

Hochschule Heilbronn

Fakultät für Mechanik und Elektronik

# Weiterentwicklung einer Airhockey-Simulation und Training eines Reinforcement Learning Agents

im Masterstudiengang Automotive Systems Engineering

**Masterprojekt**

**Autor:** Rico Steinke  
MatNr. 196949  
[rsteinke@stud.hs-heilbronn.de](mailto:rsteinke@stud.hs-heilbronn.de)

**Version vom:** 13. Juni 2023

**1. Betreuer:** Prof. Dr. -Ing. Nicolaj Stache  
**2. Betreuer:** M. Eng. Pascal Graf

## Kurzfassung

Reinforcement Learning (RL) ist eine Methode des maschinellen Lernens, die darauf abzielt, intelligente Agents zu entwickeln, die durch Interaktion mit ihrer Umgebung lernen und sich verbessern. Es hat breite Anwendungen in verschiedenen Bereichen und ist besonders effektiv in komplexen, dynamischen Umgebungen. Im Rahmen dieser Arbeit wurde ein RL-Agent entwickelt und trainiert, um in dem Geschicklichkeitsspiel Airhockey gegen einen realen Spieler konkurrieren zu können. Zum Training wurde eine Airhockey-Simulation weiterentwickelt. Hierbei trägt die Integration der Multi-Joint Dynamics with Contact (MuJoCo) Physik-Engine zur Simulationsgenauigkeit bei. Die Implementierung eines Frameworks zur Anwendung von Domain-Randomization-Techniken erleichtert den Übergang des Reinforcement Learning (RL)-Agents vom simulierten zum realen Airhockey-Tisch. Das bestehende Reinforcement Learning (RL)-Framework der Hochschule Heilbronn, Modular Reinforcement Learning (MRL), wurde optimiert, um den Trainingsprozess und die Auswertung der Trainingsergebnisse zu verbessern. Zur Verbesserung der Bewertung der relativen Spielstärke des Agents wurde der Glicko2-Algorithmus eingeführt. Der Soft Actor-Critic (SAC)-Algorithmus wurde zum Training des Agents verwendet. Die Trainingsergebnisse zeigten nach 72 Stunden fortgeschrittene Fähigkeiten in der Spielstrategie und der Fähigkeit, die Puck-Trajektorie zu antizipieren. Um das Domain-Gap zu überbrücken, wurden Domain-Randomization-Methoden implementiert. Dies führte zu verbesserten Anpassungsfähigkeiten des Agents an veränderte Umgebungsparameter und erhöhte die Robustheit und Generalisierbarkeit des Agents. Das Training unter Anwendung von Domain-Randomization-Techniken führte auch zu einer Verbesserung der Spielstrategie der künstlichen Intelligenz.

## Abstract

Reinforcement Learning (RL) is a machine learning method aimed at developing intelligent agents that learn and improve through interaction with their environment. It has broad applications in various domains and is particularly effective in complex, dynamic environments. In this study, an RL agent was developed and trained to compete in the skill game of air hockey against a real player. An air hockey simulation was further developed for training purposes. The integration of the Multi-Joint Dynamics with Contact (MuJoCo) physics engine contributes to the accuracy of the simulation. The implementation of a framework for applying domain randomization techniques facilitates the transition of the Reinforcement Learning (RL) agent from the simulated to the real air hockey table. The existing Reinforcement Learning (RL) framework of Heilbronn University, Modular Reinforcement Learning (MRL), was optimized to improve the training process and the evaluation of the training results. To enhance the evaluation of the relative playing strength of the agent, the Glicko2 algorithm was introduced. The Soft Actor-Critic (SAC) algorithm was used to train the agent. The training results demonstrated advanced capabilities in game strategy and the ability to anticipate the puck trajectory after 72 hours. To bridge the domain gap, domain randomization methods were implemented. This led to improved adaptability of the agent to changed environmental parameters and increased the robustness and generalizability of the agent. Training using domain randomization techniques also resulted in an improvement of the game strategy of the artificial intelligence.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>iii</b>
<b>Tabellenverzeichnis</b>	<b>iv</b>
<b>Abkürzungsverzeichnis</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 Stand der Technik</b>	<b>3</b>
2.1 Grundlagen des Reinforcement Learning (RL) . . . . .	3
2.1.1 Die zentralen Elemente des Reinforcement Learning . . . . .	4
2.1.2 Grundlagen des Deep Reinforcement Learning (DRL) . . . . .	5
2.1.3 Soft Actor-Critic (SAC)-Algorithmus . . . . .	5
2.1.4 Domain Randomization . . . . .	6
2.2 Modular Reinforcement Learning (MRL) Framework . . . . .	7
2.3 Airhockey-Simulation . . . . .	8
2.4 Multi-Joint Dynamics with Contact (MuJoCo) . . . . .	9
2.5 Rating-Algorithmen . . . . .	10
2.6 Bewertung von RL-Agents . . . . .	11
<b>3 Weiterentwicklung der Airhockey-Simulation</b>	<b>13</b>
3.1 Integration der Physik-Engine Multi-Joint Dynamics with Contact (MuJoCo) . . . . .	13
3.2 Implementierung von Domain-Randomization-Techniken . . . . .	16
3.3 Visuelle Weiterentwicklung der Airhockey-Simulation . . . . .	19
3.4 Entwicklung von Methoden zur Evaluation von Airhockey-Agents . . . . .	21
<b>4 Weiterentwicklung des Modular Reinforcement Learning Framework</b>	<b>24</b>
4.1 Algorithmus zum Bewerten von RL-Agents . . . . .	24
4.1.1 Auswahl und Anpassung eines Rating-Algorithmus . . . . .	24
4.1.2 Implementierung und Integration des Glicko2-Algorithmus . . . . .	26
4.2 Hinzufügen eines Command-Line Interface (CLI) . . . . .	28
4.3 Implementierung von grafischen Darstellungen für Trainings- und Rating-Daten	28
4.4 Exportieren von Reward- und Observation-Zusammensetzung . . . . .	30
<b>5 Training des Reinforcement Learning (RL)-Agents</b>	<b>31</b>
5.1 Zusammensetzung der Reward-Struktur . . . . .	31
5.2 Hyperparameter . . . . .	32
5.3 Observations . . . . .	33
<b>6 Evaluation der Ergebnisse</b>	<b>35</b>
<b>7 Zusammenfassung</b>	<b>53</b>
<b>8 Ausblick</b>	<b>55</b>
<b>Literaturverzeichnis</b>	<b>57</b>
<b>Anhang</b>	<b>62</b>

## Abbildungsverzeichnis

1	Eine Übersicht zu den drei Bereichen des Airhockey-Projekts. . . . .	1
2	Die Interaktion zwischen Agent und Umgebung im Reinforcement Learning [1]. . . . .	3
3	Interaktion zwischen Actor, Critic und Umgebung beim SAC-Algorithmus in der Airhockey-Umgebung. . . . .	6
4	Der visuelle Stand der Technik der Airhockey-Simulation. . . . .	9
5	Berechnung der Mausposition in Weltkoordinaten. . . . .	14
6	Darstellung der Funktionsweise des Arrive-Steering-Behaviour-Algorithmus nach Lima [2]. . . . .	15
7	Struktur der MuJoCo-Objekte in der Unity-Simulation am Beispiel des Pushers. . . . .	18
8	Einstellungsmöglichkeiten der Domain Randomization für Umgebungsvariablen im Unity-Editor. . . . .	19
9	Programmablaufplan der Verzögerung der Action-Randomization. . . . .	20
10	Airhockey-Simulation mit visuellen Anpassungen. . . . .	21
11	Beispiel einer Trajektorie-Visualisierung für Airhockey-Agents. . . . .	22
12	Beispiel einer Heatmap für die Darstellung in welcher Häufigkeit Airhockey-Agents sich in bestimmten Positionen befinden. . . . .	23
13	High-Level-Übersicht der Übermittlung des Spielergebnisses von der Simulation zum Modular Reinforcement Learning (MRL) über einen Side-Channel. . . . .	27
14	Befehlszeilenparameterübersicht für das Modular Reinforcement Learning (MRL)-Framework. . . . .	29
15	Beispiel für ein Plot der Reward-Entwicklung beim Training. . . . .	29
16	Beispiel für ein Plot der Rating-Entwicklung in einem Turnier mit drei Agents. . . . .	30
17	Actor-Netzwerk (links) und Critic-Netzwerk (rechts). Jedes Netzwerk besteht aus einem Input-Layer, Fully-Connected (Dense) Layern mit Scaled Exponential Linear Unit (SELU)-Aktivierungsfunktionen [3] und einem rekurrenten Long Short-Term Memory (LSTM)-Layer [4]. . . . .	33
18	Entwicklung des ersten Trainings eines Airhockey-Agents ohne vortrainierten Agent. . . . .	37
19	Heatmap und Trajektorie des Agents (AgentFromScratch) nach 24 Trainingsstunden. . . . .	38
20	Heatmap und Trajektorie des Agents (AgentFromScratch) nach 48 Trainingsstunden. . . . .	39
21	Heatmap und Trajektorie des Agents (AgentFromScratch) nach 72 Trainingsstunden. . . . .	39
22	Entwicklung des Trainings zur Feinabstimmung mit vortrainierter Policy. . . . .	40
23	Glicko2-Ratings für ein Round-Robin-Turnier mit dem AgentFromScratch nach 24, 48 und 72 Trainingsstunden und dem AgentFromScratchContinued nach 24 und 48 Stunden. . . . .	42
24	Entwicklung des Trainings mit Domain-Randomization-Techniken. . . . .	44
25	Trajektorie-Beispiele für einen Agent mit (AgentDomainRandomized) und ohne (AgentFromScratch) Training mit Domain-Randomization-Techniken in einer randomisierten Simulationsumgebung. . . . .	45

26	Heatmap für einen Agent trainiert mit Domain-Randomization-Techniken (AgentDomainRandomized) in einer randomisierten Simulationsumgebung im Spiel gegen sich selbst und einen Agent ohne Training mit Domain-Randomization (AgentFromScratch). . . . .	45
27	Entwicklung des Trainings zur Feinabstimmung mit vortrainierter Policy und Domain-Randomization-Techniken. . . . .	47
28	Heatmap für die Weiterentwicklung eines Agents trainiert mit Domain-Randomization-Techniken (AgentDomainRandomizedContinued) in einer randomisierten Simulationsumgebung im Spiel gegen sich selbst, seine vorherige Entwicklungsstufe (AgentDomainRandomized) und einen Agent ohne Training mit Domain Randomization (AgentFromScratch). . . . .	48
29	Trajektorie-Beispiele für einen verbesserten Agent im Training mit Domain-Randomization-Techniken (AgentDomainRandomizedContinued) in einer randomisierten Simulationsumgebung. . . . .	48
30	Glicko2-Ratings für ein Round-Robin-Turnier mit dem AgentFromScratch nach 72 Trainingsstunden, dem AgentFromScratchContinued nach 24 bzw. 48 Stunden, dem AgentDomainRandomized und AgentDomainRandomizedContinued. .	50
31	Entwicklung erfolgreicher Trainingsläufe im Vergleich mit dem AgentFromScratch (Agent 4 in Grafik). . . . .	52

## Tabellenverzeichnis

1	Beispiel für ein Round-Robin-Turnier mit vier Agents. . . . .	28
2	Beispiel für ein Center-Player-Turnier mit vier Agents. . . . .	28
3	Trainingsparameter für ein Training eines Airhockey-Agents ohne vortrainierten Agent. . . . .	36
4	Trainingsparameter für die Feinabstimmung eines Airhockey-Agents mit vortrainiertem Agent. . . . .	40
5	Trainingsparameter für ein Training eines Airhockey-Agents (AgentDomainRandomized) mit vortrainiertem Agent (AgentFromScratch) und Domain-Randomization-Techniken. . . . .	43
6	Trainingsparameter für ein Training eines Airhockey-Agents mit vortrainiertem Agent (AgentDomainRandomized) und Domain-Randomization-Techniken. .	46
7	Trainingsparameter erfolgreicher Trainingsläufe. . . . .	51

## Abkürzungsverzeichnis

**CAPS** Computer Accuracy and Precision Score

**CLI** Command-Line Interface

**CPU** Central Processing Unit

**CQL** Conservative Q-Learning

**CSV** Comma-Separated Values

**DDPG** Deep Deterministic Policy Gradient

**DL** Deep Learning

**DNN** Deep Neural Network

**DQN** Deep Q-Network

**DRL** Deep Reinforcement Learning

**FIFA** Fédération Internationale de Football Association

**FIFO** First In First Out

**GUI** Graphical User Interface

**HHN** Hochschule Heilbronn

**ICM** Intrinsic Curiosity Module

**KDE** Kernel Density Estimate

**KI** Künstliche Intelligenz

**LSTM** Long Short-Term Memory

**ML-Agents** Unity Machine Learning Agents Toolkit

**MoV** Margin of Victory

**MuJoCo** Multi-Joint Dynamics with Contact

**MRL** Modular Reinforcement Learning

**PER** Prioritized Experience Replay

**R2D2** Recurrent Replay Distributed Deep Q-Network (DQN)

**RD** rating deviation

**RL** Reinforcement Learning

**RND** Random Network Distillation

**SAC** Soft Actor-Critic

**SELU** Scaled Exponential Linear Unit

**TD3** Twin Delayed Deep Deterministic Policy Gradient

**YAML** YAML Ain't Markup Language

# 1 Einleitung

An der Hochschule Heilbronn (HHN) wird studiengangsübergreifend ein Airhockey-Tisch mit Sensorik und Aktorik aufgerüstet mit dem Ziel gegen einen menschlichen Spieler zu spielen. Der Airhockey-Tisch soll als Demonstrator dienen, um die Möglichkeiten von moderner Kinematik im Zusammenspiel mit künstlicher Intelligenz aufzuzeigen. Ziel ist es, dass die Künstliche Intelligenz (KI) zur Erkennung von Spielsituationen und zur Eigenentwicklung von Spielstrategien fähig ist. Hierfür wird in einer Simulation des Airhockey-Tisches mittels Reinforcement Learning (RL) ein KI-Modell trainiert. Airhockey ist ein anspruchsvolles und hochdynamisches Spiel, das schnelle Entscheidungen und präzise Bewegungen erfordert. Diese Herausforderungen machen es zu einem idealen Testfeld für die Anwendung von RL-Techniken, um leistungsstarke Agenten zu entwickeln, die menschenähnliche Fähigkeiten im Spiel demonstrieren können.

Reinforcement Learning (RL) ist ein maschinelles Lernverfahren, bei dem ein Agent durch Interaktion mit einer Umgebung lernt, welche Aktionen er in verschiedenen Situationen ausführen soll, um eine Belohnung zu maximieren. Der Agent trifft Entscheidungen auf der Grundlage von Rückmeldungen in Form von Belohnungen oder Bestrafungen, die er von der Umgebung erhält.

Das Airhockey-Projekt ist in drei Kategorien unterteilt. Die Extraktion von Spielinformationen, wie beispielsweise die Position des Pucks und der Pusher, erfolgt mittels Bildverarbeitungsmethoden. Die zweite Kategorie ist die Entwicklung der Spieleintelligenz, die auf Basis der extrahierten Informationen die Aktionen des Pushers bestimmen soll. Der dritte Bereich ist die Entwicklung der Hardware, die die Entscheidungen der Spieleintelligenz umsetzt und den von der KI gesteuerten Pusher bewegt. Abbildung 1 zeigt die zuvor beschriebenen Bereiche in einer Übersicht.

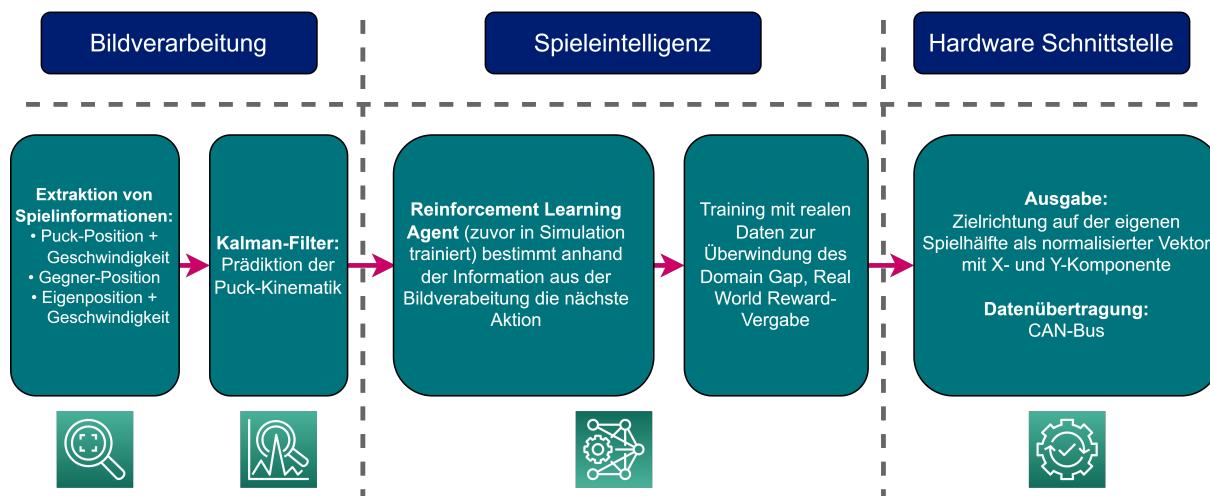


Abbildung 1: Eine Übersicht zu den drei Bereichen des Airhockey-Projekts.

Die Entwicklung von Bildverarbeitungsmethoden und der Hardware-Steuerung ist nicht Teil dieser Arbeit. Im Rahmen dieser Arbeit soll die Spieleintelligenz für das Airhockey entwickelt werden. Dies erfolgt indem:

- Die bestehende Airhockey-Simulation optimiert wird, um die realen physikalischen Gegebenheiten des Tisches möglichst genau abzubilden.
- Ein bestehendes RL-Framework um Funktionen erweitert wird, sowie gegebenenfalls die darin implementierten Algorithmen für das RL verbessert werden.
- In der Airhockey-Simulation das Training von RL-Agents erfolgt.
- Mit Hilfe von Domain-Randomization-Techniken Unterschiede zwischen Simulation und realem Tisch, der Domain-Gap, ausgeglichen werden sollen. Somit soll ein erfolgreicher Transfer von Simulation zum realen Airhockey-Tisch gewährleistet werden.

Das Ziel des Trainings ist es einen Agent zu trainieren, der beim Airhockey auftretende Spielsituationen intelligent lösen kann. Die Airhockey-KI soll gegen einen menschlichen Gegenspieler antreten können und in der Lage sein diesen zu besiegen.

Diese Arbeit ist wie folgt aufgebaut: Kapitel 2 beschreibt den Stand der Technik im Bereich des RL und der Airhockey-Simulation, wobei die zentralen Begriffe des RL erläutert werden. In Kapitel 3 wird aufgezeigt, wie die Airhockey-Simulation im Vergleich zum Stand der Technik technisch und visuell weiterentwickelt wurde. Anschließend wird in Kapitel 4 die Weiterentwicklung des HHN-RL-Frameworks beschrieben. Dabei wird die Implementierung von Methoden zum Vereinfachen des Trainingsprozesses und zur Evaluation der Trainingsergebnisse vorgestellt. Kapitel 5 zeigt die zum Training des RL-Agents verwendeten Methoden und Parameter. In Kapitel 6 werden die durchgeführten Trainings und deren Ergebnisse vorgestellt und analysiert. Abschließend wird in Kapitel 7 ein Fazit gezogen und in Kapitel 8 ein Ausblick gewagt.

## 2 Stand der Technik

Der Stand der Technik soll einen Überblick geben mit welchen Ansätzen Reinforcement Learning (RL)-Probleme gelöst werden können. Dazu werden zunächst die wichtigsten Konzepte des RL erläutert. Zusätzlich wird der gegenwärtige Entwicklungsstand der Airhockey-Simulation und des RL-Framework *Modular Reinforcement Learning (MRL)* der Hochschule Heilbronn (HHN) dargestellt.

### 2.1 Grundlagen des Reinforcement Learning (RL)

Reinforcement Learning (RL) ist eine Art von maschinellem Lernen, bei dem ein Agent durch Interaktion mit einer Umgebung lernen soll, welche Aktionen in verschiedenen Situationen am besten sind. Das Ziel dabei ist es, eine numerische Belohnung – den *Reward* – zu maximieren. Der lernende Agent bekommt keine Informationen, welche Aktionen richtig oder falsch sind. Stattdessen muss er herausfinden, welche Aktionen er ausführen muss, um die Belohnung zu maximieren [5].

Der Lernprozess im Reinforcement Learning ist, wie in Abbildung 2 dargestellt, iterativ. In jedem Schritt erhält der Agent eine Darstellung des Umgebungszustands und wählt basierend darauf eine Aktion. Die Umgebung wechselt dann in einen neuen Zustand und gibt dem Agents einen Reward basierend auf der durchgeführten Aktion [1, 5].

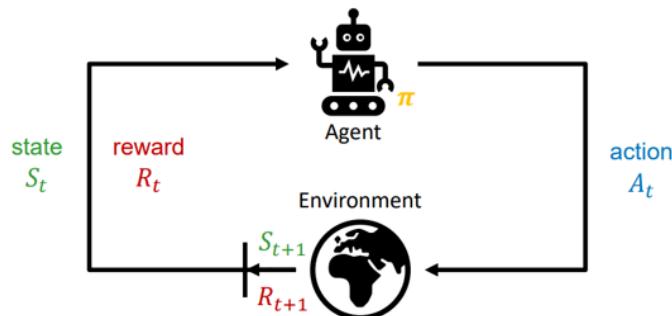


Abbildung 2: Die Interaktion zwischen Agent und Umgebung im Reinforcement Learning [1].

Das Ziel des Lernprozesses besteht darin, eine optimale Strategie zu finden, die den kumulativen Reward maximiert. Im RL besteht eine Herausforderung darin, ein Gleichgewicht zwischen Exploration und Exploitation zu finden. Der Agent muss Aktionen bevorzugen, die in der Vergangenheit erfolgreich waren, um eine hohe Belohnung zu erzielen. Aber um solche Aktionen zu entdecken, muss er auch neue Aktionen ausprobieren. Der Agent muss also sowohl Exploitation als auch Exploration betreiben, um optimal zu handeln. Beide Ansätze können jedoch nicht ausschließlich verfolgt werden, ohne dass der Agent bei der Aufgabe scheitert. Er muss

eine Vielzahl von Aktionen ausprobieren und schrittweise diejenigen bevorzugen, die am besten erscheinen [5].

Zusammengefasst ist RL ein Ansatz zur Automatisierung des zielgerichteten Lernens und der Entscheidungsfindung durch Computer. Im Gegensatz zu anderen Ansätzen betont es das Lernen eines Agents aus direkter Interaktion mit seiner Umgebung, ohne auf beispielhafte Überwachung oder vollständige Modelle der Umgebung angewiesen zu sein [5].

### 2.1.1 Die zentralen Elemente des Reinforcement Learning

Neben dem Agent und seiner Umgebung gibt es weitere zentrale Elemente beim Reinforcement Learning (RL), die den aktuellen Forschungsstand im RL-Bereich wiederspiegeln. Zudem sind diese Konzepte zum Verständnis der Ansätze und Lösungen dieser Arbeit notwendig. Die zentralen Elemente sind:

- *Policy*  $\pi$ : Policy definiert das Verhalten des Agents zu einer gegebenen Zeit  $t$ . Dass heißt eine Policy ist eine Zuordnung von wahrgenommenen Zuständen der Umgebung zu den Aktionen, die ausgeführt werden sollen, wenn sich der Agent in diesen Zuständen befindet [1].
- *Zustand (State  $S_t$ )*: Der Zustand der Umgebung zu einem bestimmten Zeitpunkt. Der Zustand enthält Informationen über die Kinematik eines Agents oder die Positionen von Objekten in der Umgebung. Ein Zustand entspricht einer vollständigen Beschreibung der Umgebung [6].
- *Observation*: Eine Observation ist eine partielle Darstellung des States, welche eventuell nur eine Teilmenge der Informationen enthält. States und Observations werden in der Regel als reellwertiger Vektor, Matrix oder Tensor dargestellt [6].
- *Aktion (Action  $A_t$ )*: Die Aktionen, die der Agent in einem gegebenen Zustand ausführen kann. Unterschiedliche Umgebungen erlauben unterschiedliche Arten von Aktionen. Die Menge aller gültigen Aktionen in einer gegebenen Umgebung wird als *Action Space* bezeichnet. Sind für den Agent nur eine endliche Anzahl an Zügen verfügbar ist der Action Space diskret. Steuert der Agent beispielsweise einen Roboter in einer physischen Umgebung, hat er einen kontinuierlichen Action Space mit reellwertigen Vektoren [6].
- *Reward-Signal*: Ein Reward-Signal definiert das Ziel eines RL-Problems. In jedem Zeitschritt sendet die Umgebung an den RL-Agents eine einzige Zahl, die als *Reward* bezeichnet wird. Das alleinige Ziel des Agents ist es, die insgesamt erhaltenen Belohnungen

über einen längeren Zeitraum zu maximieren. Das Reward-Signal definiert somit, was für den Agents gute und schlechte Ereignisse sind [1].

- *Wertefunktion (Value Function)*: Während das Reward-Signal anzeigt, was unmittelbar gut ist, gibt die Wertefunktion an, was langfristig gut ist. Grob gesagt ist der Wert eines Zustands die Gesamtmenge an Reward, die ein Agent erwartet, in Zukunft von diesem Zustand aus zu sammeln [1, 5].

Der Zusammenhang zwischen Policy, Action, State und Reward ist in Abbildung 2 dargestellt. Die Policy gibt an, welche Aktion der Agent in einem gegebenen Zustand ausführen soll. Die Aktion wird ausgeführt und der Agent erhält eine Belohnung. Die Belohnung wird verwendet, um die Policy zu aktualisieren [1].

### 2.1.2 Grundlagen des Deep Reinforcement Learning (DRL)

Deep Reinforcement Learning (DRL) erweitert herkömmliches RL durch die Integration von Deep Learning (DL)-Techniken. Im Gegensatz zu konventionellen RL-Algorithmen, die Zustands- und Aktionswerte als diskrete Tabellen oder Funktionen speichern, verwenden DRL-Algorithmen tiefe neuronale Netzwerke (Deep Neural Network (DNN)), um eine kontinuierliche Funktion zu approximieren, die die Aktionen des Agents als Funktion des aktuellen Zustands voraussagt. Dies ermöglicht es, komplexe Entscheidungsprobleme zu lösen, die in herkömmlichen RL-Algorithmen schwer zu modellieren sind [7, 8].

