ZURICH UNIVERSITY OF APPLIED SCIENCES

MASTER THESIS

# Reinforcement Learning for Complex 3D Game Playing

*Author:*
Gabriel EYYI

*Supervisor:*
Dr. Thilo STADELMANN,
Melanie Geiger

*A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Engineering*

*in the*

Information Engineering Group
Institute of Applied Information Technology (InIT)

3. April 2018

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

## Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

| Ort, Datum: | Unterschriften: |
| --- | --- |
| …… | …… |
| | …… |
| | …… |

*„Good and evil, reward and punishment, are the only motives to a rational creature: these are the spur and reins whereby all mankind are set on work, and guided."*

John Locke

# *Zusammenfassung*

Die Fortschritte im Bereich von Deep Reinforcement Learning (DRL) ermöglichen es, autonome Agenten für komplexe 3D-Umgebungen zu entwickeln. Die grosse Herausforderung dieser mächtigen DRL Algorithmen besteht allerdings im erfolgreichen Trainieren dieser Agenten. Lange Rechen- und Trainingszeiten sowie anfällige Parametereinstellungen behindern das Anwenden dieser Algorithmen auf neue Problemstellungen. Bestehende Lösungen setzen auf massenhafte Rechenleistung, aufwändige Trainings- oder Transfer Learning Methoden.

Daher beschäftigt sich die vorliegende Masterarbeit mit den generellen Herausforderungen des Anwendens solcher DRL Algorithmen für komplexe 3D-Umgebungen im Beispiel des First-Person-Shooter (FPS) Games Doom. Das Ziel dieser Arbeit ist es, die Trainingszeit der Algorithmen mittels der Imitation des menschlichen Verhaltens bei der Entscheidungsfindung oder der Schaffung einer optimalen Trainingsbedingung zu verringern.

Unter der Verwendung des Fine Grained Action Repetition (FiGAR) Frameworks und der Self-Play Trainingsmethode wurden zwei Ansätze entwickelt, implementiert und experimentell untersucht. Das FiGAR-Framework erweitert DRL Algorithmen, indem zusätzlich die Wiederholungsrate einer Aktion vorhergesagt wird. Dadurch kann eine zeitliche Abstraktion im Aktionsraum geschaffen werden. Die Self-Play Trainingsmethode stellt sicher, dass in einer kompetitiven Umgebung der Gegenspieler auf dem gleichen Level ist und somit ideale Trainingsbedingungen geschaffen werden können.

Die Ergebnisse dieser Experimente deuten darauf hin, dass das Hauptproblem beider Ansätze die Exploration der Umgebung ist und somit suboptimale Verhaltensstrategien gelernt werden. Die untersuchten Ansätze konnten somit die Trainingszeit nicht direkt verringern, liefern jedoch wertvolle Hinweise für weitere Forschungsarbeiten.

# *Abstract*

Advances in the field of deep reinforcement learning (DRL) made the development of autonomous agents for complex 3D environments possible. However, a big challenge of this powerful algorithms consists of the successful training of these agents. Long computation and training time, as well as sensitive parameter settings, limits the usage of these algorithms in other problem domains. Existing solutions use enormous computation power and time-consuming training or transfer learning methods.

In this work, general challenges of DRL algorithms are investigated for complex 3D environments in the First-Person-Shooter (FPS) Doom. The aim of this work is to reduce the training time by imitating human behavior in decision making or by creating optimal training conditions.

By using the Fine Grained Action Repetition (FiGAR) Framework and Self-Play training method two approaches are developed, implemented and examined. The FiGAR framework extends existing DRL algorithm by predicting the time scale of an action and therefore enabling temporal abstraction in the action space. The self-play training methods create optimal training conditions for competitive environments.

The results of these experiments indicate, that the major problem of both approaches is the exploration of the environment and therefore the agent learns suboptimal action policies. The examined approaches could not reduce the training time directly, nevertheless they provide valuable information for further research.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Einleitung

## 1.1 Motivation

Das Lernverhalten von Menschen ist geprägt von drei wichtigen Aktionen: beobachten, ausprobieren und anschliessendem analysieren. Die Interaktionen mit der Umgebung und die nachfolgende Analyse der Konsequenzen helfen uns, unser Wissen stetig zu verbessern. Diese Eigenschaften sind bei uns Menschen tief verankert. Sie ermöglichen uns optimale Verhaltensstrategien zu lernen sowie gelerntes Wissen auf neue Umgebungen zu adaptieren. Das Entwickeln eines Systems, welches über diese Eigenschaften verfügt ist von grossem Nutzen und einer der treibenden Kräfte hinter Reinforcement Learning. Reinforcement Learning (RL) zählt neben Supervised Learning und Unsupervised Learning zum dritten Forschungsfeld von Maschine Learning und befasst sich mit der Frage, wie eine wirksame oder sogar optimale Verhaltensstrategie in einer Umgebung gelernt werden kann. Inspiriert durch die Verhaltensforschung und Psychologie wird in RL anhand der Interaktion mit der Umgebung das optimale Verhalten gelernt und anschliessend mittels geschicktem Probieren (Trial-and-Error-Methode) verbessert. RL ist eine allgemeine Herangehensweise, um Optimierungs- und Entscheidungsprobleme lösen zu können.

Die in jüngster Zeit erzielten Resultate im Bereich von Reinforcement Learning haben das grosse Potenzial dieses Frameworks verdeutlicht. In Kombination mit Deep Learning Technologien konnten zahlreiche komplexe Optimierungs- und Entscheidungsprobleme gelernt beziehungsweise gemeistert werden. Klassische RL Techniken für die Approximation von Funktionen in hochdimensionalen Zustands- und Aktionsräumen leiden unter anderem an Stabilitätsproblemen und verhindern den Einsatz von RL Techniken in komplexeren Umgebungen. Dafür bieten sich Deep Learning Verfahren als vielversprechende Lösung an. In der Arbeit von Mnih et al. (2015) konnte in überzeugender Weise dargestellt werden, dass mithilfe von Deep Learning Verfahren und nur anhand des visuellen Inputs (Frame als Pixelmatrix), ein RL Algorithmus trainiert werden konnte, um Atari 2600 Games zu spielen. Einen weiteren Durchbruch erzielten die Entwickler von AlphaGo, als der Weltmeister in Go geschlagen werden konnte (Silver et al., 2016). Go galt lange Zeit, aufgrund des enormen Suchraumes und der Schwierigkeit Brettpositionen und Züge zu evaluieren, als einer der grössten Herausforderungen im Bereich der Künstlichen Intelligenz (KI) für Spiele (Games).

In rasanter Geschwindigkeit werden in diesem Forschungsbereich neue Methoden entwickelt sowie bestehende Methoden erweitert und verbessert. Diese Arbeit ist motiviert durch die Begeisterung von Verfahren im Bereich des maschinellen Lernens und durch die Möglichkeit, State of the Art Algorithmen für Probleme in komplexen Umgebungen genauer untersuchen zu können.

## 1.2 Problemstellung

Entscheidungs- oder Optimierungsprobleme in der realen Welt sind oft viel komplexer und beinhalten unterschiedliche Voraussetzungen. Im Vergleich zu den 2D-Games stehen dem RL Algorithmus nicht alle Informationen der Umgebung zur Verfügung. Der RL Algorithmus kann beispielsweise durch die aktuellen Sensordaten oder Kamerabilder der aktuellen Position eingeschränkt sein. Häufig sind diese Umgebungen zusätzlich in 3D. Das Hinzufügen einer zusätzlichen räumlichen Dimension stellt für RL Algorithmen eine weitere Herausforderung dar. Neben der eingeführten unvollständigen Beobachtung des Zustandes erschweren Abweichungen der Beobachtungspunkte das Lernverhalten. Gleiche Objekte können aus verschiedenen Perspektiven beobachtet werden und müssen vom RL Algorithmus erkannt werden. Aus diesem Grund muss der RL Algorithmus frühere Beobachtungen miteinbeziehen, um eine optimale Entscheidung treffen zu können.

Im Allgemeinen stellt das Trainieren eines RL Algorithmus in einer solchen Umgebung aufgrund des grossen Suchraums, des grossen Zustandsraums und der kleinen Wahrscheinlichkeit eine Rückmeldung zu erhalten, eine grosse Herausforderung dar (Bhatti et al., 2016).

Die Entwicklung und Untersuchung von neuen Algorithmen in komplexen Umgebungen sind kostspielig und aufwändig, da die RL Algorithmen viele Trainingsbeispiele benötigen, bis eine optimale Verhaltensstrategie gelernt werden kann. Aus diesem Grund werden oft Games als Testumgebungen genutzt. Insbesondere Videospiele erlauben es, komplexe 3D-Umgebungen kostengünstig zu simulieren.

In der vorliegenden Arbeit soll die Funktionsfähigkeit von Reinforcement Learning in komplexen 3D-Umgebungen genauer untersucht werden. Obwohl schon erfolgreich RL Algorithmen in solchen Umgebungen angewendet wurden, beispielsweise (Hausknecht et al., 2015), besteht das Ziel der vorliegenden Arbeit darin, vorhandene Schwierigkeiten und Engpässe von RL in solchen Umgebungen zu ermitteln und zu beheben.

### 1.2.1 Hintergrund

Eine ideale Testumgebung für komplexe 3D-Umgebungen stellt die Plattform ViZDoom (Kempka et al., 2016) dar. Die Plattform ViZDoom wurde vom Institut für Computer Science an der Poznan Univerty entwickelt und bietet ein Interface für RL Algorithmen (Agenten) an, um direkt den visuellen Input (Screen als Pixelmatrix) des Games Doom zuzugreifen. Prinzipiell ist die Umgebung in Doom nicht komplett in 3D. Objekte beziehungsweise Gegenstände werden im Game als 2D-Sprites (Grafikobjekte) dargestellt. Aus diesem Grund ist nur die Navigation in Doom in 3D und die Objekterkennung beziehungsweise Objektidentifizierung in 2D.

Die Entwickler von ViZDoom organisieren seit dem Jahr 2016 zudem einen jährlichen Wettbewerb, um die Performance von RL Algorithmen zu messen. Der Wettbewerb besteht aus zwei Deathmatch Aufgaben. Ein Deathmatch ist in diesem Kontext ein zeitlich limitierter Spielmodus, bei dem jeder Agent möglichst viele andere Agenten eliminieren muss.

## 1.3 Analyse der bestehenden Lösungen

Eine Vielzahl von unterschiedlichen Reinforcement Learning Agenten wurden entwickelt und getestet. Eine genaue Analyse der Ergebnisse beziehungsweise der Performance der Agenten ist für die Untersuchung von möglichen Engpässen und für die Entwicklung eines Agenten von grossem Nutzen. Was machen die bestehenden Agenten gut und was nicht? Wo sind die Schwächen der Agenten? Verfolgen die Agenten Strategien? Reagieren die Agenten im Vergleich zu einem Menschen sinnvoll?

Um diese Fragen beantworten zu können und um schliesslich eine mögliche Verbesserung zu erzielen, werden zuerst die bestehenden Agenten analysiert. Die Organisatoren stellen die Statistiken[1][2] und Videos[3][4] des Wettbewerbs von 2016 und 2017 zur Verfügung. Mithilfe der Analyse sollen sowohl die Stärken und Schwächen als auch die gelernten Strategien der Agenten ermittelt werden. Die Analyse erfolgt anhand einer qualitativen Untersuchung der Videos.

### 1.3.1 Qualitative Analyse: Videoanalyse der Agenten von 2016 und 2017

Die qualitative Analyse besteht aus der Analyse der Videos der besten drei Agenten des Wettbewerbs von 2016 und 2017 (Track1). Dabei werden jeweils die beste und die schlechteste Runde manuell untersucht. Das Ziel der qualitativen Analyse ist es, das Verhalten der Agenten in bestimmten Situationen zu beobachten und zu bewerten. Die Analyse soll einen ersten Eindruck über die Stärken und Schwächen der Agenten liefern. Für eine einheitliche und objektive Untersuchung werden zuerst Kriterien mithilfe von Beiträgen von Profispieler[5][6][7] zusammengestellt. Anschliessend werden die Videos anhand einer Ordinalskala bewertet. Die Ordinalskala setzt sich aus den folgenden drei Werten zusammen:

- trifft sehr zu (3) < trifft zu (2) < trifft eher nicht zu (1)

**Untersuchte Agenten:**

- **ViZDoom 2016:** $\{F1, Arnold, Clyde\}$
- **ViZDoom 2017:** $\{Marvin, Arnold2, Axon\}$

[1] http://vizdoom.cs.put.edu.pl/competition-cig-2016/results
[2] http://vizdoom.cs.put.edu.pl/competition-cig-2017/results
[3] https://www.youtube.com/watch?v=3VU6d_5ze8k
[4] https://www.youtube.com/watch?v=94EPSjQH38Y
[5] https://taskandpurpose.com/8-tips-first-person-shooters-pro-gamer/
[6] https://www.lifewire.com/be-the-best-video-game-shooter-3396182
[7] http://www.fpstactics.n.nu/

**Kriterien:**

Die Kriterien können in drei Kategorien aufgeteilt werden.

- **Navigation:** Der Agent bewegt sich in der Umgebung flüssig und läuft nicht in Hindernisse (Wände, Ecken, Gegenstände, etc.). Der Agent ist in der Lage, gezielt einen Gegenstand einzusammeln.
- **Angriff:** Der Agent erkennt Gegenspieler und kann angreifen. Der Agent ist in der Lage, Gegenspieler zu verfolgen und zu eliminieren.
- **Strategie:** Der Agent verfügt über einen Plan oder eine Strategie. Der Agent kann auf bestimmte Spielsituationen reagieren.

**Resultate Videoanalyse:**

Die Resultate der Videoanalyse sind in der Tabelle 1.1 dargestellt. In der linken Hälfte der Tabelle sind die Resultate der Agenten des Wettbewerbs von 2016 und in der rechten Hälfte die Resultate der Agenten von 2017 abgebildet.

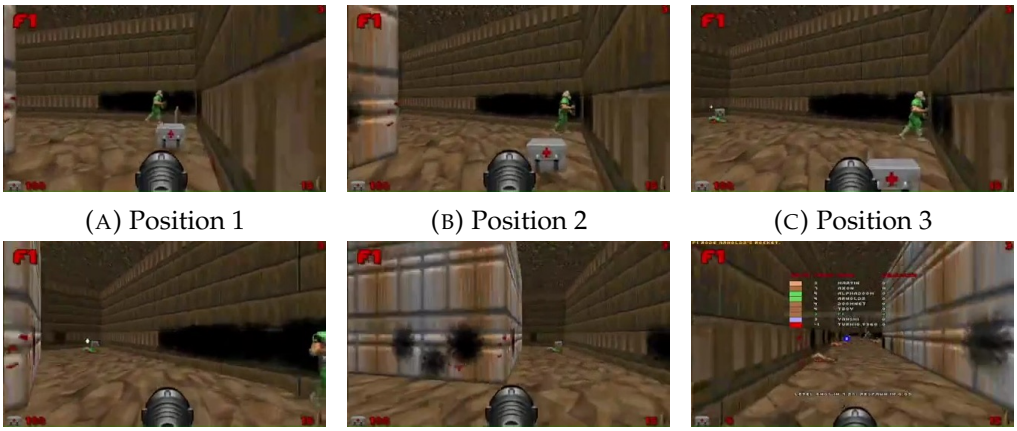
| **Kriterium** | **F1** | **Clyde** | **Arnold** | **Marvin** | **Arnold2** | **Axon** |
|---|---|---|---|---|---|---|
| ***Navigation*** | | | | | | |
| Bewegung | 3 | 2 | 3 | 1 | 3 | 1 |
| Hindernis | 3 | 2 | 2 | 2 | 2 | 2 |
| Items | 1 | 1 | 1 | 2 | 2 | 2 |
| ***Angriff*** | | | | | | |
| Treffsicherheit | 2 | 2 | 1 | 2 | 2 | 2 |
| Zielverfolgung | 2 | 1 | 1 | 2 | 3 | 2 |
| ***Strategie*** | | | | | | |
| Plan | 2 | 2 | 2 | 3 | 2 | 1 |
| Reaktion | 1 | 2 | 2 | 3 | 2 | 2 |

TABELLE 1.1: Resultat der Videoanalyse. Pro Kriterium wurden (1-3) Punkte vergeben.

- **Navigation:** Die Navigation ist bei allen sechs Agenten sehr ausgereift. Die Agenten bewegen sich flüssig und können mit Hindernissen (Wände, Sackgassen, Abbiegungen, etc.) umgehen. Allerdings wurde bei den analysierten Videos festgestellt, dass die Agenten nahezu nie bewusst Items (Health, Ammo) gesammelt haben.
- **Angriff:** Im Angriff gibt es grössere Unterschiede. Von einer unnatürlichen, aber effizienten und schnellen Hin-und-her-Bewegung bis hin zur direkten Zielverfolgung sind bei diesen sechs Agenten alle Verhaltensweisen vorhanden.
- **Strategie:** Die grössten Unterschiede sind bei der Bildung einer geeigneten Strategie ersichtlich. Abgesehen von den Agenten F1 und Marvin waren keine Strategien erkennbar.

**Videoanalyse Beispiele Szene**

In der Abbildung 1.1 ist eine Beispielszene dargestellt. In dieser Szene verfolgt der Agent (F1) die Strategie, im Korridor hin und her zu laufen, um ankommende Gegenspieler zu eliminieren. Allerdings erkennt der Agent den grünen Gegenspieler nicht und wird anschliessend eliminiert.



(A) Position 1 (B) Position 2 (C) Position 3

(D) Position 4 (E) Position 5 (F) Position 6

ABBILDUNG 1.1: Beispielsszene der Videoanalyse. F1 Bots erkennt einen Gegenspieler (grün dargestellt) nicht und wird eliminiert.

### 1.3.2 Zusammenfassung der ersten Analyse

Die Resultate der qualitativen Videoanalyse geben nur einen kleinen Einblick in die Stärken und Schwächen der Agenten und dürfen nicht zu weit interpretiert werden. Die Information, wie genau ein Agent einen bestimmten Zustand wirklich bewertet hat, ist nicht verfügbar. Zudem muss berücksichtigt werden, dass gewisse Agenten (F1) Post-Training-Rules verwenden, um ihr Modell für den Wettbewerb zu verbessern. Aufgrund dieser Tatsache sind die Resultate über die Stärken und Schwächen der trainierten Agenten mittels reiner Videoanalyse verfälscht. Nichtsdestotrotz liefern die Videos einen ersten Eindruck über die Performance der Agenten.

## 1.4 Weiteres Vorgehen

Die Resultate der Analyse haben gezeigt, dass eine ausgeprägte Navigation oder eine geeignete Strategie zum Erfolg führen kann und dass die Treffsicherheit und Zielverfolgung in Deathmatch Spiel kleinere Rollen spielen. Erfolgreiche RL Agenten für ViZDoom nutzen daher aufwändige Trainingsmethoden oder komplexe Architekturen, um ihren RL Algorithmus zu trainieren (vgl. Abschnitt 2.8).

Das erfolgreiche Trainieren von RL Agenten kann somit als Herausforderung betrachtet werden. In der vorliegenden Arbeit sollen daher unterschiedliche Ansätze untersucht werden, um die Trainingszeit (beziehungsweise Lernzeit der RL Algorithmen) zu verringern.

### 1.4.1 Gliederung

Die vorliegende Arbeit gliedert sich in die folgende Kapitel:

- Das Kapitel 2 beinhaltet eine Einführung und die Grundlagen von Reinforcement Learning. Zudem werden in diesem Kapitel die zwei Algorithmen vorgestellt und die Herausforderungen sowie die Trainingsstrategien für Games besprochen.
- Das Kapitel 3 beinhaltet die Problemanalyse und die Beschreibung von zwei Lösungsansätzen.
- Das Kapitel 4 beinhaltet eine Beschreibung des experimentellen Setups sowie der Umsetzung.
- Das Kapitel 5 beinhaltet die Resultate und Diskussionen der durchgeführten Experimente.
- Das Kapitel 6 beinhaltet das Fazit und einen Ausblick für mögliche weiterführende Forschung.

# Kapitel 2

# Grundlagen

Reinforcement Learning beschreibt ein Lernverfahren, um eine Funktion zu finden, die jeden möglichen Zustand in einer Umgebung einer treffenden Aktion zuordnet, sodass der kumulierte Reward des Agenten langfristig maximiert wird (Sutton et al., 1998). Die grosse Herausforderung bei Reinforcement Learning besteht darin, dass die Dynamik in dieser Umgebung dem Agenten nicht zur Verfügung steht. Was dies bedeutet und wie solche Problemstellungen gelöst werden können, wird in diesem Kapitel vorgestellt.

## 2.1 Reinforcement Learning

RL bezeichnet in erster Linie eine allgemeine Klasse von Algorithmen im Gebiet von Machine Learning, um sequentielle Entscheidungsprobleme durch limitierte Rückmeldungen zu lösen. Dabei werden verschiedene biologische und technische Konzepte in einem kompakten Framework zusammengefasst. Das Ziel von RL ist es, einem Agenten (dies kann beispielsweise ein Bot in einem Game, ein Roboter oder ein anderes autonomes System sein) das Verhalten in einer Umgebung anhand evaluativen Rückmeldungen zu lernen. Im Allgemeinen bezeichnet Reinforcement Learning mehr ein Learning-Problem oder -Paradigma als eine Klasse von Learning Methoden. Neben Anwendungen im Bereich der Künstlichen Intelligenz und der Computerwissenschaften bietet RL ein wertvolles konzeptionelles Framework im Bereich der Kognitions- und Verhaltenswissenschaften (Littman, 2015).

