch die partielle Beobachtbarkeit sind gegenüber einem offenen Spiel erheblich mehr Strategien möglich, da man den Gegner in die Irre führen kann, indem man eine stärkere oder schwächere Spielposition repräsentiert als es in Wirklichkeit der Fall ist und so Fehlentscheidungen des Gegners provoziert. Unabhängig von den genannten Erschwernissen wird ein Spiel immer komplizierter mit steigender Spielerzahl, weil der Einzelne weniger Einfluss auf den Spielverlauf hat.

Die Eigenschaften des Pokers sind: partiell beobachtbarer Spielzustand mit einem großem Spielbaum und vielen Spielern (mit dem Spezialfall 2 Spieler). Poker ist eines der einfachsten Spiele mit diesen Eigenschaften. Im Vergleich hierzu steht das Spiel Domino, bei dem auch nur ein partieller Spielzustand bekannt ist. Aber die Menge aller Zustände sollte nach grober Abschätzung geringer ausfallen als beim Poker und man gewinnt im Verlauf einer Spielrude eine größere Übersicht über den Spielzustand. Es liegen mehr Steine offen auf dem Tisch, die eine Abschätzung, der noch kommenden und der im Besitz des Gegners befindlichen Spielsteine erleichtert. Es sei auch das Spiel Schiffe versenken für einen Vergleich erwähnt. Dort gibt es keine Zufallskomponente. Die verdeckten Schiffe des Gegners werden durch den Gegenspieler platziert. Aber es liegt dem Spieler auch nur ein partiell beobachtbarer Spielzustand vor. Der Unterschied liegt in der Menge der möglichen Aktionen. Beim Domino ist die Aktionsmenge dynamisch, weil diese direkt von bereits gelegten Spielsteinen abhängt. Schiffe versenken hat eine Anzahl von Aktionen. Bei diesen beiden Spielen müssen vor der Spielzugplanung, zunächst einmal alle möglichen Spielzüge erkannt werden. Im Gegensatz dazu haben die verschiedenen Pokervarianten zu jedem Zeitpunkt höchstens drei Grundspielzüge zur Auswahl (fold, bet bzw. raise; check bzw. call je nach Kontext ; Holecards tauschen). Das Pokern mit fester Wetterhöhung bedarf noch nicht einmal einer Entscheidung wie groß diese Erhöhung sein sollte. Es steht also weniger die Einschränkung des Planungsraums eines Spielzuges im Vordergrund, dafür um so mehr die Absicht, eine gewünschte Reaktion des Gegners hervorzurufen.

Mit den herausgestellten Eigenschaften eignet sich das Pokerspiel vor allem für die Erforschung von Entscheidungen mit Unsicherheit. Das ist relevant für viele reale Anwendungen. Denn die meisten realen Anwendungen haben erstens viele oder unendlich viele Zustände und zweitens keine voll beobachtbare Domäne, sondern es muss grundsätzlich erst einmal ein Modell gefunden werden, um diese Domäne für das zu lösende Problem zu beschreiben. Mit jedem Modell wird die Realität abstrahiert, weil diese unmöglich vollständig zu beobachten ist. Die Abstraktion führt zur Unsicherheit, doch etwas Relevantes nicht zu beobachten und es herrscht Unsicherheit wie dienlich die Aktion wirklich war. Wird Pokern zur Erforschung der Entscheidung mit Unsicherheit herangezogen, ist die Domäne bereits bekannt und beschrie-

ben. Es bleibt die Entscheidungsfindung mit wenigen Aktionen übrig, die dadurch evaluierbar wird. Diese Überlegungen basieren auf den Inhalten der Vorlesung Einführung in die künstliche Intelligenz von Prof. Dr. Fürnkranz.

1.2 Motivation für ein Pokerframebot und dessen Pflege

An dieser Stelle sollen kurz die wichtigsten Gründe erläutert werden, weshalb die Pflege des TUD-Pokerbotframeworks kontinuierlich fortgesetzt werden soll. Hauptsächlich sollen die üblichen Wartungsarbeiten bei einer langjährig eingesetzten Software erleichtert werden, also das Verwalten von Programmcodes. Angestrebt wird auch eine Steigerung der Zuverlässigkeit, die Senkung des Ressourcenverbrauchs und auch die Erweiterbarkeit soll verbessert werden oder wenigstens erhalten bleiben. Software altert genauso wie die dazugehörigen Computer. Neue Hardware und Betriebssysteme verlangen im Allgemeinen auch die Modernisierung von Programmen, die weiterhin in einer neuen Umgebung funktionieren sollen.

1.2.1 Pokerbot-Entwicklung

Wenn ein Pokerbot implementiert werden soll, genügt es nicht nur die Entwicklung der zugrunde liegenden künstlichen Intelligenz in den Quellcode zu übersetzen. Es kommen weitere Aspekte hinzu wie beispielsweise die Anbindung zu Testgegner und Möglichkeiten zum Vergleich mit Vorgängerbots zur Evaluation; wegen dieser Aspekte müssen Netzwerkcodes und Schnittstellen festgelegt und implementiert werden. Der Entwickler muss deshalb immer die komplette Pokerdomäne modellieren. Zudem wird für die Testspiele eine Instanz benötigt, die als Dealer auftritt. Es ist einsichtig, dass ein einzelner Pokerbot eine ganze Simulationsumgebung benötigt, um dessen Funktionstüchtigkeit zu überprüfen und seine Leistungsfähigkeit einzuschätzen. Das Zusammenspiel aller Komponenten in regelmäßigen Abständen zu testen, ist zeitaufwendig. Bei Neuentwicklungen wird man spätestens beim Testen auf Implementierungsfehler stoßen, welche die Fertigstellung verzögern. Designfehler können bis zu geplanten Erweiterungen oder einer neuen Version eines Bots unbemerkt bleiben. Diese Designfehler können dann ein Projekt sehr weit zurückwerfen. Der Entwurf eines Pokerbots birgt alle Probleme des Software-Engineering, wie jedes andere ernsthafte Projekt auch.

1.2.2 Einfachere Pokerbot-Entwicklung durch ein Framework

Ein eleganter Weg die Entwicklung eines Pokerbots zu beschleunigen, ist die Verwendung eines speziell auf das Pokern zugeschnittenen Frameworks. Die est besonders sinnvoll, wenn abzusehen ist, dass verschiedene Entwicklergruppen mehrere Pokerbots entwickeln wollen. Die Vorteile liegen klar auf der Hand:

- Mit einem Framework sind bereits die meisten Infrastrukturprobleme gelöst und die Entwicklung kann sich auf künstliche Intelligenz und maschinelle Lerner konzentrieren.
- Die Pokerbots verschiedener Entwicklergruppen können verglichen werden, weil diese über gleiche Schnittstellen verfügen. Durch direktes gegeneinander Antreten werden künstliche Intelligenzen vergleichbar. Verglechbarkeit durch gleiche Schnittstellen.
- Die Verlässlichkeit der Pokerbots wird durch ein getestetes Framework erhöht. Die Fehlerquellen beschränken sich dann meistens auf den jeweiligen Bot selbst.
- Das Framework erleichtert die Teilnahme an der Anual Computer Poker Competition.

1.2.3 Anual Computer Poker Competition

Seit 2006 wird die Annual ComputerPokerCompetition Computer-Poker-Competition (2014c) von der Association for the Advancement of Artificial Intelligence (AAAI) ausgerichtet. Die Computer Poker Competition findet jeden Sommer üblicherweise mit Einreichungsende Anfang Juni statt, während der AAAI Conference on Artificial Intelligence. Sollte im gleichen Jahr die International Joint Conference on Artificial Intelligence in Nordamerika stattfinden, dann wird die Computer Poker Competition dort ausgetragen.

Ein Hauptziel der Veranstaltung ist Beschäftigung mit Pokern, weil dieses Spiel im Allgemeinen als sehr interessant empfunden wird. „Poker is a game humans find interesting: at a basic level, artificial intelligence researchers stand to gain by considering those problems humans not only face but are fascinated by.“ Computer-Poker-Competition (2014b) - KI-Forscher können sich mit einem Thema beschäftigen, das nicht nur Einsichten für Menschen liefert sondern Menschen auch fasziniert.

Das andere Hauptziel: „Poker brings many new and interesting problems that are not faced in checkers, chess, Go, or backgammon. It is not only random, there is hidden information, and a desire to maximize your winnings. Handling the hidden information is a problem that is on the edge of artificial intelligence: for instance, in video games, often the bots cheat, i.e. they are given perfect information about the world. Understanding how AI can tackle these problems is key to the next stage of AI, not only in video games, but in real-world scenarios, such as in business and the military.“ Computer-Poker-Competition (2014b) - „Poker bringt viele Probleme zum Vorschein, mit denen man sich bei der Untersuchung von Dame, Schach, Go und Backgammon nicht ausseinander setzen musste. Es ist nicht nur reiner Zufall, sondern es verlangt, mit versteckten Informationen zurechtzukommen und gleichzeitig wird ein maximaler Gewinn erstrebt. Mit versteckten Informationen umgehen ist ein fundamentales Problem für künstliche Intelligenzen. Beispielsweise mogeln die Bots in Computerspielen oft, denn sie erhalten das ganze Modell des Spiels. Zu verstehen, wie künstliche Intelligenzen dieses Problem angehen können, ist der Schlüssel zu einer höheren Stufe von künstlichen Intelligenzen nicht nur für Computerspiele, sondern auch für reale Anwendungen.“

Der Wettbewerb soll die Erforschung von künstlichen Intelligenzen durch Promotion und der Evaluation von Forschungsarbeit in ihrer Erforschung unterstützen.

Die TU Darmstadt nimmt seit 2008 an diesem Wettbewerb im Rahmen von Praktika und Abschlussarbeiten teil.

- 2008: Limit Texas Hold Em Total Bankroll 6-Ring Platz 2 von 6

- 2009: Limit Texas Hold Em Instant Runoff 3-Ring Platz 3 und 4 von 7
  Limit Texas Hold Em Total Bankroll 3-Ring Platz 3 und 4 von 7

- 2010: Limit Texas Hold Em Instant Runoff 3-Ring Platz 4 von 7
  Limit Texas Hold Em Total Bankroll 3-Ring Platz 3 von 7

- 2011: Limit Texas Hold Em Instant Runoff Heads Up Platz 10 von 19
  Limit Texas Hold Em Total Bankroll Heads Up Platz 9 von 19
  Limit Texas Hold Em Instant Runoff 3-Ring Platz 8 von 10
  Limit Texas Hold Em Total Bankroll 3-Ring Platz 8 von 10

- 2013:
  Nolimit Texas Hold Em Instant Runoff Heads Up Platz 11 von 13
  Nolimit Texas Hold Em Total Bankroll Heads UP Platz 11 von 13
  Limit Texas Hold Em Instant Runoff 3-Ring Platz 5 von 6
  Limit Texas Hold Em Total Bankroll 3-Ring Platz 5 von 6

- 2014:
  Nolimit Texas Hold Em Instant Runoff Heads Up Platz 11 von 14
  Nolimit Texas Hold Em Total Bankroll Heads UP Platz 12 von 14
  Limit Texas Hold Em Instant Runoff 3-Ring Platz 3 von 5
  Limit Texas Hold Em Total Bankroll 3-Ring Platz 3 von 5

1.2.4 Frameworkpflege

Ein Framework zieht Wartungsarbeiten und Erweiterungen nach sich. Eine regelmäßige Wartung ist erstrebenswert. Die wichtigsten Aufgaben der Pflege sollen kompakt erläutert werden.

- Jede Programmoptimierung am Framework verbessert alle Bots, die das Framework verwenden.

- Bewähren sich einzelne Module/Bestandteile eines Pokerbots so könnten diese in das Framework eingebunden werden, um wiederverwendet zu werden.

- Das TUD-Pokerframework wurde in JAVA geschrieben. Seit der Entwicklung des Frameworkkerns sind bereits drei neue JAVA Versionen erschienen, die eine Programmüberarbeitung sinnvoll machen. Zwar enthalten die Standartbibliotheken veraltete Klassen und Methoden, aber diese werden in absehbarer Zeit nicht mehr unterstützt. Es gibt also keinen Grund daran festzuhalten.

- Die ACPC hat seit 2014 einen weiteren Spielmodus. Dieser sollte in Hinblick auf zukünftige Arbeiten bereits jetzt im eigenen Framework anlegen Computer-Poker-Competition (2014e).

1.3 Ziel dieser Bachelorthesis

Das primäre Ziel ist das TUD-Pokerframework für Texas Hold Em Limit und Nolimit verwendbar zu machen. Hierzu sollen auch Nolimit Testspiele auf dem Spielserver durchführbar werden. Die Erstellung von Nolimit- und Limit-Pokerbots soll ähnlich einfach eingeleitet werden wie zuvor. Die grafische Spieloberfläche für Poker-Runden gegen die Poker-Agenten soll für Nolimit verwendbar werden. Die Frameworkstruktur soll beibehalten werden.

Der erstgenannte sekundäre Ziel ist die Einbindung von Kuhn-Poker in das Framework, genauso wie zuvor Nolimit Texas Hold Em. Kuhn-Poker in das Framework einzuführen, unter Wahrung der derzeitigen Frameworkstruktur, ist eine forderndere Aufgabe als Nolimit einzupflegen. Kuhn-Poker ist keine Variante des Texas Hold Em Pokers.

Eine weiteres sekundäres Ziel sind allgemeine Wartungsarbeiten. Die Kommunikation zwischen den Poker-Clients und dem Poker-Server ist zu verbessern. Das Nachrichtensystem soll einfacher gestaltet werden. Eine alternative Kommunikationswege zum bestehenden Nachrichtensystem sind angedacht. Ein weiteres Wartungsziel ist es den Speicherverbrauch der PokerClients zu senken und gleichzeitig Berechnungszeiten zu verringern. Das kann durch die Vereinfachung der Spielzustandserzeugung erreicht werden. Für Kuhn-Poker ist ein schnelles Framework unerläßlich, wegen drastischer Zeitlimits der ACPC. Software altert und es gilt veraltete Methoden und Klassen, die aus dem JAVA-Framework stammen, durch ihre neuen Pendants zu ersetzen und bei dieser Gelegenheit auch gleich die Programmabschnitte im Rahmen der Frameworkstruktur zu vereinfachen.

Tertiäre Ziele sind: Eine echte Serversteuerung, die Übernahme von Modulen aus Ablegerprojekten, UCT-Simulation für Nolimit Texas Hold Em, ein Gegnermodell für alle Spielvarianten und Code-Dokumentation im Framework einpflegen inklusive JUnit-Tests.

1.4 Aufbau dieser Arbeit

Im Kapitel über Grundlagen werden die wichtigsten Begriffe des Pokers und des Computerpokerns aufgegriffen. Dieses kann bei hinreichender Kenntnis der Materie übersprungen werden. Das dritte Kapitel soll die Ausgangslage des Hauptframeworks für Texas Hold Em Limit und des Ablegerprojekts für Texas Hold Nolimit darstellen. Es werden hierbei auch die übergeordneten Ziele vorgestellt. Durch Kapitel 4 werden die Thesen und Ziele dieser Arbeit im Einzelnen aufgezeigt. Dort ist auch eine Versionierung für die Arbeitsschritte zu finden. Zusätzlich werden die Ergebnisse knapp vorgestellt. Reihenfolge der Themen wird in Kapitel 5 beibehalten, welches den zentralen Raum dieser Arbeit einnimmt. Hier werden die in Kapitel 4 vorgestellten Teilaufgaben genauer ausgeführt. Dort sind die Änsätze, Konzepte, Lösungen und wichtige Details ausführlich dargestellt. Der Leser dieser Arbeit kann dieses Kapitel zunächst überspringen, wenn kein genaueres Interesse an den Feinheiten besteht. Der sechste Abschnitt ist reserviert für die Ergebnisse von Testläufen und Evaluation. Es folgen der Ausblick und das Fazit in den Kapitel 7 und 8. Der Anhang enthält im Abschnitt A eine Zusammenfassung, der verwendeten Design-Pattern mit der Angabe von weiterführender Literatur. Abschnitt B richtet sich an den Pokerbot-Entwickler und soll den Einstieg in die Pokerbot-Implementierung und des Testens erleichtern.

2 Grundlagen

Im folgenden Kapitel werden, die Regeln und die wichtigsten Begrifflichkeiten erläutert. Es soll hierbei besonders auf die Texas Hold Em Varianten und das Kuhn-Poker eingegangen werden, da diese bei der ACPC gespielt werden. Neben den Hold Em Varianten und dem Kuhn-Poker gibt es noch Stout-, Draw- und Würfel-Pokerfamilien, die vor allem durch Texas Hold Em an Popularität eingebüßt haben, aber dennoch erwähnenswert sind. Allen Pokerspielen ist gemeinsam, dass die gleiche Handstärkenhierarchie vorherrscht und dass die Aktionen links vom Dealer ausgehen. Es stehen, üblicherweise in allen Varianten, die gleichen Aktionen zu Verfügung. Im Normalfall unterscheiden sich die Pokerspiele hauptsächlich durch unterschiedliche Bietstrukturen und Kartenausteilmodi.

2.1 Das Pokerspiel

Poker ist ein Kartenspiel, welches mit einem Anglo-Amerikanischen-Kartendeck aus 52 Karten gespielt wird. Alle Pokervarianten haben eine Reihe von gemeinsamen Merkmalen: Actions, Streets, Pot, Rangfolge der Hände, einen Stack von Chips und eine Form von Mindesteinsatz.

Als Street bezeichnet man eine Wettrunde oder Bietrunde von der es in so gut wie jeder Pokervariante mehr als eine gibt. Das grundlegende Spielprinzip ist, durch Wetten auf die eigenen Karten, die anderen Spieler aus dem Spiel zu drängen oder es kommt nach einer festgelegten maximalen Anzahl von Streets zum Showdown. Beim Showdown wird aus fünf Karten, die jeder Spieler anspielt, der Gewinner ermittelt.

2.1.1 Hand, Holecards und Commoncards

Die Holecards sind in jeder Pokervariante vorhanden. Der deutsche Begriff für Holecards ist Handkarten. Diese Karten kennt nur der Spieler, der sie vom Dealer auch ausgesteilt bekommen hat. Einige Pokervarianten wie Stoutpoker kennen auch offene Holecards, die von jedem einsehbar sind, aber beim Showdown nur vom Besitzer anspielbar sind. Commoncards (Gemeinschaftskarten) werden vom Dealer offen ausgeteilt und sind von jedem beim Showdown anspielbar. Als Hand bezeichnet man eine Spielrunde mit allen ausgeteilten Karten.

2.1.2 Karten

Die Eigenschaften der Karten:

- Die Wertigkeit der Karten ist absteigend: Ace(A), King(K), Queen(Q), Jack(J), Ten(T), 9, 8, 7, 6, 5, 4, 3, 2
- Die Farben sind alle gleichwertig: Spades(s), Clubs(c), Diamonds(d), Hearts(h)
- Die Kombination Wertigkeit und Farbe kürzt die Bezeichnung einer Karte ab. So steht die Abkürzung Jd für Jack-Diamonds also Karo-Bube.

Die Rangfolge der Hände:

1. Royal Straight-Flush: Alle fünf Karten sind von einer Farbe nach Wertigkeit von Ace bis Ten.
2. Straight-Flush: Alle fünf Karten sind von einer Farbe und bilden eine durchgehende Reihe in der Wertigkeit.
3. Four of a kind: Vier Karten gleicher Wertigkeit.
4. Full-House: Drei Karten einer Wertigkeit und zwei Karten einer Wertigkeit.
5. Flush: Alle fünf Karten haben die gleiche Farbe.
6. Straight: Alle fünf Karten bilden eine durchgehende Reihe bezüglich der Wertigkeit.
7. Three of a kind: Drei Karten mit gleicher Wertigkeit.
8. Pair: Zwei Karten gleicher Wertigkeit.
9. High Card: Die Karte mit der höchsten Wertigkeit.

2.1.3 Actions - die möglichen Spielzüge eines Spielers

Die deutsche Bezeichnung für Action ist Spielzug.

- check: warten - Abgabe der eigenen Action an den nächsten Spieler. Ein check darf nur ausgeführt werden, wenn alle vorherigen Spieler in der aktuellen Street check gewählt hatten.
- bet: bieten - Der erste Spieler einer Street, der auf den Wert seiner Karten wettet, führt eine sogenannte bet durch.
- (re)raise: erhöhen - Nach einer bet haben die nachfolgenden Spieler die Möglichkeit, den Preis weiter in die Höhe zu treiben. Üblicherweise gibt es in den einzelnen Varianten Regeln zum Mindestgebot.
- call: mitgehen - Nach bet oder raise haben die darauffolgenden Spieler die Option auf call. Haben alle verbleibenden Spieler in einer Street reihum einen call getätigt, sodass keine bet oder raise unbeantwortet bleibt, dann wird in die nächste Street gewechselt oder es kommt zum Showdown nach der letzten Street.
- fold: passen/aussteigen - Wird der Preis für einen call zu hoch oder es besteht keine Chance durch einen eigenen raise die restlichen Spieler herauszudrängen, dann ist fold die letzte Option, die immer nach einem unbeantworteten raise oder bet getätigt werden darf.

2.1.4 Stack, Pot, Sidepot, Mainpot und Chips

Jeder Spieler muss seinen Stack offen auf den Tisch legen, sodass jeder beim anderen sehen kann, wieviele Chips dieser hat. Alle gesetzten Chips kommen in den Pot. Schiebt ein Spieler alle seine verbleibenden Chips in den Pot, dann spricht man vom einem All-In. Ein Spieler, der All-In ist, bleibt bis zum möglichen Showdown passiv und der Pot wird auftgeteilt in einen Mainpot und einen Sidepot. Die Chipanzahl im Mainpot wird aus der Anzahl der noch in der Hand verbleibenden Spieler (alle die den All-In mit call oder raise beantworten) multipliziert mit der Anzahl der Chips des All-In gehenden Spielers berechnet, die restlichen Chips kommen in den Sidepot. Der Auslöser des Sidepots kann nur den Mainpot gewinnen. Der Gewinner des Sidepots wird unter den restlichen Spielern ausgemacht. Es kann mehrere Sidepots gleichzeitig geben.

2.1.5 Ring Game

Als Ring Game bezeichnet man Pokerspielrunden an denen mehr als 2 Spieler beteiligt sind. Oft sind 6 bis 10 Spieler beteiligt. Die Bezeichnung Ring-Game begründet sich durch den Zwang, dass eine Street erst dann beendet ist, wenn alle Spieler als letzte Action einen call oder fold getätigt haben. Zusammen mit der Regel, dass nach einem bet oder raise alle anderen Spieler nochmals eine Action haben, kann diese neue Action wieder ein raise sein, auch wenn die vorletzte Action dieses Spielers ein call war. Es wird reihum im Uhrzeigersinn gespielt und zumeist hat ein Spieler mehrere Male das Zugrecht.

2.1.6 Heads Up

Heads Up ist eine spezielle Situation bei der entweder nur 2 Spieler am Tisch sitzen oder alle Spieler bis auf 2 ausgestiegen sind. Hier muss der Spieler seine Strategie im Vergleich zu den Ring-Games komplett verändern. Ein call eines Spieler beendet eine Street sofort und jeder Einfluss eines Spielers ist maximal, da er die Hälfte aller Knoten im Spielbaum kontrolliert. Dieses Spiel ist am berechenbarsten, weil man sich nur auf die Spielweise eines Gegners einstellen muss, was zu Kombinationen von verschiedenen Gegnerprofilen führt.

2.1.7 Acting-Player, Hero und Villains

Als Hero bezeichnet man den Spieler der entweder beobachtet wird und dessen Entscheidungen diskutiert werden oder während des Spielens bezeichnet man sich auch als Hero. Als Villains bezeichnet man die/den Gegner des Hero beziehungsweise den stärksten Gegner. Als Acting-Player, -Position oder -Seat wird derjenige bezeichnet, der das Zugrecht hat.

2.1.8 Position

Die Position ist ein essentielles Konzept. Die Positionen geben an, wann ein Spieler an der Reihe ist. Die späteren Positionen sind im Allgemeinen im Vorteil, da diese mehr Informationen durch die bereits getätigten Actions über die laufende Hand sammeln konnten als die frühen Positionen. Deswegen gibt es den Ausdruck: „Position auf jemanden haben“, wenn man nach dem betreffenden Spieler am Zug ist; analog dazu den Ausruck: „keine Positionen auf jemanden haben“. Beachtet jemand seine eigene Position nicht, dann wird er relativ schnell von erfahrenen Spielern eingeschätzt und in dessen Strategie integriert (sprichwörtlich: ausgebeutet).

Eigenschaften korrekten Positionspiels:

- In früher Position (Early Position) spielt man die stärksten Hände, weil diese am besten einem raise standhalten. Selten sollte man auch schwächere Hände spielen um weniger stark berechenbar zu sein, weil sonst lernende Gegner meistens aussteigen werden, weil starke Hände erwartet werden.
- In mittlerer Position (Middle Position) dürfen starke Hände gespielt werden. Es empfiehlt sich zu spielen, wenn in früher Position keiner spielt; genauso wie in früher Position gezielt schwächere Hände zu spielen.
- In hinterer Position (Late Position) hat der Hero viel mehr Möglichkeiten. Sind alle Spieler vor einem selbst ausgestiegen, kann man ziemlich schwache Händen spielen, weil nur noch wenige oder keine Spieler mehr auf den eigenen Spielzug antworten werden. Bei starken Händen hat man Aussicht auf einen großen Pot, falls noch weitere Spieler dabei sind. Ist der Pot sehr groß, dann kann sich auch eine relativ schwache Hand lohnen, weil sich normalerweise die Gewinnaussichten noch stark ändern können durch den Übergang in die nächste Street, da neue Karten gedealt werden.

2.1.9 Texas Hold´Em

Die beliebteste und am weitesten verbreitete Variante des Pokerns ist derzeit Texas Hold´Em. Deshalb wurde diese als Studienobjekt ausgewählt. Die Spielregeln sind relativ simpel. Jeder Spieler erhählt zwei Holecards, die privat sind. Es gibt vier Streets mit den Bezeichnungen: Preflop, Flop, Turn und River.

- Preflop: Alle Spieler kennen nur ihre Holecards und die Anzahl der Spieler. Optional können bereits Informationen über andere Spieler als Vorwissen vorliegen (aufgrund bereits gespielter Hände). Es liegen keine Gemeinschaftskarten (Commoncards) auf dem Tisch.
- Flop: Es werden die ersten drei Commoncards gedealt. Das Wettverhalten aus dem Preflop ist bekannt.
- Turn: Eine weitere Gemeinschaftskarte wird gedealt.
- River: Die letzte Gemeinschaftskarte wird gedealt. Verbleiben am Ende des Rivers mindestens 2 Spieler, dann kommt es zum Showdown.

Die Positionen nach Spielreihenfolge:

Für die üblichen Ring-Größen werden folgende Positionen verwendet. Nach jeder Hand rückt der Dealer eine Position im Uhrzeigersinn weiter. Beim 6-Ring werden die unten genannten Positionen verwendet. Beim 3-Ring nur die letzten drei und beim 9-Ring werden UTG+1, UTG+2 und MP+1 eingeführt. In einem 10-Ring kommt noch zusätzlich der MP+2 hinzu.

1. Under the gun (UTG): Der erste Spieler am Zug in der Street Preflop und ist damit eine Early Position. Ab Flop ist dieser nach dem Big-Blind am Zug.
2. Middle Position (MP): Nach den UTG(s)
3. Cut Off: Das ist die erste Late Position und kommt vor dem Dealer an die Reihe und ist somit die zweitbeste Position am Preflop. Ab der Bietrunde Flop ist er als vorletzter an der Reihe.
4. Dealer: Der Dealer ist der Spieler, der die Karten gibt (gibt ein Groupier Karten, dann gibt es trotzdem die Position Dealer). Er hat die beste Position, weil er keine Blinds zahlen muss und als letzter vor den Blinds an der Reihe ist. Ab der Bietrunde Flop ist der Dealer der letzte Spieler.
5. Small Blind: In dieser Position muss vor dem Kartenaussteilen ein halber Big-Blind in den Pot eingezahlt werden. Im Prinzip wird in dieser Position eine Zwangsaktion durchgeführt und zwar links vom Dealer. Für alle Bietrunden, exklusive des Preflops, ist er als erstes mit einem freien Spielzug an der Reihe.

6. Big Blind: Der Letzte Spieler am Preflop, der immer einen Big-Blind in den Pot zahlen muss. In dieser Position tätigt der Spieler links vom Smallblind als Zwangsaktion einen Minraise. Dadurch erklärt sich auch der Positionswechsel der ersten echten freien Aktion bezüglich Preflop und den restlichen Streets.

Den Mindesteinsatz erbringen zwingend die Spieler in den Positionen Big-Blind und der Small-Blind. Der Big-Blind ist ein festgelegter Chipwert. Jeder andere Spieler muss in der Street Preflop den Big-Blind zahlen oder einen fold tätigen. Sollte jemand eine bet oder raise gewählt haben, müssen eben diese gezahlt werden. Der Smallblind erbringt zwangsweise die Hälfte des Big-Blinds und muss wenigstens auch auf den Big-Blind mitgehen, wenn er keinen fold tätigt. Durch die Small- und Big-Blind-Positionen wird sichergestellt, dass immer Chips in den Pot gezahlt wurden.

Fixed Limit Texas Hold´Em

In dieser Variante sind die bet und raises auf einen Big-Blind für die Streets Preflop und Flop festgelegt und für Turn und River bei zwei Big-Blinds. Es sind pro Street eine bet und 3 raises möglich. Somit ist der maximale Preis ist immer berechenbar, um im Spiel zu bleiben. Es kommt hierbei nahezu nie zu einem All-In.

Nolimit Texas Hold´Em

Nolimit ist ein wenig komplizierter. Bei einer bet muss mindestens ein Big-Blind in den Pot gezahlt werden. Wird ein raise getätigt, dann muss mindestens die doppelte Menge Chips der bet in den Pot geschoben werden (Pot nach der bet subtrahiert um den Pot vor bet wird mit zwei multipliziert für einen korrekten min-raise). Alle weiteren raises müssen dann mindestens nur noch die Differenz der beiden vorherigen raises betragen. Das Maximum für bet und raise ist nur durch den eigenen Stack von Chips begrenzt und es gibt kein Maximum von raises. Beim Nolimit-Poker kommt es häufiger zu All-In Situationen als beim Limit-Poker. Wegen der variablen Menge von Chips, mit der eine bet bzw. raise belegt werden kann, ergeben sich viel mehr Knoten im Spielbaum. Damit sind Nolimit-Pokervarianten á priori immer schwieriger beherrschbar als deren Entsprechungen mit Limits.

2.1.10 Kuhn-Poker

Kuhn-Poker dient vor allem dem Studium von Entscheidungen in partiell beobachtbaren Umgebungen. Es ist eine Pokervariante, die auf das Nötigste reduziert ist. Das Kartendeck hat genau eine letzte Karte mehr als Mitspieler. Alle Karten haben unterschiedliche Wertigkeit. Jeder Spieler zahlt in den Pot eine sogenannte Ante bevor er seine Karte bekommt. Die letzte Karte bleibt für alle Spieler unbekannt. Es gibt nur eine Street. Bet und raise sind fixed limit mit genau einer Ante. Der Spieler mit Zugrecht hat die üblichen actions zur Auswahl.

2.2 Begriffe aus dem Computerpoker

2.2.1 Match und Duplicated-Match

Beim Computerpoker wird eine Serie mit festgelegter Anzahl von Händen als Match bezeichnet. Üblich sind mehrere tausend Hände pro Match. Bei einem Duplicated-Match wird ein Match zu Ende gespielt und im Anschluss mit den Handkarten eines anderen Spielers wiederholt. Wenn alle Permutationen durchgespielt sind, ist das Duplicated-Match beendet. Das Duplicated-Match soll den Glücksfaktor, der durch die ausgeteilten Karten entsteht ausgleichen. Dies ist sehr wichtig, um Pokeragenten zu vergleichen.

2.2.2 Doyle's Game Computer-Poker-Competition (2014d)

In diesem Spielmodus haben alle Spieler die gleiche Stackgröße. Nach jeder Hand wird dieser wieder auf den Anfangswert zurückgesetzt. Dieser Modus ist üblich für Computerpoker, weil die Matches für mehrere tausende Hände gehen sollen, um genügend Vergleichsdaten zu bekommen. Würden die Stacks nicht zurückgesetzt werden, dann wären oftmals Spieler vorzeitig aus einem Match ausgeschieden, was entsprechend weniger Daten liefern würde. Der andere Grund ist, dass beim Pokern immer der Spieler im Vorteil ist, der einen nennenswerten Chipvorsprung hat. Im Extremfall kann ein Spieler mit kleinem Risiko andere Spieler in ein All-In zwingen, weil er nur ein Bruchteil seiner Chips riskiert, während seine Gegner ausscheiden könnten.

Im weiteren Verlauf der Arbeit wird immer von der Anwendung von Doyle's Game ausgegangen.

3 Zustand des Pokerbot-Frameworks vor dieser Arbeit

Mit der Erläuterung, welche Funktionaltäten in den verschiedenen Pokerprojekten der KE-Group vorhanden sind und welche die Anforderungen sind, die aus der ACPC entstehen, soll ein Einstieg in die Thematik ermöglicht werden. In Kapitel 4 ist eine Versionierungshistorie zu finden.

3.1 Ausgangssituation

Zu Beginn dieser Arbeit liegen zwei Pokerbot-Frameworks vor. Das urprüngliche TUD-Pokerbotframework von 2010 ist ein Limited-Poker Framework für Heads-Up und Ring-Play, welches von Markus Zopf geschaffen wurde Zopf (2010). Es wird im Weiteren stets Limit 2010 genannt. Eine Form von Versionierung existiert zwar offiziell nicht, hilft aber, die Übersicht zu behalten. Darüber hinaus existiert ein Ablegerprojekt, im Folgenden Nolimit 2013 genannt, welches aus dem Praktikum für künstliche Intelligenzen hervorgegangen ist, um mit einem Nolimit-Texas Hold Em Heads-Up Pokerbot an der Annual Computer Poker Competition 2013 teilzunehmen. Die Entwicklung eines Ablegers aus der Version Limit 2010 war einem knappen Zeitbudget geschuldet. Beide Frameworks sollen nun zu einem einzigen vereinigt werden.

Viele Funktionen von Limit 2010 lassen sich nicht ohne Anpassungen von Nolimit-Bots verwenden und umgekehrt. Nach der Vereinigung sollen Limit- und Nolimit-Bots spieltypunabhängige Funktionen gemeinsam verwenden können, um erstens die Entwicklung neuer Bots zu vereinfachen, da mehr Grundfunktionen zur Verfügung stehen. Zweitens wird der Wartungsaufwand gesenkt, weil nur noch ein Framework gepflegt werden muss. Im Pokerbot-Framework ist eine „Grafische Benutzeroberfläche für ein Pokerspiel“ Bank (2011) eingebunden, die auch erweitert werden muss, um sie für Nolimit-Poker tauglich zu machen. Weiterhin gibt es eine Menge von Modulen, deren Wiederverwertung wünschenwert ist, um in Zukunft die mehrfache Entwicklung von ähnlicher Programmfunktionalität zu vermeiden. Diese Module sind in Ablegerprojekten diverser Abschlussarbeiten und Praktika enthalten.