### 2.1.3 Soft Actor-Critic (SAC)-Algorithmus

Der Soft Actor-Critic (SAC) ist ein DRL-Algorithmus, der sich besonders für Anwendungsfälle eignet, bei denen kontinuierliche Aktionen ausgeführt werden sollen. Dadurch ist der SAC-Algorithmus geeignet für die Steuerung eines Airhockey-Agents [6, 9].

Die Hauptkomponenten des SAC sind der *Actor* und der *Critic*. Der Actor repräsentiert die Policy des Agents. Er besteht aus einem DNN und schlägt auf Basis des aktuellen Zustands der Umgebung Aktionen vor. Der Critic schätzt den Wert der vom Actor vorgeschlagenen Aktionen. Er verwendet dazu die eine Q-Funktion, die den erwarteten kumulativen zukünftigen Belohnungen für einen gegebenen Zustand und einer Aktion entspricht. Der Critic ist ebenfalls ein neuronales Netzwerk, das trainiert wird, um die Q-Funktion zu approximieren [6, 9]. Abbildung 3 stellt die Interaktion zwischen Actor, Critic und Umgebung beim SAC dar.

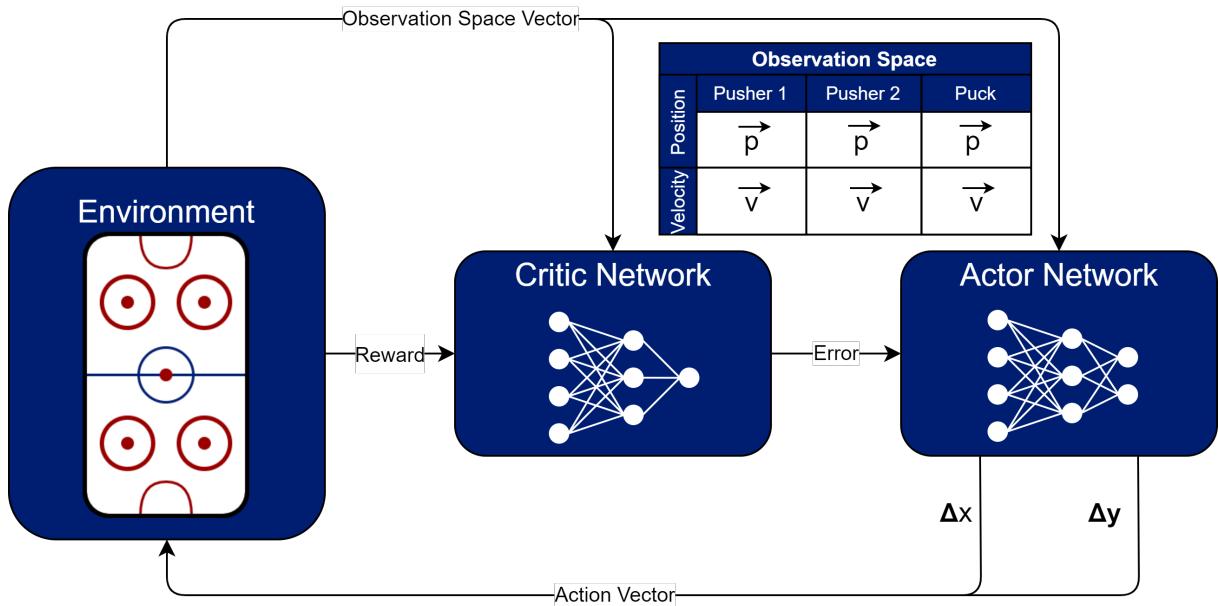


Abbildung 3: Interaktion zwischen Actor, Critic und Umgebung beim SAC-Algorithmus in der Airhockey-Umgebung.

Der SAC-Algorithmus wird in de Jesus et al [10] zur Navigation mobiler Roboter verwendet. Der Roboter erhält dabei eine Zielposition, deren erreichen mit einem positiven Reward verknüpft ist. Bestraft wird er für das Kollidieren mit Hindernissen.

Wong et al [11] verwenden den SAC-Algorithmus zur Steuerung eines zweiarmligen Roboters mit sieben Freiheitsgraden. Das Ziel dabei ist es eine Policy zu trainieren, die eine Bewegungsplanung durchführt. Dabei sollen Kollisionen der Arme, sowie Gelenkgrenzen und Singularitäten vermieden werden.

#### 2.1.4 Domain Randomization

Domain Randomization ist eine Technik, die im Reinforcement Learning (RL) verwendet wird, um Agents zu trainieren, die robust gegenüber Veränderungen in ihrer Umgebung sind. Aufgrund der Kosten für die Datenerfassung bei realen Umgebungen ist es notwendig, Modelle in einem Simulator zu trainieren, der theoretisch eine unendliche Menge an Daten liefert.

Beim Training in einer simulierten Umgebung entsteht eine Lücke zwischen dem Simulator und der realen Umgebung. Diese Lücke wird auch als *Reality Gap* oder *Domain Gap* bezeichnet. Der Domain Gap führt häufig zu Fehlern bei der Übertragung von in einer Simulation trainierten RL-Agents auf eine reale Umgebung. Der Gap entsteht durch eine Inkonsistenz zwischen physikalischen Parametern, wie z.B. Reibung, Masse und Dämpfung [12]. Außerdem wurde festgestellt, dass DRL-Agents dazu neigen Ungenauigkeiten des Simulators auszunutzen, um Verhaltensweisen zu realisieren, die in der realen Umgebung nicht möglich sind [13, 7, 14].

Domain-Randomization-Techniken wurden erfolgreich angewandt bei der Lokalisierung von Objekten, indem die reale Umgebung in einer Simulation nachgestellt und gerendert wird. In der Simulation werden die Bilder mit zufälligen Parametern wie Lichtverhältnissen, Farben und Textur generiert. Anschließend wird das DNN mit den simulierten Daten trainiert [15, 16].

Ein weiterer Bereich, indem Domain Randomization erfolgreich angewandt wurde, ist die Handhabung von Gegenständen durch Roboterhände. Hierbei werden zusätzlich zu der Randomisierung von simulierten Umgebungsaufnahmen auch physikalische Parameter und in der Simulation nicht modellierte Effekte, wie Verzögerungen und Ungenauigkeiten der Aktuatoren, berücksichtigt [17, 13].

Mehta et al [18] verwenden den Ansatz der *Active Domain Randomization*, bei welchem nach den informativsten Variationen der Simulationsumgebung innerhalb des Randomisierungsbereichs gesucht wird. Die Methode nutzt dabei Diskrepanzen von Policy-Rollouts in randomisierten und Referenz-Umgebungsinstanzen. Die Autoren stellen fest, dass häufiges Training in diesen Instanzen zu einer besseren Gesamtgeneralisierung des Agents führt.

Damken et al [19] nutzen einen *Curriculum-Learning*-Ansatz [20] für die Domain Randomization. Dabei wird die Komplexität der Umgebung schrittweise erhöht, beeinflusst durch die Performance des Agents. Die Autoren stellen fest, dass die Verwendung von Curriculum Learning die Trainingszeit verringert und stabilere Policies erzeugt.

Muratore et al [21] optimieren die Parameterverteilung der Domain Randomization während des Lernens auf Basis spärlicher Daten der Zielumgebung. Die Autoren nutzen *Bayes'sche Optimierung* um den Bereich der Parameter der Quellumgebung zu durchsuchen, sodass dies zu einer Policy führt, die reale Ziel maximiert und adaptive Verteilungen während des Trainings ermöglicht.

## 2.2 Modular Reinforcement Learning (MRL) Framework

Das Framework *Modular Reinforcement Learning (MRL)* implementiert eine Auswahl von DRL-Algorithmen und -Architekturen. Dazu gehören folgende Lernalgorithmen:

- *Deep Q-Network (DQN)* mit Erweiterungen für Double Learning, Dueling Architecture und Noisy Networks
- *Deep Deterministic Policy Gradient (DDPG)*
- *Conservative Q-Learning (CQL)* für Offline RL
- *Twin Delayed Deep Deterministic Policy Gradient (TD3)*

- *Soft Actor-Critic (SAC)* mit automatischer Temperaturanpassung

Neben der Implementierung von DRL-Algorithmen bietet das Framework zahlreiche Netzwerkarchitekturen für vektor- und bildbasierte Umgebungen. Außerdem nutzt das Framework Recurrent Replay Distributed DQN (R2D2) mit Burn-In zur Verbesserung der Stabilität und der Effizienz von DRL-Agents [22]. Des Weiteren wird der Prioritized Experience Replay (PER)-Algorithmus verwendet, um dem Agent zu ermöglichen Erfahrungen aus der Vergangenheit wiederzuverwenden [23]. Curriculum Learning unterstützt das Framework, um die Erfahrungen des Agents zu sortieren und damit die Effizienz des Lernprozesses zu verbessern [24]. Um dem in Kapitel 2.1 beschriebenen Konflikt zwischen Exploration und Exploitation gerecht zu werden, werden verschiedene Explorations-Algorithmen unterstützt. Dazu gehören Epsilon-Greedy, Intrinsic Curiosity Module (ICM) und Random Network Distillation (RND) [25, 26].

Für die Bereitstellung und Entwicklung von Simulationsumgebungen zum Training eines RL-Agents bietet das Framework Schnittstellen zu Unity und OpenAI Gym. OpenAI Gym ist eine Sammlung von Simulationsumgebungen, die für die Entwicklung und den Vergleich von RL-Agents verwendet werden können [27]. Unity ist eine Plattform für die Entwicklung von 3D-Spielen und Anwendungen. Zur Kommunikation mit Unity nutzt das Modular Reinforcement Learning (MRL) die Python-Schnittstelle des Unity Machine Learning Agents Toolkit (ML-Agents) [28].

## 2.3 Airhockey-Simulation

RL-Agents werden in Simulationen trainiert, da dies eine effektive Möglichkeit bietet eine große Anzahl von Interaktionen mit der Umgebung zu generieren. Darüber hinaus können in Simulationen Szenarien erstellt werden, die in der realen Welt nur schwer zu reproduzieren oder zu kontrollieren sind.

Die Airhockey-Simulation ist mit der Spieleentwicklungsplattform Unity entwickelt. Wie in Kapitel 2.2 beschrieben wird die Open-Source-Erweiterung Unity Machine Learning Agents Toolkit (ML-Agents) [28] genutzt zur Kommunikation mit dem Modular Reinforcement Learning (MRL)-Framework. Außerdem implementiert die Airhockey-Simulation die Basiskomponenten für das Training von RL-Agents. Dazu gehören Methoden zum Erfassen von Observations und Rewards, sowie das Ausführen von Actions. Als Physik-Engine wird *Nvidia PhysX* [29] verwendet. Dies entspricht der Standard-Engine für die Entwicklung von 3D-Spielen mit Unity.

Wie in Abbildung 4 dargestellt, ist ein Modell des realen Airhockey-Tischs in Unity importiert. Die Simulation ist aus der Top-Down-Perspektive spielbar. Die Pusher können per Tastaturein-

gabe gesteuert werden. Des Weiteren sind Collider und Skripte implementiert, die erkennen, wenn ein Tor erzielt wird.



Abbildung 4: Der visuelle Stand der Technik der Airhockey-Simulation.

## 2.4 Multi-Joint Dynamics with Contact (MuJoCo)

MuJoCo wurde speziell entwickelt, um komplexe, realistische Simulationen von Systemen in der Robotik und Biomechanik zu ermöglichen, insbesondere in Situationen, in denen Kontakte und Beschleunigungen eine wesentliche Rolle spielen. Das Design von MuJoCo beruht auf einem differenzierten Verfahren zur Vorwärtssubstitution, das es ermöglicht, Modelle mit komplexen kinematischen Strukturen, einschließlich baumartiger und geschlossener Ketten, zu simulieren [30].

Eines der zentralen Merkmale von MuJoCo ist seine Behandlung von Kontakten. MuJoCo modelliert Kontakte als weiche Kontakte mithilfe eines kontinuierlichen Modells, das eine weiche Kollisionsantwort bietet und gleichzeitig die dynamischen Eigenschaften der Interaktion beibehält. Diese Herangehensweise ermöglicht eine effiziente Simulation von Szenarien mit vielen Kontakten, wie sie in der praktischen Robotik häufig auftreten [30].

MuJoCo wird als primäre Umgebung für Benchmark-Aufgaben in RL-Frameworks wie *OpenAI's Gym* verwendet. Dort ermöglicht es dem Anwender, die Leistung von RL-Algorithmen unter realistischen physikalischen Bedingungen zu bewerten [31].

## 2.5 Rating-Algorithmen

Um die relative Fähigkeit eines Spielers zu berechnen, entwickelte Elo [32] das Elo-Rating. Im Elo-System wird jedem Spieler eine Elo-Zahl zugeordnet, welche die Spielstärke des Spielers repräsentiert. Die Elo-Zahl wird nach jedem Spiel aktualisiert, abhängig vom Ergebnis und der Spielstärke des Gegners. Elo berechnet zunächst, wie in Gleichung 1 dargestellt, den erwarteten Spieldurchgang  $E$  (hier für Spieler  $a$  berechnet).

$$E_a = \frac{1}{1 + 10^{\frac{R_b - R_a}{400}}} \quad (1)$$

Anschließend wird das Rating-Update  $R'$  basierend auf dem aktuellem Spieler-Rating  $R$  der erwarteten Gewinnwahrscheinlichkeit  $E$  und dem tatsächlichen Ergebnis  $S$  berechnet (siehe Gleichung 2). Die Konstante  $K$  ist ein Faktor, der die Anpassungsgeschwindigkeit des Ratings bestimmt. Je höher der Faktor  $K$  ist, desto schneller wird das Rating angepasst. Der K-Faktor wird in der Regel für Anfänger hoch gewählt, um die Anpassungsgeschwindigkeit für neue Spieler zu beschleunigen. Für erfahrene Spieler wird der K-Faktor niedriger gewählt, um die Stabilität des Ratings zu erhöhen [33].

$$R'_a = R_a + K(S_a - E_a) \quad (2)$$

Der Glicko-Algorithmus wurde als Weiterentwicklung des Elo-Systems entwickelt. Der Algorithmus bietet eine flexiblere und präzisere Methode zum Schätzen der Spielstärke von Spielern. Im Gegensatz zum Elo-System, das nur die Spielstärke eines Spielers berücksichtigt, berücksichtigt der Glicko-Algorithmus auch die Varianz der Spielstärke eines Spielers. Die Varianz der Spielstärke eines Spielers wird als *Rating-Abweichung* (*engl. rating deviation (RD)*) bezeichnet [34]. Die Weiterentwicklung des Glicko-Algorithmus, Glicko2, erweitert das System, um ein Maß für die Volatilität  $\sigma$  der Spielstärke eines Spielers. Die Volatilität gibt das Ausmaß der erwarteten Schwankungen in der Bewertung eines Spielers an. Die Volatilität ist beispielsweise hoch, wenn ein Spieler außergewöhnlich gute Ergebnisse erzielt, nach einer Periode mit stabilen Ergebnissen. Befindet sich der Spieler auf konstantem Level ist die Volatilität niedrig. Die RD gibt die Spielstärke eines Spielers als 95%-Konfidenzintervall an, dass den Bereich von plausiblen Werten für die tatsächliche Stärke des Spielers. Durch die Angabe des Intervalls wird der Algorithmus dem Fakt gerecht, dass das Rating eine Schätzung der unbekannten wahren Spielstärke ist. Der niedrigste Wert in dem Intervall ist das Rating des Spielers abzüglich ungefähr zwei mal die RD und der Höchstwert ist das Rating des Spielers plus zwei mal die RD. Die genaue Berechnung ist in Gleichung 3 dargestellt [35, 36]. Der vollständige Glicko2-Algorithmus ist in Anhang C dargestellt.

$$\text{Rating-Intervall} = [R - 1.96 \cdot \text{RD}, R + 1.96 \cdot \text{RD}] \quad (3)$$

Die Webseite *Chess.com* verwendet in Online-Schachspielen das Computer Accuracy and Precision Score (CAPS)-System. Das System vergleicht Züge eines menschlichen Spielers mit dem, was eine Schach-KI in derselben Position tun würde. Das Ziel des Computer Accuracy and Precision Score (CAPS) ist es eine Bewertung der Spielstärke auf Basis der Qualität der gespielten Züge zu liefern. Computer Accuracy and Precision Score (CAPS) bewertet die Spielstärke auf einer Skala von 0 bis 100, wobei 100 die höchste mögliche Bewertung ist, die bedeutet, dass alle Züge eines Spielers mit denen eines Computers übereinstimmen [37].

Der *TrueSkill*-Algorithmus ist ein von Herbrich et al [38] entwickeltes statistisches Bewertungssystem, das die Fähigkeiten von Spielern in Multiplayer-Online-Spielen schätzt. Der Algorithmus nutzt Bayes'sche Inferenz und das Gauß'sche Fehlermodell, um die Fähigkeiten von Spielern einzuschätzen und ihre Bewertungen zu aktualisieren, basierend auf den Ergebnissen der Spiele und den Bewertungen ihrer Gegner. In Teamspielen betrachtet er das gesamte Team als eine Einheit und schätzt die kombinierte Fähigkeit des Teams. Anschließend wird die Bewertung auf individuelle Ebene heruntergebrochen. *TrueSkill2* ist eine Weiterentwicklung von TrueSkill und bietet zusätzlich die Möglichkeit individuelle Spielstile und spezifische Spielmechaniken zu berücksichtigen, um eine präzisere Schätzung der Fähigkeiten eines Spielers zu ermöglichen [39].

## 2.6 Bewertung von RL-Agents

Die Spielstärke von *AlphaGo*, einer KI für das Brettspiel *Go*, wurde bewertet, indem die KI gegen verschiedene Go-Programme spielte. Diese Programme zum Vergleich mit der KI basieren nicht auf neuronalen Netzen, sondern auf *Monte-Carlo-Tree-Search*-Algorithmen. Bei der Evaluation wurde nur die Gewinnrate des AlphaGo-Agents berücksichtigt. Zusätzlich wurden fünf Spiele gegen einen professionellen Go-Spieler durchgeführt [40].

Zur Bewertung der Spielstärke von *AlphaZero* wurde das *Elo*-Rating herangezogen. Das Elo-Rating ist ein Algorithmus zur Bestimmung der relativen Spielstärke von Spielern in Nullsummenspielen mit zwei Spielern [32]. Das Rating wurde berechnet in den Spielen *Schach*, *Shogi* und *Go*. Im Schach und Shogi wurden führende Schach- bzw. Shogi-Programme als Gegner für den RL-Agents verwendet. Im Go wurden verschiedene Versionen von AlphaGo als Referenz verwendet und die Elo-Ratings berechnet [41].

Agents, welche für Jump-And-Run-Spiele oder Fahrzeug-Simulatoren trainiert wurden, wurden mit absoluten Metriken bewertet. Dazu wurde unter anderem die zurückgelegte Strecke im Level bzw. auf der Strecke gemessen [42, 43].

*OpenAI Five*, ein RL-System für das Echtzeitstrategiespiel *Dota 2*, wurde gegen menschliche Amateur- und Profispielern getestet. Gegen Amateure wurden mehrere tausend Spiele gespielt und die Gewinnrate des RL-Agents wurde berechnet. Während dem Training wurden die Agents gegen einen Pool aus Referenz-Agents mit einem angepassten TrueSkill-Algorithmus verglichen und bewertet [44].

### 3 Weiterentwicklung der Airhockey-Simulation

Ein Schwerpunkt dieser Arbeit liegt auf der Weiterentwicklung der Airhockey-Simulation in Bezug auf korrekte Physiksimulation der realen Umgebung bzw. der Entwicklung einer Simulationsumgebung zum Trainieren eines robusten, anpassungsfähigen RL-Agents. Hierfür wird zunächst in Kapitel 3.1 die Integration der Physik-Engine MuJoCo motiviert und dadurch notwendige Anpassungen beschrieben. Anschließend wird in Kapitel 3.2 dargestellt, wie die Airhockey-Simulation um ein Framework zur Domain Randomization erweitert. Darin werden Methoden zum Training eines RL-Agents mit erhöhter Generalisierungsfähigkeit vorgestellt und deren Implementierung beschrieben. Kapitel 3.3 beschreibt die visuelle Weiterentwicklung der Airhockey-Simulation. Abschließend wird in Kapitel 3.4 die Entwicklung von Methoden zur Evaluation der Airhockey-Agenten beschrieben. Um die Leistung und Effektivität der Agenten zu messen, ist es notwendig, geeignete Bewertungskriterien und -verfahren zu entwickeln. In diesem Abschnitt werden zwei Ansätze zur Evaluierung vorgestellt.

#### 3.1 Integration der Physik-Engine MuJoCo

MuJoCo ist eine Physik-Engine, die sich besonders auf die Simulation komplexer Kontaktodynamiken und Mechanismen mit mehreren Gelenken in verschiedenen Anwendungsbereichen fokussiert [30]. In Tassa et al. [45] vergleichen die Autoren mehrere Physik-Engines. Am besten bewertet werden dabei PhysX, die Standard-Physik-Engine in Unity, und die MuJoCo-Engine. Im Vergleich zu PhysX bietet MuJoCo eine vergleichbare, hohe Genauigkeit bei der Simulation von Multi-Joint-Dynamik. Bei der Simulation von Kontaktodynamiken wird PhysX übertroffen. Beim Vergleich der Leistungsfähigkeit, wird festgestellt, dass PhysX für Videospielanwendungen optimiert ist. Die MuJoCo-Engine hingegen ist stärker auf Genauigkeit und Stabilität bei der Simulation komplexer mechanischer Systeme ausgelegt [45].

Bei der Simulation der Airhockey-Umgebung wird Flexibilität und Effektivität bei der Handhabung von Kontaktodynamiken benötigt. Reibung zwischen Puck und Tisch, sowie das Kontaktverhalten von Puck und Banden bzw. Puck und Pusher sind Faktoren, die bei der Simulation berücksichtigt werden müssen. Zudem bietet MuJoCo Lösungen für die Simulation der Gelenke der Mechanik zur Steuerung des Pushers. Dies gilt unabhängig davon, ob der Pusher von einem Roboterarm oder einem Flächenportal gesteuert wird.

MuJoCo stellt ein Plug-In für Unity zur Verfügung, um Unity zur Laufzeit auf die MuJoCo-Physik-Engine zugreifen zu lassen. Aufgrund der zuvor genannten Simulationseigenschaften von MuJoCo eignet sich die Engine für die wissenschaftliche Forschung im Bereich RL. Da

mit dem Plug-In für Unity eine Integration in die bestehende Airhockey-Simulation möglich ist, wird die MuJoCo-Engine für die Simulation der Airhockey-Umgebung verwendet.

Die Hauptfunktion der Airhockey-Simulation ist die Bereitstellung einer Trainingsumgebung für RL-Agents. Zusätzlich dient die Simulation als Demonstrator für mögliche Anwendungsfälle von künstlicher Intelligenz. Neben dem Self-Play-Modus, soll es auch möglich sein in der Simulation gegen einen RL-Agenten zu spielen.

Als Bedienelement für den simulierten Pusher wird die Maus ausgewählt, da diese Art der Steuerung den Bewegungsabläufen beim Airhockey gleicht. Hierfür wird zunächst die Mausposition relativ zur Tischemebene bestimmt. Von der Kamera, die die Perspektive aufnimmt aus der die Simulation dargestellt wird, wird ein Strahl zur Bildebene berechnet. Der Schnittpunkt des Strahls mit der Ebene wird dynamisch mit der Mausposition in der Bildebene aktualisiert. Trifft der Strahl dabei auf die Tischemebene, kann daraus die Mausposition in Weltkoordinaten berechnet werden. Dadurch kann die Mausposition auf die Tischemebene projiziert werden. Abbildung 5 illustriert das zuvor beschriebene Vorgehen.

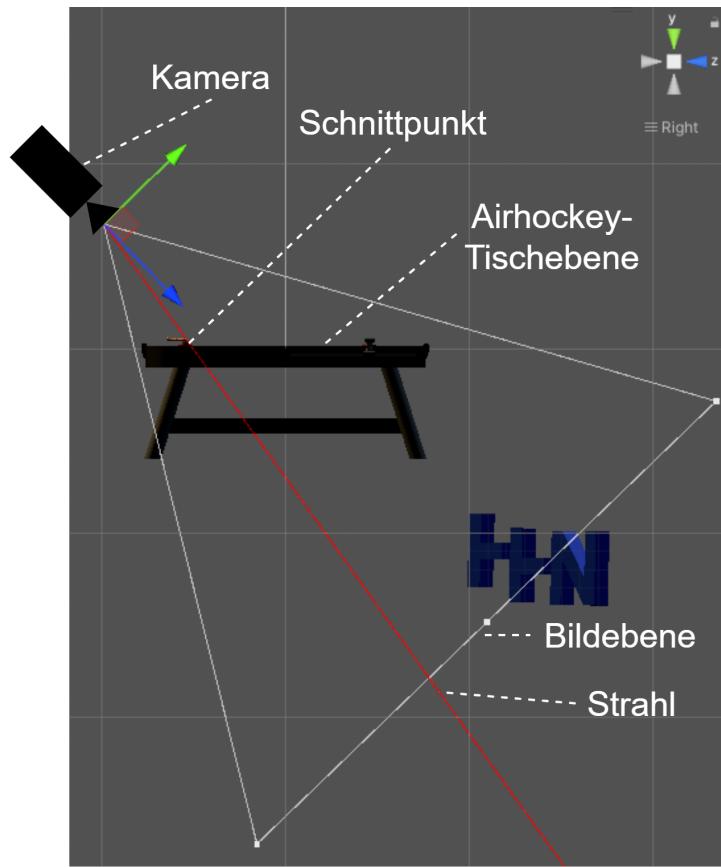


Abbildung 5: Berechnung der Mausposition in Weltkoordinaten.

In Unity3D wird jedem Objekt ein dreidimensionaler Vektor, welcher der Position des Objekts in der Simulationsumgebung entspricht, zugeordnet. Bei Verwendung der PhysX-Engine kann die Position eines Objekts beispielsweise durch Veränderung des Vektors manipuliert werden.

Durch den Einsatz von MuJoCo als Physik-Engine kann die Position von Objekten in Unity nicht mehr zur Laufzeit auf einen Vektor, wie die auf die Tischebene projizierte Mausposition, geändert werden. MuJoCo simuliert die Kinematik und Dynamik der Objekte fortlaufend. Objekte müssen mit den von MuJoCo bereitgestellten Aktorik-Systemen manipuliert werden.

