Machine Learning Projekt
PokerBot: Machine Learning Multi-Agent Simulation

Peter Behrens (Matrikelnummer: 8747156)
Nina Mergelsberg (Matrikelnummer: 6843257)
Niklas Wichter (Matrikelnummer: 7428266)

Juni/Juli 2021

Ehrenwörtliche Erklärung

Wir versichern hiermit, dass wir die vorliegende Arbeit mit dem Thema: „PokerBot: Machine Learning Multi-Agent Simulation “ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mannheim, 27.06.2021
Ort, Datum

Peter Behrens
Peter Behrens

Mannheim, 27.06.2021
Ort, Datum

Nina Mergelsberg
Nina Mergelsberg

Mannheim, 27.06.2021
Ort, Datum

Niklas Wichter
Niklas Wichter

Inhaltsverzeichnis

1 Einleitung

Software-Agenten werden in vielen verschiedenen Kontexten eingesetzt, wie zum Beispiel im E-Commerce für die Modellierung des Verhaltens von Menschen und der Umwelt, Sprachassistenten für die Modellierung von Gesprächen oder auch bei Spielen (vgl. Bădică et al., 2007; Giardini & Amblard, 2013; Silver et al., 2018). All dies sind komplexe Systeme, die aus vielen verschiedenen Variablen bestehen, die das System als Ganzes beeinflussen. Darüber hinaus sind die Variablen selbst oft komplex und ändern sich im Zeitablauf (Ramirez et al., 2019).
Im Hinblick auf Spiele ist die Entwicklung solcher AI-gestützten Modelle sehr komplex (Ramirez et al., 2019). Für AlphaGo und Schach beispielsweise kommen insbesondere Deep Reinforcement Learning (DRL)-Ansätze zum Einsatz, die die extrem große Menge der vorhandenen Merkmale berücksichtigen können. Schach und Go sind dabei Beispiele für „vollständige Informationsspiele“, bei denen alle Spieler Zugriff auf alle Informationen auf dem Brett haben, wie z. B. die Platzierung der Figuren (Ramirez et al., 2019).
Poker dagegen ist ein Spiel mit unvollkommener Information, in denen die Spieler und damit die Entscheidungsträger unterschiedliche oder keine Informationen über den Zustand des Spiels haben (Zinkevich et al., 2007). Insbesondere können die Spieler die Karten der anderen nicht sehen. Im Gegensatz zu Spielen mit vollständiger Information, bei denen es theoretisch eine optimale Vorgehensweise für jeden diskreten Spielzustand gibt, spielt daher die Unsicherheit eine große Rolle bei der Bestimmung der optimalen Strategie zu einem bestimmten Entscheidungszeitpunkt. Solche Variablen wie zum Beispiel die Wettgewohnheiten der Gegner, die Wahrscheinlichkeit, dass sich bestimmte Karten im Deck oder in den Händen anderer Spieler befinden oder ob ein Gegner blufft oder nicht, all das sind Merkmale, die es für die KI schwierig machen, die optimale Strategie zu wählen (Ramirez et al., 2019).

Im Rahmen des Machine Learning Projekts im Modul „Advanced Machine Learning“ haben wir uns mit der Modellierung und Implementierung eines PokerBots beschäftigt. Dieser basiert auf einer der erfolgreichsten Familien von Algorithmen für Spiele mit unvollkommener Information, einer Variante der Counterfactual Regret Minimization (CFR) (Zinkevich et al., 2007). Der vorliegende Projektbericht dient dabei zur detaillierten Ausführung und Erklärung der Implementierung. Zunächst wird der Ablauf eines Pokerspiels erläutert, bevor in Kapitel 3 die technische Umsetzung des PokerBots beschrieben wird. Kapitel 4 geht anschließend genauer auf die einzelnen Spielvarianten ein, die im Laufe des Projekts ausprobiert wurden. Das darauffolgende fünfte Kapitel stellt die Ergebnisse des PokerBots vor. Zum Schluss wird ein abschließendes Fazit zum Projekt gezogen. Außerdem sind die Ausführungen der Einzelleistungen am Schluss des Projektberichts zu finden.

2 Ablauf des Spiels

Zu Beginn wird ein Spieler festgelegt, der für die erste Runde Dealer - also Kartengeber - ist. Er erhält einen Button und gibt diesen in der nächsten Runde an seinen Mitspieler weiter. Die beiden Spieler links vom Dealer sind Small Blind und Big Blind. Das heißt, sie müssen einen vorgegebenen Einsatz leisten, der in den Pot wandert. Der Big Blind setzt in der Regel den doppelten Wert vom Small Blind. Die Funktion des Dealers sowie des Small und Big Blinds rotieren mit jeder neuen Runde im Uhrzeigersinn, sodass jeder Spieler mehrmals an der Reihe ist.
Nachdem alle Rollen zugewiesen worden sind, wird mit dem Austeilen der Karten begonnen. Jeder Spieler erhält zwei Karten (Hole Cards) und fünf Gemeinschaftskarten werden in der Mitte verdeckt hingelegt.