### 2.1.1 Reinforcement Learning Setting

Das RL Framework besteht im Wesentlichen aus zwei Komponenten: den Agenten und dem Environment (Umgebung, Domain). Der Agent ist imstande, Neues zu erlernen, Entscheidungen zu treffen und mit dem Environment zu interagieren. Das Environment kann als das betrachtet werden, was der Agent nicht kontrollieren kann. In den anderen Machine Learning Problemen steht der Datensatz zu Beginn zur Verfügung. Beispielsweise besteht das Ziel bei Supervised Learning Verfahren darin, anhand eines gegebenen Datensatzes die Parameter unter Berücksichtigung einer Zielfunktion zu optimieren. Im Gegensatz dazu werden in RL die Daten anhand der Interaktionen mit der Umgebung zuerst generiert und anschliessend anhand der erhaltenen Rückmeldung (Reward) optimiert. Der Reward ist eine skalare Grösse und liefert dem Agenten einen Hinweis (Indikation) über die Nützlichkeit der ausgeführten Aktion. Das Ziel des Agenten besteht somit darin, mit dem Environment so zu interagieren, dass die Summe der erhalten Rewards maximiert wird.

### 2.1.2 Reinforcement Learning Komponenten

Die Interaktion dieser beiden Komponenten bildet die eigentliche Basis eines Reinforcement Learning Problems und wird deshalb im folgenden Abschnitt beschrieben.

#### Environment

Das Environment beschreibt die Umgebung, in welcher der Agent Aktionen ausführen kann und in welcher der Agent interagieren kann. Im Environment sind der Zustandsraum $S$ (State Space), der Aktionsraum $A$ (Action Space) und die Reward Funktion definiert. Diese beeinflussen das Verhalten beziehungsweise die Dynamik des Agenten. Jedes Environment lässt sich typischerweise mithilfe vier allgemeinen Eigenschaften charakterisieren (Gatti, 2015):

- **Zeithorizont:**
  Beschreibt die Zeitdauer, in welcher der Agent lernen kann.
- **Anzahl Agenten:**
  Beschreibt die Anzahl Agenten im Environment (Single Agent/Multi Agent).
- **Stationarität des Environments:**
  Beschreibt Veränderungen, welche die Agent-Environment-Interaktion beeinflussen können.
- **Beobachtbarkeit:**
  Beschreibt, welche Informationen dem Agenten zur Verfügung stehen.

#### Agent

Der Agent besteht aus drei Elementen: der Repräsentation, des Learning Algorithmus und einer Policy für die Selektierung der nächste Aktion (Action Selection Policy).

#### Repräsentation

Für die Beschreibung der Zustände beziehungsweise für die Speicherung der Zustandswerte wird in Reinforcement Learning ein Datenspeicher (Gedächtnis) benötigt, der es erlaubt, für einen bestimmten Zustand den Wert abzurufen und zu aktualisieren. Dieser Datenspeicher (Repräsentation) ist abhängig von den Eigenschaften des Environments. In Reinforcement Learning wurden viele unterschiedliche Repräsentationen verwendet, beispielsweise Lookup-Tabellen, lineare Methoden oder neuronale Netze. Letztere verfügen gemäss Gatti (2015) über drei entscheidende Vorteile:

- **Generalisierung:** Gegenüber den Lookup-Tabellen und ähnlich wie bei linearen Methoden verfügen neuronale Netze über den entscheidenden Vorteil, dass Zustandswerte generalisiert werden können. Dies bedeutet, dass für Zustände, die nicht explizit besucht wurden, geeignete Zustandswerte vergeben werden können. Diese Eigenschaft kann die Trainingszeit enorm verkürzen.
- **Parametrisierung:** Im Vergleich zu der Grösse des Zustandsraums benötigen neuronale Netze relativ wenige Parameter.

- **Implizite Features:** Neuronale Netze verfügen über die Möglichkeit, aus den Rohdaten geeignete Features abzuleiten. Das aufwendige Feature-Engineering für die Zustandswerte fällt somit weg.

Trotz der attraktiven Vorteile von neuronalen Netzen wurden diese lange nicht favorisiert. Instabilitäten, nicht optimale Lösungen und Divergenzen sind einige Gründe, welche den Einsatz von neuronalen Netzen erschweren. Das erfolgreiche Trainieren von neuronalen Netzen ist abhängig von der Funktion, die approximiert werden soll. Für eine detaillierte Auflistung der Probleme sowie eine umfangreiche Literaturauflistung wird auf Gatti (2015) verwiesen.

**Learning Algorithmus**
Der Learning Algorithmus ist das Herzstück eines Agenten. Er erlaubt ihm, die Dynamik des Environments zu lernen, sodass der Agent ein gewünschtes Ziel (beispielsweise den Reward zu maximieren) erreichen kann (Gatti, 2015).

**Action Selection Policy**
Der Lernprozess des Agenten wird durch die Selektierung der Aktionen beeinflusst. Wann welche Aktion ausgeführt wird, ist entscheidend um eine optimale Strategie zu erlernen (vgl. Abschnitt 2.6).

### 2.1.3 Reinforcement Learning Notation

Typischerweise interagiert der Agent über mehrere diskrete Zeitschritte $t$ mit dem Environment. In jedem Zeitschritt $t$ beobachtet der Agent einen Zustand $s_t \in S$ und wählt gemäss seiner aktuellen Policy $\pi$ eine Aktion $a_t \in A$ aus. Eine Policy $\pi$ ordnet jedem Zustand $s$ eine Aktion $a$ zu, $\pi : s \mapsto a$. Die Policy $\pi$ ist somit die Verhaltensstrategie des Agenten. Für die ausgeführte Aktion erhält der Agent vom Environment eine Rückmeldung in Form einer skalaren Belohnung oder Bestrafung (Reward $r$). Jede ausgeführte Aktion beeinflusst den internen Zustand des Environments und führt das Environment und den Agenten in den nächsten Zustand $s_{t+1}$. In einem Environment mit episodischem Zeithorizont wird der Zustand nach $T$ Zeitschritten zurückgesetzt. Die Sequenz von Zuständen, Aktionen und Rewards in einer Episode stellt eine Trajektorie $\tau$ (oder Rollout) dar. Die erhaltenen Rewards in einer Trajektorie sind von der aktuellen Policy abhängig und werden aufsummiert. Der Ertrag oder Return ($R$) wird wie folgt berechnet:

$$R = \sum_{k=0}^{T} \gamma^k r_{t+k} \,. \tag{2.1}$$

Rewards, welche später erteilt werden, werden mithilfe des Diskontierungsfaktors $\gamma$ gewichtet. Je kleiner $\gamma$ ist, desto weniger werden zukünftige Rewards berücksichtigt. Das Ziel von Reinforcement Learning ist es somit, eine Policy zu finden, welche den erwarteten Return von allen Zuständen aus maximiert. Diese Policy wird als optimale Policy $\pi^*$ bezeichnet:

$$\pi^* = \arg\max_{\theta} \mathbb{E}\left[R \mid \pi\right] \,. \tag{2.2}$$

In der Abbildung 2.1 ist die Agent-Environment-Interaktion dargestellt. Im Allgemeinen wählt der Agent aufgrund seiner Wahrnehmung (Perzeption) und der Schätzung der Nützlichkeit (Wert) des darauffolgenden Zustandes eine Aktion aus. Die

Rückmeldung des Environments hilft dem Agenten, seine Schätzung über die Nützlichkeit der Zustände zu verbessern. Im Normalfall werden die Rewards in RL nach jeder Aktion bereitgestellt. Die wiederholte Interaktion mit dem Environment ermöglicht es dem Agenten, seine Schätzung laufend zu verbessern und in Zukunft eine optimalere Aktion auszuführen (Gatti, 2014).

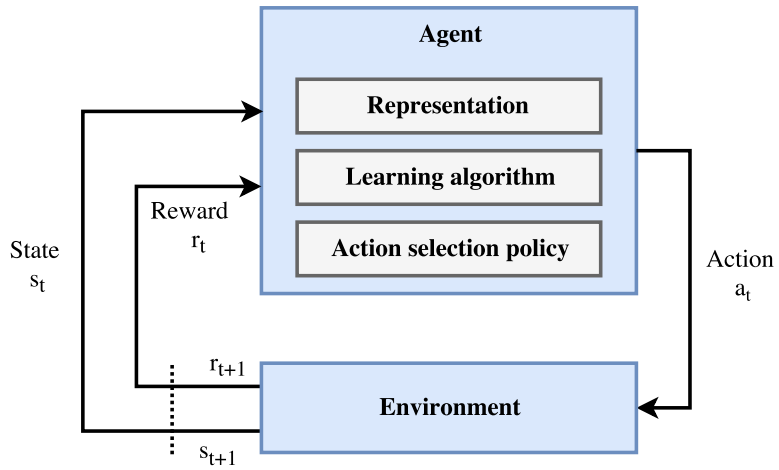



ABBILDUNG 2.1: Agent-Environment-Interaktion. Zum Zeitpunkt $t$ beobachtet der Agent den Zustand $s_t$ vom Environment. Basierend auf seiner aktuellen Policy trifft der Agent eine Entscheidung bzw. führt eine Aktion $a_t$ aus. Das Environment wird in den nächsten Zustand $s_{t+1}$ überführt und gibt dem Agent einen Reward $r_{t+1}$. Mit zunehmender Zeit lernt der Agent, geeignete Aktionen auszuführen, welche den Return maximieren (Gatti, 2014).

**Zusammengefasst:**

- Aktionen $a$ sind alle Fähigkeiten, die ein Agent zu einer bestimmten Zeit ausführen kann (z.b. schiessen, sich nach recht drehen, geradeaus laufen etc.)
- Die Beobachtung der Umgebung wird als Zustand $s$ bezeichnet.
- Der Agent muss auf Basis des aktuellen Zustands entscheiden, was er als nächstes tun soll.
- Der Reward $r$ ist die Belohnung beziehungsweise Bestrafung einer ausgeführten Aktion $a$ in einem bestimmten Zustand $s$ vom Environment.
- Eine Trajektorie $\tau$ ist die Sequenz der Zustände, Aktionen und Rewards in einer Episode mit der Länge $T$.
  - Beispiel: $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \cdots, s_{T-1}, a_{T-1}, r_T)$
  - Dabei ist $r_{t+1}$ der erhalten Reward, wenn man im Zustand $s_t$ die Aktion $a_t$ ausführt.

### 2.1.4 Partially Observable Markov Decision Processes (POMDP)

In vielen Problemfeldern ist es aufgrund der limitierten Wahrnehmung (Abtastungsmöglichkeiten) des Agenten nicht möglich, den ganzen Zustand des Environments zu beobachten. Beispielsweise kann die Beobachtung von einem Rauschsignal gestört werden oder mehrere unterschiedliche Beobachtungen können zum gleichen Zustand führen (Wiering et al., 2012). Infolgedessen wird die von vielen Reinforcement Learning Methoden vorausgesetzte Markov-Eigenschaft verletzt. Die Markov-Eigenschaft besagt, dass nur der aktuelle Zustand den nächsten Zustand beeinflusst (The future is conditionally independent of the past given the present state). Dies bedeutet, dass jede Entscheidung, die im Zustand $s_t$ getroffen wird, alleine auf dem aktuell beobachtbaren Zustand $s_{t+1}$ basiert und nicht auf den vorhergehenden Zuständen $\{s_0, s_1, ..., s_{t-1}\}$.

Eine Abhilfe bieten Partially Observable Markov Decision Processes (POMDPs) an. Im Allgemeinen werden in RL Markov Decision Processes (MDP) als mathematisches Framework genutzt (Bellman, 1957). POMDP ist eine Erweiterung (oder Generalisierung) der MDP und erlaubt eine Entscheidung unter der Bedingung der Unsicherheit (Condition of Uncertainty) zu treffen (Wiering et al., 2012). Diese Erweiterung ist in vielen komplexeren und realitätsnahen Problemfeldern zwingend notwendig (beispielsweise FPS Games). Diese Erweiterung ermöglicht dem Agenten eine optimale Entscheidung in nicht vollständig beobachtbaren Environments zu treffen (Kaelbling et al., 1998). In einem POMDP Modell erhält der Agent nicht den ganzen Zustand, sondern lediglich eine Beobachtung $o_t \in \Omega$. Die Wahrscheinlichkeitsverteilung der Beobachtung $p(o_{t+1} \mid s_{t+1}, a_t)$ ist dabei abhängig vom aktuellen Zustand des Environments und von der getroffenen Aktion. Typischerweise unterhalten POMDP Algorithmen eine Wahrscheinlichkeitsverteilung, welche frühere Beobachtungen miteinbeziehen (Arulkumaran et al., 2017).

Eine optimale Policy $\pi^*$ soll in Reinforcement Learning ohne das Wissen über die Dynamik des Systems beziehungsweise das Vorhandensein eines Modells gelernt werden. Dies bedeutet, dass die Übergangsfunktion, also die Wahrscheinlichkeit beim Ausführen einer Aktion in den nächsten Zustand zu gelangen, nicht zur Verfügung steht. Der Agent kann eine optimale Policy $\pi^*$ nur lernen, indem er mit dem Environment interagiert und die erhaltenden Informationen mittels geeigneter Algorithmen verarbeitet. Grundsätzlich existieren zwei unterschiedliche Methoden, um Reinforcement Learning Probleme zu lösen:

- **Model-free Methoden**:
  Model-free Methoden versuchen direkt eine optimale Policy $\pi^*$ zu lernen.
- **Model-based Methoden**:
  Model-based Methoden lernen zuerst ein Modell der unterstehenden Dynamik und leiten danach eine optimale Policy $\pi^*$ ab.

Diese Arbeit fokussiert sich auf Model-Free Methoden.

## 2.2 Lösungsmöglichkeiten

Grundsätzlich existieren drei unterschiedliche Möglichkeiten um RL Probleme zu lösen:

- Methoden, die auf einer **Value Funktion (Critic-only)** basieren.
- Methoden, die auf einer **Policy Funktion (Actor-only)** basieren.
- Hybride Methoden **(Actor-critic)**, welche die beiden genannten Methoden kombinieren.

In den folgenden Abschnitten werden zuerst wichtige Funktionen definiert und anschliessend populäre Algorithmen vorgestellt.

### 2.2.1 Value Funktion Methoden

Das Kernkonzept von Critic-only Methoden besteht darin, den Wert eines Zustandes zu schätzen. Dieser Wert (Bewertung) wird als State-Value bezeichnet und kann mithilfe der Value Funktion beziehungsweise der State-Value Funktion $V^{\pi}(s)$ wie folgt berechnet werden:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi\right] = \mathbb{E}\left[R \mid s, \pi\right] \tag{2.3}$$

Der Wert (Value) eines Zustandes $s$ unter Berücksichtigung der Policy $\pi$ ist somit der erwartete kumulierte Return, wenn man von $s$ aus der Policy $\pi$ folgt. Die Value Funktion gibt den Agenten Auskunft über den Wert eines bestimmten Zustandes. Da die Dynamik des System in RL nicht zur Verfügung steht, wird eine weitere wichtige Funktion benötigt, die State-Action Funktion. Ähnlich der $V^{\pi}$ ist die State-Action Funktion oder Quality Funktion $Q^{\pi}(s, a)$ wie folgt definiert:

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right] = \mathbb{E}\left[R \mid s, a, \pi\right] \tag{2.4}$$

Der Q-Value eines Zustandes ist der erwartete kumulierte Return, wenn man im Zustand $s$ die Aktion $a$ ausführt und der Policy $\pi$ folgt. Mithilfe einer gegeben Q-Funktion, kann die optimale Policy $\pi^*$ einfach ermittelt werden. In jedem Zustand wird diejenige Aktion ausgeführt, welche den grössten Q-Value aufweist (greedy). Zusätzlich zu den zwei vorhergehenden Funktionen wird die Advantage Funktion definiert:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s) \tag{2.5}$$

Die Advantage Funktion ist die Differenz der beiden Funktionen und bewertet die Ausführung einer Aktion $a$ im Zustand $s$ im Vergleich zum Mittelwert. Die Advantage Funktion ist positiv, wenn die Aktion einen Vorteil bringt und sonst negativ.

Das Ziel dieser Methoden besteht darin, eine State-Value oder State-Action Funktion zu schätzen, um eine optimale Policy $\pi^*$ abzuleiten.

**Temporal Difference Learning**

Eine Möglichkeit die Value- oder Q-Funktion zu lernen, ist Temporal Difference Learning (TD-Learning). TD-Learning Methoden sind iterative Algorithmen und passen ihren Schätzer mit jeder Iteration an. Dies bedeutet, dass die Value- oder Q-Funktion nach jeder Aktion mithilfe des beobachtbaren Rewards und mithilfe des aktuellen geschätzten Werts aktualisiert wird (Bootstrapping).

**Q-Learning**

Eine Variante von TD-Learning ist der entwickelte Q-Learning Algorithmus von Watkins et al. (1992). Q-Learning ist ein off-policy Algorithmus und ist imstande, ohne die Dynamik der Umgebung zu kennen, eine optimales Verhalten zu lernen. Off-policy Algorithmen sind in der Lage unabhängig von ihrer Policy, die State-Action Funktion Q zu approximieren, um die optimale Funktion $Q^*$ zu erhalten (Sutton et al., 1998). Die Grundlagen von Q-Learning stammen aus der Dynamischen Programmierung (Bellman, 1952). Die Q-Funktion wird daher zuerst als Bellman Gleichung umgeschrieben, beziehungsweise in rekursiver Form:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}\left[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))\right] . \tag{2.6}$$

In dieser rekursiven Form ist ersichtlich, dass mithilfe der aktuellen Schätzung der Q-Funktion die neue Schätzung verbessert wird. Dieser Vorgang wird als Bootstrapping bezeichnet und ermöglicht, die Schätzung in jeder Iteration zu verbessern.

Der Q-Learning Algorithmus sieht vor, dass im aktuellen Zustand $s$ gemäss der derzeitigen Policy $\pi$ eine Aktion $a$ ausgeführt wird. Aus dem Folgezustand $s_{t+1}$ wird dann die erfolgsversprechenste Aktion $a$ ausgewählt. Die Q-Funktion wird anhand folgender Formel angepasst:

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha\delta \, , \, \delta = r_t + \gamma \max_a Q^{\pi}(s_{t+1}, a_t) - Q^{\pi}(s_t, a_t) \, . \tag{2.7}$$

Dabei stellt $\alpha$ die Lernrate und $\delta$ den Temporal Difference (TD) Error dar. Um nun die optimale Q-Funktion zu finden, nutzt der Algorithmus folgende zwei Konzepte:

- **Policy Evaluation:**
  Hierbei wird die Schätzung der Q-Funktion verbessert, indem der TD Error anhand der aktuellen Policy minimiert wird.
- **Policy Improvement:**
  Mit einer genaueren Schätzung der Q-Funktion kann die Policy verbessert werden, indem in jedem Zeitschritt die beste Aktion gewählt wird.

Policy Evaluation und Policy Improvement werden während dem Training nicht separat durchgeführt, sondern verschachtelt (interleaving).

### 2.2.2 Policy-based

Actor-only Methoden verwalten keine Value Funktion, sondern suchen direkt nach einer optimalen Policy $\pi^*$. Mithilfe von Optimierungsverfahren werden somit die Parameter einer Policy gesucht, sodass der erwartete Return maximiert wird. Viele DRL Algorithmen setzen auf gradientenbasierte Optimierung, da weniger Stichproben benötigt werden (Sample Efficient) (Arulkumaran et al., 2017). Aus diesem Grund wird die Idee dieser Methode in folgenden Abschnitten vorgestellt.

#### Policy Gradient Methode

Die Idee der Policy Gradient Methode besteht im Wesentlichen darin, den erwartetet Return zu maximieren, indem die Parameter einer Policy mithilfe von genügenden Stichproben aktualisiert werden. Das Ziel besteht somit, den folgenden Erwartungswert zu maximieren:

$$\mathbb{E}_{\pi_\theta}\left[R_\tau\right]. \tag{2.8}$$

Dieser Erwartungswert beschreibt den erwarteten Return, welcher der Agent erhält, wenn er die Policy $\pi$ mit den Parameter $\theta$ auswertet. Die Policy Gradient Methode versucht mithilfe des Gradientenverfahren (Gradient Ascent) diesen Erwartungswert zu maximieren. Somit wird folgender Gradient benötigt:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}\left[R_\tau\right]. \tag{2.9}$$

Oft kann dieser Gradient nicht direkt berechnet werden und wird daher mithilfe der REINFORCE Regel von Williams (Williams, 1992) wie folgt geschätzt:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}\left[R_\tau\right] = \mathbb{E}\left[R_\tau \nabla_\theta \log \pi(a_t \mid s_t; \theta)\right]. \tag{2.10}$$

Anstatt den Gradienten des Erwartungswerts direkt zu berechnen, muss mithilfe dieses Schätzers lediglich der Gradient der Policy berechnet werden. Vereinfacht ausgedrückt bedeutet dies folgendes: Der Agent wertet eine Policy $\pi$ mit den Parameter $\theta$ aus und erhält eine Trajektorie $\tau$. Mithilfe von $\tau$ kann $\hat{g}(\tau) = R_\tau \nabla_\theta \log \pi(a_t \mid s_t; \theta)$ berechnet werden. Der Mittelwert von genügend solcher Trajektorien liefert einen guten Schätzer für den Erwartungswert (vgl. Formel 2.10).

