

# Bachelor Thesis

## Poker-Bot Competition

Autoren: Vito Cudemo, Michael Schärz

Hochschule: Hochschule für Technik

Studiengang: Informatik

Betreuende Dozenten: Prof. Dr. Manfred Vogel  
Dominik Frefel

Auftraggeber: Institut für Data Science,  
Fachhochschule Nordwestschweiz

Brugg, 14. August 2020

## Abstract

Im Bereich der selbstlernenden Algorithmen werden häufig Problemdomänen durch Spiele dargestellt und darauf mathematische Modelle angewendet. Der vorliegende Bericht beschreibt Modelle, die auf das bekannte Texas Holdem Poker Spiel (Kartenspiel) angewendet werden können. Dabei werden die fundamentalen Problemstellungen erläutert und durch mathematische Modelle adressiert. Der in Poker anzutreffende Zustandsraum, die nicht deterministische Vorhersage sowie das Credit Assignment Problem stehen dabei im Fokus. Mehrere Agenten-Zusammenstellungen werden als Reinforcement Learning Algorithmen implementiert und nutzen die Temporal-Difference Learning Methode, um selbstständig Muster im Spiel zu erkennen. Das führt allgemein zur Implementation eines Double Deep Q Network with Prioritized Experience Replay Modells, welcher gegen unterschiedliche Gegner Profit erzielen kann. Die Agenten treten gegenseitig in lokalen Testläufe, sowie auch auf einer dafür abgestimmten Plattform an und lassen direkte Vergleiche der Performance zu. Auswertungen zeigen zudem Grenzen auf. Die Umsetzung ist in Python unter Nutzung des Keras Frameworks erläutert.

## Ehrlichkeitserklärung

Hiermit bestätigen die Autoren, diese Arbeit selbstständig und unter Einhaltung der gebotenen Regeln sowie nur unter Benutzung der angegebenen Quellen erstellt zu haben.

Brugg, 14.08.2020

Brugg, 14.08.2020

  
\_\_\_\_\_  
Vito Cudemo  
\_\_\_\_\_  
Michael Schärz

# Inhaltsverzeichnis

<b>Abstract .....</b>	<b>2</b>
<b>Ehrlichkeitserklärung .....</b>	<b>3</b>
<b>1 Einleitung .....</b>	<b>6</b>
1.1 Ausgangslage .....	6
1.2 Zielsetzung .....	6
1.3 Einschränkungen.....	6
1.4 Herausforderungen .....	6
1.5 Vorgehen.....	7
1.6 Struktur und Leserführung .....	8
1.7 Vergleichbare Arbeiten .....	8
<b>2 Analyse.....</b>	<b>9</b>
2.1 Stakeholder .....	9
2.2 Problemdomäne .....	10
2.3 Anwendungsdomäne .....	12
2.4 Risiken .....	12
2.5 Schnittstellen und Datenbestände .....	13
2.6 Texas Holdem Poker Regeln.....	15
2.7 Texas Holdem Poker Strategien .....	16
2.8 Technische und mathematische Anforderungen.....	21
2.9 Modellwahl .....	26
<b>3 Umsetzung.....</b>	<b>29</b>
3.1 Gliederung Realisierungseinheiten .....	29
3.2 Evaluierung.....	29
3.3 RE 1 - Minimum Viable Product (MVP) mit Q-Learning .....	34
3.4 RE 2 - Deep Q Networks.....	38
3.5 RE 3 - Double Deep Q Networks.....	43
3.6 RE 4 - Prioritized Experience Replay.....	48
3.7 RE 5 – Modelleinstellungen und Testing.....	54
3.8 Features .....	65
3.9 Belohnungssystem.....	76
<b>4 Diskussion .....</b>	<b>78</b>
4.1 Rekapitulation Modell-Entwicklung .....	78
4.2 Spielweise der Agenten .....	79
4.3 Lokale Auswertungen .....	82
4.4 AICrowd .....	86
4.5 Lessons Learned.....	88
<b>5 Fazit .....</b>	<b>89</b>
5.1 Agile Entwicklung .....	89
5.2 Projekt ausserhalb der Komfortzone.....	89
5.3 Häufige Einreichungen.....	89
5.4 Grenzen .....	89
5.5 Erreichtes.....	89
5.6 Ausblick.....	90
5.7 Schlusswort .....	91
<b>6 Projektdurchführung - Einfluss Pandemie.....</b>	<b>92</b>
<b>7 Developer Guide.....</b>	<b>93</b>

7.1	Systemarchitektur .....	93
	<b>Abkürzungsverzeichnis und Glossar .....</b>	<b>102</b>
	Literaturverzeichnis .....	105
	Abbildungsverzeichnis .....	108
	Tabellenverzeichnis .....	109
	<b>Anhang .....</b>	<b>110</b>

## 1 Einleitung

### 1.1 Ausgangslage

Das Institut für Data Science an der Fachhochschule Nordwestschweiz erarbeitet in verschiedenen Anwendungsszenarien die Möglichkeiten durch Machine Learning Modelle. Im Bereich des maschinellen Lernens werden häufig Durchbrüche erlangt, in dem Modelle entwickelt werden, die komplexe Abläufe erkennen und daraus lernen sollen. Spiele eignen sich daher besonders gut. Mit dieser Arbeit wird die Anwendung von Machine Learning anhand des Spiels Texas Holdem Poker überprüft.

### 1.2 Zielsetzung

Auftrag dieser Arbeit ist es, Modelle zu entwickeln, die auf das Poker-Spiel angewendet werden sollen. Die zu verwendenden Poker-Regeln sind vom Spiel Texas Holdem Poker vorgegeben. Das Projektteam ist frei in der Auswahl, welche Machine Learning Techniken angewendet werden.

Die entwickelten Inkremeante der Modelle werden zudem auf AICrowd hochgeladen, um gegen andere Implementierungen anzutreten.

### 1.3 Einschränkungen

Im Projektrahmen bestehen einige Einschränkungen, gegeben durch die Infrastruktur:

1. Die maximale Ausführungszeit der *declare\_action* Methode soll 0.04 Sekunden nicht überschreiten.
2. Nachfolgende Abhängigkeiten und Versionen sind für eine erfolgreiche Ausführung zugelassen:
  - a. Python 3.6 - Tensorflow 2.1 - Pytorch 1.4 - scikit-learn 0.22.1
3. Es stehen keine vollständige Texas Holdem Poker Aufzeichnungen zur Verfügung.
4. Aus der AICrowd Infrastruktur erhält man nach Ablauf des Testdurchgangs lediglich die Stack-Größe. Andere Rückmeldungen aus der Infrastruktur stehen nicht zur Verfügung.

### 1.4 Herausforderungen

Im Projektteam sind keine fundierten Kenntnisse in Python und Machine Learning vorhanden. Insbesondere zu Beginn des Projektes wurde daher ein hoher Initialaufwand im Bereich dieser beiden Themen eingeplant.

Die zu entwickelnden Modelle sind noch nicht geläufig und müssen erarbeitet werden. Zudem stehen mit dem gegebenen Rahmen viele unterschiedliche Technologien zur Verfügung, die angewendet werden können. Der Freiraum sowie die fehlende Erfahrung

mit maschinellen Lernalgorithmen führen daher zu höherem Risiko für falsche Entscheidungen. Die Arbeit benötigt aufgrund dessen eine gute Planung, um schnell auf schlecht performante Modelle zu reagieren und durch bessere zu ersetzen.

## 1.5 Vorgehen

Die Projektplanung steht in Form einer Grobplanung zur Unterteilung der Entwicklungsphasen sowie eines Controllings der geleisteten Stunden zur Verfügung.

Das Projekt wird in sechs Phasen unterteilt, davon fünf Realisierungseinheiten. Jede Realisierungseinheit definiert genaue Ziele. Daraus folgende Aufgaben werden im Projektteam über den regen Austausch festgelegt und verteilt.

Gemäss den unter Kapitel 1.4 dargelegten Herausforderungen verfügt das Projektteam über keine fundierten Kenntnisse in Machine Learning und Python. Aufgrund des grossen Interesses haben sich dennoch beide Mitglieder entschieden, diese Herausforderungen anzugehen. Um ein möglichst sicheres Vorgehen festlegen zu können, haben sich die Autoren in ein von der Fachhochschule Nordwestschweiz angebotenes Modul Machine Learning [1] eingeschrieben und zusätzlich einen Python Kurs [2] über einen öffentlichen Kanal belegt. Punktuell werden weitere Kurse belegt auf der Coursera Plattform [3] belegt, um Wissen für die Optimierung der entwickelten Modelle zu sammeln.

Die Vorgehensweise zur Entwicklung von Modellen wird zu einem Grossteil aus den Learnings der Machine Learning Unterlagen nachempfunden. Die dort empfohlene Vorgehensweise besagt insbesondere:

1. Mit einem **simpelen Algorithmus** starten, welcher schnell implementiert ist. Dadurch kann das Potenzial erkannt werden, ob das gewählte mathematische Modell auf die Problemdomäne angewendet werden kann.
2. **Lernfortschritte** visualisieren, um festzustellen, ob das entwickelte Modell mehr Informationen in Form von Features benötigt.
3. **Manuelle Fehleranalyse** durchführen, um offensichtliche Fehler des Modells zu erkennen und zu adressieren. Allenfalls kann sogar eine systematische Fehlerquelle erkannt werden, die auf einen expliziten State-Zustand zurückzuführen ist. [4]

Das Team hat sich dieser Vorgehensweise angenommen und daraus die folgenden Schritte abgeleitet:

1. Problemdomäne verstehen und passende, mathematische Modelle finden.
2. Aufbau des mathematischen Modells in möglichst einfacher Variante.
3. Potenzial des Modells erkennen und entscheiden, ob das Potenzial für die weitere Verfolgung des Modells spricht. Andernfalls zurück auf Schritt 1.
4. Während den Realisierungseinheiten auf das Modell aufbauen, weiterentwickeln und dabei jeweils die vorher erwähnten Schritte **Lernfortschritte** sowie **manuelle Fehleranalyse** für die weitere Aufgaben Priorisierung nutzen.

## 1.6 Struktur und Leserführung

Der vorliegende Bericht beschreibt die Erarbeitung sowie die gewonnenen Erkenntnisse im Projektrahmen. Es werden erfolgreiche wie auch fehlgeschlagene Versuche dokumentiert.

Die Gliederung wird dem allgemeinen Projektvorgehen angeglichen. Der Aufbau führt von der mathematischen Erarbeitung der Problemdomäne, Wahl der Modelle, Entwicklung der Modelle im Laufe der Realisierungseinheiten bis zur finalen Einstellung des Netzwerks. Zum Schluss werden die Resultate mit der Zielsetzung verglichen. Einfach ausgedrückt:

1. Mathematische Annäherung und darauf basierende Wahl des Modells
2. Entwicklung und stetige Weiterentwicklung der Modelle
3. Testing sowie Vergleich der entwickelten Modelle
4. Resultate und erreichte Zielsetzung

Aufgrund der theoretischen Natur des Projekts ist im vorliegenden Bericht jeweils eine Erläuterung mitgeliefert, um allgemein die Überlegungen wie auch Verständlichkeit der Arbeit besser aufzuzeigen.

## 1.7 Vergleichbare Arbeiten

Machine Learning Ansätze werden bereits bei einer Vielzahl von Spielen eingesetzt, um eine möglichst hohe Wertung zu erreichen. Die Komplexität der Spiele ist dabei breit gestreut und reicht von ursprünglichen Arcade Spielen, wie Breakout aus dem Jahr 1976 [5], bis hin zu moderneren Multiplayer Spielen wie StarCraft II aus dem Jahr 2010. [6]

Poker bildet dabei keine Ausnahme. Auch im Rahmen von wissenschaftlichen Publikationen existieren bereits Ansätze, in denen Agenten das Pokerspielen beigebracht wird. Darunter sind auch solche, die grosse Ähnlichkeit mit dieser Projektarbeit aufweisen. In ihrer Arbeit *Building Poker Agent Using Reinforcement Learning with Neural Networks* beschreibt Annija Rupeneite beispielsweise ein Szenario, in dem ebenfalls optimale Pokerstrategien auf Basis von Umgebungsinformationen und dem Verhalten der Gegenspieler ausgewählt werden. Eine weitere Gemeinsamkeit ist der Einsatz von Q-Learning. [7]

## 2 Analyse

Dieses Kapitel beschreibt die technischen Anforderungen an die Arbeit. Es werden unter anderem die Anwendungsdomäne, Problemdomäne, Risiken, Poker Regeln und Strategien für die spätere Implementation von Features vorgestellt.

### 2.1 Stakeholder

Nachfolgend sind die im Projekt involvierten Parteien zusammengefasst.

#### 2.1.1 Fachhochschule Nordwestschweiz

Das Projekt Poker-Bot Competition wird vom Institut für Data Science in Auftrag gegeben. In diesem Fall sind die Kunden zeitgleich für die Projektbetreuung zuständig.

##### Kunde und Projektbetreuung

Institut für Data Science

Prof. Dr. Manfred Vogel

[manfred.vogel@fhnw.ch](mailto:manfred.vogel@fhnw.ch)

Dominik Frefel

[dominik.frefel@fhnw.ch](mailto:dominik.frefel@fhnw.ch)

##### Anforderung

Die Kunden wünschen die durchgeföhrten Schritte und Überlegungen nachvollziehen zu können. So sollen gewonnene Erkenntnisse und Schlussfolgerungen zur Lösung des zugrundeliegenden Problems dokumentiert werden.

#### 2.1.2 AICrowd

Die entwickelten Agenten werden auf der AICrowd Infrastruktur [8] bereitgestellt. Die AICrowd Infrastruktur stellt eine Spielwiese für die hier entwickelten Agenten sowie auch Agenten aus anderen Arbeiten dar. Dadurch können unterschiedliche Agenten-Implementierungen und deren gewählten Modelle und Strategien verglichen werden. Ziel dabei ist es, möglichst viele Iterationen auf der Infrastruktur bereitzustellen, um den Entwicklungsfortschritt erkennen zu können. Die Kunden erhalten dadurch ebenfalls die Möglichkeit, den Fortschritt zu beobachten. AICrowd steht im direkten Systemkontext.

#### 2.1.3 Studierende

Die Arbeit wird durch die berufsbegleitende Studenten Vito Cudemo und Michael Schärz durchgeführt. Sie sind mit jeweils einem 65% und 60% Pensum neben dem Studium berufstätig.

Ziel für die Studenten ist eine erfolgreiche Umsetzung der Arbeit. Sie präsentieren den Kunden zwei-wöchentlich gewonnene Erkenntnisse und daraus folgende Schlussfolgerungen.

Die Studierenden haben sich auch aufgrund persönlicher Interessen in Machine Learning für das Projekt entschieden und sich darauf beworben.

Vito Cudemo

[vito.cudemo@students.fhnw.ch](mailto:vito.cudemo@students.fhnw.ch)

Michael Schärz

[michael.schaerz@students.fhnw.ch](mailto:michael.schaerz@students.fhnw.ch)

## 2.2 Problemdomäne

Dieses Kapitel beschreibt die grundsätzlich erkannte Problemdomäne. In der Entwicklung werden Modelle eingesetzt, die auf diese angewendet werden können. Dieser Teil wird gemäss Kapitel 1.5 Vorgehen vorgezogen, um von Beginn an auf funktionierende Modelle zurückgreifen zu können.

### 2.2.1 Belohnung

Im Spiel Texas Holdem Poker besteht das Ziel, einen möglichst hohen Cashgame Stack zu erreichen. Werden Chips gewonnen oder verloren, wird im Bericht allgemein von einer Belohnung gesprochen, die positiv oder negativ sein kann. Die Belohnung ist somit ein zentraler Bestandteil. Nachfolgend sei ein Denkbeispiel gegeben.

1. Ein Baby sitzt auf dem Boden. Vor dem Baby liegt eine magische Kiste.
2. Das Baby ist neugierig und nimmt die Kiste in die Hand.
3. Es fängt an, daran zu reissen, die Kiste zu schütteln und zu werfen. Es geschieht noch nichts.
4. Dann plötzlich: ein korrekter Händegriff, jeweils die Kanten der Kiste gepackt, gefolgt von einem Auseinanderreissen, führt dazu, dass sie sich öffnet.
5. In der Kiste befinden sich Süßigkeiten.

Das Baby hat also durch Ausprobieren herausgefunden, wie sich die Kiste öffnet und Süßigkeiten als Belohnung erhalten. Liegt ein Tag später wieder dieselbe Kiste da, kann es gut sein, dass es wieder üben muss – das Ziel aber schneller erreicht, bis es mit der Kiste keine Probleme mehr hat.

Die Denkweise lässt sich nun auch auf Poker anwenden. Ähnlich der oberen Beschreibung müssen im Poker eine Reihe von Aktionen durchgeführt werden, bis eine Belohnung erreicht wird. Solange die Pokerrunde nicht abgeschlossen ist, erhält der Agent keine Belohnung.

Das Problem der Bestimmung der Handlungen, die zu einem bestimmten Ergebnis führen, ist bekannt als Credit Assignment Problem. Dieses fundamentale Problem gilt es zu lösen, denn die Interaktionskette bis zu einer entsprechenden Belohnung kann viele Aktionen erfordern. [9, p. 17]

### 2.2.2 Texas Holdem Poker

Texas Holdem Poker ist ein komplexes Karten-Spiel mit vielen unterschiedlichen Eigenschaften:

1. Eine Poker-Runde definiert alle Zustände einer einzelnen Runde – von *preflop* bis zum abschliessenden *showdown*.
2. Eine Poker-Runde besteht aus vielen aufeinanderfolgenden Entscheidungen.
3. Es gibt unterschiedliche Aktionen (*fold*, *call* und *raise*) sowie je nach Entscheidung auch unterschiedlich grosse Beträge (*amount*).
4. Mehrere aneinander gereihte Entscheidungen führen zu einem Ergebnis, wobei die Kette der Entscheidung wichtig ist.
5. Diese Entscheidungsketten – oder auch Pfade durch das Spiel – sind abhängig vom allgemeinen Spiel-Zustand. So kann derselbe Pfad in der einen Runde zu grossen Gewinnen und in der nächsten zu grossen Verlusten führen. Das Spiel ist somit nicht deterministisch. Der Weg zum Gewinn ist stark abhängig von anderen Faktoren, wie zum Beispiel Anzahl Spieler, Karten auf der Hand, Karten auf dem Tisch und so weiter.
6. Ziel des Spieles ist es, solche Entscheidungspfade zu erkennen und, basierend dem Spielzustand, die richtige Entscheidung zu treffen. Eine richtige Entscheidung kann hier beispielsweise auch sein, aus dem Spiel auszutreten (*fold*), wenn kein Gewinn mehr zu erwarten ist, aber auch genügend lange mitzuspielen, um überhaupt in eine Gewinn-Situation zu geraten.

Für ein Modell ist es also wichtig, je nach Zustand zu erkennen, welcher Pfad zu welcher Belohnung geführt hat.

#### 2.2.2.1 Erkenntnisse

Gemäss den Beschreibungen oben ist implizit gegeben:

1. Ein grosser Zustandsraum muss abgebildet werden können, da im Poker viele Parteien pro Spiel grossen Einfluss haben. Seien dies die Gegner, deren Spielweise, Karten in der Hand oder auf dem Tisch.
2. Das mathematische Modell hinter dem Modell muss das Credit Assignment Problem (CAP) adressieren können.
3. Belohnungen müssen rückwirkend dem Modell für das Training zur Verfügung gestellt werden können.

### 2.2.3 Multi-Agent Umgebung

Auf der AICrowd Infrastruktur stehen fünf weitere Modelle bereit, die für die Evaluierung verwendet werden. Das Setup entspricht einem Multi-Agenten-System (MAS). Multi-Agenten-Systeme seien dabei definiert als multiple, miteinander im Austausch stehende, intelligente Systeme, die sich an die Umgebung anpassen. [10]

In diesem Kontext lösen die Agenten nicht ein gemeinsames Problem. Die Performance des zu entwickelnden Modells ist aber abhängig von der allgemeinen Performance aller Agenten.

Das selbstlernende Modell hat das Ziel, die Belohnung zu maximieren. Sind die Gegner jedoch schwach, kann es dazu führen, dass das Modell allgemein lernt, immer den gesamten Stack einzusetzen, da damit die höchste Profitabilität erreicht werden kann.

#### 2.2.4 Zusammengefasst

Grobziele einer erfolgreichen Lösung aus Sicht des entwickelten Modells sind:

1. Das Modell ist lernfähig und kann mit gesammelten Erfahrungen trainiert werden.
2. Die Balance zwischen *fold*, *call* und *raise* finden
3. Basierend dem Risiko kann die Höhe des Einsatzes festgelegt werden.
4. Basierend auf Gegner-Verhalten reagiert das Modell, um die Gewinnchance zu erhöhen.

Im Laufe der Entwicklung wird jedes dieser Probleme adressiert.

### 2.3 Anwendungsdomäne

Die entwickelten Lösungen sind auf die Problemdomäne von Poker ausgerichtet. Entsprechend der Erläuterungen im vorherigen Kapitel ist für die korrekte Konvergenz der Modelle die Belohnung von zentraler Bedeutung.

Um die Anwendungsdomäne nicht einzuschränken, werden einzelne Bestandteile in Komponenten aufgeteilt. Je nach Anwendung kann dann beispielsweise die Belohnungskomponente ersetzt werden. Dies vereinfacht zudem das Testing und Training – die Komponenten werden dynamisch bei der Erzeugung angegeben.

### 2.4 Risiken

Dieses Kapitel beschreibt die Risiken, die in der Analyse-Phase identifiziert wurden. Eine grafische Darstellung der Auswirkungen identifizierter Risiken auf den Projektverlauf kann der Grobplanung im Anhang entnommen werden.

#### 2.4.1 Fehlende Machine Learning Kenntnisse

Zum Projektbeginn ist kein Machine Learning Wissen im Team vorhanden. Das nötige Grundwissen wird durch den Besuch des Moduls Machine Learning erarbeitet. [1] Damit Informationen aus dem Gebiet möglichst schnell umgesetzt werden können, wird das Modul in der online Variante durchgeführt. Dadurch kann die Zeiteinteilung flexibler gestaltet werden als in der regulären Vorlesung. Ziel ist es, den Coursera Kurs bis zum Ende der fünften Projektwoche abgeschlossen zu haben. Der hohe Lernaufwand führt dazu, dass zu Beginn dieser Arbeit weniger Stunden in die Umsetzung investiert werden können. Zudem sind für die drei Übungen, die absolviert werden müssen, jeweils zwei Wochen vorgesehen, in welchen die Sollstunden für die Bachelor-Thesis reduziert wurden. Das betrifft die Projektwochen 4, 5, 8, 9, 13 und 14.

## 2.4.2 Fehlende Python Kenntnisse

Zusätzlich zu den Machine Learning Grundlagen fehlen auch Kenntnisse in Python. Um diesem Umstand zu begegnen, erarbeiten sich die Studierenden das nötige Wissen. Die Grundkenntnisse werden in den ersten zehn Projektwochen mit Hilfe eines Python Tutorial auf Udemy erworben. [2]

## 2.5 Schnittstellen und Datenbestände

Nachfolgend werden die im Projektrahmen verwendeten Datenbestände sowie die zur Verfügung stehende Umgebung zusammengefasst.

### 2.5.1 Environment

Als Versionsverwaltung wird Gitlab von der Fachhochschule Nordwestschweiz verwendet (<https://gitlab.fhnw.ch>). Die Konfigurationsdateien für die entwickelten Agenten stehen für Windows sowie Unix-Systeme bereit.

### 2.5.2 PyPokerEngine

Die PyPokerEngine stellt die gesamte Poker-Infrastruktur für die zu entwickelnden Algorithmen zur Verfügung. Dazu gehören unter anderem die Regeln des Pokerspielens, der Dealer sowie eine Basisklasse, auf deren Basis Spieler implementiert werden können.

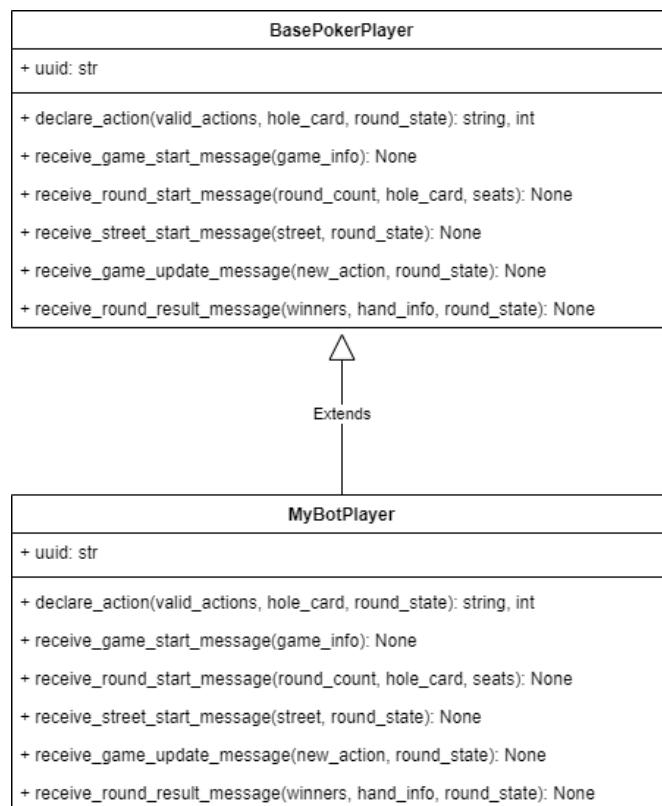


Abbildung 1 Klassendiagramm BasePokerPlayer

Abbildung 1 Klassendiagramm BasePokerPlayer zeigt die sechs Methoden, die für eine eigene Klasse überschrieben werden müssen. Die Klasse MyBotPlayer steht in dieser Abbildung stellvertretend für den eigenen Spieler. An dieser Stelle ist auch wichtig zu erwähnen, dass die Klasse BasePokerPlayer noch über weitere Methoden verfügt, um durch die PyPokerEngine eingebunden werden zu können. Die zusätzlichen Methoden sind jedoch für die Entwicklung des eigenen Spielers irrelevant und wurden weggelassen, um das Klassendiagramm übersichtlicher zu gestalten. Die Parameter der Methoden sind als JSON-Objekte abgebildet. Über folgende URL kann eine umfassende Dokumentation samt Beispielen abgerufen werden: [https://github.com/ishikota/PyPokerEngine/blob/master/AI\\_CALLBACK\\_FORMAT.md](https://github.com/ishikota/PyPokerEngine/blob/master/AI_CALLBACK_FORMAT.md) [11]

#### 2.5.2.1 Attribute BasePokerPlayer

##### **uuid**

Die uuid ist eine eindeutige Zeichenkette, welche die PyPokerEngine jedem eingebundenen Spieler zuweist, um zwischen ihnen unterscheiden zu können.

Diese Information kann beispielsweise verwendet werden, um die Parameter der einzelnen Methoden nach der eigenen uuid zu durchsuchen, um basierend dem Ergebnis gewisse Aktionen durchzuführen.

#### 2.5.2.2 Methoden BasePokerPlayer

##### **declare\_action**

Diese Methode bildet den Entscheid über die auszuführende Aktion des Spielers ab. Mögliche Aktionen sind fold, call und raise. Zudem muss ein Betrag für die gewählte Aktion angegeben werden.

##### **receive\_game\_start\_message**

Die Methode receive\_game\_start\_message wird nur einmal zu Beginn des Spiels aufgerufen. Über die Parameter können Informationen zum geltenden Regelwerk in Erfahrung gebracht werden.

##### **receive\_round\_start\_message**

Diese Methode wird zum Beginn jeder Runde aufgerufen. Sie stellt Informationen wie die aktuelle Rundenzahl und die eigenen Handkarten zur Verfügung.

##### **receive\_street\_start\_message**

Zu Beginn jeder einzelnen Street wird die Methode receive\_street\_start\_message aufgerufen.

##### **receive\_game\_update\_message**

Für jede getätigte Aktion eines Spielers wird die Methode receive\_game\_update\_message aufgerufen. Basierend auf den Informationen in den Parametern können beispielsweise Statistiken über das Verhalten der Gegner geführt werden.

##### **receive\_round\_result\_message**

Diese Methode wird am Ende jeder Runde aufgerufen. In ihr werden die Gewinner bekannt gegeben.

### 2.5.3 Baseline Spieler

Zusätzlich zum MyBotPlayer sind Baseline Spieler gegeben. Die Baseline Spieler bilden dabei einfache Umsetzungen von Agenten ab, die für Tests, zu Debug-Zwecken oder auch teilweise zu Trainingszwecken verwendet werden können.

1. **BaselinePokerPlayer:** Agent der seine Aktion anhand der Gewinnchance unter Verwendung einer Monte-Carlo-Simulation bestimmt.
2. **CallbaselinePokerPlayer:** Ein Call-Agent, der nie aus einem Spiel aussteigt.
3. **ConsolePokerPlayer:** Eine Implementation, um in der Konsole als Mensch gegen die Agenten antreten zu können.
4. **RandomPokerPlayer:** Ein Agent, welcher zufällige Aktionen ausführt.

## 2.6 Texas Holdem Poker Regeln

Bei Poker handelt es sich um eine Familie von Kartenspielen. Das Spiel wird normalerweise mit einem Kartendeck von 52 Karten gespielt.

Jeder Spieler versucht dabei, aus seinen eigenen zwei Karten, den sogenannten Hole Cards sowie den Gemeinschaftskarten, auch Community Cards genannt, eine möglichst starke Hand aus den besten fünf Karten zu bilden.

POKER HAND RANKINGS									
10	J	Q	K	A					Royal Flush
7	8	9	10	J					Straight Flush
4	4	4	4	K					Vierling - Quads
10	10	10	2	2					Full House
K	9	J	5	A					Flush
7	8	9	10	J					Straße - Straight
2	2	2	6	A					Drilling - Three of Kind
8	8	7	7	Q					Zwei Paare - Two Pair
5	5	J	10	3					Ein Paar - One Pair
K	2	8	J	6					Hohe Karte - High Card

Abbildung 2 Poker Handstärken [12]

Abbildung 2 Poker Handstärken zeigt die möglichen Kombinationen, absteigend sortiert von der Stärksten zur Schwächsten.

Als Währung dienen im Poker Chips. Durch geschicktes Wettverhalten versuchen die Spieler jeweils an möglichst viele Chips zu gelangen. [13]

## 2.7 Texas Holdem Poker Strategien

Dieses Kapitel beschreibt die Grundsätze des Pokerns und die daraus abgeleiteten Features für die Algorithmen. Die folgenden strategischen Ansätze stammen alle aus der gleichen Quelle, auf welche am Ende des Kapitels verwiesen wird.

### 2.7.1 Grundsätze eigenes Verhalten

Dieses Kapitel beschreibt allgemeine Grundsätze zum allgemeinen Verhalten, die zur Abbildung des Zustandsraumes verwendet werden können.

#### 2.7.1.1 Bezug zum Stack

Das Ziel des zu entwickelnden Algorithmus liegt darin, einen möglichst grossen Cash-game Stack zu erspielen. Beim Pokern macht es keinen Unterschied, ob der Stack aus gewonnenen oder den eigenen, nicht investierten Chips besteht. Folglich ist es auch sinnvoll, bei schlechten Händen zu folden.

#### 2.7.1.2 Anzahl Spieler

Abhängig davon, wie viele Spieler an der aktuellen Runde teilnehmen, sollten verschiedene Hände gespielt werden. Diese Information ist insbesondere im Preflop interessant, da hier darüber entschieden wird, welche Hände überhaupt gespielt werden. Grundsätzlich gilt: Je weniger Spieler in der aktuellen Runde teilnehmen, umso mehr Hände können gespielt werden.

#### 2.7.1.3 Verhalten Preflop

Bei nur zwei Spielern sollte mehr als die Hälfte der Hände gespielt werden.

Bei drei oder vier Spielern sollte der prozentuale Anteil der gespielten Hände um 30 oder weniger liegen.

Bei fünf oder sechs Spielern reichen im Normalfall weniger als 20 Prozent.

#### 2.7.1.4 Verhalten im Flop

Nachdem in der Preflop Situation darüber entschieden wurde, ob eine Hand gespielt werden soll, können im Flop prinzipiell drei Fälle eintreten.

- Falls keine der gewünschten Karten erschienen ist und keine Aussicht auf Besserung für die eigene Hand besteht, kann gecheckt werden. Ansonsten empfiehlt es sich, die Hand aufzugeben.
- Hat der Flop zu einer mittelmässigen Hand geführt, kann anhand der Gegenspieler, deren Verhalten und ihren potenziellen Händen entschieden werden, ob die eigene Hand gespielt werden soll.
- Falls sich die eigene Hand durch ein offenes Draw noch im Turn oder River verbessern lässt, kann je nach Anzahl der verbliebenen Karten entschieden werden.

#### 2.7.1.5 Verhalten im Turn

Besteht durch die vierte Karte auf dem Board keine Gefahr, dass einer der Gegner eine bessere Hand erhalten hat, kann gecallt oder geraised werden, um den Pot zu vergrössern.

Verhalten sich die Gegner jedoch aggressiv oder die vierte Karte stellt tatsächlich eine Gefahr dar, sollte lediglich gecheckt oder die Hand sogar aufgegeben werden.

#### 2.7.1.6 Verhalten im River

Wurde in der letzten Spielphase ein Out getroffen, kann weiterhin gecallt oder geraised werden. Alternativ bietet sich auch ein Bluff an, falls keine der gewünschten Karten aufgedeckt wurde. Ansonsten sollte der Showdown abgewartet werden.

#### 2.7.1.7 Verlorene Hände

Verliert ein Spieler eine Hand, die er laut der Wahrscheinlichkeitsrechnung hätte gewinnen sollen, bedeutet das nicht automatisch, dass sich ein Spieler falsch verhalten hat. Da bei der Evaluation des Agenten eine grosse Zahl von Händen gespielt werden, sollten diese Fälle nicht allzu sehr in Gewicht fallen.

#### 2.7.1.8 Ausspielen der Gegenspieler

Nebst dem eignen Verhalten müssen auch die Gegner und ihr Verhalten berücksichtigt werden. Dieses Thema wird im Kapitel 2.7.2 Verhalten Gegner ausführlicher behandelt.