Um den spielbaren Pusher dennoch in der Airhockey-Simulation zu steuern wird ein *Arrive-Steering-Behaviour* nach Reynolds [46] implementiert. Der Algorithmus implementiert ein Verhaltensmuster zur Simulation der Bewegung von autonomen Agents. Das Arrive-Steering-Behaviour ermöglicht es dem Agents sich auf ein Ziel zuzubewegen und dabei dessen Geschwindigkeit und Bewegungsrichtung so anzupassen, dass er am Ziel ankommt und dort stoppt. Abbildung 6 zeigt die Funktionsweise des Algorithmus. Der Pusher kann sich mit maximaler Geschwindigkeit bewegen bis er den Slow-Down-Radius erreicht. In diesem Radius wird die Geschwindigkeit des Pushers reduziert, sodass er im Zielradius zum Stehen kommt. Der Kreismittelpunkt entspricht der Mausposition, also der Zielposition des Pushers. Der Zielradius wird klein gewählt, sodass der Pusher näherungsweise an der Zielposition zum Stehen kommt [46, 47].

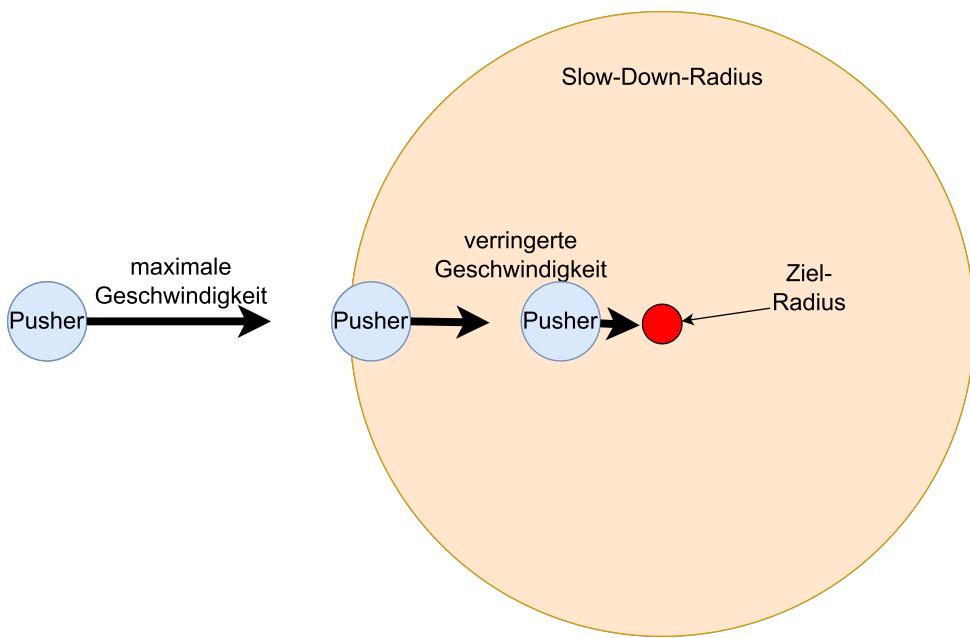


Abbildung 6: Darstellung der Funktionsweise des Arrive-Steering-Behaviour-Algorithmus nach Lima [2].

Der vollständige Algorithmus wird in Algorithmus 1 dargestellt. Es zeigt, dass der Algorithmus die aktuelle Geschwindigkeit des Pushers betrachtet und daraus die notwendige Beschleunigung berechnet, um die Zielgeschwindigkeit zu erreichen. Da die Geschwindigkeit nicht sofort geändert werden kann, wird die Beschleunigung basierend darauf berechnet, die Zielgeschwindigkeit innerhalb einer festen Zeitspanne zu erreichen [47]. Die Parameter Zielradius, Slow-Down-Radius, Höchstgeschwindigkeit, Höchstbeschleunigung und ZeitZuZielgeschwindigkeit können im Unity-Editor angepasst werden, um die Steuerung des Pushers zu beeinflussen.

---

**Algorithm 1:** Arrive-Steering-Behaviour-Algorithmus nach Millington [47]

---

**Result:** Beschleunigungsvektor

**Input:** Zielposition, Istposition, Istgeschwindigkeit, Zielradius, Slow-Down-Radius, Höchstgeschwindigkeit, Höchstbeschleunigung, ZeitZuZielgeschwindigkeit

```
1 Zielgeschwindigkeitsvektor ← Zielposition - Istposition
2 Distanz ← Betrag des Zielgeschwindigkeitsvektor
3 if Distanz < Zielradius then
4   | Istgeschwindigkeit ← Nullvektor
5   | return Nullvektor
6 end
7 if Distanz > Slow-Down-Radius then
8   | Zielgeschwindigkeit ← Höchstgeschwindigkeit;
9 else
10  | Zielgeschwindigkeit ← Höchstgeschwindigkeit × (Distanz / Slow-Down-Radius)
11 end
12 Zielgeschwindigkeitsvektor normalisieren
13 Zielgeschwindigkeitsvektor × = Zielgeschwindigkeit
14 Beschleunigungsvektor ← Zielgeschwindigkeitsvektor - Istgeschwindigkeit
15 Beschleunigungsvektor × = 1 / ZeitZuZielgeschwindigkeit
16 if Betrag des Beschleunigungsvektor > Höchstbeschleunigung then
17   | Beschleunigungsvektor normalisieren
18   | Beschleunigungsvektor × = Höchstbeschleunigung
19 end
20 return Beschleunigungsvektor
```

---

### 3.2 Implementierung von Domain-Randomization-Techniken

Um den in Kapitel 2.1.4 erklärten Domain Gap zu verringern, werden Domain-Randomization-Techniken eingesetzt. Die Verwendung von Domain Randomization bedeutet, dass die Umgebung des Agents mit zufällig generierten Umgebungsvariablen während des Trainings verändert wird. Der Agent wird dadurch auf Variationen in der realen Welt vorbereitet. Durch das Aussetzen des Agents gegenüber unterschiedlichen Umgebungen während des Trainingsprozesses soll der Agent lernen flexibel auf eine Vielzahl von Situationen zu reagieren [12, 17].

Da das Anwenden von Domain-Randomization-Techniken auf jede Art von Umgebung neu angepasst werden muss, wird ein Framework für das Einstellen der Randomisierung von Umgebungsvariablen in die Unity-Simulationsumgebung integriert. Es wird dadurch ermöglicht die Domain Randomization im Unity-Editor zu konfigurieren.

Die Domain Randomization kann in drei Kategorien aufgeteilt werden. Die erste Kategorie beinhaltet die Randomisierung von physikalischen Umgebungsvariablen, wie Reibung und Dämpfung. Dadurch werden unter anderem Unterschiede zwischen dem Reibungsverhalten und den Kontaktodynamiken des Pucks ausgeglichen. Die zweite Kategorie ist das Randomisieren der Observations. Dies modelliert das Messrauschen bei der Extraktion der Spielinformationen mittels Bildverarbeitung bzw. die Unsicherheit der Prädiktion der Puck-Kinematik durch einen Kalman-Filter (siehe Abbildung 1). Die dritte Kategorie beinhaltet das Verzögern und Stören der Actions des Agents. Das Verzögern der Actions modelliert die Latenz der physischen Steuerung des Agent-Pushers [13]. Durch das Stören der Actions wird man nicht modellierten Dynamiken des Tisches gerecht, wie beispielsweise unterschiedliche Reibungswerte an verschiedenen Stellen des Tisches oder unterschiedliche Kontaktodynamiken an unterschiedlichen Stellen an der Banne [48].

Zunächst wird ein Domain-Randomization-Controller-Skript implementiert, indem die drei Kategorien der Domain Randomization global für die Umgebung aktiviert werden können. Durch die Möglichkeit einzelne Domain-Randomization-Techniken an- bzw. abzuschalten, kann gezielt auf Schwächen in der Policy oder Veränderungen in der realen Umgebung reagiert werden. Muss beispielsweise die Kamera des Systems ausgetauscht werden und die neue Kamera hat eine andere Auflösung, kann die Randomisierung der Observations angepasst werden, um der veränderten Unsicherheit der Puck- bzw. Pusher-Detektion gerecht zu werden.

Die Randomisierung für Umgebungsparameter akzeptiert als Eingabe mehrere Objekte, auf die die aktuelle Parameterkonfiguration angewandt werden soll. Auf diese Weise können wiederholende Konfigurationen mehrfach verwendet werden. Um ein Objekt in Unity mit MuJoCo simulieren zu können, müssen einem Parent-Objekt MuJoCo-Objekte als Childs hinzugefügt werden. Die Child-Objekte enthalten die Konfiguration der physikalischen Parameter des Parent-Objekts. Dieser Zusammenhang ist in Abbildung 7 dargestellt. Als Objekte für die Randomisierung werden sowohl das Parent-Objekt, als auch die Child-Objekte akzeptiert. Wird ein Parent-Objekt als Eingabe für die Randomisierung verwendet, werden die physikalischen Parameter aller Child-Objekte mit der selben Konfiguration randomisiert.

Nach Festlegung der Objekte, auf die die aktuelle Domain-Randomization-Konfiguration angewandt werden soll, muss festgelegt werden, wie häufig die Parameter neu randomisiert werden. Das Erstellen einer neuen, randomisierten Parameterkonfiguration nach jeder Episode ist nach Muratore et al. [48] die gängigste Vorgehensweise.

Der Bereich in dem ein Umgebungsparameter randomisiert wird kann auf zwei Arten festgelegt werden. Die erste Möglichkeit ist die Eingabe eines Wertebereichs, in dem der Parameter randomisiert wird. Die zweite Möglichkeit ist die Eingabe eines prozentualen Anteils des Startwerts des jeweiligen Parameters. Der Anteil wird vom Startwert subtrahiert bzw. addiert, um

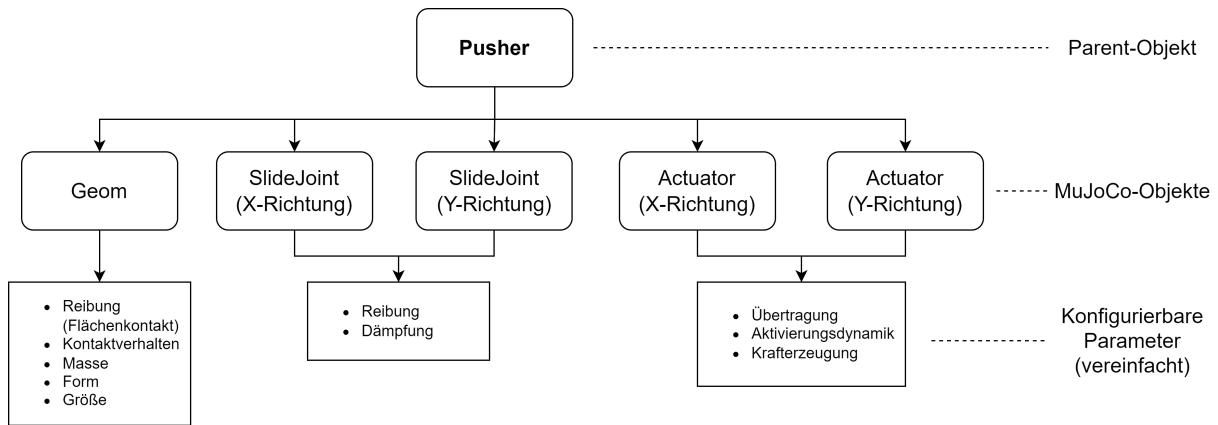


Abbildung 7: Struktur der MuJoCo-Objekte in der Unity-Simulation am Beispiel des Pushers.

den Wertebereich zu markieren. Gleichung 4 zeigt beispielhaft die Berechnung des Wertebereichs für einen prozentualen Anteil von 10% für den Startwert 1. Der Präzisionsparameter legt fest, mit wie vielen Nachkommastellen die Parameter innerhalb des Wertebereichs randomisiert werden.

Startwert: 1

Prozentualer Anteil: 10

$$\text{Berechnung: } 1 \cdot \frac{10}{100} = 0.1 \rightarrow 1 - 0.1 = 0.9 \text{ und } 1 + 0.1 = 1.1 \quad (4)$$

Wertebereich: [0.9, 1.1]

Die Auswahl der Wahrscheinlichkeitsdichtefunktion legt eine Verteilung für die Zufallszahlen innerhalb des Wertebereichs fest, die für die Randomisierung verwendet werden. Es kann ausgewählt werden zwischen einer Gleichverteilung und der Standardnormalverteilung. Abbildung 8 zeigt die Einstellungsmöglichkeiten der Domain Randomization für Umgebungsvariablen im Unity-Editor.

Observations und Actions sind zweidimensionale Vektoren. Daher werden X- und Y-Wert für die Randomisierung der Observations und das Stören von Actions nach dem Prinzip des prozentualen Wertebereichs (siehe Gleichung 4) implementiert. Zusätzlich kann die Schrittweite festgelegt werden, in der Störungen bzw. Verzögerungen angewandt werden. Des Weiteren besteht die Möglichkeit eine Wahrscheinlichkeit einzustellen, mit der die jeweilige Action-Randomisierung aktiv ist. Damit kann auch ein System simuliert werden, das in unregelmäßigen Zeitabständen verzögert bzw. gestört wird. Durch das Stören wird der Action-Vektor, der von der Policy bzw. dem DNN ausgegeben wird, im ausgewählten Bereich verändert. Somit werden nicht modellierte Dynamiken berücksichtigt. Die Verzögerung von Actions wird realisiert, indem im Unity-Editor ein Wertebereich angegeben wird, der die minimale bzw. maximale Anzahl an Actions festlegt, die verzögert werden. Ist die Verzögerung aktiv, werden die Actions

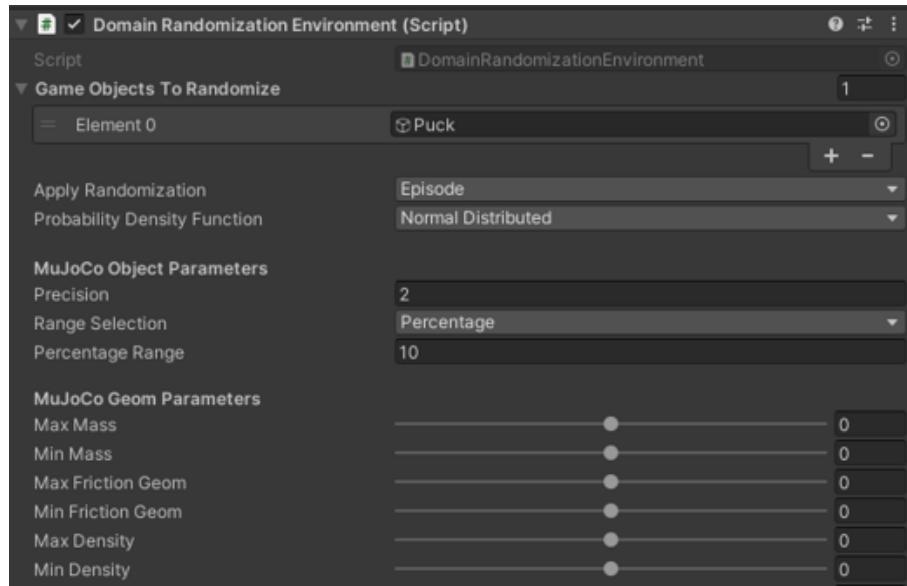


Abbildung 8: Einstellungsmöglichkeiten der Domain Randomization für Umgebungsvariablen im Unity-Editor.

in einem First In First Out (FIFO)-Buffer gespeichert. Erreicht der Buffer die maximale Anzahl an zu verzögernden Actions, wird er nach dem First In First Out (FIFO)-Prinzip geleert. Der Agent führt verzögert die Actions aus, die sich im Buffer befinden, bis der Buffer leer ist. Der Ablauf der Verzögerung ist in Abbildung 9 dargestellt.

### 3.3 Visuelle Weiterentwicklung der Airhockey-Simulation

Um die Airhockey-Simulation als Demonstrator zu nutzen wurde das visuelle Erscheinungsbild der Simulation weiterentwickelt. Ziel der Anpassungen ist es die Steuerung der Simulation für den Benutzer intuitiv zu gestalten und ein optisch ansprechendes Aussehen zu erreichen.

Hierfür werden neben dem Airhockey-Tisch Info-Panel eingeblendet. Das Panel auf der rechten Seite des Tisches erklärt in wenigen Sätzen das Themengebiet RL. In der linken oberen Ecke des Bildschirms erklärt ein Panel wie zwischen Selfplay-Modus und Spiel gegen die KI per Mauseingabe gewechselt werden kann. Zusätzlich wird auf der linken Seite die Entwicklung des Rewards in der aktuellen Episode dargestellt. Hierfür wird jedes Reward-Signal mit dem kumulativen Reward in der aktuellen Episode und der Reward-Änderung pro Schritt angezeigt. Zuletzt wird der totale Reward visualisiert. Dies ermöglicht es dem Entwickler die Entwicklung des Rewards zu verfolgen und, wie in Kapitel 5.1 erläutert, die Zusammensetzung der Reward-Struktur zu optimieren.

Des Weiteren wurden ein animiertes HHN-Logo, sowie animierte Spielstände über dem jeweiligen Tor eingefügt. Die Tischoberfläche wurde neu gestaltet und die Belichtung dieser

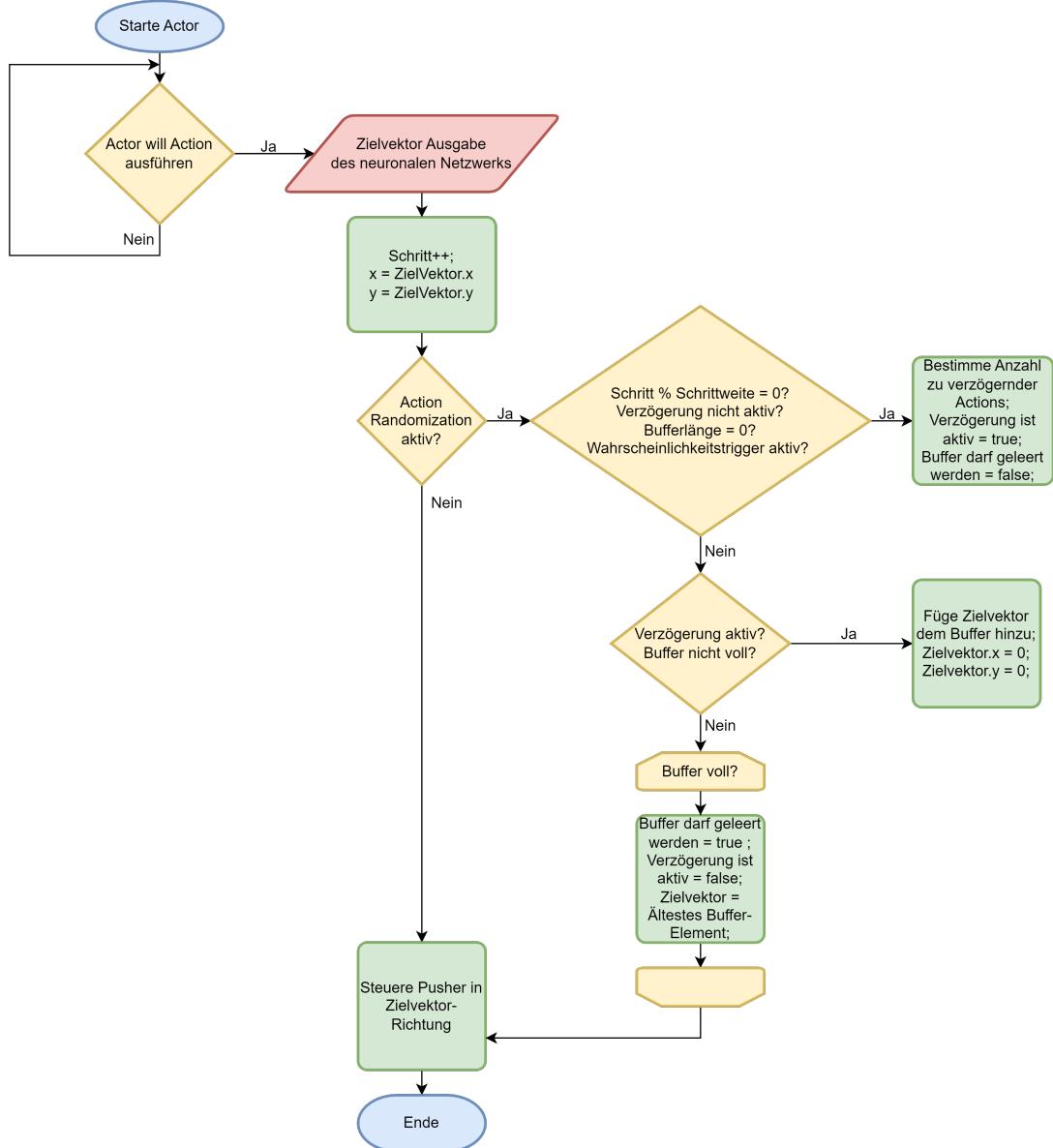


Abbildung 9: Programmablaufplan der Verzögerung der Action-Randomization.

verbessert. Kreise vor dem Tor symbolisieren, ähnlich wie beim Eishockey, den Torraum und sollen damit die Sichtbarkeit der Tore verbessern. Zusätzlich wurde ein virtueller Raum als Hintergrund für den Airhockey-Tisch eingefügt.

Die Standarddarstellung des Maus-Cursor wird ausgeblendet und durch eine virtuelle Hand ersetzt, die den Cursor darstellt. Befindet sich der Benutzer im Selfplay-Modus, wird die virtuelle Hand transparent dargestellt. Im Spiel gegen die KI wird die virtuelle Hand vollständig angezeigt. Die zuvor beschriebenen Visualisierungen werden in Abbildung 10 dargestellt. Abbildung 4 zeigt den vorherigen Stand der Airhockey-Simulation und eignet sich daher zum Vergleich.

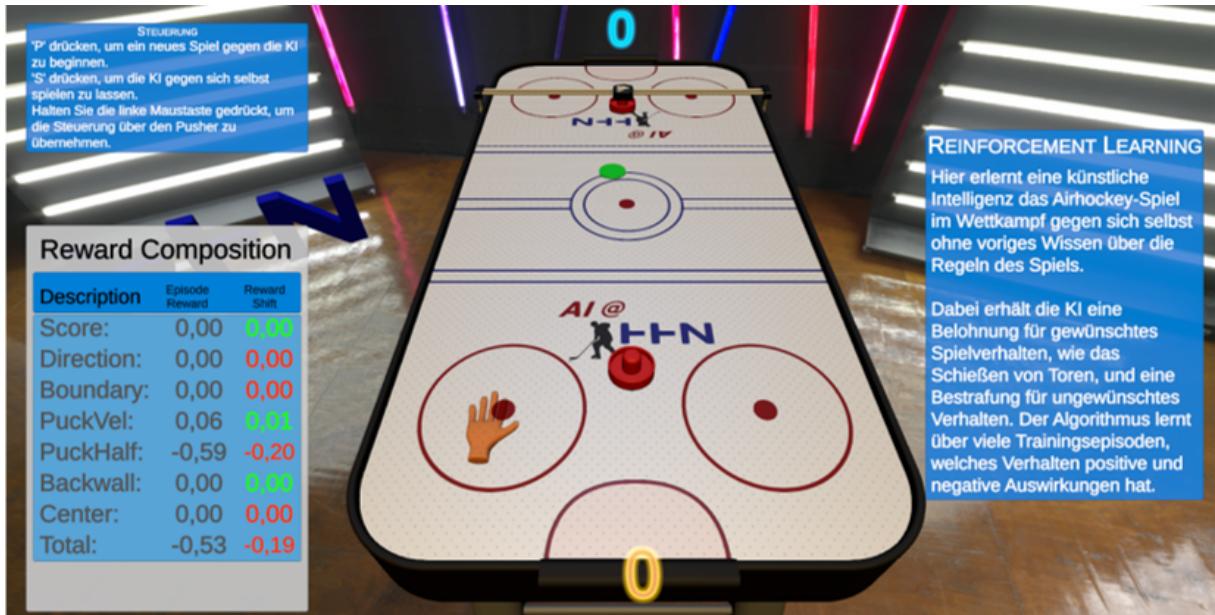


Abbildung 10: Airhockey-Simulation mit visuellen Anpassungen.

### 3.4 Entwicklung von Methoden zur Evaluation von Airhockey-Agents

Zur Evaluation des Verhaltens von Airhockey-Agents werden zwei Methoden entwickelt. Die erste Methode ist eine Visualisierung der Trajektorie der beiden Pusher und des Pucks in der Airhockey-Simulation. Die zweite Methode ist das Darstellen der Häufigkeit, mit der sich ein Agent in einer Position befindet, durch eine Heatmap.

Zur Visualisierung der Trajektorie des Pucks und der Pusher, wird in Unity ein Skript implementiert, das Screenshots der Airhockey-Simulation erstellt. Anschließend werden die Screenshots in einem Python-Skript weiterverarbeitet. Die Trajektorie der Objekte wird visualisiert, indem zu Beginn des Aufnahmezeitraums die Objekte mit hoher Transparenz dargestellt werden. Im Laufe des Aufnahmezeitraums wird die Transparenz schrittweise verringert, sodass die zurückgelegte Bahn des jeweiligen Pushers bzw. des Pucks nachvollziehbar ist. Dies ist in Abbildung 11 dargestellt. Hierfür werden zunächst die Pusher und der Puck isoliert, indem die restlichen Pixel der Bilder vollständig transparent gemacht werden. Um die Objekte in Abhängigkeit des Aufnahmezeitraums transparent darzustellen, werden die Bilder nach Aufnahmzeit sortiert und anschließend der Wert des Transparenzkanals  $\alpha$  der Bilder der Objekte in Abhängigkeit der Bildnummer und der Gesamtzahl der Bilder berechnet. Dies ist in Gleichung 5 dargestellt.

$$\alpha = \alpha_{max} * (1 - (NummerBild / AnzahlBildergesamt)) \quad (5)$$

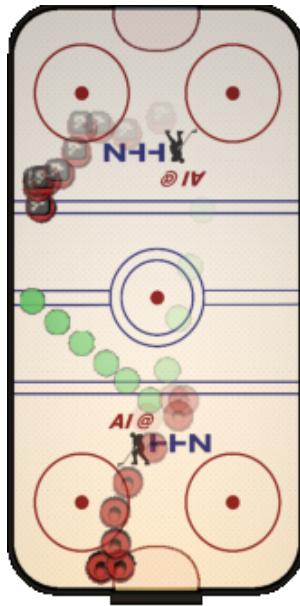


Abbildung 11: Beispiel einer Trajektorie-Visualisierung für Airhockey-Agents.

