Hochschule für angewandte Wissenschaften Coburg
Fakultät Elektrotechnik und Informatik

Studiengang: Informatik Bachelor

Bachelorarbeit

KI-Entwicklung für das Spiel "Ganz schön clever": Ein Deep Reinforcement Learning Ansatz

Schubert, Sander

Abgabe der Arbeit: 10.11.2023

Betreut durch:

Prof. Dr. Mittag, Florian, Hochschule Coburg

Inhaltsverzeichnis

1 Zusammenfassung **5**

2 Einleitung **6**
2.1 Hinführung zum Thema . . . 6
2.2 Zielsetzung und Motivation . . . 6
2.3 Aufgabenstellung . . . 7
2.4 Aufbau der Arbeit . . . 7

3 Grundlagen **8**
3.1 Allgemeine Grundlagen . . . 8
3.1.1 Ganz schön clever . . . 8
3.1.2 Maschinelles Lernen . . . 11
3.1.3 Reinforcement Learning . . . 12
3.1.4 Deep Learning . . . 14
3.1.5 Proximal Policy Optimization . . . 15
3.2 Verwendete Technologien . . . 16
3.2.1 Gymnasium . . . 16
3.2.2 Stable Baselines 3 . . . 17
3.2.3 Pytorch . . . 17
3.2.4 Matplotlib . . . 18
3.2.5 ChatGPT 4 . . . 18

4 Anforderungen und Konzeption **19**
4.1 Anforderungen . . . 19
4.1.1 Rahmenbedingungen . . . 19
4.1.2 Das Spiel . . . 19
4.1.3 Die Künstliche Intelligenz . . . 20
4.2 Konzeption . . . 21
4.2.1 Die Spielumgebung . . . 23
4.2.2 Die Künstliche Intelligenz . . . 27
4.2.3 Einschränkungen . . . 31

5 Implementierung **32**
5.1 Spielumgebung . . . 32
5.1.1 Klassenattribute . . . 32
5.1.2 Schritt Methode . . . 37
5.1.3 Methode zum Zurücksetzen der Spielumgebung . . . 39
5.1.4 Methode zur Visualisierung der Spielumgebung . . . 39

5.1.5 Würfel Methode 40
5.1.6 Methode zur Überprüfung der freigespielten Belohnungen 40
5.1.7 Methode zur Generierung des Beobachtungsraumes 41
5.1.8 Methode zur Generierung der Aktionsmaske 41
5.1.9 Methode zum Hinzufügen von Boni 43
5.1.10 Methode zum Inkrementieren von Runden 43
5.2 Künstliche Intelligenz 44
5.2.1 Methode zu anlernen des Modells 45
5.2.2 Methode zum Vorhersagen mithilfe des Modells 46
5.2.3 Methode zur Initialisierung der Spielumgebungen 47
5.2.4 Methode zum Anwenden der Aktionsmaske 47
5.3 Darstellung 48
5.3.1 Methoden zum Erstellen von Einträgen 48
5.3.2 Methode zum plotten von Historien 49
5.4 Verwendung 49

6 Ergebnisse **50**
6.1 Trainingshistorie 50
6.1.1 Prototype 50
6.1.2 Training mit und ohne Aktionsmaske 53
6.1.3 Training mit zwei und mit vier Würfeln 55
6.1.4 Optimiertes Training 57
6.2 Erstes Training mit allen Feldern und Boni 58
6.3 Finale Ergebnisse 59
6.3.1 Performance 59
6.3.2 Hyperparameter 60
6.3.3 ChatGPT 4 61

Literaturverzeichnis **62**

Ehrenwörtliche Erklärung **63**

Abb. 1:

1 Zusammenfassung

Zunehmend prägen das maschinelle Lernen und KI die Arbeit und das Leben der Menschen. Besonders präsent sind im Jahr 2023 unter Anderem potente Chatbots wie ChatGPT 4. Solche Tools ermöglichen es Benutzern komplexe sowie komplizierte Aufgaben deutlich einfacher und schneller abzuarbeiten. Hervorzuheben ist hierbei auch, dass man mit solchen Tools deutlich weniger Fachwissen benötigt um in einem Bereich aufgaben effizient lösen zu können, da es einem eine Vielzahl von Informationen zum gewünschten Thema auf anfrage bereitstellen kann. Je komplexer das Problem oder die Fragestellung allerdings sind, desto unverlässlicher werden diese Tools. Man muss seine Anfragen deshalb möglichst präzise formulieren und die Problemstellung in für das Tool angemessene Teilaufgaben zerlegen.

Auch in der Spielentwicklung spielen maschinelles Lernen und KI schon seit langem eine bedeutende Rolle. In den meisten Spielen gibt es sogenannte Bots, welche man als KI bezeichnen kann. Diese sollen bestimmte Aufgaben im Spiel erfüllen um den Spieler zu unterstützen oder im als Widersacher zu dienen. Je komplexer die Aufgabe, umso schwerer ist es einen solchen Bot zu erstellen, welcher die Aufgabe auf zufriedenstellende Weise erfüllen kann.

Das Gesellschaftsspiel "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. Ziel dieser Arbeit ist es eine KI beziehungsweise einen Bot für dieses Spiel zu entwickeln, der das Spiel effizient spielen kann, sowie zu analysieren welche Aspekte der Entwicklung dabei relevant und zu beachten sind.

Dazu mussten Spielumgebung sowie KI zunächst implementiert werden. Dies geschah mithilfe von Bibliotheken wie Stable-Baselines3 und Gymnasium. Insgesamt ergab sich dabei, dass sich mithilfe des PPO-Algorithmus von Stable-Balseslines3 auf relativ einfache Weise ein effizientes Modell für das Spiel entwickeln lässt.

2 Einleitung

2.1 Hinführung zum Thema

In den vergangenen Jahren gewann maschinelles Lernen und insbesondere die Künstliche Intelligenz zunehmend an Bedeutung, Tendenz steigend. Im Jahr 2023 ist eines der präsentesten neuen Tools 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. Auch in dieser Arbeit wurde ChatGPT 4 als unterstützendes Tool verwendet. Es wurde vor Allem dafür benutzt um fachliche Fragen zu beantworten, aber auch anfangs um Code für den Prototypen zu generieren. 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 nimmt das maschinelle Lernen und die Künstliche Intelligenz schon seit langem eine zentrale Rolle ein. In den meisten 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 effizient und richtig lösen kann.

Das Gemeinschaftsspiel "Ganz schön clever" ist ein Würfelspiel, welches 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 zu Stande. Das Spiel weist zusätzlich eine hohe Stochastizität auf, welche die Komplexität weiter erhöht.

Interessant ist wie man für ein solch komplexes Spiel einen Bot oder eine Künstliche Intelligenz entwickeln kann um dieses effizient spielen zu können. Ist die Komplexität möglicherweise zu groß, um vom Bot erfasst zu werden und wenn nicht, wie kann man einen solchen Bot implementieren und was gilt es dabei zu beachten?

2.2 Zielsetzung und Motivation

Ziel der Arbeit ist es einen Bot beziehungsweise eine Künstliche Intelligenz zu entwickeln, welche das Spiel "Ganz schön clever" möglichst effizient spielen kann. Dabei soll analysiert werden, welche Aspekte es dabei zu beachten gilt und wie sich unterschiedliche Ansätze auf das Verhalten und die Performance des Modells (des Bots) auswirken.

In den vergangenen Jahren hat sich viel getan, weshalb deutlich mehr möglich geworden ist. Mit neuen Möglichkeiten ergeben sich auch bessere oder einfachere Ansätze, die zu einem wünschenswerten Ergebnis führen. Ziel ist es auch einen geeigneten Ansatz zu finden und zu vervollständigen.

Es gibt des Weiteren 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 man diese überwinden kann.

2.3 Aufgabenstellung

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

2.4 Aufbau der Arbeit

3 Grundlagen

3.1 Allgemeine Grundlagen

3.1.1 Ganz schön clever

Die folgende Abbildung zeigt das Spielbrett des Spiels "Ganz schön clever" zu dem im Rahmen dieser Arbeit eine KI implementiert werden soll:

Abb. 1: Ganz schön clever
Quelle: [Screenshot des Android Handyspiels Ganz schön clever]

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

Es gibt sechs farbige Würfel, wobei jeder bis auf den weißen einem der 5 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 das entsprechende 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 Würfen ein. Nach jeder Runde bekommt der Spieler zudem die Möglichkeit einen Würfel auf dem Silbertablett eines Mitspielers zu wählen. Diese Wahl verhält sich so wie bei der Wahl eines eigenen Würfels bei den Würfen

des Spielers selbst. Die Anzahl der Runden werden von der Spielerzahl festgelegt. 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 ist immer drei, allerdings kann diese Anzahl reduziert werden, wenn kein Würfel mehr zum Würfeln zur Verfügung steht. Dann wird der Spielablauf fortgesetzt als hätte der Spieler seinen dritten Wurf in der Runde beendet und er kann einen Würfel vom Tablett 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 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.
Es gibt zwei Arten von Belohnungen im Spiel. Punktebelohnungen und Boni. Bei Punktebelohnungen handelt es sich um Punkte welche auf den Score des Spielers addiert werden, welcher am Ende des Spiels entscheidet wer gewonnen hat. Der Spieler mit dem höchsten Score gewinnt das Spiel. Punktebelohnungen erhält man beim gelben Feld indem man eine Spalte vollständig ausfüllt. Die Anzahl der Punkte findet sich am Ende der Spalte. Im blauen Feld erhält man Punktebelohnungen mit zunehmender Anzahl an ausgefüllten blauen Kästchen. Die Anzahl der Punkte findet sich im oberen Bereich des blauen Feldes. Im grünen Feld ebenso mit zunehmender Anzahl der ausgefüllten grünen Kästchen. Die Anzahl der Punkte findet sich auch hier im oberen Bereich des grünen Feldes. Im orangenen und lila Feld muss man für Punktebelohnungen lediglich ein Kästchen ausfüllen. Die Punktebelohnung entspricht der Augenzahl des entsprechenden ausgefüllten Kästchens. Beim orangenen Feld wird diese Augenzahl bei einigen Feldern zusätzlich mit zwei oder drei multipliziert. Außerdem gibt es eine Boni Belohnung, 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 den niedrigsten erreichten Punktewert multipliziert und zum Gesamtpunktestand 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 bei den 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 die Boni welche beim gelben Feld freigeschaltet wird indem alle diagonalen Felder von links oben nach rechts unten ausgefüllt werden.

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

Extra Wahl: Bei der Extra Wahl wird es dem Spieler ermöglicht am Ende seiner eignen Runde beziehungsweise nachdem er einen Würfel vom Silbertablett des Gegners gewählt hat erneut Würfel zu wählen und die entsprechenden Kästchen dafür anzukreuzen. Würfel die so gewählt wurden können mithilfe der Extra Wahl Boni nicht erneut gewählt werden solange keine neue Runde oder Wahl vom Silbertablett beginnt. Mit der Extra Wahl Boni können alle Würfel gewählt werden, nicht nur die, welche unter normalen Umständen gültig zur Wahl stehen würden. Das Symbol ist die +1.

Neuer Wurf: Der Neue Wurf ermöglicht es dem Spieler einen seinen 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 erhalten der Boni eines der gelben Kästchen nach eigener Wahl auszufüllen. Das Symbol ist ein Kreuz auf gelbem Hintergrund.

Blaues Kreuz: Ermöglicht es dem Spieler direkt nach erhalten der Boni eines der blauen Kästchen nach eigener Wahl auszufüllen. Das Symbol ist ein Kreuz auf blauem Hintergrund.

Grünes Kreuz: Ermöglicht es dem Spieler direkt nach erhalten der Boni das nächste freie Grüne Kästchen auszufüllen. Das Symbol ist ein Kreuz auf grünem Hintergrund.

Orangene Vier: Ermöglicht es dem Spieler direkt nach erhalten der Boni 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 erhalten der Boni 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 erhalten der Boni 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 der Boni das nächste freie lila Kästchen mit einer sechs auszufüllen. Das Symbol ist eine sechs auf lila 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 dem Extra Wahl Boni bereits gewählt worden sind. Außerdem werden Würfel, welche eine geringere Augenzahl aufweisen als der aktuell gewählte Würfel automatisch ungültig. Eine Ausnahme ist hierbei die Wahl eines Würfels vom Silbertablett des Gegenspieler oder Wahlen von Würfeln durch die Extra Wahl Boni. Würfel werden wieder gültig am Anfang jeder neuen Runde, nach Abschluss des dritten Wurfes, sowie nach der Wahl des Würfels vom Silbertablett, als auch nachdem Wahlen mit dem Extra Wahl

Boni erfolgt sind. Dabei bleiben mithilfe der Extra Wahl Boni gewählte Würfel solange ungültig, bis keine weiteren Extra Wahl Boni in diesem Spielschritt mehr verwendet werden und das Spiel normal weiter geht.

Jedes Feld hat seine eigenen Regeln, die bestimmen, wann ein Kästchen ausgefüllt werden darf. Beim gelben Feld muss die Augenzahl des Würfel 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. Dann kann das nächste Feld mit jeder beliebigen Augenzahl gewählt 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.

