IM Fakultät
Informatik und Medien

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

im Studiengang Informatik
der Fakultät Informatik und Medien
der Hochschule für Technik, Wirtschaft und Kultur Leipzig

Evaluierung von Reinforcement Learning-Algorhitmen anhand eines Würfelspiels in Unity

vorgelegt von: Tony Lenz
Geburtsort- und datum: Leipzig, den 07.01.1993
Abgabe: Leipzig, den 25. Februar 2024

Erstgutachter: Prof. Dr.-Ing. Müller ERSTGUTACHTER, HTWK Leipzig
Zweitgutachter: Prof. Dr.-Ing. Bleymehl ZWEITGUTACHTER, HTWK Leipzig

HTWK Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die an der Hochschule für Technik, Wirtschaft und Kultur Leipzig, konkret an der Fakultät Informatik und Medien (FIM), eingereichte Arbeit zum Thema Evaluierung von Reinforcement Learning-Algorhitmen anhand eines Würfelspiels in Unity selbstständig, ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe.

Alle den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen sind als solche einzeln kenntlich gemacht. Die Abbildungen in dieser Arbeit wurden von mir selbst erstellt oder mit einem entsprechenden Hinweis auf die Quelle versehen.

Diese Arbeit ist bislang keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht worden.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

. . . . . .

Unterschrift Tony Lenz
Leipzig, den 25. Februar 2024

Kurzfassung

Die Kurzfassung - präziser: Abstrakt, sollte auf eine Seite beschränkt sein. Der Abstrakt sollte das Ziel der Arbeit explizit beschreiben, die angewandten *Methoden* aufführen, die wichtigsten *Ergebnisse* aufzählen und die *Hauptschlussfolgerungen* aufzeigen.

Einen Weg zu finden, hunderte von Seiten an Informationen in wenigen Sätzen zusammenzufassen ist eine Herausforderung. Jedes Wort muss sorgfältig abgewogen werden (Gibt es eine bessere, prägnantere Möglichkeit, die Hauptaussage auszudrücken?). Die einzelnen Sätze sollten mit größter sorgfalt kombiniert werden.

Das Verfassen des Abstraktes sollte bis zum Schluss aufgeschoben werden, aus dem selben Grund, warum der Titel der Arbeit erst dann endgültige Form erhalten sollte, wenn das entsprechende Arbeit ansonsten vollständig ist.

Die nachfolgenden Ausführungen sollen nochmal ins Gedächtnis rufen was die Anforderungen an einzelne Abschnitte einer Abschlussarbeit an der FIM der Hochschule für Technik, Wirtschaft und Kultur Leipzig (HTWK) sind.

Die Kurzfassung schließt mit der Nennung von 5 bis 10 Schlagwörtern (Keywords) ab, welche der Verschlagwortung bspw. für die Recherche nutzbar gemacht werden. Genauer gesagt, sind das inhaltliche Schlüsselwörter Ihrer Arbeit.

Die Struktur dieser Arbeit gliedert sich wie folgt:

- **Theoretische Grundlagen:** In diesem Abschnitt werden die grundlegenden

Konzepte des Reinforcement Learning erläutert, darunter Neuronale Netze Markov-Entscheidungsprozesse und Probleme welche beim dem RL auftreten können. Weiterhin wird eine detaillierte Beschreibung des Spiels 'Noch mal' gegeben, einschließlich der Regeln, Spielziele und möglicher Strategien.

- **Konzeption:** Dieser Abschnitt umfasst eine Anforderungsanalyse und beschreibt den Ansatz zur Implementierung des Reinforcement Learning-Agenten für 'Noch mal'.
- **Experimente und Ergebnisse:** Es werden die Ergebnisse der Experimente präsentiert, einschließlich der Leistung des RL-Agenten beim Spielen von 'Noch mal' und einer Analyse seiner Fähigkeiten und Schwächen.
- **Diskussion:** Eine Diskussion über die Ergebnisse, die Einschränkungen der Studie und mögliche Verbesserungen wird vorgenommen.
- **Fazit und Ausblick:** Abschließend werden die wichtigsten Erkenntnisse zusammengefasst und ein Ausblick auf mögliche zukünftige Forschungsrichtungen gegeben.

Keywords: IoT, SD-WAN, Machine Learning (ML), Fiber to the Home (FTTH)

Inhaltsverzeichnis

1 Einleitung

In der heutigen Zeit stehen wir an der Schwelle einer digitalen Revolution, in der maschinelles Lernen und künstliche Intelligenz eine immer bedeutendere Rolle spielen. Insbesondere das Gebiet des Reinforcement Learning hat in den letzten Jahren enorme Fortschritte gemacht und findet Anwendung in einer Vielzahl von Bereichen, von der Robotik bis hin zu Finanzen. Diese Entwicklung bietet aufregende Möglichkeiten, komplexe Probleme zu lösen und intelligente Systeme zu entwickeln, die in der Lage sind, eigenständig zu lernen und Entscheidungen zu treffen.

Reinforcement Learning (RL) hat bereits beeindruckende Erfolge erzielt, indem es Algorithmen entwickelt hat, die komplexe Spiele wie Go auf einem kompetitiven Niveau spielen können und sogar über menschliche Fähigkeiten hinausgehen. Darüber hinaus wurde RL erfolgreich eingesetzt, um die Effizienz und Leistung von Serverfarmen bei Unternehmen wie Google zu optimieren. Diese Anwendungen verdeutlichen die Vielseitigkeit und Leistungsfähigkeit von Reinforcement Learning bei der Bewältigung verschiedenster Herausforderungen und unterstreichen seine Fähigkeit, komplexe Probleme zu lösen und neue Lösungswege zu finden.

In dieser Bachelorarbeit liegt der Fokus auf der Implementierung und dem Training eines Reinforcement Learning-Agenten für das Würfelspiel 'Noch mal'. 'Noch mal' ist ein Würfelspiel, das Strategie und Glück erfordert. Ziel dieser Arbeit ist es, einen Agenten zu entwickeln, der in der Lage ist, das Spiel zu erlernen und auf einem kompetitiven Niveau zu spielen.

Der Einsatz von Reinforcement Learning zur Bewältigung komplexer Spiele wie 'Noch mal' bietet eine interessante Herausforderung und die Möglichkeit, die Leistungsfähigkeit dieser Techniken zu demonstrieren. Durch die Implementierung eines RL-Agenten für dieses Spiel lässt sich untersuchen, wie gut maschinelle Lernmodelle in der Lage sind, komplexe Entscheidungsprobleme zu lösen und Strategien zu entwickeln, um ein definiertes Ziel zu erreichen.

Diese Bachelorarbeit zielt darauf ab, einen Beitrag zum Verständnis der Anwendung von Reinforcement Learning in der Spieleentwicklung zu leisten und Einblicke in die Leistungsfähigkeit dieser Techniken zu bieten. Durch die Implementierung eines RL-Agenten für 'Noch mal' sollen neue Erkenntnisse darüber gewonnen werden, wie maschinelles Lernen zur Entwicklung intelligenter Systeme in spielerischen Umgebungen eingesetzt werden kann.

