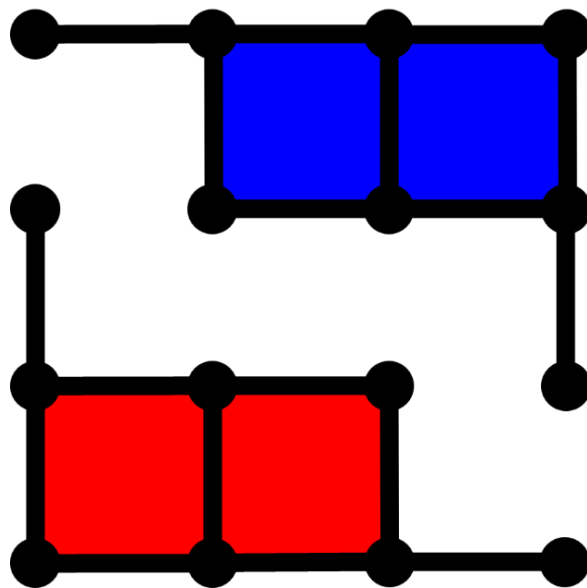




Analyse eines Deep-Q-Learning-Agenten im Spiel Käsekästchen



Maturaarbeit

von

Fabian Samuel Rütschi, Klasse M26a

betreut durch

Stefan Rothe und Dardana Jaha

Gymnasium Kirchenfeld, Abteilung MN

Bern

Oktober 2025

Inhaltsverzeichnis

1	Einleitung.....	4
2	Theoretische Grundlagen	5
2.1	<i>Reinforcement Learning</i> (RL)	5
2.2	<i>Deep Q-Learning</i>	7
2.2.1	Klassisches <i>Q-Learning</i>	7
2.2.2	Erweiterung durch neuronale Netzwerke	9
2.2.3	<i>Multilayer Perceptron</i> (MLP)	10
2.3	Python-Bibliotheken.....	12
2.3.1	<i>Gymnasium</i> und <i>Stable-Baselines3</i>	12
2.3.2	Sonstige Bibliotheken.....	13
3	Produktbeschreibung	14
3.1	Das Spiel Käsekästchen	14
3.2	Umsetzung des Spiels als <i>Gymnasium</i> -Umgebung	15
3.2.1	Der Beobachtungsraum	15
3.2.2	Der Aktionsraum	16
3.2.3	Informationsmethode	17
3.2.4	Schrittmethode.....	17
3.3	Implementierung der Künstlichen Intelligenz	17
4	Methodik.....	19
4.1	Trainingskonzept	19
4.2	Versuchsaufbau Experiment 1 (Belohnungsfunktionen)	20
4.3	Versuchsaufbau Experiment 2 (Hyperparameterwerte)	21
4.4	Evaluierungsmethoden	22
5	Ergebnisse und Diskussion	24
5.1	Experiment 1: Vergleich der Belohnungsfunktionen.....	24
5.1.1	Erfasste Daten zu Experiment 1	24
5.1.2	Vergleich und Interpretation der Belohnungsfunktionen.....	25
5.2	Experiment 2: Vergleich variierten Hyperparameterwerte	26
5.2.1	Erfasste Daten zu Experiment 2	26
5.2.2	Vergleich und Interpretation der Lernraten	29
5.2.3	Vergleich und Interpretation der Diskontierungsfaktoren	30

5.2.4	Vergleich und Interpretation der Explorationsphasenanteile.....	31
6	Fazit.....	33
6.1	Zentrale gewonnene Erkenntnisse	33
6.2	Kritische Reflexion und Verbesserungspotenzial	34
	Literaturverzeichnis	36
	Abbildungsverzeichnis	39
	Hilfsmittelverzeichnis	42
	Danksagung	43
	Anhang.....	44
	Übersicht über die trainierten Modelle.....	44
	Anleitung und Zugriff	44

1 Einleitung

Heute ist künstliche Intelligenz kaum mehr wegzudenken. Fast jede und jeder hat sie schon einmal benutzt. Sei es, um den eigenen Wissensdurst zu stillen, eine Reise zu planen oder um ein gutes Rezept zu finden. Häufig handelt es sich bei den dabei genutzten KI-Modellen um grosse Sprachmodelle. Es lassen sich aber für nahezu jede Problemstellung KI-Modelle trainieren. Ein verbreiteter dafür genutzter Algorithmus ist das *Deep Q-Learning*. Bei diesem lassen sich einzelne Hyperparameter anpassen, um das Training zu optimieren. Gleichzeitig erhalten sogenannte Agenten Belohnungen, abhängig davon, wie sich ihre gewählten Aktionen auf die Trainingsumgebung auswirken. Wofür genau diese Belohnungen ausgeschüttet werden, wird durch sogenannte Belohnungsfunktionen definiert.

Doch wie beeinflussen diese Hyperparameter tatsächlich das Training bei einem Spiel wie «Käsekästchen» und welche Auswirkungen haben verschiedene Belohnungsfunktionen auf das Lernen? Ziel meiner Maturaarbeit ist es, mithilfe eigens auf das Spiel Käsekästchen trainierter KI-Modelle auf diese Fragen einzugehen und sie mithilfe der durch Vergleiche gewonnenen Erkenntnisse zu beantworten.

In einem ersten Teil werden die theoretischen Grundlagen und Prinzipien der verwendeten künstlichen Intelligenz erläutert. Der zweite Teil setzt sich dann mit dem Spiel «Käsekästchen», der Methodik sowie dem Auswerten und Interpretieren der Daten auseinander. Im letzten Teil werden gewonnene Erkenntnisse noch einmal zusammengefasst und die Fragestellungen werden gezielt beantwortet. Zuletzt wird kritisch über diese Maturaarbeit reflektiert.

Um die Vergleichsdaten zu sammeln, werden neun KI-Modelle auf das Spiel Käsekästchen trainiert. Dabei handelt es sich um ein rundenbasiertes Zwei-Personen-Spiel, bei dem abwechselnd ein Strich gezogen wird. Ziel ist es, möglichst viele Kästchen, welche auf allen vier Seiten von Strichen umgeben sind, für sich zu gewinnen. Mithilfe der Python-Bibliothek *Gymnasium* wurde das Spiel Käsekästchen als Trainingsumgebung programmiert, in der die KI-Modelle trainiert werden können. Die Grundstrukturen der Modelle werden mithilfe der Python-Bibliothek *Stable-Baselines3* implementiert.

Im Verlauf dieser Maturaarbeit wird auf genannte Aspekte ausführlich eingegangen und sie werden fundiert erklärt. Mithilfe des vorhergehend aufgebauten Wissens werden dann gesammelte Daten interpretiert und abschliessend in einem Fazit die Antworten auf die Fragestellungen festgehalten.

2 Theoretische Grundlagen

Um besser nachvollziehbar zu machen, wie eine künstliche Intelligenz das Spiel Käsekästchen erlernen kann, werden in diesem Kapitel die theoretischen Grundlagen dafür erläutert. Dabei wird die Theorie ausgehend vom allgemeinen Lernprinzip des *Reinforcement Learnings* bis zum spezifischen *Deep-Q-Network*-Algorithmus, welcher in dieser Arbeit verwendet wurde, Schritt für Schritt erläutert.

2.1 Reinforcement Learning (RL)

Reinforcement Learning (RL) ist ein Lernverfahren, bei dem ein Agent durch Interagieren mit seiner Umgebung lernt, optimal zu handeln. In diesem Unterkapitel wird dieses Konzept erläutert.

Beim *Reinforcement Learning* (RL) lernt ein autonomer Agent, eigenständige Entscheidungen durch Interaktionen mit seiner Umgebung zu treffen. Genau genommen geschieht dies durch Versuch und Irrtum sowie mithilfe einer Belohnungsfunktion. Die für das Lernen nötigen Eingabedaten sind gesammelte, voneinander abhängige Tupel, also eine aus Zustand, Aktion und Belohnung bestehende Datenfolge [1].

Reinforcement Learning kommt dann zum Einsatz, wenn ein Ziel für ein Problem bekannt ist, jedoch nicht der Weg zum Ziel, also die Lösung. Das Ziel eines Agenten ist es, möglichst viele Belohnungen in einer Umgebung zu erhalten. Bevor er eine Aktion ausführt, wird ihm nicht verraten, welche Belohnung er dafür erhält. Wird der Agent dann in der Umgebung trainiert, lernt er mit der Zeit, welche Auswirkungen spezifische Aktionen auf die Umgebung haben. Basierend darauf ist es ihm möglich, langfristige Strategien für maximale Belohnungen zu entwickeln [2].

Eine Trainingsumgebung muss die reale Welt unter Beachtung spezifischer Punkte vereinfacht widerspiegeln. Dazu muss einerseits ein passender RL-Algorithmus gewählt werden. Andererseits müssen mögliche Zustände und Aktionen sowie das Ziel, das durch Belohnungen beschrieben wird, klar definiert werden. Bis ein Algorithmus wie gewollt funktioniert, muss er viele Transitionen durchlaufen. Diese bestehen aus einer Zustandsänderung der Umgebung durch eine Aktion und der entsprechenden Belohnung. Ein Hauptgrund für den hohen Bedarf an Erfahrung ist, dass einzelne Transitionen strategisch zusammenhängen können und somit eine Belohnung für eine gute Aktion häufig erst verzögert ausgeschüttet wird [2].

Häufig wird der in Abbildung 2.1 gezeigte Markovsche Entscheidungsprozess benutzt, um die Funktionsweise von *Reinforcement Learning* zu erklären. Dieser besteht im Wesentlichen aus dem Agenten und der Umgebung. Dabei liefert die Umgebung Informationen über ihren aktuellen Spielzustand S_t an den Agenten. Alle möglichen Spielzustände mit den jeweiligen Informationen werden als Zustandsraum zusammengefasst. Anhand des übergebenen Spielzustands S_t tätigt

der Agent eine Aktion A_t . Alle ausführbaren Aktionen werden als Aktionsraum zusammengefasst [1]. Meist handelt es sich dabei um eine endliche Anzahl an Aktionen und somit um einen diskreten Aktionsraum [3]. Erhält der Agent durch die gewählte Aktion von der Belohnungsfunktion eine positive Belohnung R_t , steigt die Wahrscheinlichkeit, dass er in der Zukunft in ähnlichen Situationen zielführende Aktionen wählt [1].

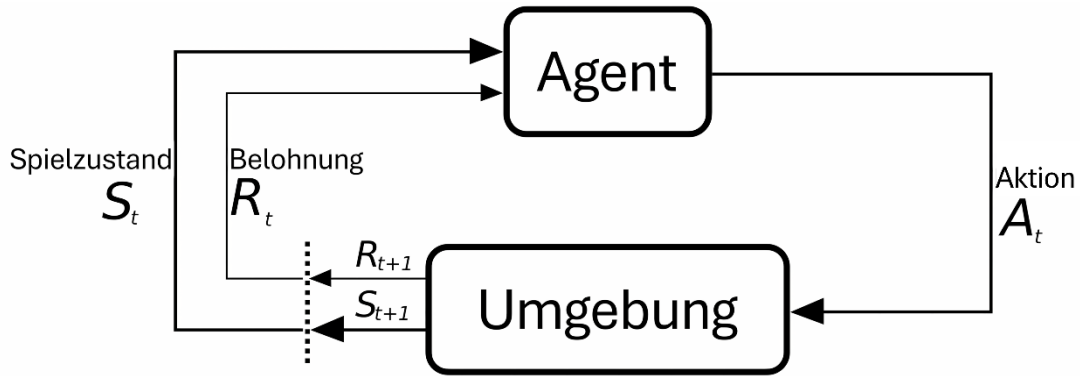


Abbildung 2.1: Schema eines Markovschen Entscheidungsprozesses im Reinforcement Learning.

Eine effektive Belohnungsfunktion zu definieren, kann eine Herausforderung sein. Es wird als sogenanntes «Reward-Engineering» bezeichnet [2]. Aus diesem Grund vergleicht eine der beiden Fragestellungen dieser Maturaarbeit die Effektivität verschiedener Belohnungsfunktionen miteinander. Eine Belohnungsfunktion ist wichtig, da sie das Ziel des Agenten formt. Sein Ziel ist es, die erwartete kumulierte Belohnung zu maximieren. Die kumulierte Belohnung setzt sich dabei aus der Summe aller Belohnungen einer Sequenz τ von Zuständen und Aktionen zusammen. Belohnungen, die noch viele Zeitschritte entfernt liegen, sind unwahrscheinlicher, eingesammelt werden zu können. Aus diesem Grund muss ein Diskontierungsfaktor, Gamma γ , definiert werden. Dieser reduziert automatisch den Einfluss von weiter entfernten Belohnungen. Der Exponent k von γ entspricht der Anzahl Zeitschritte, welche eine Belohnung vom aktuellen Schritt entfernt ist. Meist liegt der Parameterwert Gamma zwischen 0,8 und 0,999. Ein hoher Wert führt zu einer grossen Gewichtung der zukünftigen Belohnungen, während ein tiefer mehr auf die zeitnahen fokussiert. Der Faktor γ^k wird mit den Belohnungen aller Aktionen, die der Agent in einer möglichen Sequenz τ wählen würde, multipliziert. Addiert man alle zukünftigen, entsprechend gewichteten Belohnungen, erhält man die Formel (1) für die gewichtete Summe der zukünftigen Belohnungen [3].

$$R(\tau) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^k R_{t+k+1} = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1)$$

Jedem durch einen Zeitschritt erreichbaren Zustand wird der Wert seiner gewichteten Summe gegeben. Somit kann der Agent vergleichen und abschätzen, welche Aktionen zu Zuständen führen, aus denen in Zukunft eine höhere Belohnung

zu erwarten ist. Ist eine Belohnungsfunktion also schlecht definiert, ist es möglich, dass der Agent unerwünschte, nicht zielführende Strategien erlernt [3].

Ein bekanntes Dilemma von *Reinforcement Learning* ist der Konflikt zwischen Ausbeutung und Exploration. Dabei muss sich der Agent entscheiden, ob er lieber nach bewährtem, erfolgreichem Schema vorgeht oder ob er lieber eine neue Strategie erkundet und ausprobiert [1]. Das Problem dabei ist, dass sein übergeordnetes Ziel immer ist, die maximale kumulierte Belohnung zu erhalten. Entscheidet er sich für das bewährte Schema, ist die Wahrscheinlichkeit hoch, dass er dafür belohnt wird. Entscheidet er sich jedoch für eine neue Vorgehensweise, könnte er einerseits scheitern und eine geringere Belohnung erhalten. Andererseits ist es auch möglich, dass er auf eine neue, zielführendere Strategie stösst, die eine höhere Belohnung liefert, als die bisher bewährte Strategie [4]. Der Agent muss also einen Kompromiss zwischen dem Ausprobieren von Neuem und dem Setzen auf bewährte Aktionen finden [1].

Die sogenannte *Policy* π lässt sich mit dem Gehirn eines Agenten vergleichen. Ziel des Trainings ist es, die optimale *Policy* zu finden. Das ist die Strategie, mit der die erwartete, kumulierte Belohnung durch die Wahl von passenden Aktionen bei gegebenen Zuständen maximiert wird. Es gibt verschiedene Ansätze, diese optimale *Policy* zu finden, für diese Maturaarbeit sind jedoch nur wertbasierte Methoden relevant. Bei diesen ergibt sich die optimale *Policy* aus einer Wertfunktion, welche den Zuständen möglichst akkurat einen Wert zuordnet [5]. Im nächsten Unterkapitel wird der dafür verwendete Ansatz, das *Deep Q-Network* (DQN), näher erläutert.

2.2 Deep Q-Learning

Q-Learning ist ein Algorithmus, der auf dem in Unterkapitel 2.1 erläuterten *Reinforcement Learning* basiert. In diesem Unterkapitel werden das Konzept von *Q-Learning*, dessen Erweiterung durch neuronale Netzwerke sowie die Funktionsweise mithilfe eines *Multilayer-Perceptron* (MLP) erläutert. MLP ist dabei die im Produkt dieser Maturaarbeit verwendete neuronale Netzwerkarchitektur.