3.1.2 Maschinelles Lernen

"Maschinelles Lernen heißt, Computer so zu programmieren, dass ein bestimmtes Leistungsmerkmal anhand von Beispieldaten oder Erfahrungswerten optimiert wird" [Maschinelles Lernen, Seite 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. 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 des selben oder eines ähnlichen Sachverhaltes, der beim generieren der Trainingsdaten vorhanden war, zu treffen. Ein besonders weit verbreiteter Anwendungsfall ist die Herleitung von Kundenverhalten und möglicher Optimierungsmöglichkeiten für den Verkauf. 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 dann 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 [Maschinelles Lernen, Seite 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. [ML, Seite 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 [ML, Seite 3].

Und das sind nur einige wenige der zahlreichen möglichen Bereiche in denen maschinelles Lernen bereits Anwendung findet.

Die Datenbestände und das World Wide Web werden immer größer und die Suche nach relevanten Daten kann nicht mehr manuell vorgenommen werden [ML, Seite 3].

"Das maschinelle Lernen ist aber nicht nur für Datenbanken relevant, sondern auch für das Gebiet der künstlichen Intelligenz" [ML, Seite 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 [ML, Seite 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.

3.1.3 Reinforcement Learning

Reinforcement Learning (im deutschen: bestärkendes Lernen) heißt so, weil es die Aktionen des Agenten (beziehungsweise des Modells) bestärkt. Man kann sich das in etwa so vorstellen, 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 [RL, Seite 11].

Im Reinforcement Learning sind vor allem Folgende Begriffe wichtig:

Agent (oder Modell): 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, Seite 11]. In dieser Arbeit ist diese Entität ein Modell, welches mithilfe der Bibliothek Stable-Baselines3 erstellt wird.

Umgebung: Dabei handelt es sich um die Außenwelt des Agenten [RL, Seite 11]. der Agent interagiert mit dieser und erhält je nach Zustand der Umgebung und seiner gewählten Aktion ein entsprechendes Feedback.

Aktion: Eine Aktion beschreibt das Verhalten des Agenten [RL, Seite 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 [RL, Seite 11]. In dieser Arbeit ist der Zustand von den Eigenschaften des Spielbrettes, der Würfel, der Rundenanzahl und der erspielten Boni abhängig.

Belohnung: Positive oder negative Vergeltung je nachdem wie gut der Zustandswechsel von Zustand x nach Zustand y gewesen ist [RL, Seite 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 nächsten Aktionen wählt [RL, Seite 11]. In dieser Arbeit wird die Policy durch ein Multilayer Perceptron (siehe Deep Learning) abgebildet.

Episode: Eine Menge an Zusammenhängenden Aktionen, welche endet, wenn das Ziel erreicht worden ist [RL, Seite 11]. In dieser Arbeit ist eine Episode ein kompletter Spieldurchlauf von Ganz schön clever.
Die folgende Abbildung beschreibt einen Lernzyklus im Reinforcement Learning:

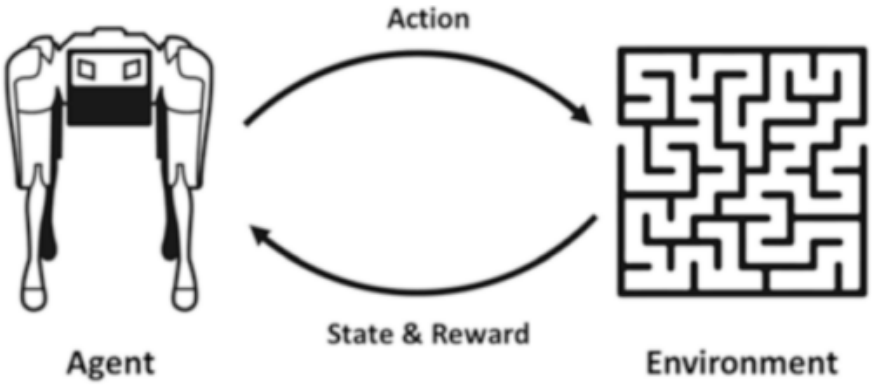


Abb. 2: Reinforcement Learning
Quelle: [RL, Seite 12]

Der Agent führt 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 gesamte erreichte Belohnung zu maximieren. Demnach wird die Policy dementsprechend angepasst, dass dies begünstigt wird [RL, Seite 12f].

Doch dabei 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 bevorzugen, 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 und Ausbeutung. 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 oder bereits bekannte, welche eine gute Belohnung zu bringen scheinen auszunutzen beziehungsweise auszubeuten [RL, Seite 13].

3.1.4 Deep Learning

Was unterscheidet Reinforcement Learning von Deep Reinforcement Learning? Im wesentlichen handelt es sich um das selbe Konzept, allerdings ist das eine Deep Learning und das andere nicht. Was ist also Deep Learning?

Von Deep Learning spricht man sobald ein Neuronales Netz mehrere versteckten Schichten hat.

Ein solches Netz besteht aus einer Vielzahl von Neuronen. [DRL, Seite 75] Ein solches Neuron setzt sich zusammen aus Inputs, Outputs, Gewichtungen dieser In- und Outputs, sowie einer Aktivierungsfunktion, welche auch gewichtet sein kann. 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 die selbe Aktivierungsfunktion. Was den Beitrag der einzelnen Neuronen zum Gesamtergebnis des Netzes steuert sind im wesentlichen seine Gewichtungen und die Position im Netz.

Die folgende Abbildung zeigt ein solches Multilayer Perceptron. Dabei sind nicht alle Verbindungen eingezeichnet, um Übersichtlichkeit zu gewährleisten:

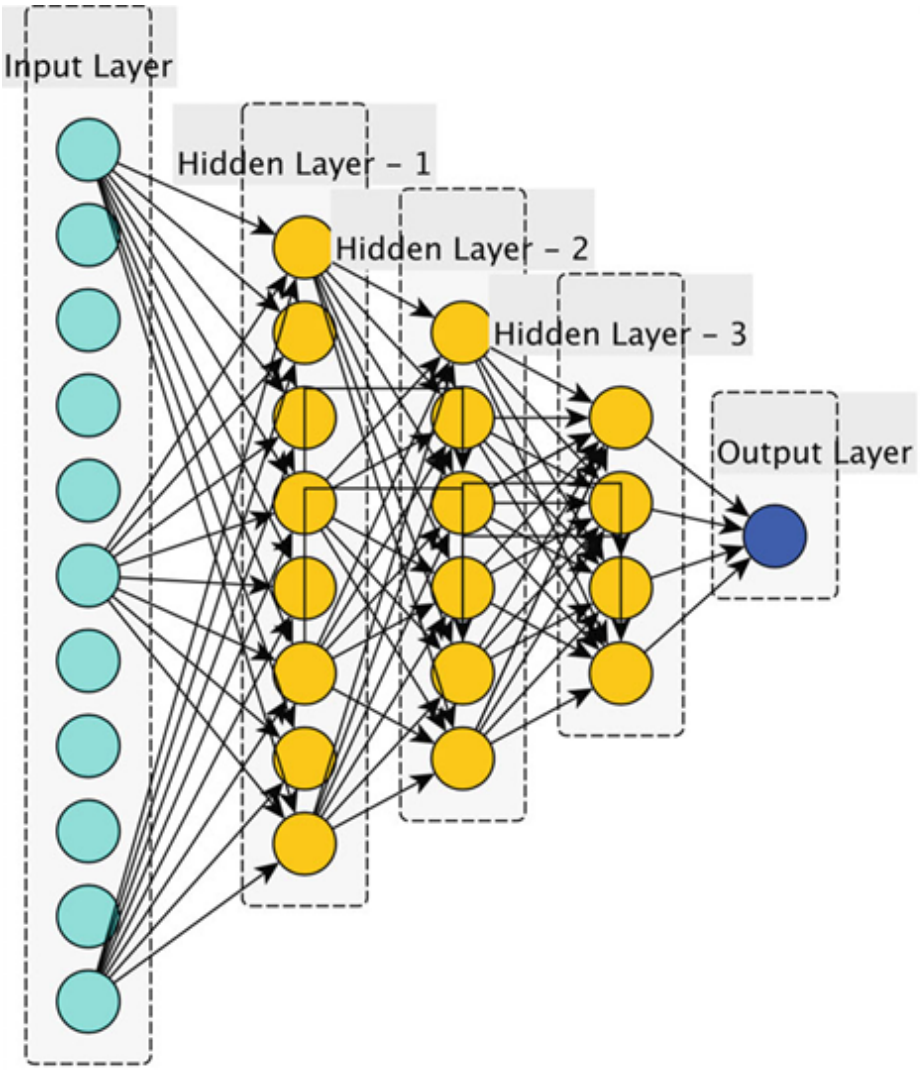


Abb. 3: Multilayer Perceptron
Quelle: [DRL, Seite 78]

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

3.1.5 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 Stichproben Komplexität. PPO zeigt eine gute Balance zwischen Stichproben Komplexität, Einfachheit und Trainingsgeschwindigkeit [PPO, Seite 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 an. Das kann man sich ähnlich wie einen Punkt auf einer Parabel vorstellen, die Policy entspricht dem Punkt und wird so lange angepasst beziehungsweise verschoben, bis sie möglichst das Minimum der Parabel erreicht. Dieses Minimum spiegelt eine optimale Policy wieder.

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.

PPO 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. Daher eignet sich PPO besonders gut für Einsteiger, die bisher nicht viel mit Deep Reinforcement Learning gearbeitet haben.

3.2 Verwendete Technologien

3.2.1 Gymnasium

Gymnasium ist die Fortführung der OpenAi Bibliothek Gym [Gym]. 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 sich mit wenig Aufwand ausprobieren zu können.

Die Bibliothek bietet Kernmethoden, welche später 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 (im Englischen step-method) und eine Methode zum zurücksetzen der Umgebung

auf den Startzustand (im Englischen 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, dies ist allerdings optional.

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.

3.2.2 Stable Baselines 3

"Stable Baselines 3 ist eine Bibliothek, welche verlässliche Implementierungen von Reinforcement Learning Algorithmen in Pytorch bietet" [SB3, Seite 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 auch eine experimentelle Version der Bibliothek, welche Stable Baseline 3 Contributing genannt wird [SB3, Seite 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 erweitert den PPO Algorithmus um die Funktionalität einer Maskierung. Diese Maskierung ermöglicht es die Wahlwahrscheinlichkeit 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.

3.2.3 Pytorch

Pytorch ist eines der beiden am weitesten verbreiteten Deep Learning Frameworks. Es bietet die Möglichkeit vereinfacht eigene Deep Learning Algorithmen zu implementieren. Dies erfolgt vor allem mit sogenannten Tensoren. Dabei handelt es sich um mehrdimensionale Matrizen, mithilfe derer die Berechnungen vorgenommen werden.

Pytorch wurde in dieser Arbeit indirekt verwendet, da Stable Baselines 3 auf Pytorch basiert.

3.2.4 Matplotlib

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

3.2.5 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 dann eine Antwort bekommt. Die Erzeugnisse des Chat-Bots sind so gut, dass er sich gut eignet, um bei der Konzeption und Programmierung der Arbeit zu unterstützen. Daher wird ChatGPT 4 in dieser Arbeit zum Teil 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.

4 Anforderungen und Konzeption

4.1 Anforderungen

Dieses Kapitel beschreibt die Anforderungen an das Projekt. Das Projekt ist die Implementierung und die Analyse einer Spiel-KI für das Spiel Ganz schön clever von Anfang bis Ende.

4.1.1 Rahmenbedingungen

Der zeitliche Rahmen der Arbeit beträgt vier Monate. In dieser Zeit ist zunächst ein Grundkonzept in Form eines Exposés vorzustellen. Anschließend soll dieses Konzept weiter verfeinert und validiert werden, was schließlich zur Vervollständigung des Projektes mit den gewünschten, sich aber zum Teil auch ändernden Anforderungen führen soll.

Es ist eine Künstliche Intelligenz zu implementieren, welche das Spiel Ganz schön clever effizient spielen 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. Dies soll mithilfe geeigneter, schlüssiger sowie verständlicher Methoden und Visualisierungstechniken erfolgen.

4.1.2 Das Spiel

Das Spiel heißt Ganz schön clever. Es ist ein Würfelspiel bei dem es darum geht möglichst viele Punkte innerhalb einer vorgeschriebenen Rundenanzahl zu erreichen [siehe Grundlagen].

Zunächst soll ein Prototyp der Spielumgebung entwickelt werden, welcher später Stück für Stück 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.

Einen Mechanismus, welcher die Würfel für die aktuelle Runde als ungültig markiert, sobald sie gewählt werden. Ungültige Würfel dürfen nicht gewählt werden [siehe Grundlagen].

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 Boni befindet.

Ein Runden-System, bei dem insgesamt sechs Runden gespielt werden, in denen jeweils bis zu drei Würfe erfolgen, falls 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 diese Würfel zufällig gewürfelt und dann drei davon auf das Silbertablett gelegt. Außerdem werden Boni am Anfang der ersten, zweiten, dritten und vierten Runde für alle Spieler freigeschaltet [siehe Grundlagen].