2 Theoretische Grundlagen

2.1 Neuronale Netze

Neuronale Netze (=NN) sind ein Modell für künstliche Intelligenz nach dem Vorbild des Gehirns. Es besteht aus mehreren Schichten verknüpften Neuronen, welche numerische Informationen verarbeiten und in einen Output umwandeln. Die Ausgänge der vorderen Neuronen sind dabei immer mit den Eingängen der Neuronen der nächsten Schicht verknüpft. Jedes Neuron besitzt eine Aktivierungsfunktion, welche entscheidet ob es aktiv ist oder nicht. Neuronen geben Werte von 0 (passiv) bis 1 (aktiv) an dahinterliegende Neuronen. Richtig trainierte Neuronale Netze können gute Antworten für komplexe Problemstellungen geben, so liefern sie Beispielsweise in der Mustererkennung gute Ergebnisse.
Wird ein NN initialisiert werden Kantengewichte zwischen den Neuronen zufällig verteilt. Diese Kantengewichte sorgen dafür wie stark einzelne Neuronen in die nachfolgende Rechnung eingehen. Deshalb liefern untrainierte NN schlechte Ergebnisse und müssen trainiert werden, bevor sie Problemstellungen richtig lösen können.
Während des Trainings eines NN werden Kantengewichte angepasst um die Heuristik an ein optimales Ergebnis anzupassen.
Für das Training von NN wird viel Rechenleistung gebraucht, da der Prozess mit sehr vielen Rechenoperationen verbunden ist. [Ertel S. 290]

Bild aufbau Neuronale Netze

2.2 Reinforcement Learning

Reinforcement Learning ist ein Bereich des maschinellen Lernens, bei welchem ein Agent durch Interaktion mit seiner Umgebung lernt, welche Aktionen in welchen Situationen am besten geeignet sind um ein bestimmtes Ziel zu erreichen. Da ein Agent keine konkrete Vorgehensweise besitzt, versucht er über Trial-and-Error herrauszufinden welche Aktionen zu Belohnungen führen.
Bestandteile des RL sind der Agent und die Umwelt. Der Agent bekommt die Informationen der Umwelt übergeben und entscheidet welche Handlungen daraus folgen sollen. Das Environment setzt die Aktionen des Agenten in Hanldungen um und bewertet diese mit numerischen Rewards, welche als Belohnung fungieren. Anhand der erhaltenen Rewards, versucht der Agent seine Aktionen anzupassen um die zukünftigen Belohnungen zu maximieren.
Ein RL-Agent nimmt seine Umgebung als eine Menge an bestimmten Zuständen war. Jeder Zustand enthält eine Vielzahl von Merkmalen und Eigenschaften welche für die Entscheidungsfindung relevant sein können. Als Zustandsraum wird die Menge aller möglichen Zustände bezeichnet.
Auf jedem dieser Zustände ist es möglich eine bestimmte Menge an Aktionen auszuführen, welche das Umfeld in einen anderen Zustand überführen. Die Menge aller möglichen Aktionen wird als Aktionsraum bezeichnet.
Wird eine Aktion auf einem bestimmten Zustand ausgeführt, so bezeichnet man die Zustandsübergänge bei Ausführung der Aktion als Zustandsübergangsfunktion.
Die Verhaltensstrategie πeines Agenten wird auch als Policy bezeichnet und liefert zu jedem Zustand eine Aktion. Die Policy wird im Verlauf des Trainings erlernt.
Das Erlernen einer Policy erfolgt durch Training des Neuronalen Netzes. Zu Beginn des Trainings werden Kantengewichte des zur trainierenden Neuronalen Netzes zufällig verteilt. Um zu überprüfen ob seine Aktionen gut oder schlecht sind, muss der Agent durch einen numerischen Rückgabewert die Information erhalten. Diese numerischen Rückgabewerte werden als Belohnungen oder auch Rewards bezeichnet. Belohnungen werden im Zustandsraum verteilt und können gutes Verhalten des Agenten durch positive Rewards und falsches Verhalten durch negative Rewards bestärken. Der Agent versucht den Wert der erhaltenen Belohnungen zu maximieren und erlernt auf diese Weise eine optimale Policy.
Probleme welche beim RL auftreten können sind unteranderem Überanpassung, Beloh-

nungsumgehung, Sparse Rewards. Wenn der Agent ein Problem gut in der Lernumgebung in welcher er trainiert wurde lösen kann allerdings nicht auf abweichenden Umgebungen spricht man von Überanpassung. Es tritt auf, wenn der Agent zu spezialisiert auf seine Trainignsumgebung ist und sich nicht an andere Situationen anpassen kann. Diese Problem lässt sich durch Generalisierung des Trainingsprozesses beheben. Ein Agent welcher auf vielen (zufällig) generierten Umgebungen trainiert, kann deutlich besser mit neuen Situationen umgehen, benötigt allerdigns auch mehr Zeit zum trainieren.
Belohnungsumgehung tritt auf, wenn der Agent lernt die Belohnungsfunktion zu manipulieren, um erhaltene Belohnungen zu maximieren, ohne das eigentliche Ziel der Aufgabe zu erreichen. Dies führt zu unerwünschtem Verhalten des Agenten. Verhindert werden kann dieses Problem, indem Belohnungen direkt mit dem Ziel verknüpft werden, also nur Belohnugnen ausgelöst werden, wenn tatsächlich positive Aktionen ausgeführt werden.
Von Sparse Rewards spricht man, wenn die Lernumgebung dem Agenten nur selten oder unregelmäßig Belohnungen vergibt. Dies führt dazu, dass der Agent Schwierigkeiten hat, die ihm gegebene Aufgabe richtig zu erlernen. Um dieses Problem zu umgehen, kann man zusätzliche künstliche Belohnungen einführen, welche dem Agenten auf den Weg zum richtigen Verhalten führen. Dies kann im Rückschluss jedoch wieder zur Belohnungsumgehung führen, wenn der Agent die Zwischenbelohnungen nutzt um maximale Belohnungen zu erhalten.
Auch ein zu großer Beobachtungsvektor kann zu Komplikationen führen. Wird dem Agenten eine Vielzahl an Observationen zugeführt, müssen erheblich mehr Parameter im Modell verarbeitet werden, was zu größerem Berechnungsaufwand führt. Dadurch wird auch die Lerngeschwindigkeit verlangsamt und der Bedarf an Speicherplatz für die Modelle steigt. So ist es möglich, dass einfach erscheinende Aufgaben mehrere Tage an Rechenzeit benötigen um ein gutes Modell zu generieren.

[Quelle Zai, Kramer]

2.2.1 Markov- Entscheidungsprozess

Jedes RL Problem lässt sich durch den Markov Entscheidungsprozess (= MDP) beschreiben. Der MDP ist ein mathematisches Modell für Entscheidungsprobleme, bei welchem der Agent abhängig von der Umwelt Entscheidungen trifft um ein bestimmtes Ziel zu erreichen. MDP setzen die Einhaltung der Markov-Eigenschaft vorraus. Diese ist erfüllt, wenn ein Zustandsübergang nur vom letzten Zustand und der letzten Aktion abhängig ist. Hauptkomponenten des MDP sind eine endliche Menge valider Zustände, eine endliche Menge valider Aktionen, Belohnungsfunktion und Verhaltensstrategie. Das Ziel eines MDP ist es eine optimale Policy, durch Maximierung der erhaltenen Belohnungen zu ermitteln.