2.2.1 Klassisches Q-Learning

Beim *Q-Learning* handelt es sich um einen Algorithmus, der zum Trainieren einer *Q*-Funktion dient, welche allen Zustand-Aktion-Paaren einen *Q*-Wert zuordnet. Alle *Q*-Werte werden dann in einer *Q*-Tabelle gespeichert. Ein Beispiel einer solchen sowie deren Funktionsweise ist in Abbildung 2.2 dargestellt. Diese stellt beim klassischen *Q-Learning* die *Q*-Funktion dar [6]. Bei dieser Lernmethode wird der Ansatz des zeitlichen Differenzlernens (TD) verfolgt. Dabei wird die Schätzung des Wertes zukünftiger Zustände basierend auf der Differenz zwischen erwarteter Belohnung und tatsächlich erhaltener Belohnung angepasst [1]. Zu Beginn sind alle Einträge in

der Q-Tabelle null. Durch Training wird die Tabelle dann ausgefüllt. Um das Dilemma zwischen Ausbeutung und Exploration zu lösen, wird der Wert Epsilon ϵ definiert. Dieser entspricht der Wahrscheinlichkeit, mit welcher der Agent eine zufällige, nicht auf früheren Erfahrungen basierende Aktion wählt. Zu Beginn des Trainings ist dieser Wert hoch und wird dann im Verlaufe des Trainings reduziert [6].

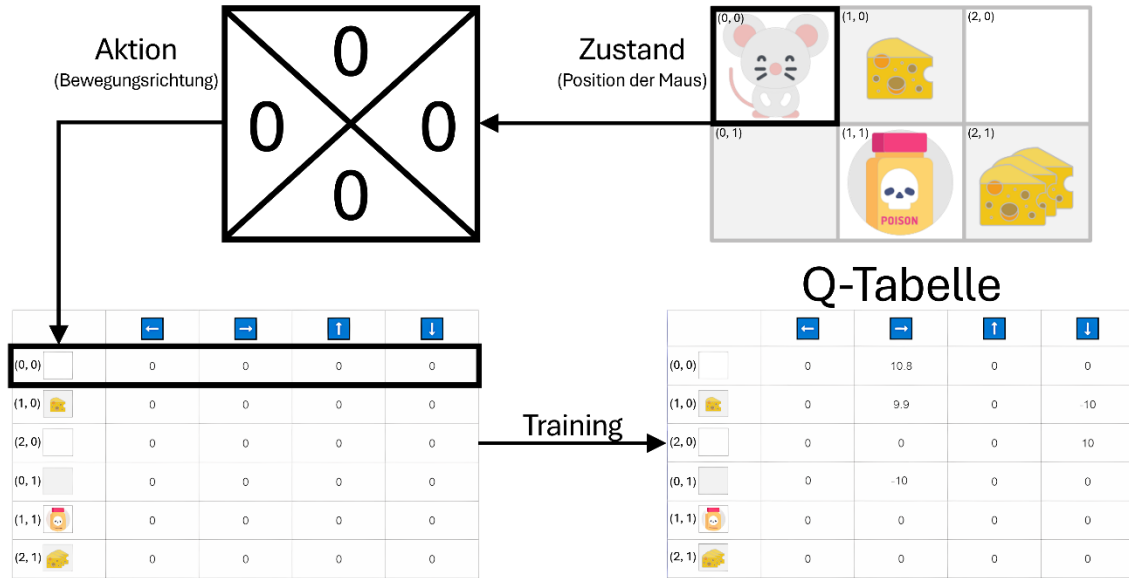


Abbildung 2.2: Aufbau und Funktionsweise einer Q-Tabelle eines Beispielspiels. Aus jedem Zustand können dieselben Aktionen ausgeführt werden. Links wird die Q-Tabelle initialisiert und rechts dann durch Training mit Werten gefüllt.

Um die optimale Q-Funktion zu finden, ist die Bellman-Optimalitätsgleichung von zentraler Bedeutung. Diese ordnet Zustand-Aktion-Paaren den jeweils optimalen Q-Wert zu [7]:

$$Q^*(s, a) = \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \quad (2)$$

Formel (2) definiert also den angestrebten, optimalen Q-Wert für das Ausführen einer Aktion a aus Zustand s als den Erwartungswert $\mathbb{E}$ der Summe aus der sofortigen Belohnung R_{t+1} und dem mit γ gewichteten, maximalen Q-Wert des Folgezustands. Diese Summe bildet somit das TD-Ziel. Aus dieser Gleichung setzt sich dann die Q-Wert-Aktualisierungs-Formel (3) zusammen. Mit dieser wird nach jedem Zug der Q-Wert des gewählten Zustand-Aktion-Paares aktualisiert [7]:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(s, a) \right] \quad (3)$$

Der neue Q-Wert ergibt sich somit aus der Summe des alten Q-Wertes und einem Anpassungswert. Dieser ist das Produkt aus der Lernrate α und dem sogenannten TD-Fehler. Der TD-Fehler stellt die Differenz zwischen dem TD-Ziel und dem alten Q-Wert dar [7]. Die Lernrate α bestimmt dabei, wie stark der Q-Wert in Richtung des

TD-Ziels angepasst wird. Je näher der TD-Fehler an null ist, desto genauer ist also die Vorhersage der Q-Funktion [6].

Ein Problem des herkömmlichen *Q-Learning* ist, dass es sich nicht gut für Umgebungen mit vielen verschiedenen Zuständen eignet. Dies liegt daran, dass die Q-Tabelle für jedes Zustand-Aktion-Paar einen Wert speichern muss und somit bei komplexen Umgebungen sehr gross wird. Bei einem Spiel wie Käsekästchen ist die Anzahl möglicher Spielzustände so hoch, dass eine Q-Tabelle rechnerisch kaum umsetzbar wäre. Der Agent müsste enorm viel Erfahrung durch zeitintensives Training sammeln, um die Tabelle sinnvoll zu füllen. Hat der Agent nicht genug Training, haben alle noch nie erlebten Zustand-Aktion-Paare den Q-Wert null, wodurch der Agent dann nicht wissen kann, ob es sich um belohnende oder bestrafende Aktionen handelt. Um dieses Problem zu lösen, wird eine Alternative benötigt [8].

2.2.2 Erweiterung durch neuronale Netzwerke

Deep Q-Learning löst das Skalierungsproblem durch eine Implementierung von neuronalen Netzwerken. *Deep Q-Learning* hat den Vorteil, dass auch bei Umgebungen, die viele mögliche Zustände und Aktionen haben, mit einer moderaten Menge an Training gute Ergebnisse erzielt werden können. Dieser Ansatz unterscheidet sich gegenüber dem klassischen *Q-Learning* durch die Nutzung eines tiefen neuronalen Netzwerks, um Q-Werte zu schätzen. Ein weitverbreiteter Algorithmus, der dieses Konzept umsetzt, heisst *Deep Q-Network* (DQN) [7]. Dabei wird die Q-Funktions-Formel durch Theta θ_i erweitert zu $Q(s, a; \theta_i)$. Theta θ_i stellt die Parameter des Q-Netzwerks zum i -ten Optimierungsschritt, also vor ihrer i -ten Anpassung, dar. Diese Q-Netzwerkparameter werden während des Trainings regelmässig in Optimierungsschritten angepasst, um die Verlustfunktion (4) zu minimieren. Dadurch wird das Q-Netzwerk besser darin, Q-Werte zu schätzen und die Abweichungen zu den TD-Zielen nehmen ab. Die Verlustfunktion (4) wird durch Quadrierung des TD-Fehlers gebildet [9]:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right] \quad (4)$$

Um das Training zu stabilisieren, verwendet der DQN-Algorithmus zwei neuronale Netzwerke: das Q-Netzwerk und das Ziel-Netzwerk. Das Q-Netzwerk mit den Parametern θ_i schätzt bei jedem Zug die Q-Werte der Zustand-Aktion-Paare. Die Parameter des Q-Netzwerks werden dann nach einer vordefinierten Anzahl an Zeitschritten angepasst. Das zweite neuronale Netzwerk, das Zielnetzwerk, besitzt die Parameter θ_i^- . Mithilfe dieses Netzwerks und der Belohnung r wird der TD-Zielwert in der Verlustfunktion (4) berechnet. Die Parameter des Zielnetzwerks werden seltener als jene des Q-Netzwerks angepasst. Bei einer Optimierung übernimmt es die Parameter des Q-Netzwerks. Um die Verlustfunktion (4) für eine

Aktualisierung der Q-Netzwerkparameter zu berechnen, werden aus dem Wiedergabepuffer D mehrere vergangene Erfahrungen komplett zufällig ausgewählt. Dabei werden Zustand s , Aktion a , Belohnung r und Folgezustand s' der ausgewählten Erfahrungen in Form von Stichproben-Minibatches verwendet. Somit werden die Korrelationen zwischen aufeinanderfolgenden Erfahrungen reduziert [9].

2.2.3 Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) ist die neuronale Netzwerkarchitektur, auf der die in dieser Maturaarbeit verglichenen KI-Modelle basieren. Wie der Name schon sagt, basiert MLP auf mehreren Schichten von Perzeptronen, welche miteinander verbunden sind. Ein Perzeptron funktioniert ähnlich wie ein menschliches Neuron. Aus diesem Grund wird es manchmal auch als Neuron bezeichnet. Ein einzelnes Neuron hat Verbindungen zu allen Neuronen der vorherigen Schicht, sofern es nicht selbst die erste Schicht ist [10]. Jede einzelne Verbindung zwischen zwei Neuronen hat eine Gewichtung w_{nm} , also vom m -ten Neuron der vorherigen Schicht zum n -ten Neuron der aktuellen Schicht. Diese Gewichtung definiert die Relevanz der jeweiligen weitergegebenen Information. Wird ein Wert x_m von einem vorherigen Neuron gesendet, wird er mit dieser spezifischen Gewichtung w_{nm} multipliziert. Addiert man alle gewichteten erhaltenen Werte eines Neurons, erhält man die Summe eines Neurons $\sum_m (x_m * w_{nm})$. Addiert man zu dieser Summe einen Bias b_n , erhält man z_n . Dieser Wert wird dann durch eine Aktivierungsfunktion transformiert, welche den Wert a_n liefert. Dieser Ausgabewert wird dann an die Neuronen der nächsten Schicht gesendet [11]. Eine dafür genutzte Aktivierungsfunktion ist beispielsweise *Rectified Linear Unit* (ReLU()). Sie liefert alle Werte über null genau so weiter, bei allen anderen Eingabewerten ist der Ausgabewert null [10]. Abbildung 2.3 illustriert die Weitergabe und Verarbeitung von Werten in einem Neuron.

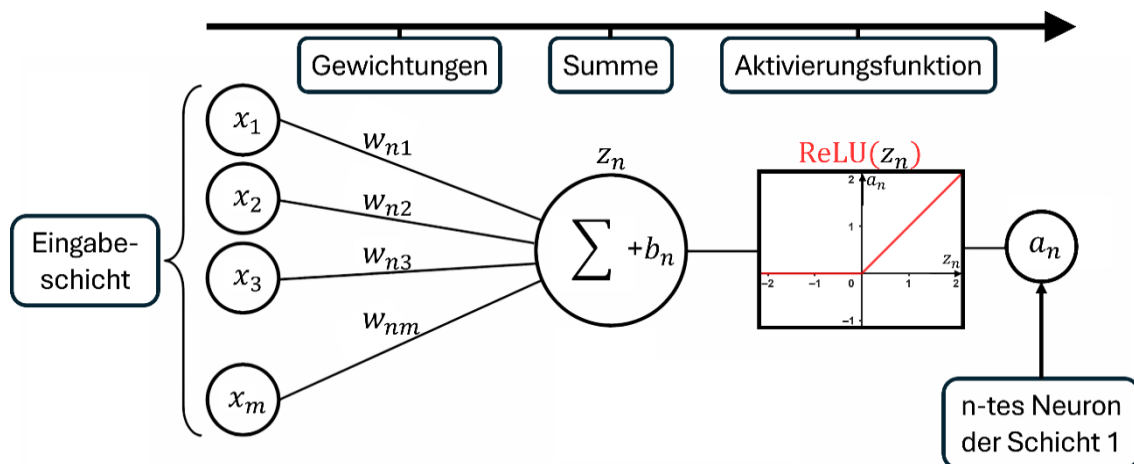


Abbildung 2.3: Schema eines Perzeptrons. Es empfängt Eingabewerte und gewichtet und summiert diese. Die resultierende Summe wird durch die ReLU-Aktivierungsfunktion transformiert. Das Ergebnis bildet den Ausgabewert des n -ten Neurons der ersten verborgenen Schicht.

Ein neuronales Netzwerk besteht aus 3 Arten von Schichten. Dabei wird unterschieden zwischen der Eingabeschicht, verborgenen Schichten und der Ausgabeschicht [10]. Jede Schicht speist die nächste mit den Ergebnissen ihrer Berechnung. Nutzt man ein MLP-Netzwerk für *Deep-Q Learning*, bildet jedes Neuron der Eingabeschicht, wovon es nur eine gibt, eine Information des Umgebungszustandes ab. Die Anzahl der verborgenen, dazwischenliegenden Schichten sowie der jeweils darin enthaltenen Neuronen ist frei wählbar. Bei der Ausgabeschicht, wovon es nur eine gibt, stellt jeder Wert den geschätzten Q-Wert für eine Aktion aus dem gegebenen Zustand heraus dar. Die Menge der Neuronen in der Ausgabeschicht entspricht also der Grösse des Aktionsraums [12]. Abbildung 2.4 veranschaulicht den Aufbau eines neuronalen Netzwerks.

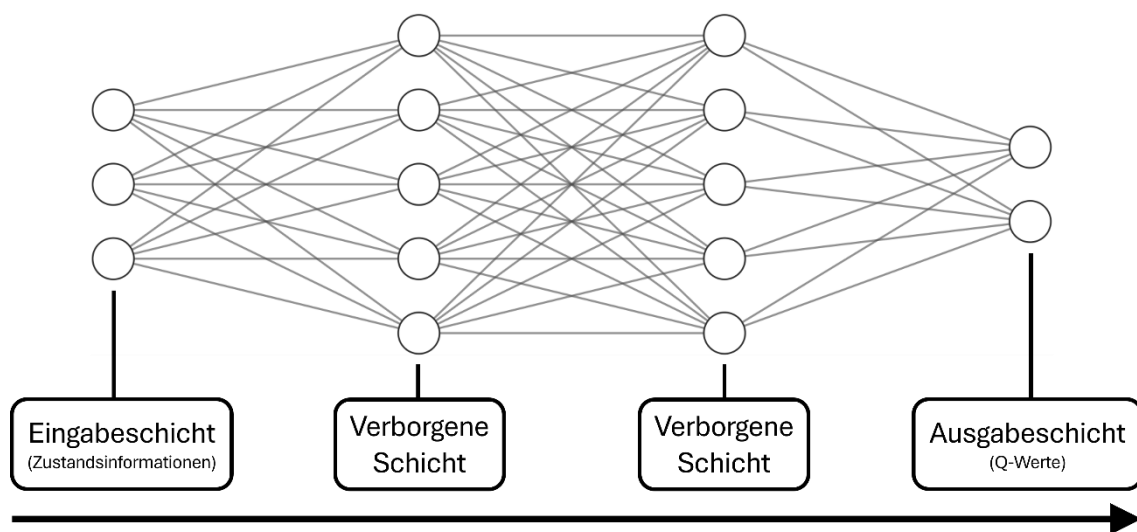


Abbildung 2.4: Schema eines Multilayer Perceptron (MLP). Das Netzwerk besteht aus einer Eingabeschicht, zwei verborgenen Schichten und einer Ausgabeschicht. Die Neuronen einer Schicht sind jeweils vollständig mit denen der folgenden Schicht verbunden. Bei einem Q-Netzwerk liefert die Ausgabeschicht die Q-Werte.

Würde der Algorithmus ohne Anpassungen stetig nur die Q-Werte berechnen, wäre das MLP-Netzwerk nicht in der Lage, zu lernen. Aus diesem Grund benutzt es Rückwärtspropagierung. Dabei werden die Gewichte und Biases, zusammen die Parameter θ , schrittweise während des Trainings angepasst. Dazu wird meist das Gradientenabstiegsverfahren verwendet. Dabei wird der erhaltene Verlustfunktionswert eines zufällig gewählten Zustand-Aktion-Paares zurückpropagiert und die Parameter, die für die schlechte Schätzung eines Q-Werts verantwortlich waren, werden angepasst. Durch das wiederholte Ausführen dieser Schritte minimiert das Netzwerk die Verlustfunktion (4). So lernt das *Deep Q-Network*, die Q-Werte immer präziser vorherzusagen und entwickelt eine zunehmend effektive Spielstrategie [13].