Nun kann das Spiel beginnen. Der Spieler links neben dem Big Blind setzt zuerst. Dabei gibt es verschiedene Möglichkeiten: Die Teilnehmer können mit dem Einsatz des vorherigen Spielers mitgehen, diesen erhöhen, passen (sofern noch nicht erhöht wurde) oder aufgeben. Haben alle Spieler ihren ersten Einsatz in den Pot gelegt, werden drei Gemeinschaftskarten aufgedeckt. Danach beginnt die zweite Setzrunde und der erste verbleibende Spieler (also ein Spieler, der in der ersten Runde nicht aufgegeben hat) links neben dem Dealer führt zuerst seinen Spielzug aus. Nachdem jeder Spieler an der Reihe war, wird die vierte Gemeinschaftskarte aufgedeckt. Auch die dritte Setzrunde beginnt mit dem ersten verbleibenden Spieler links vom Dealer. Haben alle Teilnehmer ihren Spielzug ausgeführt, wird die fünfte und letzte Gemeinschaftskarte aufgedeckt. Nachdem alle Karten sichtbar sind, haben die Spieler noch eine letzte Chance, ihren Einsatz zu erhöhen. Haben zuvor bereits alle Spieler bis auf einen aufgegeben, hat dieser automatisch gewonnen und erhält alle eingesetzten Chips. Sollte nach der letzten Setzrunde noch mehrere Teilnehmer im Spiel sein, kommt es zum sogenannten Showdown. Die Spieler decken nacheinander ihre Karten auf und die beste Pokerhand gewinnt den Pot.
Die Gemeinschaftskarten können dabei von jedem Spieler für seine Pokerhand genutzt werden, indem er beliebig viele seiner Hole Cards mit beliebig vielen Gemeinschaftskarten kombiniert. Wichtig zu beachten ist, dass eine Pokerhand immer aus fünf Karten bestehen muss. Von den sieben zur Verfügung stehenden Karten dürfen also insgesamt nur fünf verwendet werden.

2.1 Das Poker Ranking

Das Poker Ranking bestimmt die Rangfolge der Pokerhände. Es gibt also Auskunft darüber, wer die beste Kartenkombination hat und somit die Runde gewinnt. Dieses Ranking gilt für alle gängigen Pokervarianten und lautet folgendermaßen beginnend bei der schlechtesten Pokerhand:

- High Card: die höchste einzelne Karte.
- Paar: zwei Karten gleichen Rangs.
- Zwei Paare.

- Drilling: drei Karten gleichen Werts.
- Straße: fünf aufeinanderfolgende Karten.
- Flush: fünf Karten gleicher Farbe.
- Full House: ein Paar und ein Drilling.
- Vierling: vier Karten gleichen Werts.
- Straight Flush: fünf aufeinanderfolgende Karten gleicher Farbe.
- Royal Flush: höchstmögliche Straße (10-J-Q-K-A) in derselben Farbe - die höchstmögliche Pokerhand.

Sollten zwei oder mehr Spieler die gleiche Kartenkombination auf der Hand haben, zum Beispiel Spieler 2 hat ein Paar und Spieler 4 hat ebenfalls ein Paar, kann deswegen im ersten Moment kein Gewinner festgestellt werden, weil alle anderen Spieler raus sind oder niedrigere Karten haben, es zählt dann der Kartenwert. Das bedeutet Ass ist am höchsten, gefolgt von König, Dame, Bube, 10, 9, 8, 7, 6, 5, 4, 3 und am niedrigsten ist die 2. Lässt sich immer noch kein Gewinner verkünden, weil beide Spieler ein König Paar haben, zählt nicht die Farbe der Handkarten, sondern der so genannte Kicker. Das ist die Beikarte der Hole Cards eines Spielers, also diejenigen der fünf Karten, die nicht zu den Karten gehören, die den Rang der Hand bestimmen. Sollte hier wiederum Gleichstand herrschen, entscheidet der zweite Kicker und ggf. auch der dritte (Christl, 2021).

3 Technische Umsetzung

Dieses Kapitel unterteilt sich in drei Unterkapitel. Zu Beginn wird die Grundlage der Pokersimulation erläutert. Darauf folgend wird der verwendete Algorithmus aufgezeigt und anhand von den verwendeten Algorithmen und Techniken erläutert. Im letzten Kapitel wird auf die Verwendungen einer GUI für die Interaktion mit dem PokerBot eingegangen.

3.1 Grundlage der Pokersimulation

Für die Trainingssimulation eines Pokerspiels wird das Modul *PyPokerEngine* verwendet. Ein wichtiger Bestandteil besteht in der Klasse *BasePokerPlayer*. Diese besitzt alle Handlungsmöglichkeiten eines Pokerspielers und bietet die Grundlage für die spätere Simulation eines Pokerspielers.

Das Modul *PyPokerEngine* bietet verschiedene Grundeinstellungen für den Ablauf eines Pokerspiels. Diese können für verschiedene Trainings- und Testszenarios angepasst werden (GitHub/ishikota, 2016; Ishikota, 2017).

1. max_round : Anzahl an Runden
2. initial stack : Anfangsbestand eines jeden Spielers
3. small blind : Betrag des Small Blind
4. ante : Betrag des Mindesteinsatzes pro Spiel

3.2 Verwendeter Algorithmus

Für die Umsetzung des PokerBots wurde als Basis ein Counterfactual Regret Minimization Algorithmus verwendet. Der Anwendungsfall beim Poker ist für einen Algorithmus nicht einfach zu berechnen, da es sich um ein Spiel mit unvollständigen Informationen handelt. Im Vergleich zu anderen Algorithmen ist der CFR-Algorithmus von der Performance jedoch bedeutend besser, da er nicht auf alle Informationen angewiesen ist, sondern anhand der Ergebnisse eine optimalere Handlungsweise ableitet.

