

Bearbeitungsbeginn: 01.03.2023

Vorgelegt am: 21.08.2023

# **Thesis**

zur Erlangung des Grades

**Bachelor of Science**

im Studiengang Medieninformatik

an der Fakultät Digitale Medien

***Alexander Reiprich***

***Matrikelnummer: 263006***

**Computer Vision als Werkzeug zur Steuerung von  
Videospielen – Analyse und Entwicklung eines  
Programmes basierend auf Reinforcement Learning**

*Erstbetreuer:* Prof. Dr. Uwe Hahne

*Zweitbetreuer:* Prof. Dr. Ruxandra Lasowski



## Abstract

Videospiele erfreuen sich seit Jahren immer weiter wachsender Beliebtheit (Bitkom Research 2023). Während Videospiele sich schon immer an den Menschen als Spieler richten, gab es in den letzten Jahren enorme Fortschritte im Bereich künstliche Intelligenz, was zur Folge hat, dass auch Videospiele mittlerweile von Programmen erlernt und gespielt werden können (Mnih et al. 2013). In dieser Arbeit soll diesbezüglich ein Modell entwickelt werden, welches das Spiel „StepMania 5“ durch rein visuellen Input erlernt und spielt. Davor wird das Konzept und die Theorie hinter maschinellern Lernen erklärt, als auch Beispiele im Themenbereich „Künstliche Intelligenz und Videospiele“ aufgeführt. Abschließend wird ein Fazit aus den Erkenntnissen gezogen und ein Ausblick in die Zukunft von künstlicher Intelligenz in Medien, und insbesondere in Videospielen, gegeben.

The popularity of video games has increased for years (Bitkom Research 2023). While video games have always been aimed at humans as players, there has been enormous progress in the field of artificial intelligence in recent years, resulting in the fact that video games can now also be learned and played by computer programs (Mnih et al. 2013). In this thesis a model will be developed, which learns and plays the game „StepMania 5“ by pure visual input. Before that, the concept and theory behind machine learning will be explained, and examples in the topic area „Artificial Intelligence and Video Games“ are shown. Finally, a conclusion is drawn from the findings and an outlook on the future of artificial intelligence in media, and especially in video games, is given.



## Inhaltsverzeichnis

Abstract .....	III
Inhaltsverzeichnis.....	V
Abbildungsverzeichnis.....	VII
Tabellenverzeichnis.....	VIII
Listingverzeichnis .....	IX
Abkürzungsverzeichnis .....	XI
1. Einführung .....	1
1.1. Motivation und Ziel der Arbeit.....	1
1.2. Aufbau der Thesis.....	2
2. Grundlagen.....	3
2.1. Was ist KI? .....	3
2.2. Was ist StepMania 5? .....	4
2.3. Machine Learning.....	6
2.3.1. Reinforcement Learning.....	8
2.3.2. Deep Learning.....	10
2.3.3. Evolutionary Algorithms.....	13
3. KIs im Bereich Unterhaltung.....	17
3.1. Künstliche Intelligenz in Videospielen .....	17
3.2. Künstliche Intelligenz in Unterhaltungsmedien .....	20
4. Entwicklung.....	23
4.1. Zielsetzung und Anforderungen .....	23
4.2. Umsetzung der verschiedenen Bestandteile .....	23
4.2.1. Aufsetzen eines Environments.....	23
4.2.2. Jupyter Notebook .....	32
4.2.3. Custom Theming.....	34

4.2.4.	Tesseract und Template Matching .....	37
4.2.5.	Erster Lernversuch .....	42
4.2.6.	Neues Reward-System .....	44
4.2.7.	Datenkomprimierung .....	51
4.2.8.	Abschließender Trainingslauf .....	55
4.3.	Bewertung und Einordnung .....	57
5.	Fazit und Ausblick .....	59
Literaturverzeichnis.....		63
Eidesstattliche Erklärung.....		71

## Abbildungsverzeichnis

Abb. 1 - Screenshot vom Gameplay von StepMania 5 .....	5
Abb. 2 - Screenshot des Resultscreens nach Abschluss eines Charts .....	6
Abb. 3 - Interaktion zwischen Agent und Environment (aus: Russell und Norvig, 2016, S.35, Figure 2.1).....	7
Abb. 4 - Prozess der Evolution (Sloss und Gustafson 2020, S.314, Fig. 16.3) .....	14
Abb. 5 - Ablaufdiagramm des Lernprozesses, Teile im Environment und Teile im Modell sind farblich unterteilt (Eigene Darstellung) .....	28
Abb. 6 - Zwei Code-Cells eines Jupyter Notebooks innerhalb der Entwicklungsumgebung Visual Studio Code .....	33
Abb. 7 - Screenshots des Gameplays, bei welchem die durch den Template Matching Algorithmus erkannten Elemente markiert wurden .....	34
Abb. 8 – Auszug der metrics.ini-Datei im StepMania 5 Ordner.....	35
Abb. 9 – Auszug der Other.lua-Datei im StepMania 5 Ordner.....	36
Abb. 10 - User Interface nach den Änderungen am Theme.....	36
Abb. 11 - Erkannte Zahl als Output (oben) mit dem Screenshot der Zahl als Input (unten) .....	37
Abb. 12 - Ergebnisscreen eines Charts .....	38
Abb. 13 - Ergebnis des genannten Funktionsaufrufs (oben) und die dazu gehörende Observation (unten).....	39
Abb. 14 - Ergebnis des Benchmarktests mit Tesseract .....	41
Abb. 15 - Ergebnis des Benchmarktests nach beiden Änderungen .....	42
Abb. 16 - Screenshot von Tensorboard - Darstellung der Mittelwerte der Episodenlänge, des Rewards, sowie der Wert der Exploration Rate des ersten Lernversuchs über die Lerndauer .....	42
Abb. 17 - Screenshot von Tensorboard - Darstellung der Learning Rate und der Loss Rate des ersten Lernversuchs über die Lerndauer.....	43
Abb. 18 - Screenshot von Tensorboard - Darstellung der Mittelwerte der Episodenlänge, des Rewards, sowie der Wert der Exploration Rate nachdem das Reward-System überarbeitet wurde .....	49
Abb. 19 - Screenshot von zwei "Holds" .....	50
Abb. 20 - Bildschirmaufnahmen des Gameplays, in Graustufen.....	51