2.3 Python-Bibliotheken

Um für diese Maturaarbeit eine künstliche Intelligenz auf ein Spiel zu trainieren, sind verschiedene Python-Bibliotheken nötig. In diesem Unterkapitel wird erklärt, wozu diese dienen und wie sie es ermöglichen, in einer Trainingsumgebung einen Agenten zu trainieren.

2.3.1 *Gymnasium*¹ und *Stable-Baselines3*²

Gymnasium ist eine Bibliothek, die die Grundlagen für *Reinforcement Learning* 2.1 bereitstellt. Mit ihr ist es möglich, eine Umgebung zu programmieren, in der ein Agent trainiert werden kann. Zu den wichtigsten Funktionen gehören unter anderem `Env.reset()`, `Env.step()` und `Env.render()`. Der Ablauf des Lernens folgt dabei dem Prinzip des in Unterkapitel 2.1 erklärten Markovschen Entscheidungsprozesses [14].

Tabelle 2.1: Übersicht über die zentralen Methoden der *Gymnasium*-Umgebung.

Methode	Beschreibung
<code>Env.reset()</code>	Wird nach dem Starten der Umgebung ausgeführt, um die Umgebung zurückzusetzen und den Zustand der Umgebung zu erfassen. Diese Methode wird nach jeder Episode, also einem beendeten Spiel bestehend aus Transitionen, aufgerufen [14].
<code>Env.step()</code>	Wird ausgeführt, nachdem der Agent eine Aktion gewählt hat. Dabei wird diese Aktion in der Umgebung ausgeführt, welche dann aktualisiert wird. Danach erhält der Agent zum einen eine Belohnung, dementsprechend wie sich seine Aktion auf die Umgebung ausgewirkt hat, und zum anderen Informationen über den neuen Zustand der Umgebung. Es wird auch überprüft, ob die Episode zu Ende ist und diese Information wird gegebenenfalls weitergegeben. Jeder einzelne Aufruf dieser Funktion wird als ein Zeitschritt bezeichnet. Mehrere Zeitschritte ergeben dann folglich zusammen eine Episode [14].
<code>Env.render()</code>	Ist für das Darstellen eines Bildes verantwortlich. Dies jedoch nur, falls der entsprechende Modus beim Erstellen der Umgebung definiert wurde. Wurde kein Modus definiert, wird kein Bild angezeigt und diese Funktion nicht aufgerufen [14].

Stable-Baselines3 (SB3) ist eine Bibliothek, welche Implementierungen von *Reinforcement-Learning*-Algorithmen wie DQN zur Verfügung stellt. SB3 greift dabei intern auf die Python-Bibliothek *PyTorch* für die Implementierung der neuronalen Netzwerke zurück. Die SB3-Bibliothek vereinfacht das Training von KI-Modellen erheblich. Dabei können verschiedene Parameter wie z. B. die Lernrate `learning_rate` oder der Diskontierungsfaktor `gamma` direkt beim Erstellen des Modells über den Konstruktor definiert werden. Damit ermöglichen sie viel

¹ <https://gymnasium.farama.org/>

² <https://stable-baselines3.readthedocs.io/>

Flexibilität beim Trainingsprozess. Ein weiteres nützliches Argument, das dem Konstruktor übergeben werden kann, ist `tensorboard_log=«(Dateipfad)»`. Damit werden verschiedene Metriken des Trainings regelmässig gespeichert und können grafisch dargestellt werden. Um diese zu visualisieren, ist eine weitere Bibliothek namens *Tensorboard* nötig. Für die Visualisierung der Daten muss dann die Eingabeaufforderung geöffnet werden und der Befehl `tensorboard --logdir (Dateipfad)` gegeben werden. Dadurch wird ein lokaler Server gestartet und unter der URL `http://localhost:6006/` sind die visualisierten Daten ersichtlich. Die Bibliothek *TensorBoard* ist jedoch keine Anforderung für das Ausführen des Codes. Sie dient ausschliesslich der Visualisierung der Daten. Die wichtigste Methode der SB3-Bibliothek ist `Learn()`. Bei der Verwendung des *Deep Q-Networks* wird diese Funktion genutzt, um beispielsweise die Anzahl der Zeitschritte des Trainings festzulegen, oder eine Umgebung für *Callbacks* zu definieren. *Callbacks* sind Funktionen, die während des Trainings regelmässig ausgeführt werden, um Aktionen durchzuführen. Sie dienen beispielsweise dazu, die Gewinnrate eines Modells während des Trainings in einer separaten Evaluierungsumgebung aufzuzeichnen, um so eine objektivierte Leistungsevaluierung vornehmen zu können. Zudem trifft das Modell durch einen *Callback* seine Entscheidung vollständig basierend auf Ausbeutung, also Epsilon-Wert 0. Im Unterkapitel 4.1 wird weiter auf die Gründe für eine Unterscheidung zwischen Trainings- und Evaluierungsumgebung eingegangen. Die Klassenmethode `Load()` ermöglicht eine weitere Funktionalität. Damit werden fertig trainierte Modelle unter Angabe des Speicherorts geladen [15].

2.3.2 Sonstige Bibliotheken

*NumPy*³ ist eine Bibliothek, die numerische Berechnungen mit Python ermöglicht. Sie bietet eine Vielzahl an Möglichkeiten. Darunter sind auch sogenannte *ndarrays*. Das ist eine Datenstruktur, die mehrere Werte des gleichen Typs speichern kann. Ein *Array* kann beliebig viele Dimensionen haben. Die Grösse jeder Dimension wird definiert. Ist das *Array* dreidimensional, lässt es sich zur Veranschaulichung als Quader darstellen, der in kleinere Quader unterteilt ist, welche Werte enthalten. Der Zugriff auf die Daten erfolgt wie bei Listen mit Indizes `[x, y, z]` [16]. Eine weitere Funktionalität von NumPy, die genutzt wurde, ist `numpy.random.Generator.integers`. Damit ist es möglich, zufällige ganze Zahlen in einem definierten Bereich generieren zu lassen [17].

*Pygame*⁴ ist eine Bibliothek, die zur Programmierung von Multimedia-Anwendungen wie Spielen dient. Sie ermöglicht die Darstellung von 2D-Grafiken, das Abspielen von Klängen sowie die Erfassung von Maus- und Tastatureingaben [18].

³ <https://numpy.org/>

⁴ <https://www.pygame.org/>

3 Produktbeschreibung

In dem folgenden Kapitel werden die Funktionsweise des Spiels «Käsekästchen» sowie die Umsetzung des Spiels als Trainingsumgebung für einen Agenten beschrieben.

3.1 Das Spiel Käsekästchen

In diesem Unterkapitel wird das Spielprinzip und die Regeln des Spiels «Käsekästchen» erläutert.

Bei dem Spiel «Käsekästchen» handelt es sich um ein zugbasiertes Zwei-Personen-Spiel. Das Spiel beginnt mit einem leeren Punkteraster. Dabei ist die Anzahl der vertikalen und horizontalen Punkte vorher beliebig wählbar. Abwechselnd wird dann ein horizontaler oder vertikaler Strich zwischen zwei Punkten gezogen. Ein einzelnes Kästchen ist definiert durch die vier umgebenden Striche. Die Punkte bilden die Ecken. Wenn bereits auf drei Seiten des Kästchens Striche zwischen den Punkten gezogen sind, kann ein Spieler den Strich auf der letzten Seite ziehen, um dieses Kästchen für sich zu gewinnen. Hat ein Spieler ein Kästchen gewonnen, darf er direkt anschliessend noch einen zweiten Strich ziehen. Er ist so lange an der Reihe, bis durch seinen Strich kein Kästchen mehr geschlossen wird. Dies führt zum strategischen Konzept von sogenannten Ketten. Dies sind mehrere ungeschlossene Kästchen, welche offen miteinander verbunden sind. Dabei besitzen diese ungeschlossenen Kästchen jeweils erst zwei Striche. Ein einziger unvorsichtig gezogener Strich reicht dann aber aus, um dem anderen Gegner die Möglichkeit zu geben, alle Kästchen dieser Kette nacheinander zu schliessen und an sich zu reißen. Denn sobald der Gegner das angefangene Kästchen mit drei Strichen der Kette vervollständigt, entstehen neue Möglichkeiten, weitere angrenzende Kästchen dieser Kette, die nun drei Striche haben, zu schliessen. Wird der Gegner also dazu gebracht, eine solche Kette zu öffnen, sprich den auslösenden Strich zu ziehen, können somit alle Kästchen der Kette nacheinander gewonnen werden. Ziel des Spiels ist es dann, möglichst viele Kästchen für sich zu gewinnen. Das Spiel ist zu Ende, wenn alle möglichen horizontalen und vertikalen Striche gezogen wurden und somit jedes Kästchen vergeben ist. Ein Spielverlauf könnte z. B. wie in Abbildung 3.1 aussehen.

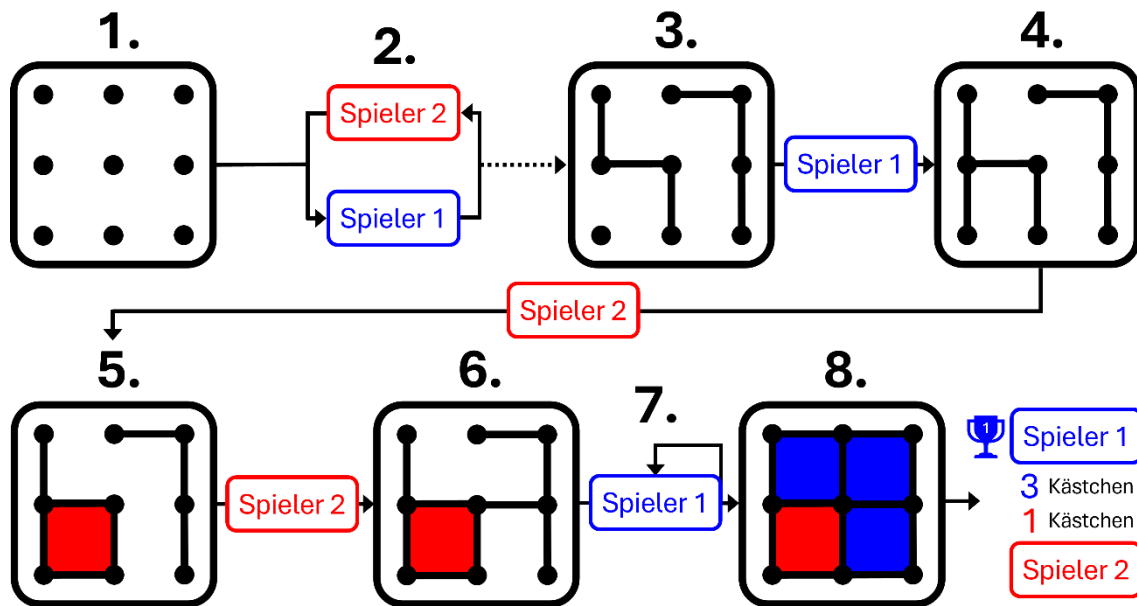


Abbildung 3.1: Ablaufschema eines Beispiel-Käsekästchen-Spiels mit strategischem Opfer. Die Abfolge zeigt das strategische Element der Ketten. Spieler 1 opfert strategisch ein einzelnes Kästchen an Spieler 2. Dadurch wird Spieler 2 gezwungen, die verbleibende lange Kette von potenziellen Kästchen zu öffnen. Anschliessend kann Spieler 1 diese Kette vervollständigen und gewinnt dadurch das Spiel.

3.2 Umsetzung des Spiels als *Gymnasium*-Umgebung

Das Spiel wurde mithilfe des in 2.3.1 erklärten Prinzip als eine *Gymnasium*-Umgebung umgesetzt. In diesem Unterkapitel folgt eine Erläuterung der wichtigsten Codeteile, die spezifisch für die Umsetzung des Käsekästchen-Spiels sind.

3.2.1 Der Beobachtungsraum

Der Beobachtungsraum `self.observation_space`: Der Beobachtungsraum, also jene Informationen, die an den Agenten weitergegeben werden, wird als dreidimensionales *NumPy*-Array definiert. Dieses ist nach *NumPy*-Achsenkonvention folgendermassen aufgebaut: `[x, y, z]`. Der x-Index repräsentiert dabei die vertikale Achse der Position der Punkte. Die Grösse dieser Achse entspricht also der Anzahl Punkte in vertikaler Richtung. Der y-Index repräsentiert die horizontale Achse der Punkte und die Grösse davon ist somit durch die horizontale Anzahl an Punkten definiert. Der z-Index entspricht der Ebene. Dabei entspricht die z-Ebene `[x, y, 0]` einem horizontalen Strich rechts des durch x- und y-Index definierten Punktes. Währenddessen entspricht die z-Ebene `[x, y, 1]` einem vertikalen Strich unterhalb des jeweiligen Punktes. Es gibt somit nur zwei z-Indexe, nämlich null und eins. Zu Beginn eines Spiels wird die Umgebung zurückgesetzt und somit der Wert aller Werte im Array auf 0 gesetzt. In Abbildung 3.2 ist ein durch 3D-Quader visualisiertes Array dargestellt.

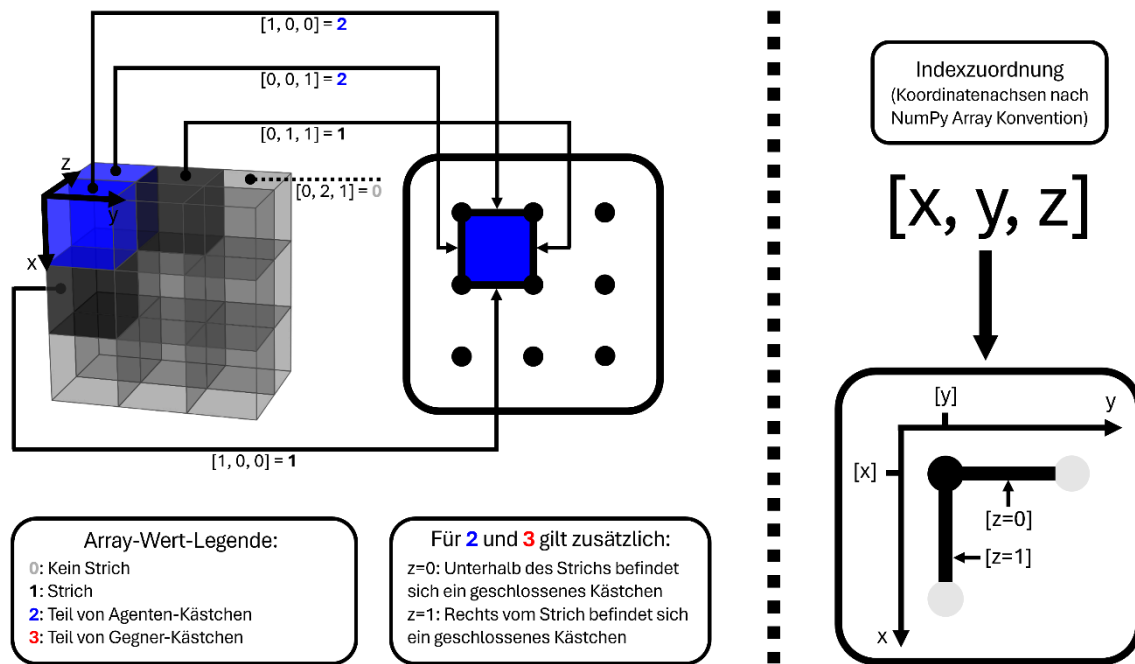


Abbildung 3.2: Darstellung des Beobachtungsraums und seiner Indexzuordnung. Links ist der Beobachtungsraum als dreidimensionales Array visualisiert. Die Striche, die zum geschlossenen Kästchen auf dem mittleren Spielfeld gehören, sind entsprechend ihrer Array-Werte farblich hervorgehoben (siehe Werte-Legende unten). Rechts wird die Zuordnung der einzelnen Indizes zur Position eines Strichs im Spielfeld veranschaulicht.

Es gibt in einem Array, welches dieses Prinzip der Indizierung nutzt, mehr Werte, als es mögliche Striche auf dem dazugehörigen Käsekästchenspielfeld gibt. Aus diesem Grund enthält das Array auch irrelevante Werte. Diese irrelevanten Indizes werden in der Abbildung 3.3 bildlich in einem 3D-Array veranschaulicht.

