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 12. 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 12. Februar 2024

Vorwort (optional)

In einem Vorwort können persönliche Danksagungen, Ihre eigene Motivation sowie persönliche Erkenntnisse niederschreiben. Inhaltliche Angaben zum Werk werden jedoch nicht gemacht. Zudem ist es möglich hier eine Gendererklärung abzugeben. Gendererklärung: In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit es für die Aussage erforderlich ist.

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.

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

Inhaltsverzeichnis

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

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.

Die Struktur dieser Arbeit gliedert sich wie folgt:

- **Theoretische Grundlagen des Reinforcement Learning:** In diesem Abschnitt werden die grundlegenden Konzepte des Reinforcement Learning erläutert, darunter Markov-Entscheidungsprozesse, Politiken, Wertfunktionen und verschiedene RL-Algorithmen.
- **Das Würfelspiel 'Noch mal':** Hier wird eine detaillierte Beschreibung des Spiels 'Noch mal' gegeben, einschließlich der Regeln, Spielziele und möglicher Strategien.
- **Methodik:** Dieser Abschnitt beschreibt den Ansatz zur Implementierung des Reinforcement Learning-Agenten für 'Noch mal'. Hier werden die Wahl des RL-Algorithmus, die Modellarchitektur und die Trainingsparameter erläutert.
- **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.

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.

1.1 Methodik

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

1.1.1 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.

1.1.2 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.

1.1.3 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.

1.1.4 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.

1.1.5 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.

2 Theoretische Grundlagen

2.1 Neuronale Netze

Neuronale Netze sind ein Modell für künstliche Intilligenz nach 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.

[Ertel S. 290]

2.1.1 Training von NN

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] [Quelle?]

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.

2.2.1 Agent und Umwelt

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.

2.2.2 Markov- Entscheidungsprozess

Jedes RL Problem lässt sich durch den Markov Entscheidungsprozess (Markov decision Prozess) beschreiben. Der MLP 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. Die Abbildung1 stellt den Ablauf eines MDP dar.

Hauptkomponenten des MDP:

- endliche Menge valider Zustände
- endliche Menge valider Aktionen
- Belohnungsfunktion
- Verhaltensstrategie

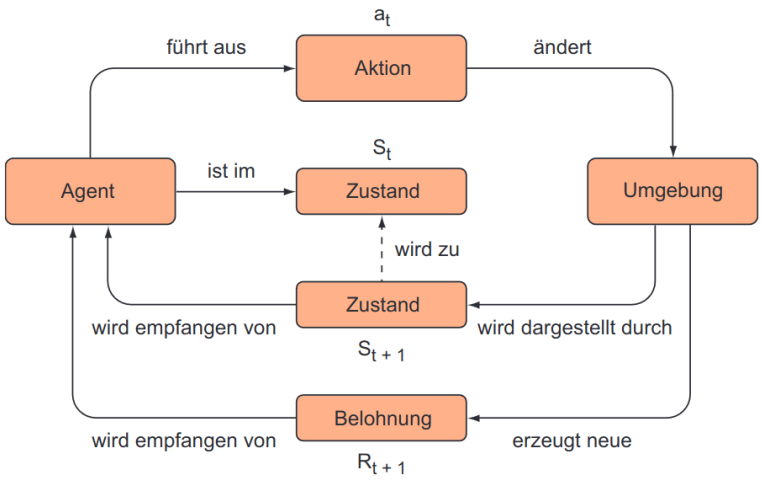


Abbildung 2.1: Quelle:Einstieg DeepLearning Zai/Brown S27

2.2.3 Zustände, Aktionen, Verhaltensstrategie und Belohnungen

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.

[Quelle Zai, Kramer]

2.2.4 Bewertungsmetriken für RL-Agenten

Average Cumulative Rewards

Policy Loss und was noch ?

QUELLE Suchen

2.2.5 PPO

Hier sollte eines Übersicht des PPO alghorhitmus stehen - Entwickelt von John Schulman 2017 wurde zum default reinforcement learning Algorhitmus von OpenAI - viele experts called PPO state of the Art - gute balance zwischen Performance and Comprehention er ist verglichen zu anderen RL Algorithmen simple stable and sample efficiency

3 Konzeption

3.1 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.

3.1.1 Spielablauf Einspieler Variante

Der Spieler Würfelt alle 4 Würfel bestehend aus 2 Farb und 2 Zahlenwürfeln. Der Spieler hat 30 Züge dazu 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.

3.1.2 Spielregeln

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

3.1.3 Optimale Spielstrategie

Um im Spiel 'Noch mal' eine möglichst hohe Anzahl an Punkten zu erhalten, ist es wichtig nach folgenden Strategien zu spielen:

1. **Priorisierung äußerer Spalten**: Äußere Spalten geben mehr Punkte, weshalb es wichtig ist diese komplett auszufüllen.
2. **Beenden von Farben**: Vollständig ausgefüllte Farben geben viele Extrapunkte. Im späten Spielverlauf kann es besser sein Farben komplett zu füllen anstatt Spalten zu werten.
3. **Priorisieren von Sternfeldern**: Jedes ausgefüllte Sternfeld gibt 2 Punkte, eine gute Spielweise ist es so viele Sternfelder wie möglich auszufüllen.
4. **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.

3.1.4 Punktevergabe

Gelingt es dem Spieler eine Spalte komplett auszufüllen, erhält er je nach Spalte Punkte dafür. Äußere Spalten mehr Punkte vergeben werden 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.

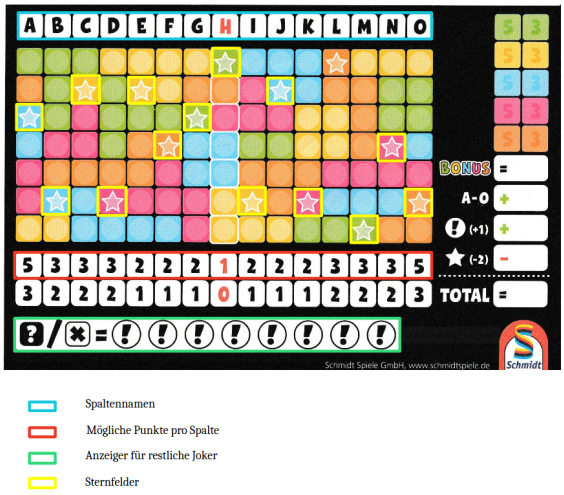


Abbildung 3.1: Quelle: Aus Noch mal! von Schmidt Spiele, Grafik zur Erklärung der Punktevergabe

Abbildung 3.2: Quelle: Aus Noch mal! von Schmidt Spiele, Grafik zur Bewertung der gesammelten Punkte der Einspieler Variante

3.2 Vorgehensweise

Gehört das hier tatsächlich hin? Und wenn ja was soltle dewn nhjier bittte rein?

4 Implementation

4.1 Umsetzung in Unity

Wie wurde Programmiert UML? Flussdiagramm?

4.1.1 Controller

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?>

4.1.2 NumberDice

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> <Bild der Sprites>

4.1.3 ColorDice

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>

4.1.4 GameField

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

4.1.5 FieldSquare

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

- Feld ist ein Sternfeld -> bool
- Farbe des Feldes -> string
- Feld ist ausgefüllt -> bool
- Feld ist verfügbar -> bool
- Clustergröße -> int
- X Koordinate des Feldes -> int
- Y Koordinate des Feldes -> int

4.1.6 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>

4.1.7 Agent

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.

4.2.1 Observations

Zu Beginn jeder Episode (=SChritt) 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 sic hso die Anzahl der Observations unterscheiden würden.

Aufbau der Observations:

- Anzahl der verbleibenden Joker: int[0-11]
- Anzahl der gespielten Runden: int[0-30]
- Ergebnis der Zahlenwürfel: int[0-5]
- Ergebnis der Farbwürfel: Vector6[0,1]

Für jedes Feld:

4.2.2 Bewertungsmetriken für RL-Agenten

- Farbe des Feldes: Onehot vector5[0,1]
- Ist das Feld verfügbar: bool
- Ist das Feld bereits abgestrichen: bool
- Ist das Feld ein Sternfeld: bool

<skript Collect Observations>

//TODO 2 verschiedene Varianten was mach ich damit?

4.2.3 Actions

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:

- Index des gewählten ZahlenWürfels: int (0-1)
- Index des gewählten Farbwürfels: int(0-1)
- Jokerzahl int 0-4
- X Koordinate des gewählten Feldes: int (0-14)
- Y Koordinate des gewählten Feldes: int(0-7)
- 4 x Continous Action für die Auswahl der Nachbarn

what to do unterscheidliche actions

4.2.4 Erklärung des Alghorhytmus mit dem Beispielhaften Ausgabevektor (1,1,0,3,4, 0.6, 0.5, 0.4, 0.8)

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.

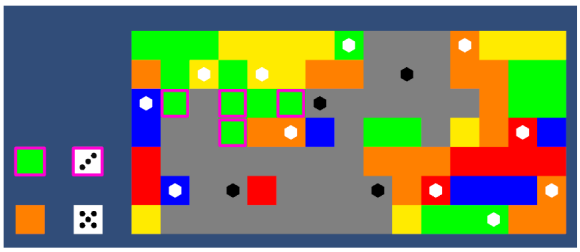


Abbildung 4.1: Valide Felder für das gewählte Würfelergebnis wurden markiert

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.

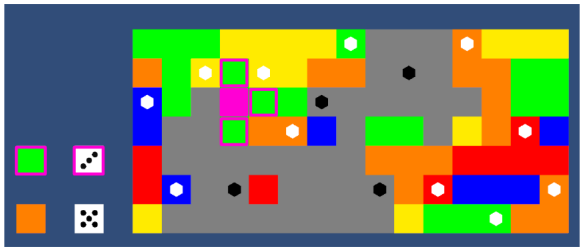


Abbildung 4.2: Feld(3,4) wird in die pickedField Liste aufgenommen und benachbarte Felder werden zurückgegeben.

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.

- Feld(3,5) 0 bis 0.33
- Feld(4,4) 0.33 bis 0.66
- Feld(3,3) 0.66 bis 0.99

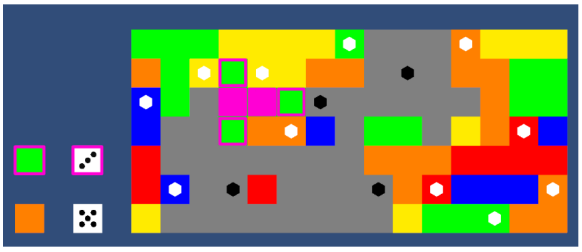


Abbildung 4.3: Feld(4,4) ist das nächste gewählte Feld und wird in pickedFields aufgenommen

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.

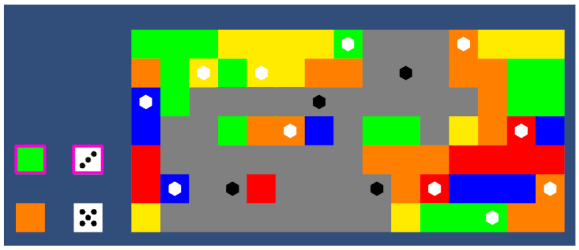


Abbildung 4.4: Felder wurden gewählt und ausgefüllt

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

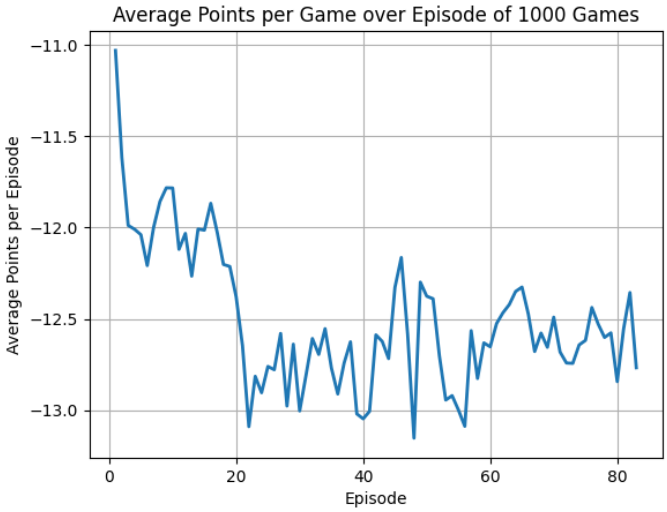


Abbildung 4.5: Das Logo von Informatik und Medien

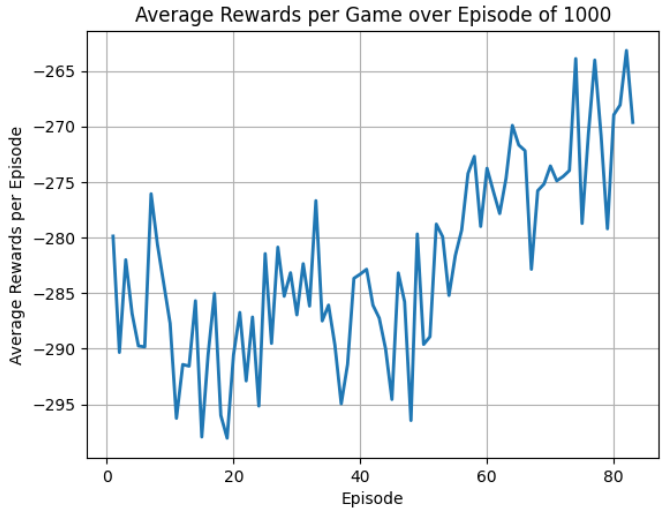


Abbildung 4.6: Das Logo von Informatik und Medien

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.

4.3.3 Training des Agenten ohne Zusatz Belohnungen

Da auch der vorhergehende Versuch keinen klaren trend zu Verbesserung aufzeigte, entschied ich mich, einzig die Punktebewertung des Spiel als Rewards zu benutzen. Auf diese Weise bekommt der Agent nur Rewards für Punkte welche er auch für das spielen erhält. Auf diese Weise, kann er sich keine 'Falschen' Strategien erarbeiten, welche aus falsch festgelegten Rewards resultieren.

Rewards gibt es nun für: Ausfüllen von 'Sternfeldern' -> 2.0f Auffüllen einer kompletten Spalte -> (je nach Spalte 5-1) Ausfüllen einer kompletten Farbe -> (5 Punkte) Übrig gebliebene Joker nach Ablauf der 30 Spielrudnen -> 1 Punkt pro Joker

4.3.4 Verbesserung des Trainings

Verlängerung der Spieldauer

Training mit Speziellen Feldern

Verkleinerung des Feldes

Änderung der Policy

4.3.5 Training Auswahl Initiales Feld

Da der Agent keine großen Fortschritte erzielen konnte, entscheid 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.

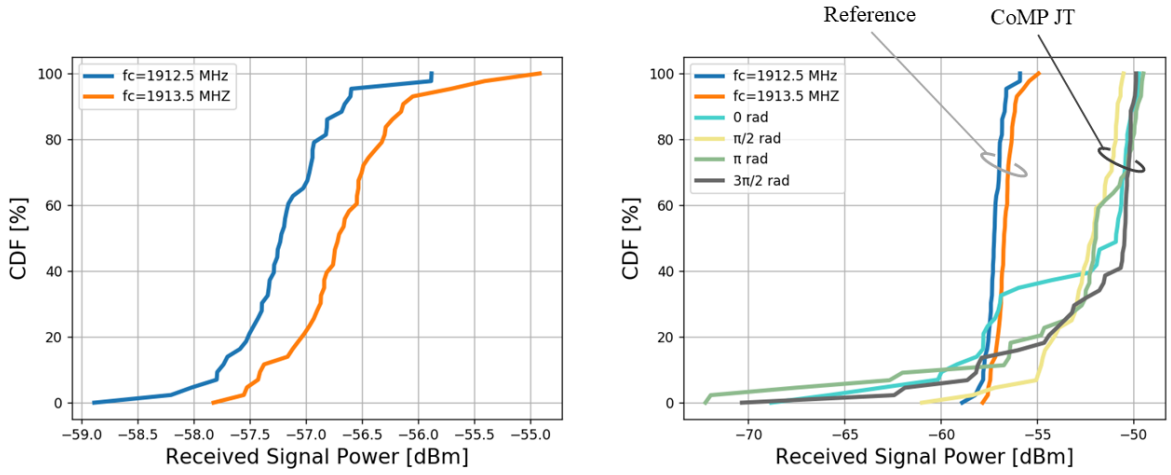
Zusammenhängende Ergebnisse werden vorteilhaft in Form von Tabellen (wie in 4.1) dargestellt.

Tabelle 4.1: Laufzeiten in sec, gemessen mit einem Intel xy-8 Ghz Prozessor

	Verfahren X	Verfahren Y	Verfahren Z
Problem A	300	340	210
Problem B	730	580	540
Problem C	610	420	440
Problem D	895	790	520

Funktionale Zusammenhänge hingegen lassen sich in Form von Diagrammen oder Grafiken (Siehe 4.7) darstellen.

Ein guter Ergebnisteil erfordert Literaturzitate (vielleicht auch gar keine) und die Bekanntgabe des Ergebnisses sollte von einer Interpretation und Diskussion begleitet werden.



(a) Reference Measurement

(b) Reference and CoMP JT with Phase regulation Measurement

Abbildung 4.7: Statistical analysis of the received power

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

- Setzen Sie nun den Plan aus dem dritten Kapitel wenigstens prototypisch um
- Ziel ist es Ihre Überlegungen und Darlegungen zu beweisen
- Diskutieren von Problemen
- Zusätzlich zu der Umsetzung bedarf es (von Ihnen zu definierender) Tests, um die Funktionalität zu beweisen
  - Laufzeittests, Lasttests, etc.
  - Produzieren Sie Ergebnisse um zu zeigen, dass Ihr Ansatz funktioniert und besser/schlechter ist
  - „schlechter“ sein, bedeutet nicht, dass Ihre Arbeit schlecht ist! Am Ende zählt Ihr Ansatz und die Idee

5 Auswertung und Ausblick

5.1 Bewertung der Ergebnisse

5.2 Weitere 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.

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.