#### 2.7.1.9 Positionen

Die Reihenfolge, in der Spieler am Zug sind, ändert mit jeder Runde, jeweils beginnend beim Small Blind und endend beim Dealer. Grundsätzlich gilt: Je später ein Spieler am Zug ist, desto besser. Das ergibt sich aus der Tatsache, dass Spieler in einer frühen Position agieren müssen, ohne zu wissen, wie die anderen Spieler reagieren werden. Spieler in einer späten Position werden auf das bisher Geschehene reagieren können. Folglich können in einer späten Position auch schwächere Hände gespielt werden, wenn lediglich gecallt werden muss, um möglicherweise günstig einen guten Flop zu treffen. Die Position des eigenen Spielers wird zu diesem Zweck in drei Bereiche unterteilt: Frühe (early), mittlere (middle) und späte (late) Position. [14]

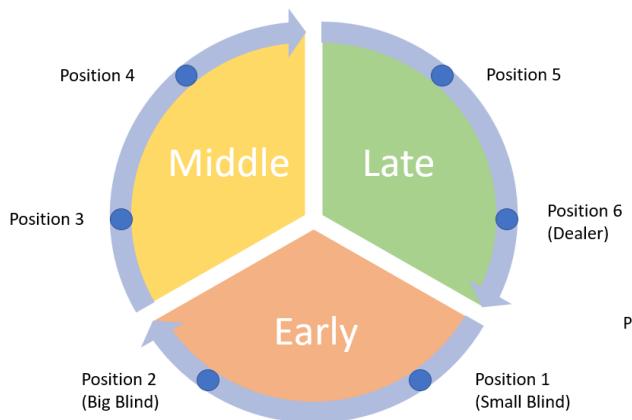


Abbildung 7 Positionen bei sechs Spielern

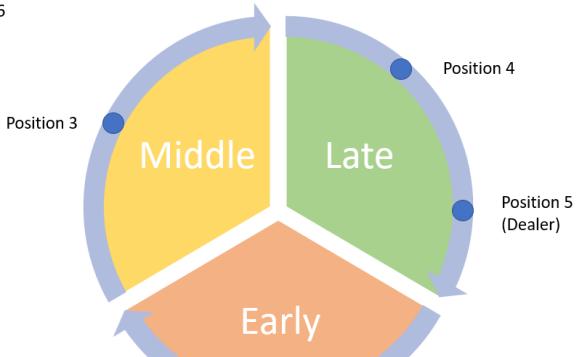


Abbildung 7 Positionen bei fünf Spielern

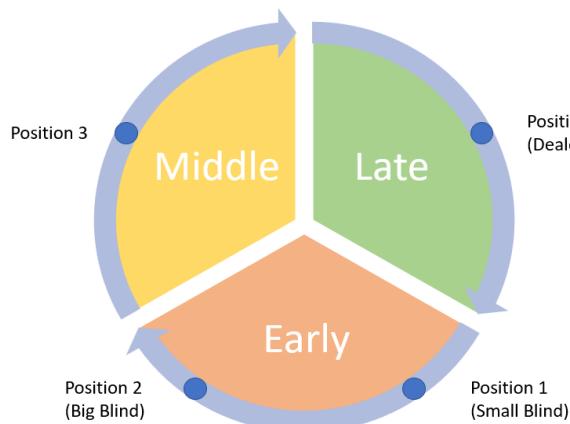


Abbildung 7 Positionen bei vier Spielern

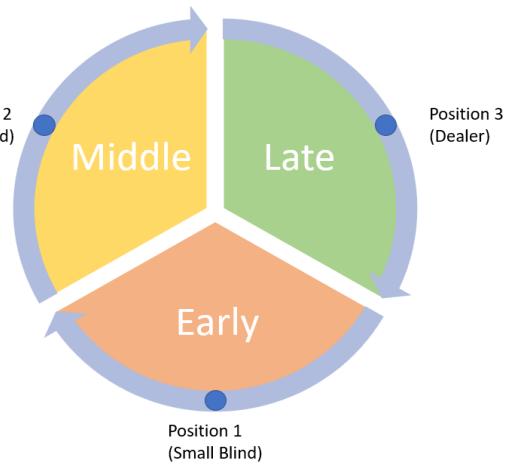


Abbildung 7 Positionen bei drei Spielern

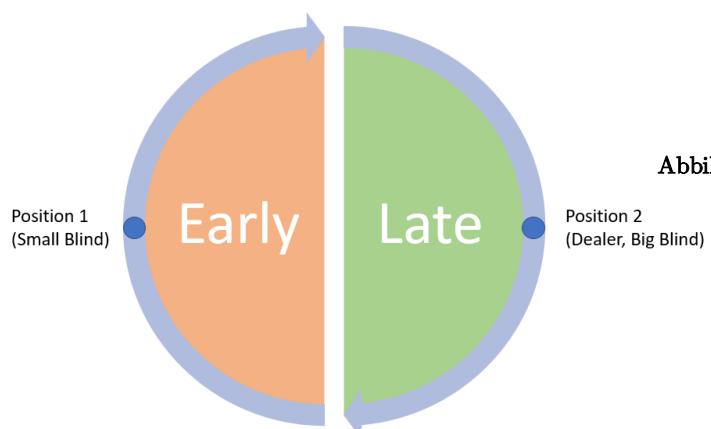


Abbildung 7 Positionen bei zwei Spielern

### 2.7.1.10 Fazit

Die beschriebenen Grundlagen bieten eine solide Ausgangslage für die Features des Algorithmus. Insbesondere die Position am Pokertisch, die aktuelle Runde (street) und die Qualität der eigenen Karten bieten sich dafür an.

### 2.7.2 Verhalten Gegner

#### 2.7.2.1 Begriffe

Beim Verhalten von Pokerspielern wird primär zwischen zwei Faktoren unterschieden: Der Qualität der Hände, die gespielt werden, und das Wettverhalten.

Spieler, welche nur gute Hände spielen, werden als *tight* bezeichnet. Das Gegenstück dazu sind Spieler, die auch schlechtere Hände spielen, diese werden als *loose* bezeichnet. Beim Wettverhalten wird zwischen *aggressiven* und *passiven* Spielern unterschieden. Aggressive Spieler zeichnen sich durch häufiges Wetten und Erhöhen aus. Passive Spieler hingegen wetten selten, gehen oft mit oder folden.

Aus jeweils einer dieser beiden Eigenschaften aus den beiden Kategorien lassen sich vier Verhaltenskategorien von Pokerspielern bilden. Jede dieser Strategie hat seine Vorteile, Nachteile und mögliche Konter-Strategien. Das folgende Bild veranschaulicht die Gegenüberstellung der vier Verhaltenskategorien. Die angegebenen Strategien stammen aus der Quelle *Poker Playing Styles*. [15]

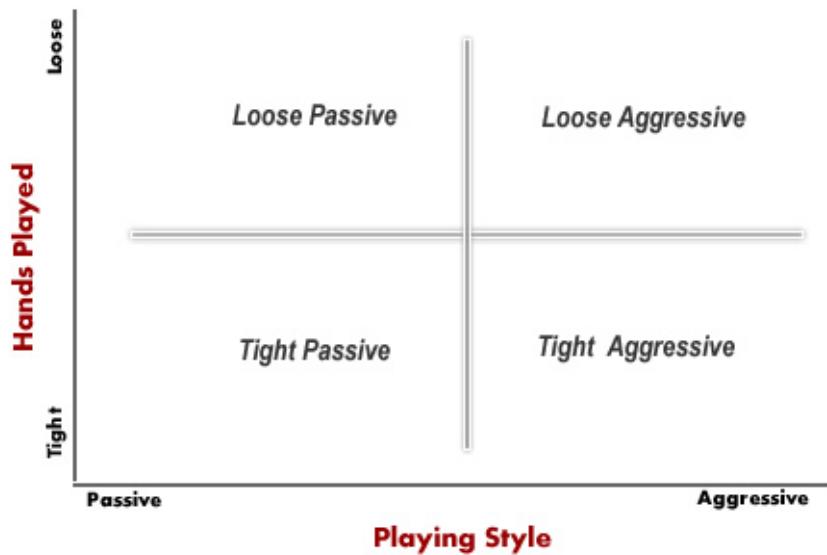


Abbildung 8 Die vier Verhaltenskategorien von Pokerspielern [15].

#### 2.7.2.2 Loose-Aggressives Verhalten

Loose-Aggressive Spieler gewinnen oft grosse Pots, zudem ist das Mitgehen für die Gegner oft sehr teuer. Dieses Verhalten kann ausgenutzt werden, indem man solche Spieler in die eigene starke Hand laufen lässt. Sprich, man lässt loose-aggressiven Spieler selbst erhöhen, statt das selbst zu tun.

#### 2.7.2.3 Tight-Aggressives Verhalten

Die tight-aggressive Spielart ist allgemein als beste Spielweise bekannt. Durch dieses Verhaltensmuster entstehen hohe Gewinnchancen bei den gespielten Händen. Gegner können durch das Image getäuscht werden. Eigene schlechte Hände werden durch Gegner oft nicht erkannt, weil sie nicht gecallt werden. Der Nachteil dieser Spielart ist die Berechenbarkeit, da das Verhalten leicht zu durchschauen ist. Am besten begegnet man solchen Spielern durch eine eigene, tight Spielweise.

#### 2.7.2.4 Tight-Passives Verhalten

Tight-Passive Spieler haben den Vorteil, dass sie keine grossen Risiken eingehen und konstante, dafür kleine, Gewinne einfahren. Diese kleinen Gewinne können im Lauf der Pokerrunde jedoch zum Problem werden, da diesen Spielern oft die Chips ausgehen. Als Konter haben sich das nicht Antreten gegen diese Spieler oder das Bluffen etabliert.

#### 2.7.2.5 Loose-Passives Verhalten

Der loose-passive Spielstil ist allgemein als schlechteste Spielweise bekannt. Dieser Stil bietet keine Vorteile, fährt zudem hohe Verluste ein. Diesen Spielern begegnet man am besten mit einer tighten Spielweise, ohne zu bluffen [16].

#### 2.7.2.6 Fazit

Aufgrund der zentralen Bedeutung des Verhaltens der Gegner und der dadurch bedingten Gegenmassnahmen sollen diese Verhaltenskategorien in den Algorithmus einfließen. Um an diese Features zu gelangen, muss das Verhalten der Gegner laufend beurteilt werden. Dafür bietet sich das Führen von Statistiken an.

### 2.7.3 Einschätzung Hand

Beim Pokern spielt die Stochastik eine zentrale Rolle. Das gilt vorwiegend bei der Einschätzung der eigenen Hand, beziehungsweise wie diese noch verstärkt werden könnte.

#### 2.7.3.1 Begriffe

Um die folgenden Überlegungen zu erklären werden zuerst zwei neue Begriffe eingeführt.

- Als *Outs* werden Karten bezeichnet, welche noch für ein starkes Blatt benötigt werden.
- Die Wahrscheinlichkeit, die Outs noch zu treffen, wird als *Odds* bezeichnet.

#### 2.7.3.2 Berechnung Odds

Die Berechnung der Odds ist insbesondere ab dem Flop sinnvoll. Mit der folgenden Formel kann die Wahrscheinlichkeit, ein Out im Turn oder River zu treffen, berechnet werden.

$$Odds = 1 - \frac{47 - Outs}{47} * \frac{46 - Outs}{46}$$

Von der garantierten Eintrittswahrscheinlichkeit wird die Wahrscheinlichkeit auf einen Treffer im Turn, multipliziert mit der Wahrscheinlichkeit auf einen Treffer im River abgezogen. Die Zahl 47 repräsentiert die Karten, die sich während dem Flop noch im Deck befinden. Die 46 steht entsprechend für die Karten, die sich während dem Turn noch im Deck befinden.

#### 2.7.3.3 Berechnung Pot-Odds

Unter Pot-Odds versteht man das Verhältnis zwischen der Grösse des Pots und der zu investierenden Chips, um diesen gewinnen zu können.

$$\text{Pot Odds} = \frac{\text{Zu investierende Chips}}{\text{Pot}}$$

Folglich lohnt es sich, eine Hand zu spielen, wenn die Odds grösser oder gleich den Pot-Odds ist.

Sind die Odds jedoch tiefer als die Pot-Odds, empfiehlt es sich, die Hand nicht zu spielen.

#### 2.7.4 Fazit

Sowohl die Odds als auch die Pot-Odds bieten sich als Features für den zu entwickelnden Algorithmus an.

### 2.8 Technische und mathematische Anforderungen

Die allgemeine Ausgangslage, Zielsetzungen, Schnittstellen sowie Poker-Regeln und Zustände sind bekannt. Dieses Kapitel beschreibt nun die mathematischen sowie technischen Anforderungen an das Modell.

#### 2.8.1 Markow-Entscheidungsproblem (MDP)

Das vorliegende Problem ist in der Mathematik als Markow-Entscheidungsproblem (MDP) bekannt. MDP ist eine mathematische Betrachtungsweise und wird in Kombination mit Reinforcement Learning seit Jahrzenten als Stand der Technik für Deep Learning angesehen. [17, p. 195]

MDP ist ein formelles Modell, um ein mögliches Ergebnis basierend dem gegebenen Zustand mathematisch zu definieren. Gemäss der allgemeinen Definition gilt:

1. In Bezug auf Reinforcement Learning beschreibt MDP eine Umgebung
2. Die Umgebung ist vollständig beobachtbar
3. Der aktuelle Zustand charakterisiert den Prozess komplett

Das entspricht nicht ganz der gegebenen Ausgangslage. Die Poker-Umgebung ist nicht vollständig beobachtbar. Tatsächlich sind in jedem Zustand viele Variablen unbekannt, wie etwa die Karten der Gegenspieler oder auch das Board.

Entsprechend kommt eine erweiterte Definition von MDP zum Zuge: Partially observable Markov Decision Process (POMDP). Diese sei folgend definiert.

Gegeben ist ein 7-Tuple

$$(S, A, T, R, \Omega, O, \gamma)$$

wobei

*S eine Gruppe von Zuständen*

*A eine Gruppe von Aktionen*

*T eine Reihe von bedingter Übergangswahrscheinlichkeit zwischen Zuständen*

*R:  $S \times T \rightarrow R$  die Belohnungsfunktion (Reward)*

*O eine Menge von bedingten Beobachtungswahrscheinlichkeiten*

*$\gamma \in [0, 1]$  der Discount – Faktor*

Der POMDP besagt nun: Zu einer Zeitperiode befinden wir uns in einem Zustand  $s \in S$ , führen eine Aktion  $a \in A$  aus, und treten mit Wahrscheinlichkeit  $T(s, a)$  in Zustand  $s' \in S$  über. Dabei kann allgemein durch die Transition eine Beobachtung  $O(s', a)$  basierend dem gegebenen Zustand und der durchgeführten Aktion erfassst sowie eine Belohnung  $r \in R$  zurückgegeben werden.

Ein Reinforcement Learning Ansatz ist entsprechend, diese Belohnung zu maximieren, indem der Agent die beobachteten Zustände erkennt und basierend der Übergangswahrscheinlichkeit zu einem gewünschten Zustand übergeht. Dieses 7-Tuple wird entsprechend in ähnlicher Form als Speicher abgebildet.

### 2.8.2 Stochastic game

Das POMDP gibt die mathematische Grundlage und entspricht in der Anwendung dem Single-Agent Fall. Gemäss Erläuterung in Kapitel 2.2.3 Multi-Agent Umgebung ist der Agent abhängig von der gesamten Umgebung. Oder anders ausgedrückt: der Spielzustand ist definiert durch die gemeinsame Aktion aller Agenten. Ein solcher Multi-Agent Fall ist in der Spieltheorie auch bekannt als Stochastic game und stellt eine Verallgemeinerung des Markow-Entscheidungsproblems dar, wobei das Tuple mit der gemeinsamen Aktion aller Agenten erweitert wird. Die Belohnung, die das Modell erhält, ist stark an die Aktion der gegnerischen Agenten gebunden.

[10, p. 6]

Im Poker haben alle Agenten dasselbe Ziel: die Belohnung und damit den eigenen Cashgame Stack zu maximieren – ein kooperativ stochastisches Spiel. Dabei erhält der Agent im Falle der PyPokerEngine ein Feedback seiner Aktion am Ende einer Runde.

Die Funktion, die auf diesen Multi-Agent Fall angewendet werden kann, ist bekannt als *optimal state-value function*. [10, p. 4]

### 2.8.3 Temporal-Difference Learning

Bisher sind folgende Voraussetzungen und Anforderungen erläutert

- CAP: Credit Assignment Problem
- POMDP: partiell observierbares Markow-Entscheidungsproblem
- Multi-Agent Fall: stochastic game

Allgemein führen diese Voraussetzungen und Anforderungen zu einer Methode bekannt als Temporal-Difference Learning (TD Learning).

Im Poker-Spiel ist der nächste Zustand unbekannt. Er ist abhängig von vielen Variablen. Was bekannt ist, ist die eigene Stärke (das heutige Wetter). Wenn das Modell diese als Gut empfindet, spielt der Agent mit und tritt in den nächsten Zustand über (das morgige Wetter). Nun kennt er den neuen Zustand und kann damit die Stärke neu evaluieren und dadurch entsprechend reagieren. Diese Erfahrung kann der Agent nutzen, um beim nächsten Mal das Wetter für übermorgen etwas besser vorherzusagen. TD Learning löst das Vorhersage-Problem mit Hilfe der state-value Funktion. Die Anwendung sei nachfolgend zusammengefasst und wird in den späteren Realisierungseinheiten von hoher Bedeutung sein:

---

Für jede Episode:

Initialisiere Zustand S

Für jeden Schritt in der Episode:

A <- Aktion durch Policy für Zustand S

Aktion A ausführen, observiere Belohnung R und  
nächsten Zustand S'

Wende die state-value Funktion an

S <- S'

solange S nicht terminiert ist

---

**Listing 1 Aktualisierung state-value mit TD Methode [9, p. 120]**

Listing 1 Aktualisierung state-value mit TD Methode zeigt das allgemeine Vorgehen, wie Temporal-Difference Methoden angewendet werden. Dieses Vorgehen wird später in den Realisierungseinheiten verwendet, um ein lernendes Modell zu erhalten.

Nachfolgend sei die *state-value* (V) Funktion gegeben.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (1)$$

Der zu aktualisierende Wert V

im State S

zu Zeitpunkt t

wird mit Geschwindigkeit  $\alpha$  aktualisiert,

wobei die Belohnung zu Zeitpunkt ( $R_{t+1}$ ) addiert wird

mit der Differenz zwischen dem nächsten Zustand ( $S_{t+1}$ ) und aktuellen Zustand ( $S_t$ ).

Die fortlaufende Aktualisierung führt allgemein zur Richtlinie für die Vorhersage von  $V_\pi$ . [9, p. 120] Mit Hilfe der Richtlinie können demnach zu einem bestimmten Zustand passende Aktionen abgeleitet werden.

Weitere Informationen zur Funktionsweise von Temporal-Difference Learning sind im Buch *Reinforcement Learning – An Introduction* von Richard S. Sutton und Andrew G. Barto gegeben. [9, p. 119]

#### 2.8.4 Supervised oder Unsupervised Learning

Eine der fundamentalen Entscheidungen zu Beginn des Projekts ist die Klärung der Frage, ob ein Supervised oder Unsupervised Learning Ansatz verfolgt werden sollte. Ausschlaggebend für den Entscheid sind allen voran die folgenden Überlegungen:

- Bei Supervised Learning Ansätzen ist die Lösung jeweils bekannt. In diesem Projekt ist das nicht zwingend der Fall, da in einer Pokerrunde die Hände der Gegner nicht in jedem Fall gezeigt werden, sondern nur, wenn es zu einem Showdown kommt. Somit ist zum Beispiel nicht klar, ob eine hohe Wette eines Gegners ein Bluff oder das Anspielen einer starken Hand war, wenn alle anderen Spieler ihre Hand bereits aufgegeben.
- Es lassen sich nur vage Schlüsse über potenzielle weitere Spielverläufe ziehen, wenn die eigene Hand beispielsweise zu früh aufgegeben oder das Wettverhalten anders gewählt worden wäre. Somit kann die korrekte Lösung nicht einfach vorgegeben werden.
- Das Projekt verwendet eine spezielle Version von Cashgame Poker. Dabei werden die Chips, die einem Spieler zur Verfügung stehen, jede Runde auf 200 zurückgesetzt. Ein passendes Testdatenset wurde nicht bereitgestellt. Der Aufwand würde einerseits durch die fehlende Python Erfahrung verstärkt, zeitgleich könnte sich das Team weniger mit der fundamentalen Problemstellung des Projektes beschäftigen.
- Unsupervised Learning versucht selbstständig, Muster und Beziehungen zwischen den zur Verfügung gestellten Daten zu erkennen. [18] Da die PyPokerEngine durch das Ausführen von Evaluationen diese Daten laufend generiert, erscheint dieser Ansatz sinnvoller.

**Entscheid:** Der Unsupervised Learning Ansatz wird verfolgt.

#### 2.8.5 Exploration versus Exploitation

Damit Unsupervised Learning Muster und Beziehungen im Poker-Zustandsraum erkennen kann, ist es wichtig, dass das Modell im Training eine Exploration durchführt. Heisst: Das Modell muss zufällige Aktionen ausführen und die Zustandsübergänge als Erfahrungen sammeln. Das Netzwerk nutzt diese Erfahrungen, um davon zu lernen.

Das Modell soll die Exploration solange durchführen, bis es genügend weiß, um mit der Exploitation – also der Verwertung des Erlernten - zu beginnen. Eine Richtlinie, die dieser Idee entspricht, ist die Epsilon Greedy Strategy.

### 2.8.5.1 Epsilon Greedy Strategy

Epsilon Greedy Strategy wird ein Algorithmus genannt, welcher eine Strategie zur Balance zwischen Exploration und Exploitation vorgibt. Nachfolgender Ablaufdiagramm soll dies graphisch veranschaulichen.

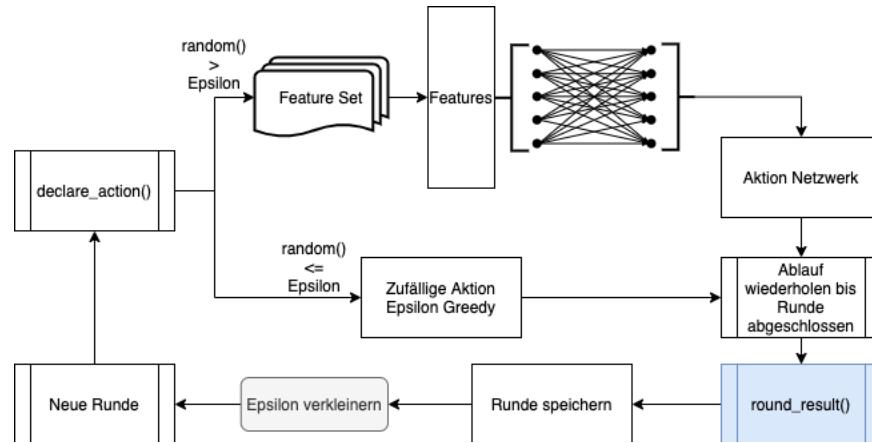


Abbildung 9 Anwendung Epsilon Greedy Algorithmus

Die in der Abbildung unten links startende Runde führt zum Aufruf der `declare_action(...)` Methode. Typischerweise beginnt das Training mit 100% Exploration. Ist ein zufällig generierter Wert im Bereich kleiner oder gleich dem Epsilon Wert, wird eine zufällige Aktion ausgeführt, ansonsten wird die Aktion durch Exploitation (das Netzwerk) berechnet. Nach Ausführung wird der Epsilon Wert, solange er nicht beim Minimum von beispielsweise 1% ist, mit dem Epsilon-Decay Wert vermindert.

Das heisst: Der Algorithmus startet mit  $\text{Epsilon} = 100\%$  Exploration und vermindert diesen Wert, bis nur noch 1% Exploration durchgeführt wird.

Wichtig ist daher bei der Auswahl der als Hyperparameter geltenden Werte (`epsilon`, `epsilon_decay` und `epsilon_min`) eine dem Zustandsraum passende Initialisierung zu verwenden. So ist bei grossem Zustandsraum, wie dies im Poker der Fall ist, eine länger andauernde Exploration hilfreich.

## 2.9 Modellwahl

Dieses Kapitel beschreibt weitere Überlegungen – aufbauend auf die vorhandene Problemdomäne gemäss Kapitel 2.2 Problemdomäne sowie den Anforderungen gemäss vorherigem Kapitel, die zur Verwendung von Q-Learning und Variationen davon geführt haben.

### 2.9.1 Reinforcement Learning Methoden

Ein Reinforcement Learning System besteht aus vier zentralen Elementen: dem Agenten, einer Policy (Richtlinie), der Belohnung sowie der Wertefunktion. Die Richtlinie beschreibt dabei das Verhalten des Agenten zu einem bestimmten Zeitpunkt.

Diese Richtlinie wird grob in zwei Sektoren unterteilt – modellbasiert sowie modellfrei.

- Modellbasierte Reinforcement Learning (RL) Methoden versuchen generell eine Wertefunktion zu modellieren. Der Algorithmus ist weniger abhängig von Exploration, stattdessen kennt er die Regeln und passt sich diesen Regeln an.
- Dagegen beruhen modellfreie RL Algorithmen auf echten Erfahrungen aus der Umgebung, welche durch Exploration gesammelt werden.

Wenn der Agent nach dem Lernen Vorhersagen über den nächsten Zustand und die Belohnung machen kann, bevor er die Aktion ausführt, handelt es sich um einen modellbasierten RL-Algorithmus. Umso mehr Variablen (beispielweise Multi-Agenten Systeme), umso schwieriger wird es, dieses Ziel zu erreichen. Modellfreie RL-Algorithmen hingegen verwenden eine Policy und versuchen diese so zu optimieren, dass die Belohnung zu maximieren. [19]

### 2.9.2 Unterschied on-policy / off-policy

Modellfreie RL-Algorithmen unterscheiden zusätzlich zwischen on-policy und off-policy Methoden. Die zu trainierende Policy sagt dem Modell, wie es sich in einer bestimmten Situation verhalten soll.

- On-Policy Algorithmen lernen den Wert basierend der ausgeführten Aktion. So geht ein On-Policy Algorithmus allgemein davon aus, dass die aktuell gültige Richtlinie (verfolgter Ansatz) auch in Zukunft gültig sein wird.
- Off-Policy Algorithmen lernen den zu optimierenden Wert unabhängig von der Handlung des Agenten. Allgemein wird eine zukünftig erwartete, diskontierte Belohnung zur Wahl der Aktion verwendet. Off-Policy Algorithmen eignen sich damit in Anwendungsdomänen, bei denen neue Wege durch den Zustandsraum gefunden werden sollen, um die Belohnung möglichst zu erhöhen. Sie sind damit stark auf genügend lang andauernde Exploration angewiesen. [9, p. 129]

### 2.9.3 Algorithmen im Vergleich

Die im vorherigen Kapitel beschriebene optimal state-value function ist für die Anwendung in einem Multi-Agenten System ausgelegt. Es soll die Anwendung des Temporal Difference Learnings Modells verfolgt werden, wodurch der Fokus auf drei Algorithmen eingeschränkt wird: Actor-Critic, Sarsa (on-policy) und Q-Learning (off-policy).

#### 2.9.3.1 Actor-Critic Methoden

Actor-Critic Methoden stellen eine Form von Algorithmen dar, die allgemein zwei miteinander funktionierende Modelle kombinieren – den Actor (Aktor) und den Critic (Kritiker).

Während der Actor die durchzuführende Aktion wählt, gibt der Critic Feedback, wie gut oder schlecht diese Aktion war. In Advantage Methoden von Actor-Critic gibt der Critic statt einfaches Feedback dem Actor an, wie viel besser oder schlechter eine andere Aktion gewesen wäre. [9, p. 331]

#### 2.9.3.2 SARSA

SARSA ist ein Akronym für State-Action-Reward-State-Action. Unter Verwendung des Temporal-Difference Learnings will SARSA durch ein State-Action Paar (Zustand-Aktion Paar) allgemein der Richtlinie folgen, die erwartete Belohnung zu maximieren. Die Richtlinie verknüpft dabei die durchzuführende Aktion bei jedem Zustand.

Das heißt: Nach jedem State-Action (S – A) Paar erhält das Modell eine Belohnung (R), führt erneut eine Aktion (A) für die nächste Belohnung (B) aus und aktualisiert nun den Belohnungswert S – A mit der Belohnung B. [9, p. 129]

#### 2.9.3.3 Q-Learning

SARSA und Q-Learning sind zwei sehr ähnliche Algorithmen – Q-Learning wird teilweise auch referenziert als SARSA-max. Im Unterschied zu SARSA verfolgt Q-Learning keiner Richtlinie und gilt damit als off-policy Ansatz. In Q-Learning wird allgemein die maximale Aktion (maximal berechneter Q-Wert) unabhängig der vorherig durchgeführten Aktion angewendet. Q-Learning verfolgt den Greedy Ansatz, während SARSA den sicheren Weg wählt.

Nachstehende Grafik soll dies verdeutlichen.

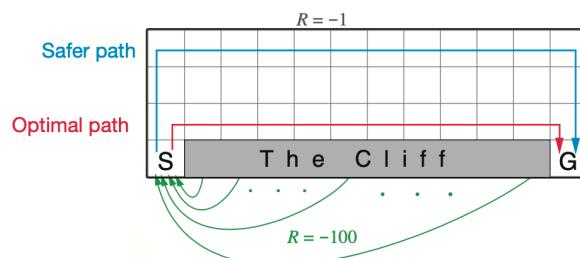


Abbildung 10 Vergleich SARSA und Q-Learning am Beispiel [9, p. 132]

Der blaue Pfad repräsentiert SARSA, der rote Pfad Q-Learning. Während SARSA den sicheren Pfad wählt, wobei eine Fehlertoleranz sichergestellt ist, verfolgt Q-Learning den Greedy Ansatz, was gleichzeitig auch der optimale, schnellste Weg ist. Ein Fehler führt aber zum Cliff und damit einer negativen Belohnung. [9, p. 132]

Für Q-Learning existieren viele Implementierungen. Einfache, die mit State-Action Tabellen implementiert werden, bis zu unzähligen Anwendungsfällen mit neuronalen Netzwerken.

#### 2.9.4 Wahl Algorithmus erste Realisierungseinheit

**Entscheid:** Das Team verfolgt den Q-Learning Ansatz.

##### Gründe

1. Q-Learning ist ein verständliches Modell, um sich mit dem Aktionsraum, dem State sowie der Temporal-Difference Learning Methode auseinanderzusetzen.
2. Das Modell kann gut ohne Anwendung von Frameworks entwickelt werden. Dies erlaubt dem Projektteam das Auseinandersetzen mit der im Modell verwendeten mathematischen Funktion und deren Auswirkungen.
3. SARSA und Q-Learning haben grosse Ähnlichkeiten. SARSA ist dabei eine generellere Version von Q-Learning. Um auf SARSA umzustellen, muss lediglich die Update-Funktion während dem Lernen angepasst werden.
4. Die gegebene Anwendungsdomäne schränkt das Modell in Bezug auf Sicherheit nicht ein. SARSA ist aufgrund der «sicheren» Policy bei Anwendungen vorzuziehen, wo beispielsweise kein Risiko eingegangen werden soll. Im Falle von Poker darf das Modell aber durchaus Risiken eingehen – das Schlimmste, was passieren kann, ist ein Verlust des Stacks.
5. Q-Learning ist allgemein verbreitet und es existieren viele Abwandlungen davon – Deep Q Networks, Distributional DQN, nashQ, und minmaxQ, um wenige Implementierungen zu nennen. Das ermöglicht, erkannte Schwächen durch Anpassung des mathematischen Modells zu adressieren, ohne dabei die fundamentale Theorie hinter dem Modell neu erarbeiten zu müssen.

### 3 Umsetzung

Dieses Kapitel beschreibt die Vorgehensweise, die entwickelten Modelle, die Modelloptimierungen sowie die gewonnenen Resultate.

#### 3.1 Gliederung Realisierungseinheiten

Die Umsetzung ist in fünf Realisierungseinheiten zu jeweils vier Wochen gegliedert.

Meilenstein	Fokus
Realisierungseinheit 1:	Minimum Viable Product mit Q-Learning
Realisierungseinheit 2:	Deep Q Networks
Realisierungseinheit 3:	Double Deep Q Networks
Realisierungseinheit 4:	Double Deep Q Networks mit Prioritized Experience Replay
Realisierungseinheit 5:	Modelleinstellungen und Testing

Die Realisierungseinheiten sind aufeinander aufbauend. Die hier dokumentierten Realisierungseinheiten umfassen jeweils die Ziele, Anforderungen, Modellwahl, Umsetzung, Resultate sowie Erkenntnisse und Beschlüsse. Der dokumentierte Stand entspricht jeweils dem finalen Entwicklungsstand am Ende der Realisierungseinheit und umfasst die fundamentalen Learnings.

Die implementierten und verwendeten Features sowie auch die Belohnungssysteme werden im Anschluss an die Realisierungseinheiten erläutert, um die Nachvollziehbarkeit der Modell-Entwicklung in den Fokus zu stellen.

#### 3.2 Evaluierung

Im Laufe der Realisierungseinheiten werden unterschiedliche Modelle vorgestellt und die Implementations dazu abgeliefert. Um die Weiterentwicklung miteinander vergleichen zu können, werden (ausser im Falle von RE1 Q-Learning) jeweils dieselben Input-Features verwendet. Die Evaluation wird immer mindestens auf eine Million Spiel-Durchläufe für das Training festgelegt.

##### 3.2.1 Mittel zur Performanceanalyse

Um die Performance der Modelle einschätzen zu können, werden Informationen in Form von Logs oder Statistiken benötigt. Nebst der Stack-Grösse stehen keine weiteren Feedbacks aus AICrowd zur Verfügung<sup>1</sup>.

---

<sup>1</sup> Siehe [Beilagen/Sitzungen/200324\_IP6\_Pokerbot\_Protokol\_Sitzung02]

### 3.2.1.1 Babysitting durch Episode-Printouts

Um die allgemeine Spielweise sowie auch weitere Runden-Zustände zu beobachten, werden während den Spiel-Einheiten Episoden auf der Konsole ausgegeben.

Die Ausgabe wird folgendermassen zusammengestellt:

```
Episode: xxxx, won: 1, acc_reward: 961.00, folded: 0, c_stack: 54531.67,  
total_bet: 180.00, hand_ranking: 0.22, risk_factor = 0.21, hand_strength: Two  
Pair, pref_allin: 0, players: 2
```

**Listing 2 Episode Printout**

Jede Episode (Runde von `receive_round_start_message()` bis `receive_round_result_message()`) zeigt uns

- *won*: Hat der Agent diese Runde gewonnen
- *acc\_reward*: Die über alle Rundenaktionen (also Interaktionskette, beispielsweise *Call – Call – Raise*) erhaltene, akkumulierte Belohnung
- *folded*: Ist der Agent ausgestiegen
- *c\_stack*: Aktueller Cashgame Stack
- *total\_bet*: Akkumulierter Einsatz über die gesamte Interaktionskette
- *hand\_ranking*: Berechnete Hand-Stärke, berechnet bei der zuletzt durchgeföhrten Aktion in der `declare_action()` Methode. Die Berechnung der Hand-Stärke wird in einer späteren Realisierungseinheit erläutert.
- *risk\_factor*: Allgemeiner Risikofaktor, berücksichtigt eigene Position, Hand-Stärke sowie skaliert an Anzahl Spieler, die noch im Spiel sind. Die Berechnung des Risikofaktors wird ebenfalls in einer späteren Realisierungseinheit erläutert.
- *hand\_strength*: Falls der Agent gewonnen hat, mit welcher Hand?
- *pref\_allin*: Hat der Agent im Preflop sein gesamtes Stack verwettet?
- *players*: Anzahl Spieler, die noch im Spiel sind (inkl. der Agent selbst)

Daraus kann gefolgert werden:

1. Zu beliebigem Zeitpunkt im Training kann beobachtet werden, wie profitabel sich die einzelnen Agenten verhalten. Es bietet eine Echtzeitüberwachung.
2. Über eine Reihe von Episoden kann abgelesen werden, wie sich der *c\_stack* entwickelt.
3. Ein trainierter Agent sollte bestenfalls *folded* = 1 haben, und wenn nicht, *won* = 1 mit hohem Anteil von *won* = 1.
4. Der Agent lernt bei schlechter Hand ein niedriges *total\_bet* zu setzen. Wir sehen, dass der Agent nichts oder nur die Blinds setzt, wenn er sich für *fold* entscheidet oder gratis Calls ausgenutzt hat.
5. Der Agent konvergiert nicht zu *pref\_allin*. Heisst: Der Agent geht nicht zu häufig All-In im Preflop Zustand.

Das Babysitting mit den Episode Printouts gibt hingegen keine detaillierten Informationen zum Spiel-Zustand. Beispiel:

1. Die Hand-Stärke ist nicht allein aussagekräftig genug, um festzustellen, ob das Modell gut oder schlecht spielt.
2. Wenige Episoden sagen nichts über die allgemeine Performance des Modells aus.
3. Große Verhaltensveränderungen können aus der Episode nicht erkannt werden. Es findet dauernd ein Model-Fitting statt.
4. Allgemeine Performance über das gesamte Spiel – der `c_stack` kann zwar steigend sein, ist jedoch vom Verhalten der gegnerischen Agenten abhängig, die allenfalls viele Chips verlieren.