3.2.1 Counterfactual Regret Minimization

Der Counterfactual Regret Minimization Algorithmus (CFR) ist ein Reinforcement Learning Algorithmus, der am Ende eines Spiels oder einer Handlung eine Aktion bewertet. Dabei werden die möglichen Aktionen mit der jeweiligen ausgeführten Aktion verglichen und überprüft, welche Aktion die ideale Aktion gewesen wäre. Die Entscheidungen trifft der Algorithmus, indem er für jede mögliche Aktion Wahrscheinlichkeiten errechnet, mit welcher er durch eine Zufallszahl die Aktion auswählt. Die Entscheidung selbst beeinflusst die späteren Wahrscheinlichkeiten nur indirekt; ausschlaggebend für die Anpassung ist das allgemeine Ergebnis des Spiels.

Das Ziel des Algorithmus ist es, das Bedauern im Sinne einer schlechten Entscheidung auf ein Minimum zu reduzieren. Für die Verbesserung seiner Handlungsweise trainiert er ein Modell, welches das Ziel hat, möglich nah an das Nash Equilibrium zu gelangen, welches im nächsten Abschnitt erklärt wird (Brown & Sandholm, 2019; Zinkevich et al., 2007).

3.2.2 Nash Equilibrium (Nash-Gleichgewicht)

Das Nash Equilibrium beschreibt den anzustrebenden Zustand des CFR-Algorithmus. Je nachdem, wie komplex das ursprüngliche Problem ist, dauert es kürzer oder länger bis dieser Zustand erreicht werden kann. Bei sehr komplexen Problemen, wie das Poker spielen, kann dies nicht immer erreicht werden.

Basierend auf den möglichen Ausgangssituationen wird dadurch eine Wahrscheinlichkeitsverteilung aufgestellt, welche alle Wahrscheinlichkeiten für alle Aktionen in einem Spiel enthält. Diese können dann für den Ablauf des Algorithmus genutzt werden.

Für den PokerBot wurde sich auf aufgrund der Komplexität des Pokerspiels nur auf die Evaluierung mit zwei Handkarten bezogen, da bereits die nächste Stufe die Technik überforderte. Grundlegend ist der Ablauf aber derselbe - es dürfte nur bedeutend länger dauern bis ein ausreichend gutes Modell trainiert werden kann.

Berechnung der Anzahl der möglichen Kartenkombinationen

Für die Grundlage der späteren Anpassungen wurde eine Datei erstellt, welche alle möglichen Kartenkombinationen (für 2 Handkarten) enthält. Diese Datei passt der Algorithmus nach jedem Durchlauf anhand der Ergebnisse an, um das Nash Equilibrium zu erreichen.

$$Formel : \binom{\text{Anzahl aller Karten}}{\text{Anzahl der gezogenen Karten}}$$

Beispiel für 2 Handkarten:

$$\binom{52}{2} \text{ oder } \frac{52 * 51}{2} = 1326$$

Index der Speicherdatei:

String, welcher aus einer Liste der Handkarten besteht (ab dem zweiten Fall zusätzlich eine Liste der Gemeinschaftskarten)

Aufgrund von Performance-Problemen beschränkt sich die Bestimmung des Nash Equilibrium nur auf den ersten Fall mit zwei Handkarten. Für die spätere Berechnung der weiteren Züge wird ein Monte-Carlo-Algorithmus verwendet.

Tabelle 1: Mögliche Kartenkombinationen

Handkarten	Gemeinschaftskarten	Kartenkombinationen
2	-	1.326
2	3	2.598.960
2	4	20.358.520
2	5	133.784.560

Tabelle 2: Spalten der Speicherdatei

Aktion	
Fold	Enthält die Wahrscheinlichkeit für die Aktion „Fold“
Call	Enthält die Wahrscheinlichkeit für die Aktion „Call“
Raise	Enthält die Wahrscheinlichkeit für die Aktion „Raise“
Counter	- Zählt alle Spiele, welche mit dieser Kartenkombination gespielt wurde - Summe der gesamten Spalte ergibt Anzahl aller gelernten Spiele

3.2.3 Problem des Algorithmus

Bei einfachen Anwendungsfällen, wie beispielsweise Kuhn Poker (2 Spieler und 3 Karten) oder Schere-Stein-Papier, erlangt der Algorithmus schnell das Nash Equilibrium. Wohingegen bei komplexeren Problemen wie Limit-Poker, die Lernrate mit steigenden Iterationen massiv abnimmt und eine höhere Anzahl an weiteren Iterationen benötigt wird, um in die Nähe des Nash Equilibrium zu kommen.

3.2.4 Verbesserungsmöglichkeiten

Um die Performance des CFR-Algorithmus zu verbessern gibt, es verschiedene Möglichkeiten. Durch Kombinationen mit anderen Algorithmen wie dem Monte Carlo Algorithmus lassen sich die benötigte Dauer und Anzahl an Iterationen reduzieren.

