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: 01.11.2023

Betreut durch:

Prof. Dr. Mittag, Florian, Hochschule Coburg

Inhaltsverzeichnis

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 auch KI zunehmend an Bedeutung, Tendenz steigend. Im Jahr 2023 ist eines der am meisten präsenten neuen Tool ChatGPT 4. Dieses Tool ist ein Chat-Bot, welches es dem Benutzer ermöglicht mit ihm zu kommunizieren und ihm, dem Chat-Bot, vor allem 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 KI schon seit langem eine zentrale Rolle ein. In den meisten spielen gibt es eine oder meist 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 (der sogenannte Aktionsraum) für den Spieler und die vielen Zusammenhänge des Belohnungssystems im Spiel zu Stande. Das Spiel weist allerdings auch eine hohe Stochastizität auf, welche die Komplexität zusätzlich erhöht.

Interessant ist wie man für ein solch komplexes Spiel einen Bot oder eine KI 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

Ziel der Arbeit ist es einen Bot beziehungsweise eine KI zu entwickeln, welche das Spiel "Ganz schön clever" möglichst effizient spielen kann. Dabei soll analysiert und erforscht 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 gewünschten Ergebnis führen. Ziel ist es auch einen solchen 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 Aufschlüsse 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 KI gespielt werden kann. Dieser Prozess wird analysiert und bewertet um Schlüsse darüber zu ziehen was vorteilhaft und was nachteilig für ein solches Vorhaben ist.

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: [Google Play Store, de.brettspielwelt.ganzschoenclever]

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 anderen Würfel betrachtet werden. Wenn bestimmte Bedingungen erfüllt sind kann der Spieler einen Würfel wählen und das entsprechende Subfeld 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 Subfelder 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 finden ist.
Es gibt zwei Arten von Belohnungen im Spiel. Punktebelohnungen und Boni. Bei Punktebelohnungen handelt es sich um Punkte welche auf dem 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 mit zunehmender Anzahl der ausgefüllten blauen Subfelder. Die Anzahl der Punkte findet sich oben im blauen Feld. Im grünen Feld ebenso mit zunehmender Anzahl der ausgefüllten grünen Subfelder. Die Anzahl der Punkte findet sich auch hier oben im grünen Feld. Im orangenen und lila Feld muss man dafür lediglich ein Subfeld ausfüllen. Die Punktebelohnung entspricht der Augenzahl des entsprechenden ausgefüllten Subfeldes. 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 Subfeldern beziehungsweise am Rande von Spalten und Zeilen eingezeichnet und können freigeschaltet werden indem man diese 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.

Extra Wahl: Bei der Extra Wahl wird es dem Spieler ermöglicht am Ende seiner Würfe beziehungsweise nachdem er einen Würfel vom Silbertablett des Gegners gewählt hat erneut Würfel zu wählen und die entsprechenden Felder dafür anzukreuzen. Würfel die so gewählt wurden

können mithilfe der Extra Wahl Boni im selben Wurf nicht erneut gewählt werden. Es können alle Würfel gewählt werden, nicht nur die, welche unter normalen Umständen gültig zur Wahl stehen. 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 die im Kreis angeordnet sind.

Gelbes Kreuz: Ermöglicht es dem Spieler direkt nach erhalten der Boni eines der gelben Subfelder 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 Subfelder 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 Subfeld auszufüllen. Das Symbol ist ein Kreuz auf grünem Hintergrund.

Orangene Vier: Ermöglicht es dem Spieler direkt nach erhalten der Boni das nächste freie orangene Subfeld 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 Subfeld 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 Subfeld 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 Subfeld 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 Subfeld auszufüllen und ob er ein gültiger Würfel ist:

Ist ein Würfel ungültig kann dieser nicht gewählt werden. Würfel werden ungültig indem sie in der aktuellen Runde 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 der Runde, nach Abschluss des dritten Wurfes, sowie nach der Wahl des Würfels vom Silbertablett,als auch nachdem Wahlen (nicht nach jeder einzelnen Wahl, sondern wenn Extrawahlen erfolgt sind und das Spiel danach fortgesetzt wird) mit dem Extra Wahl Boni erfolgt sind.