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 [siehe Grundlagen]. Die Felder beinhalten verschiedene Boni, welche freigespielt werden, wenn bestimmte Kästchen oder Kombinationen aus diesen ausgefüllt worden sind [siehe Grundlagen].

Die sieben verschiedenen Boni, Fuchs, Extra Wahl, Neu Würfeln, Gelbes Kreuz, Blaues Kreuz, Grünes Kreuz, Orangene Vier, Orangene Fünf, Orangene Sechs sowie Lila Sechs [siehe Grundlagen]. Das beinhaltet sowohl das Erhalten, Speichern, sowie die Benutzung der Boni.

Ein Mechanismus, welcher Würfel, die mithilfe der Extra Wahl Boni gewählt worden sind, als ungültig markiert, damit diese nicht erneut in der selben Runde durch die Extra Wahl Boni 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, welche nach der Wahl vom Silbertablett des Gegners erfolgen.

4.1.3 Die Künstliche Intelligenz

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

Die wichtigsten dieser Parameter sind unter Anderem die Trainingsdauer, welche festlegt wie lange trainiert wird, Gamma, welches bestimmt wie zukunftsorientiert die KI ihre Entscheidungen fällt, die Größe und Art des neuronalen Netzen, das verwendet wird, welches bestimmt wie und präzise Merkmale von der Künstlichen Intelligenz erfasst werden können und 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 vorhergesehenen zeitliche Rahmen der Arbeit eingehalten werden kann.

4.2 Konzeption

In diesem Kapitel wird das Grundkonzept der Künstlichen Intelligenz, sowie der Spielumgebung und deren Zusammenspiel beschrieben.
Die Folgende Abbildung zeigt das Zusammenspiel von Spielumgebung und KI während des Trainingsprozesses:

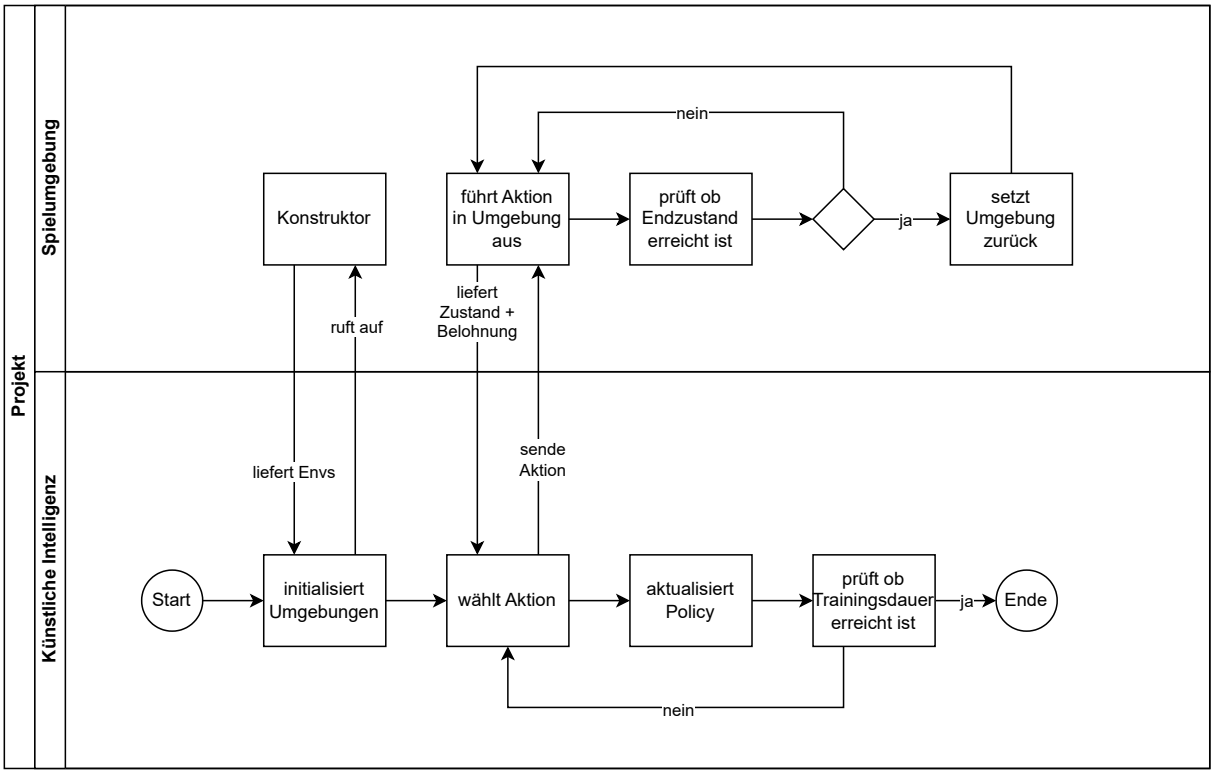


Abb. 4: Swimlane-Diagramm des Projektes

Der Trainingsprozess wird angestoßen, die Umgebung beziehungsweise die Umgebungen werden initialisiert. Dazu wird der Konstruktor der Umgebung aufgerufen. Daraufhin wird von der KI eine Aktion ausgewählt und an die Umgebung weitergeleitet. Die Umgebung führt diese Aktion aus und prüft ob der Endzustand des Spiels erreicht worden ist oder nicht. Wenn der Endzustand erreicht wurde, wird die Umgebung auf den Anfangszustand zurückgesetzt. Wenn der Endzustand nicht erreicht worden ist, wird der Prozess ohne Weiteres fortgesetzt. Daraufhin liefert die Umgebung der KI eine Belohnung für den ausgeführten Schritt und den neuen Zustand der Umgebung nach Ausführung der Schrittes/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 geführt haben und welche nicht.

4.2.1 Die Spielumgebung

Die folgenden beiden Abbildungen 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:

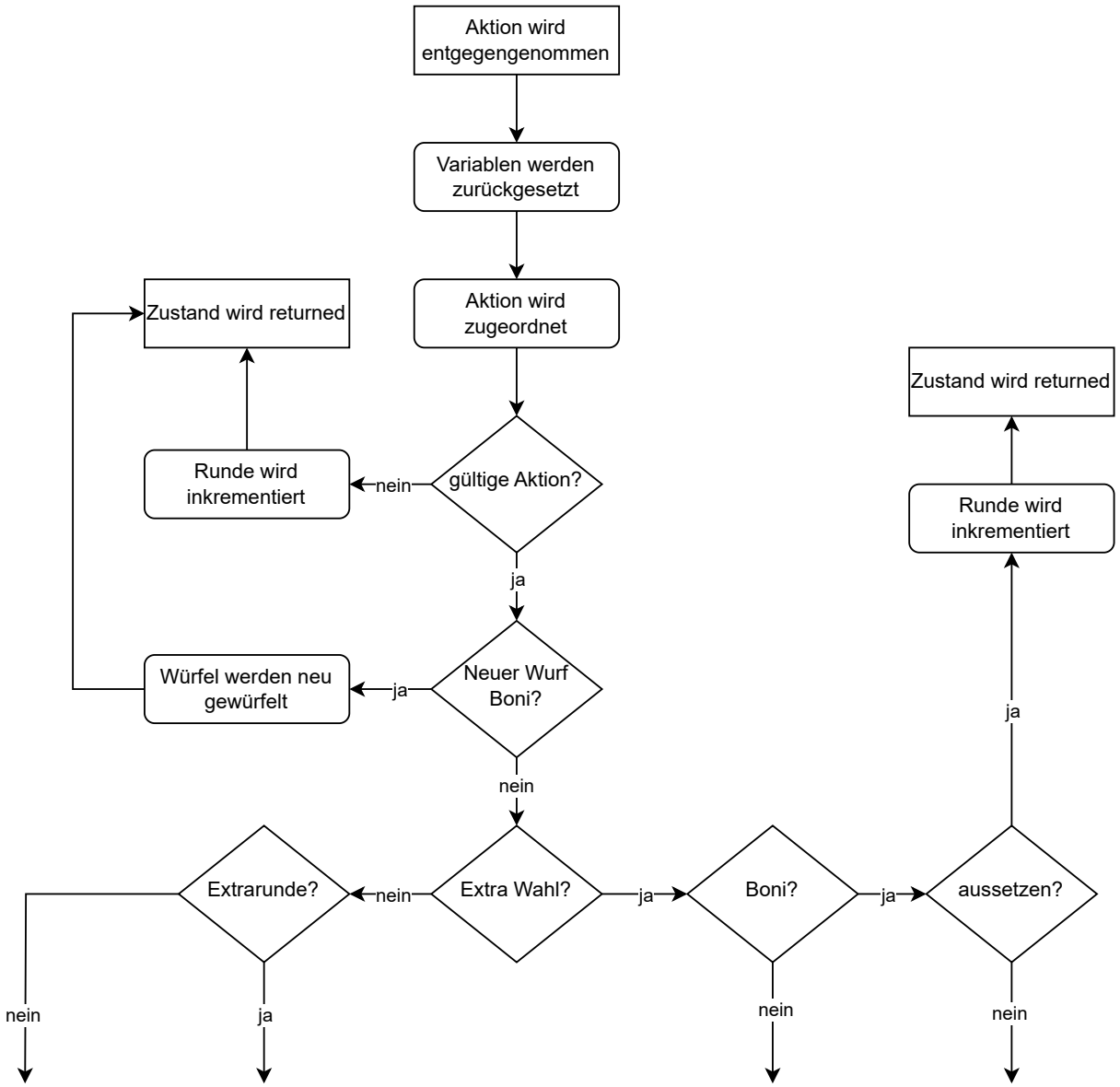


Abb. 5: Ablauf-Diagramm der Schritt Funktion 1

Nachdem die Schritt Funktion angestoßen wird, werden die Variablen dieser 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 Würfen, 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 gewürfelt 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 der Extra Wahl Boni und Wahlen vom Silbertablett des 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, die es erlauben eines der farbigen Felder auszufüllen [siehe Grundlagen].

Handelt es sich nicht um eine Extrarunde muss es eine normale Wahl nach einem Standardwurf in einer der sechs Runden sein [siehe Grundlagen].

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

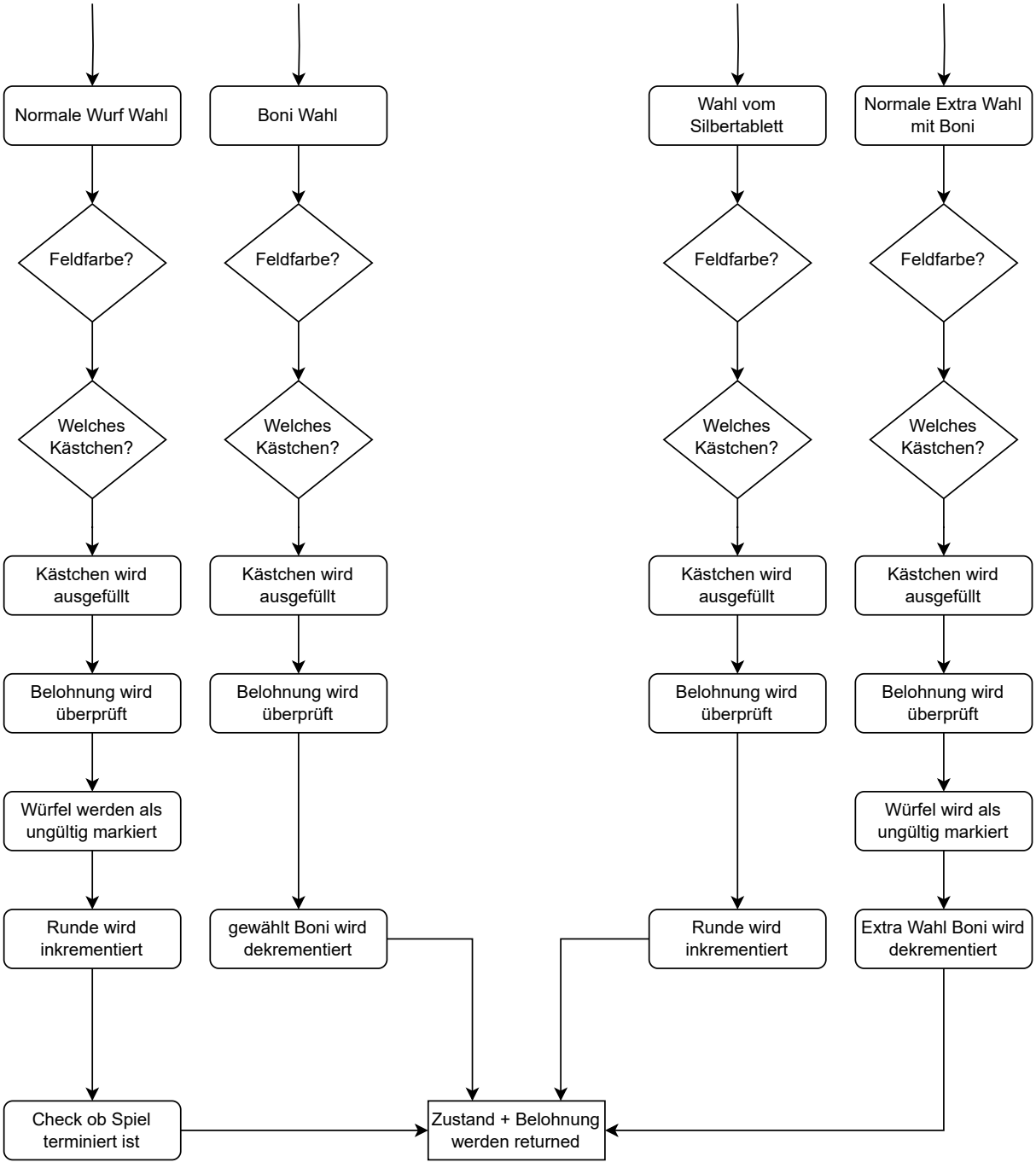


Abb. 6: Ablauf-Diagramm der Schritt Funktion 2

Der folgende Ablauf der Funktion gestaltet sich relativ ähnlich bei allen vier Varianten. Bei allen Vieren wird das Feld und das Kästchen ermittel, 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 der normalen Wahl nach einem Standardwurf werden die entsprechenden Würfel als ungültig markiert [siehe Grundlagen]

und die Runde wird inkrementiert. Außerdem prüft die Funktion bei dieser Variante am Schluss ob das Spiel terminiert ist oder nicht.