Zum Erstellen der Heatmap wird zunächst in Unity ein Skript implementiert, dem Simulationsobjekte übergeben werden können. Das Skript speichert dann die Position der Objekte in einer Comma-Separated Values (CSV)-Datei. Auf Basis der gespeicherten Positionsdaten wird in einem Python-Skript eine Heatmap als Kernel-Dichteschätzung (engl. Kernel Density Estimate (KDE)) erstellt. Eine horizontale und eine vertikale Linie in der Mitte der Grafik in blauer Farbe markieren die Spielfeldmitte. Ebenfalls werden schwarze Linien gezeichnet, die die Tore auf beiden Seiten des Spielfeldes repräsentieren. Abbildung 12 zeigt ein Beispiel der Heatmap.

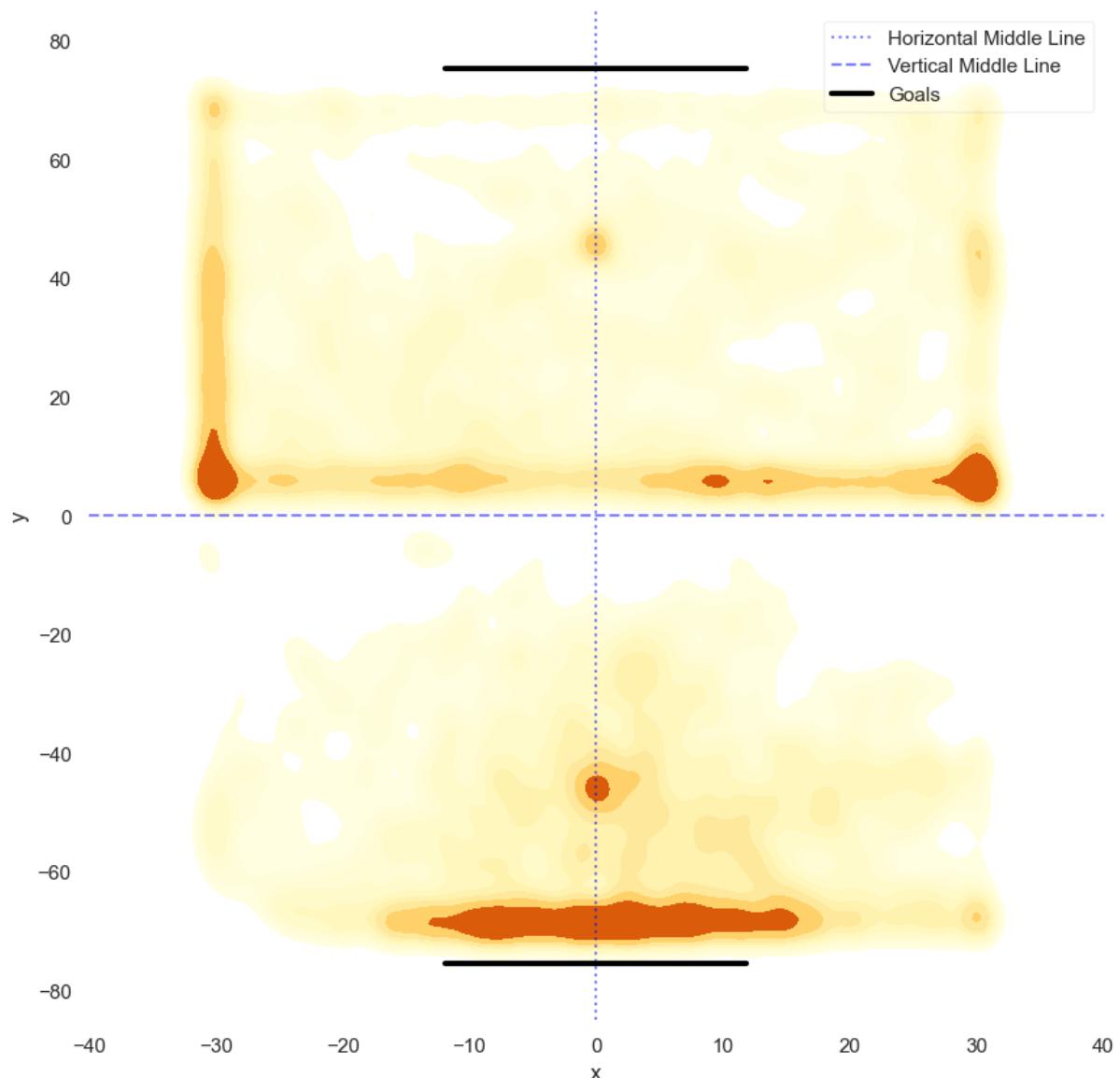


Abbildung 12: Beispiel einer Heatmap für die Darstellung in welcher Häufigkeit Airhockey-Agents sich in bestimmten Positionen befinden.

## 4 Weiterentwicklung des Modular Reinforcement Learning Framework

Neben der Weiterentwicklung der Airhockey-Simulation ist es Teil dieser Arbeit das Modular Reinforcement Learning (MRL)-Framework weiterzuentwickeln. Im Folgenden wird aufgezeigt, welche Änderungen am MRL-Framework vorgenommen wurden, um die Handhabung und das Monitoring des Trainingsprozesses zu verbessern und die Möglichkeiten zur Evaluation der Agents zu erweitern. In Kapitel 4.1 wird zunächst aufgezeigt, weshalb ein Rating-Algorithmus zur Bewertung von Airhockey-Agents notwendig ist. Es wird dargestellt weshalb der Glicko2-Algorithmus für das MRL geeignet ist und wie der Algorithmus implementiert wird. Zusätzlich zeigt Kapitel 4.2 die Vorteile eines Command-Line Interface (CLI) für das MRL. Abschließend beschreibt Kapitel 4.3 die Implementierung von grafischen Darstellungen für die Daten während des Trainingsprozesses und der Evaluation mittels des Rating-Algorithmus.

### 4.1 Algorithmus zum Bewerten von RL-Agents

Zur Evaluation der RL-Agents sollen objektive Bewertungsmetriken herangezogen werden. Dies ist zum Einen notwendig, um den Trainingsfortschritt der Agents zu überprüfen und zum Anderen, um die Leistungsstärke der Agents zu vergleichen.

Im Folgenden wird aufgezeigt, weshalb ein Rating-Algorithmus zur Bewertung von Airhockey-Agents notwendig ist. Anschließend wird dargestellt, wie der *Glicko2*-Algorithmus nach Glickman [35] zur Bewertung und Evaluation von RL-Agents ausgewählt wurde. Zusätzlich wird aufgezeigt, wie der *Glicko2*-Algorithmus in das MRL-Framework integriert wurde.

#### 4.1.1 Auswahl und Anpassung eines Rating-Algorithmus

Welche Bewertungsmaßnahmen aussagekräftig für den Trainingsfortschritt bzw. die Spielstärke eines Agents sind, hängt von der Anwendung und Umgebung des Agents ab. Wird ein Agent für ein Jump-And-Run-Spiel, wie in Karakovskiy et al. [42] trainiert, kann der zurückgelegte Weg in einem Level ausreichend zur Bewertung sein. In einem solchen Szenario befinden sich keine anderen Agents oder Spieler in der Umgebung. Verschiedene Agents können unabhängig voneinander bewertet werden und eine absolute Maßnahme, wie beispielsweise der zurückgelegte Weg, liefert einen aussagekräftigen Vergleichswert [42]. Dieses Prinzip gilt ebenfalls bei der Anwendung von RL-Algorithmen für autonomes Fahren. Wang et al. [43] benutzen zur Bewertung eines Models zur Steuerung eines Car-Racing-Simulators absolute Maßnahmen, wie die durchschnittlich zurückgelegte Strecke, die durchschnittliche Geschwindigkeit und die durch-

schnittliche Abweichung von Rennstreckenmittelpunkt. In Nullsummenspielen, wie beispielsweise Schach oder Go, sind absolute Metriken abhängig von der Spielstärke des Gegners. Eine absolute Metrik, wie beispielsweise *Anzahl der Spielzüge bis zum Sieg*, ist nur aussagekräftig, wenn man verschiedene Agents stets gegen einen standardisierten Benchmark-Gegner antreten lässt.

Für die Bewertung von Airhockey-Agents, in der in diesem Projekt verwendeten Umgebung, sind keine Benchmarks vorhanden. Daher soll mit einem Rating-Algorithmus bestimmt werden können, wie ein Agent im Vergleich zu anderen Airhockey-Agents abschneidet.

Im Vergleich zum Schachspiel mit menschlichen Akteuren, wofür das Elo-System ursprünglich entwickelt wurde [32], gibt es beim Airhockey mit RL-Agents Unterschiede zu berücksichtigen. Ein RL-Agent wird typischerweise in deutlich mehr Spielen trainiert und evaluiert als Menschen in Schachturnieren. Des Weiteren ist die Volatilität eines neu trainierten Agents unbekannt. Elos K-Faktor basiert auf empirischer Schätzung der Varianz des menschlichen Spiels und der Annahme, dass Anfänger schnell lernen, während ein Schachgroßmeister seine Fähigkeiten nur noch langsam verbessern kann. Ein RL-Agent lernt nur während dem Training. Das Berechnen des Ratings während dem Training ist nur sinnvoll, um den Trainingsfortschritt zu überwachen. Da der Agent, wie in Kapitel 2.1 beschrieben, während dem Training explorieren muss sind die Ergebnisse der Spiele daher nicht aussagekräftig für eine Bewertung der tatsächlichen Spielstärke. Nach dem Training verbessert sich der Agent nicht mehr. Hat der Agent in bestimmten Bereichen des Spiels Stärken und in anderen Bereichen Schwächen, wird er auch nach einer hohen Anzahl an Spielen diese nicht verbessern und inkonsistente Ergebnisse erzielen [49, 32].

Zusätzlich betrachtet das Elo-System als Ergebnis nur Sieg, Unentschieden oder Niederlage. In vielen Spielen deutet ein Sieg mit einem deutlichen Vorsprung auf eine höhere Spielstärke des Siegers hin, als ein knapper Sieg. Das Elo-System betrachtet diesen *Margin of Victory (MoV)* nicht [49, 32].

Der Glicko2-Algorithmus ist für die Bewertung von RL-Agents in einem Nullsummenspiel wie Airhockey besser als das Elo-System geeignet. RL-Agents müssen in der Lage sein eine Vielzahl von Situationen zu lernen und daraus ihre Policy zu optimieren [5]. Daher kann es sein, dass ein Agent in bestimmten Spielsituationen eine gute Leistung zeigt, während er in anderen Spielsituationen keine passende Lösung hat. Ein Airhockey-Agent könnte zum Beispiel außerordentlich gut darin sein Tore zu erzielen, aber ebenso anfällig für Gegentore sein. Ob dieser Agent ein Spiel gewinnen kann hat eine große Zufallskomponente. Die Varianz der Spielstärke ist hoch. Die RD und Volatilität im Glicko2-Algorithmus ermöglichen es die Unsicherheit und Schwankungen in der Leistung eines RL-Agents besser zu erfassen und somit eine genauere Einschätzung der aktuellen Leistungsfähigkeit eines Agents zu liefern. Da sich die Leistung

eines Agents im Laufe des Trainingsprozesses ändern kann, ist es sinnvoll die RD und Volatilität zu berücksichtigen. Eine niedrige RD und Volatilität bei hohem Rating deuten auf einen stabilen Agents hin, der in der Lage ist, in einer Vielzahl von Spielsituationen gute Ergebnisse zu erzielen. Der Glicko2-Algorithmus betrachtet den MoV nicht [35].

Der in Kapitel 2.6 erwähnte TrueSkill-Algorithmus [38] ist besonders für die Bewertung der Spielstärke von Spielern in Mehrspieler-Spielen geeignet. Da es sich beim Airhockey um ein Nullsummenspiel mit jeweils einem Spieler handelt, ist der Glicko2-Algorithmus besser als der TrueSkill-Algorithmus geeignet.

Um den MoV in einem Rating-Algorithmus für ein RL-Framework zu implementieren ist zu beachten, dass in verschiedenen Spielen der MoV unterschiedlich gewichtet werden muss. Dies soll an folgendem Beispiel verdeutlicht werden: Beim Eishockey ist es üblich, dass das in Rückstand liegende Team den Torhüter durch einen zusätzlichen Spieler ersetzt, wenn noch wenige Minuten zu spielen sind und der Rückstand weniger als drei Tore beträgt. Durch den zusätzlichen Feldspieler erhöht sich die Wahrscheinlichkeit, dass das Team mit Rückstand ein Tor erzielt und das Spiel ausgleichen kann. Jedoch erhöht sich durch das leere Tor die Wahrscheinlichkeit sehr stark, dass die führende Mannschaft bei Gewinn des Pucks ein Tor erzielt. Der MoV wird dadurch verzerrt und gibt nicht die eigentliche Spielstärke der Mannschaften wieder. Im Fußball hingegen verlässt der Torhüter seltener das Tor bei Rückständen. Dementsprechend fallen weniger Tore bei leerem Tor. Der MoV ist in diesem Fall ein besserer Indikator für die Spielstärke der Mannschaften. Wird im Tennis eine Partie als gewonnen betrachtet, indem ein Spieler zwei Sätze gewinnt, ist der MoV kein guter Indikator für die Spielstärke der Spieler. Es wäre möglich, dass ein Spieler den ersten Satz mit 7-6 spielen gewinnt, den zweiten Satz mit 0-6 verliert und den Entscheidungssatz wieder mit 7-6 gewinnt. Der Spieler gewinnt die Partie mit einem negativen MoV nach Spielen von -4 [50]. Die Beispiele zeigen, dass der MoV in verschiedenen Spielen unterschiedlich gewichtet werden muss. Für ein allgemeines RL-Framework wie das MRL ist es nicht sinnvoll Spezialfälle im Rating-Algorithmus zu berücksichtigen. Zudem erlaubt die Simulation eine Vielzahl von Spielen in kurzer Zeit zu simulieren. Dadurch ist der Algorithmus in der Lage eine genaue Einschätzung der Fähigkeiten der Agents zu liefern und der MoV kann in einem Rating-Algorithmus für das MRL vernachlässigt werden.

#### 4.1.2 Implementierung und Integration des Glicko2-Algorithmus

Die Berechnung eines Ratings ist, wie in Kapitel 2.6 und Kapitel 4.1.1 beschrieben, kein Problem exklusiv für den Anwendungsfall einer Airhockey-KI. Daher wird das Berechnen von Ratings in das MRL-Framework integriert.

Um ein Rating-Update nach einer gespielten Partie zu berechnen, wird zunächst das Spielergebnis benötigt. Da die Spiele in der Unity-Simulationsumgebung gespielt werden, müssen die Ergebnisse von Simulationsseite an das MRL übermittelt werden. Eine Lösung ist die Entwicklung eines benutzerdefinierten *Side-Channels*. Ein Side-Channel ist ein Kommunikationskanal zwischen C# (Unity) und Python (MRL), um benutzerdefinierte Datenstrukturen zu versenden. Wie in Abbildung 13 dargestellt, wird das Ende einer Partie als Event benutzt, um das Versenden einer Nachricht über den Side-Channel zu triggern. Die Nachricht enthält den Spielstand zum Spielende und eine Spiel-ID, die der Anzahl der gespielten Partien entspricht. Wird eine Nachricht vom MRL über den Side-Channel empfangen, wird eine Funktion aufgerufen, die das Spielergebnis in eine Comma-Separated Values (CSV)-Datei abspeichert. Abschließend kann auf Basis der gespeicherten Ergebnisse das Rating-Update berechnet werden.

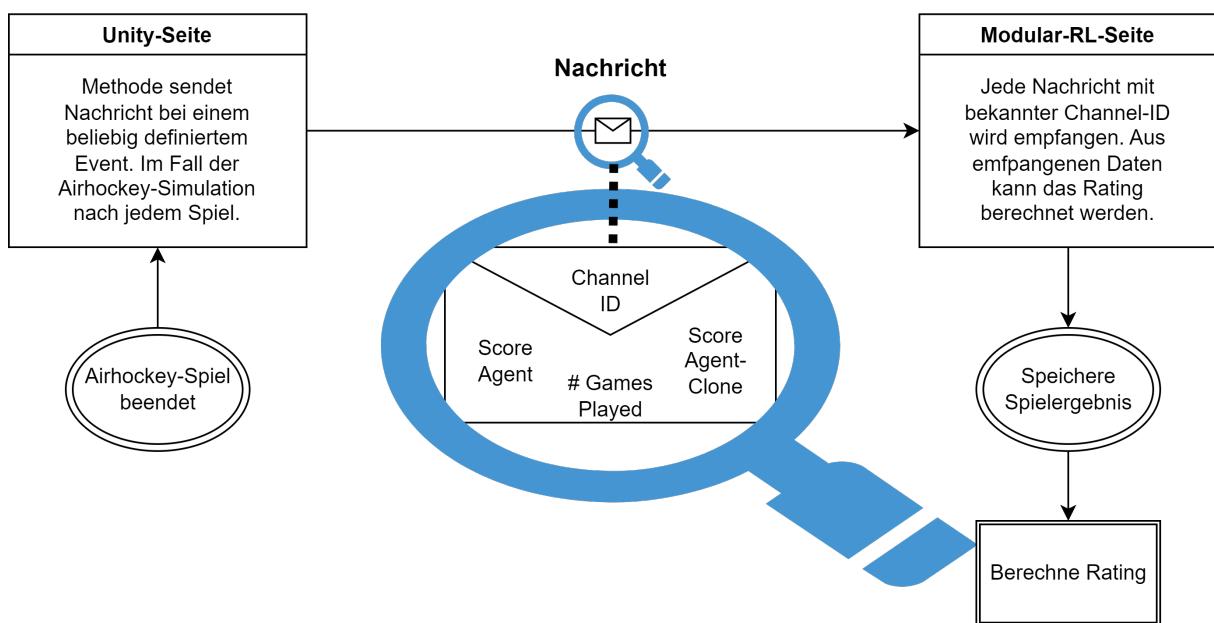


Abbildung 13: High-Level-Übersicht der Übermittlung des Spielergebnisses von der Simulation zum MRL über einen Side-Channel.

Um unter einer Vielzahl an RL-Agents, welche in verschiedenen Spielsituationen verschiedene Stärken und Schwächen aufweisen, den stärksten Agent mit Hilfe des Rating-Algorithmus zu bestimmen, wird ein Turniersystem implementiert. Das MRL-Framework stellt dafür einen Turnierplaner zur Verfügung. Es kann eine beliebige Anzahl an Agents, die am Turnier teilnehmen sollen festgelegt werden. Verfügbare Turniertypen sind *Round Robin* und *Center Player*. Beim Round-Robin-Turnier spielt jeder Turnierteilnehmer einmal gegen die anderen Teilnehmer. Beim Center-Player-Turnier tritt jeder Teilnehmer einmal gegen den Center-Player an. Turniertypen mit Eliminierungen werden nicht unterstützt, da die Agents auf Basis möglichst vieler Spiele bewertet werden sollen, um statistische Ausreißer zu vermeiden. Tabelle 1 und Tabelle 2 zeigen jeweils ein Beispiel für die beiden unterstützten Turniersysteme. Außerdem ist konfigurierbar, wie häufig der Spielplan wiederholt wird.

Des Weiteren ist zu berücksichtigen, dass der Glicko2-Algorithmus – im Gegensatz zum Elo-System – nicht vorsieht ein Rating-Update nach jeder Partie durchzuführen. Stattdessen wird das Rating-Update nach einer festgelegten Anzahl an Partien durchgeführt. Diese Serie an Begegnungen wird als *Rating-Periode* bezeichnet. Die Länge einer Rating-Periode wird als Parameter beim Starten eines Turniers festgelegt. Nach jeder Rating-Periode wird das Rating-Update durchgeführt. Glickman [35] empfiehlt eine Rating-Periode von durchschnittlich 10 bis 15 Partien pro Spieler.

Tabelle 1: Beispiel für ein Round-Robin-Turnier mit vier Agents.

Runde	Spiel 1	Spiel 2
1	Agent1 vs Agent2	Agent3 vs Agent4
2	Agent1 vs Agent3	Agent2 vs Agent4
3	Agent1 vs Agent4	Agent2 vs Agent3

Tabelle 2: Beispiel für ein Center-Player-Turnier mit vier Agents.

Runde	Spiel
1	Agent1 vs Agent2
2	Agent1 vs Agent3
3	Agent1 vs Agent4

## 4.2 Hinzufügen eines Command-Line Interface (CLI)

Zur Verbesserung der Handhabung des MRL-Frameworks wird dem Benutzer die Möglichkeit gegeben ein Command-Line Interface (CLI) zu nutzen. Die Bedienung von Software über ein Command-Line Interface (CLI) ist bekannt aus Tools wie *Git* oder *Docker*. Durch das Verwenden von Tastenkombinationen und Textbefehlen können Parameter, wie beispielsweise der RL- oder Exploration-Algorithmus, vor dem Starten eines Trainings effizient angepasst werden. Zuvor mussten die Parameter im Quellcode geändert werden. Zusätzlich können die Parameter in einem Command-Line Interface (CLI) in einem Skript gespeichert und automatisiert ausgeführt werden. Dies erleichtert die Wiederholung von Trainings mit den wechselnden Parametern. Wie Abbildung 14 zeigt, wird dem Benutzer bei Eingabe des Hilfe-Befehls oder bei ungültiger Eingabe eine Übersicht und Erklärung der verfügbaren Befehle angezeigt.

## 4.3 Implementierung von grafischen Darstellungen für Trainings- und Rating-Daten

Zur Darstellung von Trainings- und Rating-Daten wird dem MRL eine Komponente zum grafischen Darstellen dieser Daten hinzugefügt. Der Benutzer kann das Command-Line In-

```

usage: main.py [-h] [-m MODE] [-mp MODEL_PATH] [-cp CLONE_PATH] [-ti TRAINER_INTERFACE] [-ep ENV_PATH]
               [-ta TRAIN_ALGORITHM] [-dp DEMO_PATH] [-ea EXPLORATION_ALGORITHM] [-pa PREPROCESSING_ALGORITHM]
               [-pp PREPROCESSING_PATH] [-rmoc REMOVE_OLD_CHECKPOINTS] [-p TRAINING_PARAMETERS TRAINING_PARAMETERS]
               [-gpf GAMES_PER_FIXTURE] [-rts REPEAT_TOURNAMENT_SCHEDULE] [-tt TOURNAMENT_TYPE] [-cpl CENTER_PLAYER]
               [-gr GAME_RESULTS_PATH] [-rp RATING_HISTORY_PATH]

optional arguments:
  -h, --help            show this help message and exit
  -m MODE, --mode MODE  Choose between 'training', 'testing', 'fastTesting' or 'tournament'
  -mp MODEL_PATH, --model_path MODEL_PATH
                        If you want to test a trained model or continue learning from a checkpoint enter the model path here.
  -cp CLONE_PATH, --clone_path CLONE_PATH
                        If defined this path defines the clone's weights for selfplay training / testing. Otherwise, model_path will be used.
  -ti TRAINER_INTERFACE, --trainer_interface TRAINER_INTERFACE
                        Choose from 'MLAgentsV18' (Unity) and 'OpenAIGym'
  -ep ENV_PATH, --env_path ENV_PATH
                        If you want to run multiple Unity actors in parallel you need to specify the path to the Environment 'exe' here. In case of 'OpenAIGym' enter the desired env name here instead, e.g.'LunarLanderContinuous-v2'. If you want a CQL agent to learn from demonstrations, an environment can be used to evaluate the model on a regular basis. Please provide a path or type None to connect directly to the Unity Editor. Otherwise, type 'NoEnv' to proceed without evaluation.
  -ta TRAIN_ALGORITHM, --train_algorithm TRAIN_ALGORITHM
                        Choose from 'DQN', 'DDPG', 'TD3', 'SAC', 'CQL'
  -dp DEMO_PATH, --demo_path DEMO_PATH
                        In case you want to train the agent offline via CQL please provide the path for demonstrations.
  -ea EXPLORATION_ALGORITHM, --exploration_algorithm EXPLORATION_ALGORITHM
                        Choose from 'None', 'EpsilonGreedy', 'ICM', 'RND', 'ENM', 'NGU'
  -pa PREPROCESSING_ALGORITHM, --preprocessing_algorithm PREPROCESSING_ALGORITHM
                        Choose from 'None' and 'SemanticSegmentation'
  -pp PREPROCESSING_PATH, --preprocessing_path PREPROCESSING_PATH
                        Enter the path for the preprocessing model if needed
  -rmoc REMOVE_OLD_CHECKPOINTS, --remove_old_checkpoints REMOVE_OLD_CHECKPOINTS
                        Determines if old model checkpoints will be overwritten
  -p TRAINING_PARAMETERS TRAINING_PARAMETERS, --training_parameters TRAINING_PARAMETERS TRAINING_PARAMETERS
                        Parse the trainer configuration (make sure to select the right key (training algorithm) e.g. <path to yaml> <key>)

```

Abbildung 14: Befehlszeilenparameterübersicht für das MRL-Framework.

terface (CLI) nutzen, um zwischen dem Plotten der Historie von Trainingsdaten, wie der Reward-Entwicklung beim Training, oder die Rating-Entwicklung in einem Turnier zu wählen. Die Quelle der Daten und Speicherort der Plots können ebenfalls über das Command-Line Interface (CLI) konfiguriert werden.

Die Reward-Entwicklung in einer Episode während eines Trainings unterliegt starken Schwankungen. Der Reward wird daher in der Darstellung geglättet. Zur Glättung wird nach dem Vorbild des Visualisierungs-Framework *TensorBoard* [51] der exponentiell gewichtete gleitende Durchschnitt des Rewards berechnet. Abbildung 15 zeigt ein Beispiel für ein Plot der geglätteten Reward-Entwicklung nach einem Training.

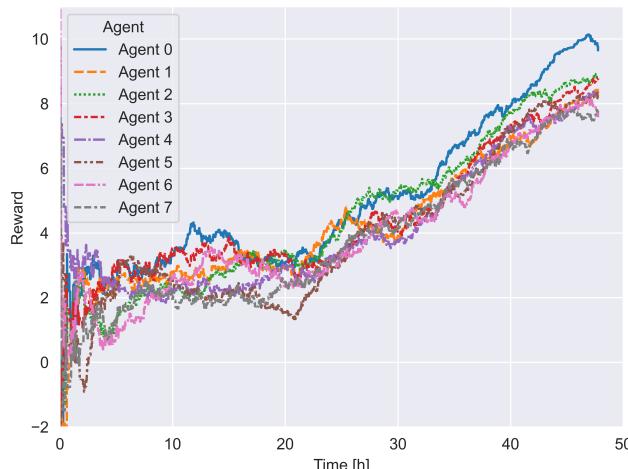


Abbildung 15: Beispiel für ein Plot der Reward-Entwicklung beim Training.