Umgesetzt wird diese Analyse durch die Methode `print_episode(...)` in der Klasse [/source/agent/MVBasePokerPlayer.py]. Sie steht allen entwickelten Agenten zur Verfügung. Zusätzlich wird via [/source/configuration/agent\_config.ini] gesteuert, welcher Agent die Episoden ausgeben soll. In der Konfigurationsdatei kann zusätzlich das Flag `debug_printouts` auf `True` gesetzt werden. Dadurch werden für alle aktiven Agenten weitere Informationen auf der Konsole ausgegeben. Die Agenten sind mit Debug-Meldungen ergänzt, die ein umfassendes Bild während dem Training ermöglichen.

### 3.2.1.2 Statistik Printouts bei Spielende

Die Nachteile vom Babysitting sollen nun durch Führen und Ausgeben von Statistiken eingedämmt werden, um eine bessere Einschätzung über die Modell-Performance zu erhalten. Ein exemplarisches Beispiel sei nachfolgend gegeben:

```
Model weights have been saved to ./model/DdqnPerPlayer_6.keras
```

```
Game Statistics for: DdqnPerPlayer_6
```

```
Reward model used: Simple
```

```
Split network enabled: False
```

```
Play blinds: False
```

```
Times small called: 24
```

```
Times big called: 0
```

```
Times hand rank based raise: 0
```

```
Times all in: 0
```

```
Times preflop all in: 0
```

```
Times high bet fold: 0
```

```
# = amount > 25% initial_stack
```

```
Times low hand rank fold: 79
```

```
# = hand_rank < 0.3 fold
```

```
Average won amount: 3.3076923076923075
```

```
Total won amount: 43.0
```

```
Length won amount: 13
```

```
Average lost amount: 0.5402298850574713
```

```
Total lost amount: 47
```

```
Length lost amount: 87
```

```
Average number of round actions: 1.17204301075268. # avg Aktionen pro Runde
```

```
Percentage of games won: 13.0%
```

```
Percentage of played games won (without folded): 81.25%
```

```
Percentage of folded games: 84.0%
```

Die fett markierten Stellen sind bei den Modell-Analysen und für die Einstellungen jeweils von besonders grossem Wert. Allgemein ermöglicht diese Statistik:

1. Performance-Aussagen über das gesamte Spiel und damit einen guten, allgemeinen Überblick erhalten.
2. Basierend den Ausarbeitungen zu Poker-Strategien gemäss Kapitel 2.7 die Umsetzung des Modells überprüfen.
3. Angaben zu allgemeinem Verständnis des Netzwerks über den Zustandsraum – wie oft gewinnt der Agent wenn er mitspielt.
4. Angaben zu durchschnittlicher Spiellänge – wie viele Aktionen werden durchschnittliche pro Runde gespielt.
5. Aggressivität des Agenten – wie häufig ist im Preflop ein All-In beobachtet worden.
6. Allgemeine Einstellungen des Agenten zu Dokumentationszwecken.

### 3.2.1.3 Statistik-Mail Versand

Als wichtiges Instrument – allen voran zur Einschätzung über die AICrowd Performance sowie auch zu Dokumentationszwecken – wird die Methode `_send_statistics_mail` und damit die Funktionalität zum Versenden der Statistiken implementiert.<sup>2</sup>

Die via Mail versendete Statistik ist analog den Statistik Printouts gemäss vorherigem Kapitel, ergänzt um einige Informationen zur Umgebung:

#### **Opponent statistics aggressivity:**

```
{'cpoixfvrcctqtnzfibop': {'number_of_raises': 16955, 'number_of_actions': 16955, 'aggressivity': 1.0}, ...}
```

#### **Opponent statistics tightness:**

```
{'cpoixfvrcctqtnzfibop': {'hands_folded': 0, 'tightness': 0.0}, ...}
```

#### **Seats:**

```
[{'name': 'BaselinePlayer_1', 'uuid': 'cpoixfvrcctqtnzfibop', 'stack': 26, 'cash-game_stack': -619479.6666666666, 'state': 'folded'}, ...]
```

Dies ermöglicht zusätzlich zur Statistik die Spielweise und Strategie der gegnerischen Agenten einzuschätzen. Dieselben Statistiken werden teilweise auch als Zustandsraum für die Agenten verwendet. Die Berechnung der Gegner-Statistiken ist in Kapitel 3.8.1 Implementierte Features gegeben.

### 3.2.1.4 Debugging und ConsolePlayer

Das Babysitting wie auch die Statistiken ermöglichen es, auf schnelle Weise und ohne grossen Aufwand die Agenten zu überwachen. Es fehlt jedoch eine detailliertere Einheit, welche Hände der Agent wie spielt.

Jeweils gegen Ende einer Erweiterung sowie nach Anwendung der Trainings-Einheiten wird der Agent manuell überprüft – sei dies via Debugging, um die Zustände des Spiels

---

<sup>2</sup> Die Implementierung ist in [source/agent/MVBasePokerPlayer.py] gegeben

zu beobachten, oder via ConsolePlayer, wobei in der Konsole gegen den Agenten angekommen werden kann. Diese Methode ist zeitaufwändig.

### 3.2.1.5 Plots

Um Verhaltens-Änderungen während eines Spiels verfolgen zu können, werden Plots eingerichtet. Zu den visualisierten Informationen zählen der Cashgame Stack, die Anzahl gewonnener und verlorener Chips sowie die gewählten Aktionen. In der Konfigurationsdatei kann angegeben werden, wie gross die Step-Size sein soll. Dabei wird beispielsweise bei 500'000 Iterationen mit einer Step-Size von 10'000 jeweils der Durchschnitt dargestellt.

Über den Cashgame Stack kann beurteilt werden, ob der Agent das primäre Projektziel, einen hohen Cashgame Stack zu erspielen, erreicht. Die gewonnenen und verlorenen Chips geben Auskunft über das Wettverhalten. Der Plot über die Aktionen ist am besten geeignet, um das eigene Spielerverhalten zu beurteilen.

### 3.2.1.6 Tensorboard

Mit Hilfe von Tensorboard wird zusätzlich der Optimierungsalgorithmus (objektive Funktion) überwacht.

In der Konfigurationsdatei [source/configuration/agent\_config.ini] kann Tensorboard unter [tensorboard] für einzelne Agenten aktiviert werden. Zusätzlich muss der Pfad zu den angelegten Tensorboard-Dateien angegeben werden.

### 3.2.1.7 Declare Action Time

In der Konfigurationsdatei [source/configuration/agent\_config.ini] kann für alle Agenten der Flag *measure\_declare\_action\_time* auf *True* gesetzt werden, um die Ausführungszeit zu beobachten. Damit soll die gegebene Einschränkung von 0.04 Sekunden im Auge behalten werden.

### 3.3 RE 1 - Minimum Viable Product (MVP) mit Q-Learning

Die erste Realisierungseinheit dauert vom 2. März 2020 bis und mit 29. März 2020.

#### 3.3.1 Ziele

Für die erste Realisierungseinheit sind folgende Ziele festgelegt:

1. Die Autoren sollen sich mit der PyPokerEngine vertraut machen.
2. Es soll ein Reinforcement Learning Algorithmus eruiert werden, welcher auf ein simplifiziertes Poker-Modell angewendet werden kann.
3. Das zugrunde liegende Credit Assignment Problem soll mit dem gewählten Modell adressiert werden können.
4. Der Algorithmus soll auf die AICrowd Infrastruktur geladen werden können und lauffähig sein.
5. Als mathematisches Modell soll gemäss Analysephase Q-Learning implementiert werden.

#### 3.3.2 Anforderungen Modell

Basierend auf den gesetzten Zielen für die erste Realisierungseinheit wird nach einem Modell gesucht, welches folgenden Ablauf ermöglicht:

1. Die Aktionen *fold*, *call* und *raise* können als Aktionen festgelegt werden.
2. Das Modell ermöglicht im State die *Street* der aktuellen Poker-Runde festzulegen.
3. Basierend auf einen gegebenen Zustand, finde eine Aktion, um einen bestmöglichen nächsten Zustand zu gelangen. Bestmöglich bezieht sich hierbei auf einen möglichst maximalen Gewinn.
4. Ist ein neuer Zustand gegeben, merke die erhaltene Belohnung.

#### 3.3.3 Umsetzung Modell

Gemäss den gesteckten Zielen ist für das MVP ein vereinfachter Zustandsraum zu verwenden. Als Aktionen möchten wir *fold*, *call* und *raise* unterscheiden können. Nachfolgend sei daher die Q-Table gegeben, die von Q-Learning benötigt wird:

Tabelle 1 RE1: Q-Table für Q-Learning Modell

State	Action	call	raise	fold
Preflop		0	0	0
Flop		0	0	0
Turn		0	0	0
River		0	0	0
Showdown		0	0	0

:

Tabelle 1 RE1: Q-Table für Q-Learning Modell gibt je nach Zustand (1. Spalte) eine Aktion (1. Reihe). Die in der Q-Table ausgefüllten Werte werden zu Beginn mit 0 initialisiert und entsprechen den Qualitätswerten, die mit Hilfe der Q-Learning-Gleichung berechnet und aktualisiert werden. Die Initialisierung mit 0 entspricht dem Standard. Andere Implementationen verwenden teilweise für die Initialisierung auch zufällige Werte. [9, p. 131]

### 3.3.3.1 Definition Bellman-Gleichung

Q-Learning verwendet zur Aktualisierung der Qualitätswerte die Bellman-Funktion.

Die Gleichung sei beschrieben durch

$$Q(s, a) = r + \gamma \max Q(s', a') \quad (2)$$

Wobei die maximale, zukünftige Belohnung gegeben ist durch den aktuellen Zustand und die erhaltene Belohnung, sowie die Aufsummierung aller zukünftigen Belohnungen, die von diesem Zustand zu erreichen sind. Dabei sind gegeben

$s, a$	=	Zustand $s$ , Aktion $a$
$r$	=	Belohnung (Reward)
$\gamma$	=	Discount Faktor Lambda

Die maximale, zukünftige Belohnung durch den gegebenen Zustand dient danach dem Treffen einer Entscheidung. Daraus folgt nun die Q-Learning Gleichung.

### 3.3.3.2 Funktionsweise Q-Learning

Die Q-Learning Gleichung sei nun beschrieben durch

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha (r_{t+1} + \gamma \max Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)) \quad (3)$$

wobei Q-Wert in neuem Zustand  $t + 1$  sei aktualisiert durch

$Q_t$	=	Q-Value zu Zeitpunkt $t$ ( <i>alter Wert</i> )
$s_t, a_t$	=	Zustand $s$ , Aktion $a$ zu Zeitpunkt $t$ ( <i>durchgeführte Aktion bei altem Zustand</i> )
$\gamma$	=	Discount Faktor Lambda
$\alpha$	=	Lernrate Alpha

[20, p. 2]

Die Q-Values sind damit direkt mit der Belohnung verknüpft. Die Q-Values für ein Zustands-Aktions-Paar werden mit der diskontierten, zukünftigen Belohnung aktualisiert. Die Lernrate *alpha* gibt dabei an, wie stark diese Änderungen – gegeben durch die Erfahrungen – übernommen werden sollen.

Der Fokus von Q-Learning liegt in der Berechnung der Q-Funktion, wodurch eine Aktion daraus abstrahiert werden kann –  $\max(Q)$ . Statt der *state value function*  $V$  verwendet Q-Learning die *state-action value function*  $Q$ . Während die Wertefunktion  $V$  allgemein zu einem gegebenen Zustand der Richtlinie folgt, nutzt Q-Learning das Zustands-Aktions-Paar. [9, p. 131]

### 3.3.4 Implementation

Aus der [/agent/MVQLearningPlayer.py] Klasse seien nachfolgend einige Details gegeben. Die Implementierung wird nicht im Detail erläutert, da das einfache Q-Learning Modell später durch ein Neuronales Netzwerk ersetzt wird.

1. Das Spiel wird gestartet.
2. `declare_action(...)` wird von der PyPokerEngine aufgerufen.
3. MVQLearningPlayer.py initialisiert den Zustand, darunter
  - a. Street
  - b. win\_rate
  - c. pot
  - d. eigener Stack
4. Unter Anwendung der Epsilon Greedy Strategy und Berücksichtigung der erlaubten, nächsten Aktionen, wird nun die nächste Aktion berechnet.
5. Die Aktion wird validiert und der *amount* entsprechend den erlaubten Werten gesetzt. Im Falle von *raise* wird der festzulegende Amount unter Berücksichtigung der Gewinnchance gesetzt.
6. Am Ende einer Runde wird die Q-Table aktualisiert. Das Update wird mit Hilfe der Gleichung (2) oben durchgeführt.

### 3.3.5 Erkenntnisse

Der Q-Learning Agent [agent/MVQLearningPlayer.py] ist gemäss vorgestelltem Modell implementiert. Die Evaluation unter Verwendung der in Kapitel 3.2 Evaluierung vorgestellten Methoden hat folgende Erkenntnisse offen gelegt:

1. Der kleine Zustandsraum reicht nicht aus, damit sich das Modell an die Komplexität des Pokerspiels anpassen kann. Der Agent zeigt eine grosse Konvergenz in eine Richtung. Die Erwartungen haben die Belohnungsfunktion zu stark dominiert, wodurch das Modell eine schlecht balancierte Belohnung erhalten hat, was zu Konvergenz in eine Richtung führt.
2. Q-Learning eignet sich eher für die Anwendung bei einfachen Umgebungen. Umso grösser der Zustands- und Aktionsraum, umso schwieriger wird eine allfällige Umsetzung.
3. Bei  $n$  Zuständen ist die Q-Table eine  $n \times \text{Aktionsraum}$  grosse Matrix von Q-Werten.
4. Die Möglichkeit, auf das Credit Assignment Problem zu reagieren und die Q-Werte zu aktualisieren, ist mit massivem Aufwand verbunden.
5. Ein gutes Reward System ist notwendig, um die Qualitätswerte der Q-Table zu berechnen.
6. Das Führen einer zweiten Q-Learning Tabelle zur Generalisierung ist notwendig, um das Konvergieren der Qualitäts-Werte über mehrere Spiele besser zu homogenisieren. Grund: Mit nur einer Tabelle konnte festgestellt werden, dass die Q-Werte

starkes flackern aufweisen, da Aktualisierungen basierend einer Runde stark ins Gewicht fallen. Aufgrund des nicht deterministischen Pokerspiels sollen einzelne Runden keine grosse Anpassung in der Strategie auslösen.

### 3.3.6 Beschlüsse

Die Q-Learning Gleichung verfügt über das Potenzial, den Anforderungen zu entsprechen. Der Optimierungs-Ansatz durch die Q-Learning-Gleichung wird weiterverfolgt und wird als korrekter Ansatz interpretiert. Die oben beschriebenen Erkenntnisse können durch Erweiterungen des Modells adressiert werden.

Gemäss allgemeinem Vorgehen steht nun die Ausbaustufe von Q-Learning an – das Modell soll durch ein neuronales Netzwerk ergänzt werden, um einen grösseren Zustandsraum modellieren zu können. Damit einher kommt auch die Wichtigkeit von Feature Engineering ins Spiel.

Der Fokus der Realisierungseinheit zwei ist somit das Anwenden der Learnings aus RE1 – Q-Learning und damit der allgemeinen Q-Learning Gleichung in einem neuronalen Netzwerk.

### 3.4 RE 2 - Deep Q Networks

Die zweite Realisierungseinheit dauert vom 30. März 2020 bis und mit 26. April 2020.

#### 3.4.1 Ziele

1. Das Q-Learning Modell soll erweitert werden.
2. Ein möglichst grosser Zustandsraum muss verwendet werden können.
3. Der Zustandsraum soll durch Features abgebildet werden, um dem Modell möglichst viele Informationen für das Training geben zu können.
4. Für das neuronale Netzwerk wird auf ein Framework zurückgegriffen.

#### 3.4.2 Framework

Als Framework für das neuronale Netzwerk wird Tensorflow verwendet. Dabei wird auf eine Spezialisierung bekannt als Keras zurückgegriffen, die auf Tensorflow aufbaut. [21]

#### 3.4.3 Layer

Ein neuronales Netzwerk ist in Layer unterteilt – Input Layer, Hidden Layer und Output Layer. Jedes dieser Layer besteht aus Neuronen.

Der Input Layer nimmt die Informationen in Form von Parametern entgegen, die in das Netzwerk für den Entscheid zur Verfügung gestellt werden. Die durch die Neuronen bezeichneten Werte werden mit gewichteten Graphen zum nächsten Layer übergeben. Jedes Neuron wendet die Aktivierungsfunktion an und gibt das Resultat an die nächste Stufe von Neuronen. Die letzte Stufe bildet der Output Layer. Weitere Informationen zur Funktionsweise sind unter der angegebenen Quelle zu finden. [4]

#### 3.4.4 Neuronales Netzwerk

Dieses Kapitel beschreibt die Funktionsweise des angewendeten, neuronalen Netzwerks. Zum Einsatz kommt ein Deep Feed Forward Neural Network.

Bei jedem Aufruf der *declare\_action(...)* Methode durch die PyPokerEngine soll das mit Keras erstellte, neuronale Netzwerk für den Entscheid der nächsten Aktion verwendet werden. Als Zustandsraum für den Input kommen die durch Feature Engineering entwickelten Parameter in das Netzwerk und laufen durch die Aktivierungsfunktionen der Neuronen durch. Der Output Layer gibt uns nun drei q-Werte an – jeweils für jede Aktion einen.

Nachfolgende Grafik zeigt das Netzwerk in vereinfachter Form zum besseren Verständnis.

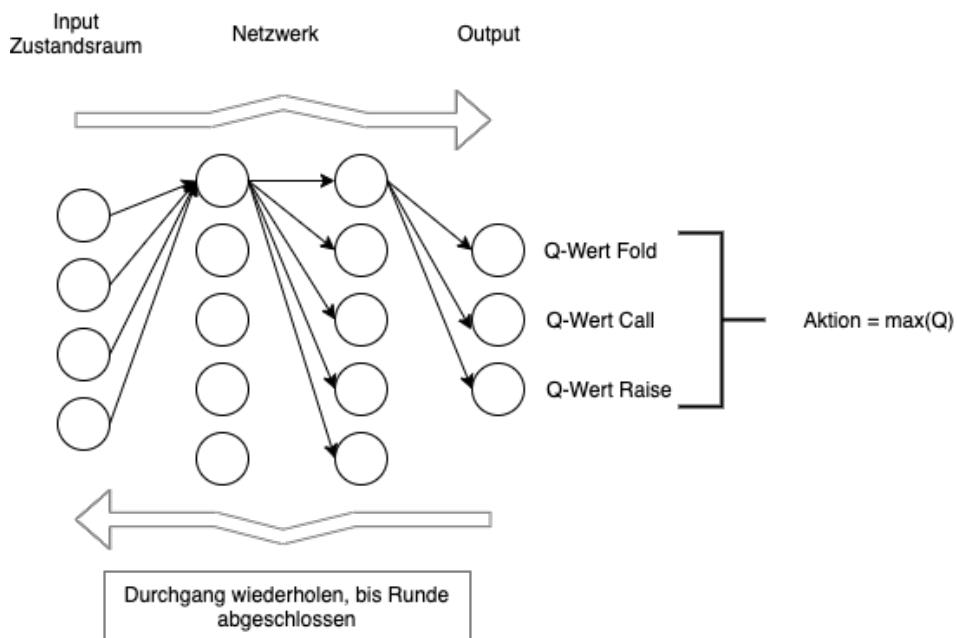


Abbildung 11 Vereinfachte Form des neuronalen Netzwerks

Dabei sind entsprechend im Output Layer in oberer Abbildung:

- Index 0 = fold
- Index 1 = call
- Index 2 = raise

Gemäss der Q-Learning Gleichung wird  $\max(Q)$  als nächste Aktion verwendet. Durch die Belohnung und das Model-Fitting werden die Gewichte zwischen den Neuronen so gesetzt, dass ein Zustand entsprechend zu der Aktion führt, die schlussendlich den Cashgame Stack des Agenten erhöhen soll.

Tabelle 2 Konfiguration [/source/agent/MVDqn.py]

Konfiguration	Parameter
Anzahl Hidden Layer	2
Grösse Output Layer	3 – fold, call, raise
Grösse Input Layer	23

### 3.4.5 Agent MVDeepQPlayer

Der Agent beinhaltet alle Hyperparameter sowie das neuronale Netzwerk. Die Hyperparameter werden mit empfohlenen Werten initialisiert und sollen erst zu einem späteren Zeitpunkt optimiert werden.

```
1. def _remember(self, state, action, reward, next_state, done):
```

*remember(...)* speichert den Zustandsübergang inklusive getätigter Aktion und Belohnung im MemoryBuffer. Das 5-Tuple *state*, *action*, *reward*, *next\_state*, *done* wird für das Lernen gemäss Q-Learning benötigt. Der MemoryBuffer entspricht in dieser Iteration einer Liste, die direkt vom Agenten geführt wird.

```
2. def _replay(self, batch_size):
```

*replay(...)* verwendet den MemoryBuffer und die als Hyperparameter eingegebene *batch\_size*, um die vorhandenen Beobachtungen zu wiederholen und dabei den Q-Learning Gradient zu optimieren. Dabei wird jeweils zufällig ein Sample des MemoryBuffers generiert und darüber iteriert. Hier findet somit das Model-Fitting - das eigentliche Lernen - statt.

Die eigentliche Vorhersage in *declare\_action(...)* findet unter Berücksichtigung der gemäss Kapitel 2.8.5.1 Epsilon Greedy Strategy erläuterten Explorations-Strategie statt:

```
3. if np.random.rand() <= self.epsilon:
4.     action = random.choice(possible_moves)
5. else:
6.     act_values = self.model.predict([state, np.o
7. nes(self.out_layer_size).reshape(1, self.out_layer_size)])
8.     action = valid_actions[np.argmax(act_values[0])]['action']
```

**Listing 3** Vorhersage unter Berücksichtigung Epsilon Greedy Strategy

Für die Vorhersage wird ein State in Form unserer Features in das neuronale Netzwerk geführt. Im Output Layer mit drei Neuronen werden schlussendlich die drei Q-Werte ausgegeben. Der Index des grössten Q-Werts entspricht der durchzuführenden Aktion.

### 3.4.6 Ablaufdiagramm

Aus den vorangehenden, einzelnen Bestandteilen setzt sich nachfolgende Ablaufdiagramm zusammen, welcher den Ablauf graphisch darstellt.

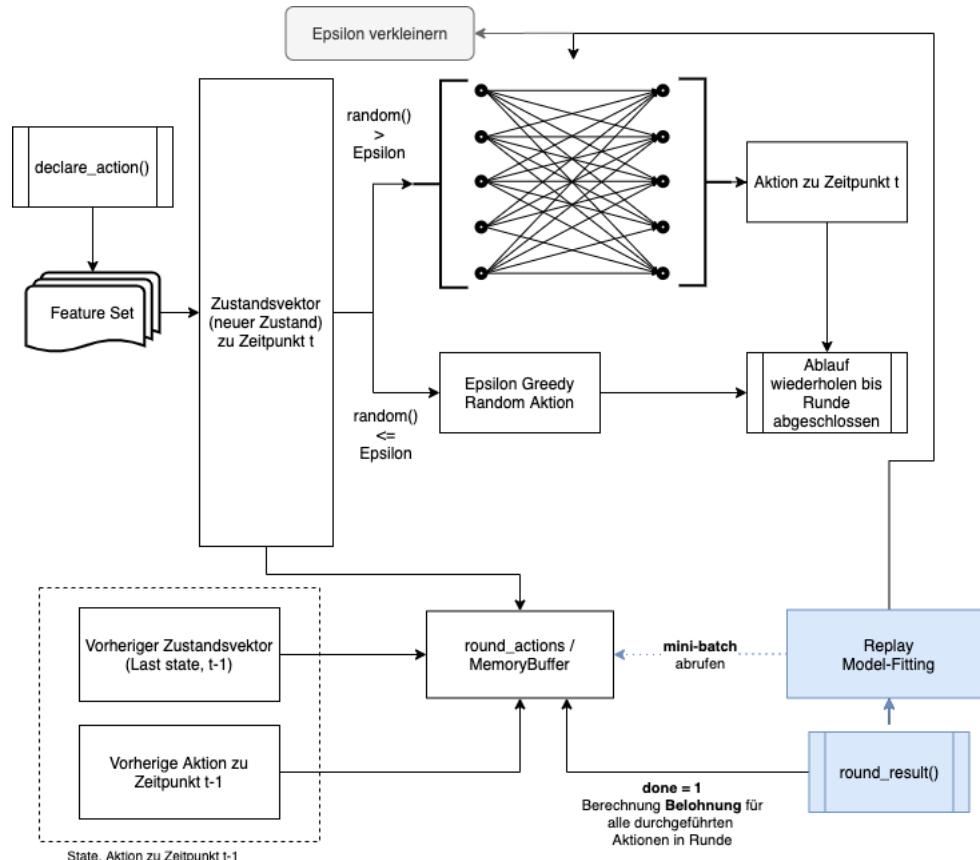


Abbildung 12 Ablaufdiagramm [MVDqn.py]

Aus Sicht des Diagramms in Abbildung 12 Ablaufdiagramm [MVDqn.py] wird der mit dem Aufruf von `declare_action()` gestartet. Der allgemeine Zustand (State) wird berechnet und unter Anwendung der Epsilon Greedy Strategy eine Aktion ausgewählt. Ab der zweiten Runde (Zeitpunkt t) wird der Zustand und die Aktion zusammen mit dem vorherigen Zustand und der vorherig durchgeföhrten Aktion (Zeitpunkt t-1) im MemoryBuffer gespeichert.

Ist die Runde abgeschlossen, wird in `round_result()` die Belohnung für alle Zustandsübergänge berechnet und im `MemoryBuffer` ergänzt. Danach findet das Model-Fitting mit der `Replay` Funktion statt.

### 3.4.7 Erkenntnisse

Der Deep Q Network Agent [source/agent/MVDqn.py] ist gemäss vorgestelltem Modell implementiert. Die Evaluation hat folgende Erkenntnisse offengelegt

1. Die Konvergenz findet langsam statt.
2. In den lokalen Auswertungen kann eine hohe Varianz festgestellt werden. Das Verhalten des Agenten ist nicht nachvollziehbar.
3. Auf AICrowd kann kein positiver Cashgame-Stack erzielt werden.
4. Das Lernen dauert zu lange. Dies ist der *replay(...)* Funktion geschuldet, die über den MemoryBuffer iteriert und dafür viel Zeit beansprucht.
5. Das Netzwerk weist hohes Flackern und damit Instabilität auf.
6. Die Erkenntnisse aus RE1 wiederholen sich. Der Zustandsraum wird aufgrund Beobachtung über Babysitting noch als zu schwach angesehen. Die Monte Carlo Simulation für die Berechnung der Gewinnchance benötigt viel Zeit und wurde deshalb auf 50 – 100 Simulationen festgelegt. Die Prüfung der berechneten Werte zeigt, dass die Vorhersage meist bei über 70% liegt, was als unrealistisch erachtet wird.

### 3.4.8 Beschlüsse

1. Die Gewinnchance soll durch ein besseres Feature ersetzt werden, um die Hand Stärke einzuschätzen.
2. Der Zustandsraum soll erweitert werden.
3. Das Model-Fitting muss schneller stattfinden, um eine bessere Performance und damit verbessertes Training zu erhalten.
4. Das Netzwerk soll stabilisiert werden.

### 3.5 RE 3 - Double Deep Q Networks

Die dritte Realisierungseinheit dauert vom 27. April 2020 bis und mit 24. Mai 2020.

#### 3.5.1 Ziele

Basierend den vorangehenden Beschlüssen werden folgende Ziele festgelegt.

1. Der Zustandsraum soll erweitert werden.
2. Die Hand-Stärke soll neu statt der Gewinnchance verwendet werden.
3. Das Model-Fitting basiert auf Matrix-Multiplikation statt auf Iteration, um die *replay* Performance zu erhöhen.
4. Das Deep Q Network soll stabilisiert werden.

#### 3.5.2 Double Deep Q Network

Die mit Deep Q Network festgestellte Varianz ist ein bereits bekanntes Problem von Q-Learning. Ursache für diese Instabilität ist die starke Überschätzung der Aktionswerte, welche mit der Belohnung verknüpft sind. Q-Learning verwendet dabei einen Single Estimator – ein Modell für die Auswahl der Aktion sowie für die darauffolgende Auswertung. Das Model-Fitting, wobei die Gewichtungen der Neuronen aktualisiert werden, führt auf dem Netzwerk zu grossen Anpassungen. [22, p. 1]

Um dem entgegenzuwirken wird ein zweites Netzwerk eingeführt. Daraus entsteht der Double Estimator. Die Idee besteht darin, häufige Anpassungen der Strategie zu reduzieren, indem die maximale, durchzuführende Aktion sowie die darauffolgende Auswertung (während dem Replay) teilweise entkoppelt werden. Es wird eine Anpassung im Replay erforderlich. Eine neue Komponente, aufbauend auf RE2, wird entwickelt – [source/components/DdqnModel.py].

#### 3.5.3 Replay mit Double Estimator

Die in Realisierungseinheit 1 entwickelte Replay Funktion des Agenten muss erweitert werden. Die Implementierung basiert auf der vom Paper Double Deep Q Network vorgegebenem Algorithmus. Die Umsetzung sieht wie folgt aus:

##### 1. Netzwerk

Neu werden zwei separate Netzwerke mit identischer Grösse generiert – das Eval- und das Target-Netzwerk.

- ```
1. self.dqn_eval = self._build_model(...)  
2. self.dqn_target = self._build_model(...)
```

## 2. Auswahl Aktion

Wird der Agent zur Aktions-Auswahl über `declare_action(...)` angefordert, wird das Eval-Netzwerk verwendet. Hier gibt es keinen Unterschied zum Single Estimator in Dqn:

```
3. q_values = self.dqn_eval.predict(
4. [state, np.ones(self.out_layer_size).reshape(1, self.out_layer_size)])
5. action = np.argmax(q_values[0])
```

**Listing 4 Prediction der Aktion mit Eval-Netzwerk aus [DdqnModel] `choose_action(...)`**

## 3. Lernen

Bei der eigentlichen Lern-Phase in der `replay(...)` Methode ist der hauptsächliche Unterschied zu erkennen. Unter Verwendung des Double Estimators findet hier das Model-Fitting statt.

```
6. q_next = self.dqn_target.predict(new_states)
# Estimate new next action target given the state
7. q_eval = self.dqn_eval.predict(new_states)
8. q_pred = self.dqn_eval.predict(states)
9. q_target = q_pred # needed to calculate differences
10. max_actions = np.argmax(q_eval, axis=1)
11. batch_index = np.arange(self.batch_size, dtype=np.int32)

12. q_target[batch_index, actions] = \
    rewards + self.gamma * q_next[batch_index, max_actions.astype(
        int)] * dones # done has been inverted
13. self.dqn_eval.fit(states, q_target, verbose=0)
```

**Listing 5 Replay mit Double Estimator aus [DdqnModel] `replay(...)`**

Im Vergleich zu einem Single Estimator findet das Model-Fitting mit Hilfe dem Eval und dem Target Netzwerk statt. Für die im Buffer vorhandenen Zustände werden dazu jeweils die Q-Werte berechnet. Das Q-Target entspricht danach der Liste, die für das Update ins Modell geführt wird. Weitere Informationen zum Algorithmus sind im Paper *Deep Reinforcement Learning with Double Q-learning* von Google DeepMind gegeben. [22]

### 3.5.4 Optimierung Model-Fitting

Zusätzliches Ziel der Realisierungseinheit ist das Optimieren des Model-Fittings. In der vorangehenden Version wird in der `replay(...)` Methode aus den Erfahrungen eine Liste mit zufälligen Erfahrungen geholt, über jede Erfahrung iteriert und darauf das Model-Fitting angewendet.

Es wird eine neue Komponente [/source/components/MemoryBuffer.py] entwickelt und die `replay(...)` Methode erweitert, um mit Matrix-Multiplikationen zu operieren. Die Umsetzung wird basierend dem von deeplearning.ai vorgeschlagenen Mini-batch gradient descent „Improving deep neural networks“ Kurs vorgenommen. [23]

#### 3.5.4.1 MemoryBuffer

Die MemoryBuffer Komponente wird nun für alle Agenten eingesetzt. Den Agenten stehen nachfolgende Funktionen zur Verfügung.

## 1. Erfahrungen speichern

```
1. def store_state(self, state, action, reward, new_state, done):
    index = self.mem_cntr % self.mem_size
    self.state_memory[index] = state
    self.new_state_memory[index] = new_state
    self.action_memory[index] = action
    self.reward_memory[index] = reward
    self.terminal_memory[index] = 1 - int(done)
```

**Listing 6** MemoryBuffer Komponente, Auszug store\_state(...)

Listing 6 zeigt die *store\_state(...)* Implementation, die vom Agenten über *remember* aufgerufen wird, um Erfahrungen zu speichern. Jeder Parameter *state*, *action*, *reward*, *new\_state* und *done* wird in separaten Listen gespeichert.

## 2. Erfahrungen abrufen

```
2. batch_indices = np.random.choice(max_memory, batch_size) # get random indices
3. states = self.state_memory[batch_indices]
4. actions = self.action_memory[batch_indices]
5. rewards = self.reward_memory[batch_indices]
6. new_states = self.new_state_memory[batch_indices]
7. terminals = self.terminal_memory[batch_indices]
8. return states, actions, rewards, new_states, terminals
```

**Listing 7** MemoryBuffer Komponente, Auszug get\_sample\_batch(...)

Der Abschnitt in Listing 7 generiert nun randomisierte Index-Werte im gültigen Bereich und gibt die entsprechenden Erfahrungen in fünf getrennten Listen an den Aufrufer zurück.

## 3. Model-Fitting mit Matrix-Multiplikation

Der Agent erhält fünf Listen und kann jeweils die Berechnungen direkt über alle Einträge durchführen.

```
9. q_target[batch_index, actions] = \
    rewards + self.gamma * q_next[batch_index, max_actions.astype(
        int)] * dones # done has been inverted, see MemoryBuffer terminal_memory
10. self.dqn_eval.fit(states, q_target, verbose=0)
```

**Listing 8** Model-Fitting mit neuer MemoryBuffer Komponente

Listing 8 zeigt nun das Model-Fitting des Agenten unter Benutzung der neuen MemoryBuffer Komponente. Auf Schlaufen kann nun verzichtet werden, stattdessen wird jeweils mit den Listen gearbeitet. Das eigentliche Aufbereiten des Q-Targets für das Model-Fitting wird mit Hilfe von Matrix-Multiplikation durchgeführt.

### 3.5.5 Anwendung an vortrainiertem Modell

In lokalen Auswertungen könnten gute Resultate erzielt werden. Als Test wird zusätzlich ein spezielles Setup angewendet: Ein mit einem Deep Q Network Agenten vortrainiertes Modell, welches zu einem Fold-Bot konvergiert ist, soll mit einem Stabilisierungs-Netzwerk eine verbesserte Konvergenz erreichen. Nachfolgende Grafik soll dies veranschaulichen.

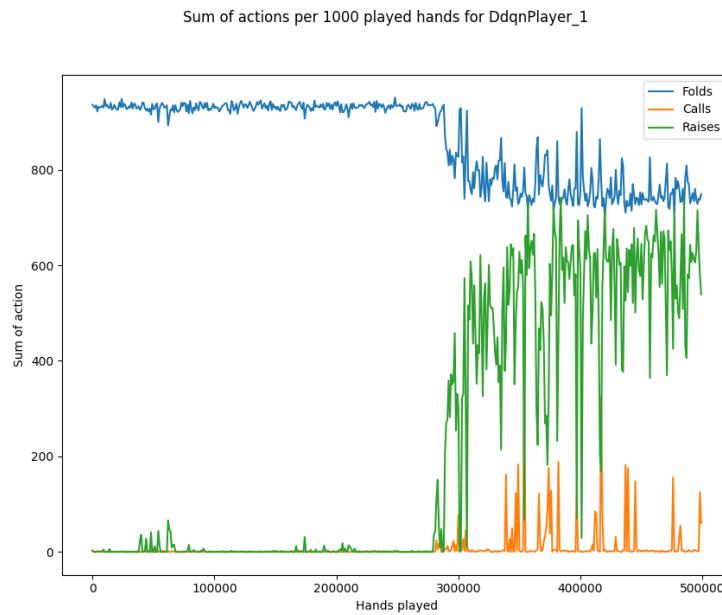


Abbildung 13 Konvergenz DdqnAgent zu Tight-Aggressive Spieler

Gemäss oberer Abbildung hat der Double Deep Q Network Agent nach ungefähr 280'000 Runden begonnen, eine verbesserte Verteilung der Aktionen zu erzielen. Die Stabilisierung des Netzwerks ermöglicht dem Modell damit eine starke Anpassung der Strategie.

