

Hochschule Darmstadt

– Fachbereich Informatik –

Effizientes Training von Tischkickeragenten mittels Multi-Agent Competition

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

vorgelegt von

Adriatik Gashi

Matrikelnummer: 768367

Referent : Prof. Dr. Gunter Grieser
Korreferent : Prof. Dr. Elke Hergenröther

ERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 26. Mai 2022



Adriatik Gashi

ABSTRACT

By means of reinforcement learning, great successes have been achieved in recent years. Whether in board- or real-time strategy games, agents have been trained over long periods of time and have even been able to defeat the respective world champions in some games. One method that is often used is the use of multiple agents that train with each other. While these successes often used large hardware systems, the question arises whether the use of multi-agent competition is also suitable for challenges with limited computational resources.

In this work, an efficient training concept for training striker and goalkeeper table soccer agents in a simulation is designed and implemented. The source code state of the module "Masterprojekt Systementwicklung" from the winter semester 2021/2022 of the University of Applied Sciences Darmstadt is used as the initial implementation. The simulation-to-reality gap of the simulation is evaluated and measures are taken to reduce it. Furthermore, a systematic definition of the agent environment takes place. Based on a literature review, individual components such as the observation and action space are defined and suitable configurations for efficient learning are determined by means of experiments. Besides the selection of a learning algorithm, different aspects for the modeling of multi-agent training are discussed on the basis of further literature research and a concept is derived from this for the use in the table soccer domain.

It could be shown that the use of multiple agents is also suitable in this domain. Despite an imbalance in the difficulty of the tasks of striker and goalkeeper agents, both agents were able to learn individual strategies while taking the respective opponent into account.

ZUSAMMENFASSUNG

Mittels Reinforcement Learning konnten in den letzten Jahren große Erfolge erzielt werden. Ob in Brett- oder Echtzeitstrategiespielen wurden hierfür Agenten über lange Zeiträume trainiert und konnten auch die jeweiligen Weltmeister in einigen Spielen besiegen. Eine hierbei oft zum Einsatz kommende Methode ist die Nutzung mehrerer Agenten, die miteinander trainieren. Während diese Arbeiten oft große Hardwaresysteme nutzten stellt sich die Frage, ob sich die Nutzung von *Multi-Agent Competition* auch für Herausforderungen mit begrenzten Rechenressourcen eignen.

In dieser Arbeit wird ein effizientes Trainingskonzept für das Trainieren von Stürmer- und Torwart-Tischkickeragenten in einer Simulation entworfen und implementiert. Dabei wird der Quellcode-Stand des Moduls "Masterprojekt Systementwicklung" vom Wintersemester 2021/2022 der Hochschule Darmstadt als Ausgangsimplementierung genutzt. Es wird der Simulation-To-Reality-Gap der Simulation evaluiert und Maßnahmen zur Reduktion getroffen. Weiterhin findet eine systematische Definition der Agentenumgebung statt. Auf Grundlage einer Literaturrecherche werden einzelne Komponenten wie der Observations- und Aktionsraum definiert und mittels Versuchen geeignete Konfigurationen für ein effizientes Lernen bestimmt. Neben der Auswahl eines Lernalgorithmus werden unterschiedliche Aspekte für die Modellierung von Multi-Agenten Training auf Grundlage weiterer Literaturrecherche diskutiert und ein Konzept hieraus für die Nutzung in der Tischkickerdomäne abgeleitet.

Dabei konnte gezeigt werden, dass die Nutzung mehrerer Agenten sich auch in dieser Domäne eignet. Trotz Ungleichgewicht im Schwierigkeitsgrad der Aufgaben von Stürmer- und Torwartagenten, konnten beide Agenten einzelne Strategien erlernen wobei der jeweilige Gegner mit berücksichtigt wurde.

INHALTSVERZEICHNIS

I THESIS

1 EINLEITUNG	2
1.1 Ziel der Arbeit	3
1.2 Gliederung	3
1.3 Verwandte Arbeiten	4
2 GRUNDLAGEN	6
2.1 Künstliche Neuronale Netze	6
2.1.1 Aufbau	6
2.1.2 Funktionsweise	8
2.1.3 Aktivierungsfunktionen	10
2.2 Reinforcement Learning	11
2.2.1 Markov Decision Process	12
2.2.2 Wertefunktionen	15
2.2.3 Monte-Carlo Methoden	17
2.2.4 Dynamische Programmierung und Temporal-Difference-Learning	18
2.3 Deep Reinforcement Learning	19
2.3.1 Taxonomie Lernalgorithmen	19
3 EFFIZIENZDEFINITION	24
3.1 Probeneffizienz	24
3.2 Recheneffizienz	24
3.3 Auswahl einer Effizienzdefinition	25
4 TISCHKICKERSIMULATION	26
4.1 Aufbau Bosch Rexroth Kickertisch	27
4.1.1 Maße und Begriffsdefinitionen	28
4.2 Überblick Implementierung der Simulation	30
4.2.1 Kommunikation zwischen Simulation und Agent	30
4.2.2 Physikalische Eigenschaften der Simulation	32
4.2.3 Interpretation der Implementierung	32
4.3 Maßnahmen zur Verringerung des Simulation-To-Reality-Gaps	33
4.3.1 Kommunikation	33
4.3.2 Physikalische Eigenschaften der Simulation	35
4.3.3 Domain Randomization	38
4.4 Zusammenfassung vorgenommener Änderungen	39
4.4.1 Evaluation Simulationsleistung	40
5 DEFINITION DER AGENTENUMGEBUNG	43
5.1 Observationsraum	43
5.1.1 Frequenz an Observationen	43
5.1.2 Form und Inhalt einer Observation	46
5.1.3 Observationskanäle beim Tischkicker	50
5.1.4 Kodierung der Observationen	55

5.2	Aktionsraum	57
5.2.1	Frequenz von Aktionen	57
5.2.2	Inhalt einer Aktion	58
5.2.3	Kodierung des Aktionsraums	59
5.3	Rewardfunktion	59
5.3.1	Potenzialbasierter Reward	61
6	LERNALGORITHMUS	65
6.1	Auswahl Lernalgorithmus	65
6.1.1	Modellbasierte oder Modelfreie Methoden	65
6.1.2	Deterministische oder stochastische Policy	65
6.1.3	Wertebasierte oder policy-basierte Methoden	66
6.1.4	On-Policy oder Off-Policy	67
6.2	Implementierung	70
6.3	Hyperparameter	72
7	EXPERIMENT ZUR BESTIMMUNG DES OBSERVATIONSRAUMS	78
7.1	Vorgehensweise und Versuchsaufbau	81
7.1.1	Observationskanäle	82
7.1.2	Aktionsraum	83
7.1.3	Trainingsdauer und Hyperparameter	83
7.1.4	Episodendefinition	84
7.1.5	Evaluationsszenario	88
7.2	Ergebnisse und Auswertung	89
8	EXPERIMENT ZUR BESTIMMUNG DES AKTIONSRÄUMS	96
8.1	Vorgehensweise und Versuchsaufbau	96
8.2	Ergebnisse und Auswertung	98
9	MULTI-AGENT COMPETITION	101
9.1	Definition Multi-Agenten Umgebung	101
9.1.1	Observationsraum	102
9.1.2	Rewardfunktion	103
9.2	Lernalgorithmus	105
9.2.1	Zentralisierter und Dezentralisierter Critic	107
9.3	Modellierung von Gegnern	108
9.3.1	Gegnerische Policies	110
9.3.2	Methode zur Gegnerauswahl	111
9.4	Vorgehensweise und Versuchsaufbau	113
9.4.1	Episodendefinition	113
9.5	Ergebnisse und Auswertung	114
9.5.1	Vergleich der Rewardkurven	114
9.5.2	Ergebnisse Evaluation	117
9.5.3	Erlernte Strategien	118
10	FAZIT	124
10.1	Ergebnisse und Einordnung	124
10.2	Offene Fragen	125
II	APPENDIX	
A	APPENDIX	127

ABBILDUNGSVERZEICHNIS

Abbildung 2.1	Zeichnung eines biologischen Neurons (links) und sein mathematisches Modell (rechts)	6
Abbildung 2.2	Zwei neuronale Netze (NN). Links mit einer und Rechts mit zwei verborgenen Schichten.	7
Abbildung 2.3	Links: Sigmoid-Funktion, Rechts: Tanh-Funktion	11
Abbildung 2.4	ReLU-Funktion	11
Abbildung 2.5	Feedback-Schleife von Agent und Umgebung	12
Abbildung 4.1	Darstellung des Bosch Rexroth Kickertisches	27
Abbildung 4.2	Winkelposition 90°	29
Abbildung 4.3	Winkelposition -90°	29
Abbildung 4.4	Winkelposition 0°	29
Abbildung 4.5	Lateral-position mit Wert 1.	29
Abbildung 4.6	Lateral-position mit Wert 0.	29
Abbildung 4.7	Lateral-position mit Wert -1	29
Abbildung 4.8	Dauer in Sekunden für 1.000 Schritte in der Umgebung. <i>Baseline</i> stellt übernommenen Quellcode-Stand dar. <i>Finale Version</i> beinhaltet die Änderungen dieses Kapitels. Dargestellt mir jeweiligem bootstrapped 95%-Konfidenzintervall.	41
Abbildung 4.9	Dauer in Sekunden für 10.000 Schritte in der Umgebung. <i>Baseline mit Kamera-Fix</i> stellt behobenen Quellcode-Stand dar. <i>Finale Version</i> beinhaltet die Änderungen dieses Kapitels. Dargestellt mir jeweiligem bootstrapped 95%-Konfidenzintervall.	42
Abbildung 5.1	<i>HalfCheetah</i> Umgebung in OpenAI Gym.	45
Abbildung 5.2	Bild des Atari-Spiels Breakout. Links: Unbearbeitet. Rechts: Monochromiert und in der Auflösung reduziert.	46
Abbildung 5.3	Darstellung des genutzten Frame-Skips beim Lernen von Atari Spielen mittels Bildern. Nur jedes vierte Bild wurde genutzt.	47
Abbildung 5.4	Genutzte Architektur für das Lernen von Atari 2600 Spielen aus Bildern. Nutzung von zwei <i>Convolutional-Layer</i> als Convolutional Neural Network (CNN), gefolgt von einem Fully-Connected Neural Network (FCNN).	47

Abbildung 5.5	Verlust der Markov-Eigenschaft dargestellt am Ball, der sich entweder entlang Pfeil A oder Pfeil B bewegen kann. Nur durch eine Observation ist nicht bestimmbar welche Richtung und Geschwindigkeit der Ball hat.	51
Abbildung 5.6	Koordinatensystem in der Simulation, bestehend aus X-, Y- und Z-Achsen.	52
Abbildung 5.7	Breite des Tores als Referenz zur Bestimmung der Potenzialfunktion $\psi_z^{\text{Stürmer}}$	63
Abbildung 7.1	Repräsentation der Dropout-Schicht: O_1, \dots, O_n stellen die Observationskanäle, D_1, \dots, D_n die Dropout-Kanäle und I_1, \dots, I_n die Eingabeschicht des NN dar	80
Abbildung 7.2	Initialzustandsverteilung des Balls	86
Abbildung 7.3	Durchschnittlicher kumulierter Reward pro Rollout. Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.	90
Abbildung 7.4	Durchschnittlicher kumulierter Reward pro Episode bei Manipulation des jeweiligen Observationskanals auf der X-Achse. "Baseline" stellt die Vergleichsmetrik ohne Manipulation dar. Darstellung beinhaltet bootstrapped 95%-Konfidenzintervalle	91
Abbildung 7.5	Importance-Score der einzelnen Observationskanäle im Vergleich zur "baseline"	92
Abbildung 7.6	Durchschnittlicher kumulierter Reward pro Rollout für den erweiterten Observationsraum mit Spielerpositionen (grün) und den erweiterten Observationsraum mit Spielerpositionen ohne den mittleren Spieler (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.	93
Abbildung 7.7	Durchschnittlicher kumulierter Reward pro Rollout. Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.	94
Abbildung 8.1	Durchschnittlicher kumulierter Reward pro Rollout für kontinuierlichen Aktionsraum (grün) und multidiskretem Aktionsraum (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.	98
Abbildung 8.2	Durchschnittlicher kumulierter Reward pro Rollout für kontinuierlichen Aktionsraum (grün) und multidiskretem Aktionsraum (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.	99
Abbildung 9.1	Durchschnittlicher kumulierter Reward pro Rollout für Spiel 1 der Torwart-Agenten (rot) und Stürmer-Agenten (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.	115

Abbildung 9.2	Durchschnittlicher kumulierter Reward pro Rollout für Spiel 2 der Torwart-Agenten (rot) und Stürmer-Agenten (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.	116
Abbildung 9.3	Durchschnittlicher kumulierter Reward pro Rollout für die Spiele 1 und 2 der Torwart-Agenten (rot) und Stürmer-Agenten (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.	117
Abbildung 9.4	Beobachtbare Positionierung des Torwerts abhängig von der Ballposition relativ zur Stürmerfigur.	119
Abbildung 9.5	Beobachtbare Bewegungsmuster des Torwerts abhängig von Ballposition und Bewegung der Stürmerstange.	120
Abbildung 9.6	Beobachtbares Offensivverhalten des Torwerts abhängig von der Ballposition.	120
Abbildung 9.7	Beobachtbare Positionierung der Stürmerfigur abhängig von der jeweiligen Ballposition.	121
Abbildung 9.8	Beobachtbares Offensivverhalten des Stürmers bei Schüssen auf das Tor, abhängig von der jeweiligen Ballposition	122
Abbildung 9.9	Beobachtbares Defensivverhalten des Stürmers, abhängig von der Laufrichtung des Balles.	123
Abbildung A.1	Beispiel eines Markov Decision Process (MDP): a stellt Aktionen dar die zu Zustandstransitionen führen können; r stellt den Reward für eine ausgeführte Aktion im jeweiligen Zustand dar. Der schwarze Knoten stellt den Initialzustand dar. [DDZ20 , S. 64]	127
Abbildung A.2	Komponentendiagramm des Quellcodes vom Wintersemester 2022 [Wil+22]	128
Abbildung A.3	Datenmodell für die Übertragung von Observationen aus der Simulation an den Agenten.	131
Abbildung A.4	Histogramme der gesammelten Observationen während einer Trainingsdauer von 2,16 Mio. Trainings-schritten (1/2)	135
Abbildung A.5	Histogramme der gesammelten Observationen während einer Trainingsdauer von 2,16 Mio. Trainings-schritten (2/2)	136

TABELLENVERZEICHNIS

Tabelle 9.1	Auswertung der Agentenstärke bei Initialisierung. Alle Werte sind aus der Sicht des Torwärts.	117
Tabelle 9.2	Auswertung nach der ersten Iteration. Alle Werte sind aus der Sicht des Torwärts.	118
Tabelle 9.3	Auswertung nach der zweiten Iteration. Alle Werte sind aus der Sicht des Torwärts.	118
Tabelle A.1	Hardware Spezifikationen der zur Verfügung stehenden Rechenressourcen	127
Tabelle A.2	Die MuJoCo Umgebungen OpenAI Gym mit ihren jeweiligen Zeitschritt-Frequenzen. [Bro+16a; Ope22] . . .	129
Tabelle A.3	Die MuJoCo Umgebungen der DeepMind Control Suite mit ihren jeweiligen Zeitschritt-Frequenzen. [Tun+20]	130
Tabelle A.4	Optimierte Hyperparameter für OpenAI Gym <i>Classic Control</i> Umgebungen [Raf20]. Leere Felder stellen die Nutzung der Stable-Baselines 3 (SB3) Standard-Hyperparameter dar.	132
Tabelle A.5	Optimierte Hyperparameter für OpenAI Gym <i>PyBullet</i> Umgebungen [Raf20]. Leere Felder stellen die Nutzung der SB3 Standard-Hyperparameter dar.	133
Tabelle A.6	Optimierte Hyperparameter für OpenAI Gym <i>Mujoco</i> Umgebungen [Raf20]. Leere Felder stellen die Nutzung der SB3 Standard-Hyperparameter dar.	134

ABKÜRZUNGSVERZEICHNIS

- ACER Actor-Critic with Experience Replay
ALE Arcade Learning Environment
CNN Convolutional Neural Network
CTDE Centralized Training for Decentralized Execution
DP Dynamische Programmierung
FCNN Fully-Connected Neural Network
GAE Generalized Advantage Estimation
gRPC general-purpose Remote Procedure Calls
IO Initialer Observationsraum
KL Kullback-Leibler
MAPPO Multi-Agent Proximal Policy Optimization
MC Monte-Carlo
MDP Markov Decision Process
MLP Mehrlagiges Perzeptron
NN neuronale Netze
OPC-UA Open Platform Communications Unified Architecture
POMDP Partially Observable Markov Decision Process
PPO Proximal Policy Optimization
PBRS Potential Based Reward Shaping
RL Reinforcement Learning
RPC Remote Procedure Calls
SB₃ Stable-Baselines 3
SGD Stochastic Gradient Descent
SPS Speicherprogrammierbare Steuerung
TD Temporal Difference
TopiCo Time-optimal Trajectory Generation and Control
TPU Tensor Processing Unit
TRPO Trust Region Policy Optimization

Teil I
THESIS

EINLEITUNG

Mittels Reinforcement Learning ([RL](#)) konnten in den letzten Jahren Erfolge in unterschiedlichen Bereichen erzielt werden. So konnten [[Alp](#)] das Programm *AlphaGo* entwickeln, mit dem 2016 erstmals der Weltmeister im Spiel Go besiegt werden konnte. In ihrer Arbeit wurde *AlphaGo* mittels einer Kombination aus überwachtem Lernen unter Nutzung von menschlichen Expertenspielen und [RL](#) mittels Self-Play, also dem Spielen gegen sich selbst, trainiert. Im darauffolgenden Jahr gelang [[Sil+17b](#)] die Entwicklung des Programms *AlphaGo Zero*, das ohne Zuhilfenahme von Expertenspielen trainiert wurde. *AlphaGo Zero* wurde durch [RL](#) mittels Self-Play trainiert und begann den Trainingsprozess "tabula-rasa". Es erreichte übermenschliche Leistung und besiegte das zuvor entwickelte Programm *AlphaGo* 100 – 0. Im selben Jahr konnte durch [[Sil+17a](#)] das Programm *AlphaZero* entwickelt werden, das mittels [RL](#) und Self-Play binnen 24 Stunden übermenschliche Leistungen in Schach, Shogi und Go erreichen konnte.

Aber auch in der Domäne der Echtzeit-Strategiespiele konnten mittels [RL](#) Erfolge erzielt werden. 2019 wurde von [[Ber+19](#)] *OpenAI Five* entwickelt, das erste Programm das die jeweiligen Weltmeister in einem E-Sports Spiel besiegen konnte. Im gleichen Jahr konnte in [[Vin+19b](#)] ebenfalls für das Echtzeit-Strategiespiel *Starcraft II* ein Agent entwickelt werden, der die jeweiligen Weltmeister schlagen konnte [[Vin+19a](#)].

Die genannten Erfolge wurden alle durch Nutzung von [RL](#) erzielt, indem mehrere Agenten miteinander trainierten. Hierbei konnte gerade durch Modellierung von Wettbewerben Agenten miteinander trainiert werden, die Strategien entwickeln um den gegnerischen Agenten zu besiegen.

Gleichzeitig haben alle genannten Arbeiten einen hohen Rechenaufwand gemeinsam. In [[Vin+19b](#)] wurden über einen Zeitraum von 44 Tagen über 32 Tensor Processing Unit ([TPU](#)s) trainiert. *AlphaGo Zero* wurde über einen Zeitraum von 40 Tagen trainiert, während *AlphaGo* nur für das Spiel gegen den Go-Weltmeister insgesamt 1920 CPUs und 280 GPUs nutzte. Für *OpenAI Five* wurden die Agenten mit einer Trainingsdauer von zehn Monaten und einem Trainingsvolumen von ungefähr zwei Millionen verarbeiteten Bildern alle zwei Sekunden trainiert [[Ope+19a](#)].

Dabei stellt sich die Frage, inwieweit das Trainieren von Agenten miteinander bei begrenzten Rechenressourcen möglich ist. Hierfür wird der halb-automatisierte Bosch Rexroth Kickertisch als Problemdomäne im Rahmen dieser Arbeit untersucht. Es wird hierzu ein Trainingskonzept entworfen für das Trainieren von [RL](#)-Agenten zum Bedienen der Stürmer- und Torwartstangen mittels Multi-Agent Competition. Dies nutzt eine Simulation des Kickertisches, welche im Rahmen des Moduls "MasterProjekt Systementwicklung" von Studierenden der Hochschule Darmstadt stetig weiterentwickelt

wird. Das entworfene Konzept wird implementiert und anschließend evaluiert. Hierbei werden durch die Ergebnisse Einblicke in die Nutzbarkeit von Multi-Agenten Training in Ressourcen begrenzten Umgebungen ermöglicht.

1.1 ZIEL DER ARBEIT

Ziel der Arbeit ist die Entwicklung eines Trainingskonzepts zum Trainieren von **RL**-Agenten für die Stürmer- und Torwartstange in Bezug auf die Simulation des Bosch Rexroth Kickertisches. Hierbei wird das Trainingskonzept unter der Nebenbedingung entworfen, möglichst Effizient in Bezug auf die zur Verfügung stehenden Rechenressourcen zu sein. Dafür ist die zu nutzende Simulation zu evaluieren und ggf. zu modifizieren. Dabei wird geprüft ob die Simulation ausreichend realitätsnah implementiert ist und ob Modifikationen in Bezug auf die Effizienz dieser nötig sind. Weiterhin ist bei der Definition der Agentenumgebung im **RL**-Framework darauf zu achten, dass die einzelnen Komponenten zu möglichst effizientem Lernen führen. Hierbei wird sich **RL**-Literatur bedient die dies behandelt. Durch Experimente finden Evaluierungen zur Festlegung einzelner Komponenten statt.

Anschließend wird ein Trainingskonzept für Multi-Agenten Training von Stürmer- und Torwartstange auf Grundlage der ausgeführten Versuche und weiterer **RL**-Literatur entworfen. Dieses wird Implementiert und evaluiert, um Aussagen über die Nutzbarkeit dieses Ansatzes an diesem Beispiel zu treffen.

Durch die Fokussierung auf die Simulation wird eine Portierung des Agenten auf den Bosch Rexroth Kickertisch im Rahmen dieser Arbeit nicht näher betrachtet.

1.2 GLIEDERUNG

Die Arbeit beginnt mit Grundlagen zu **NN** als Funktionsapproximatoren, gefolgt von Grundlagen zu **RL** und Deep-**RL**, welche eine Kombination aus **RL** in Verbindung mit **NN** darstellt. Danach wird kurz auf Effizienzdefinitionen in **RL** eingegangen und die Bedeutung dieser auf die Entwicklung des Trainingskonzepts eingegangen. In Kapitel 4 wird auf die zur Verfügung stehende Tischkicker-Simulation eingegangen. Diese wird hierbei näher erläutert und ein Überblick über die Implementierung gegeben, sowie ggf. Maßnahmen zur Anpassung der Simulation bestimmt und vorgenommen. Im darauf folgenden Kapitel wird eine Definition der Agentenumgebung im Rahmen des **RL**-Frameworks vorgenommen. Dies betrifft die Definition eines potenziellen Observations- und Aktionsraumes, welche Experimentell evaluiert werden, sowie einer Rewardfunktion. In Kapitel 6 findet die Auswahl eines Lernalgorithmus für das Trainieren der Agenten statt. Kapitel 7 und 8 stellen Versuche zur Evaluierung der potenziellen Observations- und Aktionsräume dar, die in Kapitel 5 ausgearbeitet wurden. Die hierbei gewonnenen Einblicke werden bei der abschließenden Definition der Observations- und Aktionsräume für das Multi-Agenten Training genutzt. In Kapitel 9 fin-

det eine Anpassung der Umgebungsdefinition aus Kapitel 5 statt, indem diese um einen weiteren Agenten ergänzt wird. Es wird die Modellierung des Gegners und der Ablauf des Multi-Agenten Trainings zwischen Torwart- und Stürmerstange bestimmt, sowie ein Versuch mit diesem ausgearbeiteten Trainingskonzept ausgeführt und die Ergebnisse evaluiert. Abschließend findet in Kapitel 10 eine Zusammenfassung der erzielten Ergebnisse und eine Diskussion dieser statt.

1.3 VERWANDTE ARBEITEN

Zunächst werden verwandte Arbeiten in Bezug auf die Tischkickerdomäne aufgezeigt, gefolgt von Arbeiten die Simulationsumgebungen zum Trainieren von Agenten genutzt haben. Zuletzt folgen relevante Arbeiten bezüglich Multi-Agenten Training.

In [WN03] wurde ein automatisierter Kickertisch entwickelt, dessen Steuerung mittels Decision-Tree implementiert wurde. Hinzu kommt eine durch die Autoren implementierte Simulation, die das ausprobieren unterschiedlicher Strategien durch Nutzung zweier *KiRo* Klienten ermöglicht. Ein weiterer halb-automatisierter Kickertisch wurde in [JBM10] untersucht, indem eine Methode zur Verfolgung des Balls über eine Kamera entwickelt wurde. In [DB+21] wurde mittels RL-Agenten einer Stürmerstange das Schießen von Toren beigebracht. Dabei wurde ebenfalls der in dieser Arbeit betrachtete Bosch Rexroth Kickertisch verwendet. Hierbei lag der Fokus auf dem Bereich "Sim-to-Real", indem das Tischkickerbeispiel für die Darstellung eines produktionsähnlichen Prozesses gewählt wurde, welcher mittels Deep-RL optimiert wurde. Ein weiteres Tischkickerbeispiel stellt [ZNo7] dar, indem einer Tischkickerstange verschiedene Aktionsssequenzen wie das Festhalten des Balls mittels Imitation von menschlichen Aktionen beigebracht wurde. Weiterhin wurde in [Roh+21] einem Tischkicker-Torwart das Verteidigen des Tores mittels RL beigebracht. In anderen Domänen wie Roboter-Fußball konnte in [Rie+09] erfolgreich RL-Methoden zum Erlernen wichtiger Fähigkeiten eingesetzt werden.

In [Ope+19b] konnte einer Roboterhand das Lösen eines Rubix-Würfel beigebracht werden. Hierfür nutzten die Autoren eine Simulation in der der Agent trainiert wurde um dieses Modell anschließend für die Steuerung einer realen Roboterhand einzusetzen. Dadurch konnte gezeigt werden, dass durch Nutzung einer Simulation Agenten trainiert werden können, die in der Realität vergleichbare Ergebnisse erzielen. Dies konnte auch in [Ope+18] gezeigt werden, indem visuelle Objekterkennung genutzt wurde um Objekte durch eine Roboterhand wie in einer bildlichen Darstellung zu positionieren. Weitere Beispiele hierfür stellen [Alp; Sil+17b; Sil+17a] dar, in denen Agenten das Brettspiele wie Schach, Shogi und Go auf Großmeister-Niveau lernen konnten. Hierfür nutzten die Autoren ebenfalls Simulationen, um den Agenten eine Vielzahl von Spielen absolvieren lassen zu können. In den Arbeiten [Ope+19a; Vin+19b] wurden Beispiele für die Nutzung von Multi-Agenten Training in komplexen online Echtzeit-Strategiespielen aufge-

zeigt, die unter anderem die jeweiligen Weltmeister besiegen konnten. Hierbei wurden komplexe verteilte Trainingssysteme [Ope+19a] implementiert um groß-skaliertes Training zu ermöglichen. In [Vin+19b] wurde ein Liga-System entwickelt, mit der kontinuierliches Training der Agenten sichergestellt werden konnte.

Ein weiteres Beispiel für Multi-Agenten Training stellt [Ban+17] dar, indem Agenten durch das trainieren mit einem gegnerischen Agenten in simplen Umgebungen komplexe Strategien erlernen konnten. Hierbei konnten Finten, Ducken, Treten und das Stören des Gegners erlernt werden, wobei die genutzten simplen Umgebungen kein explizites Design zum Lernen dieser Strategien besaßen. Das erlernen komplexer Verhalten konnte auch in [Bak+19] gezeigt werden, indem kompetitive-kooperative Umgebungen genutzt worden sind. Hierbei wurde das Spiel *Hide-and-Seek* von mehreren Agenten in Teams gespielt, wobei erlernt werden konnte einzelne Objekte in der Umgebung für das Blockieren der gegnerischen Agenten zu nutzen, welche wiederum Gegenstrategien entwickelten, um diesen Blockaden zu umgehen. In [WGH21] konnte ebenfalls komplexes Verhalten erlernt werden, indem zwei Agenten jeweils ein dem Menschen nachempfundenes Modell kontrollieren, die in einem Boxring eingesetzt und gegeneinander im Boxen antreten. Hierbei konnte das Ausweichen von gegnerischen Schlägen und Taktiken wie schnelle Führhandschläge erlernt, sowie ein "Gorilla"-ähnliches Verhalten der Agenten beobachtet werden.

2

GRUNDLAGEN

Zunächst werden **NN** und deren Funktionsweise als Funktionsapproximierer erläutert. Anschließend werden **RL** und in der Literatur genutzte Problemdefinitionen näher erläutert. Darauf folgend wird Deep Reinforcement Learning als Kombination aus **NN** und **RL** erklärt. In diesem Zusammenhang wird eine Taxonomie von **RL**-Algorithmen dargestellt und die einzelnen Kategorien aufgezeigt. Abschließend folgt eine Einführung in Multi-Agent Reinforcement Learning und Besonderheiten bei Problemdefinitionen dieser Art.

2.1 KÜNSTLICHE NEURONALE NETZE

In diesem Abschnitt werden künstliche **NN** näher erläutert. Dies ist wichtig, da Deep-**RL**-Algorithmen diese als Funktionsapproximatoren nutzen [Bou+22]. In diesem Rahmen wird ihr Aufbau und ihre Funktionsweise erläutert.

2.1.1 *Aufbau*

NN sind in ihrem Aufbau an den Aufbau des menschlichen Gehirns angelehnt. Das menschliche Gehirn besteht aus Neuronen, welche mittels Synapsen miteinander verbunden sind. Hierbei erhält ein Neuron ein Signal über Dentride und kann über sein Axon Signale weitergeben. Axone sind jeweils über Synapsen zu weiteren Dentriden verbunden, wodurch bei ca. 86 Millionen Neuronen ca. $10^{14} - 10^{15}$ Synapsen im menschlichen Nervensystem vorhanden sind. Abbildung 2.1 stellt den Aufbau eines menschlichen Neurons dem nachempfundenen, mathematischen Modell, einem **Perzeptron**, gegenüber.

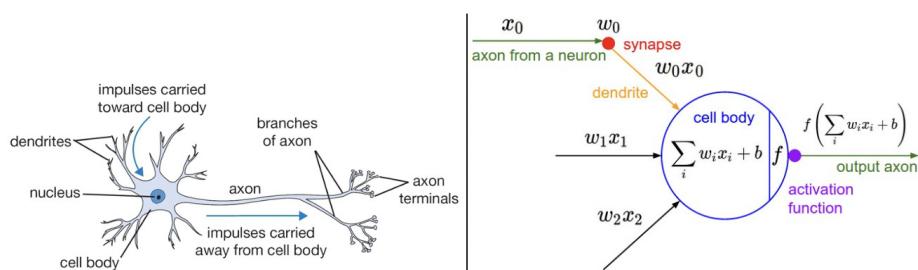


Abbildung 2.1: Zeichnung eines biologischen Neurons (links) und sein mathematisches Modell (rechts)

Quelle: [FFL22a]

Ein Perzeptron stelle das mathematische Modell eines biologischen Neurons dar. Hierbei kann ein Perzeptron mehrere Eingabe- und Ausgabeverbindungen haben. Während ein biologisches Neuron elektrische Impulse durch die eingehenden Dendriten sammelt und es bei der Überschreitung des Aktionspotenzials zu einer Weiterleitung des Signals kommt, findet in einem Perzeptron ebenfalls eine Verarbeitung von eingehenden Daten statt. Wie in Abbildung 2.1 dargestellt werden die eingehenden Daten x_0, x_1, x_2 mit jeweils eigenen Parametern w_0, w_1, w_2 multipliziert. Diese Parameter werden als **Gewichte** bezeichnet und bringen die Stärke der jeweiligen Verbindung zum Ausdruck. Somit ist ein Gewicht einer jeweiligen Verbindung zugeordnet und unabhängig vom übertragenen Wert x .

Nachdem ein eingehender Wert mit dem jeweiligen Gewicht multipliziert wurde, wird das Produkt mit einem skalaren Wert, dem **Bias**, addiert.

Dieser Bias ist dem Perzeptron selbst zugeordnet. Anschließend wird diese Summe als Eingabe für die Aktivierungsfunktion genutzt. Die Ausgabe der Aktivierungsfunktion wird dann an die nächsten Verbindungen weitergeleitet. Warum Aktivierungsfunktionen benötigt werden und Beispiele hierfür werden in Kapitel 2.1.3 genannt und erläutert.

Wie bereits beschrieben, besteht das menschliche Nervensystem aus einer Vielzahl von Neuronen, die miteinander verbunden sind. So bestehen auch **NN** aus mehreren Perzeptren, die miteinander verbunden sind. Zwei unterschiedliche Architekturen sind in Abbildung 2.2 dargestellt.

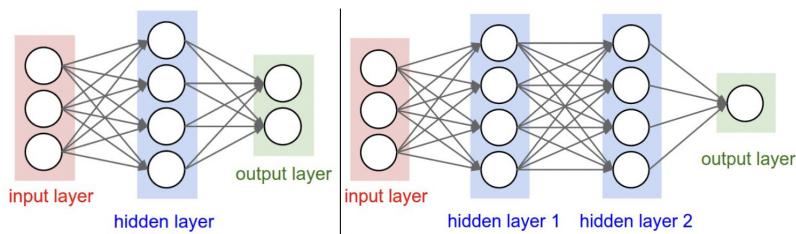


Abbildung 2.2: Zwei **NN**. Links mit einer und Rechts mit zwei verborgenen Schichten.

Quelle: [FFL22a]

Hierbei sind drei Arten von Schichten zu erkennen: Eingabeschicht (rot), verborgene Schicht (blau) und Ausgabeschicht (grün). Jede Schicht besteht aus mehreren Perzeptren. Eine solche Architektur wird als Mehrlagiges Perzeptron (**MLP**) bezeichnet. Die Eingabeschicht stellt den Beginn des **NN** dar. Hier werden die zu verarbeitenden Daten in das **NN** eingegeben. Zwischen Eingabe- und Ausgabeschicht sind verborgene Schichten vorhanden. Diese bestehen aus mehreren Perzeptren. Die Anzahl an Perzeptren in einer verborgenen Schicht gibt die Breite dieser Schicht an. Die Anzahl an verborgenen Schichten beschreibt die Tiefe eines **NN**. In Abbildung 2.2 werden zwei vollvermaschte **NN** unterschiedlicher Größe dargestellt. Hierbei ist zu beachten, dass durch diese Vollvermaschung zwar jedes Perzeptron einer vorhergehenden Schicht mit jedem Perzeptron der darauf folgenden Schicht

verbunden ist, die Perzeptren einer einzelnen Schicht jedoch nicht untereinander verbunden sind. Weiterhin ist hierbei zu beachten, dass die Verbindungen der Perzeptren gerichtet sind. Daten fließen nur in der Richtung von Eingabe- zur Ausgabeschicht. Ein solches **NN** wird als Feed-Forward **NN** bezeichnet.

Da ein Gewicht jeder Verbindung von Perzepron zu Perzepron zugeordnet ist und jedes Perzepron (außer der Eingabeschicht) einen eigenen Bias besitzt, besteht somit das linke **NN** in Abbildung 2.2 aus 20 Gewichten und 6 Bias. Das rechte **NN** in Abbildung 2.2 besitzt jedoch bereits 32 Gewichte und 9 Bias, obwohl die Ausgabeschicht nur ein statt zwei Perzeptren beinhaltet. [FFL22a]

Die Gewichte und Bias eines **NN** werden als Parameter θ eines **NN** bezeichnet. Diese Parameter können während des Lernprozesses angepasst werden um nicht-lineare Funktionen abhängig von der Eingabe x approximieren zu können [FFL22a]. Eine wichtige Funktion hierfür erfüllen die Aktivierungsfunktionen, die im nächsten Abschnitt näher erläutert werden.

2.1.2 Funktionsweise

NN werden genutzt, um nicht-lineare Funktionen zu approximieren. Beinhaltet ein **NN** mindestens eine verborgene Schicht, so ist dieses **NN** ein universeller Funktionsapproximator.

Hierfür werden die Parameter des Netzwerkes θ iterativ angepasst, bis eine gewünschte Approximationsgenauigkeit vorliegt. Das Anpassen dieser Parameter wird als Lernen bzw. Training bezeichnet und basiert auf folgenden Schritten [DDZ20, S. 16-20]:

1. Feed-Forward
2. Fehlerberechnung
3. Back-Propagation
4. Anpassung der Parameter

FEED-FORWARD beschreibt die Berechnung der Ausgabe eines **NN** auf Grundlage von Eingabewerten x . Da **NN** Funktionen approximieren, ist das Ergebnis dieser Berechnung eine Schätzung des Wertes, den die zu approximierende Funktion $f(x)$ bei Eingabe eines beliebigen x bestimmen würde. Im Rahmen dieses Abschnittes wird die zu approximierende Funktion als $f(x)$ bezeichnet, während die Ergebnisse dieser Berechnungen als y bezeichnet werden. Die berechneten Werte des **NN** $g_\theta(x)$ werden als \hat{y} bezeichnet. Somit wird in diesem Schritt $\hat{y} = g_\theta(x)$ berechnet.

FEHLERBERECHNUNG Im nächsten Schritt wird der Fehler der Schätzung berechnet. Hierbei wird unter Berücksichtigung der berechneten Werte \hat{y}

und der tatsächlichen Werte y der Schätzfehler mittels einer sogenannten Fehlerfunktion $L(\theta)$ berechnet. Für die Berechnung des Schätzfehlers gibt es mehrere Möglichkeiten. Eine oft genutzte Methode stellt die Berechnung der mittleren quadratischen Abweichung dar, wodurch folgende Formel folgen würde:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (2.1)$$

wobei n die Anzahl an Wertepaaren von Schätzung \hat{y} und tatsächlichem Wert y darstellt. [SSBD14, S. 123]

BACK-PROPAGATION bezeichnet die Rückführung des Schätzfehlers auf die einzelnen Parameter des **NN**. Hierfür wird der globale Gradient der Fehlerfunktion $\nabla L(\theta)$ berechnet. Die Funktionsweise des Back-Propagation Algorithmus basiert auf der Kettenregel, mit der lokale Gradienten der Fehlerfunktion für die einzelnen Netzwerkparameter berechnet werden können. Somit kann für alle Parameter θ eines Netzwerkes deren Einfluss auf den Schätzfehler bestimmt werden.[GBC16, S. 200-216]

ANPASSUNG DER PARAMETER Um die einzelnen Parameter des **NN** anzupassen, werden die in der Back-Propagation berechneten lokalen Gradienten der Fehlerfunktion benutzt. Eine Methode, die in vielen Deep Learning Algorithmen hierfür zum Einsatz kommt, ist Stochastic Gradient Descent (**SGD**) [GBC16, S. 149]. Das Anpassen der Parameter basierend auf einzelnen Schätzungen führt bei großen Mengen an Trainingsdaten zu einer potentiell langen Dauer bis eine Aktualisierung aller Parameter stattfinden kann. **SGD** nutzt hierfür die Tatsache, dass die Gradienten lediglich eine Erwartung sind, und diese Erwartungen durch Nutzung kleinerer Gruppen von Trainingsdaten approximiert werden kann.

Die Menge an Trainingsdaten für eine Iteration wird als Minibatch bezeichnet. Die Größe des Minibatch wird hierbei während des Trainings normalerweise konstant gehalten. Diese Schätzung des Gradienten \mathcal{G} basierend auf einem Minibatch wird dann für die Anpassung der **NN** Parameter folgendermaßen genutzt:

$$\theta = \theta - \alpha \mathcal{G} \quad (2.2)$$

wobei α die Lernrate darstellt [GBC16, S. 149-150]. Die Lernrate wird hierbei genutzt, um die Anpassungsrate der Parameter zu bestimmen. Ist diese zu klein, kann das Training eine lange Zeit in Anspruch nehmen, bis ein lokales Optimum gefunden worden ist. Ist sie zu groß, kann es passieren, dass ein lokales Optimum überschossen und somit nicht erreicht wird. [Gro]

2.1.3 Aktivierungsfunktionen

Aktivierungsfunktionen stellen einen wichtigen Baustein für **NN** dar. Ohne Aktivierungsfunktionen könnten **NN** lediglich lineare Funktionen approximieren. Wären **NN** lediglich aus Gewichten und Bias aufgebaut, könnten lediglich Funktionen ersten Grades approximiert werden. Durch Nutzung von Aktivierungsfunktionen, die das Aktionspotenzial eines biologischen Neurons modellieren, können **NN** beliebige, nicht-lineare Funktionen approximieren [DDZ20, S. 11-12]. Eine Aktivierungsfunktion $f(x)$ hat als Parameter einen skalaren Wert und bildet diesen auf einen ebenfalls skalaren Wert ab [FFL22a]. Die Aktivierungsfunktionen **Sigmoid**, **Tanh** und **ReLU** werden oft für **NN** eingesetzt und werden hier in kürze näher erläutert:

SIGMOID besitzt einen Wertebereich von $(0, 1)$ und bildet einen Eingabewert x folgendermaßen ab: $\sigma(x) = 1/(1 + e^{-x})$ [FFL22a]. Der Verlauf dieser Funktion ist in Abbildung 2.3 dargestellt.

Diese Aktivierungsfunktion besitzt jedoch einige Nachteile, weshalb sie weniger häufig als Aktivierungsfunktion in **NN** zum Einsatz kommt. Zum einen kann es dazu kommen, dass ein Perzeptron relativ große bzw. kleine Werte als Ausgabe berechnet. Diese werden auf der Sigmoid-Funktion auf ≈ 1 bzw. ≈ 0 abgebildet. Da in der Back-Propagation die lokalen Gradienten bezüglich der Netzwerkparameter berechnet werden, können diese gegen 0 tendieren für jeweilige Perzeptren, die relativ große bzw. kleine Ausgaben berechnet haben. Da jedoch durch die Nutzung der Kettenregel für die Bestimmung der einzelnen lokalen Gradienten eine Multiplikation dieser durch alle Schichten des Netzwerkes vollzogen wird, wird der Gradient pro Ebene exponentiell kleiner. Kleine Gradienten haben zur Folge, dass die Parameter des **NN** gerade in den ersten Ebenen des **NN** nur gering angepasst werden. Dieses Problem wird als *Vanishing Gradient Problem* bezeichnet und stellt einen Grund dar, warum die Sigmoid-Funktion weniger häufig in **NN** genutzt wird. Ein weiterer Grund ist, dass der Wertebereich der Sigmoid-Funktion nicht null-zentriert ist, was bei Nutzung von Minibatches von kleiner Größe zu nicht-wünschenswerten Zick-Zack Bewegungen bei Aktualisierung der Parameter führen kann.[FFL22a]

TANH stellt eine null-zentrierte Sigmoid-Funktion mit einem Wertebereich von $(-1, 1)$ dar: $\text{Tanh}(x) = 2\sigma(2x) - 1$. Sie ist in Abbildung 2.3 dargestellt. Aufgrund ihrer Null-Zentrierung wird sie in der Praxis im Vergleich zur Sigmoid-Funktion bevorzugt. Dennoch kann es bei dieser Aktivierungsfunktion immer noch zum Vanishing Gradient Problem kommen. [DDZ20, S. 12]

RELU bzw. Rectified Linear Unit wird durch folgende Funktion berechnet und ist in Abbildung 2.4 dargestellt: $\text{ReLU}(x) = \max(0, x)$. Diese Funktion besitzt zum einen den Vorteil, dass sie einfacher als Sigmoid oder Tanh

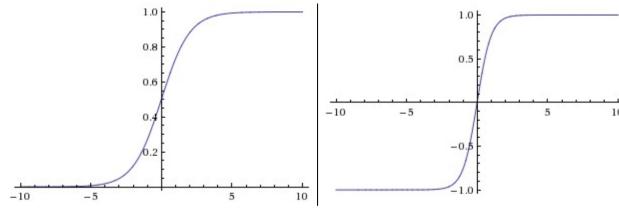


Abbildung 2.3: Links: Sigmoid-Funktion, Rechts: Tanh-Funktion

Quelle: [FFL22a]

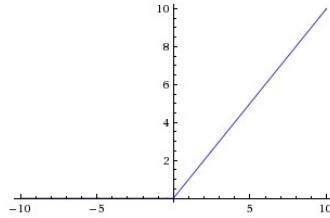


Abbildung 2.4: ReLU-Funktion

Quelle: [FFL22a]

berechnet werden kann, welche exponentielle Berechnungen durchführen. Weiterhin wurde in [KSH12] durch Nutzung dieser Aktivierungsfunktion das Training um Faktor sechs beschleunigt, was durch ihre Linearität und ungesättigte Form argumentiert wird. Ein Nachteil ist jedoch das sogenannte Dying ReLU Problem. Werden bei der Back-Propagation große lokale Gradienten berechnet, so kann dies dazu führen, dass einzelne Perzeptren so stark angepasst werden, dass diese "sterben" und keine Aktivierung mehr erzielen können. [FFL22a]

2.2 REINFORCEMENT LEARNING

Reinforcement Learning stellt einen Teilbereich des maschinellen Lernens dar [WO12, S. XI]. Hierbei sind die maßgeblichen Komponenten der *Agent* und die *Umgebung* [DDZ20, S. 3]. Ein Agent interagiert mit der Umgebung und hat ein Ziel zu erreichen [WO12]. Für diese Interaktion besitzt ein Agent eine Reihe von Aktionen, die genutzt werden können. Diese zur Verfügung stehenden Aktionen werden als Aktionsraum bezeichnet. Aktionen können diskreter oder kontinuierlicher Natur sein [DDZ20; WO12, S. 48, S. 10]. Durch Ausübung von Aktionen kann sich der Zustand der Umgebung bzw. des Agenten in dieser Umgebung verändern. Um dies wahrzunehmen werden dem Agenten Observationen zuteil. Durch diese Observationen ist der Agent in der Lage, Änderungen in der Umgebung wahrzunehmen und diese Informationen bei der Wahl einer nächsten Aktion zu berücksichtigen. Auf Grundlage des definierten Ziels findet eine Konditionierung des Agenten mittels Rewards statt [SB18, S. 6]. Rewards sind skalare Werte, die dem Agenten zeigen ob ein erreichter Zustand in der Umgebung nützlich für

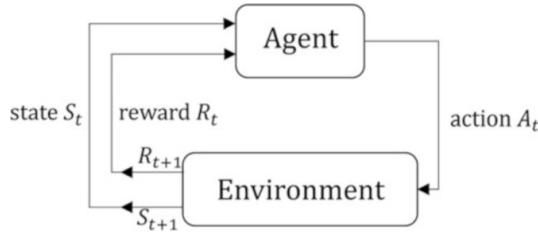


Abbildung 2.5: Feedback-Schleife von Agent und Umgebung

Quelle: [DDZ20, S. 48]

den Agenten ist oder nicht. Durch Wahrnehmung und Interaktion mit der Umgebung, sowie Konditionierung mittels Rewards entsteht eine Feedback-Schleife, die das Erlernen eines Verhaltens zur Maximierung des Rewards ermöglicht. Diese Feedback-Schleife ist in Abbildung 2.5 dargestellt.

Dies stellt eine grobe Erläuterung vom **RL**-Framework dar. Für die Formulierung von **RL**-Problemen eignen sich **MDP** [WO12, S. 10, 12]. Im nächsten Abschnitt werden **MDP** näher erläutert und deren Nutzung in **RL**. Anschließend wird auf Policies und Wertefunktionen in diesem Zusammenhang eingegangen und eine Reihe von Optimierungsverfahren für **RL**-Probleme näher erläutert.

2.2.1 Markov Decision Process

Um ein **RL**-Problem formal zu definieren eignen sich **MDP** [WO12, S. 10, 12]. Dieses besteht aus einem Aktionsraum \mathcal{A} , einem Zustandsraum \mathcal{S} , einer Rewardfunktion \mathcal{R} und einer Zustandsübergangsfunktion \mathcal{P} . Wie bereits erläutert stellt der Aktionsraum \mathcal{A} alle einem Agenten zur Verfügung stehenden Aktionen dar. Diese Aktionen sind die einzige Möglichkeit des Agenten mit der Umgebung zu interagieren. Der Zustandsraum \mathcal{S} stellt alle möglichen Zustände der Umgebung dar. Hierbei ist wichtig zwischen dem Zustandsraum \mathcal{S} und dem Observationsraum \mathcal{O} zu unterscheiden. Observationen stellen Repräsentationen eines Zustandes dar. Ist eine Umgebung vollständig beobachtbar durch den Agenten, so entspricht eine Observation einem Zustand in der Umgebung. Ist dies nicht der Fall, da die Umgebung nur teilweise beobachtbar ist, so entspricht eine Observation nicht dem tatsächlichen Zustand der Umgebung. Im folgenden wird jedoch eine vollständig beobachtbare Umgebung angenommen, weshalb nur der Zustandsraum weiter referenziert wird.

Interaktionen zwischen Agent und Umgebung finden zu diskreten Zeitpunkten $t = 0, 1, 2, \dots$ statt. Zu jedem diskreten Zeitpunkt erhält der Agent eine Repräsentation des Zustandes der Umgebung $S_t \in \mathcal{S}$ und wählt auf dieser Grundlage eine Aktion $A_t \in \mathcal{A}$ zum Ausführen aus [SB18, S. 48]. Die Rewardfunktion \mathcal{R} stellt eine Funktion dar, die auf Grundlage eines Zustands zu einem bestimmten Zeitpunkt $S_t \in \mathcal{S}$ und einer Aktion zu diesem Zeitpunkt $A_t \in \mathcal{A}$ einen Reward $R_{t+1} \in \mathbb{R}$ bestimmt, abhängig vom erreichten

Zustand $S_{t+1} \in \mathcal{S}$. Die Wahrscheinlichkeit zum Erreichen von Zustand S_{t+1} durch Ausführung von Aktion A_t in Zustand S_t wird durch die Zustandsübergangsfunktion \mathcal{P} dargestellt. Damit ist $\mathcal{P}(S_t, A_t, S_{t+1}) = p(S_{t+1}|S_t, A_t)$ die bedingte Wahrscheinlichkeit den Zustand S_{t+1} zu erreichen, unter der Bedingung in Zustand S_t Aktion A_t auszuführen.

Durch Interaktion von Agent und Umgebung entstehen Transitionen, die als Tupel $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$ dargestellt werden können. Diese Sequenz von Transitionen wird als Episode τ bezeichnet [DDZ20, S. 50]. Eine Episode ist beendet wenn ein terminaler Zustand im MDP erreicht wird. Um die zeitliche Abfolge von Transitionen darstellen zu können, wird die Länge einer Episode mit T dargestellt, wobei t einen beliebigen diskreten Zeitpunkt repräsentiert. Weiterhin gilt hierfür $t \geq 0$ sowie $t \leq T$. Daraus folgt, dass die Länge einer Episode $\text{len}(\tau) = T$ entspricht.

Im Rahmen dieser Arbeit stellen A , S , R und P konkrete Werte und s , a , r und p allgemeine Konzepte von Zustand, Aktion, Reward und Zustandsübergangswahrscheinlichkeit dar. \mathcal{A} , \mathcal{S} , \mathcal{R} und \mathcal{P} stellen den Zustands- und Aktionsraum sowie die Reward- und Zustandsübergangsfunktion dar. Somit gilt:

- $A_t \in \mathcal{A} | \forall t \in T$, wobei A_t eine konkrete Aktion des Aktionsraumes \mathcal{A} zu einem beliebigen Zeitpunkt t darstellt.
- $S_t \in \mathcal{S} | \forall t \in T$, wobei S_t einen konkreten Zustand des Zustandsraumes \mathcal{S} zu einem beliebigen diskreten Zeitpunkt t darstellt.
- $R_{t+1} = \mathcal{R}(S_t, A_t, S_{t+1}) | \forall t \in T$, wobei R_t einen konkreten Reward als Wert der Rewardfunktion \mathcal{R} darstellt.
- $P_t \in \mathcal{P}(S_t, A_t, S_{t+1}) | \forall t \in T$, wobei P_t die Wahrscheinlichkeit für die Transition von Zustand S_t zu Zustand S_{t+1} unter Ausführung von Aktion A_t zu einem beliebigen diskreten Zeitpunkt t darstellt.

Der Agent interagiert mit der Umgebung um ein Ziel zu erreichen. Das Ziel des Agenten ist die Maximierung der zu erwartenden summierten Rewards, der auf lange Sicht erhalten wird [SB18, S. 6]. Dieser summierte Reward wird als Return bezeichnet. Als G_t wird der Return ab dem Zeitpunkt t dargestellt [SB18, S. 55]. Somit wird definiert [SB18, S. 54]:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T \quad (2.3)$$

Diese Return-Definition ist jedoch problematisch in Umgebungen, die keine natürlichen Terminalzustände besitzen und somit $T = \infty$ gilt. Dies führt zu einem Return der selbst unendlich werden könnte, wenn beispielsweise zu jedem Zeitpunkt ein positiver Reward zuteil wird. Daraus würde in diesem Beispiel folgen, dass sowohl zufälliges Verhalten als auch optimales Verhalten zu einem unendlichen Reward führen würde. Um dieses Problem zu umgehen wird ein Diskontierungskonzept genutzt [SB18, S. 57]. Durch Nutzung eines Diskontierungsfaktors γ wird der Return über die Zukunft

maximierbar. Daraus ergibt sich folgende Return-Definition die für MDP mit und ohne Terminalzustände genutzt werden kann:

$$G_t \doteq \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (2.4)$$

wobei $0 \leq \gamma \leq 1$ und $T \leq \infty$ (aber nicht $T = \infty$ und $\gamma = 1$) gilt [SB18, S. 57]. Dieser Diskontierungsfaktor ist Teil der Umgebungsdefinition, wodurch ein MDP durch folgendes Tupel aus Zustandsraum \mathcal{S} , Aktionsraum \mathcal{A} , Zustandsübergangsfunktion \mathcal{P} , Rewardfunktion \mathcal{R} und Diskontierungsfaktor γ dargestellt werden kann $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ [DDZ20, S. 127].