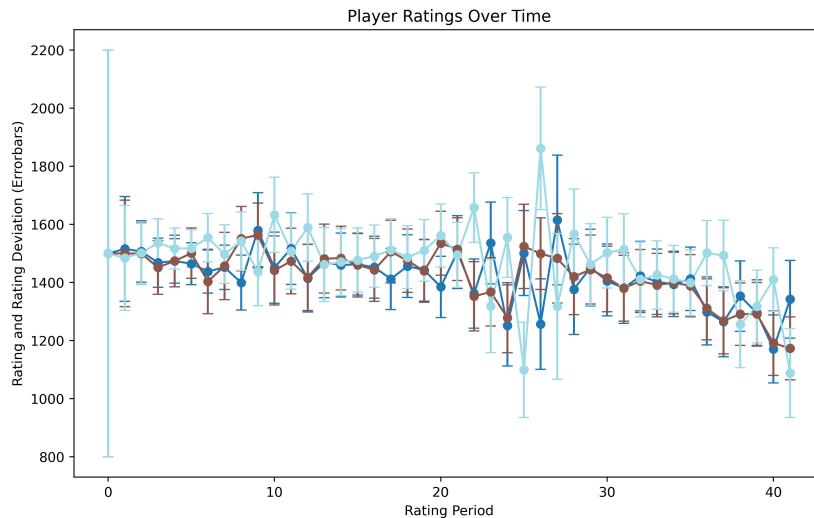


Abbildung 16: Beispiel für ein Plot der Rating-Entwicklung in einem Turnier mit drei Agents.

Die Rating-Historie wird in einer für den Glicko2-Algorithmus passenden Darstellung geplottet. Abbildung 16 zeigt ein Beispiel für ein Plot der Rating-Entwicklung in einem Turnier. Die Rating-Entwicklung wird für jede Rating-Periode dargestellt. Das 95%-Konfidenzintervall der RD wird als Fehlerbalken dargestellt.

#### 4.4 Exportieren von Reward- und Observation-Zusammensetzung

Das MRL exportiert Trainingsparameter wie die Konfiguration der Hyperparameter, sobald ein Training gestartet wird. Die Informationen werden in einer YAML Ain't Markup Language (YAML)-Datei gespeichert.

Nicht gesichert werden die Reward-Zusammensetzung bzw. das Reward-Scaling und die verwendeten Observations in der Umgebung. Diese Informationen sind für die Reproduzierbarkeit eines Trainings wichtig. Rewards und Observations werden in der Unity-Simulation konfiguriert. Die Daten können daher, wie in Kapitel 4.1.2 beschrieben, per Side-Channel von der Airhockey-Simulation an das MRL gesendet werden. Hierfür wird bei der Initialisierung eines Trainings eine Kommunikation mit dem Side-Channel aufgebaut. Werden Daten empfangen werden diese an die bestehende Struktur zum Sichern der Trainingsparameter mit YAML Ain't Markup Language (YAML) angehängt und gespeichert.

## 5 Training des Reinforcement Learning (RL)-Agents

Zum Training des Airhockey-Agents wird der in Kapitel 2.1.3 beschriebene Soft Actor-Critic (SAC)-Algorithmus verwendet. Der SAC ist ein DRL-Algorithmus, der sich besonders für Anwendungsfälle eignet, bei denen kontinuierliche Aktionen ausgeführt werden sollen. Dadurch ist der SAC-Algorithmus geeignet für die Steuerung eines Airhockey-Agents [6, 9].

In Kapitel 5.1 die Zusammensetzung und Skalierung der Reward-Struktur dargestellt. Kapitel 5.2 stellt die wesentlichen Hyperparameter vor. Zuletzt werden die verwendeten Observations beschrieben.

### 5.1 Zusammensetzung der Reward-Struktur

Wie in Kapitel 2.1.1 beschrieben ist die Zusammensetzung der Reward-Struktur ein zentrales Element für die erfolgreiche Anwendung von RL-Algorithmen. Die Reward-Struktur motiviert den Agents bestimmte Aktionen auszuführen und bestimmte Ziele zu erreichen [5].

Bei der Gestaltung der Reward-Struktur ist zwischen inkrementellen und endgültigen Reward-Signalen zu unterscheiden. Inkrementelle Reward-Signale werden für jede Aktion ausgegeben, die der Agent ausführt. Endgültige Reward-Signale werden erst am Ende einer Episode ausgegeben. Bei der inkrementellen Reward-Signalen erhält der Agent fortlaufend Feedback. Da der SAC-Agent beim Airhockey kontinuierliche Aktionen ausführen soll, werden inkrementelle Reward-Signale verwendet [5].

Zusätzlich zur Verwendung von inkrementellen Reward-Signalen, ist die Dichte der Reward-Signale zu beachten. Die Dichte der Rewards bezieht sich darauf, wie oft der Agent ein Reward-Signal erhält. Eine zu geringe Reward-Dichte kann den Lernprozess verlangsamen [5].

Die Reward-Struktur sollte robust gegenüber *Reward-Hacking* sein. Reward-Hacking betreibt der Agent, wenn er unerwünschte Verhaltensweisen entwickelt, um die Belohnung zu maximieren [52]. Beispielsweise würde ein Airhockey-Agent, der für das Erzielen von Toren belohnt wird, aber die Tore nicht unterscheiden kann, möglicherweise beginnen Eigentore zu schießen.

Für das Training des Airhockey-Agents wurden folgende Zusammensetzung der Rewards verwendet:

- Positiver Reward, wenn der Agent ein Tor erzielt.
- Negativer Reward, wenn der Gegner ein Tor erzielt.

- Positiver Reward, wenn der Agent den Puck gegen die gegnerische Rückwand schießt.
- Negativer Reward, wenn der Puck in der eigenen Hälfte des Spielfeldes ist.
- Positiver Reward, wenn der Puck auf eine hohe Geschwindigkeit beschleunigt wird.
- Negativer Reward, für unnötige und häufige Richtungsänderungen des Agents.
- Negativer Reward, wenn der Agent nach dem Stoßen des Pucks nicht in die vertikale Spielfeldmitte (Tormitte) zurückkehrt.
- Negativer Reward, wenn sich der Agent an der Spielfeldbande bewegt

Neben der Zusammensetzung der Reward-Struktur, ist die Skalierung der Rewards zu definieren. Der Wert eines einzelnen Reward-Signal muss gewichtet werden, um einen stabilen Lernprozess zu gewährleisten [9]. Häufiger auftretende Reward-Signale, wie beispielsweise der Reward für das Beschleunigen des Pucks, sollten einen geringen Anteil des kumulativen Gesamtrewards ausmachen. Das Erzielen eines Tores stellt das Hauptziel des Airhockey-Agents dar und kommt seltener vor. Daher sollte der Anteil des Rewards für das Erzielen eines Tores höher sein.

Bei der Verwendung des SAC-Algorithmus ist die Zusammensetzung und Skalierung der Rewards einer der wichtigsten Hyperparameter. Mit einer gut gewählten Reward-Skalierung kann der SAC-Algorithmus die Balance zwischen Exploration und Exploitation finden und einen schnelleren Lernprozess mit besseren Ergebnissen gewährleisten [9, 53].

## 5.2 Hyperparameter

Wie zuvor in Kapitel 5.1 beschrieben, ist die Zusammensetzung und Skalierung der Rewards ein wichtiger Hyperparameter. Zusätzlich zu den Rewards sind weitere Hyperparameter für das Training eines RL-Agents mit dem SAC-Algorithmus zu definieren. Im Folgenden werden die wichtigsten Hyperparameter für das Training beschrieben:

- *Batch-Größe*: Die Batch-Größe beeinflusst, wie viele Erfahrungen gleichzeitig zum Aktualisieren der Agent-Parameter verwendet werden [28].
- *Lernrate*: Lernrate für den Gradientenabstieg des Actor-Netzwerks ( $\alpha$ ) und Critic-Netzwerks ( $\beta$ ) [28].
- *Discount-Faktor  $\gamma$* : Der Discount-Faktor  $\gamma$  bestimmt, wie stark zukünftige Rewards gewichtet werden. Ein hoher Discount-Faktor  $\gamma$  bewirkt, dass zukünftige Rewards stärker

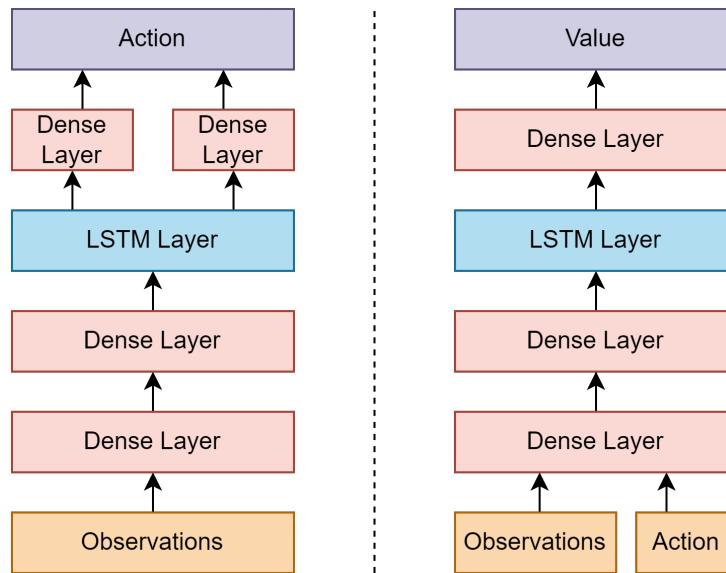


Abbildung 17: Actor-Netzwerk (links) und Critic-Netzwerk (rechts). Jedes Netzwerk besteht aus einem Input-Layer, Fully-Connected (Dense) Layern mit SELU-Aktivierungsfunktionen [3] und einem rekurrenten LSTM-Layer [4].

gewichtet werden. Ein niedriger Discount-Faktor  $\gamma$  bewirkt, dass zukünftige Rewards schwächer gewichtet werden [28].

- *Ziel-Netzwerk-Aktualisierungsfaktor  $\rho$* : Bestimmt wie häufig das Ziel-Netzwerk aktualisiert wird [28].
- *Selfplay-Netzwerk-Aktualisierungsfaktor*: Bestimmt wie häufig das Selfplay-Netzwerk, also der Gegner des Agents, aktualisiert wird.
- *Replay-Buffer-Kapazität*: Die Kapazität des Replay-Buffers bestimmt, wie viele Erfahrungen gespeichert und für das Training verwendet werden können [28].
- *Entropie-Koeffizient  $\tau$* : Der Entropie-Trade-Off-Koeffizient, auch als Temperatur bezeichnet, steuert wie stark das Critic-Netzwerk während des Modell-Updates aktualisiert werden soll [28].

Eine vollständige Liste der verwendeten Hyperparameter für den SAC-Algorithmus im MRL-Framework ist in Anhang A dargestellt. Die verwendete Netzwerkarchitektur für die DNNs für Actor und Critic ist in Abbildung 17 dargestellt.

### 5.3 Observations

Als Observations werden standardmäßig die Positionen und Geschwindigkeiten der beiden Pusher und des Pucks verwendet. Dies entspricht einem Observation-Space-Vektor mit einer

Länge von 12. Dieser Vektor dient, wie in Abbildung 3 dargestellt, als Input für das Actor-Netzwerk und Critic-Netzwerk. Zusätzlich kann der Observation-Space um den Beschleunigungswert der Objekte erweitert werden, was einem Observation-Space-Vektor mit einer Länge von 18 entspricht. Eine weitere Option zur Vergrößerung des Observation-Space ist den Abstand der Objekte zu den Toren dem Vektor hinzuzufügen. Die Länge des Observation-Space beträgt dann 26.

## 6 Evaluation der Ergebnisse

In diesem Kapitel werden die Ergebnisse des Trainingsprozesses eines Airhockey-Agents trainiert mit DRL-Methoden präsentiert und evaluiert. Das Trainingskonzept sieht vor, zunächst einen Agent von Grund auf zu trainieren, da zum Zeitpunkt der Erstellung dieser Arbeit keine vortrainierten Agents für die Airhockey-Umgebung existieren. Der vortrainierte Agent soll anschließend als Ausgangspunkt dienen, um durch gezielte Anpassungen der Trainingsparameter die Policy zu verbessern. Domain-Randomization-Techniken sollen die Generalisierungsfähigkeit des Agents verbessern. Die Evaluation soll feststellen, ob die Randomisierung der Trainingsumgebung den Agent robuster macht. Zuletzt werden zum Training eines Airhockey-Agents von Grund auf verschiedene Parameter-Konfigurationen betrachtet, um die Auswirkungen der Parameter auf die Trainingszeit und die Qualität der Policy zu untersuchen. Zur Bewertung des Spielverhaltens der Agents werden die in Kapitel 3.4 beschriebenen Heatmaps und Trajektorie-Ausschnitte verwendet. Des Weiteren wird der in Kapitel 4.1 beschriebene Rating-Algorithmus verwendet, um die Leistungsfähigkeit der Agents einzuschätzen.

Das Rating der Agents erfolgt durch ihre Teilnahme in einem Round-Robin-Turnier. Die Rating-Periode wird dabei auf durchschnittlich 12 Spiele pro Periode festgelegt. Der Zyklus wiederholt sich bis 50 Rating-Perioden gespielt sind. Dies entspricht zirka 600 Spielen pro Turnierteilnehmer. Die Systemkonstante  $\tau$  im Glicko2-Algorithmus wird auf 0.5 festgelegt. Initialisiert werden unbewertete Agents mit einem Rating von 1500, einer Rating-Abweichung von 350 und einer Volatilität von 0.06 [35].

Um einen Airhockey-Agent von Grund auf zu trainieren, wurde ein Training mit den Parametern in Tabelle 3 durchgeführt. Die gewählten Hyperparameter ergeben sich aus Werten, die in Haarnoja et al [9] und Juliani et al [28] empfohlen werden. Dieser Agent wird von nun an auch als *AgentFromScratch* bezeichnet. Der Agent wurde zunächst mit gleichbleibenden Trainingsparametern für 75 Stunden trainiert.

Zur Bewertung, ob ein Training einen positiven Verlauf nimmt, können beim RL verschiedene Metriken herangezogen werden. Dazu gehören Statistiken der Umgebung, der Policy und der Loss-Funktionen [28]. Beim Training eines Airhockey-Agents mit dem SAC-Algorithmus im MRL sind folgende Metriken von Interesse:

- Der durchschnittliche kumulative Reward pro Episode.
- Der durchschnittliche Verlust des Wertefunktions-Updates.
- Der durchschnittliche Betrag der Policy-Loss-Funktion.

Tabelle 3: Trainingsparameter für ein Training eines Airhockey-Agents ohne vortrainierten Agent.

<b>Reward</b>	<b>Wert</b>
Agent erzielt Tor	10
Gegner erzielt Tor	-5
Gegnerische Rückwand	0.5
Puck in Agent-Hälfte	-0.05
Puck-Beschleunigung	0.15
Häufige Richtungsänderungen	-0.3
Spielfeldmitte	-0.025
Bande vermeiden	-0.25
<b>Observations</b>	-
Observation-Space-Vektor-Länge	12
<b>Hyperparameter</b>	-
Anzahl Actor	12
Batch-Größe	32
Lernrate Actor $\alpha$	0.0005
Lernrate Critic $\beta$	0.0005
Discount-Faktor $\gamma$	0.99
Ziel-Netzwerk-Aktualisierungsfaktor $\rho$	300
Selfplay-Netzwerk-Aktualisierungsfaktor	500
Replay-Buffer-Kapazität	300000
Entropie-Koeffizient $\tau$	0.01

Die Episoden-Länge wird nicht zur Evaluierung des Trainingsfortschritts herangezogen. Bei einem Nullsummenspiel wie Airhockey kann eine Episode lange dauern, wenn beide Agents eine gute Policy haben und Gegentore erfolgreich verhindern. Eine Episode kann ebenso lange dauern, wenn beide Agents aufgrund schlechter Spielfähigkeiten nicht in der Lage sind ein Tor zu erzielen. Die Episoden-Länge ist somit nicht ausreichend aussagekräftig für den Trainingsfortschritt.

Abbildung 18a zeigt, dass der Agent in den ersten zehn Trainingsstunden seinen durchschnittlichen Reward pro Episode von -2 auf 1.5 steigern kann. Ausreißer zu Beginn des Trainings sind auf durch Zufall gefallene Tore zurückzuführen, die auf Grund fehlender Vorergebnisse nicht geglättet werden. Nach dem steilen Anstieg zu Beginn steigert sich der Reward pro Episode nur noch langsam. Dies deutet dennoch auf einen positiven Trainingsverlauf hin, da aufgrund des in Tabelle 3 dargestellten niedrigen Selfplay-Netzwerk-Aktualisierungsfaktors der Gegner des Agents durch eine aktuelle verbesserte Version des Agents ersetzt wird. Die in Kapitel 5.1 beschriebene Reward-Struktur zeigt, dass gegen einen stärkeren Gegner auch ein niedrigerer Reward zu erwarten ist.

Der durchschnittliche Verlust des Wertefunktions-Updates ist in Abbildung 18b dargestellt. Dies korreliert damit, wie gut das Modell die Wertefunktion approximiert. Der Graph sollte

steigen solange der Agent lernt und anschließend wieder fallen, wenn sich der Reward stabilisiert [28]. Der Graph zeigt, dass der Agent lernt. Eine Stabilisierung des Rewards ist nicht zu erkennen.

Der durchschnittliche Betrag der Policy-Loss-Funktion ist in Abbildung 18c dargestellt. Dieser korreliert mit der Häufigkeit, in der sich die Policy verändert. Der Betrag sollte sich während eines erfolgreichen Trainings verringern [28]. Der Graph zeigt, dass der Policy-Loss zu Beginn des Trainings stark abnimmt und sich anschließend nur noch langsam verringert. Nach zirka 65 Stunden ist der Policy-Loss nahezu konstant. Dies deutet darauf hin, dass nur noch wenige Veränderungen der Policy stattfinden. Die stagnierende Reward- und Policy-Loss-Entwicklung legen Nahe das Training zu beenden und die Ergebnisse zu evaluieren.

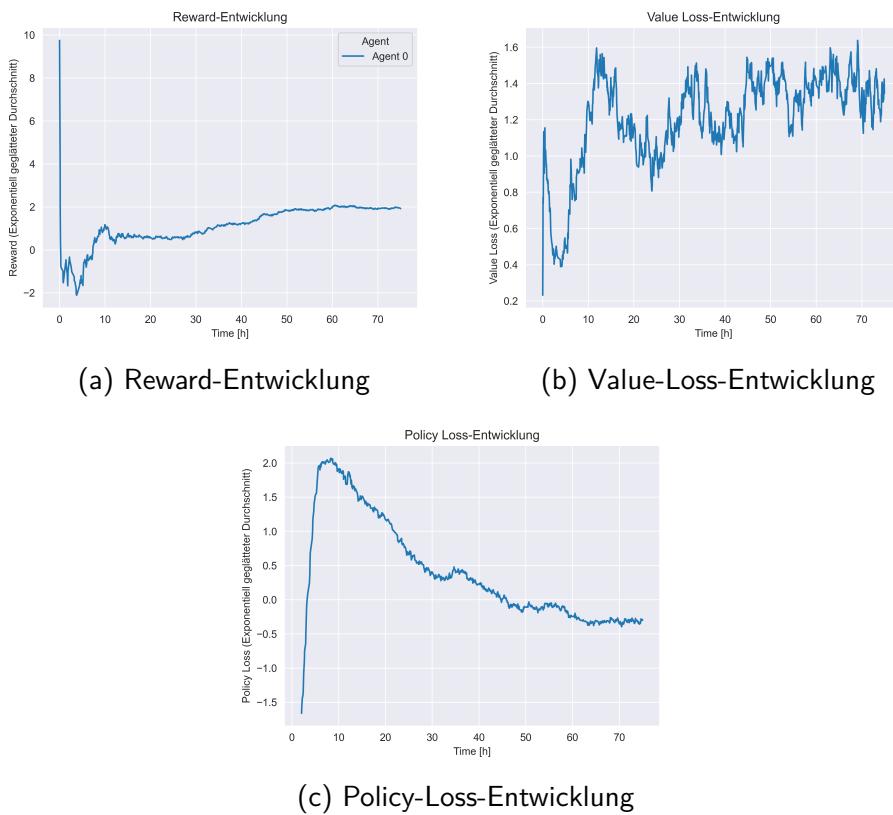
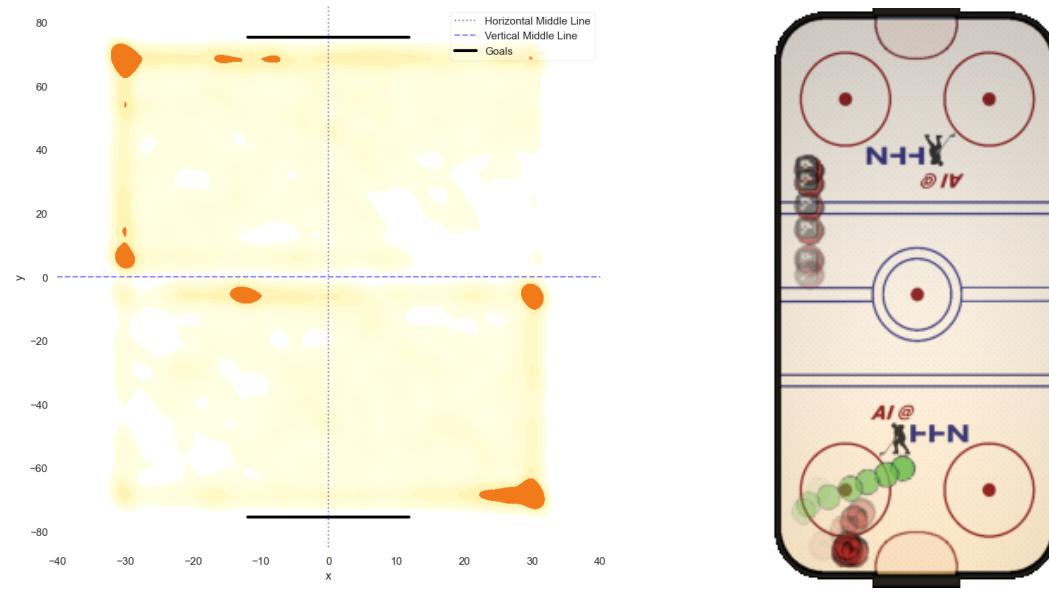


Abbildung 18: Entwicklung des ersten Trainings eines Airhockey-Agents ohne vortrainierten Agent.

Nach 24 Stunden zeigt der Agent kein intelligentes Spielverhalten. Die Heatmap in Abbildung 19a stellt den AgentFromScratch in einer Episode im Spiel gegen sich selbst dar. Die Grafik zeigt, dass der Agent sich hauptsächlich am Spielfeldrand aufhält und sich häufig weit vom eigenen Tor entfernt. Dies ist nicht als optimale Spielstrategie zu werten, da der Agent so nicht in der Lage ist, den Puck bei einem schnellen Schuss auf das eigene Tor zu blocken. Der Ausschnitt der einer Trajektorie der Airhockey-Objekte in Abbildung 19b verdeutlicht, dass der Agent suboptimal handelt. Der Puck bewegt sich von der linken unteren Spielfeldhälfte in die Mitte der unteren Hälfte. Dennoch bewegt sich der Agent in der oberen Spielfeldhälfte

an der linken Bande entlang. Besser wäre es, wenn der Agent sich zurück auf die horizontale Mittellinie bewegen würde, um einen möglichen Schuss auf das eigene Tor zu blocken.

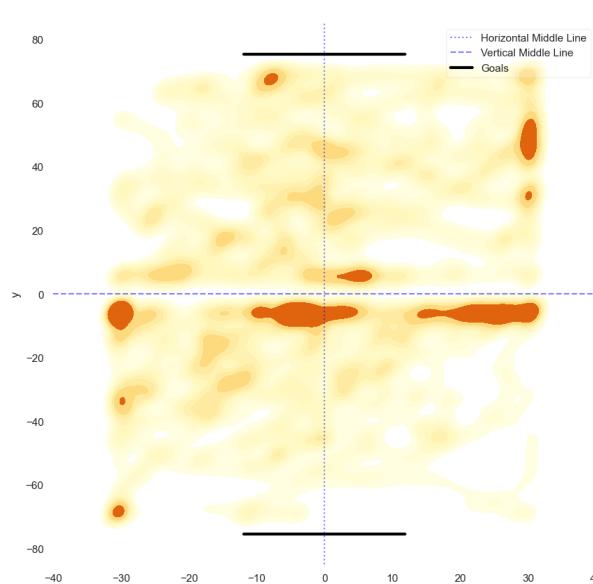


(a) Heatmap für eine Episode. (b) Trajektorie-Beispiel für den Agent.

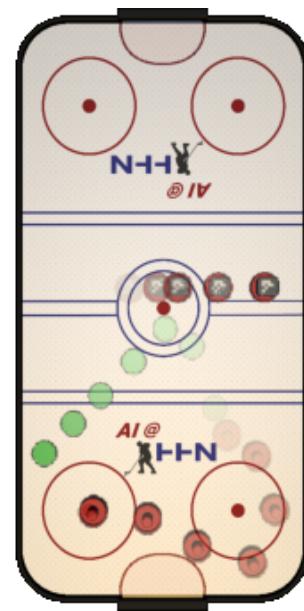
Abbildung 19: Heatmap und Trajektorie des Agents (AgentFromScratch) nach 24 Trainingsstunden.

Nach 48 Stunden Trainingszeit fokussiert der Agent sich weniger auf bestimmte Punkte auf dem Spielfeld als zuvor. Die Heatmap in Abbildung 20a zeigt dennoch, dass der Agent sich häufig an der Mittellinie bzw. den Banden aufhält. Die Trajektorie-Darstellung in Abbildung 20b zeigt, dass der Agent die Verfolgung des Pucks verbessern konnte. In der unteren Spielfeldhälfte ist zu sehen, dass der Pusher den Puck auf der rechten Spielfeldseite anspielt. Anschließend bewegt er sich, wenn auch nicht auf direktem Weg, in Richtung Puck. Der Agent zeigt dabei ein als intelligent zu wertendes Verhalten, da er sich so zum Puck bewegt, dass der direkte Weg zum eigenen Tor blockiert wird. Der Abprallwinkel des Pucks wird vom Agent gut vorhergesessen. Negativ zu bewerten ist, dass der Agent in der oberen Spielfeldhälfte, nach dem er den Puck in der Nähe des Mittelpunkts anspielt, ohne erkennbaren Grund zur rechten Bande fährt.