Abb. 21 - Bildschirmaufnahmen des Gameplays, als Binärbild .....	52
Abb. 22 - Bildschirmaufnahme des Gameplays, als Vergleichsbild.....	52
Abb. 23 - Bildschirmaufnahme des Gameplays, auf welchem u.a. zwei „Hold“-Pfeile zu sehen sind, als Binärbild.....	53
Abb. 24 - Bildschirmaufnahme des Gameplays, Vergleichsbild von Abb. 21 .....	53
Abb. 25 - Weitere Bildschirmaufnahme des Gameplays, als Vergleichsbild .....	54
Abb. 26 - Graph in TensorBoard, welcher den durchschnittlichen Reward pro Episode über 1,4 Millionen Schritte zeigt. ....	55
Abb. 27 - Performance der unterschiedlichen DQN-Variationen über 57 Atari-Spiele (aus: Hessel et al. 2017, S.1, Figure 1).....	56

## Tabellenverzeichnis

Tabelle 1 - Anzahl Tastendrucke für jede Taste nach 10 Minuten .....	43
Tabelle 2 - Anzahl Tastendrucke pro Taste, nach 5.000, 10.000 und 25.000 Schritten .....	44



## Listingverzeichnis

Listing 1 - Code der init-Funktion des Environments.....	27
Listing 2 - Code der step-Methode des Environments.....	29
Listing 3 - Code der reset-Methode des Environments.....	30
Listing 4 - Code der get_observation-Methode des Environments.....	31
Listing 5 - Code der get_over-Methode des Environments .....	31
Listing 6 - Code der get_reward-Methode des Environments .....	32
Listing 7 - Code der analyze_results-Funktion im Pattern Recognition Modul .....	40
Listing 8 - Code der analyze_score-Methode im Pattern Recognition Modul.....	41
Listing 9 – Code der überarbeiteten get_reward-Methode des Environments.....	47
Listing 10 – Code der input_expected- und check_for_color-Methoden des Pattern Recognition Moduls .....	48
Listing 11 - Code der downscaleImage-Methode im Image Analysis Modul .....	51



## Abkürzungsverzeichnis

CNN .....	<i>Convolutional Neural Network</i>
DDR .....	<i>Dance Dance Revolution</i>
DQN .....	<i>Deep-Q-Network</i>
et al. ....	<i>et alia</i>
etc. ....	<i>et cetera</i>
GPT .....	<i>Generative Pretrained Transformer</i>
KL.....	<i>Künstliche Intelligenz</i>
LLM .....	<i>Large Language Model</i>
MDP .....	<i>Markov Decision Process</i>
OCR.....	<i>Optical Character Recognition</i>
RL.....	<i>Reinforcement Learning</i>
u.a. ....	<i>unter anderem</i>
U.S.....	<i>United States (of America)</i>
WGA .....	<i>Writers Guild of America</i>
z.B. ....	<i>zum Beispiel</i>



In dieser Arbeit wird das generische Maskulinum genutzt, um Personenbezeichnungen in neutraler Form zu verwenden. Alle männlichen Bezeichnungen beziehen sich implizit auf Personen jeglichen Geschlechts und sollen keinerlei Ausschluss oder Vorurteil gegenüber anderen Geschlechtern darstellen. Ziel ist es, den Lesefluss zu erleichtern, ohne dabei die Gleichbehandlung und Inklusion aller Geschlechter zu vernachlässigen.

## 1. Einführung

### 1.1. Motivation und Ziel der Arbeit

Künstliche Intelligenz ist derzeit im Trend wie noch nie zuvor. Laut dem Artificial Intelligence Index Report 2022 der Stanford Universität (Zhang et al. 2022) hat sich die Anzahl der Publikationen im Bereich KI im Vergleich von 2010 zu 2021 mehr als verdoppelt, insbesondere in den Themenbereichen Pattern Recognition und Machine Learning. Sowohl durch den Hype des Chat-Bots „ChatGPT“ Anfang 2023 als auch durch das wachsende Interesse an KI-Kunst durch Dienste wie „midjourney“ sind die Fähigkeiten von KI ein aktuelles Thema. Dabei findet dieser Diskurs nicht nur im Bereich der Informatik, sondern in der ganzen Gesellschaft statt. Laut Bayerischem Rundfunk wird an manchen Hochschulen und Universitäten beispielsweise das Nutzen von künstlicher Intelligenz, z.B. in Form von ChatGPT, bereits verboten (Barthel und Ciesielski 2023).

Mit der Generierung von Texten oder Bildern durch künstliche Intelligenz kommen natürlich auch Probleme – sowohl moralische als auch juristische. Nachdem 2022 ein Werk einer KI als Sieger eines Kunstwettbewerbes gekürt wurde kam es zu Diskussionen, inwiefern Bilder, welche von KI erzeugt wurden, tatsächlich Kunst, und die Ersteller tatsächlich Künstler sind (Roose 2022). Ebenfalls war es unklar, ob mit KI erstellte Bilder rechtlich durch Copyright geschützt sind. Nutzer behaupten, dass die eingegebenen Befehle, die zu diesem Bild geführt haben, ihr eigenes Werk wären. Dies ist jedoch nach einem Urteil des U.S. Copyright Office nicht der Fall (Novak 2023).

Beschäftigt man sich mit der Thematik „kreative künstliche Intelligenz“ etwas genauer, stößt man schnell darauf, dass es mittlerweile auch Kanäle auf der Livestream-Plattform Twitch gibt, welche komplett KI-gesteuert sind. Eine der bekanntesten Kanäle in diesem Bereich ist „Neuro-Sama“. In den Livestreams dieses Kanals sitzt keine Person vor der Kamera, sondern eine KI, welche Videospiele spielt und auf ihre Zuschauer eingeht (Xiang 2023).

All diese Dinge haben mich dazu inspiriert das Thema KI im Bereich Medien, und insbesondere im Bereich Videospiele, genauer zu untersuchen. Diese Arbeit ist dabei in zwei Teile unterteilt – im theoretischen Teil soll zuerst die Funktionsweise von KI erklärt werden, wobei dort genauer auf die Funktionsweise einzelner Methoden und Algorithmen eingegangen werden soll. Ebenso soll die Existenz von KI in Videospielen und Unterhaltungsmedien betrachtet werden. Im praktischen Teil dieser Arbeit soll selbst eine KI zur Steuerung eines Videospieles entwickelt und trainiert werden. Dabei soll der Entwicklungsprozess und die Herangehensweise beschrieben und erklärt, sowie schlussendlich retrospektiv analysiert werden.