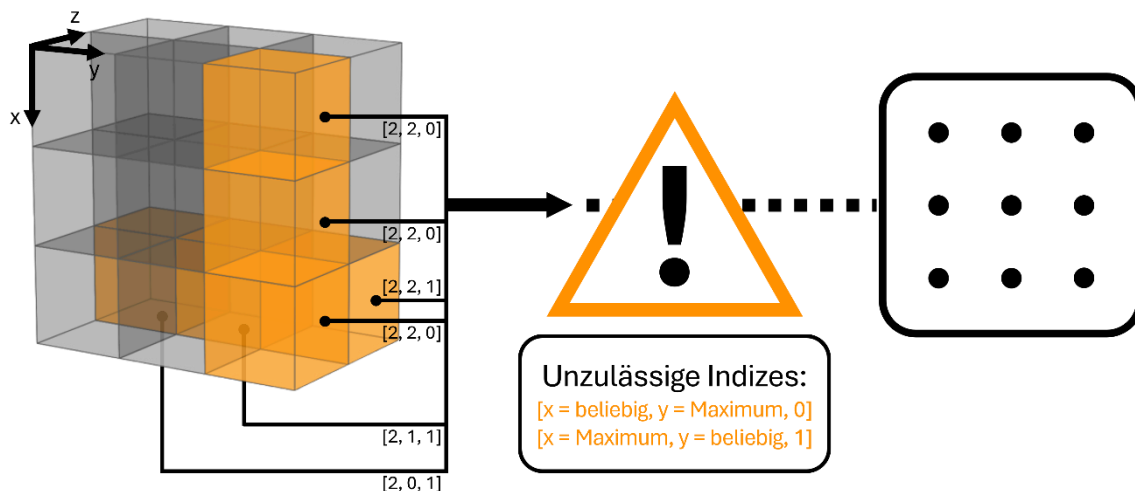


Abbildung 3.3: Darstellung der ungültigen Rand-Indizes im Beobachtungsraum. Jene Positionen des Arrays, welche keinem Strich des Spielfelds zugehörig sind, sind im Array links orange hervorgehoben. Diese Array-Positionen werden während des Spiels ignoriert und behalten immer den Wert 0.

3.2.2 Der Aktionsraum

Der Aktionsraum `self.action_space`: Der Aktionsraum definiert, welche Aktionen es gibt. Das genutzte MLP-Netzwerk ist nicht in der Lage, direkt eine Position im Array zu wählen, um einen Strich zu ziehen. Aus diesem Grund wurde

ein diskreter Aktionsraum, welcher durch eine Zahlenmenge definiert ist, verwendet. Diese Zahlenmenge entspricht der maximalen Anzahl an theoretisch möglichen Strichen eines leeren Spielfelds. Eine separate Methode rechnet dann die gewählte Aktion in die entsprechenden *Array*-Indizes um.

3.2.3 Informationsauswertungsmethode

Die Umgebungsinformationsmethode `_get_info(self)` wird nach jeder Aktion aufgerufen und dient dazu, die hervorgerufenen Veränderungen auszuwerten und den Spielstand zu aktualisieren. Als Erstes wird überprüft, ob durch die Aktion neue Kästchen geschlossen wurden. Wurde kein Kästchen geschlossen, wechselt der Spieler. Hat der Agent ein Kästchen geschlossen, wird er belohnt. Hat aber der Gegner ein Kästchen geschlossen, wird der Agent bestraft. Dies geschieht, indem zu der Belohnung `self.belohnung` eine Belohnung addiert oder subtrahiert wird. Die Belohnungsausschüttung variiert jedoch zwischen den verschiedenen Belohnungsfunktionen, welche in dieser Maturaarbeit verglichen werden. Als Nächstes werden in dieser Methode die Striche, welche noch benötigt werden, um ein Kästchen zu schliessen, in einer Liste gesammelt. Der Trainingsgegner des Agenten kann diese Striche später potenziell ziehen, um selbst Kästchen für sich zu gewinnen.

3.2.4 Schrittmethode

Die Schritt-Methode `step(self)` bildet den Kern der Interaktion zwischen dem Agenten und der Umgebung. Sie wird jeden Zeitschritt einmal aufgerufen. Der Agent ist immer als Erstes an der Reihe, da ihm sonst nicht die aktuellsten Informationen über die Umgebung zur Verfügung stehen. Schliesst der Agent ein Kästchen, wird die Methode für die Aktionswahl des Gegners nicht aufgerufen und der Zeitschritt endet. Im nächsten Zeitschritt macht der Agent dann wieder als Erstes einen Zug. Dies geschieht so lange, bis der Agent durch seine Aktion kein Kästchen mehr schliesst. Dann kommt der Gegner an die Reihe. Kann dieser dann ein Kästchen schliessen, ist er innerhalb dieses einen Zeitschritts so lange am Zug und tätigt Aktionen, bis er kein Kästchen mehr schliesst.

3.3 Implementierung der Künstlichen Intelligenz

Wie in Abschnitt 2.3.1 beschrieben, wird die Bibliothek Stable-Baselines3 verwendet, um eine KI, also ein neuronales Netzwerk, in der Umgebung trainieren zu lassen. Im folgenden Unterkapitel wird die Implementierung davon erläutert.

Zuerst muss ein auf DQN basierendes Modell erstellt werden mit `model.DQN(...)`. Dafür wird die in Abschnitt 2.2.3 beschriebene MLP-Netzwerkarchitektur verwendet. Standardmässig enthält das Netzwerk der SB3-Implementierung zwei verborgene Schichten mit je 64 Neuronen [19]. In der Eingabeschicht stellt jeder

Wert einer Position im *Array* eine Information dar. Bei der Ausgabeschicht stellt jeder Wert eines Ausgabeneurons den geschätzten Q-Wert für das Ziehen eines bestimmten Strichs bei gegebenem Zustand dar. Die Anzahl der Ausgabeneuronen ist gleich der Anzahl möglicher Striche auf einem leeren Spielfeld. Beim Erstellen eines Modells können verschiedene Hyperparameter angepasst werden. Die in dieser Maturaarbeit relevanten Hyperparameter sind in Tabelle 3.1 beschrieben.

Tabelle 3.1: Übersicht der untersuchten Hyperparameter und deren Beschreibung.

Hyperparameter	Beschreibung
learning_rate	Die Lernrate bestimmt, wie sehr die Parameter des Netzwerks pro Optimierungsschritt verändert werden. Ein hoher Wert, kann zwar das Training beschleunigen, macht es aber auch instabiler. Ein tiefer Wert führt zu mehr Stabilität des Trainings durch kleinere Veränderungen, benötigt jedoch erheblich mehr Zeit.
gamma	Der Diskontierungsfaktor gewichtet die zukünftigen Belohnungen. Ein hoher Wert führt dazu, dass der Agent langfristige Strategien verfolgt, während ein tiefer Wert den Fokus mehr auf kurzfristige Belohnungen legt.
exploration_fraction	Der Explorationsphasenanteil legt fest, über welchen prozentualen Anteil der gesamten Trainingszeitschritte der Epsilon-Wert von 1 auf seinen finalen Wert von 0,05 abgesenkt wird. Er bestimmt also, wie viele Aktionen anteilmässig zufällig gewählt werden, abhängig davon wie fortgeschritten das Training ist. Durch einen hohen Wert exploriert der Agent länger und erkundet mehr. Währenddessen legt ein tiefer Wert den Fokus früher auf Ausbeutung.

Weitere wichtige Hyperparameter, die in dieser Maturaarbeit nicht variiert wurden, sind die Anzahl der Trainingszeitschritte und die Wiedergabepuffergrösse, welche in Tabelle 3.2 beschrieben werden.

Tabelle 3.2: Übersicht weiterer Hyperparameter, die nicht untersucht wurden, und deren Beschreibung.

Hyperparameter	Beschreibung
total_timesteps	Die Anzahl der Zeitschritte (Standardwert: 2500000) bestimmt die Länge des Trainings. Dabei wird nach jedem vierten Schritt mit einem Datensatz aus dem Wiedergabepuffer ein Optimierungsschritt des Q-Netzwerks durchgeführt [15].
buffer_size	Die Wiedergabepuffergrösse (Standardwert: 100000) bestimmt die maximale Anzahl an vergangenen Erfahrungen, welche im Wiedergabepuffer gespeichert werden. Die Grösse des Wiedergabepuffers ist begrenzt, sodass jüngere Erfahrungen bevorzugt werden und ältere, sobald der Speicher voll ist, durch neue ersetzt werden.

4 Methodik

Im folgenden Kapitel werden das Trainingskonzept, der Versuchsaufbau und die Evaluierungsmethoden erläutert, welche zur Erfassung und Auswertung der Daten genutzt wurden.

4.1 Trainingskonzept

In diesem Unterkapitel wird das Trainingskonzept erläutert, welches für die Erfassung der Daten genutzt wurde.

Die Spielfeldgrösse wurde auf vier mal vier Punkte festgelegt, da dies einen guten Kompromiss aus Strategiekomplexität und Menge an benötigtem Training einer KI bildet.

Um die Zufallskomponente des Gegners reproduzierbar zu machen, wurde der seed-Wert auf 0 gesetzt. Dieser Wert stellt sicher, dass bei Verwendung der gleichen Hyperparameterwerte sich die Trainingsergebnisse reproduzieren lassen.

Für das Training wurde ein regelbasierter Trainingsgegner verwendet. Er wählt seine Aktion basierend auf zwei Prioritäten. Zuerst probiert er, Kästchen zu schliessen. Falls dies nicht möglich ist, tätigt er eine zufällige gültige Aktion.

Um die Gewinnrate und kumulierte Belohnung der KI-Modelle möglichst gegen menschenähnliches Spielverhalten zu ermitteln, wurde dazu eine separate Evaluierungsumgebung mit einem komplexen Evaluierungsgegner verwendet. Dieser Evaluierungsgegner ist so konstruiert, dass er menschliches Spielverhalten möglichst imitiert, indem er seine Aktionen basierend auf einer komplexeren Prioritätenhierarchie wählt. Er wählt seine Aktion basierend auf vier Prioritäten: 1. Kästchen schliessen, 2. keinen dritten Strich eines Kästchens ziehen, um dem Agenten keine Vorlage zum Schliessen eines Kästchens zu geben 3. nicht in der Mitte einer Kette einen Strich ziehen, der diese für den Agenten öffnen würde, 4. irgendeinen dritten Strich setzen. Dieser Gegner ist deutlich ausgeklügelter als der Trainingsgegner. Er weist jedoch trotzdem Schwachstellen auf, da er durch die dritte Priorität die Wahrscheinlichkeit, lange Ketten zu öffnen, nur verringert, aber nicht ausschliesst. Dies liegt daran, dass nur Striche ausgeschlossen werden, die gleichzeitig der dritte Strich von zwei Kästchen wären. Striche am Anfang oder Ende von Ketten können dadurch jedoch weiterhin gewählt werden.

Es gibt zwei Argumente, weshalb es sinnvoll ist, den Trainingsgegner und nicht den komplexen Evaluierungsgegner für das Trainieren zu verwenden. Ein Grund ist die Überanpassung. Würde der Agent nämlich auf den Evaluierungsgegner mit den vier Prioritäten trainiert werden, ist es wahrscheinlich, dass sich das KI-Modell zu sehr daran anpassen würde. Das heisst, der Agent würde gezielt Strategien entwickeln, die spezifische Schwachstellen des Evaluierungsgegners ausnutzen. Diese

Schwachstellen gibt es, da die komplexere Prioritätenhierarchie menschliches Verhalten nicht akkurat widerspiegelt. Die erlernten Taktiken wären gegen einen spielvertrauten Menschen somit nicht effektiv, da dieser nicht diese spezifischen Schwachstellen aufweist. Es ist deshalb sinnvoll, einen nicht zu komplexen Gegner für das Training zu verwenden, um unterschiedliche Strategien zu fördern. Trotzdem kann es auch beim Trainingsgegner zur Überanpassung des KI-Modells kommen. Aus diesem Grund ist es sinnvoll, für die Evaluierung einen separaten Evaluierungsgegner zu verwenden, damit die Überanpassung an den Trainingsgegner erkennbar wird. Zudem wird dadurch eine verlässliche Evaluierung der Gewinnrate eines KI-Modells gegen menschenähnliches Spielverhalten gewährleistet [20]. Der zweite Grund für die Verwendung eines Trainingsgegners für das Training ist das Erlernen der Grundlagen. Würde der Agent gegen den Evaluierungsgegner trainiert werden, bräuchte er deutlich länger, um die Spielgrundlagen zu erlernen. Dies, da der Agent aufgrund des komplexen Evaluierungsgegners kaum je dazu kommt, Kästchen zu schliessen. Somit sammelt er nur sehr begrenzt Erfahrungen, in denen er belohnt wird. Dadurch hat er Schwierigkeiten Muster zu erkennen und nachzuvollziehen, wofür er Belohnungen erhält. Der Trainingsgegner zieht hingegen, falls kein Kästchen geschlossen werden kann, zufällig einen Strich. Dadurch ergibt sich für den Agenten häufiger die Möglichkeit, eine unabsichtliche vom Gegner geschaffene Vorlage zu nutzen und ein Kästchen zu schliessen. Somit kann der Agent mehr Erfahrungen sammeln, in denen er belohnt wird und lernt somit schneller die Grundlagen.

4.2 Versuchsaufbau Experiment 1 (Belohnungsfunktionen)

In diesem Unterkapitel werden die verglichenen Belohnungsfunktionen und deren Unterschiede erläutert.

Die erste Fragestellung dieser Maturaarbeit lautet: Welche Auswirkungen haben verschiedene Belohnungsfunktionen auf das Lernen eines KI-Agenten im Spiel Käsekästchen? Zur Beantwortung dieser Fragestellung wurden drei verschiedene Belohnungsfunktionen entworfen. Zu Beginn jedes Zeitschritts wird die Belohnung auf 0 gesetzt. Abhängig von der gewählten Belohnungsfunktion erfolgt dann die Belohnung und Bestrafung wie in Tabelle 4.1 aufgeführt.

Tabelle 4.1: Zeigt die Belohnungsstruktur der verschiedenen Belohnungsfunktionen (B1), (B2) und (B3).

	Belohnungs- funktion 1 (B1)	Belohnungs- funktion 2 (B2)	Belohnungs- funktion 3 (B3)
Grundbelohnungen			
Eigenes Kästchen		+1	
Ungültiger Zug		-2	
Sieg/Niederlage		+1 / -1	

Spezifische Bestrafung			
Gegnerkästchen	-1	-	-0,5
Vorlage für Gegner erzeugt	-	-1	-0,5

Die Unterschiede zwischen den drei Belohnungsfunktionen liegen in den Bedingungen, die für eine Bestrafung erfüllt sein müssen. Bei der Belohnungsfunktion 1 (B1) wird der Agent reaktiv bestraft, wenn der Gegner ein Kästchen geschlossen hat. Dabei kann es sein, dass der Gegner innerhalb eines Zeitschritts mehrere Kästchen schliesst. Dann addiert sich die Bestrafung. Bei Belohnungsfunktion 2 (B2) wird der Agent vorausschauend für Vorlagen, also noch offene Kästchen mit drei Strichen, die er entweder nicht nutzt oder durch seine gewählte Aktion dem Gegner eröffnet, bestraft. Diese gegebenen Vorlagen werden in jedem Zeitschritt, sobald der Gegner an die Reihe kommt, überprüft. Durch dieses Vorgehen ist die Bestrafung unabhängiger von der gewählten Aktion des Gegners, was sinnvoll ist, da der Gegner menschliche Spielweise nicht exakt imitiert. Belohnungsfunktion 3 (B3) kombiniert diese beiden Ansätze. Somit setzt sich die Bestrafung halb aus (B1) und halb aus (B2) zusammen.

Aus den zuvor erläuterten Unterschieden ergibt sich folgende Hypothese 1 (H1): Es wird angenommen, dass die kombinierte Bestrafung (B3) zum Agenten mit der höchsten Gewinnrate führt.

4.3 Versuchsaufbau Experiment 2 (Hyperparameterwerte)

In diesem Unterkapitel werden die verglichenen Hyperparameter und die gewählten Testwerte erläutert.