24 Stunden später zeigen sich weitere Verbesserungen im Spielverhalten des Agents. Die Heatmap in Abbildung 21a verdeutlicht, dass der Agent nun gelernt hat sich weniger an den Rändern der jeweiligen Spielfeldhälfte aufzuhalten. Abbildung 21b zeigt, wie der Agent in der oberen Spielfeldhälfte, nach dem er den Puck in die gegnerische Hälfte gespielt hat, sich zurück in die Spielfeldmitte vor das eigene Tor orientiert. Der Agent in der unteren Spielfeldhälfte beschützt ebenfalls das eigene Tor bevor er den Puck in einem Bogen anfährt, um ihn über die Bande in Richtung gegnerisches Tor zu spielen. Im Vergleich zu Abbildung 20b ist der Weg zum Puck deutlich direkter gewählt. Zudem zeigt Abbildung 21b, dass die Trajektorie, die der Agent wählt, einer glatten Kurve ohne abrupte Richtungsänderungen gleicht.

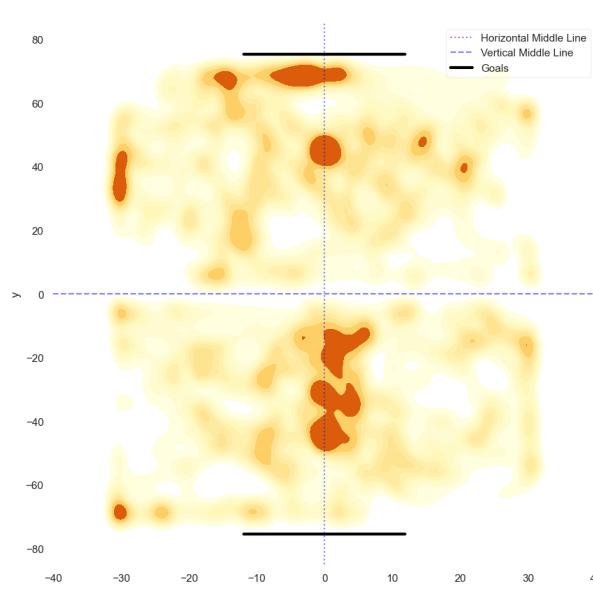


(a) Heatmap für eine Episode.

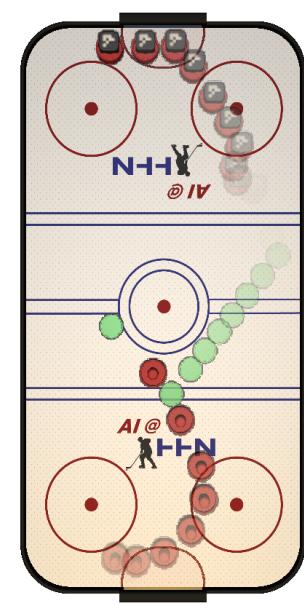


(b) Trajektorie-Beispiel für den Agent.

Abbildung 20: Heatmap und Trajektorie des Agents (AgentFromScratch) nach 48 Trainingsstunden.



(a) Heatmap für eine Episode.



(b) Trajektorie-Beispiel für den Agent.

Abbildung 21: Heatmap und Trajektorie des Agents (AgentFromScratch) nach 72 Trainingsstunden.

Um die Spielfähigkeiten des AgentFromScratch weiter zu verbessern, wird auf Basis des trainierten Agents ein weiterer Agent trainiert. Dieser Agent wird mit einer vortrainierten Policy initialisiert und anschließend mit einer angepassten Reward-Funktion wie der AgentFromScratch trainiert. Die abgestimmten Parameter für das Training sind in Tabelle 4 dargestellt. Die Obser-

vations und Hyperparameter werden nicht verändert und entsprechen den Trainingsparametern in Tabelle 3. Der Agent wird von nun an als AgentFromScratchContinued bezeichnet.

Die in Abbildung 22 dargestellten Entwicklungen während des Trainings zeigen nach 48 Stunden ein ähnliches Bild, wie der in Abbildung 18 dargestellte Trainingsverlauf des AgentFromScratch. Im Vergleich zum AgentFromScratch zeigt die fallende Value-Loss-Entwicklung, dass der Reward sich stabilisiert.

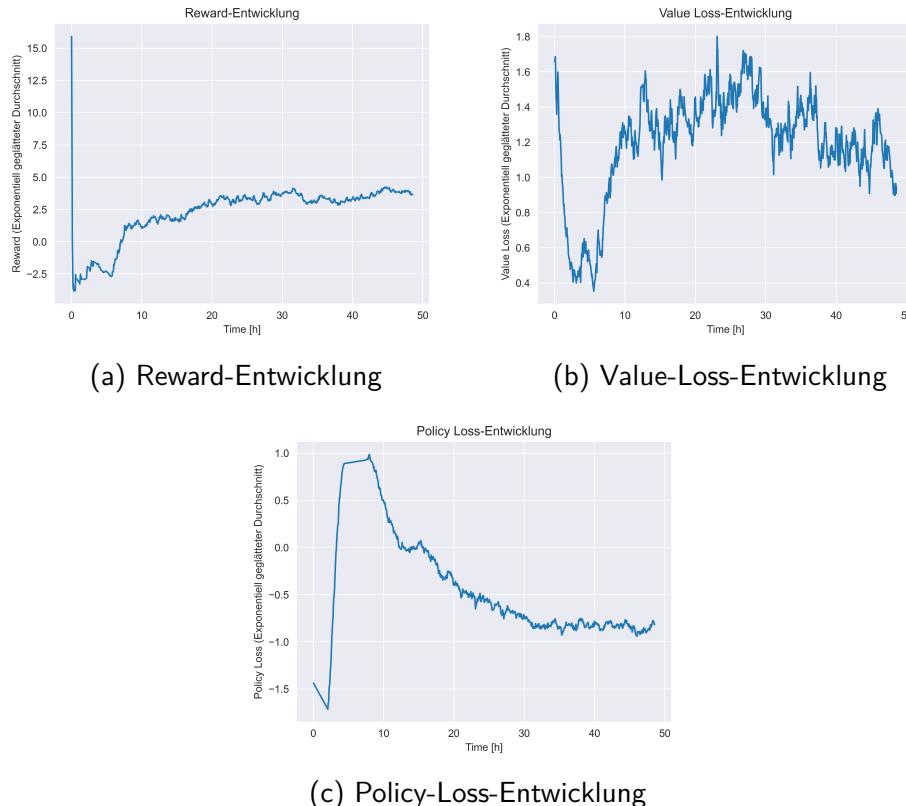


Abbildung 22: Entwicklung des Trainings zur Feinabstimmung mit vortrainierter Policy.

Tabelle 4: Trainingsparameter für die Feinabstimmung eines Airhockey-Agents mit vortrainiertem Agent.

Reward	Wert
Agent erzielt Tor	10
Gegner erzielt Tor	-7
Gegnerische Rückwand	0.5
Puck in Agent-Hälfte	-0.05
Puck-Beschleunigung	0.2
Häufige Richtungsänderungen	-0.37
Spielfeldmitte	-0.04
Band vermeiden	-0.25

Die Analyse des AgentFromScratchContinued mittels Trajektorie-Darstellung und Heatmap zeigen keine eindeutige Verbesserung des Spielverhaltens im Vergleich zum AgentFromScratch in Abbildung 21. Auf eine Darstellung der Heatmap bzw. eines Trajektorie-Beispiels wird daher verzichtet.

Abbildung 23 zeigt die Glicko2-Ratings für ein Round-Robin-Turnier mit dem AgentFromScratch nach 24, 48 und 72 Trainingsstunden und dem AgentFromScratchContinued nach 24 und 48 Stunden. Die Ergebnisse des Turniers und die Ratings bestätigen die zuvor beschriebene Beobachtung, dass der AgentFromScratch nach 24 Stunden Trainingszeit kein intelligentes Spielverhalten gelernt hat. Der Agent wird bereits in den ersten zehn Rating-Perioden stark abgewertet und stagniert anschließend bei einem Rating von ungefähr 850 bei einer RD von 60. Die Volatilität unterliegt keinen nennenswerten Schwankungen. Dies ist darauf zurückzuführen, dass der Agent von 604 Begegnungen nur acht gewinnen kann und somit konstant schlecht performt. Die Version des Agents nach 48 Trainingsstunden erreicht bei geringer Volatilität ein Rating von zirka 1350 bei einer Rating-Abweichung von ungefähr 45. Dies zeigt, dass der AgentFromScratch sich innerhalb von 24 Trainingsstunden um 500 Rating-Punkte steigern konnte. In der Konfidenzintervall-Betrachtung hat sich der Agent demnach mit 95 prozentiger Sicherheit um mindestens 290 Rating-Punkte verbessert. Der Agent gewinnt 182 Spiele von 602 Partien. Weitere 24 Stunden später erreicht der Agent erneut ein verbessertes Spielniveau mit einem Rating von ungefähr 1650 bei einer RD von 38 und bestätigt die zuvor beschriebene Beobachtung in Abbildung 21. Demnach kann sich der Agent von 48 auf 72 Stunden Trainingszeit mit einer Konfidenz von 95% um mindestens 134 Rating-Punkte steigern. Im Turnier gewinnt der Agent 345 seiner 600 Partien. Der AgentFromScratchContinued erreicht nach 24 Stunden ein Rating von zirka 1815 bei einer RD von 38. Im 95%-Konfidenzintervall betrachtet kann sich der Agent um mindestens 13 Punkte verbessern. Der Agent gewinnt 462 seiner 600 Begegnungen. Nach 48 Trainingsstunden erreicht der AgentFromScratchContinued ein Rating von ungefähr 1870 bei einer Rating-Abweichung von 39. Die 95%-Konfidenzintervalle überlappen sich nun. Der Agent gewinnt 505 seiner 600 Partien. Die Analyse der Ratings zeigt, dass die Feinabstimmung der Reward-Skalierung in Tabelle 4 dargestellt so wie weiteres Training die Policy verbessern und der Agent eine höheres Spielniveau erreicht. Die Ratings in Abbildung 23 zeigen auch, dass der Agent sich nach den ersten 24 Stunden am stärksten gesteigert hat. Anschließend erreichen die Agents durch jeweils weitere 24 Stunden Trainingszeit stets ein höheres Rating. Der Vorsprung zur jeweils 24 Stunden älteren Version wird jedoch geringer.

Auf Basis der Policy eines Agents, der vergleichbares Spielverhalten und vergleichbare Leistungsfähigkeit zeigt, wie der zuvor beschriebene AgentFromScratch nach 72 Stunden (siehe Abbildung 21), wurde ein weiterer Agent mit Domain-Randomization-Techniken trainiert. Da der Entwicklungsstand des MRL das in Kapitel 4.4 beschriebene Exportieren der Reward-Zusammensetzung nicht zuließ, wurde zur Hinführung der zuvor beschriebene Agent gewählt.

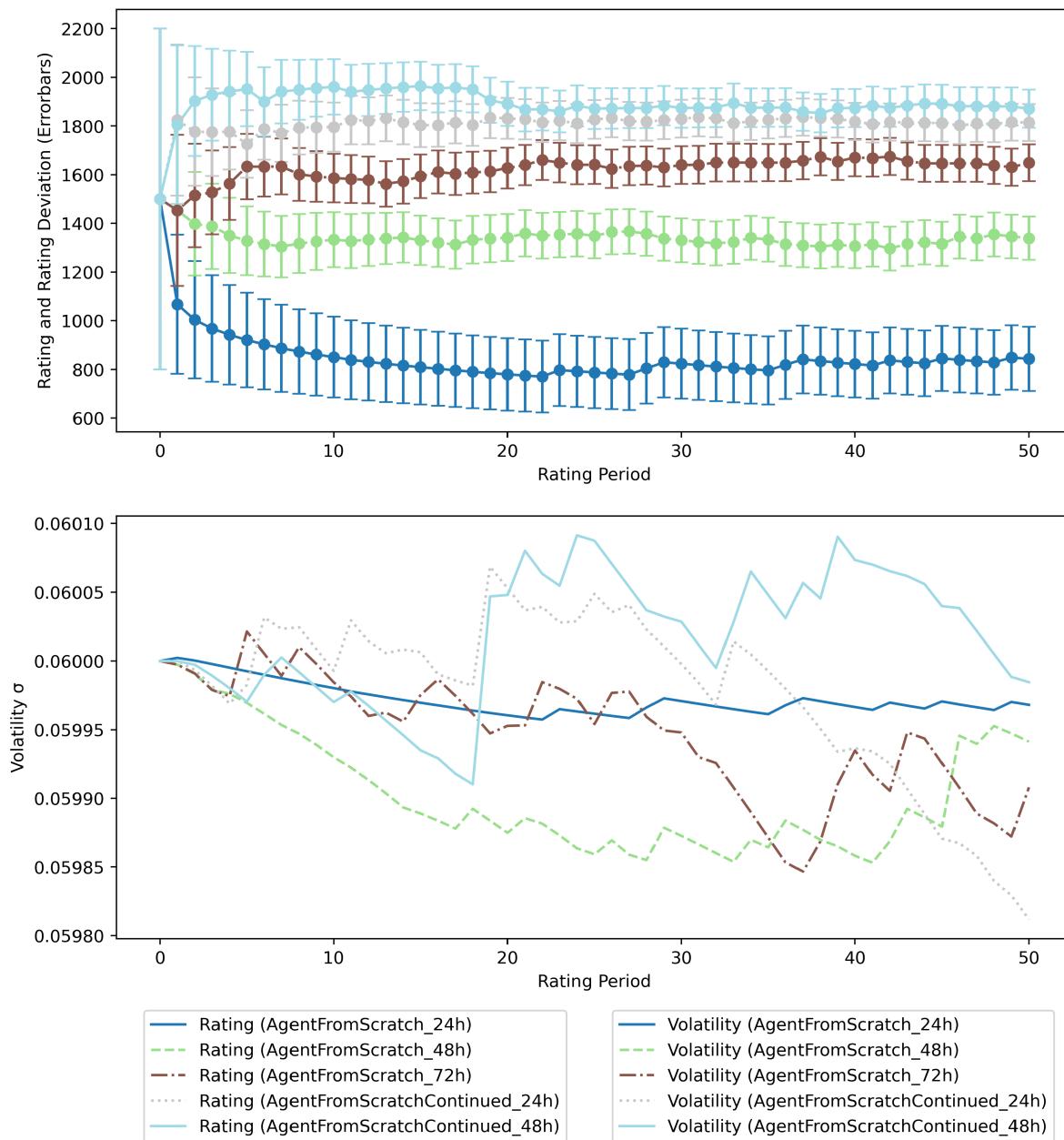


Abbildung 23: Glicko2-Ratings für ein Round-Robin-Turnier mit dem AgentFromScratch nach 24, 48 und 72 Trainingsstunden und dem AgentFromScratchContinued nach 24 und 48 Stunden.

Es wird zunächst nicht jede der in Kapitel 3.2 beschriebenen Techniken angewendet. Dies ermöglicht eine nachvollziehbare Analyse des Verhaltens des Agents in Bezug auf die jeweils angewandte Domain-Randomization-Methode. Zunächst werden die Observations und Teile der Simulationsumgebung randomisiert. Die Observations werden auf Basis der in Gleichung 4 dargestellten Formel mit einem prozentualen Anteil von fünf randomisiert. Die Randomisierung der physikalischen Parameter findet an den beiden Pushern Anwendung. Wie in Abbildung 7 dargestellt, werden für die Parent-Objekte alle MuJoCo-Objekte im Fünf-Prozent-Bereich gewählt.

Als Verteilung im Wertebereich wird für beide Domain-Randomization-Techniken die Standardnormalverteilung gewählt.

Für das Training des Agents mit Domain-Randomization-Techniken wurden die in Tabelle 5 aufgeführten Parameter gewählt. Der Agent wird von nun an auch als *AgentDomainRandomized* bezeichnet. Da die Policy des Agents bereits gelernt hat die Spielfeldmitte zu besetzen, wird die Bestrafung für Abweichungen von der Mitte reduziert. Durch das Randomisieren der Observations ist zu erwarten, dass der Agent zunächst Schwierigkeiten hat sich selbst und den Puck zu lokalisieren. Daher wird die Bestrafung für abrupte Richtungsänderungen erhöht, um zu vermeiden, dass der Agent lernt die Ungenauigkeit in der Lokalisierung durch häufige Richtungsänderungen zu kompensieren.

Tabelle 5: Trainingsparameter für ein Training eines Airhockey-Agents (*AgentDomainRandomized*) mit vortrainiertem Agent (*AgentFromScratch*) und Domain-Randomization-Techniken.

<b>Reward</b>	<b>Wert</b>
Agent erzielt Tor	10
Gegner erzielt Tor	-5
Gegnerische Rückwand	1
Puck in Agent-Hälfte	-0.1
Puck-Beschleunigung	0.15
Häufige Richtungsänderungen	-0.7
Spielfeldmitte	0
Banden vermeiden	-0.2
<b>Observations</b>	<b>-</b>
Observation-Space-Vektor-Länge	12
<b>Hyperparameter</b>	<b>-</b>
Anzahl Actor	12
Batch-Größe	64
Lernrate Actor $\alpha$	0.0005
Lernrate Critic $\beta$	0.0005
Discount-Faktor $\gamma$	0.99
Ziel-Netzwerk-Aktualisierungsfaktor $\rho$	300
Selfplay-Netzwerk-Aktualisierungsfaktor	3000
Replay-Buffer-Kapazität	300000
Entropie-Koeffizient $\tau$	0.01

Abbildung 24a zeigt, dass der geglättete kumulative Reward sich nach wenigen Stunden bei Werten zwischen drei und vier bewegt. Nach ungefähr 70 Stunden Trainingszeit wird erstmals nachhaltig die Grenze von vier überschritten. Den Höchstwert erreicht der *AgentDomainRandomized* nach zirka 95 Trainingsstunden. Zur gleichen Zeit steigt die Value-Loss-Entwicklung (siehe Abbildung 24b) an, was auf zunehmende Instabilität des Rewards hindeutet. Der Policy-Loss in Abbildung 24c zeigt ebenfalls keinen fallenden Trend. Daraus folgt, dass für diesen Trainingslauf die bestmögliche Policy nach ungefähr 95-100 Stunden gefunden wurde.

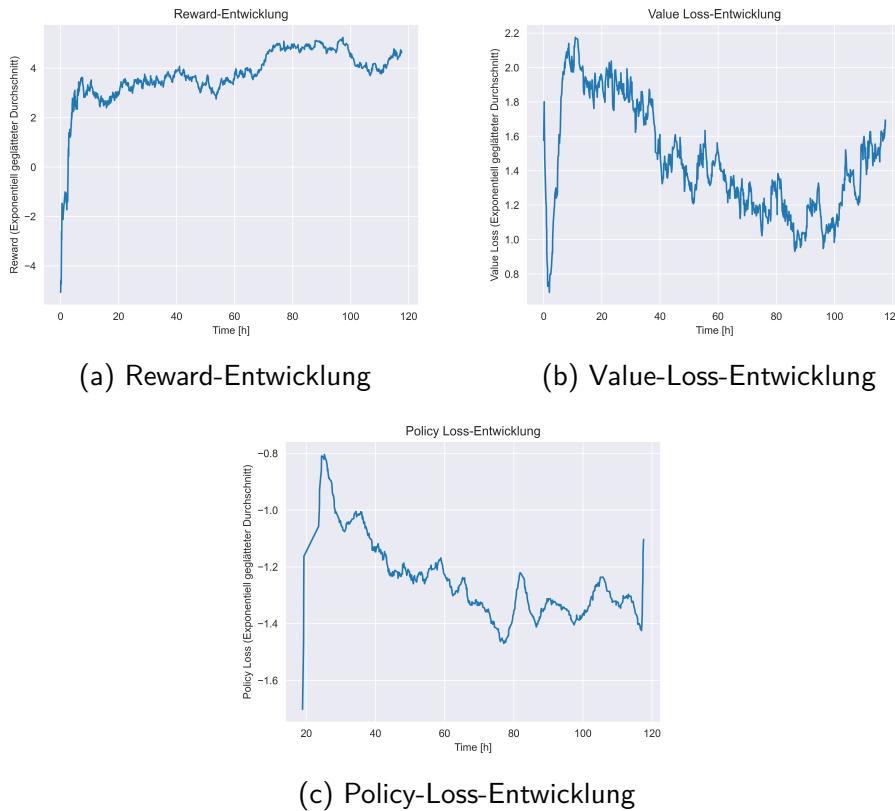
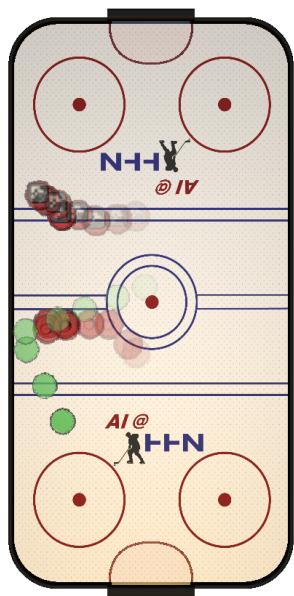


Abbildung 24: Entwicklung des Trainings mit Domain-Randomization-Techniken.

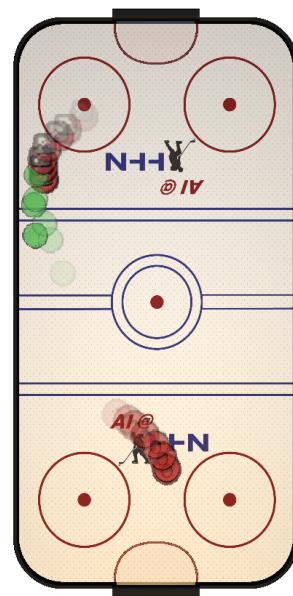
Abbildung 25a zeigt, dass der Agent, der für 72 Stunden ohne Domain-Randomization-Techniken trainiert wurde (*AgentFromScratch*), Schwierigkeiten in der randomisierten Umgebung bekommt. In der Abbildung wird deutlich, dass der Pusher sich in die Richtung des Pucks bewegt, jedoch den schräg von der Spielfeldmitte kommenden Puck verfehlt. Das Verpassen des Pucks führt unmittelbar zu einem Gegentreffer nach dem Abprallen des Pucks an der Bande. Die Weiterentwicklung des Agents mit Domain-Randomization-Techniken zeigt, dass der Agent in der randomisierten Umgebung lernt den Puck trotz des simulierten Rauschens der Werte der Observations zu lokalisieren. In Abbildung 25b ist zu erkennen, dass der Agent den Puck direkt an der Bande abfängt und in Richtung des Gegners zurück spielt.

Abbildung 26b zeigt, dass der Agent nach dem Training mit Domain-Randomization-Techniken dazu neigt sich direkt an der Bande oder abweichend von der Spielfeldmitte aufzuhalten. Daher wird ein weiteres Training mit angepasster Reward-Skalierung durchgeführt, um das Spielverhalten zu verbessern. Dieser Agent wird als *AgentDomainRandomizedContinued* bezeichnet. Die in Tabelle 6 aufgeführten Parameter werden für das Training gewählt. Die Bestrafung für Gegentreore und das Aufhalten an der Spielfeldbande wird erhöht, um zu erreichen, dass der Agent vermehrt aus der Spielfeldmitte agiert.

Die Reward-Entwicklung in Abbildung 27a zeigt, dass der Agent den totalen Reward in den ersten 65 Trainingsstunden kontinuierlich steigern kann. Anschließend stagniert der kumulative

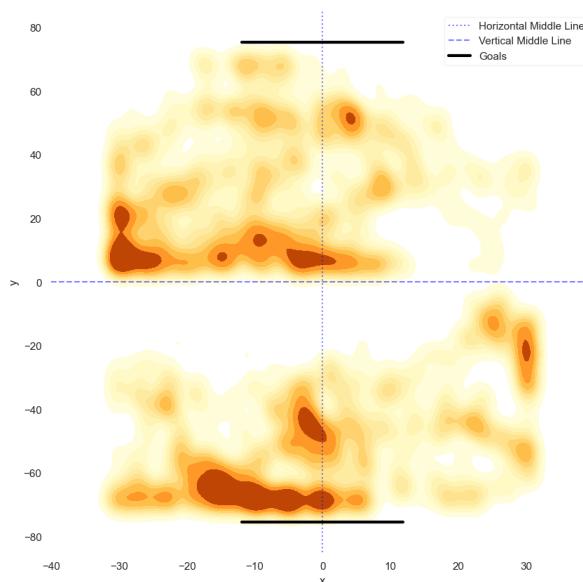


(a) Training ohne Domain Randomization

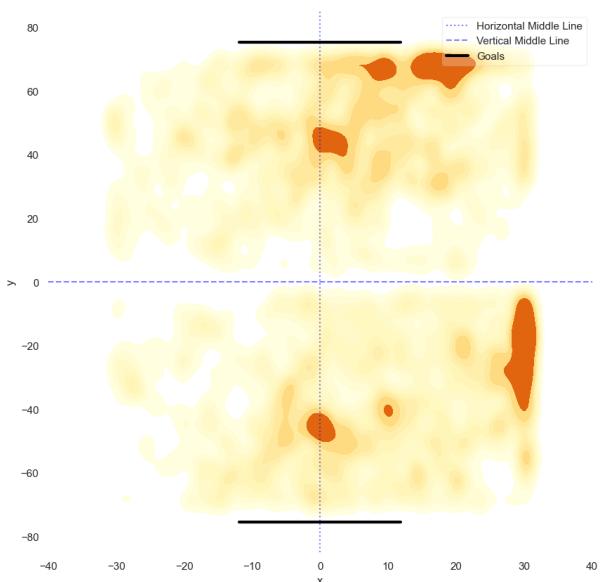


(b) Training mit Domain Randomization

Abbildung 25: Trajektorie-Beispiele für einen Agent mit (AgentDomainRandomized) und ohne (AgentFromScratch) Training mit Domain-Randomization-Techniken in einer randomisierten Simulationsumgebung.



(a) Oben: AgentDomainRandomized  
Unten: AgentFromScratch



(b) AgentDomainRandomized im Selfplay-Modus

Abbildung 26: Heatmap für einen Agent trainiert mit Domain-Randomization-Techniken (AgentDomainRandomized) in einer randomisierten Simulationsumgebung im Spiel gegen sich selbst und einen Agent ohne Training mit Domain-Randomization (AgentFromScratch).