Die Berechnung des Gradienten basiert auf den empirischen Return der Trajektorien und besitzt somit eine grosse Varianz. Eine Möglichkeit um diese Varianz zu reduzieren, ist den Gradienten nicht mit den ganzen $R_\tau$ zu gewichten, sondern nur mit relativen Vorteil.

## 2.3 Deep Reinforcement Learning

Speicherkomplexitäts-, Rechenkomplexitäts- und Samplekomlextitäts-Probleme erschweren das Anwenden von RL Algorithmen auf Problemstellungen mit grossen Zustands- und Aktionsräumen, wie beispielsweise komplexe 3D Games (Strehl et al., 2006). Diese Probleme konnten teilweise mithilfe von Deep Learning (DL) überwunden werden. In vielen Bereichen von Machine Learning verhalf der Einsatz von DL, den State of the Art Ansatz in verschiedenen Aufgaben, wie beispielsweise der Objekterkennung, Spracherkennung oder Übersetzung, drastisch zu verbessern (LeCun et al., 2015; Arulkumaran et al., 2017).

Deep Learning verfügt über wichtige Eigenschaften: Deep Neural Networks (DNN) sind sehr mächtige Funktionsapproximatoren. Diese DNN sind in der Lage, automatisch kompakte Features (Repräsentationen) aus hochdimensionalen Daten (beispielsweise Bilder, Text oder Audio) zu finden und zu extrahieren.

Im Bereich von Reinforcement Learning erlaubt der Einsatz von Deep Learning, Entscheidungsprobleme zu lösen, welche früher aufgrund ihrer grossen Zustands- und Aktionsräume nicht lösbar waren. Die Benutzung von DL Algorithmen innerhalb von RL definiert den Bereich von Deep Reinforcement Learning (DRL). Im Vergleich zu den klassischen Methoden, welche auf Tabellen oder nichtparametrisierten Funktionen basieren, können DRL effizienter mit dem rapiden Anstieg der Komplexität in grösseren Environments (vgl. Curse of Dimensionality in Abschnitt 2.6) umgehen. In DRL werden neuronale Netze verwendet, um die optimale Policy Funktion $\pi^*$ oder die optimalen Value Funktionen $V^*$, $Q^*$ oder $A^*$ zu approximieren. Für einen umfassenden Überblick über Deep Learning wird an dieser Stelle auf die Arbeit von LeCun et al. (2015) verwiesen.

## 2.4 Algorithmen

### 2.4.1 Deep Q-Network (DQN)

Ein populärer Deep Reinforcement Learning Algorithmus ist Deep Q-Network (DQN). DQN basiert auf einer Value-Funktion und kombiniert Reinforcement Learning (Q-Learning) mit Deep Learning, indem die Q-Value Funktion anhand eines Deep Neural Networks geschätzt wird. Die Idee des Algorithmus besteht darin, die Parameter $\theta$ des neuronalen Netzes so anzupassen, dass für $Q_\theta(s,a) \approx Q^*(s,a)$ gilt. Für die Erkundung des Environments nutzt DQN die $\epsilon$-Greedy Explorationstrategie.

DQN lernt die optimale $Q$-Funktion $Q^*$, indem die erhaltenen Rewards verwendet werden, um die Schätzung der Q-Funktion anzupassen. Die Parameter (beziehungsweise die Gewichte des Netzwerks) werden angepasst, indem der mittlere quadratische Fehler zwischen der aktuellen Schätzung und der neuen Schätzung berechnet wird und durch das Netzwerk rückpropagiert wird. DQN nutzt unterschiedliche Techniken, um das Training zu stabilisieren. Beispielsweise wird die Q-Funktion nicht in jeder Iteration neu geschätzt, sondern für mehrere Iterationen beibehalten. Dafür nutzt DQN ein zusätzliches neuronales Netz (Target Network), welches nicht trainiert wird, sondern die Parameter vom aktuellen Netz in periodischen Intervallen erhält.

Die entsprechende Zielfunktion von DQN ist demnach wie folgt:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s^{t+1}}\left[(Q^*_\theta(s,a) - y)^2\right], \text{mit } y = r + \gamma \max_\theta \bar{Q}^*(s_{t+1}, a_{t+1}). \tag{2.11}$$

$\bar{Q}$ stellt hierbei die Schätzung der Q-Funktion des Target Networks dar. Zusätzlich zu Target Network wird ein Pufferspeicher (Experience Replay Buffer) verwendet, um Korrelationen zwischen den Samples zu brechen und so ein Overfitting zu verhindern. In diesem Speicher werden Transitionen beziehungsweise Erfahrungen des Agenten, bestehend aus $(s, a, r, s_{t+1})$ abgespeichert, welche genutzt werden, um das Netzwerk zu trainieren.

DQN verwendet eine Architektur bestehend aus mehre Convolutional Neural Networks (CNN) gefolgt von mehreren Fully-connected Layer (FCN) (Mnih et al., 2015).

### 2.4.2 Asynchronous Advantage Actor Critic (A3C)

Ein weiterer populärer Deep Reinforcement Learning Algorithmus ist der Asynchronous Advantage Actor Critic (A3C) Algorithmus. Actor Critic beschreibt ein Konzept, das ursprünglich von (Witten, 1977) und dann von (Barto et al., 1983) eingeführt wurde. Actor Critic Algorithmen kombinieren die zwei RL Lösungsmöglichkeiten in einer Architektur. Die Begriffe Actor und Critic wurden von (Barto et al., 1983) eingeführt. Actor-only Methoden passen die Policy direkt an, wohingegen Critic-only Methoden auf der Evaluation von Value- oder Q-Funktionen basieren. Actor Critic Algorithmen unterhalten somit Parameter Schätzungen sowohl für die Policy $\pi_{\theta_a}(a \mid s)$ als auch für die Value Funktion $V_{\theta_c}(s)$. Die Schätzungen der Value Funktion werden genutzt, um die Varianz der Policy Gradient Updates zu reduzieren.

In (Mnih et al., 2016) haben die Autoren das Actor Critic Konzept genutzt, um den Asynchronous Advantage Actor Critic (A3C) Algorithmus zu entwickeln. Dabei werden $k$ Learner Threads (Worker) genutzt, welche $k$ Kopien der Policy asynchron ausführen und ihre Parameter Updates an einen zentralen Parameter Server in regelmässigen Zeitintervallen senden. Die Policy und Value Funktion werden von einem Deep Neural Network parametrisiert. A3C ist ein On-Policy Algorithmus, da die $k$ Learner Threads mit der gleichen Policy arbeiten. Die asynchronen Learner Threads wurden eingeführt, um die Exploration des Environments zu stärken, da die verschieden Threads verschiede Teile des Zustandsraum parallel explorieren. Zusätzlich wird das Lernen stabilisiert und beschleunigt, da die zeitliche Korrelation von Transitions mit den parallelen Learner Threads aufgebrochen wird. Die Autoren verwenden für den Actor-Teil folgende Zielfunktion:

$$\mathcal{L}(\theta_a) = \log \pi_{\theta_a}(a_t \mid s_t)(G_t - V(s_t)). \tag{2.12}$$

$G_t$ ist hierbei eine Schätzung des Returns im Zeitschritt $t$. Die Differenz $G_t - V_{s_t}$ ist die Schätzung der Advantage Funktion $A_t$ und sagt aus, welchen Vorteil die Ausführung der Aktion $a_t$ im Zustand $s_t$ bringt. Die Value Funktion (Critic-Teil) wird separat aktualisiert mithilfe des $n$-step TD Erros:

$$\mathcal{L}(\theta_c) = \left(\hat{V}(s_t) - V_{\theta_c}(s_t)\right)^2 \tag{2.13}$$

## 2.5 Reinforcement Learning in FPS Games

Für die Forschung im Bereich von Reinforcement Learning eignen sich Games exzellent als Testumgebung. Im Allgemeinen sind Games so konzipiert, dass sie in erster Linie den Spieler unterhalten, amüsieren und herausfordern. Zudem müssen in einem Game eine Reihe von interessanten Entscheidungen vom Spieler gefällt werden, um ein bestimmtes Ziel zu erreichen. Aufgrund dieser Eigenschaften stellen Games eine komplexe Domäne dar. Durch die Untersuchung von unterschiedlichen Games (Strategie, FPS etc.) erhoffen sich die Wissenschaftler, mehr über die menschliche Intelligenz herauszufinden; beispielsweise die genauen Herausforderungen und Anforderungen, um eine bestimmte Aufgabe lösen zu können (Wiering et al., 2012).

### 2.5.1 Herausforderung von RL in Games

Das Anwenden von RL Methoden für die Entwicklung eines intelligenten Agenten (Bot) für ein 3D Environment, wie beispielsweise Doom, ist eine anspruchsvolle Aufgabe. Grundsätzlich liegen die Herausforderungen des Anwendens von RL Methoden im Bereich von Games beim Finden einer geeigneten Repräsentation, beim Verwenden einer effizienten Explorationsmethode, beim Anwenden einer geeigneten Trainingsmethode, beim Umgang mit fehlenden Informationen und beim Modellieren des Verhaltes der Gegenspieler (Wiering et al., 2012).

Im Vergleich zu den Atari 2600 Games ist das Doom Game komplexer. Der Agent muss nicht nur mit einem grösseren Aktionsraum zurechtkommen, sondern er muss eine geeignete Strategie für eine spezifische Situation formen. Der Agent muss ausserdem in der Lage sein, durch das Environment (Map) zu navigieren, Items einzusammeln, Gegenspieler zu erkennen und diese zu bekämpfen. Des weiteren verhindert die zusätzliche räumliche Dimension, dass der Agent den vollständigen Zustand des Environments beobachten kann. Der Agent muss somit Entscheidungen mit fehlenden Informationen treffen (Lample et al., 2017; Bhatti et al., 2016).

#### Beispiel: Agent für ein FPS Game

Für das FPS Game Doom erhält der Agent in jedem Zeitschritt den aktuellen Screen als Pixelmatrix vom ViZDoom Environment. Der Agent kann nun eine verfügbare zulässige Aktion auswählen (beispielsweise ATTACK) und ausführen. Dies führt dazu, dass das Environment das Game einen Zeitschritt vorrückt. Falls nun der Agent einen Gegenspieler getroffen und eliminiert hat, wird er vom Environment mit einem positiven Signal (Reward = +1) belohnt. Andernfalls erhält er ein neutrales Signal (Reward = 0). Danach wiederholt sich der Ablauf. Der Agent erhält vom Environment wieder eine neue Beobachtung und hat wieder die Möglichkeit, eine Aktion auszuwählen und auszuführen.

## 2.6 Praktische Probleme in Reinforcement Learning

In komplexen Environments stellt das Trainieren eines Reinforcement Learning Algorithmus eine grosse Herausforderung dar. Zeitlich verzögerte und spärliche (sparse) Rewards sowie die Erkundung von grossen Zustandsräumen erschweren den Lernfortschritt und die Stabilität des Algorithmus. Zu den Herausforderungen zählen gemäss Heidrich-Meisner et al. (2007) die in diesem Abschnitt beschriebenen Punkte.

#### The Temporal Credit Assignment Problem

Für eine ausgeführte Aktion erhält der Agent den Reward oftmals zeitlich verzögert. Dies bedeutet, dass der erzielte Reward nicht zwingend mit der letzten Aktion korreliert. Der Agent muss in der Lage sein, frühere Aktionen auf den erzielten Reward abzubilden. Dieses Problem wird als **Temporal Credit Assignment Problem** bezeichnet (Heidrich-Meisner et al., 2007). Es lässt sich am besten mit dem Brettspiel Schach veranschaulichen. Der Spieler weiss erst nach Beendigung des Spiels, ob seine Züge (Aktionen) zum Erfolg geführt haben. Ein ausgeführter Zug kann im Augenblick keine grosse Auswirkung auf das Spiel haben, allerdings kann er entscheidend sein für das Ergebnis. Natürlich kann diese zeitliche Verzögerung auch viel kürzer sein.

#### The Exploration-Exploitation Dilemma

In einem unbekannten Environment muss der Agent zuerst den Zustands-Aktions-Raum (State Action Space) explorieren, um die Struktur und die Eigenschaften des Environments zu lernen. Anhand der Trial-and-Error-Methode sammelt der Agent die nötige Erfahrung über das Environment. Mit zunehmendem Wissen über das Environment wird die Exploration weniger relevant und der Agent kann sein aktuelles Wissen verwenden, um den Reward zu maximieren. Im Kontext von Reinforcement Learning bedeutet das Explorieren eine für den aktuellen Zustand und Wissensstand nicht optimale Aktion auszuführen. Die grosse Schwierigkeit besteht darin, den Zeitpunkt herauszufinden, wann das bestehende Wissen des Agenten ausgenutzt werden soll und wann eine nicht optimale Aktion beziehungsweise das Environment exploriert werden soll. Dieses Problem wird als **Exploration-Exploitation Dilemma** bezeichnet (Heidrich-Meisner et al., 2007).

Eine beliebte Explorationsstrategie beispielsweise für DQN ist die $\epsilon$-Greedy Strategie. Bei $\epsilon$-Greedy wird eine zufällige Aktion mit der Wahrscheinlichkeit $\epsilon \in [0,1]$ veranlasst, ansonsten wird eine optimale (in Bezug auf dem aktuellen Wissensstand) Aktion ausgeführt. Der Wert von $\epsilon$ wird im Verlauf des Trainings linear bis auf 0 verringert, sodass der Agent am Ende des Trainings nur noch seinen Wissensstand auswertet.

### The Curse of Dimensionality

Um sicherzustellen, dass ein RL Algorithmus konvergiert, müssen die Zustände oder Zustands-Aktions-Paare genügend oft aufgesucht werden. In komplexen Environments ist dies aufgrund der vielen Zustände nicht möglich, beziehungsweise nicht praktikabel. Dieses Problem ist bekannt als **The curse of dimensionality** (Bellman, 1961). Der Agent muss daher in der Lage sein, zu generalisieren. Aus seiner Erfahrung muss er Eigenschaften lernen, die für noch nicht gesehene Zustände oder Zustands-Aktions-Paare verwendet werden können. Diese Generalisierung kann durch Funktionsapproximation erreicht werden (Heidrich-Meisner et al., 2007).

Zusätzlich zu den vorgestellten Problemen stellt das Finden von geeigneten Parametern ein Problem dar. Die Trainingszeiten von DQN und A3C für komplexe Reinforcement Learning Problemen können von einigen Stunden bis zu mehren Tagen andauern und erschweren somit eine exzessive Parametersuche.

## 2.7 Trainingsstrategien für Games

Eine allgemeine Trainingsstrategie besteht darin, die Erfahrungen beziehungsweise die Parameter der Algorithmen (DQN und A3C) von anderen Environments zu übernehmen. Um diese Parameter übernehmen zu können, werden typischerweise die Beobachtungen (Pixelmatrix) und die Rewards normalisiert (Mnih et al., 2013). In den folgenden Abschnitten werden weitere Strategien vorgestellt.

### 2.7.1 Frame Skip

Typischerweise führen RL Agenten in jedem Zeitschritt eine Aktion aus beziehungsweise interagieren in jedem Zeitschritt mit dem Environment. Die Häufigkeit der Interaktionen des Agenten mit dem Environment ist entscheidend für den Lernerfolg (Braylan et al., 2015). In Games bedeutet dies, dass ein Agent für jedes erhaltene Frame eine Entscheidung treffen muss. Beispielsweise läuft das ViZDoom Environment mit 35 Frames in der Sekunde. Ein Agent hat somit die Möglichkeit, für jedes Frame eine Entscheidung zu treffen. Um den Lernprozess des Agenten zu beschleunigen, werden die Anzahl Entscheidungen reduziert, indem die gleiche Aktion mehrere Frames lang wiederholt wird. Diese Methode wird als **Frame Skip** bezeichnet. Existierende RL Ansätze im Bereich Videos Games setzen auf eine statische Anzahl Frame Skips, das heisst, die Anzahl der Wiederholungen wird zu Beginn des Trainings definiert und bleibt konstant.

Die Wichtigkeit von Frame Skip wurde von Braylan et al. (2015) im Kontext der Arcade Learning Environment (ALE) (Bellemare et al., 2013) untersucht. Die Autoren haben festgestellt, dass Frame Skip ein mächtiger Parameter ist, um einen Agenten für Atari 2600 Games trainieren zu können. Gemäss den Autoren bestehen die Vorteile von Frame Skip bei rechenintensiven Environments, in der signifikanten Reduktion der Simulationszeit und der Vorbeugung von Super-Human-Reflex Strategien (Hausknecht et al., 2014). Die Reduktion der Simulationszeit kann nur auf Kosten von verpassten Gelegenheiten erzielt werden. Die Autoren haben gezeigt, dass Agenten mit höheren Frame Skips schneller trainiert werden können und dass die Agenten eine bessere Fähigkeit besitzen, um Assoziationen zwischen zeitlich distanzierten Zuständen und Aktionen zu lernen. Diese Eigenschaft ist vor allem in nicht vollständig beobachtbaren Environments von grossem Vorteil (vgl. Abschnitt 2.1.4).

Neben Frame Skip werden in nicht vollständig beobachtbaren Environments ausserdem Recurrente Neuronale Netze (RNNs) eingesetzt (Hausknecht et al., 2015). RNNs sind dynamische Systeme die sich dadurch kennzeichnen, dass Verbindungen von Neuronen einer Schicht zu anderen Neuronen derselben oder einer vorangegangenen Schicht existieren. Solche Rückkoppelungen helfen, zeitlich codierte Informationen in den Daten zu entdecken. Im Bereich der sequentiellen Entscheidungsprobleme haben sich Long Short-Term Memory (LSTM), eine Variante von RNNs, bewährt (Mnih et al., 2013).

### 2.7.2 Reward Shaping

In komplexen Environments ist es möglich, dass der Agent den Reward verzögert erhält. Beispielsweise in Games erhalten die Agenten nur einen Reward, wenn sie ihre Gegenspieler eliminieren (+1) oder von einem Gegenspieler getötet werden (-1). Zu Beginn des Trainings benötigt der Agent sehr viele Versuche, bis er einen Gegenspieler trifft und einen Reward erhält. Eine effektive, aber kritische Möglichkeit ist das Hinzufügen von zwischenliegenden (intermediären) Rewards. Diese Vorgehensweise wird als Reward Shaping bezeichnet (Ng et al., 1999; Devlin et al., 2011). Allerdings besteht das Risiko bei diesem Vorgehen, dass der Agent aufgrund der zwischenliegenden Rewards eine nicht optimale Policy (Strategie) lernt, da falsche Annahmen (Bias) getroffen werden (Irpan, 2018).

### 2.7.3 Transfer Learning für RL

Transfer Learning im Bereich von Reinforcement Learning ist ein eigenes Forschungsgebiet mit dem Ziel, das Learning des Agenten zu beschleunigen und im optimalen Fall zu verbessern. Im Allgemeinen wird versucht, das Gelernte von einem einfachen Task (Source Task) für einen komplexeren Task (Target Task) zu übernehmen (Narvekar, 2016).

#### Curriculum Learning

In komplexen Environments (sparse Rewards, grosse Zustands- und Aktionsräume) benötigen viele Algorithmen sehr viele Trainingsschritte bis sie zu einer zufriedenstellenden Lösung konvergieren. Eine weitere Möglichkeit um dieses Problem zu beheben oder zumindest einzudämmen, ist Curriculum Learning (Bengio et al., 2009). Dabei werden dem Agenten während dem Training stufenweise komplexere Environments gegeben.

### 2.7.4 Self-Play und Tutoring

Das Erlernen eines neuen Games oder das Meistern einer neuen Fähigkeit erfolgt am effizientesten in der passenden Lernumgebung. Im Bereich von FPS Games besteht eine passende Lernumgebung aus einem Game mit einem Gegenspieler, der in etwa das gleiche Niveau besitzt wie der zu trainierende RL Agent. Eine Möglichkeit diese Lernumgebung zu realisieren, ist, den RL Agent entweder gegen sich selber, gegen eine ältere Instanz von sich oder gegen einen weiteren RL Agenten spielen zu lassen. Diese Trainingsmethode wird in der Literatur als Self-Play bezeichnet und ist im Bereich von RL in Games sehr beliebt. Allerdings bringt diese Trainingsmethode auch einen Nachteil mit sich. Gemäss Wiering et al. (2012) besteht das Hauptproblem darin, dass mit nur einem Gegenspieler das Environment zu wenig exploriert wird und somit der Reward nicht maximiert werden kann. Aus diesem Grund sollte der Agent gegen mehrere Instanzen (am besten mit unterschiedlicher Stärke) von sich selber spielen.

## 2.8 Related Work

Die Visual Doom AI Competition hat eine grosse Anzahl an Interessenten gefunden. Zahlreiche wissenschaftliche wie auch praktische Arbeiten wurden in der Literaturrecherche im Bereich des Erstellens und Trainierens eines intelligenten Agenten in Doom gefunden. Die Entwickler von ViZDoom (Kempka et al., 2016) haben in ihrer Arbeit gezeigt, dass einfache Szenarien in Doom mithilfe von DQN gelernt und sogar menschenähnliches Verhalten erreicht werden kann.