Beim ausfüllen eines Feldes mittels Boni (keine Extra Wahl Boni) wird der entsprechende Boni dekrementiert.

Bei der Extra Wahl vom Silbertablett wird die Runde inkrementiert.

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

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

4.2.2 Die Künstliche Intelligenz

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

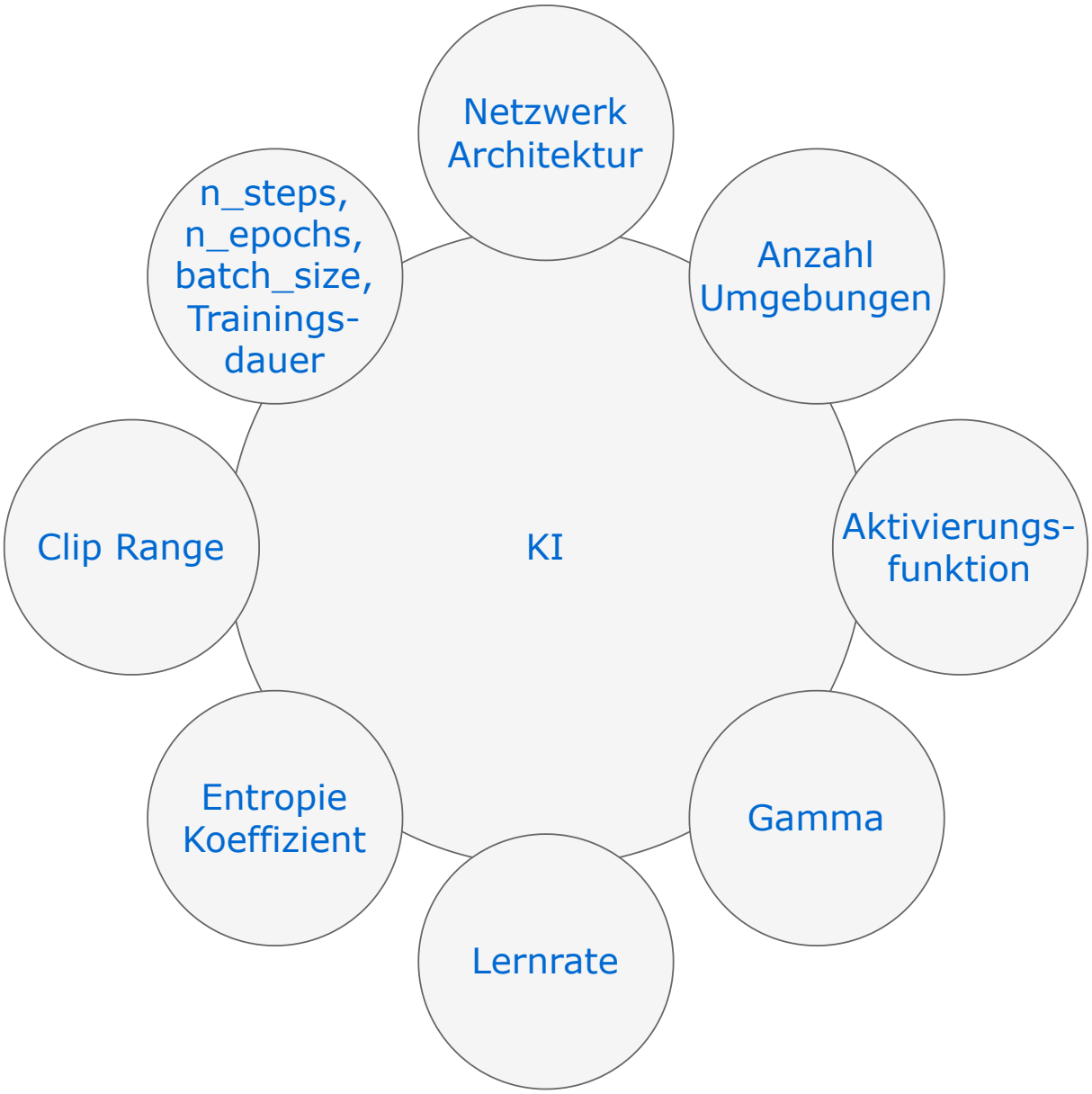


Abb. 7: Hyperparameter der Künstlichen Intelligenz

Die Künstliche Intelligenz basiert auf der MaskablePPO Version von Stable Baselines 3 Contributing. Weiterhin sind Konfigurationen zu treffen, welche das Training beeinflussen und steuern. Diese Einstellungen nennt man Hyperparameter. Sie legen die Ausprägung gewisser Steuerelemente des MaskablePPO Algorithmus und ähnlicher Algorithmen fest.

Die wichtigsten Hyperparameter für das Projekt sind folgende:

Netzwerk Architektur: Die Netzwerk Architektur legt die Anzahl der Schichten und der Neuronen pro Schicht des Neuronale Netzes fest, welches für die Abbildung der Policy verwendet wird. Außerdem legt sie fest wie viele Neuronen es pro Schicht gibt [siehe Grundlagen]. Die Netzwerk Architektur ist so zu bestimmen, dass die Künstliche Intelligenz im Stande ist die Komplexität der Problemstellung mithilfe dieser zu erfassen und gleichzeitig zu gewährleisten, dass die Trainingsdauer, die Systemauslastung dadurch nicht zu hoch steigen und es möglichst keine Überanpassung an die Trainingsdaten gibt. 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 [siehe Grundlagen]. 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 der Trainingsdaten unterscheiden.

Anzahl der Umgebungen: Eine erhöhte Anzahl an Trainingsumgebungen ermöglicht es vor allem 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 somit 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 laufen zu lassen.

Aktivierungsfunktion: Die Aktivierungsfunktion legt fest, wann und wie stark Neuronen 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 zu erzielen scheint. Die Formel der ReLU Funktion lautet: f(x) = max(0,x). Die ReLU Funktion gibt, wenn der Wert kleiner Null ist Null zurück und ansonsten den Wert selbst. Die folgende Abbildung zeigt den Graphen der ReLU Funktion:

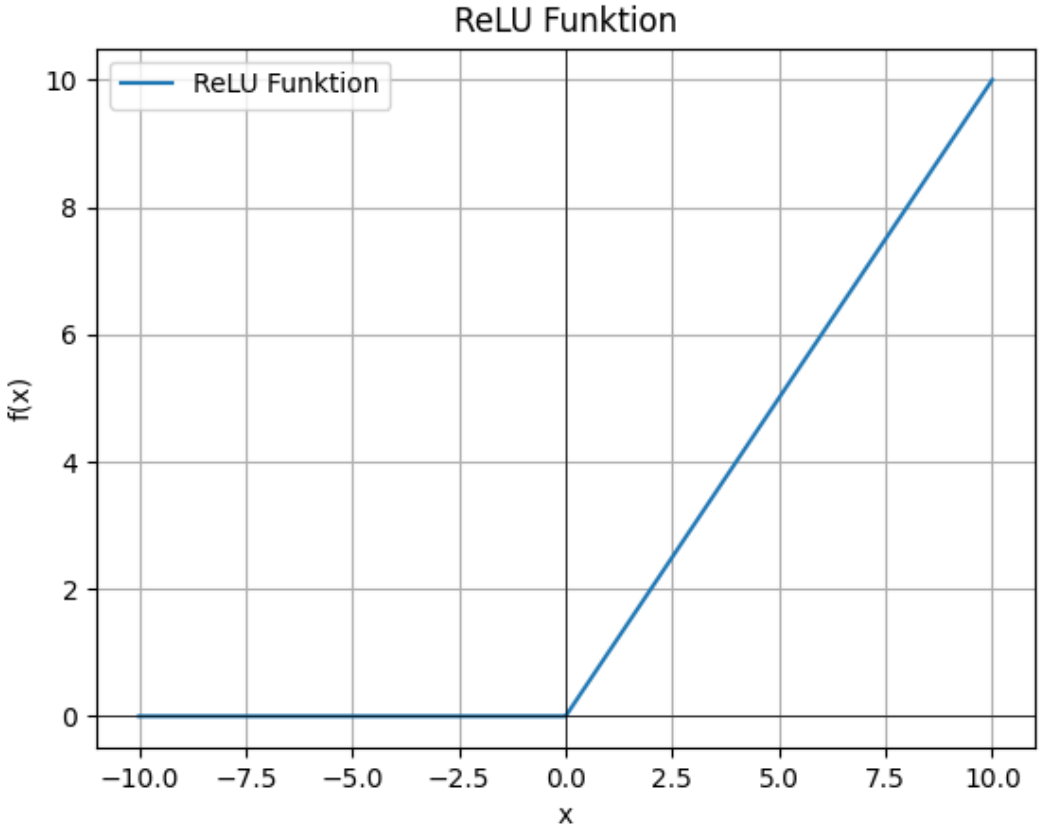


Abb. 8: ReLU Funktion

Gamma: 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 nur 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.

Die Lernrate: Die 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 gewählten Trainingsdatensatz abhängen. Zu niedrige Werte führen wiederum zu einer Erhöhung der nötigen Trainingsdaten und

Trainingsdauer. Der Wert für die Lernrate bewegt sich im Projekt üblicherweise zwischen Werten von 0.0003 bis zu 0.0003x32.

Entropie Koeffizient: Der Entropie Koeffizient bestimmt das Maß an Exploration des Modells. Je höher der Entropie Koeffizient ist desto stärker Belohnt das Modell neue beziehungsweise bisher wenig erkundete Aktionen oder Zustände. Ein zu hoher Wert führt dazu, dass das Modell nicht lernt bereits funktionierende Taktiken genügend 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. Sie legt fest wie stark Policy Updates sein dürfen und ab welchem Schwellwert die Updates abgeschnitten werden. Mit Abschneiden ist gemeint, dass die Updates zwar noch 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: nSteps legt fest, wie viele Aktions-Zustands-Paare gesammelt werden bevor ein Update des Neuronales Netzes erfolgt.

nEpochs: nEpochs legt fest wie oft das selbe 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 beträgt 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 das selbe Gesamtdatenpaket durch ein hohes nEpochs viele male verwendet wird. Batch Size legt fest wie groß diese kleineren Datenpakete (Batches) sind.

Die Trainingsdauer: Die Trainingsdauer legt fest wie lange trainiert wird. Die Implementierung von Stable Baselines 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 nur durch Erhöhung der Umgebungsanzahl bei gleichbleibender Trainingsdauer in Timesteps.

4.2.3 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 die Extra Wahl Boni kein letztes mal benutzt werden und es gibt in der letzten Runde auch keine Wahl vom Silbertablett. Außerdem werden beim solo Spiel nicht die niedrigsten sechs Würfel des Wurfes auf das Silbertablett des viruellen Mitspielers gelegt, sondern drei zufällige.

Des weiteren ist ChatGPT nicht zur Codegenerierung über den Prototypen hinaus zu nutzen. Der Code soll nach Erstellung der Prototypen selbstständig geschrieben werden. Auch soll ChatGPT nicht genutzt werden um Text für die Bachelorarbeit selbst zu generieren.

5 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.

5.1 Spielumgebung

Die Implementierung besteht aus zwei wesentlichen Klassen. Eine davon ist die Spielumgebung. Zunächst werden in diesem Kapitel die wesentlichen Attribute der Spielumgebung und anschließend ihre Methoden erläutert.

5.1.1 Klassenattribute

Der folgende Code 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 Runden

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) zu 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.

Der folgende Code 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.

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

```
self.score
self.score_history
self.initialized
```

Code 3: Klassenattribute für Punktestand

Das Attribut score repräsentiert den aktuellen Score 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 Episoden beziehungsweise Spieldurchläufen beim Training. Das Attribut initialized wird verwendet, um zu gewährleisten, dass nur Einträge in der Score-History eingetragen werden, nachdem ein Spiel abgeschlossen worden ist. Es trifft eine Aussage darüber ob die Spielumgebung bereits einmal initialisiert worden ist oder nicht.

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

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

Code 4: Klassenattribute für farbige Felder

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 beziehungsweise freigeschaltet worden sind.