Reward bis Stunde 100 bei einem Wert von vier. Daraufhin sinkt der Reward fast bis auf zwei ab. Bis zum Ende des Trainingslaufs nach zirka 200 Stunden steigt der Reward wieder

Tabelle 6: Trainingsparameter für ein Training eines Airhockey-Agents mit vortrainiertem Agent (AgentDomainRandomized) und Domain-Randomization-Techniken.

Reward	Wert
Agent erzielt Tor	10
Gegner erzielt Tor	-7
Gegnerische Rückwand	1
Puck in Agent-Hälfte	-0.1
Puck-Beschleunigung	0.15
Häufige Richtungsänderungen	-0.5
Spielfeldmitte	0
Banden vermeiden	-0.4
<b>Observations</b>	-
Observation-Space-Vektor-Länge	12
<b>Hyperparameter</b>	-
Anzahl Actor	12
Batch-Größe	64
Lernrate Actor $\alpha$	0.0005
Lernrate Critic $\beta$	0.0005
Discount-Faktor $\gamma$	0.99
Ziel-Netzwerk-Aktualisierungsfaktor $\rho$	300
Selfplay-Netzwerk-Aktualisierungsfaktor	5000
Replay-Buffer-Kapazität	300000
Entropie-Koeffizient $\tau$	0.01

auf einen Wert von vier. Der Verlauf der Value-Loss-Kurve in Abbildung 27b zeigt, dass der Reward sich bei ungefähr 150 Trainingsstunden stabilisiert. Die Entwicklung des Policy-Loss in Abbildung 27c lässt den Schluss zu, dass an der Policy nach 100 Stunden keine starken Veränderungen vorgenommen wurden.

In Abbildung 28 sind Heatmaps des Agents mit angepasster Reward-Skalierung im Spiel gegen sich selbst, den vorherigen Stand mit Domain Randomization und im Spiel gegen den Agent ohne Domain Randomization dargestellt. Alle Heatmaps zeigen, dass das Anpassen der Rewards den gewünschten Effekt erzielt und der Agent sich nun vermehrt in der Spielfeldmitte aufhält. Die Spielfeldränder werden nur noch aufgesucht, wenn die Spielsituation dies erfordert. Abbildung 28 zeigt außerdem, dass der Agent gelernt hat die *Dreieck-Strategie* anzuwenden, um das eigene Tor zu verteidigen. Bei der Dreieck-Strategie wird der Pusher in Form eines gleichschenkligen Dreiecks zwischen Puck und Tor bewegt. Beim Airhockey gilt es sich im wesentlichen gegen drei Arten von Schüssen zu verteidigen:

- Ein gerader Schuss auf das Tor
- Ein schräger Schuss auf das Tor

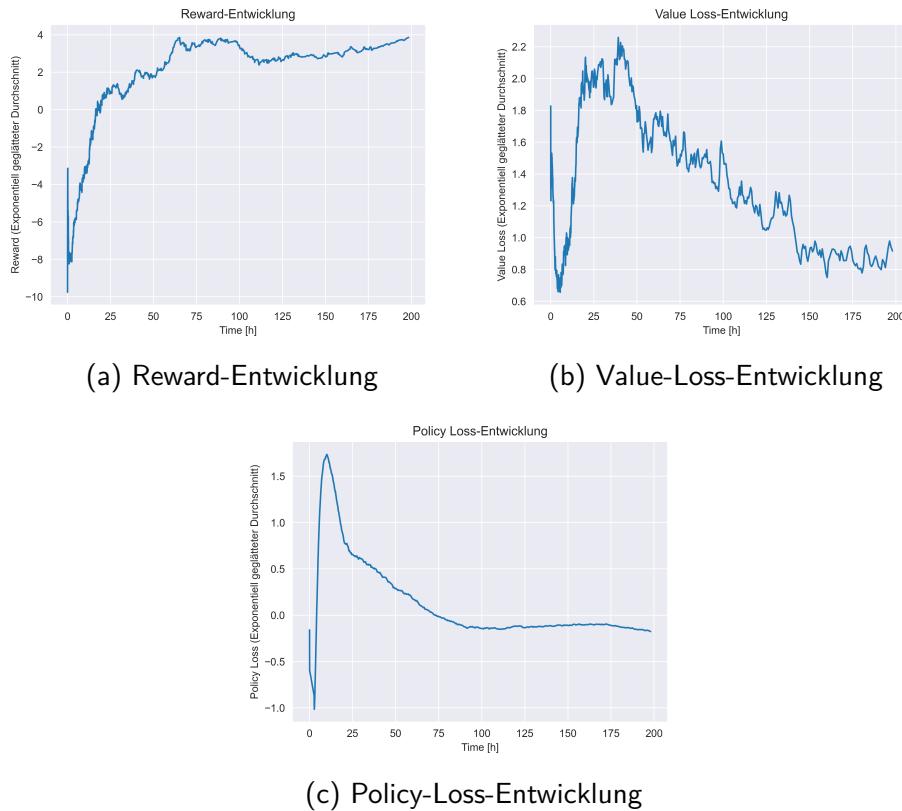


Abbildung 27: Entwicklung des Trainings zur Feinabstimmung mit vortrainierter Policy und Domain-Randomization-Techniken.

- Ein Schuss über Bande auf das Tor

Die Dreieck-Strategie ist eine gute Verteidigungsstrategie gegen die zuvor genannten Schussarten, weil stets kurze Wege nötig sind, unabhängig von der Schussart [54]. Der Bereich auf dem Spielfeld, der von der Dreieck-Verteidigungsstrategie abgedeckt wird ist in Abbildung 28 grün umrandet dargestellt.

Abbildung 29 zeigt Trajektorie-Beispiele für den verbesserten Agent (AgentDomainRandomizedContinued) im Spiel gegen sich selbst und im Spiel gegen seine Vorstufe den Agent-DomainRandomized. Die Trajektorie in Abbildung 29a zeigt, wie der Agent in der unteren Spielfeldhälfte den Puck auf Höhe der blauen Linie in seiner Hälfte anspielt und anschließend vor sein Tor zur Ausgangsposition der Dreieck-Strategie zurückkehrt. In Abbildung 29b ist der aktuelle Agent (AgentDomainRandomizedContinued) in der oberen Spielfeldhälfte gegen seine Vorentwicklung (AgentDomainRandomized) zu sehen. Die Darstellung zeigt, wie der Agent-DomainRandomizedContinued aus der Dreieck-Position einen kurzen Weg zum Puck hat. Der Agent in der oberen Spielfeldhälfte erkennt, dass sein Gegenüber sich an der rechten Bande befindet. Der AgentDomainRandomizedContinued hat gelernt den Puck schräg anzufahren, um ihn an die rechte Bande zu spielen. Der AgentDomainRandomized, der die Dreieck-Strategie

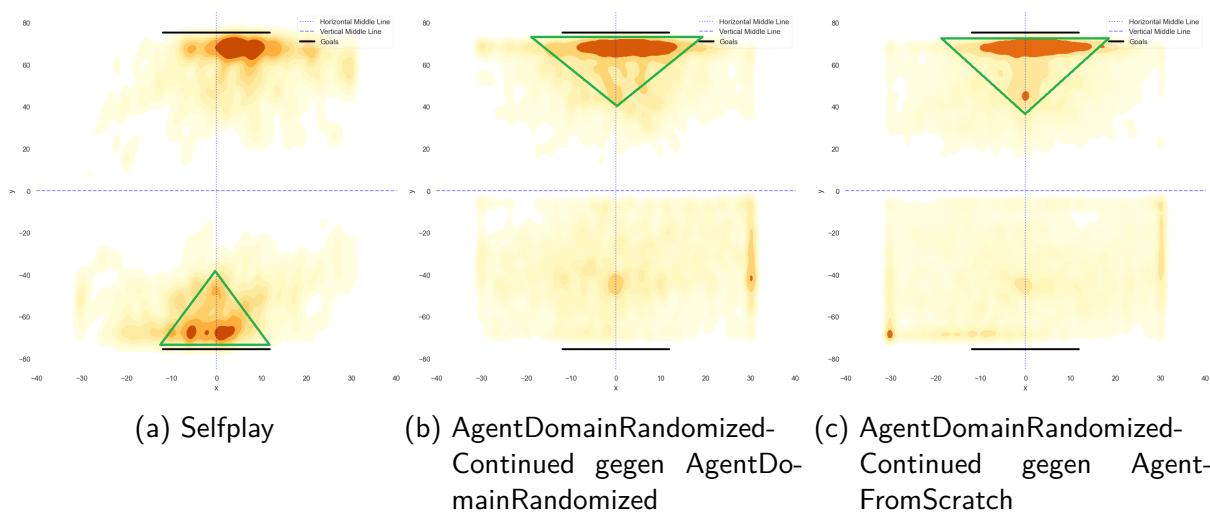


Abbildung 28: Heatmap für die Weiterentwicklung eines Agents trainiert mit Domain-Randomization-Techniken (AgentDomainRandomizedContinued) in einer randomisierten Simulationsumgebung im Spiel gegen sich selbst, seine vorherige Entwicklungsstufe (AgentDomainRandomized) und einen Agent ohne Training mit Domain Randomization (AgentFromScratch).

nicht gelernt hat, hat damit den maximalen Weg zum Abfangen des Pucks zurückzulegen. Dies gelingt nicht und der Schuss kann nicht verteidigt werden.

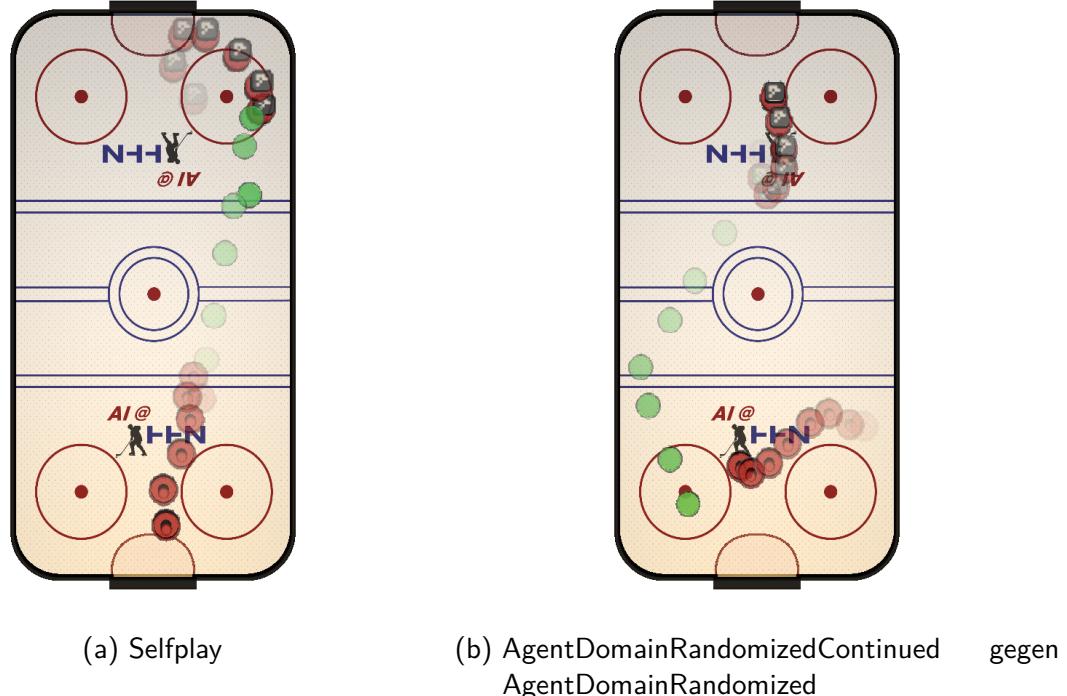


Abbildung 29: Trajektorie-Beispiele für einen verbesserten Agent im Training mit Domain-Randomization-Techniken (AgentDomainRandomizedContinued) in einer randomisierten Simulationsumgebung.

Zur Bestimmung der Ratings in Abbildung 30 wurde das Turnier in einer Simulationsumgebung gespielt, welche die zuvor beschriebenen Domain-Randomization-Techniken anwendet. Das Turnier wird gespielt mit dem nicht-randomisierten Agent `AgentFromScratchContinued` und den beiden Agents, die mit der Anwendung von Domain-Randomization trainiert wurden. Der `AgentFromScratchContinued` ist nach den Ratings in Abbildung 23 mit hoher Wahrscheinlichkeit der spielstärkste Agent. Daher wird er verwendet, um zu evaluieren, ob die Verwendung von Domain Randomization die Robustheit der Policy für abweichende Parameter der Physik und Observations erhöht. Der Agent wird dafür mit seinen bereits errechneten Rating-Daten initialisiert. Die Darstellung des Ratings bzw. der RD zeigt, dass der Agent-`DomainRandomized` in der randomisierten Simulationsumgebung bessere Leistungen als der nicht-randomisierte `AgentFromScratchContinued` erreicht. Nach 50 Rating-Perioden wird die Spielstärke des `AgentDomainRandomized` bei einem Rating von 2020 eingeschätzt. Der Agent-`FromScratchContinued` wird auf 1615 Rating-Punkte abgewertet. Unter Berücksichtigung der RD bzw. des Konfidenzintervalls ist der `AgentDomainRandomized` zu 95% um mindestens 219 Rating-Punkte stärker einzuschätzen. Des Weiteren zeigt Abbildung 30, dass der Agent-`DomainRandomizedContinued` sein Vorgängermodell mit einem Rating von 2728 übertreffen kann. Die RD ergibt, dass der Agent seinen Vorgänger zu 95% um mindestens 482 Punkte überlegen ist. Die Turnierergebnisse bestätigen damit die zuvor beschriebenen Beobachtungen der Heatmaps bzw. Trajektorie-Beispiele. Die Darstellung der Volatilität zeigt, dass der Agent-`FromScratchContinued`, ohne die Anwendung von Domain-Randomization-Techniken beim Training, in der randomisierten Umgebung deutlich inkonsistenter Ergebnisse erzielt als die anderen Turnierteilnehmer. Bei den Agents, die mit Domain-Randomization-Techniken trainiert wurden, zeigt sich, dass das Volatilitätsmaß weniger Ausschläge vorweist. Die konstanten Leistungen bestätigen, dass die Agents in der randomisierten Umgebung konstant ihr Spielniveau halten können. Sie sind demnach robust für sich verändernde Parameter. Die Domain Randomization konnte erfolgreich beim Training eines Airhockey-Agents mit dem SAC-Algorithmus angewandt werden.

Neben den zuvor analysierten erfolgreichen Trainingsläufen, wurden weitere Trainings durchgeführt, mit denen kein intelligenter Airhockey-Agent trainiert werden konnte. Die Trainingsparameter sind in Tabelle 7 dargestellt. Der *Agent 0* in Tabelle 7 wurde darauf ausgelegt häufige Richtungsänderungen zu vermeiden. Außerdem wurde der Selfplay-Netzwerk-Aktualisierungsfaktor sehr hoch gesetzt, um zu testen, ob der Agent in der Lage ist gegen einen gleichbleibenden Gegner zu lernen. Die Trainingsentwicklung in Abbildung 31 zeigt, dass der Agent nicht in der Lage ist, eine erfolgreiche Policy zu erlernen. Die Reward-Entwicklung zeigt keinen positiven Trend. Bei der Entwicklung des Policy-Loss ist kein fallender Trend erkennbar, was die Beobachtung eines erfolglosen Trainings bestätigt. Die übrigen Agents, welche in Tabelle 7 aufgeführt sind, wurden mit der gleichen ausgewogenen Reward-Skalierung trainiert. Diese entspricht der des `AgentFromScratch` in Tabelle 3. Der *Agent 1* und *Agent 2* wurden mit dem

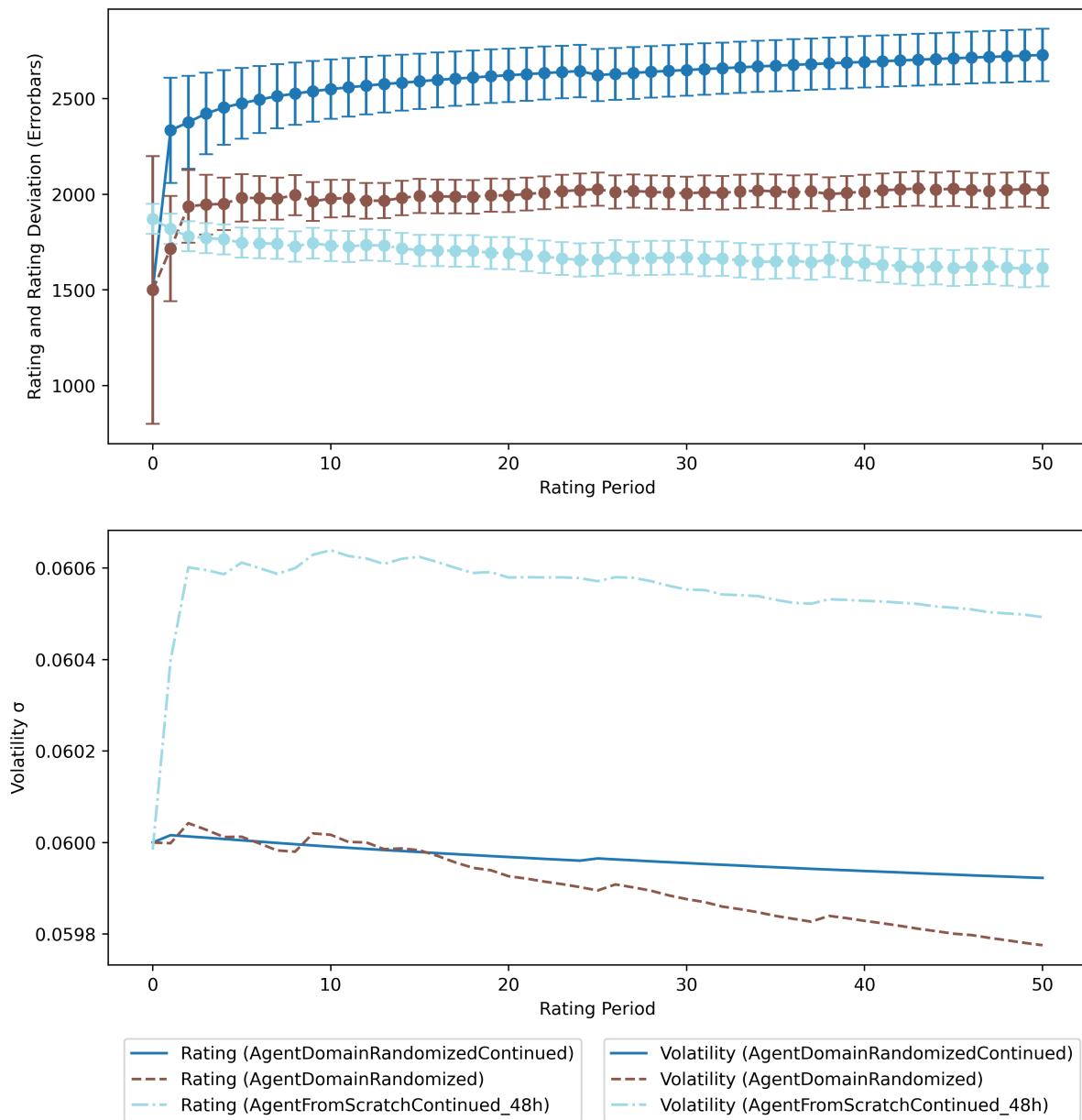


Abbildung 30: Glicko2-Ratings für ein Round-Robin-Turnier mit dem AgentFromScratch nach 72 Trainingsstunden, dem AgentFromScratchContinued nach 24 bzw. 48 Stunden, dem AgentDomainRandomized und AgentDomainRandomizedContinued.

maximalen Observation-Space-Vektor trainiert. Der Agent 2 mit einer erhöhten Batch-Große von 256. Der Agent 3 wurde mit einer im Vergleich zum AgentFromScratch (Batch-Größe 32) deutlich erhöhten Batch-Große von 512 trainiert. Zum Vergleich der Trainingsentwicklung der Agents wird die Trainingsentwicklung des AgentFromScratch in Abbildung 31 als Agent 4 dargestellt. Der Agent 1 und Agent 0 weisen im Vergleich zum AgentFromScratch einen höheren Gesamt-Reward auf. In den Kurven zur Value-Loss-Entwicklung in Abbildung 31b ist kein fallender Trend und somit keine Verbesserung der Approximation der Wertefunktion erkennbar. Die Policy-Loss-Entwicklung in Abbildung 31c zeigt für den Agent 1 einen fallenden Trend. Nach ungefähr 38 Trainingsstunden steigt die Kurve stark an. Bei einem erfolgreichen

Training sollte die Policy-Loss-Kurve stets einen fallenden Trend aufweisen, wie bei Agent 4 in Abbildung 31c. Der Policy-Loss des Agents 2 entspricht zumeist einem fallenden Trend. Im Vergleich zu Agent 4 (AgentFromScratch) zeigt die Policy-Loss-Entwicklung, dass Agent 2 länger benötigt, um zu einer Policy zu finden. Der Agent 3 benötigt im Vergleich zum Agent 4 (AgentFromScratch) länger um im geglätteten Durchschnitt einen positiven totalen Reward zu erzielen. Nach ungefähr 70 Trainingsstunden erreichen beide Agents einen Gesamt-Reward von zwei im Schnitt. Die Policy-Loss-Entwicklung des Agents 3 zeigt keinen fallenden Trend und somit auch kein positives Trainingsergebnis. Die Value-Loss-Entwicklung der Agents 2 und 3 zeigt, dass größere Batches zu weniger Ausschlägen des Value-Loss führen. Eine bessere Annäherung der Wertefunktion durch größere Batches ist nicht erkennbar. Die in diesem Abschnitt beschriebenen Ergebnisse können nicht abschließend klären, welche Zusammensetzungen von Rewards, Observations und Hyperparametern zu einem erfolgreichen Training führen. Wie in Islam et. al [55] und Helfenstein [56] untersucht, unterliegen Trainings mit RL-Methoden vielen Quellen von Instabilität und Varianz. Die Ergebnisse können nur Anhaltspunkte dafür geben, welche Trainingsparameterzusammensetzungen erfolgversprechend sind. Um eine Aussage über die Qualität der Trainingsparameter zu treffen, müssten diese mit einer größeren Anzahl an Trainingsläufen getestet werden [55].

Tabelle 7: Trainingsparameter erfolgloser Trainingsläufe.

<b>Reward</b>	<b>Agent 0</b>	<b>Agent 1</b>	<b>Agent 2</b>	<b>Agent 3</b>
Agent erzielt Tor	10	10	10	10
Gegner erzielt Tor	-5	-5	-5	-5
Gegnerische Rückwand	0	0.5	0.5	0.5
Puck in Agent-Hälfte	-0.15	-0.05	-0.05	-0.05
Puck-Beschleunigung	0.1	0.15	0.15	0.15
Häufige Richtungsänderungen	-1	-0.3	-0.3	-0.3
Spielfeldmitte	0	-0.025	-0.025	-0.025
Banden vermeiden	-0.32	-0.25	-0.25	-0.25
<b>Observations</b>	-	-	-	-
Observation-Space-Vektor-Länge	12	26	26	12
<b>Hyperparameter</b>	-	-	-	-
Anzahl Actor	8	12	12	12
Batch-Größe	32	32	256	512
Lernrate Actor $\alpha$	0.0005	0.0005	0.0005	0.0005
Lernrate Critic $\beta$	0.0005	0.0005	0.0005	0.0005
Discount-Faktor $\gamma$	0.99	0.99	0.99	0.99
Ziel-Netzwerk-Aktualisierungsfaktor $\rho$	200	300	300	300
Selfplay-Netzwerk-Aktualisierungsfaktor	100000	500	500	500
Replay-Buffer-Kapazität	1000000	300000	300000	300000
Entropie-Koeffizient $\tau$	0.01	0.01	0.01	0.01

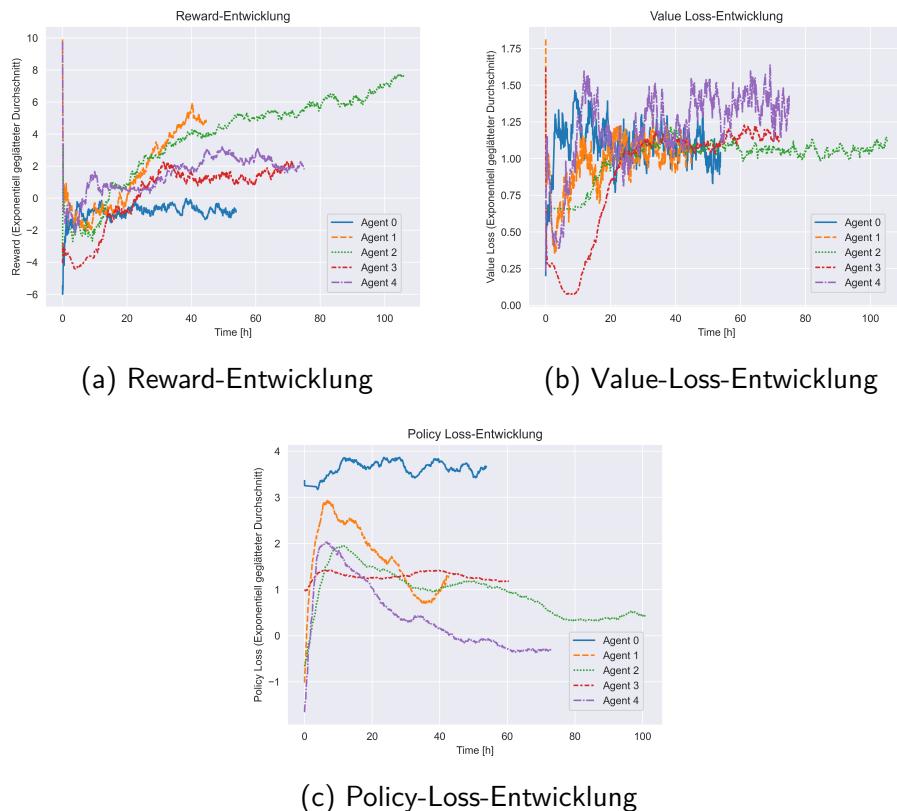


Abbildung 31: Entwicklung erfolgloser Trainingsläufe im Vergleich mit dem AgentFromScratch (Agent 4 in Grafik).