Der Wettbewerb ist in zwei unabhängige Aufgaben unterteilt. Die erste Wettbewerbsaufgabe (Track 1) besteht im Wesentlichen darin, einen Agenten für ein Deathmatch Game in einer bekannten Umgebung (Map) zu entwickeln. Ein Deathmatch ist ein Spielmodus in einem FPS Game. Das Ziel in diesem Spielmodus besteht darin, möglichst viele Gegenspieler in einer bestimmten Zeit zu eliminieren. Dabei werden getötete Spieler automatisch bewaffnet wiederbelebt. In der zweiten Wettbewerbsaufgabe (Track 2) treten die trainierten Agenten ebenfalls in einem Deathmatch Game gegeneinander an, allerdings ist die Umgebung unbekannt. Die Gewinner-Agenten in beiden Wettbewerbsaufgaben werden anhand der Anzahl Frags bestimmt. Die Anzahl Frags ist die Differenz zwischen den eliminierten Gegenspielern und den Selbstmorden. Unter den Teilnehmern des Wettbewerbs wurden mehrheitlich Agenten mithilfe von Deep Recurrent Q-Learning (DRQN), DQN und A3C trainiert. DRQN sind Deep-Q-Networks mit einer zusätzlichen RNN Schicht. Die Architekturen der Gewinner werden in folgenden Abschnitten kurz beschrieben.

Eine ausführliche Beschreibung des Trainierens eines Agenten für Doom ist in der Arbeit von Lample et al. (2017) gegeben. Die Autoren vertreten in ihrer Arbeit die These, dass zwei separate Netzwerke, ein Netzwerk für die Navigation und eines für die Aktionen, die Trainingszeit reduzieren und die Performance, im Vergleich zu einem herkömmlichen DRQN, steigern können. Für das Navigationsnetzwerk wurde ein simples DQN verwendet. Für das Aktionsnetzwerk wurden zusätzliche Informationen (beispielsweise Informationen, ob ein Gegenspieler im Frame sichtbar ist) während der Trainingsphase mitgegeben. Im Vergleich zu den herkömmlichen DRQN zeigten die Autoren, dass mit ihrer Architektur das Lernen mit überwachten Hilfstasks (Supervised Auxilary Tasks) zwar beschleunigt werden kann, allerdings das Trainingsverfahren erschwert wird. Während dem Training müssen diese Informationen zur Verfügung stehen. Die Autoren erreichten mit ihrem Agenten den zweiten Platz im Wettbewerb von 2016 (Track 1).

Die Gewinner des Wettbewerbs von 2016 (Track 1) setzten auf den Actor Critic Algorithmus A3C und ein spezielles Transfer Learning Methode (Wu et al., 2016). Die Autoren vertraten die Ansicht, dass die Agenten Menschen imitieren sollten und so wurde nur der visuelle Input (aktuelle Frames als Pixelmatrix) bereitgestellt und Curriculum Learning angewendet.

Die Gewinner der zweiten Wettbewerbsaufgabe verwendeten einen anderen Ansatz, Direct Future Prediction (DFP), der DQN, DRQN und A3C übertreffen konnte (Dosovitskiy et al., 2016). Die Autoren haben gezeigt, wie Supervised Learning Methoden verwendet werden können, um das Agieren in komplexen 3D Environments zu lernen.

# Kapitel 3

# Methodik

## 3.1 Problemanalyse

Das Reinforcement Learning Problem der vorliegenden Arbeit besteht aus dem Erstellen eines Agenten für das FPS Game Doom. Doom ist ein First-Person-Shooter (FPS) Game aus den 1990er Jahren. Im Wesentlichen bestehen die Herausforderungen darin, ein Agent in einer 3D-Umgebung zu entwickeln. Im Gegensatz zu den Atari 2600 Games (beispielsweise Pong) benötigt der Agent weit mehr Fähigkeiten, um die Aufgabe zu bewältigen. Der Agent muss unter anderem in der Lage sein, sich durch eine Map zu navigieren, Items zu sammeln und gleichzeitig Gegenspieler zu erkennen und zu eliminieren.

### 3.1.1 Das Doom Environment

Im Doom Environment beobachtet der Agent in jedem Zeitschritt den aktuellen Screen oder Frame als Pixelmatrix. Der Agent erhält vom ViZDoom Environment somit eine 2D-Projektion der Umgebung aus seiner aktuellen Position und aus seiner aktuellen Perspektive. Zusätzlich zu dem Frame stellt das ViZDoom Environment weitere Informationen, wie beispielsweise die aktuelle Anzahl der Munition, den aktuellen Gesundheitsstand (Health Points) und weitere spezifische Informationen zur Verfügung.

#### Zustandsraum

Der Zustandsraum von Doom wird somit aus den möglichen Frames und den Wertebereichen der Gameinformationen aufgespannt.

- **Frames:**
  - dreidimensionaler Vektor (Screen Breite, Screen Höhe, Farbkanal)
- **Gameinformationen:**
  - eindimensionaler Vektor (Anzahl Munition, Anzahl Lebenspunkte, Anzahl Frags etc.)

### Aktionsraum

Der Aktionsraum in Doom besteht aus üblichen Aktionen beziehungsweise Tastenknöpfen eines FPS-Games. Insgesamt beinhaltet der Aktionsraum 43 diskrete und kontinuierliche Aktionen[1]. Der Aktionsraum kann pro Map eingeschränkt werden, diskrete und kontinuierliche Aktionen können miteinander verwendet werden und die Aktionen lassen sich miteinander kombinieren. Es ist möglich, dass man gleichzeitig vorwärts läuft und einen Schuss abgibt (Kempka et al., 2016).

- **Diskrete Aktionen**: Stellen einen Tastendruck dar:
  - Wertebereich: $\{0, 1\}$
  - Beispiel: $A = \{ATTACK, MOVE\_LEFT, MOVE\_RIGHT\}$
- **Kontinuierliche Aktionen**: Stellen Mausbewegungen dar:
  - Wertebereich: $[-100, 100]$, wobei bei einem Wert von 0 keine Bewegung durchgeführt wird.
  - Beispiel: $A = \{TURN\_LEFT\_RIGHT\_DELTA\}$

Zusammengefasst kann das ViZDoom Environment als ein nicht stationäres, Multi-Agent und nicht vollständig beobachtbares Environment mit episodischem Zeithorizont betrachtet werden.

### Beispiel: Agent-Environment-Interaktion in ViZDoom

Der Agent erhält vom ViZDoom Environment den aktuellen Frame und kann nun eine zulässige Aktion auswählen (z.b. ATTACK) und ausführen. Dies führt dazu, dass das ViZDoom Environment das Spiel einen Zeitschritt vorrückt. Falls nun der Agent einen Feind getroffen und eliminiert hat, wird er vom Environment mit einem positiven Reward (+1) belohnt, andernfalls erhält er einen neutralen Reward (0). Danach wiederholt sich der Ablauf. Der Agent erhält vom ViZDoom Environment wieder eine neue Beobachtung und hat wieder die Möglichkeit, eine Aktion auszuwählen und auszuführen.

[1] `https://github.com/mwydmuch/ViZDoom/blob/master/doc/Types.md`

## 3.2 Entwicklung von Lösungsansätze

Es existieren unterschiedliche Ansätze, um die Trainingszeit (Lernzeit) von Reinforcement Learning Agenten zu verbessern. Gemäss Tian (2017) können neue oder verbesserte Algorithmen entwickelt oder Trainingsmethoden optimiert werden.

Die Literaturrecherche hat gezeigt, dass alle entwickelten Agenten in ihren Modellen auf statische Frame Skips (vgl. Abschnitt 2.7.1) setzen. Das Training kann dadurch beschleunigt und stabilisiert werden, da weniger Entscheidungen getroffen werden müssen und der Agent weniger durch bedeutungslose Zustände irritiert werden kann. Beispielsweise bewegt sich im Atari 2600 Game Pong der Ball hin und her und ist dabei in einigen Frames nicht zu sehen. Diese Frames können das Lernen negativ beeinflussen. Unterschiedliche Wiederholungsraten von Aktionen (Frame Skips) wurden von Kempka et al. (2016) in Doom untersucht. Die Untersuchung beschränkt sich allerdings auf verschiedene konstante Wiederholungsraten (Für wie viele Frames eine Aktion wiederholt wird, ändert sich während dem Spiel nicht). Einen wesentlichen Unterschied zwischen einem menschlichen Spieler und einem RL Agenten besteht in der Anzahl Entscheidungen (Aktionen), die während dem Spiel getroffen werden. Je nach Situation wird von einem Menschen eine Aktion unterschiedlich lang durchgeführt beziehungsweise die Taste wird länger oder weniger lang gedrückt. Die Anzahl der Wiederholungen ist somit abhängig vom aktuellen Zustand des Games und ist dynamisch. Wenn sich beispielsweise ein Spieler entscheidet, Munition einzusammeln, die am Ende eines Korridors liegt, und stünden keine Hindernisse im Weg, dann würde ein menschlicher Spieler die Pfeiltaste nach oben drücken und solange geradeaus laufen, bis er die Munition einsammeln könnte. Ein Mensch würde in diesem Fall mit einer einzigen Entscheidung auskommen. Ein RL Agent hingegen würde in einem periodischen Intervall (beispielsweise in jedem vierten Frame) eine neue Entscheidung treffen.

Die Literaturrecherche hat zudem gezeigt, dass die entwickelten Agenten unterschiedliche Architekturen und Trainingsmethoden verwenden. Die Gewinner des Wettbewerbs von 2016 (Track 1) verwendeten beispielsweise die Trainingsmethode Curriculum Learning (vgl. Abschnitt 2.7.3) und trainierten ihren Agenten über mehrere Tage bis sie eine zufriedenstellende Lösung hatten (Wu et al., 2016). Neben den langen Trainingszeiten besteht der Nachteil dieser Trainingsmethode darin, dass manuell ein passendes Curriculum erstellt werden muss. Eine für Games alternative Trainingsmethode ist Self-Play (vgl. Abschnitt 2.7.4). Self-Play wurde kürzlich bei Silver et al. (2016) und bei Sukhbaatar et al. (2017) erfolgreich angewendet. Bei der Self-Play Trainingsmethode kann der Aufwand für das Erstellen eines Curriculums gespart werden, da die Lernumgebung (beispielsweise Stärke des Gegners) im Laufe des Trainings komplexer wird.

Das Trainieren eines RL Agenten für komplexere Szenarien in Doom ist aufwendig, benötigt viel Rechenleistung beziehungsweise Rechenzeit und Hyperparameter Tuning. Um eine mögliche Verbesserung erzielen zu können, sollen in der vorliegenden Arbeit aus diesem Grund folgende zwei Lösungsansätze experimentell untersucht werden:

- **Lösungsansatz 1:** Verbesserung des Algorithmus indem dynamische Wiederholungsraten verwendet werden.
- **Lösungsansatz 2:** Den Einsatz von Self-Play als Trainingsmethode.

## 3.3 Lösungsansatz 1

Eine Möglichkeit, um dynamisch die Anzahl Wiederholungen einer ausgewählten Aktion zu bestimmen, bietet das Framework Fine Grained Action Repetition (FiGAR) von (Sharma et al., 2017) an. FiGAR ist ein generisches Framework für DRL Algorithmen, die auf einer Policy Funktion basieren, wie beispielsweise A3C (Mnih et al., 2016), DDPG (Lilli- crap et al., 2015) oder TRPO (Schulman et al., 2015).

Das Framework bietet die Möglichkeit, eine zeitliche Abstraktion in Form einer zeitlich erweiterten beziehungsweise verlängerten Aktion, einer sogenannten Marco-Action (Hauskrecht et al., 1998; McGovern et al., 1998), zu lernen. In FiGAR sind Macro-Actions für den Agenten Aktionen mit einer dynamischen Wiederholungsrate. Anstatt eine Aktion einmal auszuführen (d.h. einen Zeitschritt lang), wird die gleiche Aktion mehrere Zeitschritte lang ausgeführt. Während dieser Wiederholung besucht der Agent zwar zeitweise verschiedene Zustände, kann aber keine neue Aktion ausführen. Dennoch werden die erhaltenen Rewards in den besuchten Zuständen kumuliert.

### 3.3.1 FiGAR für Actor Critics Algorithmen

FiGAR erweitert Actor Critic Algorithmen, indem zusätzlich die Wiederholungsrate vorhergesagt wird. Neben der Wahrscheinlichkeitsverteilung für die Aktionen, den Value für den aktuellen Zustand, wird zusätzlich eine Wahrscheinlichkeitsverteilung für die Wiederholungsrate gelernt und vorhersagt (vgl. Abbildung 3.1). Diese Erweiterung lässt sich am besten anhand der Vorgehensweise des Algorithmus beschreiben.

#### Vorgehensweise

Der Algorithmus erhält zum Zeitpunkt $t = 0$ den Initialzustand $s_0$ des Environments als Input. Für $s_0$ sagt der Algorithmus das Tuple $(a_0, x_0)$ voraus. Das Tuple besteht aus der Aktion $a_0$ und der Wiederholungsrate $x_0$. Dabei wird die Aktion $a_0$ anhand der Policy $\pi_{\theta_a}(s_0)$ bestimmt und die Wiederholungsrate anhand der Policy $\pi_{\theta_x}(s_0)$. Die gleiche Aktion $a_0$ wird $x_0$ Zeitschritte lang ausgeführt. Eine solche Vorhersage (Tuple) wird als eine Aktionsentscheidung bezeichnet.

Nach $x_0$ Zeitschritte trifft der Algorithmus, basierend auf dem aktuellen Zustand $s_1$, eine neue Entscheidung und gibt das Tuple $(\pi_{\theta_a}(s_1), \pi_{\theta_x}(s_1))$ aus. Der Zustand $s_1$ bezeichnet den Zustand, nachdem der Agent eine Aktionsentscheidungen getroffen hat. Allgemein bezeichnet der Zustand $s_j$ den Zustand nach $j$ solcher Aktionsentscheidungen. Die Häufigkeit der Entscheidungen ist abhängig von der Vorhersage der Wiederholungsrate im aktuellen Zustand. Eine Entscheidung wird jeweils zum Zeitpunkt $t = \sum_{i=0}^{k} x_i$ anhand des Zustands $s_{k+1}$ getroffen.

**Beispiel**

Falls der Algorithmus für den Initialzustand das Tuple (MoveForward, 6) voraussagen würde, würde der Agent vom Zeitpunkt $t = 0$ bis zum Zeitpunkt $t = 6$ die Aktion *MOVE_FORWARD*, also vorwärts bewegen, ausführen.

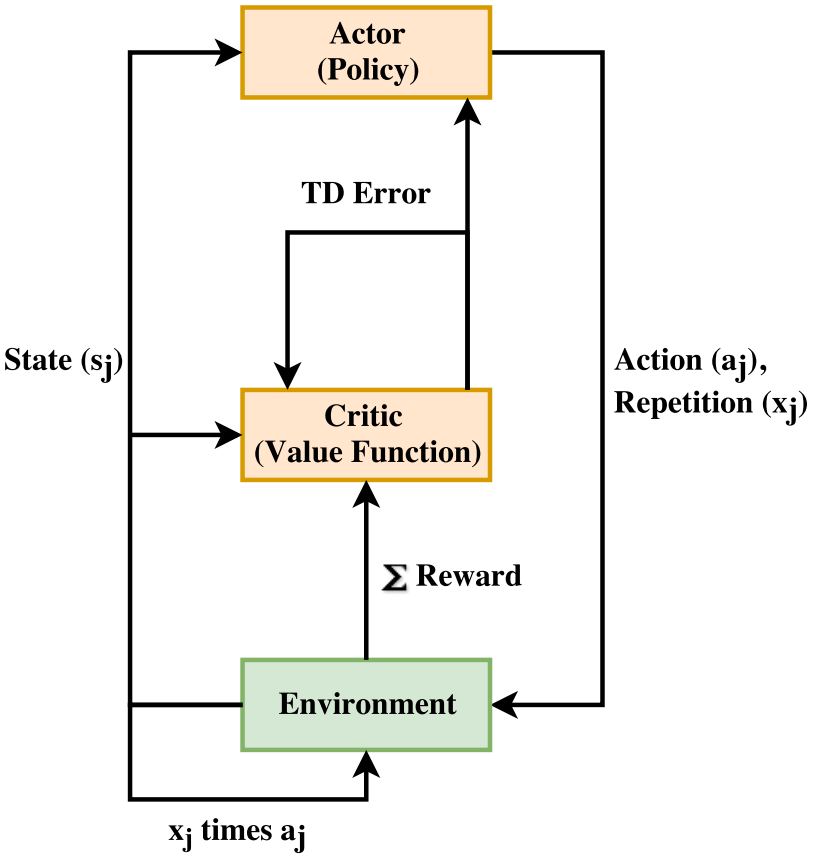



ABBILDUNG 3.1: Übersicht des FiGAR Frameworks für Actor Critic Algorithmen. Der Actor (Policy) erhält den Zustand $s_j$ vom Environment und wählt eine Aktion $a_j$ und eine Wiederholungsrate $x_j$ aus. Zur gleichen Zeit erhält die Critic (Value Funktion) ebenfalls den Zustand $s_j$ und die Summe der Rewards von der letzten Interaktion. Mithilfe dieser Informationen berechnet die Critic den TD Error und aktualisiert sich selber und den Actor (nach (Sutton et al., 1998)).

### Architektur

Die Architektur von FiGAR sieht vor, dass die Policy (Strategie) für die Wahl der entsprechenden Aktion von der Policy für die Wiederholungsrate einer Aktion entkoppelt wird. Der Agent hat somit die Möglichkeit, eine Aktion unabhängig von der Wiederholungsrate auszuwählen. Die Entscheidung, welche Aktion ausgeführt wird, ist unabhängig von der Wahl der Wiederholungsrate. Im Gegensatz zu Lakshminarayanan et al. (2017) und Vezhnevets et al. (2016) ermöglicht diese Eigenschaft, eine zeitliche Abstraktion zu finden, ohne den Aktionsraum unnötig auszuweiten. Diese Entkoppelung der Policies wird von den Autoren als Structured and Factored Representation of the Policy bezeichnet (vgl. Abbildung 3.2 ).

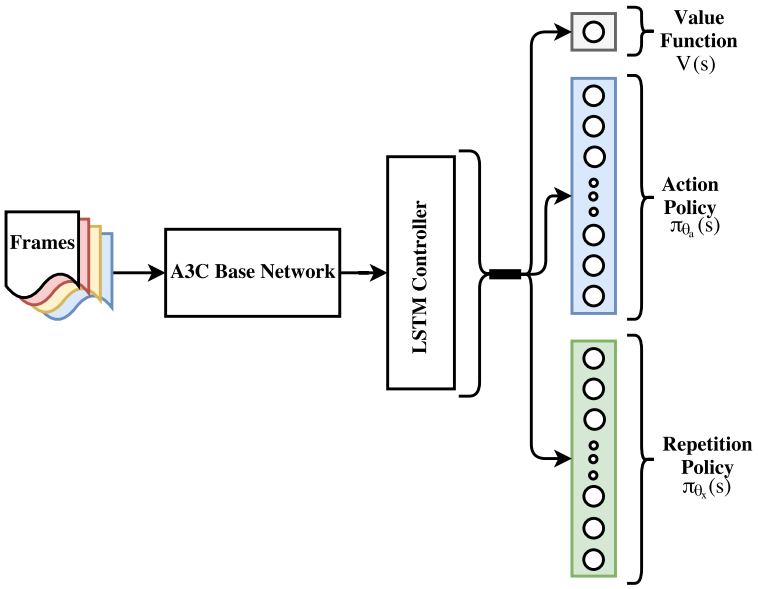



ABBILDUNG 3.2: FiGAR Architektur für A3C-LSTM. Die FiGAR Architektur basiert auf der A3C-LSTM Architektur von (Mnih et al., 2016) und beinhaltet zusätzlich einen weiteren Layer.

### Algorithmus

Die zentrale Idee des FiGAR Frameworks besteht darin, dass die Parameter der beiden Policys von einer Zielfunktion aktualisiert werden. Dies bedeutet, dass die Parameter $\theta_a$ von der Aktion Policy $\pi_a$ und die Parameter $\theta_x$ der Aktion Repetition Policy $\pi_x$ gleichzeitig gelernt werden. Im folgenden Pseudocode (Algorithmus 1) als $\mathcal{L}$ dargestellt.