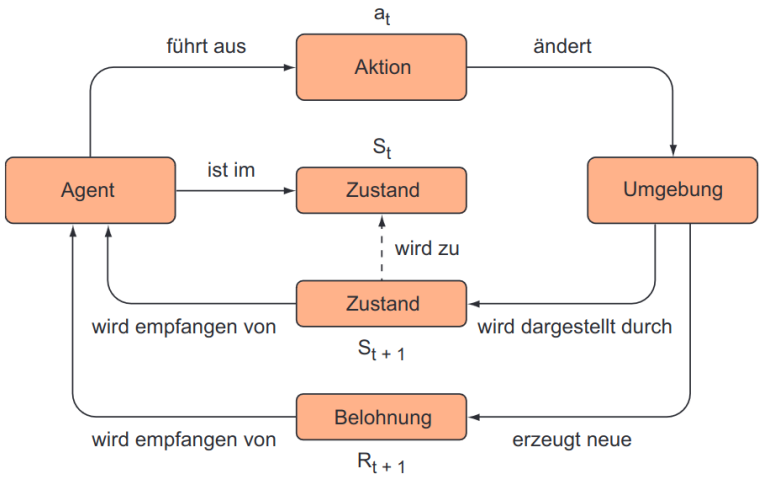

Abbildung 2.1: Ablauf eines MDP, Quelle:Einstieg DeepLearning Zai/Brown S27

Abbildung2.1 stellt den Ablauf eines MDP dar. Zu Beginn einer Episode, befindet sich die Lernumgebung in einem gewissen Zustand. Dieser Zustand wird dem Agenten übergeben. Anhand der Beobachtungen führt der Agent eine Aktion aus welche die Umgebung verändert. Die Veränderung der Umgebung erzeugt einen neuen Zustand, welcher im nächsten Schritt wiederum dem Agenten übergeben wird. Durch auslösen von Aktionen in der Lernumgebung werden Belohnungen erzeugt welche dem Agenten einen numerischen Wert liefern, ob die Aktion gut oder schlecht war. Der Agent versucht die erhaltenen Belohnungen zu maximieren. Eine hohe Belohnung impliziert das richtige Verhalten zum Lösen des Problems welches der Agent bewähltigen muss. Dieser Prozess wiederholt sich bis das Problem gelöst wurde.

2.3 Das Würfelspiel: Noch Mal

Im modellierten Spiel 'Noch Mal!' geht es darum so viele Kästchen anzukreuzen und möglichst viele Spalten und gleichfarbige Kästchen auszufüllen. Farb- und Zahlenwürfel müssen kombiniert werden um entsprechend zusammenhängende Felder der gewählten Farbe abzukreuzen. Im Spiel gibt es folgende Regeln:

1. Felder in der Spalte H sind von Beginn an verfügbar
2. Alle Kreuze müssen immer zusammenhängend in genau einem Farbblock der gewählten Farbe platziert werden
3. Kreuze müssen waagerecht oder senkrecht benachbart zu einem bereits abgekreuzten Feld oder Teil der Spalte H sein um verfügbar zu werden
4. Es müssen genau so viele Felder angekreuzt werden wie das Ergebnis des gewählten Zahlenwürfels
5. Es könnte nicht mehr als 5 Kästchen in einem Zug abgekreuzt werden
6. Wird ein Zahlenjoker gewählt, darf der Spieler eine Zahl von 1-5 bestimmen
7. Wird ein Farbjoker gewählt, darf der Spieler eine Farbe bestimmen

Um im Spiel 'Noch mal' eine möglichst hohe Anzahl an Punkten zu erhalten, ist es wichtig nach folgenden Strategien zu spielen:
Priorisierung äußerer Spalten: Äußere Spalten geben mehr Punkte, weshalb es wichtig ist diese komplett auszufüllen.
Beenden von Farben: Vollständig ausgefüllte Farben geben viele Extrapunkte. Im späten Spielverlauf kann es besser sein Farben komplett zu füllen anstatt Spalten zu werten.
Priorisieren von Sternfeldern: Jedes ausgefüllte Sternfeld gibt 2 Punkte, eine gute Spielweise ist es so viele Sternfelder wie möglich auszufüllen.
Strategische Nutzung von Jokern: Ungenutzte Joker geben zum Ende des Spiels Punkte. Es ist gut diese so wenig wie möglich zu nutzen um Extrapunkte zu bekommen. Jedoch können mit Hilfe von Jokern einfach bestimmte Felder gewählt werden, welche benötigt werden um eine Wertung zu erzielen.

Abbildung2.2 ist ein Spielfeld aus 'Noch mal!'. Die blau markierten Felder geben den Spaltennamen der Spalten an. Die rot markierten Felder zeigen an, wie viele Punkte beim

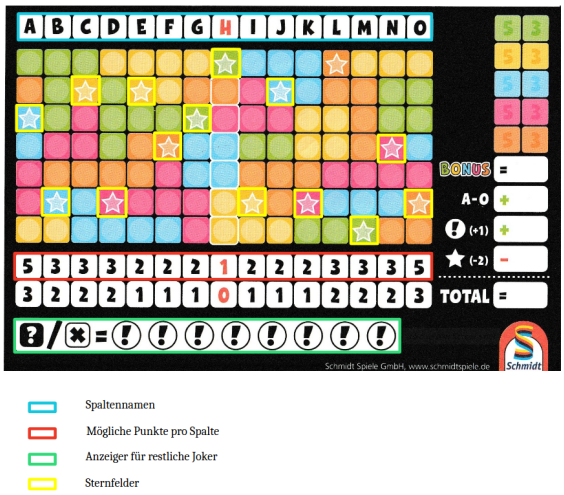

Abbildung 2.2: Vorlage des Spielfedes mit Indikation der Punktewertung

ausfüllen der jeweiligen Spalte erzielt werden. Das grün markierte Feld zeigt die Anzahl der verbleibenden Joker an, wird ein Joker benutzt, muss eines der Felder abgestrichen werden. Zum Ende des Spiels erhält der Spieler Extrapunkte für die verbleibenden Joker. In jeder Spalte befindet sich ein geld markiertes Sternfeld. Diese geben zum Ende des Spiels Minuspunkte, weshalb es wichtig ist alle Sternfelder auszufüllen.

2.3.1 Spielablauf Einspieler Variante

Der Spieler Würfelt alle 4 Würfel bestehend aus 2 Farb- und 2 Zahlenwürfeln. Der Spieler hat 30 Züge Zeit maximale Punkte zu erreichen. Anschließend wählt er ein paar aus Farb- und Zahlenwürfel aus und kreuzt entsprechend des gewürfelten Paares verfügbare Felder auf dem Spielfeld ab. Ein Spieler darf immer entscheiden ob er Würfelwürfe zum ankreuzen verwenden möchte oder nicht. Um Kästchen anzukreuzen, wählt der Spieler eine Kombination aus Zahlen bzw Farbwürfel aus. Wählt er Beispielsweise 'Grün' und '2' so müssen 2 zusammenhängende Grüne Felder angekreuzt werden.
Gelingt es dem Spieler eine Spalte komplett auszufüllen, erhält er je nach Spalte Punkte dafür. Für äußere Spalten mehr werden mehr Punkte vergeben als für die inneren Spalten. Für das komplette Ausfüllen einer Farbe erhält der Spieler fünf Punkte pro ausgefüllter Farbe. Jedes nicht angekreuzte Sternfeld gibt zum Spielende zwei Minuspunkte. Für jeden übrig gebliebenen Joker erhält der Spieler zum Ende des Spiels einen Punkt.