Eine weitere Verbesserungsmöglichkeit ist es, die neuste Iteration stärker zu gewichten, sodass mit steigender Anzahl an Iterationen die Lernrate stabil bleibt und nicht gegen null geht. Die letzte genannte Verbesserungsmöglichkeit wurde auch im Code umgesetzt.

3.3 GUI

Die grafische Darstellung der Pokerkomponenten und die Möglichkeit, selbst mit der gelernten Poker KI zu interagieren, wurde mit dem Modul *PyPokerGUI* umgesetzt (GitHub/ishikota., 2016).
Nachdem der PokerBot mit Hilfe der Erklärungen in unserem GitHub Repository ausgeführt wurde, wird man im Browser zunächst auf die Registrierungsseite geleitet. Dort registriert

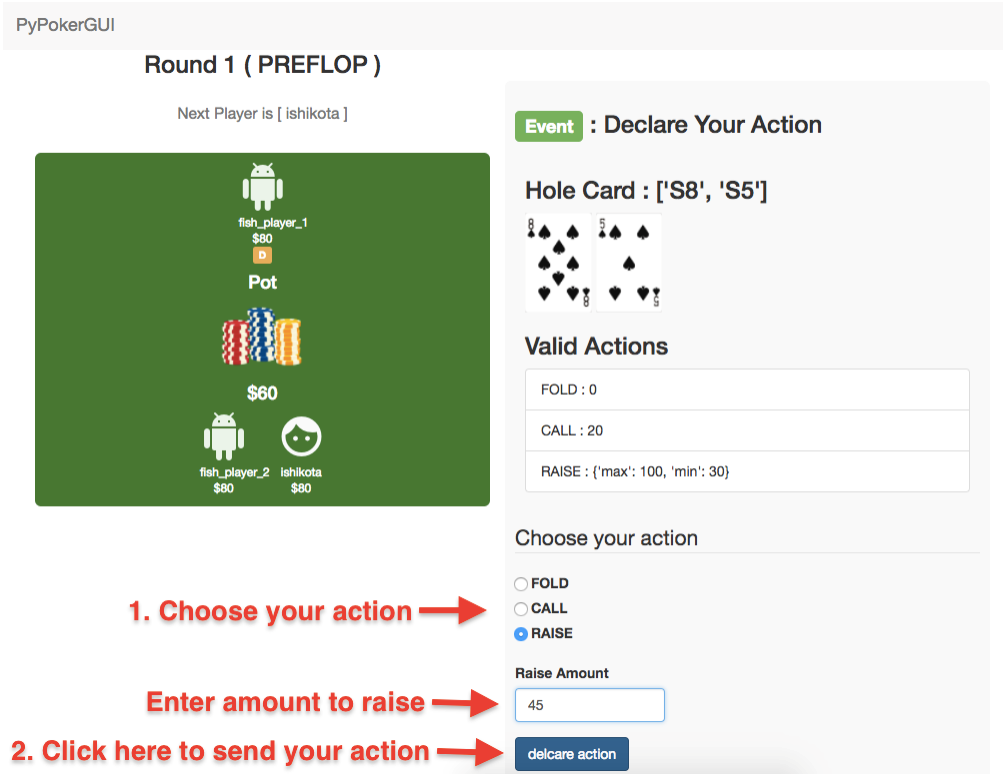

Abbildung 1: Abbildung und Erklärung der Poker GUI (Quelle: https://github.com/ishikota/PyPokerGUI)

man sich mit einem freigewählten Usernamen und startet das Spiel.

Abbildung 1 veranschaulicht das User Interface während des Spiels. Dabei wird die aktuelle Runde, der aktuelle Einsatz und die beteiligten Spieler angezeigt. Des Weiteren sieht man selbst, welche Karten man auf der Hand hat und welche zulässigen Aktionen gemacht werden können. Um zu spielen, wählt man in jeder Runde die entsprechende Aktion, die man spielen und den Wert des Einsatzes, den man setzen möchte. So spielt man auf eine intuitiv bedienbare Weise gegen den PokerBot.

4 Einzelne Spielvarianten/Gegner

Im Nachfolgenden werden die einzelnen Spielvarianten bzw. Gegner, die bespielt werden können, vorgestellt. Neben den Bots kann über ein GUI selber gegen die KI gespielt werden.

4.1 Monte-Carlo-Simulation

Die Monte-Carlo-Simulation ist ein Verfahren aus der Stochastik, bei dem eine sehr große Zahl gleichartiger Zufallsexperimente die Basis darstellt. Ziel ist es, analytisch nicht oder nur aufwendig lösbare Probleme mit Hilfe der Wahrscheinlichkeitstheorie numerisch zu lösen. Diese Simulation stellt eine Alternative zu einem adversialen Suchalgorithmus dar. Dabei wird die Breite des Suchbaums limitiert und durch zufällige Knotenauswahl erhält man ein Blatt mit einem bestimmten Evaluierungswert. Durch eine wiederholte Ausführung kann der Nutzen der darauf folgenden Aktionen des aktuellen physischen Zustands an den tatsächlichen Nutzen angenähert werden (Czarny & Schumann, 2009, S. 3). Die Zufallsexperimente können entweder real durchgeführt werden oder in Computerberechnungen mittels Monte-Carlo-Algorithmen.