## 1.2. Aufbau der Thesis

Im ersten Kapitel wird zuerst die Motivation dargestellt, um anschließend den theoretischen und praktischen Teil dieser Ausarbeitung zu erläutern. Im darauf folgenden Unterkapitel wird der Aufbau der Arbeit beschrieben.

Das zweite Kapitel beschäftigt sich mit den Grundlagen, welche dargestellt werden, um ein genaueres Verständnis des Themas und der nachfolgenden Kapitel zu gewährleisten.

Nachdem die Grundlagen des Themas im zweiten Kapitel definiert wurden, werden diese genutzt, um im dritten Kapitel genauer auf den Bereich KI in Videospielen und Unterhaltungsmedien einzugehen. Hier werden Beispiele aufgeführt, welche in diese Bereiche eingeordnet werden können.

Das vierte Kapitel beinhaltet den praktischen Teil der Arbeit. Hier wird zuerst der Aufbau des Projektes sowie die Anforderungen beschrieben, um dann in den

darauffolgenden Unterkapiteln auf den Entwicklungsprozess einzugehen. Abschließend wird das Vorgehen analysiert, bewertet und kritisiert.

Im fünften Kapitel werden die Ergebnisse der vorangegangenen Kapitel in einem Fazit zusammengefasst und es wird auf die Zukunft von künstlicher Intelligenz in Medien eingegangen.

## 2. Grundlagen

### 2.1. Was ist KI?

Die Abkürzung „KI“ steht für „künstliche Intelligenz“ – doch was genau versteht man unter diesem Begriff? Eine klare, einheitliche Definition gibt es laut dem Buch „Artificial Intelligence: A Modern Approach“ der Autoren Stuart Russell und Peter Norvig nicht, da es basierend auf der vertretenden Ansichtsweise und der zu betrachtenden Kategorie unterschiedliche Perspektiven gibt, das Konzept künstliche Intelligenz zu definieren (Russell und Norvig 2016, S. 1–5). Der US-amerikanische Informatiker Nils John Nilsson definiert künstliche Intelligenz als „...*activity devoted to making machines intelligent...*“, wobei Intelligenz hier als „...*quality that enables an entity to function appropriately and with foresight in its environment...*“ betrachtet wird (Nilsson 2013, S. 13). Die Europäische Kommission bezeichnet KI als „*Systeme mit einem „intelligenten“ Verhalten, die ihre Umgebung analysieren und mit einem gewissen Grad an Autonomie handeln, um bestimmte Ziele zu erreichen*“ (Europäische Kommission 25.04.2018). Beide Definitionen sind relativ vage und könnten genauso auf ein Thermostat zutreffen, welches die Temperatur misst („Umgebung analysieren“) und reguliert („mit einem gewissen Grad an Autonomie handeln“) um die eingestellte Temperatur zu erreichen („um bestimmte Ziele zu erreichen“). Viele würden dies jedoch nicht unter dem Begriff „künstlicher Intelligenz“ verstehen (Sheikh et al. 2023, S. 16).

Für ein besseres Verständnis dieser Arbeit muss eine einheitliche Definition festgelegt werden. Daher soll der Begriff „künstliche Intelligenz“ im Folgenden ein Programm beschreiben, welches selbstständig durch die Verwendung von Algorithmen lernt oder

gelernt hat, Muster zu erkennen um basierend auf diesen Mustern Schlüsse zu ziehen, Entscheidungen zu treffen und Aktionen auszuführen.

Das Thema KI gerat durch „ChatGPT“ und die Möglichkeiten, die das Tool brachte, stark in die Öffentlichkeit. Dabei ist ChatGPT nur ein Teil des Fortschrittes, den OpenAI in den letzten Jahren gemacht hat. Generative Pretrained Transformer 4, kurz GPT-4, ist ein sogenanntes Large Language Model (LLM), welches mit einer enormen Anzahl von Daten aus dem Internet trainiert wurde. Das Modell basiert, wie auch ChatGPT, auf dem Konzept, basierend auf Inhalt und dem Kontext eines Satzes, Wörter vorherzusagen um Antworten zu generieren (Bubeck et al. 2023). Die Möglichkeiten scheinen dabei endlos – Aufgaben wie Gedichte verfassen oder Code schreiben sind nur die Spitze des Eisberges. Während ChatGPT „nur“ auf der Vorläuferversion GPT-3 basiert, ist dieses Modell trotzdem ein enorm starkes Tool, mit dem sich die oben genannten Aufgaben problemlos bewältigen lassen. Dennoch ist dieses Modell kein Alleskönner. *„ChatGPT hat Grenzen hinsichtlich seines Wissens, Kontextverständnisses, ethischen Überzeugungen, Zugriffs auf vertrauliche Informationen und kreativen Fähigkeiten.“* – so beschreibt die KI ihre Grenzen selbst. Die Entwicklung von GPT-4 zielt darauf ab, diese Schwächen auszugleichen – insbesondere das Kontextverständnis, welches im direkten Vergleich logischer und zusammenhängender wirkt (Bubeck et al. 2023).

Durch die Präsenz von ChatGPT in den Medien ist künstliche Intelligenz als Ganzes vollständig in der Gesellschaft angekommen, wobei das Verständnis, was genau KI eigentlich macht und wo es eingesetzt wird, laut einer Umfrage des TÜV Verbands noch Lücken aufweist (TÜV Verband 2021). Während ein LLM wie ChatGPT unter der festgelegten Definition von KI fällt, hat künstliche Intelligenz viele andere Formen: In dieser Arbeit soll beispielsweise eine künstliche Intelligenz entwickelt werden, welche ein Videospiel lernt.

## 2.2. Was ist StepMania 5?

Für den praktischen Teil dieser Arbeit soll eine KI für StepMania 5 entwickelt werden. StepMania 5 ist sowohl ein Open-Source Rhythmusspiel als auch eine Spielengine, welche es dem Nutzer ermöglicht, beliebte Rhythmusspiele auf dem PC zu simulieren



und zu spielen. Der Gameplay-Aspekt ist vergleichbar mit beliebten Arcade-Spielen wie Dance Dance Revolution (DDR), jedoch kann StepMania auch mit der Tastatur oder einem Controller gespielt werden und basiert, anders als DDR, nicht nur auf dem Konzept einer Tanzmatte als Eingabegerät (StepMania 2020).