**Algorithm 1** Create FiGAR - Z (Sharma et al., 2017)

**function** MAKEFIGAR(DRLAlgorithm $Z$, ActionRepetitionSet $W$)
  $s_t \leftarrow$ state at time $t$
  $a_t \leftarrow$ action taken in $s_t$ at time $t$
  $\pi_a \leftarrow$ action policy of $Z$
  $f_{\theta_a}(s_t) \leftarrow$ action network for realizing action policy $\pi_a$
  $\mathcal{L}(\pi_a, s_t, a_t) \leftarrow$ A's objective function for improving $\pi_a$
  $\pi_x \leftarrow$ construct action repetition policy for FiGAR-Z.
  $f_{\theta_x}(s_t) \leftarrow$ repetition network with output of size $|W|$ for action repetition policy $\pi_x$
  $\mathcal{L}(\pi_x, s_t, a_t) \leftarrow \mathcal{L}$ evaulated at $\pi_x$
  $T(s_t, a_t) \leftarrow \mathcal{L}(\pi_a, s_t, a_t) * L(\pi_a, s_t, a_t)$ ▷ Total Loss
  **return**

**FiGAR-A3C**

Das FiGAR Framework erweitert den A3C Algorithmus (vgl. Abschnitt 2.4.2) mit einem zusätzlichen neuronalen Netzwerk für die Vorhersage der Wiederholungsrate. In FiGAR-A3C wird neben der Policy für die Aktionen $\pi(a \mid s_j)$ und der Value Funktion $V(s_j)$ zusätzlich eine Policy für die Wiederholungsrate der Aktionen $\pi(x \mid s_j)$ gelernt. Der Output setzt sich aus drei Teilen zusammen: einem Vektor mit der Grösse des Aktionsraums, einem Skalar für die Value Funktion und einem Vektor mit der Grösse der möglichen Wiederholungsraten (vgl. Abbildung 3.2). Die Autoren haben die Zielfunktion des Actors wie folgt erweitert:

$$\begin{aligned} \mathcal{L}(\theta_a, \theta_x) &= \log f_{\theta_a}(a \mid s_j) \cdot A(s_j, a, x) + \log f_{\theta_x}(x \mid s_j) \cdot A(s_j, a, x) \\ &= \big(\log f_{\theta_a}(a \mid s_j) + \log f_{\theta_x}(x \mid s_j)\big) \cdot A(s_j, a, x) \; . \end{aligned} \tag{3.1}$$

$A(s_j, a, x)$ repräsentiert dabei die Advantage Funktion und gibt den (relativen) Vorteil für das Ausführen der Aktion $a$ für $x$ Zeitschritte im Zustand $s_j$ aus. Dieser wird berechnet, um die Varianz zu reduzieren (Mnih et al., 2016).

Die Zielfunktion für den Critic ist beinahe identisch mit der Zielfunktion von A3C (vgl. Gleichung 2.13). Allerdings ist die Schätzung der Value Funktion für die Zielwerte (Targets) abhängig von den Aktionsentscheidungen.

## 3.4 Lösungsansatz 2

Für den Einsatz der Trainingsmethode Self-Play werden zusätzliche Agenten im Environment benötigt. Reinforcement Learning mit mehreren Agenten wird als Multi-agent RL bezeichnet und stellt die Schnittstelle zwischen der Spieltheorie und dem Reinforcement Learning Forschungsgebiet dar. Multi-agent RL Probleme wurden bisher weit weniger untersucht und stellen nicht nur technisch eine grössere Herausforderung dar. Neben den herkömmlichen Problemen von RL (siehe Abschnitt 2.6) sind beispielsweise mehrere Gleichgewichtszustände (Multiple Equilibria) oder unklare Lernziele zu bewältigen. Folglich werden ein genaues Verständnis des zu lösenden Problems und genaue Evaluationskriterien benötigt (Li, 2017).

Die experimentelle Untersuchung der Self-Play Trainingsmethode soll anhand der Arbeit von Tampuu et al. (2017) durchgeführt werden. In dieser Arbeit wurde die Self-Play Trainingsmethode für das Atari 2600 Game Pong untersucht. Die Autoren konnten zeigen, wie durch unterschiedliche Rewardschemata kompetitive und kollaborative Verhaltensmuster hervorgebracht werden können. In ihrer Arbeit verwendeten sie für jeden Agenten unabhängige DQNs. Sie begründeten den Einsatz von unabhängigen Agenten (dezentrale Eigenschaft) mit der Möglichkeit, konsistente Resultate zu erzeugen und mit dem geringen Rechenaufwand. Zudem ist der Einsatz von unabhängigen Agenten die einfachste Methode, um ein Multi-Agent RL Problem zu lösen.

Das Ziel der Untersuchung ist in erster Linie herauszufinden, welche Auswirkungen eine geeignete Lernumgebung auf die Trainingszeit beziehungsweise auf die gelernte Strategie hat. Aus diesem Grund wird ebenfalls auf eine möglichst einfache Methode zurückgegriffen.

**Reward Schema für ein kompetitives Verhalten**

| | Player 1 punktet | Player 2 punktet |
|---|---|---|
| Reward für Player 1 | +1 | -1 |
| Reward für Player 2 | -1 | +1 |

TABELLE 3.1: Reward Schema für ein kompetitives Szenario mit zwei Spielern (nach (Tampuu et al., 2017)) .

# Kapitel 4

# Umsetzung

## 4.1 Software

Die Umsetzung der Lösungsansätze wurde mithilfe des Reinforcement Learning Research Frameworks Coach von Intel (Caspi et al., 2017) durchgeführt. Der modulare Aufbau von Coach ermöglicht es, einen Agenten durch Kombinationen von verschiedenen Bausteinen (beispielsweise Netzwerkschichten) aufzubauen, zu erweitern und ihn in verschiedenen Environments zu trainieren.

Das Framework ist in der Programmiersprache Python implementiert und ist eine ideale Plattform, um RL Algorithmen in verschiedenen Environments experimentell testen zu können. Zudem beinhaltet es eine Sammlung von State of the Art Reinforcement Learning Algorithmen und wichtige Debugging Informationen können visuell dargestellt werden.

#### Aufbau des Frameworks

Für alle RL Algorithmen werden neuronale Netze als Funktionsapproximatoren verwendet. Die neuronalen Netze der Agenten sind modular aufgebaut und bestehen im Wesentlichen aus den folgenden drei Komponenten:

- **Input Embedder:** Der Input Embedder ist die erste Komponente des Netzwerks und konvertiert den Input in eine geeignete Repräsentation. Das Framework stellt zwei Input Embedder zur Verfügung:
  - Image Embedder (CNN)
  - Vector Embedder (FCN)
- **Middleware:** Die Middleware ermöglicht eine zusätzliche Verarbeitung der Daten. Dabei können entweder die Outputs von verschiedenen Input Embedders kombiniert oder zusätzliche Schichten hinzugefügt werden (FCN, LSTM).
- **Output Heads:** Die Output Heads bilden die letzten Komponenten. Sie ermöglichen es, die benötigten Werte vorherzusagen (beispielsweise Q-Values, Values oder Policy). Es können mehrere Output Heads gleichzeitig verwendet werden. In den Output Heads sind die jeweiligen Loss-Funktionen definiert. Beispielsweise benutzen Actor Critic Agenten zwei Output Heads: einen Head für die Policy und einen Head für die Values.

Der Aufbau des Netzwerks ist in der Abbildung 4.1 dargestellt.

### 4.1.1 Preprocessing

Die Inputdaten des neuronalen Netzes bestehen aus den Beobachtungen (Frames) des Agenten. Für alle Experimente wird ViZDoom mit der kleinstmöglichen Auflösung (160 × 120 Pixels) konfiguriert. Zudem werden die Daten zuerst vorverarbeitet, um den Berechnungs- und Speicheraufwand zu reduzieren. Die Beobachtungen werden in Graustufenbilder konvertiert und anschliessend wird die Auflösung auf 76 × 60 Pixels verringert (Downsampling). Diese Vorgehensweise wurde von Intel Coach Framework vorgegeben und wurde beibehalten.

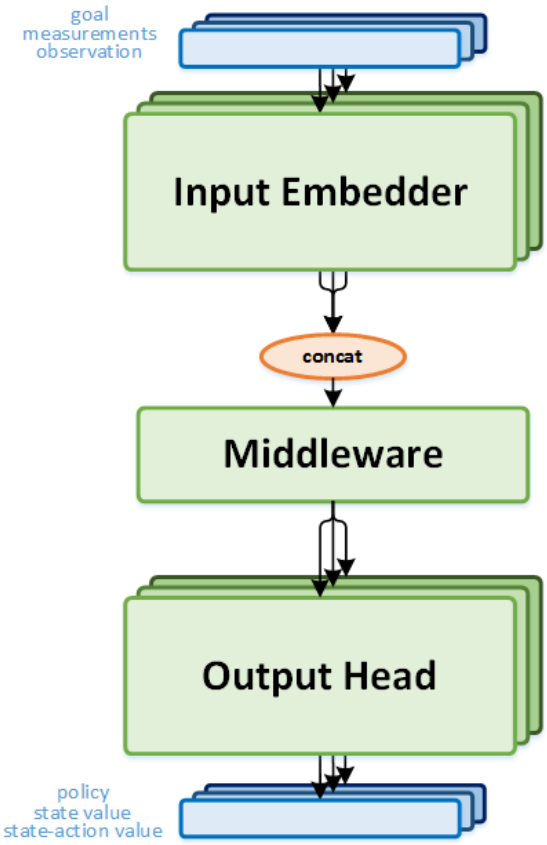



ABBILDUNG 4.1: Übersicht von Intel Coach (Caspi et al., 2017).

### 4.1.2 Experimentielles Setup

Die Experimente wurden mithilfe des GPU-Clusters[1] des Datalabs[2] durchgeführt. Der GPU-Cluster wird von mehreren Gruppen gleichzeitig benutzt und besteht aus vier Knoten (Server) mit je

- 2 × Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz 12 Cores
- 8 × Nvidia TITAN Xp

[1] `https://gpu-cluster.cloudlab.zhaw.ch/`
[2] `https://www.zhaw.ch/datalab`

### 4.1.3 Erfahrungen mit Intel Coach

Das Framework befand sich zu Beginn der Arbeiten noch im Anfangsstadium. Für die experimentelle Untersuchung mussten zuerst grundlegende Fehler behoben werden. Beispielsweise konnten trainierte Modelle nicht geladen werden und es konnte keine LSTM Schicht verwendet werden.

### 4.1.4 Doom Map Editor

Änderungen an den Szenarien (Maps) wurden mithilfe von SLADE3[3] durchgeführt. SLADE3 ist ein Editor für Games, die auf der Doom-Engine basieren. SLADE3 verfügt über einen grafischen Editor und bietet die Möglichkeit, neue Maps zu erstellen oder bestehende Maps zu bearbeiten.

[3] `http://slade.mancubus.net/index.php?page=news`

## 4.2 Umsetzung Lösungsansatz 1

Für die experimentelle Überprüfung des ersten Lösungsansatzes wird ein neuer Agent im Framework Coach erstellt. Im Wesentlichen wird der A3C Agent im Framework um einen neuen Output Head erweitert, zusätzlich werden die entsprechenden FiGAR-A3C Zielfunktionen implementiert. Für das Hinzufügen von eigenen Rewardfunktionen (Reward Shaping) und die Konfiguration des Environments wird der Doom Environment Wrapper von Intel Coach jeweils für die einzelnen Experimente den Bedürfnissen entsprechend angepasst.

### 4.2.1 Evaluation

Die Evaluation des ersten Lösungsansatzes wird anhand eines direkten Vergleiches mit dem A3C Algorithmus durchgeführt. In einem ersten Schritt werden die Experimente mit dem A3C Algorithmus durchgeführt und passende Parameter gesucht. Diese dienen als Baseline für den direkten Vergleich. In einem zweiten Schritt werden die gleichen Experimente mit denselben Parametern mit dem FiGAR-A3C Algorithmus durchgeführt. Der Vergleich wird mit den erhaltenen Rewards und anhand einer manuellen Videoanalyse durchgeführt.

Für die Überprüfung der Performance der Algorithmen werden die durchschnittlichen erzielten Rewards eines menschlichen Spielers als Richtwert genommen. In der Tabelle 4.1 ist der durchschnittliche Reward von zehn Episoden aufgeführt. Die Szenarien werden im Abschnitt 4.2.2 vorgestellt.

| **Scenario** | **Richtwert (Reward)** |
|---|---|
| Basic | 85.5 |
| Health Gathering | 2100 |
| Deadly Corridor | 2182.2 |

TABELLE 4.1: Durchschnittlicher Reward eines menschlichen Spielers.

### 4.2.2 Experimente

Das Ziel der Experimente des ersten Lösungsansatzes ist es, zu überprüfen, ob die Erweiterung des Algorithmus die Trainingszeit verkürzen kann.

Zu diesem Zweck werden die Experimente in zwei Kategorien unterteilt. In der ersten Kategorie befinden sich Experimente, die zur Sicherstellung der Implementation dienen und spezifische Fähigkeiten (Angriff, Items sammeln, etc.) des Agenten überprüfen sollen. In der zweiten Kategorie werden die gesammelten Erfahrungen genutzt, um einen Agenten für die erste Wettbewerbsaufgabe aus dem Jahr 2017 zu trainieren.

Die Experimente werden mithilfe der Standardszenarien[4] (Maps) von VizDoom und mit der Map der ersten Wettbewerbsaufgabe von 2017 durchgeführt. ViZDoom verfügt über diverse Standardszenarien mit vorgegebenen Rewardfunktionen.

[4] `https://github.com/mwydmuch/ViZDoom/blob/master/scenarios/README.md`

**Experiment 1: Überprüfung der Implementation**

Der implementierte Agent und die Funktionsweise des Environments werden anhand des einfachen Basic Scenarios überprüft. In diesem ersten Szenario befindet sich der Agent in einem rechteckigen Raum mit grauen Wänden, grauer Decke und grauem Boden. Der Agent ist mit einer Pistole bewaffnet und steht an der Wand. Der Agent kann sich in diesem Szenario nur nach links und nach rechts bewegen. Auf der gegenüberliegenden Seite befindet sich ein Monster. Das Ziel des Szenarios besteht darin, das Monster zu eliminieren. Ein Spiel (Episode) ist beendet, wenn das Monster eliminiert wurde oder die Spielzeit abgelaufen ist. Das Basic Scenario ist in der Abbildung 4.2 dargestellt.

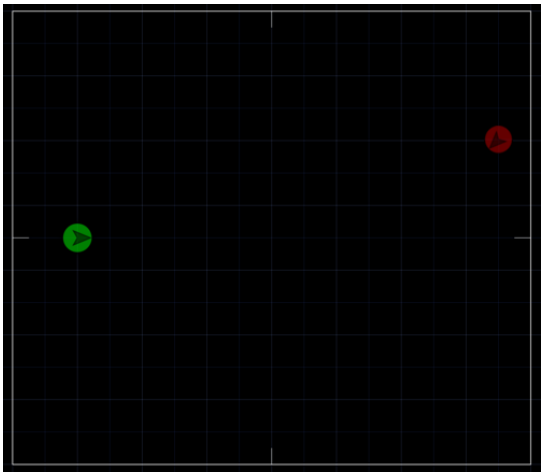


ABBILDUNG 4.2: Grundriss Basic Map.

| | **Beschreibung** |
|---|---|
| **Rewardfunktion:** | $+101$ für das Eliminieren des Gegners (Monster)<br>$-5$ für einen Fehlschluss<br>$-1$ fürs am Leben bleiben (Living Reward pro Zeitschritt) |
| **Zustandsraum:** | Screen als Pixelmatrix |
| **Aktionsraum:** | $\{MOVE\_LEFT, MOVE\_RIGHT, ATTACK\}$ |
| **Spielzeit:** | ca. 8.5 s (300 Zeitschritte bei 35 FPS) |

TABELLE 4.2: Die Konfiguration des ersten Experiments.

**Experiment 2: Health Gathering (Items einsammeln)**

Im zweiten Szenario befindet sich der Agent ebenfalls in einem rechteckigen Raum. Allerdings ist der Boden grün und mit einer säurehaltigen Substanz bedeckt. Diese säurehaltige Substanz fügt dem Agenten periodisch einen kleinen Schaden zu. Um in diesem Szenario zu überleben, muss der Agent Healthpacks einsammeln. Diese sind pseudozufällig über die ganze Map verstreut und regenerieren den Agenten. Das Ziel besteht darin, möglichst lange zu überleben. In diesem Szenario soll der Agent lernen, Healthpacks einzusammeln.

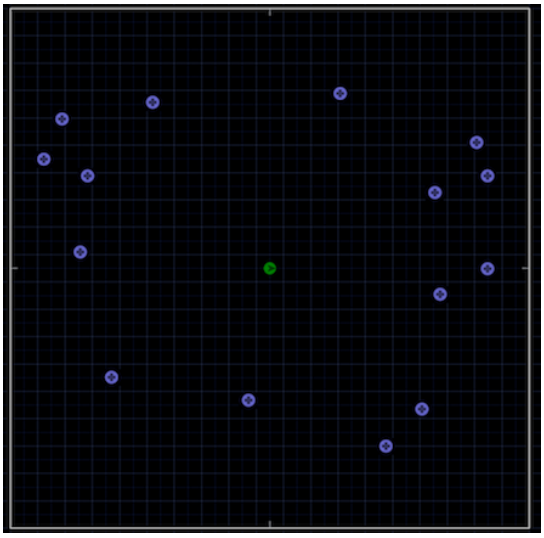


ABBILDUNG 4.3: Grundriss Health Gathering Map.

| | **Beschreibung** |
|---|---|
| **Rewardfunktion:** | $+1$ fürs am Leben bleiben (Living Reward pro Zeitschritt)<br>$-100$ für das Sterben |
| **Zustandsraum:** | Screen als Pixelmatrix<br>Gameinformation (Health) |
| **Aktionsraum:** | $\{TURN\_LEFT, TURN\_RIGHT, MOVE\_FORWARD\}$ |
| **Spielzeit:** | 60 s (2100 Zeitschritte bei 35 FPS) |

TABELLE 4.3: Die Konfiguration des zweiten Experiments.

**Experiment 3: Deadly Corridor (Ziel erreichen)**

Im dritten Szenario befindet sich der Agent, bewaffnet mit einer Pistole, in einem Korridor. Das Ziel in diesem Szenario besteht darin, den Agenten so zu trainieren, dass er bis ans Ende des Korridors gelangt und eine Weste einsammelt.

Auf beiden Seiten des Korridors befinden sich Monster, die den Agenten angreifen können. Um an die Weste zu gelangen, muss der Agent die Monster eliminieren. Der Reward in diesem Szenario ist proportional zur Distanzänderung zwischen dem Agenten und der Weste.

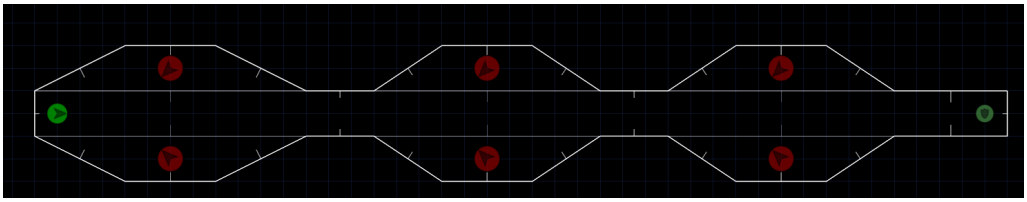


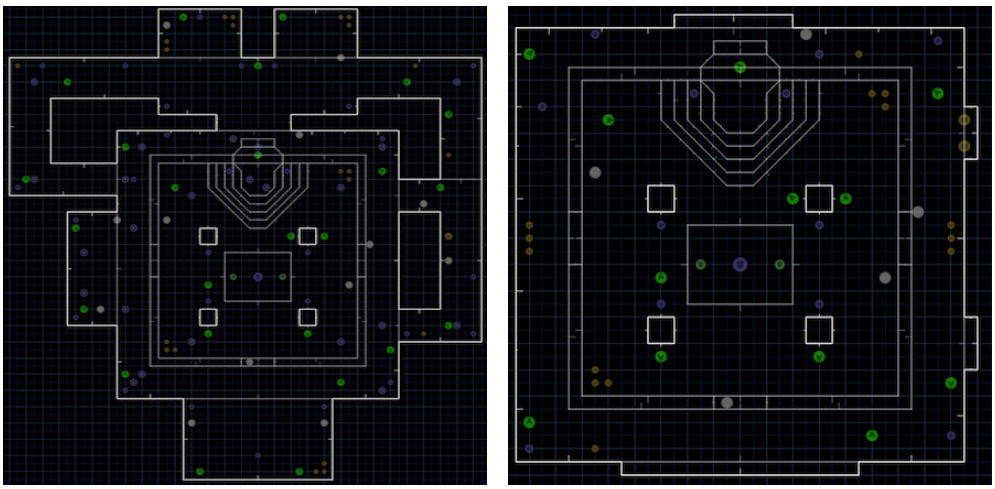
ABBILDUNG 4.4: Grundriss Deadly Corridor Map.

| | **Beschreibung** |
|---|---|
| **Rewardfunktion:** | Positiver Reward, wenn die Entfernung zur Weste sinkt.<br>Negativer Reward, wenn die Entfernung zur Weste steigt. |
| **Zustandsraum:** | Screen als Pixelmatrix<br>Gameinformation (Health) |
| **Aktionsraum:** | $\{TURN\_LEFT, TURN\_RIGHT, MOVE\_FORWARD$<br>$MOVE\_RIGHT, MOVE\_LEFT, ATTACK\}$ |
| **Spielzeit:** | 60 s (2100 Zeitschritte bei 35 FPS) |

TABELLE 4.4: Die Konfiguration des dritten Experiments.