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

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

Code 5: Klassenattribute für freizuschaltende Boni

Die Attribute yellow_rewards, blue_rewards, green, 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 Boni aufweisen, was dem Wert None entspricht.

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

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

Code 6: Klassenattribute für Punktestand

Im Attribut yellow_rewards sind im Spaltenbereich (col) die Punktebelohnungen 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.

Der folgende Code 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 sie nach dem Spiel wieder zurückgesetzt werden:

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

Code 7: Klassenattribute für Belohnungsflags

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

```
self.yellow_field_score
self.blue_field_score
self.green_field_score
self.orange_field_score
self.purple_field_score
```

Code 8: Klassenattribute für Punktestände der Felder

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

```
self.extra_pick
self.re_roll
self.fox
self.yellow_cross
self.blue_cross
self.green_cross
self.orange_four
self.orange_five
self.orange_six
self.purple_six
```

Code 9: Klassenattribute für freigespielte Boni

Der folgende Code 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 dieses Attributes inkrementiert. Diese Attribute dienen nicht dem Spielablauf selbst, sondern zur Nachvollziehbarkeit der Strategie des Modells:

```
self.picked_yellow
self.picked_blue
self.picked_green
self.picked_orange
self.picked_purple
```

Code 10: Klassenattribute für gewählte Kästchen in den farbigen Feldern

Der folgende Code zeigt die Klassenattribute für den Aktions- sowie Beobachtungsraum:

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

Code 11: Klassenattribute Aktionsraum und Beobachtungsraum

Das Attribut number_of_actions repräsentiert die Gesamtanzahl an möglichen Aktionen des Modells. Die Attribute low_bound und high_bound setzten 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.

5.1.2 Schritt Methode

Der folgende Pseudocode zeigt die Funktionsweise der Schritt Methode (step method) der Spielumgebung. Diese Methode führt Spielschritte beziehungsweise Aktionen in der Spielumgebung aus. Dabei ist zu beachten, dass es sowohl eine Aktion/Ausführung für jedes Kästchen auf dem Spielfeld mit dem weißen (Aktion 61-121, 183-243) und dem jeweils zum Feld passenden farbigen Würfel (Aktion 0-60, 122-182) gibt:

```
step(Aktion):
    Setze Methodenattribute zurück

    if not Extrapickaktion:
        if Aktion im gelben Feld:
            Fülle entsprechendes gelbes Kästchen aus
            if Gelbes Kreuz Boni >= 1
                Bonusrunde = True
                Gelbes Kreuz -= 1
            if not Bonusrunde:
                Setze entsprechende Würfel auf ungültig
        if Aktion im blauen Feld:
            Fülle entsprechendes blaues Kästchen aus
            if Blaues Kreuz Boni >= 1:
                Bonusrunde = True
                Blaues Kreuz -= 1
            if not Bonusrunde:
                Setze entsprechende Würfel auf ungültig
```

```
        if Aktion im grünen Feld:
            Fülle entsprechendes grünes Kästchen aus
            if Grünes Kreuz Boni >= 1:
                Bonusrunde = True
                Grünes Kreuz -= 1
            if not Bonusrunde:
                Setze entsprechende Würfel auf ungültig
        if Aktion im orangene Feld:
            Fülle entsprechendes orangenes Kästchen aus
            if Orangene Feld Boni >= 1:
                Bonusrunde = True
                Orangene Feld Boni -= 1
            if not Bonusrunde:
                Setze entsprechende Würfel auf ungültig
        if Aktion im lila Feld:
            Fülle entsprechendes lila Kästchen aus
            if Lila Sechs Boni >= 1:
                Bonusrunde = True
                Lila Sechs Boni -= 1
            if not Bonusrunde:
                Setze entsprechende Würfel auf ungültig

    if Extrapickaktion:
        Bonusrunde = True
        if Extra Pick Boni benutzt:
            Extra Pick Boni -= 1
        Finde Kästchen zum ausfüllen und fülle es aus
        Setze gewählten Würfel auf ungültig
        if Extra Pick Boni <= 0 or Wahl erfolgte vom Silbertablett:
            Runde wird inkrementiert

    if Neu Würfeln Boni wird benutzt:
        Würfle Würfel neu
        Neu Würfeln Boni -= 1
        Aktualisiere Aktionsmaske
        return Beobachtungsraum, Punktebelohnung, Spiel terminiert?

    if Statt Extra Pick Boni gepasst wird:
        Runde wird inkrementiert
        return Beobachtungsraum, Punktebelohnung, Spiel terminiert?
    if ungültige Aktion gewählt da keine gültigen Aktionen vorhanden:
```

```
60          Nichts tun
61
62      Punktebelohnung += erspielte Belohnung in diesem Zug
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 12: Schritt Methode

5.1.3 Methode zum Zurücksetzen der Spielumgebung

Der folgende Pseudocode zeigt die Funktionsweise der Methode zum Zurücksetzen der Spielumgebung (reset method). Sie setzt Umgebungen, in denen ein Spiel abgeschlossen worden ist auf den Anfangszustand zurück, damit eine weitere Runde gespielt werden kann:

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

Code 13: Methode zum Zurücksetzen der Umgebung

5.1.4 Methode zur Visualisierung der Spielumgebung

Der folgende Pseudocode zeigt die Funktionsweise der Funktion zur Visualisierung der Spielumgebung (render method). Die Visualisierung dient einer verbesserten Nachvollziehbarkeit der Ereignisse innerhalb der Spielumgebung:

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

Code 14: Methode zur Visualisierung der Spielumgebung

5.1.5 Würfel Methode

Der folgende Pseudocode zeigt die Funktionsweise der Methode zum Würfeln der Würfel. Würfel müssen nach jedem Wurf, bei Anfang jeder Spielrunde und beim Einsetzen des Neu Würfeln Boni neu gewürfelt werden:

```
roll_dice():
    Würfle die Werte aller gültigen Würfel neu
```

Code 15: Würfel Methode

5.1.6 Methode zur Überprüfung der freigespielten Belohnungen

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

```
check_rewards():
    Überprüfe ob im gelben Feld Belohnungen freigeschaltet wurden
    Schalte im gelben Feld freigeschaltete Belohnungen frei

    Überprüfe ob im blauen Feld Belohnungen freigeschaltet wurden
    Schalte im blauen Feld freigeschaltete Belohnungen frei

    Überprüfe ob im grünen Feld Belohnungen freigeschaltet wurden
    Schalte im grünen Feld freigeschaltete Belohnungen frei

    Überprüfe ob im orangenen Feld Belohnungen freigeschaltet wurden
    Schalte im orangenen Feld freigeschaltete Belohnungen frei

    Überprüfe ob im lila Feld Belohnungen freigeschaltet wurden
    Schalte im lila Feld freigeschaltete Belohnungen frei

    return Punktebelohnung
```

Code 16: Methode zur Überprüfung der freigespielten Belohnungen

5.1.7 Methode zur Generierung des Beobachtungsraumes

Der folgende Pseudocode zeigt die Funktionsweise der Methode zur Generierung des Beobachtungsraumes (_get_obs method). Diese Methode vereinfacht es den Beobachtungsraum zu generieren, welcher jedes mal benötigt wird, wenn ein Schritt in der Umgebung ausgeführt wird:

```
_get_obs():
    Erstelle Numpy Arrays für farbige Felder
    Erstelle Numpy Array für Würfelergebnisse
    Erstelle Numpy Array für ungültige Würfel
    Füge Arraywerte für Boni und Rundenzahl hinzu
    Beobachtungsraum = Verbinde alle Arrays miteinander

    return Beobachtungsraum
```

Code 17: Methode zur Generierung des Beobachtungsraumes

5.1.8 Methode zur Generierung der Aktionsmaske

Der folgende Pseudocode zeigt Die Funktionsweise der Methode zur Generierung der Aktionsmaske (valid_action_mask method). Einsen stehen für gültige Aktionen und Nullen für ungültige. Die Wahlwahrscheinlichkeit des Modells für ungültige Aktionen wird auf Null gesetzt. Jede Aktion wird jeweils einem Kästchen auf dem Spielbrett zugewiesen. Ausnahmen bilden hierbei die Boni Aktionen Neu Würfeln (244), passen bei Extra Wahlen (245) und die ungültige Aktion (246). Dabei entsprechen die Aktionen 0-15, 61-76, 122-137 und 183-198 Kästchen im gelben Feld. Die Aktionen 16-27, 77-88, 138-149 und 199-210 entsprechen Kästchen im blauen Feld. Die Aktionen 28-38, 89-99, 150-160 und 211-221 entsprechen Kästchen im grünen Feld. Die Aktionen 39-49, 100-110, 161-171 und 222-232 entsprechen Kästchen im orangenen Feld, und die Aktionen 50-60, 111-121, 172-182 und 233-243 entsprechen Kästchen im lila Feld. Diese Reichweiten spiegeln jeweils die Größe jedes Feldes in Anzahl an Kästchen wider. Außerdem sind die Reichweiten den verschiedenen Arten von Würfel- oder Boni-Aktionen zuzuordnen, welche zur Ausfüllung von Kästchen führen. Werte von 122-243 stehen für Extra Wahlen[siehe Grundlagen]:

```
valid_action_mask():
    Setze alle Werte auf Eins

    Werte für bereits ausgefüllte gelbe Kästchen = 0
    Werte für gelbe Kästchen ohne passende Würfelergebnisse = 0
```

```
6
7       Werte für bereits ausgefüllte blaue Kästchen = 0
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 Akionen 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 Würfel = 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 18: Methode zur Generierung der Aktionsmaske

5.1.9 Methode zum Hinzufügen von Boni

Der folgende Pseudocode zeigt die Funktionsweise der Methode zum Hinzufügen von Boni (add_reward method). 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 19: Methode zum Hinzufügen von Boni

5.1.10 Methode zum Inkrementieren von Runden

Der folgende Pseudocode zeigt die Funktionsweise der Methode zum Inkrementieren von Runden (increment_rounds method). Diese Methode ist dafür zuständig das Runden-System des Spiels voranzutreiben und zu managen:

```
1  increment_rounds():
2      if Extra Wahl nach eignener 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             Würfle Würfel neu
15         Wahl vom Silbertablett = False
16         Setze alle Würfel auf gültig
```

```
17
18      elif Extra Wahl nach Wahl vom Siblertablett:
19          Extra Wahl nach Wahl vom Silbertablett = False
20          Würfle 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              Würfle Würfel neu
33              Setze drei zufällige Würfel auf ungültig
34      else:
35          Wurf in Runde += 1
36          Würfle Würfel neu
```

Code 20: Methode zum Inkrementieren von Runden

5.2 Künstliche Intelligenz

Die Implementierung besteht aus zwei wesentlichen Klassen. Eine davon ist die Künstliche Intelligenz. In diesem Kapitel werden die Methoden der Künstlichen Intelligenz mithilfe von Pseudocode erläutert. Diese Methoden beinhaltet Methoden der Spielumgebung [siehe Spielumgebung] und Methoden zur Visualisierung, auf die im nächsten Kapitel genauer eingegangen wird.

5.2.1 Methode zu anlernen des Modells

Der folgende Pseudocode zeigt die Funktionsweise der Methode zum anlernen des Modells (model_learn). Es werden Hyperparameter festgelegt, welche genutzt werden, um ein MaskablePPO Modell zu bilden und mithilfe der Spielumgebung zu trainieren. Alternativ kann ein bereits trainiertes Modell geladen und weiter trainiert werden:

```
model_learn(Hyperparameter):
    Initialisiere Spielumgebungen
    Modell = Initialisiere MaskablePPO Modell mit Hyperparamter
    if Modellname in Hyperparameter:
        Modell = Lade Modell mit Modellname
        Modell.Spielumgebungen = Initialisieren Spielumgebungen
        Setze Entropie Koeffizient
    Lerne Modell an
    Setze Gamma für Vorhersagen
    Setze Entropie Koeffizient für Vorhersagen
    Lerne Modell erneut an
    Speichere Modell
```

Code 21: Methode zu anlernen des Modells

5.2.2 Methode zum Vorhersagen mithilfe des Modells

Der folgende Pseudocode zeigt die Funktionsweise der Methode zum Vorhersagen mithilfe des Modells(model_predict). Diese Methode ermöglicht es mithilfe des Modells Vorhersagen über günstige Aktionen in einem gegebenen Zustand der Spielumgebung zu bestimmen und anhand dessen, zusammen mit einer solchen Spielumgebung, einen Spielverlauf zu simulieren:

```
model_predict(Schrittanzahl):
    Lade Modell
    Initialisiere Spielumgebungen und Historien
    Setze Beobachtungsraum
    for i in range(Schrittanzahl):
        Aktualisiere Aktionsmaske
        Aktion = Vorhersage der nächsten Aktion
        Führe Aktion in Spielumgebung aus
        Trage Werte in Historien ein
        Visualisiere Spielumgebung
    Plotte Historien
```

Code 22: Methode zum Vorhersagen mithilfe des Modells

5.2.3 Methode zur Initialisierung der Spielumgebungen

Der folgende Pseudocode zeigt die Funktionsweise der Methode zur Initialisierung der Spielumgebungen(_init_envs). Mit der 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:

```
_init_envs(Anzahl, Punktestände, Fehlversuche):
    _init():
        Spielumgebung = Initialisieren eine Spielumgebung
        Setze Aktionsmasker für die Spielumgebung
        return Spielumgebung
    Initisalisiere Verktorumgebung(Anzahl, _init)
    if not Punkstestände and not Fehlversuche:
        return Vektorumgebung
    if Punktestände and not Fehlversuche:
        Erstelle Variablen für Punkteständehistorie
    if not Punktestände and Fehlversuche:
        Erstelle Variablen für Fehlversuchehistorie
    if Punktestände and Fehlversuche:
        Erstelle Variablen für Punkteständehistorie
        Erstelle Variablen für Fehlversuchehistorie
    return Vektorumgebung, Variablen für Historien
```

Code 23: Methode zur Initialisierung der Spielumgebungen

5.2.4 Methode zum Anwenden der Aktionsmaske

Der folgende Pseudocode zeigt die Funktionsweise der Methode zum Anwenden der Aktionsmaske(mask_fn). Die Methode dient dazu dem Modell die Aktionsmasken der einzelnen Spielumgebungen zu übergeben:

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

Code 24: Methode zum Anwenden der Aktionsmaske

5.3 Darstellung

Dieses Kapitel erläutert die Methoden zur Visualisierung mithilfe von Pseudocode.