Somit sagt einzig die Rewardfunktion aus, welche erreichten Zustände gut oder schlecht für den Agenten sind. Um einen möglichst hohen Reward sammeln zu können, muss vom Agenten ein Verhalten erlernt werden, sodass in jedem Zustand $S_t \in \mathcal{S} \mid \forall t \in T$ die Aktion $A_t \in \mathcal{A} \mid \forall t \in T$ ausgeführt wird, die den maximalen Reward zur Folge hat. Dieses Verhalten, das eine Zuordnung von Aktionen auf Zustände darstellt, bezeichnet man als Policy π [SB18, S. 6]. Eine Policy $\pi(a|s)$ stellt eine Wahrscheinlichkeitsverteilung von Aktionen zu Zuständen dar, welche die Wahrscheinlichkeit definiert mit der eine Aktion a in einem Zustand s ausgeführt wird [DDZ20, S. 65]. Diese Wahrscheinlichkeitsverteilung kann stochastischer oder deterministischer Natur sein.

Der erwartete Return ist eine Erwartung einer Policy an die Returns über alle möglichen Episoden hinweg. Gegeben einer Policy π und einer Initialzustandsverteilung \mathbb{P}_0 für den Beginn einer Episode, so ist die Wahrscheinlichkeit für eine Episode τ mit der Länge T [DDZ20, S. 65]:

$$p(\tau|\pi) = \mathbb{P}_0(S_0) \prod_{t=0}^{T-1} \mathcal{P}(S_{t+1}|S_t, A_t) \pi(A_t|S_t) \quad (2.5)$$

wobei $p(\tau|\pi)$ die bedingte Wahrscheinlichkeit für das Eintreten einer Episode τ unter π darstellt [DDZ20, S. 66]. Hieraus ergibt sich die Formel für den erwarteten Return $J(\pi)$ [DDZ20, S. 65]:

$$J(\pi) = \int_{\tau} p(\tau|\pi) \mathcal{R}(\tau) = \mathbb{E}_{\tau \sim \pi} [\mathcal{R}(\tau)] \quad (2.6)$$

wobei $\mathcal{R}(\tau)$ den Return einer Episode τ darstellt und $\tau \sim \pi$ eine unter der Policy π entstandene Episode darstellt [DDZ20, S. 65]. Eine optimale Policy π_* maximiert somit den erwarteten Return folgendermaßen [DDZ20, S. 66]:

$$\pi_* = \arg \max_{\pi} J(\pi) \quad (2.7)$$

Für die visuelle Darstellung von MDP eignen sich Graphen, dessen Knoten Zustände und Kanten Aktionen repräsentieren. Ein Beispiel hierfür findet sich im Anhang in Abbildung A.1.

In [SB18, S. 49] wird aufgezeigt, dass in einem MDP die Zustandsübergangsfunktion \mathcal{P} die Dynamiken der Umgebung vollständig beschreibt. In einem MDP erfüllt jeder Zustand S die Markov-Eigenschaft [Sil15, Kap. 2]. Grundsätzlich kann davon ausgegangen werden, dass die Transition in einen Zustand S_{t+1} abhängig von allen zuvor gegangenen Transitionen $\langle S_0, A_0, S_1, A_1, \dots, S_t, A_t \rangle$ ist [SB15, S. 64]. Wird jedoch die Markov-Eigenschaft erfüllt bedeutet dies, dass die Wahrscheinlichkeit für die Transition in einen beliebigen Zustand S_t lediglich von dem vorausgegangenen Zustand S_{t-1} und der zu diesem Zeitpunkt ausgeführten Aktion A_{t-1} abhängig ist. So mit ist die Historie von vorausgegangenen Zuständen nicht erforderlich zur Bestimmung der Wahrscheinlichkeit zur Transition zum Zustand S_{t+1} basierend auf S_t, A_t [Sil15, Kap. 2]:

$$p(S_{t+1}|S_t, A_t) = p(S_{t+1}|S_0, A_0, \dots, S_t, A_t) \quad (2.8)$$

Dies wird als Markov-Eigenschaft bezeichnet und stellt sich als Restriktion auf die Zustandsrepräsentation dar. Weiterhin führen die Autoren aus, dass ein Zustand somit alle für die Zukunft relevanten Informationen beinhalten muss, basierend auf der Vergangenheit der Interaktion von Agent und Umgebung. Die Markov-Eigenschaft ist in RL wichtig, weil angenommen wird, dass Entscheidungen nur vom aktuellen Zustand abhängen. Damit diese Entscheidungen effektiv und informativ sind, muss die Zustandsdarstellung informativ ebenfalls sein [SB98, Kap. 3.5].

Eine weitere wichtige Eigenschaft von MDP ist deren Stationarität. Ist ein MDP stationär, so bleibt die Wahrscheinlichkeitsverteilung für die Zustandsübergangsfunktion und somit den Reward über einen beliebigen Zeitraum gleich [SB18, S. 30, 32]. Ist ein MDP nicht-stationär, so können sich diese Zustandsübergangsfunktion im Laufe der Zeit ändern. Während in stationären MDP lediglich die besten Aktionen für einen jeweiligen Zustand gefunden werden müssen, können sich diese bei nicht-stationären MDP im Laufe der Zeit ändern. Somit ist im Rahmen von nicht-stationären MDP auch nicht-optimale Aktionen Überblick zu behalten um beobachten zu können, ob, wann und wie sich die Dynamiken der jeweiligen Umgebung verändern. [DDZ20, S. 58-59]

2.2.2 Wertefunktionen

Die meisten RL-Algorithmen nutzen Wertefunktionen um zu bestimmen, wie nützlich ein Zustand oder eine Zustands-Aktions-Kombination in der Umgebung sind [SB18, S. 58]. Es gibt zum einen die Zustandswertfunktion sowie die Aktionswertfunktion. Die Zustandswertfunktion $V_{\pi(s)}$ beschreibt die Nützlichkeit des Zustandes s für den Agenten unter Nutzung der Policy π . Die Nützlichkeit wird anhand des zu erwarteten Returns ausgehend vom betrachteten Zustand ausgedrückt und ist abhängig von der genutzten Policy. Somit ist der Zustandswert eines beliebigen Zustandes s der erwartete

Return G_t wenn der Agent im Zustand s startet und von dort aus der Policy π folgt [SB18, S. 58]:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi [G_t | S_t = s] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \text{ für alle } s \in \mathcal{S} \text{ und alle } a \in \mathcal{A} \end{aligned} \quad (2.9)$$

Analog hierzu stellt die Aktionswertfunktion $Q_\pi(s, a)$ den erwarteten Return ausgehend vom betrachteten Zustand s dar, wenn in diesem Zustand die Aktion a ausgeführt und anschließend der Policy π gefolgt wird [SB18, S. 58]:

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}_\pi [G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \text{ für alle } s \in \mathcal{S} \text{ und alle } a \in \mathcal{A} \end{aligned} \quad (2.10)$$

Formuliert man die Gleichung 2.9 rekursiv um, so erhält man die *Bellman* Gleichung für V_π [SB18, S. 59]:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi [G_t | S_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) [\mathcal{R}(s, a, s') + \gamma \mathbb{E}_\pi [G_{t+1} | S_{t+1} = s']] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) [\mathcal{R}(s, a, s') + \gamma V_\pi(s')] \end{aligned} \quad (2.11)$$

In der letzten Gleichung in 2.11 wird durch die rekursive Beziehung der Wert eines Zustandes in einen sofortigen Teil (dargestellt durch $\mathcal{R}(s, a, s')$) und einen zukünftigen Teil (dargestellt durch $V_\pi(s')$ mit $S_{t+1} = s'$). Hierdurch wird Bellmans Prinzip der Optimalität erfüllt. Dieses besagt, gegeben einer beliebigen Policy π und Initialzustand S_0 , wenn π als optimale Policy betrachtet wird, die verbleibenden Entscheidungen von π , unabhängig von dem Ausgangszustand S_0 einer optimalen Policy π_* entsprechen müssen, damit π als optimale Policy angesehen werden kann [DDZ20, S. 164]. Analog kann folgende rekursive Beziehung auch auf die Aktionswertfunktion angewandt werden [DDZ20, S. 68]:

$$Q_\pi(s, a) = \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) \left[\mathcal{R}(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s', a') \right] \quad (2.12)$$

2.2.2.1 Optimale Wertefunktionen

Wie bereits erläutert besteht das Ziel eines Agenten darin, eine Policy zu lernen die den erwarteten Return auf Dauer maximiert. Existiert eine Policy, die besser oder genauso gut wie alle anderen Policies ist, so ist diese eine op-

timale Policy π_* [SB18, S. 62]. Optimale Policies teilen alle dieselbe Zustands- und Aktionswertfunktionen, die folgendermaßen definiert sind [SB18, S. 63]:

$$V_*(s) \doteq \max_{\pi} V_{\pi}(s) \text{ für alle } s \in \mathcal{S} \quad (2.13)$$

$$Q_*(s, a) \doteq \max_{\pi} Q_{\pi}(s, a) \text{ für alle } s \in \mathcal{S} \text{ und } a \in \mathcal{A} \quad (2.14)$$

Weiterhin existiert folgende Beziehung zwischen optimaler Zustands- und Aktionswertfunktion:

$$Q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma V_*(S_{t+1}) | S_t = s, A_t = a] \quad (2.15)$$

Leitet man die Bellman Gleichung für die optimale Zustands- und Aktionswertfunktion ab, so erhält man folgende Bellman Optimalitätsgleichungen [DDZ20, S. 70]:

$$V_*(s) = \max_a \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s, a)} [\mathcal{R}(s, a, s') + \gamma V_*(s')] \quad (2.16)$$

$$Q_*(s, a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s, a)} \left[\mathcal{R}(s, a, s') + \gamma \max_{a'} Q_*(s', a') \right] \quad (2.17)$$

wobei $s' \sim p(\cdot|s, a)$ den nächsten Zustand s' bestimmt, basierend auf der Zustandsübergangsfunktion \mathcal{P} und Ausgangszustand s und -aktion a . Indem diese Bellman Optimalitätsgleichungen für eine Umgebung gelöst werden, erhält man die optimale Policy π_* , da hierdurch für jeden Zustand die optimale Aktion berechnet werden kann [DDZ20, S. 47].

2.2.3 Monte-Carlo Methoden

Zu Monte-Carlo (MC)-Methoden werden stichprobenbasierte Ansätze gezählt [DDZ20, S. 78-79]. Einer Probe in RL wird durch eine Episode dargestellt. Hierbei werden potentiell große Mengen an Episoden in einer Umgebung gesammelt um eine akkurate Schätzung der Wertefunktion vorzunehmen. Die entstehende Einschätzung der Nützlichkeit einzelner Zuständen in der Umgebung basiert somit lediglich auf gesehenen Erfahrungen. Es werden somit keine Schätzungen von den Werten einzelner Zustände vorgenommen, sondern lediglich die von der Umgebung gesammelten Rewards genutzt. Hierfür werden Episoden vollständig durchlaufen und der kumulierte Reward G_t für die einzelnen Zeitschritte $t \in T$ in einer Episode auf Grundlage des beobachteten Returns berechnet [DDZ20, S. 81]. Dies hat den Vorteil, dass diese Methoden geringes Wissen über die Umgebung benötigen, da durch diesen Trail-And-Error Ansatz und die dadurch gesammelten Returns eine Abschätzung der Werte der einzelnen Zustände stattfinden kann [DDZ20, S. 78]. Dies unterliegt jedoch der Annahme, dass genügend Transitionen in der

Umgebungen gesammelt werden können [SB18, S. 92].

Andererseits können einzelne Episoden eine große Länge erreichen und sub-optimale Transitionen beinhalten. Dies kann zu einer hohen Varianz in der Einschätzung der Nützlichkeit einzelner Zustände führen [SB18, S. 96]. Weiterhin kann durch zufällig generierte Episoden in der Umgebung nicht sichergestellt werden, dass jedes Zustands-Aktions-Paar vom Agenten gesehen worden ist. Zwar gibt es Ansätze hierzu, wie dies in kleinen Zustands- und Aktionsräumen sichergestellt werden kann, dies setzt jedoch ebenfalls voraus, dass eine die Menge der gesammelten Transitionen in der Umgebung groß genug ist [SB18, S. 92].

2.2.4 *Dynamische Programmierung und Temporal-Difference-Learning*

Temporal Difference (TD)-Methoden nutzen eine Kombination aus MC-Ansätzen, welche nur die tatsächlich generierten Erfahrungen in einer Umgebung nutzen, und der Dynamischen Programmierung (DP). Hierfür wird zunächst DP kurz erläutert.

2.2.4.1 *Dynamische Programmierung*

DP stellt eine Methode dar um komplexe Probleme zu lösen, indem diese in kleine Teilprobleme aufgeteilt werden [Bel54]. Hierbei werden diese kleineren Teilprobleme gelöst und die Ergebnisse hierfür zur Lösung des komplexen Gesamtproblems zusammengefügt. Aufgrund dieses Ansatzes müssen Probleme drei Kriterien erfüllen, damit diese Methode dort angewendet werden kann [DDZ20, S. 73]. Diese sind:

1. Vollständiges Wissen über die Problemdomäne
2. Optimale Teilstruktur
3. Überlappende Teilprobleme

Bezogen auf ein RL-Problem bedeutet dies, dass (1) die Reward- und Zustandsübergangsfunktion eines MDP vollständig bekannt sein muss [DDZ20, S. 73]. Weiterhin (2) muss gegeben sein, dass die optimale Lösung eines Problems in Teillösungen kleinerer Probleme aufteilbar sein muss [Saj18, S. 2]. Und zuletzt (3) muss die Anzahl der Teilprobleme endlich sein, diese Teilprobleme rekursiv auftreten und ihre Ergebnisse speicherbar sein müssen [Saj18, S. 3].

Da RL-Probleme als MDP dargestellt werden können und die optimale Lösung eines Problems durch Lösung der in Kapitel 2.2.2.1 bestimmten Bellman Optimalitätsgleichungen 2.16 und 2.17 bestimmt werden kann, sind die Kriterien (2) und (3) bereits erfüllt [DDZ20, S. 73]. Somit muss für die Nutzung von DP in RL lediglich Bedingung (1) erfüllt werden.

2.2.4.2 Temporal-Difference Learning

Während somit **MC**-Methoden lediglich tatsächlich gesammelte Erfahrungen in einer Umgebung zum Lernen nutzen, nutzt **DP** die Lösungen einzelner Teilprobleme für die Lösung eines komplexeren Gesamtproblems. Hierdurch ist die Schätzung des Wertes eines Zustandes in **MC**-Methoden von den gesehenen Episoden abhängig und kann somit eine hohe Varianz aufweisen. Andererseits ist dies bei **DP**-Methoden abhängig von den Lösungen der Teilprobleme. Dieses Bootstrapping kann zu einem höheren Bias in den Schätzungen führen. [SB18, S. 89]

TD-Learning bildet eine Brücke zwischen beiden Methoden, indem aktuell gesehene Erfahrungen mit Bootstrapping kombiniert werden [SB18, S. 119]. Hierfür wird für die Evaluierung nicht wie bei **MC**-Methoden bis zum Ende einer Episode gewartet, sondern die während der Episode beobachteten Werte einzelner Zustände genutzt. Es wird der Unterschied zwischen dem gesehenen Werten eines Zustandes in der aktuellen Episode und dem geschätzten, antizipierten Werte des Zustandes bestimmt um die Wertefunktion zu aktualisieren. Es wird somit Teile aus **MC** genutzt, indem der aktuell beobachtete Wert eines Zustandes betrachtet wird, sowie Teile aus **DP**, da der antizipierte Wert dieses Zustandes mit berücksichtigt wird. Dies hat den Vorteil, dass wie bei **MC** nicht die Reward- und Zustandsübergangsfunktion vorher bekannt sein muss, aber auch nicht wie bei **MC** bis zum Ende einer Episode für die Evaluierung und Optimierung der Wertefunktion gewartet werden muss, sondern dies nach einem beliebigen Zeitschritt in der Episode stattfinden kann. Findet diese Evaluierung nach jedem Zeitschritt $t + 1$ statt wird dies als Ein-Schritt oder $TD(0)$ bezeichnet. Dies kann um einen beliebigen Zeitschritt n zu $TD(n)$ erweitert werden. [DDZ20, S. 83]

2.3 DEEP REINFORCEMENT LEARNING

Deep Reinforcement Learning (DRL) kombiniert *Deep-Learning* auf Grundlage von **NN** mit **RL** [DDZ20, S. XXVI]. Klassisches **RL** basiert auf Nutzung von Tabellen, in denen Zustands- und Aktionswerte einzelner Zustände und Zustands-Aktions-Kombinationen festgehalten werden. In diesen Fällen können oft exakte Lösungen für **RL**-Probleme gefunden werden [SB18, S. 23]. Durch die Kombination von **NN** und **RL** werden auch hochdimensionale Zustandsräume für **RL** zugänglich. Durch die Approximationsfähigkeit nichtlinearer Funktionen von **NN** können beispielsweise relevante Informationen des Zustandsraumes für das approximieren von Wertefunktionen genutzt werden.

2.3.1 Taxonomie Lernalgorithmen

RL-Algorithmen lassen sich in verschiedene Kategorien unterteilen. In diesem Abschnitt werden einzelne Kategorien und Unterschiede aufgezeigt, um in der späteren Arbeit passende **RL**-Algorithmen auf Grundlage ihrer Ei-

genschaften auswählen zu können. Hierbei findet eine Unterscheidung zwischen modellbasierten und modelfreien Methoden, wertebasierten, Policy-basierten und Actor-Critic Methoden, On-Policy und Off-Policy Methoden sowie stochastische und deterministische Policies statt.

2.3.1.1 Modellbasiert und Modelfreie Methoden

Modellbasierte und modelfreie RL-Methoden unterscheiden sich in dem vorausgesetzten Wissen über die Umgebung. Im Kontext von modellbasierten und modelfreien RL-Methoden bezeichnet ein Modell eine Ansammlung von Wissen über die jeweilige Umgebung. Wie bereits erläutert wird eine Umgebung durch ein MDP mit den Elementen $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ beschrieben. Besteht vollständiges Wissen über all diese Eigenschaften der Umgebung können Planungsmethoden genutzt werden die keine Interaktion mit der Umgebung benötigen. Dies ist jedoch oft nicht der Fall, da ein Agent in der Regel keine Kenntnisse über die Rewardfunktion \mathcal{R} und die Zustandsübergangsfunktion \mathcal{P} besitzt. [DDZ20, S. 127]

Modellbasierte Methoden wird ein Modell der Umgebung vorgegeben oder sie versuchen dieses Modell durch Interaktionen mit der Umgebung zu erlernen. Hierdurch ist es möglich Planungsmethoden zu nutzen, was zu einer hohen Probeneffizienz dieser Algorithmen führt. Dies bedeutet, dass eine geringere Anzahl Trainingsdaten für das Trainieren des Agenten benötigt werden, da die Dynamiken der Umgebung vorgegeben sind [DDZ20, S. 128]. Unter anderem konnte mit modellbasierten Methoden der Agent AlphaZero [Sil+17a] trainiert werden, indem der Agent explizites Wissen über die Spielregeln der Spiele Schach, Go und Shogi verfügte.

Durch Interaktion mit der Umgebung kann auch ein Modell dieser erlernt werden [DDZ20, S. 128]. Hierbei werden Proben durch Interaktion einer zufälligen Policy gesammelt, welche genutzt werden um ein Vorhersagemodell der Umgebung zu lernen. Dieses Modell wird anschließend genutzt um die Policy zu optimieren.

Hierfür wird jedoch vorausgesetzt, dass das Modell die jeweilige Umgebung präzise beschreibt. Dies stellt einen Nachteil dieser Methoden dar, da ein explizites Modell der Umgebung oft nicht vorhanden ist oder die Dynamiken der Umgebung komplex und schwer erlernbar sind. Hierbei ist zu beachten, dass Abweichungen zwischen Modell und Umgebung dazu führen können, dass der Agent eine optimale Policy für das Modell lernt, diese jedoch nicht optimal in der jeweiligen Umgebung ist [DDZ20, S. 128].

Alternativ hierzu gibt es modelfreie Methoden. Diese brauchen kein Modell der Umgebung und versuchen nicht ein Modell zu erlernen. Stattdessen wird die jeweilige Policy direkt optimiert, was auch die Implementierung von Algorithmen dieser Art vereinfacht. Gleichzeitig sind modelfreie Methoden weniger probeneffizient im Vergleich zu modellbasierten Methoden, was zu Problemen bei deren Einsatz in der Realität führt. [DDZ20, S. 128]

2.3.1.2 Wertebasierte, policy-basierte und Actor-Critic Methoden

Wie bereits erläutert kann der Wert eines Zustandes durch die Zustandswertfunktion quantifiziert werden. Gleichzeitig wurde in Kapitel 2.2.2 gezeigt, dass durch Lösung der Bellman Optimalitätsgleichungen eine optimale Policy erlernt werden kann.

WERTEBASIERTE METHODEN versuchen die Wertefunktion der jeweiligen Umgebung zu lernen und auf Grundlage dieser die Aktionen auszuführen, um den zu erwarteten Wert zu maximieren. Hierfür wird in der Regel die Aktionswertfunktion $Q_\pi(a, s)$ genutzt [DDZ20, S. 129]. Somit wird nach der Aktionswertfunktion optimiert und nicht nach der Policy. Die Policy ist somit nur implizit erlernt, ein Beispiel hierfür ist die Nutzung einer ϵ -greedy Policy [Sil15, Kap. 7.4]:

$$\pi(s) = \begin{cases} \arg \max_a Q_\pi(s, a) \text{ mit Wahrscheinlichkeit } 1 - \epsilon \\ \text{Zufälliges } a \in \mathcal{A} \text{ mit Wahrscheinlichkeit } \epsilon \end{cases} \quad (2.18)$$

Vorteile dieser Methoden sind, dass sie eine hohe Probeneffizienz aufweisen und der Wert eines Zustandes mit geringer Varianz geschätzt werden kann, sowie dass diese Methoden nicht leicht in ein lokales Optimum fallen. Zu den Nachteilen gehören, dass diese normalerweise nicht für kontinuierliche Aktionsräume geeignet sind und, dass durch die Nutzung des max Operators in einer ϵ -Greedy Strategie zu einer Überschätzung der Werte geneigt wird [DDZ20, S. 129]. Weiterhin können wertebasierte Methoden nur deterministische Policies lernen [Sil15, Kap. 7]. Zwar ist durch die Nutzung einer ϵ -greedy Policy eine gewisse Stochastizität gegeben, da mit einer Wahrscheinlichkeit von $1 - \epsilon$ bzw. ϵ eine optimale bzw. zufällige Aktion ausgeführt wird. Hierdurch kann jedoch keine explizite stochastische Policy erlernt werden, wie sie beispielsweise bei Schere-Stein-Papier oder einer aliasierten Gridworld erforderlich wäre [Sil15, Kap. 7].

POLICY-BASIERTE METHODEN lernen im Gegensatz dazu keine Wertefunktion sondern optimieren die Policy direkt. Diese Optimierung findet iterativ statt bis der gesammelte Return maximiert ist [DDZ20, S. 130]. Somit ist das Ziel von policy-basierten Methoden die direkte Optimierung einer Policy π um den diskontierten erwarteten Return $J(\pi)$ zu maximieren [DDZ20, S. 104].

Vorteile dieser Methode sind, dass sie von einer besseren Konvergenz profitieren und auch in kontinuierlichen oder hoch-dimensionalen Aktionsräumen nutzbar sind [DDZ20, S. 130]. Weiterhin können diese Methoden auch stochastiche Policies lernen. Nachteile hierbei sind jedoch ihre Tendenz eher in lokalen Optima zu konvergieren. [Sil15, Kap. 7.5]

ACTOR-CRITIC METHODEN stellen eine Kombination zwischen werte- und policy-basierten Methoden dar. Hierbei werden wertebasierte Methoden genutzt um eine Wertefunktion zu lernen um die Probeneffizienz zu erhöhen,

sowie policy-basierte Methoden um eine Policy zu lernen, die sowohl in diskreten als auch kontinuierlichen Aktionsräumen eingesetzt werden kann [DDZ20, S. 130]. Der Actor repräsentiert hierbei die optimierte Policy und der Critic die zu lernende Wertefunktion. Es findet eine Optimierung des Actors unter Berücksichtigung der vom Critic bereitgestellten Informationen. [KToo]

Actor-Critic Methoden vereinen die Vorteile von wertebasierten und policy-basierten Methoden, was ihren Einsatz populär macht. Gleichzeitig bringen sie aber auch Nachteile beider Methoden mit, indem beispielsweise das Problem der Überschätzung von Werten wie bei wertebasierten Methoden oder das Problem der geringen Exploration von policy-basierten Ansätzen. [DDZ20, S. 130]

2.3.1.3 Off-Policy und On-Policy Methoden

Die Unterscheidung zwischen Off-Policy und On-Policy Methoden richtet sich nach der für die Evaluierung bzw. Optimierung bestehenden Anforderungen an die genutzte Policy in der jeweiligen Umgebung. On-Policy Methoden setzen voraus, dass die zu optimierende Policy selbst in der Umgebung eingesetzt wird und Entscheidungen für Aktionen trifft. Mittels dieser Daten findet dann eine Evaluierung und Optimierung der Policy statt. Ein Beispiel hierfür ist ein Algorithmus, der auf Grundlage der angewandten Aktionen und dem erhaltenen Return eine Optimierung der Policy vornimmt. Im Gegensatz hierzu stehen Off-Policy Methoden. Diese haben keine Anforderungen an die genutzte Policy in der Umgebung. Somit kann eine beliebige Policy in der Umgebung genutzt werden um die Ziel-Policy zu optimieren. Daraus folgt, dass mit Off-Policy Methoden auch Erfahrungen anderer Agenten in der Umgebung für die Optimierung einer Policy genutzt werden können. Ein Beispiel hierfür ist ein Algorithmus, der die angewandte Aktion und den zu diesem Zeitpunkt erhaltenen Reward für die Optimierung betrachtet, aber die zukünftigen Rewards auf Grundlage der eigenen Policy schätzt. Hierdurch reichen die Transitionswerte $S_t, A_t, R_{t+1}, S_{t+1}$ für eine Optimierung der Zielpolicy. [DDZ20, S. 131-132]

2.3.1.4 Stochastische und Deterministische Policy

Eine stochastische Policy $\pi(s|a)$ stellt eine Wahrscheinlichkeitsverteilung dar basierend auf Zuständen $S \in \mathcal{S}$ und Aktionen $A \in \mathcal{A}$. Gegeben eines Zustandes S_t wird somit eine Aktion A_t aus der Wahrscheinlichkeitsverteilung $\pi(\cdot|S_t)$ entnommen. [DDZ20, S. 70]

Stochastische Policies können jedoch auch in deterministische umgewandelt werden, indem stets die Aktion mit der höchsten Wahrscheinlichkeit gewählt wird. Dies hat zur Folge, dass unter Berücksichtigung eines Zustandes S_t eine deterministische Policy $\pi(S_t)$ immer die gleiche Aktion A_t bestimmt. [DDZ20, S. 70-71]

Während somit eine stochastische Policy eine Wahrscheinlichkeitsverteilung basierend auf der Kombination von Zuständen und Aktionen darstellt, ist ei-

ne deterministische Policy eine direkte Zuordnung von Aktion zu Zustand.
[DDZ20, S. 71]

EFFIZIENZDEFINITION

Im Rahmen dieser Arbeit werden [RL](#)-Algorithmen genutzt um Agenten das Tischkickerspielen beizubringen. Ziel der Arbeit ist es, das hierfür benötigte Training möglichst effizient zu gestalten. Hierfür ist zunächst zu klären, was unter "effizientem" Training in dieser Arbeit verstanden wird und welche Kriterien hieraus auf die Arbeit folgen. Zunächst werden unterschiedliche Effizienzdefinitionen erläutert und für diese Arbeit eine Definition festgelegt. Im Anschluss werden die Folgen dieser Definitionswahl für die Arbeit diskutiert.

Effizienz kann in maschinellem Lernen von verschiedenen Faktoren abhängig gemacht werden. Gerade in Bezug auf Algorithmen wird oft in den Kategorien Probeneffizienz und Recheneffizienz unterschieden [[Kako3](#); [Yu18](#); [Che+21](#)]. Es werden nun beide Effizienzklassen näher erläutert und anschließend eine Wahl für diese Arbeit getroffen.

3.1 PROBENEFFIZIENZ

Probeneffizienz bezieht sich auf die Anzahl der vorhandenen Proben, die während des Trainings vom Agenten genutzt werden, um eine bestimmte Leistung zu erzielen [[Yu18](#)]. Die Leistung kann in [RL](#) durch den gesammelten Return dargestellt werden. Wie in den Grundlagen bereits definiert kann eine Probe in [RL](#) durch eine Transition in einer Umgebung als folgendes Tupel dargestellt werden $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$.

Vergleicht man probeneffiziente und nicht-probeneffiziente Lernalgorithmen, so benötigen probeneffiziente Algorithmen weniger Proben um die gleiche Leistung wie nicht-probeneffiziente zu erzielen. Ist somit die Anzahl an Probendaten begrenzt, so ist ein möglichst probendateneffizienter Lernalgorithmus von Vorteil, da dieser aus den vorhanden Daten eine möglichst gute Policy erlernen kann. [[DDZ20](#), S. 249-250]

3.2 RECHENEFFIZIENZ

Die Recheneffizienz gibt die Sparsamkeit von Algorithmen in Bezug auf Rechenressourcen an [[Kla12](#)]. Diese Rechenressourcen können der Speicher darstellen oder die benötigte Zeit für Berechnungen. Da [NN](#) vor allem eine hohe Zeitkomplexität für deren Optimierung aufweisen, ist somit die Zeitkomplexität vordergründig [[Roj13](#), S. 274-275]. Üblicherweise erfolgt die Darstellung mittels \mathcal{O} -Notation, indem die eine asymptotische Schranke für die benötigten Ressourcen definiert [[Kla12](#)]. Vergleicht man recheneffiziente mit nicht-recheneffizienten Algorithmen in Bezug auf deren Zeitkomplexität, so haben recheneffiziente Algorithmen eine geringere Laufzeit als nicht-

recheneffiziente Algorithmen, um zum gleichen Ergebnis zu kommen. Ein Beispiel hierfür sind verschiedene Sortieralgorithmen, wobei *Merge Sort* mit einer Zeitkomplexität von $\mathcal{O}(n \log(n))$ effizienter ist bzw. eine geringere Laufzeit aufweist als *Insertion Sort* mit einer Komplexität von $\mathcal{O}(n^2)$ [RS14].

3.3 AUSWAHL EINER EFFIZIENZDEFINITION

Nun ist zu Prüfen, welche dieser Definitionen sich zur Nutzung im Rahmen dieser Arbeit besser eignet. Hierfür ist zu bestimmen ob Probendaten oder Rechenressourcen begrenzter sind. Im Rahmen dieser Arbeit werden Agenten in einer Simulation trainiert. Somit gibt es grundsätzlich keine Begrenzung von Probendaten die erstellt werden können.

Jedoch stehen nur zwei Laptops mit den Spezifikationen aus Tabelle A.1 im Rahmen dieser Arbeit zur Verfügung. Gleichzeitig sind potenziell mehrere Versuche durchzuführen. Daraus folgt, dass im Rahmen dieser Arbeit die Rechenressourcen begrenzt sind und diese somit auch die Generierung von Probendaten begrenzen. Es ist somit keine der beiden Effizienzdefinitionen explizit auszuschließen, sondern eine Balance zu finden mit der Agenten in möglichst kurzer Zeit trainiert werden können. Hierfür können in Bezug auf Algorithmen zum einen welche genutzt werden, die eine bessere Recheneffizienz aufweisen als andere, oder probeneffiziente Algorithmen, die weniger Proben benötigen um eine gewisse Leistung zu erzielen. Grundsätzlich ist also für die einzelnen Komponenten des Trainingskonzept zu beachten, inwieweit diese das RL-Problem einfacher zu erlernen machen und somit die benötigte Trainingsdauer reduzieren. Auf diese Bedingung ist bei der Auswahl der Komponenten stets zu achten.

4

TISCHKICKERSIMULATION

Das Training der Agenten kann auf dem Bosch Rexroth Kickertisch oder in einer Simulation auf einem Rechner stattfinden. Es existieren jedoch eine Vielzahl von Gründen die dafür sprechen, das Training des Agenten in einer Simulation stattfinden zu lassen. Das Training von [RL](#)-Agenten kann lange Zeit in Anspruch nehmen [[Ope+19a](#), S. 8]. Oft werden hierfür mehrere zehntausende Episoden benötigt. Gleichzeitig muss nach Ablauf einer Episode die nächste Episode vorbereitet werden. Dies wird durch Nutzung einer Simulation effizienter gestaltet. Zudem wird dadurch Unabhängigkeit gewonnen, da die Simulation auf einem beliebigen Rechner laufen und ggf. parallelisiert werden kann.

Jedoch hat die Nutzung einer Simulation auch Nachteile. Eine zur Realität identische Simulation lässt sich praktisch schwer umsetzen [[GK19](#)]. Es entstehen Diskrepanzen zwischen der Simulation und der Realität, die als *Simulation-To-Reality-Gap* beschrieben werden [[GK19](#)]. Durch den Simulation-To-Reality-Gap entspricht eine optimale Policy für die Simulation nicht notwendigerweise einer optimalen Policy für die Realität, wodurch der Transfer der erreichten Leistung des in der Simulation trainierten Agenten auf die Realität erschwert wird [[Ope+18](#), S. 4]. Diskrepanzen zwischen der Realität und der Simulation wären am Beispiel des Kickertisches die physikalischen Kräfte, die bei Zusammenstößen zwischen dem Ball und der Bande stattfinden, oder auch die genauen Kräfteinwirkungen auf den Spieler und den Ball während eines Schusses.

Es darf dabei nicht vergessen werden, dass die Simulation am Beispiel des Kickertisches nur ein Mittel zum Zweck ist, um das Training des Agenten zu vereinfachen. Gerade in [RL](#)-Experimenten mit Robotern, werden Agenten oft in einer Simulation trainiert und müssen ihre Fähigkeiten dann aber in der Realität unter Beweis stellen [[Ope+18](#)]. Somit ist es erforderlich, den Simulation-To-Reality-Gap möglichst gering zu halten. Darauf ist auch in dieser Arbeit zu achten.

Zunächst wird der Aufbau des Kickertisches von Bosch Rexroth erläutert. Anschließend wird ein Überblick der genutzten Komponenten für die Simulation und die Kommunikation mit dem Agenten dargestellt. Diese Arbeit baut auf dem Quellcode-Stand des Moduls "Masterprojekt Systementwicklung" der Hochschule Darmstadt auf. In diesem Modul werden eine Simulation- und Agentenimplementierung in Bezug auf den Bosch Rexroth Kickertisch von Studierenden stetig weiterentwickelt. Der in dieser Arbeit genutzte Quellcode-Stand der Simulations- und Agentenimplementierung entsprechen dem Abschluss vom Wintersemester 2021/2022. Zuletzt wird darauf eingegangen, ob der Stand der Implementierung weitere Maßnahmen benötigt, um den Simulation-To-Reality-Gap weiter zu verringern. Hier-

bei ist bei Entscheidung für weitere Maßnahmen darauf zu achten, ob diese Einflüsse auf die Trainingseffizienz haben könnten.

4.1 AUFBAU BOSCH REXROTH KICKERTISCH

Der Bosch Rexroth Kickertisch wurde bereits in weiteren Arbeiten genutzt. Eine detaillierte Beschreibung des Kickertischbaus kann in [DB+21] nachvollzogen werden. In diesem Abschnitt wird ein Überblick über den Aufbau gegeben, um die Qualität der Simulation beurteilen und ggf. weitere Maßnahmen treffen zu können, um die Simulation realistischer zu gestalten.

Der Bosch Rexroth Kickertisch ist in Abbildung 4.1 dargestellt.



Abbildung 4.1: Darstellung des Bosch Rexroth Kickertisches

Quelle: [DB+21, S. 3]

Es handelt sich um einen professionellen Kickertisch des Modells *Ullrich Sport U4P* [DB+21]. Dieser wurde so modifiziert, dass eine Seite von einem Menschen und die gegnerische Seite durch ein Antriebs- und Steuerungssystem bedient wird. Hierfür werden pro Stange zwei Motoren der Marke Bosch Rexroth genutzt, um die Spielerstangen rotieren und lateral bewegen zu können. Für die Rotation werden synchrone *IndraDyn S MS2N03* Motoren verwendet, während für die laterale Bewegung lineare *IndraDyn L MCL020* verwendet werden. Diese Motoren werden an Antriebsumrichter des Typs *IndraDrive Cs* angeschlossen. Die Speicherprogrammierbare Steuerung (**SPS**) *IndraControl XM21* und *XM22* kommunizieren mit den Motoren über die Sercos-Schnittstelle und übernehmen die Steuerung aller elektrischen Komponenten.

Da die einzelnen Kickerstangen verschiedene Grenzwerte für die laterale Bewegung haben, sind die Steuerungsmöglichkeiten eingeschränkt. Positioniert man die einzelnen Stangen lateral mittig, so können die einzelnen Stangen von dieser Position aus folgendermaßen lateral bewegt werden:

- Torwartstange: ± 120 mm
- Verteidigerstange: ± 180 mm
- Mittelfeldstange: ± 55 mm
- Stürmertstange: ± 115 mm

Weiterhin kann jede Stange rotiert werden. Während es an einem regulären Kickertisch keine physische Begrenzung der Rotation gibt, ist am Bosch Rexroth Kickertisch eine Begrenzung der Drehwinkel von $[-90^\circ, 90^\circ]$ implementiert [DB+21, S. 7]. Grund für diese Begrenzung ist die Nutzung einer Lichtschranke. Diese Lichtschranke des Typs *Sick miniTwin4* ist über dem Spielfeld angebracht und sorgt dafür, dass alle Motoren stoppen sobald die Lichtschranke durchbrochen wird, beispielsweise durch das Greifen in das Spielfeld.

Die Grenzen für Drehwinkel und Lateralposition werden durch das System erzwungen. Somit können Zustände außerhalb dieser Grenzen nicht erreicht werden. Dies gilt auch für die kinematischen Grenzen der Motoren. Diese sind der maximaler Ruck, Beschleunigung und Geschwindigkeit.

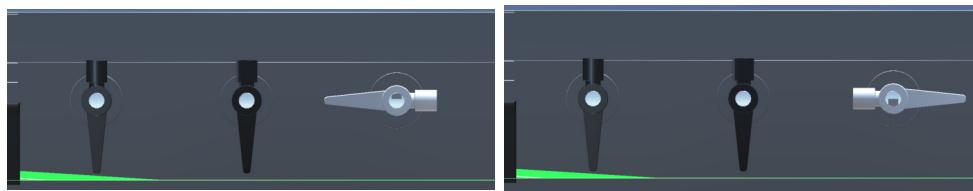
Die aktuellen Drehwinkel und Lateralpositionen der Stangen können mittels Open Platform Communications Unified Architecture ([OPC-UA](#)) Client aus dem System ausgelesen werden. Um die Ballposition erfassen zu können ist über dem Spielfeld eine Kamera angebracht, die das Spielgeschehen aufzeichnen kann.

Mittels [OPC-UA](#) Client können auch Werte für Drehwinkel und Lateralposition der einzelnen Stangen an die [SPS](#) gesendet werden. Die Bewegung der Industriemotoren wird von den Antriebsumrichtern übernommen, die über die mit Sercos übertragenen Zielwegpunkte gesteuert werden. Durch die vom Agenten übertragene Position, bestehend aus gewünschtem Drehwinkel und Lateralposition, werden vom Antriebsregler diese Zielwegpunkte als Punkte auf einer rückbegrenzten Trajektorie berechnet. [DB+21, S. 6-7]

4.1.1 Maße und Begriffsdefinitionen

In diesem Abschnitt folgen weitere Maße für den Spielfeldbereich zwischen Stürmer- und Torwartstange, die von besonderer Bedeutung für diese Arbeit sind. Hierfür erfolgt zunächst eine Begriffsdefinition für die einzelnen Tischkickerkomponenten.

In den Abbildungen [4.2](#), [4.3](#) und [4.4](#) werden die Ausprägungen der Rotation einer Kickerstange gezeigt.

Abbildung 4.2: Winkelposition 90° .Abbildung 4.3: Winkelposition -90° .Abbildung 4.4: Winkelposition 0° .

Diese Ausprägung wird im Rahmen dieser Arbeit als **Winkelposition** bezeichnet, wobei jeder Winkelposition ein Wert im Bereich $[-90^\circ, 90^\circ]$ zugewiesen wird. Ist eine Stange so ausgerichtet, dass der Spieler orthogonal zum Spielfeld steht, bekommt diese Winkelposition den Wert 0° zugewiesen. Diese Position ist in Abbildung 4.4 dargestellt. Ist der Spieler in Richtung zum gegnerischen Tor rotiert, wie in Abbildung 4.2 dargestellt, wird dieser Winkelposition der Wert 90° zugewiesen. Ist der Spieler in Richtung zum eigenen Tor rotiert, wie in Abbildung 4.3 dargestellt, wird dieser Winkelposition der Wert -90° zugewiesen.

In den Abbildungen 4.5, 4.6 und 4.7 werden die horizontalen Positionen der Kickerstange dargestellt.



Abbildung 4.5: Lateral-position mit Wert 1.



Abbildung 4.6: Lateral-position mit Wert 0.

Abbildung 4.7: Lateral-position mit Wert -1 .

Hierbei werden drei verschiedene Positionen dargestellt. Die Ausprägung der Position wird im Rahmen der Arbeit als **Lateralposition** bezeichnet, wobei jeder Lateralposition ein Wert im Bereich $[-1, 1]$ zugewiesen wird. In Abbildung 4.6 ist die Stange so positioniert, dass zu beiden Tischkickerwänden der gleiche Abstand eingehalten wird. Dieser Lateralposition wird im Rahmen der Arbeit der Wert 0 zugewiesen. Die Ausprägungen der Lateralpositionen werden wie bei der Winkelposition relativ zur Stange gesetzt. So

spielt auch hier die Ausrichtung der jeweiligen Stange und die natürliche *Blickrichtung* der Spieler zum gegnerischen Tor eine Rolle bei der Zuweisung des Wertes für die jeweilige Lateralposition. In Abbildung 4.5 ist der Abstand der Stange zur rechten Tischkickerwand minimiert, wodurch dieser Lateralposition der Wert 1 zugewiesen wird. Ist der Abstand der Stange zur linken Tischkickerwand minimiert, wie in Abbildung 4.7, wird dieser Lateralposition der Wert -1 zugewiesen. Diese Werte beziehen sich nur auf die Kontrolle der Simulation und sind somit nicht notwendigerweise von **RL**-Agenten in dieser Form zu nutzen. Diese werden in Kapitel 5 behandelt.

Nachdem die Begriffsdefinition erfolgt ist, werden weitere Maße die relevant für diese Arbeit sind aufgezeigt.

Wie bereits genannt besitzt das Tischkickerspielfeld eine Dimension von 120 cm x 68 cm. Die Spieler haben vom Mittelpunkt der Stange bis zur Fußspitze eine Länge von 7,2 cm. Beträgt die Winkelposition der Spieler 0° , beträgt die Distanz zwischen "Fußspitze" eines Spielers und dem Spielfeldboden 6 mm. Der Ball hat einen Durchmesser von 3,5 cm. Beide Tore haben eine Breite von 20 cm. Weiterhin beträgt die Distanz der Stürmerstange zum Tor 20,4 cm.

4.2 ÜBERBLICK IMPLEMENTIERUNG DER SIMULATION

Die Simulation ist in Unity3D mit Version 2020.3.32f1 implementiert. Unity3D, von nun an Unity genannt, ist eine Spiele-Engine die das Modellieren von zwei- und dreidimensionalen Spielen erlaubt [CM+14]. Diese genutzte Unity-Simulation des Wintersemesters 2021/2022 besitzt bereits eine maßstabsgerechte Nachbildung des Tischkickers von Bosch Rexroth. Der Tischkicker in der Simulation wurde mit Hilfe eines CAD-Modells von Bosch Rexroth implementiert [DB+21, S. 8]. Nun folgt ein Überblick über die genutzte Softwarearchitektur und Kommunikation zwischen der Simulation und dem Agenten.

4.2.1 Kommunikation zwischen Simulation und Agent

Im Anhang stellt Abbildung A.2 einen Überblick über die genutzten Komponenten dar. Der Agent ist mittels Python implementiert kommuniziert mittels der Unity-Simulation über das Netzwerk. Für die Kommunikation wird das Publish-Subscribe-Pattern genutzt, die konkrete Implementierung nutzt Zero-MQ [Zmqb]. Da das Publish-Subscribe-Pattern nur unidirektionale Datenfluss unterstützt, sind sowohl Agent als auch Simulation jeweils Publisher und Subscriber. Während die Simulation auszuführende Aktionen abboniert, abboniert der Agent die Observationsdaten der Simulation. Die Observationsdaten werden jedoch in Teilen an den Agenten gesendet.

Es gibt zwei Arten von Observationsdaten: Balldaten und Stangendaten. Da beide Daten mittels demselben Kanal übertragen werden, kann immer nur eine Information nach der anderen empfangen werden. Diese Informatio-

nen werden dann von der Komponente *GameStateTracker* aus Abbildung A.2 aktualisiert. Somit werden nie gleichzeitig Ball- und Stangendaten an den Agenten übertragen.

Neben den Observations- und Aktionsdaten gibt es auch eine ereignisbasierte Kommunikation. Folgende Ereignisse sind aktuell implementiert:

- GoalEvent
- ResetEvent
- PlayerTouchedBallEvent

Findet ein Ereignis in der Simulation statt so werden Ereignisdaten publiziert. Hierfür wird derselbe Publikationskanal genutzt wie für die Observations. Aktuell wird somit beim stattfinden eines Ereignisses keine Observation an den Agenten gesendet.

Weiterhin lässt sich feststellen, dass die Simulation ein *ResetEvent* an den Agenten senden kann. Dies ist der Fall, da in der Unity-Simulation die Regeln für das Beenden einer Episode definiert werden. Wird ein *ResetEvent* vom Agenten empfangen, beendet dieser die jeweilige Episode für den Agenten. Somit besitzt die Simulation aktuell die Hoheit darüber, wie eine Episode aussieht und wann diese zu Ende ist.

Wie bereits erläutert wird das Publish-Subscribe-Pattern für die Kommunikation genutzt. Dieses Pattern findet bei Multicast oder Gruppennachrichten Anwendung. Es werden alle Nachrichten an alle Empfänger gesendet. Ziel des Patterns ist es möglichst Skalierbar zu sein. Es sollen große Datenmengen schnell an viele Empfänger versandt werden können. Um dies zu erreichen wird sogenanntes *back-chatting* eliminiert.[Hin13]

Somit ist der Nachrichtenfluss unidirektional. Dadurch gibt es jedoch keine Möglichkeit der Koordination zwischen Sender und Empfänger, da Empfänger keine Antwort an den Sender schicken können. Da Publisher nur mit maximaler Geschwindigkeit senden können und Empfänger nicht mit dem Publisher kommunizieren können, kann sich eine Warteschlange an Nachrichten beim Empfänger bilden.[Hin13] Damit der Empfänger bei zu vielen Nachrichten nicht abstürzt ist ein Grenzwert gesetzt für die maximale Länge der Warteschlange. Dieser Wert heißt bei Zero-MQ *high-water mark*. Wird dieser Wert überschritten, gehen neue Nachrichten verloren [Zmq].

KOMMUNIKATION VON AKTIONEN Wie zuvor erwähnt werden auch Aktionen mittels Publish-Subscribe-Pattern der Unity-Simulation mitgeteilt. Die aktuelle Implementierung sendet eine Zielposition (Winkel- oder Lateralposition) an die Simulation, zu der sich der Agent bewegen soll. Diese Zielposition wird jedoch wie die Observation nur in Teilen übertragen. So werden Zielwinkelposition und Ziellateralposition einzeln über denselben Kanal vom Agenten publiziert. Somit kann sich eine Stange gleichzeitig nur in einer Dimension bewegen, entweder kann sie rotieren oder sich lateral bewegen.

4.2.2 Physikalische Eigenschaften der Simulation

In diesem Abschnitt wird die aktuelle Implementierung in Bezug auf die Bewegung der Stangen und das Kollisionsverhalten des Balles näher betrachtet. Beise Aspekte haben großen Einfluss auf den Simulation-To-Reality-Gap von Simulation zum realen Kickertisch.

BEWEGUNG DER STANGEN Die Stangen und Figuren sind als kinematische Objekte in Unity implementiert und werden mittels Interpolation bewegt. Dies bedeutet, dass die Figuren nicht von der Physik-Engine beeinflusst werden und somit keine physikalischen Kräfte auf diese einwirken können. Dies hat zum einen den Vorteil, dass nicht-kinematische Objekte nach belieben reibungslos zu Zielpositionen bewegt werden können. Dabei haben Kraftauswirkungen keinen Einfluss auf diese Objekte.[Tecb] Dies ist jedoch nicht der Fall in der Realität. Eine Stange kann nicht mit beliebiger Geschwindigkeit beliebig oft die Position ändern, da physikalische Kräfte wie Trägheit und Reibung in der Realität eine Rolle spielen. Aktuell findet dies jedoch keine Berücksichtigung in der Simulation.

KOLLISIONSVERHALTEN Wie bereits erläutert sind Stangen und Figuren kinematische Objekte in der Simulation. Dies hat jedoch negative Auswirkungen auf Kollisionen, da diese Objekte nicht von der Physik-Engine betrachtet werden und somit die Kollisionserkennung nur spekulativ erfolgen kann, was bei hohen Geschwindigkeiten fehleranfällig ist [Tecc][Teca]. Findet eine Kollision zwischen nicht-kinematischem Ball und kinematischer Figur statt, so hat die Kollision nur eine Kraftausübung auf den Ball. Da die Erkennung der Kollision für kinematische Objekte spekulativ erfolgt, kann es bei hohen Geschwindigkeiten dazu kommen, dass eine Figur den Ball durchdringen kann und erst anschließend eine Kollision erkannt wird. Dies hat zur Folge, dass die Bewegungsrichtung die der Ball nach der Kollision haben sollte falsch sein kann. Betroffen ist hiervon jedoch nicht nur die anschließende Bewegungsrichtung des Balls. Da die Figuren nicht von der Physik-Engine betrachtet werden, können diese zum einen keine Kraft erfahren, zum anderen aber auch keine Kraft auf andere Körper ausüben. So mit kann es bei Kollisionen zu unrealistischen Geschwindigkeiten des Balles kommen.[Tecd]

4.2.3 Interpretation der Implementierung

Damit sichergestellt werden kann, dass der Agent immer die neusten Observationsdaten der Umgebung besitzt, muss die Kommunikation zwischen Agenten und Simulation überarbeitet werden. Zum einen muss der Kommunikationsfluss der Anwendungen dahingehend angepasst werden, das Informationen wie Observationen nicht in Teilen sondern vollständig gesendet und empfangen werden. Gleichzeitig sollte keine Observation verloren gehen, wenn ein Ereignis in der Simulation stattfindet. Zudem ist sicherzu-

stellen, das der Agent immer die aktuellsten Observationen der Simulation erhält und es nicht zur Bildung einer Warteschlange beim Empfänger kommen kann.

Da beim realen Kickertisch die Steuerung der Stangen durch zwei unabhängige Motoren übernommen wird, ist es dort möglich gleichzeitig die Stange zu rotieren und lateral zu bewegen. Damit die Simulation in dieser Hinsicht die Bewegungsmöglichkeiten der Stangen korrekt darstellt, muss es die Möglichkeit geben, gleichzeitig Zielwinkel- und Ziellateralposition der Simulation mitteilen zu können, um eine Bewegung beider Achsen zeitgleich zu ermöglichen.

Da die Kollisionserkennung und -behandlung zwischen nicht-kinematischem Ball und kinematischen Figuren aktuell zu unrealistischem Verhalten führt, ist dies zu korrigieren um die Dynamiken der Umgebung realistischer zu gestalten.

4.3 MASSNAHMEN ZUR VERRINGERUNG DES SIMULATION-TO-REALITY-GAPS

In diesem Abschnitt werden Maßnahmen getroffen und implementiert, um die herausgearbeiteten Punkte zu beheben und den Simulation-To-Reality-Gap zu verringern.

4.3.1 Kommunikation

Zunächst ist die Kommunikation zwischen Simulation und Agenten anzupassen. Hier muss sichergestellt werden, dass zu jedem Zeitpunkt vollständige und die aktuellsten Observationsdaten an den Agenten geschickt werden können. Dies hat auch eine Anpassung des zur Kommunikation genutzten Datenmodells zur Folge. Im diesem Abschnitt werden Anforderungen für beide Teile abgeleitet und ein Konzept bestimmt und implementiert, das diese Anforderungen erfüllt.

4.3.1.1 Kommunikationsmethode

Wie in den Grundlagen beschrieben werden Aktionen von Agenten auf Grundlage von wahrgenommenen Observationen vorgenommen. Somit ist eine bidirektionale Kommunikation zu ermöglichen. Erreichen Observationen nun verspätet den Agenten, so kann es sein, dass sich der Zustand der Umgebung bereits verändert hat wenn eine Aktion vom Agenten ausgewählt worden ist. Gleichermaßen gilt für eine verzögerte Ausführung der Aktionen. Werden Aktionen verspätet ausgeführt, so kann es sein, dass sich der Zustand der Umgebung im Vergleich zur zuletzt gesehenen Observation verändert hat. Dies kann zur Folge haben, dass eine eben noch nützliche Aktion nun kontraproduktiv sein kann. Ein Beispiel hierfür wäre das Rotieren einer Spielerstange. Wird diese zu spät rotiert, so kann sich die Ballposition verändert haben und der Ball in eine unerwünschte Richtung geschossen werden. Dar-

aus folgt, dass die Kommunikation zwischen Simulation und Agenten möglichst schnell stattfinden soll.

Da es sich beim Tischkicker um ein Echtzeitspiel handelt ist der aktuelle Zustand der Umgebung von Bedeutung um eine geeignete Aktion zu wählen. Es darf somit nicht zur Warteschlangenbildung von Observationen kommen, mit der der Agent Aktionen auf Grundlage vergangener Observationen trifft, die keine Relevanz mehr für den aktuellen Zustand der Umgebung sind. Es muss somit eine möglichst schnelle, bidirektionale Kommunikation möglich sein, die keine Warteschlangen nutzt. Hierbei ist festzuhalten, dass die Kommunikation zwischen Agenten und Umgebung von synchroner Natur ist. Es wird eine Aktion ausgeführt und eine neue Observation wahrgenommen.