**Experiment 4: Deatmatch (Track 1)**

Das vierte Szenario besteht aus der Wettbewerbsaufgabe von 2017. In diesem Deathmatch Szenario befindet sich der Agent mit acht weiteren Built-in Bots in einem mehrstöckigen Raum. Auf der ganze Map sind Healthpacks, Schutzschilder und Muntion verteilt. Alle Spieler kämpfen gegeneinander und sind mit einem Raketenwerfer bewaffnet. Das Ziel des Szenarios ist es, möglichst viele Gegner zu eliminieren und dabei selber nicht zu sterben. Der Agent kann entweder von den Gegenspieler getötet werden oder er kann sich mit dem Raketenwerfer selber verletzten und sterben.



(A) CIG 2017 Track 1 Map. (B) CIG 2017 Track 1 Map (verkleinert).

ABBILDUNG 4.5: Grundriss Deathmatch Map.

Im diesem Szenario erhält der Agent standartdmässig keinen Reward. Aus diesem Grund wird zuerst eine passende Reward Funktion implementiert. Zur Beschleunigung der Trainingszeit wird die Map verkleinert (vgl. Abbildung 4.5) und Reward Shaping eingesetzt (vgl. Abschnitt 2.7.2). Die Werte für das Reward Shaping werden von F1 Agent übernommen (Wu et al., 2016).

| | **Beschreibung** |
|---|---|
| **Rewardfunktion:** | $+1$ für das Eliminieren eines Gegners |
| | $-1$ für das Sterben |
| | $-0.008$ fürs am Leben bleiben |
| | $-0.04$ aufsammeln von Lebenspunkten |
| | $-0.05$ verlieren von Lebenspunkten |
| | $+0.15$ aufsammeln von Munition (Raketen) |
| | $-0.008$ aufbrauchen von Munition |
| **Zustandsraum:** | Screen als Pixelmatrix |
| | Gameinformation (FRAGCOUNT, HEALTH, AMMO5, ARMOR) |
| **Aktionsraum:** | $\{TURN\_LEFT, TURN\_RIGHT, MOVE\_FORWARD$ |
| | $MOVE\_RIGHT, MOVE\_LEFT, ATTACK$ |
| | $TURN\_LEFT\_RIGHT\_DELTA\}$ |
| **Spielzeit:** | 60 s (2100 Zeitschritte bei 35 FPS) |

TABELLE 4.5: Konfiguration des vierten Experiments.

## 4.3 Umsetzung Lösungsansatz 2:

Die Überprüfung des zweiten Lösungsansatzes erfolgt ebenfalls mit dem Reinforcement Learning Framework Coach. Der Doom Environment Wrapper von Intel Coach wird so angepasst, dass ein Deathmatch Game mit mehreren trainierbaren Agenten gestartet werden kann. Dies bedeutet, dass eine Agent gleichzeitig als Hostsystem dient. Alle Experimente werden auf dem GPU-Cluster (vgl. 4.1.2) durchgeführt. Auf dem GPU-Cluster können nur Experimente in Docker gestartet werden. Um ein Multiplayer-Spiel starten zu können, wird ein Netzwerk zwischen mehreren Containern aufgebaut.

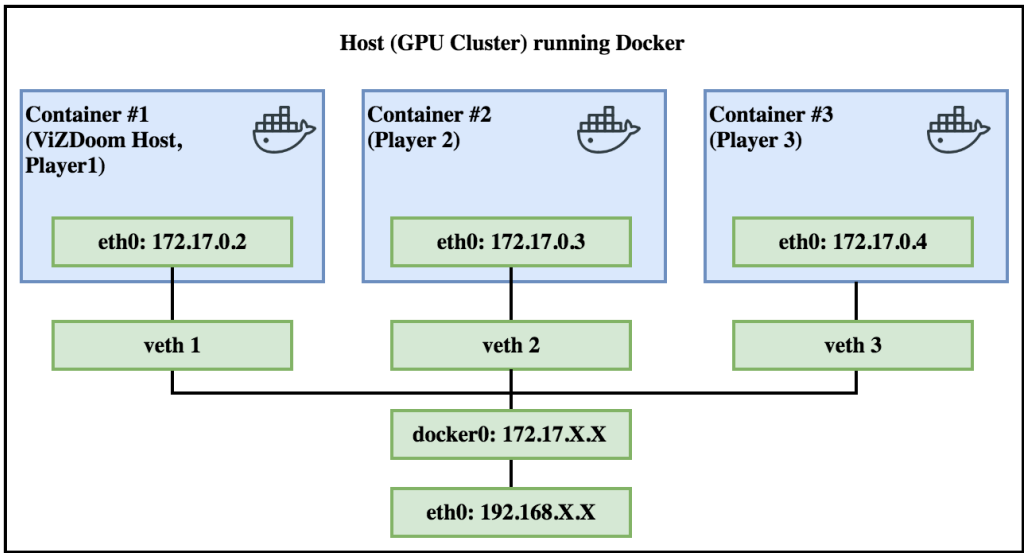



ABBILDUNG 4.6: Netzwerkübersicht der Docker Containers für die Self-Play Trainingsmethode (nach (Kereki, 2015)).

Beim Start eines Containers erstellt Docker standardmässig eine virtuelle Netzwerkschnittstelle mit einer eindeutigen Bezeichnung (vethX) und einer IP-Adresse auf dem Hostsystem. Die Container können mithilfe von Docker verlinkt werden und sind dann im gleichen Subnetz. In der Abbildung 4.6 ist die Netzwerkübersicht der Container dargestellt. Für ein Multiplayer-Spiel wird in einem Container (beispielsweise Player 1) das ViZDoom Environment als Host gestartet. In den anderen Containern wird das ViZDoom Environment so konfiguriert, dass es dem Spiel beitreten kann.

### 4.3.1 Evaluation

Die Evaluation erfolgt, indem die trainierten Agenten in zehn Episoden gegeneinander antreten und die Videos der besten Episode manuell analysiert werden.

### 4.3.2 Experimente

Das Ziel der Experimente besteht darin, zu überprüfen, ob mit der Self-Play Trainingsmethode geeignete Strategien gelernt werden können. Dies ist die Voraussetzung für einen direkten Vergleich mit den klassischen Trainingsmethoden (Agent spielt gegen Built-in Bots).

**Experiment 1: Duell**

Die Self-Play Trainingsmethode wird zuerst in einem Zweikampf getestet. Die beiden Agenten befinden sich in einem rechteckigen Raum und sind durch eine tödliche Substanz voneinander getrennt. Beide Agenten sind mit einer Schrotflinte (Shotgun) bewaffnet und können sich nur auf ihrer Plattform bewegen.

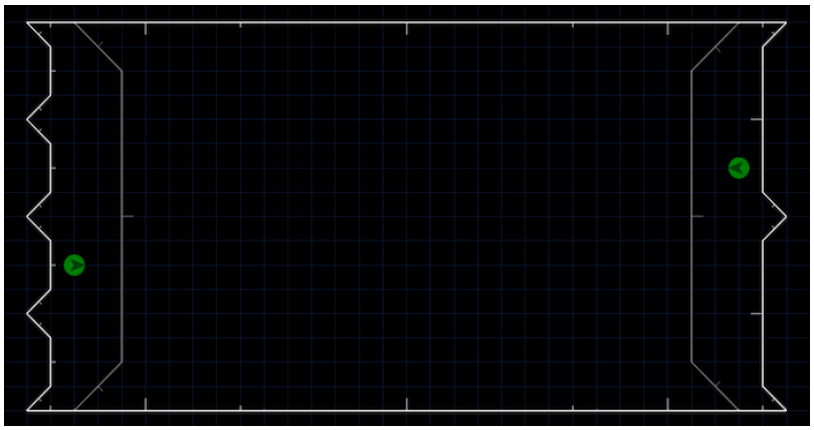


ABBILDUNG 4.7: Self-Play - Grundriss der Zweikampf Map.

Dieses Szenario ist ebenfalls Bestandteil der Standardszenarien von ViZDoom. Allerdings wurde für dieses Experiment die Bewaffnung geändert. Im Standardszenario erhalten die Agenten einen Raketenwerfer und benötigen nur einen Treffer, um den Gegenspieler zu eliminieren. Um sicherzustellen, dass der Agent bewusst den Gegner anvisiert und angreift, wird die Waffe geändert. Mit einer Schrotflinte benötigt der Agent mehrere Treffer, bis der Gegenspieler eliminiert ist.

| | **Beschreibung** |
|---|---|
| **Rewardfunktion:** | $+1$ für das Eliminieren des Gegenspielers<br>$-1$ für das Sterben |
| **Zustandsraum:** | Screen als Pixelmatrix |
| **Aktionsraum:** | $\{MOVE\_LEFT, MOVE\_RIGHT, ATTACK\}$ |
| **Spielzeit:** | 60 s (2100 Zeitschritte bei 35 FPS) |

TABELLE 4.6: Die Konfiguration des ersten Experiments.

**Experiment 2: Duell (erweitert)**

Das zweite Experiment basiert auf dem ersten Experiment. Der Aktionsraum wird mit $TURN\_LEFT, TURN\_RIGHT$ erweitert. Das Ziel des zweiten Experiments besteht darin, das Verhalten des Agenten bei einer kleinen Änderung des Aktionsraums zu beobachten.

**Experiment 3: Triell**

Die Self-Play Trainingsmethode soll in diesem Experiment in einem Triell getestet werden. Triell ist eine Variante des Duells mit drei Spielern, wobei jeder gegen jeden kämpft. Das Hauptproblem von Self-Play besteht darin, dass beide Agenten eine suboptimale Strategie lernen (vgl. Abschnitt 2.7.4). Um diesem Verhalten entgegenzuwirken, wird ein zusätzlicher Spieler hinzugefügt. Das Ziel dieses Experiments besteht darin, die Auswirkungen eines zusätzlichen Spielers zu beobachten.

Für dieses Experiment wird die Duell Map von Experiment 1 entsprechend angepasst, damit ein zusätzlicher Spieler das Spiel antreten kann. Zudem wird die tödliche Substanz entfernt und Munition sowie Healthpacks werden auf der Map verstreut. Alle Änderungen sind in der Abbildung 4.8 dargestellt.

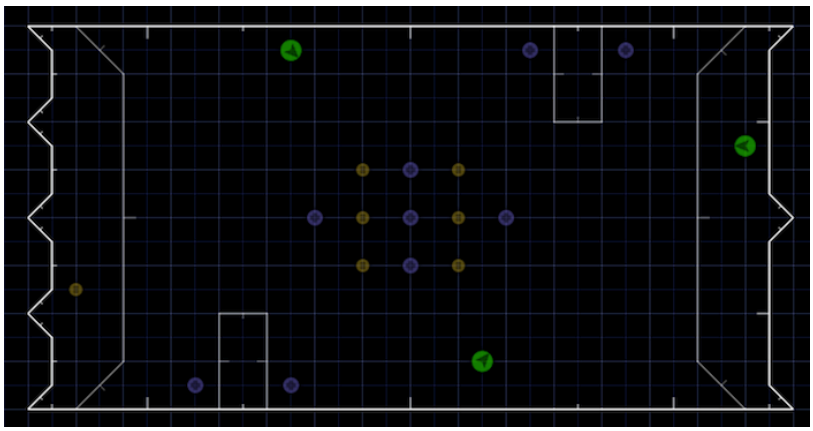


ABBILDUNG 4.8: Self-Play - Grundriss der 3 Spieler Map.

| | **Beschreibung** |
|---|---|
| **Rewardfunktion:** | $+1$ für das Eliminieren des Gegenspielers<br>$-1$ für das Sterben |
| **Zustandsraum:** | Screen als Pixelmatrix |
| **Aktionsraum:** | $\{MOVE\_LEFT, MOVE\_RIGHT, ATTACK,$<br>$TURN\_LEFT, TURN\_RIGHT,$<br>$MOVE\_FORWARD, MOVE\_BACKWARD\}$ |
| **Spielzeit:** | 60 s (2100 Zeitschritte bei 35 FPS) |

TABELLE 4.7: Die Konfiguration des dritten Experiments.

# Kapitel 5

# Resultate und Diskussion

Die Resultate beider Lösungsansätze werden in diesem Kapitel diskutiert. Neben der Analyse der Rewards werden die Charakteristiken der gelernten Policy anhand von Videos analysiert. Für die Auswertung der gelernten Policy wurde nach dem Training für jedes Szenario ein Video mit der besten Episode erstellt.

## 5.1 Resultate Lösungsansatz 1

Die Konfigurationen und Paramtern der Experimente des ersten Lösungsansatzes basieren auf den Erfahrungen von Mnih et al. (2016) und Kempka et al. (2016). Alle Experimente wurden mit 16 Learning Threads durchgeführt.

Im Allgemeinen alle wurden folgende Parameter verwendet:

- Optimizer: RMSProp with decay Parameter 0.99
- Lernrate: $\alpha = 10^{-5}$
- Diskontierungsfaktor: $\gamma = 0.99$
- Rewards wurde normalisiert: $[-1, +1]$ oder $[0, 1]$.

Die genauen Konfigurationen der Experimente sind im Anhang A.2 aufgelistet.

### 5.1.1 Resultat Experiment 1: Überprüfung der Implementation

In der Abbildung 5.1 ist der total erzielte Trainingsreward pro Episode des ersten Szenarios (Basic) dargestellt. Beide Algorithmen sind in unterschiedlicher Farbe (A3C blau, FiGAR orange) dargestellt und beide Algorithmen konvergieren bei rund 300 Episoden bei einem Reward von rund 95. In der Abbildung ist zudem ersichtlich, dass die Streuung der Rewards beim FiGAR-A3C Algorithmus in den ersten 300 Episoden grösser ist und länger andauert.

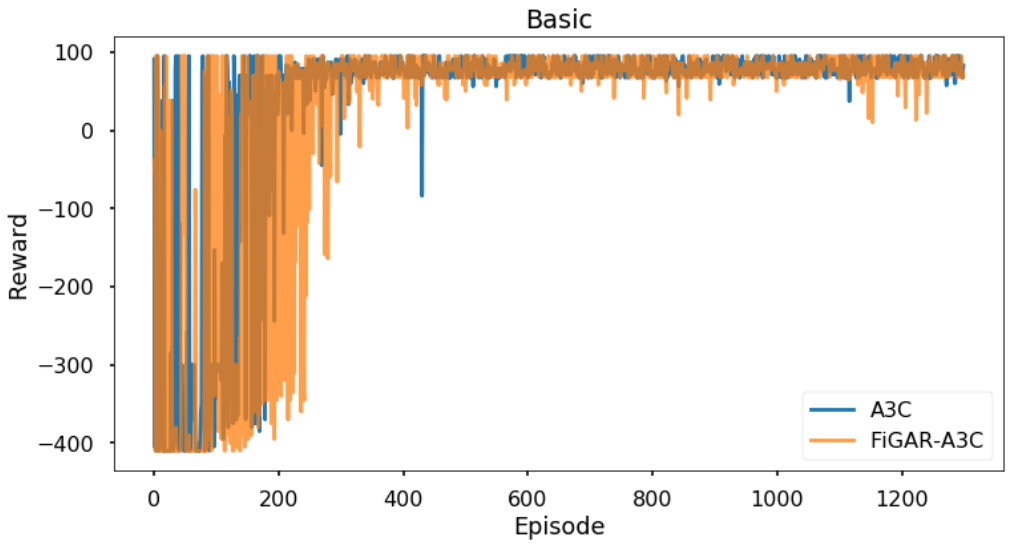



ABBILDUNG 5.1: Vergleich von A3C vs. FiGAR-A3C für das Szenario Basic.

#### Charakteristik der gelernten Strategie

Dieses Szenario stellt für beide Algorithmen kein Problem dar. Beide Agenten verfügen über ein ähnliches Verhalten. In diesem einfachen Szenario mit nur drei diskreten Aktionen lernt der Agent zuerst, möglichst viel zu schiessen und anschliessend, sich in Richtung des Gegners zu bewegen. Beide Agenten verfügen über ein ähnliches Verhalten.

### 5.1.2 Resultat Experiment 2: Health Gathering (Items einsammeln)

Im Health Gathering Szenario kann ein Reward von maximal 2100 erzielt werden, da die Spielzeit eine Minute beträgt und die Tickrate bei 35 Frames pro Sekunde eingestellt wurde. In der Abbildung 5.2 (links) sind die totalen Trainingsrewards pro Episode des zweiten Szenarios (Health Gathering) dargestellt. Es ist ersichtlich, dass der A3C Algorithmus in den ersten 250 Episoden praktisch keinen Reward erzielt hat. Nach 250 Episoden fängt der A3C Algorithmus rasant an zu lernen und konvergiert bei rund 1100 Episoden bei einem Reward von rund 2000. Im Gegensatz dazu, erzielt der FiGAR-A3C Algorithmus schon in den ersten 150 Episoden einen positiven Reward von über 500. Allerdings fällt der Reward nach rund 200 Episoden wieder und steigt nicht mehr an.

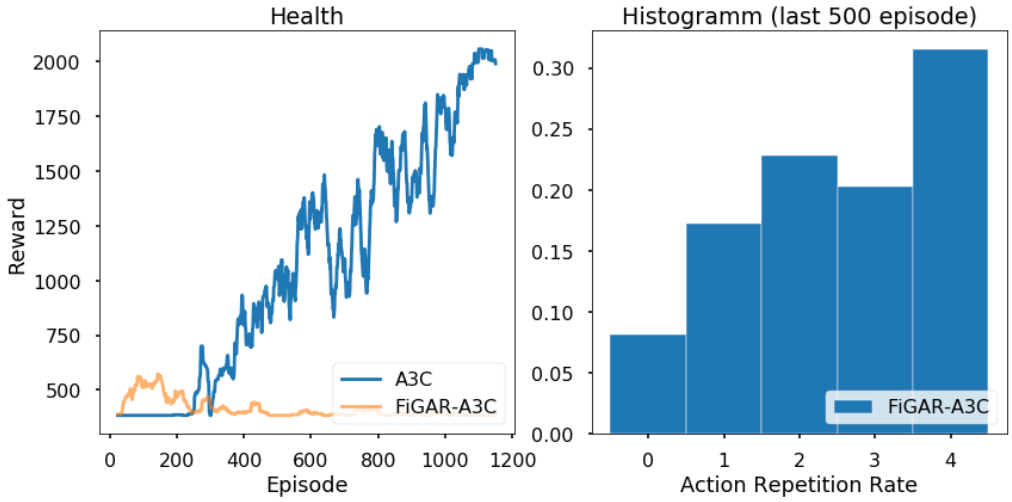



ABBILDUNG 5.2: Vergleich von A3C vs. FiGAR-A3C für das Scenario Health Gathering.

Für dieses Experiment wurde mit unterschiedlichen Wiederholungsraten experimentiert, welche jedoch keine grossen Auswirkungen auf das Endresultat hatten. Der Agent war nicht in der Lage eine optimale Strategie zu lernen. Für das in der Abbildung 5.2 dargestellte Ergebnisse wurde eine Wiederholungsrate von $[0, 4]$ mit einem Frameskip von 2 Frames verwendet. Die Aktionen konnten somit jeweils für $2, 4, 6, 8, 10$ Frames wiederholt werden. In der Abbildung 5.2 (rechts) ist das Histogramm der letzten 500 Episoden normiert dargestellt. Im Histogramm ist ersichtlich, dass in 30 Prozent der Fälle eine Aktion viermal wiederholt wird und somit 10 Frames lang ausgeführt wird.

#### Charakteristik der gelernten Strategie

In diesem Szenario lernt der A3C Agent, sich in Richtung der Healthpacks zu bewegen. Beinahe in jedem Frame entscheidet der Agent bewusst, sich in Richtung der Healthpacks zu bewegen. Der FiGAR-A3C Agent findet keine passende Strategie. Der Agent startet von der Mitte aus und entscheidet sich mit der maximalen Wiederholungsrate, in eine Richtung zu laufen. Da die Healthpacks um den Spieler verstreut sind, ist die Wahrscheinlichkeit gross, einige Healtpacks einzusammeln.

### 5.1.3 Resultat Experiment 3: Deadly Corridor

Die Resultate des dritten Experiments sind in der Abbildung 5.3 dargestellt. Das dritte Experiment wurde pro Algorithmus in zwei Varianten durchgeführt:

- A3C & A3C mit zusätzlicher LSTM Schicht
- FiGAR & FiGAR mit zusätzlicher LSTM Schicht

Das Deadly Corridor Szenario verfügt über eine wiederholende Struktur. Die Umgebung und die Gegner (zwei auf jeder Seite) wiederholen sich dreimal (vgl. Abbildung 4.4). Die zusätzliche LSTM Schicht wurde hinzugefügt, um diese Wiederholungen besser erkennen zu können (vgl. Abschnitt 2.7.1).