Jedes Feld hat seine eigenen Regeln, die bestimmen, wann ein Subfeld beziehungsweise Kästchen in ihm ausgefüllt werden darf. Beim gelben Feld muss die Augenzahl des Würfel mit der Zahl des Subfeldes übereinstimmen. Beim Blauen Feld muss die Summe der Augenzahlen des blauen

und des weißen Würfels mit der Zahl des Subfeldes übereinstimmen. Beim grünen Feld muss die Augenzahl des Würfels größer oder gleich der Zahl im Subfeld sein. Zudem kann immer nur das nächste freie Feld ausgefüllt werden, beginnend von links. Im orangenen Feld kann immer das nächste Subfeld eingetragen werden. Auch hier beginnend von links. Beim lila Feld muss die Augenzahl des Würfels größer sein als die Zahl im zuletzt ausgefüllten Subfeld. Eine Ausnahme bildet hier der Fall in dem eine sechs im zuletzt ausgefüllten Subfeld 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 sozusagen aus. Auch hier gilt die Reihenfolge von links nach rechts.

3.1.2 Machine Learning

"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 aus beliebigen Daten des selben oder eines ähnlichen Sachverhaltes, der beim Training vorhanden war, zu treffen. Ein besonders weit verbreiteter Anwendungsfall ist die Herleitung von Kundenverhalten und möglicher Optimierungsmöglichkeiten für den Verkauf. Wenn man ein Programm mithilfe von maschinellem Lernen trainiert hat, nennt man dieses dann Modell. Ein solches Modell wird häufig erst auf allgemeinen Datensätzen und später auf immer spezifischeren 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 Schlüsse über den gesamten Sachverhalt zu schließen und auf Grund dessen zu agieren zu können oder zumindest Vorhersagen über diesen zu treffen [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 Aktienmark einzuschätzen. Aber auch 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 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 Felder des Spielbrettes zum ausfüllen als Aktionen aus. Außerdem entscheidet er ob er bestimmte Boni 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:

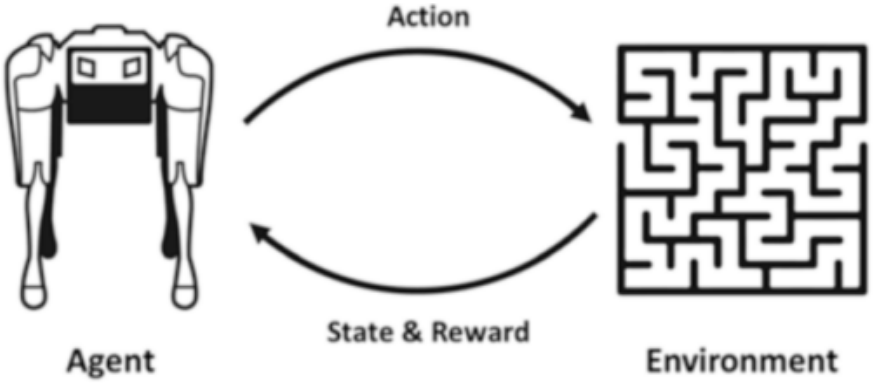

Abb. 2: Reinforcement Learning
Quelle: [RL, Seite 12]

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 einiges 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 die unterschiedlichen Gewichtungen der Neuronen und ihre Position im Netz.

Die folgende Abbildung zeigt ein solches Multilayer Perceptron:

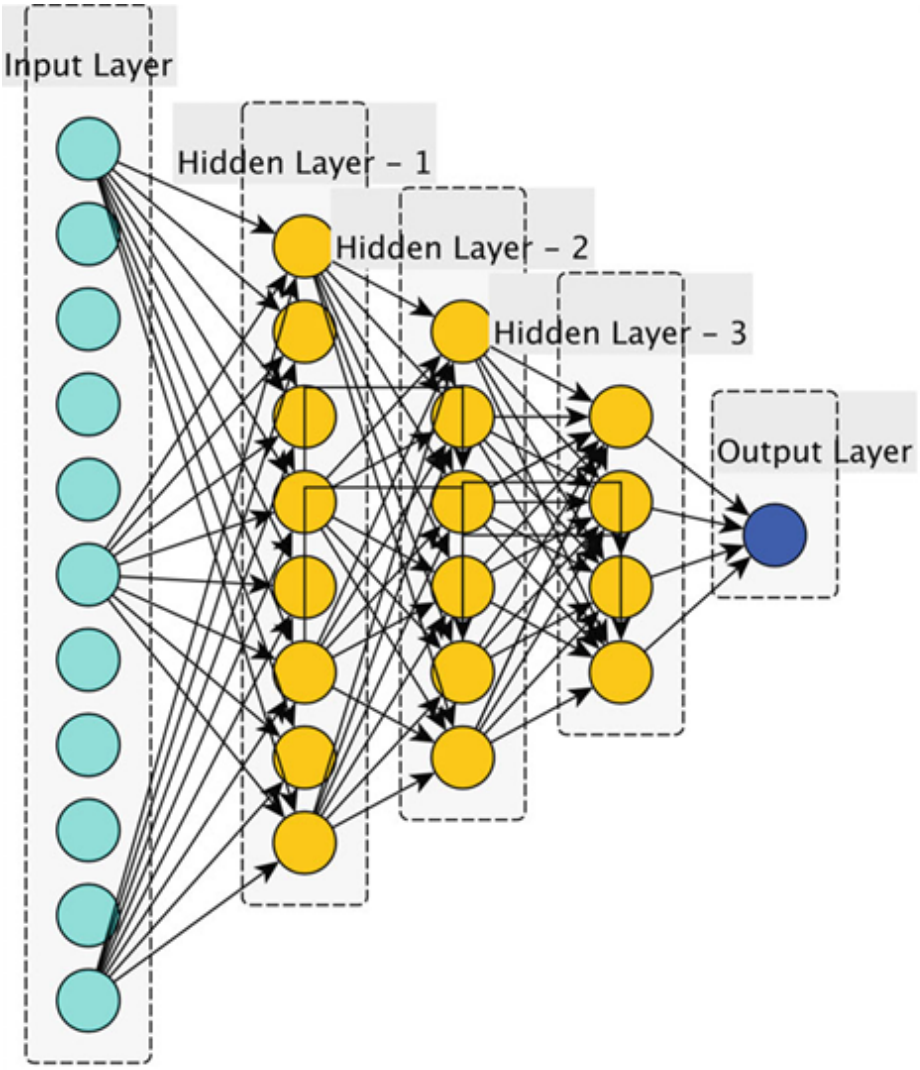

Abb. 3: Multilayer Perceptron
Quelle: [DRL, Seite 78]

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 implementie-

ren 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 Performance zu ermöglichen. Dies ermöglicht es mehr Updates mit weniger Gesamtdaten durchzuführen, was vor allem hilfreich ist, wenn nicht genügend Daten zur Verfügung stehen.

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.

Bei der Trusted Region Policy Optimization gibt es eine vertrauenswürdige Region innerhalb 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 abgeschnitten im übertragenen Sinne und es kommt zu einer abgeschwächten Änderung der Policy.

PPO ist ein verhältnismäßig simpler, einfach 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 zum Machine Learning verwendet werden. 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 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 Stable Baselines 3, welches 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 benutzt, damit diese anschließend mit 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. 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 die PPO um die Funktion 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 dieser 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 Prototyps auch zum generieren von Programmcode 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 hierbei 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 Ereignisses selbst zu analysieren. Dies soll mithilfe geeigneter, schlüssiger und 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 nach und nach erweitert werden soll, bis das Spiel mit all seinen 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.

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 bei 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 per Zufall 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 Subfelder (beziehungsweise Kästchen) auf diesem Feld anzukreuzen [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]. Damit ist sowohl das Erhalten, Speichern, sowie die Benutzung der Boni gemeint.

Ein Mechanismus, welcher Würfel, welche mithilfe der Extra Wahl Boni gewählt wurden 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 einen oder mehrere Faktoren, welche das Ausmaß von Erkundung und Ausbeutung steuern sollen.