Der Spieler kann aus verschiedenen sogenannten Charts wählen. Diese sind entweder von anderen Nutzern erstellt oder beim Download des Spiels bereits enthalten. Jeder Chart besteht aus Pfeilen, welche sich von unten nach oben bewegen und dabei dem Rhythmus des im Chart spielenden Liedes folgen. Der Spieler hat hier die Aufgabe, zum richtigen Zeitpunkt, wenn die sich bewegenden Pfeile auf die stationären Pfeile am oberen Bildschirmrand treffen, die passenden Tasten zu drücken. Dabei bekommt der Spieler Punkte, je nach Genauigkeit des Treffzeitpunktes. Hat er den Chart zu Ende gespielt, folgt der Resultscreen, auf welchem der Spieler eine Note bekommt, welche auf der Anzahl und der Genauigkeit der getroffenen Pfeile basiert.

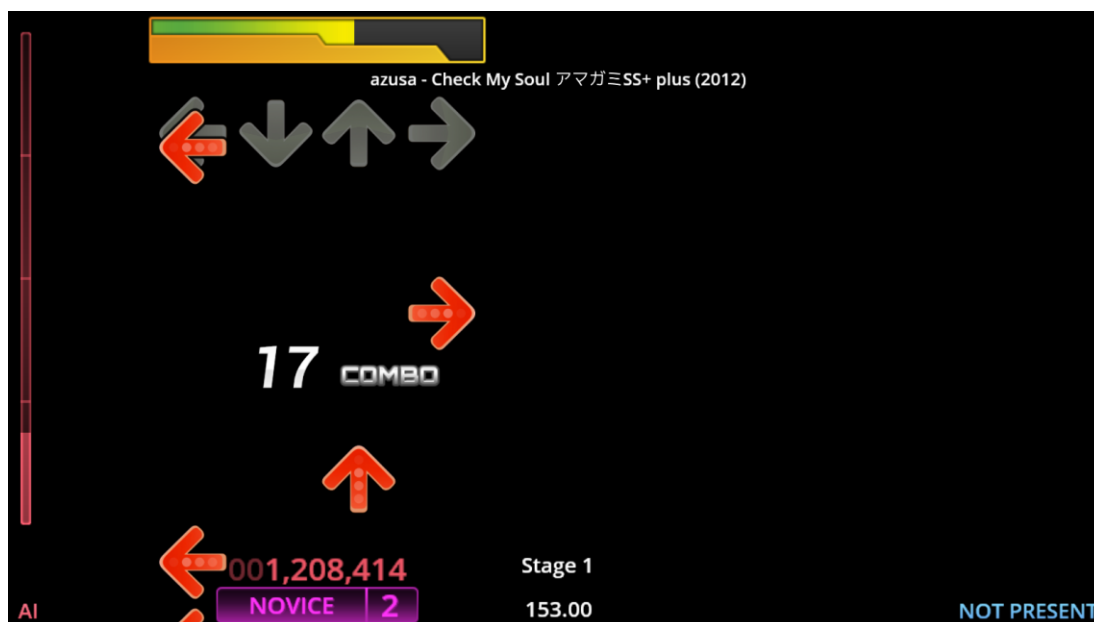


Abb. 1 - Screenshot vom Gameplay von StepMania 5

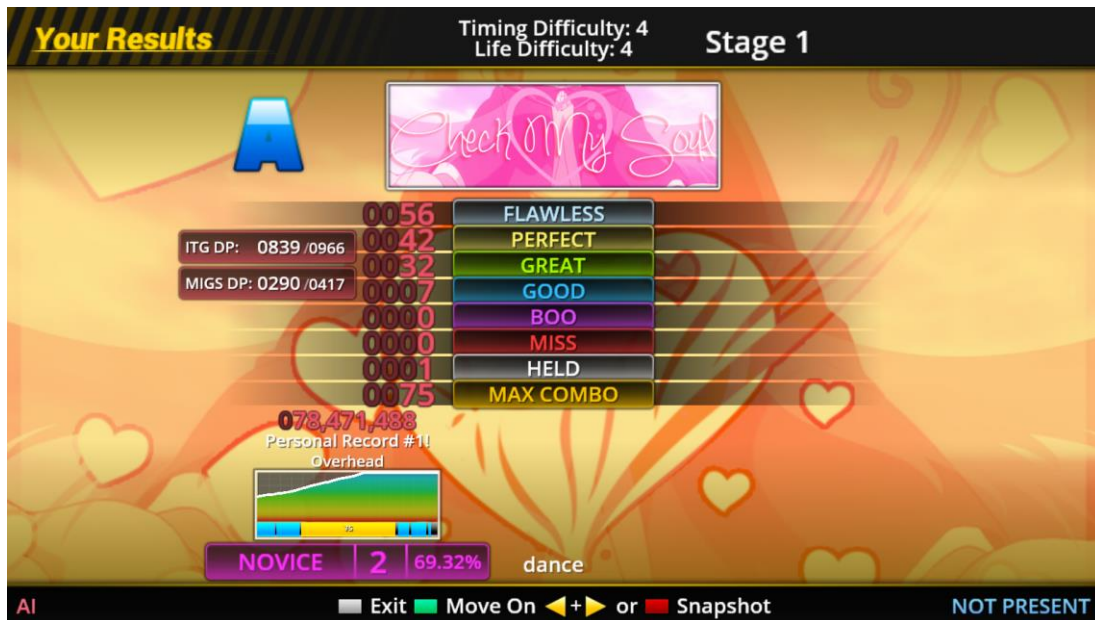


Abb. 2 - Screenshot des Resultscreens nach Abschluss eines Charts

### 2.3. Machine Learning

In der Praxis besteht eine künstliche Intelligenz oft aus zwei Teilen – einem Agent (hier wird der englische Begriff genutzt, und nicht der gleich geschriebene, deutsche) und einem Environment. Das Grundkonzept der Interaktion zwischen den beiden Teilen ist einfach: Der Agent nimmt Informationen aus dem Environment, und führt, basierend auf diesen Informationen, Aktionen im Environment aus. Was für eine Information der Agent bekommt, wie er sie erhält und schlussendlich verarbeitet, kann dabei je nach Art der künstlichen Intelligenz stark variieren (Russell und Norvig 2016, S. 34–35).