Die zweite Fragestellung dieser Maturaarbeit lautet: Wie beeinflussen die wichtigsten Hyperparameter von DQN das Training eines KI-Agenten im Spiel Käsekästchen? Zur Beantwortung dieser Fragestellung wurden sieben Modelle mit Belohnungsfunktion 1 trainiert. Eines davon diente als gemeinsame Vergleichsbasis und setzte nur auf die Standardparameterwerte von *Stable-Baselines3*. Somit ist dieses Referenzmodell identisch mit dem Modell (B1) aus Experiment 1. Bei den anderen sechs Modellen hatte jeweils eines einen höheren und eines einen tieferen Wert als der SB3-Standardwert des entsprechenden Hyperparameters. Somit wurde der Wert jedes betrachteten Hyperparameters, also der Lernrate, des Diskontierungsfaktors und des Explorationsphasenanteils, zweimal variiert. Die tiefen (T1a-T1c) und hohen (T3a-T3c) Testwerte der Hyperparameter wurden von den SB3-Standardwerten ausgehend und basierend auf den während der Erprobung gewonnenen Erkenntnisse gewählt, um einen aussagekräftigen Bereich abzudecken. Für das Experiment 2 wurden Hyperparameterwerte getestet und dazugehörige Hypothesen aufgestellt. Diese sind in Tabelle 4.2 aufgeführt.

Tabelle 4.2: Übersicht über die in Experiment 2 variierten Hyperparameter und den zugehörigen Hypothesen.

Hyperparameter	Parameterwerte	Hypothese (H2a-c)
Lernrate	Tief (T1a): 0,00005 Standard (T2): 0,0001 Hoch (T3a): 0,001	(H2a): Es wird angenommen, dass (T3a) zum stärksten Agenten führt. (T3a) wird das Training zwar voraussichtlich instabil machen, trotzdem wird seine hohe Lernrate überwiegen, da die begrenzte Trainingsdauer für (T1a) und (T2) voraussichtlich nicht ausreicht.
Diskontierungsfaktor	Tief (T1b): 0,9 Standard (T2): 0,99 Hoch (T3b): 0,999	(H2b): Es wird angenommen, dass der ausgeglichene Wert (T2) zum stärksten Agenten führt, weil (T1b) langfristige Kettenstrategien vernachlässigt und (T3b) zu riskante Spielweisen fördert.
Explorationsphasenanteil	Tief (T1c): 0,05 Standard (T2): 0,1 Hoch (T3c): 0,25	(H2c): Es wird angenommen, dass (T1c) zum stärksten Agenten führt. Dies basiert auf der Annahme, dass der durch die kurze Explorationsphase erzielte Zeitgewinn für die Optimierung einer gefundenen Strategie die Vorteile einer längeren Explorationsphase überwiegt.

4.4 Evaluierungsmethoden

In diesem Unterkapitel werden die zur Evaluierung und Auswertung hinzugezogenen Metriken erläutert.

Tabelle 4.3: Übersicht über die zwei zur Evaluierung der Modelle verwendeten Metriken, deren Zweck sowie die dafür verwendeten Daten.

Metrik	Zweck der Metrik	Verwendete Daten
Gewinnrate gegen den Evaluierungsgegner	Dient der Bestimmung der Gewinnrate eines Modells gegen einen Gegner mit einem menschenähnlichen Spielverhalten während des Trainings.	Mit dem Evaluierungs-Rückruf (<code>EvalCallback(...)</code>) wurden alle 10'000 trainierten Zeitschritte 100 Spiele gegen den Evaluierungsgegner in der Evaluierungsumgebung gespielt. Daraus entstanden für jedes Modell 250 Datenpunkte. Um Rauschen auszugleichen, wurde eine Trendlinie über 30 Datenpunkte erzeugt, die den gleitenden Durchschnitt zeigt. Für die finalen Gewinnraten wurde jedes Modell zusätzlich am Ende über 1'000 Spiele separat evaluiert.

Kumulierte Belohnung eines Spiels	Experiment 1: Dient der Analyse der erlernten Lernstrategie eines Modells. Zeigt, wie gut und wie schnell ein Agent lernt, die Belohnung nach der jeweiligen Belohnungsfunktion zu maximieren. Experiment 2: Dient der Analyse des Lernprozesses, der Lerngeschwindigkeit und der Lernstabilität bei Verwendung der gleichen Belohnungsfunktion.	Die Metrik <i>rollout/ep_rew_mean</i> wurde während des Trainings immer wieder nach vier Episoden geloggt. Da eine Episode nicht immer gleich viele Zeitschritte lang dauert, gibt es je nach Modell unterschiedlich viele Datenpunkte, meist sind es ca. 50'000. Um Rauschen auszugleichen, wurden Mittelwerte über die vorherigen 500 Datenpunkte berechnet, die den gleitenden Durchschnitt zeigen. Diese Mittelwerte werden als Liniendiagramm visualisiert.
-----------------------------------	--	---

Um die Teilaussage von Hypothese (H2a) zu testen, dass das Training des Modells (T3a) durch grössere Anpassung pro Optimierungsschritt instabiler ist, wurden verschiedene statistische Tests durchgeführt. Um vergleichbare Werte zu erhalten, wurde für jede geloggte kumulierte Belohnung, für die ein Mittelwert über die letzten 500 Datenpunkte vorhanden ist, das Residuum berechnet. Dies ist die Differenz zwischen dem Mittelwert der letzten 500 Datenpunkte und dem tatsächlichen Wert dieses Datenpunkts. Je grösser die Varianz, also die Streuung, dieser Residuen ist, desto instabiler ist das Training. Da nicht alle tatsächlichen Daten geloggt wurden, geht man von der sogenannten Nullhypothese aus. Diese sagt aus, dass sich die tatsächlichen Werte der Gruppen nicht unterscheiden und beobachtete Differenzen nur durch Zufall zustande kommen. Der sogenannte p-Wert gibt die Wahrscheinlichkeit an, genau diese beobachteten Differenzen zu erhalten, unter der Annahme, dass diese Nullhypothese wahr ist. Ist der p-Wert unter 0,05, gilt die Nullhypothese als widerlegt. Das heisst, die untersuchten Werte der Gruppen unterscheiden sich statistisch signifikant.

Da in diesem Fall drei Varianzen miteinander verglichen wurden, wurde zuerst eine einfaktorielle ANOVA als Test durchgeführt. Bei einer solchen bedeutet die Widerlegung der Nullhypothese jedoch nur, dass sich mindestens eine der drei Gruppen (T1a, T2 und T3a) statistisch signifikant von einer anderen unterscheidet [21]. Damit schlussendlich bewiesen werden kann, dass sich die Varianzen aller drei Gruppen signifikant voneinander unterscheiden, werden sie danach in einem sogenannten Tukey-Kramer post-hoc Test paarweise miteinander verglichen. Ist der p-Wert jedes paarweisen Vergleichs dann unter 0,05, gilt als bewiesen, dass sich die Varianzen aller drei Gruppen signifikant voneinander unterscheiden [22]. Für die Durchführung dieser statistischen Tests wurde als Hilfsmittel eine Webseite genutzt, welche die ANOVA und verschiedene post-hoc Tests durchführte [23].

5 Ergebnisse und Diskussion

In diesem Kapitel werden die erfassten Daten zu Experiment 1 und 2 jeweils einzeln beschrieben und danach separat interpretiert.

5.1 Experiment 1: Vergleich der Belohnungsfunktionen

In diesem Unterkapitel werden die erfassten Daten zu Experiment 1 beschrieben und interpretiert.

5.1.1 Erfasste Daten zu Experiment 1

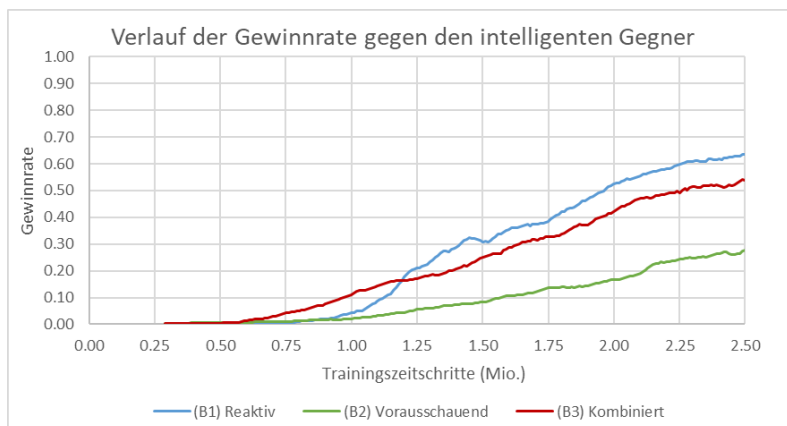


Abbildung 5.1: Verlauf der Gewinnrate gegen den Evaluierungsgegner (intelligenter Gegner). Abgebildet sind die geglätteten Lernkurven (eval/success_rate) der drei Modelle über 2,5 Millionen Trainingszeitschritte.

Abbildung 5.1 und Tabelle 5.1 zeigen, dass das Modell (B1) mit 0,66 die höchste Gewinnrate erzielt. Auffällig ist, dass das Modell (B3) als Erstes einen Leistungsanstieg aufweist, jedoch kurz vor 1,25 Mio. Zeitschritten von Modell (B1) übertroffen

wird.

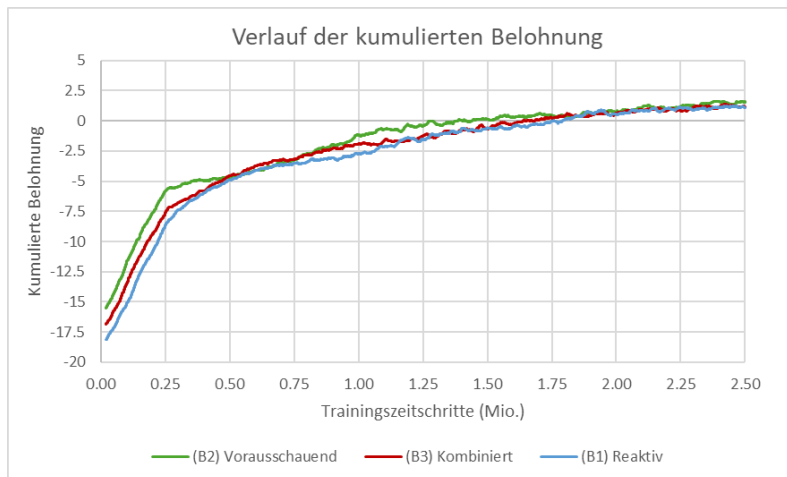


Abbildung 5.2: Verlauf der kumulierten Belohnung. Abgebildet sind die geglätteten Lernkurven (rollout/ep_rew_mean) der drei Modelle über 2,5 Millionen Trainingszeitschritte.

Abbildung 5.2 zeigt, dass Modell (B2) die höchste kumulierte Belohnung erzielt, jedoch alle Modelle ähnliche Endresultate liefern. Es fällt auf, dass sich die kumulierten Belohnungen der drei Modelle schon zu Beginn unterscheiden.

Interessant ist zudem, dass die drei Lernkurven zu zwei unterschiedlichen Zeitpunkten konvergent zu sein scheinen, zwischendurch jedoch auseinandergehen.

Tabelle 5.1: Zeigt die finalen Gewinnraten von (B1-B3) über 1000 Spiele gegen den Evaluierungsgegner.

Bezeichnung des Modells	Datei (.zip)	Belohnungsfunktion	Gewinnrate gegen Evaluierungsgegner	Rang
(B1)	B1	Reaktiv	0,66	1
(B2)	B2	Vorausschauend	0,32	3
(B3)	B3	Kombiniert	0,50	2

5.1.2 Vergleich und Interpretation der Belohnungsfunktionen

In Abbildung 5.1 ist zu erkennen, dass das Modell (B1) gegen den Schluss des Trainings die beste und Modell (B2) die schlechteste Gewinnrate gegen den Evaluierungsgegner erzielt. Gleichzeitig erhält (B2) in nicht konvergent scheinenden Abschnitten die höchste kumulierte Belohnung (Abbildung 5.2). Diese Daten stellen einen scheinbaren Widerspruch zueinander dar. Es existiert jedoch eine logische Erklärung dafür.

Eine Analyse der Belohnungsfunktionen zeigt, dass in Belohnungsfunktion 2 und 3 ein Schlupfloch existiert, wodurch gewisse geschlossene Kästchen des Gegners weniger oder gar nicht bestraft werden. Dies wurde dadurch begünstigt, dass der Trainingsgegner so aufgebaut ist, dass er immer, wenn möglich, offene schliesst. Das Problem dabei ist, dass Belohnungsfunktion 2 so konstruiert ist, dass nur beim Spielerwechsel überprüft wird, wie viele offene Kästchen mit drei Strichen existieren und den Agenten entsprechend bestraft. Schliesst der Gegner aber in einer vom Agenten geöffneten Kette ein Kästchen und schafft sich damit selbst eine neue Vorlage, die er nutzt, wird der Agent nur für die erste Vorlage bestraft, jedoch nicht für mögliche folgende. Mit Belohnungsfunktion 3 wird zwar jedes Kästchen des Gegners bestraft, jene, die jedoch durch das Ausnutzen des Schlupflochs entstehen, nur halb so stark. Die reaktive Bestrafung in Belohnungsfunktion 1 und in Belohnungsfunktion 3 überprüft anders als die vorausschauende Bestrafung nach jedem gezogenen Strich, ob der Gegner ein Kästchen schliessen konnte und bestraft dementsprechend den Agenten.

Es ist anzunehmen, dass dieses Schlupfloch schlussendlich dazu führte, dass die Modelle (B2) und (B3) suboptimale *Policies* entwickelten. Diese verfolgten Strategien, die das Schlupfloch nutzten, um weniger bestraft zu werden. Dies liefert eine mögliche Erklärung dafür, warum diese Modelle in der Trainingsumgebung phasenweise eine höhere kumulierte Belohnung als (B1) erzielen konnten.

In Abbildung 5.2 ist ein erster Konvergenzzeitraum zwischen 0,5 Mio. und 0,75 Mio. Zeitschritten zu erkennen. Dies deutet darauf hin, dass die drei Modelle zu diesem Zeitpunkt wichtige Spielprinzipien wie das Unterlassen, bereits existierende Striche neu zu ziehen, verstanden und umgesetzt hatten. Anschliessend konnten die Modelle (B2) und (B3) jedoch durch das Ausnutzen der Bestrafungsinkonsistenz des

Schlupflochs ihre kumulierten Belohnungen weiter steigern. Dies führte dazu, dass die Werte der kumulierten Belohnungen dann auseinanderliefen.

Die Bestrafungsinkonsistenz liefert auch eine mögliche Erklärung für den frühen Leistungsanstieg von Modell (B3). Da die Belohnung für ein eigenes Kästchen (+1) die Bestrafung (-0,5) für ein Kästchen, das durch Ausnutzung des Schlupflochs entsteht, überwiegt, konnte dieser Agent riskantere Strategien erlernen. Aus diesem Grund verfolgte (B3) zu Beginn riskantere Strategien als (B1) verfolgte, wodurch es anfänglich eine höhere Gewinnrate gegen den Evaluierungsgegner erzielte. Es wird angenommen, dass das Modell (B3) ab 1 Mio. das Schlupfloch nicht weiter ausnutzen konnte und sich stattdessen auf defensivere Strategien wie jene von Modell (B1) fokussierte.

Da die Belohnungsfunktion 1 alle geschlossenen Kästchen des Gegners gleich bestraft, sind Bestrafungen für das Öffnen von Ketten sehr hoch. Diese hohen Bestrafungen führten zu Beginn wahrscheinlich dazu, dass der Lernprozess von (B1) instabil war. Der Grund dafür ist, dass die daraus resultierenden hohen Werte der Verlustfunktion (4) dazu führten, dass Netzwerkparameter stärker angepasst wurden. Das Modell (B1) entwickelte daher erst nach ca. 1,1 Mio. Zeitschritten konstant gewinnbringende Strategien für das Spiel, was dann zu einem starken Anstieg der Gewinnrate führte. Dadurch lässt sich auch die daraus resultierende Annäherung der Werte der kumulierten Belohnung erklären. Letztlich führte die strikte Bestrafung der Belohnungsfunktion 1 dazu, dass das darauf trainierte Modell am Ende mit 0,66 die höchste finale Gewinnrate gegen den Evaluierungsgegner erzielte, wodurch die aufgestellte Hypothese (H1) widerlegt wird.