Abbildung 2.3: Grafik zur Bewertung der gesammelten Punkte der Einspieler Variante

Die Abbildung2.3 ist aus der Spielanleitung des Spiels. Sie zeigt an, wie viele Punkte erreichbar sind und bewertet das Ergebnis.

3 Konzeption

3.1 Anforderungsanalyse

Einfache Implementierung Visualisierung um Agent besser überwachen zu können Frameworks für RL Frameworks für Überwachung

3.2 Auswahl Entwicklungsumgebung

3.3 Methodik

In diesem Abschnitt wird der methodische Ansatz zur Implementierung des Spiels und des Trainings des Reinforcement Learning-Agenten beschrieben.

3.3.1 Implementierung des Spiels

Das Spiel 'Noch mal' wurde mithilfe der Unity Engine implementiert, um eine interaktive und visuell ansprechende Umgebung zu schaffen. Die Programmierung der Spielregeln, die Benutzerinteraktion sowie die visuelle Darstellung des Spielzustands wurden dabei in Unity realisiert.

3.3.2 Entwicklung des Reinforcement Learning-Agenten

Der Reinforcement Learning-Agent wurde mithilfe der Unity ML-Agents-Bibliothek entwickelt, um das Spiel 'Noch mal' zu erlernen und zu spielen. ML-Agents bietet eine umfassende Sammlung von Werkzeugen und Algorithmen für die Implementierung von RL-Agenten in Unity.

3.3.3 Trainingsprozess und Anpassungen

Der Agent wurde in der implementierten Unity-Umgebung trainiert, wobei mehrere Trainingsdurchläufe durchgeführt wurden. Während des Trainingsprozesses wurden kontinuierliche Anpassungen vorgenommen, einschließlich der Feinabstimmung der Hyperparameter, der Modifikation der Netzwerkarchitektur und der Einführung von Maßnahmen zur Förderung der Exploration. Diese Anpassungen wurden iterativ durchgeführt, um die Leistung und die Lernfähigkeit des Agenten zu verbessern und sicherzustellen, dass er die Spielumgebung optimal erfasst und Strategien entwickelt, um die gestellten Ziele zu erreichen.

3.3.4 Überwachung und Analyse

Der Fortschritt des Trainings wurde kontinuierlich überwacht und analysiert. Dies umfasste die Überprüfung von Trainingsmetriken wie der durchschnittlichen Belohnung, der Verfolgung der Lernkurven und der Analyse von Trainingsfehlern.

3.3.5 Iteration und Abschluss

Der Trainingsprozess wurde iterativ durchgeführt, wobei der Agent kontinuierlich verbessert wurde, bis eine zufriedenstellende Leistung erreicht wurde. Nach Abschluss des Trainings wurde der finale Agent auf seine Fähigkeit getestet, das Spiel 'Noch mal' zu spielen, und seine Leistung wurde bewertet. Dieser methodische Ansatz ermöglichte es,

einen effektiven Reinforcement Learning-Agenten für das Spiel 'Noch mal' zu entwickeln und zu trainieren, der in der Lage ist, das Spiel auf einem angemessenen Niveau zu spielen.

4 Implementation

4.1 Umsetzung in Unity

Der **Controller** stellt alle Funktionalitäten bereit, welche gebraucht werden um die Lernumgebung zu initialisieren. Er besitzt Prefabs des Agents, des GameFields und der Würfel und initialisiert diese zum Start. Weiterhin implementiert der Controller die Funktionalität der Punktevergabe, welche für eine Mehrspielervariante genutzt werden kann. Der Controller ist das Parent aller anderen Elemente und so ist er das Zentrale Element der Steuerung. Auch das wiederholte rollen der Würfel wird im Controller angestoßen. Der Controller war sehr hilfreich beim erstellen paralleler Trainings, da dieser einfach mehrfach in die Szene aufgenommen werden musste um mehrere Spielfelder, welche gleichzeitig bespielt werden zu initialisieren.

<Code ausschnitt?>

Der **NumberDice** implementiert die Logic, welche für das Würfeln und Visualisieren der Zahlenwürfel benötigt wird. Die Visualisierung funktioniert mit selbst angefertigten Sprites welche in einem Sortierten Array liegen und je nach gewürfelter Zahl initialisiert und gerendert werden. Beim wiederholten Würfeln, wird das initialisierte Sprite destroyed und ein neues erzeugt. Damit ist gewährleistet, dass immer das aktuelle Würfelergebnis angezeigt wird. Die Zahl des Würfels wird als Integer wert gespeichert, wobei er die Zahlen 1-6 annehmen kann. Die Zahl sechs entspricht dem Zahlenjoker.

<Code>

Wie der Zahlenwürfel implementiert der **ColorDice** die Funktionalität des Würfelns der

Farben. Diese werden als String dargestellt und kann folgende Werte annehmen: {'blue', 'green','red', 'yellow', 'orange', 'joker'} Zur Visualisierung wird ein Sprite erstellt, was in der gewürfelten Farbe eingefärbt wird. Ein schwarzes Feld entspricht dem gewürfelten Farbjoker.

<Code>

Das **GameField** stellt das tatsächliche Spielfeld dar. Es implementiert die benötigten Methoden um die SquareFields zu verwalten und rückzusetzen. Außerdem wird die Anzahl der Joker in ihm gehalten.

Funktionalitäten:

- Visualisierung des Spielfeldes
- Aktualisieren der Gruppen aller Felder
- Abkreuzen der Felder
- Berechnen der validen Nachbarn der Felder
- Berechnen der verbleibenden Felder einer bestimmten Farbe
- Rückgabe der validen Felder für die aktuell gewählten Würfel.
- Reduzieren der verbleibenden Joker
- Rücksetzen der Felder um ein neuest Spiel zu Starten

Die **FieldSquares** stellen die einzelnen Teilfelder des Spielfeldes dar. Gehaltene Informationen:

Beschreibung	**Typ**	**Wertebereich**
Feld ist ein Sternfeld	Boolean	True / False
Farbe des Feldes	String	-
Feld ist ausgefüllt	Boolean	True / False
Feld ist verfügbar	Boolean	True / False
Clustergröße	Integer	1-6
X-Koordinate des Feldes	Integer	0-14
Y-Koordinate des Feldes	Integer	0-6

Tabelle 4.1: Beschreibung, Typ und Wertebereich der Feldinformationen

4.1.1 Visualisierung

Die Visualisierung des Spielfeldes erfolgt über ein angefertigtes Prefab. In diesem wurden die 105 Kästchen in einem Raster von 15x7 instanziiert und manuell mit den Informationen versehen. Dieses manuell angefertigte Spielfeld wurde als Prefab gespeichert und dient als Umgebung für den Agenten. Zu Beginn des Spiels, werden die Felder in die Farben der hinterlegten Information in den richtigen Farben eingefärbt. Ausgefüllte Kästchen werden grau eingefärbt, diese Funktionalität wird im Fieldsquare Prefab ausgeführt.