Überträgt man dieses Prinzip auf das Poker-Szenario bedeutet das, dass nicht der gesamte Game Tree berücksichtigt werden muss, sondern lediglich zufällig verschiedene Pfade durch den Baum abgegangen werden. Das geht schnell und kann sehr häufig wiederholt werden, so dass der Durchschnitt dieser Stichproben zu einer guten Bewertung des Ausgangszustands des Spiels konvergiert. Die Monte-Carlo-Simulation begrenzt die Suchbreite an jedem Knoten und verwendet zufällige Auswahlfunktionen an den Entscheidungsknoten. Beim No-limit Poker existieren über 10^{71} verschiedene Knoten bzw. Wahrscheinlichkeiten, die möglich sind, so dass der Game Tree zu groß zum vollständigen traversieren ist (Czarny & Schumann, 2009).

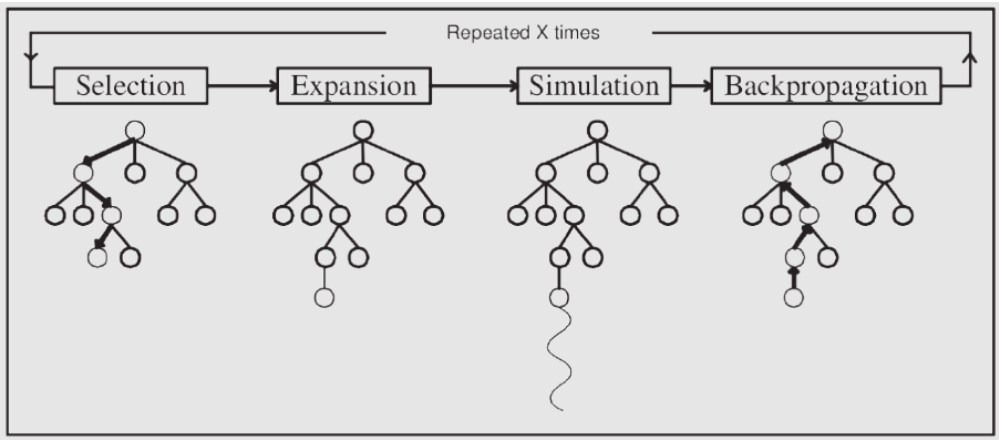

Abbildung 2: Monte-Carlo-Simulation (Quelle: Van den Broeck, 2019)

Die Monte-Carlo-Simulation baut inkrementell einen Teilbaum des gesamten Spielbaums im

Speicher auf. Für jeden Knoten speichert das Standard-MCTS den erwarteten Wert der Belohnung dieses Knotens zusammen mit einem Zähler, der die Anzahl der erfassten Spiele speichert, die zu der Schätzung geführt haben. Der Algorithmus beginnt mit der Wurzel des Baums und wiederholt die folgenden vier Schritte, bis ihm die Rechenzeit ausgeht.

1. Auswahl (eng. Selection): Ausgehend von der Wurzel wählt der Algorithmus in jedem gespeicherten Knoten den Zweig aus, den er weiter untersuchen möchte, bis er ein gespeichertes Blatt erreicht. Das muss nicht zwingend ein Blatt des vollständigen Game Trees sein.

2. Erweiterung (eng. Expansion): Ein (oder mehrere) Blätter werden dem Baum als Kind(er) des im vorherigen Schritt erreichten Blattes hinzugefügt.

3. Simulation: Beispielhaft wird ein Spiel, beginnend mit dem hinzugefügten Blatt, bis zum Abschluss gespielt.

4. Backpropagation: Die Schätzungen der Erwartungswerte und der Zähler auf dem erforschten Pfad werden mit den neuen Informationen aktualisiert. Abschließend wählt eine action-selection Strategie eine gute Aktion aus, die auf Grundlage der gespeicherten Werte in den einzelnen Kindern der Wurzeln ausgeführt werden soll (van den Broeck et al., 2009, S. 1f.).

4.2 Call Bot

Neben der Monte-Carlo-Simulation, die Entscheidungen auf Grundlage von Berechnungen des Erwartungswerts trifft, gibt es den Call Bot, der immer mitgeht. Das bedeutet, der Bot geht immer mit ungeachtet davon, welche zwei Karten er auf der Hand hat bzw. in der Mitte liegen. In der Theorie sollte dementsprechend die Monte-Carlo-Simulation besser performen, da die Spielweise auf reinem Zufall beruht.

4.3 Honest Player

Der Honest Player ist ebenfalls ein Bot, der seine Entscheidungen auf Basis von Berechnungen trifft. Im Vergleich zum Call Bot stellt er somit einen smarteren Bot ähnlich zur Monte-Carlo-Simulation dar. Die Berechnungen hinter dem Honest Player sind jedoch nicht so komplex. Er simuliert eine gewisse Anzahl an Runden und errechnet, wie viele er gewinnen würde. Anschließend beurteilt er auf Grundlage dieser Wahrscheinlichkeit, ob er im Spiel bleiben sollte.
Wie der Name schon andeutet, callt dieser Bot und bleibt somit im Spiel, wenn er gute Karten errechnet hat und verlässt das Spiel (pass bzw. fold), wenn er schlechte Handkarten bzw. Gemeinschaftskarten besitzt. Das bedeutet, er spielt sehr ehrlich, da die Mitspieler direkt sehen können, ob er gute bzw. schlechte Karten hat. Je mehr Spieler an einer Runde teilnehmen, desto unwahrscheinlicher ist die Gewinnchance und damit auch die Wahrscheinlichkeit, dass der Honest Player rausgeht.