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:

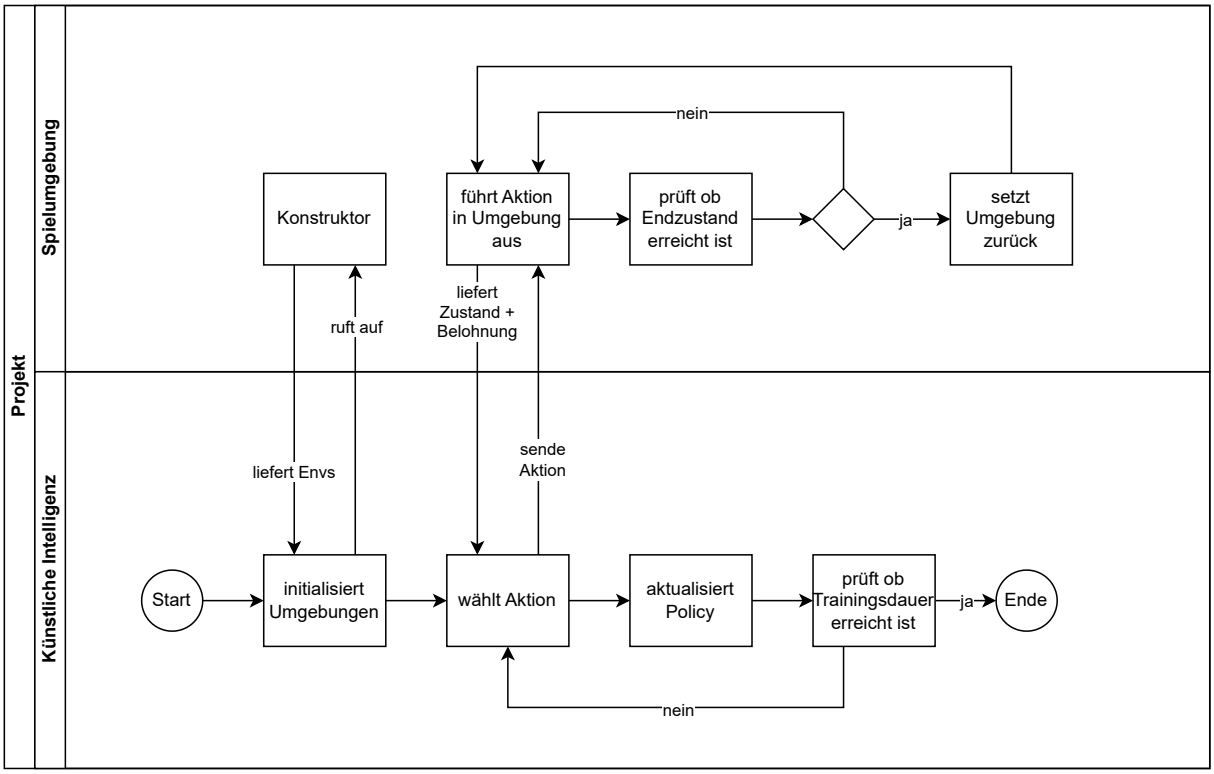

Abb. 4: Swimlane-Diagramm des Projektes

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.

4.2.1 Die Spielumgebung

Die folgenden beiden Abbildungen zeigen 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:

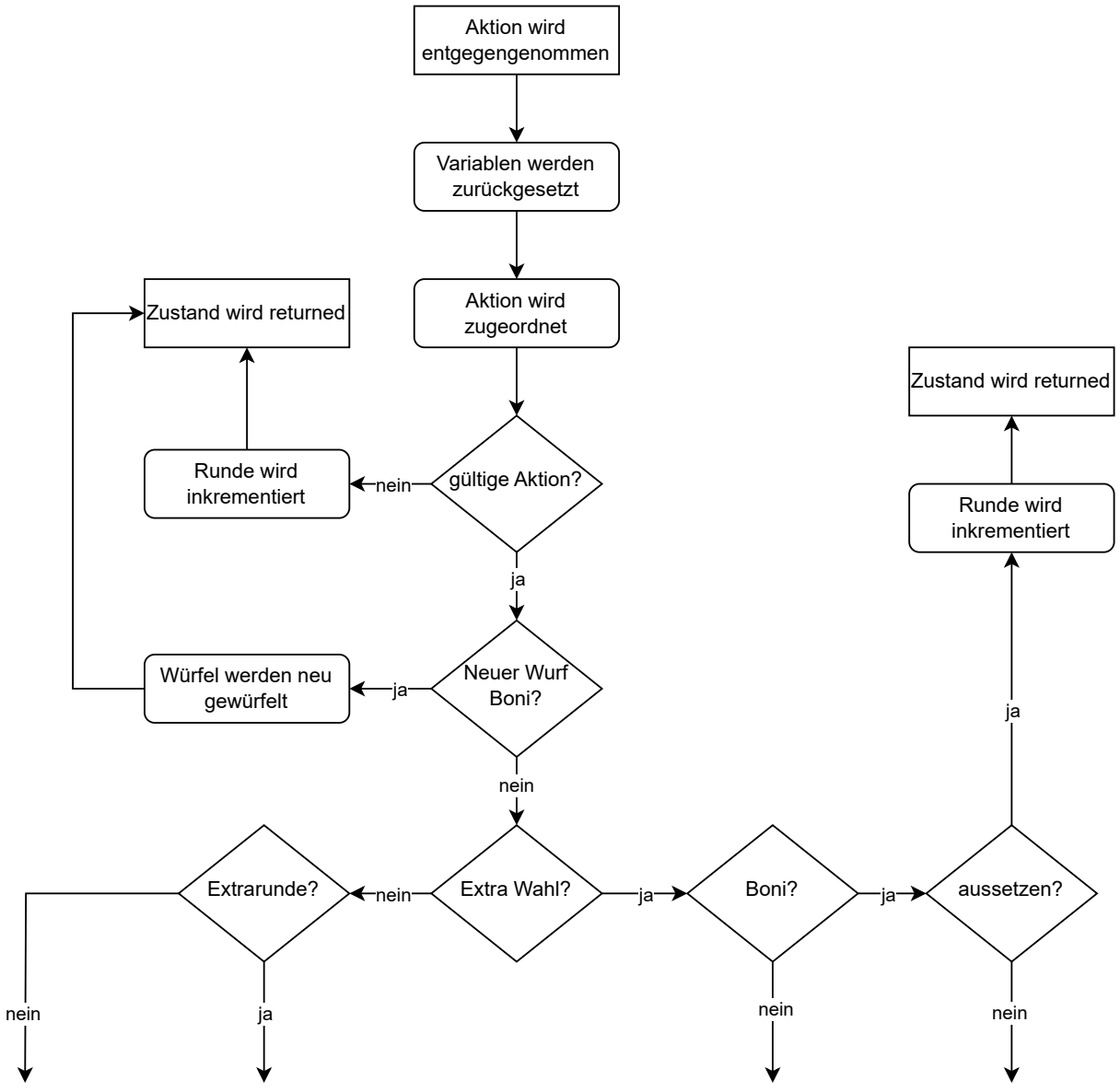

Abb. 5: Ablauf-Diagramm der Schritt Funktion 1

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.