<Code instanziierung der Felder?>

<Code einfärben der Felder>

4.2 Implementierung des Agenten

Der Agent ist die Schnittstelle zwischen dem Environment und dem RL. Dem Agent werden alle nötigen Informationen des Spielfeldes übergeben. Diese werden in ein Neuronales Netz übertragen, welches wiederum die Ausgabewerte in einem Vektor zurück an den Agent leitet. Anschließend wird der Vektor verarbeitet und die gewählten Aktionen werden ausgeführt. Für gute Aktionen erhält der Agent positive Rewards, bei schlechten Aktionen wird der Zug übersprungen.
Zu Beginn jeder Episode, welche einem Spielzug entspricht, muss dem Agenten der aktuelle Zustand des Feldes übermittelt werden, aus welchem er die bestmögliche Option für einen Zug berechnet. In der ML Agents Bibliothek gibt es hierfür eine vorgefertigte Methode mit dem Namen CollectObservations. Die bearbeitet einen Observationsvektor zu welchem die Informationen hinzugefügt werden. Wärend des Trainings eines Neuronalen Netzes, muss die Anzahl der Observations gleich bleiben. Das bedeutet es ist nicht ohne weiteres möglich ein Model auf unterschiedlichen Spielfeldern zu trainieren, da sich so die Anzahl der Observations unterscheiden würden.

Aufbau der Observations:

<skript Collect Observations>

Index	Beschreibung	Type	Wertebereich
0	Anzahl der verbleibenden Joker	Float	$[0-1]$
1	Anzahl der gespielten Runden	Float	$[0-1]$
2	Ergebnis des ersten Zahlenwürfels	Float	$[0-1]$
3	Ergebnis des zweiten Zahlenwürfels	Float	$[0-1]$
4-9	Ergebnis des ersten Farbwürfels	Vector6 (Binary)	$(0,1)^6$
10-15	Ergebnis des zweiten Farbwürfels	Vector6 (Binary)	$(0,1)^6$
16-24	Informationen für Feld 1	-	-
25-33	Informationen für Feld 2	-	-
...	...	...	...
953-961	Informationen für Feld 105	-	-

Tabelle 4.2: Zusammenfassung der Observations und Feldinformationen

Stelle im Vektor	Beschreibung	Type	Wertebereich
$k+16$ - $k+21$	Farbe des Feldes k	Vector6 (Binary)	$(0,1)^6$
$k+22$	Ist Feld k verfügbar	Boolean	True / False
$k+23$	Ist Feld k abgestrichen	Boolean	True / False
$k+24$	Ist Feld k ein Sternfeld	Boolean	True / False

Tabelle 4.3: Observation jedes einzelnen Feldes

Anhand der Observations berechnet das Neuronale Netz einen Ausgabevektor. Mit diesem führt der Agent nun bestimmte Aktionen aus und versucht sein Ergebnis (Rewards) zu maximieren. Anhand der gesammelten Rewards wird das Neuronale Netz nun angepasst um das bestmögliche Ergebnis zu erreichen.

Aufbau Actionbuffer:

Tabelle 4.4: Index und Beschreibung der Variablen

Index	Beschreibung	Typ	Wertebereich
1	Index des gewählten ZahlenWürfels	Integer	0-1
2	Index des gewählten Farbwürfels	Integer	0-1
3	Jokerzahl	Integer	0-4
4	X-Koordinate des gewählten Feldes	Integer	0-14
5	Y-Koordinate des gewählten Feldes	Integer	0-7
6	Action 1 für die Auswahl der Nachbarn	Continuous	-
7	Action 2 für die Auswahl der Nachbarn	Continuous	-
8	Action 3 für die Auswahl der Nachbarn	Continuous	-
9	Action 4 für die Auswahl der Nachbarn	Continuous	-

4.2.1 Erklärung des Alghorithmus

Im folgenden wird der Ablauf zum Wählen der Felder erläutert. Im Beispiel wird der Ausgabevektor **(1 , 1 , 0 , 3 , 4 , 0.6 , 0.5 , 0.4 , 0.8)** verwendet.
Die ersten beiden Stellen des Ausgabevektors entsprechen den gewählten Würfeln. Im ersten Schritt werden alle Feldes des Spielfedes untersucht, ob sie ein valides Ziel für das gewürfelte Ergebnis bilden. Die ergibt sich aus der Gruppe der Spielfelder, der Farbe und ob das Kästchen verfügbar ist. Valide Felder werden in eine Liste (availableFields) aus Verfügbaren Feldern geschrieben.

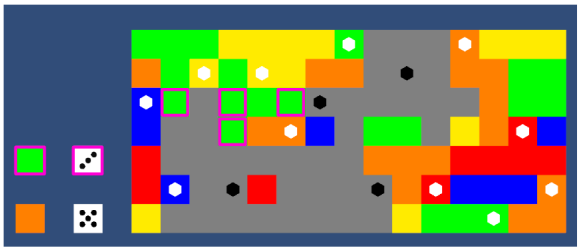

Abbildung 4.1: Valide Felder für das gewählte Würfelergebnis wurden markiert

Anschließend wird geprüft, ob die gewählten Koordinaten in availableFields vorhanden sind. Wenn nein wird die Episode abgebrochen und der Agent überspringt seinen Zug. Sofern der Agent ein valides Feld gewählt hat, wird dieses in eine weitere Liste (pickedFields) geschrieben und benachbarte Felder der selben Gruppe werden zurückgegeben.

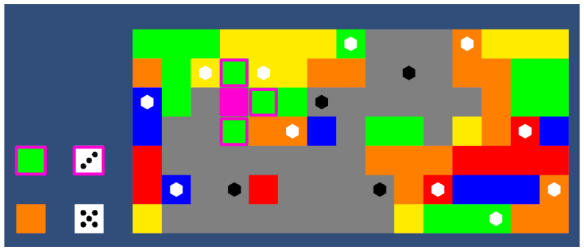

Abbildung 4.2: Feld(3,4) wird in die pickedField Liste aufgenommen und benachbarte Felder werden zurückgegeben.

Im nächsten Schritt wird jedem der verfügbaren Nachbarn abhängig der Gesamtanzahl ein Wertebereich zwischen 0 und 1 zugewiesen. Anhand des discreten Wertes des Ausgabevektors wird das Zugehörige Feld in pickedFields geschrieben.

FeldKoordinaten	**von**	**bis**
(3,5)	0	0.33
(4,4)	0.33	0.66
(3,3)	0.66	0.99

Tabelle 4.5: Bereiche für bestimmte Felder

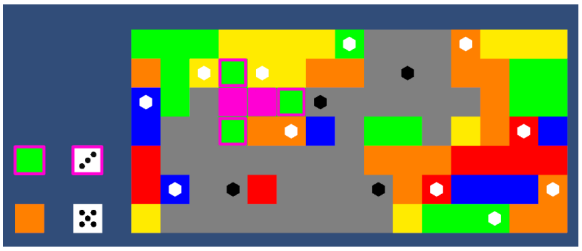

Abbildung 4.3: Feld(4,4) ist das nächste gewählte Feld und wird in pickedFields aufgenommen

Für alle Feldes in Picked Field werden die benachbarten Felder zurückgegeben und der vorherige Schritt wiederholt. Wenn so viele Felder gewählt wurden, wie erwürfelt wurden, werden die Felder anschließend ausgefüllt und auf Rewards überprüft.