### 3.5.6 Erkenntnisse

1. Die Performance gegen Baseline Player ist nicht vergleichbar mit Agenten auf AICrowd.
2. Die hohe Varianz ist noch beobachtbar, konnte jedoch gegenüber der Deep Q Network Implementierung verminder werden. Die Statistiken sowie auch Tensorboard Analysen führen aber zur Annahme, dass das Netzwerk über zu wenige Iterationen sowie mit schlechten Trainingspartnern trainiert wurde.
3. Zusätzlich wird in dieser Iteration noch mit einem komplexen Belohnungssystem gearbeitet. Das Belohnungssystem muss eine bessere Balance haben. Das Netzwerk hat Mühe, die grosse Variabilität der Belohnungen korrekt einzuschätzen.

### 3.5.7 Beschlüsse

1. Bessere Trainingspartner für lokales Training werden damit notwendig, mit unterschiedlicher Spiel Strategie. Die entwickelten Agenten sollen konfigurierbar bezüglich ihres verwendeten Q-Learning Modells, Features und Reward-Systems werden und auf dieselben Funktionen zurückgreifen.
2. Möglichkeiten sollen eruiert werden, um die Konvergenz zu beschleunigen.
3. Feature Engineering gilt nach wie vor als zentraler Bestandteil. Basierend den Analysen sollen weitere Features abstrahiert werden, um den Zustandsraum besser darzustellen.
4. Das Belohnungssystem soll besser balanciert und zusätzlich ein einfacheres Modell als Vergleich eingesetzt werden.

### 3.6 RE 4 - Prioritized Experience Replay

Die dritte Realisierungseinheit dauert vom 25. Mai 2020 bis und mit 21. Juni 2020.

#### 3.6.1 Ziele

Gemäss den in Realisierungseinheit 3 erläuterten Beschlüsse werden folgende Ziele priorisiert.

1. Die Konvergenz soll beschleunigt werden.
2. Das Netzwerk benötigt mehr Informationen, um sich an das Spiel anzupassen.  
Eine Erweiterung der Features ist notwendig.

#### 3.6.2 Prioritized Experience Replay

Im Laufe der Spiele sammelt das Netzwerk im MemoryBuffer unzählige Erfahrungen – bei jedem *declare\_action(...)* Aufruf wird der Zustand gesichert. Viele dieser Zustände sind jedoch für das Modell kaum entscheidend – beispielsweise haben Preflop Folds kaum Aussagekraft. Das Netzwerk soll bei unerwarteten Ereignissen trainiert werden – also etwa, wenn die Vorhersage komplett daneben war.

Im Paper Prioritized Experience Replay (PER) [24] wird eine Erweiterung einer Memory Implementierung beschrieben, wodurch Modell Erfahrungen mit unterschiedlicher Priorität im Replay-Buffer gespeichert und abgerufen werden. Zusätzlich wird die Methode in Kombination mit Deep Q Networks angewendet, wodurch in 42 von 49 Fällen eine massive Verbesserung beobachtet werden konnte. [24, p. 1]

##### 3.6.2.1 Motivation

Im Falle von Poker besteht eine grosse Motivation nach priorisierten Erfahrungen. Der MemoryBuffer besteht zu einem grossen Teil aus Fold-Erfahrungen. Diese sind für das Model-Fitting dann von Priorität, wenn bereits der eingesetzte Stack gross ist und danach gefolded wird. Das führt zu einer grossen, negativen Belohnung. Wiederum Erfahrungen, bei denen im Preflop Zustand gefolded wurde, sind weniger relevant für das eigentliche Model-Fitting. Solche Aktionen sollen im Netzwerk priorität für die Lerneinheiten verwendet werden.

##### 3.6.2.2 Erweiterung MemoryBuffer Komponente

Die in RE3 entwickelte MemoryBuffer.py Komponente bietet nun die Flexibilität, zusätzlich als Prioritized Memory Buffer zu fungieren. Die Komponente wird somit in dieser Realisierungseinheit dahingehend erweitert, wodurch sich über den Klassen-Initialisator angeben lässt, ob PER verwendet werden soll.

```
1. def __init__(self, max_size, number_of_parameters, with_per=True)
```

Wird PER aktiviert, werden neu folgende Änderungen beim Speichern sowie Abrufen von Erfahrungen vorgenommen:

## 1. Speichern

Neue Erfahrungen werden standardmässig mit maximaler Priorität gespeichert. Dadurch werden neue Erfahrungen, unabhängig der Wichtigkeit, mit hoher Wahrscheinlichkeit für das Fitting verwendet.

```
2. if max(self.priority_memory) == 0:
    self.priority_memory[index] = 1 # initialize with max priority so that
    it is being selected

else:
    self.priority_memory[index] = max(self.priority_memory)
```

[MemoryBuffer.py, *store\_state(...)*]

Die Priorität wird bei jedem Abruf des Agenten neu gesetzt. Handelt es sich um eine Erfahrung mit niedrigem Temporal-Difference Error, sinkt die Priorität stark und die Erfahrung wird in Zukunft mit kleinerer Wahrscheinlichkeit gezogen.

## 2. Abrufen

Wird nun über die *replay(...)* Methode ein *get\_sample\_batch(batch\_size, priority\_scale=1.0)* vom MemoryBuffer angefordert, werden gemäss Paper nachfolgende Schritte durchgeführt.

- Berechne die Wahrscheinlichkeit proportional zu den vorhandenen Prioritäten, die für jede Erfahrung gesetzt sind. Skaliere die Priorität basierend auf allen vorhandenen Erfahrungen mit Hyperparameter Beta (*priority\_scale*).

```
1. def get_probabilities(self, priority_scale):
2.     scaled_priorities = np.array(self.priority_memory[0:max_memory])
3.             ** priority_scale
4.     sample_probabilities = scaled_priorities / sum(scaled_priorities)
5.     probabilities = self.get_probabilities(priority_scale)
```

**Listing 9** Berechnung der Wahrscheinlichkeit für jede Erfahrung

- Generiere ein zufälliges Set von Index-Werten mit Berücksichtigung der Wahrscheinlichkeit, dass die Erfahrung gezogen wird.

```
5. np.random.choice(max_memory, batch_size, p=probabilities)
```

**Listing 10** Zufällige Auswahl Erfahrungen (Index) unter Berücksichtigung Wahrscheinlichkeit

- Berechne die Gewichtigkeit dieser Erfahrungen (*importances*). Die Gewichtigkeit ist das Kriterium, anhand dessen das Netzwerk weiss, wie viel es von einer bestimmten Erfahrung lernen kann.

```
6. importance = 1 / batch_size * 1 / probabilities
    importance_normalized = importance / max(importance)
    return importance_normalized
```

**Listing 11** Berechnung der Gewichtigkeit der gezogenen Erfahrungen

Die gezogenen Erfahrungen werden der Replay Methode mit der Gewichtung zurückgegeben.

### 3. Lernen

Desto mehr das Netzwerk die Exploration durchgeführt hat, desto mehr soll es basierend den wichtigen Erfahrungen trainiert werden. Entsprechend werden die Gewichtigkeiten mit Epsilon skaliert. Das Netzwerk erhält nun die entsprechenden Erfahrungen und nutzt diese für das Model-Fitting.

```
7. sample_weight_importances = importances ** (1 - self.epsilon)
8. self.dqn_eval.fit(previous_states, q_target, verbose=0, epochs=1,
                     sample_weight=sample_weight_importances)
```

Listing 12 Model-Fitting unter Berücksichtigung sample\_weight

### 4. Prioritäten aktualisieren

Das neuronale Netzwerk hat das Fitting und damit den nächsten Gradient Schritt vorgenommen. Die Prioritäten werden neu berechnet. Dafür wird der Temporal-Difference Error Algorithmus angewendet [24, p. 5]:

$$\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg \max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1}) \quad (3)$$

Ist die Differenz zwischen Vorhersage und tatsächlicher Beobachtung nun kleiner, ist auch die entsprechende Erfahrung nicht mehr von hoher Bedeutung. Beispiel: Konvergiert das Netzwerk in eine Richtung, in der die Fold/Raise Balance nicht dem optimalen Poker-Spiel entspricht, werden die hohen, negativen Belohnungen priorisiert, was wieder zu mehr Folds führt. Das Netzwerk erreicht dadurch eine schnellere Reaktionszeit. Die aktualisierten Prioritäten werden an den MemoryBuffer zurückgegeben.

```
9. self.memory.set_priorities(batch_indices, td_errors)
```

Listing 13 Rückgabe der aktualisierten Prioritäten an MemoryBuffer

Die Berechnungen entsprechen dem Algorithmus gemäss Prioritized Experience Replay von Google DeepMind. [24, p. 5]

### 3.6.3 Ablaufdiagramm

Das ergänzte Ablaufdiagramm mit PER sieht wie folgt aus:

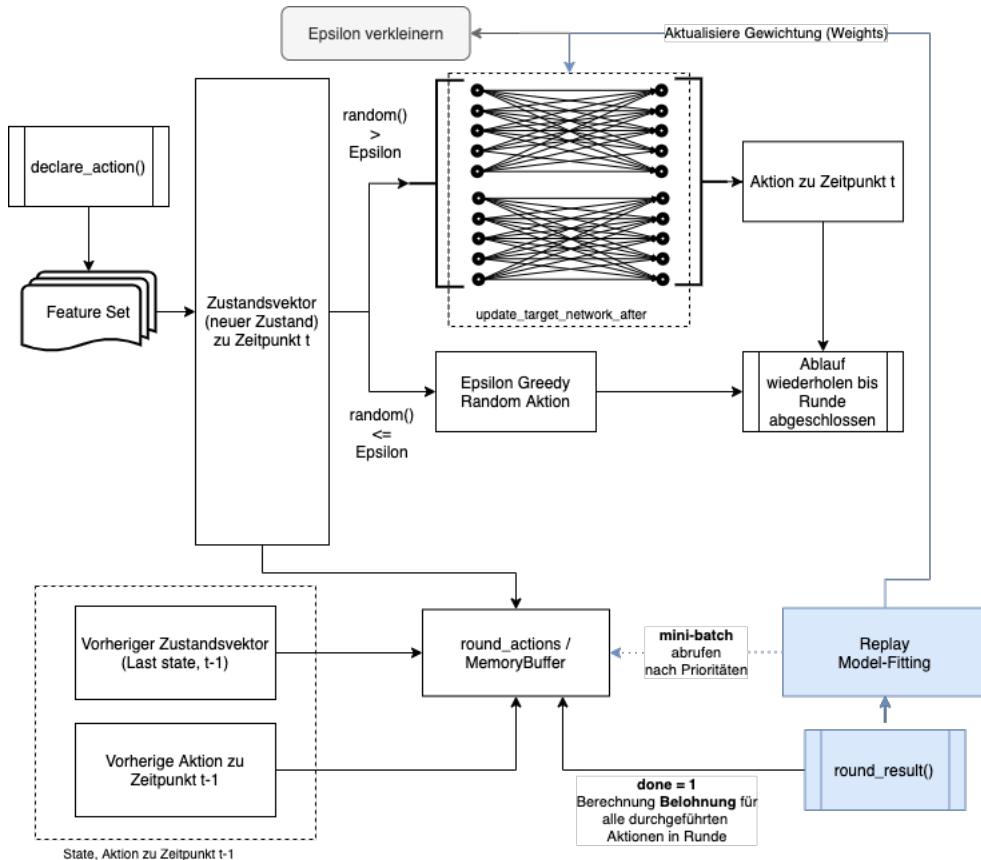


Abbildung 14 Ablaufdiagramm [MVDdqnPer]

Ddqn hat gegenüber Dqn lediglich ein weiteres Netzwerk eingeführt, welches mit Hilfe des `update_target_network_after` Hyperparameter gesteuert wird. Zusätzlich wird in Abbildung 14 Ablaufdiagramm [MVDdqnPer] nach `round_result()` die Anpassung des Netzwerks mit den Prioritäten sowie Anpassung des Netzwerks basierend der Gewichtung ergänzt. Das Diagramm zeigt damit den groben Ablauf einer Pokerrunde. Aktionen, die der Agent während den Runden durchführt, um beispielsweise die Statistiken zu aktualisieren, sind hier nicht dargestellt.

Der gesamte Ablauf sei nun zum besseren Verständnis nachfolgend gegeben.

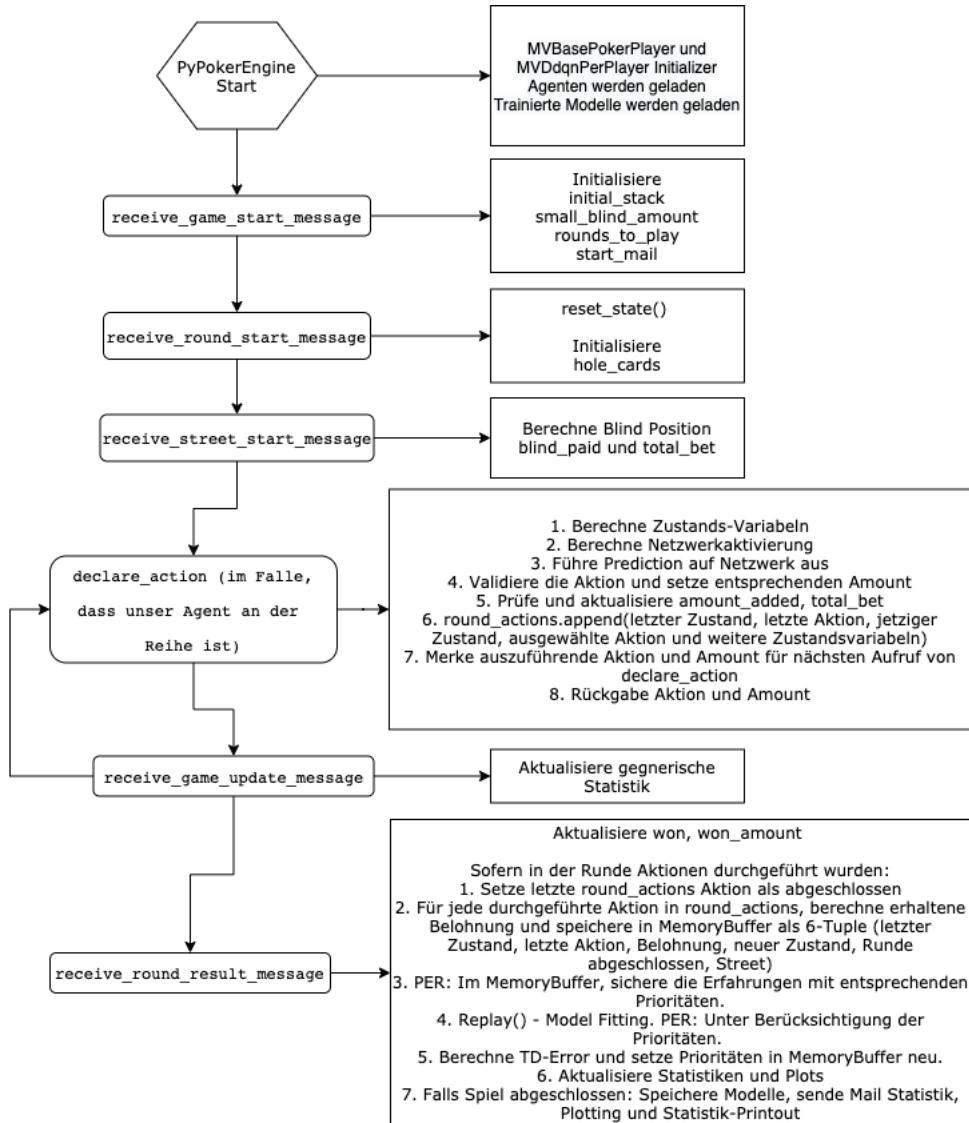


Abbildung 15 Ablauf Pokerrunde [MVBasePokerPlayer]

Abbildung 15 zeigt den Ablauf einer Pokerrunde und die wichtigsten Aktionen, die vom Agenten durchgeführt werden, grob auf.

### 3.6.4 Erkenntnisse

Der Prioritized Experience Replay Agent erreicht die besten Resultate. Nachfolgende Abbildung zeigt die Cashgame Stack Entwicklung von jeweils drei Agenten mit und drei ohne PER.

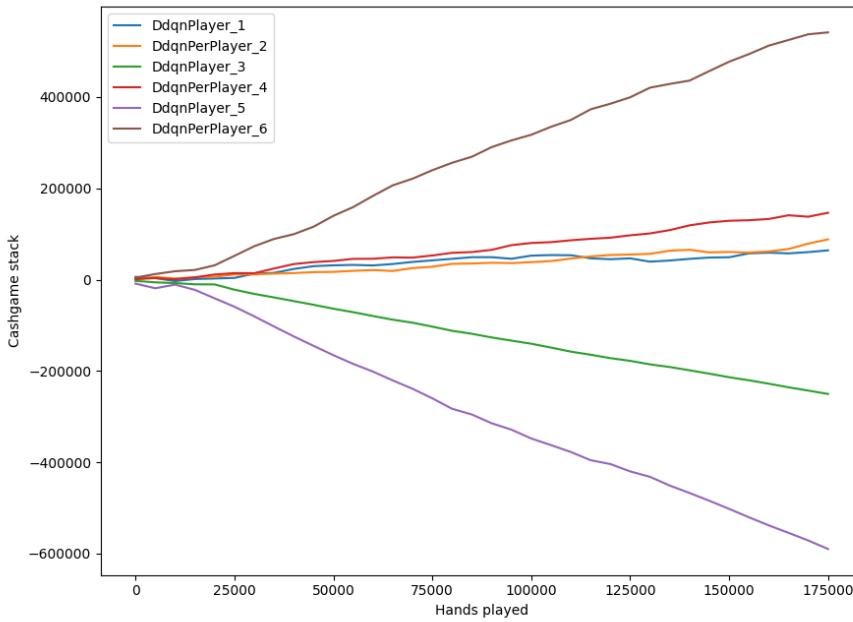


Abbildung 16 Entwicklung Cashgame Stack Vergleich Ddqn versus DdqnPer

Wie auf oberer Abbildung ersichtlich, sind die ersten drei Plätze von PER Agenten belegt. Zwei der Agenten ohne PER schaffen es nicht, den negativen Trend wieder zu korrigieren. Dies wird allen voran auf die hohe Anzahl an Erfahrungen in Kombination mit kleiner Anzahl an wirklich wichtigen Erfahrungen zurückgeführt.

### 3.6.5 Beschlüsse

1. Das Modell Ddqn in Kombination mit Prioritized Experience Replay soll beibehalten werden.
2. Die Einstellungen des Modells sollen optimiert werden. Unzählige Hyperparameter sind bisher mit Standard-Einstellungen initialisiert.
3. Die Features sollen systematisch getestet werden.
4. Ein Feature zur Abbildung der Outs soll geprüft werden, um die Gewinnrate weiter zu optimieren.
5. Das Trainingssetup soll verbessert werden. Dabei sollen Modelle über mehrere, unabhängige Trainingseinheiten mit unterschiedlichen Gegner-Konfigurationen verwendet werden.

## 3.7 RE 5 – Modelleinstellungen und Testing

Die fünfte und letzte Realisierungseinheit dauert vom 22. Juni 2020 bis und mit 19. Juli 2020. Im Fokus liegen Hyperparameter, Netzwerk-Grösse, objektive Funktion sowie AICrowd Resultate. Diese Realisierungseinheit soll zudem die entwickelten Modelle miteinander vergleichen.

### 3.7.1 Ziele

1. Die über die Realisierungseinheiten entwickelten Modelle wurden grösstenteils mit Standard-Hyperparameter Werten initialisiert. Hyperparameter, wie beispielsweise die Anzahl Layer, Epsilon Greedy decay, mini-batch Grösse sollen nun systematisch verbessert werden.
2. Das Potenzial der Features sowie des Aktionsraums soll bestmöglich ausgereizt werden.
3. Eine korrekte Double Deep Q Network Implementation soll sichergestellt sein und gegen die von Keras zur Verfügung gestellten Ddqn Implementation als Referenz antreten.
4. Verbesserte AICrowd Resultate rücken damit ebenfalls mehr in den Fokus.

#### 3.7.1.1 Schwierigkeiten

Modell Anpassungen erfordern allgemein viel Zeit. Umso mehr Hyperparameter Kombinationen miteinander verglichen werden sollen, umso länger dauert die Testphase. Zudem müssen unveränderte Agenten zur Evaluierung gegen angepasste Agenten antreten, um Verbesserungen feststellen zu können.

Es existiert keine Formel, die die optimalen Parameter-Einstellungen liefern kann. Aufgrund der hohen Anzahl an Kombinationsmöglichkeiten soll nun ein systematisches Vorgehen definiert werden, um die Zeit so zu nutzen, dass möglichst gute Ergebnisse erzielt werden können.

#### Hyperparameter Skalierung

Für die Hyperparameter sollen passende Bereiche festgelegt werden. Damit soll beispielsweise nicht der gesamte Bereich zwischen 0.1 und 1.0 getestet werden, stattdessen wird dieser sinnvoll eingeschränkt, was wiederum die Tests beschleunigt. Wir möchten nun *sinnvoll* in diesem Kontext definieren:

1. Für viele Hyperparameter existieren bewährte Praktiken, die von der Machine Learning Community über die Jahre als sinnvoll erachtet werden. Beispielsweise ist bei Prioritized Experience Replay ein *priority-scale* Wert von 0.7 empfohlen. [24, p. 6]
2. Basierend dem Input Vektor des Netzwerks kann die Anzahl Neuronen sowie Anzahl Hidden Layer eingeschränkt werden. [25]

### 3.7.2 Hyperparameter

Im Laufe der Entwicklung mussten an unzähligen Orten Hyperparameter definiert werden. Nachfolgend sei eine Auflistung gegeben.

[components/D\*Agent.py]

| Hyperparameter               | Wert   | Auswirkung                                                                                                                                  | Betrifft   |
|------------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------|------------|
| $\alpha$ - Alpha             | 0.01   | Lernfaktor alpha, regelt die Geschwindigkeit, mit der das Modell mit jeder Iteration die objektive Funktion (objective Function) optimiert. | Alle       |
| $\gamma$ - Gamma             | 0.9    | Gewichtung zukünftige Belohnung                                                                                                             | Alle       |
| memory_size                  | 10'000 | Rückwirkend Erfahrungen die berücksichtigt werden sollen                                                                                    | Alle       |
| batch_size                   | 32     | Performance – Wert gewählt basierend best practices                                                                                         | Alle       |
| replace_target_network_after | 10'000 | Anzahl Aktionen, bis das Target Netzwerk die Gewichte des Eval Netzwerks erhalten soll                                                      | DDQN / PER |
| epsilon                      | 1.0    | Wahrscheinlichkeit, dass zufällige Aktion durchgeführt wird                                                                                 | Alle       |
| epsilon_decay                | 0.9    | Multiplikator, um Epsilon Wahrscheinlichkeit nach jeder Entscheidung abzuschwächen                                                          | Alle       |
| epsilon_min                  | 0.01   | Minimale Epsilon-Exploration Rate, die beibehalten werden soll                                                                              | Alle       |
| priority_offset              | 0.1    | Keine Priority 0 in Memory Buffer, daher offset                                                                                             | PER        |
| priority_scale               | 0.7    | Skalierungsparameter für die Priorität                                                                                                      | PER        |
| hl*_dims                     | -      | Anzahl Neuronen, die die Methode <code>_build_model</code> für die Erzeugung des Netzwerks verwenden soll                                   | Alle       |

### 3.7.2.1 Hyperparameter Justierung

Nachfolgende Justierung wird mit Hilfe lokaler Ausführungen, Tensorboard-Analysen und unter Berücksichtigung der Evaluierungs-Methoden (Statistiken) durchgeführt.

| Hyperparameter               | Alter Wert | Neuer Wert |
|------------------------------|------------|------------|
| $\alpha$ - Lernrate Alpha    | 0.01       | 0.05       |
| $\gamma$ - Gamma             | 0.9        | 1          |
| Grösse Output Layer          | 3          | 4          |
| epsilon_decay                | 0.9        | 0.9999     |
| epsilon_min                  | 0.01       | 0.005      |
| batch_size                   | 32         | 128        |
| replace_target_network_after | 10000      | 5000       |

Abbildung 17 Hyperparameter Anpassung der Einstellungen

1. Die Einstellungen  $\alpha$ , `batch_size` sowie `replace_target_network_after` sind der verminderten Evaluationsgrösse auf AICrowd geschuldet. In den Rohdaten der AICrowd Resultate ist ersichtlich, dass ab dem 10.07.2020 statt 100'000 lediglich 50'000 Runden gespielt werden. Um ein schnelleres Adaptieren zu ermöglichen, werden die Hyperparameter dementsprechend angepasst.
2. Epsilon\_min wird auf 0.5% minimale Exploration festgelegt. Auf AICrowd werden vortrainierte Modelle hochgeladen, die möglichst keine zufälligen Aktionen durchführen sollen. Bei lediglich 50'000 Iterationen führt dies andernfalls schnell zu hohen Verlusten.
3. Epsilon\_decay wird stark vermindert, um längere Exploration durch den grossen Zustandsraum sicherzustellen. Ein Wert von 0.9999 bedeutet damit über 53'000 Runden, um von 100% auf 0.5% Exploration zu fallen.
4. Im Output Layer wird neu zwischen small call und big call unterschieden. Der Agent erhält damit die Möglichkeit, nur bis zu einem bestimmten Betrag mitzugehen (festgelegt auf 50% des Stacks). Zusätzlich wird für Raise der zu erhöhenden Betrag basierend der Hand-Stärke festgelegt. Die Validierung findet in [MVBasePokerPlayer.py, `_validate_action(...)`] statt.

### 3.7.3 Feature Importance

Neuronale Netze haben typischerweise die Reputation, ein Black Box Algorithmus zu sein. Es ist nicht bekannt, was in den inneren Layer (Hidden Layer) des Netzwerks passiert und welche Input Parameter zu welchen Veränderungen führen. Die Gewichtung der einzelnen Graphen kann zwar exportiert werden, diese stehen jedoch in keinem Kontext.

Die Features sollen analysiert werden, um deren Auswirkungen auf das Netzwerk zu erkennen. Hier kommen Feature Permutationen ins Spiel. Das Prinzip ist einfach: Zwei Features werden im Inputvektor auf einem bereits trainierten Netzwerk vertauscht. Danach wird ein neuer Trainingslauf gestartet, dabei können zwei Szenarien eintreffen:

1. Die Werte der objektiven Funktion steigen stark an. Die gespeicherten Gewichte im Modell entsprechen nicht mehr dem bereits bekannten Mustern, wodurch das Netzwerk keine guten Entscheidungen treffen kann. Eine grosse Fehlerrate weist auf hohe Wichtigkeit der vertauschten Features hin.
2. Die Optimierungswerte bleiben unverändert. Dies ist ein Indiz dafür, dass die im Modell gespeicherten Gewichte der vertauschten Features keinen signifikanten Einfluss auf die Optimierung aufweisen. [26]

### 3.7.3.1 Vorgehen Testing

Als Ausgangslage für die Tests werden die eigenen Player-Klassen verwendet. Um die Ergebnisse nicht zu stark mit dem Erkunden des Netzwerkes zu verzerrn, wurde der Hyperparameter Epsilon für den Benchmark und die Feature Permutationen von 1.0 auf 0.01 gesetzt.

Sämtliche durch die Player-Klassen verwendeten Modelle wurden während zwei Millionen Runden trainiert. Um die trainierten Modelle mit den folgenden Permutationen vergleichen zu können, wurden zuerst 20'000 Runden mit der originalen Feature Reihenfolge gespielt. Die abgeflachte Konvergenz der Optimierungs-Funktion dieser Aufzeichnung dient als Benchmark und ist in der folgenden Abbildung ersichtlich.

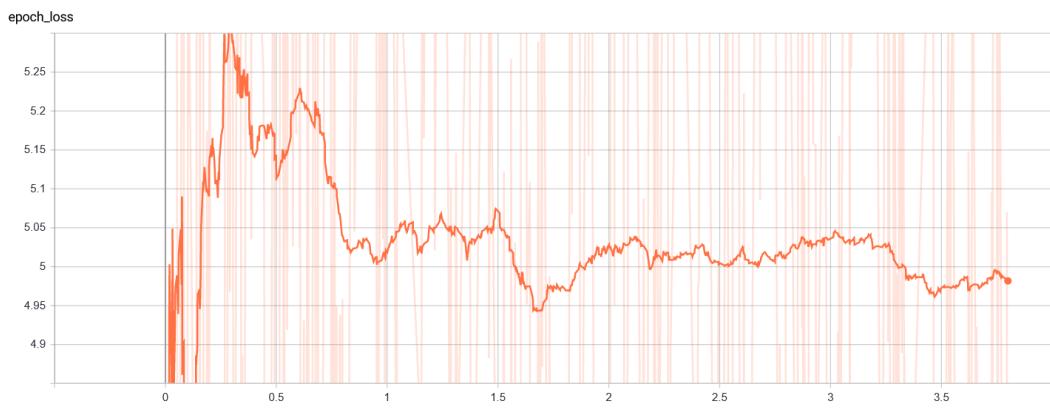


Abbildung 18 Benchmark Feature Importance Tests

Die Angaben auf der x-Achse entsprechen der relativen Zeit seit Beginn der Auswertung in Stunden. Der Plot zeigt, wie sich die Loss-Funktion in den ersten 0.8 Stunden dem Wert fünf annähert und danach nicht mehr gross divergiert. Um den Einfluss der Feature Permutationen nachvollziehen zu können, werden für jede Permutation 20'000 Runden gespielt.

### 3.7.3.2 Resultate Testing

#### Anzahl Gegner und Street

Für den ersten Test wurden die Positionen der Features Anzahl Gegner und Street vertauscht. Die daraus resultierende Loss-Funktion ist im folgenden Plot dargestellt.

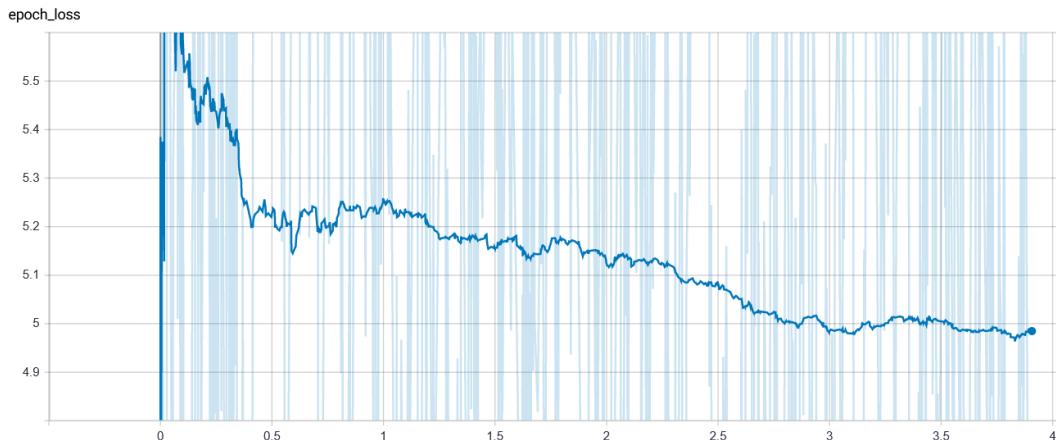


Abbildung 19 Loss Funktion nach Feature Swap Anzahl Gegner und Street

Man beachte bei diesem Plot die unterschiedliche Skalierung der y-Achse im Vergleich zum Benchmark. Aus der Aufzeichnung geht hervor, dass die Werte aus der Optimierungsfunktion nach dem Vertauschen der beiden Features zuerst einen höheren Verlust aufweisen und danach deutlich mehr Zeit benötigt wird, bis sie zum Wert fünf konvergiert. Im Gegensatz zu den ursprünglichen 0.8 Stunden vergingen hier ungefähr 2.7 Stunden. Es mussten deutlich mehr Hände gespielt werden, um die Gewichte der betroffenen Features wieder korrekt zu setzen. Daraus lässt sich gemäss dem beschriebenen Vorgehen folgern, dass die beiden Features für das Netzwerk wichtig sind.

#### Bet Size und Pot Size

Das Vertauschen der beiden Features Bet Size und Pot Size lieferte ein anderes Ergebnis. Hier blieb der Plot der Loss-Funktion beinahe unverändert, was dafürspricht, dass diese beiden Features keinen signifikanten Einfluss die Modell-Optimierung haben.

### 3.7.3.3 Weitere Analyse-Methoden

Nebst dem oben beschriebenen Feature-Swap wurden die eingesetzten Features auch anhand der Mail Auswertungen überprüft. Die folgende Abbildung zeigt eine interessante Tatsache über das Verhalten der Gegner.

Opponent statistics aggressivity:

```
{'lztmrnenixdpmudrllostbne': {'number_of_raises': 15652, 'number_of_actions': 53157, 'aggressivity': 0.29444852042064074},  
'fvsknosvusyymxnltafcup': {'number_of_raises': 13988, 'number_of_actions': 60770, 'aggressivity': 0.23017936481816687},  
'mlgmcpgiigugvhgztqjsw': {'number_of_raises': 19442, 'number_of_actions': 78976, 'aggressivity': 0.24617605348460292},  
'wzqdjuyjehcqjmuitsajd': {'number_of_raises': 16000, 'number_of_actions': 65256, 'aggressivity': 0.245188181929631},  
'xalfbxnbcpulitaaknveuz': {'number_of_raises': 14587, 'number_of_actions': 63419, 'aggressivity': 0.23000993393147165}}
```

Abbildung 20 Auszug aus AICrowd Statistiken mit beinahe identischer Aggressivität aller Gegner

Sämtliche Gegner scheinen ähnlich aggressiv vorzugehen. Gemäss der Kategorisierung in Kapitel 3.8.1.15 würde somit, unabhängig von den teilnehmenden Spielern, jedes Mal der gleiche kategorische Wert ausgewählt.

Dies hat allgemein darauf hingewiesen, dass das Feature für das Netzwerk zu erschwertem Training führt. Um die Gewichte im Netzwerk korrekt trainieren zu können, müssen Agenten mit unterschiedlichem Verhaltensmuster im Training antreten. Das Feature wird aufgrund zusätzlich erschwerendem Trainings-Verfahren entfernt.

### 3.7.4 Loss-Funktion

Die Loss-Funktion gilt als Mass für unser Machine Learning Modell, um festzustellen, wie genau der Agent in der Lage ist, ein erwartetes Ergebnis vorherzusagen. Ziel des Modells ist dabei, möglichst diese Loss Funktion, also das Delta zwischen vorhergesagtem und tatsächlich beobachtetem Zustand, zu minimieren.

In unserem Fall werden die Q-Values berechnet, die allgemein der diskontierten Belohnung entsprechen. Umsso grösser der Loss-Wert, umso schlechter hat das Modell gespielt.

Der Entscheid für eine korrekte Loss-Funktion hat einen grossen Einfluss auf die allgemeine Performance des Netzwerks. Aufgrund dessen wird dieser Entscheidungsprozess hier dokumentiert. Zu Beginn der Optimierung verwendet der Agent die Mean Squared Error (MSE) Loss-Funktion.