5.3.1 Methoden zum Erstellen von Einträgen

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

```
make_fail_entries(Punktebelohnungen, Anzahl, Fehlversuche):
    if Punktebelohnung < 0:
        Inkrementiere Fehlversuche der Umgebung

make_score_entries(Punktebelohnungen, Anzahl, Punktestände):
    if Punktebelohnung > 0:
        Addiere Punktebelohnung zum Punktestand der Umgebung

make_fail_history_entry(Fehlversuche, Fehlversuchshistorie)
    if Umgebung terminiert:
        Hänge Fehlversuche an Fehlversuchshistorie an

make_score_history_entry(Punktestände, Punktehistorie)
    if Umgebung terminiert:
        Hänge Punktestand der Umgebung Punktehistorie an
```

Code 25: Methoden zum Erstellen von Einträgen

5.3.2 Methode zum plotten von Historien

Der folgende Pseudocode zeigt die Funktionsweise der Methode zum plotten von Historien(plot_history). Diese Methode visualisiert die generierten Historien:

```
plot_history(Historie):
    Plotte jeden Eintrag der Historie
    Setze Titel
    Setze Labels
    Zeige Grafik an
```

Code 26: Methode zum plotten von Historien

5.4 Verwendung

Der folgende Code 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 (40000 Schritte lang) simuliert:

```
def main():
    model_learn(total_timesteps=1110000, ent_coef=0.1, gamma=1,
     model_name="maskableppo_ganzschoenclever_193avg_v3")

    model_predict(n_envs=1, render=True, n_steps=40000,
     model_name="maskableppo_ganzschoenclever")
```

Code 26: Anwendungsbeispiel

6 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 des Modells und der Spielumgebung beinhaltet, um sicherzustellen dass das Vorhaben im Rahmen der Vorgaben umsetzbar ist und, um Einblicke in möglichen Problemstellungen des Projektes zu erhalten. Danach wurde der Prototyp Stück für Stück erweitert, bis schließlich das komplette Spiel mit allen Funktionalitäten umgesetzt worden ist und das Modell damit trainiert werden konnte.

6.1 Trainingshistorie

6.1.1 Prototype

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

Die folgende Abbildung zeigt das erste erfolgreiche Training des Modells:

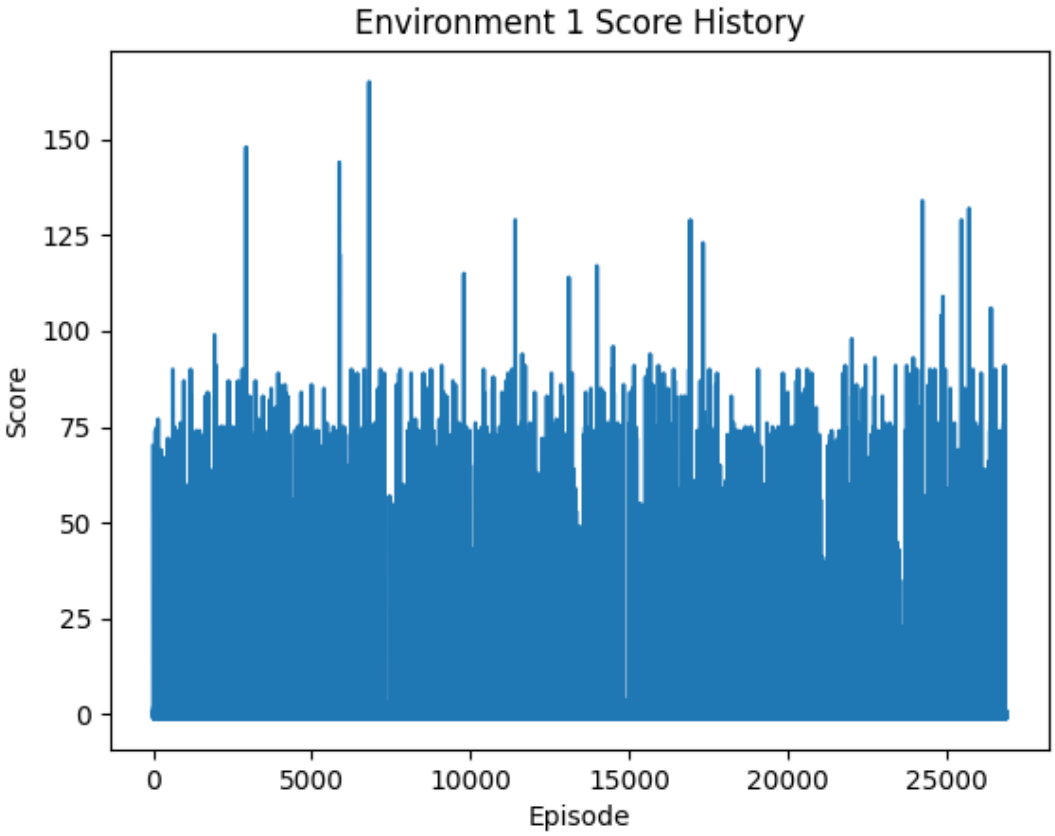


Abb. 9: Erstes erfolgreiches Training

In der Abbildung erkennt man, dass das Modell innerhalb einer Episode (eines Spiels) des öfteren in einen Punktebereich von 70 zu kommen scheint. Teilweise erzielt das Modell sogar über 150 Punkte. Fragwürdig ist, dass die maximale Punktezahl eigentlich bei 86 liegt und das Modell dennoch zu diesem Zeitpunkt zum Teil Punktewerte erzielt, die höher liegen. Dies ist darauf zurückzuführen, dass die Spielumgebung erzielte Punkte zum Teil mehrfach verwertete und somit mehr Punkte zu erspielen waren, als vorgesehen.

Die folgende Abbildung zeigt das erste erfolgreiche Training des Modells in 100 Schritten:

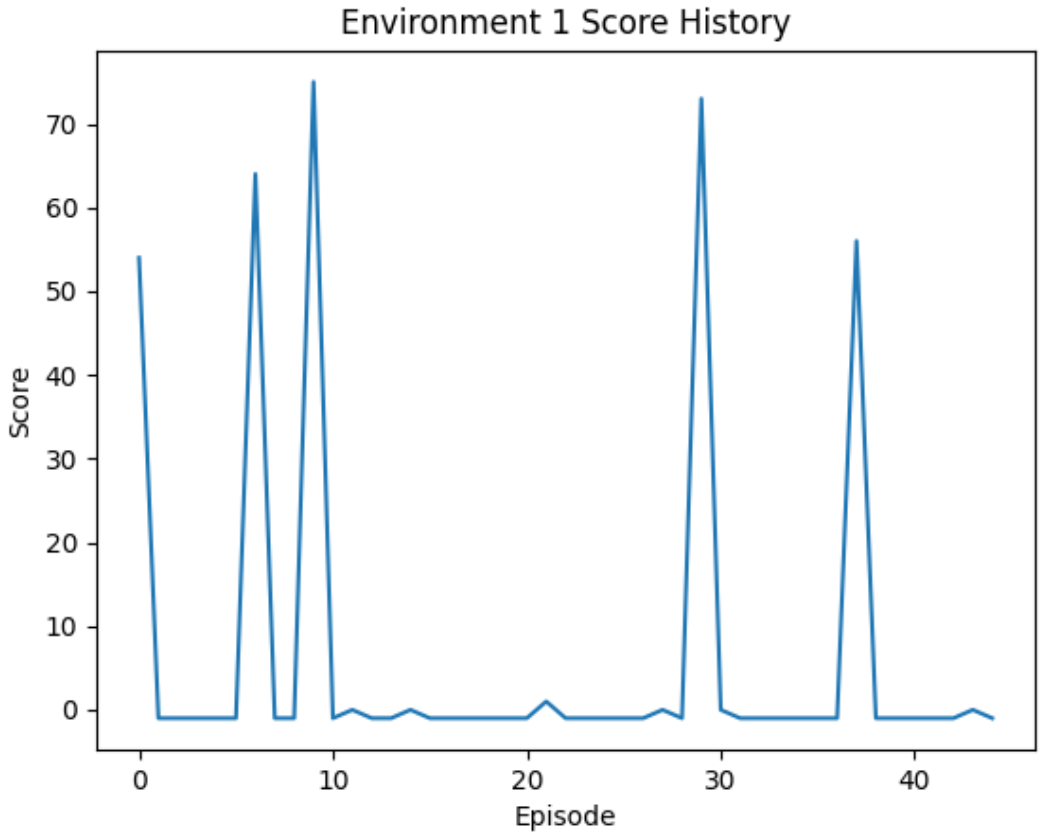


Abb. 10: Erstes erfolgreiches Training 100 Schritte

In dieser Abbildung erkennt man, dass das Modell zwar zum Teil 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 des Projektes das Spiel unverzüglich beendet, wenn eine ungültige Aktion gewählt wird. Daraus kann man schließen, dass das Modell überwiegend ungültige Aktionen wählt, da bei nur 100 Schritten bereits über 40 Spiele abgeschlossen werden. Vorgesehen sind zu diesem Zeitpunkt pro Spiel zehn Spielschritte. Somit wurden über vier mal so viele Spiele abgeschlossen als unter optimalen Bedingungen vorgesehen. Fragwürdig ist hierbei auch, dass das Modell in manchen Episoden mindesten 50/86 Punkten erzielt und in den anderen nur 0 bis 2. Die Punkte werden in Abständen von 10 bis 20 Punkten erspielt, daher handelt es sich hierbei nicht um vereinzelte Erfolge im engeren Sinne.

6.1.2 Training mit und ohne Aktionsmaske

Die folgende Abbildung zeigt die erzielten Punkte des Modells ohne Aktionsmaske in 200 Schritten:

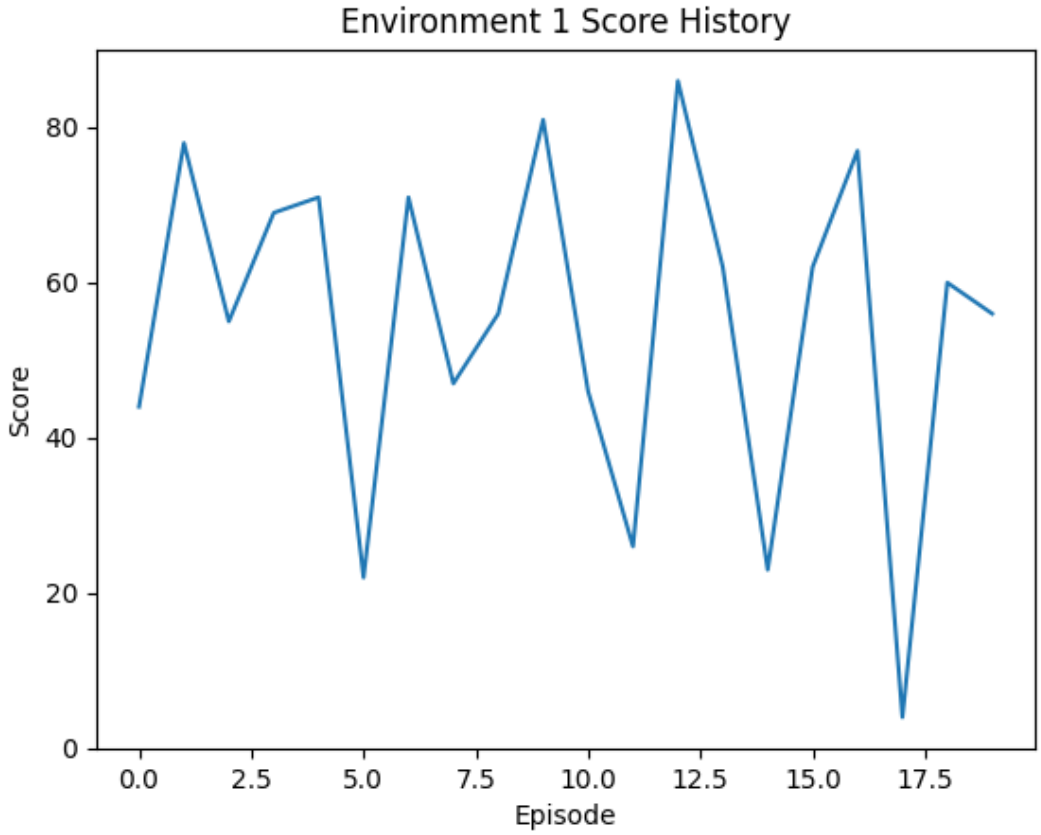
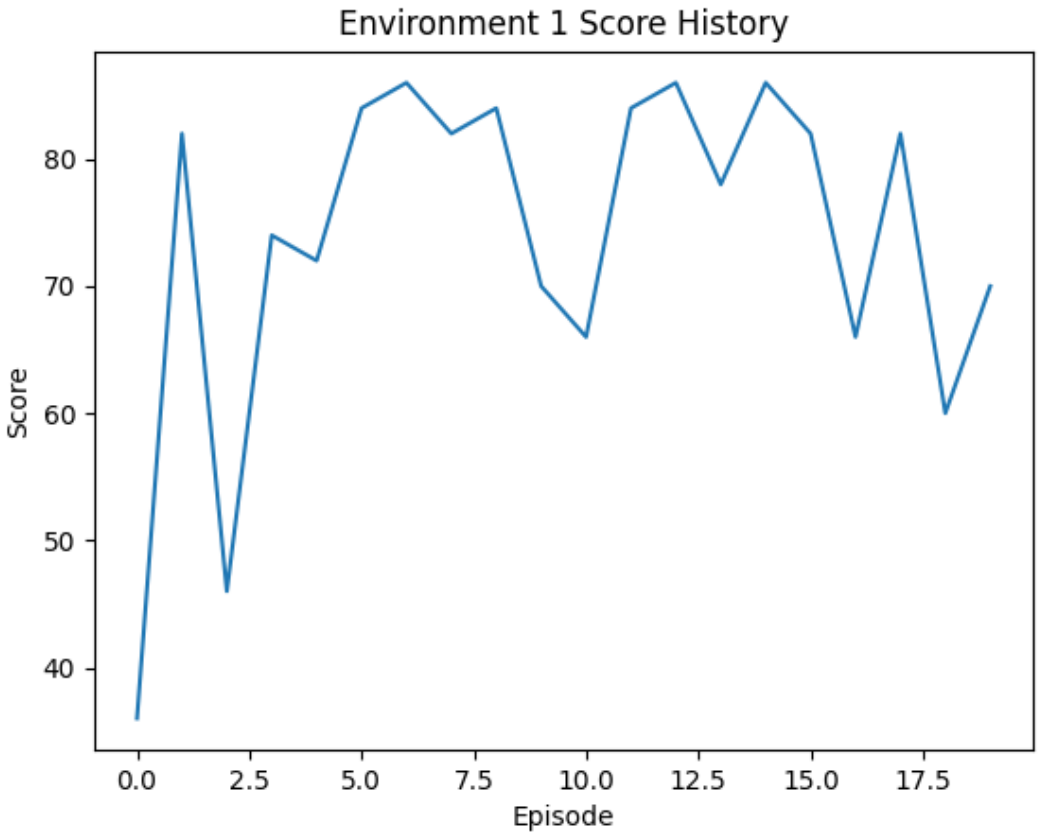


Abb. 11: Training ohne Aktionsmaske

Das Training erfolgte zunächst ohne die Verwendung einer Aktionsmaske. Das anfänglicher 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 eine Trainingsverfahren mit negativer Belohnung bei ungültigen und positiver Belohnung bei gültigen Aktionen gewählt. Diese Belohnungen sind in dieser Abbildung nicht zu sehen, da sie herausgefiltert worden ist. Die Abbildung zeigt nur die aufsummierten Belohnungen, welche 10 oder mehr Punkte betragen. Insgesamt zeigt sich eine verhältnismäßig gute Performance bei der das Modell im Durchschnitt um die 50 Punkte erzielt. Bedenkt man, dass das Maximum bei 86 Punkten liegt ist dieses Ergebnis durchaus vertretbar. Das Modell erzielt somit im Durchschnitt ungefähr 60 Prozent der maximalen Punktezahl.

Die folgende Abbildung zeigt die erzielten Punkte des Modells mit Aktionsmaske in 200 Schrit-



ten:

Abb. 12: Training mit Aktionsmaske

In dieser Abbildung sieht man, dass die Performance im Gegensatz zu der Variante ohne Aktionsmaske zugenommen hat. Das Modell erzielt im Durchschnitt ungefähr 65 Punkte was einem Zuwachs von circa einem Drittel entspricht. Die erzielten Punkte steigen somit von ungefähr 60 Prozent des Maximums auf 75 Prozent. Zwar lässt sich sagen, dass die Performance des Vorgängermodells passabel war, allerdings erzielt die Variante mit Aktionsmaske ohne größere Umstände ein wesentlich besseres Ergebnis. Deshalb wurde das Projekt ab diesem Zeitpunkt mit Aktionsmaske fortgesetzt.

6.1.3 Training mit zwei und mit vier Würfeln

Die folgende Abbildung zeigt die ungültigen Züge einer Spielsimulation von 200 Schritten mit zwei Würfeln:

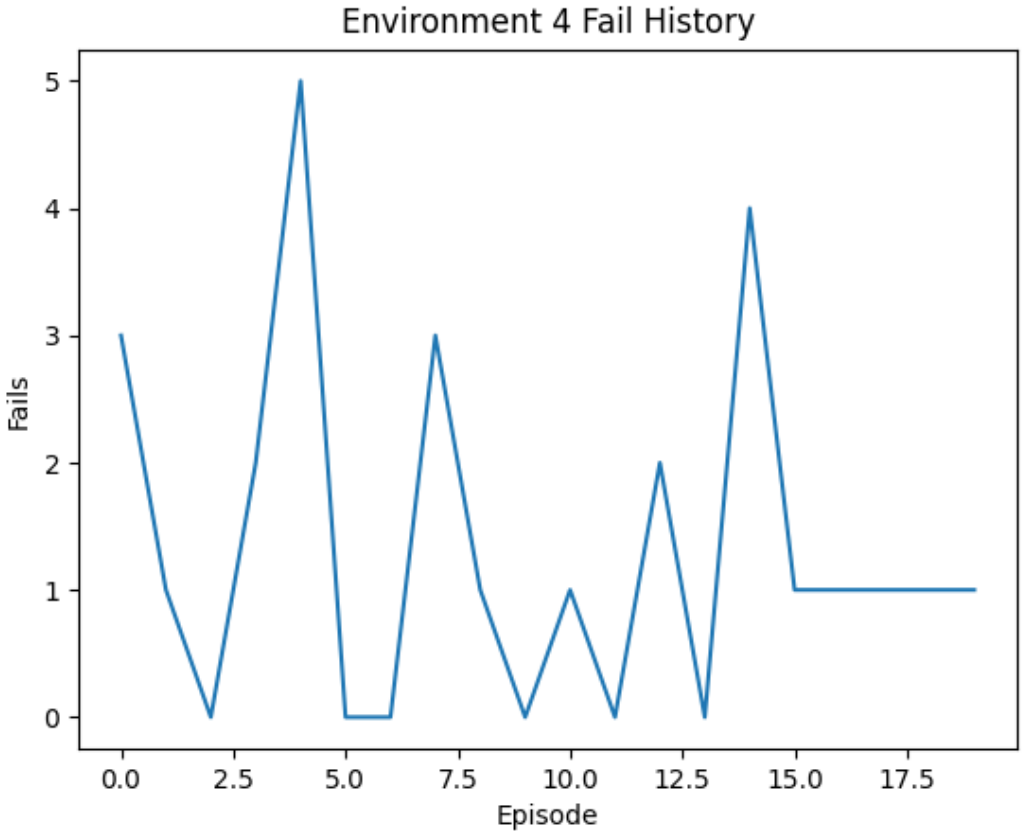


Abb. 13: Ungültige Züge mit zwei Würfeln

In der Abbildung ist zu erkennen, 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. Was passiert allerdings, wenn es keine gültige Aktion gibt? Die Antwort auf die Frage ist: Das Modell wählt eine zufällige Aktion und berücksichtigt die Aktionsmaske nicht. Da nur zwei Würfel zur Verfügung stehen kommt es oft dazu, 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.

Die folgende Abbildung zeigt die ungültigen Züge einer Spielsimulation von 200 Schritten mit vier Würfeln:

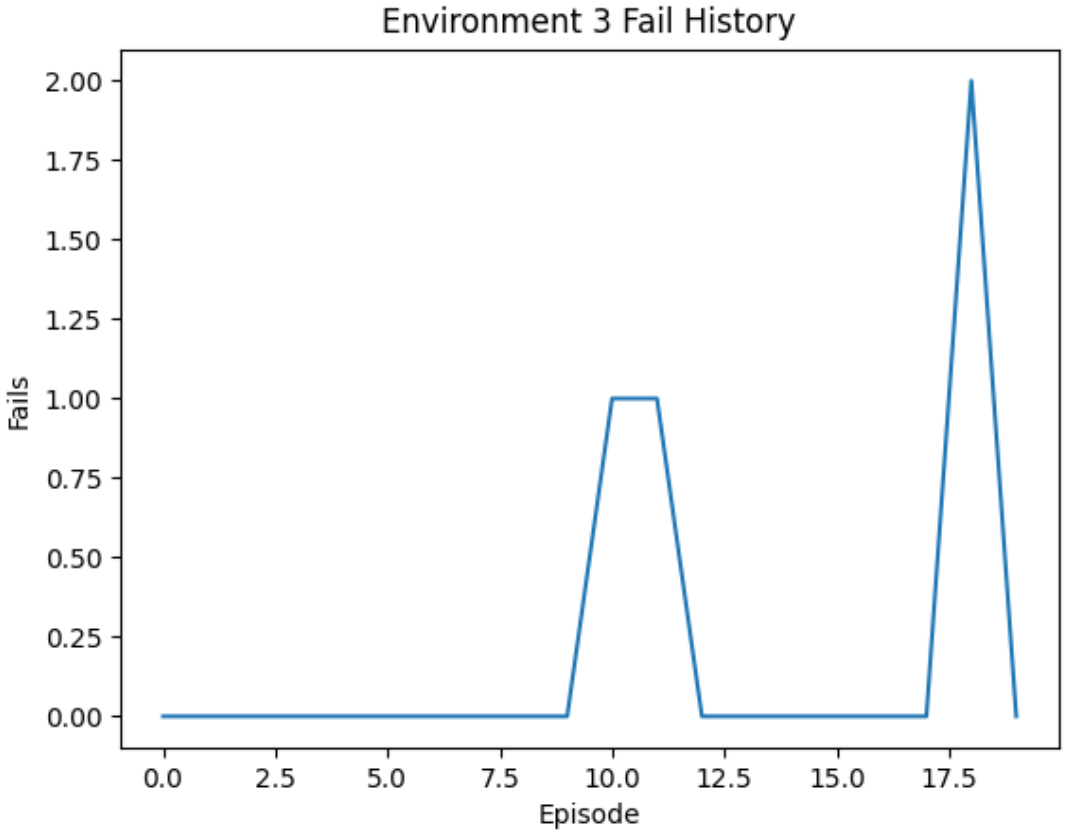


Abb. 14: Ungültige Züge mit vier Würfeln

In der Abbildung erkennt man, 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 dem Modell nichts anderes übrig bleibt, als eine ungültige Aktion zu wählen, allerdings ist die Wahrscheinlichkeit dafür bei vier Würfeln deutlich geringer.