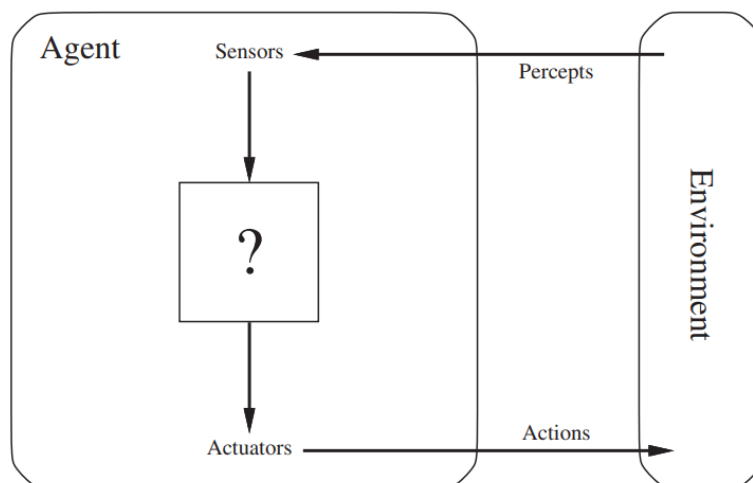


Abb. 3 - Interaktion zwischen Agent und Environment (aus: Russell und Norvig, 2016, S.35, Figure 2.1)

Die ausgeführte Aktion wird dabei von dem Environment bewertet, und der Agent bekommt Feedback. Dieses Feedback nutzt der Agent um zu lernen (Russell und Norvig 2016).

Dieses Grundkonzept ist stark runtergebrochen und nicht auf jede KI, die unter die etablierte Definition fällt, anwendbar. Muhammad Usama et al. haben in dem Paper „Unsupervised Machine Learning for Networking“ beispielsweise das Konzept des unbeaufsichtigten Lernens untersucht. Dieser Begriff des „unsupervised Learning“, also unbeaufsichtigtes Lernen, beschreibt die Verwendung von Daten ohne vorab definierte Wertung. Hier wird kein direktes Feedback gegeben. In diesem Fall muss sich das Modell selbstständig orientieren und ohne Hilfe Muster oder Strukturen in den unbearbeiteten Datensätze erkennen (Usama et al. 2019).

Während unbeaufsichtigtes Lernen eine Art des maschinellen Lernens ist, welche vom klassischen Konzept stark abweicht, gibt es viele Subkategorien, welche dem Konzept folgen, aber sich in der Art des Lernens voneinander unterscheiden. In diesem Kapitel sollen zwei unterschiedliche Varianten vorgestellt werden. Zum einen wird das Konzept des „Reinforcement Learning“, also „verstärktes Lernen“ behandelt, welches im praktischen Teil dieser Arbeit genutzt wird. Zum anderen werden auch sogenannte „Evolutionary Algorithms“ vorgestellt. Li et al. beschreiben in dem Paper „Survey on Evolutionary Deep Learning“ dieses Konzept als Algorithmen, welche nach den biologischen Prinzipien von natürlicher Selektion und Genetik arbeiten, um so durch Abstammung optimales Verhalten zu erlernen (Li et al. 2022a). Beide Varianten sind relevant für den Einsatz von KI in und mit Videospielen.

### 2.3.1. Reinforcement Learning

Reinforcement Learning (RL) beschreibt ein Machine Learning Konzept, bei welchem der Agent durch pures Ausprobieren, auch „Trial and Error“ genannt, lernt. Die ausprobierten Aktionen werden durch einen Reward, also einer Belohnung, bewertet, welchen der Agent nutzt, um die nächsten Aktionen zu bestimmen. Die zwei essentiellen Bestandteile, die RL von anderen Arten des Lernens abgrenzen, ist die eben angesprochene „Trial and Error“-Methodik und die Reward-Vergabe, welche zur Folge hat, dass eine Aktion Auswirkungen auf die Rewards der nachfolgenden Aktionen hat. Nach Sutton und Barto in dem Buch „Reinforcement Learning. An Introduction“ lässt sich RL nicht in unbeaufsichtigtes Lernen, aber auch nicht in beaufsichtigtes Lernen einordnen, da die Kriterien für beide Konzepte nicht erfüllt werden: das Modell baut zwar nicht auf im Voraus gewertete Daten auf, wie es bei einem unbeaufsichtigtem Lernmodell der Fall ist, es versucht aber auch nicht, Strukturen aus Daten zu erkennen, sondern fokussiert sich darauf, die erhaltene Belohnung zu maximieren, wie es bei einem beaufsichtigtem Modell der Fall ist (Sutton und Barto 2020).

Eine zentrale Rolle des Reinforcement Learnings spielt der Markov Decision Process (MDP). Der MDP ist ein mathematischer bzw. stochastischer Entscheidungsprozess, welcher, anders als andere Entscheidungsprozesse, nur den aktuellen Stand des Systems und die aktuell gewählte Aktion betrachtet. Der Ablauf des Prozesses ist vergleichbar mit dem Grundkonzept des Reinforcement Learnings: Steht eine Entscheidung an, wird eine Aktion ausgewählt. Dies hat zwei Konsequenzen – zum einen bekommt der Agent durch die Aktion Feedback in Form eines Rewards und zum anderen wird eine Wahrscheinlichkeit berechnet, wie sich das System nach der Aktion verändert haben könnte. Das Ziel des Prozesses ist die Optimierung einer sogenannten Behavior-Policy, also eine Abfolge von Aktionen, welche die Performance des Systems über einen langen Zeitraum hin optimiert (Puterman 1990). Im Kontext des RL bedeutet dies, dass zu jedem Input der bestmögliche Reward gefunden wird und das Modell so perfektioniert wurde. Dieser Prozess wird so lange wiederholt, bis ein Endpunkt erreicht wurde oder eine vorher bestimmte Anzahl von Wiederholungen oder eine Zeitspanne vergangen ist (Shao et al. 2019).

Neben dem MDP ist ein weiterer wichtiger Teil des Reinforcement Learning die Unterscheidung zwischen Off-Policy und On-Policy Algorithmen. Der Unterschied

zwischen den beiden liegt darin, wie viele Policies das Modell nutzt. In dem Paper „A Survey of Deep Reinforcement Learning in Video Games“ beschreiben die Autoren Shao et al. die Abgrenzung wie folgt: *„Off-policy RL algorithms mean that the behavior policy used for selecting actions is different from the learning policy. On the contrary, behavior policy is the same with the learning policy in on-policy RL algorithms.“* (Shao et al. 2019).

Bei Off-Policy Algorithmen wird zwischen einer Strategie für Exploration, im Folgenden „Exploration-Policy“ genannt, und einer Strategie für Exploitation, „Exploitation-Policy“, unterschieden. Ersteres beschreibt das Lernen, zweiteres das Ausführen von Gelerntem (García und Fernández 2015, S. 1438).