5.2 Experiment 2: Vergleich variierten Hyperparameterwerte

In diesem Unterkapitel werden die erfassten Daten zu Experiment 2 beschrieben und danach einzeln für jeden Hyperparameter interpretiert.

5.2.1 Erfasste Daten zu Experiment 2

Tabelle 5.2: Zeigt den Mittelwert, die Standardabweichung und die Varianz der Residuen zwischen über 500 Datenpunkten geglätteter und tatsächlicher kumulierter Belohnung der Modelle (T1a), (T2) und (T3a) während des Trainings.

Bezeichnung des Modells	Datei (.zip)	Mittelwert	Stichproben-Standardabweichung	Stichproben-Varianz
(T1a)	T1a	0,0692	0,3307	0,1093
(T2)	B1	0,0920	0,3966	0,1573
(T3a)	T3a	0,1287	0,5113	0,2614

Tabelle 5.2 zeigt, dass das Modell (T1a) den kleinsten Mittelwert und die geringste Stichproben-Standardabweichung aufweist, während das Modell (T3a) die

höchsten Werte besitzt. Die ANOVA der Residuen ergibt einen p-Wert von $1.1102e-16$. Tukey-Kramer post-hoc Tests danach für alle paarweisen Vergleiche der Modelle (T1a), (T2) und (T3a) ergeben p-Werte unter 0,05.

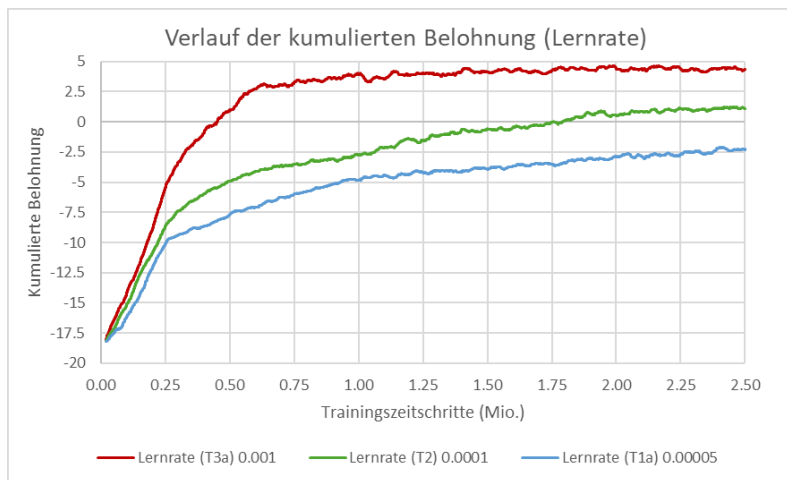


Abbildung 5.3: Verlauf der kumulierten Belohnung bei unterschiedlichen Lernraten. Abgebildet sind die Verläufe der geglätteten Lernkurven (rollout/ep_rew_mean) für drei verschiedene Lernraten über 2,5 Millionen Trainingszeitschritte.

Abbildung 5.3 zeigt, dass das Modell (T3a) früh ein sehr hohes, annähernd flaches Niveau erreicht. Das Modell (T1a) hingegen erzielt durchs gesamte Training hindurch die geringste kumulierte Belohnung. Auffällig ist, dass bei 0,25 Mio. Zeitschritten nur (T1a) einen klaren Knick aufweist.

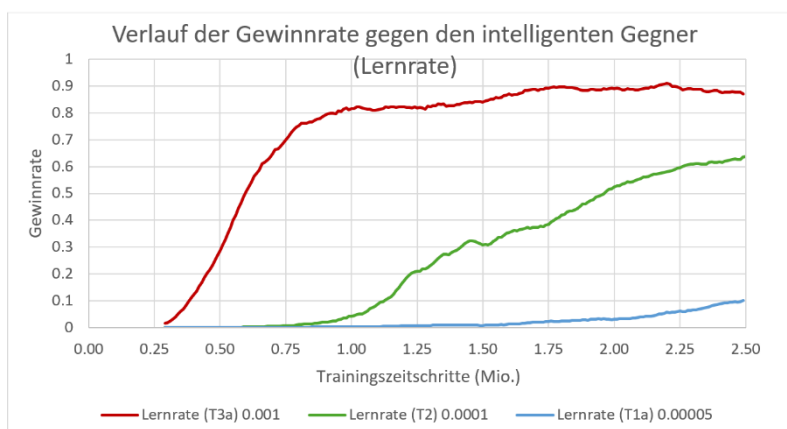


Abbildung 5.4: Verlauf der Gewinnrate gegen den Evaluierungsgegner (intelligenter Gegner) bei unterschiedlichen Lernraten. Abgebildet sind die geglätteten Lernkurven (eval/success_rate) für drei verschiedene Lernraten

Abbildung 5.4 und Tabelle 5.3 zeigen, dass die Gewinnrate von (T3a) schnell steigt und bei 1,0 Mio. Zeitschritten ein Leistungsplateau erreicht, das bis zum Ende des Trainings gehalten wird und eine finale Gewinnrate von 0,77 erzielt.

Währenddessen weist (T1a) erst ab 1,5 Mio. Zeitschritten einen geringen Anstieg auf.

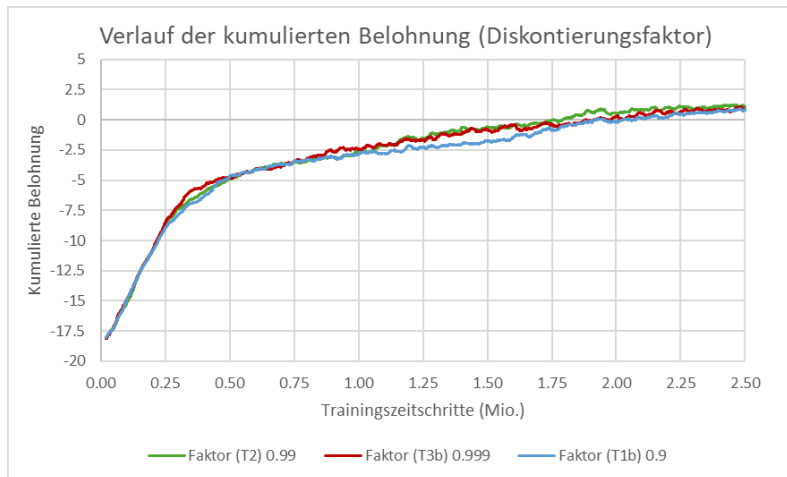


Abbildung 5.5: Verlauf der kumulierten Belohnung bei unterschiedlichen Diskontierungsfaktoren. Abgebildet sind die Verläufe der geglätteten Lernkurven (rollout/ep_rew_mean) für drei verschiedene Diskontierungsfaktoren über 2,5 Millionen Trainingszeitschritte.

Abbildung 5.5 zeigt, dass die Modelle (T1b), (T2) und (T3b) einen sehr ähnlichen Verlauf der kumulierten Belohnung haben. Interessant ist, dass (T1b) zwischen 1 Mio. und 1,75 Mio. Zeitschritten etwas tiefere Werte als die anderen beiden Modelle erzielt.

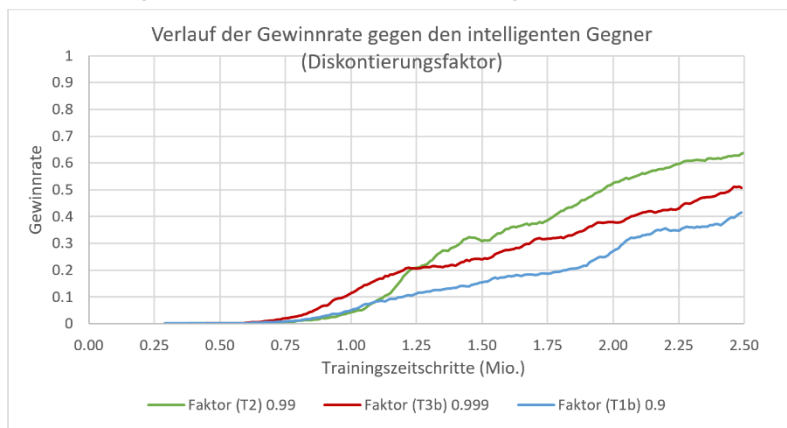


Abbildung 5.6: Verlauf der Gewinnrate gegen den Evaluierungsgegner (intelligenter Gegner) bei unterschiedlichen Diskontierungsfaktoren. Abgebildet sind die geglätteten Lernkurven (eval/success_rate) für drei verschiedene Diskontierungsfaktoren über 2,5 Millionen Trainingszeitschritte.

Abbildung 5.6 zeigt, dass Referenzmodell (T2) die höchste finale Gewinnrate erzielt. Es sticht heraus, dass (T3b) bis circa 1,25 Mio. Zeitschritten höhere Gewinnraten als (T2) erzielt, ab dann jedoch davon übertroffen wird.

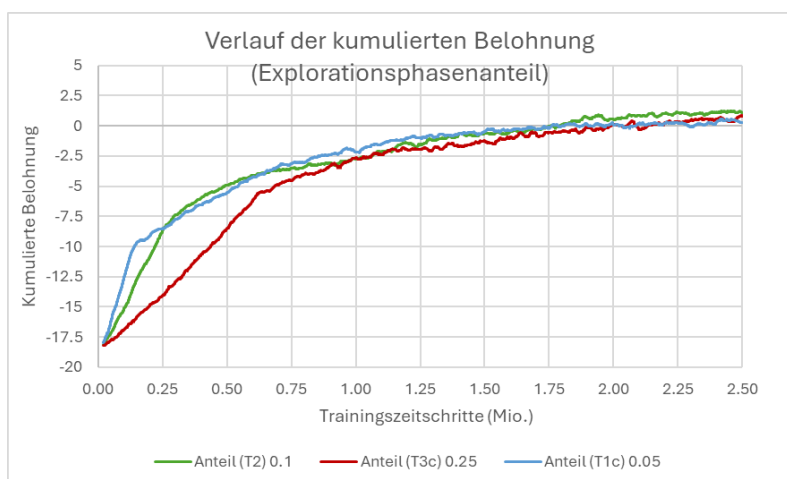


Abbildung 5.7: Verlauf der kumulierten Belohnung bei unterschiedlichen Explorationsphasenanteilen. Abgebildet sind die Verläufe der geglätteten Lernkurven (rollout/ep_rew_mean) für drei verschiedene Explorationsphasenanteile über 2,5 Millionen Trainingszeitschritte.

Abbildung 5.7 zeigt, dass jedes der drei Modelle zu Beginn eine scheinbare Steigung aufweist, die sich von den zwei anderen unterscheidet. Die Steigungen halten jeweils unterschiedlich lange an. Ab 1 Mio. Zeitschritten liegen die Kurven dann nahe beieinander.

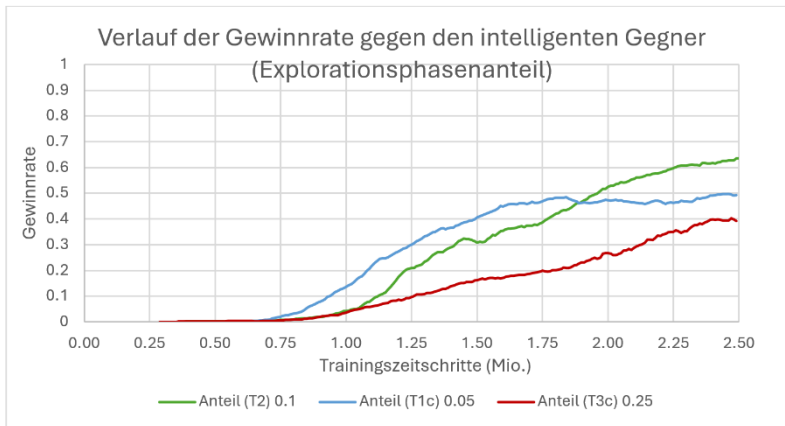


Abbildung 5.8: Verlauf der Gewinnrate gegen den Evaluierungsgegner (intelligenter Gegner) bei unterschiedlichen Explorationsphasenanteilen. Abgebildet sind die geglätteten Lernkurven (eval/success_rate) für drei verschiedene Explorationsphasenanteile über 2,5 Millionen

Abbildung 5.8 zeigt, dass Referenzmodell (T2) die höchste finale Gewinnrate erzielt. Auffallend ist, dass das (T1c) zwar als Erstes einen Leistungsanstieg verzeichnet, jedoch ab circa 1,75 Mio. Zeitschritten ein Plateau erreicht, auf dem es bis zum Ende des Trainings bleibt.

Tabelle 5.3: Zeigt die finalen Gewinnraten von (T1a-T1c), (T2) und (T3a-T3c) über 1000 Spiele gegen den Evaluierungsgegner. Der Rang bezieht sich auf die Gewinnrate innerhalb der jeweiligen Hyperparametergruppe.

Bezeichnung des Modells	Datei (.zip)	Parameterwert	Gewinnrate gegen Evaluierungsgegner	Rang
Vergleich der Lernraten				
(T1a)	T1a	0,00005	0,11	3
(T2)	B1	0,0001	0,66	2
(T3a)	T3a	0,001	0,77	1
Vergleich der Diskontierungsfaktoren				
(T1b)	T1b	0,9	0,52	2
(T2)	B1	0,99	0,66	1
(T3b)	T3b	0,999	0,5	3
Vergleich der Explorationsphasenanteile				
(T1c)	T1c	0,05	0,44	3
(T2)	B1	0,1	0,66	1
(T3c)	T3c	0,25	0,45	2

Auffällig ist, dass in Tabelle 5.3 die Rangfolgen und finalen Gewinnraten der Modelle (T1b) und (T3b) sowie (T1c) und (T3c) nicht mit den Daten auf Abbildung 5.6 bzw. Abbildung 5.8 übereinstimmen. Diese Abweichungen haben mehrere Ursachen. Erstens wurden die Daten zu den Gewinnraten über 30 Datenpunkte, sprich 300'000 Zeitschritte, geglättet. Zweitens basierte jeder Datenpunkt während des Trainings auf 100 gegen den Evaluierungsgegner gespielten Spielen, während zur Erhebung der finalen Gewinnrate 1'000 Spiele simuliert wurden.

5.2.2 Vergleich und Interpretation der Lernraten

Die Daten zu den Modellen (T1a), (T2) und (T3a) für den Lernratenwertevergleich zeigen, dass (T3a) mit der höchsten Lernrate von 0,001 mit 0,77 die höchste finale Gewinnrate erzielte. Die anderen beiden Modelle scheinen tiefere Plateaus anzustreben, sodass sie das Niveau von (T3a) nicht erreichen.

Es wird angenommen, dass die Trainingsdauer für die langsamere Lerngeschwindigkeit von Modell (T1a) und (T2) zu gering war und dass dies der Grund für ihre geringeren finalen Gewinnraten von 0,11 respektive 0,66 ist. Eine mögliche Erklärung für den «Knick» der kumulierten Belohnung von (T1a) bei 0,25 Mio. Zeitschritten in Abbildung 5.3 ist das Ende der Explorationsphase in Kombination mit der geringen Lernrate. Nach Abschluss der Explorationsphase konnten die zuvor gefundenen simplen Strategien nur noch leicht verfeinert, jedoch nicht weiter ausgebaut oder gross verbessert werden. Es ist also wahrscheinlich, dass diese beiden Modelle eine längere Explorationsphase und ein längeres Training benötigt hätten, um komplexere Strategien entwickeln zu können. Dies liegt daran, dass sie aufgrund ihrer kleinen Anpassungen der Netzwerkparameter pro Optimierungsschritt konservativer agieren, wodurch das Entwickeln komplexer Strategien mehr Zeit benötigt. Dies hat zwar den Vorteil, dass der Lernprozess stabiler ist und die Trainingsergebnisse konstanter sind, braucht jedoch deutlich länger.

Eine ANOVA und drei paarweise durchgeführte Tukey-Kramer post-hoc Tests bestätigten, dass die Varianzen der Residuen der kumulierten Belohnungen von Modell (T1a), (T2) und (T3a) sich statistisch signifikant voneinander unterscheiden. Das bedeutet, dass der Lernprozess von (T3a) während des Trainings tatsächlich instabiler war als die beiden anderen Modelle. Daraus wird gefolgert, dass (T3a) durch eine progressivere, jedoch instabilere Vorgehensweise schon früh Strategien entwickeln und damit Erfahrungen sammeln konnte, die ihm halfen, daraus komplexere Strategien zu entwickeln.

Da die beiden anderen Modelle nicht schon während der Explorationsphase solche komplexen Strategien entwickeln konnten, blieben sie während des restlichen Trainings an einfachen, zuvor erprobten Strategien gebunden. Danach war aufgrund des niedrigen verbleibenden Epsilon-Werts von 0,05 nicht mehr ausreichend Exploration möglich, um die komplexen Strategien von (T3a) zu erlernen. Es ist anzunehmen, dass der Unterschied zwischen (T1a) und (T2) auch aufgrund des Lernratenunterschieds zustande kam. Wie diese Ergebnisse zeigen, führt eine Lernrate von 0,001 wie in Modell (T3a) im Spiel Käsekästchen verglichen mit (T1a) und (T2) tatsächlich zum leistungsstärksten Modell. Damit wird die Hypothese (H2a) bestätigt.

5.2.3 Vergleich und Interpretation der Diskontierungsfaktoren

Die Daten zu den Modellen (T1b), (T2) und (T3b) zeigen, dass das Referenzmodell (T2) mit Diskontierungsfaktorwert 0,99 die höchste finale Gewinnrate und kumulierte Belohnung erzielte. Modell (T3b) erzielte jedoch mit einem Diskontierungsfaktorwert von 0,999 den frühesten erkennbaren Anstieg der kumulierten Belohnung.

Es wird angenommen, dass ein höherer Diskontierungsfaktorwert wie in Modell (T3b) zum Erlernen der Grundlagen des Spiels von Vorteil war. Modell (T3b) zeigte den frühesten Leistungsanstieg, stagnierte dann jedoch leicht kurz vor 1,25 Mio. Zeitschritten. Eine mögliche Erklärung dafür ist, dass der Agent von (T3b) bis zu diesem Zeitpunkt risikoreiche Strategien in Kauf nahm, um auch mögliche Belohnungen, die weit in der Zukunft lagen, zu erhalten. Dabei vernachlässigte er jedoch gleichzeitig kleinere Chancen und handelte zu langfristig orientiert. Bei langfristigen Strategien kam es jedoch häufig vor, dass die Strategie nicht wie geplant umgesetzt werden konnte, da der Gegner aufgrund der Zufallskomponente nicht berechenbar handelte. Passierte das, befand sich der Agent dann in einer Falle, die eigentlich er dem Gegner gestellt hatte. Irgendwann kam der Agent an einen Punkt, an dem er diese langfristigen Strategien ausgereizt hatte. Ab dann konnte er mit diesem Vorgehen die kumulierte Belohnung nicht mehr weiter maximieren. Aus diesem Grund nutzte er dann wohl vermehrt auch kleinere Chancen, sodass er nicht mehr selbst in seine eigenen Fallen tappte und die kumulierte Belohnung maximieren konnte.

Es ist anzunehmen, dass das Referenzmodell (T2) hingegen schon von Beginn an kleinere Chancen nutzte, jedoch noch ohne die übergeordneten strategischen Zusammenhänge zu verstehen. Deshalb unterlag es zu Beginn (T3b), das zu diesem Zeitpunkt noch von seiner Risikobereitschaft profitierte. Als das Referenzmodell (T2) dann aber die Zusammenhänge der strategischen Komponenten verstand, konnte es auf der Grundlage des bereits Erlernten sehr starke Strategien entwickeln und übertraf (T3b).

Modell (T1b) entwickelte wohl nur kurzfristige Strategie. Aufgrund seines tiefen Diskontierungsfaktors von 0,9 fokussierte es vor allem darauf, möglichst jede Möglichkeit, die sich anbietet, zu nutzen. Dies auch dann, wenn es dem Gegner möglicherweise im späteren Spielverlauf ermöglichte, lange Ketten zu schliessen. Die unmittelbare Belohnung für den Agenten überwog mögliche langfristige Konsequenzen.

Wie diese Ergebnisse zeigen, führt ein Diskontierungsfaktor von 0,99 wie im Referenzmodell (T2) zum leistungsstärksten Modell, wie in Hypothese (H2b) angenommen. Dadurch wird dieser Teil der Hypothese (H2b) bestätigt.

5.2.4 Vergleich und Interpretation der Explorationsphasenanteile

Die Daten zu den Modellen (T1c), (T2) und (T3c) für den Vergleich der Explorationsphasenanteile zeigen, dass das Referenzmodell (T2) mit einem Explorationsphasenanteil von 0,1 die höchste finale Gewinnrate und kumulierte Belohnung erzielte. Dies ist auffällig, da bis kurz vor Ende des Trainings das Modell (T1c) mit einem Explorationsphasenanteil von 0,05 (T2) übertraf. Eine Erklärung

dafür liefert das im Unterkapitel 2.1 erklärte Dilemma zwischen Ausbeutung und Exploration.

Das Modell (T1c) war in der Lage, bereits nach 5 % der Trainingszeitschritte mit der Optimierung der Netzwerkparameter zu beginnen. Dies führte zu einem sehr frühen Anstieg der Leistung. Dieser Anstieg blieb bestehen, bis ein Leistungsplateau erreicht war. Dieses Plateau wurde erreicht, weil (T1c) aufgrund der wenigen Erfahrung zwar komplexe Strategien entwickeln konnte, diese jedoch den Strategien von (T2) unterlegen waren, da diese auf mehr Spielszenarien übertragbar waren.

Das Modell mit der längsten Explorationsphase (T3c) verbrachte zu viel Zeit mit dem Ausprobieren neuer Strategien. Es entdeckte dadurch zwar potenziell eine grössere Vielfalt an Strategien als die beiden anderen Modelle, hatte jedoch zu wenig Zeit, um die beste Strategie ausreichend zu erlernen und zu verfeinern.

Das Referenzmodell (T2) ging den Mittelweg zwischen den zwei anderen Modellen mit einem Explorationsphasenanteil von 0,1 und bildete somit einen guten Kompromiss zwischen Exploration und Ausbeutung. Damit war es ihm möglich, in der Explorationsphase genug auszuprobieren, um auch komplexere Strategien zu finden. Danach blieb ihm noch genug Zeit, um diese Strategien zu verfeinern.

Wie diese Ergebnisse zeigen, führt ein Explorationsphasenanteil von 0,1, wie beim Modell (T2), zum leistungsstärksten Modell, entgegen der Annahme in der Hypothese (H2c), dass der Wert von Modell (T1c) zur höchsten Gewinnrate führt. Damit wird die Hypothese (H2c) widerlegt.

6 Fazit

Dieses Kapitel fasst die gewonnenen Erkenntnisse zusammen und reflektiert danach kritisch über diese sowie über mögliches Verbesserungspotenzial an den Experimenten.

6.1 Zentrale gewonnene Erkenntnisse

In diesem Unterkapitel werden die zentralen gewonnenen Erkenntnisse zusammengefasst.

Die Ergebnisse verdeutlichen, dass das Aufstellen einer Belohnungsfunktion sowie das Wählen der Hyperparameter nicht so trivial ist, wie es auf den ersten Blick scheinen mag. Der Grund dafür ist, dass Belohnungsfunktion und Hyperparameter mit den restlichen Komponenten, aus denen eine Trainingsumgebung aufgebaut ist, stark zusammenhängen.

Die Interpretation der gesammelten Daten der drei Belohnungsfunktionen zeigt, dass es Sinn macht, Bestrafungen für einen Agenten reaktiv an die Spielzüge des Gegners zu koppeln. Ein Grund dafür ist, dass es schwer ist, korrekt mögliche Spielverläufe vorherzusehen. Dies zeigte sich gerade bei dem im Abschnitt 5.1.2 erklärten Schlupfloch. Durch dieses war es dem Agenten möglich, teilweise nicht für vom Gegner geschlossene Kästchen bestraft zu werden, da diese im Voraus nicht erkannt wurden. Ein weiterer Grund, weshalb die reaktive Bestrafung zum leistungsstärksten Modell führte, war die Logik des Trainingsgegners. Dieser hatte nur zwei Prioritäten, aber die erste davon war entscheidend. Zuerst schloss er immer, wenn möglich, offene Kästchen. Diese erste Priorität führte zu Situationen, in denen eine reaktive Bestrafung gut durch ihre erzieherische Wirkung funktionierte. Es ist anzunehmen, dass mit einem Gegner, der basierend auf reinem Zufall agiert hätte, die Ergebnisse anders ausgefallen wären. Es kann somit gesagt werden, dass es für das Spiel Käsekästchen bei einem regelbasierten Trainingsgegner, der eine simple Spielstrategie verfolgt, am effektivsten ist, in der Belohnungsfunktion reaktiv zu bestrafen.

Die Interpretation der gesammelten Daten zu den Hyperparameterwertevergleichen führte zu mehreren Erkenntnissen. Bei der Lernrate zeigte sich, dass eine hohe Lernrate trotz instabilem Training klar zum leistungsstärksten Modell führt. Durch grössere Optimierungsschritte war es dem Agenten möglich, sich komplexere Spielstrategien anzueignen, die er mit tieferen Lernraten innerhalb der gegebenen Trainingszeit nicht erlernen konnte.

Bei dem Diskontierungsfaktor zeigte sich, dass das Spiel Käsekästchen eine ausgewogene Strategie erfordert. Ein tiefer Diskontierungsfaktor führte zu suboptimalem kurzfristigem Handeln. Ein hoher Diskontierungsfaktor führte zu strategisch langfristigem Handeln, das jedoch häufig aufgrund des hohen Risikos

ineffektiv war. Obwohl also maximal langfristige Strategien vorteilhaft erscheinen, sind sie fragil und führen im Spiel Käsekästchen nicht zum besten Ergebnis.

Schliesslich zeigte der Vergleich der Explorationsphasenanteile das klassische Dilemma zwischen Ausbeutung und Exploration. Eine zu kurze Explorationsphase verhinderte das Entdecken überlegener Strategien. Gleichzeitig konnte eine ausgiebige Explorationsphase den Nachteil einer kürzeren Optimierungsphase nicht ausgleichen.

6.2 Kritische Reflexion und Verbesserungspotenzial

In diesem Unterkapitel wird über diese Maturaarbeit und die durchgeführten Experimente kritisch reflektiert.

Eine zentrale methodische Entscheidung, die die Ergebnisse der Experimente stark beeinflusst hat, war die Wahl des regelbasierten Trainingsgegners. Durch diesen wurde der Agent mit einem teilweise vorhersehbaren, menschenähnlichen Spielverhalten konfrontiert. Ein Experiment mit einem Trainingsgegner, welcher basierend auf einer definierten Wahrscheinlichkeit zufällig oder regelbasiert handelt, könnte eine sinnvolle Erweiterung sein. So könnte untersucht werden, wie effektiv eine reaktive Belohnungsfunktion verglichen mit einer vorausschauenden und kombinierten ist, wenn der Trainingsgegner weniger vorhersehbar handelt. Es ist wahrscheinlich, dass in einem solchen Versuch Modelle, welche mit vorausschauender Belohnungsfunktion trainiert wurden, gegen den Evaluierungsgegner anderen Modellen überlegen wären. Ein mit reaktiver Belohnungsfunktion trainiertes Modell könnte Strategien entwickeln, welche vor allem die Zufallskomponente ausnutzen, jedoch gegen den Evaluierungsgegner ineffektiv wären. Gleichzeitig wäre das Schlupfloch der Modelle (B2) und (B3) bei einem solchen Trainingsgegner weniger effektiv nutzbar.

Im zweiten Experiment dieser Maturaarbeit wurden für drei Hyperparameter jeweils drei Werte verglichen. Dabei wurden Kombinationen variierten Parameterwerte jedoch nicht untersucht. Es wäre jedoch mit den vorhandenen Ressourcen sehr schwer, ein systematischer Vergleich kombinierter variierten Parameterwerte durchzuführen. Zum einen ist es nicht möglich, mit einer kleinen Anzahl an Modellen den Bereich an Kombinationen sinnvoll abzudecken und zum anderen nimmt das Training viel Zeit in Anspruch.

Eine weitere mögliche Erweiterung wäre, dass mehr einzeln variierte Werte der Hyperparameter verglichen würden, da die Testwerte jeweils lediglich den Standardwert von *Stable-Baselines3* sowie einen tieferen und einen höheren Wert umfassten. Da diese Werte über ein grosses Spektrum verteilt sind, wäre es sinnvoll, mehr Parameterwerte mit kleineren Zwischenschritten zu vergleichen.

Wie in Unterkapitel 3.3 erläutert, gibt es noch weitere Hyperparameter, wie beispielsweise die Anzahl der Trainingszeitschritte, die in dieser Maturaarbeit nicht variiert wurden. Eine mögliche zukünftige Erweiterung dieser Maturaarbeit wäre also der Vergleich von Parameterwerten nicht untersuchter Hyperparameter.

Diese Maturaarbeit beschränkte sich auf eine Spielfeldgrösse von vier mal vier Punkten. In Zukunft wäre ein Vergleich verschiedener Spielfeldgrössen interessant.

Literaturverzeichnis

- [1] J. Murel und E. Kavlakoglu, «Was ist verstärkendes Lernen?», IBM, 25 März 2024. [Online]. Available: <https://www.ibm.com/de-de/think/topics/reinforcement-learning>. [Zugriff am 9 Juni 2025].
- [2] Alexander Thamm GmbH, «Reinforcement Learning: Kompakt erklärt,» Alexander Thamm, 22 Juni 2023. [Online]. Available: <https://www.alexanderthamm.com/de/blog/einfach-erklaert-so-funktioniert-reinforcement-learning/>. [Zugriff am 9 Juni 2025].
- [3] T. Simonini, O. Sanseviero und S. Paul, «The Reinforcement Learning Framework,» Hugging Face, 8 August 2023. [Online]. Available: <https://huggingface.co/learn/deep-rl-course/unit1/rl-framework>. [Zugriff am 9 Juni 2025].
- [4] T. Simonini, O. Sanseviero und S. Paul, «The Exploration/Exploitation trade-off,» Hugging Face, 18 April 2023. [Online]. Available: <https://huggingface.co/learn/deep-rl-course/unit1/exp-exp-tradeoff>. [Zugriff am 9 Juni 2025].
- [5] T. Simonini, O. Sanseviero und S. Paul, «Two main approaches for solving RL problems,» Hugging Face, 27 Juni 2023. [Online]. Available: <https://huggingface.co/learn/deep-rl-course/unit1/two-methods>. [Zugriff am 9 Juni 2025].
- [6] T. Simonini, O. Sanseviero und S. Paul, «Introducing Q-Learning,» Hugging Face, 2 Juni 2025. [Online]. Available: <https://huggingface.co/learn/deep-rl-course/unit2/q-learning>. [Zugriff am 23 Juni 2025].
- [7] Europäisches IT-Zertifizierungsinstitut, «Was ist die Bellman-Gleichung und wie wird sie im Kontext von Temporal Difference (TD)-Lernen und Q-Learning verwendet?,» Europäisches IT-Zertifizierungsinstitut, 11 Juni 2024. [Online]. Available: <https://bit.ly/Kurzlink-zu-Bellman-Gleichung-und-TD-Lernen-und-Q-Learning-von-EITCA>. [Zugriff am 24 Juni 2025].
- [8] N. Van Otten, «Deep Q-Learning [Reinforcement Learning] Explained & How To Example,» Spot Intelligence, 12 Mai 2025. [Online]. Available: <https://spotintelligence.com/2025/05/12/deep-q-learning-reinforcement-learning/>. [Zugriff am 23 Juni 2025].
- [9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie,

A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg und D. Hassabis, «Human-level control through deep reinforcement learning,» *Nature*, Bd. 518, Nr. 7540, pp. 529-533, 2015.