## 7 Zusammenfassung

Das Ziel der Arbeit ist es, einen Reinforcement Learning (RL)-Agent zu trainieren, der auf menschlichem Niveau gegen einen realen Spieler bei dem Geschicklichkeitsspiel Airhockey bestehen kann. Hierfür wurde eine Airhockey-Simulation weiterentwickelt, die die realen physikalischen Gegebenheiten des Airhockey-Tischs möglichst genau abbildet. Zu diesem Zweck war die Auswahl und Integration der Physik-Engine MuJoCo zur Verbesserung der Simulationstreue der physikalischen Eigenschaften der Airhockey-Objekte ein maßgeblicher Bestandteil. Die Implementierung eines Frameworks zur Anwendung von Domain-Randomization-Techniken bildet die Grundlage für den Transfer des RL-Agents vom simulierten zum realen Airhockey-Tisch. Zudem wurde die Simulation optisch ansprechend gestaltet und für den Menschen spielbar gemacht.

Das RL-Framework der Hochschule Heilbronn (HHN), Modular Reinforcement Learning (MRL), wurde um Funktionen erweitert, die den Trainingsprozess optimieren und die Evaluation der Trainingsergebnisse ermöglichen. Dies beinhaltete die Einführung des Glicko2-Algorithmus zur Bewertung der relativen Spielstärke der Agents. Eine Command-Line-Schnittstelle und Exportoptionen für Reward- und Observation-Daten erleichtern die Handhabung des Framework während des Trainingsprozesses. Eine Funktion zum Erstellen grafischer Darstellungen für Trainings- und Rating-Daten vereinfacht die Visualisierung der Trainingsergebnisse.

Für das Training des Agents wurde der Soft Actor-Critic (SAC)-Algorithmus verwendet. Die Zusammensetzung der Reward-Struktur und die ausgewählten Hyperparameter, die zum Training des Airhockey-Agents verwendet wurden, wurden vorgestellt.

Die Evaluation der Ergebnisse zeigt, dass die Zielsetzung erreicht werden konnte. Es wurde ein Airhockey-Agent trainiert, der eine fortgeschrittene Spieleintelligenz aufweisen kann. Die Arbeit veranschaulicht, wie ein RL-Agent von Grund auf trainiert werden kann. In den ersten 24 Trainingsstunden kann der Agent keine intelligente Spielstrategie lernen. Nach 48 Trainingsstunden konnten wesentliche Fortschritte festgestellt werden. Der Agent zeigt sich deutlich verbessert im Antizipieren der Puck-Trajektorie. Weitere 24 Stunden führten den Agent zu einem fortgeschrittenen Spielniveau. Er kann nun die Puck-Trajektorie antizipieren und den Puck gezielt in Richtung des gegnerischen Tores spielen. Der Agent platziert sich je nach Spielsituation sinnvoll in seiner Spielfeldhälfte. Die Bewertung der relativen Spielstärke des Agents mit dem Glicko2-Algorithmus in 24-Stunden-Intervallen zeigt, dass die Policy sich im Zeitraum von 24 auf 48 Stunden Trainingszeit, am stärksten verbessert. Anschließend kann durch weiteres Training die Policy weiter optimiert werden. Allerdings zeigen die berechneten Glicko2-Ratings, dass die relative Spielstärke mit zunehmender Trainingsdauer geringer steigt.

Zur Überwindung des Domain-Gap wurden verschiedene Methoden zur Domain-Randomization implementiert. Die Techniken umfassen das Randomisieren der Puck-Detektion, den physikalischen Parametern der Airhockey-Umgebung und den Aktionen des Agents. Die Ergebnisse zeigen, dass der Agent lernt, die veränderten Umgebungsparameter zu berücksichtigen. Der Vergleich zwischen Agents, die mit Domain-Randomization trainiert wurden und Agents, die herkömmlich trainiert wurden, bestätigt den Lerneffekt. Die randomisierten Agents haben gelernt, ihre Trajektorie auf Unsicherheit in der Genauigkeit der Puck-Detektion und verändertes Kontaktverhalten der Objekte anzupassen. Darüber hinaus ist erkennbar, dass das Spielverhalten durch das Training weiter verbessert wird. Die Robustheit und Generalisierungsfähigkeit des Agents wird durch die Anwendung von Domain-Randomization-Techniken erhöht.

Zusammenfassend zeigt diese Arbeit die Eignung von RL zur Anwendung in komplexen und dynamischen Umgebungen, wie der des Airhockey-Spiels. Die Ergebnisse unterstreichen das Potential von RL und bieten die Grundlage für weitere Forschung und Entwicklungen in diesem Bereich. Der in Kapitel 8 folgende Ausblick beschreibt mögliche Ansätze für zukünftige Untersuchungen.

## 8 Ausblick

Aus den im Zuge dieser Arbeit gewonnenen Ergebnissen und Erkenntnissen eröffnen sich neue Möglichkeiten für die Weiterentwicklung der Airhockey-Simulation und des MRL-Framework. Außerdem bieten sich Anknüpfungspunkte für zukünftige Forschungen im Bereich RL. Im Folgenden werden Empfehlungen für potenzielle Entwicklungen gegeben.

Die im Verlauf dieser Arbeiten trainierten Agent-Versionen können als Benchmarks für weitere Airhockey-Agents herangezogen werden. Mit Hilfe des Rating-Algorithmus kann dann ermittelt werden, ob Agents mit anderen Trainingsparametern in der gleichen Trainingszeit bessere Ergebnisse erzielen bzw. ab welcher Trainingszeit die Benchmark übertroffen wird.

Eine zusätzliche Möglichkeit die Trainingsparameter zu optimieren ist der Einsatz eines Hyperparameter-Tuners. Die manuelle Suche nach optimalen Hyperparametern ist zeitaufwendig und fehleranfällig. Der Einsatz eines Hyperparameter-Tuners ermöglicht eine konsistente Suche durch Algorithmen wie *Grid-Search* oder *Random-Search* [57]. Außerdem können fortgeschrittene Tuning-Methoden, wie *Bayes'sche Optimierung*, die Effizienz des Prozesses der Hyperparameter-Suche erhöhen [57]. Das Framework *Ray* [58], das im MRL bereits zur Parallelisierung verwendet wird, integriert mit *Ray Tune* eine Erweiterung zum Hyperparameter-Tuning. Die Bibliothek unterstützt Algorithmen nach aktuellem Stand der Technik zur Hyperparameter-Optimierung und ist mit dem MRL kompatibel [59].

Der Trainingsprozess bei Verwendung des MRL kann beschleunigt werden, indem der Ansatz zur Parallelisierung des Trainings erweitert wird. Nach aktuellem Stand bietet das Framework die Möglichkeit mehrere Instanzen der verwendeten Simulationsumgebung zum Training zu nutzen. Das Modell kann so in kürzerer Zeit mehr Zustände erfahren und lernt demnach schneller. Die Anzahl an parallelen Instanzen ist jedoch durch die Anzahl verfügbarer Central Processing Unit (CPU)-Kerne begrenzt. Die verfügbaren Actor können erhöht werden, wenn das MRL das parallele Training auf mehreren Maschinen zur gleichen Zeit unterstützt. Eine weitere Möglichkeit zur Skalierung ist eine Kompatibilität des Frameworks mit Cloud-Diensten herzustellen [60].

Der in Kapitel 6 beschriebene Bereich, in dem zufällige Werte im Zuge der Domain-Randomization generiert wurden, ist nicht optimiert auf den realen Demonstrator. Zum Zeitpunkt der Anfertigung dieser Arbeit sind die Parameter des realen Demonstrators nicht vollständig bekannt. Beispielsweise kann der Bereich, in dem die Observations randomisiert werden eingeschränkt werden, wenn die Puck-Prädiktion mit dem Kalman-Filter (vgl. Abbildung 1) einsatzbereit ist. Die maximale Schätzung der Unsicherheit mit dem Kalman-Filter und deren Verteilung entspricht dem optimalen Wertebereich für die Domain-Randomization der Observations [61]. Etwaige Verzögerungen bei der Ansteuerung der Motoren der Kinematik am

realen Demonstrator, die bei der Action-Randomization berücksichtigt werden müssen, sind ebenfalls unbekannt. Eine geringere Verteilung der Wertebereiche würde die Komplexität des Lernens verringern [19].

Weitere Möglichkeiten zur Weiterentwicklung der Domain-Randomization-Techniken ist die Verwendung spezieller Algorithmen zur Steuerung der Randomisierung. Ein Ansatz ist die *Active Domain Randomization* nach Mehta et al [18]. Dabei wird nach konkreten Schwächen in der Policy bei bestimmten Parameterkonfigurationen im vordefinierten Wertebereich gesucht. Diese Konfigurationen wird dann zusätzliche Trainingszeit eingeräumt. Andere Ansätze, wie die *Self-Paced Domain Randomization* nach Damken et al [19], setzen auf *Curriculum-Learning*-Methoden [20].

Offline RL verfolgt einen anderen Ansatz, als der in dieser Arbeit gewählte Weg den Agent in einer Simulationsumgebung zu trainieren und anschließend mit Domain-Randomization-Techniken auf den realen Demonstrator zu übertragen. Beim Offline RL werden bereits gesammelte Daten – bevorzugt in der Zielumgebung – verwendet, um die Policy zu trainieren [62]. Der in dieser Arbeit verwendete Ansatz ist ein Online RL. Das Sammeln von Daten am realen Demonstrator und anschließende Training mit Offline-RL-Techniken kann genutzt werden, um die Anwendung von Offline RL im MRL-Umfeld zu evaluieren. Der zuvor beschriebene Ansatz mit Benchmark-Agents kann genutzt werden, um die Eignung von Offline RL einzuschätzen. Die Agents, die mit Domain-Randomization-Techniken trainiert wurden, eignen sich, um zu überprüfen, ob in der Zielumgebung gesammelte Daten den Domain Gap schließen.

Der im Rahmen dieser Arbeit implementierte Glicko2-Algorithmus eignet sich, wie in Kapitel 4.1 beschrieben, zur Bewertung von RL-Agents in Nullsummenspielen ohne Berücksichtigung des MoV. In zukünftigen Arbeiten könnte der Algorithmus erweitert werden den MoV zu berücksichtigen, sodass er für jede Anwendung angepasst wird. Des Weiteren können Ansätze die Berechnung des MoV zu generalisieren, wie in Szczerbinski [63] beschrieben, Anwendung finden. Zusätzlich besteht die Möglichkeit, weitere Bewertungsalgorithmen für RL-Agenten in das MRL-Framework einzubinden. Zusätzlich besteht die Möglichkeit, weitere Bewertungsalgorithmen für RL-Agenten in das MRL-Framework einzubinden. Zur Berücksichtigung von Mehrspieler-Anwendungen können Ansätze wie der TrueSkill-Algorithmus [38] betrachtet werden. Airhockey ist ein symmetrisches Spiel, bei dem beide Spieler die gleichen Aktionen ausführen können. In verschiedenen Brettspielen, Videospielen oder Sportarten ist die Umgebung asymmetrisch. Dass heißt die Spieler haben unterschiedliche Aktionen zur Verfügung oder beginnen die Episode mit einem Handicap. Der Rating-Algorithmus kann angepasst werden, um die Bewertung von Agents in asymmetrischen Umgebungen zu ermöglichen [49].

## Literaturverzeichnis

- [1] N. Stache and P. Graf, "Autonomous systems: Deep learning reinforcement learning introduction," 2022. Nicht öffentlich zugänglich.
- [2] E. S. de Lima, "Artificial intelligence - lecture 07 – steering behaviors," 2020.
- [3] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," 2017.
- [4] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.
- [5] R. Sutton and A. Barto, *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series, MIT Press, 2018.
- [6] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.
- [7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.
- [8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.
- [9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.
- [10] J. C. de Jesus, V. A. Kich, A. H. Kolling, R. B. Grando, M. A. d. S. L. Cuadros, and D. F. T. Gamarra, "Soft actor-critic for navigation of mobile robots," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 2, p. 31, 2021.
- [11] C.-C. Wong, S.-Y. Chien, H.-M. Feng, and H. Aoyama, "Motion planning for dual-arm robot based on soft actor-critic," *IEEE Access*, vol. 9, pp. 26871–26885, 2021.
- [12] L. Weng, "Domain randomization for sim2real transfer," *lilianweng.github.io*, 2019.
- [13] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Mai 2018.
- [14] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," 2017.
- [15] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," 2017.
- [16] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," 2017.

- [17] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *CoRR*, 2018.
- [18] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," 2019.
- [19] F. Damken, H. Carrasco, F. Muratore, and J. Peters, "Self-paced domain randomization,"
- [20] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," vol. 60, p. 6, 06 2009.
- [21] F. Muratore, C. Eilers, M. Gienger, and J. Peters, "Bayesian domain randomization for sim-to-real transfer," 03 2020.
- [22] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," Mai 2019.
- [23] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015.
- [24] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," 2020.
- [25] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," 2018.
- [26] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," 2017.
- [27] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
- [28] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2020.
- [29] Nvidia, "Physx." <https://developer.nvidia.com/physx-sdk>, 2023. [Zugriff am 31.03.2023].
- [30] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.
- [31] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
- [32] A. E. Elo, *The Rating of Chessplayers, Past and Present*. New York: Arco Pub., 1978.
- [33] "B. Permanent Commissions / 02. FIDE Rating Regulations (Qualification Commission) / FIDE Rating Regulations effective from 1 July 2017 till 31 December 2021 (with amendments effective from 1 February 2021) / FIDE Handbook — handbook.fide.com." <https://handbook.fide.com/chapter/B022017>. [Zugriff am 08.04.2023].

- [34] M. Glickman, "A comprehensive guide to chess ratings," in *American Chess Journal*, 3, pp. 59–102, 1995.
- [35] M. Glickman, "Example of the glicko-2 system," 2022.
- [36] M. Glickman, "The glicko system," 2016.
- [37] D. R. (DanielRensch), "Better Than Ratings? Chess.com's New 'CAPS' System — chess.com." <https://www.chess.com/article/view/better-than-ratings-chess-com-s-new-caps-system>, 2017. [Zugriff am 24.05.2023].
- [38] R. Herbrich, T. Minka, and T. Graepel, "Trueskill(tm): A bayesian skill rating system," in *Advances in Neural Information Processing Systems 20*, pp. 569–576, MIT Press, Januar 2007.
- [39] T. Minka, R. Cleven, and Y. Zaykov, "Trueskill 2: An improved bayesian skill rating system," Tech. Rep. MSR-TR-2018-8, Microsoft, March 2018.
- [40] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Januar 2016.
- [41] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," Dezember 2017.
- [42] S. Karakovskiy and J. Togelius, "The mario ai benchmark and competitions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 55–67, 2012.
- [43] S. Wang, D. Jia, and X. Weng, "Deep reinforcement learning for autonomous driving," 2019.
- [44] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," *CoRR*, vol. abs/1912.06680, 2019.
- [45] Y. Tassa, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," Mai 2015.
- [46] C. Reynolds, "Steering behaviors for autonomous characters," Juni 2002.
- [47] I. Millington and J. Funge, *Artificial Intelligence for Games*. CRC Press, 2009.
- [48] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," 2022.

- [49] B. Wise, "Elo ratings for large tournaments of software agents in asymmetric games," 2021.
- [50] M. Ingram, "How to extend elo: a bayesian perspective," *Journal of Quantitative Analysis in Sports*, vol. 17, pp. 203 – 219, 2021.
- [51] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.
- [52] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," 2016.
- [53] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," 2019.
- [54] AirHockeyNerds, "Air hockey strategy: 7 tips to win every time." <https://airhockeynerds.com/air-hockey-strategy/>, 2023. [Zugriff am 07.05.2023].
- [55] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," 2017.
- [56] F. Helfenstein, "Benchmarking deep reinforcement learning algorithms." [https://www.ias.informatik.tu-darmstadt.de/uploads/Team/DavideTateo/felix\\_thesis.pdf](https://www.ias.informatik.tu-darmstadt.de/uploads/Team/DavideTateo/felix_thesis.pdf), 2021.
- [57] P. L. Bajo, "Hyperparameters in Deep RL — towardsdatascience.com." <https://towardsdatascience.com/hyperparameters-in-deep-rl-f8a9cf264cd6>. [Zugriff am 17.05.2023].
- [58] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging ai applications," 2018.
- [59] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.
- [60] "Massively Large-Scale Distributed Reinforcement Learning with Menger — ai.googleblog.com." <https://ai.googleblog.com/2020/10/massively-large-scale-distributed.html>. [Zugriff am 17.05.2023].
- [61] D. Tomer, "What I Was Missing While Using The Kalman Filter For Object Tracking — towardsdatascience.com." <https://towardsdatascience.com/what-i-was-missing-while-using-the-kalman-filter-for-object-tracking-8e4c29f6b795>. [Zugriff am 18.05.2023].

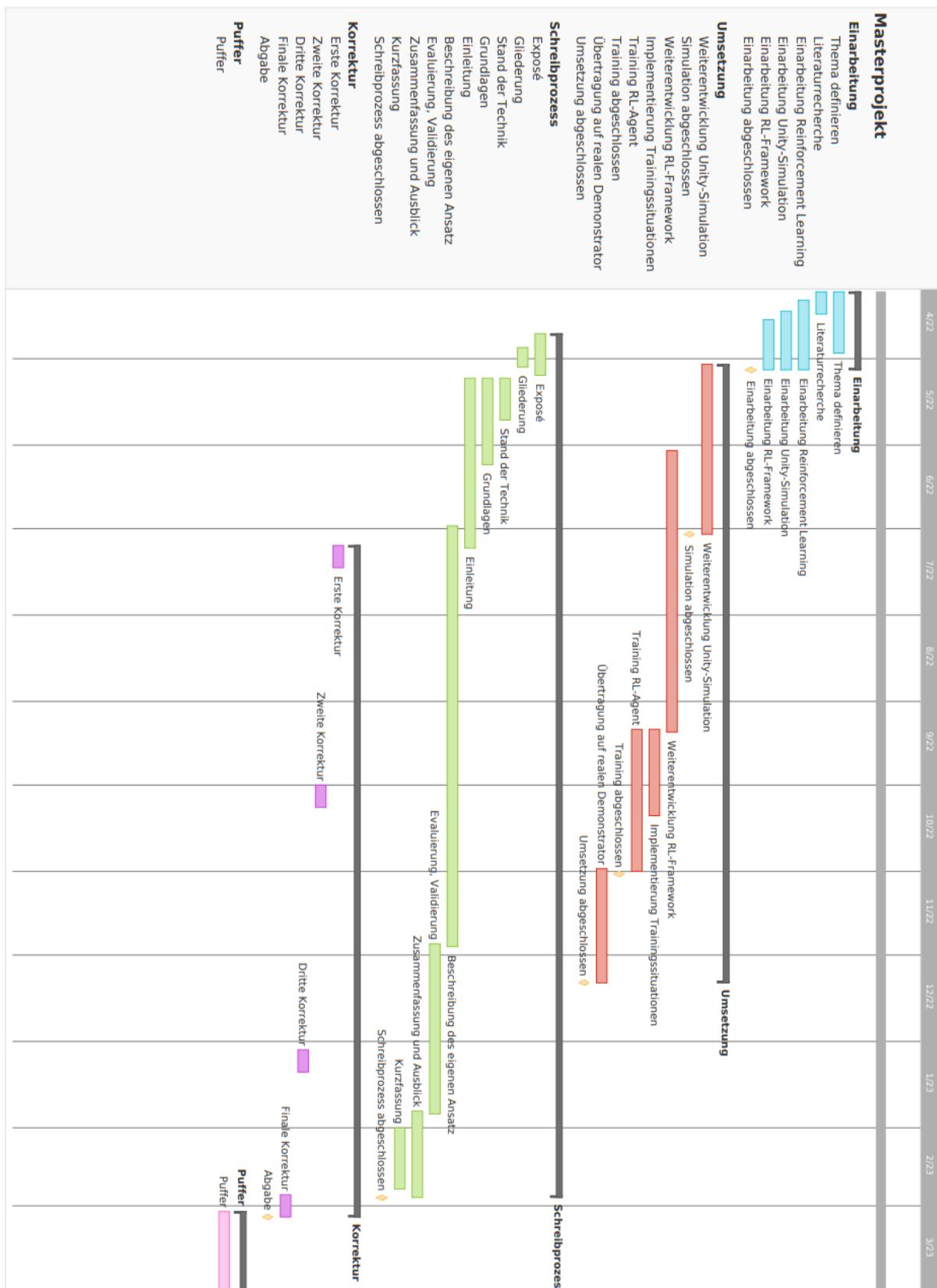
- [62] S. Levine, A. Kumar, G. Tucker, and J. Fu, “Offline reinforcement learning: Tutorial, review, and perspectives on open problems,” 2020.
- [63] L. Szczecinski, “G-elo: Generalization of the elo algorithm by modelling the discretized margin of victory,” 2022.



## A. Hyperparameter

Hyperparameter	Beispiel	Erklärung
Recurrent	true	Verwendung von rekurrenten Netzwerken
BurnIn	0	Anzahl der Burn-In-Schritte
AdaptiveSequenceLength	false	Anpassung der Sequenzlänge
ActionFeedback	false	Rückmeldung der Aktion an den Agenten
RewardFeedback	false	Rückmeldung der Belohnung an den Agenten
PolicyFeedback	false	Rückmeldung der Policy an den Agenten
SequenceLength	12	Länge der Sequenzen
Overlap	6	Überlappung der Sequenzen
ActorNum	8	Anzahl der parallelen Actor
BatchSize	32	Größe der Minibatches
ClipGrad	10.0	Gradientenclipping-Schwellenwert
FeatureSpaceSize	18	Größe des Feature-Space
CuriosityScalingFactor	0.03	Curiosity-Skalierungsfaktor
LearningRate	0.00005	Lernrate
Epsilon	0.15	Epsilon für Exploration
EpsilonDecay	0.9995	Epsilon-Abklingfaktor
EpsilonMin	0.01	Minimales Epsilon
StepDown	false	Abklingen der Lernrate
ObservationNormalization	false	Normalisierung der Beobachtungen
kNearest	10	Anzahl der nächsten Nachbarn
MaxIntRewardScaling	0.3	Max. Skalierungsfaktor für Belohnungen
ClusterDistance	0.008	Abstand zwischen Clustern
KernelEpsilon	0.01	Epsilon für den Kernel
MaximumSimilarity	4	Maximale Similarity
EpisodicMemoryCapacity	600	Kapazität des Episoden-Speichers
Gamma	0.99	Diskontierungsfaktor
LearningRateActor	0.0005	Lernrate des Actors
LearningRateCritic	0.0005	Lernrate des Critics
LogAlpha	0.0	Logarithmus von Alpha
NSteps	4	Anzahl der Schritte für die Berechnung des Ziels
AdaptiveExploration	false	Anpassung der Exploration
NetworkUpdateFrequency	300	Häufigkeit der Netzwerkaktualisierung
SelfPlayNetworkUpdateFrequency	1000	Häufigkeit der Klon-Netzwerkaktualisierung
Filters	32	Anzahl der Filter

## B. Gantt-Diagramm



## C. Glicko2-Algorithmus

Schritte des Glicko2-Algorithmus nach Glickman [35]:

1. Initialisiere Spieler mit dem Rating 1500, RD 350 und Volatilität 0.06 oder bereits existierenden Werten. Bestimme die Systemkonstante  $\tau$  zwischen 0.3 und 1.2.
2. Skalierung von  $R$ :

$$\mu = (r - 1500) \cdot q = (r - 1500) \cdot \frac{\ln(10)}{400} = \frac{r - 1500}{173,7178}$$

3. Berechnung der Größe  $v$  (Geschätzte Varianz des Spieler-Ratings basierend auf dem Spielergebnis):

$$v = \left( \sum_{j=1}^m g_j^2 \cdot E_j \cdot (1 - E_j) \right)^{-1}$$

Gewichtungsfaktor  $g$ , der die Varianz der Verteilung beeinflusst:

$$g(\phi_j) = \frac{1}{\sqrt{1 + \frac{3}{\pi^2} \phi_j^2}} = g_j$$

Gewinnwahrscheinlichkeit  $E$ :

$$E_j = \frac{1}{1 + e^{-g(\phi_j)(\mu - \mu_j)}}$$

4. Berechne  $\Delta$ , die geschätzte Änderung des Ratings im Vergleich zur Rating-Periode davor.

$$\Delta = v \sum_{j=1}^m g(\phi_j) (s_j - E(\mu, \mu_j, \phi_j))$$

5. Update der Rating-Abweichung auf den neuen Wert vor der Bewertungsperiode (vorläufige Abweichung)  $\phi^*$ :

a)  $a = \ln(\sigma^2)$ , und definiere

$$f(x) = \frac{e^x (\Delta^2 - \varphi^2 - v - e^x)}{2(\varphi^2 + v + e^x)^2} - \frac{x - a}{\tau^2}$$

Definiere Konvergiertoleranz  $\varepsilon$ . Ein Wert von  $\varepsilon = 0.000001$  ist eine ausreichend kleine Toleranz.

b) Bestimme die Initialwerte des iterativen Algorithmus:

- $A = a = \ln(\sigma^2)$
- Wenn  $\Delta^2 > \varphi^2 + v$ , dann ist  $B = \ln(\Delta^2 - \varphi^2 - v)$ . Wenn  $\Delta^2 \leq \varphi^2 + v$ , dann führe folgende Iteration durch:

- i.  $k = 1$
- ii. Wenn  $f(a - k\tau) < 0$ , dann
  - $k \leftarrow k + 1$
  - Gehe zu (ii).
- iii.  $B = a - k\tau$ .

c)  $f_A = f(A)$  und  $f_B = f(B)$ .

d) Solange  $|B - A| > \varepsilon$ , führe die folgenden Schritte aus:

- i.  $C = A + (A - B)f_A/(f_B - f_A)$  und  $f_C = f(C)$ .
- ii. Wenn  $f_C f_B \leq 0$ , dann  $A \leftarrow B$  und  $f_A \leftarrow f_B$ ; ansonsten ist  $f_A \leftarrow f_A/2$ .
- iii.  $B \leftarrow C$  und  $f_B \leftarrow f_C$ .
- iv. Stoppe wenn  $|B - A| \leq \varepsilon$ . Ansonsten wiederhole die oberen drei Schritte.

e) Sobald  $|B - A| \leq \varepsilon$  ist, setze

$$\sigma' \leftarrow e^{A/2}$$

6. Update Abweichung  $\phi'$ :

$$\phi' = \frac{1}{\sqrt{\frac{1}{\phi^{*2}} + \frac{1}{v}}}$$

7. Update Rating  $\mu'$ :

$$\mu' = \mu + \phi'^2 \cdot \sum_{j=1}^m g_j \cdot (s_j - E_j)$$

8. Konvertiere Rating und Abweichung zurück:

$$r' = 173.7178\mu' + 1500$$

$$\phi' = 173.7178\phi'$$