Zuletzt kann man RL-Ansätze in zwei Kategorien differenzieren: Ansätze, die auf einem Modell basieren („model-based“) und Ansätze, die ohne Modell funktionieren („model-free“). Der Begriff Modell bezeichnet hier die Existenz eines fiktiven Environments, an welchem Vorhersagen über das Ergebnis der Handlungen und das daraus resultierende Feedback getroffen werden. Normalerweise befindet sich der Agent in einem Environment und hat keinen Zugriff darauf. Mit einem model-based Ansatz besitzt der Agent ein eigenerstelltes Modell seines Environments und kann dementsprechend Aktionen in verschiedenen Zuständen simulieren, ohne sie im tatsächlichen Environment auszuführen (Atkeson und Santamaria 1997; Luo et al. 2018).

Innerhalb der model-free Ansätze wird zwischen value-based und policy-based RL-Ansätzen unterschieden. Der Unterschied liegt hier in der Optimierung der Entscheidungsstrategie des Agents. Auch hier fassen Shao et al. das Konzept gut zusammen: *„In value-based RL, agents update the value function to learn suitable policy, while policy-based RL agents learn the policy directly.“* (Shao et al. 2019).

Der Zwischenschritt über die Wertefunktion wird also bei policy-based Algorithmen übersprungen. Bei einem value-based Algorithmus basieren die Entscheidungen hingegen (Shao et al. 2019) auf der Wertefunktion – die Policy des Modells ergibt sich hier nur implizit aus den Werten dieser Funktion (Karunakaran 2020a; Shao et al. 2019).

Beide Konzepte können auch kombiniert werden können, wie beispielsweise im Actor-Critic-Ansatz. Hier gibt es zwei Parteien, den Actor und den Critic.

Karunakaran, Professor der Universität Sydney, beschreibt das Verfahren wie folgt: *„The actor decided which action should be taken and critic inform the actor how good was the action and how it should adjust. The learning of the actor is based on policy gradient approach. In comparison, critics evaluate the action produced by the actor by computing the value function“* (Karunakaran 2020b).

Für den Anwendungsfall dieser Arbeit wird ein DQN genutzt. Unter welche der angesprochenen Kategorien dieser Algorithmus fällt, wird im folgenden Kapitel weiter erläutert.

### 2.3.2. Deep Learning

Die Einsatzmöglichkeiten von klassischen Machine Learning Methoden sind begrenzt. Im Bereich der Verarbeitung von rohen Datensätzen stoßen konventionelle Systeme, welche von Hand auf ihre Anwendungsfälle spezialisiert wurden, schnell an ihre Grenzen (LeCun et al. 2015). Bei diesen wird sogenanntes „Representation Learning“ angewendet, also das Lernen, was genau die einzelnen Daten repräsentieren, was dem Programm dabei hilft auch komplexere Datenstrukturen zu verstehen. Eine Verbesserung dieser konventionellen Prinzipien ist die Kombination mit neuronalen Netzen, sogenannte Deep Learning Methoden, um das Representation Learning weiter zu verbessern (Bengio et al. 2013). Deep Learning beschreibt in diesem Kontext, dass diese Funktionen, welche die Repräsentationen erarbeiten, in Layer aufgeteilt sind. Dabei wird eine erste Repräsentation aus dem rohen Input erstellt, aus welcher eine neue, abstraktere Repräsentation abgeleitet wird. Diese wird dann an den nächsten Layer gegeben und weiter abstrahiert. Nach genug Transformationen ist es dem Modell möglich, mithilfe der Abstraktion komplexe Funktionen zu erstellen, welche mit den Daten besser umgehen können. Der Turing-Award Gewinner Yann LeCun beschreibt das Konzept an einem Beispiel der Bildklassifikation wie folgt: *„... the learned features in the first layer of representation typically represent the presence or absence of edges at particular orientations and locations in the image. The second layer typically detects motifs by spotting particular arrangements of edges, regardless of small variations in the edge positions. The third layer may assemble motifs into larger combinations that correspond to parts of familiar objects, and subsequent*

*layers would detect objects as combinations of these parts.*“ (LeCun et al. 2015, S. 436)

LeCun beschreibt den wichtigsten Punkt des Deep Learning, als das Selbsterlernen dieser Layer durch das Programm selbst. Ein Deep Learning Programm analysiert die gegebenen Daten also selbst und formt basierend auf den Ergebnissen passende Layer (LeCun et al. 2015).

Die neuronalen Netze, die so aufgebaut werden, können in verschiedene Kategorien aufgeteilt werden. Relevant im Kontext der Arbeit ist das sogenannte „Convolutional Neural Network“ (CNN), also ein gefaltetes neuronales Netz.

Die Architektur von CNNs ist von der menschlichen visuellen Wahrnehmung inspiriert (Li et al. 2022b). Die verschiedenen Funktionen und Layer innerhalb des Netzes sind daher speziell auf die Bildverarbeitung ausgelegt, weshalb CNNs in Kombination mit Computer Vision z.B. für Mustererkennung genutzt werden (O'Shea und Nash 2015). In dem Anwendungsfall dieser Arbeit wird ein CNN genutzt, da es sich beim Input, den die KI erhält, um ein Bild des momentanen Spielzustands handelt.

Eine KI besteht nicht nur aus dem CNN – ebenfalls relevant ist das Modell, welches den Input verarbeitet und aktiv lernt diesen zu interpretieren. Dafür wird in dieser Arbeit ein Deep-Q-Network (DQN) genutzt. Es wurde sich explizit für ein DQN entschieden, da sich dieses im Vergleich zu anderen Modellen im Kontext von Videospielen durchgesetzt hat (Mnih et al. 2015, S. 530–532).

Der zentrale Bestandteil eines DQN ist die sogenannte Q-Funktion. Dies ist die Funktion, die den erwarteten Reward bestimmt, wenn in einem beliebigen Zustand eine beliebige Aktion ausgeführt wird. Dabei wird diese Funktion während des Lernprozesses an die Geschehnisse angepasst und lernt so die Zusammenhänge von Zuständen, Aktionen und Rewards. Endresultat ist eine optimale Version dieser Q-Funktion, welche zu jedem Zustand die Aktion bestimmen kann, die den höchsten Reward erzeugt (Roderick et al. 2017, S. 1–2).