5 Ergebnisse

Kommen wir nun zu den Lernerfolgen des PokerBots. Der Algorithmus wurde an einem Tisch trainiert, an dem fünf unterschiedliche Poker-Algorithmen mitgewirkt haben:

1. MCCFR Player 1 (lernt das Modell und setzt es um)
2. MCCFR Player 2 (setzt das Modell um)
3. Monte Carlo Player
4. Honest Player
5. Call Player

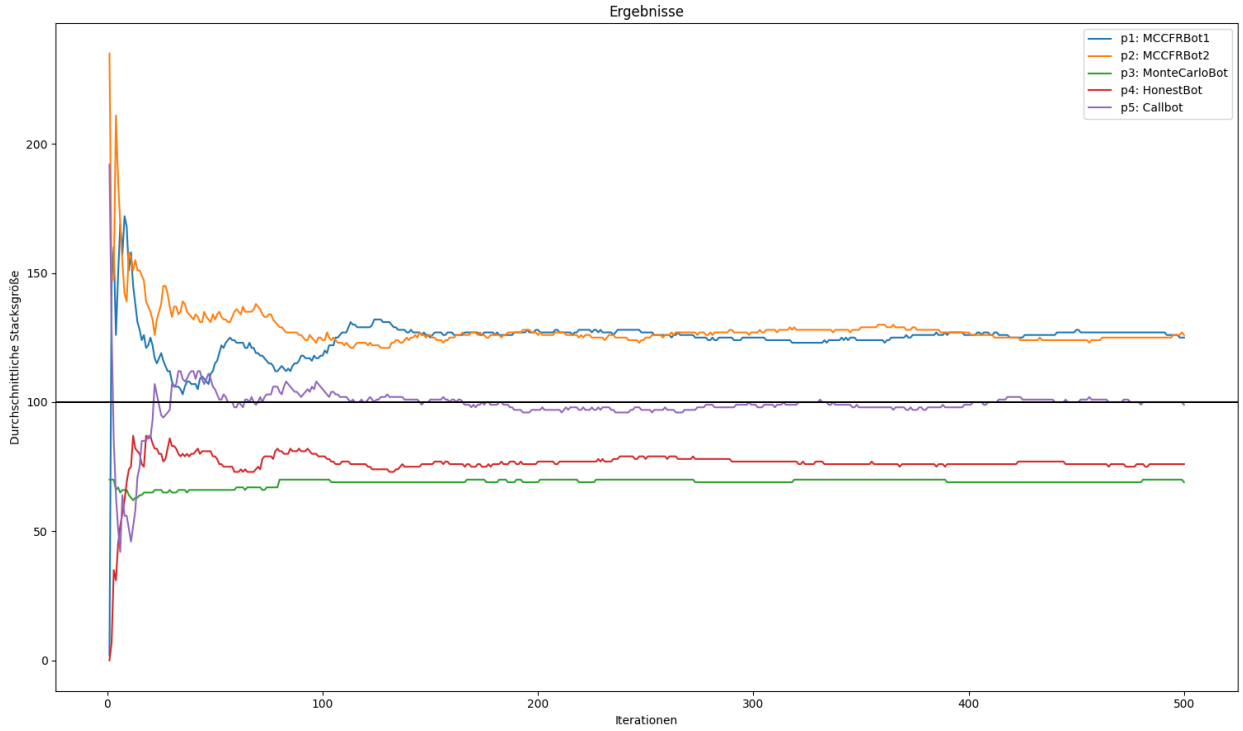

Abbildung 3: Ergebnisse fortlaufender Durchschnitt 500 Iterationen

Anhand der Grafik ist deutlich zu erkennen, dass die MCCFR-Player (Player 1 und 2) am besten abschneiden und sich gegenüber den anderen deutlich unterscheiden.
Der Monte Carlo-Algorithmus alleine zeigt dabei die geringste Stackgröße und schneidet damit am schlechtesten ab. Auch der Honest Player und Call Player können durch durchschnittliche Ergebnisse erreichen. Dies bestätigt die Annahme, dass der Monte Carlo-Bot in Kombination mit dem CFR-Algorithmus bessere Ergebnisse erzielt und daher am geeignetsten für den Anwendungsfall des PokerBots ist.

5.1 Ideen für Verbesserungen

Für eine spätere Verbesserung des Algorithmus können verschiedene weitere Szenarien hinzugefügt werden. Derzeit ist der Algorithmus nur auf dieses Szenario trainiert, sodass nicht sichergestellt werden kann, ob er auch gegen eine andere Gegnerkombination gut performt. Dies müsste weiter ausgetestet werden.

In einem normalen Pokerspiel Mensch gegen Mensch können diese unabhängig von den eigenen Karten *Bluffen*. Das bedeutete, sie wollen dem Gegner vermitteln, dass man beispielsweise gute Karten hat, obwohl dies nicht der Fall ist. Dadurch wollen sie ihn durch gezielte Erhöhung des Einsatzes verunsichern und zum Aufgeben zwingen. Da man selbst seine Karten nicht zeigen muss, wenn alle gegnerischen Spieler in einem Spiel aufgeben haben, wird in einem solchen Fall keiner erfahren, ob nun geblufft wurde oder nicht.