Hierfür eignet sich die Nutzung von synchronen Remote Procedure Calls ([RPC](#)), die einen Funktionsaufruf zwischen unterschiedlichen Systemen ermöglichen. Eine auch in anderen hoch-performanten [RL](#)-Systemen genutzte Implementierung hierfür ist general-purpose Remote Procedure Calls ([gRPC](#)) [[Ope+19a](#), S. 51]. [gRPC](#) bietet eine leichtgewichtige Implementierung an um schnell synchrone Funktionsaufrufe zwischen zwei Systemen zu gewährleisten und eignet sich besonders für effiziente Echtzeitkommunikation zwischen Umgebungen die unterschiedliche Sprachen nutzen. Die übertragenen Daten werden in binärem Datenformat übertragen, was das Parsen der Daten effizienter macht. [[Aut](#)] In ihrer Arbeit [[Ope+19a](#), S. 51] nutzten die Autoren einen [gRPC](#)-Server in der Spiele-Umgebung, über den die Kommunikation zwischen Agent und Umgebung stattfand. Sowohl die in [[Ope+19a](#)] genutzte Umgebung als auch der Ausgangsquellcode dieser Arbeit nutzen das Umgebungsmodell OpenAI Gym [[Bro+16a](#)]. Dieses sieht vor, dass bei Ausführung einer Aktion durch den Agenten eine Observation als Rückgabewert durch die Umgebung geliefert wird. Dies kann so dargestellt werden, dass die Simulation eine Methode `execute_action(action a)` anbietet, die als Parameter eine Aktion a nutzt und als Antwort eine Observation bereitstellt.

4.3.1.2 Datenmodell

Damit zu jedem Zeitpunkt die ganze Umgebung observiert werden kann, muss zu jedem Zeitpunkt Zugriff auf alle aktuellen Observationsdaten vorhanden sein. Hierfür wird ein Datenmodell benötigt, dass Informationen bezüglich des Balls, der Stangen und Ereignisse beinhaltet kann. So gehen keine Informationen verloren, wenn ein Ereignis stattfindet und die Ball- und Stangenpositionen sind vom gleichen Zeitpunkt, wodurch zeitliche Verzögerungen zwischen den Positionsdatenständen vermieden werden.

Aktuell wird als Datenformat [JSON](#) für die Kommunikation zwischen Simulation und Agenten genutzt. Für das neue Datenmodell werden die unterschiedlichen Nachrichtenarten (Ball-, Stangen- und Ereignisdaten) in eine Nachrichtenart gebündelt. Dieses ist im Anhang in Abbildung [A.3](#) dargestellt. Hierdurch kommt es zu keiner Verzögerung zwischen den einzelnen

Datenständen und es kann zu keiner zeitlichen Verzögerung einzelner Nachrichtenartefakte kommen.

4.3.2 *Physikalische Eigenschaften der Simulation*

Weiterhin sind die physikalischen Eigenschaften für Stangenbewegungen und Kollisionen zu überarbeiten. Sowohl die Stangenbewegung, als auch das Kollisionsverhalten sind realistischer zu gestalten. In diesem Abschnitt werden hierfür Anforderungen abgeleitet und ein Konzept bestimmt und implementiert, um die Simulation realistischer zu gestalten.

4.3.2.1 *Bewegung von Stangen*

Es muss darauf geachtet werden, dass die Bewegungen der Stangen in der Simulation den Bewegungen am echten Kickertisch möglichst ähnlich sind. Wie bereits erläutert werden am Bosch Rexroth Kickertisch zum Erreichen einer Zielposition Zielwegpunkte genutzt. Diese Zielwegpunkte werden vom Antriebsregler berechnet. Dieser berechnet die Zielwegpunkte als Punkte auf einer ruckbasierten Trajektorie. In [DB+21] wird hierfür die Bibliothek *opt_control* genutzt, die effiziente und parallelisierbare Generation voller Trajektorien ermöglicht. Im Rahmen dieser Arbeit wird der Nachfolger dieser externen Bibliothek genutzt, Time-optimal Trajectory Generation and Control (*TopiCo*) [BB17]. *TopiCo* erzeugt effizient («1ms pro Achse pro Trajektorie) glatte, dynamisch machbare, zeitoptimale ruckbasierte Trajektorien von beliebigen Start- zu beliebigen Zielzuständen [BB17]. Es wurde erfolgreich als modellprädiktiver Regler für verschiedene Drohnen in verschiedenen Forschungsprojekten und Roboterwettbewerben eingesetzt. Da die Bibliothek in *MATLAB* implementiert ist, wird ein .NET-Assembly kompiliert und in Unity integriert. Auch in der Arbeit [DB+21] fand so die Integration von *opt_control* in die Unity-Simulation statt.

Wie bereits erläutert werden für die Bewegung der Tischkickerstangen vom Antriebsregler Zielwegpunkte auf einer ruckbasierten Trajektorie berechnet. Diese Zielwegpunkte werden auf Grundlage von aktueller Stangen- und Zielposition berechnet. Da *TopiCo* ebenfalls als Berechnungsgrundlage den Ruck der Motoren nutzt, werden die Zielwegpunkte im Bosch-Kickertisch und in der Unity-Simulation somit nach derselben Methode berechnet. Neben den Zielwegpunkten liefert *TopiCo* auch die aktuelle Geschwindigkeit, Beschleunigung und Ruck zu jedem Zielwegpunkt. Somit sind auch zu jedem Zielwegpunkt präzise Bewegungsinformationen zu den einzelnen Stangen verfügbar.

Um eine Trajektorie von *TopiCo* generieren zu lassen sind die Geschwindigkeits-, Beschleunigungs- und Ruckgrenzwerte der Kickertischmotoren zu bestimmen. Hierfür wurden mittels *OPC-UA* Client die Werte aus dem *SPS* Modul des Bosch-Kickertisches ausgelesen und als Grenzwerte für die Trajektoriengenerierung durch *TopiCo* genutzt. Da für *TopiCo* lediglich die Beträge der einzelnen Dimensionen von Relevanz sind, ist darauf zu achten, dass alle Ab-

leitungen die gleiche Einheit nutzen. Als Grenzwerte für Geschwindigkeit, Beschleunigung und Ruck wurden für die laterale Bewegung der Stangen folgende Werte ausgelesen:

- Geschwindigkeit: min. -2 m/s , max. 2 m/s
- Beschleunigung: min. -9 m/s^2 , max. 9 m/s^2
- Ruck: min. -500 m/s^3 , max. 500 m/s^3

Für die Rotationsmotoren wurden folgende Werte ausgelesen:

- Geschwindigkeit: min. $-104,72 \text{ rad/s}$, max. $104,72 \text{ rad/s}$
- Beschleunigung: min. -3.000 rad/s^2 , max. 3.000 rad/s^2
- Ruck: min. -300.000 rad/s^3 , max. 300.000 rad/s^3

Neben diesen Werten ist für die Berechnung von [TopiCo](#) noch die Startposition, Endposition und ein Zeitinkrement anzugeben. Für die Startposition werden bei Berechnung einer Trajektorie die aktuellen Stangenpositionen aus der Unity-Simulation ausgelesen. Die Zielposition wird abhängig von der Agentenaktion bestimmt. Das Zeitinkrement legt fest, wie viele Zielwegpunkte berechnet werden sollen. Bei einem Zeitinkrement von beispielsweise 10 ms werden die Zielwegpunkte jede 10 ms bis zum Erreichen der Zielposition berechnet. Um diesen Parameter zu bestimmen, wird kurz erneut auf Unity eingegangen. Unity besitzt einen Parameter, mit dem die Anzahl an Physikberechnungen pro Sekunde bestimmt werden. Dieser Parameter heißt *Fixed-Timestep* [[Tece](#)]. An solch einem Zeitpunkt werden alle Physikberechnungen durchgeführt. Beispielsweise werden hier die Stangenbewegungen umgesetzt. Das Zeitinkrement für [TopiCo](#) muss dem Parameter Fixed-Timestep entsprechen, da die berechneten Zielwegpunkte diese Zeitspanne für das Erreichen eines Zielwegpunktes voraussetzen. Weichen Zeitinkrement und Fixed-Timestep voneinander ab, so erreichen die Stangen entweder zu langsam oder zu schnell den jeweiligen Zielwegpunkt.

4.3.2.2 Kollisionsverhalten

Aufgrund der Nutzung kinematischer Figuren und Stangen können diese reibungslos bewegt werden. Dies vereinfacht die Bewegung der Stangen drastisch, da lediglich darauf zu achten ist, dass die Stangen der von [TopiCo](#) generierten Trajektorie folgen. Um das Kollisionsverhalten zwischen Ball und Figur zu beheben, muss die Kollisionserkennung und die Kollisionsauflösung verbessert werden.

Die Kollisionserkennung zwischen kinematischen und nicht-kinematischen Objekten wird in Unity spekulativ vorgenommen [[Tecc](#)]. Eine möglicher Lösungsansatz ist die Nutzung nicht-kinematischer Figuren statt kinematischer. Dadurch werden die Figuren von der Physik-Engine mit berücksichtigt und es findet eine akkurate Kollisionserkennung durch Unity statt. Auch die Kollisionsauflösung würde somit von Unity übernommen werden. Jedoch

hat dies den Nachteil, dass die Stangen nicht mehr reibungslos bewegt werden können. Da durch [TopiCo](#) die genaue Trajektorie basierend auf den Motorspezifikationen berechnet wird, kann durch Implementierung von nicht-kinematischen Figuren diesen Trajektorien nicht reibungslos gefolgt werden, was zu unrealistischeren Bewegungen der Stange führen kann. Weiterhin ist die durch Unity übernommene Kollisionsauflösung abhängig von vielen physikalischen Parametern, die akkurat am realen Kickertisch zu bestimmen sind, wie beispielsweise einzelne Reibungskoeffizienten und Trägheitsmomente der einzelnen Komponenten. Gleichzeitig geht damit ein großer Implementierungsaufwand einher, da die aktuelle Implementierung auf Nutzung kinematischen Stangen basiert.

Eine weitere Möglichkeit ist, den Parameter Fixed-Timesteps in Unity zu erhöhen. Dadurch wird die Wahrscheinlichkeit verringert, dass Kollisionen nicht erkannt werden. Weiterhin wird auch die Wahrscheinlichkeit verringert, dass Kollisionen erst bei Penetration des Balles erkannt werden. Die Anpassung der Fixed-Timesteps kann als Parameter ohne weiteren Implementierungsaufwand vorgenommen werden. Auch die kinematischen Eigenschaften der Stangenbewegung bleiben somit erhalten und es kann durch Verfolgung der von [TopiCo](#) generierten Trajektorie die realistische Umsetzung der Stangenbewegung erfolgen.

Aufgrund dessen fällt die Wahl zur Verbesserung der Kollisionserkennung auf die Anpassung bzw. Erhöhung der Fixed-Timesteps. Die Standardeinstellung des Fixed-Timestep Parameter beträgt 0,02 s [[Tece](#)]. Dies entspricht der Ausführung von physikalischen Berechnungen alle 20 ms. Um dies öfter zu ermöglichen ist ein kleinerer Wert zu bevorzugen. Dieser Wert wird unter Berücksichtigung der zur Verfügung stehenden Rechenressourcen ausgewählt, da häufigere Phsyikberechnungen eine höhere Rechenlast zur Folge haben. Wie zuvor erläutert stehen zwei Laptops mit den Spezifikationen [A.1](#) im Rahmen dieser Arbeit als Rechenressourcen zur Verfügung.

Für die Wahl des Parameters wurde der Ball vor der Stürmerstange platziert und diese mit maximaler Rotationsgeschwindigkeit rotiert. Es war zu sehen, dass bei hohen Werten weniger Kollisionen korrekt erkannt wurden. Gleichzeitig konnte bei niedrigen Werten eine erhöhte Rechenlast beobachtet werden. Hiebei wurde ein Wert von 0,01 s für den Fixed-Timestep Parameter bestimmt. Dieser Wert erzielte auf den vorhandenen Rechenressourcen eine gute Balance zwischen Kollisionserkennung und Rechenlast.

Als nächstes ist die Kollisionsauflösung zu verbessern. Da weiterhin kinematische Figuren genutzt werden, ist eine eigene Kollisionsauflösung zu implementieren, da diese nicht von Unity übernommen werden kann. Um eine realistische Kollisionsauflösung zu approximieren werden die von [TopiCo](#) berechneten Trajektorien genutzt. Diese beinhalten nicht nur die Zielwegpunkte, die die Stange verfolgen soll, sondern auch aktuelle Geschwindigkeiten der jeweiligen Achsen zu jedem Zeitpunkt. Somit können diese akkuraten Geschwindigkeiten der Figuren für die Berechnung der Kollisionsauflösung genutzt werden. Indem die laterale und rotations Stangengeschwindigkeit genutzt werden, kann mittels dieser beider Geschwindigkeiten ein Bewe-

gungsvektor berechnet werden, der die Richtung und Geschwindigkeit einer Figur beschreibt. Da die Geschwindigkeit der Lateralbewegung in m/s angegeben und die Rotationsbewegung in rad/s durch [TopiCo](#) angegeben wird, ist hierbei auf eine akkurate Umrechnung in die gleiche Einheit m/s zu achten.

Die akkurate Geschwindigkeiten des Balles werden von Unity bereitgestellt. Mithilfe beider Geschwindigkeitsvektoren kann eine Endgeschwindigkeit des Balles approximiert werden. Indem die Fixed-Timesteps erhöht werden sind, werden auch öfter und akkurate Kollisionen zwischen Figuren und dem Ball erkannt. Diese erkannten Kollisionen geben den exakten Kontakt-Punkt zwischen Ball und Stange an. Dieser Kontakt-Punkt beinhaltet auch die Höhe des jeweiligen Kontaktes zwischen Ball und Figur. Indem die Länge einer Figur und die Höhe des Ballkontakte berücksichtigt wird, kann ein genauer Rotationsradius der Stange berechnet werden. Mithilfe dieses Rotationsradius lässt sich die Rotationsgeschwindigkeit in eine Lateralgeschwindigkeit mit der Einheit m/s umrechnen, indem $v_{lateral} = rw$, wobei r den Radius und w die Rotationsgeschwindigkeit angibt [TS22]. Dies stellt jedoch lediglich eine Approximation des tatsächlichen Geschwindigkeitsvektors einer Figur dar.

Indem die einzelnen Geschwindigkeitsvektoren der Stange und des Balls miteinander subtrahiert werden, ergibt sich ein Endgeschwindigkeitsvektor für den Ball. Wird dieser mit dem normalisierten Kontakt-Punkt zwischen Ball und Figur multipliziert ergibt sich ein Richtungsvektor, der zum einen die Richtung des Balls angibt, die dieser nach der Kollision verfolgt, zum anderen auch die Ballgeschwindigkeit nach der Kollision durch dessen Magnitude ausdrückt. Da die Stangen von den Motoren in der jeweiligen Position gehalten werden und der Ball nur 20 Gramm wiegt, werden die Effekte einer Kollision auf die Stange vernachlässigt. Dies verringert den Implementierungsaufwand und die Bewegung der Stange wird vollständig von [TopiCo](#) übernommen, was eine realistische Stangenbewegung ermöglicht.

Die Kollisionsauflösung in der Simulation wurde mit dem Kollisionsverhalten des Balls am realen Kickertisch verglichen, indem einzelne Schüsse durch die Stürmerstange evaluiert wurden.

4.3.3 *Domain Randomization*

Eine weitere oft genutzte Methode um den Simulation-To-Reality-Gap zu verringern stellt Domain-Randomization dar [Tob+17]. Domain-Randomization baut auf der Prämisse auf, dass eine Simulation in vielerlei Hinsicht nicht exakt der Realität entspricht. Die Funktionsweise von Domain-Randomization wird im folgenden am Tischkickerbeispiel erklärt. Der Simulation-To-Reality-Gap in Bezug auf den Ball bezieht sich auf die physikalischen Eigenschaften des Balles in der Realität im Vergleich zu dessen Umsetzung in der Simulation. Beispielsweise könnten sich die Ballgröße oder das Kollisionsverhalten zwischen Simulation und Realität unterscheiden. Bei Nutzung von Domain-Randomization werden nun einzelne Parameter der Simulation va-

riert, beispielsweise die Ballgröße oder die anschließende Geschwindigkeit nach einer Ballkollision. Diese Variation findet auf Grundlage einer zuvor definierten Verteilung für den jeweiligen Parameter statt. So könnte eine Randomisierung der Ballgröße dazu führen, dass der Agent ein robusteres Verhalten lernt und auch mit Bällen mit leicht abweichender Größe in der Realität gut spielen kann.

Dies kann jedoch negative Auswirkungen auf die Trainingsdauer und Trainingsdateneffizienz haben [Ban+17, S. 8]. Das Randomisieren kann zwar zu robusteren Verhalten führen, aber ein Agent mit randomisierten Elementen in der Simulation lernt in der Regel langsamer, als ein Agent mit einer statischen Simulation. Grund hierfür ist, dass durch die Nutzung dieser Technik die Umgebung an Stationarität verliert. Diese nicht-stationarität kann zu Dynamiken in der Umgebung führen, die schwer für einen RL-Agenten zu analysieren sind [BM19, S. 1-2]. Somit ist eine Abwägung zu treffen, welche und wie viele Parameter randomisiert werden sollten, um zum einen die Trainingseffizienz nicht zu stark zu verringern, zum anderen jedoch einen Agenten mit stabilerem Verhalten hervorzubringen. Hierfür könnte die graduelle Randomisierung von Dimensionen der Umgebung als Trade-off gesehen werden, wie in [Ban+17] beschrieben. Dadurch wird dem Agenten zu Beginn Zeit gegeben, die Dynamiken der Umgebungen zu erlernen und später graduell einzelne Dimensionen randomisiert.

4.4 ZUSAMMENFASSUNG VORGENOMMENER ÄNDERUNGEN

In diesem Abschnitt werden die vorgenommenen Änderungen zur Reduktion des Simulation-To-Reality-Gaps zusammengefasst. Hierbei findet auch eine Evaluierung der Implementierung statt.

Zum einen ist die veränderte Kommunikationsstruktur zwischen Simulation und Agenten zu nennen. Hierbei wurde die Nutzung eines `gRPC`-Servers auf Unity-Seite implementiert, wodurch eine synchrone Kommunikation zwischen Simulation und Agenten ohne die Möglichkeit zur Warteschlangenbildung etabliert wurde. Hierbei wurde auch das verwendete Datenmodell angepasst, wodurch stets Informationen des selben Zeitpunktes an den Agenten übertragen werden. Durch die Ablösung zweier ZeroMQ-Server ist hier mit einer Leistungssteigerung der Simulation zu rechnen. Diese wird im folgenden Abschnitt quantitativ evaluiert.

Des weiteren sind die angepassten physikalischen Eigenschaften der Simulation zu nennen. Hierbei ist eine realistischere Stangenbewegung durch Nutzung von `TopiCo` implementiert worden. Dabei wurden die Grenzwerte der einzelnen Motoren des Bosch Rexroth Kickertisches ausgelesen und hierfür ein .NET-Assembly für die Integration der Bibliothek in der Unity Simulation kompiliert. Dadurch wird sichergestellt, dass die Stangen in der Simulation einer realistischen Trajektorie folgen. Hierbei wurde darauf geachtet, dass die Berechnungsmethode der Trajektoriengenerierung sowohl bei der Simulation als auch beim Bosch Rexroth Kickertisch die selben sind.

Zuletzt ist eine Anpassung des Kollisionsverhaltens vorgenommen worden,

um eine akkurate Kollision zwischen Ball und Figuren in der Simulation zu ermöglichen. Hierbei wurden die Geschwindigkeitsdaten von [TopiCo](#) genutzt, um den Bewegungsvektor des Balls nach einer Kollision berechnen zu können. Dies wurde qualitativ Evaluierter, indem Schüsse der Stürmerstange in der Simulation und auf dem realen Kickertisch miteinander verglichen wurden, um den Simulation-To-Reality-Gap in dieser Hinsicht zu reduzieren.

4.4.1 Evaluation Simulationsleistung

In diesem Abschnitt findet eine Evaluierung der Simulationsleistung vor und nach den implementierten Änderungen. Hierbei wurde der Ausgangsstand mit der Version verglichen, die alle in diesem Kapitel genannten Änderungen beinhaltet. Dabei wurden minimale Anpassungen an der Agentenimplementierung vorgenommen, um die Kommunikationsmethode [gRPC](#) zu nutzen. Für die Evaluierung der Leistungsfähigkeit wurden zehn Versuche mit unterschiedlichen Seeds für die Zufallsgeneratoren der Simulation genutzt. Alle Versuche fanden auf dem gleichen Laptop *ThinkpadP1Gen2* (siehe Tabelle A.1) statt. Hierbei wurde darauf geachtet, dass keine weiteren Programme auf dem Rechner liefen und die für die Ausführung beider Programme die gleichen Bedingungen auf dem System vorliegen. Die Messungen wurden unter Nutzung eines Netzteils vorgenommen damit ein unterschiedlicher Batteriestand keine Auswirkung auf die Ergebnisse haben kann. Für beide Implementierungen wurde in der Simulation die maximale Framerate gewählt. Weiterhin wurde das Python Profiling-Werkzeug *yappi* zur Analyse der Laufzeit des Programmes genutzt. Da dieses Programm die einzelnen Aufrufe im Programmcode misst, ist somit von einem Overhead in den gemessenen Werten auszugehen. Da *yappi* jedoch für alle Evaluierungen genutzt wurde, sollte dies keine Implementierung gegenüber einer anderen im Vergleich benachteiligen.

Zur Evaluierung wurden 1.000 Agentenschritte in der Umgebung vorgenommen. Hierbei wurden lediglich die Sensordaten der Simulation genutzt und keine Kamerainformationen.

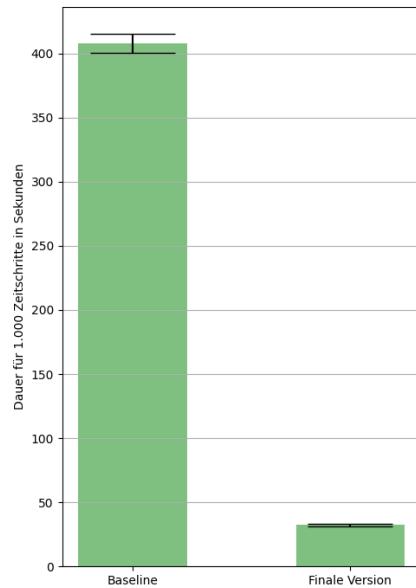


Abbildung 4.8: Dauer in Sekunden für 1.000 Schritte in der Umgebung. *Baseline* stellt übernommenen Quellcode-Stand dar. *Finale Version* beinhaltet die Änderungen dieses Kapitels. Dargestellt mir jeweiligem bootstrapped 95%-Konfidenzintervall.

Abbildung 4.8 zeigt den Durchschnitt von zehn verschiedenen Versuchen pro Implementierung an. Hierbei ist ein signifikanter Leistungsunterschied zu beobachten, wobei die Ausgangsversion fast zehn mal länger benötigt als die neue Implementierung. Betrachtet man die Ergebnisse von *yappi* so ist der Grund hierfür die Implementierung einer *DummyCamera*, welche konstant Daten von ZeroMQ abfragt obwohl die Kameraeinstellung deaktiviert ist.

Um den Vergleich fair zu gestalten, wird somit dieser Abschnitt aus dem Ausgangsquellcode entfernt und somit keine *DummyCamera* mehr genutzt. Für die Evaluierung dieses modifizierten Ausgangsquellcodes und der neuen Simulationsimplementierung werden erneut zehn verschiedene Versuche nach dem gleichen Versuchsaufbau durchgeführt. Hierbei werden nun jedoch 10.000 Agentenschritte pro Versuch vorgenommen. Das Ergebnis dieses Versuches ist in Abbildung 4.9 dargestellt.

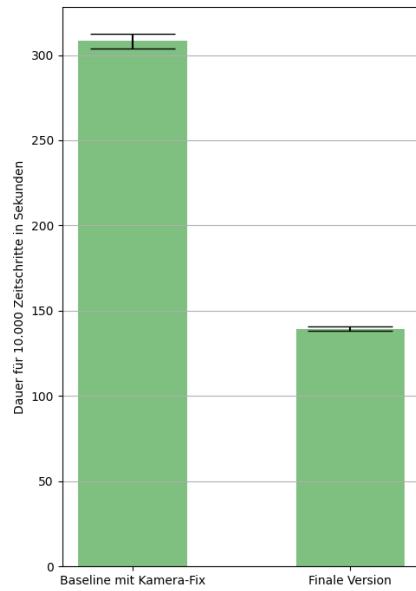


Abbildung 4.9: Dauer in Sekunden für 10.000 Schritte in der Umgebung. *Baseline mit Kamera-Fix* stellt behobenen Quellcode-Stand dar. *Finale Version* beinhaltet die Änderungen dieses Kapitels. Dargestellt mir jeweili- gem bootstrapped 95%-Konfidenzintervall.

Wie zu sehen ist trotz der Behebung der Nutzung der *DummyCamera* die neue Implementierung deutlich schneller. Hierbei ist zu berücksichtigen, dass diese neue Implementierung sowohl *TopiCo* für die Generierung der Trajektorien und einen doppelt so hohen Fixed-Timestep Parameter im Vergleich zur Ausgangssimulation nutzt, was zu doppelt so vielen physikalischen Berechnungen in der neuen Simulation führt. Dennoch konnte statt durchschnittlich über 300 Sekunden für 10.000 Zeitschritte in der Umgebung eine Reduktion auf durchschnittlich unter 150 Sekunden erfolgen, was eine signifikante Leistungssteigerung der Simulation belegt.

DEFINITION DER AGENTENUMGEBUNG

Ziel der Arbeit ist es, das Training von Tischkickeragenten möglichst effizient zu gestalten. Diese Arbeit bezieht sich auf den Stürmer und den Torwart als Tischkickeragenten. Damit eine systematische Auswahl der Komponenten für das Training getroffen werden kann, wird eine Problemdefinition vorgenommen. Wie in den Grundlagen bereits erläutert kann ein **RL**-Problem durch ein **MDP** dargestellt werden. Durch das **MDP** wird somit die Umgebung und die Dynamiken dieser Umgebung definiert. Für die Problemdefinition wird das **MDP** aus Sicht des Stürmer-Agenten formuliert. Grundsätzlich gilt folgende Problemdefinition auch aus Sicht des Torwart-Agenten. Diese Ausgangsdefinition des **MDP** aus Sicht eines Agenten wird in Kapitel 9 um die Nutzung mehrerer Agenten erweitert. Da durch Nutzung einer Simulation die Zustandsübergangsfunktion \mathcal{P} durch diese Vorgegeben und unbekannt ist, werden in diesem Kapitel der Observationsraum \mathcal{O} , Aktionsraum \mathcal{A} und die Rewardfunktion \mathcal{R} bestimmt. Hierbei wird darauf geachtet, dass die einzelnen Komponenten sowohl für den Torwart- als auch für den Stürmeragenten genutzt werden können. Anpassungen hierfür in Bezug auf die jeweiligen Agenten für das anschließende Multi-Agenten Training finden ebenfalls in Kapitel 9 statt.

5.1 OBSERVATIONSRAUM

Wie in den Grundlagen erläutert stellen Observationen Zustandsrepräsentationen dar. Hierfür ist die Darstellung einer Observation zu definieren. Weiterhin ist das zeitliche Intervall bzw. die Frequenz der Observationen zu bestimmen. Zunächst wird die Frequenz der Observationen bestimmt. Um die Beschreibung der Observationsfrequenz zwischen bildlichen- und Vektor-Observationen gleich zu halten, wird in dieser Arbeit einheitlich die Einheit Hertz Hz genutzt, um die Frequenz zu quantifizieren.

5.1.1 Frequenz an Observationen

Bei der Problemdomäne Tischkicker handelt es sich um ein Echtzeitspiel, bei dem zwei bzw. vier Personen gegeneinander spielen. Im Gegensatz zu Spielen wie beispielsweise Schach oder Poker, ist dieses Spiel nicht zugbasiert. Beide Teams können gleichzeitig Aktionen ausführen und kontinuierlich Observationen wahrnehmen. Somit ist darauf zu achten, die Observationsfrequenz nicht zu gering zu wählen, damit die Umgebung möglichst präzise observiert werden kann um Rückschlüsse bezüglich der gewählten Aktionen ziehen zu können. Da das Lernen in der Simulation stattfindet ist die Observationsfrequenz nur durch technische Grenzen eingeschränkt,

wie beispielsweise die benötigten Rechenressourcen für deren Umsetzung. Jedoch ist die Auswahl einer maximalen Observationsfrequenz nicht immer wünschenswert. Hierbei ist der grundlegende Zweck der Simulation in Erinnerung zu rufen. Die Simulation in dieser Arbeit dient wie zuvor in Kapitel 4 beschrieben dazu, das ggf. lange Trainieren von Tischkickeragenten zu ermöglichen. Somit dient die Simulation und letztlich auch das MDP lediglich als Abstraktion des realen Tischkickerproblems. Um den Simulation-To-Reality-Gap gering zu halten ist somit in Bezug auf die Observationsfrequenz zu achten, dass diese Frequenz realistisch zu wählen ist.

Zusammenfassend gilt für die Observationsfrequenz, dass diese zum einen möglichst hoch zu wählen ist, um präzise Rückschlüsse bezüglich der gewählten Aktionen ziehen zu können, zum anderen jedoch realistisch bezüglich der realen Problemdomäne zu wählen ist, damit der Simulation-To-Reality-Gap gering bleibt. Hierfür werden zunächst etablierte RL-Benchmark Umgebungen und die dort genutzten Observationsfrequenzen betrachtet, um Referenzwerte für diese Arbeit zu bestimmen. Anschließend werden Einschränkungen für die Frequenz bestimmt, um diese in einem realistischen Rahmen zu halten.

Es gibt eine Vielzahl etablierter Benchmark-Umgebungen für RL, die in vielen Arbeiten genutzt werden um unterschiedliche RL-Ansätze zu untersuchen [Mni+13] [KH20]. Dazu zählen unter anderem Emulationen von Atari 2600 Spielen wie in der Arcade Learning Environment (ALE) [Bel+13] implementiert, aber auch dreidimensionale Physiksimulationen wie in OpenAI Gym [Bro+16a] und DeepMind Control Suite [Tas+18] bereitgestellt. Für diese ausgewählten Benchmark-Umgebungen werden nun die jeweils genutzten Observationsfrequenzen bestimmt.

ARCADE LEARNING ENVIRONMENT ALE stellt ein Framework für die Nutzung von RL-Algorithmen zum Lernen von Atari 2600 Spielen dar. Hierfür wird eine hochperformante Emulation genutzt, die bis zu 6000 Bilder die Sekunde emulieren kann [Bel+13, S. 255]. Die Standardfrequenz ist jedoch mit 60 Hz deutlich geringer. Diese Frequenz wurde auch in unterschiedlichen Arbeiten genutzt [Mni+13; Bra+15].

OPENAI GYM beinhaltet eine Vielzahl unterschiedlicher RL-Probleme. Dies beinhaltet einfache zweidimensionale Kartenspiele bis zu komplexen Physiksimulationen [Ope22]. In Abbildung 5.1 ist ein Bild von dem RL-Problem *HalfCheetah* dargestellt, das auf der Physiksimulation MuJoCo basiert [Ope22]. Betrachtet man die Physiksimulationen von OpenAI Gym, haben diese unterschiedliche Observationsfrequenzen, wie in den Konfigurationsdateien der Probleme ersichtlich [Bro+16b]. Eine Liste der genutzten Frequenzen kann im Anhang A.2 eingesehen werden. Die genutzten Frequenzen reichen bei MuJoCo-Physiksimulationen von 50 bis 500 Hz, wobei die durchschnittliche Frequenz 197,22 Hz beträgt und der Median 100 Hz.

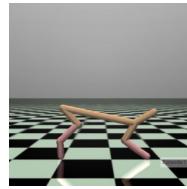


Abbildung 5.1: *HalfCheetah* Umgebung in OpenAI Gym.

Quelle: [Ope22]

DEEPMIND CONTROL SUITE stellt eine Sammlung von **RL**-Umgebungen dar, in denen Agenten kontinuierliche Kontrollaufgaben lösen können [Tas+18; Tun+20]. Die Aufgaben sind standardisiert aufgebaut, dreidimensional und nutzen ebenfalls die Physiksimulation *MuJoCo*. Die genutzten Frequenzen für die einzelnen Umgebungen sind ebenfalls im Anhang abgebildet A.3. Hierbei reichen die genutzten Frequenzen von 33,33 bis 1.000 Hz, mit einem Durchschnitt von 266,67 Hz und einem Median von 200 Hz.

Somit nutzen die betrachteten etablierten **RL**-Benchmarks Frequenz im Bereich von 33,33 Hz bis 1.000 Hz, wobei der Median von **ALE** 60 Hz, von OpenAI Gym 100 Hz und von DeepMind Control Suite 200 Hz beträgt. Nun sind mögliche Einschränkungen zu bestimmen, die benötigt werden um die Simulation dennoch möglichst realistisch darzustellen. Da Tischkicker von Menschen gespielt wird, kann die Frequenz des menschlichen Auges als Einschränkung genutzt werden. In [Pot+13] konnten Bilder vom Menschen verarbeitet werden, die lediglich für einen Zeitraum von 13 ms zu sehen waren. Dies entspricht einer Frequenz von ca. 75 Hz. Weitere Arbeiten [DHL15] schreiben dem menschlichen Auge eine maximale Observationsfrequenz von 50 – 90 Hz zu.

Zur Bestimmung einer Frequenz ist im Rahmen dieser Arbeit somit ein Wert zu wählen, der im Rahmen der menschlichen Observationsfrequenz liegt, damit der Tischkicker Agent dieselben Voraussetzungen zur Kontrolle wie ein Mensch hat. Auf Grundlage der genutzten Observationsfrequenzen etablierter **RL**-Benchmarks, sowie der Observationsfrequenz des menschlichen Auges, wird für diese Arbeit eine Observationsfrequenz von 60 HZ gewählt. Somit ist die Frequenz zum einen realistisch und nahe an der Frequenz des menschlichen Auges, zum anderen ist diese Frequenz eine oft genutzte in etablierten **RL**-Benchmarks.

5.1.2 Form und Inhalt einer Observation

Nun ist zu bestimmen, welche Form und welchen Inhalt eine Observation beinhalten soll. Zunächst wird auf die Form der Observation eingegangen, gefolgt von den Informationen, die diese beinhalten soll.

5.1.2.1 Form

Observationen können verschiedene Formen annehmen. So nutzen [Mni+13] Bilder von Atari-Spielen, um einen Agenten zu trainieren, diese Spiele zu spielen. Sie nutzten hierfür die Bilder eines Atari-Emulators und bearbeiteten diese zunächst, bevor sie als Observationen für den Agenten genutzt wurden. Die Bilder wurden in ihrer Auflösung reduziert und monochromiert, um die Dimensionalität der Daten gering zu halten. In Abbildung 5.2 sind die vorgenommenen Transformationen am Beispiel des Atari-Spiels *Breakout* zu sehen.

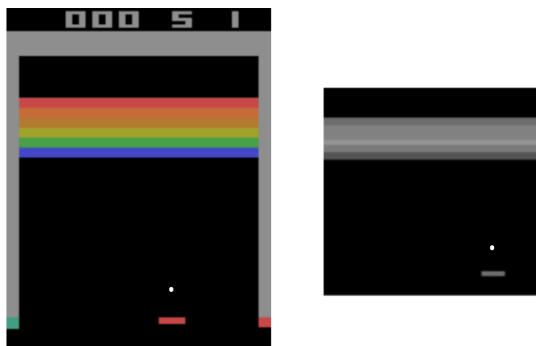


Abbildung 5.2: Bild des Atari-Spiels Breakout.

Links: Unbearbeitet.

Rechts: Monochromiert und in der Auflösung reduziert.

Quelle: In Anlehnung an [AY] erstellt.

Am Beispiel von *Breakout* sieht man jedoch, dass einzelne Bilder nicht alle Information einer Umgebung beinhalten. Beispielsweise ist keine Information zur Ballgeschwindigkeit im Bild erkennbar, oder eine Richtung, in der der Ball sich bewegt. Um auch diese Informationen für den Agenten sichtbar zu machen nutzten sie mehrere Bilder als eine Observation. Abbildung 5.3 zeigt, wie dies vorgenommen wurde.

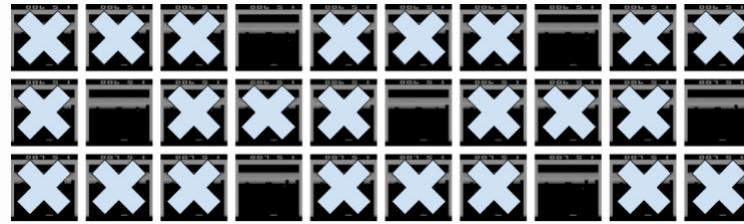


Abbildung 5.3: Darstellung des genutzten Frame-Skips beim Lernen von Atari Spielen mittels Bildern. Nur jedes vierte Bild wurde genutzt.

Quelle: [Fra]

Es wurden vier Bilder zusammengelegt als eine Observation, und diese dem Agenten mitgeteilt. Dabei wurden nicht vier aufeinander folgende Bilder genutzt, sondern nur ein Bild alle vier Bilder. Dies wird als Frame-Skipping bezeichnet [Bra+15]. Dadurch wird eine erneute Reduzierung des Observationsraums vorgenommen. Die Idee dahinter ist, dass ein Agent auch ohne das Wahrnehmen aller Bilder in der Lage ist eine gute Policy zu erlernen, und dadurch gleichzeitig die Trainingsgeschwindigkeit erhöht wird. Ihre Ergebnisse zeigten, dass ihr Agent eine gute Policy mit 10 Mio. Bildern erlernen konnte.

Um Informationen aus den Bildern extrahieren zu können, nutzte der Agent von [Mni+13] zum Verarbeiten der Bilder ein CNN. CNNs extrahieren mittels Faltungs- und Pooling-Schichten relevante Informationen aus Bildern[DDZ20, S. 26-27]. Hierfür müssen diese jedoch trainiert werden. Der von [Mni+13] genutzte Agenten verband dieses CNN mit einem weiteren FCNN. Die Architektur des genutzten Netzes ist in Abbildung 5.4 zu sehen.

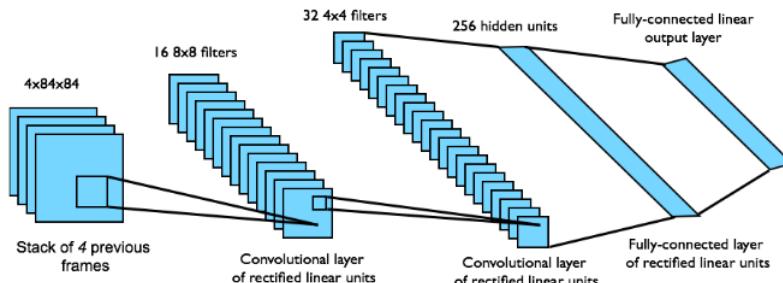


Abbildung 5.4: Genutzte Architektur für das Lernen von Atari 2600 Spielen aus Bildern. Nutzung von zwei *Convolutional-Layer* als CNN, gefolgt von einem FCNN.

Quelle: [Sil15]

Durch Nutzung der Rewards der Umgebung wurde nun das gesamte NN trainiert, um die gewünschten Ergebnisse in der jeweiligen Umgebung zu erreichen. Dies entspricht am Beispiel von Atari das Erreichen einer hohen Punktzahl im Spiel. Es wurde zum einen der hintere Teil des Netzes, das

[FCNN](#) trainiert, dass für die Auswahl der optimalen Aktion zuständig ist, als auch das vordere [CNN](#), das für das Extrahieren von relevanten Daten aus den Bildern zuständig ist. Diese Kombination ermöglichte auch in weiteren Arbeiten Erfolge in mehreren Domänen, auch in der Robotik [[ZQW20](#); [Ope+18](#)].

Dabei lässt sich feststellen, dass im genannten Beispiel das [NN](#) mehrere Aufgaben erlernen musste. Zum einen musste das [CNN](#) erlernen, welche Bereiche in den Bildern relevante Informationen beinhalten. Zum anderen musste das [FCNN](#) aus den relevanten Informationen lernen, die optimale Aktion herzuleiten. Es ist davon auszugehen, dass wenn das [CNN](#) bereits die relevanten Informationen aus den Bildern extrahieren könnte und nicht trainiert werden müsste, durch die von Beginn an hohe Qualität an Daten schneller eine gute Policy für die jeweilige Umgebung erlernt werden könnte. In der Tat gibt es mehrere Arbeiten in der [RL](#)-Literatur und mehrere [RL](#)-Benchmark-Umgebungen, die nicht Bilder der jeweiligen Umgebung zum Trainieren des Agenten nutzen, sondern sogenannte Feature-Vektoren [[Ope+19a](#); [Bro+16a](#); [Tas+18](#)].

Diese Feature-Vektoren sind Vektoren, die bereits die relevanten Informationen zum Finden der optimalen Policy beinhalten. In vielen Benchmarkumgebungen wie beispielsweise OpenAI Gym [[Bro+16a](#)] und DeepMind Control Suite [[Tas+18](#)], sind diese Feature-Vektoren als Observationen bereits vordefiniert. Mit Nutzung von Feature-Vektoren entfällt somit das Trainieren eines [CNN](#), um die relevanten Informationen aus Bildern zu extrahieren, womit sich das Training effizienter gestalten lässt. Außerdem ist hervorzuheben, dass Bilder mehr Speicher benötigen als Feature-Vektoren und deren Nutzung rechenaufwendig ist [[Mni+13](#), S. 5]. Somit ist davon auszugehen, dass die Nutzung von Feature-Vektoren im Vergleich zur Nutzung von Bildern einen positiven Effekt auf die Trainingseffizienz hat.

Einen Vergleich zwischen Bildern und Feature-Vektoren als Observationsformen wurde in [[WU19](#)] vorgenommen. Dort werden unter anderem drei verschiedene Observationsräume miteinander verglichen:

- Bilder
- Feature-Vektoren
- Feature-Vektoren mit Objekterkennung

Dieser Vergleich findet in den beiden Umgebungen *Reacher-v2* und *FrankaReacher-v0* von OpenAI Gym statt und wird durch Nutzung von drei Lernalgorithmen ausgeführt. Die Ergebnisse dieser Arbeit zeigen ein Trend bezüglich des Einflusses des Observationsraums auf die Leistung der Agenten über alle drei Lernalgorithmen und beide Umgebungen hinweg. Es konnte beobachtet werden, dass durch Nutzung von Feature-Vektoren der geringste *Regret* erzielen konnten, wobei *Regret* den erwarteten Verlust an Reward unter Ausführung einer Policy bezeichnet [[KLM96](#)]. Weiterhin konnte durch Nutzung von Feature-Vektoren eine bessere Policy erlernt werden im Vergleich zu den anderen getesteten Alternativen.

Da sich die Nutzung von Feature-Vektoren im Rahmen von effizientem Training eher eignen als die Nutzung von Bildern, und in [WU19] durch Nutzung von Feature-Vektoren bessere Ergebnisse erzielt werden konnten fällt die Wahl für die Repräsentationsart für den Observationsraum auf die Nutzung von Feature-Vektoren. Hierbei ist jedoch darauf zu achten, dass der Agent die zu extrahierenden Informationen nicht mehr selbst bestimmen kann und der Inhalt der Feature-Vektoren somit mit bedacht zu wählen ist.

5.1.2.2 *Inhalt der Observation*

Nachdem die Nutzung von Feature-Vektoren als Form der Observationen bestimmt worden ist, ist nun der Inhalt eines Feature-Vektors zu bestimmen. Hier spielt vor allem eine große Rolle, ob der gesamte Zustand der Umgebung in einer Observation dargestellt werden kann, da dies wie in den Grundlagen beschrieben die Voraussetzung zur Erfüllung der Markov-Eigenschaft ist. Hierfür ist das Tischkickerspiel als Problemdomäne näher zu betrachten.

Kann der gesamte Zustand der Umgebung in einer Observation dargestellt werden, so erfüllt diese Observation die Markov-Eigenschaft. Um die Wichtigkeit der Markov-Eigenschaft zu verdeutlichen, wird dies in einem Vergleich zwischen den Spielen Schach und Poker erläutert. Bei Schach handelt es sich um ein sogenanntes Perfect-Information-Game [Alp, S. 19]. Spiele mit dieser Eigenschaft sind vollständig observierbar [PM17, Kap. 11.2.2]. Am Beispiel von Schach bedeutet dies, dass alle Informationen zu jedem Zeitpunkt von beiden Spielern beobachtbar sind. Es ist bei Schach nicht notwendig, die Historie an Zuständen zu kennen, um die optimale Aktion für einen Zustand wählen zu können. Somit stellt Schach ein Perfect-Information-Game dar, dessen Observationen die Markov-Eigenschaft erfüllen. Dadurch wird sichergestellt, dass die zukünftigen Zustände des Spiels nur von dem aktuellen Zustand abhängig sind und nicht von der Historie des gesamten Spiels. Anders sieht dies aus beim Spiel Poker. Bei Poker hat jeder Spieler ein Blatt von Karten. Ein Spieler sieht jedoch nur die eigenen Karten. Die Karten der anderen Spieler bleiben einem verborgen. Es handelt sich bei Poker um ein Imperfect-Information-Game und ist somit nur in Teilen observierbar [PM17, Kap. 11.2.2]. Durch die partielle Observierbarkeit der Umgebung erfüllen Observationen eines Spielers nicht die Markov-Eigenschaft und die zukünftigen Ausgänge sind auch von vergangenen Observationen abhängig. Eine geeignete Formulierung für diese Spiele stellen Partially Observable Markov Decision Process (POMDP)s dar. Hier entspricht eine Observation nämlich nicht dem Zustand des Spiels, da Informationen verborgen bleiben. Ein Agent weiß somit nicht, in welchem Zustand er sich aktuell in der Umgebung befindet. [SB18, S. 467]

Observationen können in POMDP genutzt werden, um einen internen *Belief-State* des Agenten zu formen, der eine Abschätzung des tatsächlichen Zustandes des Spiels darstellen soll [KLC98, S. 106].

Betrachtet man nun den Tischkicker als Problemdomäne, so wird klar, dass es sich um ein Perfect-Information-Game handelt, da alle relevanten Informationen Spieler zu jedem Zeitpunkt ersichtlich sind. Somit kann ein Observationsraum definiert werden, der die Markov-Eigenschaft erfüllt. Dies ist wichtig, da wie in den Grundlagen beschrieben für effektive und informative Entscheidungen eine informative Zustandsdarstellung nötig ist.

Eine naheliegende Idee wäre nun, eine Observation mit so vielen Informationen wie möglich zu gestalten, damit der Agent allerlei Informationen über die Umgebung hat, um somit die beste Entscheidung treffen zu können. Hierbei ist jedoch zu beachten, dass dies nicht zwangsläufig zum effizientesten Training des Agenten führen muss. Dies wird klar, wenn die Aufgabe eines Agenten in [RL](#) nochmal explizit formuliert wird. Der Agent versucht eine Policy zu erlernen, die einen möglichst hohen erwarteten Return zur Folge hat. Diese Policy stellt eine Funktion dar, die einer Observation der Umgebung und einer möglichen Aktion eine Wahrscheinlichkeit zur Ausführung zuweist. Das Verhalten ist somit eine Abbildung von Wahrscheinlichkeiten auf Observations-Aktions-Paare. Ein Agent mit einem optimalen Verhalten besitzt eine Policy, die bei jeder Observation die Aktion auswählt, die den maximal möglichen Return zur Folge hat.

Je komplexer nun die Observation ist, desto schwieriger ist es für den Agenten, relevante von nicht-relevanten Informationen zu unterscheiden. Es kann somit von Nachteil sein, möglichst viele Observationen hinzuzufügen, da hierdurch der Agent länger beim Training brauchen könnte, um die Nützlichkeit der einzelnen Observationskanäle zu erlernen. Dies wird auch als *Curse of Dimensionality* bezeichnet [[BMo3](#)]. Gleichzeitig muss starkes Minimieren der Dimensionen des Observationsraum auch nicht zwangsläufig zu effizienterem Training führen und kann unter Umständen das Lernverhalten negativ beeinflussen. Grund hierfür ist, dass dem Agenten möglicherweise nützliche Informationen für das Erlernen einer guten Policy verwehrt bleiben. Es gilt somit einen Observationsraum zu definieren, der eine Balance zwischen dessen Größe und Relevanz an Informationen beinhaltet. [[SB15](#), S. 63-65] Somit müssen ggf. manche Observationskanäle bzw. Dimensionen aus dem Observationsraum entfernt werden. Es stellt sich nun die Frage, wie eine Auswahl der zu nutzenden Observationskanäle zu treffen ist.

Zunächst werden mögliche Observationskanäle ausgearbeitet, die für die Nutzung der Tischkickeragenten in Frage kommen. Anschließend folgt in Kapitel [7](#) eine Auswahl der zu nutzenden Observationskanäle. Grund hierfür ist, dass für die Auswahl der zu nutzenden Observationskanäle zunächst weitere Komponenten des [MDP](#) zu bestimmen sind, unter anderem der Aktionsraum und die Rewardfunktion.

5.1.3 Observationskanäle beim Tischkicker

Einen Ansatz für das Trainieren eines Tischkicker-Stürmers mittels [RL](#) zeigen [[DB+21](#)]. Sie nutzen einen sechsdimensionalen Observationsraum, um

einem Stürmer das Schießen beizubringen. Dieser besteht aus folgenden Observationskanälen:

- Lateralposition der Stange
- Winkelposition der Stange
- Lateralgeschwindigkeit der Stange
- Rotationsgeschwindigkeit der Stange
- X-Position des Balls
- Z-Position des Balls

In der zum Training genutzten Simulation wurde von den Autoren die Sensordaten diese genutzt. Wurde der Agent auf dem Bosch Kickertisch ausgeführt, so konnten präzise Stangeninformationen aus den Motoren ausgelesen werden. Für Informationen zum Ball wird eine Kamera genutzt, die mit 60 Hz von oben herab das Spielfeld filmte. Die genauen Ballkoordinaten auf dem Spielfeld werden dann mittels analytischer Computergrafik aus dem Videomaterial extrahiert und als X- und Z-Koordinate ebenfalls dem Agenten mitgeteilt. In ihrer Arbeit nutzten sie keine Ballgeschwindigkeit als Observationskanal aufgrund fehlender verlässlicher Daten hierzu. Dies ist eine interessante Entscheidung, da hierdurch die Markov-Eigenschaft des Observationsraumes verloren geht. Dies wird in Abbildung 5.5 dargestellt.

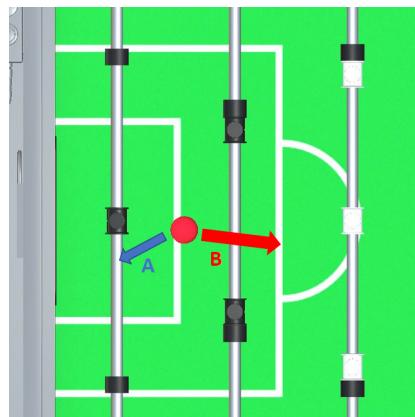


Abbildung 5.5: Verlust der Markov-Eigenschaft dargestellt am Ball, der sich entweder entlang Pfeil A oder Pfeil B bewegen kann. Nur durch eine Observation ist nicht bestimmbar welche Richtung und Geschwindigkeit der Ball hat.

Sowohl Fall A als auch Fall B werden durch diesen Observationsraum gleich dargestellt. Jedoch könnte es sein, dass eine Aktion für Fall A optimal, aber für Fall B nicht optimal wäre, da sich der Ball in B mit beispielsweise einer viel höheren Geschwindigkeit bewegt. Neben der Geschwindigkeit des Balles kann auch keine Aussage über die Bewegungsrichtung des Balles getroffen werden. So kann es sein, dass der Ball auf den Stürmer oder auf das

Tor zuläuft. Beide Fälle könnten jeweils andere Aktionen erfordern. Dennoch konnte in ihrer Arbeit ein Stürmer trainiert werden, der aus unterschiedlichen Positionen Tore schießen konnte.

Da in dieser Arbeit das Trainieren in der Simulation im Vordergrund steht, können genaue Daten zu den jeweiligen Positionen und Geschwindigkeiten von Ball und Stange genutzt werden. Dies lässt auch bei späterem Transfer auf den realen Kickertisch Schlüsse darüber ziehen, wie relevant diese zusätzlichen Informationen für das Lernen des Agenten sind.

Wie bereits zuvor erläutert, hat die Erfüllung der Markov-Eigenschaft Auswirkung auf das Trainingsverhalten eines Agenten. Somit ist auch bei Gestaltung des Observationsraumes für den Tischkicker darauf zu achten, dass diese erfüllt wird. Hierfür müssen die relevanten Informationen zur Stange und zum Ball dem Agenten mitgeteilt werden.

Zunächst werden die Informationen zur Stange ausgearbeitet. Relevante Informationen stellen die Lateral- und Winkelposition der Stange dar. Um die Bewegung entlang der verschiedenen Dimensionen zu beschreiben sind die einzelnen Achsbeschreibungen in Abbildung 5.6 dargestellt.

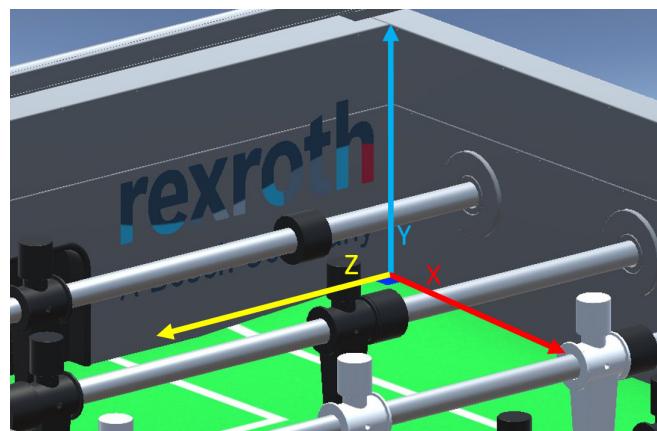


Abbildung 5.6: Koordinatensystem in der Simulation, bestehend aus X-, Y- und Z-Achsen.

Da die Kickerstange entlang der X- und Y-Achse fixiert ist, kann sie sich nur entlang der Z-Achse lateral bewegen. Daraus folgt, dass der Wert der Z-Achse ausreicht um die genaue Lateralposition der Stange zu definieren. Bezuglich der Winkelposition sieht es ähnlich aus. Die Stange lässt sich lediglich um die Z-Achse rotieren, weshalb zur Beschreibung der Winkelposition ebenfalls ein einzelner skalarer Wert genügt. Somit ist die genaue Position des Agenten mittels zwei skalarer Werte definierbar. Wie zuvor erläutert wird jedoch auch die Geschwindigkeit für die Erfüllung der Markov-Eigenschaft benötigt.

Hierfür bieten sich zwei Möglichkeiten an. Eine Möglichkeit ist die Nutzung von gestaffelten Observationen wie in [Mni+13]. Wie bereits zuvor zu diesem Beispiel beschrieben, werden mehrere Bilder des Atari-Emulators genutzt um eine Observation für den Agenten darzustellen. Durch die Po-

sitionsdaten aus vier unterschiedlichen Bildern ist die Geschwindigkeit des Agenten implizit Teil einer Observation.