3.2 Einfluss der Annual Computer Poker Competition der Jahre 2013 und 2014

Die Teilnahme an der ACPC 2013 hat gezeigt, dass es nicht einfach ist, ein auf Limited-Poker ausgelegtes Framework innerhalb von 6 Wochen - inklusive der Entwicklung eines Bots - in ein gutes Nolimit-Pokerframework zu überführen. In der Kürze der Zeit wurden nur die Kommunikationsschnittstelle zum Server der ACPC, die Spielzustandserzeugung und der Spielzustand angepasst. Dieses Framework hat zwar funktioniert. Elegantere Lösungen sind jedoch vorstellbar.

Die ACPC kennt verschiedene Bewertungsmodi. Die beiden wichtigsten sind die „Total Bankroll“ Computer-Poker-Competition (2012c) und „Total Bankroll Instant Runoff“ Computer-Poker-Competition (2012b). Im „Total Bankroll“ Modus sollen die gegnerischen Bots so stark wie möglich ausgenutzt werden, um die maximale Anzahl von Chips zu erreichen; während es beim „Bankroll Instant Runoff“ wichtig ist, eine maximale Anzahl an Gegnern so sicher wie möglich (also mit geringem Risiko) zu besiegen. Es spielt jeder gegen jeden und der Teilnehmer mit den wenigsten Siegen scheidet aus. Das Turnier ist beendet, wenn nur noch ein Teilnehmer verbleibt. Aus den verschiedenen Anforderungen geht die Sinnhaftigkeit, dem eigenen Bot ein Ziel übermitteln zu können oder spezielle Bots für den jeweiligen Modus zu schreiben hervor. Das Framework könnte dieses unterstützen. Dank der zurückliegenden Teilnahmen an der ACPC ergibt sich ein weiterer Aspekt für das Testen, der bei der Fortentwicklung der Pokerbots und auch des Frameworks eine Rolle spielen soll: Testläufe gegen die gegnerischen Bots aus dem Vorjahr. Als ehemaliger Teilnehmer darf man bis zur nächsten Pokerbot-Challenge Testspiele gegen die Teilnehmer auf den Servern der University of Alberta durchführen. Das bedeutet, zu testende eigene Pokerbots werden lokal gestartet und übermitteln ihre Spielzüge an den Server und diese erhalten als Antwort die Züge der Gegner. Dies ist eine völlig andere Art des Testens als innerhalb des eigenen Frameworks gegen eigene Bots zu spielen, bei dem man die volle Kontrolle über den Server und die angemeldeten Clients hat. Der Testmodus auf den Servern der University of Alberta sollte besser in das Pokerframework eingebunden werden, sodass er leichter zu handhaben wird. Mit der Pokerbot-Challenge 2014 wurde ein neuer Spielmodus eingeführt, das Kuhn-PokerComputer-Poker-Competition (2014e). Für diese Pokervariante wird unter Umständen eine neue oder zusätzliche Datenmodellierung für das TUD-Pokerframework benötigt. Die Veranstalter der Pokerbot-Challenge wünschten bereits im Vorfeld der Pokerbot-Challenge, dass die Bots in Zukunft weniger Ressourcen verbrauchen sollenComputer-Poker-Competition (2014a).

3.3 Features aus dem Pokerframework aus dem Jahr 2010 von Markus Zopf

Das Limit Pokerbotframework aus dem Jahre 2010 wird in an dieser Stelle mit seinen wichtigsten Eigenschaften und Modulen vorgestellt. Zopf (2010)

- Es ist ein mehrstufiges Nachrichtenzustellsystem vorhanden.
- Ein Server mit Accountverwaltung, auf dem Games ausgeführt werden.
- Die Klasse Game hat im Prinzip die Aufgabe eines Dealers und sorgt dafür, dass jeder Teilnehmer auf neuestem Stand ist, die Karten austeilt und das Endergebnis ermittelt.
- Eine Klasse PokerClient, von der die Pokerbots erben, welche eingehende Daten in Form eines GameState bereitstellt und ausgehende Daten(normalerweise Spielzüge) versendet.
- Eine Klasse namens GameState, welche den aktuellen Spielzustand abbildet. Die PokerClients haben ein Attribut dieser Klasse, welches vom Clientterminal des Nachrichtenzustellsystems geupdated wird.
- GameStateFactory erzeugt aus einem String eine GameState-Instanz. Die ACPC-Server und das Nachrichtenzustellsystem verwenden die gleiche String-Repräsentation der Spielzustände.
- TUDPokerClient ist eine Zwischenstufe der Kommunikation zwischen dem Nachrichtensystem und dem PokerClient. Der TUDPokerClient hat einen passiven Modus, womit er nach außen keinen Einfluss ausübt und einen aktiven Modus mit folgenden Hauptfunktionen: Einen Server starten dürfen, Pokerbot-Instanzen einem Game zuordnen und Games starten dürfen. Die Nebenfunktion ist, eine Basis für eine GUI zu schaffen.
- Eine GUI, die zum Spielen gegen Pokerbots und zum Einstellen des Servers (für aktive TUDPokerClients) dient.
- Es existiert ein Loggingserver, der am Nachrichtenzustellsystem angeschlossen ist.
- Eine UCT-Simulation wurde umgesetzt und wird im kleinen Umfang vom PokerClient CheckNorris angewendet.
- Im Package divat existieren einige Metriken um den Spielzustand einschätzen zu lassen. Einige dieser Metriken werden vom PokerClient RichieRich demonstriert.
- Es gibt einen HandRanger für die Einschätzung der eigenen Handkarten mit den Commoncards.

3.4 Features aus dem Nolimit-Pokerframework aus dem Jahr 2013

Die Feature des Nolimit 2013 Frameworks sind für den Pokerbot ArizonaStu Tomasz Gasiorowski (2013) erstellt worden. Sie sind bereits durch ACPC Teilnahmen erprobt.

- Ein Gegnermodell für Texas Hold em Heads Up. Die Basisklasse heißt OpponentModelBase. In die Updatemethode werden abgeschlossene Hände übergeben. Packagename: pokertud.opponentModel2p

  Features:

  - OpponentModelBase erhebt einfache statische Daten wie fold-Verhalten für die einzelnen Straßen, gespielte Hände in den Straßen, Showdown- und WonAtShowdown-Zähler jeweils für beide Positionen. Eine Methode namens getwhat(String) kann nach pokertypischen Abkürzungen befragt werden, welche dann eine Kennzahl liefert. Alle folgenden Add-Ons werden von der OpponentModelBase mitgeupdated.
  - Die Klasse ActionTree baut einen Baum auf, dessen Kanten komplette Spielzugfolgen einer Bietrunde sind. Betsizes werden nicht beachtet. Der Baum listet GameStates auf, die nach Spielzugfolgen sortiert sind, auf. Das Argument der Suche sind Prefixe der Spielzugfolgen oder komplette Spielzugfolgen. Diese Listen können dann weiter untersucht werden, um beispielweise herauszufinden, wie oft der Gegner unter den gewählten Voraussetzungen zum Showndown kam oder mit welchen Karten der Gegner das Spiel zum Showdown kommen lässt. Viele weitere Suchfunktionen sind darin enthalten.
  - Im ActionTreeObject, also einem Knoten können einfache Erhebungen über den eigenen Zustand und den des Gegners abgefragt werden. Diese Klasse hält eine Liste von zutreffenden GameStates vor. Jedes ActionTreeObject hat einen einfachen Ranger, den ShowdownRanger.
  - Der ShowdownRanger führt Buch über die Showdowns, welche das zugehörige ActionTreeObject gesehen hat. Ein Pokerbot kann den ShowdownRanger nach der Liste der Handstärken, dem Median, nach Quartillen oder nach einem Intervall der Handstärke fragen. In gleicher Weise können auch die zugehörigen GameStates geliefert werden.

  - Es existieren verschiedene Erben der Klasse StandardPipe. Eine Pipe ist in der Länge einstellbar und beobachtet eine Historie von GameState. Pipes sollen einen Durchschnittswert über kleine Anzahlen von Händen liefern, um ein Indiz zu erhalten, ob der Gegner seine Strategie anpasst. Ein Durchschnittswert über alle gespielten Hände verändert sich mit zunehmender Anzahl gespielter Hände immer langsamer, bis man Änderungen am Spielerverhalten nicht rechtzeitig bemerkt. Beispielsweise ist eine Pipe für Preflop fold-Verhalten umgesetzt.
  - Das Gegnermodell ist serialisierbar, damit ist es persistierbar und ermöglicht es in späteren Sitzungen bereits Gelerntes weiterzuverwenden.
  - JUnit-Tests sind für weite Teile des Gegnermodells vorhanden.

- Das Nolimit 2013 hat einen verbesserten EV-Calculator (Expected Value), der zwei bis drei Größenordnungen schneller ist als sein Vorgänger aus Limit 2010. Ein einfacher Pokerbot namens EVBot basiert auf diesem verbesserten EV-Calculator. Zudem ist dieser EV-Calculator gut als Fallback-Strategie verwendbar. Er berechnet All-In-Equity gegen eine Range von Karten, gegen zufällige Karten und eine All-In-Equity gegen eine gegnerische Hand bei Variationen der Commoncards.

  Package: pokertud.divat.ev

- Der Converter ist eine Klasse, die Ergebnisse von der ACPC zu PokerTracker Log-Dateien umwandelt. Der PokerTracker ist ein Unterstützungstool für Online-Pokerspiele mit vielen Analysenfunktionen und Replayfunktionen.

  Package: pokertud.util.converterFormACPCfilesToPokerTrackerFiles

- Der HandTypeRanger und AllinEquityRanger liefern beide mit Hilfe des Opponentmodels Listen von Karten, die der EV-Calculator nutzen kann, um die Berechnung präziser durchzuführen, indem der gegnerische Kartenbereich eingeschränkt wird.

  Package: as.rangeTest

- Nolimit 2013 besitzt eine regelbasierte Pokerbot-Steuerung. Das Regelwerk ist vorgesehen für Heads Up. Die Steuerung besteht aus folgenden Komponenten:
  - Einem Regelparser mit Scaner, Tokenizer und Parser. Die Regeln sind textbasiert und somit auch für Menschen schreib- und lesbar. Sie bestehen aus einer Actionsequenz als Bedingung. Im Body werden die einzelnen Fälle jeweils einer Zeile beschrieben. Relationen werden mit Klammern und logischen Junktoren verbunden. In den Regeln werden pokertypische Abkürzungen genutzt, um Konzepte zu beschreiben.

    Package: as.rulebot.parser
  - Der Regelinterpreter der Steuerung unterstützt eine Menge von einzelnen Statementtypen, die von der Klasse Statement abgeleitet sind, welche wiederum von der Klasse AST erbt, welches das Visitor-Pattern umsetzt.

    Package: as.rulebot.ast
  - Einem Regelinterpreter, der eine Anbindung an das OpponentModel und die Ranger zulässt.

    Package: as.rulebot
  - Der Rulebot selbst ist auch bekannt unter den Namen ArizonaStu oder KEmpfer

4 Einordnung der geprüften Thesen und angegangene Ziele

Die in Kapitel 1.3 beschriebenen übergeordneten, abstrakten Aufgaben werden in diesem Kapitel als konkrete Ziele und Thesen vorgestellt (sortiert nach Bearbeitungsbeginn). Die Implementierung der Neuentwicklungen ist am 16 Juni 2015 eingestellt worden (Bugs wurden weiterhin beseitigt). Im Folgenden wird der Zustand der Teilaufgaben dokumentiert. Genauere Beschreibungen der Lösungen, Konzepte und Evaluation der Teilprobleme finden sich in Kapitel 5. Mit der folgenden Übersicht wird eine Versionslegende eingeführt, die das Lesen der Ausarbeitung erleichtern wird. Diese Bachelorthesis baut Limit 2010 und Nolimit 2013 auf. Version Nolimit 2014 ist die erste Weiterentwicklung im Rahmen dieser Arbeit.

- Limit 2010: Die Version des TUD-Pokerframeworks von Markus Zopf für Fixedlimit Texas Hold Em.
- Nolimit 2013: Eine eingeschränkte Ablegerversion des TUD-Pokerframeworks für Texas Hold Em Nolimit.
- Nolimit 2014: Weiterentwicklung von Nolimit 2013 mit Vorbereitungen für Version 1.0.
- Limit und Nolimit Version 1.0: Texas Hold em Limit und Nolimit in einem Framework.
- Limit und Nolimit Version 1.1: Erweiterung um den Spieltyp Kuhn-Poker.
- Limit und Nolimit Version 1.2: Texas Hold em Limit und Nolimit mit verbessertem Netzwerkverkehr.
- Limit und Nolimit Version 1.3: GameState-Updates können effizienter durchgeführt werden.
- Limit und Nolimit Version 1.4: Es können Module aus anderen Arbeiten übernommen werden.
- Limit und Nolimit Version 1.5: Neuentwurf einer Serversteuerung.
- Limit und Nolimit Version 1.6: GUI für Menschen.
- Limit und Nolimit Version 1.7: Beobachtungs-GUI.
- Limit und Nolimit Version 1.8: Abgabeversion mit Unterstützung von Legacy-Bots, basierend auf Version 1.2 mit den Verbesserungen aus den Versionen 1.3, 1.4, 1.6 und 1.7.
- Limit und Nolimit Version 1.X: Ist eine Abkürzung und steht für alle Versionen zwischen Version 1.0 und 1.8, sowie auf diesen basierende Nachfolger. Es ist eine Sammelbezeichnung.
- TUD-Pokerframework Version 2.0: Neugestaltung des Pokerclients, des Servers und des Dealers.

Limit und Nolimit Version 1.8 ist die Abgabeversion im Rahmen dieser Thesis. In den folgenden Kapiteln dieser Bachelorthesis wird nur die Versionsnummer angegeben für alle Limit und Nolimit Versionen.

4.1 Das TUD-Pokerframework kann für Texas Hold Em Nolimit adaptiert werden

Durch direktes Portieren konnte die Version Nolimit 2013 in Nolimit 2014 überführt werden und an Praktikanten der KE-Group übergeben werden, um Pokerbots zu testen. Es gab keine andere Möglichkeit für Testspiele, da der ACPC-Server der University of Alberta wegen technischer Probleme bis zum November 2014 ausgefallen war.

4.2 Im TUD-Pokerframework können die Texas Hold Em Varianten Limit und Nolimit koexistieren mit Offenhaltung einer Erweiterung für Kuhn-Poker

Da die Vereinigung beider Texas Hold Em Varianten gelungen ist, liegt nun die Version 1.0 eines gemeinsamen Frameworks vor.

Diese Version wurde ebenfalls sofort nach einer kurzen Testphase an die Praktikanten ausgeliefert. Es wird eine Vererbungszwischenschicht zwischen PokerClient und Bot eingesetzt, um spezielle Verhaltensweisen für Limit und Nolimit zu implementieren, eine kleine Erweiterung des Loginvorgangs und es gibt nun Erben von der Klasse GameState für beide Texas Hold Em Varianten.

4.3 Arbeitsthese: Kuhn-Poker ist in das bestehende TUD-Pokerframework direkt einbindbar

Kuhn-Poker ist nicht direkt einbindbar und wurde nicht zu Ende implementiert.

Das TUD-Pokerframework ist nicht ohne größere Umbauaufgaben am Server, Game und GameStateFactory möglich. Die konkreten Probleme werden im entsprechenden Kapitel 5 beschrieben. Eines der Hauptprobleme ist die fehlende Dokumentation zum Quellcode von Limit 2010. Einige Verbesserungen am TUD-Pokerframework wie Code-Vereinfachungen sind das Ergebnis dieser gescheiterten Version 1.1. Kuhn-Poker wurde für die restliche Bearbeitungszeit nicht weiterverfolgt.

4.4 Überarbeitung des Netzwerkverkehrs über das bestehende Nachrichtensystem

Die Kommunikation konnte verbessert werden.

Es wurden immer wieder bei Tests Anomalien im Nachrichtenfluss registriert. Sodass der Auftrag erteilt wurde, diese zu untersuchen. Die Netzwerkstruktur konnte vereinfacht werden, was zu der Version 1.2 führt. Die Anomalien konnten vollständig entfernt werden. Leider wurde hierdurch die GUI, welche von Max Bank Bank (2011) geschaffen wurde, nicht mehr in dieser Form verwendbar. Das Ergebnis ist ein schnelleres, stabileres und einfacheres Framework.

4.5 Gamestate Updates können effizienter durchgeführt werden

GameState Updates sind effizienter und eleganter durchführbar.

Mit weitergehenden Änderungen am Framework, die das Datenmodel des GameState betreffen könnte die Arbeitsgeschwindigkeit weiter erhöht werden. Nebenbei konnte die Klasse Game unabhängig von anderen Klassen vereinfacht werden und die Verantwortlichkeit der Auswertung von Actions der Handausgängen vollständig an die Klasse Game in ihrer Funktion als Dealer delegiert werden. Somit parsen die PokerClients nur beim Spielen mit dem Alberta-Server aufwendig Strings zu einem GameState, was an dieser Stelle gerechtfertigt ist, da dort plattformunabhägig ASCII Zeichenketten übermittelt werden. Beim TUD-Pokerframework ist es unerheblich, ob eine zu serialisierende Nachricht nur mit String beladen ist oder gleich mit einem korrekten GameState für genau einen Empfänger abgesendet wird. Die Erzeugung der Strings bzw. jetzt GameStates konnte nicht vereinfacht werden. Das Ergebnis ist Version 1.3 mit weniger Nachrichten Overhead und Arbeitsschritten.

4.6 Es können Module aus Ablegerprojekten und Abschlussarbeiten übernommen werden

Dieser Bearbeitungsabschnitt ist nicht vollständig durchgeführt worden.

Es gibt relativ viele Arbeiten von verschiedener Quellcode- und Dokumentationsqualität, die einen stark unterschiedlichen Aufwand beim Einbinden verursachen. Einige Module können direkt verwendet werden, anderen genügt eine Parametrisierung, um diese für Nolimit und Limit verwendbar zu machen. Selten wird auch das Nachrichtenwesen aus der Zeit vor Version 1.2 benötigt. Bei anderen ist die verwendete JAVA-Version bereits stark veraltet. Der Schwierigkeitsgrad kann zu einem eigenständigen Praktikum ansteigen, Module auf Nolimit zu übertragen, wenn diese konzeptionell nur für Texas Hold Em Limit ausgelegt sind. Prinzip bedingt kann es Module geben, die niemals für Nolimit geeignet sein werden. In Kapitel 5.6 gibt es eine Auflistung von übernommenen Modulen und eine Untersuchung der Übernahmeproblematik.

4.7 Neuentwurf einer Serversteuerung

Zustand: Das Konzept ist an einem Testserver erprobt worden mittels JUnit-Tests, Commands sind zum Teil fertiggestellt. Die Abbänderungen am TUD-Poker-Server stehen aus. Version 1.5 ist nicht fertiggestellt.

Derzeit hat der Server keine echte Steuerungsschnittstelle. Im Konsolenmodus arbeitet er weitestgehend im Batchbetrieb. Die Pokerclients verbinden sich mit einem gestarteten Server und der „Operator“ kann lediglich ein Game starten, wenn sich alle Teilnehmer verbunden haben und ansonsten hat er keinen weiteren Einfluss. Einstellungen müssen am Quellcode vorgenommen werden. Im GUI-Modus können sogenannte aktive Spieler in der StartGame Message einige einmalige Einstellungen vornehmen:

- Die Anzahl der Hände, die gespielt werden.
- Die Gegner des Sendenden starten.
- Ob der Sendende ein Beobachter sein soll.

- Ob, die Namen, der Spieler angezeigt werden sollen.
- Ob ein sogenanntes duplicate-Match vorliegt, bei dem alle Permutationen einer Hand bezüglich der Holecards in getrennten Matches ausgespielt werden.

Die GUI von Markus Bank Bank (2011) bietet ein wenig mehr Komfort. Leider ist diese GUI nicht mehr verwendbar seit den Änderungen im Nachrichtenverkehr. In Hinblick auf neue Spieltypen und deren Parameter wird eine facettenreichere Steuerung immer wichtiger, welche die alte GUI nicht bietet zumal diese nur für Texas Hold Em Limit konzeptiert ist und in jedem Fall eine starke Überarbeitung benötigt. Hierzu wurde die Entwicklung des sogenannten ServerCommander begonnen. Die Entwicklung wurde wegen des veralteten Servers eingestellt, der drigend wegen deprecated Methoden überarbeitet werden muss. Der ServerCommander nimmt textuelle Befehle entgegen, die in mehreren Stufen überprüft werden. Er ist mit einer eigenen Kommunikationschnittstelle ausgestattet und arbeitet als eigenständiger Thread als Bestandteil des Servers. Die ServerCommander-Clients brauchen nur die passenden Befehle als String zu erzeugen. Dies eröffnet die Optionen, dass eine GUI die Befehle generiert und sendet oder dass mit einem Terminal-Programm wie Netcat oder Telnet die Befehle direkt durch den Benutzer eingegeben werden. Es wurde bei diesem Konzept besonders darauf geachtet, dass der Server als Konsolenprogramm gestartet werden kann und sich der „Operator“ mit seinem Steuerungsclient immer wieder am Server anmelden kann.

4.8 Spieloberfläche Mensch gegen Pokerbot

Zustand: Erledigt. Version 1.6 ist fertiggestellt. Wegen der Änderungen am Nachrichtensystem ist die vorhandene GUI nur mit Anpassungen nutzbar. Auch in Hinblick auf neue Spieltypen müssen Änderungen vorgenommen werden. Eine Neukonzeption wurde begonnen und schneller wieder verworfen. In der vorhandenen Spieloberfläche ist nur eine Unterstützung von Texas Hold Em Nolimit praktikabel. Um nicht für jeden Spieltyp eine eigene GUI entwerfen zu müssen, ist es nötig den GameState abzuändern. Hilfreich ist eine Aufteilung des GameStates in einen Controller, einem Model und einer Strategie mit Spielregeln und Model-Update-Befehlen, die vom Controller konsumiert werden. Die Spieloberfläche der Version 1.6 sieht bereits schon vor diese GUI, wie ein PokerClient agieren zu lassen und die Steuerung des Servers an den ServerCommander zu delegieren. Mit dieser Spieloberfläche kann sich der Spieler von einem Pokerbot beraten lassen. Für das TUD-Pokerbotframework 2.0 werden massive folgen müssen.

4.9 Spielbeobachtung

Zustand: Erledigt. Version 1.7 ist fertiggestellt. Die gleichen Probleme wie die GUI für Spiele Mensch gegen Pokerbots hat die Spielbeobachtung. Derzeit ist nicht abschließend geklärt, ob die Spielbeobachtung über die Schnittstelle des ServerCommander erfolgen soll oder über das Nachrichtensystem der PokerClients. Die Spielbeobachtung ist für Versionen 1.X zufriedenstellend.

4.10 Unterstützung von Legacy-Pokerbots

Zustand: Version 1.8 ist abgeschlossen als Abgabeversion. Es gibt einige Texas Hold Em Limit PokerClients, die über die Jahre erstellt wurden. Diese sollen mit der Überarbeitung des TUD-Pokerframeworks weiterhin funktionstüchtig sein. Mit den Jahren gab es immer wieder kleinere Änderungen am TUD-Pokerframework, die Inkompatibilitäten hervorrufen, welche sich durch falsche Aktionen oder Fehlinterpretation der GameState Strings äußern. Der Server und das Game wurden, angepasst LegacyBot-konforme GameState-Updates zu versenden. Der Server und das Game verhalten sich nun genauso wie es Legacy-Bots erwarten. Alle Legacy-PokerClients, die zu Tests herangezogen, wurden laufen einwandfrei.

4.11 Von der Modularisierung zum TUD-Pokerframework 2.0

Die Arbeiten an TUD-Pokerframework 2.0 sind begonnen.

Um die ausstehenden Aufgaben elegant zu lösen, sind Änderungen an der Spiellogik und der internen Zustandsrepräsentation zweckmäßig. Die Zustimmung zur Abänderung wurde erst spät in der Bearbeitungszeit erteilt. Als wichtigste Maßnahme ist vorgesehen, Updates des GameStates unabhängig von der GameStateFactory zu machen; d.h. eine PokerClient Aktion soll nicht mehr die komplette Neuerzeugung des GameStates nach sich ziehen. Die Controllerfunktionalitäten werden aus dem GameState gezogen. Die Texas Hold Em Limit und Nolimit Regeln werden aus dem GameState und seinen Erben gezogen und jeweils in einer Strategie gelagert, die dann der Controller konsumiert. Der GameState verbleibt als reines Modell, in dem die Spieldaten in verschiedene Container eingelagert werden. Die PokerClients sollen

ihre Action nur noch über den Controller abwickeln, welche diese auf Regelverletzungen prüft und gegebenenfalls den GameState updated und den Spielzug als Nachricht weiterleitet.

Die Klasse Game soll abgeschafft werden und durch einen PokerClient ersetzt werden, der vom Server die Erlaubnis bekommt, die Methoden eines Dealers auszuführen. Damit ist sichergestellt, dass eine Abbänderung der Strategie sowohl PokerClients wie auch den Dealer erreicht, da deren Controller die selbe Strategie verwendet.

Der Server braucht Überarbeitung in seiner Thread-Verwaltung, diese hat den Stand von JAVA 1.4. Dabei können die Anpassungen für den ServerCommander und den Dealer miterledigt werden.

Ein weiterer Schritt zur Modularisierung ist die Parametrisierung des GameStates, Controllers und der Strategie bezüglich Bietrunden und der Positionen. Die Teilcontainer des GameStates sind voll generisch und können damit verschiedenste Pokertypen abbilden. Mit der Strategie wird es in Zukunft einfach, neue Pokertypen zu implementieren, da genau 3 Klassen implementiert werden müssen. Es werden keine Erben von GameState, PokerClient und Game für jeden Spieltyp geschaffen. Der Umbau hebt das TUD-Pokerframework auf JAVA 1.8.

Es ist eine weitreichende Überarbeitung, die wohl zurecht TUD-Pokerframework 2.0 genannt werden darf.

Zustand: Strategie für Fixed-Limit implementiert, jedoch noch die Dealer Regeln dafür. Der Controller fertig gestellt. Die Implementierung des GameState ist in einem fortgeschrittenen Stadium und mit JUnit (soweit fertiggestellt) getestet. Konzeptionell sind PokerClient und der neue Dealer fertiggestellt. Der Server ist noch nicht bearbeitet worden. Die Fertigstellung wird nicht mehr im Rahmen dieser Bachelorthesis geschehen.

4.12 Einbinden von UCT in die modularisierte Variante des TUD-Pokerframeworks

Diese Aufgabe ist als angedacht einzustufen. Jedoch sollte diese Aufgabe einmal gut durch das TUD-Framework 2.0 erfüllbar sein.

5 Konzepte, Ansätze und Implementierungen der Thesen und Ziele

Je nach Entwicklungsstand der vorgestellten Modularisierungsstufen, stellt dieses Kapitel nicht nur Ansätze und Konzepte vor, sondern zeigt auch wichtige Details über die Implementierung und gegebenenfalls auch deren Evaluation. Die Themen wurde wegen ihren Verzahnungen oftmals parallel bearbeitet, wodurch die Lösungsfindung gegenseitig beinflusst wurden.

5.1 Die Adaptierung des TUD-Pokerframework für Texas Hold Em Nolimit

Die Adaptierung des TUD-Pokerframework für Texas Hold Em Nolimit ist im Rahmen dieser Thesis implementiert und erprobt worden. Sie wird Nolimit 2014 genannt. Es werden von Nolimit 2013 die Klassen GameStateNolimit und GameStateFactoryNoLimit verwendet und verbessert. Es sind lediglich Anpassungen von GameState und GameStateFactory, aber keine Erben. In Nolimit 2013 auch die Klassen ActionNolimit und PlayerNolimit verwendet, die bis Version 1.8 jedoch abschafft wurden, um die Adaption zu vereinfachen.

5.1.1 Konzept der Adaptierung für Texas Hold Em Nolimit

Um Nolimit statt Limit zu spielen, genügt es, im GameState eine Regel durch eine andere zu ersetzen. Die Cap-Regel, welche besagt, dass es nur 4 bet beziehungsweise raise in einer Street geben darf, wird ersetzt durch die All-In-Regeln. Ein Spieler der All-In ist, wird passiv gesetzt bis zum Showdown. Der Sidepot wird nicht beachtet, da im Wettbewerbsformat der ACPC alle Spieler zu Beginn einer neuen Hand wieder einen vollen Stack haben und beim Nolimit nur Heads Up gespielt wird. Im Falle eines All-In hat der Dealer eine zusätzliche Regel, weil er noch die restlichen Commoncards austeilen muss.

Das Nachrichtensystem benötigt eine Anpassung, weil es nun nicht mehr ausreicht nur die gewählte Action an den Dealer zu übertragen, denn es kommt nun die bet-size beziehungsweise die raise-size hinzu. Somit muss nun auch die Erzeugung des Spielzustands sowie der Spielzustand selbst geändert werden, da diese auch nur auf das Enum Action ausgelegt sind. Als letzter Schritt ist die Erzeugung der String-Repräsentation von ACPC-Notation zu übernehmen. Der Dealer, also die Klasse Game, wird von GameState auf GameStateNolimit umgestellt.

5.1.2 Wichtige Implementierungsdetails der Adaptierung für Texas Hold Em Nolimit

Die Umsetzung der All-In-Regel wurde durch eine Erweiterung der Klasse Player, die eine Kompenente des GameStateNolimit ist, begonnen. Im Gegensatz zu Limit 2010 und Nolimit 2013, ist bei Nolimit 2014 in der Klasse Player ein neues Enum Feld namens PlayerState mit folgenden möglichen Werten eingeführt worden:

- Bet
- Raise
- Reraise
- Cap
- HigherNoLimitRaise
- MinRaise
- Call
- Fold
- AllinBet
- AllinRaise
- AllinCall

Die Belegungen MinRaise und AllinCall finden in dieser direkten Adaption noch keine Anwendung, da AllinCall ein Ring-Game erfordert und MinRaise derzeit nicht geprüft wird.

Neuerdings wird das Feld PlayerState bei jeder Auswertung einer Action neu berechnet. Üblicherweise findet die Neuberechnung analog zu den restlichen Feldern des Players (während der Erzeugung eines GameStateNoLimit) statt oder wenn der Dealer eine Action von einem Spieler annimmt. Zweck des Feldes ist, die Erzeugungs- und Steuerungslogik zu vereinfachen. Dadurch sind im GameState folgende Methoden realisiert worden:

- public boolean allChecked()
- public boolean allCalled()
- public boolean allAllIn()
- public boolean isShowDown()
- public boolean isFinished()

Diese Methoden erfüllen alle den gleichen Zweck, eine neue Street einzuleiten und den Dealer gegebenenfalls ein neue Commoncard austeilen zu lassen oder beenden die Hand. Damit vereinfacht sich die Logik. Mit dem Wegfall der Klasse PlayerNolimit, die beispielsweise boolsche Felder für den Allinzustand oder auch für den fold verwendet hat, konnten leicht ungültige Zustände entstehen. Für eine möglichst fehlerfreie Funktion ist ebenfalls wichtig, dass die Änderungen an Server und Game minimal bleiben. Denn Server und Game arbeiten mit einer Klasse namens Account, um bei der Kommunikation Spielzüge bestimmten Player zuzuordnen (wenn PlayerNolimit verwendet worden wäre, hätte weitere Änderungen nach sich gezogen). Somit liegt die Verantwortung für Handhabung für die Allins nicht mehr bei den Bots selbst, sondern beim GameStateNolimit. Als Korolar erhält der Bot-Entwickler genauere Informationen über die Player, ohne diese selbst berechnen zu müssen.

Das Enum Action war nicht ausreichend und die Klasse ActionNolimit hatte keinen Mehrwert gegenüber der Integer-Repräsentation der Actions. So wird folgende Codierung in Nolimit 2014 eingeführt:

- -1 steht für ein fold
- 0 steht für check beziehnungsweise call
- Ganze Zahlen größer 0 stehen betsizes beziehungsweise raisesizes

Mit dieser neuen Codierung werden alle Felder und Methoden, die Actions enthalten, refactored. Das Refactoring umfasst besonders die Klassen GameStateNolimit, GameSateFactoryNoLimit und Game bezüglich der Zustandsrepräsentation.

Im Nachrichtenzustellsystem wird die ActionMessage um ein String Feld erweitert, welches von der Klasse ClientPokerClient (das Clientterminal des Nachrichtenzustellsystems) beschrieben wird. Der PokerClient hat die Verantwortung, dem ClientPokerClient eine korrekte Action in String Repräsentation zukommen zu lassen. Die Codierung der Action in einer ActionMessage sieht jetzt wie folgt aus:

- „f“ für fold
- „c“ für call oder check
- „r#“ für eine bet oder raise auf den Chipmenge mit dem der Spieler am Pot investiert sein möchte.

Texas Hold Em Limit und Nolimit benutzen ab Version 1.0 die gleiche Übertragungsmethodik (ein Nachrichtentyp für Limit und Nolimit). Auf das Parsen und Erzeugen von Spielzuständen in String Repräsentation wird nicht näher eingegangen.

5.1.3 Evaluation der Adaption für Texas Hold Em Nolimit

Zu Testzwecken ist ein lokaler ACPC-Server aufgesetzt worden um die Funktionstüchtigkeit mit dem AlbertaClient (Kommunikationsschnittstelle zu ACPC-Servern) sicherzustellen. Der angepasste TUD-Server wurde mehrere Male getestet und dann an wartende Praktikanten weitergereicht. Die Praktikanten stellten mangelnde Threadsicherheit des Servers fest. Das Problem konnte durch stärkere Synchronisierung des GameState-Update-Vorgangs erreicht werden.

5.2 Die Vereinigung beider Hold Em Varianten mit Erweiterungsmöglichkeit für Kuhn-Poker

In der Version 1.0 sind nun beide Hold Em Varianten mit Erweiterungsmöglichkeit für Kuhn-Poker implementiert und erprobt. Die Klasse GameStateFactoryNolimit wird nicht weiter benötigt.

5.2.1 Konzepte der Vereinigung von Limit- und Nolimit-Framework

Zuerst werden die clientseitigen Änderungen zwischen der Version 1.0 und ihren Vorgängerversionen beschrieben. Die erste Maßnahme ist eine Vererbungsschicht zwischen dem PokerClient und dem eigentlichen Pokerbot einzuführen. Der GameState analog zu den PokerClient-Zwischenschichten vererbt.

Änderungen auf der Clientseite - Neue Erben des PokerClients

In den neuen abstrakten Erben von PokerClient mit den Namen PokerClientLimit, PokerClientNolimit und PokerClient-Kuhn sind die Klassennamen für den verwendeten GameState und Spielvariante nun hart codierte Felder. Beim Instanzieren des PokerBots werden diese beiden Felder jetzt ausgelesen und mit der AccountloginMessage an den Server gesendet. Der Konstrukter des Bots setzt den Klassennamen des GameStates in die GameStateFactory ein. Die PokerClient-Erben implementieren die Methode zur Erzeugung, die String-Repräsentation der Spielzüge für ActionMessages.

Die Änderungen am Nachrichtenzustellsystem sind nicht sehr umfangreich. Die AccountLoginMessage enthält nun Informationen zum Pokeragenten bezüglich der Pokervariante und des verwendeten GameState-Erben. Die ActionMessage wurde aus der Version NoLimit 2014 übernommen für alle PokerClient-Erben.

Serverseitige und clientseitige Veränderungen - GameState und seine Erben

Bei Nolimit 2014 liegen die Spielzustände in den beiden Ausführungen GameState und GameStateNolimit vor. In Version Nolimit 2014 wurde GameState nicht verwendet, stattdessen nur GameStateNolimit, welches aber nur eine angepasste Variante für Nolimit ist. Im PokerClient der Version 1.0 soll ein Feld der Klasse GameState (keine Erben) verwendet werden, um möglichst unaufwendig ein PokerBot erschaffen zu können. Ein einfaches Konzept ist es, den bisherigen GameState zum GameStateLimit zu refactoren. Es wird nun für Version 1.0 eine neue abstrakte Klasse mit dem Namen GameState geschaffen, die aus der maximalen Schnittmenge gleicher Methoden und Felder aus den Klassen GameStateLimit und GameStateNolimit besteht. Während gleichnamige Methoden verschiedenes Verhalten als abstrakte Methoden eingetragen sind. Die Methoden allAllIn, isFinished, isShowDown, allChecked und allCalled wurden von GameStateNolimit in den abstrakten GameState übernommen. Diese Methoden vereinfachen die Erzeugungs- und Steuerungslogik und geben dem Botentwickler einige nützliche Hilfsmethoden. In jedem Fall wird die Lesbar- und Wartbarkeit des Frameworks verbessert. Hierzu wird eine Vereinfachung für eine komplizierte Stelle in der Steuerungs- und Erzeugungslogik im GameStateLimit demonstriert:

```
if (((!preflop || (maxBetsizeThisStreet != FixValues.BIG_BLIND_VALUE))  &&
        (getPositionToAct() == lastRaisePosition)) ||
        ((preflop && (maxBetsizeThisStreet == FixValues.BIG_BLIND_VALUE)) &&
        (handeledActionsThisStreet >= (players.size() - 1)))) {
                positionToAct = Position.INVALID;
}
```

Die GameState-Klasse erhält Methoden der Klasse GameStateNolimit. Womit sich gezeigte die CodeStelle zu folgendem vereinfacht:

```
if (this.allPlayersChecked()){
        positionToAct = Position.INVALID;
}

if( this.allPlayersCalled()){
        positionToAct = Position.INVALID;
}

if (this.isShowDown()) {
        positionToAct = Position.INVALID;
        currentStreet = Street.SHOWDOWN;
}
```

Die in Nolimit 2014 beschriebene Codierung der Action hat auch den Weg in den abstrakten GameState gefunden. Ebenso wird die Klasse Player von Nolimit 2014 übernommen. Die Klassen GameStateLimit und GameStateNolimit sind jetzt Erben von GameState.

Die GameStatefactory erzeugt nun GameStateLimit oder GameStateNolimit. Hierzu wird der Klassenpfad vom Konstruktor des GameState an die GameStateFactory übermittelt, welche dann weiterhin den größten Teil des Spielzustands

erzeugt. Die Methode parsebettingString(), welche aus den Strings der einzelnen Streets Actions parst, wurde als abstrakte Methode an den GameState ausgelagert. Diese Auslagerung passt in das Schema, da bereits die Analysemethoden, welche die Actions in chronologischer Reihenfolge auswerten und den Spielzustand ändern, im GameState enthalten sind. Die GameStateFactory wendet nun das Abstract-Factory Pattern an. (Gamma u. a., 1995, S.87ff)

Die serverseitigen Konzepte der Version 1.0

Vor einem Spielstart vergleicht der Server, ob alle AccountLoginMessages, den gleichen GameState-Erben und Spieltyp enthalten. War das nicht der Fall, dann bricht der Server den Start des Matches ab. Vor dem Spielstart stellt der Server seine eigene GameStateFactory-Instanz auf das gewünschte Produkt ein. Ganz analog wird die GameFactory auf den zu spielenden Pokertyp eingestellt. Analog zu den GameStates gibt es Erben von der jetzt abstrakten Klasse Game mit den Namen GameTexasHoldEmLimit und GameTexasHoldEmNolimit. Letztere stammt aus der Version Nolimit 2014 mit der Erweiterung, dass falsche Spielzüge dem Spieler als call ausgelegt werden. Damit verhält sich das Game wie der Server der ACPC.

5.2.2 Wichtige Implementierungsdetails der Framework-Vereinigung

In den abstrakten Klassen PokerClientLimit und PokerClientNolimit sind die benötigten Klassennamen für die GameStateFactory und der GameFactory als Klassenpfad in String-Repräsentation hinterlegt mit Sichtbarkeit protected und Modifier final. Die Namen der Felder sind GameStateType und GameType. Der Konstruktor der PokerClient-Erben ruft den Konstruktor des PokerClient auf, der wiederum die statische Methode setGameRuleObject(String gameState) aufruft und in das entsprechende statische Feld speichert. Bei der Instanziierung eines Spielzustandes wird die Reflection Class.forname(String name) benutzt um den passenden GameState zu schaffen.

Für das Starten eines Pokerbots wird die Klasse ClientRunner beauftragt. Er interpretiert die Startparameter und wählt die gewünschte Kommunikationsschnittstelle aus. Dann werden der PokerClient und die Kommunikationsschnittstelle verbunden. Da der AlbertaPokerClient und der TUDPokerClient beide die statische Methode parseGameStateString der GameStateFactory nutzen, um GameStates zu instanziieren, sind Einstellungen der GameStateFactory schon alleine durch die Existenz eines PokerClient korrekt gesetzt.

Wird der TUDPokerClient genutzt, wird (sofort nach dem Verbindungsaufbau durch den ClientMessageClient der im TUD-PokerClient und PokerClient gekapselt ist und das Clientterminal des Nachrichtenzustellsystem ist), die AccountLoginMessage gesendet. Der TUDPokerClient und der ClientMessageClient haben jeweils ein Attribut, das den PokerClient referenziert und können somit die Klassenpfade für den GameState und den GameType in der AccountLoginMessage übermitteln.

Der Server nimmt AccountLoginMessages an und produziert (wie in den Vorgängerversionen) Accounts, entnimmt die Informationen über den GameStateType und GameType, die jeweils in einer Liste gesammelt werden. Kommt es zum Startbefehl für ein neues Game, dann wird zuerst die Methode checkBotTypes() gestartet, welche die Listenposten der GameState-Pfade auf Gleichheit prüft und bei einem positiven Ergebnis die serverseitige GameStateFactory einstellt. Analog dazu wird mit der Liste der GameTypes verfahren, jedoch mit der Methode checkGameTypes() für die GameFactory. Die Erben der Klasse Game unterscheiden sich durch die enthaltenen GameState-Erben, damit ist ein Teil des unterschiedlichen Verhaltens von Limit und Nolimit dorthin ausgelagert. Insbesondere differiert die Methodt handleAction(Account, String) zwischen den verschieden Game-Erben, weil dort die Kontrollabläufe unterschiedlich sind. Beispielsweise werden beim GameNolimit bezüglich des All-Ins zusätzliche Methoden gebraucht, wie checkNoLimitAllPlayerAllin() und dealShowdown(). Die Game-Klassen wurden passend zu ihren korrespondierenden GameState-Klassen refactored.

5.2.3 Verbesserung durch die Vereinigung beider Texas Hold Em Varianten in Version 1.0

Durch die Erben der Klasse PokerClient sind bei der verbesserten Version 1.0 auf den Spieltyp zugeschnittene Sende-Methoden umgesetzt worden, um älteren Pokerbots einen einfachen Umstieg zu erlauben. Die Min-Bet Regel des Nolimit Spiels wird nun von der Sende-Methode geprüft. Diese hebt zu kleine bet oder raises auf die Min-bet an. Durch dieses Vorgehen ist der Pokerbot besser vor unnötigen calls bei illegalen Spielzügen geschützt, trotz Nachlässigkeiten der Pokerbotentwickler.

Ein Pokerbotentwickler braucht nichts weiter zu machen als seinen Pokerbot vom passenden PokerClient erben zu lassen. Als einzige abstrakte Methode verbleibt handleGameStateChange(). Alles andere wird vom TUD-Pokerframework getan.

Der Server prüft selbstständig vor dem Start und bricht den Startvorgang eines Matches mit einer Fehlermeldung ab, wenn PokerClientLimit- und PokerClientNolimit-Erben zusammen spielen sollen. Zusätzlich sind mehrere Dutzend Codestellen ausgebessert worden, die der Compiler mit Warnungen versehen hat, wegen der alten Java-Version in der

das Framework programmiert wurde. Das waren oftmals nicht korrekte geschlossene I/O Schnittstellen zum Netzwerk oder zu Dateien aus der Zeit vor Java 7.

Das TUD-Pokerframework beherrscht mit Version 1.0 nun Texas Hold Em Nolimit Ring Spiele. Zuvor war in der Version Nolimit 2014 nur Texas Hold Em Heads up möglich. Der fehlende Sidepot ist derzeit kein Nachteil, weil nach den Regeln der ACPC gespielt wird und nach jeder Hand der Stack für alle Spieler wieder gleich voll ist (Doyle's Game).

5.3 Die Einbindung von Kuhn-Poker in die Version 1.1

Das neue Wettbewerbsformat der ACPC ist alleine Grund genug gewesen eine Frameworkerweiterung zu starten. Die Voraussetzungen wurden bereits in Version 1.0 geschaffen, die es erlauben passende Zwischenschichten in der Vererbungshierarchie einfügen.

5.3.1 Konzept für die Einbindung von Kuhn-Poker

Das Grundkonzept ist einfach und versucht sich möglichst an die Texas Hold Em Varianten anzunähern. Es wird naiv versucht, soviel wie möglich aus dem bereits vorhandenen zu übernehmen.

- Die PokerClient Klasse erhält einen neuen abstrakten Erben namens PokerClientKuhn, der im Prinzip identisch ist mit dem PokerClientLimit. Die einzigen Unterschiede sind die Klassenpfade zu GameKuhn und GameStateKuhn sowie die Verwendung von GameStateKuhn.
- Die Klasse GameStateKuhn ist Erbe von GameState. Die Idee ist, den Kontrollfluss während der Erzeugung anzupassen, wodurch eine neue GameState-Analyse nötig wird. Dabei muss darauf geachtet werden, dass es nur eine Street gibt und jeder Spieler nur eine Holecard bekommt und das jeder Spieler die Ante zahlt. Commoncards brauchen nicht beachtet zu werden. Der erste Spieler am Zug ist links vom Dealer.
- Die GameStatefactory braucht keine Änderungen, da das Format von Texas Hold Em Limit kompatibel zu Kuhn-Poker ist.
- Das Nachrichtenzustellsystem erfährt keine Änderungen zur Version 1.0
- Der Server arbeitet konform nach Version 1.0 und trägt Verantwortung für den korrekten Start eines Games.
- Die GameFactory erhält ein zusätzliches Produkt, das GameKuhn.
- Die Klasse GameKuhn ist ähnlich schwierig umzusetzen wie GameStateKuhn. Die Liste der wichtigsten Verantwortlichkeiten der Klasse GameKuhn:
  - Die Klasse GameKuhn erzeugt die Holecard nach den Regeln des Kuhn-Pokers für jeden Spieler, was ein eingeschränktes Kartendeck bedeutet. Ein Hilfsobjekt names CardShuffler schafft Abhilfe, indem es verschiedene Kombinationen von Holecards und Commoncards generieren kann mit möglichen Einschränkungen des Kartendecks. Vor dieser Version wird die Anzahl von zu spielenden Händen eines Matches im Game eingestellt. Beim Match-Beginn werden die Holecards und Commoncards jeder Hand festgelegt durch den Anfangsabschnitt der Methode run() von Game und im Speicher gehalten. Der Cardshuffler soll instant Karten generieren.
  - Die Spielsteuerung für beendete Hände ist anders wie die von Texas Hold Em.
  - Um möglichst viele Methoden aus GameState und Game übernehmen zu können, müssen Steuerungssignale äquivalent sein.

5.3.2 Probleme während der Implementierung von Kuhn-Poker

Die Klassen GameStateKuhn und GameKuhn wurden nicht fertig implementiert aufgrund ihrer Erbschaft von GameState beziehungsweise Game. Das Erben ist zwar unproblematisch für die Texas Hold Em Varianten, aber hat sich als relativ ungeeignet für andere Pokervarianten herausgestellt.

Die Klasse GameState vererbt eine Vielzahl von Methoden, welche die Existenz von Commoncards vorraussetzen. Ebenso sind viele Methoden auf den Umgang mit verschiedenen Bietrunden ausgelegt. Lässt man diese Methoden bestehen, werden dem Pokerbot-Entwickler viele Möglichkeiten geboten, unpassende oder funktionslose Methoden zu benutzen,

was von keinem guten Design zeugt und Fehlerquellen erröffnet. Es fehlt eine Metrik zur Auswertung der Kuhn-Poker-Hände, den outcomes Berechnung ist designed worden um Showdowns mit Kombinationen aus 5 Commoncards und 2 Holecards für jeden Spieler zu berechnen.

Um den GameStateKuhn umsetzen zu können, hätten in der Klasse GameState mehr Methoden abstrakt werden müssen und viele Methoden, die Commoncards erfordert hätten verbannt werden müssen. Genauso hätten Methoden, die von Streets abhängen auch aus dem GameState verbannt werden müssen. Andernfalls liefern solche Methoden nur Ergebnisse, die von einer einzigen existenten Street abhängen, was schlechtes Framework-Design ist, wenn jegliches Argument erlaubt ist und das Ergebnis gleich ist. Die Veränderungen würden mit der Hauptbedingung aus Kapitel 1.3 kollidieren - zur Verdeutlichung soll die Abbildung 5.1 einen Vergleich des GameState aus der Abgabeversion (Version 1.8) mit einem GameState, der abstrakt genug für Kuhn-Poker ist, liefern.

Um den Game-Erben GameKuhn implementieren zu können, ist ein funktionierender GameStateKuhn unerläßlich. Das Game trifft Entscheidungen für eine neue Street, Showdown, nächster Spieler und neue Hand durch einige Enumerations der Typen Street und Position, die in entsprechenden Feldern des GameState Steuerungswirkung auf das Game und die PokerClients haben. Durch die Mischung von Spieldaten und Steuerungsdaten müssen diese Felder präzise mit Werten belegt werden, sonst gibt der Dealer keine Commoncard oder gibt diese zu früh. Beispiel Enumeration Street mit seinen Einträgen:

- Preflop
- Flop
- Turn
- river
- Showdown
- Invalid

„Showdown“ und „Invalid“ sind keine Texas Hold Em Streets. Das sind reine Steuerdaten, die nicht für die PokerClients gedacht sind, aber sie dennoch auch von den PokerClients berechnet, wegen der gleichen Datenrepräsentation für Game und PokerClients, welche die GameStateFactory produziert. In welchen Situationen „Invalid“ oder „Showdown“ gesetzt werden soll, ist nicht dokumentiert. Aufschlussreiche JUnit-Tests sind in Limit 2010 nicht enthalten. Enumeration Position ist ein ähnlicher Fall, dort ist auch ein Eintrag namens Invalid enthalten. Dort stellt sich die gleiche Frage, wann muss „Invalid“ verwendet werden. Wie wirken sich Kombinationen beiden Enumerations zusammen mit weiteren Steuerungsdaten aus. Methoden, die nicht der Game-Steuerung, dienen, filtern die Steuerdaten oftmals heraus(falls diese in Felder geraten könnten, wo diese Werte nicht auftauchen dürfen oder durch die schiere Möglichkeit des Methodenaufrufs nicht enthalten sein sollten) und beenden den Prozess mit einer Runtimeexception. Das ist schlechtes Design und schwer nachzuvollziehen. Erschwerend kommt hinzu, dass für KuhnPoker eigentlich andere Streets und Positions gebraucht werden, weil die verwendeten Enumerations unpassend sind, weil es keine Blinds in Kuhn-Poker gibt. Leider werden diese Enumerations ausgiebig genutzt im GameState. Es hätte fast alles vom GameState wegabstrahiert werden müssen. Die Erben von Game haben keinen echten Einfluss auf den GameState, weil eine korrekte Abänderung eines GameState nur durch die Kombination aus GameStatefactory und den nachgeschalteten Analysemethoden möglich ist. Mit anderen Worten, hier offenbart sich eine Verschmelzung von Model und Controller, die äußerst unflexibel ist. Die Umsetzung von Kuhn-Poker hat nun eine niedrige Priorität bekommen, um andere Aufgaben anzugehen, auch wenn die Implementierung von GameKuhn und GameStateKuhn teilweise fertiggestellt ist.

Abbildung 5.1.: Klassenvergleich zwischen GameState der Version 1.8 und einer Variante für Kuhn-Poker. Die Abbildung soll lediglich vor Augen führen, wie wenig übrig bleiben würde. Es würde fast nichts mehr übrig bleiben, wenn auch noch die Positions bezogenen Felder und Methoden entfernt (bzw. abstrakt) würden

5.3.3 Ergebnisse aus dem Kuhn-Poker-Versuch

Die Vererbung von GameState und Game hat Grenzen. Damit können nur Texas Hold Em Varianten oder stark ähnliche Pokervarianten dem TUD-Pokerframework hinzugefügt werden. Es wurden Vereinfachungen an Server und Game vorgenommen, so dass eine verbesserte Version 1.0 entstanden ist. Praktikanten haben diese Version getestet und als schneller ausführend bewertet. Es wurde ein Kartengenerator geschaffen, der einstellbar ist bezüglich der Anzahl der Commoncards, der Anzahl von Holecards für jeden Spieler und der bei Bedarf nur eine Auswahl von Karten aus dem 52 Kartendeck zum Austeilen nutzt.

5.3.4 Weitere Ansätze und Überlegungen

Es zeigt sich zunehmend, dass die Unabänderlichkeit bis zur Neugenerierung des GameState ein Problem darstellt. Derzeit sind drei Controllerfragmente aktiv. Das ist der PokerClient-Erbe, der den GameState auslesen muss, ob die Cap- bzw. Min-Bet Regel eingehalten wurde. Der zweite ist das Game, welches auch den GameState mit Updates versorgen muss, aber dies nicht vollständig korrekt erledigt. Nur die GameStateFactory, die aber nur während der Instanziierung aus der String-Repräsenation aktiv wird, kann immer gültige GameStates erschaffen. Also ist es erstrebenswert einen Controller für den Dealer beziehungsweise PokerClient zu entwerfen, der GameState-Änderungen, Spielregelüberprüfung und Teile der jetzigen „Analysemethoden“ in sich aufnimmt und gegebenenfalls an Subcontroller deligiert, um sich den Umweg über die GameStateFactory zu ersparen.

Der GameState muss flexibler werden, um andere Pokervarianten ohne größeren Aufwand umzusetzen, im speziellen wird die Austauschbarkeit der Enums Position und Street für verschiedene Pokertypen gebraucht.

Das direkte Erben von PokerClient, Game und GameState wird für jede neue Pokervariante mindestens drei neue Klassen mit Erweiterungen in anderen Klassen bedeuten. Viele Klassen machen viel Wartungsaufwand.

5.4 Überarbeitung und Vereinfachung des Netzwerkverkehrs

Hintergrund dieser Überarbeitung, die zur Version 1.2 führt, ist die Beobachtung von zu vielen versendeten Nachrichten an die PokerClients. Jede SetGameStateMessage bedeutet, den kompletten Vorgang für die Erschaffung eines GameStates aus einem String durchzuführen. Diese Operation ist relativ aufwendig gestaltet. Es wird gescant, in Tokens zerlegt und jede Menge Unterobjekte geschaffen. Immer wenn eine Nachricht einen PokerClient erreicht, unterbricht dieser begonnene Operationen, um beispielsweise ein GameState-Update durchzuführen. Unter Umständen kann dies zu inkonsistenten Zuständen im PokerClient führen. Im Weiteren soll geprüft werden, ob auch eine Kommunikation ohne eine TCP Verbindung möglich ist.

5.4.1 Konzept der Überarbeitung des Netzwerksverkehrs

Die Überarbeitung des Netzwerkverkehrs ist nach erfolgter Analyse des Problems relativ einfach. Die Klassen MultiplePokerClientManager und TUDPokerClient werden entfernt. Der ClientRunner verbindet, beim Einsatz des TUD-Pokerservers, mittels loser Kopplung über das Observer Pattern, nun direkt ClientMessageClient und PokerClient gegenseitig als Oberserver und Observable ohne Umwege. Server und Game senden jetzt zu den PokerClients keine UpdateClientMessages mehr, die vorher von der Zwischenschicht TUDPokerClient verarbeitet werden sollten. Damit gibt es keine redundanten Nachrichten mehr am PokerClient und es gibt eine Zwischenstufe weniger, die für einen Pokerbot keine Vorteile bot. Der Preis ist, dass die GUI von Max Bank nicht mehr verwendet werden kann.

5.4.2 Gründe für die Entfernung von MultiplePokerClientManager, TUDPokerClient und UpdateClientMessage

Die Problematik, die der Einsatz des MultiplePokerClientManager nach sich zieht, ist kurz umrissen die Verwendung eines einzelnen Observers, der alllen TUDPokerClients gleichzeitig ein Update eines GameStates zukommen lässt. Der MultiplePokerClientManager ist durch das Singleton Designpattern (Gamma u. a., 1995, S.127ff) genau einmal vorhanden. Das heißt, es werden alle PokerClients an seinem Observer registriert und alle TUDPokerClients observieren den MultiplePokerClientManager. Folgende Probleme entstehen daraus:

1. Es können durch die UpdateClientMessages, die auch an alle TUDPokerClients weitergegeben werden, Accountdaten inklusive UID des Nachrichtenzustellsystems ankommen. Somit könnte sich ein Pokerbot als ein anderer Pokerbot ausgeben. Diese UpdateClientMessages waren eigentlich nur für sogenannte aktive Clients gedacht, die rudimentäre Steuerungsmöglichkeiten haben sollen um beispielsweise mittels der GUI von Max Bank, einen Server zum Start eines Matches zu veranlassen und dessen Zustand einsehen zu können.

2. PokerClients könnten Nachrichten erhalten, die nicht für diese gedacht waren, weil die Accountdaten bekannt sind. Somit können prinzipiell auch die Holecards der Villians ausgelesen werden.

3. Durch das gemeinsame Nutzen des Nachrichtenzustellsystem für Serversteuerung und Serverbeobachtungsdaten kann jeder PokerClient auch als passiver PokerClient Steuerungsnachrichten senden, weil beim Eingang einer Steuerungsnachricht keine Überprüfung stattfindet, ob der Client berechtigt war.

4. MultiplePokerClient ist nur Halfplex, er behandelt nur Nachrichten vom Server kommend. Die PokerClients haben über den ClientMessageClient eine zweite vollwertige Verbindung zu Server. Das heißt der MultiplePokerClient-Manager leitet eine Nachricht weiter, weil ihm der Ziel-PokerClient bekannt ist und der ClientMessageClient erhält ebenso die Nachricht vom Server durch das Nachrichtenzustellsystem. Auf dem Rückweg gehen Nachrichten vom PokerClient direkt über den ClientMessageClient. Der MultiplePokerClientManager kann nicht korrigierend eingreifen, wenn ein PokerClient die HashMap mit allen Verbindungen abfragt und nicht mehr seine eigene nutzt.

5.4.3 Implementierte Maßnahmen zur Verbesserung des Netzwerkverkehrs

- Entfernen des MultiplePokerClientManagers
- keine UpdateGameMessage versenden
- keine StartGameMessage mehr annehmen
- Der ClientRunner nimmt eine PokerClient-Instanz entgegen und startet eine Kommunikationsschnittstellen-Instanz. Beide observieren sich mittels Observer-Pattern gegenseitig. Damit brauchen sich der Pokerbot und die Kommukationsschnittstelle nicht mehr zu kennen, sobald der ClientRunner auch die Klassenpfade für die AccountLoginMessage weitergegeben hat.

5.4.4 Folgen der Netzwerkumgestalltung

Für Texas Hold Em Limit steht keine GUI mehr zur Verfügung. Ein Update der GUI stand alleine schon wegen Texas Hold Em Nolimit an. Es wird eine neue Möglichkeit gebraucht, Matches zu beobachten und den Server zu administrieren. Bei dieser Gelegenheit können Steuerungsdaten und Beobachtungsdaten vom Netzwerkverkehr der Pokerbots getrennt werden.

5.4.5 Ausblick bezüglich des Netzwerkverkehrs

Ein alternatives AccountLogin-Verfahren, welches beispielsweise auf UTF Strings basiert, würde es ermöglichen auch andere Programmiersprachen als JAVA zu verwenden, wenn man den TUD-Server nutzen möchte. Ebenso könnte auch mit den ActionMessages und SetGameStateMessages verfahren werden. Zurzeit ist dies nicht möglich, weil die Nutzlast der Nachrichten in einen Erben der Klasse Message eingelegt wird und vor dem Senden serialisiert wird.

Derzeit ist die Kommunikation nur in geschützten LANs sicher, da keinerlei Sicherheitskonzepte enthalten sind. Ein erster Schritt zur Absicherung wäre eine einfache Publik-Key-Infrastruktur zwischen Server und PokerClients zur Verschlüsselung von Nachrichten und auch zur gegenseitigen Authentifizierung.

5.5 Effizientere GameState-Updates

Der Dealer übermittelt nach jeder Action jedem PokerClient einen für diesen gültigen Spielzustand. Die Verarbeitung von Actions der PokerClients ist sehr aufwendig. Folgenden Vorgang gilt es in Version 1.4 zu vereinfachen:

1. Ein ActionMessage kommt beim Game in der Methode handleAction(Account, String) an und der alte GameState wird gecloned. Wird der GameState nicht gecloned funktioniert der Updatevorgang nicht.

2. Dann wird die Methode currentPlayerMakesAction(String) der GameState-Klasse benutzt, welche die Spielregeln prüft und die Argumente der nächsten Stufe vorbereitet. Es wird nun handleAction(...) der Klasse GameState aufgerufen.

3. handleAction(...) kommt auch bei der Instanzierung von GameState innerhalb der GameState-Analyse zum Einsatz. Die ankommende Action wird durch handleAction() in den GameState eingepflegt.

4. Danach ruft das Game die Methode updateGameStatesAndSendUpdates() auf, welche den GameState in seine String-Repräsentation umwandelt, um danach die GameStateFactory zu bemühen, um aus dem String ein völlig neues, korrektes GameState Objekt zu schaffen. Aus einem String einen GameState zu erzeugen, bedeutet neben dem Parsen des Strings auch alle Actions nochmals nachzuspielen, um einen gültigen Spielzustand zu erhalten. Danach wird der GameState in einen individuellen GameState in String-Repräsentation umgewandelt, da ein PokerClient Informationen über seine Position braucht und nur seine eigenen Holecards kennen soll, welche dann an den jeweiligen Account versendet werden. Der ClientMessageClient eines PokerClients läßt den GameState-String wieder zu GameStates umarbeiten.

5. Im Nachgang wird geprüft, ob eine weitere Karten gedealt werden sollen und im Fall des Dealens durch die Methoden dealNextStreet() und updateGameStatesAndSendUpdates() aufgerufen, die weitere Karten den Commoncards hinzufügt beziehnungsweise das in 4. Beschriebene nochmals durchführen.

6. Die zu sendenden Strings werden letzlich vom ServerMessageClient SetGameStateMessage eingesetzt und beim Senden serialisiert.

Dieser Prozess stellt kein Optimum dar. Mit der geltenden Anweisung, das Framework möglichst voll kompatibel zu vorherigen Versionen zu halten und die Struktur beizubehalten sind dennoch einige Optimierungen möglich.

5.5.1 Konzepte zur Optimierung der GameState-Updates

Das Vorgehen von Version 1.3 für die GameState-Updates:

1. Reduktion der SetGameStateNachrichten-Anzahl. Derzeit konnte eine Action mehrere Nachrichten auslösen, genau immer dann, wenn eine Street zuende geht. Die Methode sendUpdate() wird nur einmal am Ende von handleAction() ausgeführt. Es wird einmal das Parsen und Generieren eines GameStates eingespart, sowie das Erzeugen und Senden von SetGameStateMessage für alle Spieler.

2. Innerhalb des Games werden GameState nur noch neu generiert, falls eine Street zu Ende geht.

3. In die SetGameStateMessage werden anstatt der String-Repräsentation von GameStates nun einfach die GameStates direkt übertragen. Damit wird die Verantwortung für das Erzeugen der Spielzustände vollständig an den Dealer abgegeben, somit gibt es nur noch eine Stelle, die hierfür zuständig ist. Das Serialisieren von SetGameState-Messages mit kompletten GameState-Objekten, hat den gleichen Aufwand, wie GameStates in String-Repräsention zu serialisieren. Somit kann der Zwischenschritt, einen GameState zu einem String umzuformen, um ihn nach dem Deserialisieren wieder in ein Objekt zu übersetzen, eingespart werden. Die Serialisierungsschnittstelle ist darauf ausgelegt, aus Java-Objekten einen Text zu machen und umgekehrt, sie arbeitet schneller als die Serialisierung durch die Methode toGameStatestring() und die Deserialisierung durch die GameStateFactory.

5.5.2 Implementierung der optimierten GameState-Updates

Es werden Implementierungsdetails von Version 1.3 vorgestellt. Die Reduktion der Nachrichtenmenge konnte durch das Platzieren der Methode sendUpdates() an das Ende der handleAction() Methode in der Klasse Game erreicht werden. Alle anderen Aufrufe von updateGameStatesAndSendUpdates() wurden aus den Methoden handleAction() und dealNextStreet() verbannt; damit ist die Menge gesendeter Nachrichten reduziert auf genau eine pro action und Spieler. Als Ersatz für updateGameStatesAndSendUpdates() in der Methode dealNextStreet() wird der GameState über die GameStateFactory neuerzeugt. Der Vorteil ist, dass nur noch im Bedarfsfall ein neuer GameState angefordert wird.

Das Senden der serialisierten GameStates funktioniert. Leider ist es nicht möglich; gültige GameStates ohne den Umweg über die GameStatefactory zu gehen. Folgender Versuch wurde unternommen:

- Es wird unmittelbar vor dem Senden der SetGameStateMessage für Spieler ein Clone des vorliegenden GameState erstellt.

- Es wird der erste Spieler wird für die Versendung ausgewählt.

- Es werden alle Holecards im Clone des GameState gelöscht bis auf die des Nachrichtenempfängers.

- Die Nachricht wird mit dem GameState-Clone versendet.

- Die Löschung der Holecards im Clone wird rückgängig gemacht.

- Die Sendeschleife schaltet einen Spieler weiter.
- Schritt 3 bis 5 werden wiederholt bis alle Spieler eine SetGameStateMessage erhalten haben.

Diese Lösung ist nicht ausreichend. Jedes Mal, wenn ein PokerClient eine Action tätigen darf, dann ist er Hero und gleichzeitig auf der ActingPosition. Das wird durch einen Vergleich ermittelt getHero().getPosition == actingPosition. Bei obigen Versuch wurde nicht bedacht, dass für alle Spieler-Updates immer der gleiche Spieler Hero ist, also denken alle PokerClients, sie wären Hero mit der gleichen Position wie der ActingPlayer (im Game auf dem Server). Das Feld „boolean isHero“ und „Position position“ der Klasse Player ist final. Den beschriebenen Versuch erfolgreich zu Ende zu führen, würde bedeuten, die Felder isHero und position verlieren den Modifier final. Beide Felder würden dann zusammen mit den Holecards umgeschrieben werden. Die Auswirkungen sind zur Zeit nicht bekannt. Nach jetzigem Stand wird für jeden Spieler eine neue GameState-Instanz durch GameStateFactory angelegt und versendet.

Durch Testläufe konnte ein Bug in der Klasse GameStateNolimit gefunden werden, so dass nun in der handleAction() Methode des Game kein Clon des GameStates benötigt wird, wie es in Kapitel 5.5 beschrieben ist.

5.5.3 Weiterführende Arbeit an den GameState-Updates

Die Untersuchung, ob in der Klasse Player bei den Feldern isHero und position, der Modifier final wegfallen darf, steht aus. Es müssen hierzu alle relevanten Ableger Projekte in Version 1.3 eingebunden werden und Testläufe gefahren werden, ob negative Auswirkungen vorhanden sind. Sollten die Testläufe ein zufriedenstellendes Ergebnis liefern, dann reicht eine weitere Methode im GameState aus, um die GameStateFactory für die Action-Updates des Games überflüssig zu machen. Es würde dann reichen, vor dem Senden, den HeroPlayer mit dem Zielaccount gleichzusetzen und alle Holecards ausser denen des Hero auszublenden.

5.6 Übernahme von Modulen aus anderen Arbeiten

Es gibt eine Reihe von Projekten, die wiederverwertbare Module enthalten könnten. Eine hinreichend große Menge von übertragenen Modulen wird zu Version 1.4 führen. Die inspizierten Module werden im Folgenden grob bezüglich ihrer Übernahmemodalitäten klassifiziert. Die Klassifizierung gilt für dieses Kapitel und wird durch die runden Klammern mit Klassennummer (#) dargestellt:

1. Module, die ohne größere Anpassungen für beide Texas Hold Em Varianten übernommen werden konnten.
2. Übernehmbar für Limit und Nolimit jedoch nur für Heads Up geeignet.
3. Lauffähig übernehmbares Modul für Limit und Nolimit, jedoch mit eingeschränkter Funktion für eine der Pokervarianten.
4. Solche, die mit Refactoring für beide Varianten übernommen werden können.
5. An eine Pokervariante gebundene Module, die keine Portierung zulassen, aber für einen Pokerspieltyp verwendet werden können.
6. Module, die an Pokervarianten gebunden sind und zusätzlich für den Einsatz in Version 1.X refactored werden müssen.
7. Nicht eingebundene Module.

Bei den an eine Pokervariante gebundenen Modulen, können prinzipiell durch tiefere Einarbeitung in die Themengebiete möglicherweise generalisierte Versionen geschaffen werden, die für beide Texas Hold Em Varianten benutzbar sind. Im Rahmen einer Bachelorthesis ist das ausführliche Durcharbeiten mehrerer Abschlussarbeiten anderer Studenten nicht leistbar.

5.6.1 Inspizierte Module

- Module aus Limit 2010:
  - UCT Simulation(3)Zopf (2010)

    Stand der Übernahme: übernommen

    Package: pokertud.utc.*

- MetricsCalculator(1)

  Die Klasse liefert AllinEquityZopf (2010), ImmediateHandRankZopf (2010), SevenCardHandRankZopf (2010) und EffectiveHandRankZopf (2010).

  Stand der Übernahme: übernommen

  Package: pokertud.metrics.divat.*

- Divat2Max(2)Thomas Hartmann (2009)

  Dient der Post-Match-Analyse für limit heads up Poker.

  Stand der Übernahme: übernommen

  Package: pokertud.metrics.divat.*

- Aus Nolimit 2013 konnten folgende Module in das TUD-Pokerframework Version 1.4 übernommen werden:
  - Das Opponentmodel(2) Tomasz Gasiorowski (2013)

    Stand der Übernahme: übernommen

    Package: pokertud.opponentmodel2P.*

  - Der verbesserte EV-Calculator(1)Tomasz Gasiorowski (2013)

    Stand der Übernahme: übernommen

    Package: pokertud.metrics.divat.ev.*

  - Der Converter(2)Tomasz Gasiorowski (2013)

    Stand der Übernahme: übernommen

    Package: pokertud.util.converterACPCtoPokertracker

  - HandTypeRanger und AllinEquityRanger(2)Tomasz Gasiorowski (2013)

    Packages: boardtypes, opponentmodel2P, pokertud.metrics.divat.ev.* werden zur Anwendung benötigt. Stand der Übernahme: übernommen

    Package: pokertud.metrics.opponentandboardtypeRanger.*

- boardtypes(1)Tomasz Gasiorowski (2013)

  Stand der Übernahme: übernommen

  Package: pokertud.cards.boardtypes.*

- Parser(2)Tomasz Gasiorowski (2013)

  Stand der Übernahme:

  Package: pokertud.cards.boardtypes.*

  Die regelbasierte Pokerbot-Steuerung(4)Tomasz Gasiorowski (2013)

  - Regelparser(1) mit Scaner, Tokenizer und Parser

    Package: as.rulebot.parser.*

  - Die Statementtypen(1) Package: as.rulebot.ast
  - Ein Regelinterpreter(4)

    Package: as.rulebot.*

  - Der Rulebot selbst, auch bekannt unter ArizonaStu oder KEmpfer, ist als Nolimit Pokerbot konzipiert.

  Stand der Übernahme: im Framework

  Package: as

- Die Bachelorthesis über Ensemblebots von Tobias ThielThiel (2013) ist noch nicht eingepflegt und benötigt noch Überarbeitung. Diese Arbeit würde das TUD-Pokerframework um folgende Module erweitern:
  - Einen Server für die Verwaltung von Ensemblebots(6)

    Stand der Übernahme: im Framework

    Package: pokertud.ensemble.server.*

- Es wird auf einem erweiterten Server aus dem pokertud.ext.* gegen die Ensemblebots gespielt.
- Einen Auswahlalgorithmus für die Wahl der besten Aktion(6)

  Stand der Übernahme: im Framework

  Package: pokertud.ensemble.server.*
- Einen erweiterten GameState(6)

  Stand der Übernahme: im Framework

  Package: pokertud.ensemble.ext.gamestate.*
- Einen PokerClient für Ensemblebots(6)

  Package: pokertud.ensemble.*
- Ein Loggingsystem für den EnsembleServer

  Stand der Übernahme: im Framework

  Package: pokertud.ensemble.server.*

Die Standalonevariante der Ensemblebots können mit der Version 1.8 spielen. Insgesamt sind Überarbeitungen am Netzwerk und am Package Pokertud.ext.GameState nötig, um diesen Bot in das Framework als Limit-Erweiterung zu integrieren.

- Der Bot FatTony(7) ist ein Limit-Bot von Johannes Dorn und kann mit dem TUD-Pokerframework 1.8 spielen. Er würde folgende Module zum TUD-PokerFramework beisteuern:
  - Ein Gegnermodell auf statistischer Basis für 2 Spieler.
  - Ein neuronales Netzwerk als Gegnermodell ebenfalls für 2 Spieler.
  - MonteCarlo Entscheidungsbaum für Heads Up auf Basis des AllInEquity des MetricsCalculator mit dem statistischen Model, nur Limit geeignet.
  - UCT Entscheidungsbaum, der das neuronale Netz nutzt, um die UCT-Simulation des TUD-Framework einzustellen, nur Limit Heads Up geeignet.
  - Einen Ranger, der das statistische Gegnermodell nutzt, um die Villian-Holecards einzuschränken. Das Ergebnis ist eine Vorhersage über den zu erwarteten Gewinn.
- Der Bot FatBetty(7) ist ein Limit-Bot von Victor Negoescu. Er kann zwar mit Version 1.8 spielen ist, aber nicht fehlerfrei. Während des Spielens wirft dieser Pokerbot permanent Exceptions. Die Korrektheit ist auf keinen Fall sichergestellt und wird nicht in das Framework übernommen.
- Theo Kischkas PokerbotsKischka (2014) haben die Tests für Legacybots bestanden und können mit der Version 1.8 bespielt werden. Im Rahmen dieser Thesis wurde Kischkas Bot nicht reviewed, weil noch an einem fortführenden Praktikum seiner Bachelorthesis gearbeitet wird.
- Weitere Bots die noch nicht betrachtet wurden, sind der CFR-Bot von Peter Glöckner, der DonBot und der OwnBot. Letztere beiden liegen nur als beschädigte Projekte vor. Der CFR-Bot wurde nicht aus dem SVN heruntergeladen als er zur Untersuchung anstand, weil fälschlicherweise angenommen wurde, dass der Download nicht funktioniert. Es hat sich im Nachhinein herausgestellt, dass dieser Pokerbot insgesamt 42GB groß ist und lange für den Download braucht.

5.6.2 Module aus dem Original TUD-Pokerframework

Der MetricsCalculator ist lauffähig gemacht worden, durch kleine Fixes am Dateiladevorgang.

Er kann vollständig für beide Varianten genutzt werden, da er nur die Klassen Card und Cards benötigt. Die UCT-Simulation ist im FrameWork enthalten und kann nach kleinem Refactoring für Nolimit mit Einschränkungen übernommen werden. Der UCT-Baum hat genau einen Ast für Spielzüge vom Typ raise. Da es beim Limit Texas Hold Em nur eine Betsize gibt, genügen drei Äste (fold, call und raise). Um diese Simulation sinnvoll für Nolimit nutzen zu können, ist das fixe Entscheidungstripel aufzuspalten in eine Liste, die reservierte Indizes für call und fold nutzt und in den weiteren Komponenten bet Actions verschiedener Values einlagert. Gegebenenfalls ist die UCT-Simulation für Nolimit mit einer weiteren Metrik, wie Pottodds oder einem Gegnermodell zu koppeln. Die Gewinnchancen hängen auch von den bet-Sizes ab, weshalb der Baum mehr Äste braucht. Oder anders formuliert kann die Frage gestellt werden: Lohnt es sich das Risiko, das mit verschiedenernVillian bet-Sizes verbunden ist, auszusetzen.

Es werden die RichieRich PokerClients, welche die AllInEquity in verschiedenen Einstellung nutzen im Framework mitgeliefert. Der Bot Check-Norris verwendet ein einzelnes DecisionTripel zu Demonstrationszwecken.

5.6.3 Module aus der Teilnahme an der ACPC 2013

Das Gegnermodel, die verbesserte Ev-Berechnung, der HandTypeRanger und der AllInEquityRanger wurden generalisiert und sind jetzt für Limit und Nolimit verfügbar. Das Gegnermodel ist nur für Heads Up ausgelegt und ist zusammen mit den Rangern noch relativ nahe am Texas Hold Em Limit. Durch das einsetzen der GameState-Klasse statt GameStateNolimit war das Refactoring nahezu abgeschlossen und diese Klassen sind nun eingepflegt.

Die Klasse Converter konnte mit leichten Änderungen übernommen werden. Die Änderungen befähigen ihn nun log-Dateien von Limit ACPC Matches auszulesen.

Die Regelbasierte-Steuerung des Pokerbots KEmpfer / ArizonaStu ist zwar eingepflegt, weil sie zur Zeit für nur für Nolimit Heads Up geeignet ist, liegt sie in einem Package abseits des pokertud.* . Die restlichen Bestandteile sind relativ frei von weiteren Anpassungen.

Der PokerClient für Nolimit KEmpfer / ArizonaStu ist nicht eingepflegt, da sein RegelInterpreter noch nicht generalisiert wurde. Er nutzt das Opponentmodel und auch die Ranger aus dem Praktikum.

Insgesamt verlief das Einbinden zufriedenstellend, weil recht nahe am Framework gearbeitet wurd.

5.6.4 Die Ensemblebots von Tobias Thiel

Die Portierbarkeitsfrage auf Nolimit ist nicht geklärt. Thiels Abschlußarbeit und der Pokerbot samt seiner Infrastruktur müssen eingehend studiert werden, um diese Frage zu klären. Insgesamt erscheint die Struktur nach begonnenem Studium als ziemlich komplex. Um die Ensemblebots fest in das TUD-Pokerframework eingliedern zu können, werden einige Änderungen nötig sein. Die Bots eines Ensemble sind gewöhnliche PokerClients.

EnsembleServer

In dem Projekt ist ein Server, an dem sich Ensemble-Teilnehmer verbinden, enthalten mit dem Klassennamen EnsembleServer. Er setzt genauso wie der eigentliche Spielserver, das Interface IServer um und ist ein Erbe von PokerClientWithReset(auch ein Erbe des PokerClients). Die Ensemble-Bots melden sich über das Nachrichtensystem anstatt beim Spielserver nun beim Ensembleserver an. Die Kommunikation von Ensemblebots und anderen Pokerbots nutzt die gleiche Infrastruktur, was elegant zu sein scheint. Der EnsembleServer tritt dabei als PokerClient auf und verhält sich wie jeder andere Pokerbot zum Spielserver und dem Dealer. Jedoch ist der EnsembleServer auch Erbe von PokerClientWithReset und zwar aus gutem Grund, denn für ein korrektes Duplicated-Match müssen PokerClients zurückgesetzt(etwaige Gegnermodelle oder Historien geleert) werden können. Im EnsembleServer wird der MultiplePokerClientManager verwendet, der eigentlich nicht gebraucht wird. Tobias Thiel warnt, im Quelltext und auch im Betrieb, keine aktiven PokerClients zu nutzen, weil diese nicht unterstützt wird. Also wird im Umkehrschluss auch kein MultiplePokerClientManager gebraucht. Statt des MultiplePokerClientManager könnte von diesem EnsembleServer an dieser Stelle auch die Methode sendSetGameStateMessage des ServerMessageClient verwenden, weil der ServerMessageClient ein Bestandteil des EnsembleServer ist. Im Unterschied zum Dealer im Spielserver würde EnsembleServer den gleichen GameState an alle Ensemblebots senden. Eine einfachere Alternative für den jetzigen Ensemblebot, wäre einen PokerClient mit einer Liste von zu instanziierenden PokerClient als Argument zu übergeben. Diese PokerClients würden gewrappt werden und der Wrapper jeweils als Thread gestartet werden. Die Wrapper wären in einer geeigneten Datenstruktur einzulagern und mittels Observer-Pattern mit dem EnsembleServer verbunden.

GameServer

Der GameServer der das Reset-Command versendet, konnte nicht gefunden werden. Möglicherweise ist er in einem Kompilat einer Bibliothek enthalten.

EnsembleDecider

Der EnsembleServer hat als einen wichtigen Bestandteil eine abstrakte Klasse namens EnsembleDecider, die einen weiteren abstrakten Erben namens EnsemblePokerClient hat, von dem die eigentlichen Entscheider erben. Der EnsembleServer nutzt diesen EnsembleDecider, um aus den vorgeschlagenen Spielzügen, der am EnsembleServer angemeldeten PokerClients, einen Spielzug auszuwählen. Die Entscheider sind alle für Texas Hold Em Limit ausgelegt und verfügen über keinerlei Methodik, die verschiedene bet-Sizes in Buckets einzuordnen, was ein nötiger Schritt für Nolimit ist.

GameStateExt und ImmutableGameState

Die Entscheider verwenden einen Erben namens GameStateExt des GameStates, der für das derzeitige TUD-Pokerframework unterhalb Version 2.0 in einer Limit und einer Nolimit Variante vorliegen muss. Denn in JAVA ist keine Mehrfachvererbung erlaubt. Desweiteren wird noch ein zweiter Erbe von GameState für statistische Zwecke eingesetzt. Die Instanzen dieses Erben werden in einem Modul eingesetzt, das die EnsembleBots nach einer Hand neu bewertet, um diesen mehr oder weniger Gewicht bei Abstimmungen zu geben. Wieder werden zwei Versionen dieses GameState gebraucht. Zusätzlich muss auch die Auswertung angepasst werden.

5.6.5 Pokerbot FatTony von Johannes Dorn

FatTony bringt, wie Thiels EnsembleBots, interessante Erweiterungen mit. Ob dieser Pokerbot auch auf Nolimit portierbar ist, ist noch nicht geklärt.

Gegnermodell auf statistischer Basis

Das Gegnermodell basiert auf Statistik und betrachtet typische Poker-Kennzahlen, die unabhägig von bet-Sizes sind. Somit ist es leicht für Nolimit und Limit Texas Hold Em anpassbar. Es ist nur für den Preflop gedacht. Leider ist es nur Heads Up geeignet und eine Umstellung auf Ring-Games wird wegen der Auswertungsschwierigkeiten des GameState ziemlich kompliziert, denn schon die Auswertung des Heads-Up ist schwierig zu verstehen. Für jede Action wird der Ausführende bestimmt und ob der GameState in einem auswertbarem Zustand ist. Immerhin liefert es solide Kennzahlen, die sich gut in Regeln verwenden lassen.

Neuronales Netzwerk als Gegnermodell

Ein Neuronales Netzwerk ist als Teil des Postflop-Gegnermodells enthalten. Es versucht eine Empfehlung für die erste Action des Heros am Flop zu machen, genau dann wenn der Flop noch keinen Spielzug gesehen hat. Eine Anpassung für Nolimit könnte relativ einfach sein, weil das neuronale Netzwerk Spielzüge in Zahlenwerte umsetzt; es könnte bereits weitestgehend passend für Nolimit Poker sein. Ob dieses neuronale Netzwerk auch bet-Sizes liefern könnte, muss noch geprüft werden. Sollte das der Fall sein, wäre endlich die Frage für Nolimit gelöst, wie ein Pokerbot relativ dynamisch bet-Sizes bestimmen kann.

PostFlopHandRangeEstimation auf Basis eines neuronalen Netzes

Der Ranger benutzt das gleiche neuronale Netzwerk wie die Vorhersage der ersten Action des Heros. Er wird für alle anderen Züge eingesetzt. Wegen des neuronalen Netzes scheint der Ranger auch gut für Nolimit-Poker anpassbar zu sein. Er verwendet eine Klasse namens BoardEvaluator, die dem package boardtype aus Nolimit 2013 ähnlich ist. Der Ranger wird für das Postflop-Gegnermodel eingesetzt.

Monte-Carlo mit Postflop-Gegnermodell

Die Klasse DecisionPredictionMCHeadsUp produziert Entscheidungstripel für Monte Carlo Bäume des TUD-Pokerframeworks. Das Postflop-Gegnermodell liefert statistische Werte, um die Monte-Carlo-Suche von Hand zu Hand anzupassen. Der eine Teil der statistischen Werte wird durch die Klasse PostFlopHandRangeEstimation geliefert. Funktioniert für Nolimit mit der Einschränkung, dass keine bet-size geliefert wird, sondern nur die Action.

UCT Simulation mit Postflop-Gegnermodell

Die Klasse DecisionPredictionUCTHeadsUp liefert Entscheidungstripel für die UCT-Simulation des TUD-Pokerframeworks. Das Postflop-Gegnermodell benutzt das neuronale Netzwerk für die erste Postflop-Action und benutzt ebenfalls die statistischen Werte des Postflop-Gegnermodells und die gleichen Einschränkungen für Nolimit wie die Monte-Carlo-Simulation.

5.6.6 Pokerbot FatBetty von Victor Negoescu

Dieser Bot verwendet die gleichen Heads-Up-Gegnermodelle wie FatTony. Die aus dem Gegnermodell erzeugten Entscheidungstripel werden jedoch für die Berechnung eines Nash-Gleichgewichts herangezogen.

In den Testläufen hat sich gezeigt, dass der Bot nicht gut funktioniert. Er wirft jede Menge Exceptions und ist relativ langsam. Sein Bruder FatTony funktioniert besser. Dabei hat sich der Bot-Entwickler ziemlich viel Mühe gegeben, die Probleme unter Kontrolle zu bekommen. Er ging soweit, dass für Betty ein GameStateWrapper entworfen wurde, der die Actions des gewrappen GameState nochmals als PlayerAction Listen speichert. PlayerAction macht zwar nichts weiter als einer Action den Spieler zuzuordnen, der sie tätigt, aber es erspart viele Zeilen Quellcode und Berechnungszeit, diese Berechnungen nicht zu wiederholen. Wenigstens erfordert der Wrapper für den GameState keine zweite Variante für Nolimit.

Das Preflop-Gegnermodell liegt hier auch in der Ausführung für Ring-Spiele vor.

5.6.7 Der Pokerbot von Theo Kischka

Es handelt sich hierbei um einen Pokerbot, der WEKA(Eine Software für Maschinelle-Lerner) verwendet. Er lernt die Spielweise anderer Poker-Spieler. Da die Arbeit an diesem Bot derzeit noch nicht abgeschlossen ist, werden keine Module von diesem ins TUD-Pokerframework übernommen. Die Ergebnisse seiner Review in Hinblick auf die zukünftige Entwicklung des TUD-Pokerframeworks werden nun angeschnitten.

Die Packages utils und dataStructures beinhalten diverse Konverter und Datenstrukturen, die den GameState in andere Repräsentationen bringen. Die Klasse ActionData ist ein gutes Beispiel. Sie trägt nur Daten, aber erleichtert die Arbeit des Bot-Entwicklers sehr stark.

```
package dataStrutures;

import pokertud.gamestate.Action;
import pokertud.gamestate.Player;
import pokertud.gamestate.Position;
import pokertud.gamestate.Street;

public class ActionData {
        public Action action;
        public String currentAction;
        public Street street;
        public Player player;

        public ActionData(Action myAction, Player myPlayer, String myCurrentAction,
            Street myStreet){
                action = myAction;
                player = myPlayer;
                currentAction = myCurrentAction;
                street = myStreet;
        }

        @Override
        public String toString(){
                return player.getPlayerName() + " " + action + " current action: " +
                    currentAction + " " + street;
        }
}
```

Die GameStates unterhalb Version 2.0 haben keine Konzentration von Daten in den Listen der Spielzüge. Es werden dort einfach nur in chronologischer Reihenfolge Action Enums beziehungsweise Integers, die Actions repräsentieren, in Listen gesammelt. Aus dem Action Enum ist nicht ersichtlich wer den Spielzug getätigt hat, es gibt keine zusätzlichen Informationen wie in ActionData im Feld currentAction. Action Enum kennt nur 3 Spielzüge:

- raise
- call

- fold

Es ist eine ausreichende Repräsentation für Steuerungen oder Datenübertragung. Actions haben mehrfache Bedeutungen. Die Action call kann je nach Spielsituation einen echten call repräsentieren oder auch ein check und bezogen auf Texas Hold Em sollte im Preflop zwischen limp und call unterschieden werden. Bei raise wird es noch schwieriger, denn der erste raise einer Street heißt bet, der zweite ist der eigentliche raise, der dritte ist der reraise und danach folgt der cap. Aber eigentlich müssten die Bezeichnungen zwischen Limit und Nolimit ab dem vierten raise unterschieden werden. Diese Bedeutungen müssen beim einfachen Enum Action oder in der Darstellung als Integer immer wieder berechnet werden, wenn diese gebraucht werden. Das macht Programme unübersichtlich und sperrig. Ein kleiner Vorgriff auf Kapitel 5.11: Die Entwicklung von Version 2.0 hat als Ausgangspunkt eine Klasse namens ActionObject mit ähnlichen Feldern in größerer Anzahl, die unabhängig von dieser Arbeit entstanden ist.

5.6.8 Erkenntnisse aus den Modul-Reviews

Die Review alter PokerBots und deren Modulen hat die Ansätze für das TUD-Pokerframework 2.0 bestätigt. Ebenso werden in Version 2.0 Erkenntnisse aus den Code-Reviews Beachtung finden. Insbesondere werden die Implementierungserfahrungen, die in Projekten enthalten sind, für die Umgestaltung von Kernmodulen benutzt. Das Einpflegen von alten Modulen wird durch das Design von Limit 2010 und allen Updates (Versionen 1.X) nicht gut unterstützt. Es gibt eine Vielzahl von Gründen, Version 2.0 voranzutreiben.

Es gilt die andauernde Implementierung ähnlicher Funktionalitäten in den verschiedenen Projekten einzudämmen. Denn ähnliche Funktionalität bedeutet nicht automatisch, dass diese mit minimalen Anpassungen in anderen bestehenden Projekten verwendbar sein wird. Im schlimmsten Fall sind ähnliche Funktionalitäten - abgesehen von kleinen semantischen Unterschieden - völlig anders designed und nicht zu ersetzen. Also darf eine Funktionalität nur einmal in einem Framework enthalten sein, sonst die Übersicht verloren geht. Dann stellt sich die Frage, welcher Kandidat der Beste ist oder ob ein neues Konstrukt für das TUD-Pokerframework mit Dokumentation, Tests und unter Berücksichtigung der Erfahrungen aus Vorgängerimplementierungen entstehen soll.

Die Datenmodellierung stellt ein weiteres Problem dar. GameState wird oftmals in andere Repräsentationen übersetzt. Nahezu jedes Projekt hat eine oder mehrere eigene Repräsentationen. Fast immer sind diese für ganz themenspezifische Anforderungen zugeschnitten worden. Auch wenn es eine gewisse Schnittmenge zwischen den Implementierungen gibt, wäre es unpassend, für ein Framework alle Varianten einzubinden. In den Unterkapiteln werden die verschiedenen Formen der Spielzustandsdarstellung diskutiert.

GameState-Erben

Eine einfache Möglichkeit dem GameState weitere Methoden oder anderes Verhalten zukommen zu lassen, ist die direkte Vererbung. Aus Sicht des Pokerbot-Entwicklers ist das sehr bequem, auf diese Art eine neue Funktionalität hinzuzufügen. Aus Sicht des Framework-Entwicklers ist es sehr unbequem, weil er dann die Aufgabe hat, den GameState Erben zurückzuführen auf GameState und sein spezielles Verhalten sowie Methoden zu einer eigenen Klasse auszulagern. Das gleich gilt, wenn alternativ versucht wird eine Erbschafts-Linie aufzubauen. Gerade bei einem Toplevel Modul wie dem GameState wirken sich Erben verheerend aus.

In Hinblick auf Nolimit und Limit müssten immer zwei Erben für einen projektbezogenen GameState geschaffen werden, weil der GameStateLimit und der GameStateNolimit vom abstrakten GameState erben. Werden die GameState-Erben der Projekte nicht korrekt eingepflegt, dann haben Bot-Entwickler nur eine praktikable Möglichkeit: es werden genauso viele spezielle parallele GameStates verwendet, wie Features vom TUD-Pokerframework eingesetzt werden sollen.

Eine andere besonders wartungsaufwändige Lösung wäre, dass alle Erbschaftskombinationen für das Framework geschaffen werden (die sich aus den Featurekombinationen ergeben könnten). Die letztgenannte Lösung wäre auch Hinblick der Verwendbarkeit besonders unpraktikabel, denn es können Methoden in den verschiedenen Vererbungsschichten mehrfach überschrieben sein. Ein Feature benötigt üblicherweise eine genau definierte Verhaltensweise einer Methode, auf die der Programmierer nur dann Zugriff hat, wenn entsprechende Hilfsmethoden in den Erben eingearbeitet wurden, um an die überschriebenen Methoden in der Vererbungshierarchie heranzukommen. Über mehrere Vererbungsschichten entsteht eine gewisse Hilfsmethodenmenge. Vererbung ist zu vermeiden, viele Klassen sorgen für großen Wartungs- und Pflegeaufwand.

Die Wrapper des GameState

Ein Wrapper (Gamma u. a., 1995, S.139ff) verpackt eine Instanz einer Klasse mit dem Ziel weitere Methoden, die auf der verpackten Klasse arbeiten, zur Verfügung zu stellen. Die Wrappermethoden sind seiteneffektfrei bezüglich des Wrappers. Die Seiteneffektfreiheit wird bei Anwendung der Wrappermethoden nicht für das verpackte Objekt gefordert. Diese

Methodik ist bequem für den Bot-Entwickler. Anders als bei der Vererbung ist es leichter, die benötigte Funktionalität für ein Modul oder eines Features auf einen GameState anzuwenden. Beispielsweise benötigen zwei Features zwei Spielzustandsrepräsentationen, dann würde die Lösung mit Wrappern folgendermaßen realisierbar sein: Es werden beide Wrapper für die veränderte Spielzustandsrepräsentation eines GameStates instanziiert, aber die Konstruktoren beider Wrapper bekommen dieselbe Referenz zu einem GameState übergeben; damit sind beide Wrapper immer gleich aktuell. Diese parallele Konstruktion hat eine Einschränkung: Die Wrappermethoden dürfen das verpackte Objekt nicht durch ein anderes austauschen.

Die Aufgabe des Framework-Entwicklers ist es hierbei, eine sinnvolle Packagestruktur aufzubauen. Die Wrapper und das dazugehörige Feature oder Modul eindeutig erkennbar im Framework anordnet. Wrapper stellen eine geschickte Möglichkeit dar, fehlende Funktionalität des GameStates auszugleichen. Sie sollten bei zukünftigen Projekten statt Vererbung und Decorator-Klassen angewendet werden.

Decorator-Klassen für den GameState

Das Decorator-Pattern (Gamma u. a., 1995, S.175ff) ist mit Wrapper-Pattern verwandt, aber mit dem entscheidenden Unterschied, dass auch zusätzliche Felder in der umgebenden Klasse enthalten sind. Der Einsatz von parallen Decorator-Klassen für eine zu wrappende Klasse hat eine wichtige Anforderung mehr als Wrapper-Klassen. Diese Anforderung lautet, tätigt ein Decorator eine Methode auf dem verpackten Objekt, dann müssen alle Decorator prüfen, ob ihr Zustand noch konform zum neuen Zustand des verpackten Objekts ist. Keiner der Decorator, welche in den Projekten eingesetzt werden, kann einen Test durchführen, ob der eigene Zustand noch konform mit dem GameState ist und Abweichungen in den eigenen Feldern korrigieren. Ein anderer Weg parallele Decorator aktuell zu halten, wäre der Einsatz von Observern durch den Decorator und der GameState wird ein Observable-Erbe. Leider ist den GameState der Versionen 1.X keine Vererbung von Observable vorgesehen. Zumal die derzeitigen nicht das Observer-Interface umsetzen. Damit sind GameState-Decorator auch eine schlechte Wahl für GameState-Erweiterungen, wenn die geschaffenen Module später in das Framework verankert werden sollen.

Von GameState unabhängige Spielzustandsrepräsentationen

Solche Konstrukte sind bequeme Konstrukte für einen Bot-Entwickler. Der GameState wird übersetzt in eine angenehmerere Repräsentation, die passend für das Feature oder das zu lösende Problem ist. Möglicherweise sind die Konstrukte zweckmäßig und effizient, aber im Einsatz mit anderen Features sind Nachteile zu Wrappern zu erkennen. Wrapper folgen einem Design-Pattern und sind eine bewährte Vorgehensweise. In den parallelen Wrappern wird immer das gleiche Objekt verpackt und erweitert, während die unabhängigen Repräsentationen redundante Daten produzieren. Zum Vergleich: Zwei parallele Wrapper tragen genau einen GameState in sich. Zwei unabhängige Repräsentationen formen jeweils den gleichen GameState um. Dabei wird mit großer Wahrscheinlichkeit die Datenmenge genau verdoppelt, die dann redundant ist. Es müssen dann zwingend immer zwei Datenmodelle mit Updates versorgt werden.

Der Aufwand für Framework-Entwickler ist bei der Einpflege derartiger Strukturen ähnlich hoch wie bei der Einpflege von Wrappern. Das Feature braucht, zusammen mit seinem Datenmodell und Übersetzer, eine verständliche Packagestruktur, um es als komplettes Feature in einem Featurebereich einzulagern.

Überlegungen für mehr Modul- und Featurerecycling

Aus den vorangegangenen Überlegungen geht hervor, dass von Vererbung abgesehen werden sollte. Die Wiederverwertbarkeit von Modulen und Features leidet darunter am stärksten. Zusätzliche Datenstrukturen sind zwar zugeschnitten auf die Anwendungen, aber ihre Existenz könnte schon ein Indiz für Unzulänglichkeiten im Framework sein, sonst würden diese nicht gebraucht werden. Das Verwenden von Wrappern ist die eleganteste Möglichkeit für eine Erweiterung. Sie bewirken keine Änderungen in der Vererbungshierarchie und verhindern redundante Daten, aber auch hier gilt: Je nach umgesetzter Funktionalität könnten im Datenmodell des Frameworks bereits die Probleme liegen.

Ein Teil der Konzepte für Version 2.0 wurden aus den Reviews alter Projekte übernommen. Konzepte für Version 2.0, die bereits vor einem Teil der Reviews entstanden sind, wurden durch die Reviews bestätigt. Beispielweise ist die Klasse ActionObject für Version 2.0 bereits vor der Review von Theo Kischkas Arbeit entstanden, welche die Klasse ActionData mit ähnlicher Funktion enthält.

Das TUD-Pokerframework braucht ein neues Datenmodell, das weniger Erweiterungen braucht und final ist. Es sollen keine weiteren Erben vom GameState entstehen. Zugleich soll das neue Datenmodell für eine möglichst große Anzahl von Pokervarianten ohne Spezialisierungen verwendbar sein. Zwar werden durch Vererbung relativ einfach mehrere Spieltypen umgesetzt, aber sie verlangt im Zweifel auch von den Modulen in mehreren Versionen vorzuliegen, wenn diese beispielsweise Nolimit und Limit tauglich sein sollen. Kontroller und Datenmodell sollten stärker voneinander

getrennt werden als vorher. Erst dann wird wirklich Modularisierung möglich und es besteht die Möglichkeit, mehr Modul-Recyling zu betreiben. Das bereits angefangene TUD-Pokerframework 2.0 ist genau auf diese Anforderungen zugeschnitten.

In jedem Fall sollten auch Schnittstellen/Interfaces zu üblichen Erweiterungen wie Metrik-Berechner, Gegnermodellen, Spielbäumen und Simulatoren erstellt werden. Die Implementierungen dieser Schnittstellen, werden austauschbar und somit modular. Was besonders wichtig ist: Die Module werden vergleichbar.

Generell ist stärker auf Kommentierung und JAVA-Docs wertzulegen, um den Verwendungszweck von Methoden und Klassen schneller erfassen zu können. JUnit-Tests sind auch nur selten enthalten. Diese sind nicht nur für den Nachweis von Korrektheit wichtig, sie sind auch unerlässlich bei der Modulpflege, weil damit schnell getestet werden kann, ob nach einem Refactoring noch alles funktioniert. Außerdem stellen JUnit-Tests ein weiterer Weg zu Codedokumentation dar.

5.7 Serversteuerungsschnittstelle und Serversteuerungs-GUI

Der ServerCommander ist als Servererweiterung konzipiert und schon teilweise implementiert. Es existiert keine Möglichkeit zur Serverüberwachung, Serverkonfiguration und Serversteuerung. Durch die Änderungen im Netzwerk-Stack (kein TUDPokerClient und MultiplePokerClientManager) gibt es derzeit keine Benutzerschnittstelle, um mit dem Server zu interagieren. Wären der TUDPokerClient und der MultiplePokerClientManager aktiv, würde mit der derzeitigen Serverimplementierung eine Sicherheitslücke entstehen. Jeder PokerClient hätte die Möglichkeit die neuen adminstrativen Werkzeuge zu nutzen oder zu missbrauchen. Der Server nimmt von jedem PokerClient Steuerungsbefehle entgegen. In diesem Kapitel wird gezeigt, warum eine mächtigere und maximal generalisierte ServerSteuerung gebraucht wird und wie ein Konzept hierfür gestaltet sein kann. Sobald dann der ServerCommander fertiggestellt wäre, wäre Version 1.5 abgeschlossen.

5.7.1 Die bereits bestehende GUI für die Serverkontrolle und der TUD-Pokerserver

Die bereits implementierte GUI Bank (2011) kann dem TUD-Pokerserver nur mit zwei echten Steuerungsbefehlen beeinflussen. Der erste Befehl ist der Spielstart mit einigen Parametern. Der zweite ist „quit“, welcher das Spiel per Knopfdruck abbrechen lässt. Der Spieler oder Beobachter versendet im laufenden Match eine disconnectMessage. Der Server bricht beim disconnect eines Teilnehmers das ganze Match direkt ab und schließt auch die Verbindung zu diesem Teilnehmer. Nach dem Verwenden von „quit“ wird eine Neuanmeldung zum Server fällig. Mit diesem Verhalten stellt sich „quit“ als kein brauchbarer Serversteuerungsbefehl dar. Spiele, die nicht von Menschen bespielt oder am Tisch beobachtet werden, lassen sich nicht abbrechen. Es sei denn: Ein Pokerbot meldet sich als aktiver Client an mit einer entsprechenden Implementierung, „quit“ anzuwenden. Spiele die beobachtet werden, sollen nicht zwangsweise abgebrochen werden, wenn der Beobachter beschließt, nicht mehr zusehen zu wollen.

An der Spieler-View (Pokertisch) sind weitere Einstellungen vorhanden, wie „pause on showdown", einen Slider für die Anzeigedauer eines Spielzuges, sowie die Einblendung von Statistiken oder der Log-Historie. Alle diese Befehle haben nur eine Wirkung auf die GUI. Jedoch fehlen wichtige Befehle an den Server und er läuft weitestgehend im Batch-Modus.

Zeitlimits

Unabhängig von speziellen Zeiteinstellungen bei Anwesenheit von menschlichen Spielern oder Beobachtern ist es nicht möglich, den Server bezüglich eines Zeitlimits für Spielzüge, ganze Hände oder gar ganze Matches einzustellen. Bei der ACPC gibt es solche Limits, die dringend bei Testspielen eingehalten werden müssen, denn die SpielregelComputer-Poker-Competition (2012a) lautet: ein Spieler hat beim Texas Hold Em genau 7 Sekunden Zeit multipliziert mit der Anzahl der Hände und ein einzelner Spielzug darf maximal 10 Minuten dauern. Für KuhnPoker ist die Matchdauer begrenzt auf 100 Hände pro Sekunde multipliziert mit der Anzahl der zu spielenden Hände. Diese Einstellungen sollten global voreinstellbar sein und auch für einzelne Clients angepasst werden können, um beispielsweise menschlichen Spielern unbegrenzt Zeit zu geben. Hierfür gibt es keine Serverbefehle und somit auch keine echten Testläufe für die ACPC.

Logging von Spieldaten

Es existieren zwei Logger für Spieldaten. Einer existiert in der GUI des Pokertisches. Seine Aufgabe ist es den Matchverlauf mit den Spielzügen zu protokollieren. Der bessere Ort hierfür ist der Server oder das Game, weil erstens solch ein Logg auch ohne die GUI geschrieben werden können soll, zweitens genügt es, wenn der Logger des Servers Textnachrichten an die GUI versendet und drittens braucht die GUI durch das Versenden der fertigen Textnachrichten weniger Logik.

Der andere Logger ist Teil des Nachrichtenzustellsystems. Er dient insbesondere zu Debuggingzwecken, um die Nachrichtenverläufe zu analysieren. Hiermit ist eine rudimentäre Timinguntersuchung möglich, weil die Sende-Time-Stamps in der Methode receive und das Weiterleiten von der Methode send jeweils mitprotokoliert werden. Jede Nachricht, die das Nachrichtenzustellsystem passiert, wird von zwei Message-Handlern behandelt und somit von beiden Message-Handler geloggt.

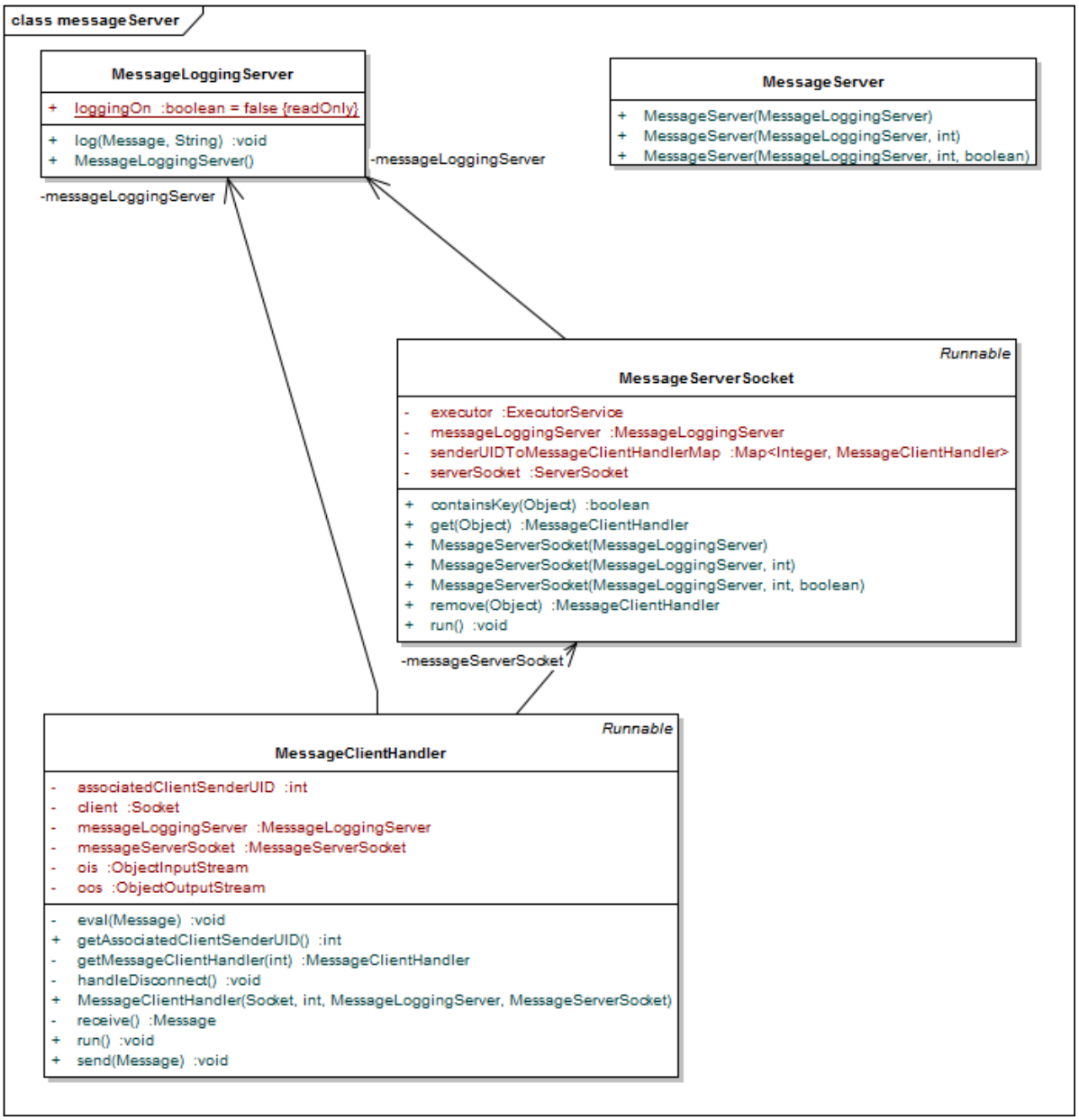

Abbildung 5.2.: Die Klasse MessageServer instanziiert ein MessageServerSocketObjekt und übergibt diesem den MessageLoggingServer. Der MessageServerSocket wartet in der run Methode auf eingehende Verbindungen. Diese werden durch den MessageClientHandler zu einer Session erhoben. Der Konstruktor des MessageClientHandlers erhält als Referenz den MessageLoggingServer. In der receive und send Methode des MessageClientHandler wird geloggt.

Der Logger ist am zentralen Knotenpunkt des Nachrichtenzustellsystems in der Lage, jegliche Nachricht zu protokollieren und ist somit an einer sinnvollen Stelle platziert. Jedoch fehlen einige wichtige Optionen. Das Loggen findet nur auf der Konsole statt und kann nur am Quellcode ein- und ausgeschaltet werden (MessageLoggingServer: boolean loggingOn). Wegen der Vielzahl von Nachrichten wäre es hilfreich, Zieldateien für das Nachrichtenloggen angeben zu können. Gegebenenfalls wären auch weitere kleine Erweiterungen wie Sender- oder Empfängerfilter hilfreich. Dies alles erfordert jedoch neue Serverbefehle.

Mit den Spieltypen Nolimit Texas Hold Em und in absehbarer Zukunft mindestens zusätzlich Kuhn-Poker wird es wichtiger, allgemeine Servereinstellungen vornehmen zu können. Die Lobby von Max Banks GUI ist in der Lage, das TUD-Pokerframework nach PokerClients zu durchsuchen und für ein bestimmtes Game zu starten. Diesen Vorgang gilt es zu erweitern, um die PokerClients nach Spieltyp zu sortieren. Schnittstellen zu Turniermodi könnten offengehalten werden, um beispielsweise die Formate der ACPC als Preset spielen zu können. Aber einfache Einstellungsmöglichkeiten, wie Stacks oder Big-Blind ändern zu können wären Optionen. Jedoch werden hierzu wiederum neue Serverbefehle gebraucht.

5.7.2 Grundkonzept des ServerCommanders

Er ist ein eigenständiger Serverthread, der Servermethoden aufrufen darf. Er kapselt einen eigenen Serversocket, um Wartungs- und Steuerungsverbindungen eingehen zu können ohne das Nachrichtenzustellsystem nutzen zu müssen. Die ausgetauschten Nachrichten sind ausschließlich Texte, die für Menschen verständlich sind. Mit diesem Vorgehen ist es möglich, diverse Clients zu verwenden. Es können Terminalprogramme wie netcat und telnet verwendet werden, aber auch ConsolenClients oder GUI-Clients. Die GUI-Clients sind einfach zu erstellen, weil das Bedienen von Steuerelementen nur die Texte zu generieren braucht, die man sonst per Hand eingeben würde. Alle textuellen Befehle müssen nicht unbedingt nicht von den Clients geprüft werden, denn das erledigt bereits der ServerCommander. Ein Vorteil, der sich aus der textuellen Befehlsentgegennahme ergibt, ist das Vermeiden einer Vielzahl von Message-Erben. Die bisherigen Steuerungsnachrichten waren jeweils ein Erbe von Message, was eine Fallunterscheidungen der Klassen durch instanceof oder andere Reflections nötig macht. Workflow vereinfacht sich von: Befehl parsen und Klasse der Nachricht bestimmen, Serialisieren, Versenden und am Server zu Deserialieren mit anschließender Bestimmung der Nachrichtenklasse zu einem einfacheren Workflow: Die Nachricht versenden, die Nachricht zerlegen und dem Nachschlagen, ob der Befehl bekannt ist. Der ServerCommander liefert grundsätzlich kurze Rückmeldungen über den Ausführungsversuch und bei Aufforderung auch erweiterte Meldungen. Zu jedem Command sind Hilfstexte vorhanden, welche die Syntax, Einsatzbedigungen und Zweck der Commandbausteine erläutern. Ein wichtiges Ziel beim Design des ServerCommanders war, die Vererbung überflüssig zu machen, um anderes Verhalten zu erreichen.

5.7.3 Aufbau des ServerCommanders

Der Servercommander besteht im Kern aus einer HashMap, in der die Commands mit ihrem Namen hinterlegt sind, einer Methode, die Verbindungen annimmt (awaitingClient()) und einer Methode für das Interpretieren von Nachrichten (interprete()). Das Enum CommandState gibt Rückmeldung, ob die Argumente eines Commands zulässig waren oder ob der Server bzw. das Game den ServerCommand ausführen durfte oder ausgeführt hat.

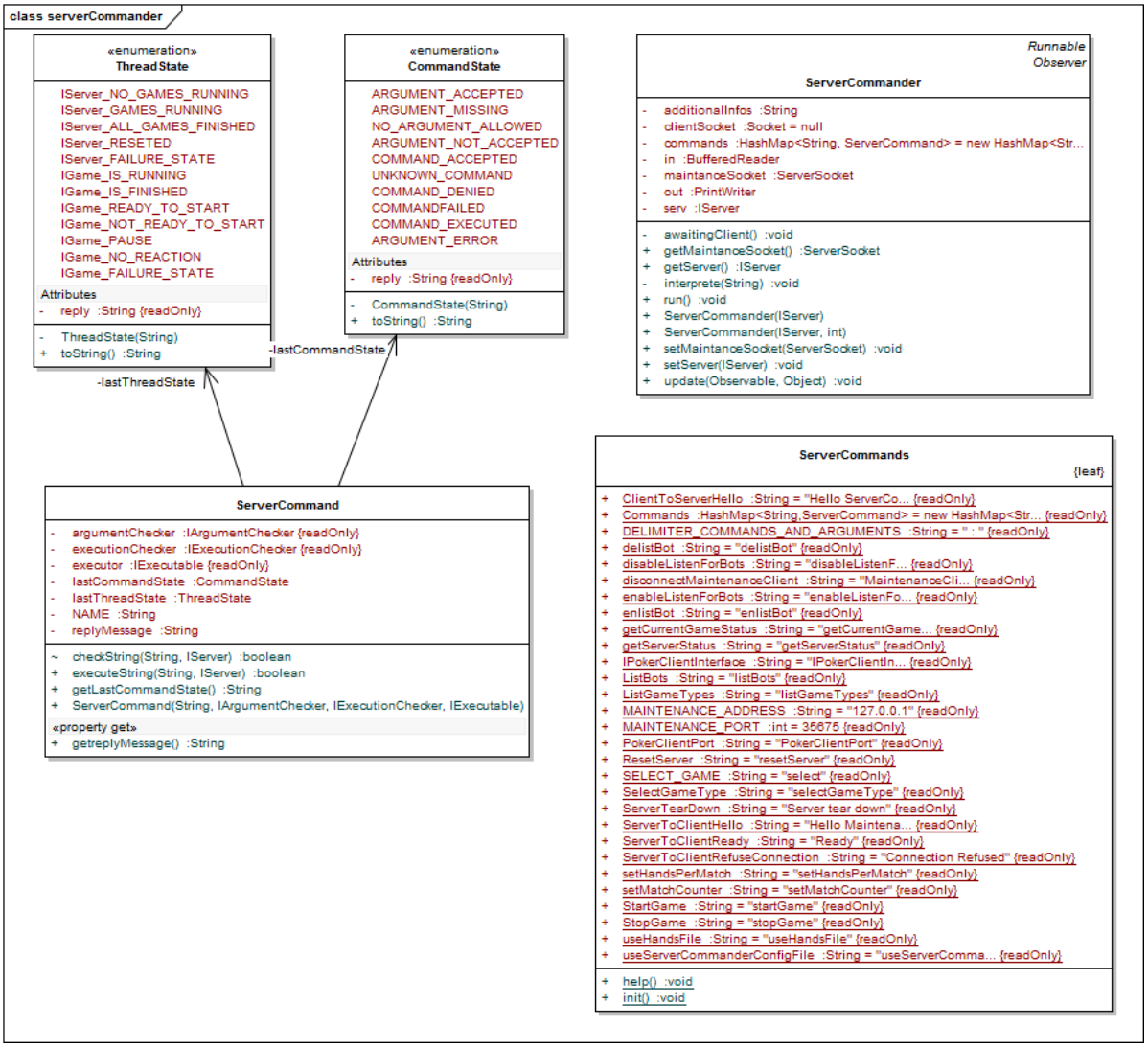

Abbildung 5.3.: ServerCommander Grundaufbau. ServerCommands liefert die Namen der Commands. In ServerCommander.commands werden die Namen der ServerCommands und die ServerCommands selbst hinterlegt.

Das Enum ThreadState wird von Servern und Games geliefert, welche die erweiterten Interfaces IServer und IGame umsetzen. Die Commands prüfen, ob Server oder Game in einem zulässigen Zustand sind, bevor das Command durchgeführt wird. Es ist Aufgabe von Server und Game, ihren Status korrekt zu publizieren. Der Aufbau des ServerCommanders folgt dem Command-Pattern.

5.7.4 ServerCommands und deren Anwendung

Ein ServerCommand besteht aus genau drei Bestandteilen, die beliebig kombiniert werden dürfen. Das erste Bauteil ist der IArgumentChecker, der Argumente prüft. Dessen check-Methode wird von ServerCommander.interprete zuerst ausgeführt. War dieser check zufriedenstellend, wird ein IExecutionChecker bemüht. Objekte dieses Interfaces tragen Verantwortung, nur in zulässigen Zuständen von Server oder Game, die Methode interprete() fortfahren zu lassen. In dritter Stufe folgt ein Objekt vom funktionalen Interface IExecuters mit der Methode IExecutable. Ein Lambda-Ausdruck ist an dieser Stelle zweckmäßig, weil sich jedes ServerCommand in seiner Wirkung unterscheiden soll; also gibt es auch keinen Grund für echte Klassen. Die Objekte beider Checker-Klassen sind regelmäßig wiederverwertbar und werden in den entsprechenden Packages gesammelt.

5.7.5 IArgumentChecker und IExecutionChecker

Die IArgumentChecker sollen die Argumente einer Nachricht überprüfen, bevor Parameter daraus entstehen dürfen. Je nach Aufgabe unterscheiden sich die Konstruktoren dieser Checker. Beispielweise ist der NominalValueChecker dar-

auf ausgerichtet, einen Array mit Strings anzunehmen, um später Argumente auf Gleichheit mit einem der Strings zu matchen. Das Prüfen übernimmt immer die Methode checkArguments(String) mit der Rückgabe eines CommandState Enums. Dabei nimmt checkArgument stets die ganze Nachricht entgegen. Die Methode helpChecker() liefert bei Aufruf Hilfestellung, was vom IArgumentChecker-Objekt geprüft wird, wozu es eingesetzt wird und wie die akzeptierte Syntax ist. Bei Bedarf kann durch getAdditionalCheckInfo() eine genauere Meldung über den letzten Einsatz dieses Objekts geliefert werden. Diese Methode soll die Fehlersuche beim administriern des Servers erleichtern.

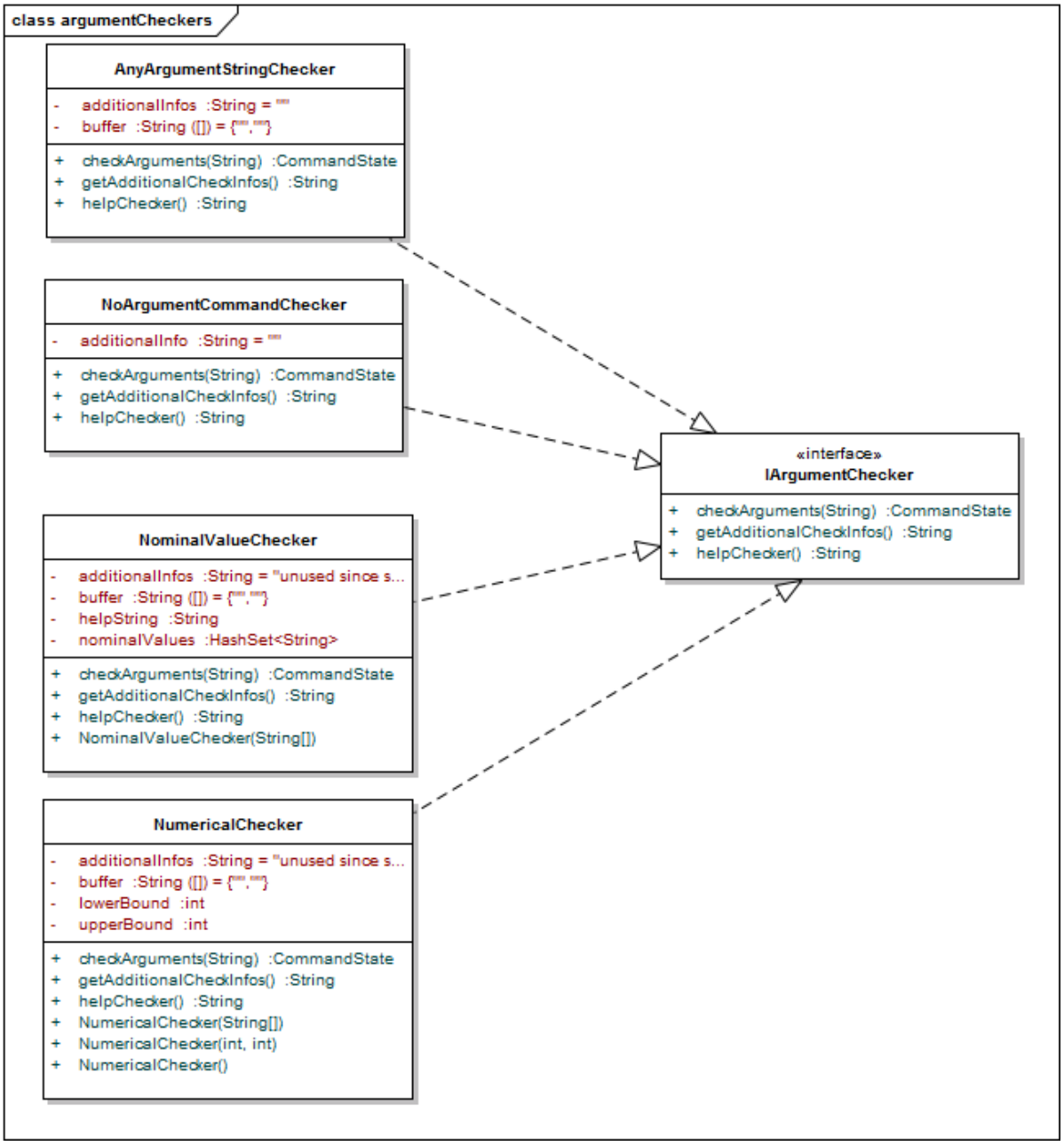

Abbildung 5.4.: Implementierungen von IArgumentchecker. AnyArgumentChecker prüft nur, ob die Trennzeichen korrekt sind.

Die IExecutionChecker sind im Aufbau ziemlich ähnlich. Die Funktion der Methoden helpChecker() und getAdditionalInfo() sind analog zu denen im ArgumentChecker.

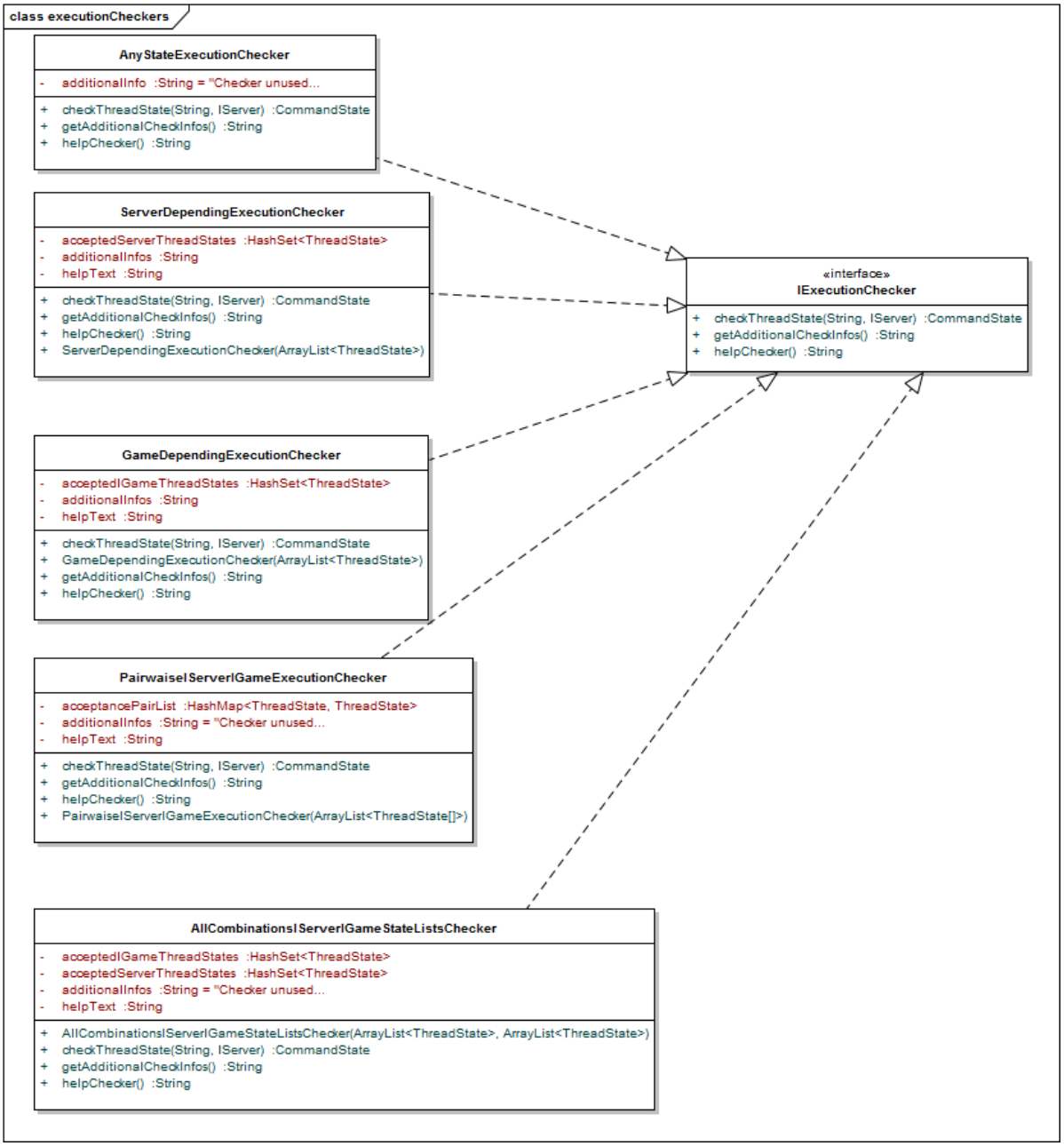

Abbildung 5.5.: Implementierungen von IExecutionChecker.

Die IExecutionChecker haben die Aufgabe, darüber zu wachen, ob ein IExecutable ausgeführt werden darf. Hierzu bekommen die Konstruktoren eine Liste von zulässigen ThreadStates überreicht. Die verschiedenen Varianten von IExecutionChecker beziehen sich auf den Server oder das Game oder beides. Dieser Checker braucht keine Parser, da er nicht mit Eingaben von einem Benutzer oder mit einer Netzwerkverbindung in Berührung kommt.

Beide Checkertypen sind seiteneffektfrei und dürfen jederzeit ausgeführt werden.

5.7.6 IExecutable

Mit JAVA 1.8 wurden lambda-Ausdrücke eingeführt. Diese sind ideal für die Umsetzung von Strategy- und Command-Pattern. Commands haben die Eigenschaft, dass diese oftmals genau eine Methode umsetzen, die auch nur genau einmal gebraucht wird. Eine Klasse dafür zu erstellen, erhöht den Wartungsaufwand, der sich kaum lohnt. Nested-classes sind bereits besser geeignet, aber immer noch ziemlich sperrig. Lambda-Ausdrücke (hier in Java eher Lambda-Objekte) sind kompakt und eignen sich für den ServerCommander ausgezeichnet.

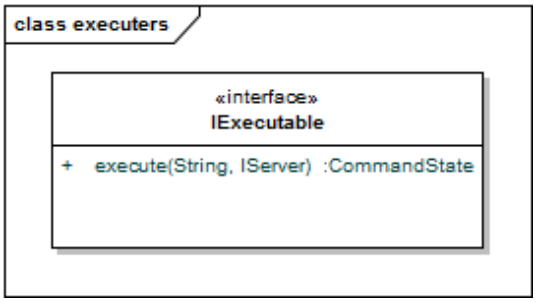

Abbildung 5.6.: Das funktionale Interface IExecutable

Die IExecutables bekommen durch den Parameter incomingCommand die komplette Nachricht (Befehl) übermittelt. Diese braucht nur am Trennzeichen geteilt zu werden, um die Argumente zu erhalten. Der andere Parameter ist der IServer, auf dem der Befehl ausgeführt werden soll. An dieser Stelle werden Methoden am Server oder Game aufgerufen. Durch die Erweiterung der Schnittstelle IServer ließe sich immer noch eine dispatcher-Methode umsetzen, um eine zweite Methodik für Methodenaufrufe zur Verfügung zu haben.
Hier ist ein Beispiel für die Einlagerung eines ServerCommands in eine HashMap mit einem lambda-Ausdruck als IExecutable.

```
this.commands.put(ServerCommands.SELECT_GAME,
        new ServerCommand(ServerCommands.SELECT_GAME,
                new AnyArgumentStringChecker(),
                new AnyStateExecutionChecker(),
                (( String arg, IServer server) -> {   // begin Lambda
                        if(server.setCurrentGame(arg)) {
                                this.additionalInfos="parameter: "+arg+" could be
                                    processed.";
                                return CommandState.COMMAND_EXECUTED;
                        }
                        else{
                                this.additionalInfos="parameter: "+arg+" could not be
                                    processed.";
                                return CommandState.ARGUMENT_ERROR;
                        }
                })));  //end Lambda
```

5.7.7 Anmeldevorgang am ServerCommander

In der run Methode wird auf eingehende Verbindungen gewartet und mit einem einfachen Protokoll sichergestellt, dass der Client mit dem ServerCommander verbunden sein möchte:

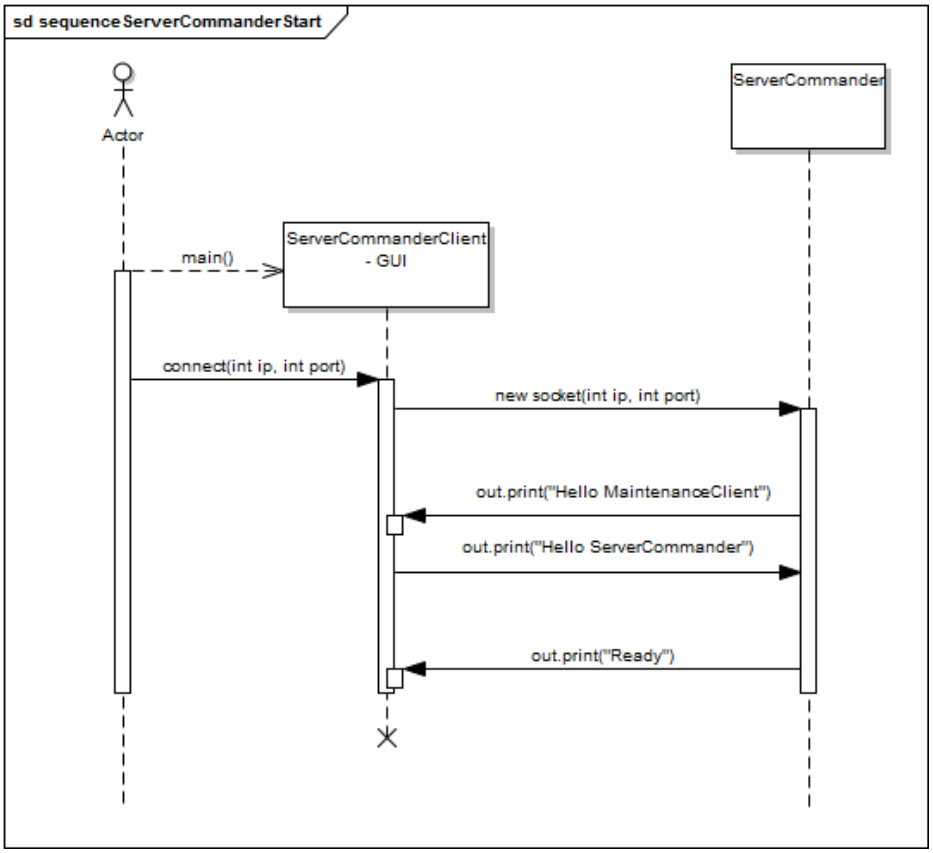

Abbildung 5.7.: Anmeldeprotokoll: Es wird in der Methode run() unter Aufruf von awaitingClient() durchgeführt.

Erfolgt keine Begrüßung durch den Client, wird die Verbindung abgebrochen mit der Nachricht:"Client on Socket: "+clientSocket.toString()+"not understood, refusing Connection". Ist ein Client angemeldet, dann ist der ServerCommander bereit, Nachrichten zu interpretieren.

5.7.8 Nachrichten an ServerCommander

Die Nachrichten an den ServerCommander sind ganz einfach gestaltet:
<Name des Commads> <Trennzeichen> <Argument>
Ausformuliertes Beispiel: selectgame : ArizonaStu_EVBot
Das Trennzeichen ist eine globale Variable, die derzeit: „ : “ lautet und in der Klasse ServerCommands.DELIMITER_COMMANDS_AN
zu finden ist.

```
String nextCommand="selectgame : EVBot_vs_ArizonaStu";
String delimiter =ServerCommands.DELIMITER_COMMANDS_AND_ARGUMENTS;

this.commands.get(nextCommand.split(delimiter)[0].execute(nextCommand,
    this.getIServer);
```

Die Nachrichten wurden entworfen mit dem Ziel, so einfach wie möglich für Menschen schreibbar zu sein. Es lassen sich ServerCommands mit mehreren Argumenten durchaus realisieren. Dafür ist nur ein entsprechender Checker notwendig, der nach Möglichkeit bestehende Checker konsumiert. Scripte können einfach realisiert werden, indem einfach in jede Zeile ein Command geschrieben und Zeile für Zeile ausgeführt wird.

5.7.9 Fortschritt und Qualitätssicherung

Die IArgumentChecker und IExecutionChecker sind fertig implementiert, mit JAVA-DOCs dokumentiert und sind mit JUnit-Tests versehen. Der IExectionChecker wurde mit einer Testimplementierung von IServer getestet. Die Checker wurden mit einem strengen Exceptionhandling ausgestattet. Die ServerCommands wurden getestet und vorgestellt. Die anschließende Diskussion zwischen dem Autor dieser Arbeit und seinem Zweitgutachter über die Notwendigkeit der

Überarbeitung von Server, Game und GameState ergaben: Die Situation hat sich grundsätzlich verändert, es werden mehr Spieltypen (am dringensten Kuhn-Poker für ACPC-Tuniere) gebraucht und der Server soll administrierbar werden, statt Einstellungen direkt am Quellcode durchführen zu müssen. Zusammen mit der erschwerten Übernahme von Modulen aus anderen Pokerprojekten (Kapitel 5.6) und auch wegen des anstehenden Umbaus der Spieloberfläche, wurde die ehemalige Hauptbedingung, möglichst nahe am Limit 2010 Framework zu bleiben, fallen gelassen. Dafür rückt nun TUD-Pokerframework 2.0 wie es in Kapitel 5.11 beschrieben wird in den Vordergrund. Die Fertigstellung von Version 1.5 verbleibt als Teilaufgabe für das TUD-Pokerframework 2.0.

5.8 Spiel Mensch gegen Pokerbot

Für das Spiel Mensch gegen Pokerbot wurde von Max Bank Bank (2011) eine grafische Benutzeroberfläche entwickelt. Die Erweiterungen für Nolimit sind bereits eingepflegt. Die parallele Entwicklung einer neuen Spieloberfläche (die auch Kuhn-Poker hätte unterstützen sollen) für Version 1.6 wurde, wegen der komplexen Auswertung des GameStates beziehungsweise seiner Erben, recht schnell nach Entwicklungsbeginn zurückgestellt. An der bestehenden GUI ist die Neuregelung des Netzwerkzugangs durchgeführt worden. Der TUDPokerClient und der MultiplePokerClientManager sind aus dem Nachrichtenzustellsystem entfernt worden, aber werden von der GUI immer noch verwendet. Damit geht auch die Abtrennung der Lobby aus der alten GUI einher. Die Spieloberfläche ist als Standalone Version für das TUD-Pokerframework 1.8 verfügbar.

5.8.1 Grenzen der Benuterzoberfäche von Max Bank

Diese GUI braucht Anpassungen für zukünftige Spieltypen, weil die jetzige nur für Texas Hold Em Fixed Limit ausgelegt ist. Das zugrundeliegende Model, welches den GameState aus Limit 2010 umformt, und der dazugehörige Controller sind nur in der Lage, Texas Hold Em Spiele handzuhaben. Die View ist ebenfalls starr für Texas Hold Em konstruiert. Diese Umsetzung ist nicht portierbar für andere Spieltypen.

5.8.2 Poker-Benutzoberfläche nach der Anpassung

In der GUI nach Version 1.6 soll der Benutzer durch Argumente am Programmstart die IP-Adresse, den Port, die Kommunikationsschnittstelle und den Spieltyp einstellen. Es stehen eine Verbindung zum Alberta-Server nach Variante der ACPC und zum TUD-Pokerframework zur Auswahl. Fällt die Wahl auf das TUD-Pokerframework, dann meldet es sich mit seiner GUI an wie jeder andere PokerClient auch. Es wird keine aktive Anmeldung am Server geben. Die GUI wird keine Verwaltungsaufgaben mehr wahrnehmen, diese sollen später an ServerCommander und seine Clients deligiert werden. Sie dient nur noch dem Spielen.

Die Lobby wird entfernt, weil sie mit der Änderung des Nachrichtenzustellsystems nutzlos geworden ist. Der TUD-PokerClient und der MultiplePokerClientManager wurden vor der Anpassung im Nachrichtenzustellsystem ausschließlich für die Lobby der GUI verwendet. Ein weiterer Grund die Lobby zu entfernen: Ein menschlicher Spieler braucht zum Spielen keine zusätzliche Funktionalität im Vergleich zu einem Pokerbot. Damit ist ein PokerClient mit einer GUI ausreichend. Der Verwaltungsbereich (siehe Kapitel 5.7) der GUI ist nur so mächtig wie die Nachrichten, die zugestellt werden können. Keines der beiden Konstrukte rechtfertigt die entfernten Zwischenschichten. Es gibt eine Erweiterung für das Spielen zusammen mit einem Bot. Der macht seinen Vorschlag und der Mensch entscheidet.

5.8.3 Grundlage für ein Ensemble-Bot-System

Diese Erweiterung für Zusammenspiel von Bot und Mensch stellt auch gleichzeitig eine Grundlage für ein schlankeres Ensemble-Bot-System dar. Es wird hierzu lediglich ein geeigneter Container für die verschiedenen PokerClients benötigt, die bei einem neuen GameState ein Update erhalten. Im Gegenzug teilen diese dem Masterbot per Observer-Pattern ihre eigene Referenz mit dem beabsichtigten Spielzug mit. Der Masterbot braucht nur noch die übermittelten 2-Tupel für die primitivste Lösung in einer Map einzulagern und abzustimmen, wenn genügend Ergebnisse vorliegen. Alles Weitere wie Gewichtung der Abstimmer, Abstimmungsmodus und Betsize-Bucketing sind Themen für eine eigenständige Arbeit.

5.8.4 Implementierung der Anpassung

Das Model verwendet nun statt des originalen GameStates aus Limit 2010 den abstrakten GameState aus den Versionen 1.X. Für die Controller- und die Model-Klasse der GUI wirkt sich das nur auf das Auslesen der Actionlisten aus. Statt des

Action Enum wird nun die Anzeige durch die Auswertung der Integers (wie in Kapitel 1.1 für Nolimit 2014 beschrieben) vorgenommen. Alle anderen verwendeten Methoden des GameStates beziehen sich auf die Texterzeugung. Jegliche weitere Funktionalität (außer das Spielen und das Verbinden mit dem Server) werden verbannt. Für das Nolimit-Spiel gibt es nun eine editierbare ComboBox für bet-Sizes. Im Limit-Modus wird weiterhin der Raise-Button benutzt. Diese GUI ist auf einfachem Wege wieder in eine umfangreichere Benutzeroberfläche integrierbar, weil der Startvorgang der Spieloberfläche in eine statische Methode GUIPanel.guiPanelFactory(Controller, Model) ausgelagert worden ist und nur die Übergabe von passenden Argumenten benötigt. Es mussten zwei Klassen PokerClient-Klassen geschaffen werden und ein Interface für diese beiden Klassen. Die HumanPokerClient-Klassen sind die Verbindung zwischen Nachrichtenzustellsystem und GUI. Für beide Kommunikationsrichtungen wurden jeweils Observer verwendet. Die Spieloberfläche blendet jeweils das nicht benötigte Bedienelement aus (Combobox beim Limit-Spiel und Raise-Button bei Nolimit). Das Spiel zusammen mit einem Bot wird per Argumentübergabe beim Start der GUI festgelegt. Der eingebettete PokerClient erhält vom HumanPokerClient per Observer zur gleichen Zeit Spielzustandsupdates wie die GUI. Hat der eingebettete PokerClient seine Berechungen durchgeführt, dann meldet er dies durch den Observer, worauf die GUI ein Update durchführt. Für diesen Spielmodus wurde ein neuer Button eingeführt, der mit seiner Beschriftung den vorgeschlagenen Spielzug des gekapselten PokerClients anzeigt. Der Spieler kann nun entscheiden, ob dieser Spielzug durchgeführt werden soll. Die Kapselung ist nicht für sämtliche alte PokerClients getestet, sie wird jedoch mindestens für alle PokerClients, die mit diesem Framework ausgeliefert werden, funktionieren.

5.8.5 Neuimplentierung der Spieloberfläche

Die Neuimplentierung der Spieloberfläche, die auch Kuhn-Poker einbinden sollte, wurde relativ schnell - unter den vorherigen Eindrücken durch Kuhn-Poker (Version 1.2), die Übernahme von Modulen (Version 1.4) und dem Studium der bestehenden Spieloberfläche - in der Priorisierung herabgestuft. Die Auswertung der GameState-Erben ist mühsam und nicht zielführend, weil nicht Kuhn-Poker unterstützt werden kann auf dieser Modellplattform. Aufgrund des Studiums der GUI von Max Bank haben sich Konzepte für das TUD-Pokerframework 2.0 herauskristallisiert. Die wichtigste Erkenntnis ist eine neue Auftrennung des GameState zu einem Controller und einem Model, welche sich auch durch eine View, einen Pokertisch anzuzeigen und zu bedienen, also ein echtes MVC-Pattern eignet. Der Begriff View soll hier nicht auf die GUI beschränken, sondern auch dem Pokerbot soll eine View geboten werden (die Pokerbot gerecht ist, also keine GUI). Das neue Model soll die Objekte des Pokerns abbilden. Es werden dann Seats, Spieler, ein Pokertisch, Wettrunden, Actions, der Dealerbuttom und all die anderen Poker-Objekte im Model als echte Objekte abgebildet, statt wie bisher auf eine stark minimierte Darstellung zu setzen, die schlecht manipulierbar und ohne komplexe Verarbeitung kaum informativ ist. Mit diesem neuen GameState ließe sich auch eine anpassungsfähige Spieloberfläche realisieren.

5.8.6 Pathologisches Problem: GameState

Auch bei der Implementierung der GUI wurde der GameState dekoriert und zwar durch die Klasse Model. Zudem ist auch die Klasse Player für die GUI nochmals im Package: pokertud.clients.swingclient.* beziehungsweise jetzt in poker.clients.swingclient2015 implementiert worden. Es wird erneut gezeigt, dass der Umgang mit dieser Klasse GameState umständlich ist.

5.9 Beobachtungs-GUI für laufende Bot-Spiele

Die Beobachtung der Pokermatches an einem Spieltisch wurde durch die GUI von Max Bank realisiert. Für Version 1.7 ist eine Auffrischung des Beobachtungsmodus geplant. Als Grundlage wird die in Version 1.6 geschaffene GUI benutzt. Die Beobachtungsoberfläche von Max Bank ist auch von den Netzwerkumstellungen betroffen, damit sind auch in diesem Bereich Anpassungen durchzuführen. In der ursprünglichen Variante konnte sich ein aktiver Spieler wahlweise als Beobachter oder Spieler anmelden.

5.9.1 Verbesserungen an der bestehen Beobachtungs-GUI

Die Beobachtungsoberfäche ist nun in der Lage Limit und Nolimit Texas Hold Em darzustellen. Die Verbesserungen beginnen bereits am Server. Es werden in diesem nun getrennte Listen zwischen Beobachtern und Spielern geführt, welche dann einem zu startenden Spiel zugeführt werden. Die Game-Instanz arbeitet dadurch nun korrekt und lässt nur als Beobachter ausgewiesenen Accounts den kompletten Spielzustand zukommen. Beobachter melden sich als solche an und werden daran gehindert auf das Spiel Einfluß zu nehmen. Es werden von Beobachtern nur noch Abmeldenachrichten verarbeitet. Es können sich derzeit nur die beiden HumanPokerClient Klassen, wegen Ihres HumanInterface, als Beobachter anmelden.

5.9.2 Implementierung

Einige Implementierungsdetails betreffen den Server und das Nachrichtenzustellsystem, damit sind sie von Interesse und werden an dieser Stelle vorgestellt.

1. Die AccountLoginMessage hat ein weiteres Feld bekommen: boolean spectator = false. Dieses Feld ist per Default auf false für alle PokerClients, die nicht HumanInterface verwenden.

2. Die Accounts des Servers haben ein weiteres Feld erhalten: int UID. Es dient der Absicherung gegen illegale Spielzüge durch Beobachter und erschwert die illegale Anforderung eines kompletten GameState für gewöhnliche PokerClients. Die UID ist die Absendernummer einer Nachricht und wird einem Sender beim Verbindungsaufbau zugewiesen.

3. Der Server führt nun ein Feld für Beobachter: LinkedList<Account> spectators. Dort werden alle Beobachter getrennt von den Spielern aufbewahrt, um diese einem Game zu übergeben.

4. Die Methode Server.handleMessage(Message) trennt bei Ankunft einer AccountLoginMessage zwischen Beobachtern und Spielern. Im Fall einer ActionMessage fragt handleMessage bei den laufenden Games nach, ob sich der Absender in der Beobachterliste oder Spielerliste befindet. Spielzüge von Beobachtern werden verworfen. Ist ein Account in beiden Listen enthalten, dann meldet der Server in der Konsole (über den Error-Out), um welches Spielerkonto es sich handelt, aber leitet die Action zum Game weiter (ist durch Auskommentieren einer Zeile verhinderbar).

5. Das Game versorgt die Spieler mit partiellen Informationen und die Beobachter mit dem vollen Spielzustand. Hat ein Pokerbot zwei ClientMessageClient Instanzen implementiert, dann erhält er zwei Updates pro Spielzug, das gilt insbesondere für eine Anmeldung als Spieler und Beobachter.

6. Die Klasse ClientRunner trägt die Verantwortung die Beobachter korrekt zu erkennen und diese entsprechend anzumelden. Dazu wurde die Hilfsmethode decideSpectator hinzugefügt, die nur HumanInterface Inhaber erlaubt Beobachter werden zu dürfen. Wird schummeln generell erwünscht, dann ist diese Methode durch client.isSpectator() zu ersetzen.

7. Im Nachrichtenterminal ClientMessageClient wurde eine neue connect-Methode hinzugefügt, die es nun gestattet, die erweiterten AccountLoginMessages zu instanziieren.

8. In der abstrakten Klasse PokerClient ist die Methode isSpectator() erhalten, welche immer false liefert. Für schummelde PokerClients ist die Methode zu überschreiben.

5.9.3 Schummeln durch Beobachtung - Lernen durch Schummeln

Den eigenen PokerClients kann, das Schummeln durch 4 Schritte beigebracht werden. Versehentliches Schummeln soll so weit wie möglich ausgeschlossen werden. Der wie folgt beschriebene Mogel-Bot versendet zwei AccountLoginMessages und hat zwei verschiedene UID:

- isSpectator() überschreiben.
- humanInterface implementieren.
- An die Methode setBot(PokerClient) einen PokerClient überreichen und mit Observer ausstatten.
- Den hinterlegten PokerClient mit dem ClientRunner starten.

Eine Verwendung für den mogelnden PokerClient wäre die Spielzugverifikation. Der Basis-Bot mit dem vollständigen GameState berechnet einen optimalen Spielzug nach einer geeigneten Heuristik, während der untergeordnete PokerClient ein Ergebnis liefert und versendet. Nach dem Match kann dann der untergeordnete PokerClient bezüglich der Heuristik evaluiert werden. Eine zweiter Ansatz wäre das direkte Lernen aus den gegnerischen Spielzügen noch während des Spielens. Der PokerClient mit unvollständiger Information tätigt die Spielzüge, während der andere beispielsweise ein gemeinsames Gegnermodell befüllt. Damit ließe sich möglicherweise die Funktion eines Gegnermodells nachweisen und in seiner Wirkung einordnen.

5.9.4 Weitere Arbeiten am Beobachtungsmodus

Genau wie in die Spieloberfläche werden in die Beobachtungsoberfläche in Hinblick auf das TUD-Pokerframework 2.0 Modernisierungen eingepflegt werden. Für den Beobachter soll es ermöglicht werden, sich bei einem laufenden Match an- und abzumelden. Im Beobachtungsmodus könnten spezialisierte Bots für ausgefeilte Logging-Verfahren zum Einsatz kommen. Es ist zu überlegen, ob der Beobachtungsmodus in den ServerCommander verschoben oder auch dort angeboten wird.

5.10 Unterstützung von Legacybots

Die Anforderung alte Pokerbots zu unterstützen ist erfüllt. Es wird hierzu das alte Anmeldeverfahren weiterhin unterstützt und die Annahme getroffen, dass ein alter PokerClient immer ein Texas Hold Em Limit Spieler ist. Dabei sind einige unvorhergesehene Probleme aufgetreten, die jedoch gelöst werden konnten.

5.10.1 Konzept für die Unterstützung von Legacybots

Das neue Anmeldeverfahren liefert in der AccountLoginMessage den Klassenpfad des zu verwendenden GameState. Der Server entscheidet, ob ein Match in der Bot-Konstellation zulässig ist (wie in Kapitel 5.2 zu Version 1.0 beschrieben). Die AccountLoginMessage wurde derart gestaltet, dass ein leerer Klassenpfad eine null-Referenz ist. Der Server interpretiert null-Referenzen als GameStateLimit. Die Klasse Account wurde um das Feld: boolean legacy erweitert. Anmeldungen mit null-Referenzen im Klassenpfad des GameStates werden mit legacy=true markiert. Das Game sendet an die als legacy markierten Accounts im alten Verfahren mit der Übermittlung des GameState-Strings, während an die Accounts neuer Pokerbots ein angepasster GameState, wie in Kapitel 5.4 für Version 1.3 beschrieben ist, übermittelt wird. Die ActionMessage ist auch für die Verwendung alter PokerClients umgeschrieben worden. LegacyBots tragen in die ActionMessage in das Feld action vom Typ Enum Action ihren Spielzug ein, alle anderen PokerClients benutzen den entsprechenden String.

5.10.2 Implementierung des Legacy-Modus

Die AccountLoginMessage wurde lediglich am Feld PokerClientType um den Defaultwert null erweitert. Beim Server wurde in der Methode handleMessage in der Fallunterscheidung für AccountLoginMessages eine Erweiterung vorgenommen. Es wird eine Meldung auf der Konsole ausgegeben, wenn eine null-Referenz für den GameState-Pfad auftritt. Der Account wird erzeugt mit der Einstellung legacy=true und es wird im Server-Feld List<String> botTypes mit GameStateLimit.class.getName() eingetragen. Die Anmeldung wird mit den der anderen PokerClients überprüft durch die Server-Methode checkBotTypes(). Die Klasse ActionMessage hat einen weiteren Konstruktor erhalten, der Spielzüge in String-Repräsentation annimmt und im Feld String actioncode einlagert. Der alte Konstruktor nimmt weiter Action-Enums an, aber wandelt diese zusätzlich in die String-Repräsentation um, so dass die getAction Methode der ActionMessage immer einen String liefern kann. Erst mit diesem Schritt genügt genau eine Klasse für Spiele (die Klasse Game), welche auf der abstrakten GameState Klasse arbeitet. Die Game-Klassen für Nolimit und Limit werden nicht weiter benötigt. Denn der Server stellt nun die GameStateFactory auf den GameState-Erben ein, für die sich die PokerClients angemeldet haben. Also liegt im Game immer die passende Implementierung des GameState vor. Der Weg zu dieser Lösung ist durch vielfaches Testen mit dem PokerBots von Theo Kischka entstanden.

Bei den Testläufen ist aufgefallen, dass sich Game und GameState für Limit und Nolimit leicht im Verhalten unterscheiden und leider auch vom Verhalten für Legacybots abweichen.

Die Probleme entstanden durch die abweichende Benutzung der Steuerungszustände der Enums Position und Street. Nun verhalten sich alle Modi gleich nach gleichzeitiger Entfernung von kleinen Fehlern.

Die Klassen GameStateLimit und GameStateNolimit haben nun eine derartige Ähnlichkeit, dass bei einem nächsten Bearbeitungsschritt beide GameStates wieder vereinigt werden könnten, wenn die restlichen ungleichen Methoden in eine Interface ausgelagert werden. Die Implementierungen dieses Interface würden dann im GameState eingesetzt werden, um die Unterschiede zwischen Nolimit und Limit Texas Hold Em zu erzeugen. Die beiden restlichen Analysemethoden, handleAction(), currentPlayMakesAction(), einige Hilfsmethoden und die toString-Methoden würden mit den Implementierungen dieses Interfaces austauschbar werden. Das Strategy-Pattern wird im GameState angewendet.

Es wird nun korrektes Texas Hold Em mit allen PokerClients gespielt.

5.11 Von der Modularisierung zum TUD-Pokerframework 2.0

Während der Bearbeitung der Ziele und dem Prüfen der Thesen sind Schwächen im bestehenden Framework aufgedeckt worden. Problematische Bereiche sind der Spielzustand, der Server, der Dealer, die Benutzerschnittstellen und die

Eingliederung von Modulen. Oftmals sind Teilprobleme dieser Bereiche ineinander verzahnt. Das erfolgreiche Erstellen einer Lösung wird oftmals nur seine Wirkung entfalten, wenn auch die angrenzenden Bereiche in einen guten Zustand versetzt werden. Das Nachrichtenzustellsystem könnte Veränderungen erfahren, aber diese sind nicht so kritisch wie die der anderen Bereiche. In den Unterkapiteln werden die Ansätze und Teil-Implementierungen vorgestellt. Die Verzahnung der Problemebereiche wird nach vorliegendem Kenntnisstand näher gebracht.

5.11.1 Spielzustand

Im Bereich des Spielzustandes sind folgende Teilprobleme erkennbar: Der Spielzustand selbst, das Updaten und die Erzeugung des Spielzustandes, sowie die Trennung von Model und Controller. Alle Teilprobleme sind miteinader verwoben. Die erste ist die Parametrisierung des GameState bezüglich der Bietrunden und Positionen. Verschiedene Pokertypen haben unterschiedliche Anzahlen und Bezeichnungen für Bietrunden und Positione; damit ist es zweckmäßig, dass sich der GameState anpassen kann. Die zweite Maßnahme ist, alle Update- und Spielregeln aus dem GameState zu verbannen und an einen Controller zu delegieren. Danach folgt die Zerlegung des Spielzustands in die Objekte der Pokerdomäne. Die Parametrisierung von GameState und Model erfolgt durch die Wahl der Generics am PokerClient.

Der Controller für den neuen GameState

Die Klasse PokerController ist fertig implementiert, jedoch noch nicht mit JUnit-Tests ausgestattet. Diese Klasse ist Teil eines PokerClients und hat die Verantwortung, Spielzüge vom PokerClient oder vom Nachrichtensystem entgegenzunehmen und die resultierenden Updates des GameStates durchzuführen. Als zusätzlicher Dienst können Spielzüge auf Legalität getestet werden ohne ein Update. Die Update Methode nimmt ActionObject, Action und die String-Repräsentation an. Eine weitere Updatemethode nimmt Spielkarten entgegen, die vom Dealer ausgeteilt werden. Mit dieser Methode können eigene verdeckte, offene Holecards anderer Spieler und natürlich Commoncards übermittelt werden. Damit wird es wesentlich einfacher andere Poker-Varianten als Texas Hold Em und Kuhn-Poker zu spielen wie Draw- oder Stout-Poker.

Die Update-Methode ruft in fester Reihenfolge fünf private Methoden auf:

- checkContainer: fragt alle Untercontainer des GameState, ob diese bereit sind.
- checkCommonPokerRules: testet alle Regeln, die für jeden Pokertyp gleich sind bezüglich eines legalen Spielzugs.
- checkPreconditions: testet alle Spielregeln des aktuell gespielten Pokertyps.
- manipulateGameState: führt alle Änderungen am GameState durch.
- checkPostConditions: führt Prüfungen auf Korrektheit nach einem Spielzug durch oder erledigt nachgelagerte Einstellungen.

Der Controller ist vollständig parametrisiert wie der GameState. Die Spielregeln werden durch das Strategy-Pattern in den Controller eingelagert. Alle privaten Methoden außer checkContainer beziehen die Spielregeln aus einem Erben der abstrakten Klassen APokerClientConfig. Die Check-und Manipulator-Methoden erwarten Listen von seiteneffektfreien Lambda-Ausdrücken. Wenn ein Lambda-Ausdruck fehlschlägt, wird der Updatevorgang abgebrochen. Damit ist die Klasse PokerController voll modular.

Die Strategie des PokerController

Die abstrakte Klasse APokerClientConfig ist fertig gestellt. Sie besteht aus einer privaten HashMap für die commonPokerRules und drei HashMaps für die Pre- und Postconditions und den Manipulatoren. Es sind public final static String Konstanten mit den Namen der gemeinsamen Pokerregeln, die in den HashMaps als Schlüssel verwendet werden. Die anderen HashMaps nehmen ebenfalls Strings als Schlüssel. Als Value dienen Lambda-Ausdrücke vom funktionalen Interface IGameStateConstraint, die stets ein ActionObeject und GameState als Parameter übergeben bekommen. Es gibt eine private Methode initCommonRules, die während der Instanziierung die entsprechende HashMap mit Regeln befüllt. Analog sind drei abstrakte init-Methoden für die anderen drei HashMaps vorgesehen. Vier Getter stellen die Lambda-Ausdrücke für den Controller bereit. Es sind drei überladene abstrakte Methoden namens produceActionObject vorgeschrieben. In den drei Varianten wird für den Spielzug als Argument entweder ein String oder Action-Enum oder Action-Enum mit einem Integer verlangt und jede der Methoden erwartet den GameState. Aus der Kombination von Spielzuge und Spielzustand wird ein korrektes ActionObject für den vorliegenden Spieltyp zu produziert. Alle Felder und nicht abstrakten Methoden sind final.

Die Erben für die konkreten Spieltypen implementieren in den Init-Methoden, die Spielregeln durch Lambda-Ausdrücke. Es können private Hilfsmethoden geschrieben werden, die von den Lambda-Ausdrücken verwendet werden, dann wird jedoch die Portierbarkeit schwieriger. Wir bei der Implementierung der Lambda-Ausdrücke auf private Hilfsmethoden in der Strategie verzichtet, dann könnten Lambda-Ausdrücke, von der einen Strategie zur anderen kopiert werden. Die Kopiermethode ist noch nicht implementiert. Zur Zeit existiert bereits eine Konfiguration für Texas Hold Em Limit mit den Tunierregeln der ACPC.

ActionObject

Die Klasse ActionObject ist implementiert und getestet und ersetzt alle vorangegangenen Typen und Klassen, die einen einzelnen Spielzug repräsentieren. Die Erfahrungen von Nolimit 2013 sind in diese Klasse eingeflossen. Bei der Inspektion der anderen Pokerprojekte wurde diese bestätigt. Von der Klasse ActionNolimit aus Limit 2013 sind das Enum vom Typ Action und ein int für den Value übernommen worden. Wie beim Pokerbot FatBetty werden einer Action auch dem Player zugeordnet. Theo Kischka hat bei der Klasse ActionData zusätzlich zum Player auch die Street und die Position zusammengeführt. Beim ActionObject wurde bis jetzt auf die Street verzichtet, aber auch ein Feld für die Position implementiert. Die Position ist generisch für verschiedene Pokerspielen anpassbar zu sein. Es wurde darauf geachtet, alle relevanten Daten bezüglich eines Spielzuges zusammenzuführen, so dass erstens Berechnungen nur einmal durchgeführt werden und zweitens weniger Bedarf besteht, den GameState für Projekte abzuändern.

```
public final int actionValue;
public final Action action;
public final Player<PositionLabel> actor;
public final PositionLabel actingPosition;
public final int minRaiseTo;
public final int afterActionPotsize;
public final int afterActionActivePlayers;
public final StreetState afterActionStreetState;
```

StreetObject ein Container für ActionObjects

Die Implementierung und JUnit-Tests des StreetObjects sind abgeschlossen. Der StreetContainer hat eine Reihe von Verantwortungen. Er lagert ActionObjects ein und kann die Einlagerung rückgängig machen. Er hat verschiedene Getter, die ActionObjects mit bestimmten Eigenschaften suchen. Das StreetObject liefert bietrundenspefizifische Daten wie maximumInvestedThisStreet, die getätigten Bets und Name der Bietrunde, um einige zu nennen. Die Klasse ist generisch bezüglich der Street und Position. Das verwendete Enum für die Parametrisierung StreetLabel, gibt den Namen der Bietrunde vor. Das Enum für PositionLabel gibt die Parametrisierung der ActionObject-Liste an, weil ActionObjects selbst bezüglich der Position parametrisiert sind. Für die Verwaltung durch einen umgebenden Container sind Methoden zur initialen Einstellung, zum Clonen, dem Rücksetzen der Bietrunde und Abfrage des StreetObject-Zustands vorhanden. Das StreetObject hat seine eigene Zustandsverwaltung, die folgende Zustände kennt:

- NEW_STREET Eine Bietrunde, in der noch Spielzüge keine getätigt wurden.
- PLAYING Eine offene Bietrunde, die noch Spielzüge annimmt.
- ALL_CHECKED Eine geschlossene Bietrunde, in der alle Spieler gepasst haben.
- ALL_CALLED Es sind alle Spieler mitgegangen außer dem Erhöhenden. Bietrunde ist geschlossen.
- ALL_FOLDED Alle Spieler bis auf einen sind ausgestiegen und die Bietrunde ist geschlossen.
- ALL_ALLIN Eine geschlossene Bietrunde in der alle Allin sind.

In den ActionObjects ist der StreetState festgehalten und kann zurückverfolgt werden.

StreetContainer verwaltet StreetObject

Die Klasse ist implementiert und mit JUnit-Tests ausgestattet. StreetContainer ersetzt eine Menge von Feldern aus den GameStates der Versionen 1.X:

- preflop-, flop-, turn-, riverAction : Listen von Spielzügen
- maxBetsize, potsize, betSizeToCall und maxBetsizeThisStreet : Zahlenwerte von Chipmengen
- handeledBetsThisActionsThisStreet, handeledBetsThisStreet, preflop-, flop-, turn-, und riverBets : Zähler für Spielzüge
- currentStreet : Zeiger für die aktuelle Bietrunde

Alle Felder sind weiterhin vorhanden und sind durch deprecated Methoden abfragbar, welche die alten Felder emulieren. Diese Klasse ist genau wie StreetObjects bezüglich Position und Street parametrisiert. Bei der Instanziierung eines StreetContainers, werden in der HashMap streets genauso viele Schlüssel angelegt, wie das Parameter Enum für das StreetLabel Zustände hat. Die dazugehörigen Werte der Schlüssel sind die StreetObjects. Damit passt sich der StreetContainer den Pokerspielvarianten in Hinblick auf Bietrunden an. Wegen der Positionsparametrisierung StreetObjects und ActionObjects erfolgt die Anpassund bezüglich verschiedener Positionsbezeichnung. Anstatt eines Konstruktors wird das Factory–Method-Pattern (Gamma u. a., 1995, S.107ff) verwendet, um fertig konfigurierte StreetContainer zu liefern. Die statische Methode streetContainerFactory setzt das Pattern um. Im StreetContainer gibt es eine Zustandsverwaltung:

- CURRENT_STREET_NOT_SETUP Es wurde in der aktuellen Bietrunde noch kein Initialzustand übergeben, also nimmt der StreetContainer keine ActionObjects an.
- CURRENT_STREET_NO_ACTION Die aktuelle Bietrunde hatte noch keinen Spielzug.
- CURRENT_STREET_ACTIVE Bietrunde ist offen und nimmt ActionObjects an.
- CURRENT_STREET_FINISHED Es steht der Übergang zur nächsten Bietrunde an.
- HAND_FINISHED Die Hand ist beendet.

Die Steuerungszustände vom StreetContainer und auch der StreetObjects werden vom PokerController beziehungsweise von dessen Konfiguration benutzt. Der StreetContainer ermöglicht den Zugriff auf StreetObjects und insbesondere auf die jetzt offene Bietrunde. Es ist eine Reihe von Gettern implementiert, die üblicherweise in zwei Ausführungen vorliegen. Eine Ausführung arbeitet auf der aktuellen Bietrunde und die zweite mit einem Argument für die zugesprechende Bietrunde. Sie sollen bestimmte ActionObjects, Spieler oder Kennzahlen suchen und falls das Gesuchte existiert, dann auch liefern. Neben Gettern stehen zwei Arten von Verwaltungsmethoden zu Verfügung. Die einen dienen der Verwaltung des StreetContainers, wie Cloner und Resetter. Alle anderen manipulieren StreetObjects, von denen die addActionObjectToStreet-Methoden besonders wichtig sind. Damit ist die Spielzugverwaltung modular, parametrisch, stateful und bildet die Pokerdomäne ab.

CardsContainer

Eine Klasse namens CardsContainer ist noch nicht implementiert. Sie wird bezüglich der Bietrunden parametrisiert und wird Objekte der Klasse Cards aufnehmen und verwalten. Auch diese Klasse erhält ein Enum für Zustände:

- CURRENT_STREET_COMMONCARDS_NOT_DEALT
- CURRENT_STREET_COMMONCARDS_DEALT
- CARDSCONTAINER_NOT_READY
- CARDSCONTAINER_SHOWDOWN

Die Steuerzustände werden von der Strategie des Controllers verwendet. Die Klasse wird die Felder flop, turn, river der GameStates der Versionen 1.X ersetzen. Die Getter und Verwaltungsmethoden werden denen des StreetContainers ähneln. Dies ist weiterer Schritt zur Modularisierung.

PlayerContainer

Der PlayerContainer ist nicht implementiert und liegt derzeit nur als Konzept vor. Er wird bezüglich der Positionen parametrisierbar sein, weil die Klasse mit ihrem Feld für die aktuelle Position des Spielers auch parametrisiert sein wird. Der Container erhält ein Zustandsfeld, welches die Strategie des Controllers benutzen wird. Die Belegungen könnten folgende sein:

- PLAYERS_NOT_SETUP Die Einstellungen an den Player-Objekten wurden noch nicht vorgenommen.
- PLAYERS_NOT_SEATED Die Spieler haben noch keine Position beziehungsweise Platz am Spieltisch eingenommen.
- NO_PLAYERS Der PlayerContainer ist leer.
- PLAYING_HEADS_UP Es wird im Heads Up Modus gespielt.
- PLAYING_RING_GAME Der Ring-Spiel Modus ist zu wählen.

Der PlayerContainer wird ähnliche Methoden für Verwaltung und Datenerhebung erhalten wie der StreetContainer. Insbesondere übernimmt der PlayerContainer die Instanziierung von Player-Objekten. Das Feld player im GameState der Versionen 1.X wird durch diese Klasse ersetzt.

Player

Die Abänderung dieser Klasse steht aus. Sie wird erweitert um die statistischen Kennzahlen des Players aus der GUI von Max Bank. Der PlayerState bleibt erhalten. Ein Feld mit einer eindeutigen Nummer wird implementiert, damit die Spielauswertung nicht mehr anfällig für Namensgleichheit ist. In Matches mit mehreren Instanzen einer PokerClient-Klasse kommt es zu Kollisionen in der HashMap des Auswerters, weil der selbe Spieler, in den verschiedenen GameStates der Hände, in verschiedenen Objekten vorliegt und nur über den String identifiziert wird.

PokerTable

Der Spieltisch ist implementiert und wurde mit JUnit-Tests geprüft. Die Klasse PokerTable ist bezüglich der Positionen parametrisiert. Eine Factory-Methode(Gamma u. a., 1995, S.107ff) erzeugt durch eine Namensliste für jeden Spieler einen Sitzplatz, der eine Position aus der Positionsparametrisierung erhält. Die Sitzplätze werden durch die Factory-Methode zu einem Ring angeordnet, denn ein Seat kennt seinen Nachfolger. Der Klasse ist zentrales Hilfsobjekt für den Poker-Controller und vereinfacht die Umsetzung von Spielregeln sehr stark. Der PokerTable ist der Ersatz für Steuerungsfelder bestehend aus Typen Position und Street sowie die GameState-Analyse-Methoden der GameStates der Versionen 1.X. Folgende Felder und Methoden werden durch den PokerTable obsolet:

- positionToAct
- Position.INVALID, Street.INVALID und STREET.SHOWDOWN sind Steuerdaten. Die Enum Position und Street waren in den Versionen 1.X kontextabhängig zu interpretieren. Im Steuerungskontext sind die bereits erwähnten Belegungen sinnvoll, jedoch bei der Verwendung als Spielinformationen haben sich diese Belegungen oftmals als Fehlerquelle herausgestellt. Was häufig nach der Fehlerbehebung für einen der Kontexte wiederum zu neuen Fehlern im anderen Kontext geführt hat.
- analyzeGameStateAndUpdatePlayers() startet die playerSetupNewStreet- und mehrfach die analyzeStreetAction-Methode.
- playerSetupNewStreet hat die Spieler bei einem Bietrundenwechsel korrekt eingestellt.
- analyzeStreetAction hat im Preflop alle initialen Einstellungen und die nötigen Einstellungen für die anderen Bietrunden vorgenommen. Danach wurde für jeden Spielzug einer Bietrunde handleAction aufgerufen.
- handleAction hat jede Veränderung, die ein Spielzug an einem Feld oder Objekt des GameState hervorrufen kann, durchgeführt. In Limit 2010 gab es kaum Hilfsmethoden für handleAction, was die Methode unverständlich macht.

Der PokerTable kapselt nun einen erheblichen Teil der Pokerspiellogik ein. Der PokerController verwendet lediglich die Methoden des PokerTable und braucht sich nicht um die Berechnung des aktiven Spielers beziehungsweise dessen Position zu kümmern. Die Berechnungen, ob ein Spielzug zu einer neuen Bietrunde führt, werden abgenommen. Auch ob eine Bietrunde endet, ist in Methoden gekapselt. Es werden die wichtigsten Methoden vorgestellt:

- moveDealerButton gibt dem nächsten Seat im Ring die Position Dealer; anzuwenden zu Beginn einer neuen Hand.
- giveActionToNextPlayer setzt den Player des nächsten Seat im Ring als ActingPlayer am PokerTable. Es werden die Seats nach links durchgegangen bis der erste Spieler, der nicht Allin ist oder gefoldet hat, gefunden ist.
- setNextStreetReached(T startposition) setzt als ActingPlayer, den Inhaber der Position des Parameters startposition. Falls der ausgewählte Spieler nicht mehr in der Hand aktiv ist, wird der nächste Mögliche ermittelt.
- haveAllCalled, haveChecked, haveAllFolded und areAllAllin sind selbstredend.
- willAllhaveCalled, willAllhaveFolded, willAllhaveChecked und willBeAllAllin haben als Parameter eine Action und sie liefern Aussagen über den zukünftigen Spielzustand.
- getActingPlayer emittelt den Spieler mit Zugerlaubnis.

Weitere Methoden sind implementiert, die weitere Informationen zum Spieltisch liefern oder den Spielern Sitzplätze zuweisen oder sie zum Aufstehen auffordern. Der PokerTable ist mit einem Zustandsanzeiger ausgestattet:

- ALL_CALLED
- ALL_CHECKED
- ALL_FOLDED
- NO_PLAYERS_ON_SEATS
- READY_FOR_ACTION
- SHOWDOWN

Dem Botentwickler bleibt die Steuerung verborgen, für den Spieltyp-Entwickler vereinfacht sich die Arbeit extrem, weil nur die Spielregeln in einer Konfiguration festgelegt werden müssen. Der GameState wird übersichtlicher, die Steuerung verständlich und modular durch Parametrisierung.

Seats

Die Implementierung ist abgeschlossen. Die JUnit-Tests sind mit dem PokerTable durchgeführt worden. Die Seats haben eine Parametrisierung für Position-Enums. Die Aufgabe der Seats ist es, einem Spieler einen Platz am Pokertisch zu geben. Sie tragen zur Modularisierung des Frameworks bei, in dem Spieler ausgetauscht werden können. Das ist wichtig, falls Cash-Games oder Tuniere an verschiedenen Tischen durchgeführt werden sollen. Bei Pokertunieren mit mehreren Tischen ist es üblich, dass Spieler von sich leerenden Tischen zusammengeführt werden und Ausgeschiedene Platz machen für neu Hinzukommende. Es sind einige Getter bezüglich der Sitzplatzeigenschaften vorhanden. Die Klasse Seat ist ein Decorator für Player-Objekte (Gamma u. a., 1995, S.175ff).

GameState

Die Implementierung ist angefangen worden, aber es steht noch die Implementierung des CardsContainer und des PlayerContainers aus, um den GameState für Tests bereit zu machen. Es besteht die Überlegung den GameState insgesamt mit dem Modifier final auszustatten. Damit soll in Zukunft die Integrierbarkeit von Features aus zukünftigen Pokerbots vereinfacht werden, weil die Bot-Entwickler gezwungen werden Wrapper zu verwenden statt Vererbung zu betreiben. Die GameStateFactory braucht eine Überarbeitung, um GameStateStrings der APCP-Server in einen neuen GameState oder in ein Update zu überführen. Das wichtigste Argument für den neuen GameState ist die Anwendbarkeit für alle Pokerspieltypen und die Auftrennung in einen Controller und ein Model. Der neue GameState ist eine zentrale Verbesserung des TUD-Pokerframework und der Ausgangspunkt für die Version 2.0.

GameState-Update

Mit dem neuene GameState (bestehend aus dem PokerTable, dem StreetContainer, dem PlayerContainer und dem CardsContainer) wird es endlich möglich alle, GameState-Updates ohne die Erzeugung eines GameStateStrings und der anschließenden Nutzung der GameStateFactory durchzuführen. Die Methoden der GameStateFactory werden einfacher. Im Alberta-Modus wird die Factory nur noch den GameStateString tokenizen und die relevanten Daten an den PokerController übergeben. Der TUD-Server kann auf die Übermittlung von GameStateStrings oder dem kompletten GameState verzichten. Stattdessen könnte der Server spezialisierte Nachrichten versenden. Ein Beispiel wäre eine Nachricht für ausgeteilte Karten. Sinnvoll wären auch Nachrichten mit Commands an die PokerClients wie resetBot oder resetGameState.

Erweiterungen für den GameState und LegacyInterface

Der GameState sollte nach seinem Grundaufbau noch einige Erweiterungen bekommen. Die wichtigste Erweiterung ist ein LegacyInterface, welches für alte Pokerbots alle Methoden der GameState aus den Versionen 1.X bereitstellt. Eine andere Idee ist es, die Flexibilität zu erhöhen durch das Einbinden eines Containers für Custommethoden. Das könnten Lambda-Ausdrücke sein oder ein eigenes Interface, welches genau für diesen Zweck designed ist. Der Bot-Entwickler bekommt eine Möglichkeit den GameState ohne Vererbung und Wrapper zu erweitern. Dadurch wird die Wiederverwertbarkeit von Features erhöht, weil die zusätzlichen Methoden nur in dem Container zusammengeführt werden brauchen. Analog dazu könnte auch ein Container für Customfelder hinzugefügt werden oder alternativ kann auch der Custommethodencontainer verallgemeinert werden. Der Zweck ist es, Vererbung und den Bedarf für Decorator Klassen zu verringern; womit die Modularisierung für den GameState abgeschlossen wäre.

PokerClient

Es liegen nur Ideen vor, einen neuen PokerClient zu schaffen. Er wird in jedem Fall den PokerController und den neuen GameState nutzen. Die Verbindung zum Nachrichtenzustellsystem oder Alberta-Client sollte weiterhin durch gegenseitiges observieren geschaffen werden, weil beides asynchron arbeitende Threads bleiben werden. Es steht noch zur Diskussion, ob der PokerController ähnliche CustomFelder wie der GameState erhalten soll oder derartige Felder im PokerClient vorgesehen werden sollen. In jedem Fall steigt die Übertragbarkeit von Features und Modulen bei der Verwendung von derartigen Strukturen. Vielleicht ist es dadurch sogar möglich, Pokerbot ohne Erben von PokerClients zu erschaffen. Die Aussicht wäre Pokerbots zu schaffen indem mehrere PokerClients vereinigt werden und man nur eine neue Aktionplanung zu entwerfen braucht. Dieses Vorhaben könnte sich als mächtiges Hilfsmittel erweisen komplexe Pokerbots zu erstellen. Es ist zu überlegen, ob eine weitere Schnittstelle für spezielle Logger nützlich ist. Für ACPC-Tuniere sind Begrenzungen des Zeit- und Speicherverbrauchs einzuhalten. Diese Bedingungen könnten durch Logdateien festgehalten werden.

5.11.2 Dealer

Die Klasse Game übernimmt in den Versionen 1.X die Aufgabe des Dealers und ist im Prinzip ein Decorator für den GameState, der in einem eigenen Thread seiner Tätigkeit nachgeht. Die Idee des neuen Dealers ist es im PokerController eine Referenz zum Server zu implementieren. Die Strategie APokerClientConfig wird um eine weitere Liste oder HashMap mit Manipulatoren erweitert. Das können wieder Lambda-Ausdrücke sein oder Implementierungen eines speziellen Interfaces. Dort werden Methoden für die Handhabung von eingehenden Nachrichten und die Versendung ausgehender Nachrichten vorgehalten. Diese Methoden sind nur für einen PokerClient zugänglich, der vom Server als Dealer gestartet wurde. Die Spielregeln und Manipulatoren bleiben auf eine Klasse pro Pokerspielvariante begrenzt. Der Dealer ist dann ein PokerClient ohne besondere Intelligenz und wird automatisch mitgeliefert, sobald ein Pokerspieltyp beschrieben worden ist.

5.11.3 Server

Für den Server existieren Anforderungen, die noch nicht von Konzepten abdeckt sind. Der in den Versionen 1.X enthaltene Server besteht aus veraltetem JAVA-Quellcode, bei dem besonders die Threadverwaltung deprecated ist. Er ist nicht steuerbar während des Betriebs. Es können in ein laufendes Match keine Beobachter hinzugefügt oder entfernt werden, andernfalls bricht der Server das Match ab. Die Liste der Anforderungen an einen Server:

- Eine erneuerte Threadverwaltung wird gebraucht
- Zusammenarbeit mit dem ServerCommander für eine modulare Steuerung
- Ein Konfigurations-Interface ähnlich dem des PokerController für Servereigenschaften wie Zeitlimits oder Tuniermodi
- Mehrere Dealer gleichzeitig verwalten
- Dealermengen für Tuniere
- Persistierung von Matches
- Pseudozufallszahlengeneratoren für reproduzierbare Matches

- Wahlweise direkte Anbindung von PokerClients an den Server mittels Observer oder einer Verbindung über das auf TCP/IP basierende Nachrichtenzustellsystem
- Eine flexiblere Anmeldung von Beobachterclients an die laufenden Matches

Damit würde eine modulare Grundlage für einen Server geschaffen werden, der sich besser für den Einsatz mit GUIs und veränderbaren Spielbedingungen eignet.

5.11.4 Benutzerschnittstellen

Die bestehende Spiel- und Beobachtungsoberfläche ist fest auf Texas Hold Em ausgelegt. Diese sollten den neuen GameState konsumieren, um möglichst viele Pokerspieltypen darzustellen, ohne Anpassungen vornehmen zu müssen. Die Panelstruktur ist zweckmäßig; dadurch lassen sich eigenständige Oberflächen schaffen oder die Panels werden in eine Gesamt-GUI eingesetzt. Analog wäre ein Panel für den ServerCommander-Client erstrebenswert, um diesen auch wahlweise als eigenständiges Programm zu starten oder in einer Gesamt-GUI, wie bei Max Bank, einzubinden. Zusätzliche Konsolen-Clients könnten auch in Betracht gezogen werden. Für die Spieloberfläche sollte weiterhin an der Imitation eines Pokerbots aus der Sicht des Servers festgehalten werden.

5.12 UCT Bäume für TUD-Pokerframework 2.0

Das TUD-Framework 2.0 kann eine direkte Unterstützung für Spielbäume liefern. Diese Zielsetzung ist nur auf Konzeptebene bearbeitet worden. Die Idee ist, ein Interface n-Tree für die ActionObject Klasse zu entwerfen. Das letzte ActionObject hält n-Nachfolger von ActionObjects, um verschiedene Bet-Sizes-Buckets abzubilden. Im Fall von Fixed-Limit werden nur drei Nachfolger verwendet. Der Controller erhält erweitert e Precondiations, Manipulatoren und Postconditions. Immer wenn ein Spielzug getätigt wurde, wird entweder aus der aktuellen Wurzel das zutreffende Kind ausgewählt und als nächstes ActionObject verwendet oder wenn kein Kind zutreffend war ein neuer Baum erzeugt. Optional können bei einem zutreffenden Kind weitere Äste zugefügt werden oder auch die Tiefe des Baumes erhöht werden. Damit ist zumindest das Datenmodell erklärt. Es handelt sich um das Composite-Pattern (Gamma u. a., 1995, S.195ff). Die Simulationen stellen ein eigenes Thema dar, aber haben zumindest eine gemeinsame Datenstruktur, welche diese direkt vergleichbar macht.

6 Testergebnisse und Evaluation des TUD-Pokerbot-Framework 1.8

6.1 Legacybots im Test

Die Test-Matches hatten 500 Hände und liefen im Duplicated-Modus. Grundsätzlich wird mit der Eigenschaft „Version 1.8 kompatibel“ angezeigt, dass der Pokerbot ein Fixed Limit Vergleichsgegner ist. Trotzdem ist auf Fehler zu achten. Pokerbots die Spielregeln verletzen, könnten unter falschen Vorraussetzungen eigene Spielzüge planen. Falsche Voraussetzungen könnten dann entstehen, wenn ein Update des inneren Pokerbot-Zustands geschieht, bevor der Spielzug vom Dealer bestätigt wurde. Fehlerhafte Pokerbots müssen nicht unbedingt schlecht spielen. Die Berechnungsdauer ist in diesem Testlauf nicht so wichtig, sie soll nur eine Notiz am Rande sein.

Tabelle 6.1.: Testläufe der Legacybots auf dem TUD-PokerFramework 1.8 gegen TunedRichieRich

Legacybot	Dauer	Autor	kompatibel	Fehler
FatTony	2:25:41	Johannes Dorn	ja	verletzt Cap-Regel
PerspectiveFutureStatsTrainingBot	0:24:13	Theo Kischka	ja	Datenzugriffsexceptions
SelfObservingBot	-	Theo Kischka	-	-
LearnedTunedRichieRich	-	Theo Kischka	-	-
FixedTransitionEnsembleBot	-	Tobias Thiel	nein	Laufzeitfehler
MajorityVoteEnsembleBot	0:15:14	Tobias Thiel	ja	
RandomEnsembleBot	0:14:08	Tobias Thiel	ja	selten: wird fold zu check
StatisticsEnsembleBotMax	0:11:20	Tobias Thiel	ja	-
StatisticsEnsembleBotMaxDraw	0:12:28	Tobias Thiel	ja	selten: fold zu check
StatisticsEnsembleBotMaxDrawAction	0:12:18	Tobias Thiel	ja	selten: fold zu check und Cap-Regel
StatisticsEnsembleBotProp	0:11:50	Tobias Thiel	ja	selten: fold zu check und Cap-Regel
StatisticsEnsembleBotPropSig	0:13:43	Tobias Thiel	ja	selten: Cap-Regel
StatisticsEnsembleBotPropSigAction	0:13:20	Tobias Thiel	ja	-
FatBetty	3:11:43	Victor Negoescu	ja	selten Cap-Regel

Eine kurze Erläuterung zu den Fehlern: „fold wird zu check“ bedeutet, dass der Dealer aus einem fold ein check gemacht hat, ganz analog zu den Regeln der ACPC. „Cap-Regel“ besagt, das ein fünfter raise in einer Bietrunde vom Dealer zu einem call umgewandelt wurde.

6.2 ArizonaStu gegen ArizonaStu - Version 1.0 gegen Nolimit 2013

Hier traten der Pokerbot ArizonaStu (KEmpfer) mit dem Nolimit 2013 Framework, wie er zur ACPC 2013 eingereicht wurde gegen ArizonaStu mit einem Framwork-Update auf Version 1.0 gegeneinander an. Es ist ein Vergleich zwischen den Frameworks Nolimit 2013 und Version Version 1.0 einschließlich gewisser Verbesserungen aus 1.3. ArizonaStu ist für beide Frameworks gleich eingestellt. Da ArizonaStu mit Nolimit 2013 keine Unterstützung für den TUD-Server bietet wurde ein lokaler Server durch das Programm „dealer“ der ACPC-Ausrichter eingerichtet. Mit diesem Testlauf wird der Client-Bereich der beiden Frameworks miteinander verglichen. Ab Version 1.0 sind die Spielregeln der ACPC in den Klassen PokerClientLimit und PokerClientNolimit direkt implementiert. Das bedeutet insbesondere für Nolimit Pokerbots: zu kleine Bets werden auf den Min-Raise angehoben, um einen serverseitigen call zu verhindern. Bei Nolimit 2013 wurden falsche Züge nicht gänzlich vom PokerClient korrigiert; diese Fehler werden vom ACPC Server regelmäßig als call ausgelegt wird.
Die Weiterentwicklung des Frameworks hat sich gelohnt sich. Denn ArizonaStu hält nun mit Version 1.0 zwangsweise die

Tabelle 6.2.: Match-Ergebnisse Version 1.0 gegen Nolimit 2013

Position 1	Chips 1	Chips 2	Position 2
ArizonaStu2013	-27323	27323	ArizonaStu2014
ArizonaStu2014	356159	-356159	ArizonaStu2013
ArizonaStu2014	477212	-477212	ArizonaStu2013
ArizonaStu2013	-48951	48951	ArizonaStu2014

Spielregeln der ACPC ein und gewinnt gegen die alte Frameworkversion. Es wurden in den Matches jeweils 5000 Hände gespielt mit Seed 0. Zwischen den Matches wurden die Sitzplätze getauscht und die Pokerbots neugestartet. Dieser Test ist erfolgreich verlaufen.

6.3 Version 1.0 mit Modifikation gegen Version 1.8

Hier trat die Version 1.0 einschließlich gewisser Verbesserungen aus 1.3 gegen die Version 1.8 des Frameworks an. Der Pokerbot ArizonaStu wird verwendet. Die Hauptunterschiede zwischen den Versionen 1.0 und 1.8 sind Bugfixes und Vereinfachungen in der Logik. Es wurde ein leichter positiver Einfluß auf ArizonaStu erwartet. Diesmal ist der auf dem TUD-Pokerserver eingesetzt worden. Die Matches hatten jeweils 5000 Hände und wurden einer Datei hands_20150729-011241_2.txt entnommen. ArizonaStu war für beide Frameworks gleich eingestellt. Danach wurde mit vertauschten Positionen wiederholt. Überraschenderweise hat Framework Version 1.0 gewonnen. Die Dauer ist keine relevante Information und ist nur informationshalber mit aufgeführt. Es konnten keine Spielfehler festgestellt werden. Die Einstellungen der Bots wurden überprüft und geben auch keinen Aufschluß über das Ergebnis.

Tabelle 6.3.: Match-Ergebnisse Version 1.0 gegen Version 1.8

Position 1	Chips 1	Chips 2	Position 2	Dauer
ArizonaStu V1.8	-25863	25863	ArizonaStu V1.0	0:15:03
ArizonaStu V1.0	227948	-227948	ArizonaStu V1.8	0:15:48
ArizonaStu V1.8	-51325	51325	ArizonaStu V1.0	0:14:59
ArizonaStu V1.0	247210	-247210	ArizonaStu V1.8	0:14:46
ArizonaStu V1.8	-64675	64675	ArizonaStu V1.0	0:14:44
ArizonaStu V1.0	183088	-1830880	ArizonaStu V1.8	0:14:40

6.4 Laufzeitvergleich zwischen Version 1.8 und Limit 2013

Bei diesem Test wurden die PokerClients der Version 1.8 und des Limit 2013 Frameworks miteinander verglichen. Auf den PokerClients der beiden Frameworks saßen jeweils zwei einfache Limit-Raise-Bots auf. Diese überbieten sich solange bis die Cap-Regel greift und die Hand zu Ende ist. Dadurch wurde sichergestellt, das beide PokerClient-Versionen die gleiche Anzahl von Spielzügen durchführen. Es wurde der TUD-Server aus Version 1.8 verwendet und 3000 Hände pro Match gespielt. Aus Gründen der Fairness wurde die Fehlerprotokollierung des Games auskommentiert, weil das Limit 2013 Framework vom Server der Version 1.8 ausgebremst werden würde. Es wurde erwartet, dass beide Matches ungefähr gleich schnell sein würden. Die beiden RaiseBots, die auf dem PokerClientLimit der Version 1.8 aufsitzen, haben zwei Geschwindigkeitsnachteile: Die ein- und ausgehenden Daten müssen eine Vererbungsschicht zusätzlich passieren und sie prüfen ihre eigenen Spielzüge vor dem Absenden auf Spielregelkonformität. Dem stehen eine einfachere Kommunikationsschicht und Codeoptimierungen gegenüber. Der Server und das Game wurden auch soweit dies möglich war verbessert. Diese Verbesserungen finden für alle Pokerbots Anwendung. Das Ergebnis ist noch am Rande dessen was erwartet wurde. Die Version 1.8 ist langsamer als Limit 2010. Dafür bietet Version 1.8 mehr Sicherheit beim Einhalten der Spielregeln. Zudem ist der Quellcode sicherer, es wurden viele Compiler-Warnungen beseitigt, was doch eher zu langsamerer Ausführungsgeschwindigkeit führt als zu einer Beschleunigung. Gemessen an der durchschnittlichen Handanzahl pro Sekunde sind beide Frameworks viel zu langsam, um beim Kuhn-Poker mithalten zu können. Die Hand-Rate müsste sich um 2 Größenordnungen verbessern. Genauere Aussagen lassen sich erst treffen, wenn an die Pokerbots Logger-Prozesse angeschlossen werden. Die Logger sollten das GameState-Update an der Kommunikationsschnittstelle und den PokerClienten beim Senden observieren, um die Stoppuhr präzise zu bedienen. Bei diesem Vorgehen wird die Messung einen möglichst geringen Einfluß haben.

Tabelle 6.4.: Match-Ergebnisse Version 1.8 gegen Limit 2013

	Version 1.8	Limit 2010	Differenz	Version 1.8 in Hände/s	Limit 2010 in Hände/s
1	1433s	1279s	154s	0,477	0,426
2	1562s	1264s	298s	0,506	0,421
3	1447s	1243s	204s	0,483	0,414
4	1411s	1260s	151s	0,470	0,420
Durchschnitt	1463,25s	1261,5s	201,75s	0,488	0,4205

7 Ausblick

An einem Framework gibt es immer etwas zu tun. Bis zur Vollendung des TUD-Pokerframework 2.0 bleiben bleibt noch eine Reihe anderer Aufgaben zu untersuchen und zu implementieren. Ein großes Themengebiet ist das Portieren von Limit-Features zu Nolimit. Die Version 1.8 kann jedenfalls mit Abschluß dieser Arbeit direkt weitergepflegt werden.

7.1 Ausblick in Richtung TUD-Pokerframework 2.0

In Kapitel 5.11 wurde eine neue Version beschrieben, die als TUD-Pokerframework 2.0 bezeichnet wurde, und in der vieles anders umgesetzt werden sollte als bei den Vorgängern. In diesen Ausblick fließen auch die Erfahrungen mit den Ablegerprojekten ein. Die wichtigsten Merkmale des TUD-Pokerframeworks 2.0 sollen im Folgenden hervorgehoben werden, mit den Möglichkeiten, die sich daraus ergeben. Ebenso werden die anzugehenden Aufgaben zusammengefasst.

7.1.1 Spielzustandsrepräsentation

Die Flexibilität und Generalisierung des Spielzustands sollte so großzügig wie möglich sein. Damit wird die Arbeit des Framework-Entwicklers einfacher. Für Pokerbot-Entwickler wird es bequemer mit dem GameState zu arbeiten, weil er leichter zu bedienen sein wird als sein Vorgänger. Der Weg dazu:

- Ein konfigurierbarer GameState und ein konfigurierbarer Controller, die alle Pokervarianten abbilden, sind unerlässlich.
- Der Controller soll für alle relevaten Views Verwendung finden. Zu diesen Views gehören insbesondere PokerClients, Dealer und Benutzeroberflächen.
- Eine Lehre, die aus den Ablegerprojekten gezogen werden sollte: Vererbung über zentrale Elemente wie dem GameState macht die Wiederverwertbarkeit von Features schwierig. Der GameState und der Controller sollen ohne Vererbung mit weiterer Funktionalität ausgestattet werden können. Also wird eine Schnittstelle für Pluggins gebraucht.
- Die Objekte des GameState bilden die Poker-Domäne in echten Objekten und brauchbaren Containern ab. Angestrebt wird, dass der User des Pokerframeworks intuitiv verstehen kann, wie der GameState aufgebaut ist und bereits aufbereitete Spielinformationen bereitstehen. Damit sollen auch mehrfache Implementierungen von ähnlichen Methoden in den verschiedenen Projekten verringert werden.
- Anzustreben ist die Kompatibilität zu der alten Spielzustandsrepräsentation durch eine Legacy-Schnittstelle.
- Die Spielzustandsrepräsentation braucht ein möglichst schmales Interface als Fassade. Dieses Interface ermöglicht die Implementierung einer alternativen Spielzustandsrepräsentation, die kompatibel zum PokerController ist. Die Tests aus Kapitel 6.4 zeigen, dass absolut minimalistische Lösungen für Kuhn-Poker gefragt sind. In diesem Spielmodus sind Millisekunden sehr viel Zeit.

Erst diese Eigenschaften werden andere Pokervarianten als Texas Hold Em überhaupt ermöglichen. Das TUD-Pokerframework 2.0 wäre mit den genannten Vorschlägen genau darauf ausgelegt, diese Leistungen zu erbringen. In Zukunft wird es dann leichter möglich sein, Features von zukünftigen Projekten in das Framework einzubinden, weil keine Vererbung mehr erlaubt ist. Die Unterstützung beim Programmieren eines Pokerbots wird sich verbessern. Neue Spieltypen werden bereits durch eine Implementierung der abstrakten Klasse AStrategy realisiert, die eine neue Konfiguration für den PokerController ist (Gamma u. a., 1995, S.315ff). Die begonnene Implementierung des TUD-Pokerframework 2.0 weiterzuführen, bietet sich als direkt an diese Thesis anknüpfendes Projekt an.

7.1.2 Server und Nachrichtenzustellsystem

Das Nachrichtenzustellsystem muss für die neuen Anforderungen durch Kuhn-Poker (100 Hände pro Sekunde) so schlank wie möglich gehalten werden.

- Server und Nachrichtensystem brauchen eine Trennung von Steuerdaten für den Server, Beobachterinformationen und Spielerkommunikation. PokerClients dürfen keinen Einfluss auf den Server haben.

- Die Theadsteuerung des Servers ist stark veraltet und benötigt wegen bereits enthaltenen deprecated Methoden Überarbeitung.

- Der Server kennt keine Zustände, die von einem Steuerungsunterprogramm wie ServerCommander abgefragt werden könnten. Ohne Serverzustände wird das Ausschließen von temporär illegalen Methoden nahezu unmöglich.

- Der Server braucht ein Steuerungsunterprogramm, weil er derzeit fast komplett im Batchbetrieb läuft. Mit einem Steuerungsunterprogramm wird es möglich, nach dem Matchstart Einfluss auf den Server zu nehmen. Sinnvolle Steuerbefehle können dann den Betriebsablauf ändern oder gezielt Informationen zum Betrieb des Servers einholen.

- Die PokerClients sollten durch ein Server-Kommando zu reseten sein, wie im Abschnitt über die EnsembleBots gezeigt wurde. Damit werden Steuerungsnachrichten an die PokerClients notwendig.

Diese Verbesserungen sind angegangen worden, aber erst TUD-Pokerframework 2.0 wird allen Anforderungen gerecht sein. Mit Fertigstellung eines neuen Servers und dem ServerCommander wird es möglich, effektive Spielsteuerungsclients zu schreiben und zu benutzen. Ohne das Resetten der Pokerbots ist zwar ein Testbetrieb mit Duplicated-Matches möglich. Jedoch werden die duplizierten Hände zurzeit noch einfach dem Match angehängt. Pokerbots mit Gedächtnissen, wie eine Gegnermodellierung oder eine Spielzustandshistorie, werden nicht zurückgesetzt. Würden diese Bots zurückgesetzt werden, wären die Testbedingungen passender für Wettkämpfe der ACPC, bei denen die Pokerbots immer nach jedem Match neugestartet werden, um das Gedächtnis zu leeren. Möglicherweise führen verbesserte Testumgebungen auch zu besseren Tunierergebnissen.

7.1.3 User Interface für Version 2.0

Eine überarbeitete Spieloberfläche, die möglichst viele Pokertypen inklusive eines Beobachtermodus unterstützt, wird fällig. Genauso wäre eine neue Serveradministrations-GUI zu implementieren. Erstrebenswert wäre es sicherlich für den Benutzer, die Wahl zu haben, beide GUI jeweils als Standalone-Programm oder in einem gemeinsamen Programm zur Verfügung zu haben. Konsolen-Clients könnten alternative Überwachungs- und Steuerungswege darstellen. In jedem Fall gibt es auch auf dem Gebiet der Benutzerschnittstellen unerledigte Arbeit.

7.1.4 TUD-Pokerframework 2.0 vollenden

Derzeit führt kein Weg an weiterer Arbeit am Framework vorbei. Viele Funktionalitäten der untersuchten Pokerbots könnten in das alte Framework eingepflegt werden, jedoch wäre das Ergebnis nahezu unwartbar, da sämtliche Bereiche eher mit mehr Funktionen gepatched werden würden, weil die Struktur nicht für solche Maßnahmen gedacht ist. Die Aussicht ist hierbei eine neue, elegantere Pokerbotgeneration, die möglicherweise stärker von Vorgängerarbeiten profitieren kann. Die angefangene Version 2.0 wäre ein Ausgangszustand für dieses Projekt.

7.2 TUD-Pokerframework von Version 1.8 auf 1.9 heben

Die Vererbung von GameState zu den Klassen GameStateLimit und GameStateNolimit könnte schon bald wieder aufgehoben werden. Hierzu könnte der Weg über ein Interface führen, wie in Kapitel 5.10.2 beschrieben. Danach wäre der GameState nicht mehr abstrakt. In den PokerClient-Klassen würde eine kleine Abänderung genügen: Statt des Pfades der GameState-Klasse wird der Pfad der Strategie in der AccountLoginMessage übermittelt. Aber auch die PokerClient-Erben PokerClientLimit und PokerClientNoLimit könnten wieder zu einer Klasse vereinigt werden.

Wie in Kapitel 5.5.3 dargestellt, wäre eine Untersuchung erstrebenswert, ob die Felder isHero und position in der Player-Klasse wirklich mit dem Modifier final ausgestattet sein müssen. Falls dies nicht nötig ist, könnten die GameStates direkt vor dem Senden an einen Spieler manipuliert werden. Dann wäre es ausreichend, den GameState direkt nach dem Einstellen der Holecards, des Hero-Status und der Position, in einer SetGameStateUpdateMessage zu versenden.

Sind die beiden vorherigen Schritte erledigt, dann wäre es an der Zeit, die neuen Updatemöglichkeiten auch der Alberta-Kommunikationsschnittstelle zur Verfügung zu stellen. Sowie eine Code-Erneuerung bei dieser Kommunikationseinrichtung. Seit Anhebung des Frameworks auf JAVA 1.8 und der Weiterentwicklung zur Version 1.8 (Legacy-Bot-Unterstützung) funktioniert diese Kommunikationseinrichtung nicht mehr korrekt. Die letzten Tests hierfür fanden auf Version 1.0 statt.

Grundsätzlich gibt es viele Stellen bei denen der Quellcode bezüglich der Ausführungsgeschwindigkeit noch weiter optimiert werden könnte. Zur Verständlichkeit würden mehr Kommentare und JAVA-Docs sehr hilfreich sein.

7.3 Überarbeitung der Ensemblebots

Eine Überarbeitung der Ensemblebots wäre anzustreben. Die Aufgaben wären dann, EnsembleBots auf Nolimit zu heben und die Infrastruktur zu vereinfachen. Die Packagestruktur müsste nochmals geprüft werden, um gegebenenfalls Frameworkanteile und Pokerbotbestandteile sauber zu trennen, damit die Frameworkanteile in das TUD-Pokerframe übernommen werden können. Eine umfassende Evaluation mit Limit und Nolimit Pokerbots sollte durchgeführt werden. Der Quellcode ist in der Limitversion nicht vollständig kommentiert und mit einigen Java-Docs versehen. Das Hinzufügen von JUnit-Test wäre hilfreich, um später noch zu verstehen wie die Ensembles funktionieren und um kontrollieren zu können, ob Quellcodeänderungen schädlich waren.

7.4 Überarbeitung von FatTony

Ein weiterer Vorschlag wäre FatTony für Nolimit zu überarbeiten und zu prüfen, ob neuronale Netzwerke ein guter Ansatz für Nolimit-Pokerbots sind oder wenigstens brauchbare bet-Sizes liefern könnten.

7.5 Standart-Module entwerfen

Dem Framework könnten parametrisierbare Module hinzugefügt werden. Beispielsweise ein generelles Gegnermodell für Ring und Heads-Up Spiele, was für Limit und Nolimit Poker tauglich ist und das Spielzüge in die Kontexte Commoncards, Position, Betsizes und Holecards einordnet. Wünschenwert sind auch parametrisierte Ranger zur Einschätzung von Villain-Holecards unter Beachtung der bereits getätigten Spielzüge. Üblicherweise betrachten Ranger für diesen Zweck nur die bekannten Karten. Speziell für Nolimit wird ein Bucketing-Verfahren gebraucht, welches bet-Sizes des Gegners bewertet.

7.6 Aspektorientiertes Programmieren und Lambda-Ausdrücke

Nachdem Lambda-Ausdrücke sich als sehr hilfreich bei der Framework-Entwicklung erwiesen haben, stünde auch eine Untersuchung bezüglich der Verwendbarkeit bei Pokerbots an. Anonyme Methoden könnten nützlich für Regel-basierte Pokerbots sein. Aspektorientiertes Programmieren hat noch keine Verwendung im Framework gefunden. Dabei ist vor allem an ausgefeilte Logging-Systeme zu denken, um beispielsweise den Zeitverbrauch von PokerClients zu messen.

8 Fazit

Die Modularisierung des TUD-Pokerframeworks hatte die Zielvorgabe, bestehende Ablegerprojekte zu einem wohl strukturierten Framework zusammenzuführen. Ein Teil der Integrationsaufgaben wurde erfüllt und durch die eingehende Beschäftigung mit dem Framework wurden viele weitere Aufgaben entdeckt. Das wichtigste Ziel wurde verwirklicht: Das TUD-Pokerframework ist nun für Texas Hold Em Limit und Nolimit verwendbar. Der Nachweis für die Funktionstüchtigkeit ist erbracht worden durch Test-Matches am ACPC-Server und am TUD-Server.

In dieser Arbeit wurde eine Sichtung und eine Klassifizierung der im TUD-Pokerframework bisher eingesetzten Module und der von anderen Autoren kreierten Verbesserungsvorschläge vorgenommen (siehe die Übersicht über die Versionen in Abschnitt 4). Bereits bestehende Erfahrungen im Echtbetrieb und gezielt durchgeführte Testläufe haben den Gang dieser Untersuchung ebenfalls beeinflußt. Nach konkreten Programmierungen und vielen Tests befindet sich das Poker-Framework nach Abgabe dieser Arbeit auf dem Level der Version 1.8. Diese Arbeitsergebnisse bezüglich der einzelnen Versionen sind in der Abgabeversion gebündelt worden: Version 1.0 Zusammenführung der Nolimit- und Limit-Frameworks (wie bereits erwähnt); Version 1.2 Vereinfachung des Netzwerkverkehrs; Version 1.3 Verbesserte GameStateUpdates; Version 1.6 eine Standalone GUI als Spieloberfläche für Limit und Nolimit; Version 1.7 ein Beobachtermodus für die Spieloberfäche; Version 1.8 zufriedenstellende Unterstützung alter Pokerbots und der eingebundenen Module verschiedener Projekte durch Version 1.4.

Die Erneuerung ist bei weitem noch nicht abgeschlossen. Mit dieser Arbeit sind einige Verbesserungen erzielt worden. Darüber hinaus konnte ein geordneterr Überblick über die nach Abschluss dieser Thesis noch offenen Aufgaben im Framework geschaffen werden, so dass eine strukturierte Weiterarbeit möglich ist. Die Bearbeitung und Skizzierung der offenen und teilweise erfüllten Ziele (Version 1.1 Einbindung von Kuhn-Poker; Version 1.4 Übernahme von Projektmodulen; Version 1.5 eine Serversteuerung und Version 2.0 Umbau bestehender Module) wurde deshalb Wert gelegt. Die Thesis bildet darum eine reichhaltige konzeptionelle Grundlage für weitere integrative Arbeiten am TUD-Pokerframework 2.0.

Die Modularisierung des TUD-Pokerframeworks stellte sich als schwieriger heraus als erwartet. Bevor die Modularisierung wirklich angegangen werden konnte, waren Wartungsarbeiten zu erledigen. Die Gesamtheit aller Ablegerprojekte und Vorgänger ist ziemlich groß. Die vielen Ablegerprojekte sind alle sehr unterschiedlich implementiert. In Kapitel 5.6.8 wird auf die Problematik der Modulübernahme eingegangen. Viele Features und Module sind nicht in einem Zustand, diese direkt einzupflegen. Die Erfahrungen aus den Code-Reviews der Ablegerprojekte sollten in die neue Version des Frameworks einfließen. Die wichtigsten Erfahrungen zielen auf die Beachtung der Möglichkeiten zur Erweiterung und Flexibilisierung der Hauptbestandteile, wie auf den Spielzustand zu achten und Schnittstellen für Erweiterungen festzulegen.

Es empfiehlt sich bei der Arbeit an einem Framework, einzelne Aspekte mutig neu zu implementieren und mit Tests und Dokumentation auszustatten. Im Verlauf dieser Bachelor-Thesis wurde lange auf die Erhaltung der bisherigen Frameworkstruktur gesetzt, was einige wesentliche Ziele wie Kuhn-Poker zu ermöglichen, zwar nicht unmöglich macht, aber leider zu keiner zufriedenstellenden Lösung geführt hätte. Eine mutige Neuimplementierung von Toplevelmodulen birgt zwar neue Risiken auf der Systemebene, hat jedoch den Vorteil, dass das zeitraubende Auffinden und Deuten von Einzelfehlern in komplexen Strukturen vermieden wird. Die Neuimplementierung mit Unit-Tests von Frameworkaspekten hätte schneller zum Ziel geführt, als ein Framework, das einem Zweck (Fixed Limit Texas Hold Em) dient, auf andere Zwecke zu übertragen (Kuhn Poker und Texas Hold Em Nolimit). Die dabei entstehenden Fehler sind zu meist offensichtlich, aber das fehlerhafte oder nicht angepasste Detail zu finden war ermüdend. Glücklicherweise ergaben die Tests gegenüber Nolimit 2013 Fortschritte in der Frameworkunterstützung für die Pokerbots.

In Kapitel 1.3 wurde das sekundäre Ziel der allgemeinen Wartungsarbeiten erwähnt. Viele Compilerwarnungen wurden beseitigt. Der Quellcode wurde vereinfacht, abgesichert, erneuert, dokumentiert oder auch für gut befunden. Es wurden zusätzliche Methoden für Pokerbot-Entwickler hinzugefügt. Auch dieser Aspekt sei erwähnt, denn ihm wurden viele Stunden teilweise mühsamer Bearbeitungszeit gewidmet, die nur schlecht in einem Text darstellbar sind. Erst diese Wartungsarbeiten hinter den Fassaden haben ein tieferes praktisches Verständnis für den Aufbau dieses Frameworks und den allgemeinen Aufbau von verteilten Systemen erbracht.

Für die nächste Computer Poker Competition steht nach dieser Arbeit ein in den skizzierten Punkten überarbeitetes TUD-Pokerframework 1.8 zur Verfügung. Der Weg zu Version 1.9 und Version 2.0 wurde im Kapitel Ausblick aufgezeigt und parallel dazu wurde mit der Implementierung von Version 2.0 begonnen.

A Design-Pattern

In dieser Arbeit wurde eine Vielzahl von Design-Pattern aufgeführt und angewendet. Die Design Patterns wurden dem Buch Design Patterns: Elements of Reusable Object-Oriented Software Gamma u. a. (1995) entnommen, wo sie vertieft beziehungsweise ausführlich dargestellt sind.

A.1 Creational Patterns

Das ist Klasse von Pattern, die Objekte erzeugt.

- Abstract Factory: Produziert verschiedene Objekte einer Vererbungshierarchie oder eines Interfaces. Hierzu werden der Factory Einstellungen übergeben und diese übernimmt die Verantwortung, Objekte mit den gewünschten Eigenschaften zu liefern. Wird in der Klasse GameStateFactory angewendet. (Gamma u. a., 1995, S.87ff)

- Factory Method: Produziert fertig eingestellte Objekte auf der Vererbungshierarchie einer Klasse. Üblicherweise keine eigene Klasse sondern eine statische Methode einer Klasse. Wird in den Klassen PokerTable und StreetContainer verwendet. (Gamma u. a., 1995, S.107ff)

- Singleton: Stellt sicher, dass nur ein Objekt einer Klasse in einem Prozess existiert. Wurde beim MultiplePokerClientManager verwendet. (Gamma u. a., 1995, S.127ff)

A.2 Structural Patterns

Diese Pattern geben Programmen eine verständliche Struktur und sorgen maßgeblich für die Wiederbenutzbarkeit von bereits funktionierenden Klassen.

- Adapter (oder auch Wrapper): Übersetzt Methoden und Daten von einer Repräsentation in eine andere. Die eigentlichen Adapter-Methoden sind üblicherweise seiteneffektfrei. Ein Adapter kann ein Interface ersetzen. Sie werden in den Ablegerprojekten oftmals eingesetzt, um den GameState in geeigneter Weise anzusprechen. (Gamma u. a., 1995, S.139ff)

- Composite: Ein Container der Komponenten (Blätter) und Kompositien (Knoten) enthält. Kompositien verhalten sich wie Blätter, falls diese wie ein Blatt angesprochen werden, aber können weitere Kompositien oder Komponenten enthalten. Dieses Pattern soll die ActionObjects erweitern für den Aufbau von UCT-Bäumen in Version 2.0. (Gamma u. a., 1995, S.163ff).

- Decorator: Eine Klasse wird in eine andere Klasse eingesetzt, um Funktionalität zu erhalten. Der GameState wird häufig dekoriert. Decorator kann prinzipiell Vererbung ersetzen. Die Klasse Seat dekoriert Player am PokerTable von Version 2.0. Der GameState wird oftmals von Botentwicklern dekoriert. (Gamma u. a., 1995, S.175ff)

- Facade: Subsysteme werden hinter einem Interface oder einem Objekt verborgen. Es werden komplizierte Interaktionen fast wörtlich hinter einer Fassade versteckt. Das Pattern wird in Version 2.0 oftmals benutzt: StreetContainer, PokerTable und der PokerController bilden Fassaden. (Gamma u. a., 1995, S.185ff)

- Flyweight: Häufig auftretende kleine gleiche Objekte, werden nicht mehr jedes Mal neu instanziiert, sondern durch einen Pool von Repräsentaten ersetzt - es wird die gleiche Referenz benutzt. Der GameState von Version 2.0 verwendet einen gemeinsamen Pool von Player-Objekten in den Klassen StreetContainer, StreetObject, ActionObjet und PokerTable. (Gamma u. a., 1995, S.195ff)

A.3 Behavioral Patterns

Solche Designs ändern das Verhalten von Objekten ab. Zumeist wird in das abzuändernde Objekt ein anderes eingelagert, das dann über eine Schnittstelle angesprochen wird.

- Command: Clients senden Befehle an einen Server. Im Server werden die Befehle durch eine Klasse dargestellt in der mindestens eine Methode zur Ausführung des Befehls enthalten ist. Es können Unterobjekte eingelagert werden, das Verhalten der Klasse zu ändern. Die Befehle können stationär an eine Referenz gebunden sein oder einem ausführenden Controller übermittelt werden. Dieses Konzept wird beim ServerCommander und beim PokerController eingesetzt. (Gamma u. a., 1995, S.233ff)

- Observer: Viele Beobachter warten auf eine Veränderungsmeldung eines Objekts. Nun gibt es zwei Möglichkeiten: Das observierte Objekt übermittelt an alle seine Beobachter die Veränderung oder die Beobachter treten bei aktuellem Interesse an das Object heran und entnehmen die benötigten Daten selbständig. Dieses Pattern wird zwischen PokerClients beziehungsweise dem Server und deren Netzwerkzugängen eingesetzt, um ein- und ausgehende Nachrichten weiterzureichen. Für GUIs ist dieses Design-Patter fast unersetzbar. (Gamma u. a., 1995, S.293ff)

- Strategy: Eine Klasse hält eine Schnittstelle offen, um durch die in der Schnittstelle festgelegten Methoden ein dahinterliegendes Objekt anzusprechen. Je nach Situation kann dieses Objekt ausgetauscht werden, damit ein anderes Verhalten beim Aufruf von Schnittstellenmethoden herbei geführt werden kann. Beispielsweise setzt der PokerController solch eine Strategie ein. Es können in ihm Konfigurationen für die verschiedenen Pokerspieltypen abgelegt werden. (Gamma u. a., 1995, S.315ff)

- Visitor: Ein Klasse von Objekten passiert andere Objekte, die durch ihre Methoden den Zustand des Besuchers verändern. Es ist damit möglich Objekte zu ändern ohne diese zu erweitern. Der PokerController und die APokerClientConfig-Klassen werden vom GameState besucht und durch Lambda-Ausdrücke der APokerClientConfig-Klassen verändert. Der GameState braucht für die verschiedenen Pokertypen nicht geändert zu werden. (Gamma u. a., 1995, S.331ff)

B Verwendung des TUD-Pokerframework 1.8

Dieser Anhang soll den Einstieg in das Entwickeln und Testen von Pokerbots mit dem TUD-Pokerframework 1.8 erleichtern.

B.1 Erstellung eines einfachen Nolimit Texas Hold Em Heads Up Pokerbots

Ein einfacher Texas Hold Em Nolimit braucht nur die Erbschaft von PokerClientNoLimit: Durch die Erbschaft von PokerClientNoLimit ist durch den impliziten Konstruktor von PokerClientNoLimit bereits alles Relevante eingestellt. Die vorliegenden Spielzustände sind von der Klasse GameStateNoLimit. Die GameStateFactory ist bereits mit der Instanziierung eines PokerClientNoLimit konfiguriert worden und produziert Objekte der Klasse GameStateNoLimit.

```
package pokertud.clients.nolimit;

import pokertud.clients.framework.PokerClientNoLimit;

public class TutorialBot extends PokerClientNoLimit {

        @Override
        public void handleGameStateChange() {
                // TODO Auto-generated method stub
        }

        /**
         * @param args
         */
        public static void main(String[] args) {
                // TODO Auto-generated method stub

        }
}
```

In Version 1.8 muss zwingend handleGameStateChange() implementiert werden. Wichtig ist der Test, ob der Hero (also der eigene Bot) am Zug ist; dieser Test wird durch den aktuellen GameState mit der Methode isHeroActing durchgeführt. Im Fall isHeroActing == true wird der Aufruf von sendAction(int action) erwartet. Desweiteren brauchen PokerClients üblicherweise den ClientRunner zum Starten. Es genügt normalweise die Argumente des Programmstarts an den ClientRunner zusammen mit einer Instanz eines PokerClients zu übergeben. Die Methode runClient(String[] args, PokerClient client) stellt die Kommunikationsschnittstelle ein und wertet Argumente bezüglich der Spieloberfläche aus.

```
public class TutorialBot extends PokerClientNoLimit {

        /**
         * @param args
         */
        public static void main(String[] args) throws   UnknownHostException {
                        ClientRunner.runClient(args, new TutorialBot());
        }

        @Override
        public void handleGameStateChange() {

                if (this.getGameState().isHeroActing()){
                        this.sendAction(0);
                }
}
```

Die nächste Stufe des TutorialBots soll nicht nur check und call durch this.sendAction(0) tätigen. Es werden durch einen Zufallszahlengenerator mit 20% eine -1 und jeweils mit 40% eine 0 oder eine größere Zahl erzeugt. Die 0 bildet

auf check/call ab, die -1 auf fold und >0 sind bets. Speziell für die bets bei Nolimit-Bots ist es wichtig, die min-bet einzuhalten. Dafür stellt der PokerClientNoLimit die Methode getMinBet(GameStateNoLimit gameState) zur Verfügung. Für die Wettbewerbe der ACPC ist es wichtig, die min-bet Regeln einzuhalten, sonst werden zu kleine Bets als call ausgelegt. Mit jeder falschen Bet wird entweder eine Chance verpaßt, einen Villain zum Folden zu bringen, oder einen größeren Pot zu gewinnen (je nach Absicht).

```
public class TutorialBot extends PokerClientNoLimit {

        /**
         * @param args
         */
        public static void main(String[] args) throws   UnknownHostException {
                        ClientRunner.runClient(args, new TutorialBot());
        }

        Random rand= new Random();
        private int command, betsize=0;

        @Override
        public void handleGameStateChange() {

                if (this.getGameState().isHeroActing()){
                        this.command = this.rand.next(-1+5 \% 3);
                        this.betsize = this.getMinBet(this.getGameState());
                        this.betsize += this.getGameState().getPotsize() \% command;
                        if(command<1)
                                this.sendAction(command);
                        else
                                this.sendAction(this.betsize);
                }
}
```

Die Methode sendAction(int actioncode) ist final im PokerClientNoLimit implementiert. Die Methode prüft vor dem Absenden, ob die min-bet Regeln eingehalten wurden, falls nicht dann wird die bet auf die min-bet-size angehoben. Im Falle eines folds wird ermittelt, ob ein check möglich wäre und das günstigere Ergebnis wird abschickt. Ob eine bet-size größer ist als der Stack wird getestet und gegebenenfalls auf diesen begrenzt. Damit ist das Senden einer Action sicher bezüglich der Spielregeln. Der Bot-Entwickler kann ungeprüfte Spielzüge senden, jedoch muß der Bot-Entwickler Anweisungen zur Überprüfung von Spielzügen verwenden, immer dann wenn die Spielzüge noch vor dem Senden (exakt: vor der Antwort des Dealers) in Historien, Gegnermodellen oder Simulationen verwendet werden sollen, um während der Wartezeit Berechnungen anzustellen.

B.2 Erstellung eines einfachen Limit Texas Hold Em Ring-Game Pokerbots

Die Texas Hold Em Limit Pokerbots werden ganz ähnlich zu den Nolimit-Bots erschaffen. Diese erben von PokerClient-Limit. Der implizite Konstruktur stellt das Nötige auf dem PokerClient ein. Das betrifft wieder besonders die GameState-Factory. Genauso wie Tutorialbot für Nolimit ist der ClientRunner für den Start unerlässlich. Die isHeroActing() Methode wird in jedem Fall gebraucht, um den eigenen Spielzug zu ermitteln.

```
package pokertud.clients.limit;

import pokertud.clients.framework.PokerClientLimit;

public class SqeezeBot extends PokerClientLimit {

        @Override
        public void handleGameStateChange() {
                if (this.getGameState().isHeroActing()){
                        this.sendCall();
                }
        }
```

```
        /**
         * @param args
         */
        public static void main(String[] args) {
                ClientRunner.runClient(args, new SqueezeBot());
        }

}
```

Es folgt ein etwas klügerer Pokerbot. Der SqueezeBot wird um eine Regel namens isSqueeze() erweitert. Ein Squeeze bedeutet, der Spieler vor einem selbst hat eine bet gecallt und man selbst erhöht. Da wenigstens 2 weitere Spieler aktiv sind, lohnt sich dieses Vorgehen. Mit der Mettode isSqueeze() soll gezeigt werden, wie mit den Listen der Actions (Bietrunden) umzugehen und der PlayerState einzusetzen ist.

Eine zweite Regel verwendet ein einfaches Gegnermodell, das für jeden Spieler eine Kennzahl über folds im PreFlop als erste Action führt. Die Regel lautet dontPlaywithNit - nicht mit Geizhälsen spielen.

Der Bot berechnet im Normalfall seine Chancen mit Hilfe des MetricsCalcalutor. Es wurde eine Metric gewählt, welche die Chancen der eigenen Holecards mit den bereits bekannten Commoncards gegen unbekannte Holecards berechnet.

```
package pokertud.clients.limit;

import pokertud.clients.framework.PokerClientLimit;
import pokertud.*;

public class SqeezeBot extends PokerClientLimit {
        MetricsCalculator mcalc = new MetricsCalculator();
        HashMap<String, Double> opponentModel = new HashMap<String, Integer>();
        int opponentModelRoundCounter = 0;

        @Override
        public void handleGameStateChange() {
                if (this.getGameState().isHeroActing()){
                        if(isSqueeze()){
                                this.sendraise();
                        }
                        else if(dontPlaywithNit){
                                this.sendFold();
                        }
                        else{
                                double ihr = mcalc.getImmediateHandRank(
                                                this.getGameState().getHero().getHolecards(),
                                                this.getGameState().getBoard(),
                                                this.getGameState().getActivePlayerCount(),
                                                10000);
                                if(ihr < 0.15)
                                        this.sendFold();
                                else if(ihr< 0.3)
                                        this.sendCall();
                                else
                                        this.raise();
                        }
                }
                else if(this.getGameState().isFinished()){
                        this.opponentModellRoundCounter++
                        this.updateOpponentModel(this.getGameState());
                }
        }

        /**
         * @param args
```

```
         */
        public static void main(String[] args) {
                ClientRunner.runClient(args, new SqueezeBot());
        }

        private boolean isSqueeze(){
                if(this.getGameState().getCurrentStreetAction().size()>1){
                        if(this.getGameState().getLastActingPlayer()
                                        .getPlayerState.hasCalled()){
                                if (this.getGameState().getCurrentStreetAction()
                                                .get(this.getGameState()
                                                .getCurrentStreetAction()
                                                .size()-2).getPlayerState.hasBetted()){
                                        return true;
                                }

                        }
                }
        return false;
        }

        private boolean dontPlaywithNit(){
                for(Player player : this.getGameState().getPlayers()){
                        if(!player.isHero() && player.hasBetted()){
                                return (double)
                                    this.opponentModel().get(player.getName) /
                                    this.opponentModelRoundCounter < 0.04;
                        }
                }
        }

        private updateOpponentModel(GameState gs){
                ....
        }
}
```

Das Update des OpponentModels soll hier nicht im Vordergrund stehen. Es folgt nun der Schritt zur Modularisierung. Für einen Pokerbot ist es ratsam, ein eigenes Package anzulegen. Module, die eine eigene Funktionalität darstellen, sind am besten auch in eigene Packages einzulagern, wie es hier mit dem OpponentModel und den Rules geschehen ist. Die eigentliche Auswahllogik des Pokerbots wurde in squeezeBotLogic gekapselt, welche eine Klasse aus dem Package squeezebot ist. Eine saubere Trennung hilft bei der Übernahme von Modulen bei späteren Arbeiten.

```
package pokertud.clients.limit;
import pokertud.clients.framework.PokerClientLimit;
import pokertud.*;
import squeezebot.opponentmodel;
import squeezebot;
import squeezebot.rules;

public class SqeezeBot extends PokerClientLimit {
        private SqeezeBotLogic squeezeBotLogic = new SqueezeBotLogic();
        private OpponentModel opponentModel = new opponentModel();

        @Override
        public void handleGameStateChange() {
                if (this.getGameState().isHeroActing()){
                        SqueezeBotRule rule =
                            this.squeezeBotLogic.decideRule(this.getGameState(),
                            this.opponentModel);
                        this.sendAction(rule.getAction);
```

```
                }
                else{
                        this.opponentModel.updateOpponentModel(GameState);
                }
        }

        /**
         * @param args
         */
        public static void main(String[] args) {
                ClientRunner.runClient(args, new SqueezeBot());
        }
}
```

B.3 Testspiele der vorgestellten Pokerbots innerhalb des Frameworks

Der Server kann durch eine Argument-Übergabe beim Starten einige wenige Einstellungen vornehmen:
<Dateiname || Anzahl der Hände> <Duplicated-Match-Modus> <Port>

- <Dateiname || Anzahl der Hände> Mit dem Dateiname können Hände aus einer Datei geladen oder mit der Angabe einer Zahl, die Anzahl der zu spielenden Hände. Der Pfad kann relativ oder absolut sein. Es sind die Eigenheiten der Betriebsysteme diesbezüglich zu beachten.
- <Duplicated-Match-Modus> Mit der Einstellung -single werden die Hände nur einfach gespielt. Mit -duplicated wird ein Duplicated-Match gestartet.
- <Port> Die Portnummer ist optional, es ist jedoch darauf zu achten, dass der Port auch wirklich frei ist. Wird kein Port gewählt kommt der Defaultport 35676 zum Einsatz.

Sind wenigstens zwei Spieler mit dem Server verbunden, dann reicht die Eingabe von „s“ an der Serverkonsole. Der TUD-Server prüft automatisch, ob die Spielerkonstellation gestartet werden darf. Nach dem Spielen wird eine Logg-Datei mit den Händen im Projektpfad angelegt.

Ein Beispiel für einen Serverstart:
- 3000 -duplicated -20000

Der Server wird ein Match mit 3000 Händen im Duplicated-Match-Modus starten. Für Pokerbots ist der Server auf seiner derzeitigen IP-Adresse am Port 20000 zu finden.

B.4 Testspiele auf lokalem ACPC-Server

Die University of Alberta stellt ihre Serversoftware auf der Hompage der ACPC Computer-Poker-Competition (2014c) zum Download zur Verfügung. Das Programm „dealer“ liegt als Quellcode in der Programmiersprache C vor. Durch den Aufruf des makefile wird es für Linux-Betriebssysteme kompiliert. Im Projekt-Verzeichnis der Version 1.8 ist bereits ein Kompilat für Ubuntu-Linux enthalten. Das executable-flag muss vor dem Erstgebrauch gesetzt werden. Nach jetzigen Erkenntnisstand wird empfohlen, dieses Programm im Root-Verzeichnis des Projektes zu starten.

Der Aufruf hat folgende Struktur:
dealer<tag> <config> <# runs> <seed> <name1> <name2>....<name n>

1. <tag> gibt dem Spiel einen Namen, der auch für die Benennung der Loggdateien benutzt wird.
2. <config> enthält die Einstellungen des Servers. Es werden die Dateien holdem.nolimit.2p.reverse, holdem.limit.2p.reverse und holdem.limit.3p.reverse mitgeliefert, welche den ACPC-Turnierbedingungen entsprechen.
3. <# runs> legt die Anzahl der zu spielenden Hände fest.
4. <seed> ist der Startwert für den Pseudozufallszahlengenerator, um Matches reproduzieren zu können.

5. <name1> <name2>... <name n> sind die Namen, welche nach Anmeldereihenfolge vergeben werden sollen.

Weitere Optionen können dem Programm durch einen Aufruf ohne Argumente entnommen werden.
Der Server startet mit dem Match, sobald alle Pokerbots mit diesem vernetzt sind. Wenn die Port-Option nicht genutzt wird, zeigt „dealer“ auf dem Terminal die zu verwendeten ports an. Im Default-Modus wird eine Loggdatei für jedes Match angelegt.
Ein Bespiel für einen Aufruf von „dealer“:
./dealer local1 holdem.nolimit.2p.reverse_blinds.game 100 2 p1 p2 -p 30000,30001 -q
Das Match heißt local1 und es wird Texas Hold Em Nolimit mit der Spielregel Reserse Blinds gespielt. 100 Hände werden vom Pseudozufallsgenerator mit Startzahl 2 generiert. Die Spieler heißen p1 und p2 und bekommen die Ports 30000 und 30001 zugewiesen. Die Option -q unterdrückt die Anzeige des Spielverlaufs, aber nicht etwaige Fehlermeldungen.
Zuletzt noch ein Hinweis: Es können eigene Server-Konfigurationen geschrieben werden. Die vorhandenen Konfigurationen sind ziemlich selbsterklärend.

B.5 Testspiele der vorgestellten Pokerbots gegen die ACPC-Gegner

Im Download Paket für Serversoftware liegt auch der Quellcode für weitere Testsoftware bei. Das Programm „bm_run_matches“ liegt ebenfalls im Quellcode vor und wird zusammen mit „dealer“ durch das „makefile“ kompiliert. Vor der Benutzung braucht das Programm das executable-flag. Nach dem kompilieren liegt ein Linux-Programm vor.

Eine Anmeldung an den Remote-User-Account des ACPC-Host-Servers erfolgt durch das Secure Shell Protokoll. Dort werden die Loggdateien der Testspiele abgelegt und dort liegen auch die Serverkonfigurationen, die auch mit dem Programm “dealer“ geliefert wurden.
sudo ssh -v -i <private Key der KE-Gruppe> -l <Username> 54.166.192.196
Falls der eigene Computer beim ACPC-Host-Server bekannt gemacht wurde, dann läßt sich alternativ ein Dateimanager verwenden:
nautilus ssh://<Username>@54.166.192.196/home/<Username>

Hinweis: Der private key muss normalerweise im Betriebssystem Ubuntu-Linux registriert werden, um ihn benutzen zu können. Bei anderen Linux-Distributionen ist die Dokumentation zu konsultieren.

Ein weiterer Vorbereitungsschritt ist zu erledigen: Für den Aufruf von “bm_run_matches“ wird ein Shellscript für den zu startenden Pokerbot benötigt. Dieses hat den Zweck, die IP-Adresse und den Port des Servers an den Pokerbot weiterzuleiten.
Es wird hier als Beispiel das Shellscript für den EVBot gezeigt:

```
java -Xmx2000m -cp lib/pokerserver.jar:bin/:lib/commons-math-2.1.jar:lib/swt-linux.jar
    \
pokertud.clients.nolimit.EVBot $1 $2
```

1. java - Ist der Aufruf der JAVA-Virtual-Machine.
2. -Xmx2000m der JAVA-VM sollen 2000 Megabyte RAM-Speicher zur Verfügung gestellt werden.
3. -cp ist signalisiert der JAVA-VM, dass Klassenpfade folgen.
4. Die Klassenpfade sind Linux konform. Bibliotheken pokerserver.jar, commons-math2.1.jar und swt-linux.jar werden vom TUD-Pokerframework zur Zeit (Stand Ende Juli 2015) verwendet und müssen stets angegeben werden. Weitere Bibliotheken des Pokerbots müssen hier auch angegeben werden. Das Verzeichnis “bin“ enthält üblicherweise die Kompilate des Pokerbots.
5. An dieser Stelle wird der Pfad zur main-Methode des Pokerbots angegeben. Im Beispiel: “pokertud.clients.nolimit.EVBot“
6. $1 ist die vom Pokerbot zu nutzende Server-IP-Adresse.
7. $2 reicht den Server-Port weiter an den Pokerbot.

Alle PokerClients werden vom ClientRunner eingestellt, welcher die Argumente verarbeitet. Das Shellscript benötigt das executable-flag.

Nun ist alles vorbereitet für das eigentliche Testspiel gegen ehemalige ACPC-Teilnehmer. Der Grundaufruf von bm_run_matches: ./bm_run_matches <bm_hostname> <bm_port> <username> <pw>

1. <bm_hostname> ist die IP-Adresse des ACPC-Host-Servers. Bis Ende 2015 wird das die IP-Adresse 54.166.192.196 sein.
2. <bm_port> der Port des ACPC-Host-Server ist 54000.
3. <username> Logindaten sind anzufordern.
4. <pw> Logindaten sind anzufordern.

Wird hinter das Passwort “games“ angehängt, dann werden alle ehemaligen Teilnehmer nach Pokertyp aufgelistet.
Start des Testspiels:
bm_run_matches <bm_hostname> <bm_port> <username> <pw> run <Pokertyp> <local script> <# runs> <tag> <seed> <player1> <player2>...
Es werden die weiteren Argumente nach dem Passwort vorgestellt:

- run ist der Befehl für Testspiele.
- <Pokertyp> es stehen zur Auswahl 2pl für Heads Up Limit, 2pn für Heads Up Nolimit und 3pl für einen Ring dreier Spieler.
- <local script> das Shellscript für den Pokerbot-Aufruf kommt an diese Stelle.
- <# runs> gibt die Anzahl der zu spielenden Matches an.
- <tag> damit wird der Name der Match-Serie festgelegt. Die Loggdateien der einzelnen Matches werden nach dem „tag“ und einer fortlaufenden Nummer benannt.
- <seed> Die Starteinstellung des Pseudozufallsgenerators des Kartengenerators.
- <player N> Ist entweder der Name eines Pokerbots aus der Server-Liste oder der Eintrag local (der Platzhalter für das Shellscript).

Ein Beispiel:
./bm_run_matches 54.166.192.196 54000 <Username> <Passwort> run 2pn startArizonaStu.sh 1 Testlauf20 2 KEmpfer local
Die Matches können mehrere Stunden dauern, denn die Leitung ist in diesem Fall nicht nur im sprichwörtlichen Sinne wirklich lang. Bei einem engen Test-Zeitplan sind Verbindungsabbrisse einzuplanen. Der Testlauf ist entweder zu wiederholen oder können bei noch laufenden Server-Prozessen, durch den „rerun“ Befehl noch zu Ende, geführt werden.

B.6 Spiel Mensch gegen Bot

Die Klasse pokertud.clients.swingclient2015.StandaloneClient wird hierzu gestartet. Im Wesentlichen handelt es sich um die Spieloberfläche von Max Bank mit einigen Änderungen bezüglich des Starts und der Tatsache, dass nun damit auch Texas Hold Em Nolimit gespielt werden kann. Die Argumente sind folgende:
<IP des Servers> <Port des Servers> <Kommunikationsschnittstelle> <-play> <Pokertyp>
Die Bedeutung der Argumente:

- <IP des Servers> Die IP-Adresse des Servers in numerischer Darstellung wird immer erkannt. Die textuelle Auflösung ist plattformabhängig.
- <Port des Servers> Der Port des Servers wird in nummerischer Darstellung erwartet.
- <Kommunikationsschnittstelle> Mit -tud wird ein TUD-Server signalisiert und mit -alberta ein ACPC-Server.
- <-play> Die Einstellung -play läßt den Menschen spielen.
- <Pokertyp> Die Einstellung -selflimit startet das Programm im Limit-Modus und analog dazu -selfnolimit.

Die Steuerungselemente der Spieloberfläche sind für Pokerspieler selbsterklärend. Im Zweifel sollte die Lektüre des Kapitels 2.1 (Grundlagen des Pokerspiels) ausreichen, um Hände gegen die Pokerbots zu bestreiten.
Ein Beispiel: 127.0.0.1 35676 -tud -play -selfnolimit
Die Bedeutung des Beispiels ist: Die GUI-Pokerclient soll sich mit einem TUD-Server, der auf dem selben Computer läuft auf dem gerade die Spieloberfläche gestartet wird, verbinden. Der Server-Dienst ist auf dem Defaultport 35676 zu suchen. Es wird Nolimit Texas Hold Em gespielt.

B.7 Team Mensch und Bot gegen Bot

Die Klasse pokertud.clients.swingclient2015.StandaloneClient wird auch für diesen Modus genutzt. Der Startvorgang ist ähnlich wie beim Spiel Mensch gegen Bot.
<IP des Servers> <Port des Servers> <Kommunikationsschnittstelle> <-play> <-withbot> <Pfad zum Pokerbot>

- <IP des Servers> Die IP-Adresse des Servers in numerischer Darstellung wird immer erkannt. Die textuelle Auflösung ist plattformabhängig.
- <Port des Servers> Der Port des Server wird in nummerischer Darstellung erwartet.
- <Kommunikationsschnittstelle> Mit -tud wird ein TUD-Server signalisiert und mit -alberta ein ACPC-Server.
- <-play> Die Einstellung -play läßt den Menschen spielen.
- <-withbot> Die Verwendung von -withbot signalisiert, dass ein PokerClient anwesend sein wird.
- <Pfad zum Pokerbot> Hier wird der Packagepfad zu der main-Methode des Pokerbots übermittelt. Der Pokerbot muss in Version 1.8 ein Erbe von PokerClientLimit oder PokerClientNolimit sein.

Die Spieloberfläche startet wie gewöhnlich, jedoch mit einem Unterschied: Die Steuerelemente haben einen zusätzlichen Knopf, mit dem der Vorschlag des Pokerbots ausgeführt werden kann.

Abbildung B.1.: Die Beschriftung des rechten Knopfes zeigt den Vorschlag des EVBots an. Es ist ein Reraise auf 850 Chips.

Der Vorschlag von EVBot wurde akzptiert und an den Server gesendet.
Ein Beispiel sieht folgendermaßen aus:
127.0.0.1 35676 -tud -play -withbot pokertud.clients.nolimit.arizonastu
Die Spieloberfläche soll sich am selben Computer mit einem TUD-Server auf dem Defaultport vernetzen. Der Pokerbot ArizonaStu soll als Berater gestartet werden. Die Auswahl des Pokertyps erfolgt hierbei implizit durch den gewählten „Berater“.

Abbildung B.2.: Der Vorschlag von EVBot hat ArizonaStu zum Folden gebracht.

B.8 Beobachtungsmodus

Auch für diesen Modus wird die Klasse pokertud.clients.swingclient2015.StandaloneClient gestartet. Im Vergleich zum Spielmodus werden zwei Argumente geändert.
<IP des Servers> <Port des Servers> <Kommunikationsschnittstelle> <-spectate> <-Pokertyp>

- <IP des Servers> Die IP-Adresse des Servers in numerischer Darstellung wird immer erkannt. Die textuelle Auflösung ist plattformabhängig.
- <Port des Servers> Der Port des Server wird in nummerischer Darstellung erwartet.
- <Kommunikationsschnittstelle> Mit -tud wird ein TUD-Server signalisiert und mit -alberta ein ACPC-Server.
- <-spectate> Die Einstellung -spectate läßt den Menschen beobachten.
- <Pokertyp> Die Einstellung -limit beziehungsweise -nolimit startet das Programm im jeweiligen Modus. An dieser Stelle ist Option -withbot nicht vorgesehen und führt zum Abbruch des Startvorgangs.

Dem Beobachter wird der Pokertisch vollständig einsehbar.
Hinweis: In der jetzigen Implementierung von Version 1.8 läuft das Match weiter, wenn sich ein Beobachter abmeldet.
Ein Beispiel für die Beobachtung eines Limit Texas Hold Em Matches am Localhost mit dem Defaultport des TUD-Servers: 127.0.0.1 35676 -tud -spectate -limit

B.9 Konvertierung der Logdateien der Testspiele für den Pokertracker

Das Programm Pokertracker ist ein professionelles Pokerunterstützungsprogramm. Die Hauptfunktionen des Pokertrackers sind die Spieleranalyse, die Handanaylse, das automatisches Datensammeln über alle Hände die gespielt wurden oder in die Datenbank geladen worden sind. Beim Spielen ist die große Spielerleichterung das Einblenden von statistischen Daten über ein sogenanntes Head-Ups-Display, der am Pokertisch anwesenden Spieler. Diese Pokerunterstützungs-Software ist als einzige zugelassen und zertifizert von den großen Online-Casinos. Es ist eine Bereicherung für die

Spielanalyse. Deshalb wurde anläßlich der ACPC 2013 ein Konverter von ACPC- (und neuerdings auch TUD-Server) Loggdateien geschrieben.

Die Klasse ist unter folgendem Packagepfad zu finden: pokertud.util.converterACPCtoPokertracker.converter
Dieses Programm wird über eine Konsole oder über einen Terminal benutzt. Die Argumentstruktur:

1. <Pfad zu Loggs> Ein relativer oder absoluter Pfad zu einem Verzeichnis mit Loggdateien oder zu einer Loggdatei. Die Eigenheiten der Betriebssysteme bezüglich der Pfade sind zu beachten.

2. <Name des Hero> Die Konvertierung der Loggdateien erfolgt im Hinblick auf den angegebenen Hero.

Die erzeugten Dateien sind im Verzeichnis „Pokertracker“ zu finden, welches dem Projektpfad entsprechend im Dateisystem zu finden ist. Danach kann der Import in den Pockertracker beginnen.

B.10 Die Generierung von Handkartendateien

Handkartendateien sind ein Ersatz für die Handkarten-Erzeugung auf Basis von einstellbaren Pseudozahlengeneratoren. Mit diesen Dateien lassen sich Matches reproduzieren. Im Package pokertud.server ist ein Konsolenprogramm namens generateSpecialMatch enthalten. Es braucht beim Aufruf der main-Methode nur zwei Argumente. Das erste für die Anzahl der Hände und das zweite für die Anzahl der Spieler. Es liefert immer 5 Commoncards zuzüglich 2 Holecards pro Spieler. Die generierten Handkartendateien sind im Verzeichnis „data“ des Projektpfades zu finden.

Literaturverzeichnis

[Bank 2011] Bank, Max: *Eine Grafische Benutzeroberfläche für ein Poker-Spiel*, TU Darmstadt, Knowledge Engineering Group, Diplomarbeit, 2011. http://www.ke.tu-darmstadt.de/lehre/arbeiten/bachelor/2011/Bank_Max.pdf. – Bachelor's Thesis

[Computer-Poker-Competition 2012a] Computer-Poker-Competition, Chairmen: *Technical Details 2014*. Website, 2012. – Online einsichtlich http://www.computerpokercompetition.org/index.php/competitions/technical/100-technical-details-2014; abgerufen am 16. Juli 2015.

[Computer-Poker-Competition 2012b] Computer-Poker-Competition, Chairmen: *Winner Determination: Bankroll Instant Run-off*. Website, 2012. – Online einsichtlich http://www.computerpokercompetition.org/index.php/competitions/rules/82-winner-determination-bankroll-instant-run-off; abgerufen am 8. September 2014.

[Computer-Poker-Competition 2012c] Computer-Poker-Competition, Chairmen: *Winner Determination: Total Bankroll*. Website, 2012. – Online einsichtlich http://www.computerpokercompetition.org/index.php/competitions/rules/80-winner-determination-total-bankroll; abgerufen am 8. September 2014.

[Computer-Poker-Competition 2014a] Computer-Poker-Competition, Chairmen: *2014 Rules*. Website, 2014. – Online einsichtlich http://www.computerpokercompetition.org/index.php/competitions/rules/96-2014-rules; abgerufen am 8. September 2014.

[Computer-Poker-Competition 2014b] Computer-Poker-Competition, Chairmen: *About the ACPC*. Website, 2014. – Online einsichtlich http://www.computerpokercompetition.org/index.php/about; abgerufen am 18. September 2014. Kontakt: chair@computerpokercompetition.org

[Computer-Poker-Competition 2014c] Computer-Poker-Competition, Chairmen: *Annual Computer Poker Competition*. Website, 2014. – Online einsichtlich http://www.computerpokercompetition.org/; abgerufen am 8. September 2014. Kontakt: chair@computerpokercompetition.org

[Computer-Poker-Competition 2014d] Computer-Poker-Competition, Chairmen: *Annual Computer Poker Competition*. Website, 2014. – Online einsichtlich http://www.computerpokercompetition.org/index.php/competitions/results?id=82; abgerufen am 13. Juli 2015. Kontakt: chair@computerpokercompetition.org

[Computer-Poker-Competition 2014e] Computer-Poker-Competition, Chairmen: *Limit Games - 3 player Kuhn Poker*. Website, 2014. – Online einsichtlich http://www.computerpokercompetition.org/index.php/75-limit-games; abgerufen am 8. September 2014.

[Gamma u. a. 1995] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0–201–63361–2

[Kischka 2014] Kischka, Theo: *Trainieren eines Computer-Pokerspielers*, TU Darmstadt, Knowledge Engineering Group, Bachelor-Arbeit, may 2014. http://www.ke.tu-darmstadt.de/lehre/arbeiten/bachelor/2014/Kischka_Theo.pdf

[Thiel 2013] Thiel, Tobias: *Managing a Team of Poker Players*, TU Darmstadt, Knowledge Engineering Group, Bachelor-Arbeit, Mai 2013. http://www.ke.tu-darmstadt.de/lehre/arbeiten/bachelor/2013/Thiel_Tobias.pdf

[Thomas Hartmann 2009] Thomas Hartmann, Markus Z.: Strategy Evaluation - Pokerbotpraktikum / TU Darmstadt, Knowledge Engineering Group. Version: 2009. http://www.ke.tu-darmstadt.de/lehre/archiv/ss09/challenge/StrategyEvaluation.pdf. 2009. – Forschungsbericht

[Tomasz Gasiorowski 2013] Tomasz Gasiorowski, Peter G. Julian Prommer P. Julian Prommer: Praktikum für künstliche Intelligenz - Pokerbotpraktikum / TU Darmstadt, Knowledge Engineering Group. 2013. – Forschungsbericht

[Zopf 2010] Zopf, Markus: *Ein Framework zur Entwicklung und Evaluation intelligenter Pokeragenten*, TU Darmstadt, Knowledge Engineering Group, Diplomarbeit, 2010. http://www.ke.tu-darmstadt.de/lehre/arbeiten/bachelor/2010/Zopf_Markus.pdf. – Bachelor's Thesis