#### Erläuterung

Das Netzwerk erhält für die Vorhersage allgemein den Zustand basierend auf den von uns definierten Features. Es wird aber der partiell beobachtbare Zustand aus Sicht des Agenten wiederspiegelt. Informationen, wie beispielsweise die Hände der Gegner, stehen wie bereits erläutert nicht zur Verfügung.

#### Anwendung am Beispiel

Während in einer Runde ein Ass-Paar für den Sieg genügt, kann in der nächsten Runde das Ass-Paar zu einem grossen Verlust führen. Das Netzwerk soll also nicht immer bei Ass-Paar gleich entscheiden. Ein grosser Verlust bei einem Ass-Paar (hohem *hand\_ranking*) soll die Hand wiederum auch nicht zu stark bestrafen.

#### Loss-Funktion

Drei Funktionen stehen im Fokus:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Huber Loss [27]

Die Huber Loss-Funktion kombiniert dabei die beiden Funktionen MSE und MAE, um einen Mittelweg zu schaffen. Nachfolgend sei ein Vergleich der drei Funktionen gegeben:

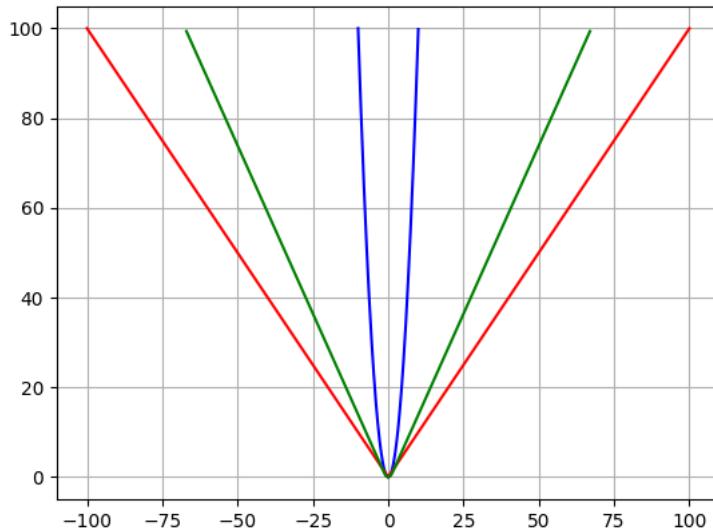


Abbildung 21 Loss Funktionen: MAE (rot), Huber (grün), MSE (blau) [27]

Während MSE (blaue Linie) als quadratische Funktion schnell skaliert (also eine falsche Vorhersage wie beispielsweise bei einem Ass-Paar quadratisch gewichtet), ist MAE (rote Linie) linear in der Skalierung. Huber hingegen verwendet für kleine Delta-Beträge (bis beispielsweise 1.0 Loss-Werte) die MSE Funktion, bei grösseren, fehlerhaften Vorhersagen hingegen die lineare MAE Funktion. Bei kleineren Werten nahe am Zentrum ist damit eine starke Gewichtung (quadratisch) gegeben, ohne bei grösseren Differenzen zwischen Vorhersage und Beobachtung die Balance des Netzwerks zu stören (was beispielsweise zu einem Netzwerk führen könnte, welches lediglich *folded*).

Umso grösser das Delta zwischen Vorhersage und Beobachtung, umso weniger Gewichtung erhält das im Modell durchzuführende Update der Gewichte. Stattdessen wird mit genügend Iterationen gearbeitet, um das Netzwerk zur Konvergenz zu verhelfen. Bei schlechten Händen lernt das Modell also langsamer, diese nicht zu spielen, dieser Faktor wird jedoch durch vorangehende Trainings abgeschwächt. Zeitgleich gewinnt der Agent damit an Stabilität.

Huber entspricht der für diese Problemdomäne angemessenen Funktion.

### 3.7.5 Optimizer

Der Optimizer definiert die Update Regeln des Netzwerks und bestimmt somit die eigentlichen Lernschritte. Der Optimizer hat grossen Einfluss auf die Performance und Geschwindigkeit der Updates, die zu einer Konvergenz führen sollen.

Auch hier wird der korrekte Optimizer für verbesserte Modell-Performance eruiert.

#### 3.7.5.1 Typen

Optimizer werden ähnlich wie neuronale Netzwerke selbst häufig als Black-Box Algorithmen angesehen. Während der Entwicklung wurde Stochastic Gradient Descent als standardmässiger Algorithmus angewendet. Im Projektrahmen soll ein Optimizer eingesetzt werden, welcher dem Stand der Technik entspricht.

Im Fokus stehen vier Optimizer

1. Stochastic Gradient Descent (SGD) gemäss unserer aktuellen Anwendung
2. SGD mit Momentum
3. Adaptive Moment Estimation (Adam)
4. RMSProp

Aktuelle Algorithmen basieren meist auf RMSProp oder Adam. Je nach Anwendungsbereich existieren heute zudem weitere Spezialisierungen, die eine höhere Genauigkeit (Accuracy) erreichen. Dabei handelt es sich meist um spezielle Anwendungsfälle wie Convolutional Neural Networks, bei denen der Fokus auf das Prozessieren von Bildinformationen liegt. Als Beispiel Spezialisierungen seien PowerSign und AddSign genannt, die beide weniger Ressourcen als sonstige Optimizer benötigen und noch schneller konvergieren. [28]

Für die mathematische Erläuterung der einzelnen Optimizer sei auf das Paper *An overview of gradient descent optimization algorithms* [29] verwiesen.

### 3.7.5.2 Auswahlverfahren

Ein sauberes Auswahlverfahren für den korrekten Optimizer mit der korrekten Einstellung würde ähnlich dem Hyperparameter-Entscheidungsproblem zu einem grossen Testverfahren führen. Ziel soll in diesem Rahmen aber sein, mit möglichst geringem Testaufwand einen Optimizer basierend den Charakteristiken zu finden, welcher möglichst schnell zu hoher Genauigkeit konvergiert.

#### **Optimizer für Anwendungsdomäne**

Hier konnte kein richtig oder falsch erkannt werden. Alle Optimizer haben jeweils dem Modell zur Konvergenz verholfen. Das Modell wird auf Adam als häufig verwendeteter Optimizer umgestellt.

### 3.7.6 Normalization

Normalization wird für die Stabilisierung der Cost-Funktion verwendet. Nebst der Normalisierung der Input Werte kann auch innerhalb des Netzwerks zwischen den Hidden Layers mit Normalisierung gearbeitet werden.

Die als Batch Normalization bekannte Anwendung funktioniert aus demselben Grund wie auch die allgemeine Feature Normalisierung – um eine ausgeglichene Verteilung der Werte zu erhalten. Mittelwert und Varianz der berechneten Komponenten sollen vor jedem Layer normalisiert werden, was allgemein die einzelnen Layer stabilisiert. Das wiederum ermöglicht jedem Layer im Netzwerk unabhängig der vorgehenden Aktivierung zu lernen. Weitere Informationen dazu können aus dem Coursera Kurs *Improving Deep Neural Networks: Hyperparameter tuning* von Andrew Ng entnommen werden. [30]

### 3.7.6.1 Exkurs Batch Normalization und Neural Networks

Das neuronale Netzwerk besteht aus einer Vielzahl Neuronen, die vom Keras Framework generiert werden. Einzig bei unserer `build_model()` möchten wir angeben, welcher Layer wie viele Neuronen beinhalten soll.

Jedes dieser Neuronen führt zwei Berechnungen durch. Für unseren ersten Neuron im ersten Layer bedeutet dies:

$$z_1^{[1]} = \text{Verarbeitung der erhaltenen Features und Gewichtung } (W^1, b^1)$$

$$a_1^{[1]} = \text{Wende nun die Aktivierungsfunktion (beispielsweise ReLU) an}$$

Batch Normalization wird nun allgemein vor der Verarbeitung des Features und der Gewichtung auf jedem Neuron angewendet. [31]

### 3.7.6.2 Anwendung

Batch Normalization wird in `build_model` ergänzt. Keras bietet hier einen Layer im `[keras.layers]` Package an. Batch Normalization entspricht lediglich einem weiteren Layer, welcher gemäss vorangehender Erläuterung jeweils vor der Aktivierungsfunktion angewendet werden soll. Unsere `build_model` Methode sieht ergänzt wie folgt aus:

```

1. # Input dimension and first Hidden Layer
2. model.add(Dense(hl1_dims, input_dim=input_layer_size))
3. model.add(BatchNormalization())
4. model.add(Activation('relu'))
5.
6. # For each Hidden Layer:
7. model.add(Dense(hl3_dims))
8. model.add(BatchNormalization())
9. model.add(Activation('relu'))
10.
11. # Output Layer
12. model.add(Dense(output_layer_size))
13. model.add(Activation('linear'))
```

**Listing 14** Auszug `build_model()` mit Batch Normalization aus Agentenklasse [source/components/D\*Agent.py]

**Hinweis:** Gemäss Listing 14 ist auf Zeile 12 ersichtlich, dass im Output Layer keine Batch Normalization ergänzt wird. Grund dafür ist, dass das Normalisieren des Output Layers zu Verfälschungen führen kann und daher nicht empfohlen wird. Die berechneten Q-Values sollen nicht normalisiert werden.

### 3.7.7 Resultate RE5

In den vorangehenden Kapiteln haben unterschiedliche Hyperparameter-Einstellungen, Optimizer, Normalizer sowie Feature Anpassungen, entweder aus Sicht von mathematischen Vorzügen oder aus Test-Gründen, zur Einstellung des Modells geführt.

Diese Einstellungen werden jeweils lokal trainiert, getestet und danach untrainiert auf AICrowd hochgeladen. Die vom Agenten aus AICrowd versendeten Statistiken via E-Mail helfen danach festzustellen, ob die Modelleinstellung zu Verbesserungen führen. Die am besten funktionierende Modelleinstellung wird danach weiterverwendet, über ein bis zwei Millionen Runden trainiert und auf AICrowd hochgeladen.

Nachfolgend sei die Statistik von AICrowd vor Anpassungen gemäss Kapitel RE 5 – Modelleinstellungen gegeben:

**Tabelle 3 Evaluation 10.07.2020 aus [Beilagen/AICrowd\_Resultate]**

| Evaluation                                       | Konfiguration                                           |
|--------------------------------------------------|---------------------------------------------------------|
|                                                  | <b>Reward:</b> Simple                                   |
| Weitere Details siehe<br>[100720_Evaluation.pdf] | <b>Split Netzwerk:</b> Nein                             |
|                                                  | <b>Modell vortrainiert:</b> Ja, lediglich 20'000 Runden |
|                                                  | <b>Blinds spielen:</b> Nein                             |
|                                                  | <b>high_bet_fold:</b> 312                               |
|                                                  | <b>raises:</b> 76'535                                   |
|                                                  | <b>folded:</b> 49.146%                                  |
|                                                  | <b>win_rate:</b> 55.07%                                 |
|                                                  | <b>cashgame stack:</b> -1'258'481                       |

Die Statistik zeigt:

1. 55% der gespielten Runden gewonnen
2. 49%, insgesamt 24'574 von 50'000 Runden ausgestiegen (folded).
3. 312 Runden über 50 Chips eingesetzt und danach trotzdem ausgestiegen (*high bet fold*)
4. Hohe *raise* Anzahl im Vergleich zu *call*: Aggressive Spielweise
5. Insgesamt 1.26 Millionen Chips an Gegner verloren

In allen Sektoren besteht Verbesserungspotenzial:

1. Die Zahl der gespielt und gewonnen Runden muss möglichst hoch sein, um wenig Chips an Gegner zu verlieren.
2. Bei sechs Spielern und einem gleichverteilten Poker-Spiel (faire Random-Selektion der Karten und damit Hand-Stärke) sollte die *Fold*-Rate eines Tight-Spielers deutlich über 50% liegen.

Nachfolgend sei die Statistik von AICrowd nach Anpassungen gemäss Kapitel RE 5 – Modelleinstellungen gegeben:

Tabelle 4 Evaluation 12.07.2020 aus [Beilagen/AICrowd\_Results]

| Evaluation                                       | Konfiguration                                                                                                                                                                                                                                                                             |
|--------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Weitere Details siehe<br>[120720_Evaluation.pdf] | <b>Reward:</b> Complex<br><b>Split Netzwerk:</b> Nein<br><b>Modell vtrainiert:</b> Ja, lediglich 20'000 Runden<br><b>Blinds spielen:</b> Nein<br><b>high_bet_fold:</b> 30<br><b>raises:</b> 28'752<br><b>folded:</b> 75.95%<br><b>win_rate:</b> 48.12%<br><b>cashgame stack:</b> -430'783 |

Die Statistik zeigt:

1. 48% der gespielten Runden gewonnen. Die Zahl ist etwas niedriger im Vergleich zu 100720\_Evaluation – dies liegt aber an der fold-Zahl. Der vorherige Agent hat untrainiert aggressiver gespielt und damit mehr gegnerische Blinds gewonnen.
2. Das Netzwerk lernt schnell, nicht viel zu setzen und dann zu folden (30 im Vergleich zu 312 Mal *high\_bet\_fold*), falls schlechte Gewinnaussichten bestehen.
3. Die fold-Rate liegt bei 75.95% gegenüber 49% mit vorheriger Einstellung und entspricht damit mehr den Erwartungen.

### Resultate

Mit Erreichen der finalen Parameter Einstellung konnte die AICrowd Performance um das 2.5-fache im Vergleich der vorangehenden Modelle verbessert werden, ohne trainiert zu sein.

Das optimierte Modell wird über eine Million Durchläufe auf einer vom Projektteam aufgesetzten Linux-Maschine trainiert.

Auf der Abbildung unten ist ein Auszug aus AICrowd gegeben. Die Evaluierung vom 19.07 (20200719.0) beinhaltete ein vtrainiertes Modell gegen Tight-Agenten zur Prüfung der Performance und erreichte 58'636.7 Chips und damit einen Podest-Platz. Im Weiteren werden noch andere Zusammensetzungen getestet, bis zur schliesslich finalen Agentenkonfiguration.

| Status | Evaluation Date | Accumulated win |
|--------|-----------------|-----------------|
| graded | 20200719.0      | 58636.7         |
| graded | 20200712.0      | -430783.0       |
| graded | 20200709.0      | -1259173.5      |
| graded | 20200628.0      | -1938040.0      |

Abbildung 22 AICrowd Resultate Hyperparameter Optimierung vom 28.06 - 19.07

## 3.8 Features

In den vorgehenden Kapiteln lag der Fokus auf den Modellen. Dieses Kapitel beschreibt nun die Features, welche auf Basis der Erhebungen aus dem Kapitel 2.7 Texas Holdem Poker Strategien implementiert und über die Iterationen ergänzt wurden. Zudem wird stellenweise auf allfällige Experimente und Entwicklungen der Features eingegangen.

### 3.8.1 Implementierte Features

In diesem Unterkapitel werden die implementierten Features erklärt. Die Auswahl der Implementierungen basiert auf dem Gedanken, dem neuronalen Netzwerk die gleichen Informationen zur Verfügung zu stellen, die ein Mensch ebenfalls brauchen würde, um die gleichen Entscheidungen zu treffen.

#### 3.8.1.1 Gewinnchance Monte-Carlo

Die Gewinnchance basiert auf der in der PyPokerEngine zur Verfügung gestellten Monte-Carlo Simulation. Das Ergebnis entspricht der Anzahl gewonnener Spiele geteilt durch die Anzahl durchgeföhrter Simulationen und nimmt somit einen Wert zwischen null und eins an. Da die Dauer für die Entscheidungsfindung im Algorithmus beschränkt ist, wurde die Anzahl Simulationen auf 100 limitiert.

In der vierten Realisierungseinheit wurde auch eine weitere Monte-Carlo Simulation getestet, die frei verfügbar ist. [33] Diese Version der Einschätzung der eigenen Handstärke lieferte deutlich genauere Angaben als die zur Verfügung gestellte. Dafür lagen die Ausführungszeiten zwischen 0.05 und 15 Sekunden. Da diese Ausführungszeiten die Einschränkung von 0.04 Sekunden deutlich überschreiten und zudem sehr inkonsistent sind, wurde die Alternative wieder verworfen.

Der Grundgedanke hinter diesem Feature ist, dem Algorithmus eine diskrete Information zum Potential der eigenen Hand zur Verfügung zu stellen. Langfristig soll der Algorithmus daraus lernen, vor allem starke Hände zu spielen, um seinen Gewinn zu maximieren.

#### 3.8.1.2 Hand-Ranking kategorisch

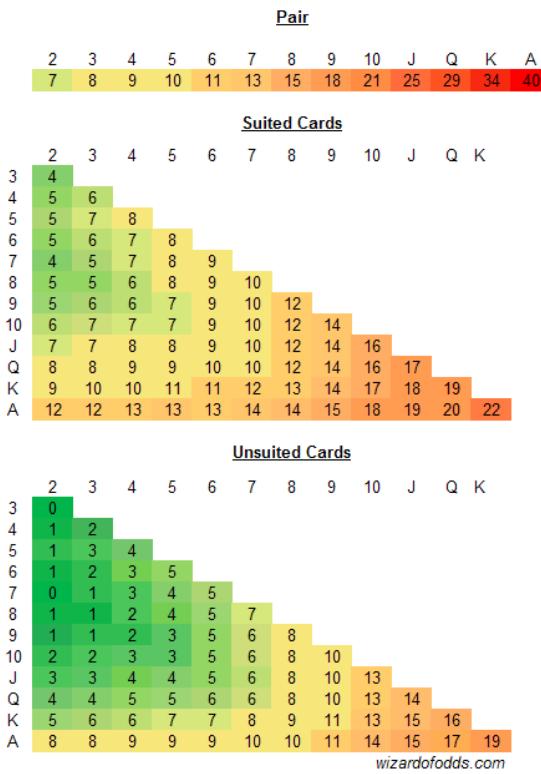
Das Hand Ranking ist eine neue Repräsentation der Stärke der eigenen beiden Handkarten in Kombination mit den Gemeinschaftskarten. Sie dienen als Nachfolger der Monte-Carlo Simulation. Tests der gegebenen Simulation haben ergeben, dass für eine Spielerzahl von sechs Spielern im Preflop durchschnittlich eine Gewinnwahrscheinlichkeit von ca. 80% berechnet wird. Sowohl das manuelle Testen mit bewusst schwachen Handkarten als auch die Ergebnisse beim Durchspielen der Hände widerlegten die Ergebnisse der Monte-Carlo Simulation deutlich. Die tatsächlichen Gewinnchancen waren deutlich tiefer. Aus diesem Grund fiel der Entscheid, diese Implementierung der Monte-Carlo Simulation nicht mehr einzusetzen.

Die neue Implementierung unterscheidet zwei verschiedene Szenarien beim Bestimmen der Stärke der eigenen Hand, im Preflop und den anderen Streets:

- **Preflop**

Im Per flop sind noch am wenigsten Informationen zum weiteren Spielverlauf und der Entwicklung der eigenen Hand vorhanden. Daher wird die Stärke der eigenen Hand basierend auf Statistiken erhoben. [34]

Power Ratings for Initial Two Cards in 6-Player Texas Hold'em



wizardofodds.com

Abbildung 23 Hand Rankings Preflop [34]

Da die Hand Ranking Werte von null bis 40 annehmen, werden sie mit 2.5% multipliziert, um Rankings im Bereich von null bis 100% zu erhalten.

- **Flop, Turn und River**

In den restlichen Streets wird die Stärke der eigenen Hand durch die Gemeinschaftskarten deutlicher. Auch hier wird der Wert wieder mit einem statistischen Ansatz ermittelt. Mit der eingebundenen Komponente Treys kann die Stärke der Hand bestimmt werden. Dabei wird die eigene Handstärke mit allen existierenden Äquivalenzklassen von Pokerhänden verglichen. [35] Nachteil dieses Ansatzes ist, dass nur die aktuelle Stärke berücksichtigt wird. Weitere mögliche Entwicklungen wie beispielsweise Flush- oder Straight-Draws im Flop oder Turn werden nicht in die Stärke der Hand einberechnet.

Nebst den verlässlicheren Gewinnchancen ist auch die Zeit zur Berechnung deutlich kürzer als bei der Monte-Carlo Simulation.

Der Ansatz, das resultierende Ranking an die Anzahl aktiver Spieler zu skalieren wurde verworfen. Die genaue Berechnung kann der Tabelle *win\_rate\_experimente.xlsx* im Anhang entnommen werden.

Das Resultat ist in jedem Fall eine prozentuale Angabe zwischen null und hundert.

Die Abbildung erfolgte zuerst in 10%-Schritten. Da diese Unterteilung nach dem Preflop jeweils über 700 Hände zusammenfasst, wurde sie in der letzten Realisierungseinheit auf 5%-Schritte verfeinert.

### 3.8.1.3 Wahrscheinlichkeit Handstärke kategorisch

Dieses Feature gibt Auskunft über die Wahrscheinlichkeit, basierend auf den eigenen Handkarten und den Gemeinschaftskarten, eine gewisse Handstärke zu erreichen. Weitere Details zur Implementierung befinden sich im Kapitel 7.1.9. Es wird für jede Handstärke (High Card, Pair, Two Pair, etc.) eine separate kategorische Abbildung erstellt. Jede dieser Abbildungen ist in 10%-Schritte unterteilt und beschreibt die Wahrscheinlichkeit, mit dieser Handstärke im Showdown zu enden.

### 3.8.1.4 Street kategorisch

Die Information, in welcher Street sich das Spiel gerade befindet, wird durch die PyPokerEngine zur Verfügung gestellt und kann nur vier Werte annehmen: Preflop, Flop, Turn und River. Da zu jedem Zeitpunkt nur einer dieser Werte zutrifft, wurde das Feature als kategorischer Wert abgebildet.

**Tabelle 5 Street One-hot**

| Preflop | Flop | Turn | River |
|---------|------|------|-------|
| 1       | 0    | 0    | 0     |
| 0       | 1    | 0    | 0     |
| 0       | 0    | 1    | 0     |
| 0       | 0    | 0    | 1     |

Tabelle 5 Street One-hot zeigt die Codierung. Es wird jeweils nur die Zeile verwendet, die der aktuellen Street entspricht.

Dieses Feature wurde gewählt, weil sich das Verhalten eines Spielers je nach Street unterscheiden sollte, wie im Kapitel 2.7.1 Grundsätze eigenes Verhalten dargelegt wurde.

### 3.8.1.5 Position kategorisch

Analog der Street, ist auch die Position des eigenen Spielers als kategorischer Wert abgebildet. Die Information wird nicht durch die PyPokerEngine direkt zur Verfügung gestellt, sondern wird anhand der eigenen *UUID*, Plätze am Tisch und Distanz zum Small Blind berechnet. Eine grafische Darstellung kann dem Kapitel 2.7.1.9 entnommen werden.

**Tabelle 6 Position One-hot**

| Early | Middle | Late |
|-------|--------|------|
| 1     | 0      | 0    |
| 0     | 1      | 0    |
| 0     | 0      | 1    |

Tabelle 6 Position One-hot zeigt die Codierung der Position des eigenen Spielers. Auch hier kann wieder nur eine Zeile zutreffen.

Die Unterteilung in nur drei Bereiche am Pokertisch hat zwei Gründe:

- **Bekanntheit**

Die Unterteilung in Early-, Middle- und Late-Position ist weit verbreitet. Es existieren weitere bekannte Einteilungen, welche beispielsweise die beiden Blind-Positionen speziell berücksichtigen. Dies wird in diesem Fall aufgrund der fixen Anzahl Spieler sowie den kleinen Blinds nicht berücksichtigt.

- **Unabhängigkeit von Anzahl Spielern**

Unsere Implementierung passt sich der Anzahl aktiver Spieler an. Der Vorteil dieser Vorgehensweise wird am besten in einem Gegenbeispiel klar, in dem die Position ebenfalls als One-hot-Encoding modelliert wurde, jedoch mit sechs Einträgen, die den Plätzen am Pokertisch entsprechen:

- a. Der eigene Spieler befindet sich zu Beginn an Position vier, was zu folgendem One-hot-Encoding führt:

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|

- b. Im Preflop beschliessen die Spieler an den Positionen eins bis drei ihre Hände nicht zu spielen und folden. Die Spieler an den Positionen fünf und sechs hingegen warten den Flop ab und callen.
- c. Das One-hot-Encoding ist nun zwar nicht falsch, der eigene Spieler sitzt immer noch auf dem vierten Platz, muss jedoch im Flop als erster agieren, was nicht mehr mit dem One-hot-Encoding übereinstimmt.
- d. Unsere Implementierung hingegen berücksichtigt diesen Umstand und bildet die Position des eigenen Spielers im Preflop als Middle- und im Flop als Early-Position ab.
- e. Die eins im One-hot-Encoding einfach an die erste Stelle zu setzen wäre auch eine Möglichkeit, dann ist jedoch nicht klar, wie viele Spieler im Anschluss noch auf die eigene Aktion reagieren werden.

Diese Implementierung zielt darauf ab, dass der Algorithmus sein Verhalten je nach Position anpassen soll. Die Grundsätze des positionsabhängigen Agierens am Pokertisch sind im Kapitel 2.7.1.9 beschrieben.

### 3.8.1.6 Anzahl Gegner kategorisch

Die Anzahl Gegner wurde aufgrund der Erkenntnisse aus dem Kapitel 2.7.1.2 in den Input Layer aufgenommen. Das Feature wird als One-hot-Encoding dargestellt wie folgender Tabelle entnommen werden kann.

Tabelle 7 Repräsentation Anzahl Gegner als kategorischer Wert

| 2 Gegner | 3 Gegner | 4 Gegner | 5 Gegner |
|----------|----------|----------|----------|
| 1        | 0        | 0        | 0        |
| 0        | 1        | 0        | 0        |
| 0        | 0        | 1        | 0        |
| 0        | 0        | 0        | 1        |

Da die Anzahl der Gegner immer zwischen zwei und sechs liegt, ergeben sich nur vier mögliche Konstellationen.

Die Abbildung als One-hot-Encoding wurde bewusst gewählt, da das Verhalten des Algorithmus je nach Anzahl der Gegner anders ausfallen sollte und somit eine Kategorisierung abgebildet werden kann.

### 3.8.1.7 Pot-Grösse

Die Grösse des Pots wird durch die PyPokerEngine zur Verfügung gestellt. Damit auch dieses Feature in einem Raum von null bis eins liegt, wird folgende Formel angewendet:

$$pot = \frac{aktuelle\_pot\_groesse}{anzahl\_spieler * initialer\_stack}$$

Durch die gegebenen Regeln dieser Projektarbeit kann der Nenner auch als Konstante betrachtet werden, da jeweils sechs Spieler teilnehmen und der Stack pro Runde 200 Chips beträgt. Folglich ist der Nenner immer 1'200.

Dadurch soll der Algorithmus erfahren, wie viele Chips in der aktuellen Situation gewonnen werden können. Diese Information soll insbesondere bei der Gewinnmaximierung helfen.

### 3.8.1.8 Pot-Grösse kategorisch

Die kategorische Abbildung der Pot Grösse erfolgt in 10% Schritten. Somit entspricht jeder Schritt 120 Chips, da die grösstmögliche Pot-Grösse 1'200 Chips beträgt.

### 3.8.1.9 Pot-Odds

Der grundlegende Gedankengang sowie die Berechnung der Pot-Odds können dem Kapitel 2.7.3.3 entnommen werden. Die Absicht dahinter ist, zu erkennen, ob es sich lohnt, einen Einsatz zu bezahlen oder nicht.

### 3.8.1.10 Pot-Odds kategorisch

Die Pot-Odds können auch kategorisch dargestellt werden. Die gewählte Unterteilung ist in folgender Tabelle abgebildet.

**Tabelle 8 Pot-Odds kategorisch**

| Gratis call | 25% von Pot | 50% von Pot | 75% von Pot | Mehr als 75% |
|-------------|-------------|-------------|-------------|--------------|
| 1           | 0           | 0           | 0           | 0            |
| 0           | 1           | 0           | 0           | 0            |
| 0           | 0           | 1           | 0           | 0            |
| 0           | 0           | 0           | 1           | 0            |
| 0           | 0           | 0           | 0           | 1            |

Der gratis Call wird dabei vor der 25% Obergrenze berechnet, um das Erkennen von kostenlosen Calls zu ermöglichen. Die 25%--, 50%- und 75%-Schritte bilden jeweils die Obergrenze der berechneten Pot-Odds, wobei der höchste, noch zutreffende Eintrag der Tabelle verwendet wird.

### 3.8.1.11 Einsatzgrösse

Die Grösse des Einsatzes berechnet sich aus der Summe aller bezahlten Einsätze einer Runde. Die relative Grösse dieser Summe wird dem Algorithmus zur Verfügung gestellt:

$$einsatz = \frac{bereits\_bezahlte\_chips}{initialer\_stack}$$

Auch diese Formel resultiert immer in einer prozentualen Angabe von null bis hundert. Der initiale Stack ist anhand der gegebenen Regeln eine Konstante und beträgt immer 200. Das Feature soll dem Algorithmus bei der Entscheidungsfindung helfen, da beispielsweise das folden bei maximalem Einsatz keinen Sinn ergibt.

### 3.8.1.12 Einsatzgrösse kategorisch

Die Repräsentation des Features wird ebenfalls von einer prozentualen Angabe in kategorische Werte geändert. Die Unterteilung ist wiederum in zehn Kategorien von identischer Grösse gegliedert. Basierend auf den gültigen Regeln ergeben sich daraus Schritte von jeweils 20 Chips.

### 3.8.1.13 Gegner aktiv

Gibt an, ob ein Gegner noch an der aktuellen Runde teilnimmt. Diese Information wird durch die PyPokerEngine im Attribut seats geführt. Ist der State eines Spielers folded, ist er bereits ausgeschieden, andernfalls gilt er als aktiv.

### 3.8.1.14 Aggressivität

Die Aggressivität wird aus den bisherigen Aktionen eines Gegners errechnet. Dazu wird eine prozentuale Angabe verwenden. Die Definition der Aggressivität eines Spielers lautet wie folgt:

$$aggressivität = \frac{anzahl\_raises}{anzahl\_aktionen}$$

Die grundsätzlichen Überlegungen zur Wichtigkeit des gegnerischen Verhaltens können dem Kapitel 2.7.2 entnommen werden. Diese Werte werden über den Zeitraum der gesamten Evaluation, fortlaufend erhoben und aktualisiert.

### 3.8.1.15 Aggressivität kategorisch

Die Aggressivität kann auch kategorisch dargestellt werden. Die gewählte Unterteilung ist in 20%-Schritte gegliedert, um eine eher grobe Abstufung zu erlauben. Beginnend bei 0%-20% für die Kategorie *sehr passiv* und endend bei 80%-100% für *sehr aggressiv*. Nachfolgend sei die Verteilung der Werte repräsentiert.

**Tabelle 9 Aggressivität Kategorien Double Deep Q Network**

| Sehr aggressiv | Aggressiv | Ausgeglichen | Passiv | Sehr passiv |
|----------------|-----------|--------------|--------|-------------|
| 1              | 0         | 0            | 0      | 0           |
| 0              | 1         | 0            | 0      | 0           |
| 0              | 0         | 1            | 0      | 0           |
| 0              | 0         | 0            | 1      | 0           |
| 0              | 0         | 0            | 0      | 1           |

Für die Abbildung wird jeweils der Mittelwert der Aggressivität jedes noch aktiven Spielers verwenden.

### 3.8.1.16 Tightness

Die Tightness wird, wie auch die Aggressivität, aus den Aktionen eines Spielers errechnet und als prozentuale Angabe abgebildet

$$tightness = \frac{\text{anzahl\_folds\_im\_preflop}}{\text{anzahl\_runden}}$$

Wie der Formel entnommen werden kann, ist die Tightness als «Hände, die im Preflop gefoldet werden» definiert. Eine andere Möglichkeit wäre beispielsweise die Tightness als «Hände, die sofort gefoldet werden» zu definieren. Diese Definition würde jedoch gratis Calls ausblenden.

### 3.8.1.17 Tightness kategorisch

Die Unterteilung ist, analog der Aggressivität, in 20%-Schritte gegliedert. Beginnend bei 0%-20% für die Kategorie *sehr loose* und endend bei 80%-100% für *sehr tight*. Die folgende Tabelle repräsentiert die Kategorisierung.

**Tabelle 10** Tightness Kategorien Double Deep Q Network

| Sehr tight | Tight | Ausgeglichen | Loose | Sehr loose |
|------------|-------|--------------|-------|------------|
| 1          | 0     | 0            | 0     | 0          |
| 0          | 1     | 0            | 0     | 0          |
| 0          | 0     | 1            | 0     | 0          |
| 0          | 0     | 0            | 1     | 0          |
| 0          | 0     | 0            | 0     | 1          |

Auch die Tightness wird nur für aktive Spieler erhoben. Abgebildet wird jeweils der Mittelwert der Tightness aller noch nicht ausgeschiedenen Gegner der aktuellen Runde. Die Zusammenfassung aller Gegner in ein einziges Feature verfolgt den Ansatz, die Stimmung am Tisch zu kategorisieren. Die Absicht hinter diesem Feature ist nach wie vor unverändert: Dem Algorithmus mitteilen, wie viele Hände von den aktiven Gegnern gespielt werden, um ihre Stärke einzuschätzen.

### 3.8.1.18 Risiko Faktor kategorisch

Die Implementierung entspricht einer Kombination aus zwei verschiedenen Features: Hand Ranking und Position.

#### **Referenz Hand Ranking**

Das Hand Ranking wird gemäss der Beschreibung im Kapitel 3.8.1.2 berechnet.

#### **Position**

Die Erhebung der Position entspricht der im Kapitel 3.8.1.5 beschriebenen Vorgehensweise. Abhängig von der Position wird eine Hand entweder als stärker, unverändert oder schwächer behandelt, wie der folgende Codeausschnitt veranschaulicht:

```

1. risk_factor = min(1, hand_rank * (len(round_state['seats']) /
2. number_of_active_players))
3. if early_position:
4.     risk_factor = risk_factor * 0.8
5. elif late_position:
6.     risk_factor = risk_factor * 1.2

```

Listing 15 Skalierung nach Anzahl Spieler und Position [/source/components/Features.py]

Dieses Feature beinhaltet somit gebündelte Informationen zur Spielbarkeit der eigenen Hand. Die Repräsentation als kategorischer Wert dient lediglich der Einstufung von Risiken. Jede Kategorie deckt dabei einen Bereich von 20% ab. *Sehr tief* steht somit für 0%-20% und *sehr hoch* für 80%-100%. Die Verteilung ist auf nachfolgender Tabelle gegeben.

Tabelle 11 Risiko Kategorien Double Deep Q Network

| Sehr tief | Tief | Mittel | Hoch | Sehr hoch |
|-----------|------|--------|------|-----------|
| 1         | 0    | 0      | 0    | 0         |
| 0         | 1    | 0      | 0    | 0         |
| 0         | 0    | 1      | 0    | 0         |
| 0         | 0    | 0      | 1    | 0         |
| 0         | 0    | 0      | 0    | 1         |

Der Gedanke hinter diesem experimentellen Feature war, dass der Algorithmus viele Informationen aus einem Feature lesen kann. Um dem Netzwerk möglichst unverfälschte Informationen zur Verfügung zu stellen wurde das Feature jedoch nach der dritten Realisierungseinheit nicht mehr verwendet.

### 3.8.2 Übersicht Features und Realisierungseinheiten

Die folgende Tabelle zeigt, welche Features in welcher Realisierungseinheit Teil des Input Layers waren.