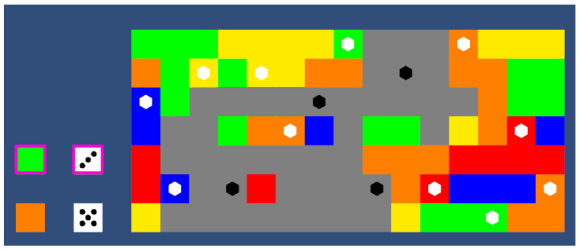

Abbildung 4.4: Felder wurden gewählt und ausgefüllt

4.3 Trainingsversuche

4.3.1 Trainingsversuche

Aufbau Actionbuffer bis dato -> würfel, würfel, feldIndex, feldIndex , feldIndex ,feldIndex, feldIndex (feldIndex int) In den ersten Trainings soltle der Agent Würfel und anschließend zufällige Felder wählen. Danach wurde überprüft ob alle ausgewählten Felder folgende Kriterien erfüllten:

- alle Felder verfügbar
- alle Felder benachbart
- alle Felder der selben farbe

Illegale Züge wurden bestraft (negative Rewards) und wurden nicht ausgeführt. Da der Agent Anfangs zufällige Felder auswählt, hatte es zur Folge, dass nahezu keine Spielzüge getätigt wurden. Dies führte dazu, dass das Spiel nicht gespielt wurde. Dementsprechend war das Training auf diese Weise nicht erfolgreich und musste überarbeitet werden.

4.3.2 Training des Agenten mit Zusatz Belohnungen

In den Bereits erläuterten Trainingsversuchen, bekam der Agent für vermeintlich gute Züge eine Belohnung und für schlechte Züge eine Bestrafung.

- Für das Abkreuzen von Feldern 0.02f pro feld - falls ein gesammmtes Cluster abgekreuzt

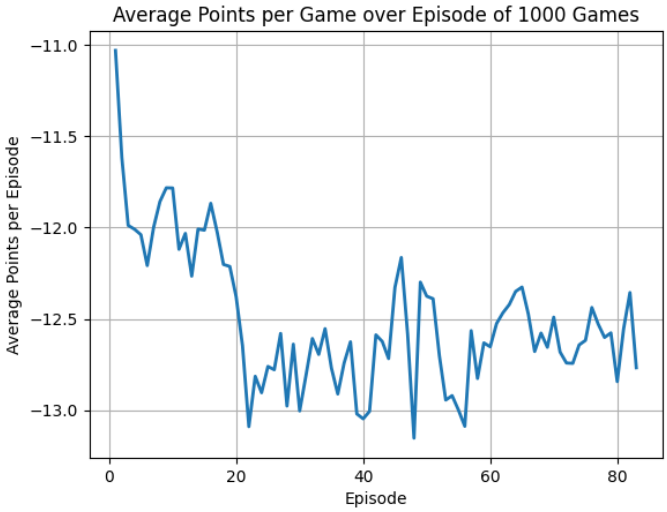

Abbildung 4.5: Das Logo von Informatik und Medien

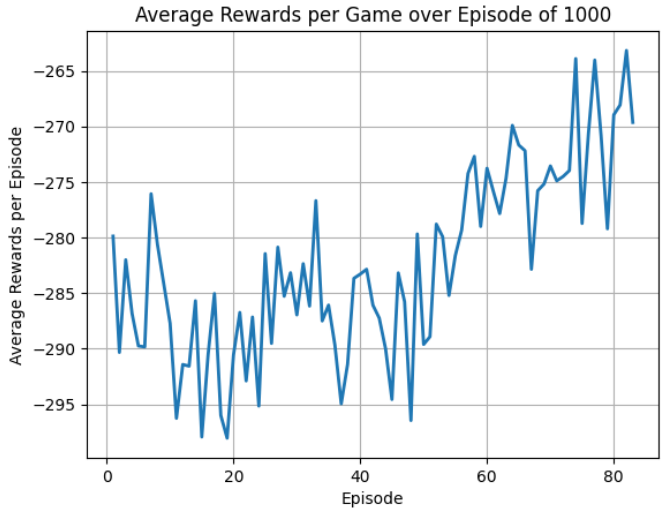

Abbildung 4.6: Das Logo von Informatik und Medien

wird -> 0.04 pro feld - wahl eines Würfelpaars für das es keine legalen Züge gibt -> -50.0f - Wahl eines Jokers obwohl keine Joker verfügbar sind -> -50.0f

Die Belohnungen führten dazu, dass der Agent nicht Versuchte Spalten abzukreuzen, da er das abkreuzen von Clustern für deutlich effizienter hielt um siene Belohnungen zu maximieren. Obwohl die Belohnungen für Erreichte Spalten oder komplett ausgefüllte Farben mehr Punkte ergaben.

Die Grafiken zeigen deutlich, dass je länger der Agent tranierte desto kleiner die Durchschnittlichen Punkte pro Spiel wurden. Dies spricht dafür, dass es zum sogenannten 'reward hacking' gekommen ist. Dieses Phänomen tritt auf, wenn der Agent sein eigentliches Ziel nicht erreichen kann, da er eine andere (falsche) Strategie erlernt, welche nicht zum eigentlichen Ziel führt. Ursache hierfür ist, dass der Agent sein eigentliches Ziel (das ausfüllen von Spalten) nie erreicht und somit nicht erlernt.

4.4 Untersuchung des Trainingsfortschritts

In diesem Versuch wird Untersucht ob das Training des Agenten erfolgreich war. In jeder Messreihe wurden 1Mio. Lernschritte druchgeführt was ca 33k gespielten Spielen entspricht. Im Anschluss wurde aus den gespielten Spielen die durchschnittliche erreichte Punktzahl kalkuliert und in einem Graphen dargestellt.

4.4.1 Trainiert vs Untrainiert

In diesem Experiment, werden die erreichten Punkte und Rewards eines untrainierten Agenten gegenüber den erziehlten Ergebnissen eines trainierten Agenten gegenübergestellt. Der trainierte Agent hat bereichts 25 Mio. Spielzüge absolviert, was ungefähr 830k gespielten Spielen entspricht. Der untrainierte Agent bekommt ein neu initialisiertes NN, welches zufällig gewählte Kantengewichte zwischen den Neuronen erhält. Wie an den Graphen zu erkennen ist, hat der trainierte Agent tatsächlich einen höheren Durchschnitt an erzielten Punkten pro Spiel. Auch die gesammlten Rewards sind bei dem trainierten Agenten höher. Dies liegt daran, dass die Rewards so festgelegt sind, dass der Agent sie nur erhält, wenn er auch im Spiel punktet. Schon während des Trainings war ein merklicher Unterschied festzustellen, deshalb war das Ergebnis dieses Experiments zu erwarten.