Der Algorithmus allein kann sich nur auf Informationen in Form der gelegten Karten beziehen und durch Kontrolle der am Ende aufgedeckten Karten seine Annahmen bestätigen. Um die menschliche Komponente und die dadurch erzeugten Daten in Form der Körpersprache und der normalen Sprache in den Lernerfolg mit einzubeziehen, könnte man durch Auswertung von Videodateien und unter Verwendung von Nature Language Processing den Algorithmus darauf trainieren, auch ein Bluffen zu erkennen und die KI so weiter verbessern.

6 Fazit

Das Ziel des Projekts und des vorliegenden Projektberichts ist es, einen PokerBot zu implementieren und zu erklären, der basierende auf den Counterfactual Regret Minimization (CFR) Algorithmus und der PyPokerEngine funktioniert. Es stellte sich heraus, dass der gewählte Algorithmus am geeignetsten ist, da dieser ein beliebter, iterativer Algorithmus zur Berechnung von Strategien in Spielen mit extensiver Form ist.

Um die Funktionalitäten des PokerBots zu erklären, wurde zunächst der allgemeine Spielablauf von Poker erläutert. Die sich anschließende Beschreibung der technischen Umsetzung ging dabei genauer auf den Algorithmus und seine weiteren Bestandteile wie das Nash-Gleichgewicht ein. Zusätzlich zu der Implementierung des eigentlichen Algorithmus beschäftigten wir uns mit verschiedenen Spielvarianten bzw. Gegnern. Dabei wurde deutlich, dass aus den getesteten Spielvarianten, die Monte-Carlo-Simulation, am besten abschneidet.

Die abschließende Präsentation der Ergebnisse zeigte deutlich, wie gut der implementierte PokerBot funktioniert, wobei die Gegner miteinander verglichen wurden. Es zeichnet sich heraus, dass sich unsere erste Annahme, nämlich eine Kombination aus Counterfactual Regret Minimization und Monte-Carlo-Simulation auszuprobieren, die besten Ergebnisse erzielte.

Poker ist jedoch ein sehr viel komplexeres Spiel. Denn durch die fehlenden Spielinformationen kann der Algorithmus seine Berechnungen nur anhand der gelegten Karten durchführen. Die wichtigste Komponente beim Poker ist jedoch der Mensch. Die Körpersprache, das Gesagte und ob der Gegenspieler daher blufft oder nicht, all diese entscheidenden Informationen sind für die KI nicht einsehbar. Aufgrund dessen gibt es auch einige Verbesserungen, die vorgenommen werden könnten, wobei dies ein Ansatzpunkt für zukünftige Forschungsarbeiten ist.

Eigenanteil Peter Behrens

Bei unserem Applied Machine Learning Projekt bestand mein Eigenanteil in der Projektkonzeption, dabei ging es vor allem um den Umfang und die spezifischen Einzelheiten des Projekts. Zudem wurden verschiedene Libraries recherchiert, die für die Umsetzung und Implementierung verwendet werden können. Bei der technischen Realisierung habe ich unterstützend mitgewirkt und einzelne Aufgabenpakete übernommen.

Bezüglich der Dokumentation war ich für das zweite Kapitel Ablauf und für das vierte Kapitel Gegner verantwortlich. Dazu wurden unter anderem die Regeln des Pokerspiels recherchiert und die einzelnen Rollen in einer Pokerrunde herausgearbeitet. Darüber hinaus wurden das Ranking der Pokerkarten erstellt, um zu identifizieren, wodurch ein Spieler eine Runde für sich gewinnen kann. Diese Informationen wurden auch für das Erstellen des AI PokerBots verwendet.
Für das vierte Kapitel wurden die einzelnen Spielvarianten, die unser Projekt bieten soll, beschrieben. Dabei wurde vor allem die Monte-Carlo-Simulation, die als umfangreichster Gegner in unseren PokerBot einfließt, berücksichtigt.

Bei der Präsentation war ich für das Design und Layout zuständig und habe neben der Erstellung eines Master Slide Decks, auch die Folien zum Spielablauf sowie die der Spielgegner übernommen.

Eigenanteil Nina Mergelsberg

Meinen Eigenanteil für das Machine Learning Projekt leistete ich in administrativen und operativen Arbeiten im Zusammenhang mit der Umsetzung und der Verschriftlichung des Projekts. Neben der Gliederung des Projektreports und dem Schreiben der Einleitung, Kapitel 3.3, Kapitel 5 und des Schlusses, habe ich mich zudem mit dem Layout, der Korrektur und den Feinheiten des Berichts beschäftigt.

Für die Verschriftlichung der Einleitung galt es dabei, sich einen Überblick über die verschiedenen Methoden zur Umsetzung von Spielen mit unvollständigen Informationen zu verschaffen und heraus zu selektieren, welches Szenario für unser Projekt am besten geeignet ist. Daraus ergab sich der „Business“-Use Case einen PokerBot zu implementieren, der auf Basis des Counterfactual Regret Minimization Algorithmus, dem Nash-Gleichgewicht und der PyPokerEngine umgesetzt wurde.