Eine weitere Möglichkeit ist die Nutzung expliziter Geschwindigkeitsdaten für den Agenten. Dies wird auch in unterschiedlichen RL-Benchmarks wie OpenAI Gym genutzt [Bro+16b]. Diese Option würde sich für die genutzte Atari Umgebung von [Mni+13] weniger eignen, da der Atari-Emulator keine Angaben bezüglich der Geschwindigkeit des Agenten macht. In Kapitel 4 wurde erläutert, dass eine zeitoptimale Trajektoriengenerierung für die Simulation der Stangenbewegung genutzt wird, um diese möglichst realistisch darzustellen. Da diese Trajektorien ebenfalls präzise Geschwindigkeitsangaben bezüglich beider Achsen der Tischkickerstange beinhaltet (Lateralbewegung und Rotation), können diese Daten als Observationskanäle genutzt werden.

Um die Geschwindigkeitsdaten zu den Observationskanälen hinzuzufügen, wird die explizite Darstellung dieser mittels der TopiCo-Trajektoriengenerierung genutzt, da hierdurch präzise Angaben der Geschwindigkeit möglich sind und keine Approximierung mittels Positionsdaten vorgenommen werden muss.

Es lässt sich zusammenfassen, dass der Observationsraum folgende Observationskanäle bezüglich der Tischkickerstange enthält:

- Lateralposition der Stange
- Winkelposition der Stange
- Laterale Bewegungsgeschwindigkeit
- Rotationsgeschwindigkeit

Nun müssen noch die Observationskanäle bezüglich des Balls hinzugefügt werden. Hierfür lässt sich wie für die Stange zunächst feststellen, dass die Ballposition eine wichtige Information darstellt. Im Gegensatz zur Stange kann sich der Ball jedoch in allen drei Dimensionen bewegen. Neben der X- und Z-Achse kann sich der Ball auch entlang der Y-Achse bewegen, da der Kickertisch angehobene Ecken besitzt. Dennoch stellen lediglich die X- und Z-Koordinaten relevante Informationen für den Agenten dar, da an den angehobenen Ecken keine Spieler platziert sind. Hinzu kommt, dass die Fläche die durch Spieler erreicht werden kann gerade ist und keine Unebenheiten aufweist.

Neben der Ballposition ist wie bei der Stange auch die Ballgeschwindigkeit wichtig, um die Markov-Eigenschaft zu erfüllen. Hier gibt es die gleichen Möglichkeiten wie bei der Stange, namentlich die implizite Darstellung der Geschwindigkeit durch Nutzung zeitversetzter Positionsdaten, oder die explizite Darstellung mittels dediziertem Wert für die Ballgeschwindigkeit. Da sich der Ball sowohl entlang der X- als auch entlang der Z-Achse bewegen kann, müssen zwei Geschwindigkeitswerte mitgeteilt werden, damit eine Antizipierung der Ballposition erlernt werden kann. Wie für die Stange werden auch für den Ball die expliziten Geschwindigkeitswerte genutzt, da diese durch Nutzung der Simulation präziser sind als die implizite Darstellung.

Mit diesen Informationen zum Ball ist nun auch die Markov-Eigenschaft hinreichend approximiert, wie bei den OpenAI Gym Umgebungen [Bro+16b]. Zusammenfassend lassen sich als grundlegender Observationsraum für den Tischkickeragenten folgende Observationskanäle festhalten:

- Lateralposition der Stange
- Winkelposition der Stange
- Laterale Bewegungsgeschwindigkeit
- Rotationsgeschwindigkeit
- X-Position des Balls
- Z-Position des Balls
- X-Geschwindigkeit des Balls
- Z-Geschwindigkeit des Balls

Nun gibt es weitere Informationen, die nützlich für das Trainieren der Tischkickeragenten sein könnten. Eine Information die dem Agenten aktuell verwehrt bleibt, ist die Anzahl und Position der Spieler einer Stange. Dies könnte für die Torwartstange von kleinerer Bedeutung sein, da diese lediglich einen Spieler in der Mitte der Stange besitzt. Jedoch besitzt die Stürmerstange eines Kickertisches drei Stürmer. Interessant wäre hier zu sehen, ob das Hinzufügen der Positionsdaten der Spieler Einfluss auf das Training eines Agenten hat. Da die Spieler fest an der Stange montiert sind und die Abstände zwischen ihnen somit auch bei Bewegung gleich bleiben, reicht hier ein skalarer Wert der Position der Spieler auf der Z-Achse. Durch Hinzufügen dieser Information müsste der Agent nicht implizit die Anzahl der Spieler und deren jeweilige Position lernen, sondern könnte diese über die Observationskanäle exakt erfassen. Als weitere mögliche Observationskanäle sind somit die jeweiligen Spielerpositionen einer Tischkickerstange festzuhalten.

In [KH20] werden Unterschiede zwischen den Observationsräumen in den RL-Benchmarkumgebungen OpenAI Gym und DeepMind Control Suite herausgearbeitet und deren Einfluss auf das Lernverhalten bei dem RL-Problem *Locomotion* analysiert. Ein solcher Unterschied ist die Nutzung von binären Kontaktinformationen. Diese binären Werte geben an, ob es einen Kontakt zwischen den jeweiligen Gelenken des Agenten gibt. In ihren Auswertungen sahen sie kleinere Leistungsunterschiede bzw. Leistungsgewinne bei Nutzung dieses Observationskanals.

Eine andere Studie [RTP20] untersuchte ebenfalls die Nützlichkeit von binären Kontaktinformationen bei Locomotion-Problemen und kam zum Schluss, dass binäre Kontaktinformationen in manchen Fällen hilfreich sein können. Binäre Kontaktinformationen können auch beim Tischkicker genutzt werden, um einen Kontakt zwischen Spieler und Ball zu identifizieren. Da bei Berührung von Spieler und Ball in der Regel eine Veränderung der Ballgeschwindigkeit stattfindet, kann hierdurch ggf. weiterer Aufschluss über die

Dynamiken des Spiels erlernt werden. Dies stellt somit einen weiteren Observationskanal, der für die Nutzung beim Training in Frage kommen könnte. Binäre Informationen wurden auch in anderen Arbeiten genutzt. So nutzten [Ope+19a] in ihrem Observationsraum unterschiedliche binär kodierte Informationen um beispielsweise die Team-Zugehörigkeit einzelner Einheiten im Spiel *Dota 2* zu kodieren. Am Beispiel einer Tischkickerstange könnte sich die Nutzung einer binären Reichweiteninformation anbieten. Hiermit würde dargestellt werden, wann der Ball in Reichweite der jeweiligen Stange liegt. Diese Information könnte von Relevanz für den Agenten sein, da auf den Ball nur Einfluss genommen werden kann wenn sich dieser innerhalb der Stangenreichweite befindet. Somit könnte dieser Observationskanal auf die Möglichkeit zum Ballkontakt hindeuten.

Daraus folgen folgende potentielle Ergänzungen für den zuvor definierten Observationsraum:

- Spielerpositionen der Stange
- Binäre Kontaktinformation
- Binäre Reichweiteninformation

Dies stellt nur eine Auswahl von möglichen Ergänzungen des Observationsraums dar. Da die Bestimmung des Observationsraums nur ein Teil des Gesamtkonzeptes ist, wurden hierfür auf Grundlage von verwandten Arbeiten mögliche Observationskanäle bestimmt, die mit relativ geringem Implementierungsaufwand umgesetzt werden können. Hierbei wurde ebenfalls darauf geachtet, dass die bestimmten Observationskanäle auch am realen Kickertisch umgesetzt werden können, beispielsweise durch Nutzung analytischen Computergrafik wie in [DB+21]. In Kapitel 7 findet eine methodische Auswahl der zu nutzenden Observationskanäle für den Observationsraum statt.

5.1.4 Kodierung der Observationen

Im vorigen Abschnitt wurden die Informationen definiert, die dem Agenten mitgeteilt werden können. Nun ist die Kodierung dieser Informationen zu bestimmen.

Da sich Ball und Stange frei in ihren jeweiligen Dimensionen bewegen können, bietet sich die Repräsentation durch kontinuierliche Werte für diese Informationen an. Diese Art der Kodierung wird auch in verschiedenen Observationsräumen basierend auf Feature-Vektoren in RL-Benchmarkumgebungen wie OpenAI Gym und DeepMind Control Suite genutzt. Dies würde folgende Observationskanäle betreffen:

- Lateralposition der Stange
- Winkelposition der Stange

- Laterale Bewegungsgeschwindigkeit
- Rotationsgeschwindigkeit
- X-Position des Balls
- Z-Position des Balls
- X-Geschwindigkeit des Balls
- Z-Geschwindigkeit des Balls

Weiterhin sind die potenziellen Ergänzungen des Observationsraums zu berücksichtigen. Die Spielerpositionen der Stange können ebenfalls durch kontinuierliche Werte dargestellt werden, da diese an der Stange befestigt sind und sich somit in kontinuierlichen Inkrementen bewegen können. Für die binären Kontakt- und Reichweiteninformationen bietet sich eine binäre Kodierung an.

Zuletzt ist zu betrachten, in welchem Wertebereich die einzelnen Observationskanäle dargestellt werden sollen. Hierbei ist das in den Grundlagen erläuterte Vanishing Gradient Problem 2.1.3 zu berücksichtigen. Werden große Wertebereiche gewählt, so kann dies zu einer hohen Aktivierung einzelner Perzeptren führen und bei der Backpropagation durch Anwendung der Kettenregel zu sehr kleinen Anpassungen der Parameter eines NN führen [Alz+21, S. 49]. Gleichzeitig sollte darauf geachtet werden, dass die einzelnen Wertebereiche der Observationskanäle in einer ähnlichen Reichweite liegen, damit einzelne Observationskanäle nicht zu einer höheren Aktivierung nur auf Grund ihres Wertebereiches führen. Indem ein null-zentrierter kleiner Wertebereich genutzt wird, wird auch die Ausdruckskraft des NN erhöht, da auch kleine Werte eine hohe Aussagekraft behalten [FFL22b]. Weiterhin wird hierdurch die Wahrscheinlichkeit für das Vanishing Gradient Problem verringert, da die Eingabewerte klein gehalten werden. Aus diesem Grund bietet sich für die Kodierung der kontinuierlichen Observationskanäle die Nutzung des Intervalls $[-1, 1]$ an.

Hierfür ist jedoch Wissen über die Grenzen der jeweiligen Observationskanäle erforderlich. In Bezug auf die Stange sind Maximal- und Minimalausprägung der Lateral- und Winkelposition der Stange bekannt. Gleichzeitig sind die Grenzen der lateralen Bewegungs- und Rotationsgeschwindigkeit bekannt. Dies trifft auch auf die X- und Z-Position des Balls zu, sowie die Spielerpositionen einer Stange. Lediglich die Grenzen der X- und Z-Geschwindigkeiten des Balls sind unbekannt. Hier ist zu beachten, dass der Ball unter Umständen hohe Geschwindigkeiten annehmen kann. In [And+20, S. 6] wird die Nutzung von Grenzwerten nahegelegt, wenn das Risiko für sehr hohe Eingabewerte besteht. Somit wird ein Grenzwert von 10 m/s bestimmt, da dieser in unterschiedlichen Arbeiten als Referenzgeschwindigkeit für schnelle Tischkickerspiele genannt wird [JBM10; WNo3, S. 132, S. 3]. Die Ballgeschwindigkeit wird somit durch das Intervall $[-10, 10]$ dargestellt. Werden höhere Ballgeschwindigkeiten erreicht, wird der jeweilige

Maximalwert des Intervalls genutzt. Die binären Kontakt- und Reichweiteninformationen können binär mit $[0, 1]$ kodiert werden, wie auch in [Ope+19a] vorgenommen.

5.2 AKTIONSRaUM

Der Aktionsraum des Tischkickeragenten muss die Einflussmöglichkeiten des Agenten in der Umgebung darstellen können. Dieser wird durch dessen Inhalt und Kodierung definiert. Wie in den Grundlagen beschrieben finden Aktionen und Observationen zu diskreten Zeitschritten statt. In Kapitel 5.1.1 wurde bereits eine Observationsfrequenz von 60 Hz festgelegt. Hierbei ist zu prüfen, ob Aktionen ebenfalls mit dieser Frequenz ausgeführt werden sollen oder mit einer geringeren Frequenz. Zunächst wird die Frequenz der Aktionen bestimmt, gefolgt vom Inhalt des Aktionsraums und dessen Kodierung.

5.2.1 Frequenz von Aktionen

Mittels Aktionen nimmt der Agent Einfluss auf die Umgebung. Es lässt sich feststellen, dass je höher die Aktionsfrequenz ist, desto feingranularer kann der Agent auf Zustandsänderungen in der Umgebung reagieren. Somit erlaubt eine hohe Aktionsfrequenz das Lernen einer potenziell besseren Policy im Vergleich zu einer niedrigen Aktionsfrequenz, da zu mehr Zeitpunkten auf kleinste Änderungen in der Umgebung reagiert werden kann. Eine hohe Aktionsfrequenz kann jedoch negativen Einfluss auf die Effizienz des Lernens haben [Bra+15]. Ein Agent muss zunächst Informationen über die Umgebung sammeln. Auf Grundlage der erhaltenen Observationen und getätigten Aktionen erfolgt eine Evaluierung unter Berücksichtigung des erhaltenen Rewards. Am Beispiel der Stürmerstange ist das Schießen eine zu erlernende Abfolge von Aktionen. Je kleinschrittiger die Aktionen des Agenten sind, desto länger kann das Erlernen einer solchen Aktionssequenz dauern, da die Länge dieser Aktionssequenz zunimmt. Es ist somit eine Balance zu finden, um zum einen die Aktionsfrequenz so hoch zu wählen, sodass der Agent genügend Kontrolle in der Umgebung hat, zum anderen jedoch auch so zu wählen, dass die Effizienz nicht stark hierunter leidet und die zu erlernenden Aktionssequenzen zu lang werden.

Eine Möglichkeit die Anzahl an auszuführenden Aktionen zu beschränken ist Frame-Skipping, wie in [Mni+13] genutzt. Hierbei werden nicht zu jeder Observation Aktionen ausgeführt, sondern alle n Observationen eine Aktion. Während der nicht wahrgenommenen Observationen wird die zuvor gewählte Aktion wiederholt, auch genannt Action-Repeat [Bra+15; Kal+21]. Dadurch findet bei rechenintensiven Aktionsbestimmungen ein Zeitersparnis statt, wodurch Simulationen mehr Schritte auf einmal simulieren können. Gleichzeitig konnte jedoch in unterschiedlichen ALE Umgebungen gerade durch Frame-Skipping gute Resultate erzielt werden. [Bra+15]

Als Alternative zum Frame-Skipping kann eine Aktion zu jedem Zeitpunkt ausgewählt werden, was einem Frame-Skip von 0 entsprechen würde. Dies

würde bei der in Kapitel 5.1.1 bestimmten Observationsfrequenz von 60 Hz zu 60 Aktionen pro Sekunde führen. Betrachtet man die für diese Arbeit relevante Umgebung so betrifft das den Bereich zwischen Stürmer- und Torgewichtsstange. Wie in Kapitel 4.1.1 erläutert beträgt der Abstand zwischen diesen beiden Stangen 20,4 cm. Geht man nun von Ballgeschwindigkeiten von bis zu 10 m/s aus, so ist diese Strecke in 20 ms vom Ball durchlaufen. Die bestimmte Observationsfrequenz von 60 Hz entspricht einer Observation alle 16,67 ms. Selbst unter Berücksichtigung geringerer Ballgeschwindigkeiten von 6 m/s würde der Ball diese Strecke in ca. 34 ms durchlaufen. Aus diesem Grund wird im Rahmen dieser Arbeit auf Frame-Skipping verzichtet, da durch die potenziell hohen Ballgeschwindigkeiten die Nutzung von Frame-Skipping zu einem potenziell hohen Kontrollverlust führen könnte. Aktionen werden somit synchron mit jeder Observation durchgeführt.

5.2.2 Inhalt einer Aktion

Zunächst ist festzuhalten, dass wie bereits beschrieben eine Tischkickerstange so montiert ist, dass diese sich lediglich entlang der Z-Achse bewegen und um diese rotieren kann. Daraus ergibt sich ein zweidimensionaler Aktionsraum. Der Agent muss somit die laterale Bewegung und Rotation der Stange bestimmen bzw. direkten Einfluss darauf nehmen können. Hierbei ist festzuhalten, dass die Position der einzelnen Figuren für das Spiel von Relevanz ist, da nur mittels der Spieler ein Ball geblockt oder geschossen werden kann. Deshalb bietet es sich an, die Position als Inhalt des Aktionsraums festzulegen, wie auch beim Tischkickeragenten von [DB+21]. Dies würde bedeuten, dass eine Aktion des Agenten eine Ziel-Lateral- und -Winkelposition beinhaltet würde.

Da der Ball lediglich durch die Spielerfiguren kontrolliert werden kann könnte es sich weiterhin anbieten, den Rotationsbereich einer Stange zu limitieren. Hierdurch kann vermieden werden, dass eine Stange extreme Winkelpositionen annimmt, wodurch kein Kontakt mehr zum Ball hergestellt werden kann. In den Grundlagen wurde bereits erläutert, dass die annehmbaren Winkelpositionen sowohl am Kickertisch, als auch in der Simulation auf $[-90^\circ, 90^\circ]$ begrenzt sind. Aufgrund der Distanz zwischen Figur und Spielfeld von 6 mm, einem Balldurchmesser von 3,5 cm und einer Figurlänge von 7,2 cm lässt sich der Winkel berechnen, indem eine Figur zuletzt den Ball berühren kann. Dies entspricht in diesem Fall ab einer Winkelposition von 55° . Da in Kapitel 4.3.2.2 jedoch erläutert wurde, dass Kollisionen spekulativ von der Simulation erkannt werden, wird ein Grenzwert von 70° genutzt, da ab dieser Höhe auch bei hohen Geschwindigkeiten von 10 m/s in der Simulation keine fehlerhaften Kollisionen erkannt wurden.

Indem somit die Winkelposition auf den Wertebereich $[-70^\circ, 70^\circ]$ begrenzt wird, werden für das zu erlernende Verhalten nicht benötigte Winkelpositionen vermieden. Dies wird auch in [KSH20] als Empfehlung für die Definition eines Aktionsraums genannt, um ein RL-Problem möglichst leicht lernbar

zu machen. Gleichzeitig ist es weiterhin möglich, die Stange so weit rotieren zu lassen, dass der Ball nicht von den Figuren berührt werden muss.

5.2.3 Kodierung des Aktionsraums

Nachdem die Aktionsfrequenz und dessen Inhalt bestimmt worden sind, ist die Kodierung des Aktionsraum zu bestimmen. Die Aktionen können mittels diskreter oder kontinuierlicher Werte mitgeteilt werden. Während Aktionen in Videospielen häufig diskret sind [Bel+13], sind die meisten Aktionen in der Realität, gerade in Bezug auf Roboter von Natur aus kontinuierlich [KCC13, S. 126]. Dies trifft auch auf den Tischkicker zu. Eine der Realität entsprechende Kodierung wäre somit die Nutzung von kontinuierlichen Werten, um die Zielposition der Stange mitzuteilen. Gleichzeitig ist zu berücksichtigen, welchen Einfluss die Kodierung auf die Effizienz des Lernens hat. In [KSH20] wurde hierzu eine Analyse durchgeführt und verschiedene Transformationen bezüglich des Aktionsraums evaluiert. Sie kamen zum Schluss, dass diskrete und multi-diskrete Aktionsräume effizienteres Lernen erlauben als kontinuierliche. In ihrer Arbeit wird jedoch im Zuge dessen genannt, dass die Bestimmung der Diskretisierung einen neuen Hyperparameter für das Training darstellt, der ebenfalls zu bestimmen ist. Die Autoren gehen jedoch nicht darauf ein, wie sie die Diskretisierung bestimmt haben oder wie dieser Hyperparameter zu bestimmen ist.

In einer weiteren Arbeit [TA19] werden kontinuierliche Aktionsräume diskretisiert und einzelne Diskretisierungsparameter getestet. Die Autoren kamen zum Schluss, dass die Diskretisierung in komplexen Umgebungen wie *Humanoid*, mit einem 367-dimensionalem Observations- und 17-dimensionalem Aktionsraum, das Training deutlich beschleunigt, dies jedoch nicht der Fall ist bei simpleren Umgebungen wie beispielsweise *Hopper*, mit einem 11-dimensionalem Observations- und 3-dimensionalem Aktionsraum [Ope22]. Die Schwierigkeit der einzelnen Umgebungen wurde von den Autoren auf Grundlage der Dimensionalität von Observations- und Aktionsraum bewertet. Hinzu kamen schlechtere Leistungen von diskretisierten Aktionsräumen in besagten simpleren Umgebungen im Vergleich zu kontinuierlichen.

Somit ist eine Diskretisierung des Aktionsraums zu prüfen, da dies einen positiven Effekt auf die Effizienz des Trainings haben könnte. Die Prüfung hierzu findet in Kapitel 8 statt, da ähnlich wie bei der Bestimmung der zu nutzenden Observationskanäle zunächst weitere Komponenten des MDP festzulegen sind. Bei Bestimmung zur Nutzung von diskreten oder kontinuierlichen Aktionen findet auch eine Kodierung des Aktionsraums statt.

5.3 REWARDFUNKTION

In diesem Abschnitt wird die zu nutzende Rewardfunktion am Beispiel der Stürmerstange definiert. Hierfür werden die Lernziele des Stürmeragenten erläutert und eine Rewardfunktion entworfen, mit der die Leistung des Agenten pro Episode quantifiziert werden kann.

Ziel des Tischkickerstürmers ist es Tore zu schießen. Hierbei ist gleichzeitig zu vermeiden, dass der Ball hinter den Stürmer gelangt. Somit wird zunächst festgehalten, dass bei einem Tor ein positiver Reward gegeben wird und ein negativer Reward, wenn der Ball hinter dem Stürmer gelangt. Ein Beispiel für eine Rewardfunktion für einen Tischkickerstürmer haben [DB+21] in ihrer Arbeit dargestellt. Deren genutzte Rewardfunktion ist folgendermaßen aufgebaut $r = r_{goal} + r_{time} + r_{backshot}$, wobei $r_{goal} = 5$ der Reward für das Schießen eines Tors ist, $r_{backshot} = -1$ der Reward wenn der Ball hinter den Stürmer gelangt und $r_{time} = -\frac{1}{MaxSteps}$ ein kontinuierlicher negativer Reward ist der jeden Zeitschritt dem Agenten mitgegeben wird. Dies ist eine oft genutzte Technik um dem Agenten das Ziel zu geben, eine Episode so kurz wie möglich zu halten [Ope22].

Zunächst werden Rewards für die Terminalzustände des MDP definiert. Diese sind zum einen das Schießen eines Tores, zum anderen wenn der Ball hinter die Stürmerstange gelangt. Eine intuitive Belegung hierfür wäre die Nutzung eines Rewards von 1 für ein Torereignis und ein Reward von -1 , wenn der Ball hinter den Stürmer gelangt. Dies hätte zur Folge, dass das Schießen eines Tores so erstrebenswert ist, wie es vermeidenswert ist, dass der Ball hinter die Stürmer gelangt. Da jedoch rechts und links vom Tor Wände sind, von denen der Ball abprallen kann, ist die Wahrscheinlichkeit bei Nutzung einer zufälligen Policy höher, dass es zu Abprallern kommt als zu Toren, da die Wände mit einer Gesamtbreite von 48 cm mehr als doppelt so Breit sind wie das Tor mit 20 cm Breite. Dies könnte schnell dazu führen, dass der Agent früh mit dem Schießen aufhört, da Tor- und Ball-hinter-Stürmer-Ereignis vom Reward equivalent sind und es deutlich häufiger zu einem Abpraller kommen würde als zu einem Tor, gerade zu Beginn des Trainings. Um dies zu vermeiden, wird ein Torereignis deutlich stärker bewertet als ein Ball-hinter-Stürmer-Ereignis. Hierdurch wird der Anreiz zum Schießen auf das Tor aufrecht erhalten. Gleichzeitig führt eine Maximierung des Rewards dazu, dass präzises Schießen auf das Tor nach genügend Training erlernt werden sollte. Da es wie in den Grundlagen beschrieben für NN von Vorteil ist, wenn die Werte in einem Wertebereich von $[-1, 1]$ liegen, wird somit für das Schießen eines Tores ein Reward von 1 bestimmt, während für das Ereignis Ball-hinter-Stürmer ein Reward von -0.1 definiert wird. Somit ist das Rewardverhältnis von positivem zu negativem Episodenausgang 10 : 1. Für die Wahl dieses Verhältnis ist lediglich wichtig, dass das Schießen von Toren dementsprechend höher belohnt wird, damit selbst bei häufigen Abprallern des Balles an der Torwand weiterhin ein Anreiz zum Schießen besteht. Hierbei könnten auch andere Verhältnisse dieses Ziel erreichen, es ist jedoch davon auszugehen, dass dieses Verhältnis ausreichend hoch ist um den gewünschten Effekt zu erzielen. Hierdurch wird auch aufwändiges Ausprobieren verschiedener Werte im Rahmen dieser Arbeit vermieden.

5.3.1 Potenzialbasierter Reward

Der Reward ist ein skalares Signal der Umgebung an den Agenten zur Beurteilung des Zustandes. Diese können zu unterschiedlichen Zuständen und/oder Aktionen zugeteilt werden. Abhängig von dem jeweiligen Design der Umgebung können Rewards beispielsweise nur selten mitgeteilt werden, zum Beispiel nur beim Erreichen von terminalen Zuständen. Eine solche Rewardfunktion wird als spärlich bezeichnet. Im Gegensatz dazu wird eine Rewardfunktion als dicht bezeichnet, wenn es mehrere Rewards gibt bzw. diese häufiger mitgeteilt werden. [Par20]

Spärliche Rewardfunktionen haben den Vorteil, dass diese deutlich leichter zu entwerfen sind als dichte Rewardfunktionen. Hierfür ist kein Expertenwissen über die Domäne nötig. Ein Beispiel hierfür wäre ein Reward von 1 bei erfolgreichem Episodenausgang und ein Reward von -1 bei nicht erfolgreichem Episodenausgang. Sie haben allerdings auch den Nachteil, dass sie schwerer zu erlernen sind als dichte Rewardfunktionen [Har19]. Grund hierfür ist, dass der Agent zunächst zufällige Aktionen ausführt und auf Grundlage der Rewards eine Anpassung der jeweils getätigten Aktionen vornimmt. Werden Aktionen durchgeführt die einen positiven Reward erzielen so wird das neuronale Netz so angepasst, dass die Wahrscheinlichkeit diese Aktionen erneut bei den jeweiligen Observationen auszuführen erhöht wird. Wenn die Umgebung nun selten Rewards vergibt, kann es lange dauern bis genügend Rewards gesammelt worden sind und die Anpassung am NN so signifikant ist, dass sich das Verhalten des Agenten maßgeblich zum Beseren ändert. Bei Nutzung von dichten Rewardfunktionen kann der Agent schneller Schlüsse darüber ziehen, welche Zustände in der Umgebung vorteilhafter sind und welche Aktionen zu diesen Zuständen geführt haben.

Um die Rewardfunktion dichter und somit einfacher lernbar zu gestalten, sollten weitere Informationen mitgeteilt werden. Dies ist jedoch kompliziert und kann unerwünschte Nebenwirkungen für das zu erlernende Verhalten haben. Ein Beispiel wird in [NHR99, S. 2] genannt, indem einem Roboter Fußballspielen beigebracht werden soll. Damit dieser Agent erfolgreich Schießen lernt könnte man vermuten, dass ein Reward bei der Ballberührung dazu führt, dass zwangsläufig Schüsse stattfinden und durch die daraus resultierenden Tore die Aufgabe vom Agenten erfolgreich erlernt wird. Die Autoren schreiben jedoch, dass der Agent sehr nah am Ball blieb und zitterte, da dies zu vielen Berührungen pro Sekunde führte und somit ein hoher Reward erzielt werden konnte. Diese Art von Zwischenfall nennt man Reward Hacking [Iba+18, S. 8]. Reward Hacking beschreibt den Fall, dass ein Agent unerwünschte Schlupflöcher der Rewardfunktion ausnutzt um den Reward zu maximieren. Durch Nutzen dieser unerwünschten Schlupflöcher lernt der Agent jedoch nicht das eigentlich zu erlernende Verhalten.

An diesem Beispiel konnte man sehen, wie das hinzufügen eines weiteren Rewards zu einer nicht wünschenswerten Policy führt. Es ist somit darauf zu achten, dass die Rewardfunktion insofern verdichtet wird, sodass sich die optimale Policy dadurch nicht ändert. Dies kann mittels Potential Based Reward Shaping (PBRS) erreicht werden [NHR99]. PBRS stellt eine Methode dar eine spärliche Rewardfunktion in eine verdichtete Rewardfunktion umzuwandeln, ohne dass dabei die optimale Policy verändert wird. Hierfür wird das MDP der Agentenumgebung in ein MDP' umgewandelt. Dieses MDP' unterscheidet sich vom MDP lediglich in der Rewardfunktion. Während in der Rewardfunktion \mathcal{R} des ursprünglichen MDP spärlich Rewards verteilt werden, ist die Rewardfunktion \mathcal{R}' des MDP' eine Rewardfunktion basierend auf \mathcal{R} und einer formenden Rewardfunktion $\mathcal{F}(S_t, A_t, S_{t+1})$:

$$\mathcal{R}'(S_t, A_t, S_{t+1}) = \mathcal{R}(S_t, A_t, S_{t+1}) + \mathcal{F}(S_t, A_t, S_{t+1}) \quad (5.1)$$

In ihrer Arbeit [NHR99] liefern die Autoren eine Form für \mathcal{F} , die garantiert, dass die optimale Policy im MDP' der optimalen Policy im ursprünglichen MDP entspricht. Um das zuvor genannte Problem der zyklischen Ausnutzung von Schlupflöchern zu umgehen, schlagen die Autoren die Nutzung von Differenzen der Zustandspotenziale vor, indem eine Potenzialfunktion $\psi(s)$ eine Funktion über die Zustände des MDP ist, die das Potenzial eines Zustandes quantifiziert. Hieraus ergibt sich für die formende Rewardfunktion \mathcal{F} :

$$\mathcal{F}(S_t, A_t, S_{t+1}) = \gamma\psi(S_{t+1}) - \psi(S_t) \quad (5.2)$$

wobei γ den Diskontierungsfaktor des MDP darstellt. Hierdurch werden zyklische Schlupflöcher eliminiert, da bei einer Transition in einen Zustand mit niedrigerem Potenzial als der Ausgangszustand, es zu einem negativen formenden Reward kommt [NHR99, S. 4]. Die Autoren beweisen die Invarianz dieser Methode in Bezug auf die optimale Policy des ursprünglichen MDP in ihrer Arbeit.

In einer Labyrinth-Umgebung wird dieser Ansatz von den Autoren empirisch untersucht mit dem Ergebnis, dass die Lerngeschwindigkeit um mehrere Größenordnungen zugenommen hat im Vergleich zur Nutzung einer spärlichen Rewardfunktion. Hier wurde als Potenzialfunktion die Manhattan-Distanz zwischen Agent und Ziel gewählt. Somit konnte ermöglicht werden, dass wenn ein Zustand erreicht wird, der Näher am Ziel des Agenten ist, dies die Ausschüttung eines positiven Rewards zur Folge hat. Weiterhin kam es jedoch zur Ausschüttung eines negativen Rewards, wenn ein Zustand erreicht wurde, der die Manhattan-Distanz zum Ziel vergrößert hat.

Am Beispiel vom Tischkicker und dem Stürmeragenten eignet sich eine Potenzialfunktion, die abhängig von der Ballposition ist. Je näher der Ball am Tor ist, desto positiver ist dieser Zustand aus Perspektive des Stürmers zu bewerten. Da die Ballposition aus X- und Z-Koordinaten besteht, ist auf beide Dimensionen Rücksicht zu nehmen. Hierfür wird $\psi_x^{\text{Stürmer}}$ als Potenzial-

funktion für die X-Position des Balles bestimmt, sowie $\psi_z^{\text{Stürmer}}$ als Potenzialfunktion für die Z-Position. Als Potenzialfunktion $\psi_x^{\text{Stürmer}}$ kann die Distanz zwischen der X-Koordinate des Balls x_{Ball} und der X-Koordinate des Tors x_{Tor} gewählt werden. Als Grenzwert ist ebenfalls die X-Koordinate bekannt, die vom Stürmer zuletzt berührt werden kann, $x_{\text{Hinter_Strmer}}$. Hierbei sollte $\psi_x^{\text{Stürmer}}$ einen hohen Wert liefern, wenn $x_{\text{Ball}} \approx x_{\text{Tor}}$, und einen niedrigen Wert liefern, wenn $x_{\text{Ball}} \approx x_{\text{Hinter_Strmer}}$. Dafür ist als nächstes der Wertebereich von $\psi_x^{\text{Stürmer}}$ zu definieren. Wie bereits bezüglich der Observationen und Aktionen beschrieben, ist auch für die Nutzung von Rewardfunktionen darauf zu achten, dass diese Werte im Wertebereich von $[-1, 1]$, um die Wahrscheinlichkeit für das Vanishing Gradient Problem zu verringern. Eine geeignete Begrenzung des Wertebereichs von $\psi_x^{\text{Stürmer}}$ erfolgt somit durch $[0, 1]$, da hierdurch der Wertebereich der formenden Rewardfunktion \mathcal{F} auf $[-1, 1]$ begrenzt wird (siehe Gleichung 5.2). Da wie in Kapitel 5.1.4 bestimmt eine normalisierte Ballposition im Wertebereich von $[-1, 1]$ genutzt wird, beträgt die X-Koordinate beim Tor -1 und hinter dem Stürmer 1 . Somit wird $\psi_x^{\text{Stürmer}}$ folgendermaßen definiert werden:

$$\psi_x^{\text{Stürmer}}(X_{S_t}) = -0,5X_{S_t} + 0,5 \quad (5.3)$$

wobei X_{S_t} der X-Koordinate des Balls zum Zustand S_t entspricht.

Bei der Definition von $\psi_z^{\text{Stürmer}}$ ist zu beachten, dass jede Z-Position welche durch das Tor abgedeckt wird ein maximal positiven Wert zur Folge haben sollte. Dies ist wichtig, damit der Agent nicht angeregt wird nur in die Mitte des Tores zu schießen. Dieser Bereich ist in Abbildung 5.7 dargestellt.

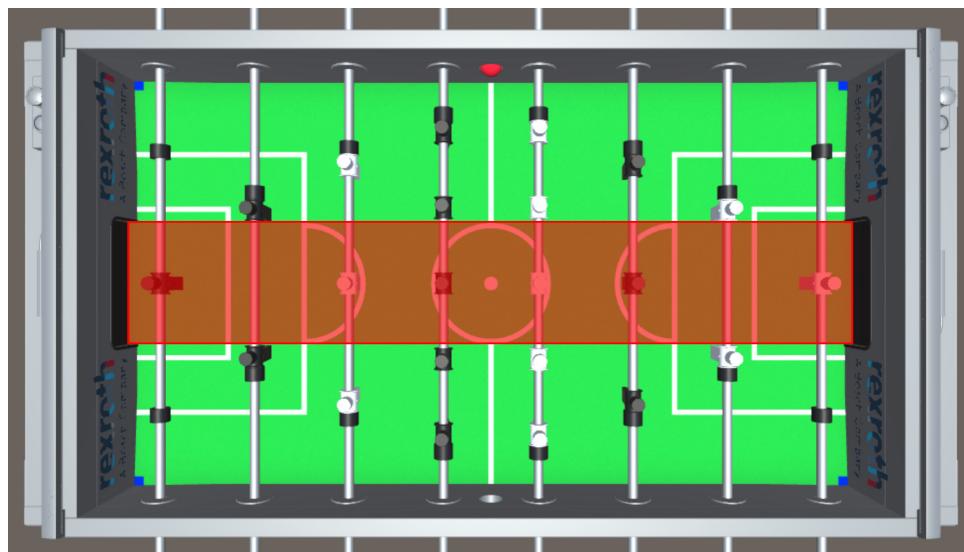


Abbildung 5.7: Breite des Tores als Referenz zur Bestimmung der Potenzialfunktion $\psi_z^{\text{Stürmer}}$

Um somit $\psi_z^{\text{Stürmer}}$ zu definieren, wird von den äußeren Kickerwänden zur Mitte des Spielfeldes ein zunehmendes Potenzial für den Ball bestimmt.

Wird auch hier eine Begrenzung des Wertebereichs auf $[0, 1]$ vorgenommen, die maximale und minimale Z-Position des Balls $z_{max} = 1$ und $z_{min} = -1$ und die Breite des Tores in diesem Wertebereich berücksichtigt, so lässt sich $\psi_z^{\text{Stürmer}}$ folgendermaßen definieren:

$$\psi_z^{\text{Stürmer}}(Z_{S_t}) = \min(-1.36|Z_{S_t}| + 1.36, 1) \quad (5.4)$$

wobei Z_{S_t} der Z-Koordinate des Balls zum Zustand S_t entspricht.

Durch Nutzung der X- und Z-Position des Balls kann somit eine Potenzialfunktion entworfen werden, die Abhängig von der Ballposition ist. Da die Potenzialfunktionen für den Wertebereich von $[0, 1]$ definiert sind, kann es beispielsweise zu einem Potenzial mit dem Wert 2 kommen, wenn sich der Ball kurz vor dem Tor befindet. Um $\psi_x^{\text{Stürmer}}$ und $\psi_z^{\text{Stürmer}}$ in einer Potenzialfunktion zu vereinen und dessen Wertebereich ebenfalls auf $[0, 1]$ zu begrenzen, wird somit folgende Potenzialfunktion als $\psi^{\text{Stürmer}}$ definiert:

$$\psi^{\text{Stürmer}}(S_t) = \frac{\psi_x^{\text{Stürmer}}(X_{S_t}) + \psi_z^{\text{Stürmer}}(Z_{S_t})}{2} \quad (5.5)$$

Als abschließende Rewardfunktion aus Perspektive der Stürmerstange wird somit definiert:

$$\mathcal{R}(S_t, A_t, S_{t+1}) = \begin{cases} 1 & \text{wenn } S_{t+1} = \text{Torereignis} \\ -0.1 & \text{wenn } S_{t+1} = \text{Ball-hinter-Stürmer-Ereignis} \\ \gamma \psi^{\text{Stürmer}}(S_{t+1}) - \psi^{\text{Stürmer}}(S_t) & \text{andernfalls} \end{cases} \quad (5.6)$$

wobei $\psi^{\text{Stürmer}}(S_t)$ Gleichung 5.5 entspricht, für alle diskreten Zeitschritte t innerhalb einer Episode beliebiger Länge T .

6

LERNALGORITHMUS

In diesem Abschnitt werden zunächst Kriterien bestimmt, die der Lernalgorithmus zur Nutzung in der Tischkickerdomäne erfüllen muss. Hierfür wird auch berücksichtigt, dass der Algorithmus für das spätere Training gegen einen weiteren Agenten geeignet sein muss. Anschließend wird ein passender Lernalgorithmus bestimmt der diese Kriterien erfüllt, gefolgt von der Auswahl einer passenden Implementierung und der Bestimmung der zu nutzenden Hyperparameter.

6.1 AUSWAHL LERNALGORITHMUS

Wie in den Grundlagen beschrieben gibt es unterschiedliche Kategorien, in denen Lernalgorithmen unterteilt werden können. Eine Taxonomie nach [DDZ20, S. 125] ist in Kapitel 2.3.1 zu sehen. Wichtig für diese Arbeit sind folgende Kategorien von Lernalgorithmen. Lernalgorithmen, die:

- modellbasiert oder modelfrei sind.
- eine deterministische oder stochastische Policy lernen.
- wertebasiert-basiert oder policy-basiert sind.
- On-Policy oder Off-Policy sind.

6.1.1 *Modellbasierte oder Modelfreie Methoden*

Wie in den Grundlagen bereits beschrieben, benötigen modellbasierte Lernalgorithmen ein Modell der Umgebung, das die Dynamiken dieser beschreibt. Jedoch ist dies bereits eine hohe Anforderung, da oft ein solches Modell nicht vorhanden ist [DDZ20, S. 127]. Im Gegensatz hierzu benötigen modelfreie Lernalgorithmen kein Modell Umgebung und Lernen durch Interaktion in der Umgebung ein Verhalten das zu einem möglichst hohen Return führt. Dies bietet sich für Fälle an, in denen kein Modell der Agentenumgebung vorhanden ist. Die im Rahmen dieser Arbeit genutzte Simulation stellt kein solches Modell der Umgebung dar. Aufgrunddessen eignet sich die Nutzung eines modelfreien Lernalgorithmus.

6.1.2 *Deterministische oder stochastische Policy*

Während eine deterministische Policy bei einem Zustand stets die gleiche Aktion auswählt, ist eine stochastische Policy eine Wahrscheinlichkeitsverteilung, die einem Paar von Zustand und Aktion eine Wahrscheinlichkeit

zuordnet. Dies hat zur Folge, dass bei identischen Zuständen nicht zwangsläufig immer die gleiche Aktion ausgeführt wird. Ob eine deterministische oder stochastische Policy zu wählen ist, hängt vom jeweiligen Problem und dessen Stationarität ab. Angenommen bei dem betrachteten RL-Problem handelt es sich um ein stationäres MDP. Wie bereits in Kapitel 2.2.1 erläutert beschreibt die Stationarität, ob sich die Zustandsübergangsfunktion \mathcal{P} eines MDP über einen Zeitraum verändert oder nicht. Bleibt \mathcal{P} konstant handelt es sich um eine stationäre Umgebung. Ist das Problem stationär, so gibt es für diese Umgebung eine deterministische optimale Policy [DD05]. Dennoch könnte für diese Umgebung auch eine stochastische Policy genutzt werden, da eine stochastische Policy in eine deterministische umgewandelt werden kann, indem zu jedem Zeitpunkt deterministisch die Aktion mit der höchsten Wahrscheinlichkeit gewählt wird.

Anders sieht es aus wenn man ein RL-Problem mit nicht-stationärem MDP betrachtet. Hier ändert sich die Zustandsübergangsfunktion \mathcal{P} mit der Zeit. Betrachtet man das Spielen von Stürmer und Torwart in der Umgebung, so kann eine deterministische Policy durch den Gegner ausgenutzt werden. Ein Beispiel hierfür bietet das Spiel Schere-Stein-Papier, indem eine deterministische Policy keine optimale Policy darstellt [Sil15, Kap. 7]. In diesen Multi-Agenten Umgebungen muss somit keine stationäre deterministische optimale Policy existieren [Lou22].

Da im Rahmen dieser Arbeit das Training mit zwei Agenten in derselben Umgebung absolviert wird, ist die Umgebung aus der jeweiligen Sicht eines Agenten nicht-stationär [Lou22]. Aus diesen Gründen wird entschieden für das Trainieren der Agenten eine stochastische Policy zu nutzen.

6.1.3 Wertebasierte oder policy-basierte Methoden

Wie in Kapitel 2.3.1.2 beschrieben unterscheiden sich wertebasierte und policy-basierte Methoden in ihren Lernzielen. Bei policy-basierten Ansätzen wird eine explizite Repräsentation einer Policy erlernt, während bei wertebasierten Methoden statt einer expliziten Policy eine Wertefunktion approximiert wird. Die Policy ist hier implizit und kann direkt aus der Wertefunktion abgeleitet werden, indem die Aktion mit dem höchsten Wert ausgewählt wird. Eine weitere Klasse stellen Actor-Critic Methoden dar, die sowohl eine Wertefunktion und eine explizite Policy erlernen.

Wie bereits zuvor festgehalten ist eine stochastische Policy zu nutzen. Da wertebasierte Algorithmen eine Wertefunktion lernen, die einzelnen Aktionen einen Wert zuweisen, eignen sich diese Algorithmen nicht in Fällen, in denen die optimale Policy stochastisch ist [Sil15, Kap. 7]. Zwar können aus Wertefunktions stochastische Policies generiert werden, dies jedoch nur in begrenztem Maße. Ein Beispiel hierfür wäre eine ϵ -greedy Policy, die mit einer Wahrscheinlichkeit $1 - \epsilon$ die Aktion auswählt mit dem höchsten Aktionswert und mit Wahrscheinlichkeit ϵ eine zufällige Aktion. Jedoch besteht hier das Problem, dass diese Methode nicht zu einer optimalen stochastischen Policy konvergieren kann, da die Policy statisch aus der Wertefunktion ab-

geleitet wird und sich nicht der Stochastizität der Umgebung anpasst. Somit sind policy-basierte Algorithmen zu wählen, da diese stochastische Policies lernen können [Sil15, Kap. 7]. Daraus folgt jedoch nicht, dass werte-basierte Algorithmen nicht von Nützlichkeit für das Erlernen einer stochas-tischen Policy sein können. Es können auch Actor-Critic Methoden genutzt werden, die eine stochastische Policy durch Nutzung eines policy-basierten Actors ermöglichen, wobei ein wertebasierter Critic bei der Evaluation der Änderungen der Policy unterstützen kann. Somit eignen sich hierfür Policy-basierte oder Actor-Critic Methoden.

6.1.4 On-Policy oder Off-Policy

Während Off-Policy Algorithmen auf Grundlage von beliebigen Policies lernen, nutzen On-Policy Algorithmen die aktuell in der Umgebung eingesetzte Policy. Hierdurch können Off-Policy Algorithmen Transitionen speichern und mehrmals zum Lernen verwenden, wodurch sie in der Regel probeneffizienter als On-Policy Algorithmen sind. Im Gegensatz dazu wird bei On-Policy Algorithmen ein *Rollout* an Transitionen in der Umgebung vom agierenden Agenten gesammelt und zum Lernen genutzt. Ein Rollout beschreibt die Ausführung einer Policy in der Umgebung und das Sammeln dieser Transitionen, um diese für die Optimierung der Policy zu nutzen. Nach Nutzung dieser Transitionen zur Aktualisierung der **NN** werden die gesammelten Transitionen jedoch verworfen und neue gesammelt. Dies macht On-Policy Algorithmen weniger probeneffizient im Vergleich zu Off-Policy Methoden, jedoch auch weniger rechenintensiv, wobei Off-Policy Methoden auch abhängig von der gewählten Anzahl an zu speichernden Transitionen auch speicherlastiger sind [GK20, Kap. 4.6]. Beispielhaft hierfür ist der Unterschied zwischen dem Off-Policy Algorithmus Actor-Critic with Experience Replay (**ACER**) und On-Policy Algorithmus Proximal Policy Optimization (**PPO**), indem **PPO** für 10.00 Schritte in der Umgebung *Atlantis-vo* ca. 60 Sekunden benötigt, während **ACER** hierfür ca. 260 Sekunden braucht [Lia+21]. Dennoch können Off-Policy Algorithmen durch ihre probeneffizientere Nutzung von Transitionen weniger Trainingsdaten und somit Schritte in der Umgebung benötigen, wodurch eine langsamere Berechnung kompensiert werden könnte.

Ein weiterer für diese Arbeit relevanter Unterschied ist die Speicherung der gesehenen Transitionen. Off-Policy Algorithmen nutzen hierfür einen Replay-Buffer [Sch+16], der oft Transitionen im Millionenbereich speichert [Mni+15; Haa+18; Lil+15]. Für die Optimierung des **NN** werden anschließend Transitionen des Replay-Buffers genutzt, indem diese entweder zufällig [Mni+13; Mni+15] ausgewählt werden oder eine Priorisierung einzelner Transitionen stattfindet [Sch+16]. Durch die Größe des Replay-Buffers wird auch zur Stabilität des Trainings beigetragen, da abwechslungsreiche Transitionen für die Optimierung genutzt werden können. Durch Nutzung von mehreren Agenten in einer Umgebung wird diese jedoch nicht-stationär. Nutzt der trainierende Agent einen Replay-Buffer mit 10^6 Transitionen, die

mit einem Gegner gesammelt worden sind, so besitzt der Replay-Buffer einen starken Bias gegenüber diesem Gegner. Wird nun ein neuer Gegner eingesetzt mit ggf. abweichendem Verhalten oder Strategie, kann es lange dauern, bis im Replay-Buffer genügend Transitionen mit dem neuen Gegner gespeichert worden sind. Dies macht das Konzept des Replay-Buffers, welches die probeneffizienz und Stabilität von Off-Policy Algorithmen stark positiv beeinflusst [Fed+20], inkompatibel mit Multi-Agenten-Umgebungen [Foe+17]. On-Policy-Methoden, die nur aktuelle Transitionen der Umgebung zur Optimierung nutzen besitzen dieses Problem nicht.

Aus diesem Grund wird im Rahmen dieser Arbeit ein On-Policy Lernalgorithmus gewählt, da dieser sich somit für Versuche in stationären Single-Agent MDP eignet, sowie für nicht-stationäre Multi-Agenten Umgebungen.

Zusammenfassend lässt sich sagen, dass ein Lernalgorithmus zu wählen ist, der folgende Eigenschaften erfüllt:

- Der Algorithmus gehört zur Klasse der modellfreien Methoden.
- Der Algorithmus lernt eine stochastische Policy.
- Der Algorithmus ist ein policy-basierter oder Actor-Critic Ansatz.
- Der Algorithmus gehört zur Klasse On-Policy.

Hier fällt die Wahl auf den Lernalgorithmus PPO [Sch+17]. Dieser erfüllt alle gestellten Anforderungen an den Lernalgorithmus. PPO ist ein On-Policy Actor-Critic Lernalgorithmus der eine stochastische Policy lernt und zur Klasse der modellfreien Lernalgorithmen gehört [Sch+17]. Dieser wurde erfolgreich in stationären Single-Agent MDP [Sch+17] sowie in nicht-stationären Multi-Agenten Umgebungen genutzt [Ban+17; Ope+19a]. PPO baut auf dem Algorithmus Trust Region Policy Optimization (TRPO) [Sch+15a] auf. Zunächst wird hierfür der Algorithmus TRPO in kürze erklärt, gefolgt von den Änderungen von PPO.

TRPO stellt einen On-Policy Algorithmus dar. On-Policy Algorithmen optimieren die in der Umgebung eingesetzte Policy direkt. Dies erfolgt indem wie in den Grundlagen beschrieben eine Zielfunktion $J(\pi_\theta)$ maximiert wird, wobei J den erwarteten Return und π_θ die genutzte Policy π mit den NN-Parametern θ beschreibt. Der Gradient der Zielfunktion $J(\pi_\theta)$ in Bezug auf die Parameter des NN θ wird als Policy Gradient bezeichnet [DDZ20, S. 104]:

$$\Delta\theta = \alpha \nabla_\theta J(\pi_\theta) \quad (6.1)$$

wobei α die Lernrate angibt. Da Policy Gradient Methoden eine hohe Varianz haben nutzt TRPO das Konzept *Trust Regions* für die Optimierung der Policy. Hierfür wird die Kullback-Leibler (KL) Divergenz $D_{KL}(\pi_{\theta_{old}} || \pi_\theta)$ zwischen der Ausgangspolicy $\pi_{\theta_{old}}$ und der aktualisierten Policy π_θ genutzt, um die Divergenz der Policy pro Aktualisierung zu begrenzen. Die KL-Divergenz gibt die Ähnlichkeit zweier Verteilungen an [DDZ20, S. 14]. Diese

KL-Divergenz wird bei **TRPO** als Bestrafungsfaktor (Penalty) genutzt, indem eine große Abweichung der Policy zu einer höheren Bestrafung führt. Hierdurch wird von **TRPO** folgende *surrogate* Zielfunktion $L(\theta)$ optimiert:

$$L^{KLPEN}(\theta) = L^{CPI} - \beta D_{KL}(\pi_{\theta_{old}} || \pi_\theta) \quad (6.2)$$

wobei β einen Hyperparameter als Multiplikator der KL-Divergenz darstellt und L^{CPI} als konservative Policy Iteration folgendermaßen definiert ist:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{Adv}_t \right] \quad (6.3)$$

wobei $r_t \theta = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ die Wahrscheinlichkeitsverhältnis zwischen der aktualisierten Policy π_θ und der Ausgangspolicy $\pi_{\theta_{old}}$ für die ausgewählte Aktion A_t in Zustand S_t beschreibt und \hat{Adv}_t den Advantage folgendermaßen berechnet:

$$\hat{Adv}(S_t, A_t) = \sum_{k=0}^{T-t} \gamma^k R_{t+k} - V_{\pi_\theta}(S_t) \quad (6.4)$$

Die Advantage-Funktion gibt an, um wie viel die ausgewählte Aktion A_t in Zustand S_t besser ist als eine zufällig ausgewählte Aktion zu diesem Zustand. Dies wird in Gleichung 7.1 durch die Wertefunktion $V_{\pi_\theta}(S_t)$ dargestellt. [Sch+15a; Sch+17]

PPO baut auf dieser Idee auf, nutzt jedoch statt der KL-Divergenz einen Clipping- Hyperparameter für die Begrenzung der Policy Anpassung. Daraus folgt folgende Zielfunktion [Sch+17]:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(L^{CPI}(\theta), clip(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{Adv}_t) \right] \quad (6.5)$$

wobei ϵ den Clipping-Hyperparameter darstellt. Der zweite Term im min-Operator $clip(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{Adv}_t$ modifiziert das Wahrscheinlichkeitsverhältnis $r_t(\theta)$ um im Intervall $[1 - \epsilon, 1 + \epsilon]$ zu bleiben. Hierdurch wird ebenfalls das starke Ändern der Policy verhindert ähnlich wie das *Trust Region* Konzept bei **TRPO**, gleichzeitig ist diese Zielfunktion einfacher zu berechnen.