**Tabelle 12 Zuweisung der Features zu Realisierungseinheiten**

|                                           | RE 1 | RE 2 | RE 3 | RE 4 | RE 5 |
|-------------------------------------------|------|------|------|------|------|
| Gewinnchance Monte-Carlo                  | X    | X    |      |      |      |
| Hand-Ranking kategorisch                  |      |      | X    | X    |      |
| Wahrscheinlichkeit Handstärke kategorisch |      |      |      |      | X**  |
| Street kategorisch                        | X    | X    | X    | X*** |      |
| Position kategorisch                      | X    |      | X    | X    |      |
| Anzahl Gegner kategorisch                 |      |      | X    | X    |      |
| Pot-Grösse                                |      | X    | X    |      |      |
| Pot-Grösse kategorisch                    |      |      |      |      | X    |
| Pot-Odds                                  |      | X    | X    |      |      |
| Pot-Odds kategorisch                      |      |      |      |      | X    |
| Einsatzgrösse                             |      | X    | X    |      |      |
| Einsatzgrösse kategorisch                 |      |      |      |      | X    |
| Gegner aktiv                              | X    |      |      |      |      |
| Aggressivität                             | X    |      |      |      |      |
| Aggressivität kategorisch                 |      | X    |      |      | X*   |
| Tightness                                 | X    |      |      |      |      |
| Tightness kategorisch                     |      | X    | X    | X*   |      |
| Risiko Faktor kategorisch                 | X    | X    |      |      |      |

\* Nur im einheitlichen Input Vektor

\*\* Nur im Input Vektor für das separate Netzwerk der Streets Flop, Turn und River

\*\*\* Nur im einheitlichen Input Vektor und dem separaten Netzwerk der Streets Flop, Turn und River

Zudem zeigt die Tabelle auch gewisse Tendenzen, die nachfolgend erläutert werden.

#### 3.8.2.1 Zunehmende Anzahl verwendeter Features

Dieser Trend ist insbesondere über die ersten drei Realisierungseinheiten zu beobachten. Die Ursache dafür ist die Tatsache, dass die Features zeitgleich mit den Modellen entwickelt wurden und in den früheren Realisierungseinheiten noch nicht zur Verfügung standen.

#### 3.8.2.2 Tendenz von prozentualen zu kategorischen Angaben

In der ersten Realisierungseinheit wurde noch auf eine einzelne, prozentuale Angabe gesetzt. Die letzte Realisierungseinheit hingegen zeigt einen Input Layer, der sich ausschliesslich aus kategorischen Werten zusammensetzt. Dieser Ansatz wurde bewusst gewählt, um den Lernprozess des neuralen Netzwerkes zu beschleunigen.

#### 3.8.2.3 Abweichung von verhaltensbasierten Features

Das separate Netzwerk für die Streets Flop, Turn und River der letzten Realisierungseinheit verzichtet komplett auf verhaltensbasierte Informationen wie die Aggressivität

und Tightness. Ursache für diesen Entscheid war die Auswertung der Mails aus den AICrowd Evaluierungen gegen die gegnerischen Bots. Darin zeigte sich, dass sämtliche Gegner tight-aggressiv spielen. Da die Features Aggressivität und Tightness somit zu Konstanten und folglich zu irrelevanten Features im Netzwerk mutierten, wurden sie aus dem Input-Vektor entfernt.

Die sonst geltende Weisheit

«*Beim Poker spielt man nicht seine Karten, man spielt seine Gegner*» [36]

Trifft in diesem Fall offenbar nicht zu, da die eingereichten Spieler auf AICrowd eher ein konstantes Verhalten aufweisen.

#### 3.8.2.4 Verwenden unverfälschter Daten

Experimentelle Features, wie der Risiko Faktor, sind in der letzten Realisierungseinheit nicht mehr Teil des Input Layers. Die Absicht dahinter ist, dem neuralen Netzwerk möglichst unverfälschte Daten zur Verfügung zu stellen.

Features wie die Monte-Carlo Simulation aus der ersten Realisierungseinheit, die falsche Angaben lieferten, wurden ebenfalls systematisch eliminiert und durch verlässlichere Informationsquellen, wie statistische Angaben, ergänzt.

#### 3.8.2.5 Split Netzwerk

In der fünften Realisierungseinheit wurden auch Agenten entwickelt, die jeweils ein separates Netzwerk für den Preflop und die restlichen Streets verwendet. Grund für diesen Entscheid für die Implementierung des Features *Wahrscheinlichkeit Handstärke kategorisch*. Die Verwendung dieses Features ergibt erst Sinn, wenn die Gemeinschaftskarten bekannt sind. Diese Einstellung wird allgemein Split Network genannt. Das Split Netzwerk hat Hürden aufgezeigt:

1. Die mathematischen Modelle benötigen für das Training mindestens eine Zustandsänderung. Im Preflop kann es aber dazu kommen, dass lediglich eine Aktion ausgeführt wird, danach kommt das zweite Netzwerk zum Zuge. In diesem Fall existiert im Buffer kein last\_state, womit die Q-Learning Funktion nicht erfüllt werden kann.
2. Dieselbe Problematik gilt für das zweite Netzwerk. Wird in der Flop-Phase lediglich eine Aktion ausgeführt und das Spiel ist danach zu Ende, findet kein Zustandsübergang und damit kein Lernen statt. Auch hier muss bei Wechsel von Preflop auf Flop der Zustand mitgeführt werden. Problem dabei ist, dass der Zustandsraum im zweiten Netzwerk mit den Outs eine andere Dimension hat und damit manuell zusammengestellt werden müsste.

#### 3.8.3 Nicht implementierte Features

Die Variante, die Handkarten abzubilden und dem Algorithmus zur Verfügung zu stellen, wurde verworfen.

Grund für diesen Entscheid ist die Tatsache, dass insgesamt über 2'500'000 verschiedene Pokerhände existieren [37]. Zusammen mit den anderen Features würde der Input

Layer schnell sehr gross, was wiederum die Trainings-Performance beeinträchtigt – grösseres Netzwerk und grösserer Zustandsraum benötigen mehr Iterationen.

Zudem wurden während einer Evaluierung auf AICrowd zuerst nur 100'000 Runden gespielt, später nur noch 50'000. Diese beiden Rundenzahlen sind, verglichen mit der Anzahl möglicher Pokerhände, zu klein.

Hinzu kommt, dass die gleiche Hand nicht zwangsläufig zum gleichen Ergebnis führt – im Gegenteil: Die gleiche Hand kann in einer Runde zu grossen Gewinnen führen, während sie in einer anderen Runde zu grossen Verlusten führt.

Noch schlimmer: Die Stärke der Pokerhände ist sehr unterschiedlich abgebildet. Die drei aufeinanderfolgenden Handstärken Strasse, Flush und Full House haben sehr verschiedene Muster (aufeinander folgende Karten, gleichfarbige Karten und gleichwertige Karten). Dieser Umstand erschwert das Erkennen von Kombinationen in einem neuronalen Netzwerk basierend auf den Handkarten.

### 3.9 Belohnungssystem

Analog den Features ist für das Netzwerk ein weiterer Bestandteil von hoher Bedeutung. Die Q-Learning Gleichung ist stark abhängig vom Parameter  $r$  – der Belohnung (Reward).

#### 3.9.1 Referenzmodelle

Vorgestellte Modelle im Netz, die beispielweise auf OpenAI Gym [38] aufbauen, verwenden eine Umgebung, die nach jedem Schritt jeweils eine Belohnung zurückgeben. Ein Schritt sei dabei definiert als Übergang vom aktuellem in den nächsten Zustand.

Die in diesem Rahmen verwendete PyPokerEngine gibt hingegen keine Belohnung zurück. Ein Belohnungssystem wird somit notwendig, um die Q-Learning Voraussetzungen einzuhalten und das CAP zu lösen.

#### 3.9.2 Umsetzung

Im Laufe der Iterationen werden unterschiedliche Belohnungssysteme implementiert und in der Komponente [/source/components/Rewards.py] ergänzt. Im finalen Setup haben allen voran zwei unterschiedliche Belohnungssysteme zu guten Resultaten geführt: Simple und Complex.

##### 3.9.2.1 Simple

Das *Simple* Belohnungssystem verwendet die tatsächlichen Chips, die gesetzt oder auch verloren werden. Die Implementierung ist demnach einfach:

```
def get_reward_per_simple(self, done, won, won_amount, amount_added, folded,
blind_paid):
    # Case Folded
    if folded:
        return -amount_added - blind_paid
    # Case Lost
    elif not won:
        return -amount_added - blind_paid
    # Won Case
    if won:
        if not done:
            reward = amount_added
        if done:
            reward = amount_added + won_amount # blind will be in won_amount
    return reward
```

Listing 16 Belohnungssystem Simple, [/source/components/Rewards.py]

Bei jedem Aufruf von *receive\_round\_result\_message(...)* in der Klasse MVBasePokerPlayer wird über alle in dieser Runde durchgeführten Aktionen iteriert und für jede Aktion die Belohnung gemäss Listing 16 berechnet.

Vereinfacht heisst das:

---

Für alle Aktionen in *round\_actions* aus:

- **Fold:** Falls der Agent mit dieser Aktion Chips sowie Blinds in den Pot gegeben hat, zähle diese als negative Belohnung.
  - **Verloren:** Analog Fold
  - **Gewonnen und nicht letzte Aktion:** Zähle die Chips, die in diesem Zustand hinzugefügt wurden, als positive Belohnung
  - **Gewonnen und letzte Aktion vor round\_result:** Zähle hinzugefügte Chips als Belohnung und addiere zusätzlich den gewonnenen Pot.
- 

### 3.9.2.2 Complex

Das als Complex bezeichnete Belohnungssystem hat eine erweiterte Logik erhalten. Grundgedanke dahinter ist, je nach Zustand zusätzlich den Agenten zu strafen oder zu belohnen. Beispiele:

- Mit Hilfe von Gegner-Statistiken, die dem Agenten als Feature zur Verfügung stehen sollen, wird der Agent gebüsst, wenn er gegen einen Gegner mit sehr stark Tight-Aggressiver Strategie viele Chips verliert.
- Gewinnt der Agent mit einem gratis Call, erhält er eine zusätzliche Belohnung.

Bei der Entwicklung des *Complex* Belohnungssystem wird darauf geachtet, dass lediglich Informationen verwendet werden, die auch dem Modell im Zustandsraum übergeben werden. Ein gratis Call muss entsprechend in den Features abgebildet werden als Pot-Odds. Die Berechnung wurde in Kapitel 2.7.3.3 Berechnung Pot-Odds erläutert.

Aufgrund der Grösse der Implementierung sei nachfolgend der Auszug für verlorene Hände aus dem Complex Belohnungssystem gegeben:

```
def get_reward_complex(...):

    # Case Lost, up to 5 * initial_stack punishment
    elif not won and action_index != 0:
        if action_index == 2 and free_call and high_risk:
            reward -= amount_added # we should have called or folded
        if high_risk:
            reward -= amount_added
        if high_bet:
            reward -= amount_added
        if high_tightness:
            reward -= amount_added
        if all_in: # punish for too aggressive play
            reward -= amount_added
```

Listing 17 Belohnungssystem Complex, Teil verlorene Hand, [source/components/Rewards.py]

Im Auszug aus dem Complex Belohnungssystem gemäss Listing 17 wird damit zwischen Risiko, Pot-Odds sowie gegnerische Spielweise und eigenem Verhalten (All-In) unterschieden. Je nach Zustand wird die negative Belohnung erhöht.

Diese negative Belohnung führt durch das Model Fitting dazu, dass der Agent bei Auftreten desselben Zustandes einer anderen Strategie folgt.

## 4 Diskussion

Dieses Kapitel beschreibt die Erkenntnisse aus den vergangenen Realisierungseinheiten, der damit verbundenen Modelle und die Wahl des idealen Kandidaten für die finale AICrowd Submission.

### 4.1 Rekapitulation Modell-Entwicklung

#### 4.1.1 Q-Learning

Als erstes Modell wurde Q-Learning verwendet. Die vergleichsweise einfache Implementierung erlaubte dem Projektteam, sich zeitgleich sowohl mit der PyPokerEngine als auch mit der Software Entwicklung mit Python vertraut zu machen. Zudem konnte eine erste Einreichung auf AICrowd durchgeführt werden. Es stellte sich jedoch heraus, dass die Verwendung einer Q-Tabelle für das Abbilden der komplexen Zustände eines Pokerspiels ungeeignet ist.

#### 4.1.2 Deep Q-Networks

Die Konsequenz aus den Erkenntnissen in Zusammenhang mit Q-Learning war die Verwendung von neuralen Netzwerken, da diese dazu in der Lage sind, den komplexen Zustand des Pokerspielens abzubilden. Um von diesem Vorteil profitieren zu können, wurde der Feature Space erweitert. Durch die Verwendung von neuralen Netzwerken entstanden jedoch neue Herausforderungen, in diesem Fall insbesondere die langsame Konvergenz und das Flackern im Netzwerk.

#### 4.1.3 Double Deep Q-Networks

Das Double Deep Q-Network ist eine Verbesserung seines Vorgängers. Insbesondere das beobachtete Flackern im Netzwerk konnte weitgehend eliminiert werden. Diese Version kann als erstes stabiles Modell betrachtet werden. Durch Einführung einer verbesserten MemoryBuffer-Komponente konnte der Trainingsprozess beschleunigt werden. Der Lernprozess fand jedoch noch zu langsam statt und musste optimiert werden.

#### 4.1.4 Double Deep Q Learning mit Prioritized Experience Replay

Durch das Ergänzen des Double Deep Q-Networks um den Prioritized Experience Replay-Faktor wurde der Lernprozess beschleunigt. Da somit die identifizierten Schwachstellen des neuralen Netzwerkes behoben waren, wurde kein weiteres Modell mehr entwickelt. Stattdessen fanden in der letzten Realisierungseinheit nur noch Optimierungen am Netzwerk und dessen Features statt.

## 4.2 Spielweise der Agenten

Nebst der Cashgame Stack Entwicklung sowie Überwachung der durchgeföhrten Aktionen über grosse Rundenzahlen, stellt sich die Frage nach der sinnvollen Spielweise.

Die Auswertung dieser Frage wird mit Hilfe der Mittel zur Performanceanalyse gemäss Kapitel 3.2.1 durchgeföhr. Unter anderem wird auch mit dem gelieferten Console-Player über kleine Runden-Zahlen gegen einem im Fokus stehenden Agenten gespielt, während die anderen Agenten im Spiel teilweise untrainiert und teilweise trainiert sind. Eine solche Analyse sei nachfolgend zusammengefasst:

### Beispiel schwache Hand DdqnPerPlayer\_1

1. Im Fokus steht Agent DdqnPerPlayer\_1. Der Agent ist an der Small Blind Position.
2. Der ConsolePlayer erhält ein schlechtes Blatt (hole card). Die sinnvolle Aktion für eine Tight-Spielweise ist in diesem Fall die Aktion *fold*.

```
-- Round 1 start (UUID = bddybheohyndnqszhvns) --
=====
-- hole card --
- ['5h', '3s']
-- players information --
- 0 : Console (bddybheohyndnqszhvns) => state : participating, stack : 200
- 1 : DdqnPerPlayer_1 (hdrcgvxfobljfttmspkomf) => state : participating, stack : 199
- 2 : DdqnPlayer_2 (ocqlagbnckkjpmptyzaago) => state : participating, stack : 198
- 3 : DdqnPlayer_3 (wfjyeaguuxtdynrhcdwbj) => state : participating, stack : 200
- 4 : DdqnPlayer_4 (ptvfwtvfqvbmumqfakpua) => state : participating, stack : 200
- 5 : DdqnPlayer_5 (zhgrichbvgraiddchwlugo) => state : participating, stack : 200
=====
```

3. Der Agent DdqnPerPlayer\_1 erhält ebenfalls ein schwaches Blatt:

```
hole_card = {[list: 2] ['3h', '2h']}
  0 = {str} '3h'
  1 = {str} '2h'
  _len_ = {int} 2
round_count = {int} 1
seats = {[list: 6] [{"name": "Console", "uuid": "bddybheohyndnqszhvns"}, {"name": "DdqnPerPlayer_1", "uuid": "hdrcgvxfobljfttmspkomf"}, {"name": "DdqnPlayer_2", "uuid": "ocqlagbnckkjpmptyzaago"}, {"name": "DdqnPlayer_3", "uuid": "wfjyeaguuxtdynrhcdwbj"}, {"name": "DdqnPlayer_4", "uuid": "ptvfwtvfqvbmumqfakpua"}, {"name": "DdqnPlayer_5", "uuid": "zhgrichbvgraiddchwlugo"}]
self = {MDdqnPerPlayer} DdqnPerPlayer_1
```

4. Ein Agent raised um 65 Chips. DdqnPerPlayer\_1 folded korrekterweise.

```
-- new action --
- DdqnPerPlayer_1 (hdrcgvxfobljfttmspkomf) declared fold: 0
-- round state --
- dealer btn : Console
- street : preflop
- community card : []
- pot : main = 68.0, side = []
- players information
- 0 : Console (bddybheohyndnqszhvns) => state : folded, stack : 200
- 1 : DdqnPerPlayer_1 (hdrcgvxfobljfttmspkomf) => state : folded, stack : 199 <= SB, CURRENT
- 2 : DdqnPlayer_2 (ocqlagbnckkjpmptyzaago) => state : participating, stack : 198 <= BB
- 3 : DdqnPlayer_3 (wfjyeaguuxtdynrhcdwbj) => state : participating, stack : 135.0
- 4 : DdqnPlayer_4 (ptvfwtvfqvbmumqfakpua) => state : folded, stack : 200
- 5 : DdqnPlayer_5 (zhgrichbvgraiddchwlugo) => state : folded, stack : 200
- action histories
- preflop
- {'action': 'SMALLBLIND', 'amount': 1, 'add_amount': 1, 'player': 'DdqnPerPlayer_1 (uuid=hdrcgvxfobljfttmspkomf)'}
- {'action': 'BIGBLIND', 'amount': 2, 'add_amount': 1, 'player': 'DdqnPlayer_2 (uuid=ocqlagbnckkjpmptyzaago)'}
- {'action': 'RAISE', 'amount': 65.0, 'paid': 65.0, 'add_amount': 63.0, 'player': 'DdqnPlayer_3 (uuid=wfjyeaguuxtdynrhcdwbj)'}
- {'action': 'FOLD', 'player': 'DdqnPlayer_4 (uuid=ptvfwtvfqvbmumqfakpua)'}
- {'action': 'FOLD', 'player': 'DdqnPlayer_5 (uuid=zhgrichbvgraiddchwlugo)'}
- {'action': 'FOLD', 'player': 'Console (uuid=bddybheohyndnqszhvns)'}
- {'action': 'FOLD', 'player': 'DdqnPerPlayer_1 (uuid=hdrcgvxfobljfttmspkomf)'}
=====
```

### Beispiel starke Hand DdqnPerPlayer\_1

1. DdqnPerPlayer\_1 sowie auch ConsolePlayer erhalten eine spielfähige Hand.

```

▶ 3 hole_card = {list: 2} ['Jh', '9h']
  0 round_count = {int} 6
▶ 3 seats = {list: 6} [{"name": "Console", "uuid": "wqt... View
▶ 3 self = {MVDdqnPerPlayer} DdqnPerPlayer_1
  ...

```

2. Im Preflop deklariert ConsolePlayer ein Raise um 50 Chips. DdqnPerPlayer\_1 called, alle anderen Agenten folden.
3. Im Flop erhält ConsolePlayer mit den Community Cards ein 4-Paar. Bis zum River deklariert ConsolePlayer ein Call.
4. Im River erhält ConsolePlayer eine weitere gute Karte und entscheidet sich, nochmals um 20 zu erhöhen. DdqnPerPlayer\_1 hat mit dem J-Paar ebenfalls höhere Gewinnaussichten und entscheidet sich für den Call.

```
=====
-- new action --
- DdqnPerPlayer_1 (lygerrewzouphdveixaov) declared call: 20
-- round state --
- dealer btn : DdqnPlayer_5
- street : river
- community card : ['6s', '3c', '4h', '8d', 'Jd']
- pot : main = 140, side = []
- players information
  - 0 : Console (wqtwmccxnfkbbioevzufqb) => state : participating, stack : 130 <= SB
  - 1 : DdqnPerPlayer_1 (lygerrewzouphdveixaov) => state : participating, stack : 130 <= BB, CURRENT
  - 2 : DdqnPlayer_2 (jkuesndctpiwrhkwyung) => state : folded, stack : 200
  - 3 : DdqnPlayer_3 (qqpszcymgaskcbiddxouok) => state : folded, stack : 200
  - 4 : DdqnPlayer_4 (qanotmqjcsxjpckyddakj) => state : folded, stack : 200
  - 5 : DdqnPlayer_5 (bvlepqcnrvyjbptattze) => state : folded, stack : 200
- action histories
- preflop
  - {'action': 'SMALLBLIND', 'amount': 1, 'add_amount': 1, 'player': 'Console (uuid=wqtwmccxnfkbbioevzufqb)'}
  - {'action': 'BIGBLIND', 'amount': 2, 'add_amount': 1, 'player': 'DdqnPerPlayer_1 (uuid=lygerrewzouphdveixaov)'}
  - {'action': 'FOLD', 'player': 'DdqnPlayer_2 (uuid=jkuesndctpiwrhkwyung)'}
  - {'action': 'FOLD', 'player': 'DdqnPlayer_3 (uuid=qqpszcymgaskcbiddxouok)'}
  - {'action': 'FOLD', 'player': 'DdqnPlayer_4 (uuid=qanotmqjcsxjpckyddakj)'}
  - {'action': 'FOLD', 'player': 'DdqnPlayer_5 (uuid=bvlepqcnrvyjbptattze)'}
  - {'action': 'RAISE', 'amount': 50, 'paid': 49, 'add_amount': 48, 'player': 'Console (uuid=wqtwmccxnfkbbioevzufqb)'}
  - {'action': 'CALL', 'amount': 50, 'paid': 48, 'player': 'DdqnPerPlayer_1 (uuid=lygerrewzouphdveixaov)'}
- flop
  - {'action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'Console (uuid=wqtwmccxnfkbbioevzufqb)'}
  - {'action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'DdqnPerPlayer_1 (uuid=lygerrewzouphdveixaov)'}
- turn
  - {'action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'Console (uuid=wqtwmccxnfkbbioevzufqb)'}
  - {'action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'DdqnPerPlayer_1 (uuid=lygerrewzouphdveixaov)'}
- river
  - {'action': 'RAISE', 'amount': 20, 'paid': 20, 'add_amount': 20, 'player': 'Console (uuid=wqtwmccxnfkbbioevzufqb)'}
  - {'action': 'CALL', 'amount': 20, 'paid': 20, 'player': 'DdqnPerPlayer_1 (uuid=lygerrewzouphdveixaov)'}
=====
```

5. ConsolePlayer gewinnt. DdqnPerPlayer\_1 ist ein Risiko eingegangen, hatte aber das höchste auf dem Board liegende Paar.

```
=====
-- winners --
- Console (wqtwmccxnfkbbioevzufqb) => state : participating, stack : 270.0
-- hand info --
- Console (wqtwmccxnfkbbioevzufqb): hand => Two Pair (['Jc', '4d'])
-- round state --
- dealer btn : DdqnPlayer_5
- street : showdown
- community card : ['6s', '3c', '4h', '8d', 'Jd']
```

Weitere Analysen sind in Form von Rohdaten unter [Beilagen/Spielweise] gegeben.

#### 4.2.1 Fazit

Allgemein konnte beobachtet werden, dass die trainierten DdqnPer Agenten die eigene Stärke in Form von fold oder call gut einschätzen konnten. Die meisten der trainierten Agenten haben jedoch meist ein Tight-Passives Verhalten gezeigt. So haben die Agenten bei starker Hand weniger schnell zu raise gegriffen und dadurch kleinere Belohnungen für sich gewinnen können.

Die passive Spielweise wird erst bei sehr starken Händen zu einer aggressiven Spielweise. In nachfolgendem Beispiel hat der vorangehende ConsolePlayer ein Raise mit 20 Chips durchgeführt. DdqnPerPlayer\_1 (K, K in der Hand) hat sich davon nicht beirren lassen und zu einem noch höheren Raise gegriffen, wie nachfolgende Auswertung zeigt.

```
-----  
-- winners --  
- DdqnPerPlayer_1 (xxkqfdwugvdsctoypqmiw) => state : participating, stack : 370.0  
-- hand info --  
- DdqnPerPlayer_1 (xxkqfdwugvdsctoypqmiw): hand => Two Pair ([`Kd`, `Kh`])  
-- round state --  
- dealer btn : DdqnPerPlayer_5  
- street : showdown  
- community card : ['As', '8s', '2h', 'Ah', 'Qh']  
- pot : main = 340.0, side = []  
- players information  
- 0 : Console (cfmermqcxciznunjdakavya) => state : participating, stack : 30.0 <= SB, CURRENT  
- 1 : DdqnPerPlayer_1 (xxkqfdwugvdsctoypqmiw) => state : participating, stack : 370.0 <= BB  
- 2 : DdqnPlayer_2 (tyudtpwenzscrnoacadmk) => state : folded, stack : 200  
- 3 : DdqnPlayer_3 (vhrrmbuhjgrdenhmstclcw) => state : folded, stack : 200  
- 4 : DdqnPlayer_4 (eksfhieikyvkloyaofhukc) => state : folded, stack : 200  
- 5 : DdqnPlayer_5 (wehykkjeniuifiyengisdn) => state : folded, stack : 200  
- action histories  
- preflop  
- {`action': 'SMALLBLIND', 'amount': 1, 'add_amount': 1, 'player': 'Console (uuid=cfmermqcxciznunjdakavya)' }  
- {`action': 'BIGBLIND', 'amount': 1, 'add_amount': 1, 'player': 'DdqnPerPlayer_1 (uuid=xxkqfdwugvdsctoypqmiw)' }  
- {`action': 'FOLD', 'player': 'DdqnPlayer_2 (uid=tyudtpwenzscrnoacadmk)' }  
- {`action': 'FOLD', 'player': 'DdqnPlayer_3 (uid=vhrrmbuhjgrdenhmstclcw)' }  
- {`action': 'FOLD', 'player': 'DdqnPlayer_4 (uid=eksfhieikyvkloyaofhukc)' }  
- {`action': 'FOLD', 'player': 'DdqnPlayer_5 (uid=wehykkjeniuifiyengisdn)' }  
- {`action': 'RAISE', 'amount': 20, 'paid': 19, 'add_amount': 18, 'player': 'Console (uuid=cfmermqcxciznunjdakavya)' }  
- {`action': 'RAISE', 'amount': 170.0, 'paid': 168.0, 'add_amount': 150.0, 'player': 'DdqnPerPlayer_1 (uid=xxkqfdwugvdsctoypqmiw)' }  
- {`action': 'CALL', 'amount': 170.0, 'paid': 150.0, 'player': 'Console (uuid=cfmermqcxciznunjdakavya)' }  
- flop  
- {`action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'Console (uuid=cfmermqcxciznunjdakavya)' }  
- {`action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'DdqnPerPlayer_1 (uid=xxkqfdwugvdsctoypqmiw)' }  
- turn  
- {`action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'Console (uuid=cfmermqcxciznunjdakavya)' }  
- {`action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'DdqnPerPlayer_1 (uid=xxkqfdwugvdsctoypqmiw)' }  
- river  
- {`action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'Console (uuid=cfmermqcxciznunjdakavya)' }  
- {`action': 'CALL', 'amount': 0, 'paid': 0, 'player': 'DdqnPerPlayer_1 (uid=xxkqfdwugvdsctoypqmiw)' }  
=====
```

Damit konnte der DdqnPerPlayer\_1 Agent einen hohen Gewinn einfahren.

Es wird davon ausgegangen, dass ein gut spielender Mensch gegen den entwickelten Agenten gewinnt. Das liegt vor allem an gewissen Mustern, die mit der Zeit beim Agenten beobachtet werden können, wie beispielsweise das Tight-Aggressive Verhalten. Große raise-Werte deuten auf eine starke Hand des Agenten hin. Ein Mensch passt sich diesem Verhaltensmuster an und reagiert entsprechend mit fold, wenn selbst keine sehr starke Hand gehalten wird.

Zusammengefasst haben unabhängig durchgeführte Auswertungen gezeigt, dass der Agent ein gutes Verständnis über die Umgebung und die Resultate der Aktionen hat.

### 4.3 Lokale Auswertungen

Die folgenden Abbildungen sollen die entwickelten Dqn, Ddqn und DdqnPer Agenten vergleichen.

1. Die Agenten verwenden jeweils dieselben Komponenten und Einstellungen (Features, Reward, Hyperparameter). Andernfalls wird ein entsprechender Hinweis gegeben.
2. Die Replay-Funktionalität ist bei den Vergleichen jeweils aktiv – das Lernen findet somit jederzeit statt. Grund dafür ist auch um allfällige Adaption der Modelle vergleichen zu können, wenn diese gegen unterschiedliche Agenten trainiert werden und aufeinandertreffen.

Weitere Rohdaten zu den Tests sind unter [Beilagen/Plots/] gegeben.

#### 4.3.1 Dqn, Ddqn und DdqnPer

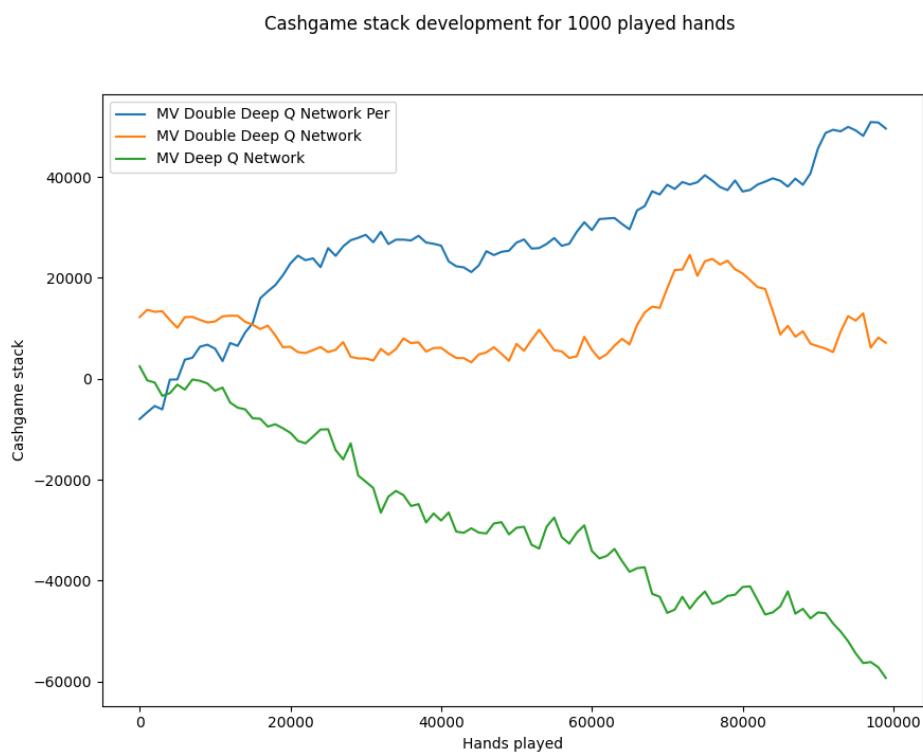


Abbildung 24 Dqn vs Ddqn vs DdqnPer

Abbildung 24 Dqn vs Ddqn vs DdqnPer vergleicht die Cashgame Stack Entwicklung über 100'000 Runden für vortrainierte Agenten. Der DdqnPer Agent (blaue Linie) verfolgt allgemein eine gewinnbringende Strategie. Der Dqn Agent (grüne Linie) zeigt zudem die erwartete, höhere Volatilität. Die Volatilität fällt im Vergleich zum DdqnPer Agent stärker aus, was gut ersichtlich wird, wenn die grüne und blaue Linie verglichen werden. Dies soll nochmals die Erkenntnisse der Realisierungseinheiten bestätigen.

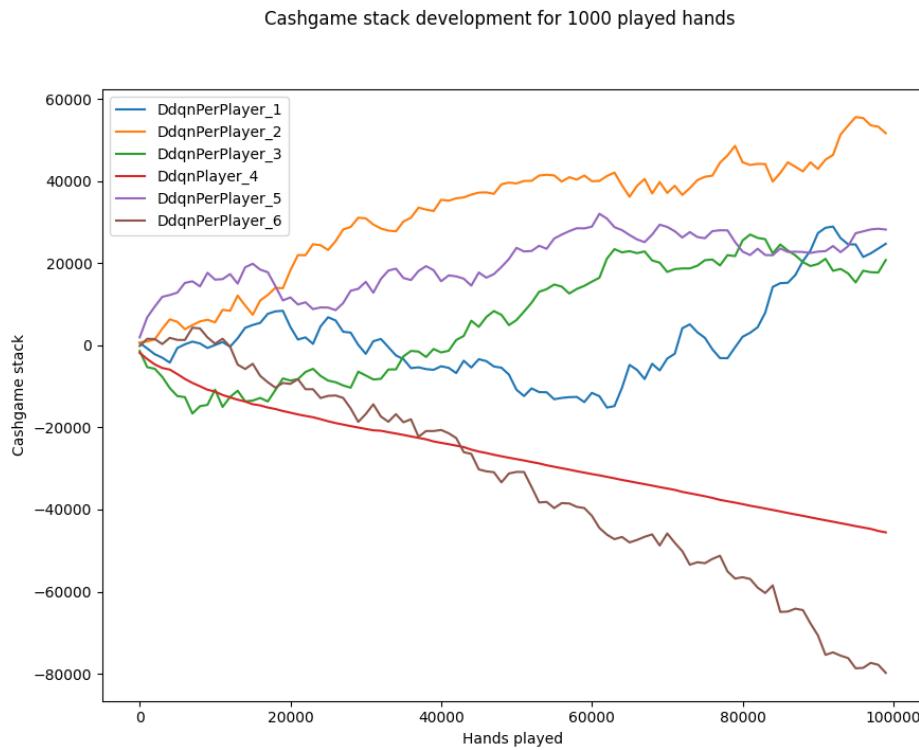


Abbildung 25 Vergleich DdqnPer und Ddqn Agenten

Tabelle 13 Konfiguration Agenten in oberer Abbildung

|                 | Split Network | Belohnungssystem |
|-----------------|---------------|------------------|
| DdqnPerPlayer_1 | Ja            | Simple           |
| DdqnPerPlayer_2 | Ja            | Simple           |
| DdqnPerPlayer_3 | Ja            | Simple           |
| DdqnPlayer_4    | Nein          | Complex          |
| DdqnPerPlayer_5 | Nein          | Simple           |
| DdqnPerPlayer_6 | Nein          | Complex          |

Ein weiterer Vergleich zeigt die Charakteristiken von Ddqn und DdqnPer. Die DdqnPer Agenten verfolgen zu diesem Zeitpunkt bereits eine Tight-Aggressive Spielweise. Der Ddqn Agent ohne Priorisierung schafft es nicht, genügend aussagekräftige Erfahrungen für das Lernen zu nutzen und konvergiert zu einem Fold-Agenten (rote Linie). Auch ersichtlich ist, dass ein PER Agent ohne Split-Netzwerk und *Complex* Belohnungssystem anfangs gut startet, dann jedoch aufgrund der hohen Belohnungs-Variabilität des *Complex*-Belohnungssystems als einziger PER Agent keinen positiven Trend erreichen kann. In unterschiedlichen Tests konnte allgemein nachvollzogen werden, dass der PER Agent mit hoher Variabilität der Belohnung keine gute Performance erreicht.

Unter Verwendung von zwei separaten Netzwerken und dem Simple Belohnungssystem erzielen die DdqnPer Agenten jeweils den höchsten Cashgame Stack. Dieses Ergebnis liess sich in mehreren voneinander unabhängig Testläufen mit jeweils mindestens einer Million gespielter Hände nachweisen. Das entwickelte Outs-Feature wird daher als wichtiges Feature erachtet.