Die Besonderheit bei einem DQN im Kontext der Q-Funktion ist, dass Deep Q-Learning angewendet wird. Das heißt, dass das Lernen der optimalen Q-Funktion durch eine CNN-Struktur erweitert wird. Das CNN wird hier genutzt um die Q-Funktion an Hand der letzten vier Frames, welche der Agent als Input bekommen hat,

genauer anpassen zu können. Hier wird ein sogenanntes „Experience Replay“ genutzt. Das Modell sammelt Erfahrungen über eine kurze Anzahl von Steps, wählt dann zufällig eine dieser Erfahrungen aus, und passt die Q-Funktion dann basierend darauf an. Dieses zufällige Auswählen wird verwendet, um die Erfahrungen extern vom Environment zu betrachten und so eine Voreingenommenheit zu vermeiden, welche die Q-Funktion negativ beeinflussen könnte. Dieser Prozess beschreibt die Exploitation Policy des DQNs. (Roderick et al. 2017, S. 2–3).

Die Exploration Policy eines DQN wird durch eine Epsilon-Greedy-Strategie gesteuert. Diese beschreibt, wie der Agent die Aktion auswählt, welche er an das Environment überträgt. Im Falle der Epsilon-Greedy-Strategie wählt das Modell die Aktion basierend auf Wahrscheinlichkeit aus. Im Folgenden beschreibt  $k$  die Anzahl der möglichen Aktionen, zwischen denen der Agent wählen kann, und  $\varepsilon$  einen vordefinierten Wert zwischen 0 und 1, die sogenannte Exploration-Rate. Die Aktion, welche laut der Q-Funktion im momentanen Zustand den höchsten Reward bringt, wird mit der Wahrscheinlichkeit  $(1 - \frac{\varepsilon(k-1)}{k})$  ausgewählt, während die übrigen Aktionen jeweils eine Wahrscheinlichkeit von  $\frac{\varepsilon}{k}$  besitzen. Konkret heißt das, dass die Exploration-Rate steuert, mit welcher Wahrscheinlichkeit das Modell etwas Neues probiert, oder es auf das bereits gelernte zurückgreift. Geht die Exploration-Rate  $\varepsilon$  Richtung 0, wählt der Agent hauptsächlich Aktionen, welche auf bisherigen Erfahrungen basieren, während bei einem Wert nahe 1 der Agent zufällige Aktionen wählt (Wunder et al. 2010, S. 1167–1168).

Für die Umsetzung eines Reinforcement Learning Ansatzes im Spiel StepMania wird ein DQN verwendet. Dieses ist ein Off-Policy Algorithmus, da zwei unterschiedliche Policies verwendet werden – die Exploitation-Policy, umgesetzt durch das Deep Q-Learning, und die Exploration-Policy, realisiert durch die Epsilon-Greedy-Strategie. Da kein Modell des Environment innerhalb des Modells existiert, ist es ein model-free Ansatz, spezieller ein model-free value-based Ansatz, da die Policy selbst nicht direkt optimiert wird, sondern die Q-Funktion.



### 2.3.3. Evolutionary Algorithms

Reinforcement Learning ist nicht der einzige Weg, wie eine KI trainiert werden kann. Eine von vielen Alternativen sind sogenannte Evolutionary Algorithms. Was unterscheidet diese von Reinforcement Learning, und wieso wurden für den praktischen Teil keine Evolutionary Algorithms genutzt?

Diese Algorithmen basieren auf Modellen der natürlichen, biologischen Evolution nach Darwin (Bäck 1996, S. 8). Hier wird jedoch zwischen zwei Perspektiven von Evolution unterschieden. Zum einen gibt es die rein biologischen Ansichtsweise, und zum anderen Evolution im Kontext der Informatik. Der biologische Ansatz der Evolution bietet dabei zwar die Basis für die Evolution in der Informatik, beide Ansätze unterscheiden sich aber dennoch voneinander. In dem Buch „Genetic Programming Theory and Practice XVII“ beschreiben die Autoren Andres N. Sloss und Steven Gustafson den Begriff „(digitale) Evolution“ wie folgt: *„Evolution is a dynamic mechanism that includes a population of entities (potential solutions) where some form of replication, variation and selection occurs on those entities [...]. This is a stochastic but guided process where the desire is to move towards a fixed goal.“* (Sloss und Gustafson 2020, S. 313).

Es gibt also eine Population von Individuen, welche sich durch Generationen hinweg verändern. Die Individuen werden dabei mit einem Fitness-Value bewertet, wobei die, die einen höheren Fitness-Value besitzen, bevorzugt für die Weiterentwicklung der Population verwendet werden. Zuerst werden anhand dieses Fitness-Values die Individuen ausgewählt, welche die nächste Generation bilden sollen. Anschließend werden diese Individuen „vermehrt“, wobei jedes Individuum Variationen erhält, welche es von anderen der selben Generation unterscheidet. Diese Variationen werden durch Selektion oder stochastische Mutationen bestimmt. Daraufhin beginnt der Prozess erneut und eine neue Generation wird erstellt. Dies stellt das Grundkonzept von Evolutionsalgorithmen dar, wobei es unterschiedliche Varianten zu diesem Konzept gibt, die auf diesem aufbauen (Sloss und Gustafson 2020, S. 314).

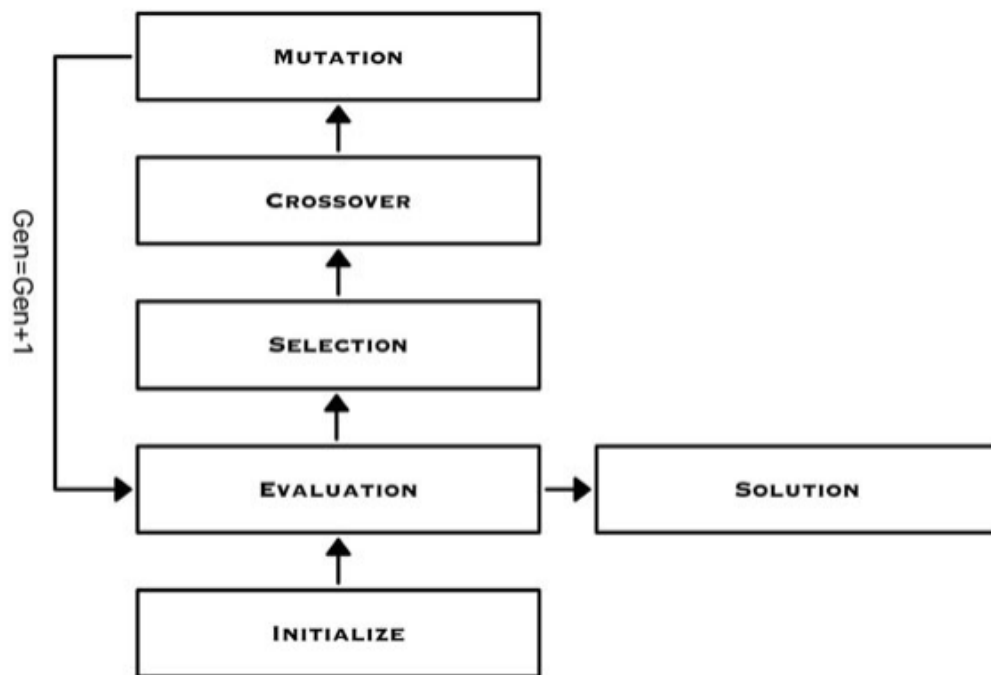


Abb. 4 - Prozess der Evolution (Sloss und Gustafson 2020, S.314, Fig. 16.3)