Da **PPO** ein Actor-Critic Algorithmus ist, lernt dieser somit eine explizite Policy und eine Wertefunktion. **PPO** ermöglicht das Nutzen eines **NN** für das Abbilden von sowohl Policy und als auch Wertefunktion, da nützliche Informationen für die Wertefunktion auch nützlich für die Policy sein können. Durch Nutzung eines **NN** für beide Aufgaben ist es nötig, die zu optimierende Zielfunktion L^{CLIP} auch um Koeffizienten für die Wertefunktion zu erweitern. Daraus folgt folgende Zielfunktion $L_t^{CLIP+VF+S}(\theta)$, welche zu maximieren gilt:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](S_t) \right] \quad (6.6)$$

wobei c_1, c_2 Koeffizienten und S einen Entropiebonus darstellen, sowie L_t^{VF} den quadrierten Fehler der Wertefunktion darstellt, berechnet durch $(V_\theta(S_t) - V_t^{Ziel})^2$. Durch Nutzung eines Entropiebonus wird eine ausreichende Exploration des Agenten sichergestellt. [Sch+17]

6.2 IMPLEMENTIERUNG

Nach Auswahl des Lernalgorithmus ist eine Implementierung des Algorithmus zu wählen. Die Auswahl der Implementierung kann großen Einfluss auf die Leistung des Agenten haben. So stellten [Eng+20] fest, das Quellcode-Optimierungen in PPO wichtiger sind bezüglich des finalen erreichten Returns als die Auswahl unterschiedlicher Lernalgorithmen, in ihrem speziellen Fall zwischen TRPO und PPO. Diese Quellcode-Optimierungen waren in der initialen PPO-Implementierung [Dha+17] vorhanden, wurden jedoch nicht oder nur teilweise in der Publikation der Autoren [Sch+17] beschrieben. In [Eng+20] werden einige implementierte Quellcode-Optimierungen von den Autoren herausgearbeitet und benannt. Von insgesamt neun gefundenen Quellcode-Optimierungen konnten aufgrund von begrenztem Zugriff auf Rechenressourcen nur vier dieser ausgiebig auf ihren Effekt getestet werden. Sie vergleichen die optimierte PPO Variante mit einer PPO-Minimal genannten Variante, die diese Quellcode-Optimierungen nicht beinhaltet, sowie mit dem Algorithmus TRPO und einer Variante TRPO+, welche um die Quellcode-Optimierungen ergänzt wurde. Sie kamen zum Schluss, dass die Quellcode-Optimierungen einen Großteil des Erfolges von PPO ausmachten. Weiterhin wurde der Erfolg gemessen und gezeigt, dass diese Quellcode-Optimierungen größeren Einfluss auf das Endergebnis haben als die Auswahl des jeweiligen getesteten Lernalgorithmus. Somit ist auch im Rahmen dieser Arbeit darauf zu achten, dass eine Implementierung von PPO verwendet wird, die diese Quellcode-Optimierungen beinhaltet.

Die gefundenen Quellcode-Optimierungen sind [Eng+20]:

1. Die Nutzung des Clipping-Mechanismus auch für das Critic- statt nur für das Actor-Netzwerk
2. Eine Normalisierung des Rewards der Umgebung. Hierbei werden die von der Umgebung erhaltenen Rewards durch die Standardabweichung dieser dividiert, wobei die Standardabweichung während des Trainings laufend aktualisiert wird.
3. Die Nutzung einer orthogonalen Netzwerkinitialisierung, die eine von Netzwerkschicht zu Netzwerkschicht unterschiedliche Skalierung der Netzwerkparameter nutzt.
4. Abhängig von der jeweiligen Umgebung eine während des Trainings immer kleiner werdende Lernrate.
5. Das Nutzen von Reward-Clipping, indem die eingehenden Rewards in einen Wertebereich von üblicherweise $[-5, 5]$ oder $[-10, 10]$ transformiert werden. Dies geschieht indem statt des Rewards die jeweilige

Grenze des Intervalls gewählt wird, wenn diese Intervallsgrenzen vom Reward überstiegen werden.

6. Die Normalisierung von Observationen, wobei diese so transformiert werden, dass sie nullzentriert sind, mit einer Varianz von 1.
7. Die Nutzung von Observations-Clipping, wobei Observationen wie beim Reward-Clipping in ein Intervall von $[-10, 10]$ übertragen werden.
8. Die Nutzung der *Tanh* Aktivierungsfunktion für das Actor- und Critic-Netzwerk.
9. Die Nutzung von Gradienten-Clipping, wobei für Actor- und Critic-Netzwerk die Gradienten so "geclipped" werden, dass diese nicht den Wert 0,5 übersteigen.

Hierbei fällt auf, dass einzelne Quellcode-Optimierungen Hyperparameter von [PPO](#) betreffen, andere wiederum die Umgebung des Agenten, wie die Normalisierung und das Clipping von Rewards und Observationen. Somit ist bei der Wahl der Hyperparameter in Kapitel 6.3 auf diese Optimierungen zu achten. Weiterhin wurde bei der Implementierung der Umgebung darauf geachtet, dass eine Normalisierung nach Punkt (2) und (6) der Quellcode-Optimierungen möglich ist. Hierfür wurde ein *Wrapper* für die Umgebung genutzt, der Standardabweichung und Durchschnitt der einzelnen Werte laufend aktualisiert. Dieser ermöglicht auch die Nutzung des in den Punkten (5) und (7) genannten Clippings für einzelne Umgebungswerte. Hierfür wurde die Referenzimplementierung von [\[Raf+21\]](#) als Wrapper genutzt.

Die aktuell im Quellcode eingesetzte [RL](#)-Bibliothek ist *ChainerRl* [\[Fuj+21\]](#). *ChainerRl* ist eine Implementierung von [RL](#)-Algorithmen auf Grundlage des Deep-Learning-Frameworks *Chainer*. *ChainerRl* ist jedoch nicht plattformunabhängig [\[PNPI15\]](#) und basiert auf einem weniger populärem Deep-Learning-Framework [\[Wen+21\]](#). Gleichzeitig wird die eingesetzte Implementierung seit Dezember 2019 nicht weiterentwickelt und befindet sich im Wartungszustand [\[Tea19\]](#). Da in [\[Eng+20\]](#) gezeigt wurde, dass die Implementierung von [RL](#)-Algorithmen einen großen Einfluss auf die erzielten Ergebnisse haben kann, wird somit im Rahmen dieser Arbeit die Bibliothek [SB3](#) genutzt [\[Raf+21\]](#). Sie ist eine Weiterentwicklung der Implementierung von OpenAI [\[Dha+17\]](#), welche in der Arbeit von [\[Eng+20\]](#) ausgewertet wurde. Hierbei wurde von den Autoren darauf geachtet, die in [\[Eng+20\]](#) genannten Quellcode-Optimierungen bei der Implementierung zu berücksichtigen. Weiterhin hat [SB3](#) eine 95% Quellcode-Testabdeckung, eine hohe Umsetzungsreife [\[KSH20\]](#), ist vollständig Dokumentiert und bietet eine benutzerfreundliche Entwicklungs-API [\[Raf+21\]](#). [SB3](#) bietet zudem eine Tensorboard integration, die verschiedene Parameter des Lernalgorithmus graphisch aufbereitet und somit das Interpretieren des Trainingsverlaufs vereinfacht.

6.3 HYPERPARAMETER

Als nächstes sind die Hyperparameter für die Nutzung von [PPO](#) zu bestimmen. Die Hyperparameterbestimmung ist oft eine Aufgabe die viel Zeit beansprucht [[HT21](#)]. Gleichzeitig sind im Rahmen dieser Arbeit die Rechenressourcen begrenzt und mehrere Versuche durchzuführen. Deshalb wird auf eine zeitintensive Hyperparametersuche verzichtet. Um dennoch Hyperparameter auszuwählen die das Erlernen einer Policy nicht stark negativ beeinträchtigen, werden unterschiedliche optimierte Hyperparameter für andere [RL](#)-Umgebungen ausgewertet, um die Wertebereiche der einzelnen Hyperparameter besser einordnen zu können. Hierfür wird das Trainings-Framework [[Raf20](#)] genutzt, welche für in [SB3](#) implementierte Algorithmen optimierte Hyperparameter für verschiedene [RL](#)-Benchmarkumgebungen beinhaltet. Unter anderem sind auch verschiedene optimierte Hyperparameter-Konfigurationen von [PPO](#) Teil der Sammlung. Für die Optimierung der Hyperparameter wurde das automatische Hyperparameter-Optimierungsframework Optuna genutzt [[Aki+19](#)]. Eine Auflistung der optimierten Hyperparameter ist im Anhang zu finden [A.4,A.5,A.6](#).

Zunächst werden die Hyperparameter von [PPO](#) näher erläutert und mit den standardmäßigen Werten der [SB3](#) Implementierung dargestellt:

- Lernrate: $3e-4$
Die Lernrate beschreibt die Schrittgröße beim Gradienten-Aufstiegs- bzw. Gradienten-Abstiegsverfahren. Eine hohe Lernrate führt zu Anpassung des [NN](#) in größeren Schritten im Vergleich zu einer niedrigen Lernrate.
- Rollout Länge: 2048
Die Anzahl an Schritten die aus der Umgebung gesammelt werden sollen. Mittels dieser wird die Optimierung der Policy mittels [SGD](#) durchgeführt.
- Minibatch Größe: 64
Die gesammelten Transitionen eines Rollouts werden in Minibatches dieser Größe eingeteilt für die Ausführung von Minibatch-[SGD](#).
- Anzahl Epochs: 10
Die Epochs geben an wie viele Iterationen an [SGD](#) mit den Minibatches ausgeführt werden.
- Diskontierungsfaktor: 0,99
Der Diskontierungsfaktor γ des [MDP](#).
- GAE_lambda: 0,95
Als Advantage-Funktion wird Generalized Advantage Estimation ([GAE](#)) genutzt, wobei λ einen Trade-Off zwischen Bias und Varianz bei der Advantage-Schätzung ermöglicht [[Sch+15b](#)].
- Clip-Parameter: 0,2
Der in Kapitel [6.1](#) erläuterte Clipping-Parameter ϵ .

- Clip-Parameter für Wertefunktion: Keine
Dieser Parameter bietet es an, das Clipping auch auf die Wertefunktion anzuwenden, wie in [Eng+20] als Quellcode-Optimierung herausgearbeitet.
- max grad norm: 0,5
Der Clipping Parameter für die Gradienten im Netzwerk, wie in [Eng+20] als weitere Quellcode-Optimierung herausgearbeitet.
- Entropie Koeffizient: 0,0
Der Koeffizient c_2 aus Gleichung 6.6.
- Wertefunktion Koeffizient: 0,5
Der Koeffizient c_1 aus Gleichung 6.6.
- Advantage Normalisierung: Wahr
Parameter, ob die Advantages während des Trainings normalisiert werden sollen.
- NN Architektur: [64, 64]
Die Architektur des NN. Dies kann ein NN für Policy- und Wertefunktion darstellen, oder getrennte NN für beide Lernziele. Standardmäßig wird *Tanh* als Aktivierungsfunktion genutzt, wie in [Eng+20] als Optimierung herausgearbeitet.
- Orthogonale Initialisierung: Wahr
Ein in [Eng+20] herausgearbeiteter Initialisierungsmechanismus, der zu verbesserten Leistungen in vielen Domänen führt. Dies deckt sich auch mit anderen Arbeiten [Rao+20].

Weiterhin werden die in [DB+21] genutzten Hyperparameter zum Trainieren eines Stürmeragenten aufgezeigt:

- Lernrate: 3e-4
- Diskontierungsfaktor: 0,95
- GAE_lambda: 0,95
- Clip-Parameter: 0,2
- Entropie Koeffizient: 0,01
- Wertefunktion Koeffizient: 0,5
- NN Architektur: [512, 512] mit *swish*-Aktivierungsfunktion
Die Autoren nutzten zwei verborgene Schichten mit jeweils 512 Perzeptren. Als Aktivierungsfunktion wurde die *swish*-Funktion genutzt.

Wie zuvor gezeigt beinhalten die in [Eng+20] aufgezeigten Quellcode-Optimierungen auch einzelne Hyperparameter für die Nutzung von PPO. Diese sind zum

einen die Nutzung der *Tanh* Aktivierungsfunktion, von orthogonaler Initialisierung, von Gradienten-Clipping vom Wert 0,5, das Nutzen des Clipping-Mechanismus für die Wertefunktion und einer immer kleiner werdenden Lernrate, diese jedoch nur für einige Umgebungen. Bis auf die Lernrate werden diese von [Eng+20] ausgearbeiteten Hyperparameter im Rahmen dieser Arbeit genutzt. Weiterhin wurden in [Sch+17] einzelne Clip-Parameter evaluiert mit dem Ergebnis, dass der Wert 0,2 über mehrere Umgebungen hinweg positive Ergebnisse erzielen konnte. Dadurch sind folgende Hyperparameter festgelegt:

- Clip-Parameter: 0,2
- Clip-Parameter für Wertefunktion: 0,2
- max grad norm: 0,5
- Wertefunktion Koeffizient: 0,5
- Orthogonale Initialisierung: Wahr

Betrachtet man die ausgewerteten optimierten Hyperparameter in A.4,A.5,A.6 so lässt sich feststellen, dass alle Umgebungen eine Advantage Normalisierung nutzen. Dies entspricht auch dem Standardwert für die SB₃ Implementierung von PPO, weshalb diese auch im Rahmen dieser Arbeit genutzt wird. Weiterhin kann für den Hyperparameter GAE_Lambda beobachtet werden, dass ein Wertebereich von 0,8 bis 1 in den Umgebungen genutzt wurde, wobei am häufigsten der Wert 0,95 genutzt worden ist. Dieser Wert ist ebenfalls von [DB+21] genutzt worden. GAE_Lambda stellt einen Koeffizienten für die Berechnung des Advantage-Wertes dar. In [Sch+15b] wird ein Wertebereich von [0,9; 0,99] als Wertebereich angeführt der normalerweise zu den besten Ergebnissen führt. Da es keinen speziellen Grund gibt im Rahmen der in dieser Arbeit genutzten Umgebung einen anderen Wert auszuwählen, wird somit der Standardwert von 0,95 genutzt.

Betrachtet man den Diskontierungsfaktor γ in den optimierten Hyperparametern, so lässt sich feststellen, dass die genutzten Werte in einem Intervall zwischen einmal 0,9 und zweimal 0,9999 liegen. Wie in den Grundlagen erläutert beeinflusst der Diskontierungsfaktor den Planungszeitraum eines Agenten. Bei einem Diskontierungsfaktor von 1 bedeutet dies, dass alle gesammelten Rewards in einer Episode gleich wichtig sind, während ein Diskontierungsfaktor von 0 zur Folge hat, dass lediglich der aktuelle Reward zählt [SB15, S. 60]. Durch Nutzung eines kleineren Diskontierungsfaktors wird somit eine Unsicherheit über die Umgebung ausgedrückt, da Rewards die weiter weg in der Zukunft liegen weniger zählen als aktuelle Rewards. In [And+20] wurden verschiedene Hyperparameter auf ihren Einfluss auf das Lernverhalten von PPO analysiert. Hierbei wurden auch unterschiedliche Werte für den Diskontierungsfaktor in unterschiedlichen Umgebungen genutzt. Ihre Ergebnisse zeigen, dass der Wert 0,99 ein durch alle Umgebungen hinweg gutes Ergebnis erzielen konnte. Aufgrund dessen wird dieser Wert als Hyperparameter gewählt.

Als nächstes werden die Rollout Länge, Minibatch Größe und die Anzahl an Epochs betrachtet. Eine geringere Rollout Länge reduziert die Erfahrungen die der Agent bis zur Policy Optimierung sammeln kann. In unterschiedlichen Versuchen wurden deswegen hohe Werte für die Rollout Länge gewählt um die Varianz der gesammelten Erfahrungen zu minimieren und stabilere Optimierungen vornehmen zu können [Ban+17; Ope+19a; Bak+19]. Betrachtet man die optimierten Hyperparameter A.4,A.5,A.6 so wurden unterschiedliche Werte für die einzelnen Hyperparameter gewählt, was eine Auswahl erschwert. Auch diese Hyperparameter wurden in [And+20] auf ihren Einfluss auf das Lernverhalten von PPO analysiert. Die Autoren kamen zum Schluss, dass eine reduzierte Rollout Länge das Training stark negativ beeinflussen kann. Anders hingegen scheint die Größe der Minibatches eine geringere Rolle für den Trainingserfolg zu spielen. Aus diesem Grund wird keine Reduktion der Rollout Länge vorgenommen, sondern der Standardwert von 2048 genutzt, da dieser am häufigsten in den optimierten Hyperparametern im Anhang genutzt wurde. Auch in [And+20] konnte mit diesem Wert durchweg gute Ergebnisse erzielt werden. Aufgrund des geringen Einfluss der Minibatch-Größe wird hier ebenfalls beim Standardwert von 64 verblieben. Dieser Wert ist auch am häufigsten genutzt in den optimierten Hyperparametern im Anhang. In [And+20] wird weiterhin bezüglich der Anzahl an Epochs dieser als wichtiger Faktor für die Probeneffizienz von PPO ausgearbeitet. Hierbei konnte mit dem Wert 10 in allen Umgebungen durchweg die höchste Punktzahl erzielt werden. Eine weitere Erhöhung des Wertes kann jedoch dazu führen, dass eine Überanpassung vorgenommen wird, welche zu einer Destabilisierung des Trainings führen kann. In komplexen Umgebungen wird deswegen die Anzahl an Epochs reduziert [Ban+17; Yu+21]. Aus diesem Grund wird hierfür der Wert 10 genutzt und nicht weiter gesteigert.

Zur Bestimmung der Lernrate kann in den optimierten Hyperparametern gesehen werden, dass 13/32 Umgebungen einen Wert in der Größenordnung $1e-5$, während 17/32 einen Wert in Größenordnung $1e-4$ nutzen. Hierbei kann beobachtet werden, dass in komplexeren Umgebungen eine kleinere Lernrate genutzt wird, gemessen an der Dimensionalität von Aktions- und Observationsraum. Dies kann auch in [And+20] gesehen werden, in denen mit der Lernrate von $3e-4$ in den meisten Umgebungen gute Resultate erzielt werden konnten, wobei eine zu hohe Lernrate gerade in Komplexen Umgebungen das Lernen stark erschweren kann. Da die Komplexität der in dieser Arbeit zu nutzenden Umgebung schwer einzuschätzen ist, wird zunächst eine kleinere Lernrate von $3e-5$ gewählt. Da kleinere Lernraten in komplexen Umgebungen das Lernen ermöglichen und in simpleren Umgebungen dies nur verlangsamen, können zu große Lernraten das Lernen verhindern. Bei Wahl dieses Wertes ist jedoch bei den Versuchen darauf zu achten, eine ausreichend lange Trainingsdauer zu bestimmen, damit die kleinere Lernrate kompensiert wird.

Zuletzt sind die NN-Architektur und die Nutzung eines Entropie Koeffizienten festzulegen. In [And+20] konnte festgestellt werden, dass die Nutzung

von zwei verborgenen Schichten für das [NN](#) zu positiven Resultaten in allen Umgebungen geführt hat, wobei die Breite des Netzwerkes ein Hyperparameter zu sein scheint, der starken Einfluss auf die Leistung des Trainings haben kann. Hierbei finden die Autoren, dass dies jeweils abhängig von der Umgebung ist und keine generelle Empfehlung ausgesprochen werden kann. Betrachtet man die optimierten Hyperparameter, so nutzen 22/32 Umgebungen die Standardkonfiguration mit zwei verborgenen Schichten und jeweils 64 Perzeptren, während die anderen 10 Umgebungen getrennte [NN](#) für Policy- und Wertefunktion mit jeweils zwei verborgenen Schichten à 256 Perzeptren nutzen. Durch Nutzung einer geringeren Netzwerkgröße müssen weniger Parameter angepasst werden, was zu einer schnelleren Konvergenz während des Trainings führen kann. Aus diesem Grund wird hierfür die Standardkonfiguration von [64, 64] als [NN](#)-Architektur genutzt. Dieser Wert ist jedoch mit Vorsicht zu betrachten, da in [[And+20](#)] die Breite des [NN](#) ein signifikanter Faktor für das Training sein kann.

Die Nutzung eines Entropie-Koeffizienten begünstigt das Ausprobieren verschiedener Aktionen durch den Agenten. Auch in [[DB+21](#)] wurde ein Entropie-Koeffizient von 0,01 genutzt um einem Stürmeragenten das Schießen beizubringen. In den optimierten Hyperparameterkonfigurationen nutzen 14 von 32 Umgebungen unterschiedliche Entropie-Koeffizienten. Auch in [[And+20](#)] wurde dieser Hyperparameter analysiert mit dem Schluss, dass dieser in manchen Umgebungen helfen kann, jedoch die Leistung des Agenten kaum negativ beeinflusst falls nicht. Aufgrund dessen wird im Rahmen dieser Arbeit ein Entropie-Koeffizient von 0,01 gewählt, da hierdurch das Ausprobieren durch den Agenten in der Umgebung verstärkt wird. Dies ist gerade als Stürmeragent wichtig um nicht in ein lokales Optimum zu fallen, indem beispielsweise keine Schüsse mehr vorgenommen werden, da der Ball oft von der Wand zurückgeprallt ist.

Somit werden folgende Hyperparameter als Ausgangswerte für die Nutzung im Rahmen dieser Arbeit festgelegt:

- Lernrate: 3e-5
- Rollout Länge: 2048
- Minibatch Größe: 64
- Anzahl Epochs: 10
- Diskontierungsfaktor: 0,99
- GAE_lambda: 0,95
- Clip-Parameter: 0,2
- Clip-Parameter für Wertefunktion: 0,2
- max grad norm: 0,5
- Entropie Koeffizient: 0,01

- Wertefunktion Koeffizient: 0,5
- Advantage Normalisierung: Wahr
- [NN](#) Architektur: [64, 64] mit *Tanh* Aktivierungsfunktion
- Orthogonale Initialisierung: Wahr

EXPERIMENT ZUR BESTIMMUNG DES OBSERVATIONSRAUMS

Es gibt unterschiedliche Arbeiten, die sich mit der Auswirkung und Wahl von Observationsräumen in [RL](#) befasst haben [[KH20](#); [RTP20](#)]. In [[KH20](#)] wurde eine Evaluierung verschiedener Observationsräume vorgenommen und ein Optimierungsalgorithmus für die automatischen Optimierung eines Observationsraums entwickelt. In ihrer Arbeit beziehen sie sich auf das [RL](#)-Problem Locomotion, d.h. einem Agenten das Laufen beizubringen. Hierfür nutzten sie die Umgebungen *HalfCheetah*, *Hopper*, *Walker2D* und *InvertedDoublePendulum* der [RL](#)-Benchmarks OpenAI Gym und DeepMind Control Suite. Beide [RL](#)-Benchmarks bieten unterschiedliche Observationsräume für dieselben Umgebungen, was die Autoren als Ausgangslage für die Entwicklung eines Algorithmus zur Bestimmung eines optimalen Observationsraums nutzen.

Um in dieser Arbeit den zu nutzenden Observationsraum zu definieren, wird der in [[KH20](#)] beschriebene Ansatz genutzt. Ziel des Experiments ist es, aus den in Kapitel [5.1](#) bestimmten potentiellen Observationskanälen einen Observationsraum zu entwerfen, der nur nützliche Observationskanäle beinhaltet, statt weniger nützliche bzw. schädliche Observationskanäle. Dies ist wichtig, da wie die Autoren betonen [RL](#)-Probleme sensitiv gegenüber der Problemdefinition sind. Durch Nutzung ihres Optimierungsalgorithmus wird sichergestellt, dass die genutzten Observationskanäle das Lernverhalten des Agenten nicht negativ beeinflussen.

Hierfür wird zunächst der Optimierungsalgorithmus aus [[KH20](#)] näher erläutert. Ihr Algorithmus besteht aus zwei Teilen, einem Such-Algorithmus [1](#) für die Bestimmung der zu testenden Observationsräume sowie einen *Dropout-Permutationstest* [2](#) zur Bestimmung schädlicher Observationskanäle:

Algorithm 1 Suchalgorithmus für Observationsräume

Input: Modell, ObsKanäle, Trainingsdauer K , Testmetrik

```

1: ausgangsModell ← erstelleModell(initialerObsRaum)
2: ausgangsModell.trainiere( $K$ )
3: initialerWert ← ausgangsModell.werteAusMit(Testmetrik)
4: bestePunktzahl, besterObsRaum ← initialePunktzahl, initialerObsRaum
5: while nicht konvergiert do
6:   aktuellerObsRaum ← besterObsRaum  $\cup$  neueObsKanalGruppe
7:   aktuellesModell ← erstelleModell(aktuellerObsRaum)
8:   aktuellesModell.trainiere( $K$ )
9:   aktuellePunktzahl ← aktuellesModell.werteAusMit(Testmetrik)
10:  if aktuellePunktzahl > bestePunktzahl then
11:    aktuellerObsRaum ← entferneSchädlicheObsKanäle(aktuellerObsRaum)
    ▷ Nutzung von Algorithmus 2
12:    bestePunktzahl, besterObsRaum ← aktuellePunktzahl, aktuellerObsRaum
13:  end if
14: end while
15: return besterObsRaum

```

Eigene Darstellung basierend auf [KH2o].

Algorithm 2 Dropout-Permutationstest zum Entfernen von ObsKanälen

Input: ObsRaum, Dropout-Wahrscheinlichkeit d , Grenzwert \hat{I} , Testmetrik

```

1: modell ← erstelleModell(ObsRaum)
2: modell.trainiere( $K$ , dropout=(modell.eingabeSchicht, dropout-wahrscheinlichkeit= $d$ ))
3: vergleichsPunktzahl ← modell.werteAusMit(Testmetrik)
4: for  $i = 1, 2, \dots, N$  do
5:   punktzahl ← modell.werteAusMit(Testmetrik,
      obs $i$ =nutzeStattOriginalerWerte(Verteilung=obs $i$ .Verteilung))
6:   importance-score ← (punktzahl – vergleichsPunktzahl) / vergleichsPunktzahl
7: end for
8: Entferne ObsKanäle wo gilt: importance-scoreobs-kanal <  $\hat{I}$ 
9: return ObsRaum

```

Eigene Darstellung basierend auf [KH2o].

Zunächst wird Algorithmus 1 näher erläutert. Ausgehend von einem initialen Observationsraum wird ein Agent mit diesem trainiert und die erreichte Leistung bestimmt. Anschließend werden diesem initialem Observationsraum iterativ einzelne bzw. gruppierte Observationskanäle hinzugefügt und das Lernverhalten des Agenten mit diesem Observationsraum gemessen.

In diesem Versuch werden die in Kapitel 5.1.3 definierten potentiell nutzbaren Observationskanäle gruppiert und eine Evaluierung mittels des Opti-

mierungsalgorithmus vorgenommen. Die Gruppierung wird hier wie in der Arbeit der Autoren semantisch vorgenommen, da dies zu besserem Lernerfolg beitrug als die Nutzung zufällig gruppierter Observationskanäle. Zur Beurteilung des Lernverhaltens werden die Rewardkurven der Agenten miteinander verglichen, indem die Steigungen und die maximalen kumulierte Rewards bzw. Return miteinander verglichen werden. Hierdurch werden Rechenressourcen gespart, da keine zusätzliche Evaluierung stattfinden muss. Dies entspricht auch der Vorgehensweise der Autoren, welche sowohl die Rewardkurven und zusätzliche Evaluationsversuche als Evaluationsmetrik auswerteten und zum Schluss kamen, dass beide Methoden vergleichbare Ergebnisse zeigten.

Verbessert sich das Lernverhalten im Vergleich zum zuvor getesteten Observationsraum, so werden die hinzugefügten Observationskanäle beibehalten und dieser neue Observationsraum auf schädliche Observationskanäle geprüft. Dies findet in Algorithmus 2 statt. Hierbei wird ein Agent mit dem auf schädliche Observationskanäle zu prüfenden Observationsraum trainiert. Bei diesem Training wird ein Dropout-Mechanismus genutzt, um den Agenten robuster gegen das Entfernen einzelner Observationskanäle werden zu lassen. In Abbildung 7.1 ist dies bildlich verdeutlicht.

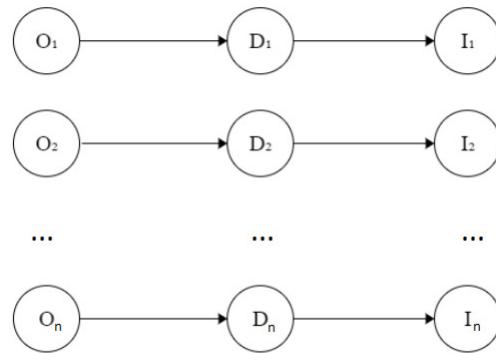


Abbildung 7.1: Repräsentation der Dropout-Schicht: O_1, \dots, O_n stellen die Observationskanäle, D_1, \dots, D_n die Dropout-Kanäle und I_1, \dots, I_n die Eingabeschicht des NN dar

Hier sieht man, dass die Dropout-Schicht zwischen den Observationskanälen und der Eingabeschicht des NN steht. Für jedes Perzeptron wird für jede Observation bestimmt, ob die reale Observation ins NN gegeben wird oder ein anderer Wert stattdessen. Die Entscheidung hierfür wird pro Observationskanal aufgrund einer definierten Wahrscheinlichkeit getroffen. In ihrer Arbeit [KH20] haben die Autoren unterschiedliche Wahrscheinlichkeiten für den Dropout evaluiert. Hierbei konnte bei hoher Dropout-Wahrscheinlichkeit von 0,3 eine starke Beeinträchtigung der Effektivität der Policy beobachtet werden, im Vergleich zu einer geringeren Dropout-Wahrscheinlichkeit von 0,01. Ist die Dropout-Wahrscheinlichkeit jedoch zu gering, so ist die Policy nicht robust genug gegenüber dem Entfernen von Observationskanälen, was die Beurteilung der Nützlichkeit einzelner Observationskanäle erschwert. In

ihrer Arbeit nutzten die Autoren deswegen eine Dropout-Wahrscheinlichkeit von 0,1 und konnten damit gute Ergebnisse erzielen. Dieser Wert wird auch in dieser Arbeit als Dropout-Wahrscheinlichkeit genutzt.

Weiterhin ist zu bestimmen, welcher Wert bei stattfinden eines Dropout an das **NN** weiterzureichen ist. Referenzimplementierungen [Ten22; Con19] nutzen hierfür einen Wert von 0, um die Deaktivierung eines Perzepron zu simulieren, wie von den Autoren der Dropout-Technik vorgeschlagen [Sri+14]. In ihrer Arbeit [KH20] wurden jedoch zufällige Werte gewählt, die realistisch sind. Hierfür wurden während des Trainings die Observationen gesammelt und eine Verteilung für jeden Observationskanal erstellt. Ist ein Observationskanal vom Dropout betroffen, wird ein zufälliger Wert dieser Verteilung gewählt. Hierdurch werden falsche, aber dennoch realistische Observationen weitergegeben. Dies wird auch im Rahmen dieser Arbeit so vorgenommen. Anschließend wird der Agent in der jeweiligen Umgebung evaluiert und der Einfluss der einzelnen Observationskanäle auf die Leistung gemessen. Hierfür wird zunächst eine Vergleichsmetrik erstellt, indem der trainierte Agent in der Umgebung eingesetzt und dessen Leistung gemessen wird. Anschließend werden iterativ einzelne Observationskanäle des Agenten in der Umgebung mit falschen Daten ersetzt und die damit erzielte Leistung in der Umgebung gemessen, wie in Zeile 5 in Algorithmus 2 dargestellt. Indem die Leistung durch Wegfall einzelner Observationskanäle mit der Vergleichsmetrik verglichen werden, wird ein sogenannter *Importance-Score* gebildet, welcher den Einfluss des jeweiligen Observationskanals auf die Agentenleistung darstellt. Anschließend werden auf Grundlage des Importance-Scores schädliche Observationskanäle entfernt und dieser bereinigte Observationsraum als neuer Ausgangspunkt für die nächste Iteration von Algorithmus 1 genutzt, bis alle Observationskanäle durchiteriert worden sind.

Die Entscheidung ob ein Observationskanal entfernt werden soll wird auf Grundlage eines bestimmten Grenzwertes \hat{I} getroffen. Liegt der erreichte Importance-Score unter diesem Grenzwert, wird der Observationskanal entfernt. Die Autoren definieren jedoch weder den genutzte Grenzwert, noch eine Methode zur Bestimmung des Grenzwertes. Es lässt sich anhand ihrer getroffenen Auswahl jedoch feststellen, dass schädliche Observationskanäle entfernt worden sind. Schädliche Observationskanäle sind durch ihren positiven Importance-Score bestimmbar. Dies bedeutet, dass durch ihre Entfernung der Agent eine bessere Leistung erzielen konnte als mit diesem Observationskanal.

7.1 VORGEHENSWEISE UND VERSUCHSAUFBAU

Für die Nutzung des Optimierungsalgorithmus in dieser Arbeit wird ein Stürmer trainiert, der lernen soll Tore zu schießen. Hierfür wird der in Kapitel 6 bestimmte Lernalgorithmus **PPO** mit der in Kapitel 5.3 bestimmten Rewardfunktion genutzt. Um die Varianz der Ergebnisse zu verringern werden die Versuche fünf mal wiederholt. Hierbei werden unterschiedliche Seeds für die Zufallsgeneratoren genutzt. Grund hierfür ist, dass dadurch die **NN**

unterschiedlich initialisiert werden. Durch diese unterschiedliche Initialisierung wird versucht sicherzustellen, dass die erzielten Ergebnisse nicht allein auf eine zufällige Initialisierung der [NN](#) beruhen. Dies ist gängige Praxis in verschiedenen [RL](#)-Arbeiten [[Hen+17](#)]. Unterschiedliche Seeds werden auch für die Unity-Simulation genutzt, damit die Ergebnisse auch unabhängiger von der zufälligen Ballpositionierung werden.

Es sind jedoch weitere Parameter zu bestimmen, um den Optimierungsalgorithmus anwenden zu können. Zum einen ist für den Lernalgorithmus zu prüfen, ob für diesen Versuch die in Kapitel [6.3](#) bestimmten Hyperparameter angepasst werden müssen. Weiterhin ist die zu nutzende Trainingsdauer und ein Aktionsraum festzulegen, da in Kapitel [5.2](#) noch eine Entscheidung offen ist ob diskrete oder kontinuierliche Aktionen zu nutzen sind, sowie die jeweilige Kodierung. Zudem ist eine Episodendefinition und ein Evaluationsszenario für den Dropout-Permutation-Test zu bestimmen. Hierbei definiert die Episodendefinition den konkreten Aufbau und die Endbedingungen einer Episode. Diese Parameter werden im folgenden Abschnitt festgelegt.

7.1.1 Observationskanäle

In Kapitel [5.1](#) wurde folgender Initialer Observationsraum ([IO](#)) entworfen, der die Markov-Eigenschaft erfüllt:

- Lateralposition der Stange
- Winkelposition der Stange
- Laterale Bewegungsgeschwindigkeit
- Rotationsgeschwindigkeit
- X-Position des Balls
- Z-Position des Balls
- X-Geschwindigkeit des Balls
- Z-Geschwindigkeit des Balls

Weiterhin wurden folgende potentiell nützliche Observationskanäle bestimmt:

- Spielerpositionen der Stange
- Binäre Kontaktinformation
- Binäre Reichweiteninformation

Für die Nutzung des Optimierungsalgorithmus von [[KH20](#)] sind Observationskanäle iterativ einem Observationsraum hinzuzufügen und deren Einfluss auf das Training zu evaluieren. Hierfür eignet sich die Nutzung von [IO](#) als Observationsraum, dem iterativ die einzelnen potenziell nützlichen

Observationskanäle zugefügt werden. Als semantische Gruppierung der iterativ hinzuzufügenden Observationskanäle bietet es sich an, die Spielerpositionen der Stange separat und die Binärinformationen gemeinsam zu gruppieren.

Daraus folgt, dass zunächst ein Agent mit dem Observationsraum IO trainiert wird, gefolgt vom iterativen Hinzufügen der Spielerpositionen der Stange im zweiten Durchlauf. In der letzten Iteration werden dann die binären Kontakt- und Reichweiteninformationen dem Observationsraum hinzugefügt.

7.1.2 *Aktionsraum*

In Kapitel 5.2 wurde aufgezeigt, dass die Nutzung diskreter Aktionsräume einen positiven Einfluss auf das Lernverhalten haben kann im Vergleich zur Nutzung von kontinuierlichen Aktionsräumen. Dies muss jedoch nicht immer der Fall sein, wie [TA19] gezeigt hat. Da für die Bestimmung der Observationsräume die Rewardkurven der jeweiligen Observationsräume miteinander verglichen werden ist davon auszugehen, dass eine gesteigerte Effizienz durch Nutzung eines diskretisierten Aktionsraums alle Observationsräume betreffen würde. Da der Aktionsraum des Tischkickers von Natur aus kontinuierlich ist und nicht davon auszugehen ist, dass durch Nutzung eines diskreten Aktionsraums ein anderes Ergebnis bei der Bestimmung der zu nutzenden Observationskanäle entsteht, wird auf die Diskretisierung des Aktionsraums für die Bestimmung der Observationsräume verzichtet und ein kontinuierlicher Aktionsraum genutzt. Hierfür wird ebenfalls ein normalisierter Wertebereich von $[-1, 1]$ gewählt, sowohl für die Lateralbewegung als auch für die Rotation der Stange. In Kapitel 8 findet dann eine experimentelle Bestimmung des Aktionsraums statt.

7.1.3 *Trainingsdauer und Hyperparameter*

Zur Bestimmung der Trainingsdauer und Hyperparameter für diesen Versuch werden die in Kapitel 6.3 bestimmten Hyperparameter als Ausgangswerte genutzt, sowie die Trainingsdauer für den trainierten Stürmeragenten in [DB+21]. Die in Kapitel 6.3 definierten Hyperparameter sind folgende:

- Lernrate: 3×10^{-5}
- Rollout Länge: 2048
- Minibatch Größe: 64
- Anzahl Epochs: 10
- Diskontierungsfaktor: 0,99
- GAE_lambda: 0,95
- Clip-Parameter: 0,2

- Clip-Parameter für Wertefunktion: 0,2
- max grad norm: 0,5
- Entropie Koeffizient: 0,01
- Wertefunktion Koeffizient: 0,5
- Advantage Normalisierung: Wahr
- **NN** Architektur: [64, 64] mit *Tanh* Aktivierungsfunktion
- Orthogonale Initialisierung: Wahr

In [DB+21] konnten die Autoren einem Stürmer das Torschießen nach 48 Stunden Training auf einem Business-Laptop mit vergleichbarer Leistung im Vergleich zu den verfügbaren Rechenressourcen beibringen A.1. Die Rewardkurve hat nach diesem Trainingszeitraum noch keine Konvergenz erreicht. Zu sehen ist, dass bereits nach 24 Stunden ein Stürmer trainiert werden konnte, der aus verschiedenen Positionen Tore schießen konnte. Hierbei ist auch die genutzte Größe des **NN** zu betrachten. Mit einer Größe von [512, 512] ist das von [DB+21] genutzte **NN** deutlich größer als das für diese Arbeit bestimmte **NN** der Größe [64, 64]. Weiterhin konnte in ihrer Arbeit innerhalb von 24 Stunden insgesamt ca. 300.000 Schritte in der Umgebung durch den Agenten durchgeführt werden. Betrachtet man die in dieser Arbeit in Kapitel 5.1.1 bestimmte Observationsfrequenz von 60 Hz, so können in 24 Stunden insgesamt 5,184 Mio. Schritte durch den Agenten in der Umgebung vorgenommen werden. In Verbindung mit der deutlich kleineren **NN**-Größe von [64, 64] ist davon auszugehen, dass eine deutlich geringere Trainingsdauer für das Beobachten erster Resultate benötigt sein sollte. Verglichen mit den positiven Resultaten nach ca. 300.000 Schritten in [DB+21] wird somit eine Trainingsdauer für diesen Versuch von 11 Stunden gewählt, was 2,16 Mio. Schritten in der Umgebung entspricht. Dies ist immer noch ein Vielfaches höher als die von [DB+21] benötigten Trainingsschritte. Dadurch ist auch davon auszugehen, dass die geringere Lernrate kompensiert wird. Gleichzeitig ist durch die kleinere Lernrate mit einer geringeren Varianz der Rewardkurve zu rechnen, da kleinschrittigere Änderungen an der Policy vorgenommen werden.

7.1.4 Episodendefinition

Nachdem der Observations- und Aktionsraum sowie die Trainingsdauer und Hyperparameter definiert worden sind, ist eine Episodendefinition zu bestimmen. Dies beinhaltet den Aufbau und die Position der einzelnen Komponenten am Anfang jeder Episode und wann eine Episode beendet wird. Im folgenden Abschnitt wird zunächst der Aufbau jeder Episode beschrieben, gefolgt von den Endbedingungen einer Episode

7.1.4.1 Aufbau einer Episode

Wie bereits in Kapitel 3 erläutert, ist in dieser Arbeit der Fokus auf die Trainingseffizienz zu setzen, da die verfügbaren Rechenressourcen begrenzt sind. Dies ist auch bei Bestimmung des Episodenaufbau zu berücksichtigen. Um das zu erlernende Verhalten einfacher zu gestalten werden die gegnerische Torwart- und Verteidigerstange während des Trainings hochgeklappt, um kein Hindernis für den Stürmer darzustellen. Dies macht das Problem einfacher zu lösen und reduziert somit die benötigte Trainingsdauer um aussagefähige Resultate durch den Optimierungsalgorithmus zu erhalten. Hierbei wird die Annahme getroffen, dass die Observationskanäle die nützlich für den Stürmer-Agenten auch nützlich für den Torwart-Agenten sind. Dies wird damit begründet, dass beide Agenten im Rahmen dieser Arbeit ein ähnliches Verhalten zu lernen haben. Während der Stürmer-Agent den Ball ins Tor schießen soll und nicht hinter die eigene Stange lassen darf, hat der Torwart-Agent ein vergleichbares Ziel, indem der Ball nicht ins Tor darf, aber jedoch am Stürmer vorbei geschossen werden soll. Durch diese Ähnlichkeit der beiden zu erlernenden Aufgaben ist davon auszugehen, dass nützliche Observationskanäle für den Stürmer-Agenten auch nützlich für den Torwart sind.

Des weiteren wurden in [RTP20] verschiedene Eigenschaften von Agenteenumgebungen und deren Einflüsse auf RL analysiert. Unter anderem wurde die initiale Zustandsverteilung als großer Faktor für die Trainingseffizienz herausgearbeitet. Diese Initialzustandsverteilung beschreibt die Verteilung der Zustände zu Beginn einer jeden Episode. Eine breite Initialzustandsverteilung bedeutet, dass zu Beginn von Episoden der Initialzustand aus einer großen Anzahl von möglichen Zuständen gewählt wird. Dementsprechend bedeutet eine schmale Initialzustandsverteilung, dass zu Beginn von Episoden ein Initialzustand aus einer kleinen Anzahl von möglichen Zuständen gewählt wird. In ihrer Studie kamen die Autoren zum Schluss, dass RL-Ergebnisse stark von der Auswahl der Initialzustandsverteilung betroffen sind. Je breiter die Initialzustandsverteilung desto schwerer kann ein RL-Problem werden. Dies führt zu einer Verschlechterung der Trainingseffizienz und dem gesammelten Reward, jedoch können diese trainierten Agenten in der Regel besser Generalisieren und sind robuster als Agenten die mit einer schmalen Initialzustandsverteilung trainiert worden sind. Somit wird für den Versuch zur Bestimmung des Observationsraums eine schmale Initialzustandsverteilung genutzt, da dies die Trainingseffizienz im Vergleich zu einer breiteren Verteilung erhöht und die erlernte Policy nur zur Auswahl der Observationskanäle dient, die zu effizienterem Training führen.

Nun ist zu bestimmen, wie die Initialzustandsverteilung in diesem Versuch konkret definiert ist. Der Stürmer soll in diesem Versuch das Schießen lernen. Somit ist der relevante Bereich des Kickertisches der Bereich zwischen der Stürmerstange und dem jeweiligen Tor des gegnerischen Teams. Für das Schießen muss die Kickerstange so bewegt werden, dass der Ball nach Kontakt mit der Stange ins Tor rollt. Zum Initialzustand gehört die initiale

Ballposition sowie die initiale Stürmerstangenposition.

Zunächst wird die Initialzustandsverteilung der Stangenposition bestimmt. Damit die Verteilung möglichst schmal ist, wird die Stürmerstange zu Beginn jeder Episode in die mittlere Ausgangsposition versetzt. Um dennoch eine gewisse Diversität an Initialzuständen zu erhalten, wird die Ballposition bei Beginn jeder Episode variiert. Hierfür ist zu bestimmen, in welchem Bereich der Ball bei Beginn einer Episode zu setzen ist. Um die Umgebung möglichst einfach erlernbar zu machen bietet es sich an, den Ball knapp vor der Stürmerstange zu platzieren. Da sich die Stürmerstange lateral bewegen und rotieren lässt und beide Fähigkeiten relevant für das Schießen von Toren sind, sollten beide Fähigkeiten in diesem Versuch trainiert werden. Somit bietet es sich nicht an, den Ball direkt vor den Stürmerfiguren so zu platzieren, dass die Stürmerstange nur rotieren muss um ein Tor zu schießen. Die Initialzustände sollten so gewählt werden, dass der Agent zum einen beide Fähigkeiten lernen muss um ein Tor zu schießen, zum anderen ist darauf zu achten, dass diese Initialzustände aus einer relativ schmalen Verteilung zu wählen sind. Hierfür wird für diesen Versuch der Ball so vor den Stürmerfiguren platziert, dass der Ball immer auf der gleichen X-Koordinate in einem Abstand von 4 cm zur Stange liegt. Dieser Abstand wurde so gewählt, da hierdurch eine signifikante Rotation zur Ballberührungsfläche erforderlich ist. Da die Stürmerstange drei Figuren besitzt, wird der Ball zu Beginn einer Episode zufällig vor einer der drei Figuren platziert. Dabei wird die Z-Koordinate so gewählt, dass sie in einem Bereich liegt der ausgehend von der Mitte der Spielfigur in beide Richtungen 4 cm beträgt. Dies ist in Abbildung 7.2 verdeutlicht.

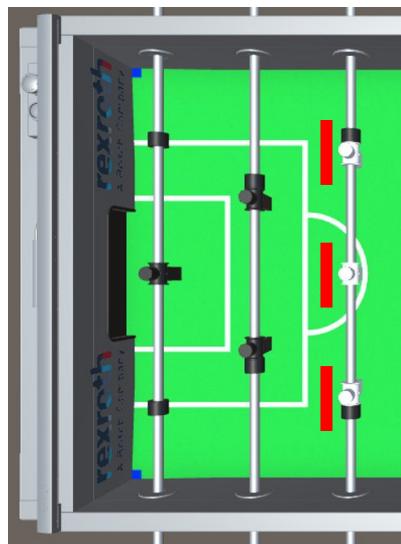


Abbildung 7.2: Initialzustandsverteilung des Balls

Dadurch wird zum einen sichergestellt, dass der Ball nicht ohne signifikante Rotation getroffen werden kann und zum anderen, dass unterschiedliche laterale Bewegungen benötigt werden um den Ball aus den verschiedenen Positionen ins Tor zu schießen.

7.1.4.2 Ende einer Episode

Nun wird das Ende einer Episode bestimmt. Hierfür sind zunächst die Terminalzustände zu bestimmen, die im Laufe des Trainings auftreten können. Hierzu zählen ein gefallenes Tor und eine Ballposition hinter der Stürmerstange, sodass der Ball von dieser nicht mehr berührt werden kann. Wie bereits zuvor beschrieben gibt es für diese Terminalzustände auch Rewards von der Umgebung.

Nun ist jedoch die Frage zu stellen, ob ein zeitliches Limit für Episoden zu setzen ist. Dies würde bedeuten, dass sobald eine Zeitüberschreitung des Limits erfolgt, die Episode beendet wird, auch wenn der Ball sich noch auf dem Spielfeld befindet. Zeitliche Limits sind oft genutzt in RL-Umgebungen, da hierdurch die Vielfalt der Observationen erhöht wird [Par+17]. Diese sind auch in dieser Arbeit erforderlich, da in den offiziellen internationalen Tischkickerregeln [fedo7, S. 7-8] eine Spielunterbrechen geregelt ist, wenn der Ball in einer von keiner Stange erreichbaren Position zum Stehen kommt. Da in diesem Versuch lediglich die Stürmerstange aktiv ist, wird die Episode beendet wenn der Ball außerhalb deren Reichweite stehen bleibt. Ein weiteres zeitliches Limit der offiziellen Regeln bestimmt, dass es in Bezug auf die Stürmerstange ein 15 Sekunden Zeitlimit für das Wechseln des Ballbesitzes gibt [fedo7, S. 16]. Da in diesem Versuch die Stürmerstange die einzige aktive Stange im Spielfeld ist, wird somit ein zeitliches Limit von 15 Sekunden für eine Episode bestimmt.

Jedoch ist ein Beenden einer Episode durch Überschreitung des zeitlichen Limits kein Terminalzustand des MDP. Hierbei ist der zu nutzende Lernalgorithmus PPO erneut zu berücksichtigen. Wie in Kapitel 6.1 erläutert nutzt PPO eine Advantage-Funktion \hat{Adv}_t um den Wert einer gewählten Aktion A_t in Zustand S_t zu bestimmen, indem dieser Wert mit dem antizipierten Wert durch zufälliges Auswählen einer Aktion verglichen wird, dargestellt durch den Zustandswert $V_\pi(S_t)$:

$$\hat{Adv}(S_t, A_t) = \sum_{k=0}^{T-t} \gamma^k R_{t+k} - V_\pi(S_t) \quad (7.1)$$

Somit nutzt diese Methode bootstrapping, da statt des tatsächlichen Zustandswert eine Approximation aufgrund der zu lernenden Wertefunktion erfolgt. PPO nutzt als Advantage-Funktion GAE [Sch+15b], was durch Nutzung eines Koeffizienten λ einen Trade-off zwischen Bias und Varianz bei Berechnung des Advantage konfigurierbar macht. Hierbei wird ein TD-Ansatz ähnlich wie in [Sut88] genutzt, indem folgender Wert y für das Aktualisieren der Wertefunktion genutzt wird:

$$y = \begin{cases} r & \text{für Terminalzustände} \\ r + \gamma V_\pi(S_{t+1}) & \text{andernfalls} \end{cases} \quad (7.2)$$

Hierbei werden jedoch in Implementierungen oft fälschlicherweise keine Unterscheidung zwischen Terminalzuständen des **MDP** und zeitlich bedingten Terminierungen einer Episode vorgenommen [Par+17]. Somit wird von den Lernalgorithmen keine Unterscheidung getroffen, ob ein Terminalzustand des **MDP** erreicht wurde oder die Episode aufgrund eines zeitlichen Limits zurückgesetzt wurde, das lediglich zur Erhöhung der Trainingseffizienz genutzt wurde und nicht Teil der Problemdomäne ist. Mit diesem Problem haben sich die Autoren von [Par+17] auseinandergesetzt. In ihrer Arbeit kamen sie zum Schluss, dass die Nutzung einer Unterscheidung zwischen Terminalzustand und Zeitüberschreitung stark positive Auswirkungen auf das Lernverhalten des Agenten hat. Somit ist bei der Implementierung darauf zu achten, dass diese Unterscheidung ebenfalls berücksichtigt wird. Dies ist bei der gewählten Implementierung von **SB₃** der Fall.

7.1.4.3 *Domain Randomization*

Wie bereits zuvor erklärt, ist Domain Randomization ein Mittel zur Erhöhung der Robustheit der erlernten Policy des Agenten. Diese wird oft genutzt, da eine Simulation nicht exakt der Realität entspricht. Durch Nutzung von Domain Randomization werden einzelne Parameter der Simulation randomisiert, damit die erlernte Policy nicht zu stark auf die Eigenheiten der Simulation abgerichtet ist. Dies führt jedoch zu einer breiteren Initialzustandsverteilung, da die Zustände sich auch innerhalb dieser gesetzten Parameter wie beispielsweise Ballbreite etc. unterscheiden können. Wie zuvor erläutert, ist der Fokus dieser Arbeit jedoch eine möglichst hohe Trainingseffizienz. Weiterhin haben einzelne Experimente gezeigt, dass ein Agent durch Nutzung von Domain Randomization Schwierigkeiten beim Lernen einer Policy haben kann, sodass in manchen Fällen kein Verhalten erlernt werden konnte [Ban+17, S. 8]. Da sich diese Arbeit jedoch nur mit der Simulation beschäftigt, nur begrenzte Rechenressourcen zur Verfügung stehen und die trainierten Agenten in der Simulation evaluiert werden und nicht auf dem realen Tischkicker, wird Domain Randomization nicht genutzt. Bei erfolgreichen Resultaten können für die anschließende Portierung der Agenten diese nochmals mittels Domain Randomization und erhöhten Rechenressourcen trainiert werden.

7.1.5 *Evaluationsszenario*

Als Evaluationsszenario werden 100 Episoden mit dem trainierten Agenten abgehalten und die Rewards protokolliert. Da fünf Agenten trainiert werden mit fünf verschiedenen Seeds, werden insgesamt fünf Evaluierungen durchgeführt, eine für jeden trainierten Agenten. Weiterhin stellt sich die Frage, welcher trainierte Agent zu evaluieren ist. Hier bieten sich zwei Möglichkeiten an. Es kann der Agent zum Ende des Trainings genutzt werden oder der Agent, der während des Trainings die beste Leistung erzielt hat gemessen am gesammelten Return pro Rollout. Es ist davon auszugehen, dass die Leistungen beider Agentenzwischenstände durch die Wahl der Observati-

onskanäle beeinflusst wird. Um eine Auswahl zu treffen ist der Zweck des Trainings hervorzuheben. Ein Agent wird für eine gewisse Dauer trainiert, um dessen Leistung zu steigern bzw. einen Agenten mit erhöhter Leistung zu erhalten. Somit ist grundsätzlich die maximale Leistung die durch das Training aufgebaut werden kann von Interesse, da ein Agent für eine Aufgabe trainiert wird um diese möglichst gut zu lösen. Daraus folgt, dass sich der beste Agent gemessen am erzielten Return pro Rollout für die Evaluierung eher eignet die statische Nutzung des Agenten am Ende des Trainings.

7.2 ERGEBNISSE UND AUSWERTUNG

Alle Versuche wurden mit fünf unterschiedlichen Seeds ausgeführt. Da Ergebnisse aus RL-Versuchen dennoch stark variieren können ist darauf zu achten, dass diese Varianz bei Auswertung der Daten berücksichtigt wird. In einer Arbeit [Hen+17] werden hierfür Methoden aufgezeigt die genutzt werden sollten, um die Aussagekraft bei RL-Versuchen zu erhöhen. Hier wird die Nutzung von bootstrapped 95%-Konfidenzintervallen empfohlen, um verlässliche Aussagen über die Verteilung der Ergebnisse machen zu können. Diese Methode wurde auch im Rahmen dieser Arbeit genutzt.

Abbildung 7.3 stellt das Ergebnis der ersten Iteration dar. Diese stellt den durchschnittlichen kumulierten Reward bzw. durchschnittlichen Return pro Rollout dar, wobei ein Rollout 2048 Schritte in der Umgebung entspricht. Hierbei wurde für die rote Kurve der initiale Observationsraum IO genutzt, während für die grüne Kurve der IO um die Spielerposition der Stange erweitert wurde. Die erzielten Werte zeigen somit, wie viele Tore im Durchschnitt während eines Rollouts durch den Agenten erzielt werden konnten. Die fixe Rolloutlänge von 2048 Schritten entspricht bei 60 Schritten pro Sekunde ca. 34 Sekunden.