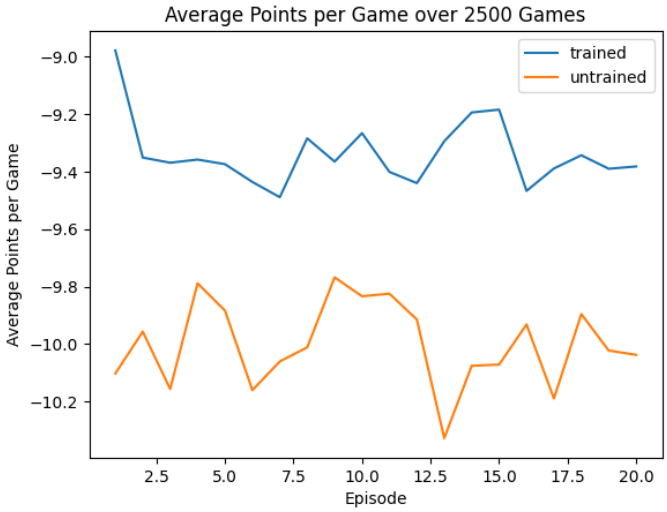

Abbildung 4.7: Durchschnitt der erreichten Punkte pro Spiel

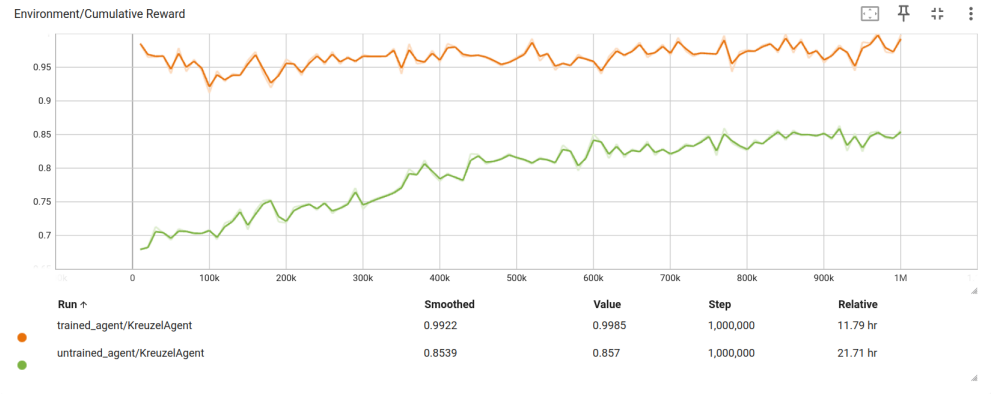

Abbildung 4.8: Übersicht der gesammelten Belohnungen

4.4.2 Trainiert vs Training mit Sonderfeldern

In diesem Experiment wurden die erreichten Puntke und Rewards des normal trainierten Agenten gegenüber einem Agenten, welcher mit Sonderfeldern trainiert wurde gegenüber gestellt. Diese speziellen Felder waren einheitlich in die verschiedenen Farben eingefärbt bzw jedes Feld wurde mit Sternfeldern versehen. Dies sollte dazu führen, dass der Agent besser zuweisen kann welche Stellen im Observationsvektor für welche Information zuständig sind.

Das Training mit speziellen Feldern führte zu einer Verschlechterung des Ergebnisses wie

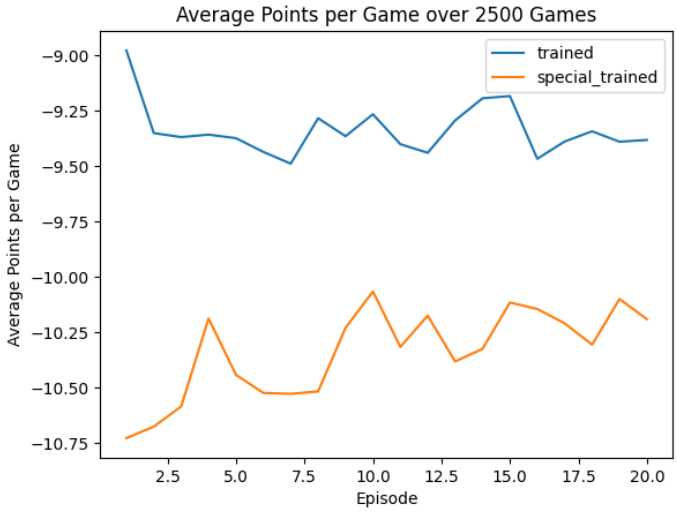

Abbildung 4.9: Durchscnbitt der erreichten Punkte beides Agenten

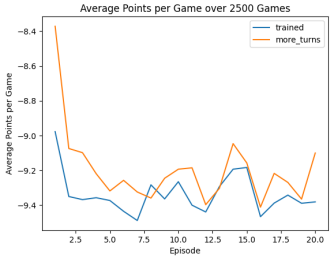

Abbildung 4.10: Durchschnitt der gesammelten Punkte

die zwei nachfolgenden Grafiken zeigen.

4.4.3 Trainiert vs Training mit mehr Spielzügen

In diesem Exderiment sollte der trainierte Agent das mit mehr zur Verfügung stehenden Spielzügen absolvieren. Dies hat zur Folge, dass während des Trainings häufiger zur Puntkewertung kommt und der Agent auf diese Ziele hin arbeiten kann. Wie die Grafiken zeigen, hatte das Experiment eine Verbesserung der Ergebnisse zur Folge.

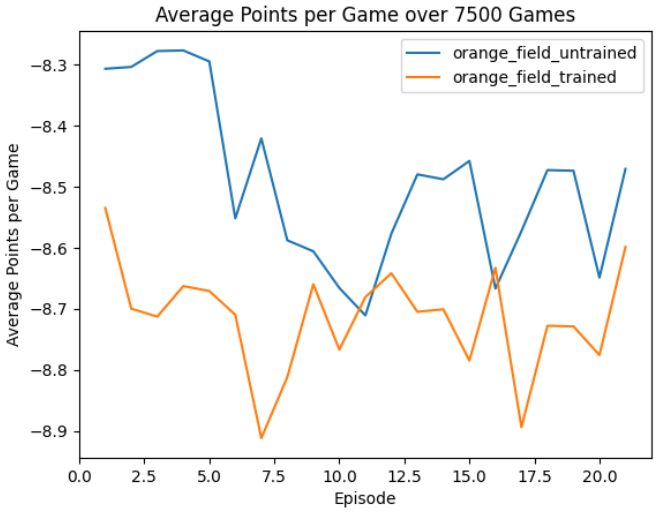

Abbildung 4.11: durchschnitt an gesammelten Punkten

Abbildung 4.12: gesammelte Rewards der beiden Agenten

4.4.4 Überprüfung auf Overfitting

In diesem Experiment, sollte der Agent auf Overfitting überprüft werden. trainierte und untrainierter Agent spielten das Spiel nach normalen Spielregeln auf einem anderen Spielfeld. In den Grafiken ist erkennbar, dass beide Agenten ungefähr die selben Rewards gesammelten haben. Der untraineirte Agent konnte im Durchschnitt jedoch etwas mehr Punkte sammeln. Dies schließt darauf, dass der Agent tatsächlich nur auf dem im Training verwendeten Spielfeld gut performen kann und neue SPielfelder erst erlernen muss.

Interessant ist weiterhin, dass der Durchschnitt aller Punkte etwa 2 Punkte über dem des anderen SPielfeldes liegt, was auf eine höhere Schwierigkeit des anderen SPielfeldes hinweist.