6.1.4 Optimiertes Training

Die folgende Abbildung zeigt die erzielten Punkte des Modells nachdem eine Vielzahl an Optimierungsversuchen durchgeführt worden ist:

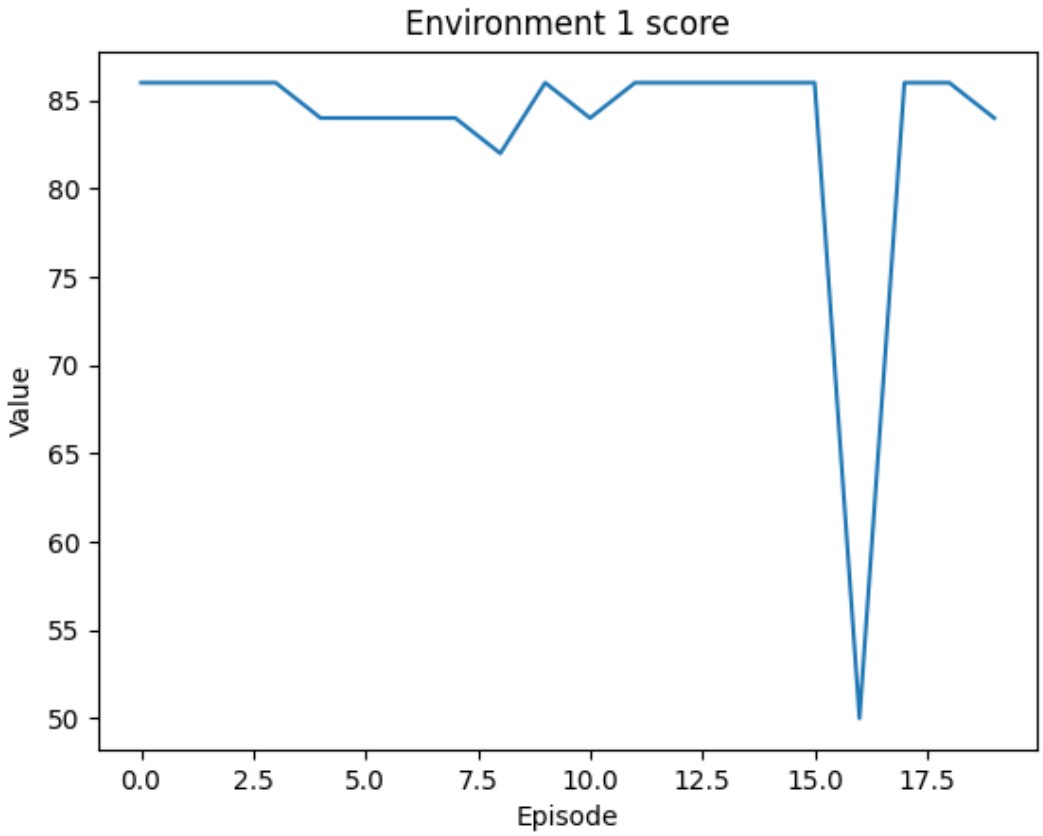


Abb. 15: Optimiertes Training

Man erkennt, 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.

6.2 Erstes Training mit allen Feldern und Boni

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

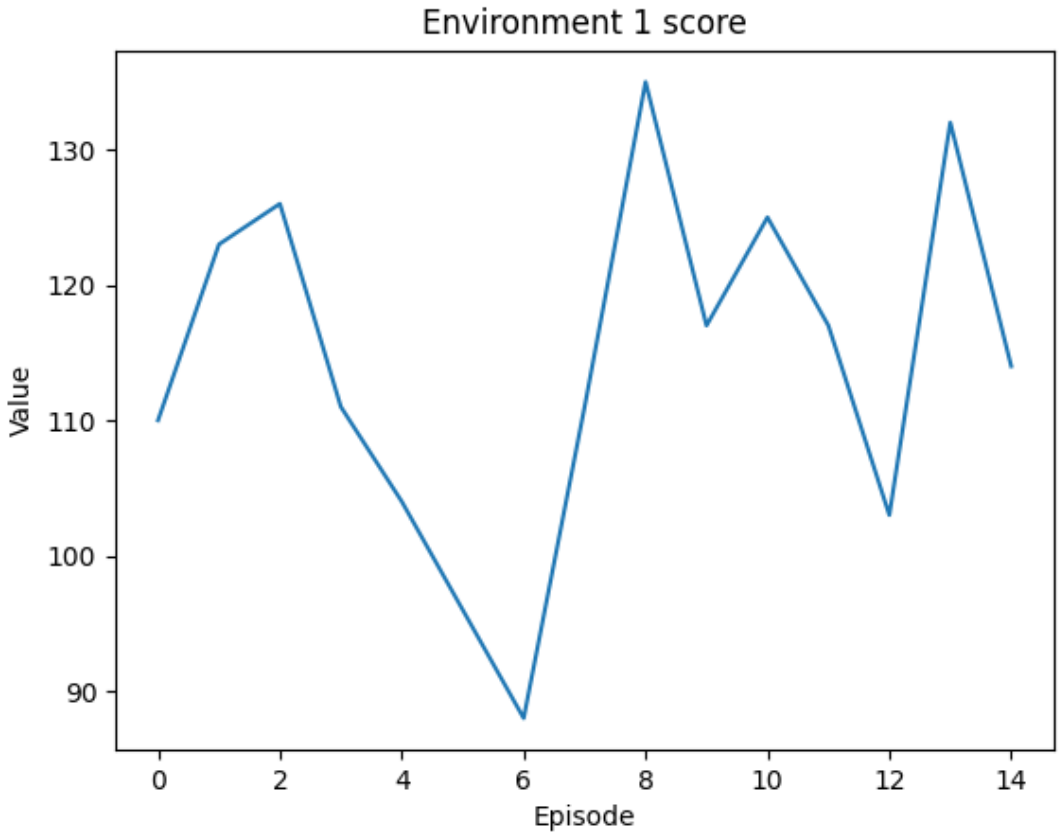


Abb. 16: Erstes Training mit allen Feldern

Man kann erkennen, 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 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.

6.3 Finale Ergebnisse

6.3.1 Performance

Die folgende Abbildung zeigt die erzielten Punkte des Modells nach eingängigem Training:

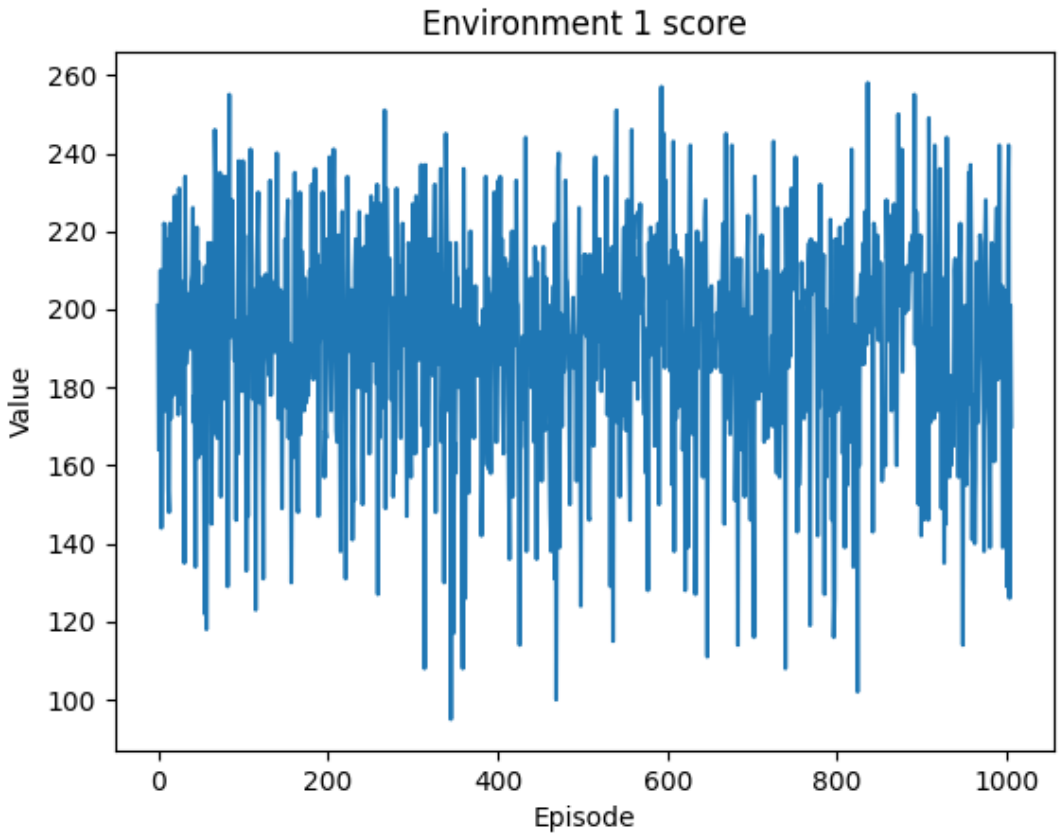


Abb. 17: Performance 40000 Schritte

Das Modell erzielt über mehr als 1000 Episoden eine durchschnittliche Punktezahl von ungefähr 205 Punkten. Das ist beachtlich, da menschliche Spieler in einem Versuch im Durchschnitt lediglich 160 Punkte erzielen konnten. Der Median liegt bei ...

Die folgende Abbildung zeigt die ungültigen Züge des genannten Modells:

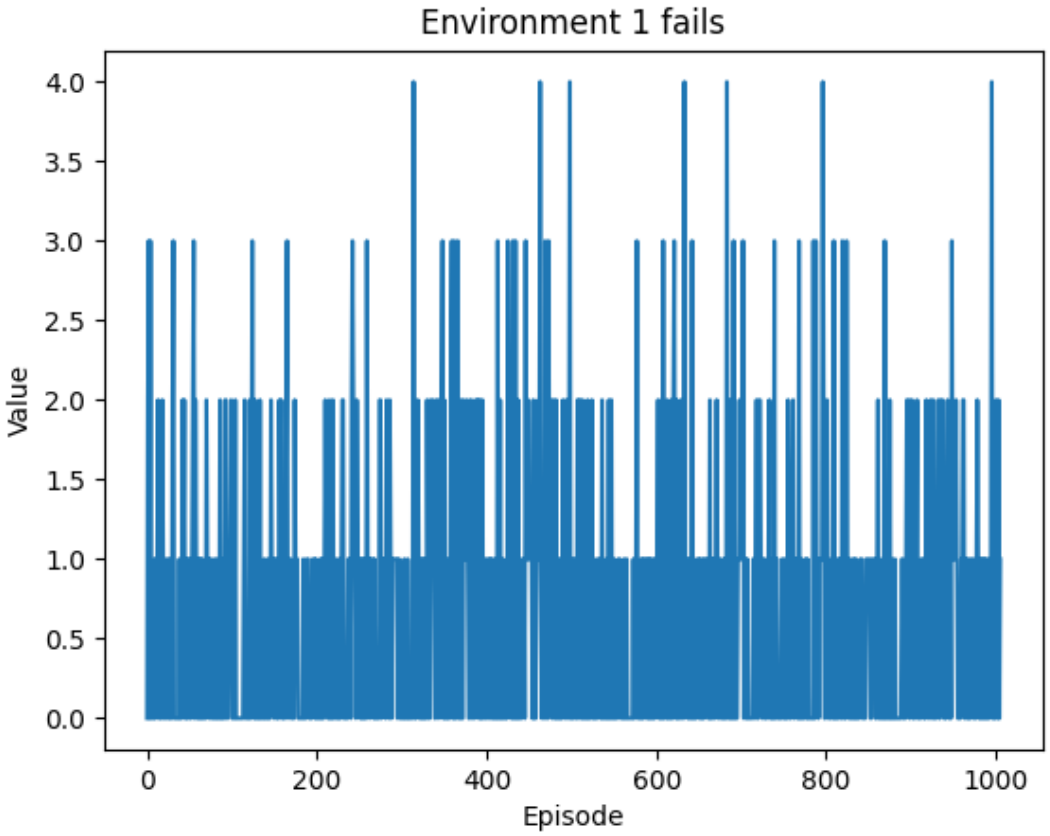


Abb. 18: Ungültige Züge 40000 Schritte

Man erkennt, dass das Modell in den meisten fällen maximal einen ungültigen Zug macht. Teilweise sind noch 2 ungültige Züge pro Episode zu erkennen und noch seltener drei oder vier. Angesichts dessen, dass das Modell pro Spielrunde ungefähr 40 Züge gemacht hat und häufig Würfel bei der Wahl ungültig gemacht werden, ist das ein beachtliches Ergebnis. Das Modell hat gelernt solche Fälle effizient zu vermeiden.

6.3.2 Hyperparameter

Die finalen Hyperparameter sind:

Das 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. Datenpaketen mit 2048 Aktions-Zustands-Paaren, welche jeweils 5 mal verwendet werden und dabei in Batches von 128 Schritten aufgeteilt werden, sowie einem Entropie Koeffizienten für das verfestigen der gelernten Schritte von 0. 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 auszunutzen/auszubeuten. Dieser Prozess wurde dann zwei weitere male mit doppelt so vielen Zeitschritten wiederholt. Zunächst betrug die durchschnittliche Punktezahl der Modells annähernd 190, danach 201 und schließlich 205 Punkte.

6.3.3 ChatGPT 4

ChatGPT wurde verwendet, um den Prototypen des Projektes bis zu seinem ersten erfolgreichen Einsatz zu implementieren. Dabei hat es sich als besonders nützlich erwiesen, um lauffähigen Code mit den gewünschten, relativ standardmäßigen Methoden zu implementieren. Dies ist gerade bei der Erstellung eines ersten Prototypen vorteilhaft, oder um die Machbarkeit des Projektes zu überprüfen. Im späteren Verlauf wurden von ChatGPT generierte Inhalte nicht mehr direkt übernommen, sondern es wurde nur benutzt, um Auskunft über geeignete oder verwendete Technologien zu geben, Ideen zu sammeln oder Syntax zu erfragen. Auch für diese Aufgaben stellte es sich als sehr nützlich heraus, solange die Fragen nicht zu spezifisch oder komplex waren. Ist die Problemstellung zu komplex oder zu spezifisch, scheint ChatGPT Schwierigkeiten dabei zu bekommen eine vollständige oder geeignete Antwort zu liefern. Außerdem ist die Datenbank des ChatGPT 4 Modells vom September 2021, was den Nutzen für neuere Sachverhalte erschwert.

Literaturverzeichnis

HOCHSCHULE COBURG

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 10.11.2023

Ort, Datum / *Place, date*