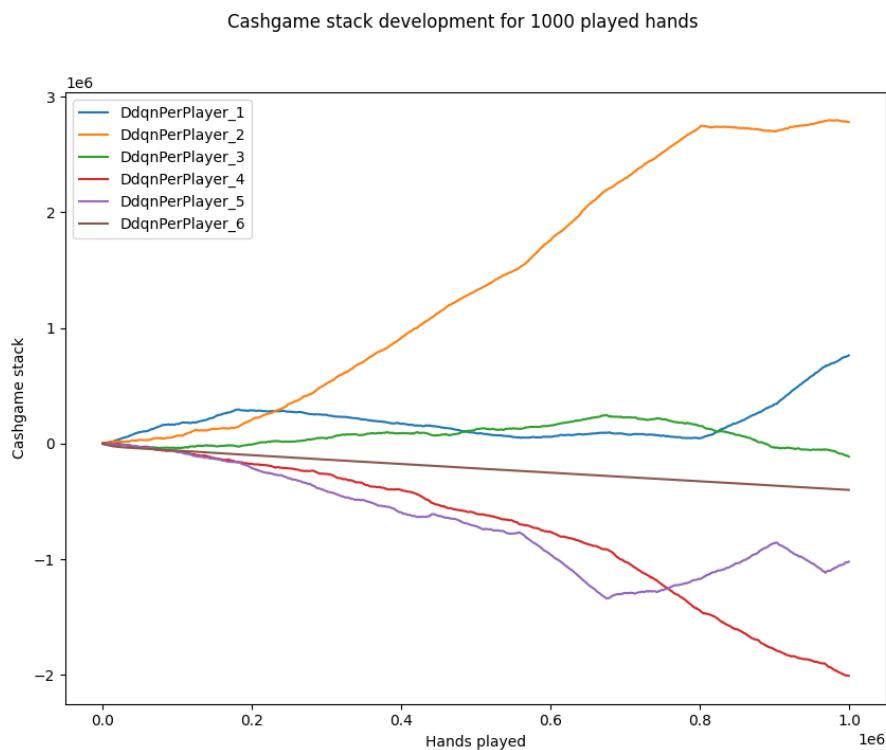


Abbildung 26 Cashgame Stack Entwicklung über eine Million Runden

Tabelle 14 Konfiguration der Player Klassen für den Modell Vergleich

|                  | Split Network | Belohnungssystem |
|------------------|---------------|------------------|
| DdqnPPerPlayer_1 | Ja            | Simple           |
| DdqnPPerPlayer_2 | Ja            | Simple           |
| DdqnPPerPlayer_3 | Nein          | Simple           |
| DdqnPPerPlayer_4 | Nein          | Simple           |
| DdqnPPerPlayer_5 | Ja            | Complex          |
| DdqnPPerPlayer_6 | Nein          | Complex          |

Tabelle 14 Konfiguration der Player Klassen für den Modell Vergleich zeigt die verwendeten Konfigurationen für die Agenten aus oberer Abbildung. In diesem Vergleich wurden eine Million Runden gespielt.

DdqnPPerPlayer\_1 und DdqnPPerPlayer\_2 belegen im Wettkampf den ersten und zweiten Platz. Beide Agenten verwenden dieselbe Konfiguration. Agent DdqnPPerPlayer\_2, welcher den ersten Platz belegt, zeigt zudem eine starke Konvergenz der Aktionen:

Sum of actions per 1000 played hands for DdqnPerPlayer\_2

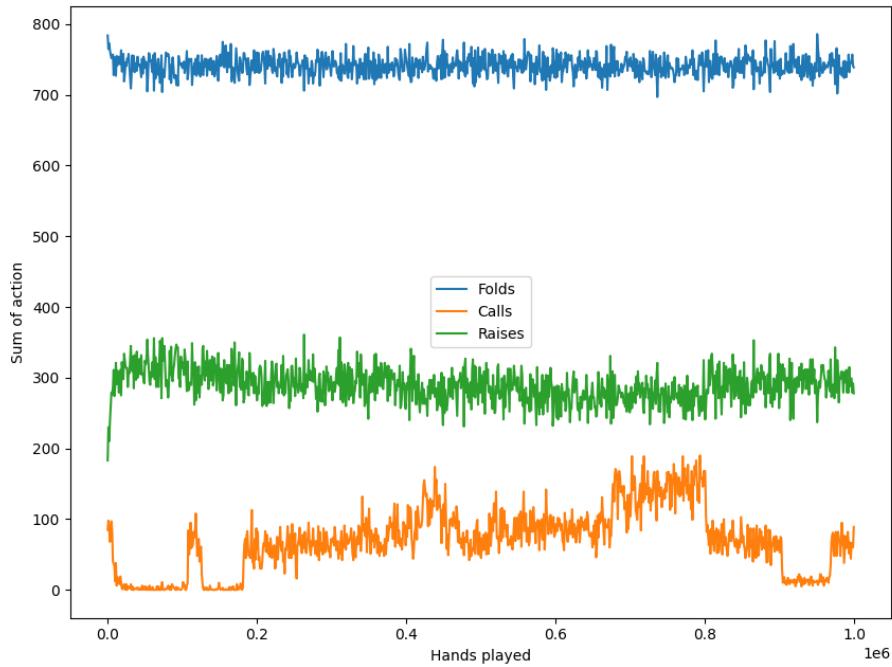


Abbildung 27 Summe der Aktionen DdqnPerPlayer\_2 für jeweils 1000 Hände

Die Strategie des Agenten entspricht der Tight-Aggressiven Spielweise.

#### 4.3.2 Fazit

Basierend den Erkenntnissen aus den vergangenen Realisierungseinheiten wird für die finale Auswertung das *Double Deep Q-Learning mit Prioritized Experienced Replay*-Modell mit dem *Simple* Belohnungssystem sowie dem Split-Netzwerk verwendet.

#### 4.4 AICrowd

Dieses Kapitel erläutert die Resultate aus AICrowd und daraus abgeleitete Schlussfolgerungen.

Unter [Beilagen/AICrowd\_Resultate] sind alle Statistiken abgelegt. Nachfolgend werden die Resultate präsentiert, zum besseren Verständnis wurden Eckdaten ergänzt.

**Tabelle 15 AICrowd Evaluationen**

| Evaluation        | Bemerkung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                   | <b>Agent:</b> DdqnPer<br><b>Split Netzwerk:</b> Nein<br><b>Modell vortrainiert:</b> Ja, gegen Baseline                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 260720_Evaluation | <b>Agent:</b> DdqnPer<br><b>Split Netzwerk:</b> Nein<br><b>Modell vortrainiert:</b> Ja, gegen Baseline<br>Agent erreicht win_rate von 43.93% und sehr aggressive Spielweise. Dies führt zu einem hohen, negativen Cashgame Stack. Der Agent belegt damit den letzten Platz.<br>Learnings: Das lokale Training ist unzureichend. Der Agent ist zudem gegen Baseline Spieler trainiert worden.                                                                                                                                                                                                                                             |
|                   | <b>Aufgrund ausserplanmässiger Evaluation wird hier derselbe Agent wie in der vorherigen Evaluation verwendet.</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 280720_Evaluation | Agent erreicht nun win_rate von 56.7%. 78.65% fold-Rate, hohe Call-Anzahl. Spielt damit Tight-Passive.<br>Ein gegnerischer Agent hat neu nahezu 0% fold-Rate.<br>Hoher Cashgame Stack. Bei looser Spielweise von Gegnern und gleichzeitiger Gewinnrate von über 50% ist ein Call-Bot eine funktionierende Spielweise. Während in der vorherigen Evaluation ein hoher negativer Stack erreicht wurde, erreicht der Agent nun einen hohen positiven Stack. Der Agent erreicht mit abgerundet 600'000 Chips den 2. Platz.<br>Dieser Unterschied zur vorherigen Evaluation zeigt den Einfluss des Multi-Agenten Systems auf die Performance. |
|                   | <b>Agent:</b> DdqnPer<br><b>Split Netzwerk:</b> Ja<br><b>Modell vortrainiert:</b> Ja, gegen Tight-Agenten                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 030820_Evaluation | Agent erreicht lediglich win_rate von 30.89%, 78.96% fold-Rate, hohe raise Anzahl – damit Tight-Aggressive Spielweise mit schlechter Gewinnchance. Ein Agent auf AICrowd hat Tightness von 0%, spielt damit (nahezu) alle Hände. Der Agent hat dadurch bei jeder Runde mindestens einen Gegner, was die win_rate senkt.<br>Cashgame Stack gut, der Agent belegt trotz niedriger Gewinnrate einen Podest-Platz.                                                                                                                                                                                                                           |

**Agent:** DdqnPer

**Split Netzwerk:** Ja

**Modell vortrainiert:** Ja, gegen Tight-Agenten über mehrere Iterationen

060820\_  
Evaluation

Agent erreicht win\_rate von 63.53%, 75.62% fold-Rate, ausgeglichene Call- und Raise-Anzahl. Gegnerische Agenten spielen nahezu alle Tight-Aggressive. Der Bereich des Cashgame Stack auf 50'000 Hände aller Agenten liegt zwischen -130'000 und 167'000.

Die Performance des Multi-Agenten System wird als gut eingeschätzt. Aufgrund dessen wird der Vergleich der Agenten in dieser Runde als massgebend erachtet.

Der Agent belegt den zweiten Platz und erreicht einen guten Cashgame Stack von nahezu 100'000. Verweis: [060820\_Leaderboard.png]

---

Für die abschliessende Evaluation wird das Training des Agenten aus der Evaluation vom 06.08 oben über weitere Iterationen durchgeführt. Zusätzlich werden teilweise auch CallBaselinePlayer im Training integriert. Dadurch konnte unter anderem auch der Cashgame Stack gegen Baseline Agenten auf 310'000 Chips bei 50'000 Runden erhöht werden.

#### 4.4.1 Fazit

Die über die Iterationen gesammelten Resultate haben allgemein zu folgenden Erkenntnissen geführt:

1. Ziel von AICrowd ist es, vortrainierte Agenten miteinander zu vergleichen.
2. Die gewählten Modelle wurden lokal geprüft, getestet und trainiert, sowie anschliessend auf AICrowd jeweils gegen weitere Modelle verglichen.
3. Die wichtigsten Aspekte für den Vergleich der Agenten sind somit:
  - a. Kann das gewählte Modell den Zustandsraum mit Hilfe der Belohnung und unter Anwendung der TD-Methode verstehen.
  - b. Ist der Agent lokal in korrekten Konfigurationen trainiert worden. Darin enthalten sind Aspekte wie: sind die Gegner genügend stark sowie wird der Zustandsraum genügend untersucht.

Die Q-Learning Gleichung erfordert eine Zustandsänderung, um erfüllt zu werden (last\_state, state). Treten beispielsweise sechs Tight-Passive und Tight-Aggressive Agenten gegeneinander an, werden die neuronalen Netzwerke auch auf grosse Anzahl Runden nur auf diesen Umstand hintrainiert. Der Zustandsraum wird damit selten korrekt erforscht, da in wenigen Fällen zwei Agenten bis zum abschliessenden River eine Call- oder Raise-Aktion durchführen. Wird nun stattdessen in einem weiteren Training eines vortrainierten Agenten ein Call-Agent dazu genommen, haben die Agenten allgemein über mehrere Zustandsänderungen hinweg Erfahrungen.

So konnten beispielsweise früh Agenten auf AICrowd geladen werden, die gegen Baseline Spieler bis zu 400'000 Chips einsammeln konnten. Diese Agenten wurden jeweils

auch mit Baseline Spielern trainiert. Gegen Tight-Passive/Tight-Aggressive Agenten konnte kein guter Cashgame Stack erreicht werden. Der Fokus wurde danach auf Training gegen ähnliche Agenten umgestellt und die Baseline Spieler vernachlässigt, wodurch gegen andere Tight-Agenten gute Resultate erzielt werden konnten.

Die Agenten wurden somit bis zur vorletzten Evaluation allen voran gegen AICrowd ähnliche Gegner trainiert, da diese Resultate im Fokus lagen und uns Informationen zur Qualität des Modells geben konnten.

#### 4.5 Lessons Learned

Gemäss den Erläuterungen werden allgemein drei ausschlaggebende Bestandteile im Projektrahmen erkannt:

1. Das korrekte Modell wählen gemäss Rekapitulation Modell-Entwicklung.
2. Den Zustandsraum mit passenden Features abbilden und die Spielweise mit jeweils kleinen Änderungen beobachten.
3. Das Training über mehrere Trainingskonfigurationen und mit unterschiedlichen Agentenkonfigurationen durchführen.

Die Trainings der Modelle nehmen viel Zeit in Anspruch. Eine lokale Evaluation, bestehend aus einer Million Runden, dauert typischerweise zwei bis drei Tage. Dadurch findet der fundamentale Lernprozess langsam statt und der Vergleich der verschiedenen Konfigurationen der Agenten benötigen jeweils viel Zeit. Ein korrektes Training hat zudem mehrere Trainingsläufe beinhaltet mit jeweils unterschiedlichen Agentenzusammstellungen.

## 5 Fazit

Dieses Kapitel beschreibt abschliessend die Erfahrungen, die sich aus der Projektdurchführung ergeben.

### 5.1 Agile Entwicklung

Die Entwicklung der Q-Learning Modelle gewährte einen tieferen Einblick in die mathematischen Grundlagen und führte zu einem besseren Verständnis. Mit Hilfe der Grobplanung konnte zudem der sonst sehr offenen Arbeit eine Struktur gegeben werden. Innerhalb dieser Leitplanken wurde agil entwickelt, um möglichst flexibel auf neue Erkenntnisse reagieren zu können und trotzdem das Ziel der einzelnen Realisierungseinheiten nicht aus den Augen zu verlieren.

### 5.2 Projekt ausserhalb der Komfortzone

Das Projekt wurde mit bisher unbekannten Techniken wie Machine Learning und Python durchgeführt. Diese zusätzliche Hürde wurde sowohl als Risiko, als auch als Herausforderung verstanden und diente als zusätzlicher Ansporn.

### 5.3 Häufige Einreichungen

Durch das regelmässige Einreichen von Agenten auf AICrowd konnten insbesondere in der zweiten Hälfte des Projektes die Vor- und Nachteile der implementierten Ansätze schnell beurteilt werden.

### 5.4 Grenzen

Eine Pokerrunde verläuft nicht deterministisch. Die gleichen Hole Cards können in einer Runde hohe Gewinne erbringen, während sie in anderen zu grossen Verlusten führen. Diese Eigenschaft wirkt sich auch auf die implementierten Lernalgorithmen aus, unabhängig von der gewählten Form. Somit unterliegt der Lernprozess der neuralen Netzwerke immer einer gewissen Ungenauigkeit.

### 5.5 Erreichtes

Trotz diesen Abweichungen fand mit jedem neuen Modell jeweils eine Verhaltensoptimierung des Spielers statt, die sich zuerst lokal und später auch auf AICrowd wieder spiegelte. Die stetig gestiegenen Gewinnraten haben direkte Hinweise auf die Verbesserungen gegeben. Die Agenten erzielen gegen unterschiedliche Gegner Profit, seien dies Baseline Agenten oder auch komplexere Implementationen. Die genutzten Modelle konnten von den Autoren mit Hilfe der zur Verfügung gestellten Paper entwickelt werden, was allgemein Flexibilität hinsichtlich der Agenten wie auch der Komponenten geboten hat.

## 5.6 Ausblick

Das Projekt bietet trotz dem Erreichten noch viele Ansätze, um weiterentwickelt zu werden. Eine Auswahl davon ist hier aufgelistet:

### 5.6.1.1 Einbinden weiterer Q-Learning Modelle

Die Weiterentwicklung der implementierten Q-Learning Modelle begrenzt sich in diesem Projekt auf das Double Deep Q-Learning mit Prioritized Replay-Modell. Die Möglichkeiten sind damit keineswegs ausgeschöpft. Weitere Modelle wie Dueling-, Distributional- und das Noisy-Double Deep Q-Network bieten Ansätze, die ebenfalls implementiert werden können, um sich mit den vorhandenen Modellen zu messen.

### 5.6.1.2 Kombination von Deep Reinforcement Learning Modellen

Nebst dem Einbinden von weiteren Modellen können diese auch kombiniert werden. Ein solcher Ansatz wird in der Publikation *Rainbow: Combining Improvements in Deep Reinforcement Learning* beschrieben. Dabei sind, je nach zugrundeliegendem Spiel, höhere Punktzahlen in der Kombination von verschiedenen Modellen als mit den einzelnen Modellen erzielt worden. [39]

### 5.6.1.3 Prognosen über Gegner implementieren

Im Rahmen dieses Projektes wurde das neuronale Netzwerk lediglich für die Bestimmung der eigenen Aktionen verwendet. Sowohl die nächste Aktion als auch die Handkarten der Gegner, bieten sich als Eigenschaften an, die ebenfalls durch die Verwendung eines neuronalen Netzwerkes vorhergesagt werden können.

### 5.6.1.4 Bestimmung der Einsatzgrösse

Die Bestimmung der Einsatzgrösse erfolgte in diesem Projekt indirekt durch den entsprechenden Knoten im Output-Layer, die jeweils mit gewissen Beträgen assoziiert waren. Es würde sich anbieten, die Bestimmung der Chips für eine Raise-Aktion beispielsweise ebenfalls über einen Machine Learning Ansatz durchzuführen. Dadurch könnte sich der ökonomische Aspekt des Pokerspielens noch stärker auf die Verhaltensweise des eigenen Agenten auswirken.

Die Liste ist nicht abschliessend. Machine Learning Ansätze werden häufig bei komplexen Zustandsräumen untersucht, um auch weitere Anwendungsdomänen zu erkennen. Die in dieser Arbeit entwickelten Agenten könnten demnach auch auf anderen Gebieten untersucht werden, bei denen ebenfalls das Credit Assignment Problem im Fokus steht.

## 5.7 Schlusswort

Die Arbeit hat grossen Wert auf die Entwicklung von Modellen sowie Features gelegt, Optimierungen passend zur vorhandenen Problemdomäne eingeführt und unterschiedliche Modelle miteinander verglichen. Passend zum grossen Zustandsraum wurden gute Agentenzusammenstellungen sowie unterschiedliche Trainingskonfigurationen als essenziell erachtet. Es wurde aufgezeigt, dass die Anwendung von Double Deep Q Networks mit Prioritized Experience Replay zu einem Agenten geführt hat, welcher eine profitable Spielweise in Texas Holdem Poker gegen unterschiedliche Gegner erlernen konnte. Die entwickelten Modelle haben den Podest-Platz im Spiel gegen fünf weitere Agenten erreicht. Der Machine Learning Ansatz zur Lösung der Poker-Problemdomäne wird als spannender Ansatz wahrgenommen. Die Autoren sind mit den Ergebnissen der Arbeit sehr zufrieden.

## 6 Projektdurchführung - Einfluss Pandemie

Zu Beginn der Arbeit kam es zu einer globalen Pandemie. Das als COVID-19 bekannte Virus führte dazu, dass die Sitzungen mit der Betreuung über Online-Videokonferenzen durchgeführt werden mussten. Der Zugang zu den Räumlichkeiten der Fachhochschule Nordwestschweiz wurde vom 17. März 2020 bis zum 8. Juni 2020 vorenthalten. Die Studierenden haben während dieser Zeit ihre Pair-Programming Tätigkeiten ebenfalls auf Online-Sitzungen umgestellt, konnten aber aufgrund gut funktionierender Arbeitsaufteilung relativ unabhängig arbeiten.

In den Sitzungen mit der Betreuung hat das Projektteam allen voran die fehlende Mimik der Betreuung und kürzer kommenden Feedbacks als weitere Schwierigkeit angesehen. Dadurch konnte das Team schlechter einschätzen, ob die Betreuung mit dem Vorgehen einverstanden ist. Um dem entgegenzuwirken, hat das Team eine kurze Umfrage<sup>3</sup> durchgeführt. Die Umfrage hat aufgezeigt, dass die Betreuung bezüglich des Vorgehens und allgemeine Einschätzung, ob das Projektteam die Problemdomäne korrekt erfasst hat, Unsicherheiten gezeigt – die Bewertung lag zwischen 6 – 8 Punkten von 10. Als Reaktion auf die Umfrage hat das Team eine freiwillige, nicht bewertete Zwischenpräsentation im Rahmen einer einstündigen Sitzung durchgeführt, um den gesamten Projektverlauf etwas näher zu erläutern und der Betreuung mehr Einblicke in das Vorgehen zu geben.

Die Autoren haben sich bemüht, die Sitzungen mit kurzen Präsentationen visuell zu untermalen. Daher wird als Protokoll-Medium von anfänglichen Fliesstext-Protokollen auf Präsentationen umgestellt.

---

<sup>3</sup> Siehe [Beilagen/Umfrage Ergebnis/ IP6 [Cudemo, Schärz] Umfrage Betreuung]

## 7 Developer Guide

Dieses Kapitel soll die Nachvollziehbarkeit des erstellten Quellcodes vereinfachen und Design Entscheidungen erläutern.

### 7.1 Systemarchitektur

Im Rahmen der Projektarbeit sind verschiedene Agenten, Features und Reward-Syste me entstanden. Um die einzelnen Implementationen schnell anpassen oder austauschen zu können, sind die Funktionalitäten in separaten Klassen abgebildet. Das vollständige Klassendiagramm befindet sich im Anhang, um die Übersicht zu gewährleisten, wurde stellenweise darauf verzichtet, sämtliche Attribute von Klassen aufzuführen. Für eine ausführliche Dokumentation der einzelnen Methoden, deren Parameter und Rückgabewerte empfiehlt sich die Konsultation der Inline-Dokumentation. Die folgenden Unterkapitel beschreiben die Ansätze, welche mit den jeweiligen Komponenten adressiert werden.

#### 7.1.1 MVQLearningPlayer

Die Klasse MVQLearningPlayer beinhaltet die erste implementierte Version von Q-Learning, welche als Minimum Viable Product auf AICrowd geladen wurde. Wie im Kapitel 3.3.6 beschrieben, wurde das einfache Q-Learning Modell, und somit auch diese Klasse, nach Abschluss der ersten Realisierungseinheit nicht weiterverfolgt.

Das Präfix MV hat keine technische Bedeutung. Es setzt sich aus den ersten Buchstaben der Vornamen der beiden Teammitglieder, Michael und Vito, zusammen und dient lediglich dem Zweck, ihr Selbstbewusstsein zu stärken.

#### 7.1.2 MVDeepQPlayer

Die Klasse MVDeepQPlayer ist der direkte Nachfolger der Klasse MVQLearningPlayer und beinhaltet die erste implementierte Version von Deep Q-Learning.

Diese Klasse verwendet noch keine eigene Basisklasse. Sämtliche Player-Klassen, die nach der Klasse MVDeepQPlayer implementiert werden, basieren auf einer einheitlichen Basisklasse.

#### 7.1.3 MVBasePokerPlayer

Die Klasse MVBasePokerPlayer dient als Elternklasse der verschiedenen Implementatio nen von Player-Klassen und leitet selbst von der gegebenen BasePokerPlayer Klasse ab.

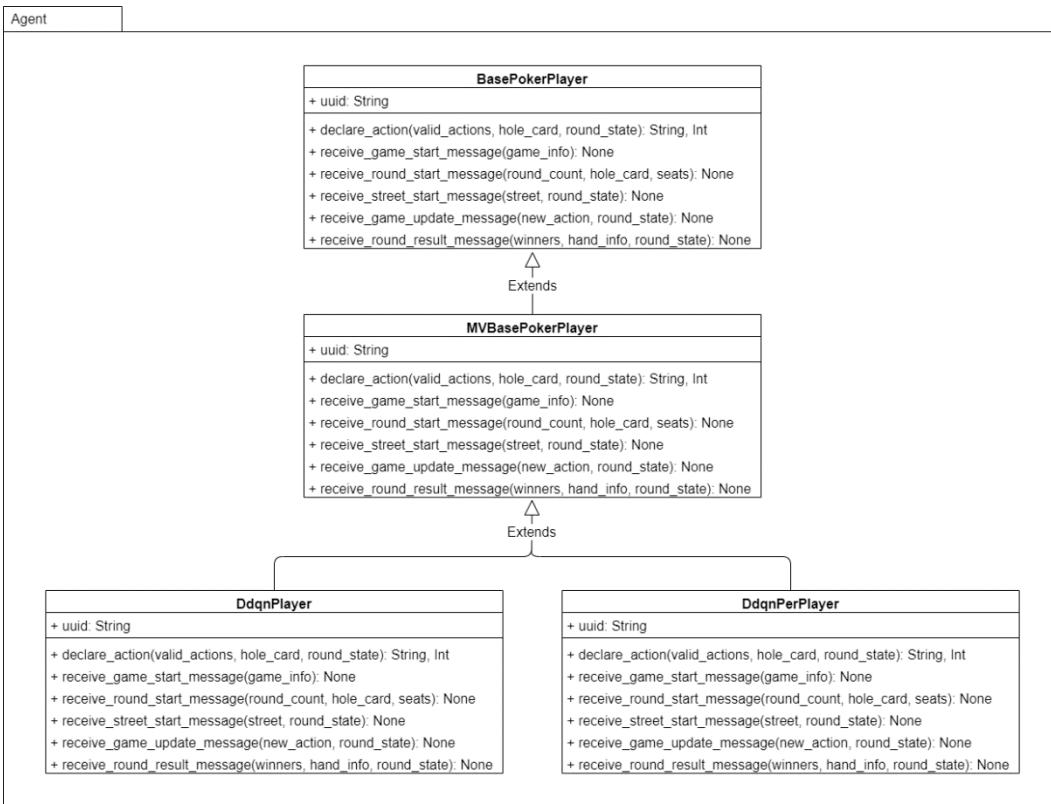


Abbildung 28 Klassendiagramm Agent Package

Abbildung 28 Klassendiagramm Agent Package zeigt diese Beziehungen. Um die Übersichtlichkeit zu gewährleisten, werden die zahlreichen Attribute der Player-Klassen weggelassen und es sind nur zwei Player-Klassen abgebildet.

Die Klasse MVBasePokerPlayer bildet die grundsätzliche Funktionalität der Player-Klassen ab. Dazu gehören das Implementieren der Methoden der Elternklasse BasePokerPlayer, das zur Verfügung stellen der Daten für den Input Layer des neuronalen Netzwerks, das Erheben von Statistiken über das Verhalten der Gegner, das Bestimmen des Rewards für getroffene Entscheidungen etc.

Durch das Verwenden einer gemeinsamen Basisklasse wird die Verwendung des gleichen Feature Standes für sämtliche Player-Klassen gewährleistet. Durch die zentrale Verwaltung kann Zeit bei der Implementierung von Player-Klassen gespart werden.

#### 7.1.4 MVDqnPlayer, MVDdqnPlayer, MVDdqnPerPlayer, MVDdqnKerasPlayer

Die vier Player-Klassen bilden jeweils eine erarbeitete Form von Deep Q-Learning ab und erben von der gemeinsamen Elternklasse MVBasePokerPlayer.

Tabelle 16 Beziehung zwischen Player-Klassen und Q-Learning Modellen

| Klasse          | Q-Learning Form                                          |
|-----------------|----------------------------------------------------------|
| MVDqnPlayer     | Deep Q Learning                                          |
| MVDdqnPlayer    | Double Deep Q Learning                                   |
| MVDdqnPerPlayer | Double Deep Q Learning mit Prioritized Experience Replay |

|                   |                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------|
| MVDdqnKerasPlayer | Double Deep Q Learning, verwendet Keras Referenzimplementierung von Double Deep Q Learning |
|-------------------|--------------------------------------------------------------------------------------------|

Tabelle 16 Beziehung zwischen Player-Klassen und Q-Learning Modellen zeigt das verwendete Q-Learning Modell für die jeweilige Klasse. Die Implementierung des jeweiligen Q-Learning Modells selbst befindet sich in der jeweiligen Modell-Klasse, siehe Kapitel 7.1.5.

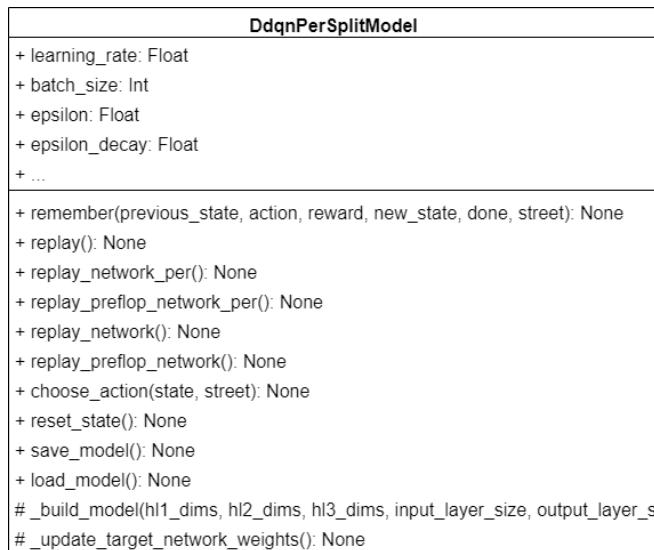
### 7.1.5 DqnModel, DdqnModel, DdqnKerasModel, DdqnPerModel, DdqnPerSplitModel

Die Model-Klassen beinhalten jeweils eine Form von Q-Learning und werden durch die Player-Klassen eingebunden.

**Tabelle 17 Beziehung zwischen Modell-Klassen und Q-Learning Modellen**

| Klasse            | Q-Learning Form                                                                                                              |
|-------------------|------------------------------------------------------------------------------------------------------------------------------|
| DqnModel          | Deep Q Learning                                                                                                              |
| DdqnModel         | Double Deep Q Learning                                                                                                       |
| DdqnKerasModel    | Double Deep Q Learning, Keras Referenzimplementierung                                                                        |
| DdqnPerModel      | Double Deep Q Learning mit Prioritized Experience Replay                                                                     |
| DdqnPerSplitModel | Double Deep Q Learning mit Prioritized Experience Replay und separaten Netzwerken für den Preflop und die restlichen Streets |

Tabelle 17 Beziehung zwischen Modell-Klassen und Q-Learning Modellen veranschaulicht, welche Klasse für die Implementierung eines spezifischen Q-Learning Modells verantwortlich ist.



**Abbildung 29 Klassendiagramm DdqnPerSplitModel**

Abbildung 29 Klassendiagramm DdqnPerSplitModel zeigt beispielhaft die Methoden der Modell-Klassen. Die Klassen weisen zum Grossteil die gleichen Methoden auf, im

Gegensatz zu den Player-Klassen wurde jedoch auf eine gemeinsame Elternklasse verzichtet, da sich die Implementierungen der einzelnen Methoden unterscheiden. Lediglich die DdqnPerSplitModel-Klasse verfügt über zusätzliche Methoden, da die verschiedenen Netzwerke für den Preflop und die anderen Streets unterschiedlich bewirtschaftet werden müssen.

### 7.1.6 MemoryBuffer

Da der Reward für die getroffenen Aktionen einer Runde erst am Ende, genauer in der receive\_round\_result\_message-Methode, bekannt wird, müssen sämtliche durch den eigenen Player getroffenen Entscheidungen zwischengespeichert werden. Diese Funktion wird durch die Klasse MemoryBuffer abgedeckt.

| MemoryBuffer                                                                                                                                |  |
|---------------------------------------------------------------------------------------------------------------------------------------------|--|
| + mem_size: Int                                                                                                                             |  |
| + mem_cntr: Int                                                                                                                             |  |
| + state_memory: List<Int>                                                                                                                   |  |
| + new_state_memory: List<Int>                                                                                                               |  |
| + action_memory: List<Int>                                                                                                                  |  |
| + reward_memory: List<Float>                                                                                                                |  |
| + terminal_memory: List<Int>                                                                                                                |  |
| + priority_memory: List<Float>                                                                                                              |  |
| + last_index: Int                                                                                                                           |  |
| + with_per: Boolean                                                                                                                         |  |
| + store_state(state, action, reward, new_state, done): None                                                                                 |  |
| + get_sample_batch(batch_size, priority_scale): List<List<Int>>, List<Int>, List<Float>, List<List<Int>>, List<Int>, List<Float>, List<Int> |  |
| + get_probabilities(priority_scale): List<Float>                                                                                            |  |
| + get_importance(probabilities, batch_size): List<Float>                                                                                    |  |
| + set_priorities(indices, td_errors, offset): None                                                                                          |  |

Abbildung 30 Klassendiagramm MemoryBuffer

Abbildung 30 Klassendiagramm MemoryBuffer zeigt die Eigenschaften und Methoden des Memory Buffers. Die gespeicherten Entscheidungen des eigenen Players und der dafür erhaltene Reward werden schlussendlich aus dem Memory Buffer in das neurale Netzwerk zurück gespiesen, um den Lernprozess zu ermöglichen.

### 7.1.7 Players

Die Klasse Players wird verwendet, um Informationen zu den Spielern am Tisch auszulesen.

| Players                                                          |  |
|------------------------------------------------------------------|--|
| + get_active_players(seats, include_own, own_uuid): List<String> |  |
| + get_index_of_player(seats, uuid, only_active_players): Int     |  |

Abbildung 31 Klassendiagramm Players

Abbildung 31 Klassendiagramm Players zeigt die angebotenen Methoden, um die Plätze am Tisch nach aktiven Spielern oder deren Position zu durchsuchen.

### 7.1.8 Features

Die Klasse Features berechnet anhand der durch die PyPokerEngine zur Verfügung gestellten Informationen die Daten für den Input Layer des neuralen Netzwerks. Die jeweiligen Gedanken hinter den Features kann dem Kapitel 3.8.1 entnommen werden.

| Features                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>+ opponent_statistics_aggressivity: Dict&lt;String, Dict&lt;String, Int&gt;&gt; + opponent_statistics_tightness: Dict&lt;Int, List&lt;Int&gt;&gt; + hand_values_pairs: Dict&lt;String, Int&gt; + hand_values_suited: Dict&lt;String, Int&gt; + hand_values_offsuit: Dict&lt;String, Int&gt; + ten_percent_one_hot: Dict&lt;String, List&lt;Int&gt;&gt; + street_one_hot: Dict&lt;String, List&lt;Int&gt;&gt; + number_of_opponents_one_hot: Dict&lt;Int, List&lt;Int&gt;&gt;  + get_hand_rank(hole_card, round_state): Float + get_hand_rank_one_hot(hole_card, round_state): List&lt;Int&gt; + get_hand_probabilities_histogram(hole_card, board): List&lt;Float&gt; + get_hand_probabilities_histogram_one_hot(hole_card, board): List&lt;Int&gt; + get_street_one_hot(street): List&lt;Int&gt; + get_position_one_hot(round_state, own_uuid): List&lt;Int&gt; + get_number_of_opponents_one_hot(seats, own_uuid): List&lt;Int&gt; + get_pot_size(pot, number_of_seats, initial_stack): Float + get_pot_size_one_hot(pot, number_of_seats, initial_stack): List&lt;Int&gt; + get_pot_odds(amount_to_play, pot): Float + get_pot_odds_one_hot(amount_to_play, pot): List&lt;Int&gt; + get_bet_size(bet, initial_stack): Float + get_bet_size_one_hot(bet, initial_stack): List&lt;Int&gt; + get_aggressivity_one_hot(seats, own_uuid): List&lt;Int&gt; + get_tightness_of_active_players(seats, own_uuid): Float + get_tightness_one_hot(seats, own_uuid): List&lt;Int&gt; + get_risk_factor(round_state, hole_card, own_uuid): Float + get_risk_factor_one_hot(round_state, hole_card, own_uuid): List&lt;Int&gt; + update_opponent_statistics(own_uuid, action, round_state): None # map_card_value_to_number(own_uuid, action, round_state): Int # get_hole_card_rank(hole_card): Float # get_cards_rank(hole_card, round_state): Float # create_opponent_statistics(seats, own_uuid): None # get_ten_percent_one_hot(percentage): List&lt;Int&gt;</pre> |

Abbildung 32 Klassendiagramm Features

Abbildung 32 Klassendiagramm Features zeigt anhand der zahlreichen Methoden die Features, welche im Rahmen des Projektes entstanden sind. Grundsätzlich ist die Klasse stateless gehalten, um einen möglichst hohen Grad der Entkoppelung von den Player-Klassen zu gewährleisten. Einzige Ausnahme bilden die Statistiken über das Verhalten der Gegner, welche als Dictionaries geführt und im Lauf einer Evaluation laufend aktualisiert werden.