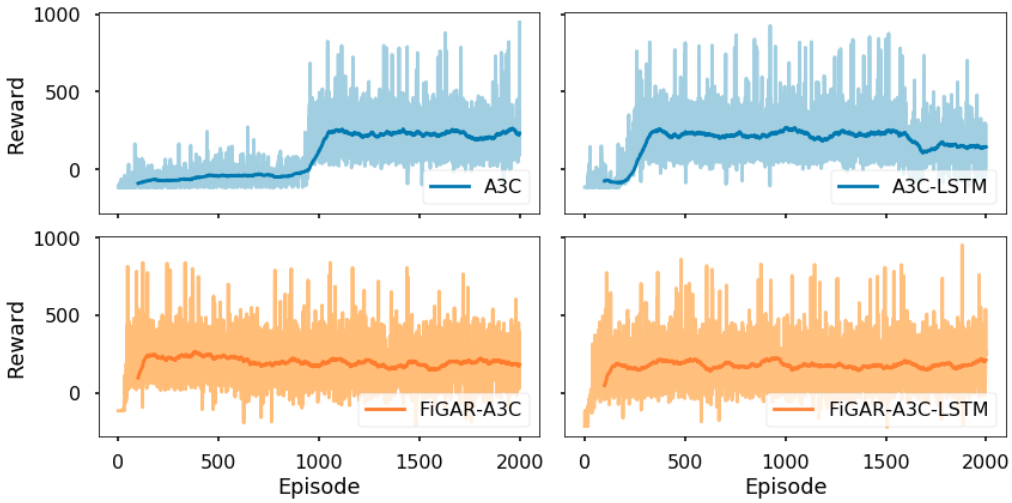



ABBILDUNG 5.3: Vergleich von A3C vs. FiGAR-A3C für das Szenario Deadly Corridor.

In der Abbildung 5.3 ist der durchschnittliche erzielte Reward jeweils als Linie dargestellt. Die durchschnittlichen Rewards der A3C Varianten sind blau dargestellt, diejenigen der der FiGAR-A3C Varianten orange.

In der Abbildung 5.3 ist erkennbar, dass der durchschnittliche Reward bei der A3C Variante bei rund 1000 Episoden sprungartig ansteigt und bei einem Wert von rund 300 bleibt. Im Vergleich dazu steigt der Reward bei der A3C-LSTM Variante schon bei rund 200 Episoden sprungartig an. Bei der A3C-LSTM Variante ist jedoch eine leichte Abnahme der durchschnittlichen Rewards erkennbar. Die Entwicklung des durchschnittlichen Rewards bei der FiGAR-A3C und bei der FiGAR-A3C-LSTM Variante sind sehr ähnlich. Der Reward steigt in den ersten 50 Episoden sprungartig an und bleibt danach bei einem Wert von rund 300. Beide Algorithmen können den Richtwert von Tabelle 4.1 nicht erreichen und bei beiden Algorithmen ist die Streuung der Rewards erheblich.

Die Resultate zeigen für den A3C Algorithmus, dass mithilfe der FiGAR Erweiterung ähnliche Effekte wie mit einer zusätzlichen LSTM Schicht erzielt werden können. Dies bestätigt die These von Braylan et al. (2015), dass Frame Skip geeignet ist, um in nicht vollständig beobachtbaren Environments eine Assoziation zwischen zeitlich verschobenen Zuständen und Aktionen zu lernen (vgl. Abschnitt 2.7.1).

### Charakteristik der gelernten Strategie

Die Videoanalyse zeigt deutlich, dass der Agent nicht in der Lage ist, bis zur Weste zu gelangen. Der Agent hat keine geeignete Strategie gelernt, um das Ziel zu erreichen. Allerdings sind bei beiden Algorithmen Ansätze erkennbar. Der Agent ist in der Lage, die ersten beiden Gegenspieler zu eliminieren und er bewegt sich langsam nach vorne.

### 5.1.4 Resultat Experiment 4: CIG 2017 Track 1 (verkleinert)

Im vierten Experiment wurden die Erfahrungen der anderen Experimente verwendet, um einen Agenten für das Deathmatch Szenario zu trainieren. Die Resultate des vierten Experiments sind in der Abbildung 5.4 dargestellt. In diesem Experiment wurde in jeder 500. Episode das Modell für 125 Episoden evaluiert. Zusätzlich ist der gleitende Mittelwert über 125 Episoden als Linie abgebildet.

In der Abbildung ist ersichtlich, dass beide Algorithmen einen negativen Reward erzielen und somit nicht in der Lage sind, eine sinnvolle Strategie zu lernen. Der Verlauf des durchschnittlichen Rewards beim A3C-LSTM Algorithmus schwankt zu Beginn zwischen $-18$ und $-22$. Nach der 6000. Episode nimmt er dann einen durchschnittlichen Wert von ca. $-18$ ein. Der Verlauf des durchschnittlichen Rewards beim FiGAR-A3C-LSTM Algorithmus ist zu Beginn bei ca. $-18$. Nach der 8000. Episode ist die Streuung massiv grösser und der durchschnittliche Reward fällt auf rund $-20$ ab.

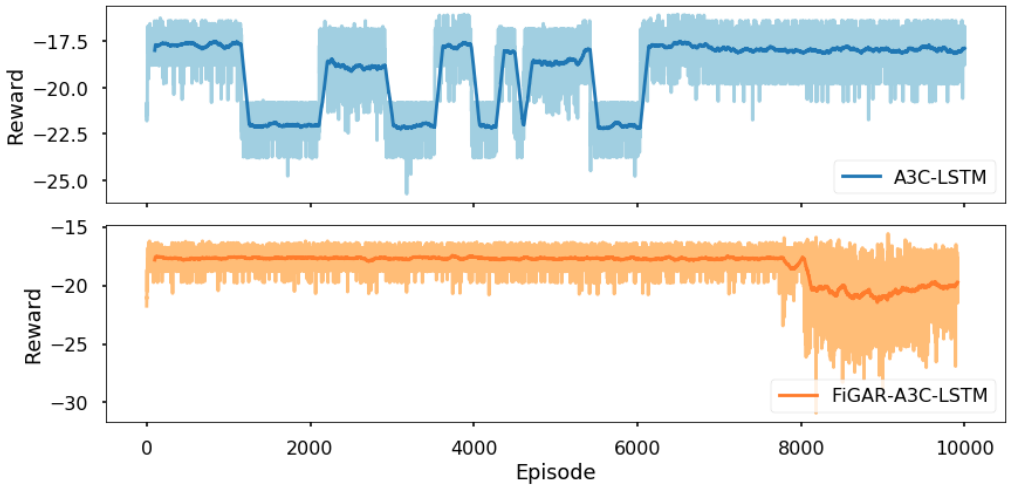



ABBILDUNG 5.4: Vergleich von A3C vs. FiGAR-A3C für das CIG 2017 Track 1 (verkleinert) Szenario.

### 5.1.5 Ursachenanalyse

Obwohl mit den gleichen Parametern und Rewardfunktionen und Netzwerkarchitekturen wie bei Ratcliffe et al. (2017) und Wu et al. (2016) experimentiert wurde, konnten beide Agenten nicht von Grund auf trainiert werden. Der Agent ist in der gegebenen Trainingszeit nicht in der Lage, eine erfolgreiche Strategie zu lernen.

**Mögliche Ursachen:**

- **Sparse Reward:** Obwohl die Map verkleinert wurde und der Agent trotzdem gegen 8 Built-in Bots antreten musste, erhält der Agent zu wenig positiven Reward. In der Abbildung 5.5 ist der maximale Reward pro Episode dargestellt sowie die maximale Anzahl Frags. Wie in der Abbildung zu sehen ist, trifft der Agent nur einmal einen Gegenspieler während dem Training, ohne dass er sich in derselben Episode selber tötet oder getötet wird.

- **Exploration:** In der Abbildung 5.6 sind die einzelne Loss-Kurven des A3C-LSTM Algorithmus dargestellt. Die Autoren des A3C addierten die Entropie der Policy zum Loss hinzu, um den Algorithmus davon abzuhalten, zu früh eine suboptimale Policy zu bevorzugen (Mnih et al., 2016). Die Entropy dient als Regularisierung. In der Abbildung ist erkennbar, dass Policy und Value Loss steigen und die Entropie abnimmt. Die Abnahme der Entropie hat zur Folge, dass der Agent weniger animiert wird zu explorieren, da eine Wahrscheinlichkeitsverteilung, die sich mehr auf einen Wert (z.B Aktion ATTACK) konzentriert, eine kleinere Entropie aufweist, als eine Wahrscheinlichkeitsverteilung, die beispielsweise gleich verteilt ist.

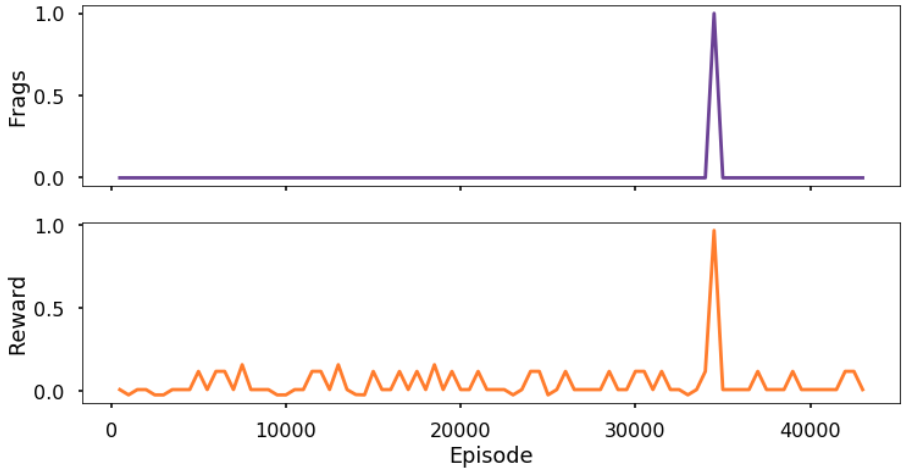



ABBILDUNG 5.5: Die maximale Anzahl Frags (violett) und der maximale Reward (orange) pro Episode des A3C-LSTM Algorithmus.

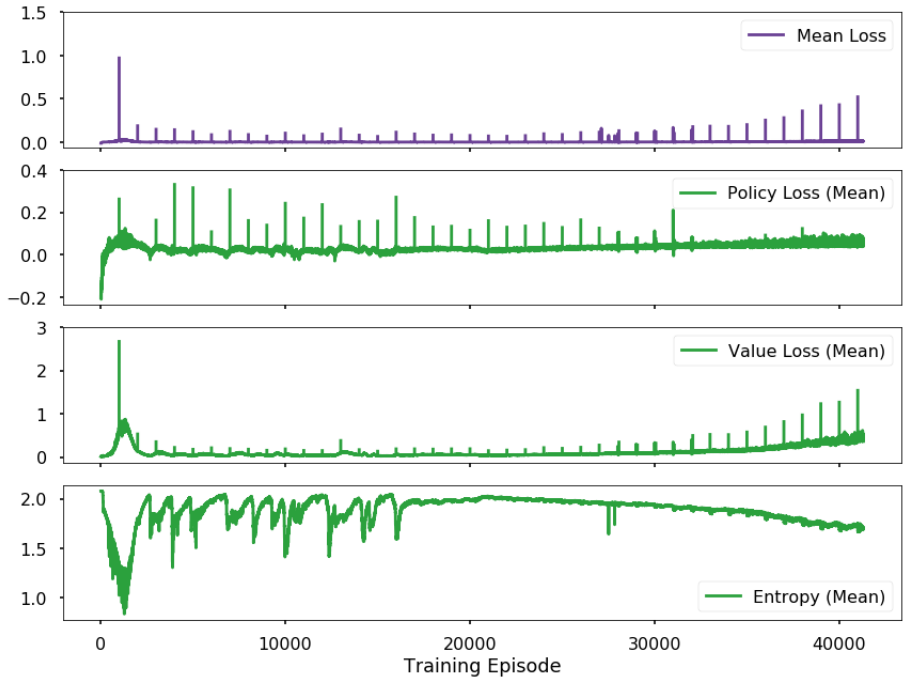



ABBILDUNG 5.6: Die Loss-Kurve des A3C-LSTM Algorithmus (violett). Diese besteht aus der gewichteten Summe des Policy- und Value-Losses sowie der Entropie (grün).

### 5.1.6 Diskussion Lösungsansatz 1

Die Resultate des ersten Lösungsansatzes haben gezeigt, dass selbst für die einfachen Standardszenarien das Trainieren der RL Agenten eine Herausforderung darstellt. Insbesondere das Trainieren eines Agenten ohne Transfer Learning Methoden einzusetzen (Training from Scratch).

Das Experimentieren und die Parametersuche waren aufgrund der langen Trainingszeiten sehr aufwendig. Folglich konnten nicht für alle durchgeführten Experimente passende Parameter gefunden werden. Selbst mit den Erfahrungen von bestehenden Lösungen konnten nicht alle Standardszenarien mit dem A3C Algorithmus erfolgreich trainiert werden. Eine Vielzahl weiterer Experimente wurden mit beiden Algorithmen erfolglos durchgeführt.

Die Resultate des ersten Lösungsansatzes zeigen jedoch Folgendes:

- In den durchgeführten Experimenten konnte beobachtet werden, dass der FiGAR-A3C Algorithmus früher einen höheren Reward als der A3C Algorithmus erzielt. Zudem konnten mit der FiGAR-A3C Variante ähnliche Resultate wie mit der LSTM Variante erzielt werden.
- Die durchgeführten Experimenten machten bewusst, wie empfindlich der A3C Algorithmus auf Parametereinstellungen ist und wie wichtig die Exploration bei RL-Algorithmen ist.

In komplexeren Szenarien konnte weder mit dem A3C noch mit dem FiGAR-A3C Algorithmus eine optimale (erfolgreiche) Strategie gelernt werden. Aus diesem Grund können grundsätzlich noch keine abschliessenden Aussagen über die Trainingszeit gemacht werden. Obwohl mit dem FiGAR-A3C Algorithmus früher einen höheren Reward erzielt werden konnte, bleibt der Algorithmus ähnlich wie der A3C Algorithmus in einem lokalen Minimum stecken beziehungsweise verwendet er eine suboptimale Policy. Daher muss zuerst gewährleistet werden, dass der Agent genügend positiven Reward erhält beziehungsweise das Environment genügend lang exploriert wird.

## 5.2 Resultate Lösungsansatz 2

Die Resultate des zweiten Lösungsansatzes sind in den folgenden Abschnitten aufgeführt. Zuerst werden die Ergebnisse mit zwei Agenten und anschliessend die Ergebnisse mit drei Agenten diskutiert.

**Konfiguration**
Die Experimente wurden mit dem DQN Algorithmus durchgeführt. Für den DQN Algorithmus wurde die gleiche Architektur wie bei Mnih et al. (2015) verwendet. Die genauen Konfigurationen der Experimente sind im Anhang A.2 aufgelistet.

### 5.2.1 Resultate Experiment 1: Zwei Spieler Duell

Die Resultate des ersten Experiments sind in der Abbildung 5.7 dargestellt. In der Abbildung ist der gleitende Mittelwert des Trainingsrewards mit einer Fenstergrösse von 25 Episoden dargestellt. In der Abbildung ist ersichtlich, dass der erste Agent (P1, blaue Kurve) im Mittel einen positiven Reward erzielt. Der zweite Agent (P2, orange Kurve) erzielt im Mittel einen negativen Reward.

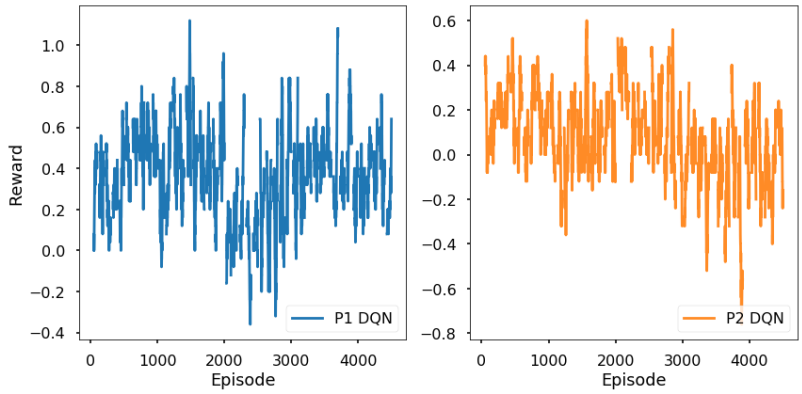



ABBILDUNG 5.7: Trainingsreward des 1. Experiments (Gleitender Mittelwert mit einer Fenstergrösse von 25).

Der Reward in der Evaluation beträgt bei beiden Agenten 0. Beide Agenten erzielen keinen Punkt.

**Charakteristik der gelernten Policies**

Die Resultate dieses Experiments zeigen erstmals eine sinnvolle Strategie der Agenten. In den Videos ist zu erkennen, dass beide Agenten unterschiedliche defensive Strategien lernen. Die Strategie des ersten Spielers (P1, blau) besteht darin, sich schnell auf die rechte Seite seiner Plattform zu bewegen und von dort aus ständig zu feuern. An der Wand angekommen, bewegt sich der erste Spieler ständig hin und her und schiesst dabei. Der erste Spieler greift den zweiten Spieler nicht direkt an, sondern wartet, bis dieser auf die rechte Seite gelangt. Seitliche Angriffe sind in diesem Szenario ausgeschlossen, da sich beide Spieler nicht drehen können. Der zweite Spieler (P2, orange) lernt eine komplett andere Strategie. Er bewegt sich nur einige Schritte von der Mitte entfernt und feuert die ganze Zeit.

### 5.2.2 Resultate Experiment 2: Zwei Spieler Duell (erweitert)

Die Resultate des zweiten Experiments sind in der Abbildung 5.8 dargestellt. In der Abbildung ist der gleitende Mittelwert des Trainingsrewards mit einer Fenstergrösse von 25 Episoden dargestellt. Beide Agenten erzielen im Mittel einen negativen Trainingsreward. Allerdings ist bei beiden Agenten ein Aufwärtstrend ersichtlich.

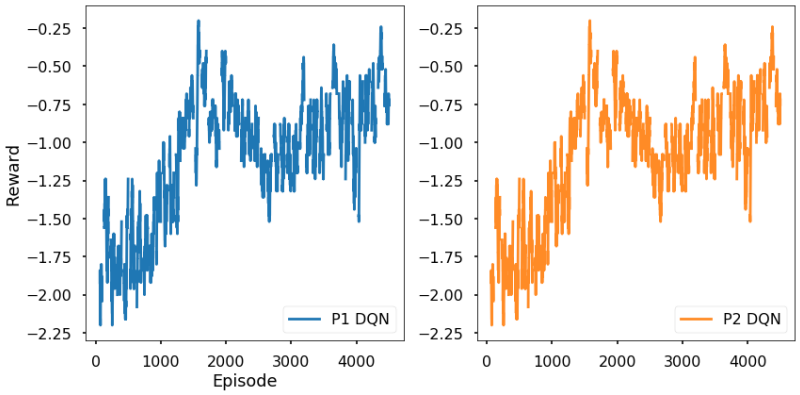



ABBILDUNG 5.8: Trainingsreward des 2. Experiments (Gleitender Mittelwert mit einer Fenstergrösse von 25).

#### Charakteristik der gelernten Policies

Die Videoanalyse zeigt deutlich, dass beide Agenten ebenfalls eine defensive Strategie lernen. Mit den zusätzlichen Aktionen $TURN\_RIGHT, TURN\_LEFT$ wird das Lernen des Agenten erschwert, da der Agent nun über die Möglichkeit verfügt, sich selber zu töten, indem er in die tödliche Substanz läuft. In der gegebenen Trainingszeit ist bei beiden Agenten zu beobachten, dass sie lediglich lernen, nicht in die tödliche Substanz zu laufen.

### 5.2.3 Resultate Experiment 3: Drei Spieler Duell

Die Resultate des dritten Experiments sind in der Abbildung 5.9 dargestellt. Die Trainingsrewards der Agenten sind einzeln dargestellt. In der Abbildung ist der gleitende Mittelwert des Trainingsrewards mit einer Fenstergrösse von 25 Episoden dargestellt. Bei allen drei Kurven ist ein Aufwärtstrend ersichtlich. Allerdings verfügen alle Agenten über einen stark negativen Reward.

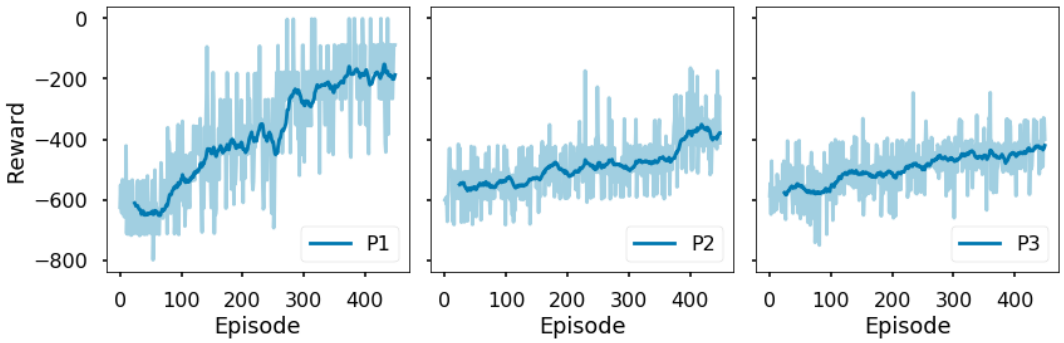



ABBILDUNG 5.9: Trainingsreward des 3. Experiments.

#### Charakteristik der gelernten Policies

Die gelernten Policies wirken sehr zufällig. In den Videos ist bei allen Agenten keine Strategie erkennbar. Die Agenten laufen in die nächste Wand oder drehen sich im Kreis.