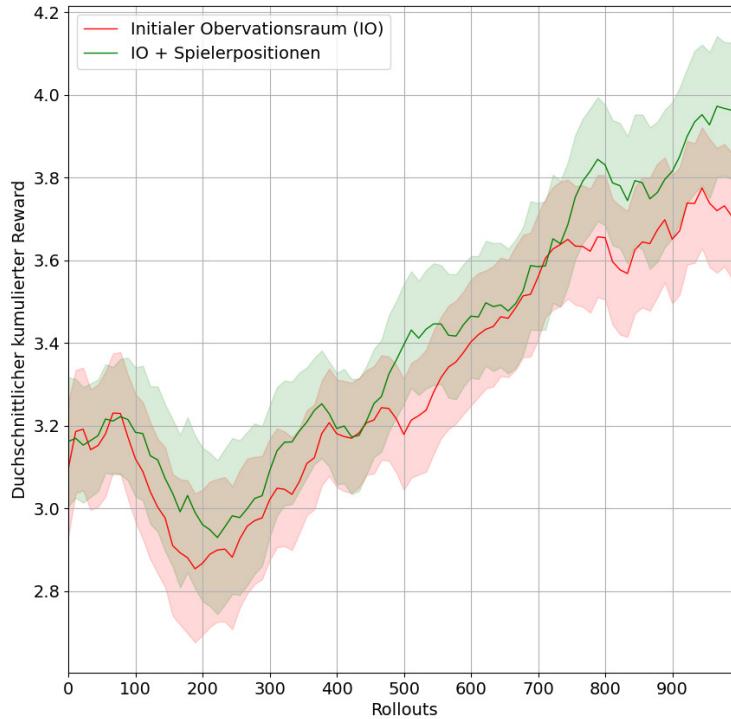


Abbildung 7.3: Durchschnittlicher kumulierter Reward pro Rollout. Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.

Hierbei kann beobachtet werden, dass durch Nutzung der Spielerpositionen gerade zum Ende des Trainings ein durchschnittlich höherer Return erzielt werden konnte. Vergleicht man den höchsten erreichten Return durch Nutzung des initialen Observationsraums (3,77) und des erweiterten Observationsraums (3,97) so wird eine leichte Verbesserung von ca. 5,3% erzielt. Somit wird die Entscheidung getroffen, die Spielerpositionen dem Observationsraum hinzuzufügen.

Als nächstes wurde dieser Observationsraum dem Dropout-Permutation-Test unterzogen und die einzelnen Einflüsse der Observationskanäle ausgewertet. Abbildung 7.4 stellt den durchschnittlichen Return pro Episode dar. Die X-Achse stellt die einzelnen Observationskanäle sowie die Vergleichsmetrik "baseline" dar. Der Wert für baseline entspricht einem Agenten, der mittels zuvor beschriebenem Dropout-Mechanismus trainiert wurde und anschließend in der Evaluationsumgebung für 100 Schüsse ausgeführt wurde, ohne dass dabei die Observationen modifiziert worden sind. Die einzelnen Observationskanäle auf der X-Achse stellen die Observationskanäle dar, die während der Ausführung in der Evaluationsumgebung manipuliert wurden. Hierbei wurde statt des originalen Wertes der Observation für den jeweiligen Observationskanal, ein falscher Wert genutzt, der wie zuvor beschrieben

aus einer Verteilung entnommen wurde, die aus den im Training gesehenen Werten für diesen Observationskanal entspricht. Die Histogramme der einzelnen Observationskanäle sind im Anhang beigefügt [A.4](#), [A.5](#).

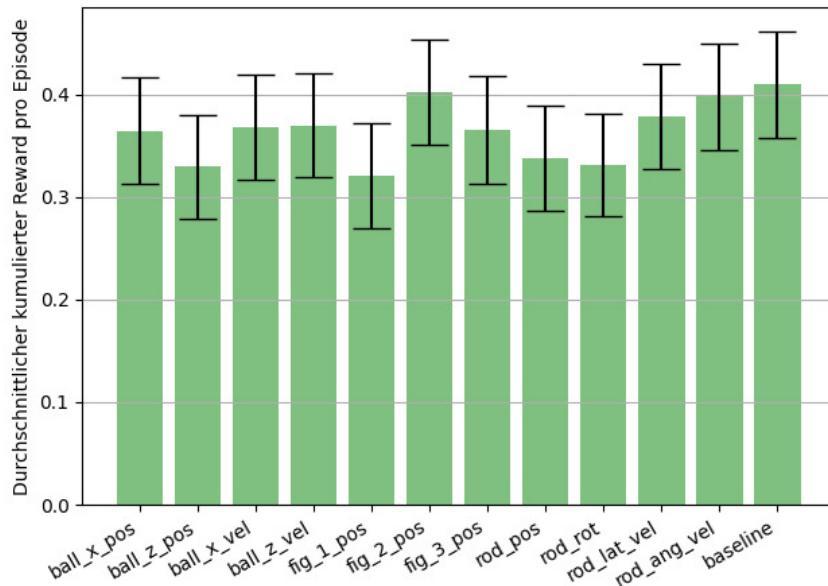


Abbildung 7.4: Durchschnittlicher kumulierter Reward pro Episode bei Manipulation des jeweiligen Observationskanales auf der X-Achse. "Baseline" stellt die Vergleichsmetrik ohne Manipulation dar. Darstellung beinhaltet bootstrapped 95%-Konfidenzintervalle

Wie im Diagramm zu sehen erzielt die Vergleichsmetrik "baseline" den höchsten Wert. Hierbei ist darauf zu achten, dass der maximal mögliche Return 1 beträgt (Schießen eines Tors) und der minimal mögliche -0,1 (Ball landet hinter Stürmer). Ein durchschnittlicher Return von 1 bedeutet somit, dass der Stürmer jede Episode ein Tor erzielt hat, während ein durchschnittlicher Return von -0,1 bedeuten würde, dass jeder Ball hinter den Stürmer gelandet ist.

Diese Werte wurden nun genutzt um die Importance-Scores der einzelnen Observationskanäle zu berechnen. Diese sind in Abbildung [7.5](#) dargestellt.

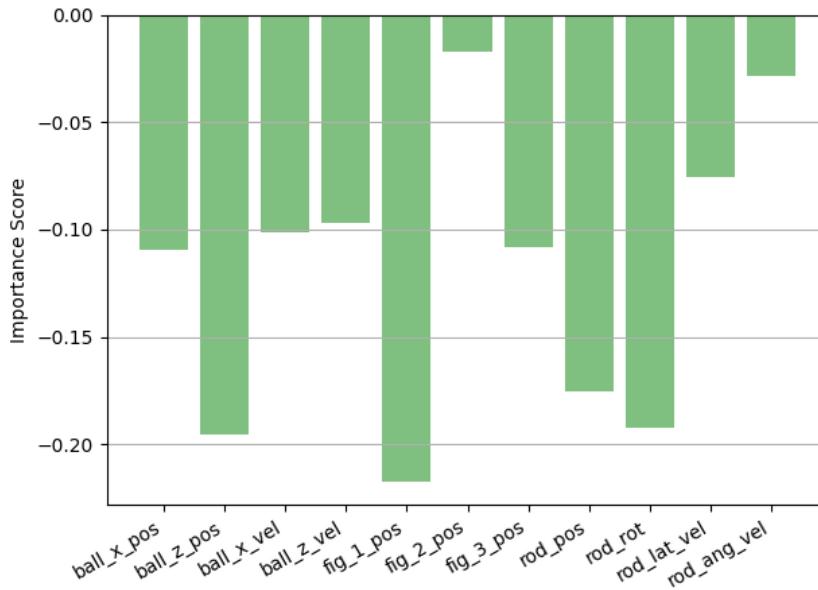


Abbildung 7.5: Importance-Score der einzelnen Observationskanäle im Vergleich zur "baseline"

Der Importance-Score quantifiziert den Einfluss eines Observationskanals auf die Leistung des Agenten. Dieser wird wie in Algorithmus 2 gezeigt folgendermaßen berechnet $IS = (score - base_score) / score$, wobei $score$ den durchschnittlichen Return pro Episode des jeweiligen Observationskanals aus Abbildung 7.4 darstellt und $base_score$ den erzielten Wert von "baseline" aus Abbildung 7.4 darstellt. Ein positiver Wert deutet darauf hin, dass ohne diesen Observationskanal ein besseres Ergebnis durch den Agenten erzielt werden konnte. Somit haben schädliche Observationskanäle einen positiven Importance-Score. Ein negativer Wert deutet darauf hin, dass ohne den Observationskanal ein schlechteres Ergebnis durch den Agenten erzielt werden konnte und zeigt somit einen nützlichen Observationskanal auf.

Hierbei ist festzustellen, dass alle Observationskanäle einen negativen Importance-Score aufweisen und somit kein besseres Ergebnis durch den Agenten erzielt werden konnte, indem ein Observationskanal entfernt worden ist. So lässt sich feststellen, dass vor allem ohne die erste Stürmerfigur, die Z-Koordinate des Balls oder die Lateral- und Winkelpositionen der Stange deutlich schlechtere Ergebnisse erzielt werden konnte. Gleichzeitig weist jedoch die mittlere Stürmerfigur nur einen geringen Mehrwert auf im Vergleich zu den andern Observationskanälen.

Da in der Arbeit der Autoren des Optimierungsalgorithmus [KH20] keine Methode zur Bestimmung des Grenzwertes aufgezeigt wird, um auf Grundlage dessen zu entscheiden welchen Observationskanal zu entfernen, wird ein Versuch ohne den Observationskanal der mittleren Stürmerfigur fig_2_pos durchgeführt. Hierbei soll festgestellt werden ob eine Reduktion des Observationsraums den Mehrwert des Observationskanal überbietet oder nicht. Das Ergebnis dieses Versuches ist in Abbildung 7.6 dargestellt.

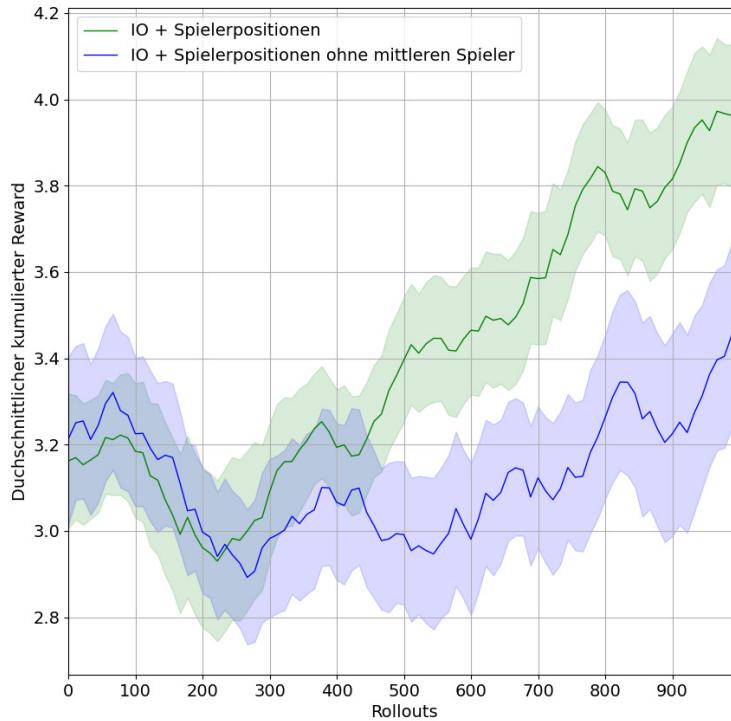


Abbildung 7.6: Durchschnittlicher kummulierter Reward pro Rollout für den erweiterten Observationsraum mit Spielerpositionen (grün) und den erweiterten Observationsraum mit Spielerpositionen ohne den mittleren Spieler (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.

Überraschenderweise sinkt die Leistung des Agenten durch entfernen des mittleren Spielers signifikant. Dies ist umso erstaunlicher, da der Wert des Importance-Scores für diesen Observationskanals am niedrigsten war im Vergleich zu den anderen Observationskanälen. Aufgrund dieses Ergebnisses wird vom Entfernen von Observationskanälen mit negativem Importance-Score abgesehen.

Zuletzt werden dem erweiterten Observationsraum bestehend aus **IO** und Spielerpositionen die binären Kontakt- und Reichweiteninformationen hinzugefügt. Das Ergebnis dieses Versuches ist in Abbildung 7.7 dargestellt.

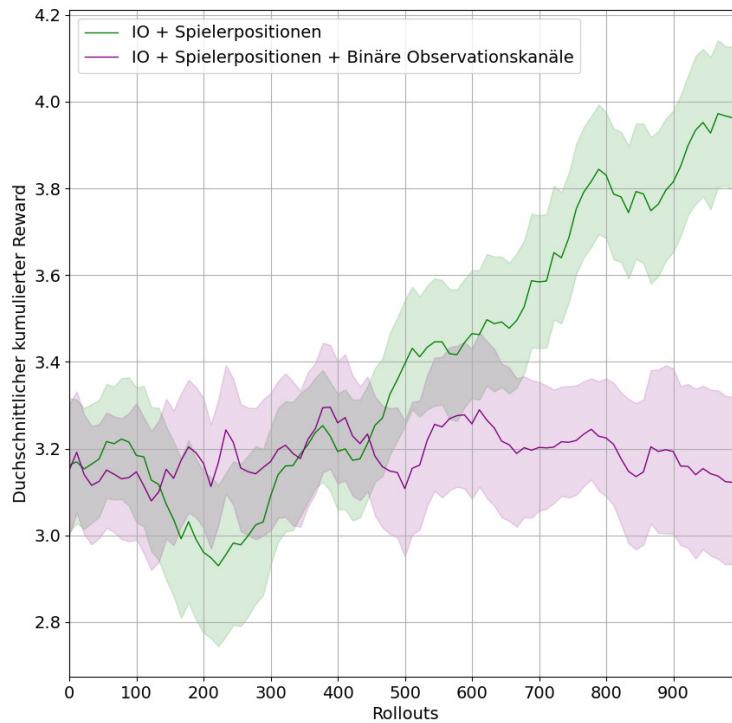


Abbildung 7.7: Durchschnittlicher kumulierter Reward pro Rollout. Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.

Hierbei ist zu sehen, dass durch hinzufügen der binären Observationskanäle es zu einer deutlichen Verschlechterung des Lernverhaltens kommt. Die Lernkurve des Agenten steigt minimal und sinkt jedoch im Laufe des Trainings leicht. Somit werden die binären Observationskanäle nicht dem Observationsraum hinzugefügt.

Abschließend lässt sich folgender Observationsraum für die Nutzung im Rahmen dieser Arbeit festhalten:

- Lateralposition der Stange
- Winkelposition der Stange
- Laterale Bewegungsgeschwindigkeit
- Rotationsgeschwindigkeit
- X-Position des Balls
- Z-Position des Balls
- X-Geschwindigkeit des Balls

- Z-Geschwindigkeit des Balls
- Spielerpositionen der Stange

Hierbei konnte gesehen werden, dass durch Nutzung der Spielerpositionen das Lernverhalten des Agenten leicht verbessert werden konnte. Gleichzeitig konnte gezeigt werden, dass die binären Kontakt- und Reichweiteninformationen zu einer deutlichen Verschlechterung des Lernverhaltens des Agenten geführt haben. Als überraschendes Ergebnis ist hierbei hervorzuheben, dass selbst durch Entfernung des Observationskanals der mittleren Spielerfigur mit relativ geringem Importance-Score im Vergleich zu den anderen Observationskanälen, es dennoch zu einer deutlichen Abnahme der Lerngeschwindigkeit gekommen ist.

Weiterhin kann bezüglich aller Trainingsverläufe festgestellt werden, dass die Trainingsdauer von 2,16 Mio. Schritten nicht lang genug war um eine Konvergenz zu erreichen. Somit ist im Rahmen dieser Arbeit bei den weiteren Versuchen darauf zu achten diese Trainingsdauer ggf. anzupassen.

EXPERIMENT ZUR BESTIMMUNG DES AKTIONSRAUMS

Nachdem der Observationsraum definiert worden ist wird der Aktionsraum näher betrachtet. Wie bereits in Kapitel 5.2 erläutert, muss ein diskreter Aktionsraum nicht immer zu effizienterem Training von Agenten führen. In komplexen Umgebungen jedoch kann durch eine Diskretisierung eine höhere Trainingseffizienz erreicht werden. Ziel dieses Versuches ist es herauszufinden, ob für die Steuerung der Tischkickerstangen ein diskreter Aktionsraum effizienteres Trainieren ermöglicht als ein kontinuierlicher.

8.1 VORGEHENSWEISE UND VERSUCHSAUFBAU

Um zu prüfen ob eine Diskretisierung des Aktionsraums zu effizienterem Training bei Tischkickeragenten führt, werden zwei Agenten trainiert die sich im genutzten Aktionraum unterscheiden. Hierfür wird ein Stürmeragent mit kontinuierlichem Aktionsraum trainiert und einer mit diskretisierten Aktionsraum. Da sowohl Stürmer- als auch Torwartstangen den gleichen Aktionsraum besitzen ist davon auszugehen, dass ein in diesem Versuch gefundener Aktionsraum der das Lernverhalten des Stürmers verbessert auch positive Auswirkungen auf den Torwart hat.

Der Versuchsaufbau ist der gleiche wie in Kapitel 7.1. Es werden hier ebenfalls die Torwart- und Verteidigerstange hochgeklappt während die Stürmeragenten das Torschießen lernen sollen. Die genutzten Observationsräume entsprechen dem in Kapitel 7.2 bestimmten Observationsraum, außerdem wird die gleiche Rewardfunktion wie in 5.3 genutzt. Die Episodendefinition ist die gleiche wie in 7.1.4, ebenso wie der genutzte Lernalgorithmus PPO mit den in 7.1.3 genutzten Hyperparametern. Wie sich im Versuch zur Bestimmung des Observationsraums gezeigt hat, konnte mit der genutzten Trainingsdauer von 2,16 Mio. Schritten keine Konvergenz der Rewardkurven erreicht werden. Aufgrund dessen wird in diesem Versuch die Trainingsdauer verdoppelt auf 4,32 Mio. Schritte. Dies wird auch zeigen, ob mit dieser längeren Trainingsdauer eine Konvergenz erreicht werden kann. Anschließend werden die Rewardkurven des Trainings miteinander verglichen um zu überprüfen, ob die Diskretisierung einen positiven Effekt auf die Trainingseffizienz hat. Wie beim Versuch zur Bestimmung des Observationsraums wird der Versuch fünf mal mit unterschiedlichen Seeds für die Zufallsgeneratoren genutzt.

In der Arbeit [TA19] werden unterschiedliche Netzwerkarchitekturen vorgeschlagen, um eine Diskretisierung vorzunehmen. Sie nutzen in ihrer Arbeit ebenfalls den Lernalgorithmus PPO und untersuchen den Einfluss einer Diskretisierung eines kontinuierlichen Aktionsraum auf das Lernverhalten

des Agenten. In ihrer Arbeit stellen sie zwei Policies vor. Zum einen eine multi-diskrete Policy, zum anderen eine multi-diskrete Policy mit ordinaler Netzwerkarchitektur. Beide Netzwerkarchitekturen erzielten in komplexen Umgebungen wie *Humanoid* signifikante Leistungszuwächse im Vergleich zu einem kontinuierlichen Aktionsraum. In simpleren Umgebungen wie *Ant* konnte ein kontinuierlicher Aktionsraum jedoch bessere Ergebnisse erzielen als die diskretisierten Varianten.

Während die ordinale Netzwerkarchitektur nicht in [SB₃](#) implementiert ist, ist die multi-diskrete Policy für den Lernalgorithmus [PPO](#) in [SB₃](#) verfügbar. Zwar ist in der Arbeit der Autoren der Quellcode beinhaltet, dieser ist jedoch mehr als zwei Jahre alt zum Zeitpunkt dieser Arbeit und wurde nicht weiter gepflegt. Hinzu kommt, dass ihre Implementierung auf einer veralteten Version des Frameworks *Tensorflow* basiert und nicht mit aktuellen Versionen ausgeführt werden kann. Gleichzeitig wurde in Kapitel [6.2](#) erläutert, dass einzelne Quellcode-Optimierung größeren Einfluss auf die Leistung eines Lernalgorithmus haben als die Auswahl unterschiedlicher Lernalgorithmen. Da die genutzte Implementierung [SB₃](#) sowohl diese Quellcode-Optimierung als auch die multi-diskrete Policy beinhaltet, wird diese Implementierung genutzt um diesen Versuch durchzuführen.

In diesem Rahmen werden somit ein kontinuierlicher und multi-diskreter Observationsraum mittels [SB₃](#) evaluiert. Auch in [\[KSH20\]](#) werden unterschiedliche Aktionsraumtransformationen evaluiert, mit dem Resultat, dass eine Diskretisierung des Aktionsraum zu schnellerem Lernen des Agenten führen können. Hierbei wird im Vergleich zwischen multi-diskreten und diskreten Observationsräumen zur Nutzung von multi-diskreten Aktionsräumen geraten, da diese auch mit höheren Dimensionen bessere Ergebnisse liefern als diskrete Aktionsräume mit ähnlicher Dimensionalität. Für die Diskretisierung wird in [\[KSH20\]](#) jedoch keine Methode vorgeschlagen. Unterschiedliche Diskretisierungsparameter werden jedoch in [\[TA19\]](#) ausgewertet, ebenso wie die Sensitivität dieses Hyperparameters. Hierfür wird von den Autoren eine Diskretisierung eines kontinuierlichen Aktionsraum in 7, 11 und 15 Kategorien vorgenommen und evaluiert. Hierbei kamen die Autoren zum Schluss, dass durch eine zu geringe Diskretisierung der Agent zu viel Kontrolle in der Umgebung verliert. Weiterhin führt eine zunehmende Diskretisierung zu einer erhöhten Rechenlast, die ungefähr linear mit Zunahme der Diskretisierung steigt. In ihren Versuchen hat ein Diskretisierungsfaktor von 11 durchweg die besten Ergebnisse erzielt. Somit wird für diesen Versuch ebenfalls dieser Diskretisierungsfaktor genutzt.

Zusammenfassend wird ein zweidimensionaler multi-diskreter Aktionsraum mit jeweils 11 Kategorien pro Dimension [11, 11] und ein zweidimensionaler kontinuierlicher Aktionsraum miteinander verglichen. Für die Kodierung des kontinuierlichen Aktionsraums wird der in [\[TA19; KSH20\]](#) genutzte Wertebereich von $[-1, 1]$ gewählt. Dieser bietet sich wie zuvor beschrieben auch in Bezug auf das zuvor erläuterte Vanishing Gradient Problem an. Der Wer-

tebereich wird für beide Dimensionen genutzt, wodurch der Aktionsraum folgendermaßen dargestellt werden kann: $[-1, 1], [-1, 1]$.

8.2 ERGEBNISSE UND AUSWERTUNG

Wie in Kapitel 7.2 werden auch hier die einzelnen Versuche fünf mal mit unterschiedlichen Seeds durchgeführt. Gleichzeitig werden die Rewardkurven mit ihrem bootstrapped 95%-Konfidenzintervall in den Diagrammen dargestellt. In Abbildung 8.1 werden die Rewardkurven des Stürmeragenten unter Nutzung des kontinuierlichen Aktionsraums mit der Nutzung des multi-diskreten Aktionsraums verglichen.

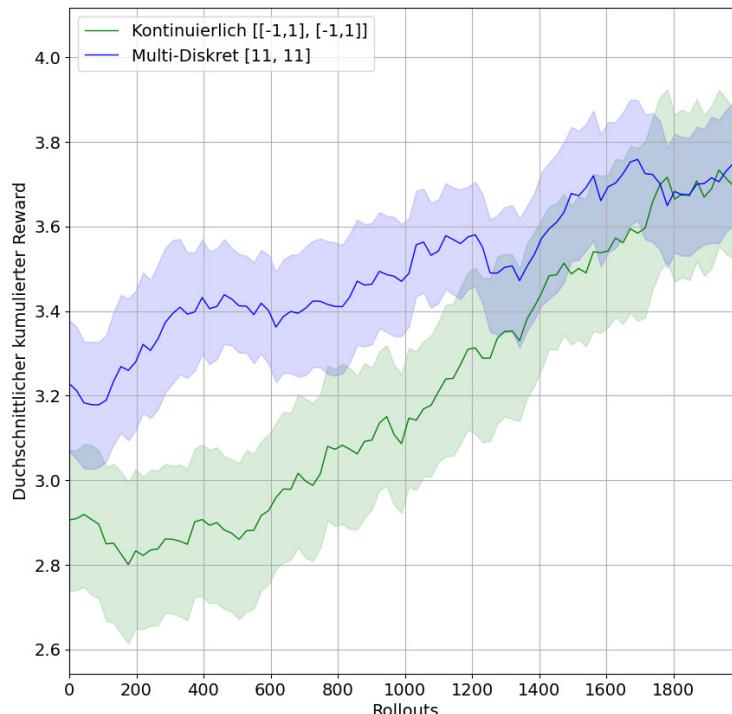


Abbildung 8.1: Durchschnittlicher kumulierter Reward pro Rollout für kontinuierlichen Aktionsraum (grün) und multi-diskretem Aktionsraum (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.

Wie zu sehen hat der multi-diskrete Aktionsraum bis ca. Rollout 1600 einen höheren durchschnittlichen Return pro Rollout. Ab Rollout 1600 bis 2000 erreicht der kontinuierliche Aktionsraum jedoch den gleichen durchschnittlichen Return pro Rollout wie der multi-diskrete Aktionsraum. Gleichzeitig ist jedoch festzuhalten, dass beide Agenten trotz doppelter Trainingsdauer nur leichte Fortschritte in der Umgebung machen. Aus diesem Grund wird der Versuch mit angepassten Hyperparametern wiederholt, um ein aus-

sagekräftigeres Ergebnis zu erhalten.

Aufgrund der langen Trainingsdauer von ca. 20 Stunden und der dennoch geringen Steigung der Rewardkurven wird die Lernrate wie in Kapitel 6.3 angekündigt auf 3e–4 erhöht. Weiterhin wird die Breite des **NN** erhöht, sodass statt [64, 64] wird ein Netzwerk der Größe [128, 128] genutzt wird, da dies wie in Kapitel 6.3 beschrieben einen großen Einfluss auf den Trainingserfolg eines **NN** haben kann. Durch diese Vergrößerung wird dessen Kapazität erhöht, was zu einer besseren Approximation komplexer Funktionen führen kann [GBC16, S. 110]. Wird ein **NN** jedoch zu stark vergrößert, kann dies zu einer Überanpassung an die jeweiligen Trainingsdaten führen und schlechte Generalisierungseigenschaften zur Folge haben [FFL22a]. Die gewählte Netzwerkarchitektur ist jedoch immer noch deutlich kleiner als die in [DB+21] genutzte Netzwerkarchitektur von zwei Schichten mit jeweils 512 Perzeptren, mit der gute Ergebnisse erzielen konnten. Somit ist nicht davon auszugehen, dass die Nutzung eines **NN** der Größe [128, 128] für dieses Problem zu einer Überanpassung führt.

Das Ergebnis des wiederholten Versuchs angepassten Hyperparameter ist in Abbildung 8.2 dargestellt.

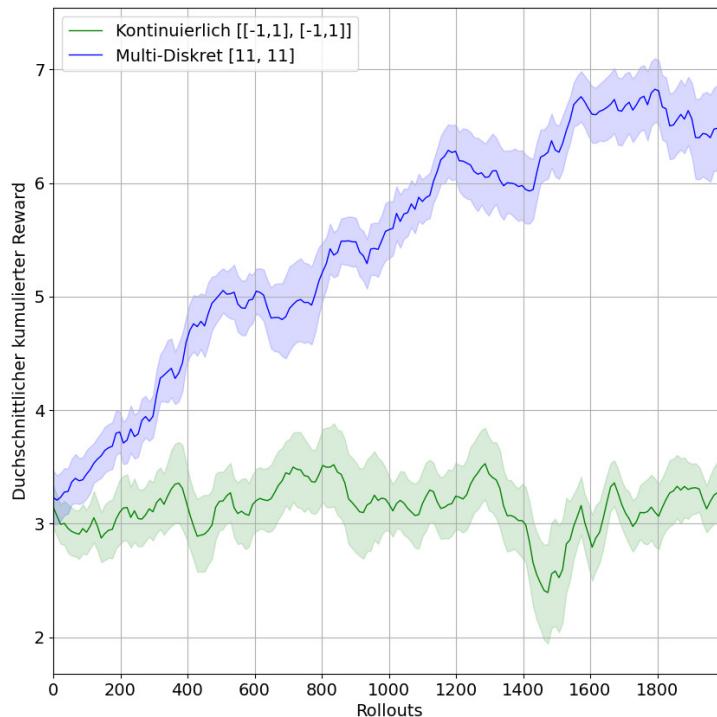


Abbildung 8.2: Durchschnittlicher kummulierte Reward pro Rollout für kontinuierlichen Aktionsraum (grün) und multi-diskretem Aktionsraum (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.

Es ist deutlich zu erkennen, dass die angepassten Hyperparameter zu einer starken Verbesserung des erlernten Verhaltens für den multi-diskreten Aktionsraum geführt haben. Bei der Nutzung des multi-diskreten Aktionsraums kann mit den getesteten Hyperparametern das Erreichen einer Konvergenz ab ca. 1500 Rollouts abgelesen werden, entsprechend ca. 3 Mio. Schritten in der Umgebung. Gleichzeitig stellt sich das Erlernen einer guten Policy mit kontinuierlichem Aktionsraum und den angepassten Hyperparametern deutlich schwerer dar. Im Vergleich zum Versuch mit geringerer Lernrate und kleinerer NN-Architektur ist eine starke Varianz der Rewardkurve zu beobachten, welche nach dem Training keine Verbesserung zeigt. Eine mögliche Erklärung hierfür ist, dass die Nutzung eines kontinuierlichen Aktionsraums das RL-Problem schwerer zu lernen machen. Dies könnte auch erklären, warum mit einer geringeren Lernrate ein besseres Ergebnis bei kontinuierlichem Aktionsraum erzielt werden konnte. Das kontinuierliche Aktionsräume schwerer zu lernen sind deckt sich mit den Ergebnissen aus [TA19; KSH20], in denen gerade in komplexen Umgebungen die Nutzung einer Diskretisierung das zu lösende RL-Problem stark vereinfachen kann.

Zusammenfassend wird somit ein zweidimensionaler multi-diskreter Aktionsraum [11, 11] für die Nutzung festgelegt, da mit diesem signifikant bessere Ergebnisse erzielt werden konnten als mit einem kontinuierlichen Aktionsraum.

MULTI-AGENT COMPETITION

Nachdem Observations- und Aktionsraum für die Agenten bestimmt worden sind, gilt es als nächstes das Konzept zum Trainieren von Stürmer- und Torwartagenten zu definieren.

Tischkicker stellt eine Domäne dar, in der zwei Spieler gegeneinander spielen. Somit ist das Spiel Tischkicker ein Mehrparteien-Spiel. Daraus folgt, dass ein guter Stürmeragent in der Lage sein muss unter anderem gut im Team und gut gegen einen aktiven Gegner spielen zu können. Selbiges gilt für alle anderen im Spiel genutzten Agenten. Somit kann zwar ein Stürmer alleine trainiert werden, um die Dynamiken des Spiels zu erlernen, diesersetzt jedoch nicht das Trainieren gegen einen aktiven Gegner, der eigene Strategien verfolgen kann. Aus diesem Grund ist für das Trainieren eines Stürmeragenten ein aktiver Gegner erforderlich. Im Falle dieser Arbeit werden hierfür Stürmer- und Torwartagenten miteinander trainiert. Hierfür ist zunächst eine Erweiterung des in Kapitel 5 definierten MDP vorzunehmen. Anschließend wird bestimmt, ob der in Kapitel 6 ausgewählte Lernalgorithmus für das Multi-Agenten Training eingesetzt werden kann und ob Anpassungen an den Hyperparametern vorzunehmen sind. Damit die Agenten miteinander trainiert werden können, ist eine Modellierung der gegnerischen Policies vorzunehmen und eine Methode zur Bestimmung der Trainingspartner zu definieren. Abschließend wird das entworfene Trainingskonzept in einem Versuch ausgeführt und die Resultate evaluiert.

9.1 DEFINITION MULTI-AGENTEN UMGEBUNG

Für eine Problemformulierung mit mehreren Agenten bieten sich Markov-Spiele an [Lit94; Lou22]. Markov-Spiele stellen eine Erweiterung von MDP dar, indem mehrere Agenten in einer Umgebung teilnehmen, mit potenziell unterschiedlichen oder rivalisierenden Zielen. Hierbei wird wie in MDP die Markov-Eigenschaft für Transitionen in der Umgebung vorausgesetzt, jedoch besitzt jeder Agent einen eigenen Aktionsraum. Für das Markov-Spiel bedeutet dies, dass ein gemeinsamer Aktionsraum \mathcal{A} existiert, bestehend aus den Aktionsräumen der einzelnen Agenten, was für n Agenten folgenden gemeinsamen Aktionsraum zur Folge hat: $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_n$.

Dies hat zur Folge, dass die Zustandsübergangsfunktion $\mathcal{P}(S_t, A_t^0, \dots, A_t^n)$ und Rewardfunktion $\mathcal{R}(S_t, A_t^0, \dots, A_t^n, S_{t+1})$ nun von dem jeweiligen Zustand S_t zum Zeitpunkt t , sowie den Aktionen aller n Agenten in der Umgebung zu diesem Zeitpunkt abhängig sind.

Da der erzielte Reward nun abhängig von den Policies aller Agenten in der Umgebung ist, ist das Ziel eines Agenten in einem Markov-Spiel das maximieren des erwarteten Returns unter der gemeinsamen Policy π , welche aus

den Policies aller Agenten in der Umgebung besteht $\pi = (\pi_1, \dots, \pi_n)$.

Daraus folgt, dass im Gegensatz zu einem MDP ein Markov-Spiel keine stationäre, optimale deterministische Policy besitzen muss, sondern eine stationäre optimale Policy hierfür stochastischer Natur sein kann [Lou22]. Ein Markov-Spiel kann jedoch zu einem MDP reduziert werden, indem nur ein Agent in der Umgebung eingesetzt wird, oder die anderen Agenten eine stationäre Policy nutzen, die sich im Verlauf der Zeit nicht verändert. Dies entspricht einer Sichtweise des Agenten, indem die anderen Agenten als Teil der Umgebung wahrgenommen werden.

Da jeder Agent somit seinen eigenen Aktionsraum besitzt, bleibt der Aktionsraum für beide Agenten jeweils ein zweidimensionaler multi-diskreter Aktionsraum mit den Dimensionen [11, 11]. Dennoch wird die Umgebung der Agenten durch hinzufügen einer gegnerischen Stange erweitert, wodurch der Observationsraum von Stürmer- und Torwartagenten um die jeweiligen Informationen bezüglich des gegnerischen Agenten erweitert werden muss. Gleichzeitig ist die in Kapitel 5.3 bestimmte Rewardfunktion zu analysieren und modifizieren, da die bestimmte Rewardfunktion lediglich auf den Stürmeragenten ausgerichtet ist und nun um den Torwartagenten erweitert werden muss. Diese werden in den folgenden Abschnitten behandelt.

9.1.1 *Observationsraum*

Der in Kapitel 7.2 bestimmte Observationsraum beinhaltet lediglich Informationen über den jeweiligen Agenten selbst sowie den Ball. Da nun ein weiterer Agent Teil der Umgebung ist, ist es wichtig Informationen über diesen Akteur mit zu berücksichtigen um eine entsprechende Policy zu erlernen. Dies wird an folgendem vereinfachten Beispiel klar.

Angenommen es wird im Rahmen von Episode τ_x eine Observation $O_t^{\tau_x}$ gemacht, aufgrund dessen der aktuelle Agent die Aktion $A_t^{\tau_x}$ wählt. Diese Observationen beinhalten keine Informationen über die gegnerische Position. Angenommen Aktion $A_t^{\tau_x}$ führt zu einem positiven Reward und verstärkt somit die Wahrscheinlichkeit, diese Aktion bei dieser Observation später zu wiederholen. Am Beispiel vom Kicker könnte der Stürmer durch diese Aktion ein Tor geschossen haben. Da nun aber ein Torwart mit im Spiel ist, könnte die Aktion erfolgreich gewesen sein, weil der Torwart nicht im Weg zwischen Ball und Tor stand. Angenommen es kommt in einer späteren Episode τ_{x+1} zu einer Observation $O_t^{\tau_x+1}$, welche inhaltlich identisch zu $O_t^{\tau_x}$ ist. Naheliegend wäre es nun, die gleiche Aktion wie in Episode τ_x zu wählen, da diese zu einem positiven Reward geführt hat. Stünde nun der Torwart jedoch an einer Stelle, die ein Tor verhindert, so würde diese Aktion in dieser Episode keinen positiven Reward zur Folge haben.

Dies kann zur Folge haben, dass das Training länger dauert aufgrund der widersprüchlichen Rewardsignale. Der gegnerische Spieler ist nicht Teil des Observationsraums und somit kann bei der Auswahl der Aktion als Folge

auf eine Observation nicht explizit Rücksicht auf die exakte Gegnerposition genommen werden.

Um dem entgegenzuwirken und das Training einfacher zu gestalten, werden Informationen zum gegnerischen Agenten zum Observationsraum hinzugefügt. Hierfür werden die Observationskanäle gewählt die Stangeninformationen beinhalten. In Kapitel 7 konnte ein Agent mit diesen Informationen eine Policy erlernen um Tore schießen zu können, durch Navigation der Stürmerstange. Gleichzeitig konnte in Abbildung 7.5 gesehen werden, dass die Leistung des Agenten stark von den einzelnen Observationskanälen bezüglich der eigenen Stangenposition und Geschwindigkeit abhängig ist. Aufgrund dessen wird der jeweilige Observationsraum von Stürmer- und Torwartagent um folgende Informationen der gegnerischen Stange erweitert:

- Lateralposition der Stange
- Winkelposition der Stange
- Laterale Bewegungsgeschwindigkeit
- Rotationsgeschwindigkeit
- Spielerposition der Stange

9.1.2 Rewardfunktion

Zum Bestimmen der Rewardfunktion der einzelnen Agenten sind die Ziele dieser zu betrachten. Das Ziel eines Torwarts ist das Verhindern von Toren, während das Ziel eines Stürmers das Schießen von Toren ist. Aufgrund dieser streng gegensätzlichen Ziele eignet sich die Definition als Nullsummen-Markov-Spiel. Ein Nullsummen-Markov-Spiel ist ein Markov-Spiel, in dem der Reward eines Agenten der negative Reward eines anderen in der Umgebung befindlichen Agenten darstellt. Somit gilt in einem Nullsummen-Markov-Spiel mit zwei Agenten und deren jeweiligen Rewardfunktionen \mathcal{R}^1 und \mathcal{R}^2 folgendes: $\mathcal{R}^1(S_t, A_t^1, A_t^2, S_{t+1}) = -\mathcal{R}^2(S_t, A_t^1, A_t^2, S_{t+1})$. [Lau, S. 13] Alternativ hierzu ist die Modellierung mittels Nicht-Nullsummen-Markov-Spiel ebenfalls möglich, indem beispielsweise das Schießen von Toren einen höheren Reward für den Stürmer bedeuten würde, als wenn der Ball hinter die Stürmerstange gelangt. Nicht-Nullsummenspiele erlauben somit ebenfalls Win-Win-Möglichkeiten, wodurch es nicht zu einem strengen Wettbewerb zwischen den Agenten kommen muss.

Im Rahmen dieser Arbeit eignet sich jedoch die Formulierung als Nullsummenspiel, da hierdurch ein starker Wettbewerb zwischen Stürmer und Torwart gefordert wird. Hierdurch müssen beide Agenten zur Maximierung ihres Returns sowohl das Schießen als auch das Verteidigen von Schüssen lernen. Weiterhin haben Nullsummenspiele aufgrund des Minimax-Theorems [Neu28] garantiert mindestens ein Nash-Gleichgewicht, was als Lösung für ein zwei Spieler Nullsummenspiel gesehen werden kann [Nas51]. Ein Nash-Gleichgewicht stellt eine Wahl von Strategien der jeweiligen Akteure dar,

wobei kein Akteur durch einseitiges Abweichen von der eigenen Strategie einen Vorteil erhalten kann.

Zur Realisierung als Nullsummenspiel stellt das Schießen eines Tores einen positiven Reward für den Stürmer und ein negativer Reward für den Torwart dar. Jedoch ist zu beachten, dass das Schießen eines Tors ein Terminalzustand des zugrunde liegenden MDP darstellt, während das Halten des Balls als Ziel des Torwarts eine andauernde Aufgabe darstellt. Im Rahmen dieses Versuches wird somit das Schießen des Balles hinter die Stürmerstrategie als positives Ziel für den Torwart definiert, da dies einer neutralisierten Torgefahr in diesem Szenario am nächsten kommt.

In Kapitel 5.3.1 wurde weiterhin die Nutzung von PBRS bestimmt, um die Rewardfunktion zu verdichten und das Erlernen einer guten Policy für den Stürmeragenten zu vereinfachen. Diese formende Rewardfunktion kann jedoch nicht ohne weiteres auf das Nullsummen-Markov-Spiel zwischen Torwart und Stürmer übertragen werden. Grund hierfür ist, dass die Potenziale der Ballposition von Stürmer und Torwart grundsätzlich anders bewertet werden.

Während für den Stürmer die Z-Position des Balls von Bedeutung ist, da das Tor nur eine begrenzte Breite hat, ist dies für den Torwart nicht relevant. Da es das Ziel des Torwarts ist, den Ball hinter den Stürmeragenten zu bekommen, spielt die Z-Position für die Bewertung des Ballpotenzials aus Sicht des Torwarts keine Rolle, sondern lediglich die X-Position. Eine Potenzialfunktion aus Sicht des Torwarts könnte so aussehen, das lediglich die X-Koordinate des Balles hierfür berücksichtigt wird:

$$\psi^{\text{Torwart}}(O_t) = 0,5X_{O_t} + 0,5 \quad (9.1)$$

wobei X_{O_t} die X-Position des Balls aus der Observation O_t zum Zeitpunkt t extrahiert.

Nutzt man jedoch für den Torwartagenten diese Potenzialfunktion 9.1 und für den Stürmeragenten die in Kapitel 5.3.1 definierte Potenzialfunktion 5.5, so ist nicht mehr gegeben, dass sich die jeweiligen Rewards zu jedem Zeitpunkt ausgleichen. Damit wäre es kein Nullsummenspiel mehr, da es Zustände in der Umgebung geben würde, in denen der Torwartagent einen größeren negativen Reward haben könnte, als der Stürmeragent positiven Reward erhalten würde. Dies wäre der Fall, wenn der Ball in einer Spielfeldecke bei dem Torwart ist. In diesem Fall wäre das Potenzial des Balls sehr gering für den Torwart, da hierfür nur die X-Koordinate eine Rolle spielt. Aus Sicht des Stürmers wäre das Potenzial aber nicht so hoch wie es im Vergleich zum Torwart sein müsste, da sich der Ball sehr weit seitlich befindet und für den Stürmer X- und Z-Position eine Rolle spielen.

Um PBRS weiterhin nutzen zu können und das Spiel als Nullsummenspiel-Markov-Spiel zu definieren, wird eine Potenzialfunktion basierend auf der

Torwart- und Stürmer-Potenzialfunktion von beiden Agenten genutzt. Daraus folgt folgende Potenzialfunktion ψ'^{Torwart} für den Torwart:

$$\psi'^{\text{Torwart}}(O_t) = (\psi^{\text{Torwart}}(O_t) - \psi^{\text{Stürmer}}(O_t))/2 \quad (9.2)$$

und folgende Potenzialfunktion $\psi'^{\text{Stürmer}}$ für den Stürmer:

$$\psi'^{\text{Stürmer}}(O_t) = (\psi^{\text{Stürmer}}(O_t) - \psi^{\text{Torwart}}(O_t))/2 \quad (9.3)$$

wobei Gleichung 9.1 als ψ^{Torwart} und Gleichung 5.5 als $\psi^{\text{Stürmer}}$ gilt. Indem die beiden Ballpotenziale miteinander verrechnet werden wird sichergestellt, dass es sich weiterhin um eine Nullsummenspiel handelt und es keinen Zustand gibt der unterschiedlich stark von beiden Agenten bewertet wird. Hierbei werden diese Potenzialfunktionen ψ' beider Agenten durch 2 geteilt, um die jeweiligen Werte im genormten Wertebereich von $[-1, 1]$ abbilden zu können.

9.2 LERNALGORITHMUS

Bezüglich des Lernalgorithmus ist zunächst festzulegen, ob der in Kapitel 6 definierte Lernalgorithmus für das Multi-Agenten Training genutzt werden kann. In Kapitel 6 wurde bereits erläutert, dass sich PPO als Algorithmus für nicht-stationäre Umgebungen wie beispielsweise Markov-Spiele eignet, da dieser eine stochastische Policy lernt. PPO ist ein Actor-Critic On-Policy Algorithmus, der somit die eingesetzte Policy in der Umgebung optimiert. Da hierdurch nur aktuelle Episoden aus der Umgebung zum Lernen genutzt werden kann somit auf geänderte gegnerische Policies direkt optimieren kann. Im Vergleich hierzu Nutzen Off-Policy Methoden oft einen Replay-Buffer [Sch+16] zum Speichern gesehener Transitionen in der Umgebung [Mni+13; Haa+18]. Dies ist unter anderem ein Faktor für ihre größere Probeneffizienz. Durch Nutzung eines Replay-Buffer werden Transitionen zum Lernen jedoch nicht direkt aus der Umgebung genutzt. Dies kann dazu führen, dass es lange dauert bis auf eine veränderte gegnerische Policy reagiert werden kann, da hierfür genügend Rollouts zu sammeln sind um für die Optimierung mehrheitlich diese neuen Samples zu nutzen, statt alter weniger relevanter Rolloutdaten.

Dennoch konnte PPO in unterschiedlichen Markov-Spielen unterschiedlicher Komplexität Erfolge erzielen. So konnte in [Ope+19a] die Weltmeister im Echtzeit-Strategiespiel *Dota 2* besiegt werden. Weiterhin konnten komplexe Verhalten wie das Nutzen von Werkzeugen oder das Bilden eines Unterschlupfs in einem *Hide-and-Seek* Spiel mit mehreren kooperativen und gegnerischen Agenten durch PPO erlernt werden [Bak+19]. In einer weiteren Arbeit [Yu+21] wurde PPO auch in unterschiedlichen anspruchsvollen Multi-Agenten Umgebungen eingesetzt mit überraschend starken Ergebnissen und einer ähnlichen Probeneffizienz im Vergleich zu Off-Policy Referenzalgorithmen. Weiterhin zeigen die Autoren Gründe für die starke Leistung

von Multi-Agent Proximal Policy Optimization ([MAPPO](#)) (im folgenden weiterhin [PPO](#) genannt) in Multi-Agenten Umgebungen auf.

Ein Grund für die starke Leistung von [PPO](#) ist die Nutzung von Normalisierung für die Wertefunktion. Diese ist bereits als wichtiger Faktor in Kapitel [6.2](#) benannt und für die erfolgten Experimente implementiert worden.

Ein weiterer Grund ist die Nutzung von globalen Informationen für das Lernen der Wertefunktion. Da die Wertefunktion lediglich beim Trainieren benötigt wird schlagen die Autoren vor, dieser weitere ggf. nicht zugängliche Informationen mitzuteilen, damit die Dynamiken der Umgebung besser eingeschätzt werden können. Im Rahmen dieser Arbeit sind jedoch bereits sowohl der Ball als auch alle beteiligten Stangen Teil des Observationsraums.

[PPO](#) sammelt Rollouts in der Umgebung und teilt die gesammelten Daten in Minibatches auf, welche für die Optimierung der Policy genutzt werden. Diese Minibatches werden mehrmals wiederverwendet, wobei die Anzahl an Wiederverwendungen durch den Hyperparameter *Epochs* definiert wird. Die Autoren [[Yu+21](#)] konnten jedoch in ihren Experimenten beobachten, dass das Teilen eines Rollouts in Minibatches zu schlechterem Lernerfolg führt. Weiterhin empfehlen die Autoren die Nutzung von 15 Epochs für einfache und 5 – 10 Epochs für schwere Aufgaben, da hierdurch bessere Ergebnisse erzielt werden konnten.

Als letzter für diese Arbeit relevanter Grund wird die Anpassung des Clip-Grenzwertes genannt. Während kleinere Clip-Grenzwerte das Lernen verlangsamen, führen diese jedoch zu stabileren Optimierungen der Policy. Dieser Wert sollte somit durch Abwägung zwischen Stabilität und Trainingseffizienz gewählt werden. In ihrer Arbeit wurden die Clip-Grenzwerte [0,05; 0,1; 0,15; 0,2; 0,3; 0,5] getestet, wobei der Wert 0,15 in 6 von 8 Versuchen überzeugen konnte.

Durch die erzielte Leistung von [PPO](#) in unterschiedlichen Multi-Agenten Umgebungen eignet sich dieser Algorithmus somit zur Nutzung in diesem Versuch. Weiterhin werden die in [[Yu+21](#)] evaluierten Hyperparameter für [PPO](#) angepasst. Dies betrifft die Nutzung von Minibatches, die Anzahl an Epochs sowie die Größe des Clip-Grenzwertes. Auf die Nutzung von Minibatches wird somit in diesem Versuch verzichtet. Als Anzahl der Epochs wird der von den Autoren empfohlene Wert 10 gewählt, da dieser zwischen den empfohlenen 15 für einfache Aufgaben und den empfohlenen 5 für schwere Aufgaben liegt. Zuletzt wird der Clip-Grenzwert auf den Wert 0,15 gesetzt, da dieser in unterschiedlichen Umgebungen konstant gute Leistungen erzielen konnte. Somit werden folgende Hyperparameter in diesem Versuch genutzt:

- Lernrate: 3e–4
- Rollout Länge: 2048
- Minibatch Größe: 2048
Dieser Wert entspricht dem weglassen von Minibatches.
- Anzahl Epochs: 10

- Diskontierungsfaktor: 0,99
- GAE_lambda: 0,95
- Clip-Parameter: 0,15
- Clip-Parameter für Wertefunktion: 0,15
- max grad norm: 0,5
- Entropie Koeffizient: 0,01
- Wertefunktion Koeffizient: 0,5
- Advantage Normalisierung: Wahr
- NN Architektur: [128, 128] mit *Tanh* Aktivierungsfunktion
- Orthogonale Initialisierung: Wahr

9.2.1 Zentralisierter und Dezentralisierter Critic

Da **PPO** ein Actor-Critic Algorithmus ist nutzt dieser ein Actor- und ein Critic-Netzwerk. Wie in den Grundlagen bereits beschrieben ist das Actor-Netzwerk dafür zuständig, in der Umgebung zu agieren, während das Critic-Netzwerk die Wertefunktion approximiert, um dem Actor eine Beurteilung der gewählten Aktionen zu geben. Das Critic-Netzwerk wird nur während des Trainings von **PPO** genutzt, um das erlernen einer guten Policy zu erleichtern. Dieser Umstand wurde in [Yu+21] wie bereits erläutert genutzt, um den Critic-Netzwerk mehr Informationen bereitzustellen, als der Agent in der anschließenden Evaluierungsumgebung zur Verfügung hat. Hierdurch soll vom Critic-Netzwerk die wahre Wertefunktion der Umgebung besser approximiert werden, wodurch bessere Policies während des Trainings vom Actor-Netzwerk erlernt werden sollen.

Einige Arbeiten nutzten dies in Multi-Agenten Umgebungen für die Implementierung eines zentralisierten Critic-Netzwerks, das von allen eingesetzten Agenten zur Beurteilung der gewählten Aktionen genutzt wird. Dieser Ansatz wird als Centralized Training for Decentralized Execution (**CTDE**) bezeichnet [Yu+21]. Hierbei wird eine Actor-Critic Architektur eingesetzt in der ein zentralisierter Critic trainiert wird. Grund für die Nutzung ist die Annahme, dass durch die Nutzung der Daten mehrerer Agenten ein besserer Critic trainiert werden kann, dessen Beurteilungen somit auch die einzelnen Actors zu besseren Policies helfen.

Der Einfluss von zentralisierten Critics wurde in [Lyu+21] untersucht. Die Autoren nutzten drei Methoden, Joint-Actor-Critic (JAC), Independent Actor Critic (IAC) und Independent Actor Centralized Critic (IACC) und testeten deren jeweiligen Einfluss in verschiedenen Umgebungen. JAC nutzt hierfür ein zentralisiertes Critic- und zentralisiertes Actor-Netzwerk für alle eingesetzten Agenten. IAC nutzt dezentralisierte Critic- und dezentralisierte

Actor-Netzwerke und IACC nutzt ein zentralisiertes Critic- und dezentralisierte Actor-Netzwerke für alle Agenten.

Die Autoren analysieren die theoretischen Implikationen von zentralisierten und dezentralisierten Critics und kommen zum Schluss, dass zentralisierte und dezentralisierte Critics äquivalent in Bezug auf die Anpassung dezentralisierter Actors sind. Weiterhin führen in der Theorie zentralisierte Critics zu einer höheren Varianz der Policyanpassungen. Ihre theoretischen Ergebnisse haben die Autoren in unterschiedlichen Umgebungen empirisch evaluiert und kamen hierbei zum Schluss, dass in der Praxis die Wahl zwischen zentralisierten und dezentralisierten Critics einer Bias-Varianz Abwägung gleich kommt, da dezentralisierte Critics durch Nutzung weniger Daten einen höheren Bias aufweisen und weniger korrekte Wertefunktion lernen als zentralisierte Critics. Gleichzeitig konnte beobachtet werden, dass dezentralisierte Critics in den genutzten Umgebungen durchweg robustere Leistungen erzielen konnten als zentralisierte Critics, da stabilere Anpassungen der Netzwerke in diesen Umgebungen wichtiger zu sein scheinen als eine korrektere Einschätzung der Wertefunktion. Als Umgebungen wurden von den Autoren sowohl klassische Matrix-Spiele, aber auch komplexe Echtzeit-Strategiespiele wie die *StarCraft Multi-Agent Challenge* [Sam+19], *Particle Environments* [MA17] sowie eine Kombination von zweidimensionalen Multi-Agenten Spielen [JA21], welches auch eine Multi-Agenten Fußballumgebung beinhaltet.

