

# Künstliche Intelligenz und Poker

Entwicklung eines intelligenten Moduls zur Verbesserung der Gegnereinschätzung

## Beschreibung von Poker

Poker ist ein Kartenspiel, das mit 52 Karten und mindestens zwei Spielern gespielt wird. Jeder Spieler setzt einen Einsatz auf die Gewinnchance seiner eigenen Hand, der Gewinner eines Spieles gewinnt alle gesetzten Einsätze (*pot*). Es gibt viele unterschiedliche Varianten von Poker, allen gemein ist, dass man letztlich mit fünf Karten eine der folgenden Hände erzielt:

- High Card (höchste Karte in der Reihenfolge der Zahlenkarten (2-10), Junge, Dame, König, Ass)
- One Pair (Paar mit zwei Karten des gleichen Wertes)
- Two Pair (zwei Paare mit jeweils zwei Karten des gleichen Wertes)
- Three Of A Kind (Drilling mit drei Karten des gleichen Wertes)
- Straight (Straße, d.h. fünf aufeinanderfolgende Karten)
- Flush (fünf Karten der gleichen Spielfarbe)
- Full House (Kombination aus einem Drilling und einem Paar)
- Four Of A Kind (vier Karten des gleichen Wertes)
- Straight Flush (Straße in einer Spielfarbe)
- Royal Flush (10, Junge, Dame, König, Ass in einer Farbe)

Für die Umsetzung der Aufgabe wurde die Pokeragenten-Plattform verwendet, die von Andreas Korol und Bent Witthold im Rahmen ihrer Diplomarbeit an der Hochschule Bremen entwickelt wurde. Die Architektur der Software ist modular aufgebaut, sodass die Funktionalität im Nachhinein durch das Einbinden weiterer Module ergänzt werden kann. Diese Anbindung wurde von den Entwicklern explizit als großer Vorteil der Software bewertet und lässt viel Raum zur weiteren Entwicklung.

Die Plattform bietet zwar die Möglichkeit verschiedene Pokervarianten zu nutzen, bisher wurde aber ausschließlich Texas Hold'em implementiert. Bei dieser Pokervariante erhält jeder Spieler zu Beginn zwei Karten verdeckt (*Hole Cards*) und es werden zuerst drei (*Flop*), dann jeweils eine (*Turn & River*) Karte offen in der Mitte aufgedeckt (*Common Cards*). Dazwischen finden Wettrunden statt, bei denen jeder Spieler abwarten (*check*) kann, falls noch nicht gesetzt wurde, oder aber selbst erhöhen (*raise*), mitgehen (*call*) oder aussteigen (*fold*) kann. Eine Wettrunde ist erst beendet, wenn alle Spieler entweder mitgegangen oder ausgestiegen sind. Am Abschluss der letzten Runde decken die übriggebliebenen Spieler ihre Karten auf, um die höchste Hand und so den Gewinner zu ermitteln (*showdown*).

# Architektur der Software

Die von Korol und Witthold entwickelte Software basiert auf einer OpenSource-Pokerplattform mit Client/Server-Architektur<sup>1</sup> für reale Spieler, nutzt aber statt des GUI-Clients einen KI-Client. Dieser Client besteht aus den vier Grundpaketen *bot*, *game*, *org* und *util*. Das *bot*-Paket ist wiederum unterteilt in *bot.config*, *bot.module* und *bot.talk*. Letzteres ist ausschließlich für die Kommunikation mit einem Server, insbesondere dem ACPC-Glassfrog-Server, zuständig. Im Hauptpaket Bot gibt es einen oder mehrere ausführbare Pokerbots, die alle von der abstrakten Klasse *bot.Bot* abgeleitet sind und nur aus der *main*-Methode, einer Methode *getBet()* und einer Methode *isPlayable()* besteht. In der *main*-Methode wird die im *bot.config* beschriebene Konfigurationsdatei geladen, *isPlayable()* fragt ab, ob der Pokerbot für die vom Server vorgegebene Pokervariante ausgelegt ist, und *getBet()* führt die eigentlichen KI-Algorithmen aus, die es dem Bot ermöglichen zu entscheiden, welche Aktion er als nächstes ausführt, also quasi die Spielerentscheidung übernimmt.