In der Abschlusspräsentation übernehme ich die Einleitung in die Präsentation, die Vorstellung der Ergebnisse des PokerBots sowie die Erklärung der Live-Demo.

Eigenanteil Niklas Wichter

Mein Eigenanteil bestand in der technischen Umsetzung und der Erstellung der Ergebnisse des PokerBots. Des Weiteren habe ich für die Dokumentation und die Präsentation die jeweiligen Seiten für die technische Umsetzung erstellt. Dafür hab ich mich mit dem Counterfactual Regret Minimization Algorithmus beschäftigt und in diesem Zusammenhang auch Nash Equilibrium behandelt.

Dazu habe ich mich mit verschiedenen Algorithmen für das Erlernen von Poker beschäftigt. Die schlussendliche Wahl fiel auf einen *Counterfactual Regret Minimization Algorithmus*. Das ist darin begründet das Poker nicht wie viele andere Spiele alle Informationen während eines Spielverlaufs jedem Teilnehmer preisgibt, sondern ein Teil der Informationen bis zum Spielende verborgen bleibt. In Form der verdeckten Karten der Mitspieler. Dadurch kann der Algorithmus diese nicht für seine Berechnungen verwenden. Als Ersatz für die fehlenden Informationen berechnet dieser Algorithmus die möglichen Ausgänge eines Spiels mit ihren Wahrscheinlichkeiten auf Grundlage der bestehenden Informationen. Dabei stellt sich dieser am Ende einer jeden Runde die Frage: Was wäre gewesen, wenn ich mich anders entschieden hätte?

In der Abschlusspräsentation übernehme ich die Vorstellung der technischen Umsetzung des PokerBots.

Aus meiner Perspektive habe ich den Aufwand und die Komplexität dieses Projektes etwas unterschätzt und würde beim nächsten Mal einen kleineren Umfang abstecken, welcher ich komplett bearbeiten kann, statt von einem sehr komplexen Thema nur teilweise bestimmte Aspekte zu erfüllen.

Literaturverzeichnis

Bădică, C., Ganzha, M., & Paprzycki, M. (2007). *Implementing Rule-Based Automated Price Negotiation in an Agent System*. Journal of Universal Computer Science, 13 (2), S.244–266. Zugriff am 25.07.2021 unter: `https://www.researchgate.net/profile/Maria-Ganzha/publication/220349766_Implementing_Rule-Based_Automated_Price_Negotiation_in_an_Agent_System/links/0912f50bf60d95a0b6000000/Implementing-Rule-Based-Automated-Price-Negotiation-in-an-Agent-System.pdf`

Brown, N., & Sandholm, T. (2019). *Libratus: The Superhuman AI for No-Limit Poker*. In C. Sierra (Hrsg.), International Joint Conferences on Artificial Intelligence (IJCAI 2017): Melbourne, Australia, 19-25 August 2017 (S. 5226–5228). Curran Associates Inc. doi: 10.24963/ijcai.2017/772

Chaslot, G., Bakkes, S., Szita, I., & Spronck, P. (2008). *Monte-Carlo Tree Search: A New Framework for Game AI*. In Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (S. 216–217). AAAI Press.

Christl, J. (2021). *Regeln für Poker: Einfach und verständlich erklärt*. Zugriff am 01.07.2021 unter: `https://praxistipps.focus.de/regeln-fuer-poker-einfach-und-verstaendlich-erklaert_100351`

Czarny, D. A., & Schumann, D. (2009). *Akuma - Online Learning Monte Carlo Poker Agent*. Zugriff am 26.07.2021 unter: `https://damianworks.files.wordpress.com/2012/12/czarny_poker_challenge_09.pdf`

Giardini, F., & Amblard, F. (Hrgs.) (2013). *Multi-Agent-Based Simulation XIII*. Springer Berlin Heidelberg.

GitHub. (2016). ishikota/*PyPokerEngine: Poker engine for poker AI development in Python*. Zugriff am 26.07.2021 unter: `https://github.com/ishikota/PyPokerEngine`

Ishikota. (2017). *PyPokerEngine: poker AI development from today*. Zugriff am 26.07.2021 unter: `https://ishikota.github.io/PyPokerEngine/`

Ramirez, A., Reinman, S., & Norouzi, N. (2019). *PokerBot: Hand Strength Reinforcement Learning*. In 2019 IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA), Sofia, Bulgaria.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*. Science (New York, N.Y.), 362 (6419), S.1140–1144. doi: 10.1126/science.aar6404

Van den Broeck, G. (2019). *Monte-Carlo tree search for multi-player, no-limit Texas hold'em poker*. Katholieke Universiteit Leuven. Zugriff am 26.07.2021 unter: `https://web.cs.ucla.edu/~guyvdb/slides/SIKS11.pdf`

Van den Broeck, G., Driessens, K., & Ramon, J. (2009). *Monte-Carlo Tree Search in Poker Using Expected Reward Distributions*. In Z.-H. Zhou & T. Washio (Hrsg.), Advances in Machine Learning (S. 367–381). Springer Berlin Heidelberg.

Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2007). *Regret minimization in games with incomplete information*. In Proceedings of the 20th International Conference on Neural Information Processing Systems (S. 1729–1736). Curran Associates Inc.