Wie in ihrer Arbeit beschrieben, führt die Nutzung eines zentralisierten Critics zu einer höheren Varianz während des Trainings. Eine potenziell hohe Varianz wird in unterschiedlichen Arbeiten durch Nutzung von großen verteilten Trainingssystemen kompensiert, um durch lange Rollouts, große Minibatches und kleinere Epochs mit diesen Batches dem Agenten kontinuierlich eine bessere Policy in kleinen Schritten lernen zu lassen. Beispiele hierfür stellen [Ope+19a] mit einer effektiven Batch-Größe von ca. 3 Mio. Zeitschritten und [Ban+17] mit einer Rollout-Länge von 409.600 Zeitschritten dar. Da im Rahmen dieser Arbeit die Rechenressourcen begrenzt sind und aufgrund der Vielfalt der von den Autoren genutzten Umgebungen sowie den durchweg von IAC erzielten positiven Ergebnisse in diesen, werden im Rahmen dieses Versuches dezentralisierte Actor-Critic Architekturen für Torwart- und Stürmeragenten genutzt.

9.3 MODELLIERUNG VON GEGNERN

Indem mehrere Agenten in einer Umgebung trainieren kommt es zu Änderungen beider trainierten Policies. Dieses Training kann so gestaltet werden, dass beide Agenten gleichzeitig das Training miteinander starten und bis zum Ende der bestimmten Trainingsdauer durchführen. Eine weitere Möglichkeit hierfür ist, dass die Trainingspartner mit der Zeit wechseln und die jeweiligen Agenten mit unterschiedlichen Trainingspartner trainiert werden. In einer Arbeit [Ban+17] wurde hierzu geprüft, wie das Training mit nur einem Trainingspartner im Vergleich zum Training mit mehreren Trainings-

partnern abschneidet. In dem Versuch mit mehreren Trainingspartnern wurden hierfür unterschiedliche Zwischenstände der jeweils trainierten Agenten genommen. Somit bestand das Trainingsensemble aus den jeweiligen aktuellsten Agenten und älteren Zwischenständen dieser. Das Training mit nur einem Trainingspartner entsprach somit dem Trainieren mit dem immer jeweils aktuellsten Agenten.

Ihre Resultate zeigten, dass beim Trainieren mit nur dem aktuellsten Trainingspartner dies zu einem Ungleichgewicht beim Training führte. Indem ein Agent früh im Training besser als der gegnerische Agent wird führte dies dazu, dass der gegnerische Agent sich hiervon nicht mehr erholen konnte und es bei einem großen Ungleichgewicht beim Training blieb. Im Gegensatz hierzu konnte durch Nutzung von verschiedenen Trainingspartnern unterschiedlicher Stärke ein Gleichgewicht erzielt werden, in dem beide Agenten sich in ihrem Verhalten verbessern konnten. Die einzelnen Trainingspartner wurden zufällig aus den erstellten Zwischenständen der jeweils gegnerischen Agenten gewählt. Dies hat zu stabilerem Training und zum Erlernen robusterer Policies durch beide Agenten geführt. [Ban+17]

Dabei muss der gegnerische Agent nicht eine von Grund auf neu trainierte Policy sein. Die Arbeiten [Sil+17a; Ope+19a] nutzten hierfür die Methode *Self-Play*. Self-Play stellt eine Trainingsmethode dar, in der ein Agent gegen sich selbst spielt. Ein Agent trainiert somit mit sich selbst als Trainingspartner. Diese Methode wurde auch in [Alp] genutzt, um erfolgreich das Spiel Go zu erlernen und den Europameister in Go im Jahr 2015 zu besiegen. Aber auch in Nachfolgerarbeiten wie [Sil+17a] konnte so ein Agent trainiert werden, der mehrere Brettspiele auf Großmeister-Niveau spielen konnte, ohne dabei je ein von Menschen gespieltes Spiel zu sehen. Ein Unterschied zwischen diesen beiden Arbeiten ist jedoch, dass in [Alp] immer gegen den aktuellsten und in [Sil+17a] immer gegen den historisch besten Agenten gespielt wurde. Auch in Echtzeitstrategiespielen wurde diese Methode erfolgreich genutzt, wie [Ope+19a] zeigt.

Im Falle von Self-Play führt eine weitere Arbeit [Ban+17] zu dem Ergebnis, dass durch zufällige Auswahl eines gegnerischen Agenten aus einem Ensemble von unterschiedlichen Zwischenständen, die aktuellste Policy gezwungen wurde beliebige alte Zwischenstände der eigenen Policy besiegen zu können. Durch diese Methode konnte durch die Autoren ein kontinuierliches Lernen gewährleistet werden.

Im Rahmen dieser Arbeit stellt sich nun die Frage, ob derselbe Agent für Stürmer- und Torwartstange zu nutzen ist oder unterschiedliche Agenten genutzt werden sollten. Da die Nutzung von nur dem aktuellsten Trainingspartner zu instabilem Training führen kann, wird ein Ensemble aus Trainingspartnern genutzt. Hierfür ist zu bestimmen, aus welchen und wie vielen Agenten dieses Ensemble bestehen soll und wie ein Trainingspartner hieraus zu bestimmen ist.

9.3.1 Gegnerische Policies

Es ist zu bestimmen, ob ein Agent für Stürmer- und Torwartstange trainiert werden sollte oder jeweils ein Agent pro Stangenart. Die Nutzung von Self-Play hat Vorteile, da hierfür nur ein Agent zu trainieren ist statt in diesem Fall zwei unterschiedliche Agenten. Dieser Agent könnte bei gleicher Trainingsdauer und gleichen Rechenressourcen die doppelte Erfahrung in der Umgebung sammeln, da sowohl die eigenen Erfahrungen als auch die gegnerischen zum Lernen genutzt werden können. In den für Self-Play genutzten Umgebungen [Alp; Sil+17b; Sil+17a; Ope+19a] handelt es sich jedoch um symmetrische Spiele. Dies bedeutet, dass beide Agenten in der Umgebung die gleichen Eigenschaften besitzen. Es spielt somit keine bedeutende Rolle, auf welcher Seite ein Agent in der jeweiligen Umgebung spielt, da sich die Ausgangslagen beider Seiten nicht relevant unterscheiden. In dieser Arbeit haben Stürmer- und Torwartstangen jedoch unterschiedliche Eigenschaften. Zum einen haben beide Stangen unterschiedliche Bewegungseinschränkungen, wie in Kapitel 4.1.1 erläutert. Weiterhin haben beide Stangen eine unterschiedliche Anzahl von Spielern pro Stange, wobei die Stürmerstange drei und die Torwartstange eine Figur besitzen. In der Nähe der Torwartstange sind die Ecken des Spielfelds angehoben und es befindet sich eine Wand hinter der Stange, von der der Ball abprallen kann. Dies steht ebenfalls im Gegensatz zur Stürmerstange.

Betrachtet man die Ziele beider Agenten lassen sich ebenfalls Unterschiede feststellen. Während der Torwart den Ball hinter die Stürmerstange schießen soll, ist die Aufgabe des Stürmers Tore zu schießen. Zwar können die Ziele von Stürmer und Torwart so abstrahiert werden, dass diese sich zusammenfassen lassen. Dies kann so formuliert werden, dass der jeweilige Agent den Ball so weit wie möglich in die Blickrichtung des Agenten bewegen soll. Doch dies ändert nicht, dass mit dem aktuellen Observationsraum die Torwände und die angehobenen Spielfeldecken nicht berücksichtigt werden können. Gleichzeitig ist davon auszugehen, dass ein Agent der gleichzeitig als Stürmer und Torwart Bälle halten und schießen können muss ein schwereres Verhalten zu lernen hat, als beispielsweise nur das Schießen und Halten als Stürmer.

Somit eignet sich aufgrund der Asymmetrie des Spiels die Nutzung unterschiedlicher Agenten für jeweils Stürmer und Torwart. Hierdurch können spezifische Eigenschaften der jeweiligen Position als statisch wahrgenommen werden, wie beispielsweise das Vorhandensein von Wänden. Zwar wird hierdurch nicht ein Agent trainiert der mehrere Positionen spielen kann, dennoch ist davon auszugehen, dass hierdurch das zu erlernende Verhalten für Stürmer und Torwart einfacher ist und somit die Trainingseffizienz gesteigert wird. Diese Methode wurde auch in [Ban+17] für asymmetrische Spiele wie *kick-and-defend* oder *you-shall-not-pass* genutzt.

9.3.2 Methode zur Gegnerauswahl

Unterschiedliche Arbeiten haben unterschiedliche Ansätze zur Wahl von Trainingspartnern gewählt. In einer Arbeit [ZZP20] werden diese unterschiedlichen Ansätze evaluiert und ein Framework zur Auswahl von Trainingspartnern entwickelt. Sie zeigen durch Versuche in unterschiedlichen Umgebungen, dass ihr Ansatz bessere Ergebnisse als die Nutzung des aktuellsten, des historisch besten oder eines zufälligen historischen Agenten erzielt. Ihr vorgestellter Algorithmus 3 zur Bestimmung des Trainingspartners ist unten abgebildet.

Algorithm 3 Störungsbasierte Self-Play Policy-Optimierung eines Ensemble von n Agenten

Eingabe: N : Anzahl von Iterationen, n : Ensemblegröße, l : Anzahl an Schritten in der Umgebung;

Ausgabe: n Agenten-Paare

```

1: for  $k = 0, 1, 2, \dots, N - 1$  do
2:   Werte  $\hat{f}(x_i^k, y_j^k), \forall i, j \in 1 \dots n$  mit Gleichung 9.4
3:   for  $i = 1, \dots, n$  do
4:     Erstelle potenzielle Kandidatenlisten sodass:
5:      $C_{y_i}^k = \{y_j^k : j = 1 \dots n\}$  und  $C_{x_i}^k = \{x_j^k : j = 1 \dots n\}$ 
6:     Bestimme  $v_i^k = \arg \max_{y \in C_{y_i}^k} \hat{f}(x_i^k, y)$ 
7:     Bestimme  $u_i^k = \arg \min_{x \in C_{x_i}^k} \hat{f}(x, y_i^k)$ 
8:     Trainieren von  $x_i^k$  mit  $v_i^k$  als stationärem Gegner für  $l$  Schritte
9:     Trainieren von  $y_i^k$  mit  $u_i^k$  als stationärem Gegner für  $l$  Schritte
10:    end for
11:  end for
```

Eigene Darstellung basierend auf [ZZP20, S. 4]

Der Algorithmus der Autoren trainiert eine Anzahl von n Agenten im Ensemble. Hierbei finden für N Iterationen Trainings der einzelnen Agenten statt, die jeweils eine Dauer von l Schritten haben. Zur Bestimmung des geeigneten Gegners für den zu trainierenden Agenten, findet zu jeder Iteration k eine Evaluierung basierend auf folgender Gleichung statt [ZZP20]:

$$\hat{f}(x, y) = \frac{1}{m} \sum_{i=1}^m \mathcal{R}(\tau_i) \quad (9.4)$$

wobei jeder Agent gegen alle Ensemblegegner über m Episoden evaluiert wird. Nach stattfinden der Evaluierung werden Kandidatenlisten in Zeile 4 und 5 erstellt, aus denen die Gegner für die als nächstes zu trainierenden Agenten x_i^k und y_i^k bestimmt werden. Hierbei werden jeweils die Agenten v_i^k und u_i^k als Gegner bestimmt, die bei zuvor gegangener Evaluierung das beste Ergebnis über m Episoden gegen die zu trainierenden Agenten x_i^k und y_i^k erzielt haben. In Zeile 8 und 9 findet dann das Training von x_i^k und y_i^k

statt, wobei die gegnerischen Policies v_i^k und u_i^k als stationäre Policies in die Umgebung eingefügt werden. Somit stellen v_i^k und u_i^k Kopien der Agenten von Iteration $k - 1$ dar, welche als Trainingspartner für die nächste Iteration genutzt und nach Abschluss des Trainings verworfen werden.

Dieser Algorithmus wird im Rahmen dieser Arbeit als Methode zur Gegnerauswahl gewählt, da hierdurch bessere Ergebnisse erzielt werden konnten [ZZP20] im Vergleich zu den genutzten Methoden in [Alp; Sil+17b; Sil+17a; Ope+19a; Ban+17].

Für die Nutzung dieses Algorithmus für das Training von Stürmer- und Torwartagenten sind somit vier Parameter zu bestimmen: Die Anzahl an Iterationen N , die Größe des Agentenensembles n , die Trainingsdauer pro Agent und Iteration l sowie die Anzahl an Episoden für die Evaluation der Agentenstärken m .

Wie in den Versuchen zuvor gesehen werden konnte, war ein Stürmer in der Lage eine gute Policy nach ca. 3 Mio. Schritten zu erlernen. Hierbei hatte der Stürmer jedoch keinen gegnerischen Torwartagenten, der versucht das Schießen von Toren aktiv zu verhindern. Da im Rahmen dieser Arbeit nur begrenzte Rechenressourcen zur Verfügung stehen wird die Anzahl an Iterationen N gering gehalten, während das Trainieren gegen einen Gegner länger durchgeführt wird. Hierdurch soll die Stationarität der Umgebung gering gehalten werden, was zu einer stabileren Entwicklung des Lernverhaltens beitragen soll. Weiterhin konnte in den Versuchen in Kapitel 8.2 gesehen werden, dass sich eine Konvergenz des Stürmeragenten nach ca. 3 Mio. Schritten abzeichnet hat, bzw. nach 1200 Rollouts (ca. 2,5 Mio. Schritten) eine sehr gute Policy erlernt werden konnte. Aufgrund dieser Werte wird die Trainingsdauer pro Agent und Iteration l auf 2,4576 Mio. Schritte festgelegt, was genau 1200 Rollouts entspricht. Die Anzahl zur Evaluierung der jeweiligen Agentenstärken m wird auf 100 festgelegt.

Eine Trainingsdauer von 1200 Rollouts entspricht ca. 12 Stunden auf den zur Verfügung stehenden Ressourcen. Gleichzeitig muss jeder Agent des Ensembles sequenziell trainiert werden, was die Trainingsdauer enorm erhöht. Aufgrund der Änderungen an der Agentenumgebung, der Umstellung von *zeroMQ* auf die Nutzung von *gRPC* und die dadurch gewonnenen Reduktion der benötigten Rechenressourcen, können das Training zweier Agenten in einer Iteration i zeitgleich in zwei Simulationen stattfinden. Hierdurch wird die benötigte Trainingsdauer halbiert. Auch die Autoren nutzten in ihrer Arbeit konkurrierendes Lernen beider Agenten in einer Iteration, um die Trainingsdauer zu reduzieren [ZZP20].

Dennoch ist für die Festlegung der Anzahl an Iterationen N zunächst die Ensemblegröße zu definieren. In ihrer Arbeit konnten die Autoren positive Ergebnisse bereits mit einer Ensemblegröße von 2 erreichen. Dies würde für diese Arbeit jeweils zwei Torwart- und zwei Stürmeragenten entsprechen. Aufgrund der begrenzten Rechenressourcen wird somit eine Ensemblegröße n von 2 gewählt und die Anzahl von Iterationen N auf 2 festgelegt. Dies ent-

spricht für alle vier Agenten jeweils eine Gesamttrainingsdauer von 4,9152 Mio. Schritten. Durch die Trainingsdauer von 1200 Rollouts pro Gegner soll gewährleistet werden, dass die Agenten genügend Zeit haben, um sich bezüglich der wechselnden Gegner anzupassen zu können. Gleichzeitig lassen sich bei 2 Iterationen ebenfalls die Auswirkung eines Gegnerwechsels erkennen.

Für den Versuch wird jede Policy mit einem eigenen Seed für die Zufallsgeneratoren initialisiert, ebenso wie die zwei parallel genutzten Simulationen.

9.4 VORGEHENSWEISE UND VERSUCHSAUFBAU

In diesem Versuch werden Stürmer- und Torwartagent miteinander trainiert. Hierfür wird der in Kapitel 9.1.1 bestimmte Observationsraum, der in Kapitel 8.2 bestimmte Aktionsraum sowie die in Kapitel 9.1.2 definierte Rewardfunktion genutzt. Es wird der Lernalgorithmus PPO mit jeweils einer Policy für Stürmer- und Torwartstange und den in Kapitel 9.2 definierten Hyperparametern genutzt. Zur Bestimmung des Trainingspartners wird der in Kapitel 9.3.2 erläuterte Algorithmus mit den dort definierten Parametern genutzt. Eine Evaluierung findet durch die in Algorithmus 3 vorgenommenen Evaluierungen statt, womit zuletzt eine Episodendefinition zu bestimmen ist.

9.4.1 Episodendefinition

Bei Bestimmung der Episodendefinition ist darauf zu achten, dass die Schwierigkeitsgrade beider Agenten möglichst ähnlich sind, damit es nicht zu einem instabilen Training mit früher Dominanz eines Agenten kommt. Gleichzeitig ist darauf zu achten, dass wie in den Versuchen zuvor die Initialzustandsverteilung möglichst schmal gewählt wird um die Trainingseffizienz zu erhöhen.

Für die Stürmerstange wurde der Ball hierfür vor den einzelnen Spielern in einer geraden Linie im X-Abstand von 4 cm zur Stange platziert, wobei der Ball in der Z-Koordinate um ± 4 cm im Vergleich zur Mitte der Stangenfigur platziert werden konnte. Dieses Intervall wird auch in diesem Versuch für den Stürmer gewählt, da hierfür eine gute Policy erlernt werden konnte, die Umgebung aber schwer genug war um nicht sofort zu konvergieren. Dies ist gerade in diesem Versuch wichtig, um den Schwierigkeitsgrad der Umgebung für Torwart- und Stürmerstange auf dem selben Niveau zu halten. Nun ist die Initialzustandsverteilung für die Torwartstange zu bestimmen. Hierbei ist zu berücksichtigen, dass die Torwartstange ein deutlich kleineres Tor zu verteidigen hat als die Stürmerstange. Gleichzeitig hat dies zur Folge, dass das Ziel des Torwarts leichter zu erreichen ist. Deshalb ist davon auszugehen, dass hierdurch der Torwart eine deutlich einfachere Aufgabe zu erlernen hat als der Stürmer. Um dies zu kompensieren wird für den Torwart eine breitere Initialzustandsverteilung bestimmt. Hierdurch soll verhindert werden, dass der Torwartagent schnell in eine dominierende Position gegenüber dem Stürmeragenten kommt. Als Initialzustandsverteilung wird

hier der Ball ebenfalls vor der Torwartfigur im X-Abstand von 4 cm platziert, die Varianz der Z-Position des Balles jedoch erhöht. Hierfür wird der Ball im Abstand zur Mitte der Torwartfigur um ± 10 cm platziert. Dies entspricht der gesamten Breite des Tores. Die Platzierung vor Stürmer- und Torwartstange findet abwechselnd zwischen den Episoden statt.

Bis auf diese Änderungen wird die gleiche Episodendefinition wie in Kapitel 7.1.4 und 8.1 genutzt, d.h. die Nutzung eines zeitlichen Limits und das Beenden einer Episode wenn der Ball für keine Stange erreichbar ist und zum Stehen kommt.

9.4.1.1 *Domain Randomization*

Wie bereits in den Versuchen in Kapitel 7 und 8 gesehen, bedarf das Trainieren eines Stürmeragenten eine lange Zeit. Selbst mit der erhöhten Trainingsdauer von 4,32 Mio. Zeitschritten in Versuch 8 konnte lediglich ein Ansatz einer Konvergenz gesehen werden. Da im Rahmen dieser Arbeit nur begrenzte Rechenressourcen zur Verfügung stehen wird somit in diesem Versuch auf die Nutzung von Domain Randomization verzichtet, da davon auszugehen ist, dass dies die Umgebung schwerer zu erlernen macht, wie bereits in den Grundlagen erläutert.

9.5 ERGEBNISSE UND AUSWERTUNG

Zunächst wird das Lernverhalten anhand der erzielten Rewardkurven der Stürmer- und Torwartagenten miteinander verglichen. Hieraus sollen Rückschlüsse über den Trainingsverlauf gezogen werden. Anschließend werden die Ergebnisse der Evaluationen aus Algorithmus 3 dargestellt. Zuletzt werden während des Trainings beobachtbare Strategien der Agenten aufgezeigt.

9.5.1 *Vergleich der Rewardkurven*

Da die Ensemblegröße 2 betrug wurden insgesamt jeweils zwei Torwart- und zwei Stürmeragenten trainiert. In Abbildung 9.1 werden die Rewardkurven für das erste Training von Stürmer- und Torwartagenten aufgezeigt.

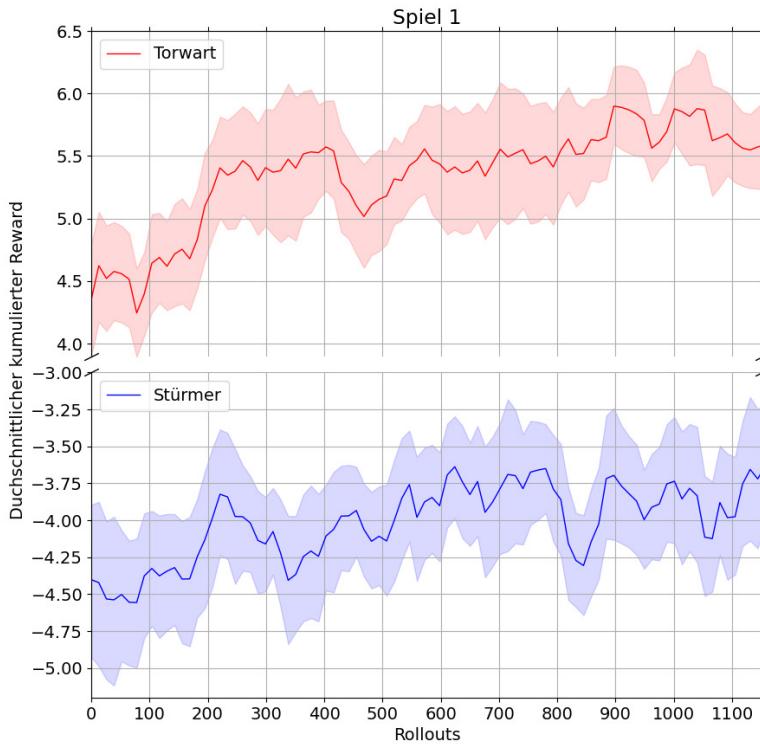


Abbildung 9.1: Durchschnittlicher kumulierter Reward pro Rollout für Spiel 1 der Torwart-Agenten (rot) und Stürmer-Agenten (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.

Hierbei ist eine deutliche Diskrepanz zwischen Torwart- und Stürmer-Rewardkurve zu erkennen. Da das Markov-Spiel als Nullsummenspiel definiert worden ist, entspricht der positive Reward eines Agenten dem negativen Reward des gegnerischen Agenten. Die hohe Diskrepanz zwischen den einzelnen Rewardkurven ist ein Indiz dafür, dass die zu erlernenden Verhalten für Stürmer- und Torwart-Stange unterschiedlich schwer sind. Da der Torwart durchweg von Beginn an einen höheren Reward erzielt als der Stürmer ist davon auszugehen, dass trotz der erschwerten Initialzustandsverteilung des Torwarts das Problem für diesen dennoch einfacher ist im Vergleich zum Stürmer.

Dennoch kann von beiden Agenten ein positives Lernverhalten beobachtet werden. In Abbildung 9.1 ist zu sehen, dass trotz des durchweg hohen Rewards für den Torwart, dieser im Laufe des ersten Spiels weiterhin ausgebaut werden konnte. Auch die Rewardkurve des Stürmers zeigt, dass die Stürmeragenten ebenfalls eine deutliche Rewardsteigerung im Laufe des Trainings erzielen konnten. Hierbei ist zu beachten, dass die jeweils trainierenden Agenten mit einer stationären Version des Gegners trainiert werden, wodurch das Steigen beider Rewardkurven trotz Modellierung als Nullsummenspiel möglich ist.

In Abbildung 9.2 ist das zweite Spiel und die Rewardkurven der jeweiligen Agenten zu sehen. Hierbei ist für die Torwarthe eine stagnierende Rewardkurve zu sehen. Dies ist dennoch positiv zu bewerten, da nun gegen einen trainierten Stürmeragenten gespielt worden ist.

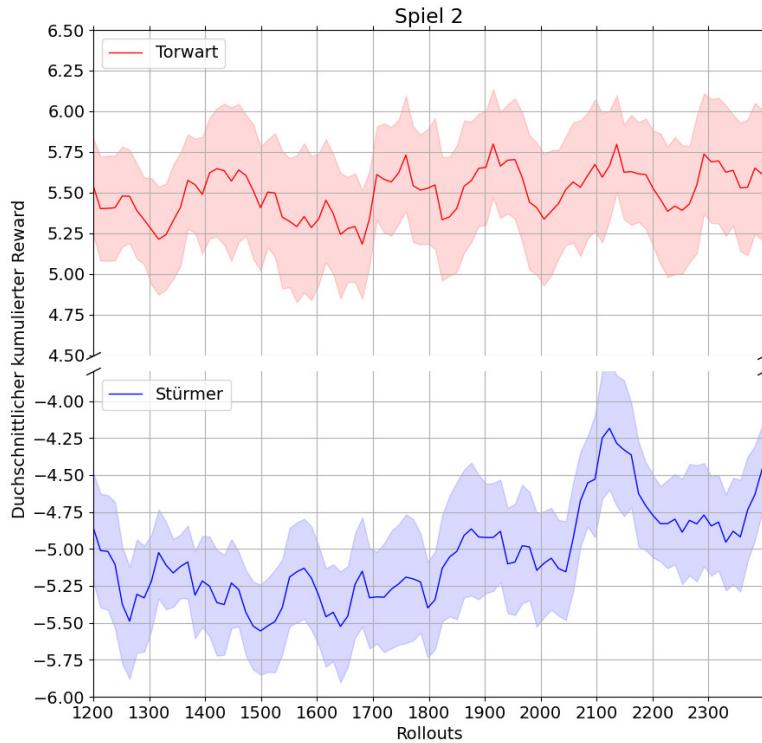


Abbildung 9.2: Durchschnittlicher kumulierter Reward pro Rollout für Spiel 2 der Torwart-Agenten (rot) und Stürmer-Agenten (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.

Für die Stürmer ist erneut eine deutliche Steigerung der Rewardkurve zu beobachten. Diese konnten im Laufe des zweiten Spiels den Reward gegen die trainierten Torwarthe ausbauen. Betrachtet man die Skalen der erreichten Rewards der Stürmer und Torwarthe kann man erkennen, dass es im zweiten Spiel zu einer deutlichen Reduktion des Stürmerrewards zu Beginn des Trainings kam.

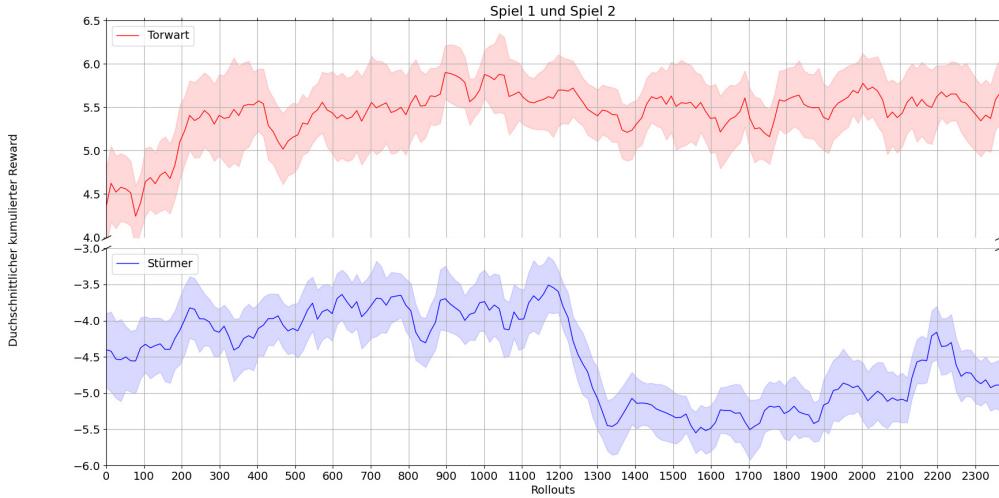


Abbildung 9.3: Durchschnittlicher kumulierter Reward pro Rollout für die Spiele 1 und 2 der Torwart-Agenten (rot) und Stürmer-Agenten (blau). Der schattierte Bereich stellt das jeweilige bootstrapped 95%-Konfidenzintervall dar.

In Abbildung 9.3 sind die konkatenierten Rewardkurven für Spiel 1 und Spiel 2 dargestellt. Bei Rollout 1200 ist hier ein deutlicher Verlust des Rewards für die Stürmer zu beobachten, während die Torwart-Rewardkurve lediglich leicht sinkt. Dies kann ein Indiz dafür sein, dass die vom Stürmeragenten im ersten Spiel erlernte Strategie gegen den trainierten Torwart nicht effektiv ist. Durch die Steigerung des Rewards der Stürmer im zweiten Spiel kann jedoch beobachtet werden, dass das erlernte Verhalten aus Spiel 1 an die trainierten Torwarte angepasst werden konnte.

9.5.2 Ergebnisse Evaluation

Als nächstes werden die im Rahmen von Algorithmus 3 entstandenen Evaluierungen der einzelnen Agentenstärken näher betrachtet. Zunächst wird die Stärke der Agenten zur Initialisierung betrachtet, welche in Tabelle 9.1 dargestellt ist.

Spiel 0	Torwart-1	Torwart-2
Stürmer-1	$0,787 \pm 0,49$	$0,741 \pm 0,57$
Stürmer-2	$0,775 \pm 0,54$	$0,781 \pm 0,56$

Tabelle 9.1: Auswertung der Agentenstärke bei Initialisierung. Alle Werte sind aus der Sicht des Torwerts.

Ein Wert von 1 bedeutet, dass in jeder Episode der Torwart den maximalen Reward erzielt hat, während ein Wert von -1 auf einen stark dominierenden Stürmer deutet. Hier bestätigt sich die Beobachtung auch aus den Reward-

kurven, dass von Beginn an ein deutliches Ungleichgewicht zwischen der Schwierigkeit der Umgebung für Torwart- und Stürmeragenten herrscht. Betrachtet man die Tabellen für die Spiele 1 [9.2](#) und 2 [9.3](#) so lässt sich erkennen, dass dieser Trend sich nicht ändert. Die Torwartagenten sind weiterhin dominierend gegenüber den Stürmeragenten. Dabei ist in Spiel 1 eine Steigerung von Torwart-2 gegenüber beiden Stürmern zu beobachten, während Torwart-1 sich nicht merklich in der erreichten Punktzahl gegenüber den Agenten steigern konnte. In Spiel 2 ist jedoch zu sehen, dass nun Torwart-2 eine deutlich geringere Punktzahl erreicht hat im Vergleich zur eigenen Leistung in Spiel 1. Gleichzeitig konnte sich nun Torwart-1 in dieser Iteration gegenüber beiden Stürmeragenten deutlich steigern.

Spiel 1	Torwart-1	Torwart-2
Stürmer-1	$0,781 \pm 0,52$	$0,811 \pm 0,42$
Stürmer-2	$0,751 \pm 0,54$	$0,881 \pm 0,40$

Tabelle 9.2: Auswertung nach der ersten Iteration. Alle Werte sind aus der Sicht des Torworts.

Spiel 2	Torwart-1	Torwart-2
Stürmer-1	$0,821 \pm 0,47$	$0,761 \pm 0,53$
Stürmer-2	$0,830 \pm 0,45$	$0,781 \pm 0,52$

Tabelle 9.3: Auswertung nach der zweiten Iteration. Alle Werte sind aus der Sicht des Torworts.

Dies bedeutet jedoch nicht notwendigerweise, dass die Stürmeragenten keinen Lernerfolg erzielt haben. Indem der erzielte Reward in einer ähnlichen Größenordnung geblieben ist kann dies ebenfalls bedeuten, dass die Stürmeragenten Strategien gegen besser werdende Torwarte erlernen konnten. Aus diesem Grund ist das erlernte Verhalten der Agenten zu untersuchen, um Schlüsse über den Lernerfolg ziehen zu können.

9.5.3 Erlernte Strategien

In diesem Abschnitt werden während des Trainings beobachtete Strategien von Stürmer- und Torwartagenten aufgezeigt. Diese wurden während des Trainings aufgezeichnet und können unter <https://github.com/kitaIRD/master-thesis-learned-agent-strategies> eingesehen werden [[Kit22](#)]. Hierfür wird zunächst auf die Torwartagenten eingegangen.

Abbildung [9.4](#) stellt die Positionierung des Torworts in Verhältnis zur Ball- und Stürmerposition dar.

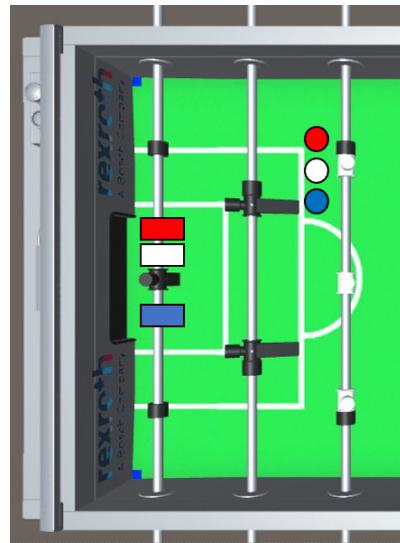


Abbildung 9.4: Beobachtbare Positionierung des Torwarts abhängig von der Ballposition relativ zur Stürmerfigur.

Dabei konnten verschiedene Verhalten beobachtet werden. Ist der Ball frontal gegenüber der Stürmerfigur positioniert, wie durch eine weiße Kugel in Abbildung 9.4 dargestellt, so ist eine Positionierung des Torwarts zwischen Tor und Ball zu erkennen, dargestellt durch ein weißes Rechteck in der Abbildung. Gleichzeitig konnte beobachtet werden, dass bei lateraler Bewegung des Stürmers nach oben (aus der Abbildungsperspektive) sich der Torwart neu positioniert hat. Durch diese Winkeländerung zwischen Stürmerfigur und Ball, dargestellt durch die blaue Kugel in Abbildung 9.4, kam es zu einer Positionierung des Torwarts in der entfernten Ecke des Tors, dargestellt durch das blaue Rechteck. Zu einer neuen Positionierung des Torwarts kam es auch, wenn sich der Stürmer lateral nach unten bewegt hat, dessen Winkeländerung als roter Ball dargestellt ist. Hierauf folgte eine Positionierung des Torwarts in der kurzen Ecke des Tores.

In Abbildung 9.5 wird dies nochmal verdeutlicht.

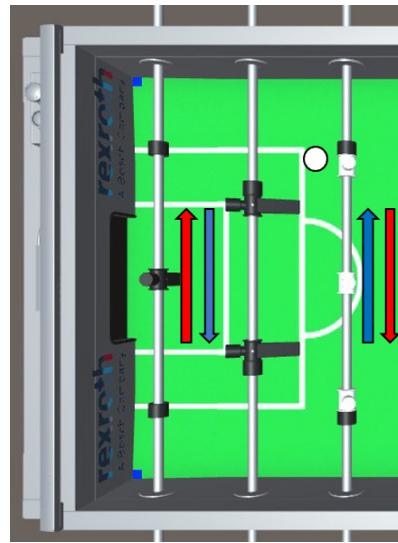


Abbildung 9.5: Beobachtbare Bewegungsmuster des Torwarts abhängig von Ballposition und Bewegung der Stürmerstange.

Die Pfeile stellen die laterale Bewegungsrichtungen beider Stangen dar. Hierbei ist zu beobachten, dass der Torwart sich abhängig zum Verhältnis zwischen Ball und Stürmerfiguren positioniert. Der Ball ist in der Abbildung weiß dargestellt. Kommt es zu einer Bewegung der Stürmerstange entlang des blauen Pfeiles, so bewegt sich die Torwartstange entlang des eigenen blauen Pfeiles ins lange Eck des Tors. Kommt es durch den Stürmer zur Bewegung entlang des roten Pfeiles so bewegt sich der Torwart ebenfalls entlang des eigenen roten Pfeiles in das kurze Eck.

In Abbildung 9.6 ist das Offensivverhalten des Torwärts dargestellt.

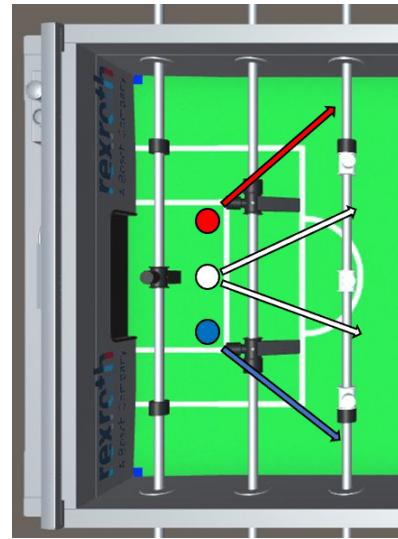


Abbildung 9.6: Beobachtbares Offensivverhalten des Torwärts abhängig von der Ballposition.

Während des Trainings konnte beobachtet werden, dass der Torwart gezielt in die Zwischenräume der Stürmerstange schießt. Abhängig von der Positionierung des Balles wurde hierbei der Raum zwischen den einzelnen Stürmerfiguren oder der Raum zwischen den äußersten Stürmerfiguren und der Wand genutzt.

Hierbei konnte bei einem Torwart-Agenten eine besondere Schusstechnik beobachtet werden, die das Ausnutzen der lediglich approximierten Simulation zeigt. Indem die Torwartstange vollständig in dessen Blickrichtung rotiert worden ist, wurden Schüsse so ausgeführt, dass wenn der Ball vor der Torwartfigur positioniert war, eine schlagartige Rotation entgegen der Blickrichtung der Torwartfigur stattfand. Dieses Rotieren von vorne nach hinten wurde zu einem Zeitpunkt ausgeführt, in dem der Ball kurz vor der Torwartfigur positioniert war. Dadurch konnten besonders schnelle Schüsse erzielt werden. Dieses Verhalten ist ebenfalls aufgezeichnet und im oben genannten Repository abrufbar.

Nun wird durch die Stürmer erlerntes Verhalten aufgezeigt. Abbildung 9.7 zeigt die Positionierung des Stürmers abhängig von der jeweiligen Ballposition in der Episode.

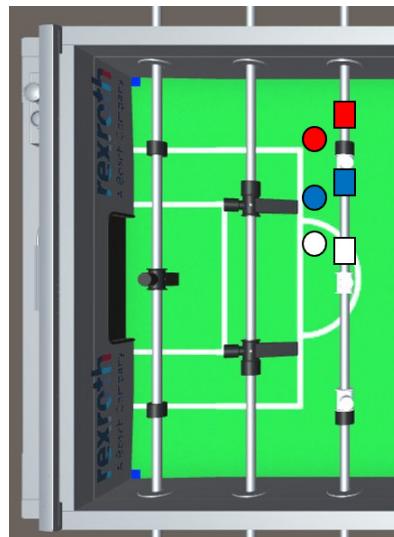


Abbildung 9.7: Beobachtbare Positionierung der Stürmerfigur abhängig von der jeweiligen Ballposition.

Ist der Ball wie durch den roten Kreis abgebildet positioniert, so positioniert die Stürmerstange den äußersten Spieler nach oben, dargestellt durch das rote Quadrat. Hierdurch können schiefe Schüsse ausgeführt werden. Anders sieht es aus im blauen Beispiel. Hierbei ist zu beobachten, dass die Stürmerstange sich nicht so weit nach außen bewegt wie im roten Beispiel. Landet der Ball zwischen zwei Figuren, dargestellt durch den weißen Ball, so konnte beobachtet werden, dass sich die Stürmerstange mit in diesem Beispiel der obersten oder der mittleren Spielerfigur dem Ball genähert hat. Hierbei war zu beobachten, dass der Stürmer in manchen Fällen zunächst

mit einer Außenfigur versucht hat sich dem Ball zu nähern, dann jedoch die mittlere Figur nutzt wenn der Ball durch die äußeren Figuren nicht erreicht werden konnte.

Abbildung 9.8 zeigt ein weiteres erlerntes Offensivverhalten des Stürmers.

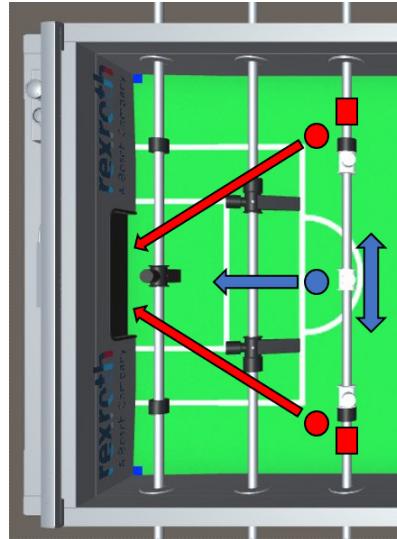


Abbildung 9.8: Beobachtbares Offensivverhalten des Stürmers bei Schüssen auf das Tor, abhängig von der jeweiligen Ballposition

Ist der Ball in günstiger direkter Schusslinie zum Tor zu Beginn einer Episode positioniert, dargestellt durch die roten Darstellungen in der Abbildung, so erfolgt ein schneller Schuss durch den Stürmer. Hierbei konnte gerade im zweiten Spiel beobachtet werden, dass der Torwart zwar oft versucht sich in die jeweilige Ecke zu bewegen, der Schuss aber oft zu schnell zum Halten ist. Anders ist es, wenn der Ball wie im blauen Beispiel dargestellt positioniert ist. Hier konnte beobachtet werden, dass es zu keinem direkten Schuss auf das Tor kommt. Stattdessen bewegt sich die Stürmerstange in kleinen Intervallen auf und ab und rotiert nur minimal. Hierbei konnte beobachtet werden, dass die Torwartstange sich abhängig zur Stürmerbewegung wie zuvor in Abbildung 9.5 erläutert ebenfalls in eine Richtung bewegt. Nach ein paar Sekunden der Bewegung wird anschließend ein Schuss durch die Stürmerfigur ausgeführt. Dies konnte ebenfalls erst in Spiel 2 beobachtet werden. Eine mögliche Interpretation hierfür ist, dass der Stürmer wartet bis sich der Torwart in eine Ecke des Tors positioniert hat, um dann in die entgegengesetzte Ecke zu schießen.

Abbildung 9.9 zeigt ein erlerntes Defensivverhalten der Stürmerstange.

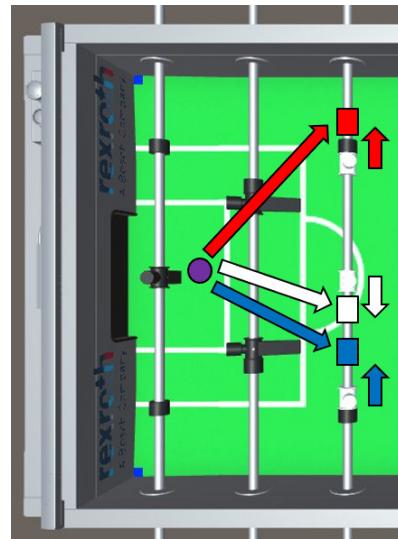


Abbildung 9.9: Beobachtbares Defensivverhalten des Stürmers, abhängig von der Laufrichtung des Balles.

Hierbei konnte beobachtet werden wie abhängig von der Laufrichtung des Balls unterschiedliche Spielerfiguren für das Halten genutzt werden. Während die Stürmerstange sich konstant leicht rotierend bewegt, konnte in Spiel 2 beobachtet werden, dass einzelne Schüsse des Torwarts bzw. Abpraller der Torwand so vom Stürmer gehalten werden konnten, dass der Ball nicht von den Spielern abprallt sondern von ihnen aktiv abgebremst wird. Anschließend konnte in manchen Fällen beobachtet werden, dass es zu einem anschließenden Schuss der Stürmerfigur gekommen ist.

FAZIT

Zunächst werden die erzielten Ergebnisse dieser Arbeit zusammengefasst und eingeordnet. Hierbei werden Schlüsse gezogen für die Nutzung von Multi-Agenten Training in der Tischkickerdomäne unter der Nebenbedingung möglichst Effizient gegenüber den Rechenressourcen zu sein. Abschließend werden offene Fragen benannt und ein Ausblick für mögliche weiterführende Arbeiten gegeben.

10.1 ERGEBNISSE UND EINORDNUNG

Ziel dieser Arbeit war das Entwerfen und Implementieren eines Trainingskonzeptes für das Multi-Agenten Training von Stürmer- und Torwart-Agenten in einer Simulation, unter der Nebenbedingung dies möglichst effizient in Bezug auf die vorhandenen Rechenressourcen zu gestalten. Hierbei wurde zunächst der Simulation-To-Reality-Gap der vorhandenen Unity-Simulation reduziert, indem eine Trajektoriengenerierung implementiert wurde die realistische Stangenbewegungen ermöglicht. Weiterhin wurde die Kommunikationsmethode zwischen Simulation und Agentenimplementierung überarbeitet, wodurch es zu einer Verdopplung der Leistung der Simulation kam. Neben den Änderungen an der Simulation wurde ein [MDP](#) für einen Stürmeragenten entworfen. Hierbei wurden systematisch einzelne Komponenten des [MDP](#) bestimmt, die das zu erlernende Verhalten möglichst effizient lernbar machen. Dabei wurden verschiedene Aspekte wie die Auswertung von Zeitlimits, der Einfluss von Initialzustandsverteilungen und das Policy-Invariante Verdichten von Rewardfunktionen im Detail beleuchtet. Des Weiteren wurden verschiedene Observations- und Aktionsräume experimentell und methodisch untersucht, um möglichst effiziente Konfigurationen dieser zu bestimmen.

Es wurde eine systematische Auswahl eines Lernalgorithmus vorgenommen und einzelne Hyperparameter durch Evaluierung verschiedener Literatur und mehrerer optimierter Konfigurationen für Benchmark-Umgebungen bestimmt. Des Weiteren wurde eine Implementierung gewählt bei der auf Quellcode-Optimierungen geachtet worden ist.

In diesem Rahmen wurde ein Konzept für das Trainieren von Stürmer- und Torwartagenten entworfen. Dabei wurde die Formulierung des Problems als Zwei-Parteien Nullsummen Markov-Spiel bestimmt. Für die Definition als Markov-Spiel wurden angepasste Observationsräume für die Agenten bestimmt und die auf [PBRS](#) basierte Rewardfunktion für das Nullsummenspiel erweitert. Weiterhin wurden für den genutzten Lernalgorithmus einzelne Modifikationen im Rahmen der Multi-Agenten Umgebung aufgezeigt und diskutiert. Es wurde die Modellierung von Gegnern definiert und eine Me-

thode zur Bestimmung von Gegnerpaaren ausgewählt und implementiert. Dabei konnte positives Lernverhalten sowohl von Stürmer- als auch von Torwartagenten anhand der Rewardkurven beobachtet werden. Weiterhin konnten erlernte Strategien beider Agenten während des Trainings beobachtet werden.

Unter Berücksichtigung der Ergebnisse lässt sich feststellen, dass das entworfene Trainingskonzept für die Tischkickerdomäne geeignet ist. Dabei konnte mit begrenzten Rechenressourcen Multi-Agenten Training erfolgen, wodurch ein Ensemble an Stürmer- und Torwartagenten miteinander trainiert werden konnten. Trotz dem Ungleichgewicht der Schwierigkeitsgrade für Stürmer- und Torwartagenten zeigten beide Arten von Agenten ein positives Lernverhalten auf, wobei beide komplexe Verhalten erlernen konnten, die das gegnerische Verhalten für die eigene Aktionswahl mit berücksichtigen.

10.2 OFFENE FRAGEN

In Kapitel 8 konnte am Versuch mit veränderten Hyperparametern gesehen werden, dass diese einen großen Einfluss auf das Trainingsverhalten der Agenten nehmen können. Hierbei wäre die Auswirkung von optimierten Hyperparametern in Bezug auf das Multi-Agenten Training ein weiterer zu beleuchtender Aspekt. Einen Überblick über automatische Hyperparameter-Optimierungsalgorithmen und Applikationen stellt [YZ20] dar.

Weiterhin konnte von Beginn an ein großes Ungleichgewicht zwischen Torwart- und Stürmeragenten beim Multi-Agenten Training beobachtet werden. Hier wäre zu untersuchen, inwieweit dies durch längeres Training oder geänderte Umgebungsbedingungen, wie eine breitere Initialzustandsverteilung kompensiert werden kann.

Weitere Arbeiten können sich ebenfalls mit der Nutzung von Imitation-Learning [Cai+21] oder Pretraining [ALE15] befassen, mit denen Agenten auf eine Anfangsstärke trainiert werden können um anschließend in Multi-Agenten Umgebungen aufbauend darauf weitere Strategien zu erlernen. Hierdurch müssen die Dynamiken der Umgebung nicht von Beginn an im erschweren Multi-Agenten Markov-Spiel erlernt werden, was die Trainingsgeschwindigkeit potenziell erhöhen könnte.

Teil II

APPENDIX

A

APPENDIX

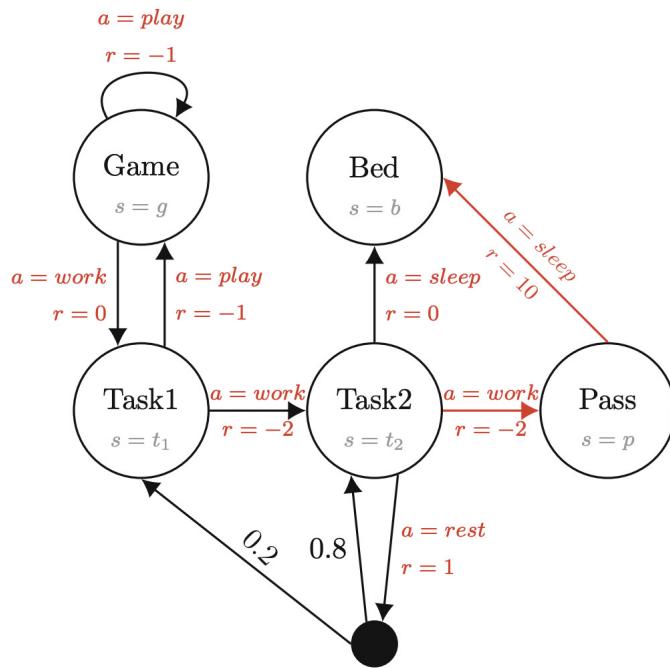


Abbildung A.1: Beispiel eines MDP: a stellt Aktionen dar die zu Zustandtransitio-
nen führen können; r stellt den Reward für eine ausgeführte Aktion
im jeweiligen Zustand dar. Der schwarze Knoten stellt den Initial-
zustand dar. [DDZ20, S. 64]

Systemmodell	ThinkpadP1Gen2	Lenovo81NX
Prozessor	Intel Core i7-9750H CPU @ 2.60GHz	Intel Core i7-9750H CPU @ 2.60GHz
Installierter RAM	32.0GB	16.0GB
Systemtyp	64-Bit-Betriebssystem x64-basierter Prozessor	64-Bit-Betriebssystem x64-basierter Prozessor
GPU	NVIDIA Quadro T1000	NVIDIA GeForce GTX 1650 mit Max-Q Design

Tabelle A.1: Hardware Spezifikationen der zur Verfügung stehenden Rechenres-
sourcen

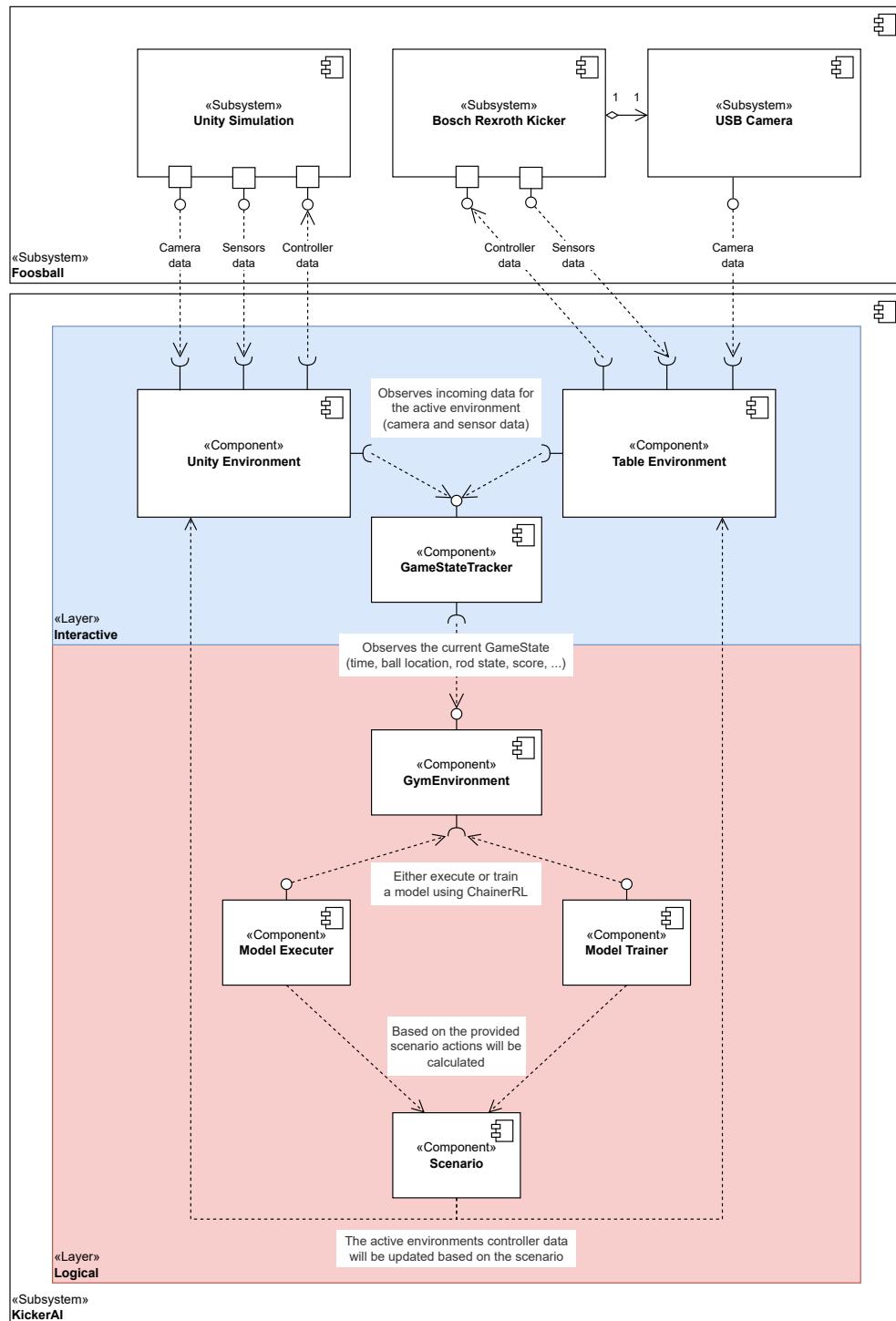


Abbildung A.2: Komponentendiagramm des Quellcodes vom Wintersemester 2022
[Wil+22]

OpenAI Gym RL-Umgebungen	Dauer pro Zeitschritt [s]	Frequenz [Hz]
Ant	0,01	100
Half Cheetah	0,01	100
Hopper	0,002	500
Humanoid	0,003	333,33
Humanoid-Standup	0,003	333,33
Inverted Double Pendulum	0,01	100
Inverted Pendulum	0,02	50
Point	0,02	50
Pusher	0,01	100
Reacher	0,01	100
Swimmer	0,01	100
Walker2d	0,002	500
<hr/>		
Durchschnitt	0,0092	197,22
Median	0,01	100

Tabelle A.2: Die MuJoCo Umgebungen OpenAI Gym mit ihren jeweiligen Zeitschritt-Frequenzen. [Bro+16a; Ope22]

DeepMind RL-Umgebungen	Dauer pro Zeitschritt [s]	Frequenz [Hz]
Acrobat	0,01	100
Cartpole	0,01	100
Cheetah	0,01	100
Dog	0,005	200
Finger	0,01	100
Fish	0,004	250
Hopper	0,005	200
Humanoid	0,005	200
Lqr	0,03	33,33
Manipulator	0,001	1000
Pendulum	0,02	50
Point Mass	0,02	50
Quadruped	0,005	200
Reacher	0,02	50
Stacker	0,001	1000
Swimmer	0,002	500
Walker	0,0025	400
<hr/>		
Durchschnitt	0,0094	266,67
Median	0,005	200

Tabelle A.3: Die MuJoCo Umgebungen der DeepMind Control Suite mit ihren jeweiligen Zeitschritt-Frequenzen. [Tun+20]

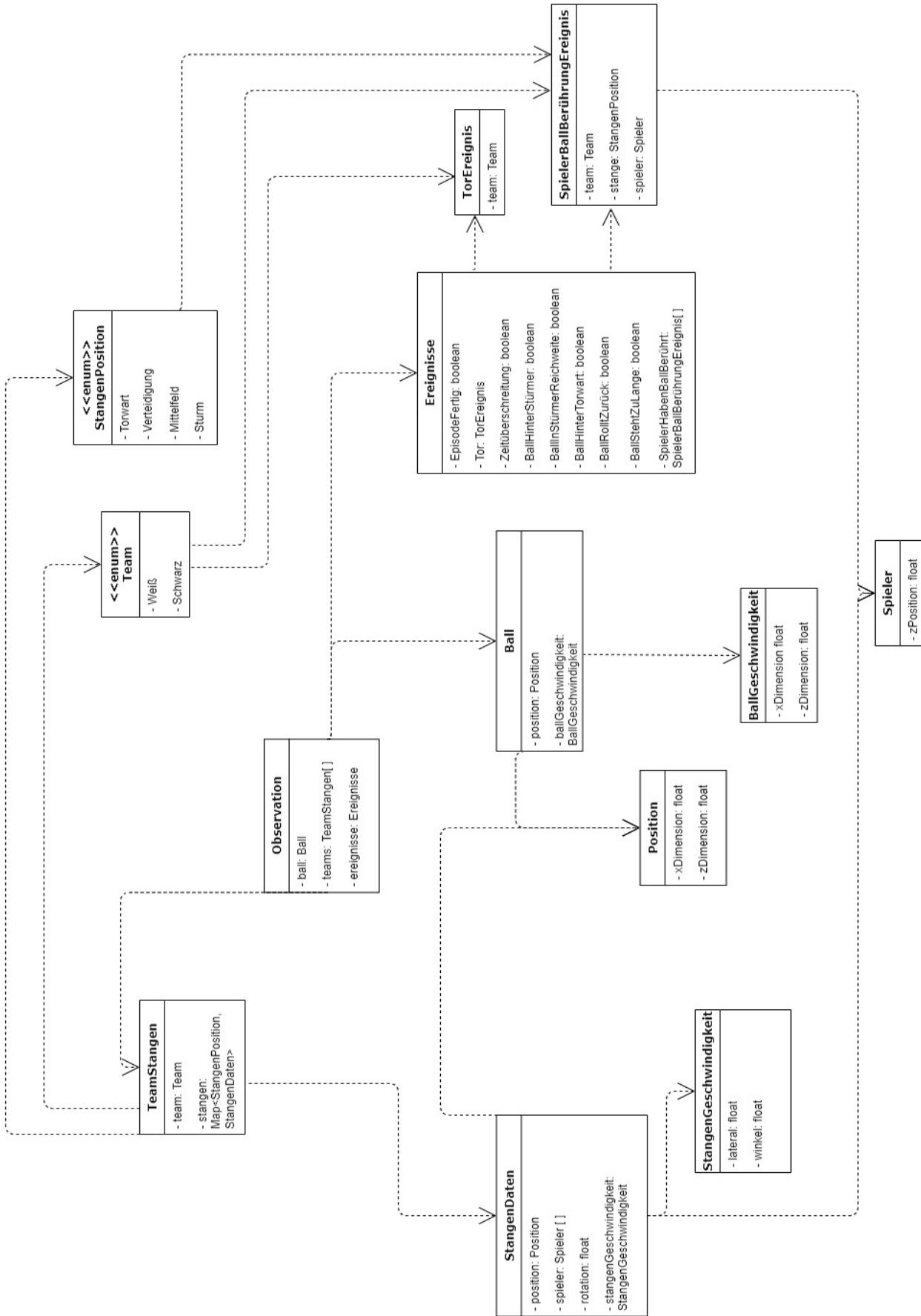


Abbildung A.3: Datenmodell für die Übertragung von Observationen aus der Simulation an den Agenten.