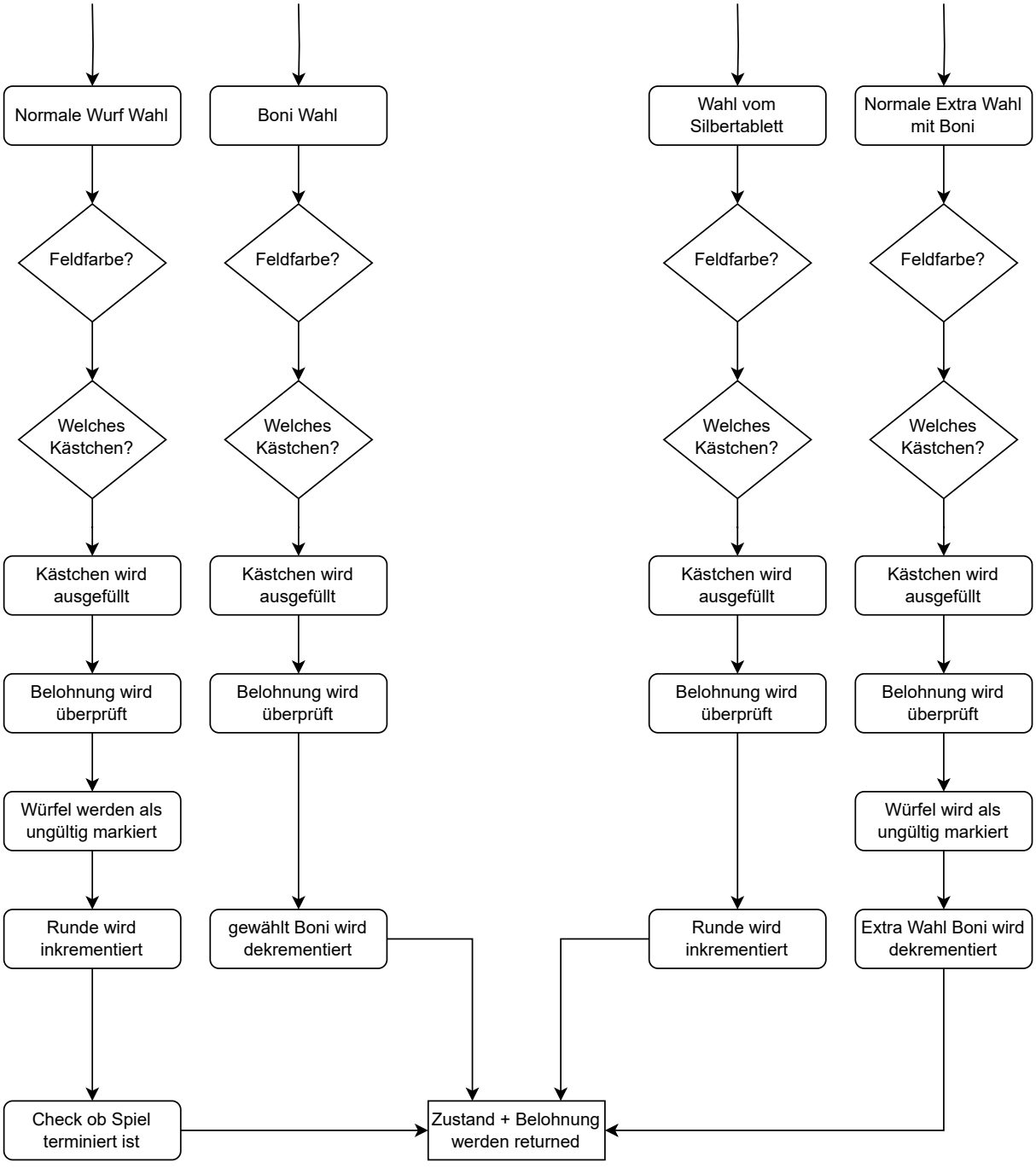

Abb. 6: Ablauf-Diagramm der Schritt Funktion 2

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 allerdings 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) 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:

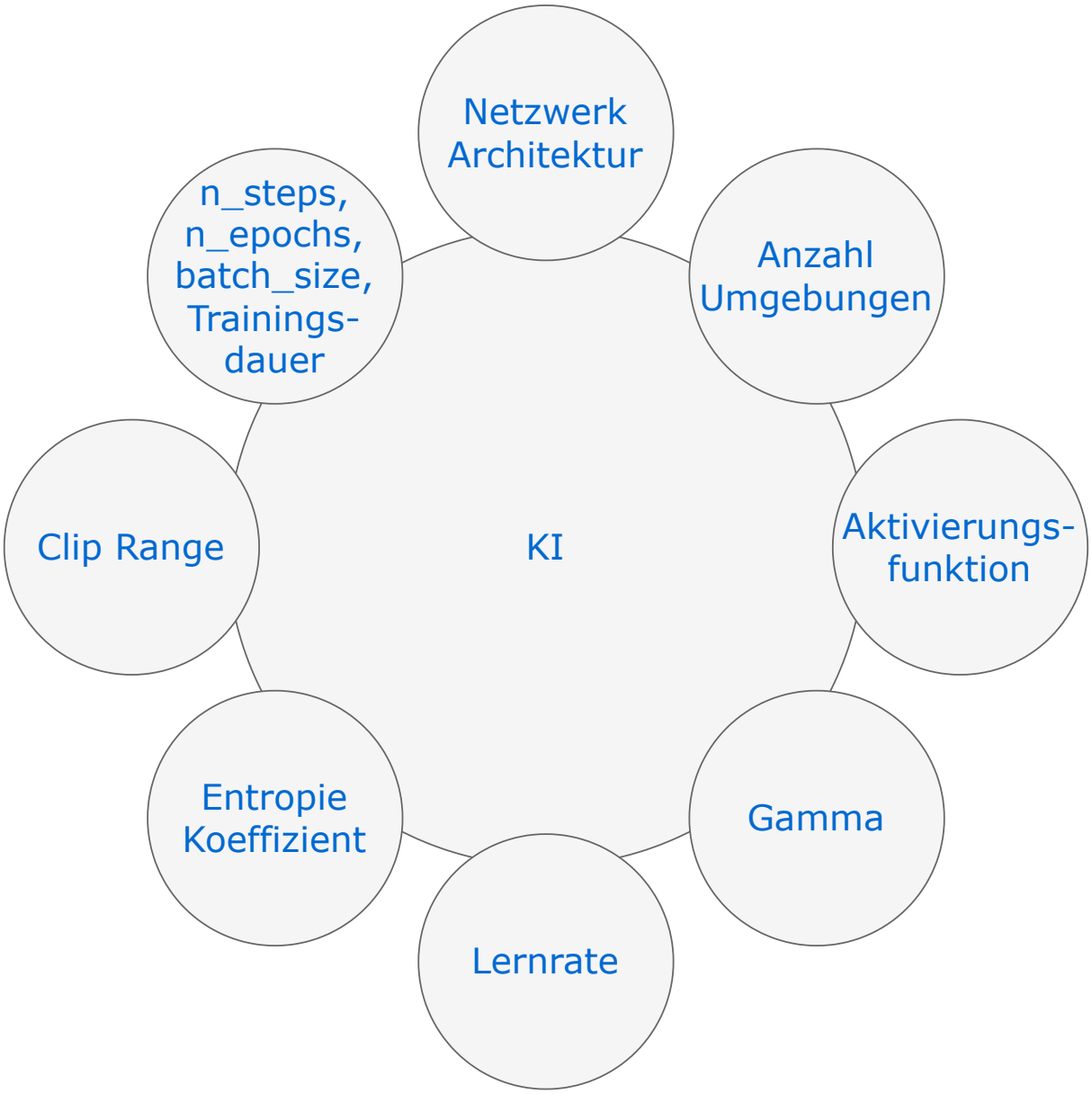

Abb. 7: Hyperparameter der Künstlichen Intelligenz

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 fest wie viele Schichten das Neuronale Netz, welches für die Abbildung der Policy und der Wertfunktion 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 und das Überanpassung dadurch nicht zu hoch steigen. 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. Komplexe Netze generalisieren somit 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 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) 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:

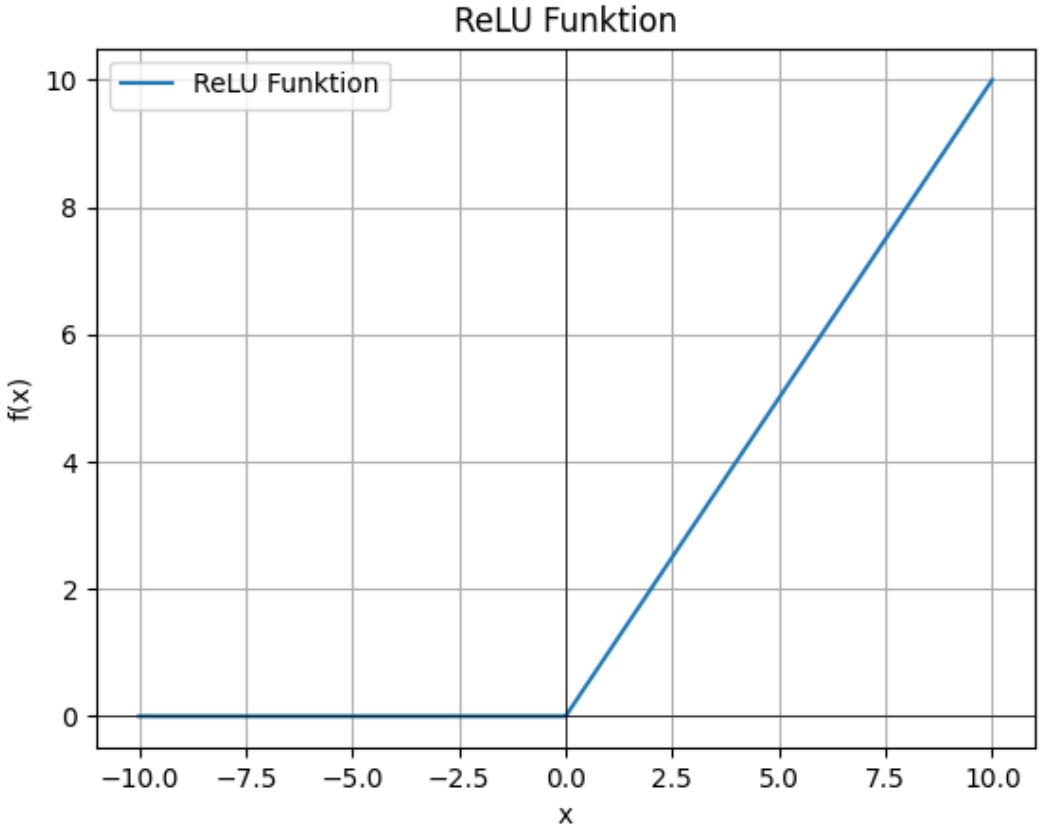

Abb. 8: ReLU Funktion

Gamma: Gamma legt fest wie stark Zukünftige Belohnungen wertgeschätzt werden. Es fungiert als eine Art Multiplikator. Belohnungen werden pro Spielschritt, die sie in der Zukunft 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 Bedeutung sind. Zu niedrige Werte von Gamma können wiederum dazu führen, dass der Agent nicht Zukunftsorientiert handelt und somit kein gutes Gesamtergebnis erzielt. 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 ausprobiert.

nSteps: nSteps legt fest, wie viele Schritte 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 Funktionen 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. Bei invalid_dice handelt es sich um ein Dictionary, welches die einzelnen Würfelfarben und einen boolschen Wert für die Gültigkeit des farbigen Würfels beinhaltet.

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 Felder 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 das beziehungsweise die entsprechenden Kästchen für die Belohnung ausgefüllt, wurde die Belohnung bereits ausgeschüttet und wird es nicht erneut, bis das Flag nach dem Spiel zurückgesetzt wird:

```
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. Diese werden aufaddiert, sobald Punkte im entsprechenden Feld erzielt worden sind und am Ende genutzt, um den Wert der Fuchs Boni zu bestimmen:

```
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. Die Attribute low_bound und high_bound definieren die Grenzen und shape definiert 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.

5.1.2 Methoden

5.1.3 Einzelne Methoden...

5.2 Künstliche Intelligenz

5.2.1 Model Learn

5.2.2 Model Predict

5.2.3 Init Envs

5.3 Darstellung

5.3.1 Make Entry

5.3.2 Plot History

5.4 Verwendung

6 Ergebnisse

6.1 Trainingshistorie

6.1.1 Version 1.1.0

6.1.2 Version 2.0

6.1.3 Version 3.0

6.1.4 Version 4.0

6.2 Finale Ergebnisse

6.2.1 Performance

6.2.2 Hyperparameter

6.2.3 ChatGPT 4

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 01.11.2023

Ort, Datum / *Place, date*