### 7.1.9 HoldemCalc

Die Implementierung ist eine vereinfachte Version des Hold'em Calculators von Kevin Tseng [33]. Im Gegensatz zu der ursprünglichen Verwendungsweise werden in der angepassten Version lediglich die eigenen Hand- sowie die Gemeinschaftskarten verwendet. Die Handkarten der Gegner werden nicht berücksichtigt.

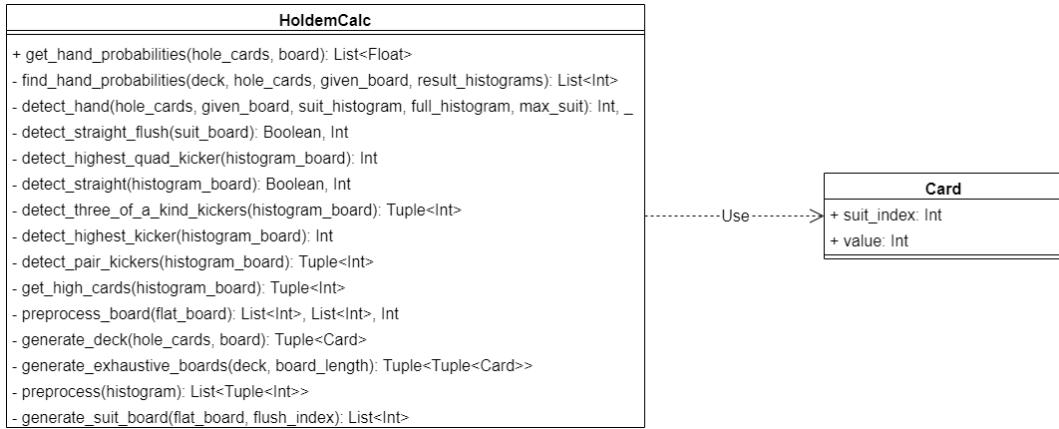


Abbildung 33 Klassendiagramm HoldemCalc

Abbildung 33 Klassendiagramm HoldemCalc zeigt die verwendete Struktur der Klasse. Die Karte Card repräsentiert eine einzelne Karte. Das Ergebnis der Methode `get_hand_probabilities` ist ein Histogramm, welches die Möglichkeiten, eine bestimmte Handstärke zu erhalten, aufzeigt.

```

Player1 Histogram:
High Card : 0.0
Pair : 0.518181818182
Two Pair : 0.384848484848
Three of a Kind : 0.0686868686869
Straight : 0.0
Flush : 0.0
Full House : 0.0272727272727
Four of a Kind : 0.0010101010101
Straight Flush : 0.0
Royal Flush : 0.0
  
```

Abbildung 34 Histogramm für mögliche Handstärken [33]

Abbildung 34 Histogramm für mögliche Handstärken zeigt beispielhaft ein solches Histogramm. Über die Wahrscheinlichkeiten im Histogramm kann die weitere Entwicklung der eigenen Handstärke beurteilt werden.

### 7.1.10 Rewards

Die erhaltenen Belohnungen für getroffene Entscheidungen sind ausschlaggebend für den Lernprozess des neuralen Netzwerkes. Das Implementieren von verschiedenen Methoden, welche jeweils ein Reward-Modell abbilden, ermöglicht sowohl das schnelle Austauschen als auch das Vergleichen der Belohnungsmodelle.

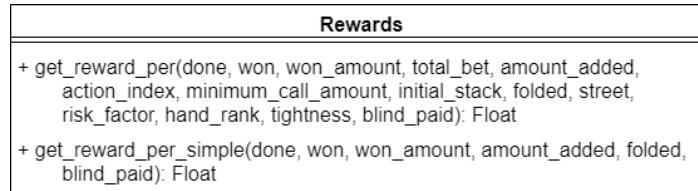


Abbildung 35 Klassendiagramm Rewards

Abbildung 35 Klassendiagramm Rewards zeigt die beiden Methoden, die derzeit im Einsatz sind. Obsolete Methoden sind im Klassendiagramm nicht mehr aufgeführt, in der Klasse jedoch noch enthalten.

### 7.1.11 Plots

Um das Verhalten der Agenten nachvollziehen zu können, sind grafische Auswertungen hilfreich. Die Klasse Plots bietet mehrere Formen dieser Auswertungen an.

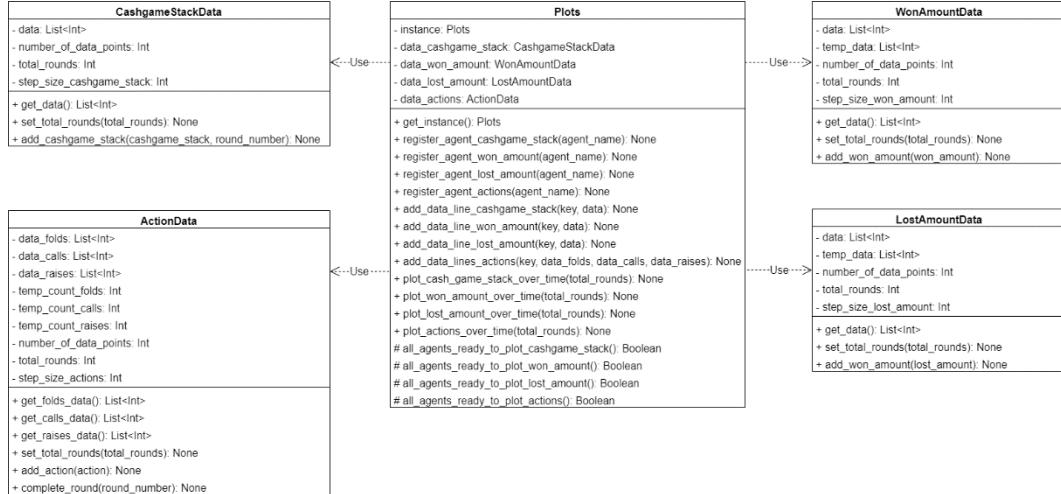


Abbildung 36 Klassendiagramm Plots

Abbildung 36 Klassendiagramm Plots zeigt die Klasse Plots und die verwendeten Data-Klassen.

Die Klasse Plots ist als Singleton implementiert. Dadurch ist gewährleistet, dass nur eine Instanz der Klasse existiert und sich mehrere Agenten registrieren und gemeinsame grafische Auswertungen erstellt werden können.

Über die register-Methoden kann sich ein Agent bei der Klasse anmelden. Die add-Methoden dienen dem Hinzufügen von Datenpunkten und über die plot-Methoden können die jeweiligen Diagramme erstellt werden. Da die Klasse von mehreren Agenten verwendet werden kann, stellen die Methoden all\_agents\_ready\_to\_plot\_[...] sicher, dass sämtliche benötigten Datenpunkte zur Verfügung stehen.

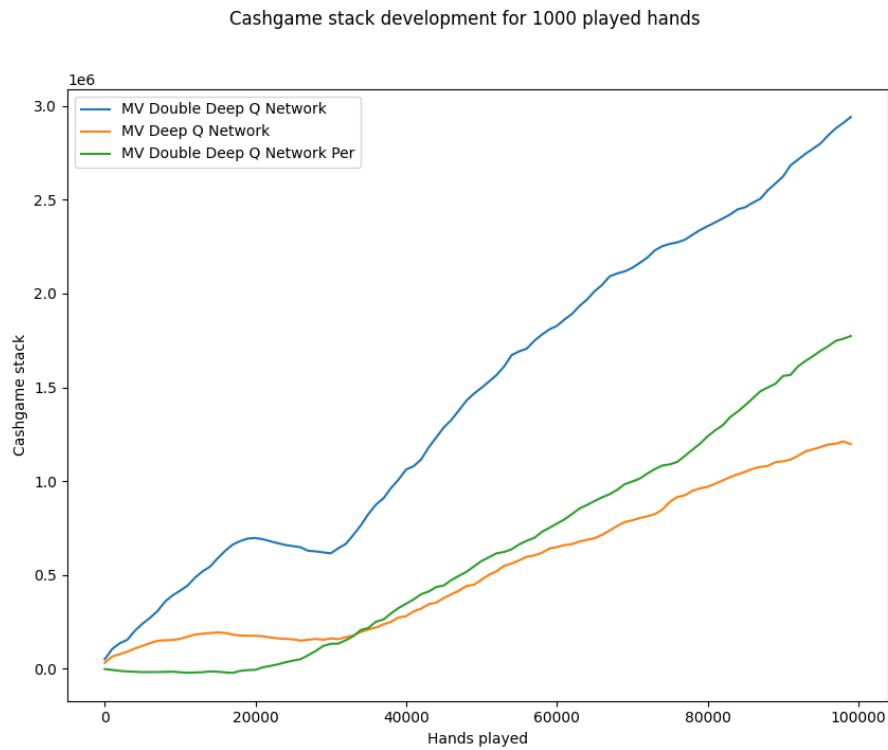


Abbildung 37 Entwicklung Cashgame Stack Plot

Abbildung 37 Entwicklung Cashgame Stack Plot zeigt einen beispielhaften Plot, an dem die Grösse des Cashgame Stacks für jeden teilnehmenden Agenten einer Evaluierung nach jeweils 1'000 gespielten Händen abgelesen werden kann.

### 7.1.12 CashgameStackData, WonAmountData, LostAmountData, ActionData

Die Data-Klassen beinhalten die Datenpunkte für die Plots. Sämtliche Data-Klassen bieten jeweils Methoden an, um Daten hinzuzufügen und die Anzahl zu spielender Runden zu definieren. Über die Attribute werden die anzuzeigenden Daten gespeichert.

### 7.1.13 Fazit

Das implementierte System erlaubt das Erstellen von neuen Agenten in einer Weise, die an einen morphologischen Kasten [42] erinnert. Im Gegensatz zu einem herkömmlichen morphologischen Kasten ist in diesem Fall eine Mehrfachauswahl des Parameters Features möglich.

**Tabelle 18 Morphologischer Kasten**

| Parameter | Ausprägung |         |          |           |     |
|-----------|------------|---------|----------|-----------|-----|
| Agent     | Dqn        | Ddqn    | DdqnPer  | DdqnKeras | ... |
| Features  | Pot-Odds   | Street  | Position | Hand Rank | ... |
| Reward    | Simple     | Complex |          |           |     |

Tabelle 18 Morphologischer Kasten zeigt ein Beispiel, in welchem ein Double Deep Q-Learning mit Prioritized Experience Replay-Agent verwendet wird. Dieser basiert auf den Features Pot-Odds, Position, Hand Rank und berechnet den Reward über das Complex Belohnungssystem.

Dank der implementierten Systemarchitektur kann innerhalb von Minuten ein neuer Agent erstellt werden. Zudem erlauben die Plots und die Statistiken, die per Mail zugesandt werden, eine Aussage über das Verhalten des eigenen Agenten oder gar die Agenten der Gegner.

Die verschiedenen Modelle können trainiert und verglichen werden, um den idealen Kandidaten für eine Einreichung auf AICrowd zu evaluieren.

## Abkürzungsverzeichnis und Glossar

### Agent

Im Kontext der Arbeit ist ein Agent als ein Programm zu verstehen, welcher selbstständig in einer vordefinierten Umgebung Aktionen ausführt., 8, 10, 11, 15, 22, 23, 26, 30, 31, 32, 33, 36, 40, 42, 44, 45, 46, 51, 53, 55, 59, 60, 62, 64, 77, 83, 86, 87, 94, 100, 101

call, 11, 12, 14, 34, 39, 63, 69, 77

Aktion beim Pokern. Callt ein Spieler, bezahlt er den geforderten Einsatz, um in der aktuellen Runde zu bleiben., 81

CAP. *Siehe* Credit Assignment Problem

Credit Assignment Problem

Das Problem der Bestimmung der Handlungen, die zu einem bestimmten Ergebnis führen. *Siehe* Kapitel 2.2.1

Ddqn

Double Deep Q Network Agent, 51, 53, 83, 101

DdqnPer

Double Deep Q Network Agent mit Prioritized Experience Replay, 53, 83, 86, 87, 101

Deep Q Network Agent, 44, 51, 101

Dqn

Deep Q Network Modell, 44

Draw

Chance die eigene Hand durch eine weitere Karte zu verbessern., 16

Exploitation

Verwertung, im Kontext von Epsilon Greedy Strategy die Nutzung der erlernten Erfahrungen, 24

Exploration

Durch Exploration neue Gebiete finden. Im Kontext von Epsilon Greedy die zufällige Auswahl von Aktionen und darauffolgende Beobachtung des Zustands., 24, 55

Feature Engineering

Als Feature Engineering wird der Prozess beschrieben, bei dem Wissen über die Problemdomäne angewendet wird, um beispielsweise Zustände oder allgemein Informationen in Werte darzustellen, die von einem neuronalen Netzwerk als Input-Vektoren verwendet werden können, 37

Flop, 16, 17, 20, 21, 34, 66, 67, 68, 73, 80

fold, 11, 12, 14, 30, 31, 34, 39, 63, 64, 79, 86, 87

Aktion beim Pokern. Folded ein Spieler, gibt er seine Hand auf und scheidet aus der aktuellen Runde aus., 81

Gemeinschaftskarten, 15, 65, 66, 67, 74

Bezeichnet die Karten, die auf dem Tisch liegen und von sämtlichen teilnehmenden Spielern verwendet werden können, um ihr Blatt zu bilden. Werden im englischen auch als Community Cards oder Board Cards bezeichnet., 98

Hole Cards, 15

Englischer Begriff für die beiden Handkarten eines Pokerspielers., 89

Kartendeck

Ein zusammengehöriges Stapel von Karten, aus dem Karten verteilt werden. Im Poker ist das Kartendeck 52 Karten stark., 15

Keras

Ein auf Tensorflow aufbauendes Framework, um Machine Learning Modelle zu entwickeln, 2, 38, 54, 62, 95

## MDP

Ausgeschrieben Markow-Entscheidungsproblem (engl. Markov Decision Process), Zustände sowie Zustandsübergänge sind voneinander abhängig und treten mit Übergangswahrscheinlichkeit auf.. *Siehe auch Kapitel 2.8.1*

## Model-Fitting

Bezieht sich im Bericht jeweils auf das aktualisieren der Q-Table und somit der Aktualisierung der Gewichte im neuronalen Netzwerk mit Hilfe des Keras Optimizers., 40, 42, 43, 44, 50

Bezieht sich im Bericht jeweils auf das Aktualisieren der Q-Table und somit der Aktualisierung der Gewichte im neuronalen Netzwerk mit Hilfe des Keras Optimizers., 31

## Monte-Carlo Simulation, 65, 66

Verfahren aus der Stochastik, bei dem ein Zufallsexperiment oft durchgeführt wird, um analytisch unlösbare Probleme numerisch zu lösen. [32], 65

## Multi-Agenten-System

Multi-Agenten-Systeme seien dabei definiert als multiple, miteinander im Austausch stehende, intelligente Systeme, die sich an die Umgebung anpassen, 11

## Pair-Programming

Agile Softwareentwicklungsmethode, bei der zwei Entwickler an der selben Station arbeiten, um komplexe Aufgaben zu lösen, 92

## POMDP

Partiell observierbares Markow-Entscheidungsproblem, bei dem der Zustand nicht vollständig bekannt ist., 22, 27

## Preflop, 16, 30, 32, 34, 48, 65, 66, 67, 68, 71, 74, 80, 95, 96

## raise, 11, 12, 14, 31, 34, 36, 39, 63, 86

Aktion beim Pokern. Durch einen Raise erhöht ein Spieler den eigenen Einsatz. Andere Spieler müssen mit ihrem Einsatz mindestens gleichziehen oder scheiden aus der aktuellen Runde aus., 81

## Reinforcement Learning

Reinforcement Learning ist ein Bereich des maschinellen Lernens. Es geht darum, geeignete Maßnahmen zu ergreifen, um die Belohnung in einer bestimmten Situation zu maximieren, 21, 22, 24, 26, 34

## River, 17, 20, 21, 34, 66, 67, 73, 80, 87

Vierte Runde beim Pokern. Zu diesem Zeitpunkt sind nebst den Handkarten auch alle fünf Gemeinschaftskarten bekannt., 16

## SARSA

### SARSA

State-Action-Reward-State-Action Policy, wendet Temporal-Difference Learning als on-policy Methode an, um mit einem Zustands-Aktionspaar eine Belohnung zu erreichen. *Siehe auch Kapitel 2.9.1.5*

State-Action-Reward-State-Action Policy, wendet Temporal-Difference Learning als on-policy Methode an, um mit einem Zustands-Aktionspaar eine Belohnung zu erreichen, 27

## Showdown, 24, 34, 67

Letzte Phase einer Pokerrunde. Im Showdown werden, vorausgesetzt es befindet sich noch mehrere Spieler in der Runde, die Handkarten der teilnehmenden Spieler aufgedeckt, um den Gewinner anhand des stärksten Blatts zu ermitteln., 17

**Singleton**

Eine Klassen, von der zu jedem Zeitpunkt maximal eine Instanz existieren darf. [41], 99

Stack, 6, 10, 12, 16, 22, 29, 30, 33, 36, 39, 48, 53, 69, 70, 83, 84, 86, 87, 100

Chips, die einem Spieler zur Verfügung stehen, 30

**stateless**

Das Ergebnis eines Stück Quellcodes ist nicht von vorhergehenden Ereignissen abhängig [40]., 97

**Stochastic game**

Eine Spieltheorie beschrieben durch Lloyd Shapley, bei welchem probalistische Übergänge im Laufe eines Spiels für einen oder mehrere Spieler definiert werden.

Verallgemeinerung des Markov-Entscheidungsprozess, 22

**Stochastic Gradient Descent**

Optimizer aus dem Keras Framework, 60, 61

Street, 34, 36, 58, 67, 73, 101

Aktuelle Runde beim Pokern. Entweder befindet sich das Spiel im Preflop, Flop, Turn, River oder Showdown., 14

**Tensorboard**

Ein Tensorflow Tool, welches Perimeter des verwendeten Netzwerks visualisieren kann, unter anderem die Loss-Entwicklung, 33

**Tensorflow**

Ein von Google entwickeltes Framework für unterschiedliche Programmiersprachen, um Machine Learning Modelle zu entwickeln, 6, 38

Turn, 16, 20, 21, 34, 66, 67, 73

**Unsupervised Learning**

Unsupervised Learning versucht selbstständig, Muster und Beziehungen zwischen den zur Verfügung gestellten Daten zu erkennen und entsprechend dem neuronalen Netzwerk ermöglichen, mit Erfahrungen zu lernen., 24

## Literaturverzeichnis

- [1] Fachhochschule Nordwestschweiz, «fhnw.ch,» [Online]. Available: <https://www.fhnw.ch/de/studium/module/9226191>. [Zugriff am 20 06 2020].
- [2] J. Seemann, «Udemy,» [Online]. Available: <https://www.udemy.com/course/python-bootcamp/>. [Zugriff am 20 06 2020].
- [3] «Coursera,» [Online]. Available: <https://www.coursera.org>. [Zugriff am 10 08 2020].
- [4] A. Ng, «Machine Learning, Week 6 Diagnostics, System Design,» Stanford University, [Online]. Available: <https://www.coursera.org/learn/machine-learning>. [Zugriff am 28 04 2020].
- [5] G. Surma, «Towards data science - Solving Games with AI 🤖 (Part 1: Reinforcement Learning),» 2 Oktober 2018. [Online]. Available: <https://towardsdatascience.com/atari-reinforcement-learning-in-depth-part-1-ddqn-ceaa762a546f>. [Zugriff am 28 Juli 2020].
- [6] A. Team, «AlphaStar: Mastering the Real-Time Strategy Game StarCraft II,» DeepMind, 24 Januar 2019. [Online]. Available: <https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii>. [Zugriff am 28 Juli 2020].
- [7] A. Rupeneite, «Building Poker Agent Using Reinforcement Learning with Neural Networks,» [Online]. Available: <https://www.scitepress.org/Papers/2014/51489/51489.pdf>. [Zugriff am 28 Juli 2020].
- [8] AICrowd, «AICrowd,» [Online]. Available: <https://www.aicrowd.com>. [Zugriff am 10 08 2020].
- [9] A. G. B. Richard S. Sutton, Reinforcement Learning: An Introduction, The MIT Press, Cambridge, 2018.
- [10] R. B. B. D. S. L. Busoniu, «Multi-agent reinforcement learning,» Delft University of Technology, 2010. [Online]. Available: [http://www.dcsc.tudelft.nl/~bdeschutter/pub/rep/10\\_003.pdf](http://www.dcsc.tudelft.nl/~bdeschutter/pub/rep/10_003.pdf). [Zugriff am 20 07 2020].
- [11] ishikota, «Github - PyPokerEngine,» 8 Dezember 2016. [Online]. Available: [https://github.com/ishikota/PyPokerEngine/blob/master/AI\\_CALLBACK\\_FORMAT.md](https://github.com/ishikota/PyPokerEngine/blob/master/AI_CALLBACK_FORMAT.md). [Zugriff am 28 Juni 2020].
- [12] «Dein Guide zu Hand Rankings beim Pokern,» OnlinePoker.de, [Online]. Available: <https://www.onlinepoker.de/hand-rankings.php>. [Zugriff am 3 August 2020].
- [13] «Vorbereitungen - Alles was Sie vor dem Spielstart benötigen und wissen müssen,» Poker.de, [Online]. Available: <https://www.poker.de/guides/poker-regeln/>. [Zugriff am 3 August 2020].
- [14] J. Meinert, «Texas Hold'em - Die Basics,» in *Die Poker Schule*, München, Knaur Taschenbuch Verlag, 2007, pp. 51-100.
- [15] «Poker Playing Styles – Introducing TAG, LAG and Bad!,» Sit and go planet, [Online]. Available: <http://www.sitandgoplanet.com/multitable/poker-playing-styles.html>. [Zugriff am 3 Mai 2020].

- [16] J. Meinert, «Texas Hold'em - Die Einteilung der Spieler,» in *Die Poker Schule*, München, Knaur Taschenbuch Verlag, 2007, pp. 101-114.
- [17] Y. M. A. N. Michael Kearns, «A Sparse Sampling Algoirthm for Near-Optimal Planning in Large Markov Decision Process,» Kluwer Academic Publishers, 2002. [Online]. Available: <https://link.springer.com/content/pdf/10.1023/A:1017932429737.pdf>. [Zugriff am 10 04 2020].
- [18] A. W. S, «Medium - Supervised vs. Unsupervised learning,» 18 Januar 2019. [Online]. Available: <https://medium.com/@aakankshaws/supervised-vs-unsupervised-learning-ba6e62cda4bf>. [Zugriff am 31 Mai 2020].
- [19] J. MacGlashan, «What is the difference between model-based and model-free reinforcement learning,» 29 04 2018. [Online]. Available: <https://www.quora.com/What-is-the-difference-between-model-based-and-model-free-reinforcement-learning>. [Zugriff am 28 07 2020].
- [20] F. S. Melo, «Convergence of Q-Learning: A simple proof,» [Online]. Available: <http://users.isr.ist.utl.pt/~mtjspa/readingGroup/ProofQlearning.pdf>. [Zugriff am 10 04 2020].
- [21] «Keras,» [Online]. Available: <https://keras.io>. [Zugriff am 30 03 2020].
- [22] A. G. D. S. Hado van Hasselt, «Deep Reinforcement Learning with Double Q-learning,» 8 12 2015. [Online]. Available: <https://arxiv.org/pdf/1509.06461.pdf>. [Zugriff am 10 05 2020].
- [23] A. Ng, «Mini-batch gradient descent,» deeplearning.ai, [Online]. Available: <https://www.coursera.org/lecture/deep-neural-network/mini-batch-gradient-descent-qcogH>. [Zugriff am 10 05 2020].
- [24] Google DeepMind, «Prioritized Experience Replay,» ICLR 2016, 2016.
- [25] A. Gad, «Beginners Ask “How Many Hidden Layers/Neurons to Use in Artificial Neural Networks?”,» 27 06 2018. [Online]. Available: <https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e>. [Zugriff am 20 07 2020].
- [26] M. Cerliani, «Feature Importance with Neural Network,» towards data science, 29 06 2019. [Online]. Available: <https://towardsdatascience.com/feature-importance-with-neural-network-346eb6205743>. [Zugriff am 21 06 2020].
- [27] G. Seif, «Understanding the 3 most common loss functions for machine learning,» towards data science, 19 05 2019. [Online]. Available: <https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3>. [Zugriff am 21 06 2020].
- [28] B. Z. V. V. Q. V. L. I. Bello, «arxiv,» 22 11 2017. [Online]. Available: <https://arxiv.org/pdf/1709.07417.pdf>. [Zugriff am 23 06 2020].
- [29] S. Ruder, «arxiv,» Insight Centre for Data Analytics, 15 06 2017. [Online]. Available: <https://arxiv.org/pdf/1609.04747.pdf>. [Zugriff am 23 06 2020].
- [30] A. Ng, «Why does Batch Norm work?,» deeplearning.ai, [Online]. Available: <https://www.coursera.org/learn/deep-neural-network/lecture/81oTm/why-does-batch-norm-work>. [Zugriff am 23 06 2020].
- [31] A. Ng, «Normalizing activations in a network,» deeplearning.ai, [Online]. Available: <https://www.coursera.org/learn/deep-neural-network/lecture/81oTm/why-does-batch-norm-work>. [Zugriff am 23 06 2020].

- network/lecture/4ptp2/normalizing-activations-in-a-network. [Zugriff am 23 06 2020].
- [32] «Mathepedia - Monte-Carlo-Methode,» [Online]. Available: <https://mathepedia.de/Monte-Carlo-Methode.html>. [Zugriff am 28 Juli 2020].
  - [33] K. Tseng, «Github - Holdem Calculator,» 15 Februar 2018. [Online]. Available: [https://github.com/ktseng/holdem\\_calc](https://github.com/ktseng/holdem_calc). [Zugriff am 27 Juni 2020].
  - [34] M. Shackleford, «Six-Player Power Ratings in Texas Hold 'Em,» Wizard of Odds Consulting, Inc., 30 September 2019. [Online]. Available: <https://wizardofodds.com/games/texas-hold-em/6-player-game/>. [Zugriff am 23 Juni 2020].
  - [35] I. Hendley, «Git - treys,» 13 August 2019. [Online]. Available: <https://github.com/ihendley/treys>. [Zugriff am 23 Juni 2020].
  - [36] «Weisheiten, Phrasen/Sprüche und Witze im Poker,» Sit and Go Poker Strategie, [Online]. Available: <https://www.sngpokerstrategie.com/poker-strategie/weitere-poker-artikel/weisheiten-phrasen-und-spruche-im-poker/>. [Zugriff am 23 Juni 2020].
  - [37] «Cactus Kev's Poker Hand Evaluator,» [Online]. Available: <http://suffe.cool/poker/evaluator.html>. [Zugriff am 23 Juni 2020].
  - [38] OpenAi, «OpenAI Gym,» [Online]. Available: <https://gym.openai.com>. [Zugriff am 01 06 2020].
  - [39] M. Hessel, J. Modayil, H. von Hasselt, T. Schaul und G. Ostrovski, «Rainbow: Combining Improvements in Deep Reinforcement Learning».
  - [40] «Stateless versus stateful,» Packt, [Online]. Available: [https://subscription.packtpub.com/book/application\\_development/9781788831437/1/ch01lvl1sec16/stateless-versus-stateful](https://subscription.packtpub.com/book/application_development/9781788831437/1/ch01lvl1sec16/stateless-versus-stateful). [Zugriff am 14 Juli 2020].
  - [41] «Wikipedia,» Singleton pattern, [Online]. Available: [https://en.wikipedia.org/wiki/Singleton\\_pattern](https://en.wikipedia.org/wiki/Singleton_pattern). [Zugriff am 14 Juli 2020].
  - [42] A. Windolph, «Projekte leicht gemacht - Der Morphologische Kasten,» [Online]. Available: <https://projekte-leicht-gemacht.de/blog/pm-methoden-erklaert/morphologischer-kasten/>. [Zugriff am 19 Juli 2020].
  - [43] «Draw.io,» [Online]. Available: <https://app.diagrams.net>. [Zugriff am 10 08 2020].

## Abbildungsverzeichnis

|                                                                                                      |     |
|------------------------------------------------------------------------------------------------------|-----|
| Abbildung 1 Klassendiagramm BasePokerPlayer .....                                                    | 13  |
| Abbildung 2 Poker Handstärken [12] .....                                                             | 15  |
| Abbildung 7 Positionen bei zwei Spielern .....                                                       | 18  |
| Abbildung 7 Positionen bei drei Spielern .....                                                       | 18  |
| Abbildung 7 Positionen bei fünf Spielern .....                                                       | 18  |
| Abbildung 7 Positionen bei sechs Spielern .....                                                      | 18  |
| Abbildung 7 Positionen bei vier Spielern .....                                                       | 18  |
| Abbildung 8 Die vier Verhaltenskategorien von Pokerspielern [15].....                                | 19  |
| Abbildung 9 Anwendung Epsilon Greedy Algorithmus.....                                                | 25  |
| Abbildung 10 Vergleich SARSA und Q-Learning am Beispiel [9, p. 132] .....                            | 27  |
| Abbildung 11 Vereinfachte Form des neuronalen Netzwerks .....                                        | 39  |
| Abbildung 12 Ablaufdiagramm [MVDqn.py].....                                                          | 41  |
| Abbildung 13 Konvergenz DdqnAgent zu Tight-Aggressive Spieler.....                                   | 46  |
| Abbildung 14 Ablaufdiagramm [MVDDqnPer] .....                                                        | 51  |
| Abbildung 15 Ablauf Pokerrunde [MVBasePokerPlayer] .....                                             | 52  |
| Abbildung 16 Entwicklung Cashgame Stack Vergleich Ddqn versus DdqnPer .....                          | 53  |
| Abbildung 17 Hyperparameter Anpassung der Einstellungen .....                                        | 56  |
| Abbildung 18 Benchmark Feature Importance Tests.....                                                 | 57  |
| Abbildung 19 Loss Funktion nach Feature Swap Anzahl Gegner und Street .....                          | 58  |
| Abbildung 20 Auszug aus AICrowd Statistiken mit beinahe identischer Aggressivität aller Gegner ..... | 58  |
| Abbildung 21 Loss Funktionen: MAE (rot), Huber (grün), MSE (blau) [27].....                          | 60  |
| Abbildung 22 AICrowd Resultate Hyperparameter Optimierung vom 28.06 - 19.07 .64                      | 64  |
| Abbildung 23 Hand Rankings Preflop [34] .....                                                        | 66  |
| Abbildung 24 Dqn vs Ddqn vs DdqnPer .....                                                            | 82  |
| Abbildung 25 Vergleich DdqnPer und Ddqn Agenten .....                                                | 83  |
| Abbildung 26 Cashgame Stack Entwicklung über eine Million Runden.....                                | 84  |
| Abbildung 27 Summe der Aktionen DdqnPerPlayer_2 für jeweils 1000 Hände .....                         | 85  |
| Abbildung 28 Klassendiagramm Agent Package .....                                                     | 94  |
| Abbildung 29 Klassendiagramm DdqnPerSplitModel.....                                                  | 95  |
| Abbildung 30 Klassendiagramm MemoryBuffer .....                                                      | 96  |
| Abbildung 31 Klassendiagramm Players .....                                                           | 96  |
| Abbildung 32 Klassendiagramm Features .....                                                          | 97  |
| Abbildung 33 Klassendiagramm HoldemCalc.....                                                         | 98  |
| Abbildung 34 Histogramm für mögliche Handstärken [33].....                                           | 98  |
| Abbildung 35 Klassendiagramm Rewards .....                                                           | 99  |
| Abbildung 36 Klassendiagramm Plots.....                                                              | 99  |
| Abbildung 37 Entwicklung Cashgame Stack Plot .....                                                   | 100 |

## Tabellenverzeichnis

|                                                                            |     |
|----------------------------------------------------------------------------|-----|
| Tabelle 1 RE1: Q-Table für Q-Learning Modell.....                          | 34  |
| Tabelle 2 Konfiguration [/source/agent/MVDqn.py] .....                     | 39  |
| Tabelle 3 Evaluation 10.07.2020 aus [Beilagen/AICrowd_Resultate].....      | 63  |
| Tabelle 4 Evaluation 12.07.2020 aus [Beilagen/AICrowd_Resultate].....      | 64  |
| Tabelle 5 Street One-hot .....                                             | 67  |
| Tabelle 6 Position One-hot.....                                            | 67  |
| Tabelle 7 Repräsentation Anzahl Gegner als kategorischer Wert .....        | 68  |
| Tabelle 8 Pot-Odds kategorisch .....                                       | 69  |
| Tabelle 9 Aggressivität Kategorien Double Deep Q Network .....             | 70  |
| Tabelle 10 Tightness Kategorien Double Deep Q Network .....                | 71  |
| Tabelle 11 Risiko Kategorien Double Deep Q Network .....                   | 72  |
| Tabelle 12 Zuweisung der Features zu Realisierungseinheiten .....          | 73  |
| Tabelle 13 Konfiguration Agenten in oberer Abbildung.....                  | 83  |
| Tabelle 14 Konfiguration der Player Klassen für den Modell Vergleich.....  | 84  |
| Tabelle 15 AICrowd Evaluationen .....                                      | 86  |
| Tabelle 16 Beziehung zwischen Player-Klassen und Q-Learning Modellen ..... | 94  |
| Tabelle 17 Beziehung zwischen Modell-Klassen und Q-Learning Modellen.....  | 95  |
| Tabelle 18 Morphologischer Kasten .....                                    | 101 |

## Anhang

### Grobplanung

|                        |            | Grobplanung Pokerbot Competition |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|------------------------|------------|----------------------------------|----------------|----------|------------|----------|-----------------|----------|-----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| Startdatum             |            | 17.02.20                         | 24.02.20       | 01.03.20 | 08.03.20   | 15.03.20 | 22.03.20        | 05.04.20 | 12.04.20  | 19.04.20 | 26.04.20 | 03.05.20 | 10.05.20 | 17.05.20 | 24.05.20 | 31.05.20 | 07.06.20 | 14.06.20 | 21.06.20 | 28.06.20 | 05.07.20 | 12.07.20 | 19.07.20 | 26.07.20 | 03.08.20 | 10.08.20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Enddatum               |            | 23.02.20                         | 01.03.20       | 08.03.20 | 15.03.20   | 22.03.20 | 29.03.20        | 05.04.20 | 12.04.20  | 19.04.20 | 26.04.20 | 03.05.20 | 10.05.20 | 17.05.20 | 24.05.20 | 31.05.20 | 07.06.20 | 14.06.20 | 21.06.20 | 28.06.20 | 05.07.20 | 12.07.20 | 19.07.20 | 26.07.20 | 03.08.20 | 10.08.20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Projektwoche           | 1          | 2                                | 3              | 4        | 5          | 6        | 7               | 8        | 9         | 10       | 11       | 12       | 13       | 14       | 15       | 16       | 17       | 18       | 19       | 20       | 21       | 22       | 23       | 24       | 25       | 26       |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <b>Meilensteine</b>    |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          | <b>A</b> |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <b>AI Crowd Upload</b> | RE1        |                                  | RE2            |          | RE3        |          | RE4             |          | RE5       |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <b>Risiken</b>         |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          | <b>A</b> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Machine Learning       |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Python                 |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <b>Produkt</b>         |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| MVP                    | Q-Learning |                                  | DeepQ Networks |          | DQN / DDQN |          | DDQN / DDQN PER |          | HP Tuning |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Major-Version 1        |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Major-Version 2        |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Major-Version 3        |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Major-Version 4        |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <b>Dokumentation</b>   |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Zwischenversion        |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Finale Version         |            |                                  |                |          |            |          |                 |          |           |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Die Grobplanung ist im Projektordner unter [Beilagen/Grobplanung\_Controlling.xlsx] im Arbeitsblatt Grobplanung zu finden.

### Klassendiagramm

Das zusammengefügte Klassendiagramm ist unter [Beilagen/Klassendiagramm\_Komplett] als PNG und draw.io [43] Datei beigelegt.