### 5.2.4 Ursachenanalyse

Die trainierten Modelle der Agenten des ersten und zweiten Experiments erzielen in der Evaluation keinen Reward. Grundsätzlich ist ein durchschnittlicher Reward von 0 bei diesem Rewardschema und bei einem gleichstarken Gegenspieler realistisch (Tampuu et al., 2017). Allerdings haben die Experimente gezeigt, dass in jeder Episode kein einziger Reward erzielt wird. In den analysierten Videos ist kein kompetitives Verhalten beobachtbar, beziehungsweise die Agenten greifen sich gegenseitig nie an.

Im dritten Experiment verhindert der stark negative Reward ein Lernen des Algorithmus. Nach einer Analyse konnte festgestellt werden, dass die verwendete (aktuellste) ViZDoom Version ein Synchronisationsproblem aufweist (vgl. Github Issue[1]). Alle durchgeführten Experimente (mit zusätzlichen Agenten) und alle Resultate sind somit inkonsistent und können nicht interpretiert werden.

#### Mögliche Ursachen

- **Lokales Minimum:** Das passive und defensive Verhalten der Agenten könnte aufgrund eines lokalen Optimums hervorgerufen werden. In den Abbildungen 5.7 und 5.8 ist ersichtlich, dass der Reward von beiden Agenten nicht konvergiert. Die Agenten lernen somit eine suboptimale Verhaltensstrategie und passen diese nicht mehr an.
- **Algorithmus:** Dieses Verhalten könnte aufgrund des genutzten Algorithmus (DQN) hervorgerufen werden. Traditionelle RL Algorithmen wie Q-Learning sind nicht besonders geeignet für den Einsatz in Multi-Agent Environments (Lowe et al., 2017).

[1] `https://github.com/mwydmuch/ViZDoom/issues/288`

### 5.2.5 Diskussion Lösungsansatz 2

Die Resultate des ersten Experiments haben gezeigt, dass in einem einfachen Szenario mit nur drei Aktionen und mit einem gegebenen Rewardschema von beiden Agenten eine unterschiedliche defensive Strategie gelernt wird. Obwohl die Agenten für das Eliminieren des Gegners gleich viel Reward erhalten, wird der Gegner nicht bewusst angegriffen.

Im zweiten Experiment wurde das Environment leicht vergrössert, indem zwei neue Aktionen hinzugefügt wurden; mit dieser kleinen Änderung lernte der Agent weder eine offensive noch eine defensive Strategie, sondern nur, sich selber nicht zu töten.

In den beiden vorangegangen Experimenten wurde festgestellt, dass der Gegner nicht direkt angegriffen und somit der Reward nicht maximiert wird. Beide Agenten lernen eine suboptimale Strategie. Die Resultate dieser Experimente bestätigt die Aussage von Wiering et al. 2014 (vgl. Abschnitt 2.7.4). Die Idee des dritten Experiments war es, einem solchen Verhalten vorzubeugen, indem ein zusätzlicher Agent hinzugefügt wird. Verschiedene Experimente mit unterschiedlichen Parametern und verschiedenen Rewardfunktionen wurden durchgeführt. Allerdings wurden die Resultate aufgrund eines Softwarefehlers des Frameworks verfälscht und lassen somit noch keine Aussage über den Nutzen eines zusätzlichen Spielers zu.

Zusammengefasst zeigen die Resultate des zweiten Lösungsansatzes Folgendes:

- Die Self-Play Trainingsmethode ist in dieser genutzten Form (dezentrale Agenten, jeder Agent besitzt sein eigenes neuronales Netz) einfach zu implementieren und führt bei jedem Agent zu einer unterschiedlichen Strategie.
- Die Self-Play Trainingsmethode mit zwei Spielern führt selbst in einfachen Environments zu einer suboptimalen Strategie, da zu wenig exploriert wird. Der Reward wird von beiden Agenten nicht maximiert beziehungsweise das Environment zu wenig exploriert.
- Kleine Änderungen im Aktionsraum führen zu neuen Zuständen und erschweren das Lernen.

Das Hauptproblem von Self-Play ist demnach die Exploration. Um diese Trainingsmethode für 3D Environments einsetzen zu können, muss sichergestellt werden, dass die Agenten das Environment genügend explorieren.

# Kapitel 6

# Fazit und Ausblick

## 6.1 Fazit

Das Ziel dieser Arbeit war es, vorhandene Schwierigkeiten und Engpässe von RL in komplexen 3D-Umgebungen (ViZDoom) zu ermitteln und zu beheben. Nach einer ersten Analyse von bestehenden Lösungen wurde festgestellt, dass das erfolgreiche Trainieren von RL Algorithmen in solchen Umgebungen eine grosse Herausforderung darstellt. Aus diesem Grund wurde der Fokus auf diese Herausforderung gelegt. Dabei wurden zwei unterschiedliche Ansätze entwickelt, implementiert und experimentell untersucht. Der erste Ansatz basierte auf die Imitation des menschlichen Verhaltens bei der Entscheidungsfindung und der zweite Ansatz auf der Schaffung von optimalen Trainingsbedingungen. Die Resultate beider Ansätze waren für einfache 3D-Umgebungen vielversprechend, konnten jedoch nicht erfolgreich auf komplexere 3D-Umgebungen angewendet werden.

Das grösste Problem beim ersten Ansatz bestand darin, den A3C Algorithmus für komplexere 3D-Umgebungen erfolgreich trainieren zu können. Obwohl gleiche Netzwerkarchitekturen, Parametereinstellungen und Rewardfunktionen verwendet wurden, konnten bekannte Resultate nicht reproduziert werden. Dieses Problem wurde ebenfalls von Henderson et al. (2017) festgestellt, die zur gleichen Zeit DRL Algorithmen hinsichtlich der Reproduzierbarkeit untersuchten. Eine Optimierung mithilfe einer Erweiterung des A3C Algorithmus konnte deshalb nicht abschliessend untersucht werden. Beim zweiten Ansatz verursachten neben der Synchronisation des Environments, die langen Rechenzeiten und enormen benötigten Ressourcen Probleme.

Dennoch lieferten beide Ansätze interessante Ergebnisse. Im ersten Ansatz konnte experimentell nachgewiesen werden, dass, ähnlich wie mit einer zusätzlichen LSTM Schicht, Assoziationen zwischen zeitlich verschobenen Zuständen und Aktionen gelernten werden. Im zweiten Ansatz konnte in den Videos interessante Strategien beobachtet werden.

Für Reinforcement Learning für komplexe 3D-Umgebung kann zusammengefasst gesagt werden, dass aus Gründen der Empfindlichkeit der Algorithmen das Training von State of the Art Algorithmen eine grosse Herausforderung darstellt und dass die Ansätze zuerst weiter untersucht werden müssen, um abschliessende Aussagen machen zu können.

## 6.2 Ausblick

### Taktischer Ausblick

Die kurzfristigen Folgearbeiten bestehen im Lösen der Synchronisationsprobleme von ViZDoom und dem Wiederholen der zusätzlichen Experimente des zweiten Lösungsansatzes. Dieses Experiment soll ausserdem mit einer unterschiedlichen Anzahl von Agenten (beispielsweise mit $[3-8]$ Agenten) durchgeführt werden. Diese Experimente könnten mögliche Ursachen über das Verhalten der Agenten liefern und das Explorationsproblem lösen.

Des Weiteren sollen die durchgeführten Experimente mit DRL Algorithmen, welche für Mutli-Agent RL Probleme optimiert wurden, wiederholt werden (beispielsweise mit Multi-Agent Actor-Critic Algorithmus von Lowe et al. (2017)).

### Strategischer Ausblick

Die vorliegende Arbeit bestätigt die Wichtigkeit der Exploration für RL Algorithmen. In beiden Ansätzen wurden suboptimale Verhaltensstrategien gelernt und nicht mehr weiter optimiert. Dies konnte besonders beim zweiten Ansatz durch eine geringfügige Vergrösserung des Aktionsraums festgestellt werden.

Eine Möglichkeit, um dieses Problem zu beheben, wäre die Kombination der Self-Play Trainingsmethode mit dem Teacher-Student Curriculum Learning (TSCL) Framework von Matiisen et al. (2017). Das TSCL ist ein Framework für automatisches Curriculum Learning und besteht aus zwei Komponenten: dem Studenten und dem Lehrer (Teacher). Die Lehrer-Komponente wählt automatisch Aufgaben aus und verteilt diesem dem Studenten. Die Aufgaben werden dabei anhand des Lernerfolgs des Studenten priorisiert und erneut verteilt. Um die Vorteile von Self-Play beizubehalten, sollen nicht unterschiedlich komplexe Aufgaben und Szenarien erstellt werden, sondern die Komplexität soll durch die Anzahl Agenten im Environment gesteuert werden. Die Lehrer-Komponente fügt in Abhängigkeit vom Lernerfolg des Studenten Agenten hinzu oder entfernt diese.

# Anhang A

# Anhang

## A.1 Offizielle Aufgabenstellung

### A.1.1 Seite 1

Zürcher Hochschule
für Angewandte Wissenschaften

zh aw School of Engineering

InIT Institut für
angewandte Informationstechnologie

MSE | MASTER OF SCIENCE IN ENGINEERING

HS 2017

**Masterarbeit**

| | | | |
|---|---|---|---|
| Studierender: | Gabriel Eyyi | Betreuer: | Dr. Thilo Stadelmann |
| Industriepartner: | - | Korreferent: | Melanie Geiger |
| | | Experte: | Ziebart Volker |

| | | | |
|---|---|---|---|
| Ausgabe: | 04. September 2017 | Umfang: | 27 ECTS (810 Ph) |
| Abgabetermin: | 30. März 2018 | Unterschrift: | |
| Präsentation: | tbd | | |

Thema: Reinforcement Learning for Complex 3D Game Playing

Die Visual Doom AI Competition 2017[1] fordert die Teilnehmer auf, mithilfe von Machine Learning Methoden einen intelligenten Kontroller (Agent, Bot) für das Spiel „Doom“, einem 3D First Person Shooter (FPS), zu entwickeln. Die grosse Herausforderung des Wettbewerbs besteht darin, einen Kontroller für eine nicht vollständig beobachtbare (partially observable) 3D Umgebung zu entwickeln, der lediglich anhand des visuellen Inputs (Frame als Pixelmatrix) trainiert werden soll. Möglichst keine internen Zustände, wie z.B. Orientierungspunkte, Positionen von Gegenspieler, etc. sollen dem Agent während dem Training mitgegeben werden.

Diese Masterarbeit widmet sich der Fragestellung, wie ein intelligenter Agent für die zweite Wettbewerbsaufgabe mithilfe von Reinforcement Learning Methoden entwickelt werden kann. Diese zweite Wettbewerbsaufgabe besteht im Wesentlichen darin, einen Agent für ein Deathmatch Spiel in einer unbekannten Umgebung (Map) zu entwickeln.

Der wesentliche Inhalt der Arbeit setzt sich folglich aus zwei Teilaufgaben zusammen. Als Erstes soll ein Agent entwickelt werden, der die grundlegende Spieldynamik von Doom sowie eine geeignete Strategie lernt. Anschliessend soll der Agent erweitert werden, um sich in einer unbekannten Umgebung (Map) orientieren bzw. navigieren zu können.

Das Spiel „Doom“ eignet sich aufgrund des einzigartigen Software Renderers (Bildsynthese) ideal als KI-Forschungsplattform, da es die Möglichkeit bietet, „Doom“ auf einem Server ohne Desktop Umgebung laufen zu lassen und trotzdem auf den Framebuffer zuzugreifen [1]. Für den Wettbewerb wird das Framework ViZDoom zur Verfügung gestellt, welches unter anderem über eine Python API verfügt [1].

[1] http://vizdoom.cs.put.edu.pl/

## A.1.2 Seite 2

Konkret widmet sich Masterarbeit folgenden Unterfragen:

- **Environment einrichten**

  Für die Entwicklung eines Agenten sowie für die experimentelle Untersuchung von Reinforcement Learning Algorithmen ist eine auf dem Framework ViZDoom basierende geeignete Umgebung (Environment) einzurichten.

- **Evaluation durchführen**

  Für die Evaluation und den Vergleich mit dem State of the Art soll ein geeigneter Baseline Ansatz entwickelt werden. Ziel ist eine kompetitive Eingabe in einer möglichen Neuauflage der ViZDoom Challenge 2018.

- **Problembewusstsein und Lösungskonzepte entwickeln**

  Basierend auf den Erfahrungen des Baseline Ansatzes sind spezifische Problemstellungen zu identifizieren und geeignete Lösungskonzepte für diese zu entwickeln, implementieren und evaluieren. Dabei sollen State-of-the-art Methoden aus dem Bereich Deep Reinforcement Learning und Transfer Learning berücksichtig werden.

### Leistungsnachweis und Bewertung

- Schriftliche Masterarbeit
- Vorbereitung einer Eingabe zu einer möglichen Neuauflage der ViZDoom Challenge 2018
- Präsentation und mündliche Prüfung (je 30 Minuten)

### Literatur

[1] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, “ViZDoom: A Doom-based AI research platform for visual reinforcement learning,” *IEEE Conf. Comput. Intell. Games, CIG*, 2017.

## A.2 Weiters

### A.2.1 Beschreibung der elektronischen Daten

Die Daten der vorliegenden Arbeit liegen auf einem USB-Stick bei. Im folgenden Abschnitt ist der Inhalt des USB-Sticks beschrieben:

- **0_Dokumente:**
  In diesem Verzeichnis ist die Masterarbeit als PDF-Datei abgespeichert.
- **1_Experimente:**
  In diesem Verzeichnis liegen die Logfiles, Konfigurationen und Videos der Experimente der beiden Lösungsansätze.
- **2_Code:**
  In diesem Verzeichnis ist der Python-Code und die erstellen Maps abgelegt.
- **3_Notebooks:**
  In diesem Verzeichnis befinden sich die Jupyter Notebooks für die Analyse der Experimente.

# Literatur


Arulkumaran, K. et al. (2017). „Deep Reinforcement Learning: A Brief Survey“. In: *IEEE Signal Processing Magazine* 34.6, S. 26–38. ISSN: 1053-5888. DOI: 10.1109/MSP.2017.2743240.

Barto, A. G., R. S. Sutton und C. W. Anderson (1983). „Neuronlike adaptive elements that can solve difficult learning control problems“. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5, S. 834–846. ISSN: 0018-9472. DOI: 10.1109/TSMC.1983.6313077.

Bellemare, Marc G et al. (2013). „The Arcade Learning Environment: An evaluation platform for general agents.“ In: *J. Artif. Intell. Res.(JAIR)* 47, S. 253–279.

Bellman, Richard (1952). „On the theory of dynamic programming“. In: *Proceedings of the National Academy of Sciences* 38.8, S. 716–719.

— (1957). „A Markovian decision process“. In: *Journal of Mathematics and Mechanics*, S. 679–684.

— (1961). „Adaptive Control Processes, Princeton, NJ“. In: *Press, Princeton, NJ*.

Bengio, Yoshua et al. (2009). „Curriculum learning“. In: *Proceedings of the 26th annual international conference on machine learning*. ACM, S. 41–48.

Bhatti, Shehroze et al. (2016). „Playing doom with slam-augmented deep reinforcement learning“. In: *arXiv preprint arXiv:1612.00380*.

Braylan, Alex et al. (2015). *Frame Skip Is a Powerful Parameter for Learning to Play Atari*. URL: https://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/10156.

Caspi, Itai, Gal Leibovich und Gal Novik (2017). *Reinforcement Learning Coach*. DOI: 10.5281/zenodo.1134899. URL: https://doi.org/10.5281/zenodo.1134899.

Devlin, Sam, Daniel Kudenko und Marek Grześ (2011). „An empirical study of potential-based reward shaping and advice in complex, multi-agent systems“. In: *Advances in Complex Systems* 14.02, S. 251–278.

Dosovitskiy, Alexey und Vladlen Koltun (2016). „Learning to act by predicting the future“. In: *arXiv preprint arXiv:1611.01779*.

Gatti, Christopher (2014). *Design of experiments for reinforcement learning*. Springer.

— (2015). „Design of Experiments“. In: *Design of Experiments for Reinforcement Learning*. Springer, S. 53–66.

Hausknecht, M. et al. (2014). „A Neuroevolution Approach to General Atari Game Playing“. In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4, S. 355–366. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2013.2294713.

Hausknecht, Matthew und Peter Stone (2015). „Deep recurrent q-learning for partially observable mdps“. In: *CoRR, abs/1507.06527*.

Heidrich-Meisner, Verena et al. (2007). „Reinforcement learning in a nutshell.“ In: *ESANN*, S. 277–288.

Henderson, Peter et al. (2017). „Deep reinforcement learning that matters“. In: *arXiv preprint arXiv:1709.06560*.

Irpan, Alex (2018). *Deep Reinforcement Learning Doesn't Work Yet*. Blog. URL: https://www.alexirpan.com/2018/02/14/rl-hard.html.

Kaelbling, Leslie Pack, Michael L Littman und Anthony R Cassandra (1998). „Planning and acting in partially observable stochastic domains“. In: *Artificial intelligence* 101.1-2, S. 99–134.

Kempka, Michał et al. (2016). „Vizdoom: A doom-based ai research platform for visual reinforcement learning“. In: *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, S. 1–8.

Kereki, Federico (2015). *Concerning Containers' Connections: on Docker Networking*. URL: `http://www.linuxjournal.com/content/concerning-containers-connections-docker-networking`.

Lakshminarayanan, Aravind, Sahil Sharma und Balaraman Ravindran (2017). *Dynamic Action Repetition for Deep Reinforcement Learning*. URL: `https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14866/14384`.

Lample, Guillaume und Devendra Singh Chaplot (2017). „Playing FPS Games with Deep Reinforcement Learning.“ In: *AAAI*, S. 2140–2146.

LeCun, Yann, Yoshua Bengio und Geoffrey Hinton (2015). „Deep learning“. In: *nature* 521.7553, S. 436.

Li, Yuxi (2017). „Deep reinforcement learning: An overview“. In: *arXiv preprint arXiv:1701.07274*.

Littman, Michael L (2015). „Reinforcement learning improves behaviour from evaluative feedback“. In: *Nature* 521.7553, S. 445.

Lowe, Ryan et al. (2017). „Multi-agent actor-critic for mixed cooperative-competitive environments“. In: *Advances in Neural Information Processing Systems*, S. 6382–6393.

Matiisen, Tambet et al. (2017). „Teacher-Student Curriculum Learning“. In: *arXiv preprint arXiv:1707.00183*.

Mnih, Volodymyr et al. (2013). „Playing atari with deep reinforcement learning“. In: *arXiv preprint arXiv:1312.5602*.

Mnih, Volodymyr et al. (2015). „Human-level control through deep reinforcement learning“. In: *Nature* 518.7540, S. 529.

Mnih, Volodymyr et al. (2016). „Asynchronous methods for deep reinforcement learning“. In: *International Conference on Machine Learning*, S. 1928–1937.

Narvekar, Sanmit (2016). „Curriculum Learning in Reinforcement Learning:(Doctoral Consortium)“. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents und Multiagent Systems, S. 1528–1529.

Ng, Andrew Y, Daishi Harada und Stuart Russell (1999). „Policy invariance under reward transformations: Theory and application to reward shaping“. In: *ICML*. Bd. 99, S. 278–287.

Ratcliffe, Dino et al. (2017). „Clyde: A deep reinforcement learning DOOM playing agent“. In:

Sharma, Sahil, Aravind S Lakshminarayanan und Balaraman Ravindran (2017). „Learning to repeat: Fine grained action repetition for deep reinforcement learning“. In: *arXiv preprint arXiv:1702.06054*.

Silver, David et al. (2016). „Mastering the game of Go with deep neural networks and tree search“. In: *nature* 529.7587, S. 484–489.

Strehl, Alexander L et al. (2006). „PAC model-free reinforcement learning“. In: *Proceedings of the 23rd international conference on Machine learning*. ACM, S. 881–888.

Sukhbaatar, Sainbayar et al. (2017). „Intrinsic motivation and automatic curricula via asymmetric self-play“. In: *arXiv preprint arXiv:1703.05407*.

Sutton, Richard S und Andrew G Barto (1998). *Reinforcement learning: An introduction*. Bd. 1. 1. MIT press Cambridge.

Tampuu, Ardi et al. (2017). „Multiagent cooperation and competition with deep reinforcement learning“. In: *PloS one* 12.4, e0172395.

Tian, Yuandong (2017). „Deep Reinforcement Learning and Games“. Tutorial. URL: `http://yuandong-tian.com/elf-tutorial/tutorial.htmly`.

Vezhnevets, Alexander et al. (2016). „Strategic attentive writer for learning macro-actions“. In: *Advances in neural information processing systems*, S. 3486–3494.

Watkins, Christopher JCH und Peter Dayan (1992). „Q-learning“. In: *Machine learning* 8.3-4, S. 279–292.

Wiering, Marco und Martijn Van Otterlo (2012). „Reinforcement learning“. In: *Adaptation, learning, and optimization* 12.

Williams, Ronald J (1992). „Simple statistical gradient-following algorithms for connectionist reinforcement learning“. In: *Reinforcement Learning*. Springer, S. 5–32.

Witten, Ian H (1977). „An adaptive optimal controller for discrete-time Markov environments“. In: *Information and control* 34.4, S. 286–295.

Wu, Yuxin und Yuandong Tian (2016). „Training agent for first-person shooter game with actor-critic curriculum learning“. In: