



Hochschule für angewandte Wissenschaften Coburg
Fakultät Elektrotechnik und Informatik

Studiengang: Informatik Bachelor

Bachelorarbeit

Entwicklung und Evaluierung einer Spiele-KI für „Ganz
schön clever“ mittels Deep Reinforcement Learning

Schubert, Sander

Abgabe der Arbeit: 22.11.2023

Betreut durch:

Florian Mittag, Hochschule Coburg

Zusammenfassung

Ziel dieser Arbeit war es, eine Künstliche Intelligenz, für das Brettspiel "*Ganz schön clever*", zu entwickeln und den Entwicklungs- sowie Trainingsprozess zu evaluieren. Diese Künstliche Intelligenz wurde erfolgreich entwickelt und kann das Brettspiel "*Ganz schön clever*" gut spielen. Zunächst wurde dafür ein Prototyp entwickelt, welcher dann schrittweise erweitert wurde, um die gesamte Komplexität des Spiels zu erfassen. Das Brettspiel "*Ganz schön clever*" ist ein Würfelspiel, welches eine hohe Komplexität aufweist. Diese kommt vor allem durch die vielen Aktionsmöglichkeiten des Spielers und die multiplen Zusammenhängen innerhalb des Belohnungssystems zustande. Außerdem weist es eine hohe Stochastizität auf, welche die Komplexität weiter erhöht. Für die Implementierung der Künstlichen Intelligenz wurde Deep Reinforcement Learning verwendet. Während der Implementierungsphase kam es zunächst zu einem Problem, bei dem die Künstliche Intelligenz es nur schwer selbstständig schaffte, gültige Aktionen zu wählen, welche nicht zum Abbruch des Spiels geführt haben. Dieses Vorgehen, bei dem das Spiel nach Wahl einer ungültigen Aktion abgebrochen wurde, wurde nach einigen Tagen verworfen. Daraufhin wurde zunächst ein Verfahren implementiert, bei dem das Spiel bei ungültigen Aktionen nicht sofort abgebrochen, sondern negativ belohnt wurde. Dieses Vorgehen stellte sich als gut geeignet heraus, wurde aber schnell von einem Vorgehen, welches eine Aktionsmaske verwendet abgelöst. Im Folgenden wurde eine Aktionsmaske implementiert, welche gewährleistet, dass nur gültige Aktionen gewählt werden können. Die Performanz stieg mit diesem Vorgehen von vorher durchschnittlich ungefähr 60 Prozent der Maximalpunktzahl auf 75 Prozent. Zudem ergaben sich Schwierigkeiten bei der Erweiterung des Runden-Systems und viele weitere Probleme, die behoben werden mussten. Das Ergebnis des finalen Trainings war eine Künstliche Intelligenz, welche durchschnittlich 208 Punkte mit einer Standardabweichung von 31 im Spiel erzielt. Im Vergleich dazu erzielten menschliche Spieler in einem Test im Durchschnitt lediglich 160 Punkte.

Abstract

The goal of this thesis was to develop an Artificial Intelligence for the board game "*Ganz schön clever*", and to evaluate the development and training process. This Artificial Intelligence was successfully developed and can play the board game "*Ganz schön clever*" well. Initially, a prototype was developed, which was then gradually expanded to capture the entire complexity of the game.

The board game "*Ganz schön clever*" is a dice game that has a high level of complexity. This is mainly due to the many action possibilities of the player and the multiple interconnections within the reward system. In addition, it exhibits a high level of stochasticity, which further increases the complexity.

Deep Reinforcement Learning was used for the implementation of the Artificial Intelligence. During the implementation phase, an initial problem occurred where the Artificial Intelligence struggled to autonomously select valid actions that did not lead to the termination of the game. This approach, in which the game was terminated after selecting an invalid action, was discarded after a few days. Subsequently, a method was implemented where the game was not immediately terminated upon invalid actions but instead negatively rewarded. This approach proved to be well-suited but was quickly replaced by a method using an action mask. An action mask was then implemented, ensuring that only valid actions could be chosen. The performance increased with this approach from previously averaging about 60 percent of the maximum score to 75 percent. In addition, there were difficulties in expanding the round system and many other problems that had to be resolved. The result of the final training was an Artificial Intelligence that achieves an average of 208 points with a standard deviation of 31 in the game. In comparison, human players scored an average of only 160 points in a test.

Inhaltsverzeichnis

Abbildungsverzeichnis	5
Codebeispielverzeichnis	6
Abkürzungsverzeichnis	7
1 Einleitung	8
1.1 Einführung zum Thema	8
1.2 Zielsetzung und Motivation	8
1.3 Aufgabenstellung	9
1.4 Aufbau der Arbeit	9
2 Grundlagen	10
2.1 Allgemeine Grundlagen	10
2.1.1 "Ganz schön clever"	10
2.1.2 Maschinelles Lernen	15
2.1.3 Reinforcement Learning	16
2.1.4 Deep Learning	19
2.1.5 Proximal Policy Optimization	20
2.2 Verwendete Technologien	21
2.2.1 Gymnasium	21
2.2.2 Stable Baselines 3	22
2.2.3 Matplotlib	22
2.2.4 ChatGPT 4	23
3 Anforderungen und Konzeption	24
3.1 Anforderungen	24
3.1.1 Rahmenbedingungen	24
3.1.2 Spiel	24
3.1.3 Künstliche Intelligenz	25
3.2 Konzeption	26
3.2.1 Spielumgebung	28
3.2.2 Künstliche Intelligenz	31
4 Implementierung	35
4.1 Einschränkungen	35
4.2 Spielumgebung	35
4.2.1 Klassenattribute	35

4.2.2	Schritt-Methode	39
4.2.3	Methode zum Zurücksetzen der Spielumgebung	41
4.2.4	Würfel-Methode	41
4.2.5	Methode zur Überprüfung der freigespielten Belohnungen	42
4.2.6	Methode zur Generierung der Aktionsmaske	42
4.2.7	Methode zum Hinzufügen von Boni	44
4.2.8	Methode zum Inkrementieren von Runden	44
4.3	Künstliche Intelligenz	45
4.3.1	Methode zum Anlernen des Modells	46
4.3.2	Methode zum Vorhersagen mithilfe des Modells	47
4.4	Anwendungsbeispiel für die Künstliche Intelligenz	47
5	Ergebnisse	48
5.1	Trainingshistorie	48
5.1.1	Prototyp	48
5.1.2	Training mit und ohne Aktionsmaske	51
5.1.3	Training mit zwei und vier Würfeln	53
5.1.4	Optimiertes Training	55
5.2	Erstes Training mit allen Feldern und Boni	56
5.3	Finale Ergebnisse	57
5.3.1	Performanz	57
5.3.2	Hyperparameter	60
5.3.3	ChatGPT 4	60
6	Zusammenfassung und Ausblick	61
	Literaturverzeichnis	64
	Glossar	65
	A1 Code für die Attribute der Spielumgebung	67
	A2 Pseudocode für Methoden der Implementierung	69
	Ehrenwörtliche Erklärung	72

Abbildungsverzeichnis

Abb. 1:	Screenshot des Spielfeldes von " <i>Ganz schön clever</i> "	10
Abb. 2:	Rundenablauf des Spiels " <i>Ganz schön clever</i> "	12
Abb. 3:	Lernzyklus im Reinforcement Learning	17
Abb. 4:	Zustandsverlauf des Spiels " <i>Ganz schön clever</i> " über zwei Runden	18
Abb. 5:	Multilayer Perceptron	20
Abb. 6:	Swimlane-Diagramm des Trainingsprozesses der KI	26
Abb. 7:	Ablauf-Diagramm der Schritt-Funktion 1	28
Abb. 8:	Ablauf-Diagramm der Schritt-Funktion 2	30
Abb. 9:	Wesentliche Hyperparameter der Künstlichen Intelligenz	31
Abb. 10:	Graph der ReLU-Funktion	33
Abb. 11:	Punktezahlen des ersten trainierten Modells für 27000 Episoden	49
Abb. 12:	Punktezahlen des ersten trainierten Modells für 45 Episoden	50
Abb. 13:	Punktezahlen des Modells ohne Aktionsmaske für 20 Episoden	51
Abb. 14:	Punktezahlen des Modells mit Aktionsmaske für 20 Episoden	52
Abb. 15:	Ungültige Züge des Modells unter Verwendung von zwei Würfeln für 20 Episoden	53
Abb. 16:	Ungültige Züge des Modells unter Verwendung von vier Würfeln für 20 Episoden	54
Abb. 17:	Punktezahlen des Modells nach optimiertem Training für 20 Episoden	55
Abb. 18:	Punktezahlen des ersten trainierten Modells mit allen Feldern	56
Abb. 19:	Punktezahlen des finalen Modells für 1000 Episoden	57
Abb. 20:	Ungültige Züge des finalen Modells für 1000 Episoden	58
Abb. 21:	Punktezahlen des finalen Modells für 100 Episoden	59
Abb. 22:	Ungültige Züge des finalen Modells für 100 Episoden	59

Codebeispielverzeichnis

Code 1:	Klassenattribute für das Runden-System	35
Code 2:	Klassenattribute für Würfel	36
Code 3:	Klassenattribute für die farbigen Felder des Spiels	36
Code 4:	Klassenattribute für freizuschaltende Boni	37
Code 5:	Klassenattribute für freizuschaltende Punktebelohnungen des gelben, blauen und grünen Feldes	37
Code 6:	Klassenattribute für freigespielte Boni	38
Code 7:	Klassenattribute des Aktions- und Beobachtungsraumes	38
Code 8:	Schritt-Methode	39
Code 9:	Methode zum Zurücksetzen der Umgebung	41
Code 10:	Methode zum Werfen der Würfel	41
Code 11:	Methode zur Überprüfung der freigespielten Belohnungen	42
Code 12:	Methode zur Generierung der Aktionsmaske	42
Code 13:	Methode zum Hinzufügen freigeschalteter Boni	44
Code 14:	Methode zum Inkrementieren des Runden-Systems	44
Code 15:	Methode zu Anlernen des Modells	46
Code 16:	Methode zum Vorhersagen von Aktionen mithilfe des Modells	47
Code 17:	Anwendungsbeispiel für die Künstliche Intelligenz	47
Code 18:	Klassenattribute für die Nachvollziehbarkeit von Punkteständen	67
Code 19:	Klassenattribute für Belohnungsflags	67
Code 20:	Klassenattribute für erreichte Punktestände der einzelnen farbigen Felder . .	67
Code 21:	Klassenattribute für die Anzahl an gewählte Kästchen innerhalb der farbigen Feldern	68
Code 22:	Methode zur Visualisierung der Spielumgebung	69
Code 23:	Methode zur Generierung des Beobachtungsraumes	69
Code 24:	Methode zur Initialisierung der Spielumgebungen	70
Code 25:	Methode zum Anwenden der Aktionsmaske	70
Code 26:	Methoden zum Erstellen von Einträgen für Historien	71
Code 27:	Methode zum Plotten von Historien	71

Abkürzungsverzeichnis

API	Application Programming Interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
KI	Künstliche Intelligenz
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
RL	Reinforcement Learning

1 Einleitung

1.1 Hinführung zum Thema

In den vergangenen Jahren gewann Künstliche Intelligenz und insbesondere maschinelles Lernen mit steigender Tendenz an Bedeutung [noac, noa23b]. Eines der präsentesten neuen Tools im Jahr 2023 ist ChatGPT 4. Dieses Tool ist ein Chat-Bot, welcher es dem Benutzer ermöglicht mit ihm zu kommunizieren und ihm Fragen oder Aufgaben zu stellen. Solche Tools ermöglichen es Benutzern zunehmend ihre Tätigkeiten zu vereinfachen und prägen somit das Leben der Menschen zunehmend [tag]. In dieser Arbeit wurde ChatGPT 4 ebenfalls als unterstützendes Tool verwendet. Vor allem wurde ChatGPT 4 zur Beantwortung fachlicher Fragen und anfänglich zur Generierung von Code für den Prototypen genutzt. Mit zunehmender Komplexität der zu bearbeitenden Aufgabe sinkt die Verlässlichkeit solcher Tools. Daher ist es wichtig, die Anfragen an den Chat-Bot möglichst präzise zu formulieren und den Aufgabenbereich angemessen einzuschränken, um das Tool nicht zu überfordern.

Auch in der Spielentwicklung nehmen Künstliche Intelligenz und maschinelles Lernen schon seit langem eine zentrale Rolle ein [noa23a]. In vielen Spielen gibt es eine oder mehrere Künstliche Intelligenzen, welche bestimmte Aufgaben erfüllen, um den Spieler bei Spiel zu unterstützen oder ihm als Widersacher zu dienen. Auch hier gilt je komplexer die Aufgabenstellung desto schwieriger ist es, einen solchen Bot zu generieren, welcher diese gut und richtig lösen kann.

Das Gemeinschaftsspiel "*Ganz schön clever*" ist ein Würfelspiel, das eine hohe Komplexität aufweist. Diese kommt vor allem durch die hohe Anzahl an möglichen Aktionen (dem sogenannte Aktionsraum) für den Spieler und die vielen Zusammenhänge des Belohnungssystems im Spiel zustande. Das Spiel weist zusätzlich eine hohe Stochastizität auf, welche die Komplexität weiter erhöht.

Interessant ist, wie man für ein solches komplexes Spiel einen Bot oder eine Künstliche Intelligenz entwickeln kann, um dieses gut spielen zu können. Dabei könnte die Komplexität der Aufgabenstellung möglicherweise zu groß sein, um vom Bot erfasst zu werden und es können sich zahlreiche weitere Problemstellungen ergeben, für die Lösungen gefunden werden müssen.

1.2 Zielsetzung und Motivation

Ziel der Arbeit ist es, einen Bot beziehungsweise eine Künstliche Intelligenz zu entwickeln, welche das Spiel "*Ganz schön clever*" möglichst gut spielen kann. Gut heißt hierbei, dass im Durchschnitt eine möglichst hohe Punktezahl im Spiel erzielen wird. Es soll analysiert werden, welche Aspekte es bei der Entwicklung zu beachten gilt und wie sich unterschiedliche Ansätze auf das Verhalten und die Performanz der Künstlichen Intelligenz auswirken.

Es gibt noch keine Untersuchungen zu einer Spiel-KI für das Spiel "*Ganz schön clever*", daher ist es interessant Erkenntnisse darüber zu gewinnen, welche Schwierigkeiten sich hierbei ergeben und wie diese überwunden werden können.

1.3 Aufgabenstellung

Es ist eine Künstliche Intelligenz für das Spiel "*Ganz schön clever*" zu implementieren. Hierbei sollen der Vorgang sowie die Ergebnisse des Prozesses analysiert und bewertet werden. Hierzu wird zunächst ein Prototyp entwickelt, welcher eines der fünf Felder des Spiels beinhaltet. Dieser Prototyp soll Einsichten über die Machbarkeit und die Rahmenbedingungen des Projektes bringen. Daraufhin werden die Künstliche Intelligenz und die Spielumgebung schrittweise um ihre jeweiligen Funktionalitäten erweitert, bis das Spiel vollständig und möglichst gut von der Künstlichen Intelligenz gespielt werden kann.

1.4 Aufbau der Arbeit

Kapitel 2 beinhaltet einen Grundlagenteil, in dem zunächst das Spiel "*Ganz schön clever*" und seine Mechaniken erklärt werden. Anschließend werden maschinelles Lernen, Reinforcement Learning, Deep Learning und Proximal Policy Optimization behandelt, um eine fachliche Grundlage für das Verständnis der Abläufe beim Training der Künstlichen Intelligenz zu bieten. Im Folgenden werden die verwendeten Technologien des Projektes behandelt. Die vorgestellten Technologien sind Gynmansion, Stable Baselines 3, Matplotlib und ChatGPT 4.

Kapitel 3 befasst sich mit den Anforderungen und der Konzeption des Projektes. Es werden Rahmenbedingungen des Projektes sowie die Anforderungen und die Konzeption für die Spielumgebung und die Künstliche Intelligenz behandelt.

Kapitel 4 zeigt und erklärt die Implementierung des Projektes. Dabei werden die Variablen und Methoden des Projektes erläutert und beschrieben. Zu einem großen Teil wird Pseudocode verwendet, um die Implementierung verständlicher zu machen und den Umfang zu begrenzen.

Kapitel 5 befasst sich mit den Ergebnissen des Projektes, wobei zunächst der Verlauf der Implementierung und dessen einzelne Meilensteine behandelt werden. Ein Prototyp wurde erstellt und anschließend schrittweise erweitert. Daraufhin folgen die Analyse und Darstellung des finalen Modells.

Das abschließende Kapitel 6 beinhaltet eine Zusammenfassung der Arbeit und Anreize zur Weiterarbeit am Projekt.

2 Grundlagen

2.1 Allgemeine Grundlagen

2.1.1 "Ganz schön clever"

Abbildung 1 zeigt das Spielbrett des Spiels "Ganz schön clever" zu dem im Rahmen dieser Arbeit eine Künstliche Intelligenz implementiert werden soll:



Abb. 1: Screenshot des Spielfeldes von "Ganz schön clever"

Quelle: Screenshot des Spiels "Ganz schön clever" von Brettspielwelt

"Ganz schön clever" ist ein Brettspiel, welches von Wolfgang Warsch entworfen und 2018 von Schmidt Spiele veröffentlicht wurde.

Im Folgenden werden der Spielablauf und die wesentlichen Regeln des Spiels erklärt.

Es gibt sechs farbige Würfel (gelb, blau, grün, orange, lila und weiß), wobei jeder bis auf den weißen einem der fünf farbigen Spielfelder zuzuordnen ist. Der weiße Würfel ist ein Sonderwürfel und kann als einer der anderen Würfel betrachtet werden. Wenn bestimmte Bedingungen erfüllt sind, kann der Spieler einen Würfel wählen und ein entsprechendes Kästchen ausfüllen. Die Felder verfügen über Belohnungen, welche freigeschaltet werden, wenn eine bestimmte Kombination oder Anzahl an Feldern freigeschaltet worden ist. Beim orangenen und lila Feld kommt es bei der Belohnung zudem darauf an, wie hoch das Würfelergebnis des gewählten Würfels ist, da die einzelnen Kästchen hier je nach Würfelergebnis zusätzlich belohnt

werden.

Das Spiel teilt sich in bis zu sechs Runden mit jeweils bis zu drei Würfeln pro Spieler ein. Nach jeder Runde eines Spielers bekommt dieser zudem die Möglichkeit, einen Würfel auf dem Silbertablett eines Mitspielers zu wählen. Diese Wahl verhält sich, wie die Wahl eines eigenen Würfels, nach Würfeln des Spielers. Die Anzahl der Runden wird durch die Spielerzahl bestimmt. Bei ein bis zwei Spielern sind es sechs Runden. Bei drei Spielern sind es fünf Runden und bei vier Spielern sind es vier Runden. Die Anzahl der Würfe pro Runde und Spieler ist immer drei, allerdings kann diese Anzahl reduziert werden, wenn kein Würfel mehr zum Würfeln zur Verfügung steht. Ist dies der Fall wird der Spielablauf fortgesetzt als hätte der Spieler seinen dritten Wurf in der Runde beendet und er kann einen Würfel vom Silbertablett des Mitspielers wählen bevor dann die neue Runde für ihn beginnt. Zu Beginn der ersten, zweiten, dritten und vierten Runde bekommt jeder Spieler zudem eine festgeschriebene Belohnung, welche oben rechts auf Abbildung 1 bei der jeweiligen Rundenzahl zu sehen ist. Die Belohnung in Runde vier steht dabei für das freie Ausfüllen eines beliebigen Kästchens mit dem jeweils maximal möglichen Wert für das Kästchen.

Abbildung 2 zeigt das Runden-System des Spiels. Dieses wird noch einige Male wichtig für das Verständnis der Arbeit werden:

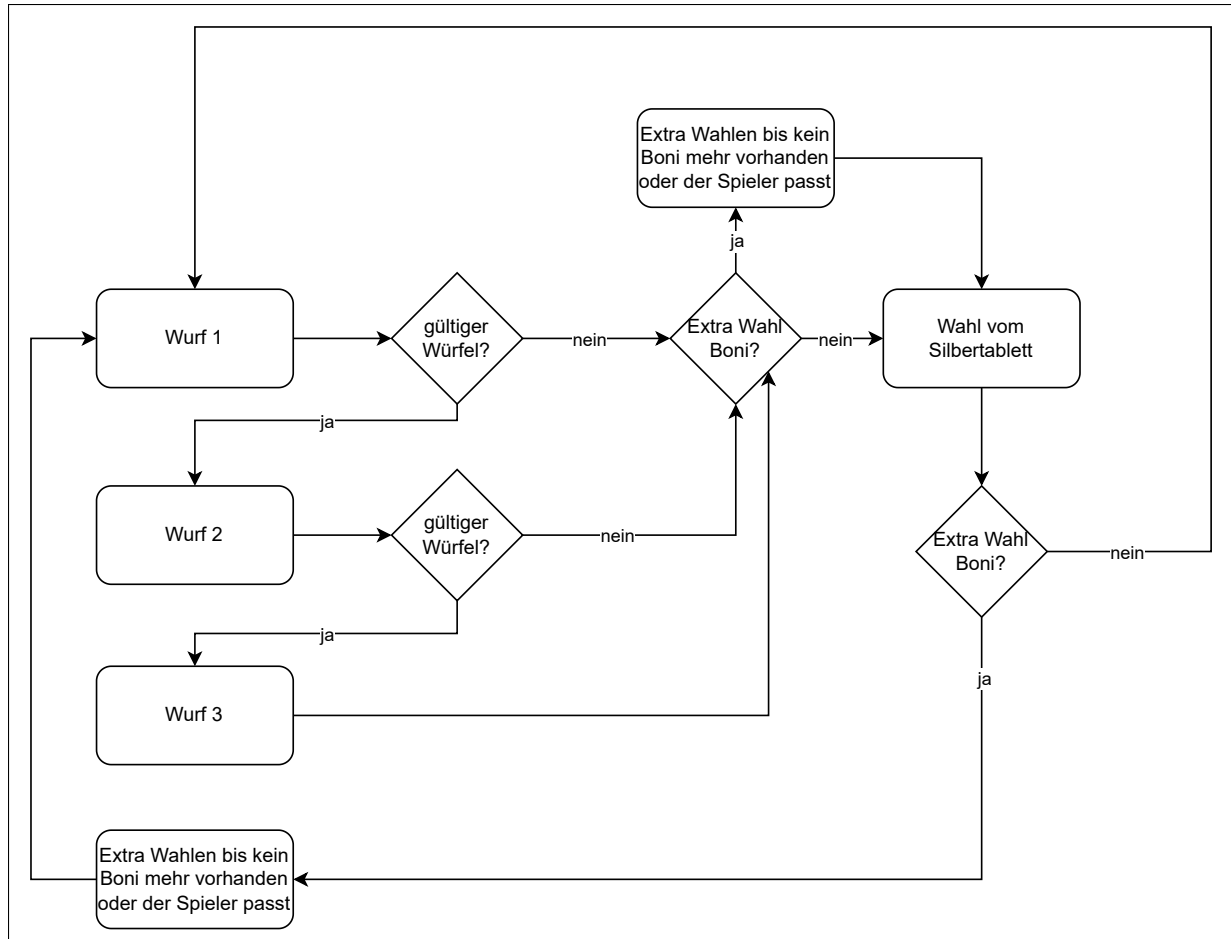


Abb. 2: Rundenablauf des Spiels "Ganz schön clever"

Quelle: Eigene Darstellung

Wie in Abbildung 2 dargestellt, beginnt der Zyklus bei Wurf 1. Wenn danach noch gültige Würfel vorhanden sind geht es weiter mit Wurf 2. Sind dann noch gültige Würfel vorhanden folgt Wurf 3. Sind in der Zwischenzeit keine gültigen Würfel vorhanden oder spätestens nach Wurf 3 hat der Spieler die Möglichkeit seine Extra-Wahl-Boni einzusetzen und einen der aktuellen Würfel zu wählen. Entscheidet sich der Spieler dafür keine Extra-Wahl-Boni einzusetzen oder besitzt er keine, geht es weiter mit der Wahl vom Silbertablett. Dort wählt der Spieler einen Würfel und hat anschließend erneut die Möglichkeit Extra-Wahl-Boni zu verwenden, falls er mindestens eine davon besitzt. Entscheidet er sich dafür die Extra-Wahl-Boni nicht zu benutzen, oder besitzt er keine, beginnt der Zyklus erneut von Wurf 1. Dieser Zyklus wird so lange wiederholt, bis das Spiel endet.

Es gibt zwei Arten von Belohnungen im Spiel, Punktebelohnungen und Boni. Bei Punktebelohnungen handelt es sich um Punkte, welche auf den Punktestand des Spielers addiert werden,

welcher am Ende des Spiels entscheidet, wer gewonnen hat. Der Spieler mit dem höchsten Punktestand gewinnt das Spiel. Punktebelohnungen erhalten Spieler beim gelben Feld, indem sie eine Spalte vollständig ausfüllen. Die Anzahl der Punkte findet sich am Ende der jeweiligen Spalte. Im blauen Feld erhalten Spieler Punktebelohnungen mit zunehmender Anzahl an ausgefüllten blauen Kästchen auf ihrem eigenen Spielbrett. Die Anzahl der Punkte findet sich im oberen Bereich des blauen Feldes, im grünen Feld erhalten Spieler ebenso mit zunehmender Anzahl der ausgefüllten grünen Kästchen Punktebelohnungen. Die Anzahl der Punkte findet sich auch hier im oberen Bereich des grünen Feldes. Im orangenen und lilafarbenem Feld muss für Punktebelohnungen lediglich ein Kästchen ausgefüllt werden. Die Punktebelohnung entspricht der Augenzahl des Würfels, welcher zum Ausfüllen des Kästchens verwendet wurde. Beim orangenen Feld wird diese Augenzahl beim vierten, siebten und neunten Kästchen mit zwei und beim elften Kästchen mit drei multipliziert. Außerdem gibt es einen Bonus, welche indirekt eine Punktebelohnung darstellt. Es handelt sich hierbei um den sogenannten Fuchs beziehungsweise die Füchse. Die Anzahl der freigeschalteten Füchse wird am Ende des Spiels mit der Anzahl der erzielten Punkte des Feldes mit dem niedrigsten erreichten Punktwert multipliziert und auf den Gesamtpunktestand des jeweiligen Spielers addiert.

Bei Boni handelt es sich um Belohnungen, welche der Spieler nutzen kann oder muss, um sich im Spiel einen Vorteil zu verschaffen. Die Boni sind unterhalb der entsprechenden Kästchen beziehungsweise am Rande von Spalten und Zeilen eingezeichnet und können freigeschaltet werden, indem man diese (Kästchen beziehungsweise Zeilen oder Spalten) ausfüllt. Eine Ausnahme bildet hier der Bonus, welcher beim gelben Feld freigeschaltet wird, indem alle diagonalen Felder von links oben nach rechts unten ausgefüllt werden.

Boni werden bei der Benutzung aufbraucht. Man kann mehr als eine dieser Boni auf einmal besitzen. Die Boni sind stapelbar. Jede Boni hat ihr eigenes Symbol. Nun folgt eine Aufzählung und Erklärung der verschiedenen Boni mit Ausnahme der Füchse:

- **Extra-Wahl:** Ermöglicht dem Spieler, am Ende seiner eigenen Runde beziehungsweise nachdem er einen Würfel vom Silbertablett des Gegners wählte, erneut Würfel zu wählen und die entsprechenden Kästchen dafür anzukreuzen. Würfel die so gewählt wurden können mithilfe des Extra-Wahl-Bonus nicht erneut gewählt werden, solange keine neue Runde oder Wahl vom Silbertablett begonnen hat. Mit dem Extra-Wahl-Bonus können alle Würfel gewählt werden, nicht nur die, welche unter normalen Umständen gültig zur Wahl stehen würden. Das Symbol ist die +1.
- **Neu-Würfeln:** Ermöglicht es dem Spieler, einen seiner Würfe zu wiederholen, ohne dabei einen der Würfel auszuwählen. Dies ermöglicht es ihm Würfe mit ungünstigen Ergebnissen neu auszurichten. Das Symbol sind die drei Pfeile, welche im Kreis angeordnet sind.

- Gelbes-Kreuz: Ermöglicht es dem Spieler direkt nach Erhalt des Bonus eines der gelben Kästchen nach eigener Wahl auszufüllen. Das Symbol ist ein Kreuz auf gelbem Hintergrund.
- Blaues-Kreuz: Ermöglicht es dem Spieler direkt nach Erhalt des Bonus eines der blauen Kästchen nach eigener Wahl auszufüllen. Das Symbol ist ein Kreuz auf blauem Hintergrund.
- Grünes-Kreuz: Ermöglicht es dem Spieler direkt nach Erhalt des Bonus das nächste freie Grüne Kästchen auszufüllen. Das Symbol ist ein Kreuz auf grünem Hintergrund.
- Orangene-Vier: Ermöglicht es dem Spieler direkt nach Erhalt des Bonus das nächste freie orangene Kästchen mit einer vier auszufüllen. Das Symbol ist eine vier auf orangenem Hintergrund.
- Orangene-Fünf: Ermöglicht es dem Spieler direkt nach Erhalt des Bonus das nächste freie orangene Kästchen mit einer fünf auszufüllen. Das Symbol ist eine fünf auf orangenem Hintergrund.
- Orangene-Sechs: Ermöglicht es dem Spieler direkt nach Erhalt des Bonus das nächste freie orangene Kästchen mit einer sechs auszufüllen. Das Symbol ist eine sechs auf orangenem Hintergrund.
- Lila-Sechs: Ermöglicht es dem Spieler direkt nach erhalten des Bonus das nächste freie lilafarbene Kästchen mit einer sechs auszufüllen. Das Symbol ist eine sechs auf lilafarbenem Hintergrund.

Nun folgen die Regeln nach denen bestimmt wird, ob ein Würfel gewählt werden kann, um ein Kästchen auszufüllen und um zu bestimmen, ob er ein gültiger Würfel ist:

Ist ein Würfel ungültig, kann dieser nicht gewählt werden. Würfel werden ungültig, indem sie in der aktuellen Runde oder bei einer Wahl mit einem Extra-Wahl-Bonus bereits gewählt worden sind. Außerdem werden Würfel, welche eine geringere Augenzahl aufweisen als der aktuell gewählte Würfel, automatisch ungültig. Ausnahmen bilden hierbei die Wahlen eines Würfels vom Silbertablett des Gegenspielers oder Wahlen mithilfe eines Extra-Wahl-Bonus. Würfel werden wieder gültig: Am Anfang jeder neuen Runde, nach Abschluss des dritten Wurfes, nach der Wahl eines Würfels vom Silbertablett sowie nachdem Wahlen mithilfe der Extra-Wahl-Boni. Dabei bleiben mithilfe eines Extra-Wahl-Bonus gewählte Würfel solange ungültig, bis keine weiteren Extra-Wahl-Boni in diesem Spielschritt mehr verwendet werden und der Spielverlauf weitergeht.

Jedes Feld hat seine eigenen Regeln, die bestimmen, wann ein Kästchen ausgefüllt werden darf. Beim gelben Feld muss die Augenzahl des Würfels mit der Zahl des Kästchens übereinstimmen. Beim blauen Feld muss die Summe der Augenzahlen des blauen und des weißen Würfels mit der Zahl innerhalb des Kästchens übereinstimmen. Beim grünen Feld muss die Augenzahl des Würfels größer oder gleich der Zahl im Kästchen sein. Zudem kann beim grünen Feld immer nur das nächste freie Kästchen ausgefüllt werden, beginnend von links. Im orangenen Feld kann immer das nächste Kästchen eingetragen werden. Auch hier beginnend von links. Beim lila Feld muss die Augenzahl des Würfels größer sein als die Zahl im zuletzt ausgefüllten Kästchen. Eine Ausnahme bildet hier der Fall in dem eine sechs im zuletzt ausgefüllten Kästchen steht. Ist dies der Fall, kann das nächste Feld mit jeder beliebigen Augenzahl ausgefüllt werden. Die sechs setzt die Voraussetzung für das lila Feld bis zum nächsten Ausfüllen im lila Feld aus. Auch hier gilt die Reihenfolge von links nach rechts.

2.1.2 Maschinelles Lernen

"Maschinelles Lernen heißt, Computer so zu programmieren, dass ein bestimmtes Leistungsmerkmal anhand von Beispieldaten oder Erfahrungswerten optimiert wird" [Alp22, S. 3].

Es gibt bis heute nach wie vor viele Problemstellungen, die von Menschen auf einfache Art und Weise lösbar sind, für die es aber keine algorithmische Lösung zu geben scheint [Alp22, S. 1]. Hier kommt das maschinelle Lernen zum Einsatz. Durch die Mustererkennung aus Trainingsdaten können Programme lernen, solche Problemstellungen zu lösen, indem sie präzise Vorhersagen über bestehende Sachverhalte beziehungsweise Muster aus beliebigen Daten desselben oder eines ähnlichen Sachverhalts, der beim Generieren der Trainingsdaten zugrunde lag, treffen. Ein besonders weit verbreiteter Anwendungsfall ist die Herleitung von Kundenverhalten und daraus resultierender Optimierungsmöglichkeiten für den Verkauf [Alp22, S. 1f].

Wenn man ein Programm verwendet, um eine Struktur mithilfe von maschinellem Lernen zu trainieren, damit diese Vorhersagen über ähnliche Sachverhalte treffen kann, nennt man diese Modell. Ein solches Modell wird häufig erst auf allgemeinen Datensätzen und später auf immer spezifischeren Daten trainiert, sodass es schließlich auf eine konkrete Aufgabe zugeschnitten werden kann [Alp22, S. 1f].

Maschinelles Lernen ermöglicht es zwar nicht, einen gesamten Prozess mit all seinen Einzelheiten zu verstehen, aber es ermöglicht, relevante Merkmale zu erkennen und mithilfe dieser Merkmale und deren Zusammenhängen Prognosen über einen gesamten Sachverhalt herzuleiten. Auf Grund dieser Prognosen kann schließlich agiert werden. [Alp22, S. 2].

Die Anwendungsgebiete von maschinellem Lernen sind zahlreich. Unter anderem ist es relevant für den Einzelhandel und Finanzdienstleister, um Kreditgeschäfte abzuwickeln, Betrugsversuche zu erkennen, oder den Aktienmarkt einzuschätzen. Aber auch in der Fertigung wird es zur

Optimierung, Steuerung und Fehlerbehebung eingesetzt. Auch in der Medizin erweisen sich medizinische Diagnoseprogramme mithilfe von Modellen, die mit maschinellem Lernen trainiert wurden als nützlich [Alp22, S. 3].

Die Datenbestände und das World Wide Web werden immer größer, weshalb die Suche nach relevanten Daten nicht mehr manuell vorgenommen werden kann. "Das maschinelle Lernen ist aber nicht nur für Datenbanken relevant, sondern auch für das Gebiet der künstlichen Intelligenz" [Alp22, S. 3]. Von Intelligenz spricht man dann, wenn das System selbstständig in einer sich verändernden Umgebung lernen und sich anpassen kann. Dadurch muss der Systementwickler nicht jede erdenkliche Situation vorhersehen und passende Lösungen dafür entwickeln [Alp22, S. 3].

Maschinelles Lernen findet seine Anwendung in dieser Arbeit in Form von Deep Reinforcement Learning. Was Reinforcement Learning ist und wie es sich von Deep Reinforcement Learning unterscheidet, wird im Folgenden beschrieben.

2.1.3 Reinforcement Learning

Reinforcement Learning (im Deutschen: bestärkendes Lernen) heißt so, weil es die Aktionen des Agenten (beziehungsweise des Modells) bestärkt. Dies verhält sich ähnlich, wie das Training eines Hundes im Park. Dieser wird jedes Mal, wenn er einen Trick richtig ausführt, mit einem Leckerli belohnt. Diese Belohnung bestärkt das Verhalten des Hundes und das Tier lernt dieses in Zukunft zu wiederholen. Eine negative Aktion kann hingegen bestraft werden, damit sie in Zukunft nicht wiederholt wird [RA23, S. 11].

Im Reinforcement Learning sind vor allem folgende Begriffe wichtig:

- **Agent:** Dabei handelt es sich um die Entität, welche mit der Umgebung interagiert und die Entscheidungen trifft. Dabei kann es sich zum Beispiel um einen Roboter oder autonomes Fahrzeug handeln [RL, S. 11].
- **Umgebung:** Dabei handelt es sich um die Außenwelt des Agenten [RA23, S. 11]. Der Agent interagiert mit dieser Umgebung und erhält je nach Zustand dieser Umgebung und seiner gewählten Aktion ein entsprechendes Feedback.
- **Aktion:** Eine Aktion beschreibt das Verhalten des Agenten [RA23, S. 11]. In dieser Arbeit wählt der Agent Kästchen des Spielbrettes aus, um diese auszufüllen. Außerdem entscheidet er, ob er bestimmte Boni zu einem gegebenen Zeitpunkt nutzen möchte oder nicht.

- Zustand: Der Zustand beschreibt den Zusammenhang zwischen Umgebung und Agent [RA23, S. 11]. In dieser Arbeit ist der Zustand von den Eigenschaften des Spielbrettes, der Würfel, der Rundenanzahl und der erspielten Boni abhängig.
- Belohnung: Der Agent erhält positive oder negative Belohnungen, je nachdem wie gut der Zustandswechsel von Zustand x nach Zustand y gewesen ist [RA23, S. 11]. In dieser Arbeit wird dies durch die jeweiligen Punktebelohnungen im Spiel verkörpert. Eine Ausnahme bildet hier eine negative Belohnung, wenn der Agent in einen Zustand gerät in dem er keine gültige Aktion tätigen kann.
- Policy: Die Policy ist die Strategie des Agenten, nach welcher er seine nächste Aktion wählt [RA23, S. 11]. In dieser Arbeit wird die Policy durch ein Multilayer Perceptron (s. Unterabschnitt 2.1.4) abgebildet.
- Episode: Eine Menge an zusammenhängenden Aktionen innerhalb einer Umgebung, welche endet, wenn das Ziel erreicht worden ist [RA23, S. 11]. In dieser Arbeit ist eine Episode ein kompletter Spieldurchlauf von *"Ganz schön clever"*.

Abbildung 3 zeigt einen Lernzyklus im Reinforcement Learning:

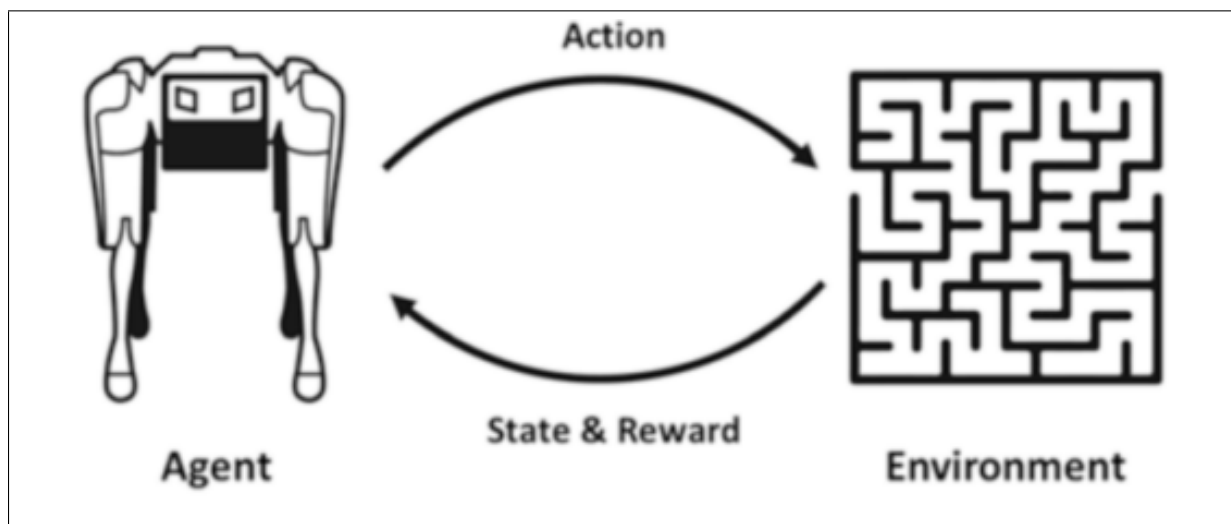


Abb. 3: Lernzyklus im Reinforcement Learning
Quelle: [RA23]

Wie in Abbildung 3 dargestellt führt der Agent eine Aktion in der Umgebung aus und erhält daraufhin eine Belohnung und den neuen Zustand der Umgebung als Feedback. Daraufhin aktualisiert er seine Policy, um in Zukunft bessere Aktionen tätigen zu können.

Hierbei ist das Ziel des Agenten die Gesamtbelohnung zu maximieren. Die Policy wird demnach so angepasst, dass sie die Maximierung der Gesamtbelohnung begünstigt. [RA23, S. 12f].

Hierbei gibt es einige Schwierigkeiten, die es zu beachten gilt. Belohnungen die in kurzer Zeit erreicht werden können, könnten wichtiger oder weniger wichtig sein als Belohnungen, die erst innerhalb vieler Schritte erreicht werden können. Ein gutes Beispiel hierfür wäre ein Balanceakt, bei dem es besonders wichtig ist kurzzeitige Belohnungen zu priorisieren, da die Episode endet, wenn das Balancieren fehlschlägt, was eine starke negative Belohnung zur Folge haben kann.

Ein weiterer Faktor ist die Balance zwischen Erkundung (englisch: exploration) und Ausbeutung (englisch: exploitation). Diese zwei Prinzipien sind wesentlich für das Reinforcement Learning. Dabei geht es darum, wie stark die Policy es vorzieht neue oder selten gesehene Zustände und Aktionen auszuprobieren (exploration) oder bereits bekannte, welche eine gute Belohnung zu bringen scheinen, auszubeuten (exploitation) [RA23, S. 13].

Im Grunde lernt das die Künstliche Intelligenz, in bestimmten Zuständen bestimmte Aktionen zu tätigen, um das beste Gesamtergebnis zu erreichen. Jeder Zustand hat seinen erwarteten Wert beziehungsweise seine erwartete Belohnung, sobald das Modell in einen bestimmten Zustand gerät, wählt es die Aktion, welche die höchste erwartete Belohnung bietet, beziehungsweise weist den möglichen Aktionen Wahrscheinlichkeiten zu, die bei bevorzugten Aktionen höher ausfallen.

Abbildung 4 zeigt einen Ausschnitt eines theoretischen Zustandsverlaufs von zwei Würfeln des Spiels "*Ganz schön clever*":

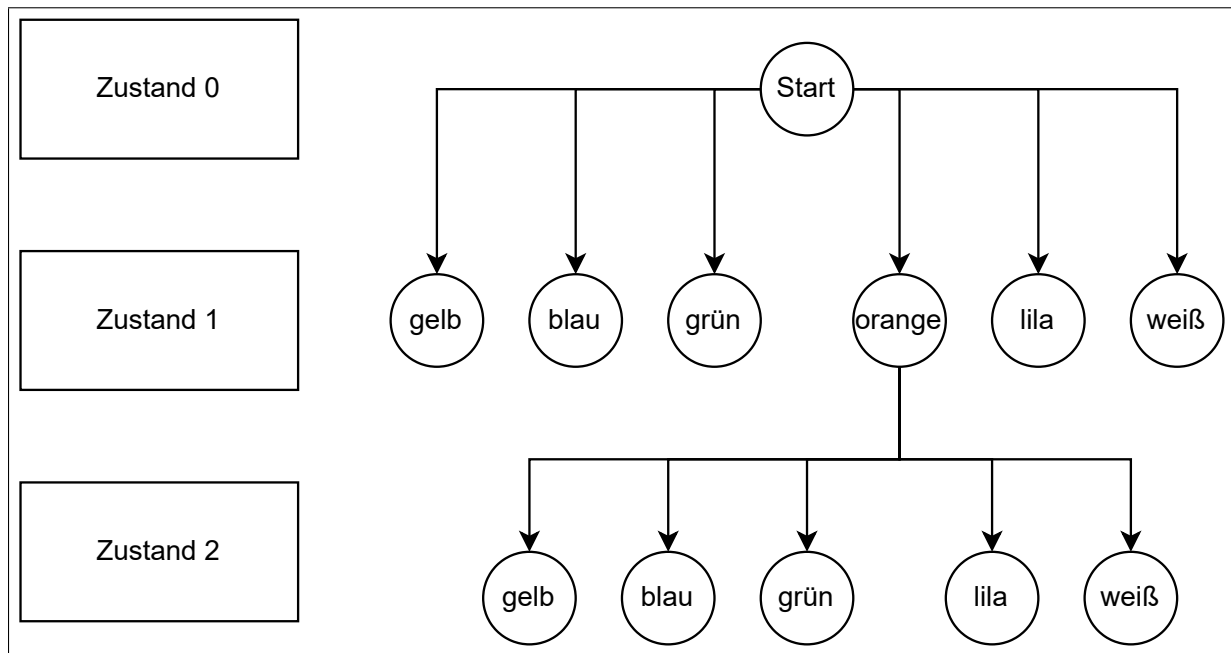


Abb. 4: Zustandsverlauf des Spiels "*Ganz schön clever*" über zwei Runden

Quelle: Eigene Darstellung

Abbildung 4 zeigt wie das Spiel beginnt. Die Würfel werden geworfen und der Agent wählt einen der sechs farbigen Würfel und füllt das entsprechende Kästchen aus. Somit gerät er in einen neuen Zustand, bei dem ein bestimmtes Kästchen ausgefüllt und ein oder mehrere Würfel ungültig sind. Die gültigen Würfel werden erneut geworfen und er wählt erneut einen der gültigen Würfel und füllt das entsprechende Kästchen aus. Es ist schnell zu erkennen, dass sich eine große Zahl an möglichen Zustands-Aktions-Paaren ergibt. Ein Zustands-Aktions-Paar beschreibt genau eine mögliche Aktion in einem spezifischen Zustand des Spiels. Die Künstliche Intelligenz lernt durch die Auswertung eben dieser Zustands-Aktions-Paare, welche Aktionen am vorteilhaftesten sind und welche sie vermeiden sollte. Die Abbildung ist lediglich eine Vereinfachung, da viele weitere Faktoren, wie erspielte Boni, eine Rolle spielen.

2.1.4 Deep Learning

Deep Learning heißt, man verwendet zum Lernen ein neuronales Netz mit mehreren versteckten Schichten.

Ein solches Netz besteht aus einer Vielzahl von Neuronen [Sew19, S. 75]. Ein solches Neuron setzt sich zusammen aus Inputs, Outputs, Gewichtungen dieser In- und Outputs, sowie einer Aktivierungsfunktion. Ein Spezialfall eines solchen neuronalen Netzes ist ein sogenanntes Multilayer Perceptron. Bei diesem sind alle Neuronen in einer Schicht mit allen Neuronen der folgenden Schicht verbunden. Häufig haben auch alle Neuronen der versteckten Schichten dieselbe Aktivierungsfunktion. Was den Beitrag der einzelnen Neuronen zum Gesamtergebnis des Netzes steuert, sind im Wesentlichen seine Gewichtungen und die Position im Netz.

Abbildung 5 zeigt ein Multilayer Perceptron. Hierbei sind jeweils alle Knoten (Inputs und Neuronen) einer Schicht mit allen Knoten der folgenden Schicht verbunden:

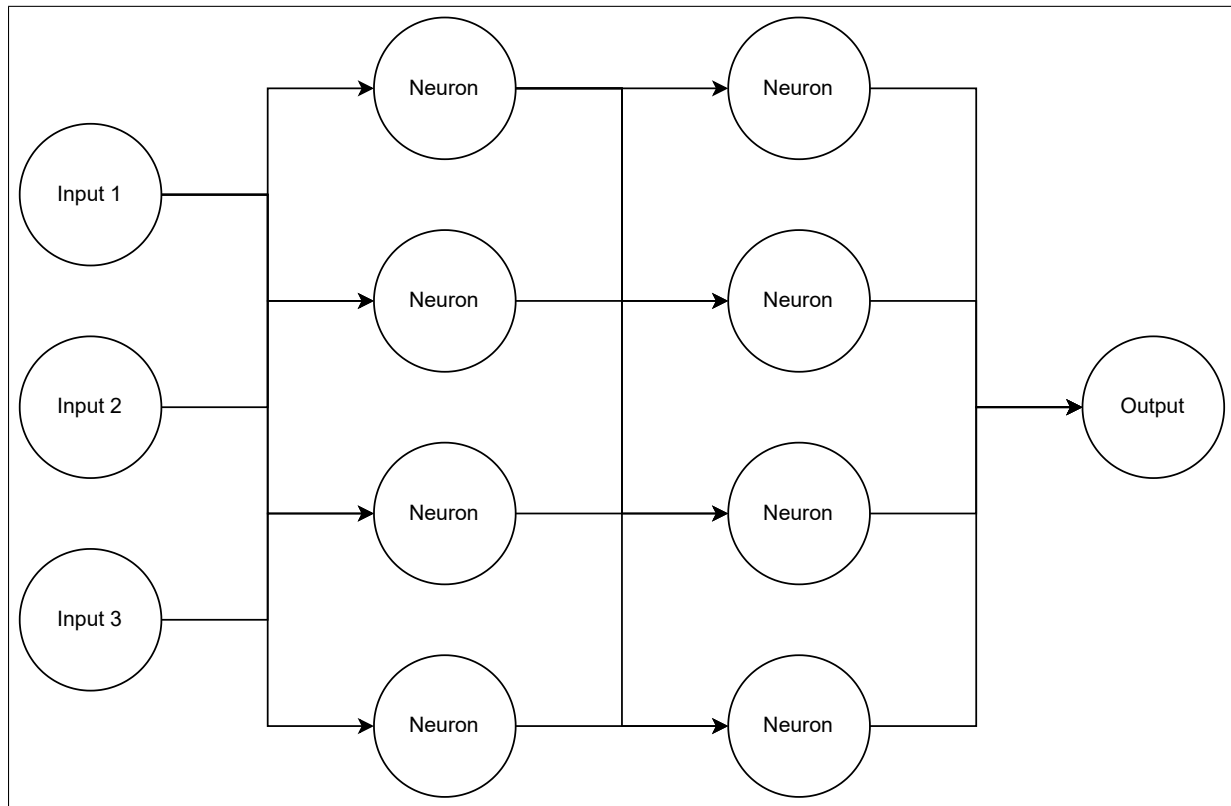


Abb. 5: Multilayer Perceptron
Quelle: Eigene Darstellung

Die Inputs des Netzes sind die Werte von Variablen der zur Verfügung stehenden Beobachtungen der Umgebung. Die versteckten Schichten verarbeiten diese Inputs mit ihren Aktivierungsfunktionen und Gewichtungen. Das Output des Netzes ist hingegen eine gewünschte Vorhersage, die mit ihrer eigenen Aktivierungsfunktion hergeleitet wird.

2.1.5 Proximal Policy Optimization

Proximal Policy Optimization oder kurz PPO ist eine neue Policy-Gradient-Methode für Reinforcement Learning. Im Gegensatz zu standardmäßigen Policy-Gradient-Methoden ist es mit dieser Methode möglich, mehrere Policy-Updates pro Datenpaket durchzuführen. PPO hat einige der Vorteile der Trusted Region Policy Optimization (kurz TRPO), ist aber simpler zu implementieren und hat auch andere Vorteile, wie bessere Generalisierbarkeit und niedrigere Stichprobenkomplexität. PPO zeigt eine gute Balance zwischen Stichprobenkomplexität, Einfachheit und Trainingsgeschwindigkeit [SWD⁺17, S. 1].

Die Stichprobenkomplexität beschreibt, wie groß ein Datenpaket sein muss, um dem Algorithmus ein gewisses Level an Performanz zu ermöglichen. Dies ermöglicht es mehr Updates mit

weniger Gesamtdaten durchzuführen, was vor allem hilfreich ist, wenn nicht genügend Daten zur Verfügung stehen oder diese nur schwer zu generieren sind.

Policy-Gradient-Methoden berechnen einen Gradienten und passen die Policy in Richtung der negativen Steigung dieses Gradienten an. Diese Anpassung verhält sich ähnlich, wie einen Punkt auf einer Parabel, welcher immer wieder in Richtung der negativen Steigung verschoben wird. Die Policy entspricht dem Punkt und wird so lange angepasst beziehungsweise verschoben, bis sie möglichst das Minimum der Parabel erreicht. Dieses Minimum spiegelt in der Theorie eine optimale Policy wieder. Es kann allerdings mehr als ein Minimum geben, was die Suche nach einer optimalen Policy erschwert.

Bei der Trusted Region Policy Optimization gibt es eine vertrauenswürdige Region, innerhalb derer die Policy abgeändert werden darf. Das soll sicherstellen, dass die Policy nicht zu stark abgeändert wird, um ein stabileres Training zu gewährleisten. Im Gegensatz zur PPO werden Änderungen, die zu stark abweichen, verworfen.

Bei der PPO kommt es zum sogenannten Clipping. Hierbei werden zu starke Änderungen im übertragenen Sinne abgeschnitten und es kommt zu einer abgeschwächten Änderung der Policy.

Der PPO-Algorithmus ist ein verhältnismäßig simpler, einfach zu verstehender und dennoch effizienter Algorithmus. Er hat eine gute Balance zwischen Stabilität und Effizienz. Außerdem erzielt er gute Ergebnisse bei einer großen Bandbreite an Aufgaben [SWD⁺17]. Daher eignet sich PPO besonders gut für Einsteiger, die bisher nicht viel mit Deep Reinforcement Learning gearbeitet haben.

2.2 Verwendete Technologien

2.2.1 Gymnasium

Gymnasium ist die Fortführung der OpenAI Bibliothek Gym [noab]. Sie kann genutzt werden, um Umgebungen zu schaffen, die für maschinelles Lernen verwendet werden können. Die Bibliothek bietet auch eine Menge vordefinierter Umgebungen, welche kostenfrei genutzt werden können, was gerade für Einsteiger den Vorteil hat, mit wenig Aufwand erste Erfahrungen zu sammeln.

Die Bibliothek bietet Kernmethoden, welche selbst gefüllt und implementiert werden müssen, um die Bibliothek mit einer benutzerdefinierten Umgebung nutzen zu können. Die wesentlichen Methoden, welche auf jeden Fall implementiert werden müssen, sind die Schritt-Methode (englisch: step method) und eine Methode zum Zurücksetzen der Umgebung auf den Startzustand (englisch: reset method). Außerdem muss eine Initialisierung der Umgebungsklasse erfolgen. Die Schritt-Methode nimmt eine Aktion entgegen und führt diese in der Umgebung aus. Außerdem liefert sie den Zustand der Umgebung nach dem Ausführen der Aktion und die Belohnung für

den ausgeführten Schritt zurück. Die Reset-Methode setzt alle relevante Werte der Umgebung auf den Ausgangswert zurück, sodass das Spiel oder die Aufgabe von vorne gestartet werden kann. Des Weiteren bietet es sich an eine Methode zu implementieren, welche den aktuellen Zustand der Umgebung zurückgibt.

Gymnasium bietet eine gute Anbindung an die Bibliothek Stable Baselines 3, welche Methoden zum Trainieren eines Modells auf Basis einer eben solchen Gymnasium-Umgebung bietet. In dieser Arbeit wird Gymnasium für die Implementierung der Spielumgebung von *"Ganz schön clever"* verwendet, damit diese anschließend mithilfe von Stable Baselines 3 trainiert werden kann.

2.2.2 Stable Baselines 3

Stable Baselines 3 ist eine Bibliothek, welche verlässliche Implementierungen von Reinforcement-Learning-Algorithmen in Pytorch bietet [noae, S. 1].

Die Algorithmen haben ein konsistentes Interface und eine umfangreiche Dokumentation, was es einfach macht verschiedene Reinforcement-Learning-Algorithmen zu testen. Die Implementierung bietet eine simple API. Modelle können in nur wenigen Codezeilen trainiert werden. Die Implementierung weist zudem eine hohe Qualität auf. Es gibt eine experimentelle Version der Bibliothek, welche Stable Baseline 3 Contributing genannt wird [noae, S. 1-3].

In diese Arbeit wird vor allem der MaskablePPO-Algorithmus aus eben dieser Contributing-Bibliothek benutzt. Dabei handelt es sich, um eine Erweiterung des PPO-Algorithmus von Stable Baseline 3. Dieser MaskablePPO-Algorithmus erweitert den PPO-Algorithmus um die Funktionalität einer Maskierung. Diese Maskierung ermöglicht es die Wahrscheinlichkeit bestimmter Aktionen auf Null zu setzen. Die Maskierung wurde in der Arbeit verwendet, um ungültige Aktionen auszuschließen, sodass das Modell nicht auf andere Weise lernen muss, diese zu vermeiden. Dies ist ein simpler und effizienter Weg, um zu gewährleisten, dass beim Training keine ungültigen Aktionen gewählt werden.

2.2.3 Matplotlib

Matplotlib ist eine umfangreiche und mächtige Bibliothek zum Plotten von Daten [noad]. In dieser Arbeit wird Matplotlib verwendet, um die erreichten Punktestände und ungültigen Züge zu visualisieren.

2.2.4 ChatGPT 4

ChatGPT 4 ist die neueste Version eines neuartigen Chat-Bots. Dieser ermöglicht es dem Benutzer Fragen zu stellen oder Aussagen zu treffen, auf die er daraufhin eine Antwort bekommt. Die Erzeugnisse des Chat-Bots sind so gut, dass er sich dafür eignet, um bei der Konzeption und Programmierung der Arbeit zu unterstützen. Daher wird ChatGPT 4 in dieser Arbeit als Hilfestellung bei Unklarheiten zur Funktionsweise von Technologien und beim Bau des Prototypen verwendet.

Zudem wird im Rahmen der Arbeit analysiert, wie gut sich ChatGPT 4 als unterstützendes Werkzeug eignet und welche Vor- und Nachteile, sowie welche Einschränkungen die Nutzung mit sich bringt.

3 Anforderungen und Konzeption

3.1 Anforderungen

Dieses Kapitel beschreibt die Anforderungen des Projektes. Das Projekt ist die Implementierung und die Analyse einer Spiel-KI für das Spiel *"Ganz schön clever"*.

3.1.1 Rahmenbedingungen

Es ist eine Künstliche Intelligenz zu implementieren, welche das Spiel *"Ganz schön clever"* gut spielen können soll. Dazu muss zunächst die Spielumgebung entwickelt und implementiert werden. Mithilfe dieser soll ein Verfahren entwickelt werden, welches die Künstliche Intelligenz in dieser Umgebung trainieren soll.

Anschließend sind die Ergebnisse des Entwicklungsprozesses sowie des Erzeugnisses selbst zu analysieren und vorzustellen. Dies soll mithilfe geeigneter, schlüssiger sowie verständlicher Methoden und Visualisierungstechniken erfolgen.

3.1.2 Spiel

Das Spiel heißt *"Ganz schön clever"*. Ziel des Spiels ist es, möglichst viele Punkte innerhalb einer vorgeschriebenen Rundenanzahl zu erreichen [s. Unterabschnitt 2.1.1].

Zunächst soll ein Prototyp der Spielumgebung entwickelt werden, welcher später schrittweise erweitert werden soll, bis das Spiel mit allen Funktionalitäten implementiert worden ist.

Diese Funktionalitäten sind:

- Die sechs farbigen Würfel, welche geworfen werden können und zufällig eine Würfelzahl von 1 bis 6 liefern. Es werden dabei immer alle aktuell gültigen Würfel gleichzeitig geworfen.
- Ein Mechanismus, welcher die Würfel für die aktuelle Runde als ungültig markiert, sobald sie gewählt werden. Ungültige Würfel dürfen nicht gewählt werden [s. Unterabschnitt 2.1.1].
- Ein Mechanismus, welcher Würfel mit einer niedrigeren Augenzahl als der aktuell gewählte Würfel als ungültig für die Runde markiert, solange sich das Spiel nicht in einer Wahl vom Silbertablett oder einer Wahl mithilfe des Extra-Wahl-Bonus befindet.

- Ein Runden-System, bei dem insgesamt sechs Runden gespielt werden, in denen jeweils bis zu drei Würfe erfolgen, solange bei einem der Würfe noch gültige Würfel vorhanden sind. Zudem erfolgt nach jeder Runde die Auswahl eines Würfels vom Silbertablett, welches die ungültigen Würfel eines anderen Spielers beinhaltet. Wenn das Spiel alleine gespielt wird und es dadurch keinen anderen Spieler gibt, werden hierzu alle Würfel geworfen und drei davon auf das Silbertablett gelegt. Außerdem werden Boni am Anfang der ersten, zweiten, dritten und vierten Runde für alle Spieler freigeschaltet [s. Unterabschnitt 2.1.1].
- Die fünf farbigen Felder, gelb, blau, grün, orange und lila. Jedes Feld hat seine eigenen Regeln, wenn es darum geht wann ein Würfel gewählt werden darf, um eines der Kästchen auf diesem Feld auszufüllen [s. Unterabschnitt 2.1.1]. Die Felder beinhalten verschiedene Belohnungen, die freigespielt werden, wenn bestimmte Kästchen oder Kombinationen aus diesen ausgefüllt worden sind [s. Unterabschnitt 2.1.1].
- Die zehn verschiedenen Boni, Fuchs, Extra-Wahl, Neu-Würfeln, Gelbes-Kreuz, Blaues-Kreuz, Grünes-Kreuz, Orangene-Vier, Orangene-Fünf, Orangene-Sechs sowie Lila-Sechs [s. Unterabschnitt 2.1.1]. Dies beinhaltet sowohl das Erhalten, Speichern, sowie die Benutzung der Boni.
- Ein Mechanismus, welcher Würfel, die mithilfe eines Extra-Wahl-Bonus gewählt wurden, als ungültig markiert, damit diese nicht erneut in der selben Runde durch mit einem Extra-Wahl-Bonus gewählt werden können.
- Ein Mechanismus, der Würfel zur richtigen Zeit wieder als gültig markiert. Dies erfolgt nach dem dritten Wurf in der Runde, nach Extra-Wahlen im eigenen Zug (nach dem dritten Wurf), nach der Wahl vom Silbertablett des Gegners und nach Extra-Wahlen, die nach der Wahl vom Silbertablett des Gegners erfolgen. Würfel werden nicht nach jedem Einsatz eines Extra-Wahl-Bonus wieder gültig, sondern erst nachdem der Spieler keinen weiteren Extra-Wahl-Bonus mehr hat oder sich entscheidet zum gegebenen Zeitpunkt keinen weiteren zu verwenden.

3.1.3 Künstliche Intelligenz

Die Künstliche Intelligenz soll das Spiel möglichst gut spielen können. Es sollen geeignete Methoden zur Verfügung stehen, um die KI trainieren und mit ihr nach dem Training Vorhersagen [s. Unterabschnitt 2.1.2] treffen zu können. Außerdem sollte sie Möglichkeiten bieten verschiedene Parameter anzupassen, um den Trainingsprozess zu optimieren.

Die wichtigsten dieser Parameter sind unter Anderem:

- Trainingsdauer, welche festlegt, wie lange trainiert wird.

- Gamma, welches bestimmt, wie zukunftsorientiert die KI ihre Entscheidungen fällt.
- Die Größe und Art des neuronalen Netzes, das verwendet wird, welches bestimmt, wie genau und präzise Merkmale von der Künstlichen Intelligenz erfasst werden können.
- Mindestens einen Faktor, welcher das Ausmaß von Erkundung und Ausbeutung steuern soll.

Außerdem soll die Implementierung auf einfache Art und Weise möglich sein, damit der zeitliche Rahmen der Arbeit eingehalten werden kann.

3.2 Konzeption

In diesem Kapitel wird das Grundkonzept der Künstlichen Intelligenz sowie der Spielumgebung und deren Zusammenspiel beschrieben.

Abbildung 6 zeigt das Zusammenspiel von Spielumgebung und KI während des Trainingsprozesses:

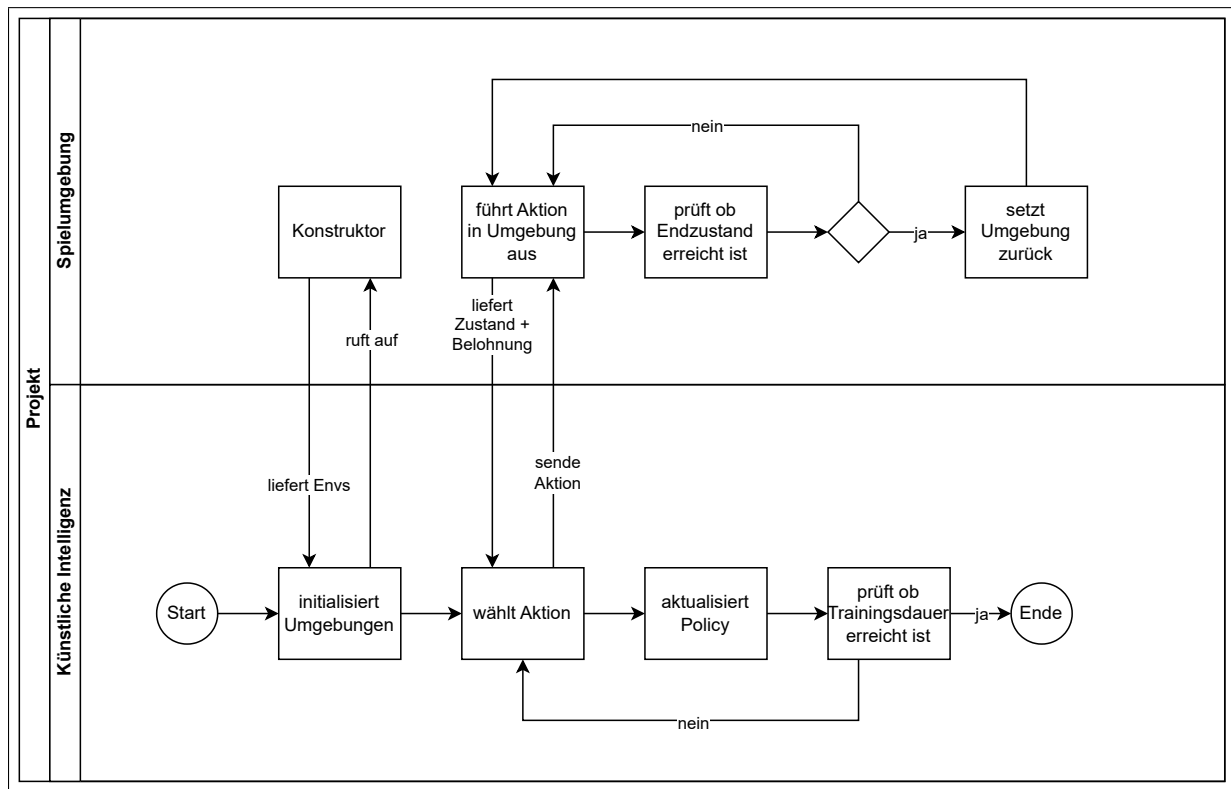


Abb. 6: Swimlane-Diagramm des Trainingsprozesses der KI
Quelle: Eigene Darstellung

Wie Abbildung 6 zeigt wird zunächst der Trainingsprozess angestoßen und die Umgebung wird initialisiert. Dazu wird der Konstruktor der Umgebung aufgerufen. Daraufhin wird eine Aktion von der KI ausgewählt und an die Umgebung weitergeleitet. Die Umgebung führt diese Aktion aus und prüft ob der Endzustand des Spiels erreicht wurde oder nicht. Wenn der Endzustand erreicht wurde, wird die Umgebung auf ihren Anfangszustand zurückgesetzt. Wenn der Endzustand nicht erreicht worden ist, wird der Prozess ohne Weiteres fortgesetzt. Daraufhin liefert die Umgebung der KI eine Belohnung [s. Unterabschnitt 2.1.3] für den ausgeführten Schritt und den neuen Zustand der Umgebung nach Ausführung der Aktion.

Daraufhin aktualisiert die Künstliche Intelligenz ihr neuronales Netz mit der Policy und überprüft ob bereits genügend Zeit vergangen ist, um das Training zu beenden. Wenn die festgelegte Zeit vergangen ist wird das Training an dieser Stelle beendet. Wenn nicht wird die nächste Aktion gewählt und das Training wird fortgesetzt.

Dies ist lediglich eine vereinfachte Darstellung des Prozesses. In Wirklichkeit wird die Policy nicht nach jedem Schritt/jeder Aktion aktualisiert, sondern es wird erst eine festgelegte Menge an Aktions-Zustands-Paaren gesammelt. Dann wird das neuronale Netz mit dieser Gesamtmenge aktualisiert. Diese Aktions-Zustands-Paare ermöglichen es festzustellen, welche Aktionen in welchen Zuständen zu einem guten Ergebnis führten und welche nicht. Hierfür wird für jeden Zustand ein geschätzter Wert über seinen zukünftigen Nutzen erstellt. Dafür wird ein neuronales Netz, das sogenannte Value Network genutzt. Dieser geschätzte Nutzen wird mit dem tatsächlich erzielten Nutzen verglichen. Die Policy wird anschließend in Richtung vorteilhafter Zustände angepasst, die einen möglichst besseren tatsächlichen als erwarteten Nutzen aufweisen.

3.2.1 Spielumgebung

Die Abbildungen 7 und 8 zeigen den prinzipiellen Funktionsablauf in der Spielumgebung. Dabei wird am Anfang eine Aktion entgegengenommen und am Schluss der neue Zustand dieser Umgebung nach Ausführung der Aktion zurückgegeben:

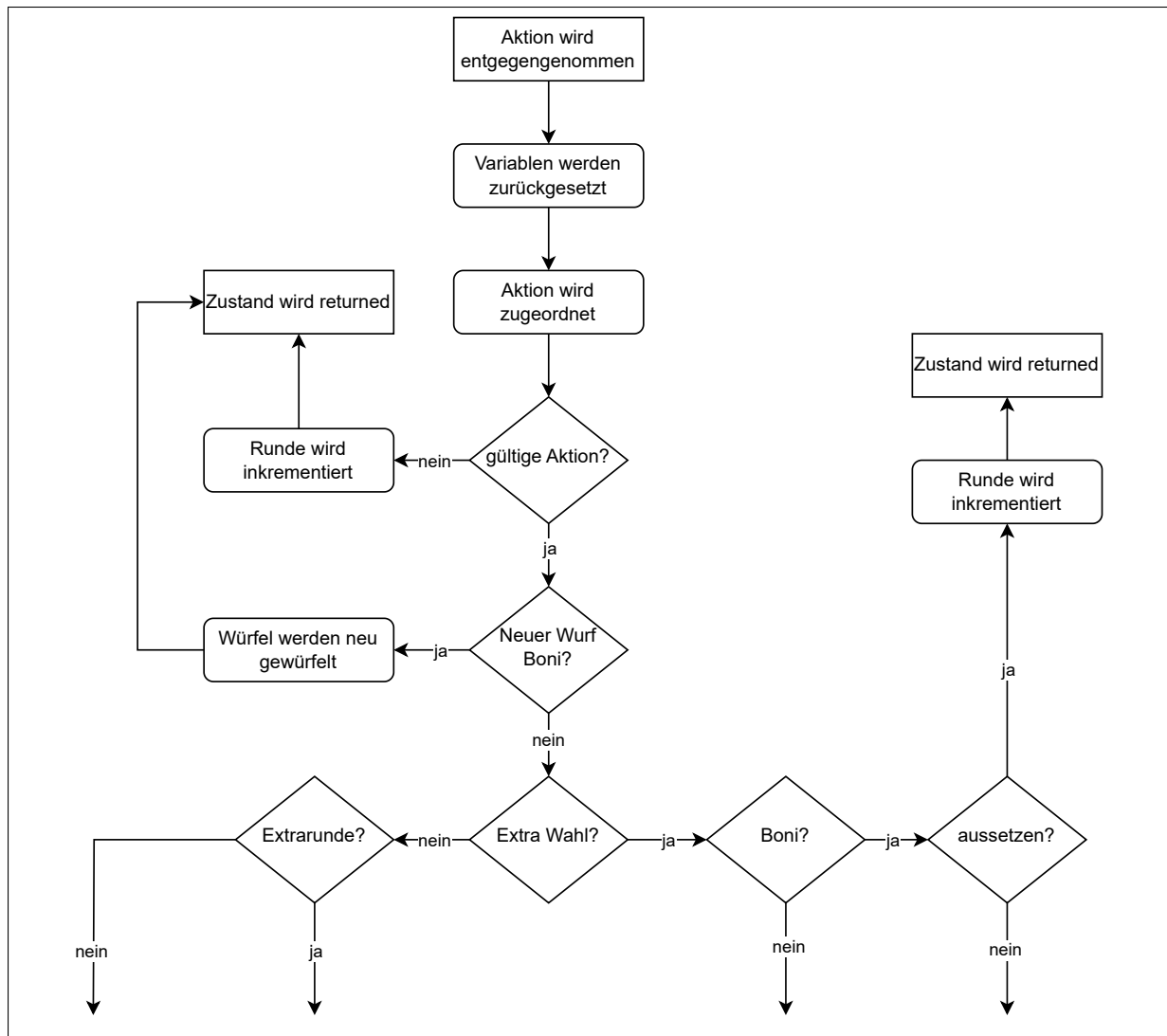


Abb. 7: Ablauf-Diagramm der Schritt Funktion 1
Quelle: Eigene Darstellung

Wie Abbildung 7 zeigt wird der Vorgang durch eine Aktion angestoßen. Nach dem Anstoßen, werden die Variablen der Funktion zurückgesetzt, um einen klaren Schnitt zwischen vergangenen Aktionen und der aktuellen Aktion zu erzielen.

Anschließend wird die Aktion ihrem entsprechenden Ablauf zugeordnet. Es gibt Aktionen für Wahlen nach normalen Würfeln, für Wahlen von Würfeln des Silbertablettes sowie für das Nutzen von Boni. Außerdem gibt es eine Aktion, die nur dann wählbar ist, wenn keine der anderen Aktionen gewählt werden kann. Dies ist die Aktion für ungültige Züge.

Ist die Aktion nicht gültig, wird die Runde inkrementiert und der Zustand der Umgebung zurückgegeben.

Wird die Neu Würfeln Boni verwendet, werden die gültigen Würfel neu geworfen und anschließend der Zustand zurückgegeben.

Ist beides nicht der Fall kommt es zu einer Auswahl zwischen den wesentlichen Aktionen der Umgebung. Es gibt sogenannte Extra-Wahlen. Diese spalten sich auf in Extra-Wahlen mit einem Extra-Wahl-Bonus und Wahlen vom Silbertablett eines Mitspielers.

Handelt es sich nicht um eine Extra-Wahl, so folgt eine Unterteilung in Extra-Runden und Normale-Runden. Bei Extra-Runden wird eine der Boni verwendet (in Abb. 8: Boni Wahl), die es erlauben eines der farbigen Felder direkt auszufüllen [s. Unterabschnitt 2.1.1].

Handelt es sich nicht um eine Extra-Runde muss es sich in diesem Fall um eine Normale-Runde (in Abb. 8: Normale Wurf Wahl) nach einem gewöhnlichen Wurf handeln [s. Abb. 2].

Handelt es sich um eine Extra-Wahl mit Boni (mit Extra-Wahl-Bonus), dann steht noch die Aktion "Passen" zur Wahl. Diese inkrementiert lediglich die Runde, sodass das Spiel weiter fortgesetzt werden kann. Dies spiegelt die Möglichkeit im Spiel wieder auf die Nutzung eines Extra-Wahl-Bonus zu verzichten.

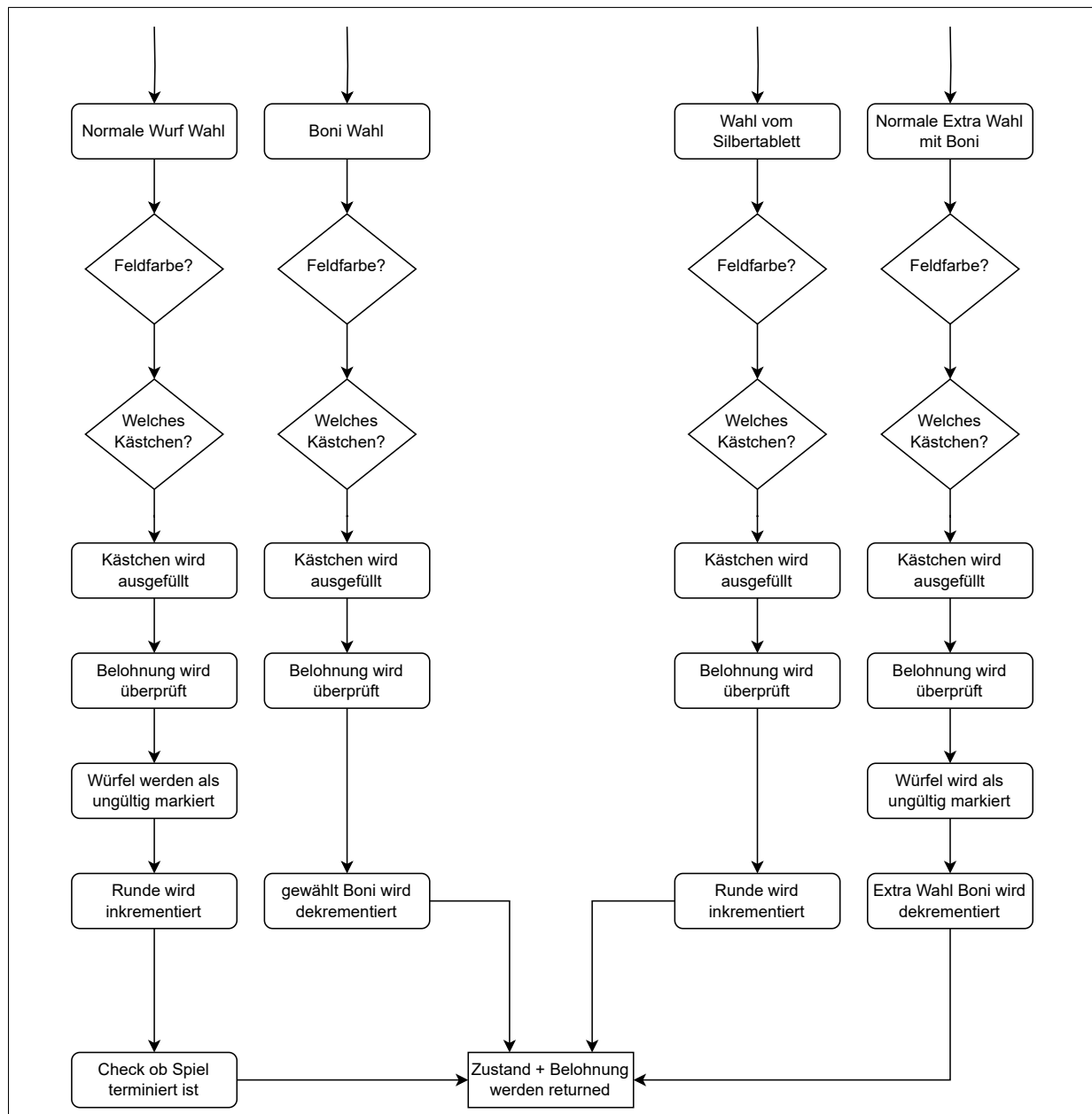


Abb. 8: Ablauf-Diagramm der Schritt Funktion 2

Quelle: Eigene Darstellung

Wie in Abbildung 8 dargestellt gestaltet sich der folgende Ablauf der Funktion relativ ähnlich bei allen vier Varianten. Es wird das Feld und das Kästchen ermittelt, welches ausgefüllt werden soll und das entsprechende Kästchen wird anschließend ausgefüllt. Dann wird die Belohnung für diesen Schritt ermittelt.

Am Ende der Funktion unterscheiden sich die vier Varianten wieder. Bei einer Normalen-Runde nach einem normalen Wurf werden die entsprechenden Würfel als ungültig markiert [s. Unterabschnitt 2.1.1] und die Runde wird inkrementiert. Außerdem prüft die Funktion bei dieser Variante am Ende, ob das Spiel terminiert ist oder nicht.

Beim Ausfüllen eines Feldes mittels eines Bonus (kein Extra-Wahl-Bonus) wird der entsprechende Bonus dekrementiert.

Bei der Extra-Wahl vom Silbertablett wird die Runde inkrementiert.

Bei der Extra-Wahl mittels Bonus wird der gewählte Würfel als ungültig markiert und der Extra-Wahl-Bonus dekrementiert.

Bei allen vier Varianten wird am Ende der Funktion der aktuelle Zustand der Umgebung sowie die erhaltene Belohnung für den Schritt zurückgegeben.

3.2.2 Künstliche Intelligenz

Abbildung 9 zeigt die wesentlichen Hyperparameter, die bei der Entwicklung der Künstlichen Intelligenz von Bedeutung sind:

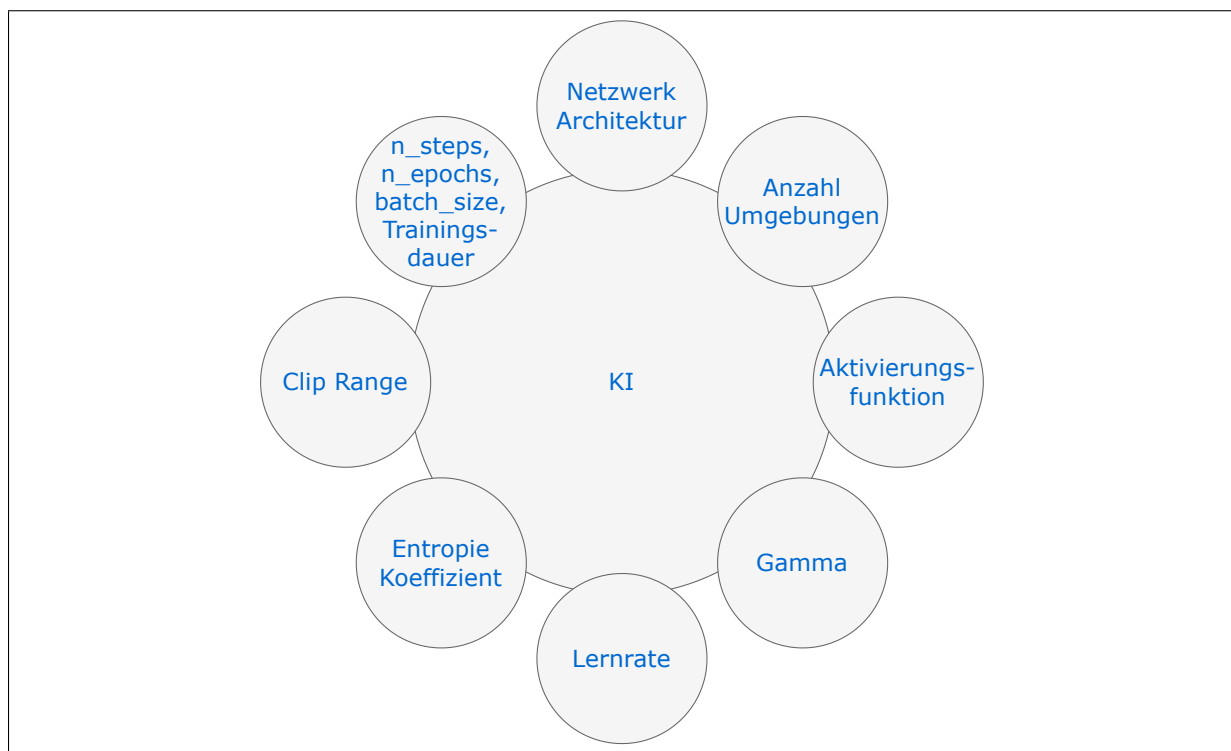


Abb. 9: Wesentliche Hyperparameter der Künstlichen Intelligenz
Quelle: Eigene Darstellung

Die Künstliche Intelligenz basiert auf der MaskablePPO-Version von Stable Baselines 3 Contributing. Es sind Konfigurationen zu treffen, welche das Training beeinflussen und steuern. Diese Einstellungen werden Hyperparameter genannt. Hyperparameter legen die Ausprägung gewisser Steuerelemente des MaskablePPO-Algorithmus und ähnlicher Reinforcement-Learning-Algorithmen fest.

Die wichtigsten Hyperparameter für das Projekt sind folgende [s. Abb. 9]:

- **Netzwerkarchitektur:** Legt die Anzahl der Schichten des neuronalen Netzes fest. Dieses Netz wird zur Abbildung der Policy verwendet. Außerdem legt die Netzwerkarchitektur fest, wie viele Neuronen es pro Schicht gibt [s. Unterabschnitt 2.1.4]. Die Netzwerkarchitektur ist so zu bestimmen, dass die Künstliche Intelligenz im Stande ist, die Komplexität der Problemstellung zu erfassen und gleichzeitig zu gewährleisten, dass die Trainingsdauer und die Systemauslastung dadurch nicht zu hoch steigen. Außerdem sollte es nicht zu einer Überanpassung an die Trainingsdaten kommen. Da es sich bei dem neuronalen Netz um ein Multilayer Perceptron handelt, sind alle Neuronen einer Schicht mit allen der vorherigen und folgenden Schicht verbunden [s. Unterabschnitt 2.1.4]. Dies führt dazu, dass die Rechenleistung für das Updaten der Gewichte der Policy exponentiell steigt, je komplexer das neuronale Netz wird. Dies führt zu einer gesteigerten Trainingsdauer und Systemauslastung. Außerdem tendieren zu komplexe Neuronale Netze dazu, sich stark an die Trainingsdaten anzupassen, was zu Überanpassung führen kann. Zu komplexe Netze generalisieren tendenziell schlechter auf neue Datensätze, die sich stark von den Trainingsdaten unterscheiden.
- **Anzahl der Umgebungen:** Eine erhöhte Anzahl an Trainingsumgebungen ermöglicht es, besonders bei Nutzung von CUDA (GPU), parallel Daten aus mehreren Umgebungen zu sammeln und zu verarbeiten. Dies erhöht die Trainingsgeschwindigkeit, da ein erheblicher Teil der Trainingsdauer in diesem Projekt davon abhängt, wie schnell die nötigen Trainingsdaten aus den Umgebungen generiert werden können. Außerdem erhöht die Nutzung von CUDA die Geschwindigkeit von Policy Updates enorm, da viele der Berechnungen im neuronalen Netz parallel abgearbeitet werden können. Allerdings erhöht die Anzahl der Umgebungen die RAM Auslastung enorm und die CPU Auslastung mittelmäßig stark, was dazu führt, dass es ineffizient wäre auf dem verwendeten System (Nvidia Geforce GTX 1070, Intel i5 8600k, 16GB RAM) deutlich mehr als 32 Umgebungen zu betreiben.
- **Aktivierungsfunktion:** Legt zusammen mit den Gewichtungen der Neuronen fest, wann und wie stark diese ihre Signale an die folgenden Neuronen weiterleiten. Im Projekt wird vor allem die ReLU-Funktion verwendet, da diese standardmäßig vom Algorithmus verwendet wird und im allgemeinen sowie spezifisch in diesem Projekt gute Ergebnisse erzielt. Die Formel der ReLU-Funktion lautet: $f(x) = \max(0, x)$. Die ReLU-Funktion gibt, wenn der Wert kleiner als Null ist, Null zurück und ansonsten den Wert selbst [?].

Abbildung 10 zeigt den Graphen der ReLU-Funktion:

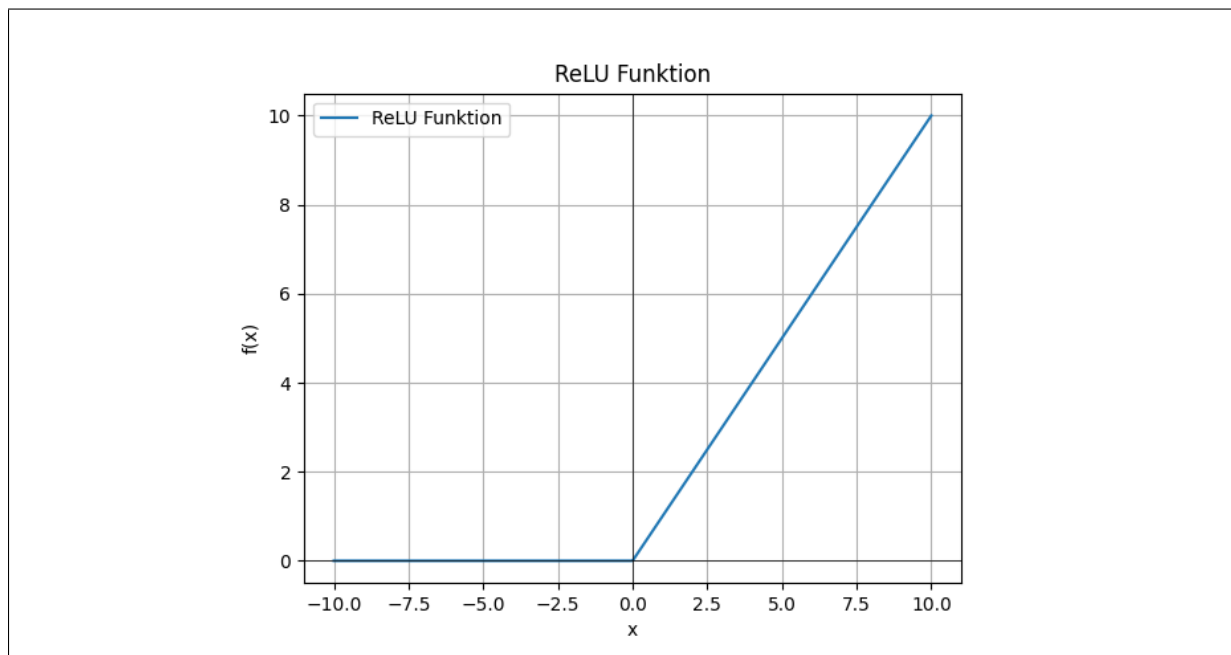


Abb. 10: Graph der ReLU-Funktion
Quelle: Eigene Darstellung

- **Gamma:** Legt fest, wie stark zukünftige Belohnungen wertgeschätzt werden. Es fungiert als eine Art Multiplikator. Potenzielle Belohnungen werden pro Spielschritt, den sie in der Zukunft entfernt liegen würden, mit Gamma multipliziert. Gamma ist so zu wählen, dass das Modell lernt, die bestmöglichen Aktionen zu wählen, um insgesamt das beste Ergebnis zu erzielen. Zu hohe Werte von Gamma können dazu führen, dass kurzzeitige Belohnungen vernachlässigt werden, was zu einem schlechteren Gesamtergebnis führen kann, wenn diese von besonderer Bedeutung sind. Zu niedrige Werte von Gamma können wiederum dazu führen, dass der Agent nicht zukunftsorientiert handelt und somit kein gutes Gesamtergebnis erzielt, da er sich hauptsächlich auf kurzfristige Belohnungen konzentriert, anstatt in die Zukunft zu schauen und dieser Wert zuzumessen. In diesem Projekt bewegt sich der Wert von Gamma im Allgemeinen in einem Bereich zwischen 0.5 und 1.
- **Lernrate:** Bestimmt, wie stark Updates des neuronalen Netzes und seiner Gewichtungen pro Updateschritt ausfallen. Zu große Werte für die Lernrate können zu einem instabilen Training führen, da die Updates somit stark vom Trainingsdatensatz abhängen. Zu niedrige Werte führen wiederum zu einer Erhöhung der benötigten Trainingsdaten und Trainingsdauer. Der Wert für die Lernrate bewegt sich im Projekt üblicherweise zwischen Werten von 0.0003 bis zu 0.0003x32.
- **Entropie-Koeffizient:** Bestimmt das Maß an Exploration des Modells. Je höher der Entropie-Koeffizient ist desto stärker Belohnt das Modell neue beziehungsweise bisher wenig er-

kundete Aktionen oder Zustände. Ein zu hoher Wert führt dazu, dass das Modell nicht lernt bereits funktionierende Taktiken ausreichend zu verfestigen. Ein zu niedriger Wert führt dazu, dass das Modell sich relativ schnell auf eine bisher vergleichsweise gut funktionierende Strategie festlegt und diese verfestigt. Der Entropie-Koeffizient bewegt sich innerhalb des Projektes meist zwischen Werten von 0.05 und 0.3, wobei sich ein Wert um die 0.1 als stabil und effizient herausgestellt hat.

- **Clip-Range:** Die Clip-Range ist spezifisch für den PPO-Algorithmus [s. Unterabschnitt 2.1.5]. Sie legt fest wie stark Policy Updates sein dürfen und ab welchem Schwellwert die Updates abgeschnitten werden. Abschneiden heißt, dass die Updates zwar durchgeführt werden, allerdings darf die Policy nach dem Update nur maximal um einen bestimmten Wert von der vorherigen abweichen. Die standardmäßige Clip-Range beläuft sich auf 0.2. Innerhalb des Projektes werden auch andere Werte getestet.
- **nSteps:** Legt fest, wie viele Aktions-Zustands-Paare gesammelt werden, bevor ein Update des neuronalen Netzes erfolgt.
- **nEpochs:** Legt fest wie oft dasselbe Datenpaket von nSteps für das Training benutzt wird bevor es verworfen wird. Je öfter man es verwendet, desto weniger Daten braucht man und desto schneller verläuft das Training. Allerdings sollte der selbe Datensatz nicht zu häufig verwendet werden, um Überanpassung zu vermeiden. nEpochs hat im Rahmen des Projektes üblicherweise einen Wert zwischen 5 und 11.
- **Batch Size:** Das Datenpaket von nSteps wird vor der Verwendung für das Updaten des neuronalen Netzes in kleinere Datenpakete, die sogenannten Batches, zerlegt. Daraufhin werden Updates mit jedem dieser Batches durchgeführt. Die Aufteilung erfolgt per Zufall, daher ist mit hoher Wahrscheinlichkeit keines der Datenpakete (Batches) wie das andere, selbst wenn dasselbe Gesamtdatenpaket durch ein hohes nEpochs viele male verwendet wird. Batch Size legt fest wie groß diese kleineren Datenpakete (Batches) sind.
- **Trainingsdauer:** Die Trainingsdauer legt fest wie lange trainiert wird. Die Implementierung von Stable Baselines [s. Kapitel 2.2.2] verwendet standardmäßig Timesteps zur Messung von Zeit. Eine Timestep ist ein ausgeführter Schritt beziehungsweise eine ausgeführte Aktion in einer Umgebung. Die Timesteps von mehreren parallel laufenden Umgebungen werden aufsummiert, somit kommt es nicht zu vermehrtem Training alleine durch Erhöhung der Umgebungsanzahl bei gleichbleibender Trainingsdauer in Timesteps.

4 Implementierung

In diesem Kapitel wird die Implementierung des Projektes vorgestellt und erläutert. Zur Darstellung der Funktionsweise der einzelnen Funktionen wird Pseudocode verwendet. Dies soll dabei helfen die Funktionsweise verständlicher und knapper aufzuzeigen.

Einige der Attribute und Methoden der Implementierung sind in diesem Kapitel nicht aufgeführt und können in den Anhängen A1 und A2 nachgeschlagen werden.

4.1 Einschränkungen

Aus zeitlichen Gründen ergeben sich einige Einschränkungen für das Design der Implementierung. Das Spiel endet nach der Wahl des Würfels nach dem dritten Wurf in der sechsten Runde. Somit kann der Extra-Wahl-Bonus kein letztes Mal benutzt werden und es gibt in der letzten Runde auch keine Wahl vom Silbertablett. Außerdem werden nicht die niedrigsten sechs Würfel des Wurfes auf das Silbertablett des Mitspielers gelegt, sondern drei zufällige.

4.2 Spielumgebung

Die Implementierung besteht aus zwei Klassen. Eine davon ist die Spielumgebung. Zunächst werden in diesem Kapitel die wesentlichen Attribute der Spielumgebung und anschließend ihre Methoden erläutert. Für die Implementierung der Spielumgebung wurde die Bibliothek Gymnasium [s. Unterabschnitt 2.2.1] verwendet.

4.2.1 Klassenattribute

Code 1 zeigt die Klassenattribute, die für das Rundenmanagement wichtig sind:

```
1 self.initial_rounds
2 self.rounds
3 self.roll_in_round
```

Code 1: Klassenattribute für das Runden-System

Das Attribut `initial_rounds` beschreibt die maximale Rundenanzahl des Spiels und wird beim Zurücksetzen der Umgebung verwendet, um die Rundenzahl auf den gewünschten Wert (im Solo-Spiel sechs) zurückzusetzen.

Das Attribut `rounds` repräsentiert die aktuell verbleibende Rundenanzahl im Spiel.

Das Attribut `roll_in_round` repräsentiert die Nummer des aktuellen Wurfes in der Runde.

Code 2 zeigt die Klassenattribute, die für das Würfeln der Würfel relevant sind:

```
1 self.invalid_dice = {"white": False, "yellow": False, ...}
2 self.dice = {"white": 0, "yellow": 0, ...}
```

Code 2: Klassenattribute für Würfel

Die Attribute `invalid_dice` und `dice` repräsentieren die Augenzahlen der Würfel, sowie die Gültigkeit der Würfel selbst. Ist der Wert von `invalid_dice` `False`, ist der Würfel nicht ungültig und somit gültig.

Das Attribut `score` repräsentiert den aktuellen Punktestand der Spielumgebung. Es wird verwendet, um erreichte Punktestände in die `score_history` einzutragen. Das Attribut `score_history` ist eine Historie über die erreichten Punktestände in den einzelnen Episoden beziehungsweise Spieldurchläufen. Das Attribut `initialized` wird verwendet, um zu gewährleisten, dass nur Einträge in der Punktestandhistorie eingetragen werden, nachdem ein Spiel abgeschlossen wurde. Es trifft eine Aussage darüber ob die Spielumgebung bereits einmal initialisiert wurde oder nicht.

Code 3 zeigt die Klassenattribute, welche die farbigen Felder des Spielbrettes repräsentieren:

```
1 self.yellow_field = [[3, 6, 5, 0], [2, 1, 0, 5], ...]
2 self.blue_field = [[0, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
3 self.green_field = [0] * 11
4 self.orange_field = [0] * 11
5 self.purple_field = [0] * 11
```

Code 3: Klassenattribute für die farbigen Felder des Spiels

Die Attribute `yellow_field`, `blue_field`, `green_field`, `orange_field` und `purple_field` stehen für die fünf farbigen Felder auf dem Spielbrett. Sie repräsentieren die eingetragenen Werte auf dem Spielbrett und bestimmen somit welche Kästchen aktuell ausgefüllt werden können (vorausgesetzt die Würfelergebnisse passen) und welche Belohnungen freigeschaltet werden.

Code 4 zeigt die Klassenattribute, welche die zu erspielenden Boni auf den farbigen Feldern repräsentieren:

```
1 self.yellow_rewards = {"row": ["blue_cross", ...], "dia": ...}
2 self.blue_rewards = {"row": ["orange_five", ...], "col": ...}
3 self.green_rewards = [None, None, None, "extra_pick", ...]
4 self.orange_rewards = [None, None, "re_roll", ...]
5 self.purple_rewards = [None, None, "re_roll", ...]
```

Code 4: Klassenattribute für freizuschaltende Boni

Die Attribute `yellow_rewards`, `blue_rewards`, `green_rewards`, `orange_rewards` und `purple_rewards` repräsentieren die freizuschaltenden Boni für die jeweiligen farbigen Felder. Für das blaue Feld sind diese in Form von Reihen (`row`) und Spalten (`col`) aufgeführt. Das gelbe Feld besitzt Boni für das Ausfüllen von Reihen (`row`) und einen Boni, der bei diagonalem Ausfüllen (`dia`) freigeschaltet werden kann. Für die Farben grün, orange und lila sind die Boni jeweils direkt einem der Kästchen im Feld zugewiesen, wobei viele der Kästchen keinen freizuschaltenden Bonus aufweisen, was dem Wert `None` entspricht.

Code 5 zeigt die Klassenattribute, für die Punktebelohnungen des gelben, blauen und grünen Feldes:

```
1 self.yellow_rewards = {"col": [10, 14, 16, 20], ...}
2 self.blue_count_rewards = [0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3 self.green_count_rewards = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Code 5: Klassenattribute für freizuschaltende Punktebelohnungen des gelben, blauen und grünen Feldes

Im Attribut `yellow_rewards` sind die Punktebelohnungen im Spaltenbereich (`col`) des gelben Feldes aufgeführt. Die Attribute `blue_count_rewards` und `green_count_rewards` repräsentieren die Punktebelohnungen, welche erspielt werden können, sobald ein blaues beziehungsweise grünes Kästchen ausgefüllt wird. Beginnend vom ersten Wert des Arrays und danach inkrementell aufsteigend, steigt die erhaltene Punktebelohnung bei jedem ausgefüllten Kästchen stetig an.

Code 6 zeigt die Klassenattribute für freigeschaltete Boni. Wird eine Boni erspielt, wird ihr Wert inkrementiert, wird sie genutzt, wird er dekrementiert:

```
1 self.extra_pick
2 self.re_roll
3 self.fox
4 self.yellow_cross
5 self.blue_cross
6 self.green_cross
7 self.orange_four
8 self.orange_five
9 self.orange_six
10 self.purple_six
```

Code 6: Klassenattribute für freigespielte Boni

Code 7 zeigt die Klassenattribute für den Aktions- sowie Beobachtungsraum:

```
1 self.number_of_actions = 247
2 low_bound = np.array([0]*16 + [0]*12 + ...)
3 high_bound = np.array([6]*16 + [6]*12 + ...)
4 self.action_space = spaces.Discrete(self.number_of_actions)
5 self.observation_space = spaces.Box(low_bound, high_bound, ...)
6 self.valid_action_mask_value = np.ones(self.number_of_actions)
```

Code 7: Klassenattribute des Aktions- und Beobachtungsraumes

Das Attribut `number_of_actions` repräsentiert die Gesamtanzahl an möglichen Aktionen des Modells. Die Attribute `low_bound` und `high_bound` setzen die obere und untere Grenze von Werten im Beobachtungsraum fest. Beispielsweise steht der erste Eintrag in beiden für Werte des gelben Feldes. Diese können von null (`[0]*16`) bis sechs (`[6]*16`) reichen. Das Attribut `action_space` repräsentiert den Aktionsraum des Modells. Es ist ein diskreter Raum mit `number_of_actions` Werten von null bis `number_of_actions` minus eins. Das Attribut `observation_space` repräsentiert den Beobachtungsraum. Shape definiert dabei die Größe des Beobachtungsraumes. Das Attribut `valid_action_mask_value` repräsentiert die Aktionsmaske. Initial handelt es sich dabei um ein Numpy Array aus Einsen. Einsen stehen für gültige Aktionen, Nullen für ungültige. Die Werte verlaufen parallel zu den Werten des Aktionsraumes. Somit repräsentieren alle Werte (beispielsweise `[0]` oder `[5]`) sowohl im Aktionsraum als auch bei der Aktionsmaske die selbe Aktion.

Die Struktur des Aktionsraumes ist für das Verständnis der Arbeit wichtig, daher wird sie hier erläutert:

Die ersten 122 Werte des Aktionsraumes von 0 bis 121 sind sowohl normalen Wahlen bei eigenen Würfeln als auch den verschiedenen Boni, welche es ermöglichen direkt ein Kästchen eines Feldes anzukreuzen, zugeordnet. Dabei stehen die ersten 16 Werte für Kästchen des gelben Feldes, die nächsten 12 für Kästchen des blauen Feldes, und die folgenden 33 zu einer Aufteilung von jeweils 11 für Kästchen im grünen, orangenen und lila Feld. Die Werte von 61 bis 121 stehen für die selben Kästchen in der selben Reihenfolge wie die vorherigen 61 Werte, allerdings wird bei diesen der weiße Würfel verwendet statt des jeweils farbigen Würfels für das spezifische Feld.

Die Werte von 122 bis 243 stehen für Wahlen mit der Extra-Wahl-Boni oder für Wahlen vom Silbertablett des Gegners. Die Struktur innerhalb dieser Reichweite ist die selbe wie bei den 122 Werte zuvor. Die ersten 16 Werte stehen für die selben Kästchen im gelben Feld und so weiter.

Der Wert 244 steht für die Neu Würfeln Boni, der Wert 245 für das Passen bei einem möglichen Einsatz von Extra-Wahl-Boni und der Wert 246 für eine ungültige Aktion, die nur möglich ist, wenn keine der anderen Aktionen getätigt werden kann.

4.2.2 Schritt-Methode

Code 12 zeigt die Funktionsweise der Schritt-Methode (`step method`) der Spielumgebung mithilfe von Pseudocode. Diese Methode führt Spielschritte beziehungsweise Aktionen in der Spielumgebung aus:

```
1 step(Aktion):
2     Setze Methodenattribute zurück
3
4     if not Extra-Wahl:
5         if Aktion im gelben Feld:
6             Fülle entsprechendes gelbes Kästchen aus
7             if Gelbes-Kreuz-Boni >= 1
8                 Bonusrunde = True
9                 Gelbes-Kreuz-Boni -= 1
10            if not Bonusrunde:
11                Setze entsprechende Würfel auf ungültig
12        if Aktion im blauen Feld:
13            Fülle entsprechendes blaues Kästchen aus
14            if Blaues-Kreuz-Boni >= 1:
15                Bonusrunde = True
```



```
16         Blaues-Kreuz-Boni -= 1
17         if not Bonusrunde:
18             Setze entsprechende Würfel auf ungültig
19         if Aktion im grünen Feld:
20             Fülle entsprechendes grünes Kästchen aus
21             if Grünes-Kreuz-Boni >= 1:
22                 Bonusrunde = True
23                 Grünes-Kreuz-Boni -= 1
24             if not Bonusrunde:
25                 Setze entsprechende Würfel auf ungültig
26         if Aktion im orangene Feld:
27             Fülle entsprechendes orangenes Kästchen aus
28             if Orangene-Feld-Boni >= 1:
29                 Bonusrunde = True
30                 Orangene-Feld-Boni -= 1
31             if not Bonusrunde:
32                 Setze entsprechende Würfel auf ungültig
33         if Aktion im lila Feld:
34             Fülle entsprechendes lila Kästchen aus
35             if Lila-Sechs-Boni >= 1:
36                 Bonusrunde = True
37                 Lila-Sechs-Boni -= 1
38             if not Bonusrunde:
39                 Setze entsprechende Würfel auf ungültig
40
41         if Extra-Wahl:
42             Bonusrunde = True
43             if Extra-Wahl-Bonus benutzt:
44                 Extra-Wahl-Boni -= 1
45             Finde Kästchen zum ausfüllen und fülle es aus
46             Setze gewählten Würfel auf ungültig
47             if Extra-Wahl-Boni <= 0 or Wahl erfolgte vom Silbertablett:
48                 Runde wird inkrementiert
49
50         if Neu-Würfeln-Bonus wird benutzt:
51             Werfe Würfel neu
52             Neu-Würfeln-Boni -= 1
53             Aktualisiere Aktionsmaske
54             return Beobachtungsraum, Punktebelohnung, Spiel terminiert?
55
56         if Statt Extra-Wahl-Boni gepasst wird:
```

```
57     Runde wird inkrementiert
58     return Beobachtungsraum, Punktebelohnung, Spiel terminiert?
59     if ungültige Aktion gewählt da keine gültigen Aktionen vorhanden:
60         Nichts tun
61
62     Punktebelohnung += erspielte Belohnung in diesem Schritt
63     if not Bonusrunde:
64         Runde wird inkrementiert
65         if Rundenanzahl == 0:
66             terminiert = True
67             Punktebelohnung += Fuchsbonipunktebelohnung
68     Aktualisiere Aktionsmaske
69     return Beobachtungsraum, Punktebelohnung, Spiel terminiert?
```

Code 8: Schritt-Methode

4.2.3 Methode zum Zurücksetzen der Spielumgebung

Code 8 zeigt die Funktionsweise der Methode zum Zurücksetzen der Spielumgebung (`reset method`) mithilfe von Pseudocode. Sie setzt Umgebungen, in denen ein Spiel abgeschlossen wurde, auf den Anfangszustand zurück, damit eine weitere Runde gespielt werden kann:

```
1 reset():
2     Setze alle Attribute für den Spielablauf auf den Startzustand
```

Code 9: Methode zum Zurücksetzen der Umgebung

4.2.4 Würfel-Methode

Code 9 zeigt die Funktionsweise der Methode zum Werfen der Würfel (`roll_dice method`) mithilfe von Pseudocode. Würfel müssen nach jedem Wurf, bei Anfang jeder Spielrunde und beim Einsetzen des Neu-Würfeln-Bonus neu geworfen werden:

```
1 roll_dice():
2     Werfe Würfel neu
```

Code 10: Methode zum Werfen der Würfel

4.2.5 Methode zur Überprüfung der freigespielten Belohnungen

Code 10 zeigt die Funktionsweise der Methode zur Überprüfung der freigespielten Belohnungen (`check_rewards` method) mithilfe von Pseudocode. Diese Methode überprüft jedes Mal, nachdem ein Kästchen in der Spielumgebung ausgefüllt worden ist, ob und welche Belohnung freigespielt wurde und fügt diese dem Inventar des Spielers hinzu:

```
1 check_rewards() :  
2     Überprüfe ob im gelben Feld Belohnungen freigeschaltet wurden  
3     Schalte im gelben Feld freigeschaltete Belohnungen frei  
4  
5     Überprüfe ob im blauen Feld Belohnungen freigeschaltet wurden  
6     Schalte im blauen Feld freigeschaltete Belohnungen frei  
7  
8     Überprüfe ob im grünen Feld Belohnungen freigeschaltet wurden  
9     Schalte im grünen Feld freigeschaltete Belohnungen frei  
10  
11     Überprüfe ob im orangenen Feld Belohnungen freigeschaltet wurden  
12     Schalte im orangenen Feld freigeschaltete Belohnungen frei  
13  
14     Überprüfe ob im lila Feld Belohnungen freigeschaltet wurden  
15     Schalte im lila Feld freigeschaltete Belohnungen frei  
16  
17     return Punktebelohnung
```

Code 11: Methode zur Überprüfung der freigespielten Belohnungen

4.2.6 Methode zur Generierung der Aktionsmaske

Code 11 zeigt Die Funktionsweise der Methode zur Generierung der Aktionsmaske (`valid_action_mask` method) mithilfe von Pseudocode. Einsen stehen für gültige Aktionen und Nullen für ungültige. Die Wahlwahrscheinlichkeit des Modells für ungültige Aktionen wird auf null gesetzt:

```
1 valid_action_mask() :  
2     Setze alle Werte auf Eins  
3  
4     Werte für bereits ausgefüllte gelbe Kästchen = 0  
5     Werte für gelbe Kästchen ohne passende Würfelergebnisse = 0  
6  
7     Werte für bereits ausgefüllte blaue Kästchen = 0
```

4 Implementierung

```
8   Werte für blaue Kästchen ohne passende Würfelergebnisse = 0
9
10  Werte für bereits ausgefüllte grüne Kästchen = 0
11  Werte für grüne Kästchen ohne passende Würfelergebnisse = 0
12
13  Werte für bereits ausgefüllte orangene Kästchen = 0
14  Werte für orangene Kästchen ohne passende Würfelergebnisse = 0
15
16  Werte für bereits ausgefüllte lila Kästchen = 0
17  Werte für lila Kästchen ohne passende Würfelergebnisse = 0
18
19  Werte für Aktionen mithilfe ungültiger Würfel = 0
20
21  if not Extra-Wahl or ungültige Extra-Wahl:
22      Werte von 122 bis 243 & Wert für das Aussetzen = 0
23  if gültige Extra-Wahl:
24      Werte von 0 bis 121 = 0
25      Wert für das Aussetzen = 1
26
27  if Neu-Würfeln-Boni <= 0 or Extra-Wahl:
28      Wert für Neu-Würfeln-Bonus = 0
29
30  if einer der Boni zum direkten Ankreuzen von Kästchen >= 1:
31      Alle Werte = 0
32
33  if Gelbes-Kreuz >= 1:
34      Werte von 0 bis 15 = 1
35  if Blaues-Kreuz >= 1:
36      Werte von 16 bis 27 = 1
37  if Grünes-Kreuz >= 1:
38      Werte von 28 bis 38 = 1
39  if (Orangene-Vier or Orangene-Fünf or Orangene-Sechs) >= 1:
40      Werte von 39 bis 49 = 1
41  if Lila-Sechs >= 1:
42      Werte von 50 bis 60 = 1
43
44  if Alle Werte außer 246 == 0:
45      Wert für ungültige Aktion = 1
46  else:
47      Wert für ungültige Aktion = 0
48
```

```
49 return Aktionsmaske
```

Code 12: Methode zur Generierung der Aktionsmaske

4.2.7 Methode zum Hinzufügen von Boni

Code 12 zeigt die Funktionsweise der Methode zum Hinzufügen freigeschalteter Boni (add_reward method) mithilfe von Pseudocode. Diese Methode wird von der Methode zur Überprüfung der freigespielten Belohnungen genutzt, um Boni dem Inventar des Spielers hinzuzufügen:

```
1 add_reward(Belohnungstyp):  
2     Inkrementiere Wert für Belohnungstyp
```

Code 13: Methode zum Hinzufügen freigeschalteter Boni

4.2.8 Methode zum Inkrementieren von Runden

Code 13 zeigt die Funktionsweise der Methode zum Inkrementieren des Runden-Systems (increment_rounds method) mithilfe von Pseudocode. Diese Methode ist dafür zuständig das Runden-System des Spiels voranzutreiben:

```
1 increment_rounds():  
2     if Extra-Wahl nach eigener Runde:  
3         Wahl vom Silbertablett = True  
4         Extra Wahl nach eigener Runde = False  
5         Setze alle Würfel auf gültig  
6         Würfele Würfel neu  
7         Setze drei zufällige Würfel auf ungültig  
8  
9     elif Wahl vom Silbertablett:  
10        if Extra-Wahl-Boni >= 1:  
11            Extra-Wahl nach Wahl vom Silbertablett = True  
12        else:  
13            Schalte Boni für erreichte Runde frei  
14            Werfe Würfel neu  
15            Wahl vom Silbertablett = False  
16            Setze alle Würfel auf gültig  
17  
18    elif Extra-Wahl nach Wahl vom Silbertablett:
```

```
19     Extra-Wahl nach Wahl vom Silbertablett = False
20     Werfe Würfel neu
21     Setze alle Würfel auf gültig
22     Schalte Boni für erreichte Runde frei
23
24     elif Wurf in Runde >= 3:
25         Rundenanzahl -= 1
26         Wurf in Runde = 1
27         Setze alle Würfel auf gültig
28         if Extra-Wahl-Boni >= 1:
29             Extra Wahl nach eigener Runde = True
30         else:
31             Wahl vom Silbertablett = True
32             Werfe Würfel neu
33             Setze drei zufällige Würfel auf ungültig
34     else:
35         Wurf in Runde += 1
36         Werfe Würfel neu
```

Code 14: Methode zum Inkrementieren des Runden-Systems

4.3 Künstliche Intelligenz

Die Implementierung besteht aus zwei Klassen. Eine davon ist die Künstliche Intelligenz. In diesem Kapitel werden die wesentlichen Methoden der Künstlichen Intelligenz mithilfe von Pseudocode erläutert. Für die Implementierung der Künstlichen Intelligenz wurde die Bibliothek Stable Baselines 3 [s. Unterabschnitt 2.2.2] verwendet.

4.3.1 Methode zum Anlernen des Modells

Code 14 zeigt die Funktionsweise der Methode zum Anlernen des Modells (`model_learn` method) mithilfe von Pseudocode. Es werden Hyperparameter festgelegt, welche genutzt werden, um ein MaskablePPO-Modell zu generieren und mithilfe der Spielumgebung zu trainieren. Alternativ kann ein bereits trainiertes Modell geladen und weiter trainiert werden:

```
1 model_learn(Hyperparameter):
2     Initialisiere Spielumgebungen
3     Modell = Initialisiere MaskablePPO-Modell mit Hyperparameter
4     if Modellname in Hyperparameter:
5         Modell = Lade Modell mit Modellname
6         Modell.Spielumgebungen = Initialisieren Spielumgebungen
7         Setze Entropie-Koeffizient
8     Lerne Modell an
9     Setze Gamma für Vorhersagen
10    Setze Entropie-Koeffizient für Vorhersagen
11    Lerne Modell erneut an
12    Speichere Modell
```

Code 15: Methode zu Anlernen des Modells

4.3.2 Methode zum Vorhersagen mithilfe des Modells

Code 15 zeigt die Funktionsweise der Methode zum Vorhersagen von Aktionen mithilfe des Modells (`model_predict` method) unter Verwendung von Pseudocode. Diese Methode ermöglicht es, mithilfe des Modells Vorhersagen über günstige Aktionen in einem gegebenen Zustand der Spielumgebung vorherzusagen und anhand dessen einen Spielverlauf zu simulieren:

```
1 model_predict(Schrittanzahl):  
2     Lade Modell  
3     Initialisiere Spielumgebungen und Historien  
4     Setze Beobachtungsraum  
5     for i in range(Schrittanzahl):  
6         Aktualisiere Aktionsmaske  
7         Aktion = Vorhersage der nächsten Aktion  
8         Führe Aktion in Spielumgebung aus  
9         Trage Werte in Historien ein  
10        Visualisiere Spielumgebung  
11        Plote Historien
```

Code 16: Methode zum Vorhersagen von Aktionen mithilfe des Modells

4.4 Anwendungsbeispiel für die Künstliche Intelligenz

Code 16 zeigt ein Anwendungsbeispiel des Projektes. Es wird ein Modell mit dem Namen `model_name` geladen und für 1110000 Schritte (x3 durch die `learn_model` methode) trainiert. Entsprechende Hyperparameter werden ebenfalls gesetzt. Daraufhin werden mithilfe des Modells Spieldurchläufe (für ungefähr 1000 Episoden) simuliert:

```
1 def main():  
2     model_learn(total_timesteps=1110000, ent_coef=0.1, gamma=1,  
3         model_name="maskableppo_ganzschoenclever_193avg_v3")  
4  
5     model_predict(n_envs=1, render=True, n_steps=40000,  
6         model_name="maskableppo_ganzschoenclever")
```

Code 17: Anwendungsbeispiel für die Künstliche Intelligenz

5 Ergebnisse

In diesem Kapitel werden die Ergebnisse des Projektes vorgestellt. Dabei werden sowohl der Verlauf der Implementierungsphase als auch die finalen Ergebnisse behandelt. Zunächst wurde ein Prototyp entwickelt, der die wesentlichen Funktionalitäten der Künstlichen Intelligenz und der Spielumgebung beinhaltet, um sicherzustellen, dass das Vorhaben im Rahmen der Vorgaben umsetzbar ist und um Einblicke in möglichen Problemstellungen des Projektes zu gewinnen. Danach wurde der Prototyp schrittweise erweitert, bis schließlich das komplette Spiel mit allen Funktionalitäten umgesetzt worden ist und die Künstliche Intelligenz vollständig trainiert werden konnte.

5.1 Trainingshistorie

5.1.1 Prototyp

Bei dieser Version handelt es sich um den ersten lauffähigen Prototypen der Umgebung. Es wurde nur eines der farbigen Felder (gelb) in der Spielumgebung implementiert. Das Modell kann mithilfe dieser Spielumgebung bereits trainiert werden und erzielt zum Teil gute Punktezahlen.

Abbildung 11 zeigt das die Punktestände des ersten erfolgreiche Trainings des Modells für 27000 Episoden:

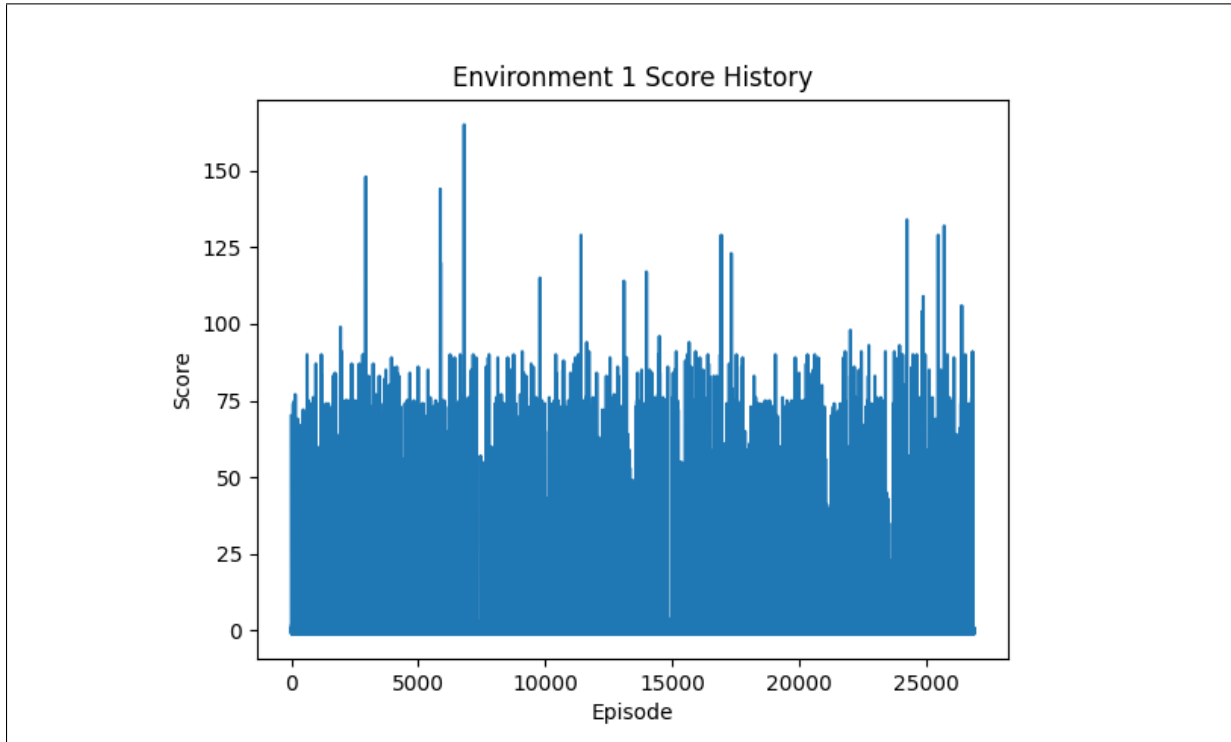


Abb. 11: Punktezahlen des ersten trainierten Modells für 27000 Episoden

Quelle: Eigene Darstellung

Abbildung 11 zeigt, dass das Modell innerhalb einer Episode (eines Spiels) des öfteren in einen Punktbereich von 70 kommt. Teilweise erzielt das Modell sogar über 150 Punkte. Fragwürdig ist hierbei, dass die maximale Punktezahl eigentlich bei 86 liegt und das Modell zu diesem Zeitpunkt dennoch teilweise Punktwerte erzielt, die höher liegen. Dies war darauf zurückzuführen, dass die Spielumgebung erzielte Punkte teilweise mehrfach anrechnete und somit mehr Punkte zu erspielen waren, als vorgesehen.

Abbildung 12 zeigt das die Punktestände des ersten erfolgreiche Trainings des Modells für 45 Episoden:

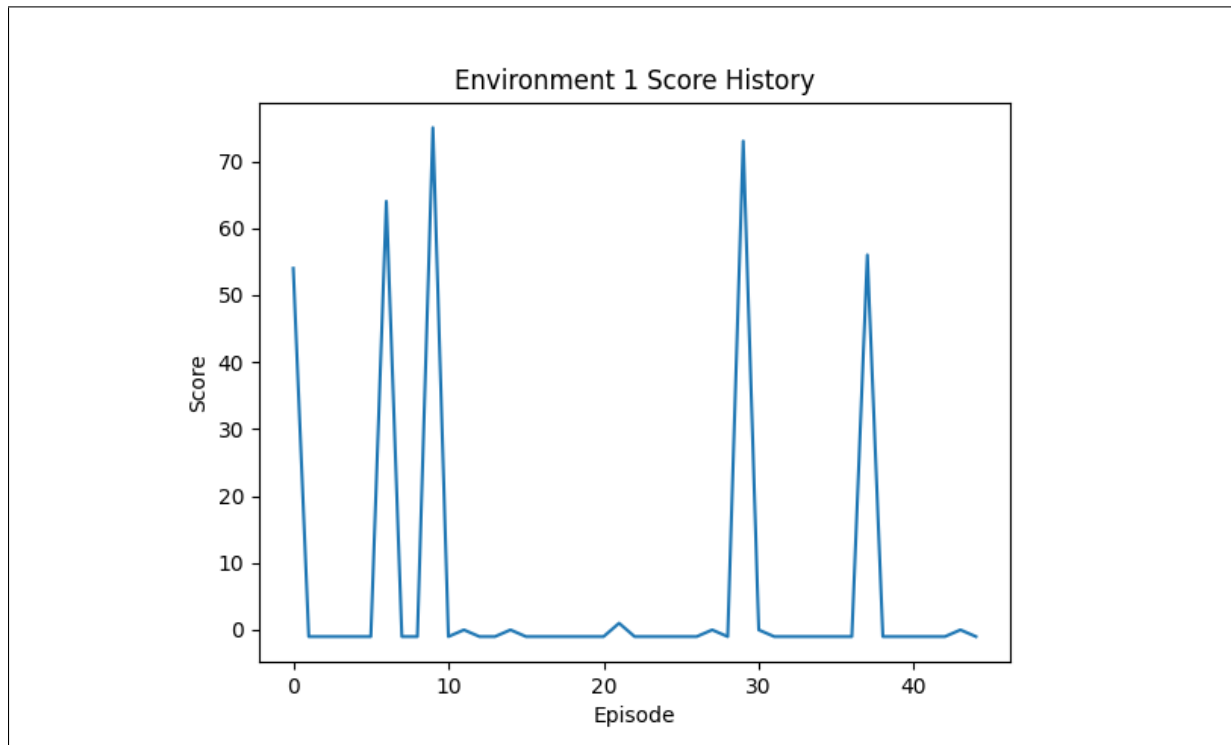


Abb. 12: Punktezahlen des ersten trainierten Modells für 45 Episoden

Quelle: Eigene Darstellung

Abbildung 12 zeigt, dass das Modell in dieser Version zwar teilweise eine gute Menge an Punkten erzielt, im Großteil der Spiele allerdings nur null oder einen bis zwei Punkte erzielt. Dies ist darauf zurückzuführen, dass die Spielumgebung bei dieser Version das Spiel unverzüglich beendet, wenn eine ungültige Aktion gewählt wird. Daraus ist zu schließen, dass das Modell überwiegend ungültige Aktionen wählt, da bei nur 100 ausgeführten Schritten bereits 45 Spiele abgeschlossen wurden. Vorgesehen sind zu diesem Zeitpunkt pro Spiel zehn Spielschritte. Somit wurden mehr als vier mal so viele Spiele abgeschlossen als unter optimalen Bedingungen vorgesehen.

5.1.2 Training mit und ohne Aktionsmaske

Abbildung 13 zeigt die erzielten Punkte des Modells ohne Aktionsmaske für 20 Episoden:

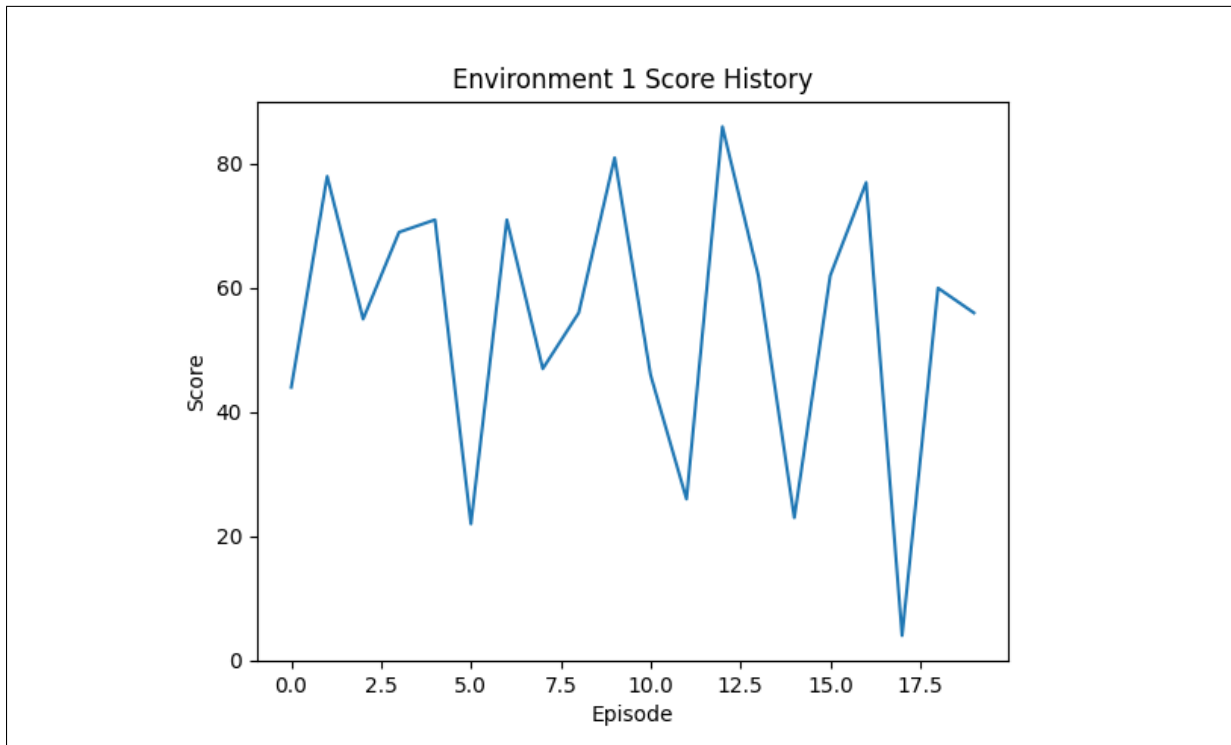


Abb. 13: Punktezahlen des Modells ohne Aktionsmaske für 20 Episoden

Quelle: Eigene Darstellung

Das anfängliche Verfahren bei dem das Spiel beendet wurde, sobald eine ungültige Aktion getätigt wurde, stellte sich als nicht geeignet heraus, da das Modell immer wieder ungültige Aktionen wählte, was zum Abbruch vieler Spielrunden führte. Daher wurde ein Trainingsverfahren mit negativer Belohnung bei ungültigen und positiver Belohnung bei gültigen Aktionen gewählt. Diese Belohnungen sind in Abbildung 13 nicht zu sehen, da sie herausgefiltert worden sind. Abbildung 13 zeigt nur die aufsummierten Belohnungen, welche 10 oder mehr Punkte betragen. Diese Belohnungen entsprechen den Belohnungen des gelben Feldes, welche mindestens 10 und maximal 20 Punkte betragen. Insgesamt zeigt sich eine verhältnismäßig gute Performanz bei der das Modell im Durchschnitt um die 50 Punkte erzielt. Bedenkt man, dass das Maximum bei 86 Punkten liegt und ungültige Aktionen getätigt werden können ist dieses Ergebnis gut. Das Modell erzielt im Durchschnitt ungefähr 60 Prozent der maximal erreichbaren Punktezahl.

Abbildung 14 zeigt die erzielten Punkte des Modells mit Aktionsmaske für 20 Episoden:

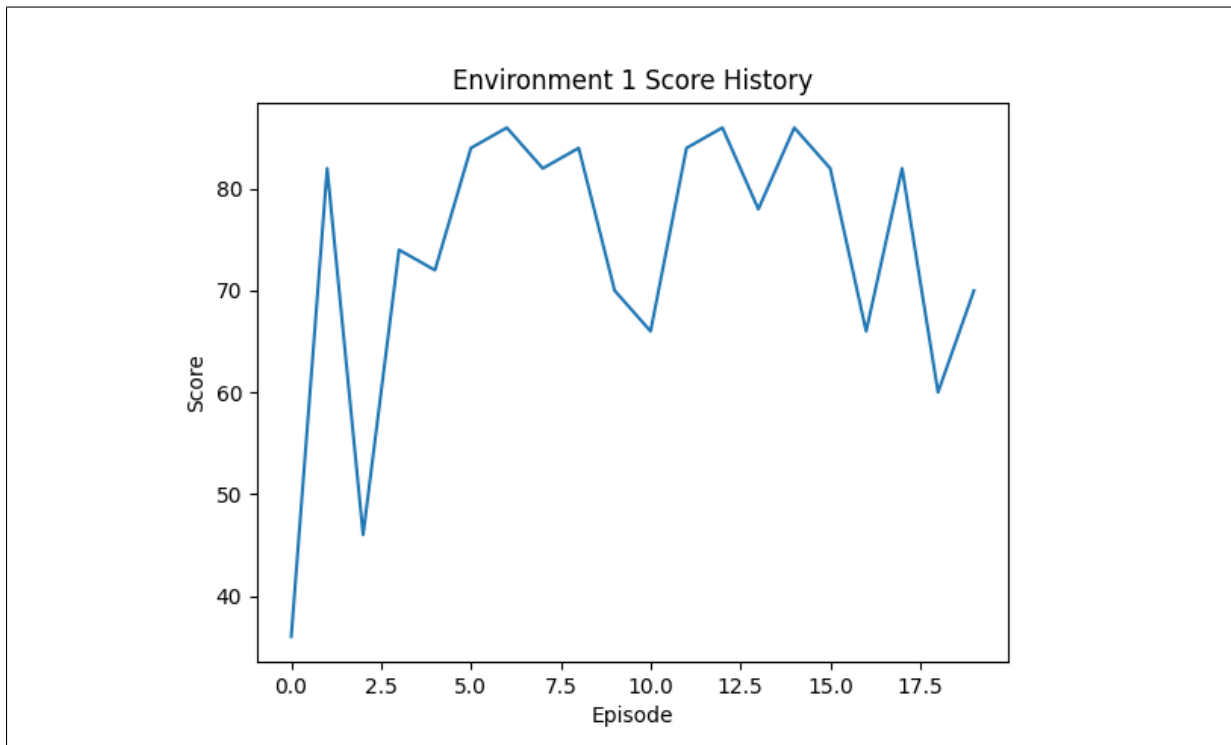


Abb. 14: Punktezahlen des Modells mit Aktionsmaske für 20 Episoden

Quelle: Eigene Darstellung

Abbildung 14 zeigt, dass die Performanz im Gegensatz zur Variante ohne Aktionsmaske zugenommen hat. Das Modell erzielt im Durchschnitt ungefähr 65 Punkte was einem Zuwachs von ungefähr einem Drittel entspricht. Die erzielten Punkte steigen somit von ungefähr 60 Prozent des Maximums auf 75 Prozent. Die Performanz des Vorgängermodells war passabel, den Umständen entsprechend gut, allerdings erzielt die Variante mit Aktionsmaske ohne größere Umstände ein wesentlich besseres Ergebnis. Deshalb wurde das Projekt ab diesem Zeitpunkt mit Aktionsmaske fortgesetzt.

5.1.3 Training mit zwei und vier Würfeln

Abbildung 15 zeigt die ungültigen Züge einer Spielsimulation mit zwei Würfeln für 20 Episoden:

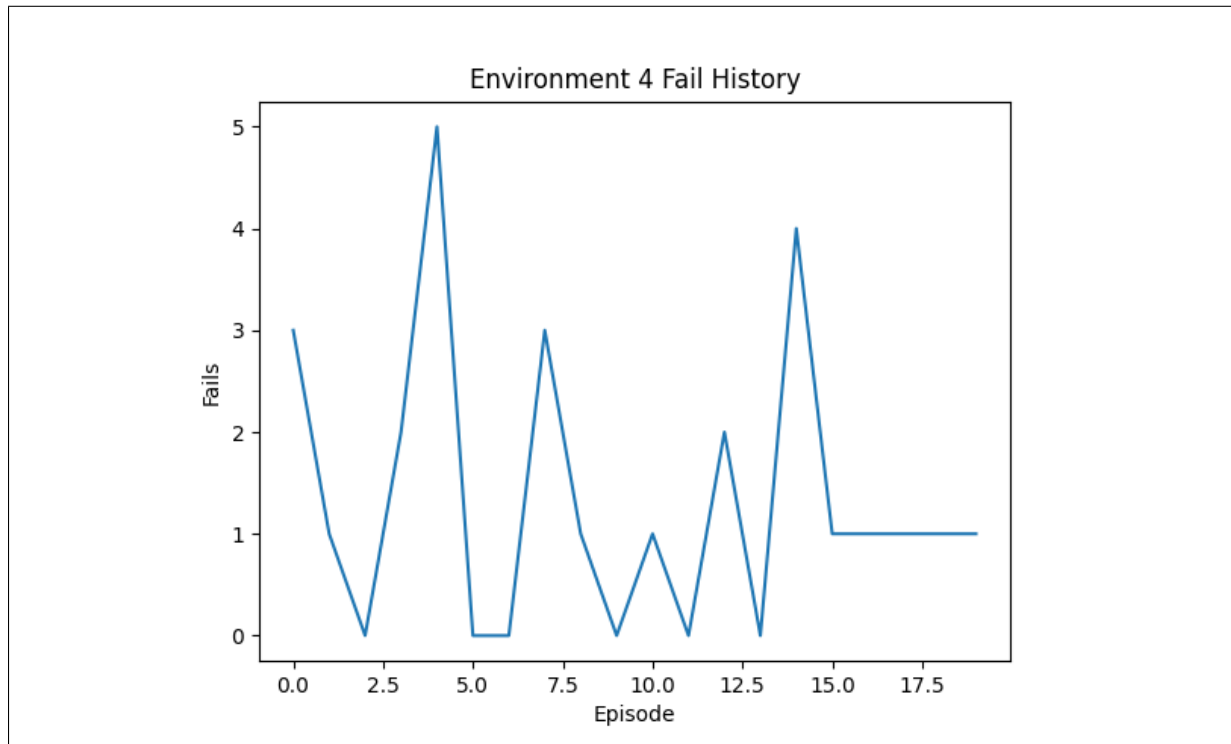


Abb. 15: Ungültige Züge des Modells unter Verwendung von zwei Würfeln für 20 Episoden
Quelle: Eigene Darstellung

Abbildung 15 zeigt, dass das Modell von null bis fünf ungültige Aktionen pro Spiel auswählt. Es ergibt sich ein Durchschnitt von ungefähr 1-2 ungültigen Zügen pro Spiel. Dies sollte nicht auftreten, wenn dem Modell gültige Aktionen zur Verfügung stehen, da die Aktionsmaske dafür sorgt, dass nur gültige Aktionen gewählt werden können. Das Problem entsteht, wenn dem Modell keine gültigen Aktionen zur Auswahl stehen. In diesem Fall wählt das Modell eine zufällige Aktion und berücksichtigt die Aktionsmaske nicht. Da in dieser Version nur zwei Würfel zur Verfügung stehen kommt es oft zu dem Fall, dass keiner der Würfel zu einem der Felder passt und somit kann keines der Felder ausgefüllt werden. Dadurch steht keine gültige Aktion zur Verfügung und es wird eine ungültige Aktion gewählt.

Abbildung 16 zeigt die ungültigen Züge einer Spielsimulation mit vier Würfeln für 20 Episoden:

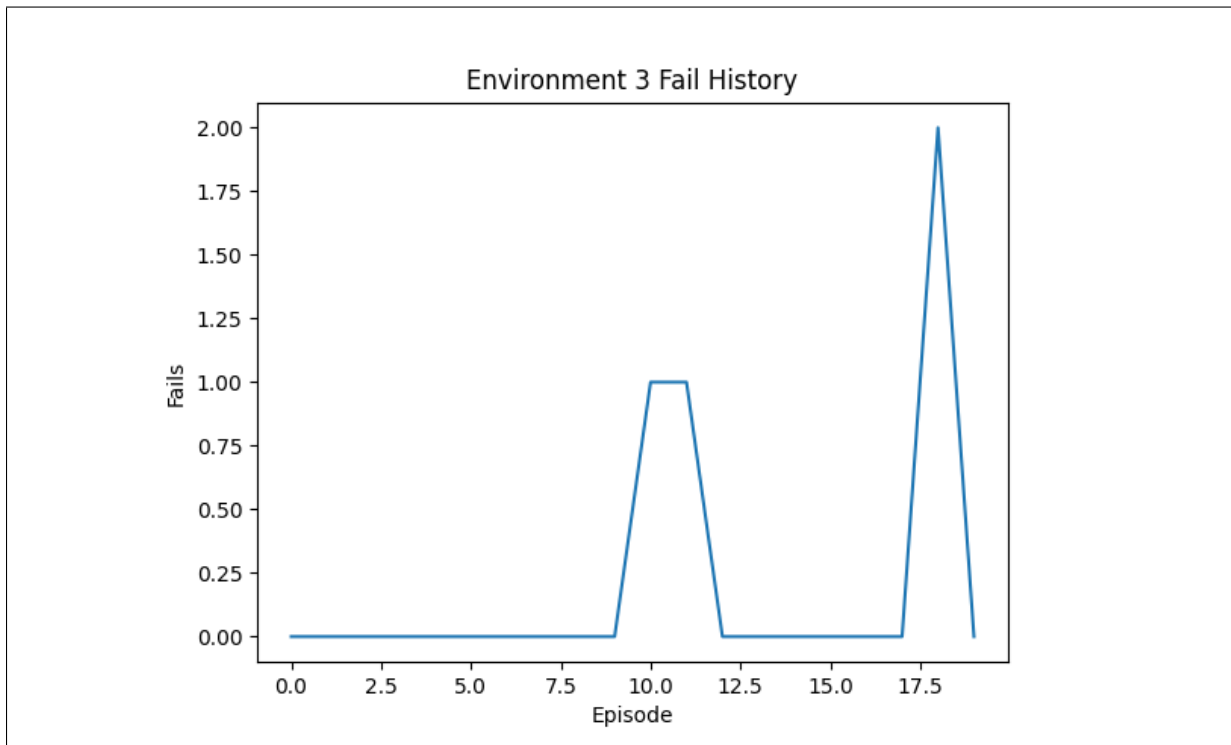


Abb. 16: Ungültige Züge des Modells unter Verwendung von vier Würfeln für 20 Episoden
Quelle: Eigene Darstellung

Abbildung 16 zeigt, dass die Anzahl an ungültigen Zügen im Gegensatz zur Variante mit zwei Würfeln drastisch abgenommen hat. Zwar kommt es vereinzelt immer noch zu Fällen bei denen das Modell ungültige Aktionen wählen muss, allerdings ist die Wahrscheinlichkeit dafür bei vier Würfeln deutlich geringer.

5.1.4 Optimierte Training

Abbildung 17 zeigt die erzielten Punkte des Modells nachdem eine Vielzahl an Optimierungen vorgenommen wurden:

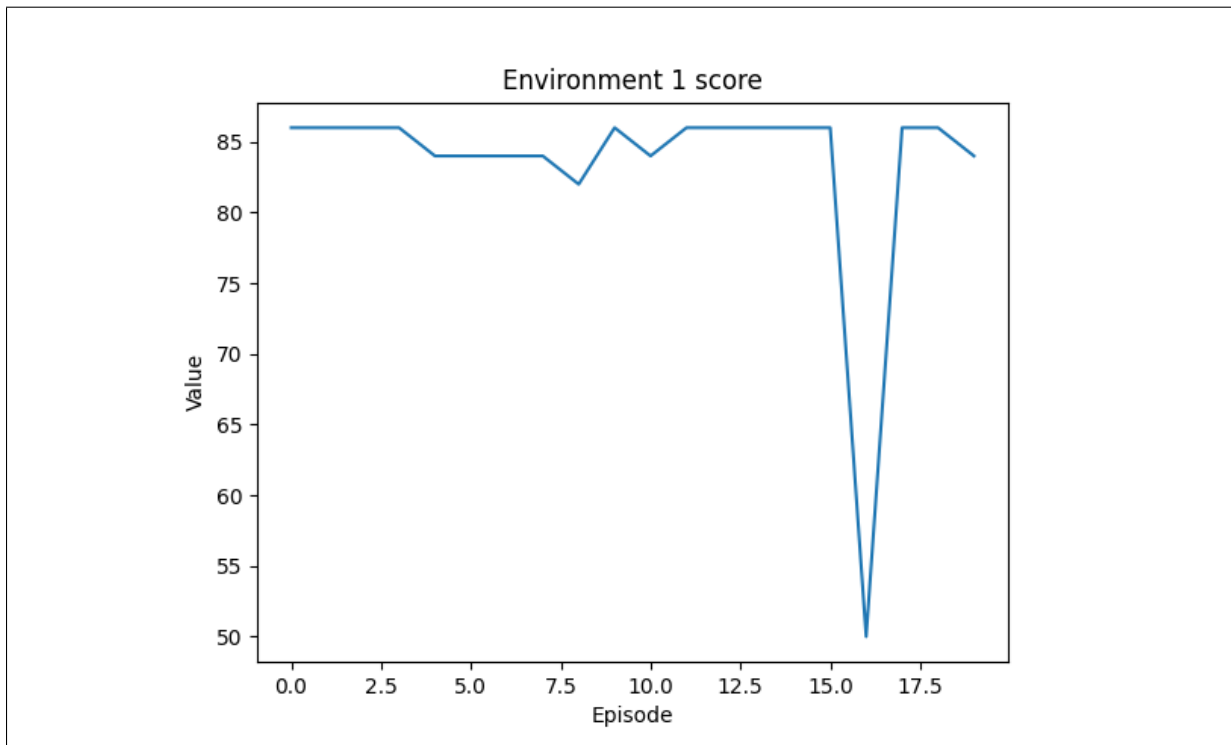


Abb. 17: Punktezahlen des Modells nach optimiertem Training für 20 Episoden
Quelle: Eigene Darstellung

Abbildung 17 zeigt, dass das Modell im Durchschnitt etwas mehr als 80 Punkte erzielt hat. Dies ist ein sehr guter Wert, da die maximale Punktezahl bei 86 liegt. Die maximale Punktezahl von 86 Punkten wurde innerhalb der zwanzig Episoden zehn mal erreicht.

5.2 Erstes Training mit allen Feldern und Boni

Abbildung 18 zeigt die Ergebnisse des ersten Trainings mit einer Spielumgebung bei der alle farbigen Felder und Boni implementiert worden sind:

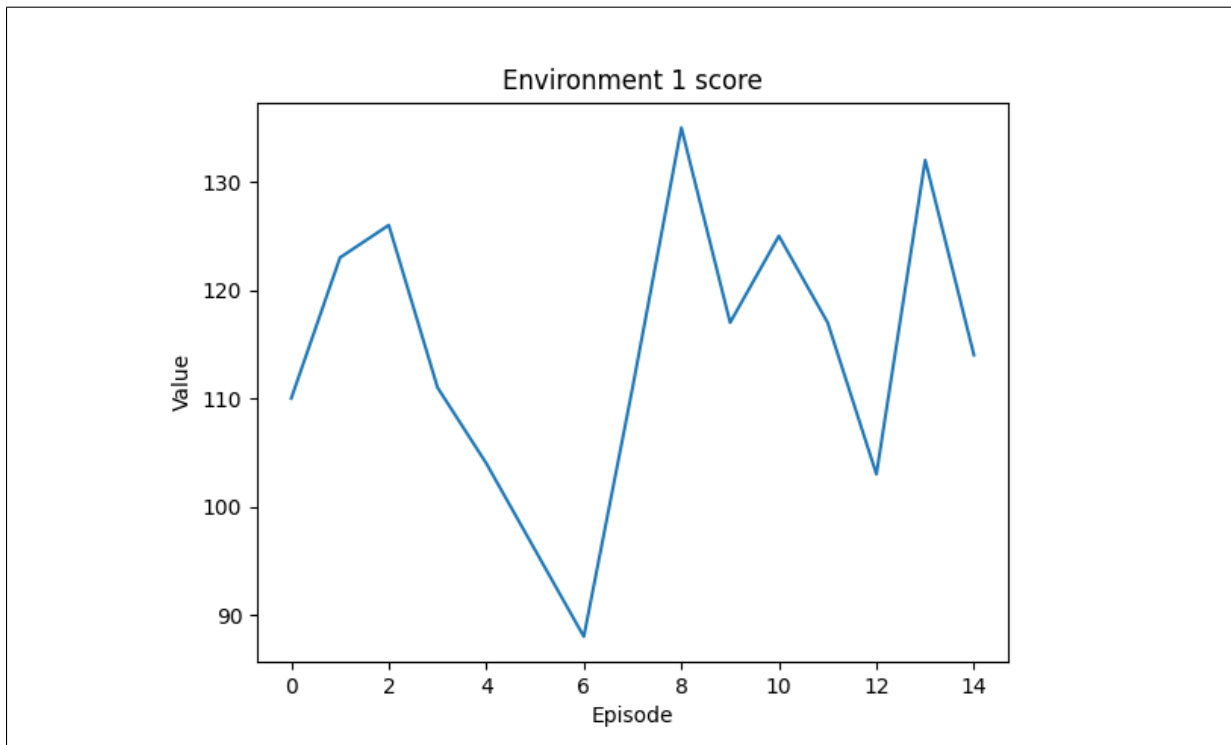


Abb. 18: Punktezahlen des erstes trainierten Modells mit allen Feldern

Quelle: Eigene Darstellung

Abbildung 18 zeigt, dass das Modell bereits vertretbare Werte erzielt, allerdings lässt sich an dieser Stelle noch viel optimieren. Im Schnitt erzielt es etwas mehr als 110 Punkte. So viele Punkte erzielt im Allgemeinen auch ein menschlicher Neueinsteiger, der das Spiel kaum kennt. Bei dieser Version des Spiels gibt es auch noch viele Bugs, die dazu führen, dass weniger Punkte erspielt werden, als es unter ordentlichen Bedingungen möglich wäre.

5.3 Finale Ergebnisse

5.3.1 Performanz

Abbildung 19 zeigt die erzielten Punktezahlen des Modells nach dem finalen Training für 1000 Episoden:

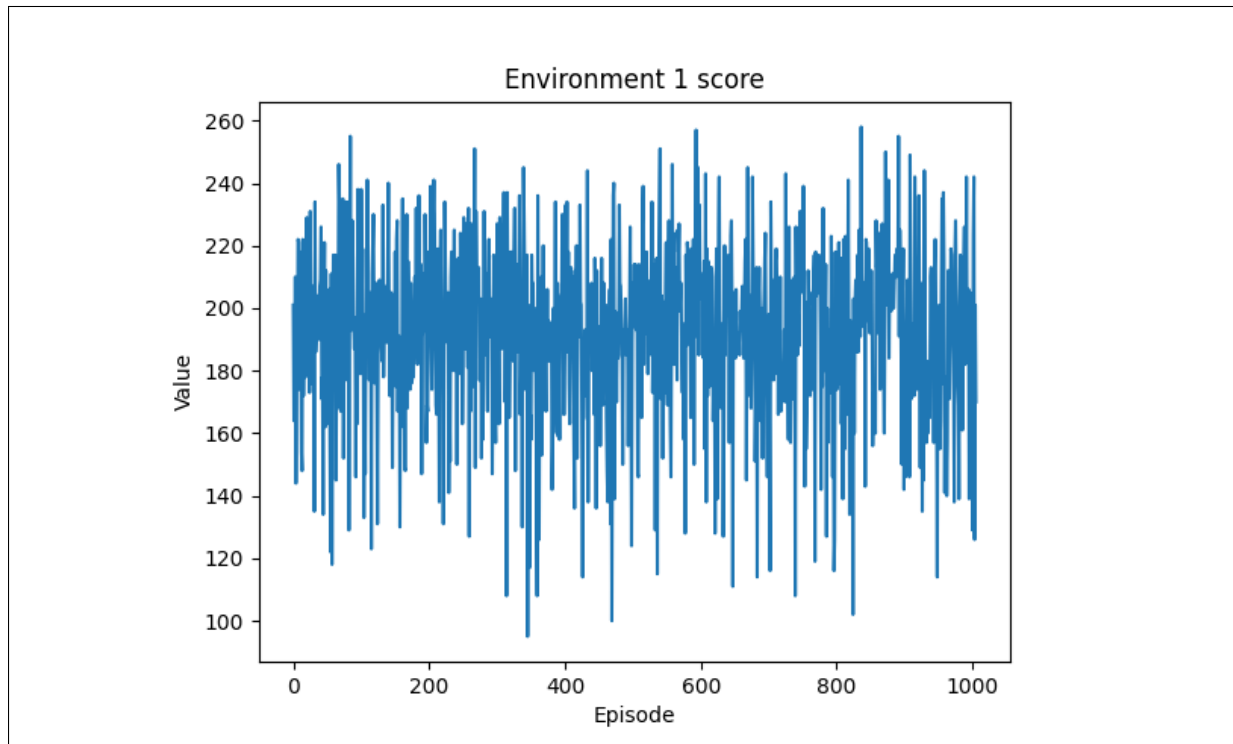


Abb. 19: Punktezahlen des finalen Modells für 1000 Episoden
Quelle: Eigene Darstellung

Abbildung 19 zeigt, dass das Modell über einen Zeitraum von mehr als 1000 Episoden eine durchschnittliche Punktezahl von ungefähr 205 Punkten erzielt. Das ist beachtlich, da menschliche Spieler in einem Versuch im Durchschnitt lediglich 160 Punkte erzielen konnten. Bei einem weiteren Test über 250 Episoden lag der Median bei 212 und die durchschnittliche Punktezahl bei 207.

Abbildung 20 zeigt die ungültigen Züge des Modells nach dem finalen Training für 1000 Episoden:

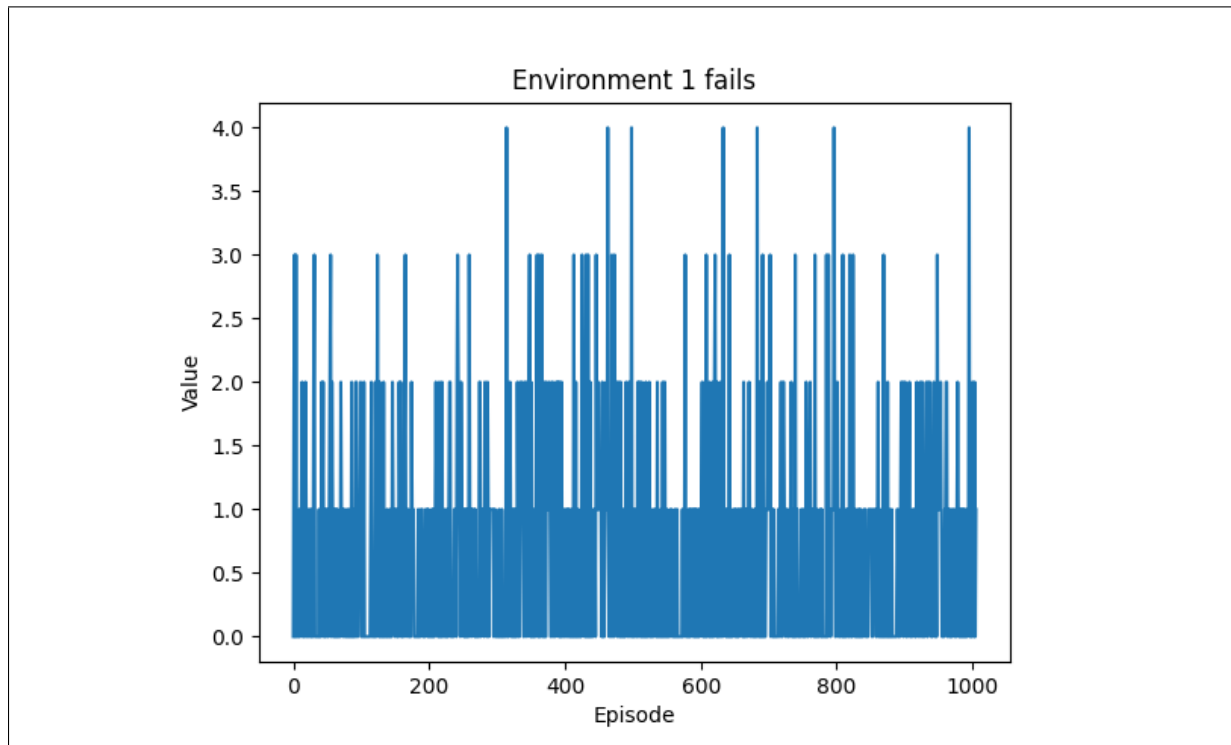


Abb. 20: Ungültige Züge des finalen Modells für 1000 Episoden
Quelle: Eigene Darstellung

Abbildung 20 zeigt, dass das Modell in den meisten Fällen maximal einen ungültigen Zug macht. Teilweise sind noch zwei ungültige Züge pro Episode zu erkennen und noch seltener drei oder vier. Angesichts dessen, dass das Modell pro Spielrunde ungefähr 40 Züge machte und häufig Würfel bei einer Wahl ungültig gemacht werden, ist das ein beachtliches Ergebnis. Das Modell lernte solche Fälle zu vermeiden.

Die Abbildungen 21 und 22 zeigen die Punktestände und ungültigen Züge des finalen Modells für 100 Episoden. Die Abbildungen sind anschaulicher als die Abbildungen für 1000 Episoden:

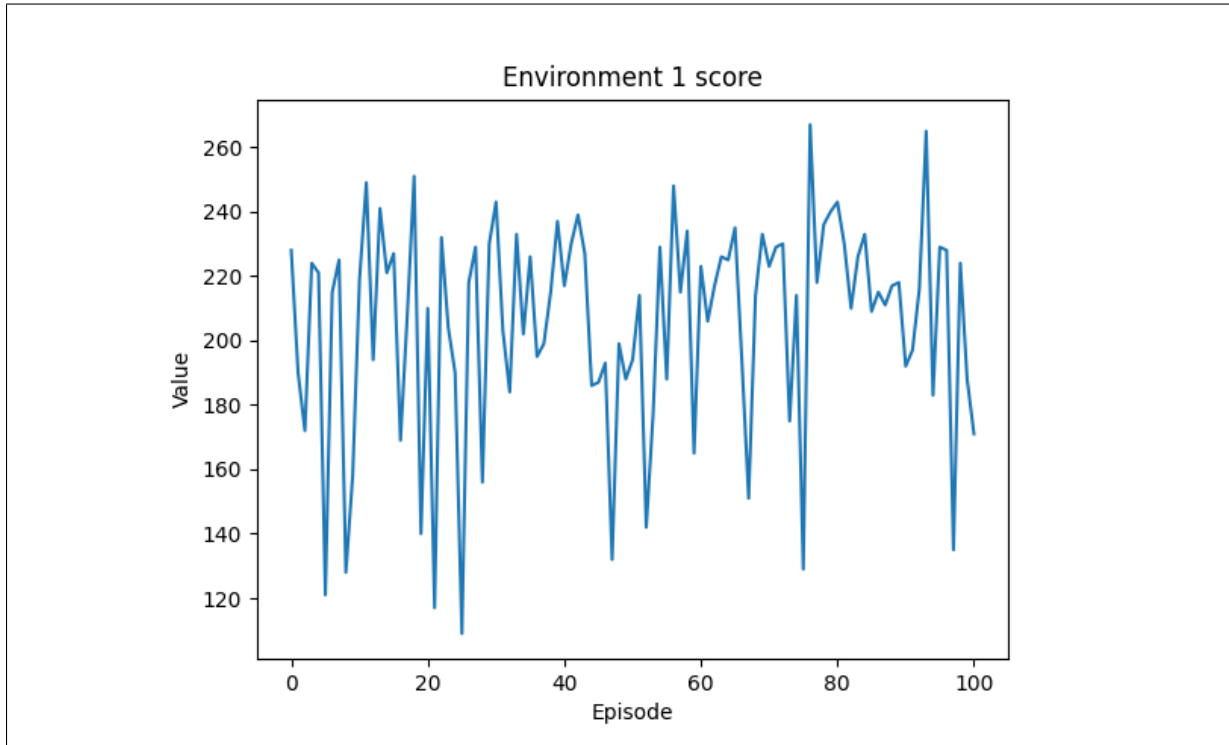


Abb. 21: Punktezahlen des finalen Modells für 100 Episoden
Quelle: Eigene Darstellung

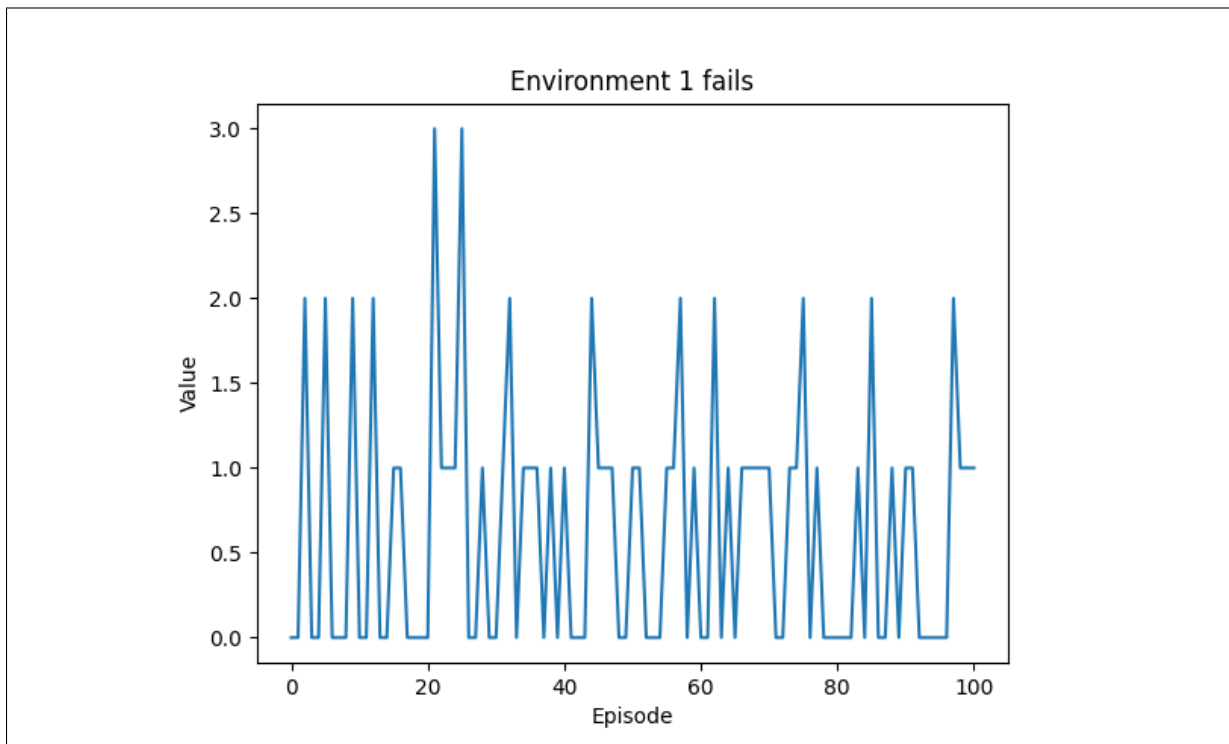


Abb. 22: Ungültige Züge des finalen Modells für 100 Episoden
Quelle: Eigene Darstellung

5.3.2 Hyperparameter

Die Hyperparameter für das finale Training sind:

- Ein neuronale Netz mit sieben Schichten von jeweils 1024 Neuronen.
- Ein Gamma von 1.
- Ein Entropie-Koeffizient von 0.1.
- Eine Lernrate von 0.0006.
- Eine Clip Range von 0.2.
- Eine Umgebungsanzahl von 32.
- Datenpakete mit 2048 Aktions-Zustands-Paaren, welche jeweils 5 Mal verwendet werden und dabei in Batches von 128 Schritten aufgeteilt werden.
- Ein Entropie-Koeffizient von 0 für das Verfestigen des gelernten Vorgehens.

Das Modell wurde in diesem Prozess zunächst für 2220000 Zeitschritte trainiert. Daraufhin wurde es mit dem Entropie-Koeffizient von 0 für weitere 1110000 Zeitschritte trainiert, um das gelernte Verhalten weiter zu verfestigen. Dieser Prozess wurde dann zwei weitere Male mit doppelt so vielen Zeitschritten wiederholt. Zunächst betrug die durchschnittliche Punktezahl des Modells annähernd 190, danach 201, danach 205 und nach weiteren Tests schließlich 208 Punkte. Die Standardabweichung im letzten Test lag bei 31 und der Median bei 214.

5.3.3 ChatGPT 4

ChatGPT 4 [s. Unterabschnitt 2.2.4] wurde verwendet, um den Prototypen des Projektes bis zu seinem ersten erfolgreichen Einsatz zu implementieren. Dabei hat es sich als besonders nützlich erwiesen, um einen lauffähigen Code mit den gewünschten, relativ standardisierten Methoden zu implementieren. Dies ist besonders bei der Erstellung eines ersten Prototypen, oder um die Machbarkeit des Projektes zu überprüfen, vorteilhaft. Im späteren Verlauf wurden von ChatGPT 4 generierte Inhalte nicht mehr direkt übernommen. ChatGPT 4 wurde in weiteren Verlauf lediglich benutzt, um Auskunft über geeignete oder bereits verwendete Technologien zu geben, Ideen zu sammeln oder Syntax zu erfragen. Auch für diese Aufgaben stellte sich ChatGPT 4 als sehr nützlich heraus. Solange die Fragen nicht zu spezifisch oder komplex waren lieferte ChatGPT 4 häufig gute und hilfreiche Antworten. Ist die Problemstellung zu komplex oder zu spezifisch, scheint ChatGPT 4 Schwierigkeiten zu haben eine vollständige oder geeignete Antwort zu liefern. Außerdem ist die Datenbank des ChatGPT 4 Modells auf den September 2021 datiert, was den Nutzen für neuere Sachverhalte erschwert.

6 Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, eine Künstliche Intelligenz, für das Brettspiel "*Ganz schön clever*", zu entwickeln und den Entwicklungs- sowie Trainingsprozess zu evaluieren. Diese Künstliche Intelligenz wurde erfolgreich entwickelt und kann das Brettspiel "*Ganz schön clever*" gut spielen. Gut bedeutet, dass sie in der Lage ist im Durchschnitt eine hohe Punktezahl zu erzielen. Zunächst wurde dafür ein Prototyp entwickelt, welcher dann schrittweise erweitert wurde, um die gesamte Komplexität des Spiels zu erfassen und um es gut spielen zu können.

Das Brettspiel "*Ganz schön clever*" ist ein Würfelspiel, welches eine hohe Komplexität aufweist. Diese kommt vor allem durch die vielen Aktionsmöglichkeiten des Spielers und die multiplen Zusammenhängen innerhalb des Belohnungssystems zustande. Außerdem weist es eine hohe Stochastizität auf, welche die Komplexität weiter erhöht.

Die Spielumgebung und die Künstliche Intelligenz mussten vollständig implementiert werden. Dies geschah mithilfe von Bibliotheken wie Stable Baselines 3 und Gymnasium. Aus zeitlichen Gründen wurde lediglich die Ein-Spieler-Variante des Spiels implementiert. Für die Implementierung der Künstlichen Intelligenz wurde Deep Reinforcement Learning verwendet. Das neuronale Netz der Künstlichen Intelligenz ist ein Multilayer Perceptron mit sieben Schichten mit jeweils 1024 Neuronen, die eine ReLU-Aktivierungsfunktion verwenden.

Während der Implementierungsphase kam es zunächst zu einem Problem, bei dem die Künstliche Intelligenz es nur schwer selbstständig schaffte, gültige Aktionen zu wählen, welche nicht zum Abbruch des Spiels geführt haben. Das Vorgehen, bei dem das Spiel nach Wahl einer ungültigen Aktion abgebrochen wurde, wurde nach einigen Tagen verworfen, da es der Künstlichen Intelligenz das Lernen erschwerte, weil die meisten Spiele bereits nach wenigen Schritten endeten und die Künstliche Intelligenz so wenig relevante Daten sammeln konnte. Außerdem war die Bewertung der Performanz ebenfalls, durch die vielen Spielabbrüche, erschwert. Daraufhin wurde zunächst ein Verfahren implementiert, bei dem das Spiel bei ungültigen Aktionen nicht sofort abgebrochen, sondern negativ belohnt wurde. Dieses Vorgehen stellte sich als gut geeignet heraus, wurde aber schnell von einem Vorgehen, welches eine Aktionsmaske verwendet abgelöst. Im folgenden wurde eine Aktionsmaske implementiert, welche gewährleistet, dass nur gültige Aktionen gewählt werden können. Somit musste die Künstliche Intelligenz nicht mehr selbstständig lernen, welche Aktionen gültig sind. Die Performanz stieg mit diesem Vorgehen von vorher durchschnittlich ungefähr 60 Prozent der Maximalpunktzahl auf 75 Prozent.

Zudem ergaben sich Schwierigkeiten bei der Erweiterung des Runden-Systems. Fast die gesamte Spielumgebung musste, an das neue Runden-System, angepasst werden. Dieser Aufwand wäre voraussichtlich deutlich geringer ausgefallen, wenn das Runden-System von Anfang an vollständig implementiert worden wäre.

Das finale Training der Künstlichen Intelligenz erfolgte in von drei Phasen. Zunächst wurde die KI in 2.220.000 Trainingsschritten vortrainiert. Anschließend wurde sie in 1.110.000 Trainingsschritten, mit einem verringerten Entropie-Koeffizienten, was einer geringeren Exploration entspricht, nachtrainiert, um das gelernte Verhalten zu verfestigen. Dieses Training erfolgte daraufhin mit der doppelten Zeit weitere zwei Male. Das Ergebnis des Trainings war eine Künstliche Intelligenz, welche durchschnittlich 208 Punkte im Spiel erzielt. Im Vergleich dazu erzielten menschliche Spieler in einem Test im Durchschnitt lediglich 160 Punkte. Für die durchschnittlichen Punktezahlen, wurden die erreichten Punkte einer abgeschlossenen Spielumgebung abgespeichert, anschließend aufaddiert und durch ihre Anzahl geteilt. Die Standardabweichung lag im letzten Test bei 31 und der Median bei 214. Einmalig erzielte die Künstliche Intelligenz 282 Punkte, was laut der offiziellen Punkteskala von Schmidt Spiele dem höchsten Level an Performanz entspricht [noaa].

Besonders hilfreich war es, einen Prototypen zu bauen und diesen anschließend schrittweise auszubauen. Dies hat dabei geholfen leichter einen Überblick über die wichtigsten Aspekte des Vorhabens zu gewinnen.

Die Implementierung der Künstlichen Intelligenz selbst gestaltete sich dank Bibliotheken wie Gymnasium und Stable Baselines 3 vergleichsweise einfach, brachte aber dennoch eine Vielzahl interessanter und nützlicher Erkenntnisse mit sich. Besonders hervorzuheben ist dabei die Anpassung von Hyperparametern, welche zum Teil zu sehr unterschiedlichen Ergebnissen geführt hat.

Der aufwändigste Teil des Projektes war die Implementierung der Spielumgebung selbst. Das Spiel hat viele kleine Interaktionen, welche zum Teil ineinander verschachtelt sind und es gestaltete sich zum Teil schwierig alle Zusammenhänge zu überschauen und zu gewährleisten, dass Änderungen an einer Funktion andere nicht negativ beeinflusst. Es sind auch vielerlei Bugs aufgetreten, die gefunden und behoben werden mussten. Der gravierendste Fehler war es, das Runden-System nicht von vornherein zu implementieren, sondern es später einzufügen, als die meisten anderen Aspekte des Spiels bereits implementiert waren. Dies führte dazu, dass viele Aspekte der Spielumgebung neu überdacht und überarbeitet werden mussten. Für die Zukunft bleibt zu sagen, dass besonders am Anfang ein Fokus auf alle grundlegenden Aspekte der Problemstellung von Vorteil ist. Sobald das Projekt mit all seinen Grundkomponenten lauffähig ist, sollte es um weniger grundlegende Aspekte erweitert werden. Der Einfluss einer Änderung des Runden-Systems wurde unterschätzt.

Für die Weiterarbeit am Projekt bietet sich die Implementierung einer grafischen Visualisierung des Lern- oder Vorhersageprozesses an. Außerdem lässt sich das Modell sicherlich mithilfe anderer Algorithmen, Trainingsverfahren oder Hyperparameter weiter optimieren. Was im Rahmen der Arbeit ebenfalls nur bedingt erfolgte, ist eine Analyse der Spielstrategien von Modellen und

wie sich unterschiedliche Strategien auf die Performance auswirken. Zwar wurden Variablen hinzugefügt, welche messen, wie oft Kästchen in bestimmten Feldern ausgefüllt wurden, allerdings lässt sich hier noch viel mehr machen. Ebenfalls wäre es sinnvoll, Tests für das Projekt zu implementieren, um sicherzustellen, dass alle Aspekte der Umgebung wie vorhergesehen funktionieren und zusammenarbeiten. Zwar erfolgten Tests durch die Auswertung von Umgebungsvariablen und Print-Statements, aber es fehlt dem Projekt eine fundierte Umsetzung professioneller Testverfahren.

Literaturverzeichnis

- [Alp22] ALPAYDIN, Ethem: Maschinelles Lernen. Version: Januar 2022. <http://dx.doi.org/10.1515/9783110740196>. In: *Maschinelles Lernen*. De Gruyter Oldenbourg, Januar 2022. – DOI 10.1515/9783110740196. – ISBN 978–3–11–074019–6
- [noaa] 49340_Ganz_schoen_Clever_ANLEITUNG_FINAL9.
- [noab] *Gymnasium Documentation*. <https://gymnasium.farama.org/index.html>
- [noac] *Künstliche Intelligenz - Weltweit | Statista Marktprognose*. <https://de.statista.com/outlook/tmo/kuenstliche-intelligenz/weltweit>
- [noad] *Matplotlib — Visualization with Python*. <https://matplotlib.org/>
- [noae] *Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations — Stable Baselines3 2.2.0a8 documentation*. <https://stable-baselines3.readthedocs.io/en/master/index.html>
- [noa23a] *Künstliche Intelligenz in Videospielen*. <https://www.doag.org/de/home/news/kuenstliche-intelligenz-in-videospielen/>. Version: August 2023
- [noa23b] *Maschinelles Lernen - Weltweit | Statista Marktprognose*. <https://de.statista.com/outlook/tmo/kuenstliche-intelligenz/maschinelles-lernen/weltweit>. Version: November 2023
- [RA23] RIS-ALA, Rafael: *Fundamentals of Reinforcement Learning*. Cham : Springer Nature Switzerland, 2023. <http://dx.doi.org/10.1007/978-3-031-37345-9>. <http://dx.doi.org/10.1007/978-3-031-37345-9>. – ISBN 978–3–031–37344–2 978–3–031–37345–9
- [Sew19] SEWAK, Mohit: *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*. Singapore, SINGAPORE : Springer Singapore Pte. Limited, 2019 <http://ebookcentral.proquest.com/lib/hs-coburg/detail.action?docID=5802517>. – ISBN 9789811382857
- [SWD⁺17] SCHULMAN, John ; WOLSKI, Filip ; DHARIWAL, Prafulla ; RADFORD, Alec ; KLIMOV, Oleg: *Proximal Policy Optimization Algorithms*. <http://dx.doi.org/10.48550/arXiv.1707.06347>. Version: August 2017. – arXiv:1707.06347 [cs]
- [tag] TAGESSCHAU.DE: *ChatGPT wächst rasant und startet Bezahlversion*. <https://www.tagesschau.de/wirtschaft/digitales/chatgpt-wachstum-bezahlangebot-abo-101.html>

Glossar

Agent

Ein Agent ist die ausführende Instanz der Künstlichen Intelligenz. Er nimmt die Umgebung wahr und führt Aktionen in dieser aus.

Bonusrunde

Eine Bonusrunde ist eine Runde bei der der Rundenablauf nicht wie üblich inkrementiert wird. Diese Einteilung ist notwendig, da bei Wahlen mit Boni andere Regeln gelten, als bei Wahlen nach dem eigenen Wurf oder nach Wahlen vom Silbertablett eines Mitspielers.

Extra-Wahl

Eine Extra-Wahl beschreibt eine Wahl mithilfe der Extra-Wahl-Boni oder vom Silbertablett eines Mitspielers.

Feld

Ein Feld ist eines der farbigen Felder (gelb, blau, grün, orange, lila) auf dem Spielbrett von *"Ganz schön clever"*.

Kästchen

Ein Kästchen beschreibt ein Rechteck des Spielbretts von *"Ganz schön clever"*, welches ausgefüllt werden kann, wenn bestimmte Bedingungen erfüllt sind.

Modell

Ein Modell beschreibt die Entität der Künstlichen Intelligenz, welche im Stande ist, Vorhersagen zu treffen

normale Wahl

Eine normale Wahl beschreibt die Wahl eines Würfels nach einem eigenen Wurf oder eine Wahl mithilfe eines der Boni, welcher es erlauben direkt eines der Kästchen in einem der farbigen Felder auszufüllen.

ungültige Aktion

Eine ungültige Aktion ist eine Aktion, die im normalen Spielablauf nicht stattfinden würde. Ungültige Aktionen treten auf, wenn dem Modell keine gültige Aktion zur Auswahl steht.

ungültiger Würfel

Ein ungültiger Würfel ist ein Würfel, der nicht mehr zur Wahl steht und nicht mehr geworfen wird.

Wahl

Eine Wahl beschreibt den Vorgang bei dem ein Würfel oder eine Boni gewählt wird, um eines der Kästchen auf dem Spielbrett auszufüllen.

A1 Code für die Attribute der Spielumgebung

Code 17 zeigt die Klassenattribute, die für das bilden einer Punktestandhistorie relevant sind:

```
1 self.score
2 self.score_history
3 self.initialized
```

Code 18: Klassenattribute für die Nachvollziehbarkeit von Punkteständen

Code 18 zeigt die Klassenattribute der Flags für die Belohnungen der farbigen Felder. Sind diese gesetzt und sind die entsprechenden Kästchen für die Belohnung ausgefüllt, wurde die Belohnung bereits ausgeschüttet und wird es nicht erneut, bis die Attribute nach dem Spiel zurückgesetzt werden:

```
1 self.yellow_reward_flags = {"row": [False] * 4, "col": ...}
2 self.blue_reward_flags = {"row": [False] * 3, "col": ...}
3 self.blue_count_reward_flags = [False] * 12
4 self.green_reward_flags = [False] * 11
5 self.orange_reward_flags = [False] * 11
6 self.purple_reward_flags = [False] * 11
```

Code 19: Klassenattribute für Belohnungsflags

Code 19 zeigt die Klassenattribute für Punktestände der einzelnen farbigen Felder. Die Punktewerte werden aufaddiert, sobald Punkte im entsprechenden Feld erspielt worden sind und am Ende genutzt, um den Wert der Fuchs-Boni zu bestimmen [siehe Unterabschnitt 2.1.1]:

```
1 self.yellow_field_score
2 self.blue_field_score
3 self.green_field_score
4 self.orange_field_score
5 self.purple_field_score
```

Code 20: Klassenattribute für erreichte Punktestände der einzelnen farbigen Felder

Code 20 zeigt die Klassenattribute für die Anzahl an gewählten Kästchen in den verschiedenen farbigen Feldern. Wenn ein Kästchen in einem der Felder ausgefüllt wird, wird der Wert des entsprechenden Attributes inkrementiert. Diese Attribute dienen nicht dem Spielablauf selbst, sondern der Nachvollziehbarkeit der Strategie des Modells:

```
1 self.picked_yellow
2 self.picked_blue
3 self.picked_green
4 self.picked_orange
5 self.picked_purple
```

Code 21: Klassenattribute für die Anzahl an gewählte Kästchen innerhalb der farbigen Feldern

A2 Pseudocode für Methoden der Implementierung

Code 21 zeigt die Funktionsweise der Funktion zur Visualisierung der Spielumgebung (`render method`) mithilfe von Pseudocode. Die Visualisierung dient einer verbesserten Nachvollziehbarkeit der Ereignisse innerhalb der Spielumgebung:

```
1 render():  
2     Zeige alle relevanten Attribute und Merkmale der Umgebung an
```

Code 22: Methode zur Visualisierung der Spielumgebung

Code 22 zeigt die Funktionsweise der Methode zur Generierung des Beobachtungsraumes (`_get_obs method`) mithilfe von Pseudocode. Diese Methode vereinfacht es, den Beobachtungsraum zu generieren, welcher jedes Mal benötigt wird, wenn ein Schritt in der Umgebung ausgeführt wird:

```
1 _get_obs():  
2     Erstelle Numpy Arrays für farbige Felder  
3     Erstelle Numpy Array für Würfelergebnisse  
4     Erstelle Numpy Array für ungültige Würfel  
5     Füge Arraywerte für Boni und Rundenzahl hinzu  
6     Beobachtungsraum = Verbinde alle Arrays miteinander  
7  
8     return Beobachtungsraum
```

Code 23: Methode zur Generierung des Beobachtungsraumes

Code 23 zeigt die Funktionsweise der Methode zur Initialisierung der Spielumgebungen (`_init_envs` method) mithilfe von Pseudocode. Mit dieser Methode werden Spielumgebungen initialisiert und einer Vektorumgebung zugewiesen. Diese Vektorumgebung ermöglicht es mehrere Spielumgebungen gleichzeitig zu bearbeiten. Außerdem werden Variablen für die Nachvollziehbarkeit der Abläufe innerhalb der Spielumgebung (Historien) initialisiert:

```
1 _init_envs(Anzahl, Punktestände, Fehlversuche):
2     _init():
3         Spielumgebung = Initialisieren eine Spielumgebung
4         Setze Aktionsmasker für die Spielumgebung
5         return Spielumgebung
6     Initialisiere Vektorumgebung(Anzahl, _init)
7     if not Punktestände and not Fehlversuche:
8         return Vektorumgebung
9     if Punktestände and not Fehlversuche:
10         Erstelle Variablen für Punkteständehistorie
11     if not Punktestände and Fehlversuche:
12         Erstelle Variablen für Fehlversuchehistorie
13     if Punktestände and Fehlversuche:
14         Erstelle Variablen für Punkteständehistorie
15         Erstelle Variablen für Fehlversuchehistorie
16     return Vektorumgebung, Variablen für Historien
```

Code 24: Methode zur Initialisierung der Spielumgebungen

Code 24 zeigt die Funktionsweise der Methode zum Anwenden der Aktionsmaske (`mask_fn` method) mithilfe von Pseudocode. Die Methode dient dazu dem Modell die Aktionsmasken der einzelnen Spielumgebungen zu übergeben:

```
1 mask_fn(Spielumgebung):
2     Caste Spielumgebung in benutzerdefinierte Spielumgebung
3     return Aktionsmaske der Spielumgebung
```

Code 25: Methode zum Anwenden der Aktionsmaske

Code 25 zeigt die Funktionsweise der Methoden zum Erstellen von Einträgen und Historien (make_fail_entries method, make_score_entries method, make_fail_history_entry method, make_score_history_entry method) mithilfe von Pseudocode. Diese Methoden erstellen Einträge und Historien von erzielten Punkteständen und der Anzahl getätigter ungültiger Aktionen innerhalb abgeschlossener Spiele:

```
1 make_fail_entries(Punktebelohnungen, Anzahl, Fehlversuche):
2     if Punktebelohnung < 0:
3         Inkrementiere Fehlversuche der Umgebung
4
5 make_score_entries(Punktebelohnungen, Anzahl, Punktestände):
6     if Punktebelohnung > 0:
7         Addiere Punktebelohnung zum Punktestand der Umgebung
8
9 make_fail_history_entry(Fehlversuche, Fehlversuchshistorie)
10    if Umgebung terminiert:
11        Hänge Fehlversuche an Fehlversuchshistorie an
12
13 make_score_history_entry(Punktestände, Punktehistorie)
14    if Umgebung terminiert:
15        Hänge Punktestand der Umgebung Punktehistorie an
```

Code 26: Methoden zum Erstellen von Einträgen für Historien

Code 26 zeigt die Funktionsweise der Methode zum Plotten von Historien (plot_history method) mithilfe von Pseudocode. Diese Methode visualisiert die generierten Historien:

```
1 plot_history(Historie):
2     Plote jeden Eintrag der Historie
3     Setze Titel
4     Setze Labels
5     Zeige Grafik an
```

Code 27: Methode zum Plotten von Historien

Persönliche Angaben / Personal details

Schubert, Sander

Familienname, Vorname / Surnames, given names

02.12.1994

Geburtsdatum / Date of birth

Informatik Bachelor

Studiengang / Course of study

01550217

Matrikelnummer / Student registration number

Eigenständigkeitserklärung***Declaration***

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und noch nicht anderweitig für Prüfungszwecke vorgelegt habe. Ich habe keine anderen als die angegeben Quellen oder Hilfsmittel benutzt. Die Arbeit wurde weder in Gänze noch in Teilen von einer Künstlichen Intelligenz (KI) erstellt, es sei denn, die zur Erstellung genutzte KI wurde von der zuständigen Prüfungskommission oder der bzw. dem zuständigen Prüfenden ausdrücklich zugelassen. Wörtliche oder sinngemäße Zitate habe ich als solche gekennzeichnet.

Es ist mir bekannt, dass im Rahmen der Beurteilung meiner Arbeit Plagiatserkennungssoftware zum Einsatz kommen kann.

Es ist mir bewusst, dass Verstöße gegen Prüfungsvorschriften zur Bewertung meiner Arbeit mit „nicht ausreichend“ und in schweren Fällen auch zum Verlust sämtlicher Wiederholungsversuche führen können.

I hereby certify that I have written this thesis independently and have not submitted it elsewhere for examination purposes. I have not used any sources or aids other than those indicated. The work has not been created in whole or in part by an artificial intelligence (AI), unless the AI used to create the work has been expressly approved by the responsible examination board or examiner. I have marked verbatim quotations or quotations in the spirit of the text as such.

I am aware that plagiarism detection software may be used in the assessment of my work.

I am aware that violations of examination regulations can lead to my work being graded as "unsatisfactory" and, in serious cases, to the loss of all repeat attempts.

Unterschrift Studierende/Studierender / Signature student

, den 22.11.2023

Ort, Datum / Place, date