4.4.5 Training Auswahl KoordinatenPicker

Da der Agent keine großen Fortschritte erzielen konnte, entschied ich den Agenten das erste Feld durch Koordinaten zu wählen. Dies setzte Vorraus, dass die Koordinaten der einzelnen Felder in die Observations mit aufgenommen werdne musste und die Observations noch größer wurden. Damit der Agent lernen kann, welche Kooordinaten zu welchen Feldern gehören, entschied ic hmic hdazu ihn auf einem Spielfeld trainieren zu lassen, wo alle Teilfelder verfügbar sind. Rewards wurden vergeben für Valide ausgewählte Felder, in Abhängigkeit der gewürfelten Zahlen.

Im nächsten Schritt wird dieses vortrainierte NN genutzt um das Spiel mit richtigen Regeln zu spielen.

5 Auswertung und Ausblick

5.1 Bewertung der Ergebnisse

5.2 Schritte zur Verbesserung des Agenten

5.3 Diskussion

Dieser Abschnitt stellt den Schlusspunkt der Arbeit dar. In diesem Abschnitt (und im Diskussionteil) erwartet der Leser, dass er Antworten auf die in der Einleitung formulierten Fragestellungen findet und sich vergewissert, dass diese wirksam verteidigt wurden und mit der von Ihnen formulierten „These“ übereinstimmen.

Die Hauptziele der Diskussion bestehen darin, eine Analyse Ihrer gesammelten Ergebnisse zu präsentieren, Ihre Ergebnisse angemessen darzustellen und eine Einschätzung der Bedeutsamkeit Ihrer Arbeit zu geben. Beachten Sie hier, den Unterschied zur Diskussion im vorherigen Kapitel. Diskutieren Sie hier vor allem den Wert und die Bedeutung Ihrer Ergebnisse, auf Basis der Interpretationen aus dem vorherigen Kapitel. Beziehen Sie sich gern auch auf Ihr beschriebenes Problem und Ihr Ziel.

Jede wichtige Schlussfolgerung, die Sie im „Ergebnisteil“ gezogen haben, muss hier erneut behandelt werden. Eine gewisse Anzahl von Wiederholungen ist unvermeidlich. Darüberhinaus, die Ergebnisse anderer Forschungsarbeiten, müssen mit eindeutigen Verweisen auf auffindbare Literaturquellen versehen sein.

Der Fazit-Teil kann als eine kurze Zusammenfassung Ihrer Diskussion betrachtet werden. Der Leser muss sich hier schnell einen Überblick über den Inhalt und die Bedeutung der Arbeit als Ganzes verschaffen.

Dieser Abschnitt soll einen Überblick präsentieren und dient dazu, dem Hauptteil Ihrer Arbeit den letzten Schliff zu geben. Die Schlussfolgerung kann auch Hinweise auf ein mögliches zukünftiges Werk enthalten.

Aber **mindestens** folgende Punkte
Inhalt des fünften Kapitels (im Allgemeinen):

- Reflektieren Sie hier nun die Ergebnisse der Tests aus dem letzten Kapitel
- Ordnen Sie diese in den Gesamtkontext ein... gut/schlecht? Was kann man verbessern/anders machen? Schätzen Sie auch ab was Veränderungen bringen könnten.
- Was wurde durch die Ergebnisse gezeigt? Geben Sie eine bewertende (selbstkritische) Aussage ab zu Ihrem Schaffen
- Geben Sie einen Ausblick was nun folgen sollte/könnte.

Quellcode 5.1: C-Code Beispiel

```c
void main() {
  const char *first_string = "abc"; //Definieren eines Strings
  const char *second_string = "abc";
  int result = mx_strcmp(first_string, second_string);

  if (result != 0) {
    printf("Vergleich schlug fehl!\n");
  }
}
```

Quellcode 5.2: Python-Code Beispiel

```python
def test_sum():
  assert sum([1, 2, 3]) == 6, "Sollte 6 sein."

if __name__ == "__main__":
  test_sum()
  print("Test bestanden.")
```

Es ist dabei auch darauf zu achten, dass Quelltexte wenn möglich natürlich

nicht über das Seitenende gehen sollten, wie z.B. beim Python-Code Bsp. Setzen Sie daher auch manuell Seitenumbrüche. Quelltexte welche eine Seite übersteigen, sind ohnehin eher als Anhang zu betrachten → siehe dazu Beschreibung im Anhang.

Quellcode 5.3: HTML-Code Beispiel

```
<!DOCTYPE html>
  <head>
    <title>A Sample HTML Document (Test File)</title>
    <meta charset="utf-8">
    <meta name="description" content="Blankes HTML-File zum Testen.">
    <meta name="author" content="Mario Hoffmann">
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <h1>Ein einfaches HTML-File zum Testen</h1>
    <p>Dient nur der stilistischen Darstellung im Latex.</p>
    <p><a href="../somewhere.html">Gehe zu...</a></p>
    <p><a href="https://example.com/html5/download-attribute/">Etwas über das HTML
     5 Download-Attribut.</a></p>
  </body>
</html>
```

Nachfolgend noch ein einfaches Beispiel ein Bild einzubinden. Da es sich um Bild*unterschriften* handelt, gehört diese somit unter die Abbildung, anders als bei Tabellen*überschriften*.
Im LaTeX-Quelltext sieht man die Optionen für das Einbinden des Bildes. Neben `scale` wäre auch `width` möglich mit dem Parameter `linewidth` oder `textwidth`.
Achten Sie vor allem auf die Lesbarkeit der auf dem Bild befindlichen Informationen, unter der stetigen Annahme, dass es sich um ein ausgedrucktes Dokument handelt. Dort gibt es keinen Zoom.

Empfehlenswert wäre soweit möglich mit skalierbaren Vektorgrafiken zu arbeiten. Allerdings müssten Sie diese vor dem Aufruf von LaTeX in PDFs wandeln (inkscape IM-logo.svg -o IM-logo.pdf) oder Inkscape installieren und LaTeX mit –shell-escape aufrufen.

Bedenken Sie bei allen Beschriftungen, egal ob Abbildungen, Tabellen oder Quelltexte,

Abbildung 5.1: Das Logo von Informatik und Medien

Abbildung 5.2: Das Logo von Informatik und Medien

dass Sie, sofern diese nicht Ihrer eigenen Schaffenskraft entsprangen, diese referenzieren. Im Beispiel der Tabelle **??** sieht man, dass sich der Beschreibungstext direkt an der Tabelle von dem im Tabellenverzeichnis unterscheidet, genau um den Punkt der Referenz.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

FIM Fakultät Informatik und Medien
FTTH Fiber to the Home
HTWK Hochschule für Technik, Wirtschaft und Kultur Leipzig
ML Machine Learning

A Anhang - Abbildungen

Grundsätzlich gehören Tabellen und Abbildungen in den Hauptteil der Arbeit. Hat man aber sehr viele oder auch lange Tabellen, die den Lesefluss im Hauptteil stören würden, dann können diese in einen separaten Anhang aufgenommen werden. Wichtig ist in jedem Fall, dass zwischen Hauptteil und Material im Anhang durch geeignete Verweise eine Beziehung hergestellt wird.

B Anhang - Tabellen

C Anhang - Quelltexte

Auch längere Quelltexte gehören nicht in den Hauptteil, sondern entweder in den Anhang oder bei großem Umfang nur auf ein erreichbares Repository (Gitlab o.ä.). Wünscht man Algorithmen im Hauptteil zu erklären, dann kann dies durch Pseudocode erfolgen.