Die abstrakte *bot.Bot*-Klasse enthält ein *game.State*-Objekt, welches die Spielstände des aktuellen Wettbewerbs speichert, ein *bot.talk.Talk*-Objekt, welches den Server aus Clientsicht repräsentiert (oder die Clientdaten an den Server übermittelt), implementiert das Interface *bot.IF\_Bot\_Talk*, welches dem *bot.talk.Talk*-Objekt und damit dem Server ermöglicht den Spielstand des Clients zu aktualisieren (oder die Serverdaten empfängt) und enthält eine Liste der verwendeten Module. Module erweitern die Möglichkeiten eines Bots um intelligente Fähigkeiten. Im Paket *bot.module* befinden sich drei zentrale Schnittstellen auf die alle KI-Module basieren und anhand derer Korol und Witthold damit sozusagen die Grundaufgaben des Pokerspiels identifizieren:

- *bot.module.IF\_GetEquity* dient zur Einschätzung der Stärke der eigenen Hand. Ein erfahrener Pokerspieler nutzt zur Einschätzung der eigenen Hand das Mittel der *PotOdds*:
  - *PotOdds* = Verhältnis zwischen Einsatz und möglichen Gewinn (*pot*)
  - *Outs* = Anzahl der zur Verbesserung der aktuellen Hand fähigen Karten
  - *Odds* = Wahrscheinlichkeit seine bisherige Hand zu verbessern  
z.B. *Odds* nach dem Turn =  $Outs / 46$  (unbekannte Karten)  
 $Odds$  nach dem Flop =  $(Outs/47) + (Outs/46) - (Outs^2/47*46)$
  - Wenn *Odds* (Gewinnchance)  $\geq$  *PotOdds* (relativer Einsatz), dann *call/raise*
  - Wenn *Odds* (Gewinnchance)  $<$  *PotOdds* (relativer Einsatz), dann *fold**PotOdds*-Berechnungen funktionieren alledings nicht für ein einzelnes Spiel, sondern nur über viele Spiele hinweg (wie bei allen Wahrscheinlichkeiten). Korol und Witthold gehen stattdessen den Weg statistisch zu berechnen, welchen Anteil des Pots ein Spieler in vergleichbaren Situationen gewinnt. Dieser Anteil wird *equity* genannt.
- *bot.module.IF\_GetEstimation* dient zur Einschätzung der Spielweise der Gegner. Dabei lassen sich beispielsweise Anspielhäufigkeit (*tight*, spielt selten mit - *loose*, spielt häufig mit) und Aggressivität (*passive*, neigt eher zu *check/call* - *agressive*, neigt eher zu (*re-raise*) oder die Häufigkeit des Bluffens bestimmen. Hierunter fiele auch eine realistische Einschätzung der Karten des Gegners.

---

<sup>1</sup>ACPC (Annual Computer Poker Competition), <http://www.computerpokercompetition.org/>, bieten eine eigene Pokerplattform für reale Spieler mit dem Server Glassfrog, dem GUI-Client Swordfish, sowie einer eigenen Kommunikationsschnittstelle.

- *bot.module.IF\_GetProposal* unterbreitet dem Spieler einen Aktionsvorschlag, z.B. basierend auf *StartingHandCharts*, die vor Ausgabe des Flops anhand der Spielerposition, des Einsatzes und der eigenen Karten, eine Aktionsempfehlung geben.

Darüber hinaus gibt es zwei Module, die kein Spielverhalten, aber zusätzliche Funktionen aktivieren:

- *game.gamestorage.IF\_SetGameStorage* ist ein Modul zum Speichern der Spielstände über einen Wettbewerb hinaus, sodass auch Spielerstatistiken über mehrere Wettbewerbe hinweg ausgewertet werden können. Dieses Modul ist nur für Texas Hold'em implementiert und speichert die Daten in einer MySQL-Datenbank.
- *extmodule.IF\_ExtModule\_Feed* lädt zusätzlich externe Module. Implementiert ist das Laden von C-Bibliotheken/Modulen über Java Native Interfaces (JNI), was für das Equity-Modul verwendet wird.

## Aufgabenstellung der Projektgruppe

Als Teilgruppe am Modulprojekt Poker haben wir uns zum Ziel gesetzt, die Gegnereinschätzung mit intelligenten Methoden zu verbessern. Slansky, ein bekannter Pokertheoretiker, beschreibt als Grundgesetz des Pokers, dass man so spielen soll, als würde man die Karten der Gegner kennen. Spielt ein Spieler seine Hand anders, als wenn er die gegnerischen Karten kennen würde, nützt das seinen Gegnern. Spielt dagegen der Gegner anders, als er es mit Wissen der eigenen Karten tun würde, dann nützt es dem Spieler. (Slansky, 2005, S. 15ff.)

Das Grundgesetz des Pokers besagt demnach, dass ein Spieler erfolgreich ist, wenn er die gegnerischen Karten gut analysieren und die eigenen Karten gut verschleiern kann.

Gegnereinschätzung ist also eine zentrale Aufgabe des Pokerspiels. Dabei gibt es unterschiedliche Ansätze: Zu einem kann es sich um eine möglichst realistische Einschätzung der verdeckten Spielkarten des Gegners handeln (*Hole Cards*), zum anderen kann eine möglichst genaue Einschätzung der nächsten Aktion des Gegners erfolgen.

Gegnereinschätzung beinhaltet auch den grundsätzliche Spielstil des Gegners oder aber die frühzeitige Erkennung von Bluffs.

Im Rahmen unseres Projekts haben wir uns vorgenommen, die bestehende Architektur und Datenstruktur der Software zu benutzen und diese durch zusätzliche Methoden zu erweitern, damit die Software ihre Entscheidungen noch intelligenter treffen kann. In der Software ist bereits ein Estimation-Modul vorhanden, aber nicht vollständig implementiert.