Classic Control Umgebungen		Pendulum-v1	CartPole-v1	MountainCarContinuous-v0	MountainCar-v0	Acrobot-v1	BipedalWalker-v3	BipedalWalkerHardcore-v3	LunarLander-v2	LunarLanderContinuous-v2
Name										
Aktionraum	1-D, kontinuierlich	1-D, diskret	1-D, kontinuierlich	2-D, kontinuierlich	2-D, kontinuierlich	1-D, diskret	4-D, kontinuierlich	4-D, kontinuierlich	1-D, diskret	2-D, kontinuierlich
Observationsraum	3-D, kontinuierlich	4-D, kontinuierlich	2-D, kontinuierlich	3-D, kontinuierlich	3-D, kontinuierlich	6-D, kontinuierlich	24-D, kontinuierlich	24-D, kontinuierlich	8-D, kontinuierlich	8-D, kontinuierlich
Advantage Normalisierung				TRUE		TRUE	TRUE	TRUE		
Trainingsdauer	100.000	100.000	20.000	20.000	1.000.000	1.000.000	5.000.000	100.000.000	1.000.000	1.000.000
Minibatch Größe	256	256	256	8	256	4096	32768	32768	64	64
Rollout Länge	4096	2048							16384	
Diskontinuierungs faktor	0.9	0.98	0.9999	0.99	0.99	0.99	0.99	0.99	0.99	0.999
GAE_lambda	0.95	0.8	0.9	0.98	0.98	0.94	0.95	0.95	0.95	0.98
Anzahl Epochs	10	20	10	4	4	10	10	10	4	4
Entropie Koeffizient	0	0	0.00429	0	0	0.001	0.001	0.001	0.01	0.01
max_grad_norm			5							
Wertefunktion Koeffizient			0.19							
Lernrate	1.00E-03	0.001	7.77E-05			2.50E-04	2.50E-04			
Clip Parameter	0.2	0.2	0.1			0.2	0.2			
Aktivierungsfunktion										
Orthogonale Initialisierung					FALSE					
NN Architektur										

Tabelle A.4: Optimierte Hyperparameter für OpenAI Gym *Classic Control* Umgebungen [Raf20]. Leere Felder stellen die Nutzung der SB₃ Standard-Hyperparameter dar.

PyBullet Umgebungen			
Name	HalfCheetahBulletEnv-v0	ReacherBulletEnv-v0	AnBulletEnv-v0
Aktionsraum	6-D, kontinuierlich	2-D, kontinuierlich	8-D, kontinuierlich
Observationsraum	17/18-D, kontinuierlich	11-D, kontinuierlich	111-D, kontinuierlich
Advantage Normalisierung	TRUE	TRUE	17-D, kontinuierlich
Trainingsdauer	2.000.000	1.000.000	
Minibatch Größe	128	64	
Rollout Länge	8192	4096	
Diskontierungsfaktor	0.99	0.99	
GAE_Lambda	0.9	0.9	
Anzahl Epochs	20	20	
Entropie Koeffizient	0	0	
max_grad_norm	0.5	0.5	
Wertefunktion Koeffizient	0.5	0.5	
Lernrate	3.00E-05	3.00E-05	3.00E-05
Clip Parameter	0.4	0.4	0.4
Aktivierungsfunktion	nn.ReLU	nn.ReLU	nn.ReLU
Orthogonale Initialisierung	FALSE	FALSE	FALSE
NN Architektur	pi = [256, 256] vf = [256, 256]	pi = [256, 256] vf = [256, 256]	pi = [256, 256] vf = [256, 256]

PyBullet Umgebungen			
Name	InvertedPendulumBulletEnv-v0	InvertedPendulumSwingupBulletEnv-v0	HopperBulletEnv-v0
Aktionsraum	1-D, kontinuierlich	1-D, kontinuierlich	3-D, kontinuierlich
Observationsraum	11-D, kontinuierlich	3-D, kontinuierlich	11-D, kontinuierlich
Advantage Normalisierung	TRUE	TRUE	
Trainingsdauer	2.000.000	2.000.000	
Minibatch Größe	64	64	
Rollout Länge	16384	16384	
Diskontierungsfaktor	0.99	0.99	
GAE_Lambda	0.95	0.95	
Anzahl Epochs	10	10	
Entropie Koeffizient	0	0	
max_grad_norm			
Wertefunktion Koeffizient			
Lernrate	2.50E-04	2.50E-04	3.00E-05
Clip Parameter	0.2	0.2	0.4
Aktivierungsfunktion		nn.ReLU	
Orthogonale Initialisierung		FALSE	
NN Architektur		pi = [256, 256] vf = [256, 256]	

Tabelle A.5: Optimierte Hyperparameter für OpenAI Gym PyBullet Umgebungen [Raf20]. Leere Felder stellen die Nutzung der SB₃ Standard-Hyperparameter dar.

MuJoCo Umgebungen						
Name	HalfCheetah-v3	Ant-v3	Hopper-v3	Walker2d-v3	Humanoid-v3	Swimmer-v3
Aktionsraum	6-D, kontinuierlich	8-D, kontinuierlich	3-D, kontinuierlich	6-D, kontinuierlich	17-D, kontinuierlich	6-D, kontinuierlich
Observationsraum	17/18-D, kontinuierlich	11/113-D, kontinuierlich	11/12-D, kontinuierlich	17/18-D, kontinuierlich	376/378-D, kontinuierlich	8-D, kontinuierlich
Advantage Normalisierung	TRUE					TRUE
Trainingsdauer	1.000.000				2.000.000	
Minibatch Größe						10.000.000
Rollout Länge						64
Diskontierungsfaktor						0.9999
GAE lambda						0.98
Anzahl Epochs						0.92
Entropie Koeffizient						0.1
max_grad_norm						0.8
Wertfunktion Koeffizient						0.6
Lernrate						0.677239
Clip Parameter						2.06E-05
Aktivierungsfunktion						0.1
Orthogonale Initialisierung						nn.ReLU
NN Architektur						pi = [256, 256] vf = [256, 256]

Walker2d-v3						
Name	Hopper-v3	HumanoidStandup-v3	Humanoid-v3	InvertedPendulum-v3	InvertedDoublePendulum-v3	Reacher-v3
Aktionsraum	3-D, kontinuierlich	17-D, kontinuierlich	17-D, kontinuierlich	1-D, kontinuierlich	1-D, kontinuierlich	2-D, kontinuierlich
Observationsraum	11/12-D, kontinuierlich	376-D, kontinuierlich	376/378-D, kontinuierlich	11-D, kontinuierlich	4-D, kontinuierlich	8/10-D, kontinuierlich
Advantage Normalisierung	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Trainingsdauer	1.000.000	10.000.000	10.000.000	1.000.000	1.000.000	1.000.000
Minibatch Größe	32	32	256	512	64	32
Rollout Länge	512	512	32	128	512	512
Diskontierungsfaktor	0.999	0.99	0.95	0.98	0.989	0.9
GAE lambda	0.99	0.9	0.9	0.8	0.9	0.95
Anzahl Epochs	5	20	5	10	5	10
Entropie Koeffizient	0.00223519	0.0000362109	0.00238306	0.0000105057	0.00000137976	0.0554757
max_grad_norm	0.7	0.7	2	0.5	0.3	0.6
Wertfunktion Koeffizient	0.835671	0.430793	0.4311892	0.695929	0.19816	0.38782
Lernrate	9.81E-05	2.56E-05	3.57E-05	1.55E-04	2.22E-04	1.04E-04
Clip Parameter	0.2	0.3	0.3	0.4	0.4	0.5E-05
Aktivierungsfunktion	nn.ReLU	nn.ReLU	nn.ReLU	nn.ReLU	nn.ReLU	nn.ReLU
Orthogonale Initialisierung	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
NN Architektur	pi = [256, 256] vf = [256, 256]	pi = [256, 256] vf = [256, 256]	pi = [256, 256] vf = [256, 256]	pi = [256, 256] vf = [256, 256]	pi = [256, 256] vf = [256, 256]	pi = [256, 256] vf = [256, 256]

Tabelle A.6: Optimierte Hyperparameter für OpenAI Gym MuJoCo Umgebungen [Raf20]. Leere Felder stellen die Nutzung der SB3 Standard-Hyperparameter dar.

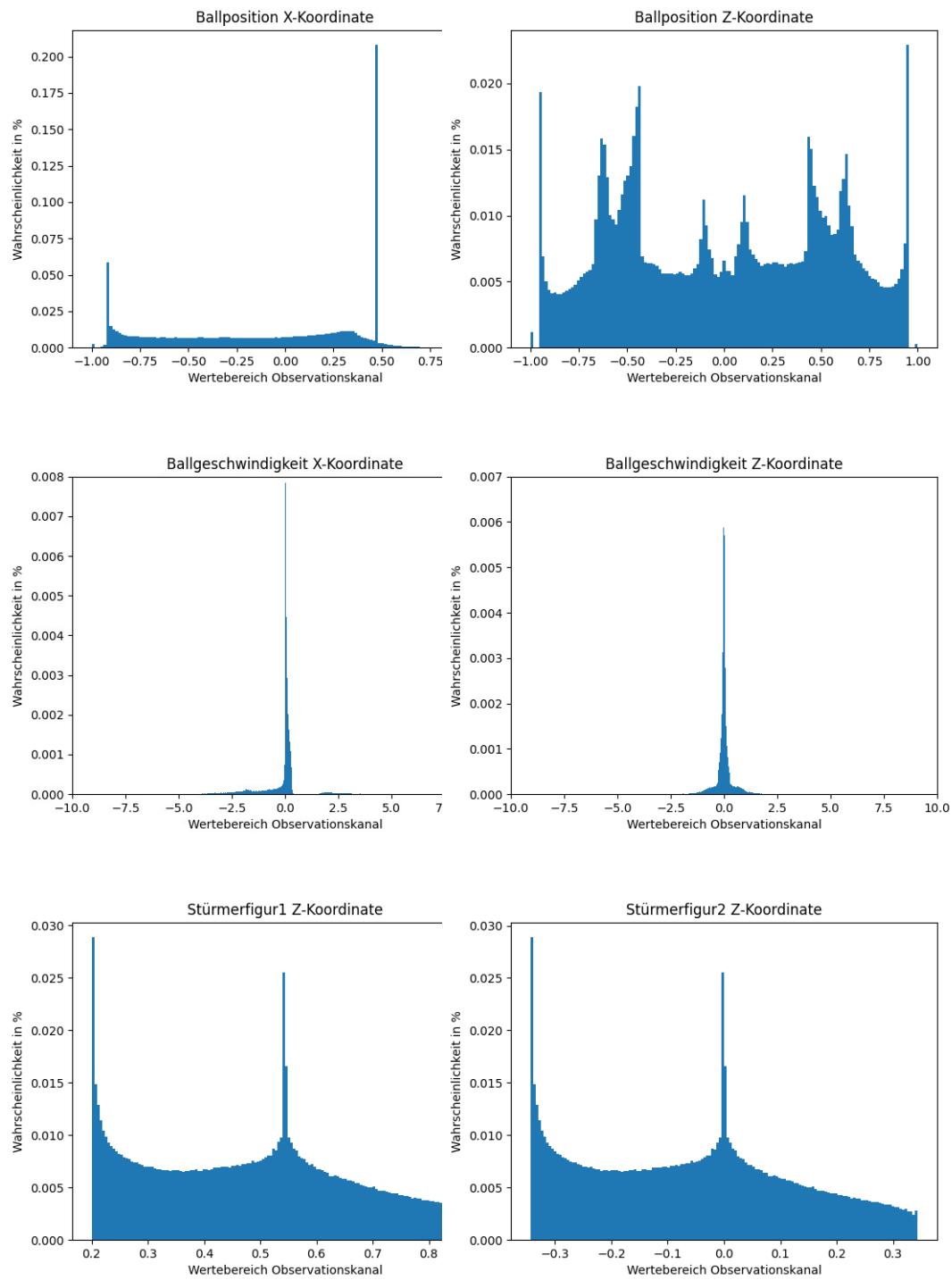


Abbildung A.4: Histogramme der gesammelten Observationen während einer Trainingsdauer von 2,16 Mio. Trainingsschritten (1/2)

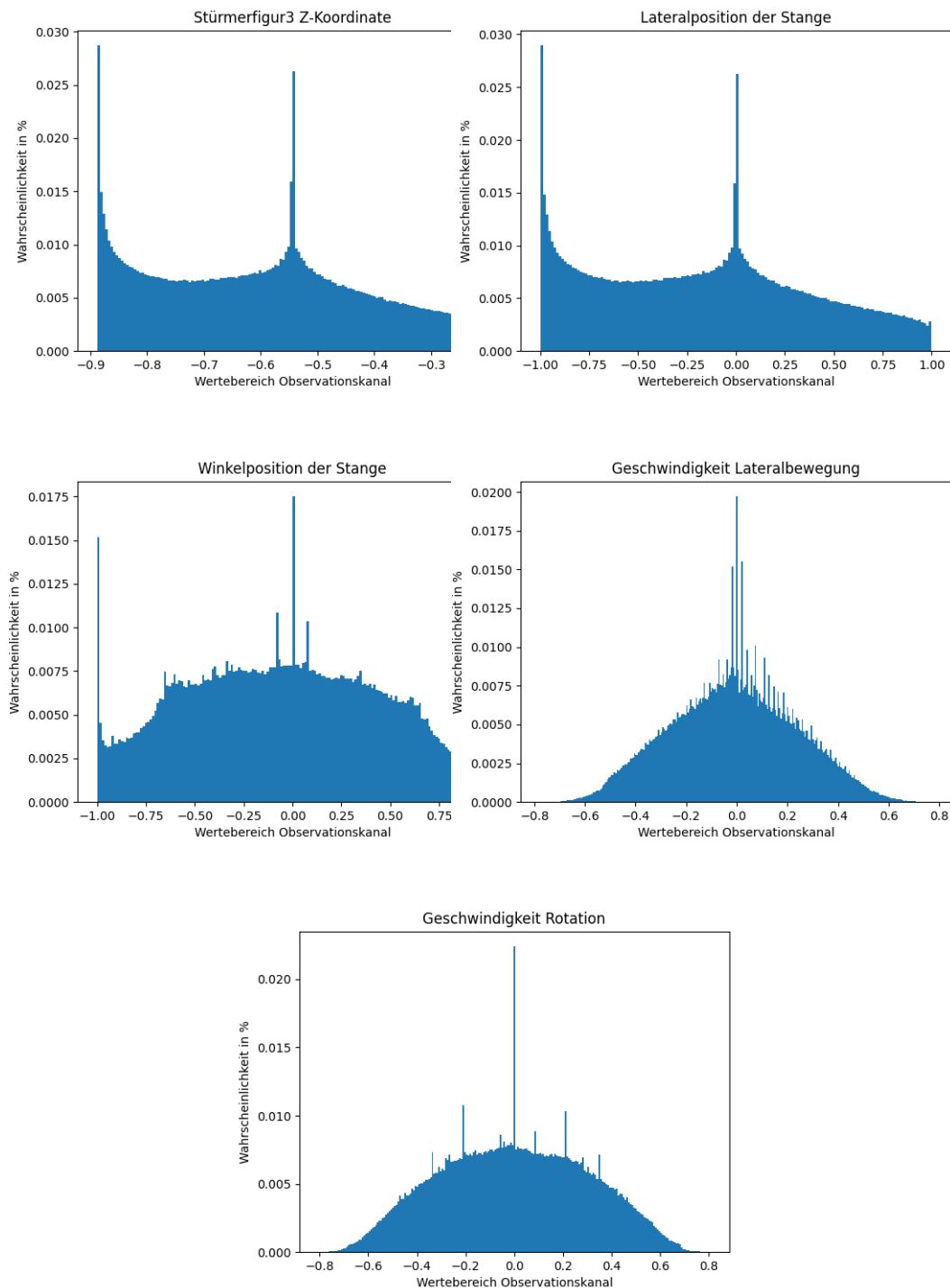


Abbildung A.5: Histogramme der gesammelten Observationen während einer Trainingsdauer von 2,16 Mio. Trainingsschritten (2/2)

LITERATUR

- [Aki+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta und Masanori Koyama. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *CoRR* abs/1907.10902 (2019). arXiv: 1907.10902. URL: <http://arxiv.org/abs/1907.10902>.
- [Alz+21] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie und Laith Farhan. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (März 2021). DOI: 10.1186/s40537-021-00444-8. URL: <https://doi.org/10.1186/s40537-021-00444-8>.
- [ALE15] Charles W. Anderson, Minwoo Lee und Daniel L. Elliott. "Faster reinforcement learning after pretraining deep networks to predict state dynamics". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, S. 1–7. DOI: 10.1109/IJCNN.2015.7280824.
- [AY] Gabriel Andersson und Martti Yap. "Learning to Play Breakout Using Deep Q-Learning Networks". In: (), S. 3.
- [And+20] Marcin Andrychowicz u. a. "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study". In: *CoRR* abs/2006.05990 (2020). arXiv: 2006.05990. URL: <https://arxiv.org/abs/2006.05990>.
- [Aut] gRPC Authors. *gRPC - Documentation*. English. Zuletzt geprüft am 16.05.2022. URL: <https://grpc.io/docs/>.
- [Bak+19] Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew und Igor Mordatch. "Emergent Tool Use From Multi-Agent Autocurricula". In: *CoRR* abs/1909.07528 (2019). arXiv: 1909.07528. URL: <http://arxiv.org/abs/1909.07528>.
- [Ban+17] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever und Igor Mordatch. "Emergent Complexity via Multi-Agent Competition". In: *CoRR* abs/1710.03748 (2017). arXiv: 1710.03748. URL: <http://arxiv.org/abs/1710.03748>.
- [BMo3] Andrew G. Barto und Sridhar Mahadevan. In: *Discrete Event Dynamic Systems* 13.1/2 (2003), S. 41–77. DOI: 10.1023/a:1022140919877. URL: <https://doi.org/10.1023/a:1022140919877>.

- [Bel+13] M. G. Bellemare, Y. Naddaf, J. Veness und M. Bowling. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47 (2013), S. 253–279. DOI: [10.1613/jair.3912](https://doi.org/10.1613/jair.3912). URL: <https://doi.org/10.1613/jair.3912>.
- [Bel54] Richard Bellman. "The theory of dynamic programming". In: *Bulletin of the American Mathematical Society* 60.6 (Nov. 1954), S. 503–516. DOI: [10.1090/s0002-9904-1954-09848-8](https://doi.org/10.1090/s0002-9904-1954-09848-8). URL: <https://doi.org/10.1090/s0002-9904-1954-09848-8>.
- [Ber+19] Christopher Berner u. a. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *CoRR* abs/1912.06680 (2019). arXiv: [1912.06680](https://arxiv.org/abs/1912.06680). URL: [http://arxiv.org/abs/1912.06680](https://arxiv.org/abs/1912.06680).
- [BB17] Marius Beul und Sven Behnke. "Fast Full State Trajectory Generation for Multirotors". In: *Proceedings of International Conference on Unmanned Aircraft Systems (ICUAS)*. Miami, FL, USA, Juni 2017. DOI: [10.1109/ICUAS.2017.7991304](https://doi.org/10.1109/ICUAS.2017.7991304). URL: <https://github.com/AIS-Bonn/TopiCo>.
- [BM19] Jonathan Plante Bhairav Mehta. *Effects of Task Distributions on Policy Optimization*. Zuletzt geprüft am 17.05.2022. 2019. URL: <https://bhairavmehta95.github.io/static/optimizationadr.pdf>.
- [Bou+22] Robert N. Boute, Joren Gijsbrechts, Willem van Jaarsveld und Nathalie Vanvuchelen. "Deep reinforcement learning for inventory control: A roadmap". In: *European Journal of Operational Research* 298.2 (Apr. 2022), S. 401–412. DOI: [10.1016/j.ejor.2021.07.016](https://doi.org/10.1016/j.ejor.2021.07.016). URL: <https://doi.org/10.1016/j.ejor.2021.07.016>.
- [Bra+15] Alexander Braylan, Mark Hollenbeck, Elliot Meyerson und Risto Miikkulainen. "Frame Skip Is a Powerful Parameter for Learning to Play Atari". In: *AAAI Workshop: Learning for General Competency in Video Games*. 2015. URL: <http://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/10156>.
- [Bro+16a] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang und Wojciech Zaremba. *OpenAI Gym*. 2016. eprint: [arXiv:1606.01540](https://arxiv.org/abs/1606.01540).
- [Bro+16b] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang und Wojciech Zaremba. *OpenAI Gym*. <https://github.com/openai/gym/tree/master/gym/envs/mujoco/assets>. 2016.
- [Cai+21] Xin-Qiang Cai, Yao-Xiang Ding, Zi-Xuan Chen, Yuan Jiang, Masaaki Sugiyama und Zhi-Hua Zhou. *Seeing Differently, Acting Similarly: Imitation Learning with Heterogeneous Observations*. 2021. DOI: [10.48550/ARXIV.2106.09256](https://doi.org/10.48550/ARXIV.2106.09256). URL: <https://arxiv.org/abs/2106.09256>.

- [CM+14] Ruzinoor Che Mat PhD, rashid shariff, Abdul Nasir Zulkifli, Mohd Rahim und Mohd Hafiz. "Using game engine for 3D terrain visualisation of GIS data: A review". In: *IOP Conference Series: Earth and Environmental Science* 20 (Juni 2014), S. 012037. DOI: [10.1088/1755-1315/20/1/012037](https://doi.org/10.1088/1755-1315/20/1/012037).
- [Che+21] Lili Chen, Kimin Lee, Aravind Srinivas und Pieter Abbeel. "Improving Computational Efficiency in Visual Reinforcement Learning via Stored Embeddings". In: *CoRR* abs/2103.02886 (2021). arXiv: [2103.02886](https://arxiv.org/abs/2103.02886). URL: <https://arxiv.org/abs/2103.02886>.
- [Con19] Torch Contributors. *Dropout — PyTorch 1.11.0 documentation*. <https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html>. Zuletzt geprüft am 17.05.2022. 2019.
- [DHL15] James Davis, Yi-Hsuan Hsieh und Hung-Chi Lee. "Humans perceive flicker artifacts at 500 Hz". In: *Scientific Reports* 5.1 (Feb. 2015). DOI: [10.1038/srep07861](https://doi.org/10.1038/srep07861). URL: <https://doi.org/10.1038/srep07861>.
- [DB+21] Stefano De Blasi, Sebastian Klöser, Arne Müller, Robin Reuben, Fabian Sturm und Timo Zerrer. "Kicker: an industrial drive and control foosball system automated with deep reinforcement learning". In: *Journal of Intelligent & Robotic Systems* 102.1 (2021), S. 1–18.
- [Dha+17] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu und Peter Zhokhov. *OpenAI Baselines*. <https://github.com/openai/baselines>. 2017.
- [DD05] Dmitri A. Dolgov und Edmund H. Durfee. "Stationary Deterministic Policies for Constrained MDPs with Multiple Rewards, Costs, and Discount Factors". In: *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*. Edinburgh, Scotland, 2005, S. 1326–1332.
- [DDZ20] Hao Dong, Zihan Ding und Shanghang Zhang, Hrsg. *Deep Reinforcement Learning*. Springer Singapore, 2020. DOI: [10.1007/978-981-15-4095-0](https://doi.org/10.1007/978-981-15-4095-0). URL: <https://doi.org/10.1007/978-981-15-4095-0>.
- [Eng+20] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph und Aleksander Madry. "Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO". In: *CoRR* abs/2005.12729 (2020). arXiv: [2005.12729](https://arxiv.org/abs/2005.12729). URL: <https://arxiv.org/abs/2005.12729>.
- [Fed+20] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland und Will Dabney. *Revisiting Fundamentals of Experience Replay*. 2020. DOI: [10.48550/ARXIV.2007.06700](https://doi.org/10.48550/ARXIV.2007.06700). URL: <https://arxiv.org/abs/2007.06700>.

- [FFL22a] Ruohan Gao Fei-Fei Li Jiajun Wu. *CS231n: Deep Learning for Computer Vision*. <https://cs231n.github.io/neural-networks-1/>. Zuletzt geprüft am 18.05.2022. 2022.
- [FFL22b] Ruohan Gao Fei-Fei Li Jiajun Wu. *CS231n: Deep Learning for Computer Vision*. <https://cs231n.github.io/neural-networks-2/>. Zuletzt geprüft am 18.05.2022. 2022.
- [Foe+17] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli und Shimon Whiteson. *Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning*. 2017. doi: [10.48550/ARXIV.1702.08887](https://doi.org/10.48550/ARXIV.1702.08887). URL: <https://arxiv.org/abs/1702.08887>.
- [Fra] *Frame Skipping and Pre-Processing for Deep Q-Networks on Atari 2600 Games*. Zuletzt geprüft am 05.05.2022. 4/24/2022. URL: <https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/>.
- [Fuj+21] Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka und Taka-hiro Ishikawa. "ChainerRL: A Deep Reinforcement Learning Library". In: *Journal of Machine Learning Research* 22.77 (2021), S. 1–14. URL: <http://jmlr.org/papers/v22/20-376.html>.
- [GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [GK19] Laura Graesser und Wah Loon Keng. *Foundations of Deep Reinforcement Learning - Theory and Practice in Python*. Amsterdam: Addison-Wesley, 2019. Kap. Epilogue. ISBN: 978-0-135-17238-4.
- [GK20] Laura Graesser und Wah Loon Keng. *Foundations of deep reinforcement learning*. Addison-Wesley Data & Analytics Series. Boston, MA: Addison Wesley, Feb. 2020.
- [Gro] Roger Grosse. *Lecture 8: Optimization*. https://www.cs.toronto.edu/~cmaddis/courses/sta314_f21/rgrosses_optimization_notes.pdf. Zuletzt geprüft am 18.05.2022.
- [Haa+18] Tuomas Haarnoja u. a. *Soft Actor-Critic Algorithms and Applications*. 2018. doi: [10.48550/ARXIV.1812.05905](https://doi.org/10.48550/ARXIV.1812.05905). URL: <https://arxiv.org/abs/1812.05905>.
- [Har19] Joshua Hare. *Dealing with Sparse Rewards in Reinforcement Learning*. 2019. doi: [10.48550/ARXIV.1910.09281](https://doi.org/10.48550/ARXIV.1910.09281). URL: <https://arxiv.org/abs/1910.09281>.
- [Hen+17] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup und David Meger. "Deep Reinforcement Learning that Matters". In: *CoRR* abs/1709.06560 (2017). arXiv: [1709.06560](https://arxiv.org/abs/1709.06560). URL: [http://arxiv.org/abs/1709.06560](https://arxiv.org/abs/1709.06560).
- [Hin13] Pieter Hintjens. *ZeroMQ*. en. Sebastopol, CA: O'Reilly Media, März 2013. Kap. 5. Pros and Cons of Publish-Subscribe.

- [HT21] Md Riyad Hossain und Douglas Timmer. "Machine Learning Model Optimization with Hyper Parameter Tuning Approach". In: *Global Journal of Computer Science and Technology* (2021).
- [Iba+18] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg und Dario Amodei. "Reward learning from human preferences and demonstrations in Atari". In: *CoRR* abs/1811.06521 (2018). arXiv: 1811.06521. URL: <http://arxiv.org/abs/1811.06521>.
- [JBM10] Rob Janssen, Jeroen de Best und René van de Molengraft. "Real-Time Ball Tracking in a Semi-automated Foosball Table". In: *RoboCup 2009: Robot Soccer World Cup XIII*. Hrsg. von Jacky Baltes, Michail G. Lagoudakis, Tadashi Naruse und Saeed Shiry Ghidary. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 128–139. ISBN: 978-3-642-11876-0.
- [JA21] Shuo Jiang und Christopher Amato. "Multi-agent reinforcement learning with directed exploration and selective memory reuse". In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 2021, S. 777–784.
- [KLM96] L. P. Kaelbling, M. L. Littman und A. W. Moore. "Reinforcement learning: A survey". In: *Journal of Artificial Intelligence Research* 4 (1996), S. 237–285.
- [KLC98] Leslie Pack Kaelbling, Michael L. Littman und Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1 (1998), S. 99–134. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/S0004-3702\(98\)00023-X](https://doi.org/10.1016/S0004-3702(98)00023-X). URL: <https://www.sciencedirect.com/science/article/pii/S000437029800023X>.
- [Kak03] Sham Machandranath Kakade. "On the sample complexity of reinforcement learning". Diss. University College London, März 2003.
- [Kal+21] Shivaram Kalyanakrishnan, Siddharth Aravindan, Vishwajeet Bagdawat, Varun Bhatt, Harshith Goka, Archit Gupta, Kalpesh Krishna und Vihari Piratla. "An Analysis of Frame-skipping in Reinforcement Learning". In: *CoRR* abs/2102.03718 (2021). arXiv: 2102.03718. URL: <https://arxiv.org/abs/2102.03718>.
- [KSH20] Anssi Kanervisto, Christian Scheller und Ville Hautamäki. "Action Space Shaping in Deep Reinforcement Learning". In: *CoRR* abs/2004.00980 (2020). arXiv: 2004.00980. URL: <https://arxiv.org/abs/2004.00980>.
- [KH20] Joanne Taery Kim und Sehoon Ha. "Observation Space Matters: Benchmark and Optimization Algorithm". In: *CoRR* abs/2011.00756 (2020). arXiv: 2011.00756. URL: <https://arxiv.org/abs/2011.00756>.

- [Kit22] Kitaird. *kitaird/master-thesis-learned-agent-strategies: Initial Release*. 2022. DOI: [10.5281/ZENODO.6582870](https://doi.org/10.5281/ZENODO.6582870). URL: <https://zenodo.org/record/6582870>.
- [Kla12] Michael Klauser. *Effizienz von Algorithmen - Eine Einführung*. <https://www.mathematik.uni-muenchen.de/~schotten/tqa/texte/EffizAlgo.pdf>. Zuletzt geprüft am 18.05.2022. 2012.
- [KToo] Vijay R. Konda und John N. Tsitsiklis. "Actor-Critic Algorithms". In: 2000, S. 1008–1014. URL: <http://papers.nips.cc/paper/1786-actor-critic-algorithms>.
- [KCC13] Petar Kormushev, Sylvain Calinon und Darwin Caldwell. "Reinforcement Learning in Robotics: Applications and Real-World Challenges". In: *Robotics* 2.3 (Juli 2013), S. 122–148. DOI: [10.3390/robotics2030122](https://doi.org/10.3390/robotics2030122). URL: <https://doi.org/10.3390/robotics2030122>.
- [KSH12] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Hrsg. von F. Pereira, C.J. Burges, L. Bottou und K.Q. Weinberger. Bd. 25. Curran Associates, Inc., 2012. URL: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.
- [Lau] Dr. Martin Lauer. *Reinforcement Learning - Learning in Multi Agent Environments*. https://ml.informatik.uni-freiburg.de/former/_media/teaching/ws1314/rl/ue13.pdf. Zuletzt geprüft am 20.05.2022.
- [Lia+21] Xingxing Liang, Yang Ma, Yanghe Feng und Zhong Liu. "PTR-PPO: Proximal Policy Optimization with Prioritized Trajectory Replay". In: *CoRR* abs/2112.03798 (2021). arXiv: [2112.03798](https://arxiv.org/abs/2112.03798). URL: <https://arxiv.org/abs/2112.03798>.
- [Lil+15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver und Daan Wierstra. *Continuous control with deep reinforcement learning*. 2015. DOI: [10.48550/ARXIV.1509.02971](https://doi.org/10.48550/ARXIV.1509.02971). URL: <https://arxiv.org/abs/1509.02971>.
- [Lit94] Michael L. Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *Machine Learning Proceedings 1994*. Hrsg. von William W. Cohen und Haym Hirsh. San Francisco (CA): Morgan Kaufmann, 1994, S. 157–163. ISBN: 978-1-55860-335-6. DOI: <https://doi.org/10.1016/B978-1-55860-335-6.50027-1>. URL: <https://www.sciencedirect.com/science/article/pii/B9781558603356500271>.
- [Lou22] Ismini Lourentzou. *Markov Games and Reinforcement Learning*. <https://isminoula.github.io/files/games.pdf>. Zuletzt geprüft am 20.05.2022. 2022.

- [Lyu+21] Xueguang Lyu, Yuchen Xiao, Brett Daley und Christopher Amato. "Contrasting Centralized and Decentralized Critics in Multi-Agent Reinforcement Learning". In: *CoRR* abs/2102.04402 (2021). arXiv: 2102.04402. URL: <https://arxiv.org/abs/2102.04402>.
- [Alp] "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), S. 484–489. DOI: 10.1038/nature16961. URL: <https://doi.org/10.1038/nature16961>.
- [Mni+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra und Martin A. Riedmiller. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: <http://arxiv.org/abs/1312.5602>.
- [Mni+15] Volodymyr Mnih u. a. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), S. 529–533. DOI: 10.1038/nature14236. URL: <https://doi.org/10.1038/nature14236>.
- [MA17] Igor Mordatch und Pieter Abbeel. "Emergence of Grounded Compositional Language in Multi-Agent Populations". In: *CoRR* abs/1703.04908 (2017). arXiv: 1703.04908. URL: <http://arxiv.org/abs/1703.04908>.
- [Nas51] John F. Nash. "Non-cooperative games". In: *Annals of Mathematics* 54 (1951), S. 286–295.
- [Neu28] J. v. Neumann. "Zur Theorie der Gesellschaftsspiele". In: *Mathematische Annalen* 100.1 (Dez. 1928), S. 295–320. DOI: 10.1007/bf01448847. URL: <https://doi.org/10.1007/bf01448847>.
- [NHR99] Andrew Y. Ng, Daishi Harada und Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *In Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann, 1999, S. 278–287.
- [Ope22] OpenAI. *Gym Documentation - API*. Zuletzt geprüft am 16.05.2022. 2022. URL: <https://www.gymlibrary.ml/content/api/>.
- [Ope+18] OpenAI u. a. *Learning Dexterous In-Hand Manipulation*. 2018. DOI: 10.48550/ARXIV.1808.00177. URL: <https://arxiv.org/abs/1808.00177>.
- [Ope+19a] OpenAI u. a. *Dota 2 with Large Scale Deep Reinforcement Learning*. 2019. DOI: 10.48550/ARXIV.1912.06680. URL: <https://arxiv.org/abs/1912.06680>.
- [Ope+19b] OpenAI u. a. *Solving Rubik's Cube with a Robot Hand*. 2019. DOI: 10.48550/ARXIV.1910.07113. URL: <https://arxiv.org/abs/1910.07113>.

- [Par+17] Fabio Pardo, Arash Tavakoli, Vitaly Levdkik und Petar Kormushev. "Time Limits in Reinforcement Learning". In: *CoRR* abs/1712.00378 (2017). arXiv: 1712 . 00378. URL: <http://arxiv.org/abs/1712.00378>.
- [Par20] Simone Parisi. "Reinforcement Learning with Sparse and Multiple Rewards". Diss. 2020. DOI: 10.25534/TUPRINTS-00011372. URL: <https://tuprints.ulb.tu-darmstadt.de/id/eprint/11372>.
- [PM17] David Poole und Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. 2. Aufl. Cambridge, UK: Cambridge University Press, 2017. ISBN: 978-0-521-51900-7. URL: <http://artint.info/2e/html/ArtInt2e.html>.
- [Pot+13] Mary C. Potter, Brad Wyble, Carl Erick Hagmann und Emily S. McCourt. "Detecting meaning in RSVP at 13 ms per picture". In: *Attention, Perception, & Psychophysics* 76.2 (Dez. 2013), S. 270–279. DOI: 10.3758/s13414-013-0605-z. URL: <https://doi.org/10.3758/s13414-013-0605-z>.
- [PNPI15] inc. Preferred Networks und inc. Preferred Infrastructure. *Tips and FAQs — Chainer 7.8.1 documentation*. <https://docs.chainer.org/en/stable/tips.html#mnist-example-does-not-converge-in-cpu-mode-on-mac-os-x>. (Accessed on 05/22/2022). 2015.
- [Raf20] Antonin Raffin. *RL Baselines3 Zoo*. <https://github.com/DLR-RM/rl-baselines3-zoo>. 2020.
- [Raf+21] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus und Noah Dormann. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), S. 1–8. URL: <http://jmlr.org/papers/v22/20-1364.html>.
- [Rao+20] Nirnai Rao, Elie Aljalbout, Axel Sauer und Sami Haddadin. *How to Make Deep RL Work in Practice*. 2020. arXiv: 2010 . 13083 [cs.LG].
- [RTP20] Daniele Reda, Tianxin Tao und Michiel van de Panne. "Learning to Locomote: Understanding How Environment Design Matters for Deep Reinforcement Learning". In: *CoRR* abs/2010.04304 (2020). arXiv: 2010 . 04304. URL: <https://arxiv.org/abs/2010.04304>.
- [Rie+09] Martin Riedmiller, Thomas Gabel, Roland Hafner und Sascha Lange. "Reinforcement learning for robot soccer". In: *Autonomous Robots* 27.1 (Mai 2009), S. 55–73. DOI: 10.1007/s10514-009-9120-4. URL: <https://doi.org/10.1007/s10514-009-9120-4>.
- [RS14] Kevin Wayne Robert Sedgewick. *Mergesort and Quicksort*. <https://www.cs.princeton.edu/courses/archive/spr14/cos226/lectures/22Mergesort.pdf>. Zuletzt geprüft am 18.05.2022. 2014.

- [Roh+21] Tobias Rohrer, Ludwig Samuel, Adriatik Gashi, Gunter Grieser und Elke Hergenröther. "Foosball Table Goalkeeper Automation Using Reinforcement Learning". In: *Proceedings of the LWDA 2021 Workshops: FGWM, KDML, FGWI-BIA, and FGIR, Online, September 1-3, 2021*. Hrsg. von Thomas Seidl, Michael Fromm und Sandra Obermeier. Bd. 2993. CEUR Workshop Proceedings. CEUR-WS.org, 2021, S. 173–182. URL: <http://ceur-ws.org/Vol-2993/paper-17.pdf>.
- [Roj13] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.
- [Saj18] Sajil N A. *Dynamic programming*. 2018. doi: [10.13140/RG.2.2.20186.39368](https://doi.org/10.13140/RG.2.2.20186.39368). URL: <http://rgdoi.net/10.13140/RG.2.2.20186.39368>.
- [Sam+19] Mikayel Samvelyan, Tabish Rashid, Christian Schröder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chi-Man Hung, Philip H. S. Torr, Jakob N. Foerster und Shimon Whiteson. "The StarCraft Multi-Agent Challenge". In: *CoRR* abs/1902.04043 (2019). arXiv: [1902.04043](https://arxiv.org/abs/1902.04043). URL: <http://arxiv.org/abs/1902.04043>.
- [Sch+16] Tom Schaul, John Quan, Ioannis Antonoglou und David Silver. "Prioritized Experience Replay". In: *CoRR* abs/1511.05952 (2016).
- [Sch+15a] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan und Pieter Abbeel. "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477 (2015). arXiv: [1502.05477](https://arxiv.org/abs/1502.05477). URL: <http://arxiv.org/abs/1502.05477>.
- [Sch+15b] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan und Pieter Abbeel. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015. doi: [10.48550/ARXIV.1506.02438](https://doi.org/10.48550/ARXIV.1506.02438). URL: <https://arxiv.org/abs/1506.02438>.
- [Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford und Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: [1707.06347](https://arxiv.org/abs/1707.06347). URL: <http://arxiv.org/abs/1707.06347>.
- [SSBD14] Shai Shalev-Shwartz und Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge: Cambridge University Press, 2014. ISBN: 978-1-107-05713-5.
- [Sil15] David Silver. *Lectures on Reinforcement Learning*. URL: <https://www.davidsilver.uk/teaching/>. 2015.
- [Sil+17a] David Silver u. a. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *CoRR* abs/1712.01815 (2017). arXiv: [1712.01815](https://arxiv.org/abs/1712.01815). URL: <http://arxiv.org/abs/1712.01815>.

- [Sil+17b] David Silver u. a. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), S. 354–359. DOI: [10 . 1038/nature24270](https://doi.org/10.1038/nature24270). URL: <https://doi.org/10.1038/nature24270>.
- [Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever und Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), S. 1929–1958. URL: [http : //jmlr.org/papers/v15/srivastava14a.html](http://jmlr.org/papers/v15/srivastava14a.html).
- [Sut88] Richard S. Sutton. "Learning to predict by the methods of temporal differences". In: *Machine Learning* 3.1 (Aug. 1988), S. 9–44. DOI: [10 . 1007 /bf00115009](https://doi.org/10.1007/bf00115009). URL: [https : //doi . org /10 . 1007 /bf00115009](https://doi.org/10.1007/bf00115009).
- [SB98] Richard S. Sutton und Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN: 978-0-262-19398-6. URL: <http://incompleteideas.net/book/first/ebook/node32.html>.
- [SB15] Richard S. Sutton und Andrew G. Barto. *Reinforcement Learning: An Introduction*. <https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf>. Zuletzt geprüft am 17.05.2022. 2015.
- [SB18] Richard S. Sutton und Andrew G. Barto. *Reinforcement Learning - An Introduction*. Cambridge: MIT Press, 2018. ISBN: 978-0-262-03924-6. URL: <http://incompleteideas.net/book/RLbook2020.pdf>.
- [TA19] Yunhao Tang und Shipra Agrawal. "Discretizing Continuous Action Space for On-Policy Optimization". In: *CoRR* abs/1901.10500 (2019). arXiv: [1901 . 10500](https://arxiv.org/abs/1901.10500). URL: [http : //arxiv . org /abs /1901 . 10500](https://arxiv.org/abs/1901.10500).
- [Tas+18] Yuval Tassa u. a. *DeepMind Control Suite*. 2018. DOI: [10 . 48550 / ARXIV . 1801 . 00690](https://doi.org/10.48550/ARXIV.1801.00690). URL: <https://arxiv.org/abs/1801.00690>.
- [Tea19] Chainer Team. *Chainer/CuPy v7 release and Future of Chainer*. [https://chainer.org/announcement/2019/12/05/released - v7.html](https://chainer.org/announcement/2019/12/05/released-v7.html). Zuletzt geprüft am 05.05.2022. Dez. 2019.
- [Teca] Unity Technologies. *Unity Documentation - Continuous collision detection (CCD)*. English. Zuletzt geprüft am 16.05.2022. URL: [https : //docs.unity3d.com /2020 . 3 /Documentation /Manual /ContinuousCollisionDetection.html](https://docs.unity3d.com/2020.3/Documentation/Manual/ContinuousCollisionDetection.html).
- [Tecb] Unity Technologies. *Unity Documentation - Introduction to Rigid-body physics*. English. Zuletzt geprüft am 16.05.2022. URL: [https : //docs.unity3d.com /2020 . 3 /Documentation /Manual /RigidbodiesOverview . html](https://docs.unity3d.com/2020.3/Documentation/Manual/RigidbodiesOverview.html).

- [Tecc] Unity Technologies. *Unity Documentation - Kinematic Rigidbody Collider*. English. Zuletzt geprüft am 16.05.2022. URL: <https://docs.unity3d.com/2020.3/Documentation/Manual/CollidersOverview.html>.
- [Tecd] Unity Technologies. *Unity Documentation - Rigidbody component reference*. English. Zuletzt geprüft am 16.05.2022. URL: <https://docs.unity3d.com/2020.3/Documentation/Manual/class-Rigidbody.html>.
- [Tece] Unity Technologies. *Unity Documentation - Time*. English. Zuletzt geprüft am 17.05.2022. URL: <https://docs.unity3d.com/2020.3/Documentation/Manual/class-TimeManager.html>.
- [TS22] Steven Schlicker Ted Sundstrom. 1.04: *Velocity and Angular Velocity*. [https://batch.libretexts.org/print?url=https://math.libretexts.org/Bookshelves/Precalculus/Book%3A_Trigonometry_\(Sundstrom_and_Schlicker\)/01%3A_The_Trigonometric_Functions/1.04%3A_Velocity_and_Angular_Velocity.pdf](https://batch.libretexts.org/print?url=https://math.libretexts.org/Bookshelves/Precalculus/Book%3A_Trigonometry_(Sundstrom_and_Schlicker)/01%3A_The_Trigonometric_Functions/1.04%3A_Velocity_and_Angular_Velocity.pdf). Zuletzt geprüft am 22.05.2022. 2022.
- [Ten22] TensorFlow. *tf.nn.dropout* | TensorFlow Core v2.9.0. https://www.tensorflow.org/api_docs/python/tf/nn/dropout. Zuletzt geprüft am 17.05.2022. Mai 2022.
- [Tob+17] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba und Pieter Abbeel. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. 2017. DOI: [10.48550/ARXIV.1703.06907](https://doi.org/10.48550/ARXIV.1703.06907). URL: <https://arxiv.org/abs/1703.06907>.
- [Tun+20] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess und Yuval Tassa. "dm_control: Software and tasks for continuous control". In: *Software Impacts* 6 (2020), S. 100022. ISSN: 2665-9638. DOI: <https://doi.org/10.1016/j.simpa.2020.100022>. URL: <https://www.sciencedirect.com/science/article/pii/S2665963820300099>.
- [Vin+19a] Oriol Vinyals u.a. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>. 2019.
- [Vin+19b] Oriol Vinyals u.a. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (Okt. 2019), S. 350–354. DOI: [10.1038/s41586-019-1724-z](https://doi.org/10.1038/s41586-019-1724-z). URL: <https://doi.org/10.1038/s41586-019-1724-z>.
- [WU19] CHING-AN WU. "Investigation of Different Observation and Action Spaces for Reinforcement Learning on Reaching Tasks". Zuletzt geprüft am 17.05.2022. Magisterarb. School of Electrical Engineering und Computer Science, 2019, S. 72. URL: <https://>

- <kth.diva-portal.org/smash/get/diva2:1415901/FULLTEXT01.pdf>.
- [WN03] Thilo Weigel und Bernhard Nebel. "KiRo – An Autonomous Table Soccer Player". In: *RoboCup 2002: Robot Soccer World Cup VI*. Springer Berlin Heidelberg, 2003, S. 384–392. DOI: [10.1007/978-3-540-45135-8_34](https://doi.org/10.1007/978-3-540-45135-8_34). URL: https://doi.org/10.1007/978-3-540-45135-8_34.
- [Wen+21] Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Hang Su und Jun Zhu. "Tianshou: a Highly Modularized Deep Reinforcement Learning Library". In: *CoRR abs/2107.14171* (2021). arXiv: [2107.14171](https://arxiv.org/abs/2107.14171). URL: <https://arxiv.org/abs/2107.14171>.
- [WO12] Marco Wiering und Martijn van Otterlo, Hrsg. *Reinforcement Learning*. Springer Berlin Heidelberg, 2012. DOI: [10.1007/978-3-642-27645-3](https://doi.org/10.1007/978-3-642-27645-3). URL: <https://doi.org/10.1007/978-3-642-27645-3>.
- [Wil+22] Bennet Wilhelm, Martin Göbel, Philipp Zimmermann, Robin Hinz, Sebastian Schuch und Tizian Rettig. "Projekt Systementwicklung - KickerAI, Optimierung der Bildverarbeitung". 2022.
- [WGH21] Jungdam Won, Deepak Gopinath und Jessica Hodgins. "Control strategies for physically simulated characters performing two-player competitive sports". In: *ACM Transactions on Graphics* 40.4 (Aug. 2021), S. 1–11. DOI: [10.1145/3450626.3459761](https://doi.org/10.1145/3450626.3459761). URL: <https://doi.org/10.1145/3450626.3459761>.
- [Yu+21] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre M. Bayen und Yi Wu. "The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games". In: *CoRR abs/2103.01955* (2021). arXiv: [2103.01955](https://arxiv.org/abs/2103.01955). URL: <https://arxiv.org/abs/2103.01955>.
- [YZ20] Tong Yu und Hong Zhu. *Hyper-Parameter Optimization: A Review of Algorithms and Applications*. 2020. DOI: [10.48550/ARXIV.2003.05689](https://doi.org/10.48550/ARXIV.2003.05689). URL: <https://arxiv.org/abs/2003.05689>.
- [Yu18] Yang Yu. "Towards Sample Efficient Reinforcement Learning". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, Juli 2018, S. 5739–5743. DOI: [10.24963/ijcai.2018/820](https://doi.org/10.24963/ijcai.2018/820). URL: <https://doi.org/10.24963/ijcai.2018/820>.
- [Zmqa] *ZeroMQ - High-Water-Mark*. Zuletzt geprüft am 03.05.2022. 5/3/2022. URL: <http://api.zeromq.org/2-1:zmq-setsockopt>.
- [Zmqb] *ZeroMQ*. Zuletzt geprüft am 03.05.2022. 5/3/2022. URL: <https://zeromq.org/>.

- [ZNo7] Dapeng Zhang und Bernhard Nebel. "Learning a Table Soccer Robot a New Action Sequence by Observing and Imitating". In: Jan. 2007, S. 61–.
- [ZQW20] Wenshuai Zhao, Jorge Peña Queralta und Tomi Westerlund. "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey". In: *CoRR* abs/2009.13303 (2020). arXiv: 2009.13303. URL: <https://arxiv.org/abs/2009.13303>.
- [ZZP20] Yuanyi Zhong, Yuan Zhou und Jian Peng. *Efficient Competitive Self-Play Policy Optimization*. 2020. doi: 10.48550/ARXIV.2009.06086. URL: <https://arxiv.org/abs/2009.06086>.
- [fed07] Tom Yore International table soccer federation. *ITSF RULE BOOK*. International table soccer federation, 2007. URL: <https://www.tablesoccer.org/rules/documents/ITSFRulesEnglish.pdf>.