- [10] C. Bento, «Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis,» TDS Archive, 21 September 2021. [Online]. Available: <https://medium.com/data-science/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141>. [Zugriff am 25 Juni 2025].
- [11] P. Caceres, «The Multilayer Perceptron - Theory and Implementation of the Backpropagation Algorithm,» pabloinsente.github.io, 20 März 2020. [Online]. Available: <https://pabloinsente.github.io/the-multilayer-perceptron>. [Zugriff am 26 Juni 2025].
- [12] GeeksforGeeks, «Deep Q-Learning in Reinforcement Learning,» GeeksforGeeks, 4 Juni 2025. [Online]. Available: <https://www.geeksforgeeks.org/deep-q-learning/>. [Zugriff am 24 Juni 2025].
- [13] Turing, «Detailed Explanation of Deep Neural Network, Multilayer Perceptron, and Deep Q-Network,» Turing, 11 April 2022. [Online]. Available: <https://www.turing.com/kb/explanation-of-deep-neural-network-multilayer-perceptron-deep-q-network>. [Zugriff am 26 Juni 2025].
- [14] Farama Foundation, «Basic Usage,» [Online]. Available: https://gymnasium.farama.org/introduction/basic_usage/. [Zugriff am 27 Juni 2025].
- [15] Stable-Baselines3, «DQN,» [Online]. Available: <https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html>. [Zugriff am 27 Juni 2025].
- [16] NumPy, «The N-dimensional array (ndarray),» [Online]. Available: <https://numpy.org/doc/stable/reference/arrays.ndarray.html#arrays-ndarray>. [Zugriff am 27 Juni 2025].
- [17] NumPy, «numpy.random.Generator.integers,» [Online]. Available: <https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.integers.html>. [Zugriff am 27 Juni 2025].
- [18] Pygame, «README,» 17 Juni 2024. [Online]. Available: <https://github.com/pygame/pygame/blob/main/README.rst>. [Zugriff am 27 Juni 2025].

- [19] Stable-Baselines3, «Policy Networks,» [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html. [Zugriff am 2 August 2025].
- [20] N. Lang, «Was ist Overfitting?,» Data Basecamp, 10 Dezember 2022. [Online]. Available: <https://databasecamp.de/ki/overfitting>. [Zugriff am 30 Juli 2025].
- [21] W. Hemmerich, «Einfaktorielle ANOVA in SPSS,» 2025. [Online]. Available: <https://statistikguru.de/spss/einfaktorielle-anova/einleitung.html>. [Zugriff am 8 August 2025].
- [22] W. Hemmerich, «Einfaktorielle ANOVA: Den Tukey post-hoc Test interpretieren,» 2025. [Online]. Available: <https://statistikguru.de/spss/einfaktorielle-anova/tukey-post-hoc-test.html>. [Zugriff am 8 August 2025].
- [23] N. Vasavada, «One-way ANOVA with post-hoc Tukey HSD Test Calculator,» 2016. [Online]. Available: https://astatsa.com/OneWay_Anova_with_TukeyHSD/_get_data/. [Zugriff am 8 August 2025].

Abbildungsverzeichnis

Abb. 2.1: Schema eines Markovschen Entscheidungsprozesses im Reinforcement Learning.

Urheber: EBattleP

Quelle: https://commons.wikimedia.org/wiki/File:Markov_diagram_v2.svg

Lizenz: Creative Commons Attribution-Share Alike 4.0 International (CC BY-SA 4.0), abrufbar unter: <https://creativecommons.org/licenses/by-sa/4.0/>

Anmerkung: Eigene Bearbeitung (Übersetzung). Die bearbeitete Abbildung steht unter derselben Lizenz. 6

Abb. 2.2: Aufbau und Funktionsweise einer Q-Tabelle eines Beispielspiels. Aus jedem Zustand können dieselben Aktionen ausgeführt werden. Links wird die Q-Tabelle initialisiert und rechts dann durch Training mit Werten gefüllt.

Urheber: Thomas Simonini, Omar Sanseviero

Quelle: <https://github.com/huggingface/deep-rl-class/blob/main/units/en/unit2/q-learning.mdx>

Lizenz: Apache License 2.0, abrufbar unter: <https://www.apache.org/licenses/LICENSE-2.0>

Anmerkung: Eigene Bearbeitung (Eigene Bildkomposition aus mehreren Abbildungen derselben Quelle mit eigenen Beschriftungen). Die bearbeitete Abbildung steht unter derselben Lizenz. 8

Abb. 2.3: Schema eines Perzeptrons. Es empfängt Eingabewerte und gewichtet und summiert diese. Die resultierende Summe wird durch die ReLU-Aktivierungsfunktion transformiert. Das Ergebnis bildet den Ausgabewert des n-ten Neurons der ersten verborgenen Schicht.

Urheber: Mayranna

Quelle: https://commons.wikimedia.org/wiki/File:Perceptron_moj.png

Lizenz: Creative Commons Attribution-Share Alike 3.0 Unported (CC BY-SA 3.0), abrufbar unter: <https://creativecommons.org/licenses/by-sa/3.0/>

Anmerkung: Eigene Bearbeitung (Es wurden eigene Elemente und Beschriftungen hinzugefügt und vorhandene Elemente entfernt.) Die bearbeitete Abbildung steht unter derselben Lizenz. 10

Abb. 2.4: Schema eines Multilayer Perceptron (MLP). Das Netzwerk besteht aus einer Eingabeschicht, zwei verborgenen Schichten und einer Ausgabeschicht. Die Neuronen einer Schicht sind jeweils vollständig mit denen der folgenden Schicht verbunden. Bei einem Q-Netzwerk liefert die Ausgabeschicht die Q-Werte.

Urheber: Alex Lenail

Quelle: <https://alexlenail.me/NN-SVG/>

Lizenz: MIT License, abrufbar unter: <https://opensource.org/licenses/MIT>

Anmerkung: Quelle ist ein Tool, zum Erstellen von Bildern von tiefen neuronalen Netzwerken. Eigene Bearbeitung (Es wurden Beschriftungen hinzugefügt). Die bearbeitete Abbildung steht unter derselben Lizenz. 11

Abb. 3.1: Ablaufschema eines Beispiel-Käsekästchen-Spiels mit strategischem Opfer. Die Abfolge zeigt das strategische Element der Ketten. Spieler 1 opfert strategisch ein einzelnes Kästchen an Spieler 2. Dadurch wird Spieler 2 gezwungen, die verbleibende lange Kette von potenziellen Kästchen zu öffnen. Anschliessend kann Spieler 1 diese Kette vervollständigen und gewinnt dadurch das Spiel.

Urheber: Fabian Samuel Rütschi

Anmerkung: Abbildung wurde selbst erstellt mithilfe von PowerPoint 15

Abb. 3.2: Darstellung des Beobachtungsraums und seiner Indexzuordnung. Links ist der Beobachtungsraum als dreidimensionales Array visualisiert. Die Striche, die zum geschlossenen Kästchen auf dem mittleren Spielfeld gehören, sind entsprechend ihrer Array-Werte farblich hervorgehoben (siehe Werte-Legende unten). Rechts wird die Zuordnung der einzelnen Indizes zur Position eines Strichs im Spielfeld veranschaulicht.

Anmerkung: Das dreidimensionale Array wurde mithilfe von Matplotlib visualisiert. Der Code dafür wurde von Gemini generiert. Abbildung wurde mithilfe von PowerPoint erstellt. 16

Abb. 3.3: Darstellung der ungültigen Rand-Indizes im Beobachtungsraum. Jene Positionen des Arrays, welche keinem Strich des Spielfelds zugehörig sind, sind im Array links orange hervorgehoben. Diese Array-Positionen werden während des Spiels ignoriert und behalten immer den Wert 0.

Anmerkung: Das dreidimensionale Array wurde mithilfe von Matplotlib visualisiert. Der Code dafür wurde von Gemini generiert. Abbildung wurde mithilfe von PowerPoint erstellt. 16

Abb. 5.1: Verlauf der Gewinnrate gegen den Evaluierungsgegner (intelligenter Gegner). Abgebildet sind die geglätteten Lernkurven (eval/success_rate) der drei Modelle über 2,5 Millionen Trainingszeitschritte.

Anmerkung: Das Punktediagramm mit gleitendem Durchschnitt über 30 Datenpunkte wurde mit Excel erstellt und basiert auf je 250 Messwerten. .24

Abb. 5.2: Verlauf der kumulierten Belohnung. Abgebildet sind die geglätteten Lernkurven (rollout/ep_rew_mean) der drei Modelle über 2,5 Millionen Trainingszeitschritte.

Anmerkung: Das Punktediagramm mit gleitendem Durchschnitt über 500 Datenpunkte wurde mit Excel erstellt und basiert auf je circa 50'000 Messwerten. 24

- Abb. 5.3:** Verlauf der kumulierten Belohnung bei unterschiedlichen Lernraten. Abgebildet sind die Verläufe der geglätteten Lernkurven (rollout/ep_rew_mean) für drei verschiedene Lernraten über 2,5 Millionen Trainingszeitschritte.
Anmerkung: Das Punktediagramm mit gleitendem Durchschnitt über 30 Datenpunkte wurde mit Excel erstellt und basiert auf je 250 Messwerten. .27
- Abb. 5.4:** Verlauf der Gewinnrate gegen den Evaluierungsgegner (intelligenter Gegner) bei unterschiedlichen Lernraten. Abgebildet sind die geglätteten Lernkurven (eval/success_rate) für drei verschiedene Lernraten über 2,5 Millionen Trainingszeitschritte.
Anmerkung: Das Punktediagramm mit gleitendem Durchschnitt über 500 Datenpunkte wurde mit Excel erstellt und basiert auf je circa 50'000 Messwerten. 27
- Abb. 5.5:** Verlauf der kumulierten Belohnung bei unterschiedlichen Diskontierungsfaktoren. Abgebildet sind die Verläufe der geglätteten Lernkurven (rollout/ep_rew_mean) für drei verschiedene Diskontierungsfaktoren über 2,5 Millionen Trainingszeitschritte.
Anmerkung: Das Punktediagramm mit gleitendem Durchschnitt über 30 Datenpunkte wurde mit Excel erstellt und basiert auf je 250 Messwerten. .28
- Abb. 5.6:** Verlauf der Gewinnrate gegen den Evaluierungsgegner (intelligenter Gegner) bei unterschiedlichen Diskontierungsfaktoren. Abgebildet sind die geglätteten Lernkurven (eval/success_rate) für drei verschiedene Diskontierungsfaktoren über 2,5 Millionen Trainingszeitschritte.
Anmerkung: Das Punktediagramm mit gleitendem Durchschnitt über 500 Datenpunkte wurde mit Excel erstellt und basiert auf je circa 50'000 Messwerten. 28
- Abb. 5.7:** Verlauf der kumulierten Belohnung bei unterschiedlichen Explorationsphasenanteilen. Abgebildet sind die Verläufe der geglätteten Lernkurven (rollout/ep_rew_mean) für drei verschiedene Explorationsphasenanteile über 2,5 Millionen Trainingszeitschritte.
Anmerkung: Das Punktediagramm mit gleitendem Durchschnitt über 30 Datenpunkte wurde mit Excel erstellt und basiert auf je 250 Messwerten. .28
- Abb. 5.8:** Verlauf der Gewinnrate gegen den Evaluierungsgegner (intelligenter Gegner) bei unterschiedlichen Explorationsphasenanteilen. Abgebildet sind die geglätteten Lernkurven (eval/success_rate) für drei verschiedene Explorationsphasenanteile über 2,5 Millionen Trainingszeitschritte.
Anmerkung: Das Punktediagramm mit gleitendem Durchschnitt über 500 Datenpunkte wurde mit Excel erstellt und basiert auf je circa 50'000 Messwerten. 29

Hilfsmittelverzeichnis

Hilfsmittel	Einsatzform	Betroffene Teile der Arbeit	Bemerkung
Gemini (Google)	Sammlung von Ideen für ein zugbasiertes Spiel	Produktteil der Maturaarbeit	
	<i>Debugging</i> von Problemen im Code	Produkt (Programmcode)	
	Textoptimierung, Feedback, Tipps zur Formulierung und Kürzung langer Sätze, Textüberarbeitung	Gesamte Arbeit	
	Lizenznennung von Abbildungen	Abbildungsverzeichnis	
	Generierung von Code für die Visualisierung der 3D-Arrays	Abbildung 3.2, 3.3	
	Verfassen des README: Verwendung von Markdown-Syntax und als Hilfe bei der Formulierung.	README.md	
ChatGPT (OpenAI)	Erstellung des Arbeitsplans	Arbeitsplan	
QuillBot	Grammatik- und Rechtschreib-überprüfung	Gesamte schriftliche Maturaarbeit	
Bitly	Linkgenerierung	Quelle [7] ⁵ im Literaturverzeichnis	Kürzerer Link, der auf die (für die Word-Quellenverwaltung) zu lange URL von EITCA führt.
Upscayl	Schärfen der Bilder	Abbildung 2.1, 2.2, 2.3	
PowerPoint	Zur Bearbeitung von Abbildungsschemas	Abbildung 2.1, 2.2, 2.3, 2.4, 3.1, 3.2, 3.3	
GeoGebra	Erstellen eines Funktionsgraphen	Abbildung 2.3	
Matplotlib (Python-Bibliothek)	Erstellen der Visualisierung des 3D-Arrays	Abbildung 3.2, 3.3	
Excel (Microsoft)	Datenauswertung und Erstellen der Diagramme	Alle Diagramme in Kapitel 5	
Astatsa [23]	Statistische Tests zu den Residuen der kumulativen Belohnung von den Modellen mit variierten Lernraten.	Unterkapitel 5.2: Modelle mit variierten Lernraten	Genutzt zur Durchführung der ANOVA und aller post-hoc Tests.

⁵ Originallink von [7]: <https://de.eitca.org/artificial-intelligence/eitc-ai-arl-advanced-reinforcement-learning/deep-reinforcement-learning/function-approximation-and-deep-reinforcement-learning/examination-review-function-approximation-and-deep-reinforcement-learning/what-is-the-bellman-equation-and-how-is-it-used-in-the-context-of-temporal-difference-td-learning-and-q-learning/>

Danksagung

Ich möchte mich herzlich bei all jenen bedanken, die mich beim Erarbeiten dieser Maturaarbeit unterstützt haben. Ohne sie wäre die Entstehung meiner Maturaarbeit nicht möglich gewesen.

Mein besonderer Dank gilt meinen Betreuern, Herrn Stefan Rothe und Frau Dardana Jaha. Für die wertvollen Ratschläge und die konstruktive Kritik, die diese Maturaarbeit massgeblich geformt haben und zu deren Gelingen beigetragen haben, bin ich sehr dankbar.

Ein ebenso herzlicher Dank gilt auch all meinen Freunden. Speziell bedanken möchte ich mich bei Marc für den hilfreichen Austausch während der Entstehung dieser Maturaarbeit.

Auch bei meiner Familie, insbesondere meinen Eltern, möchte ich mich für die moralische Unterstützung während der gesamten Zeit bedanken. Dieser Dank schliesst auch meine Haustiere - Finn und Imao - mit ein, die dafür sorgten, dass Streicheleinheiten und Spaziergänge nicht zu kurz kamen und so für den nötigen Ausgleich sorgten.

Anhang

Übersicht über die trainierten Modelle

Diese Tabelle zeigt eine Übersicht über alle trainierten Modelle und deren Hyperparameterwerte.

Tabelle A.1: Übersicht über alle trainierten Modelle. Zudem werden die Hyperparameterwerte der Modelle und die Trainingsumgebung, sprich welche Belohnungsfunktion fürs Training verwendet wurde, aufgelistet. Alle Modelle wurden 2'500'000 Zeitschritte trainiert.

Datei (.zip)	Bezeichnungen	Experiment	Belohnungs- funktion (Umgebung)	Lernrate	Diskontier- ungsfaktor	Explorations- phasenanteil
B1	(B1), (T2), Referenzmodell	1 und 2	B1	0,0001	0,99	0,1
B2	(B2)	1	B2	0,0001	0,99	0,1
B3	(B3)	1	B3	0,0001	0,99	0,1
T1a	(T1a)	2	B1	0,00005	0,99	0,1
T3a	(T3a)	2	B1	0,001	0,99	0,1
T1b	(T1b)	2	B1	0,0001	0,9	0,1
T3b	(T3b)	2	B1	0,0001	0,999	0,1
T1c	(T1c)	2	B1	0,0001	0,99	0,05
T3c	(T3c)	2	B1	0,0001	0,99	0,25

Anleitung und Zugriff

Der gesamte Code dieser Maturaarbeit steht allen Interessierten zur Verfügung. Abrufbar ist er auf einem GitHub Repository. Dazu kann entweder der Q-Code gescannt oder folgender Link aufgerufen werden:

<https://github.com/fsr07/Maturaarbeit>