## Das Estimation-Modul

Eine erfolgreiche Gegnereinschätzung ist für den Erfolg im Poker von besonderer Bedeutung. Wenn man es schafft in beliebiger Situation vorherzusagen, wie sich ein Gegenspieler verhalten würde, bedeutet das eine deutliche Verbesserung der Gewinnchancen für den Bot. Der menschliche Spieler analysiert ebenfalls ständig das Verhalten seines Gegners. Dabei berücksichtigt er sowohl das unterschiedliche Verhalten in bestimmten Phasen des Spiels, als auch eine allgemeinen Einschätzung des Gegners. Dies ermöglicht dem Spieler eigene Entscheidungen differenziert auf die Verhaltensweisen des Gegner abstimmen zu können.

Genau dies beabsichtigt das von Korol und Witthold implementierte Estimation-Modul und erlaubt es dem Bot besser an das taktische Verhalten eines menschlichen Spielers heranzukommen.

Das Modul befindet sich im Paket *bot.module.th.estimation*. Ausgangspunkt ist die Klasse *Estimation*, welche die Schnittstelle *IF\_Estimation* implementiert und u.a die Methoden *estimate()* und *getOpponentsEstimations()* besitzt. Während *estimate()* die Gegnereinschätzung durchführt, ruft man mit *getOpponentsEstimations()* die Ergebnisse der Einschätzung ab. Diese sind in der *Opponent*-Klasse gespeichert. Dort befinden sich für jeden Spieler in einem Wettbewerb die Aktionen und Verhaltensweisen, die in den jeweiligen Runden ausgeführt wurden. Jede Runde (*PreFlopShc*, *PreFlop*, *Flop*, *Turn*, *River*) führt dabei eine eigenständige Einschätzung durch, die komplett unabhängig ausgewertet kann. Allen Runden gemein ist die Implementierung von *IF\_EstimationAtRound*, die ein ganzheitliches Konzept zum Einlesen und Auswerten der Daten bietet. Fertiggestellt wurde in der Software von Korol und Witthold allerdings nur die Anbindung für das Analysieren der PreFlop-Karten in der Runde *PreFlopShc*. Hier werden die Handlungen des Gegners mit den quasi objektiven Daten der Starting-Hand-Chart (SHC) verglichen. Diese befinden sich im Paket *bot.module.th.shc* und können bei der richtigen Einstellung in der Konfigurationsdatei auch für das eigene Verhalten in der PreFlop-Phase verwendet werden. Zur Einschätzung wird sich auf die Berechnung der Anspielhäufigkeit und Aggressivität des Gegners konzentriert. Jede Abweichung von der vorgeschlagenen Handlung wirkt sich auf den Pegel dieser Werte auf. Ein *fold* statt eines vorgeschlagenen *check/call/raise* bedeutet z.B. der Spieler ist eher *tight*; *raised* der Spieler statt zu *callen* ist er eher *aggressive*.

Diese Daten sollen dann die Grundlage für die Einschätzung der nächsten Aktion des Gegners bilden. Da ein Spieler aber in verschiedenen Phasen des Spiels sein Verhalten ändern könnte, ist es wichtig sein Verhalten auch überall aufzuzeichnen, um dieses dann bewerten zu können

Als Aufgabenstellung unserer Projektgruppe ergibt sich daraus die Erweiterung des Estimation-Moduls auf alle anderen Spielphasen, um möglichst viele Daten über das Verhalten des Gegners zu sammeln und Entscheidungen auf einer sichereren Einschätzung treffen zu können. Wir leiten die Vorgehensweise aus der *PreFlop*-Phase (Vergleich zwischen "objektiver" SHC und tatsächlicher Aktion) ab und übertragen sie auf die weiteren Phasen des Spiels. Als quasi objektive Daten verwenden wir das Equity-Modul.

## Das Equity-Modul

Das Equity-Modul dient wie im Abschnitt "Architektur der Software" beschrieben zur Einschätzung der Gewinnfähigkeit der eigenen Hand. Als *equity* (eng. u.a. für "Eigenkapitalanteil") wird der Anteil des Pots bezeichnet, den ein Spieler in einer vergleichbaren Situation gewinnt und entspricht damit einer Variante der PotOdds, d.h. es wird in der Klasse *bot.module.equity.SubjectivAllinEquity* die relative Gewinnwahrscheinlichkeit, die sich aus der subjektiven Sicht eines Spielers (Kenntnis bzw. Einschätzung der gegnerischen *hole cards*) ergibt, ins Verhältnis zum *pot* gesetzt, um herauszufinden, welcher Anteil am Gewinn (in Prozent) für den Spieler bei korrekter Einschätzung zu holen ist, falls dieser sein Gesamtvermögen (*all-in*) einsetzt. Dazu werden alle bekannten Informationen des Spielers an

die Methode `bot.module.equity.SubjectiveAllinEquity.getEquity(int,int,Card[[[[]],Card[],Card[]])` übergeben. Alle *hole cards*, die dem Spieler nicht bekannt sind, werden durch einen *BeliefVector* gefüllt und an ein externes Modul zur Berechnung der *equity* weitergegeben, dieses verwendet C-Bibliotheken, die über JNI eingebunden werden. Da diese Bibliotheken für Linux kompiliert sind, lässt sich dieses Modul bisher ausschließlich unter Linux verwenden (für andere Betriebssysteme werden eigenständige Kompilierungen der C-Bibliotheken benötigt, was sich leider nur sehr umständlich realisieren lässt). Der Bot *HSBremen1* verwendet das Equity-Modul dann um zu entscheiden, ob er weiterspielt (*check* oder *call*) oder aussteigt (*fold*). Bedingung dafür ist, dass der berechnete subjektive Gewinn (*equity*-Anteil \* *pot*-Größe) mindestens so hoch sein muss wie der derzeitige Einsatz (*callsize* = Betrag, der gesetzt werden muss, um mitzugehen).

Für die Erweiterung des Estimation-Moduls wird die Entscheidungsfindung des *HSBremen1*-Bots als Vorschlag (objektive Grundlage) für die einzelnen Spielphasen verwendet und um einen Punkt erweitert: Bei der bisherigen Implementierung wird nur zwischen *call* und *fold* unterschieden, sehr gute Gewinnwahrscheinlichkeiten wirken sich bislang nicht auf den Entscheidungsprozess aus. Deshalb wird bei einem Verhältnis von Einsatz zu Gewinn, das größer als 1:4 ist, für das Estimation-Modul ein (*re-*)*raise* vorgeschlagen. Für die Einschätzung des Spielverhaltens in der aktuellen Runde werden wiederum nur die Abweichungen vom Vorschlag gezählt:

- *fold* statt *raise* bedeutet sehr *tight* (doppelt negativ gezählt)
- *call* statt *raise* bedeutet *passive* (einfach negativ)
- *fold* statt *call* bedeutet *tight* (einfach negativ)
- *raise* statt *call* bedeutet *aggressive* (einfach positiv)
- *call* statt *fold* bedeutet *loose* (einfach positiv)
- *raise* statt *fold* bedeutet sehr *loose* (doppelt positiv) und *aggressive* (einfach positiv)

Für die jeweilige Berechnung der Prozentwerte für Anspielhäufigkeit und Aggressivität wird der bisherige Wert innerhalb eines bestimmten zulässigen Änderungsbereichs (Radius) um die relative Änderung (Summe der Änderungen / Anzahl der Spiele) angepasst und anschließend auf den zulässigen Wert von 0 bis 100 beschnitten. Dadurch wirken sich Strategieänderungen nicht sofort auf die Gesamteinschätzung des Spielers auf, sondern passen sich graduell an das tatsächliche Spielverhalten an.

## Alternative Nutzung der OutsAndOdds-Klasse

Das Hauptproblem bei der Nutzung des Equity-Moduls ist die Beschränkung der typischen Plattformunabhängigkeit von Java auf ein Linuxsystem, da die extern eingebundenen C-Bibliotheken nicht in der Javaumgebung, sondern direkt auf dem System kompiliert sind und dort laufen. Eine Übersetzung von Linux-kompilierten Bibliotheken auf Windows ist nur sehr umständlich realisierbar. Alternativ zur Nutzung des Equity-Moduls bietet sich die klassische Berechnung der PotOdds.

Beim Pokerspiel muss ein Spieler seine Pot Odds (Wettchancen im Pot), Odds (Gewinnwahrscheinlichkeit) und Outs (Anzahl der Karten, die die Hand verbessern) berechnen können. Durch das Verrechnen dieser Outs mit der Anzahl der noch nicht aufgedeckten

Karten erhält man die Wahrscheinlichkeiten in einer Situation die beste Hand bekommen zu können.

Dazu nutzen wir die OutsAndOdds-Klasse die von der Teilgruppe um Sascha Ritter und Thomas Kuschmann entstanden ist. Diese Klasse wurde von der Projektgruppe mit dem Bot Nr. 5 (Jenny Bieling, Jaqueline Brinn, Tina Droste) bei der Berechnung der Odds und von uns bei der Berechnung der Outs für einen Straight erweitert und im Estimation-Modul als quasi objektive Datengrundlage eingebunden.

## Mehrwert durch Datenbankanbindung

Gegnereinschätzung ist so zentral für das Pokerspiel, das keine Teilgruppe des Projekts darauf verzichten konnte. Eine einfache Typerkennung für Anspielhäufigkeit und Aggressivität ist auch für Gruppen, die sich hauptsächlich mit dem eigenen Setzverhalten beschäftigt haben, von großer Relevanz, da die Spielweise des Gegners die eigene Strategie stark beeinflusst. Betrachtet ein Spieler jedes Spiel völlig losgelöst von den vorherigen und trifft seine Entscheidungen nur aufgrund der aktuellen Situation, wird er nie in der Lage sein auf intelligente Weise Poker zu spielen, da ein wesentlicher Erfolgsfaktor in der Analyse des Gegners liegt, wie es Slansky im Grundgesetz des Pokers beschreibt. Nur wenn die Spielweise zwischen dem Spiel mit vollständigen und unvollständigen Informationen nahezu deckungsgleich sind, kann ein Poker-Bot auf lange Sicht das Spielniveau eines Menschen erreichen.

Das Problem bei der Gegneranalyse liegt aber vor allem bei der Signifikanz der auszuwertenden Daten. Wahrscheinlichkeiten geben die relative Häufigkeit von Ereignissen über einen längeren Beobachtungszeitraum an und man kann nicht mithilfe von Wahrscheinlichkeiten ein einzelnes Ereignis vorhersagen. Erfolgreiche Gegnereinschätzung beruht demnach auf den Beobachtungszeitraum und der Menge der erhobenen Daten. Wertet man nur die Daten des aktuellen Wettbewerbs aus, sind die Einschätzungen gerade zu Beginn des Wettbewerbs irrelevant und ungenau (da man ohne Vorwissen immer von einem neutralen Spieler ausgeht und sich die ersten Spiele viel stärker auf die Einschätzung auswirken, als spätere) und führen zu fehlerhaften Entscheidungen.

Um dies zu umgehen, verwendet die Software von Korol und Witthold das Gamestorage-Modul zur Speicherung der Spieldaten über mehrere Wettbewerbe. So lässt sich für jeden Spieler, der schon zuvor an einem Wettbewerb teilgenommen hat, auch zu Beginn eines neuen Wettbewerbs eine Einschätzung des Spielertyps geben und die Einschätzung verläuft nicht ganz so ungenau. Das Gamestorage-Modul speichert die Daten in einer relationalen MySQL-Datenbank. Die Speicherung wurde für Texas Hold'em vollständig implementiert, aber die Auswertung erfolgt bisher nur für die *PreFlopShc*-Phase. Korol und Witthold haben dabei die objektrelationalen Abbildungen zum Abrufen und Schreiben der Daten selbst entwickelt. Über einen *DBProjector* lassen sich SQL-Statements erstellen und ausführen und die Ergebnisse in verschiedene einfache und komplexe Datentypen kapseln. Die Implementierung für das Gamestorage-Modul befindet sich im Paket *game.gamestorage.texas.db*, außerdem gibt es im Paket *util.sql* Hilfsklassen zur Umwandlung der Datentypen. Eine der aufwendigsten Aufgaben des Projekts war es die SQL-Anweisungen für die einzelnen Spielphasen zu erstellen. Dies hat zwar primär nichts mit Künstlicher Intelligenz zu tun, die zur Gegnereinschätzung notwendigen KI-Algorithmen basieren aber darauf, dass das verwendete Datenmaterial eine signifikante Aussagekraft besitzt. Dies lässt sich am effizientesten über die Anpassung der bereits vorhandene Schnittstelle zur Datenbank lösen.

# Verwendung des erweiterten Estimation-Moduls

Ein Bot, der unser erweitertes Estimation-Modul benutzt, kann unsere Einschätzungen und Daten verwenden um eine Voraussage machen, welche Aktion der Gegner als Nächstes ausführt. Mit Hilfe der Fuzzyzugehörigkeitsfunktionen werden für jeden Gegner die Wahrscheinlichkeiten für eine bestimmte Aktion zu einem bestimmten Zeitpunkt des Spiels berechnet.

Im Verlauf mehrerer Spiele können viele Verhaltensdaten über die Gegner gesammelt werden. Nach Verlauf mehrerer Spiele werden die Voraussagen über das Verhalten immer genauer, da die Daten über das tatsächliche Verhalten der Gegner die ganze Zeit gespeichert werden. Daraus lässt sich ein Bayesisches Netz generieren, das auch beim Hinzukommen neuer Daten die bedingten Wahrscheinlichkeitsverteilungen für jeden Knoten (jede mögliche nächste Aktion) des Netzes neu berechnet. Durch dieses Lernen des Gegnerverhaltens wird die Voraussage mit jedem gespielten Spiel immer genauer. Hier werden die Vorteile der Benutzung des Equity-Moduls und auch insbesondere der Datenbankanbindung zur Speicherung der Daten offensichtlich. Durch die Einschätzung des Gegners in jeder Phase des Spiels und der zusätzlichen Speicherung aller gesammelten Daten über die Gegner über längere Zeit kann eine viel bessere Gegnertypbestimmung sowie bessere Entscheidungen durch den Bot getroffen werden, als wenn nur die Phase des PreFlops berücksichtigt wäre.

Eine weitere Möglichkeit zur Gegnereinschätzung wäre die Voraussage, die nur darauf basiert herauszufinden, welche Karten der Gegner hat. In diesem Fall würde man den Gegner anhand der möglichen Stärke seines Blattes bewerten und daraus die entsprechenden Verhaltensweisen ziehen. Hier werden also alle Restmöglichkeiten, wie sich ein Blatt entwickeln könnte, ermittelt und die Wahrscheinlichkeiten für jede Möglichkeit berechnet. Bisher findet beim Festlegen vom oben beschriebenen BeliefVector des Equity-Moduls keine Berücksichtigung der Gegnereinschätzung statt. Doch auch das könnte bei der Kartenvoraussage eine Rolle spielen. Durch die Erweiterung des Equity-Moduls mit einer Funktion zur Kartenvoraussage der Gegner erhöht sich die Genauigkeit bei der Einschätzung der eigenen Hand erheblich.

Die erwähnten Methoden der künstlichen Intelligenz dienen also dazu die Gegnererkennung zu verwirklichen. Diese Methoden ermöglichen es die Gegnereinschätzung, die wir entwickelt haben, dafür einzusetzen, um beim Pokerspiel bessere Entscheidungen treffen zu können, also um so zu spielen, als würde man die Karten seiner Gegner kennen.

## Evaluation / Validierung des Ergebnisses

Unser Gegnereinschätzungsmodul wurde in einen Bot integriert, damit dieser auf die Ergebnisse der oben beschriebenen Auswertung zurückgreifen kann. Dafür wurde der Bot "Nr5" der Gruppe um Jenny Bieling, Jaqueline Brinn und Tina Droste so angepasst, dass er auf die Ergebnisse der Auswertung der Datenbank zurückgreifen kann.

Zum Testen der unterschiedlichen Bots wurden folgende Bots nacheinander am Server gestartet:

- HSBremen1: Original-Bot von Korol und Witthold erweitert um den Estimation-Modul, ohne Einfluss auf die Handlung

- Nr5: Original-Bot der Gruppe um Jenny Bieling, Jaqueline Brinn und Tina Droste
- Nr5Revised: Nr5 erweitert um unseren Estimation-Modul, wo an einer Stelle die Handlung des Bots von unseren Estimations beeinflusst wird.

Wenn beim Ausführen folgender Fehler erscheint:

```
Exception in thread "main" java.lang.NullPointerException
    at
game.gamestorage.texas.db.estimation.GetterDBProjector.setPreFlopShcDa
ta (GetterDBProjector.java:221)
```

bedeutet das, dass die Datenbank bereinigt werden sollte, da es im Laufe der Zeit Inkonsistenzen ansammeln.

Dafür muss auf der Datenbank alle Tabellen geleert werden. (Befehlsdatei *sql.txt* beigelegt)

Zugangsdaten zur datenbank:

```
URL: mysql://sqlhost1.informatik.uni-bremen.de/PokerBot
username = "sergejk8";
password = "b2jL9v2nq2NATBPu";
```

In den Tests ging der Nr5 Bot als Sieger heraus. Die Einbeziehung der Estimationergebnisse an nur einer Stelle im Nr5Revised-Bot scheint nicht ausreichend, um die Handlung des Bots erheblich zu verbessern.

Da sich aber der Bot Nr5 nur sehr wenig auf das Einbeziehen der Gegnereinschätzung konzentriert, kommt keine ausreichend große Benutzung unseres Moduls zustande. Die erfolgte Einbindung zeigt allerdings, dass sich das Modul problemlos in andere Bots einbinden lässt. Bei Benutzung der im vorhergehenden Abschnitt beschriebenen Methoden der künstlichen Intelligenz könnte unser Modul die Gegnereinschätzung noch erheblich verbessern.

## Zusammenfassung und Ausblick

Ziel unserer Projektgruppe ist es eine Methode zur Gegnertyperkennung zu entwickeln, die dann später von Bots als Entscheidungsgrundlage im Spiel verwendet werden kann. Ein Bot kann nur dann intelligent seinen Gegner einschätzen (und damit intelligent Poker spielen), wenn ausreichend Daten für den Spieler bereitstehen, auf denen die Entscheidungen beruhen. Unser erweitertes Estimation-Modul bietet dem Bot dieses mehr an Informationen und Daten über den Gegner. Es kann dazu benutzt werden, um Entscheidungen (basierend auf den Gegner) besser treffen zu können.

Nach Slansky gibt es viele Möglichkeiten das Wissen über das Spiel- und Setzverhalten seines Gegners einzusetzen, um bessere Entscheidungen im Spiel treffen zu können. Bei *loose*-Gegnern sollte man nicht *bluffen*, und bei gutem Blatt ebenfalls *loose* spielen, bei *tight*-Gegnern, sollte man mehr *loose* spielen als bei *Bluffs* und bei gutem Blatt lieber *tight*. Wenn ein Spieler *tight* ist und dieser *raised*, und alle andern *folden*, könnte man auch ein halbgutes Paar (z.B. Jungen) *folden*, beim *loose* würde man mitgehen. Viele Entscheidungen lassen sich also aus der Gegnereinschätzung ableiten.

Das von uns entwickelte Modul ermittelt in jeder Phase des Spiels den Zug den der Gegner machen sollte und vergleicht es dann mit der tatsächlichen Handlung. Durch diesen Prozess wird die Einordnung des Gegners im Laufe des Spiels ständig präzisiert und erreicht insofern das Ziel eine Einschätzung für den Bot zu liefern, welche er gewinnbringend benutzen kann und auf die er sich gut verlassen kann.



Weiterentwicklungspotentiale bestehen vor allem im Ausbau der intelligenten Methoden zur Entscheidungsfindung des Bots, z.B. mit Fuzzy Logic und Bayes'schem Netz. So müssen Zugentscheidungen nicht mehr statisch getroffen werden. Außerdem ließe sich der BeliefVector so erweitern, dass eine genauere Vorhersage der *hole cards* des Gegners möglich ist, um so beim eigenen Setzverhalten und bei der objektiven Einschätzung des Gegners weitere Genauigkeit der Vorhersage zu erreichen.

## Anhang

### Code-Beispiele

Unser Estimationmodul verarbeitet die Daten aus der Datenbank für alle Phasen des Spiels. In folgenden Beispielen wird gezeigt, wie die Estimation für die Flop-Runde realisiert wurde. Analog wird das auch für Turn und River in unserem Modul realisiert.

### Algorithmen im Estimationmodul

*bot.module.th.estimate.Opponent*

Hier werden die Estimations für die einzelnen Phasen des Spiels ausgerufen:

```
/**
 * instanciate estimations at rounds(line 71)
 */
private void init() {
    estimationAtRounds = new IF_EstimationAtRound[5];
    estimationAtRounds[0] = new PreFlopShc(new TableBSS());
    estimationAtRounds[1] = new PreFlop();
    estimationAtRounds[2] = new Flop();
    estimationAtRounds[3] = new Turn();
    estimationAtRounds[4] = new River();
}
```

Und werden danach weiterverarbeitet:

```
/* line 96 */
for (int i = 0; i < estimationAtRounds.length; i++) {
    estimationAtRounds[i].fetchData(gDBP);
    estimationAtRounds[i].estimate();
    estimationAtRounds[i].mergeIntoRound();
    estimationAtRounds[i].mergeIntoOverall(this);
}
```

## Verarbeitung der Daten aus der Datenbank

*bot.module.th. estimation.Flop*

```
/* equity proposal (line 91) */
int player_i = 0;
for(i=0; i<i_map.length; i++) {
    if(i_map[i]==gDBP.getPlayerId())
        player_i = i;
}
long potsize = gDBP.getHpmsFlopBetsCO().get(key)[3];
long callsize = gDBP.getCallSizeAtFlopCO(key);
long action = gDBP.getHpmsFlopBetsCO().get(key)[1];
double potodds = (((double) callsize) /
    ((double) potsize+ (double) callsize)) * 100;
int intAction = (int) action;
switch(intAction) {
    case -1: actions[0]++; break;
    case 0: actions[1]++; break;
    case 1: actions[1]++; break;
    case 5: actions[2]++; break;
    case 9: actions[3]++; break;
}
```

## Berechnung der Anspielhäufigkeit (*looseness*) und Aggressivität

Die objektiven Daten werden mit den Vorschlägen verglichen und daraus werden die Eigenschaften für die Bewertung abgeleitet.

*bot.module.th. estimation.Flop*

```
/* line 109 */
if (odds[player_i] >= 4*potodds) {
    /* raise proposed */
    if(action < 0 ) // player folded
        behaviour[2]--2; // very tight
    else if(action <= 1) // player called
        behaviour[3]--; // passive
} else if (odds[player_i] >= potodds) {
    /* call proposed */
    if(action < 0 ) // player folded
        behaviour[2]--; // tight
    else if(action>=5) // player raised
        behaviour[3]++; // aggressive
} else {
    /* fold proposed */
    if(action >= 0 && action <=1) // player called
        behaviour[2]++; // loose
    else if(action>=5) { // player raised
        behaviour[2]++2; // very loose
        behaviour[3]++; // aggressive
    }
}
```

```

    }
}
behaviour[4]++; // round is counted
if(action>0) behaviour[5]++; // round is counted as played

```

## Realisierung der dynamischen Spielerbewertung

Der Spieler wird nicht neutral, sondern mit seinem eigenen Verhalten verglichen. Er wird in Abhängigkeit von seinen tatsächlichen Aktionen und vorgeschlagenen Aktionen bewertet.

*bot.module.th.estimation.Flop*

```

/* line 142 */
public void mergeIntoRound() {
    double ratio = new Double(behaviour[2]) /
        new Double(behaviour[4]);
    double value = new Double(100-behaviour[0]) * ratio;
    behaviour[0] = behaviour[0] + (int) value;
    behaviour[0] = (behaviour[0]<0) ? 0 :
        (behaviour[0]>100 ? 100 : behaviour[0]);

    ratio = new Double(behaviour[3]) /
        new Double(behaviour[5]);
    value = new Double(100-behaviour[1]) * ratio;
    behaviour[1] = behaviour[1] + (int) value;
    behaviour[1] = (behaviour[1]<0) ? 0 :
        (behaviour[1]>100 ? 100 : behaviour[1]);
}

```

## Erweiterung von Datenbank und objektralationaler Abbildung

Beispiel für eine SQL-Abfrage aus der Datenbank, zum Herausfinden der Bets, die in Flop-Runden stattfanden.

*game.gamestorage.texas.db.estimation.GetterQuery*

```

/* line 549 */
protected static String getFlopBetsOfRound(int player_id, int
competition_id, int lastEvaldHandId) {
    // @formatter:off
    String sql = "SELECT t1.* " +
        "FROM " +
        "(SELECT bet.round_id AS round_id,
            MIN(bet.id), bet.action, bet.betsize,
            bet.potsize, bet.timestamp, bet.round_id

```

```

        AS r2 "+
        "FROM bet JOIN (round JOIN hand) ON
        (bet.round_id = round.id AND round.hand_id
        = hand.id) "+
        "WHERE bet.player_id = " + player_id + " " +
        "AND hand.competition_id = " +
        competition_id + " " + "AND hand.id > " +
        lastEvaldHandId + " " +
        "GROUP BY round.id) AS t1 " +
        "LEFT JOIN card_common ON t1.round_id =
        card_common.round_id "+
        "GROUP BY t1.round_id HAVING count(*)=3;";
    // @formatter:on
    return sql;
}

/* line 605 */
protected static String getFlopRounds(int player_id, int
competition_id, int lastEvaldHandId) {
    // @formatter:off
    String sql = "SELECT " +
        "round.hand_id, round.id, round.timestamp " +
        "FROM " +
        "bet JOIN (round JOIN hand) " +
        "ON " +
        "(bet.round_id = round.id " +
        "AND " +
        "round.hand_id = hand.id) " +
        "WHERE " +
        "bet.player_id = " + player_id + " " +
        "AND " +
        "hand.competition_id = " + competition_id
        + " " + "AND " +
        "hand.id > " + lastEvaldHandId + " " +
        "AND " +
        "round.id IN (SELECT round_id FROM
        (SELECT COUNT(*) AS num, round_id FROM
        card_common GROUP BY round_id
        HAVING num=3) AS t1)" + ";";
    // @formatter:on
    return sql;
}

```

## Einbindung des Estimationmoduls im Bot

Die *estimation.config* wird in der main-Methode im Bot aktiviert:

*bot.Nr5Revised*

```
/* line 49 */
public static void main(String[] args) throws Exception {
    String config = "estimation.config";
    // [...]
}

/* line 798 */
/**
 * Ermittlung der Tight/Looseness und Agressive/Passiveness aus
 * der DB
 */
private void playerRecognitionDB(int round) {
    String player_name_left = state.getPlayers()[1].getName();
    String player_name_right = state.getPlayers()[2].getName();
    if (!estimations.isEmpty() && estimations.get(0)
        instanceof Estimation) {
        Estimation estim = (Estimation) estimations.get(0);
        for (Map<Integer, Opponent> map :
            estim.getOpponentsEstimations().values()) {
            for (Integer player_id : map.keySet()) {
                Opponent o = map.get(player_id);
                if (o.getPlayerName().equals(player_name_left)) {
                    if (o.getBehaviourAtRounds()[round][0] > 20)
                        playerSetType[1] = "Tight";
                    else
                        playerSetType[1] = "Loose";
                    if (o.getBehaviourAtRounds()[round][1] > 50)
                        playerBetType[1] = "aggressive";
                    else
                        playerBetType[1] = "passive";
                } else if
                (o.getPlayerName().equals(player_name_right)) {
                    if (o.getBehaviourAtRounds()[round][0] > 20)
                        playerSetType[2] = "Tight";
                    else
                        playerSetType[2] = "Loose";
                    if (o.getBehaviourAtRounds()[round][1] > 50)
                        playerBetType[2] = "aggressive";
                    else
                        playerBetType[2] = "passive";
                }
            }
        }
    }
}
```

## Quellen

1. Slansky, D.: The Theory of Poker, Two Plus Two Publishing, 4th ed., 2005.
2. Korol, A.; Witthold, B.: Entwicklung einer Pokeragenten-Plattform, Diplomarbeit Bremen 29.08.2010,  
[http://pokerbot.workingcopy.de/lib/exe/fetch.php?media=diplomarbeit\\_korol\\_witthold.pdf](http://pokerbot.workingcopy.de/lib/exe/fetch.php?media=diplomarbeit_korol_witthold.pdf)
3. ACPC (Annual Computer Poker Competition),  
<http://www.computerpokercompetition.org/>