Dieses Verfahren kommt zu einem Schluss, wenn eine vorher definierte Anzahl von Generationen erreicht wurde. Die finale Generation ist dann die Lösung für das Problem – im Kontext der KI dann ein soweit optimiertes Modell für die vorliegende Umgebung (Nettleton 1994, S. 19).

Der wohl größte Vorteil bei der Nutzung eines Evolutionary Algorithmus ist die große Anzahl der möglichen Lösungen, welche gleichzeitig ausprobiert werden. Anders als beispielsweise beim Reinforcement Learning, nur ein Modell in der Umgebung zu trainieren, werden verschiedene Ansätze ausprobiert, von welchen die jeweils besten weiter verfolgt und weiter verbessert werden, um sich so einem Optimum zu nähern (Nettleton 1994, S. 19).

Evolutionary Algorithms sind für Spiele besonders interessant, da sie genutzt werden können, um interessantere und für die menschlichen Spieler spaßigere Gegner darzustellen (Lucas und Kendall 2006, S. 11). Ein Beispiel für die Entwicklung einer KI mit einem Evolutionary Algorithm ist „Blondie24“ – eine KI, welche ohne Erklärung das Brettspiel Dame gelernt und mit diesem Wissen Turniere gewonnen hat (Fogel 2001; Nettleton 1994). Hier wurde der KI nicht erklärt, wie das Spiel genau funktioniert – es wurden bloß Informationen über die Anzahl, die Position und die Farbe der Steine auf dem Spielbrett an die KI übermittelt. Informationen wie

Strategien oder eine Datenbank mit möglichen Zügen wurde nicht hinterlegt (Lucas und Kendall 2006, S. 12–13). Dame fällt unter die Spiele mit perfekter Information, da jeder Spieler zu jeder Zeit alle Informationen über den Stand des Spiels besitzt. Als Kontrast dazu existieren Spiele mit imperfekter Information, bei welchen der Wissensstand der Spieler differenziert – ein klassisches Beispiel dafür sind Kartenspiele wie beispielsweise Poker (Lucas und Kendall 2006, S. 11–13). Poker insbesondere ist für KI ein interessanter Anwendungsfall, wie Darse Billings et al. der Universität Alberta in dem Paper „The challenge of poker“ beschreiben: *„It is a game of imperfect information, where multiple competing agents must deal with probabilistic knowledge, risk assessment, and possible deception, not unlike decisions made in the real world.“* (Billings et al. 2002, S. 201).

Ähnlich, aber in komplexerer Form, sind Videospiele aufgebaut. Zustände in Videospielen können viel umfangreicher sein als die von Brett- oder Kartenspielen. Kombiniert mit der Komponente, in Echtzeit auf die Geschehnisse reagieren zu können, stellen sie damit sowohl eine Herausforderung für menschlichen Spieler, als auch für die Entwicklung einer KI dar (Lucas und Kendall 2006, S. 14–15).

Warum wurde für den praktischen Teil dieser Arbeit nun Reinforcement Learning genutzt und kein Verfahren, welches auf Evolution basiert? Das größte Problem stellt die Zeit dar, die benötigt wird, um Generationen fortzuschreiten. Für eine bessere Darstellung des Problems kann man hier das Beispiel eines Schachcomputers aufführen. Soll es pro Generation 100 Individuen geben, welche alle einmal gegeneinander spielen sollen, kommt man bei nur 10 Sekunden pro Spiel auf 825 Minuten für eine Generation. Bei 100 Generationen würde dies über 57 Tage dauern (David et al. 2014, S. 782). Man sieht, dass es bei Schach schon bei wenigen Generationen zu langen Lernzeiten kommt. Nun ist StepMania kein Schach, man muss nicht gegen andere spielen, aber das Problem der Zeit pro Spiel bleibt bestehen – ein Individuum sollte mindestens einen „Chart“ spielen, was, abgerundet und wohlwollend geschätzt, eine Minute dauert. Realistischer wäre aber eher eine Zeit von knapp 3 Minuten pro Chart. Bei 100 Individuen pro Generation sind das 100 Minuten an Spielzeit, da bei den verwendeten technischen Umständen nur eine Instanz des Spiels gleichzeitig verwendet werden kann. Rechnet man dies hoch auf 100 Generationen, entspricht dies etwa einer Woche an durchgehendem Training. Sollten unterschiedliche Trainingsdaten mit verschiedenen Charts verwendet werden, wird

sich die benötigte Zeit weiter vergrößern. Dies steht im Kontrast zum Lernablauf des Reinforcement Learning. Hier werden Charts nacheinander gespielt und nach jedem Chart das Modell angepasst. Das heißt, dass bei dem Modell bereits nach einem Chart Lernfortschritt zu erkennen ist. Da der Lernfortschritt beim evolutionären Ansatz erst nach 100 Generationen, also 100 Charts auftritt, müsste der Fortschritt mindestens dem entsprechen, den ein Reinforcement Learning Ansatz nach 100 Charts erreicht hätte. Da der Lernprozess bei dem evolutionären Ansatz erst sehr spät auftritt, und bis dahin der Erfolg auf reinem Ausprobieren aufbaut, ist dies jedoch auszuschließen. Dennoch wäre ein direkter Vergleich der beiden Methoden im Beispiel StepMania ein interessantes Konzept, welches weiter verfolgt werden könnte.

### 3. KIs im Bereich Unterhaltung

#### 3.1. Künstliche Intelligenz in Videospielen

Computergesteuerte Mit- oder Gegenspieler sind im Zusammenhang mit Videospielen nichts Neues. Bereits in den 1950ern gab es erste Konzepte von künstlicher Intelligenz in Videospielen, auch wenn diese wenig mit der in Kapitel 2.1 festgelegten Definition von KI zu tun haben. 1952 berichtete die Zeitung *The New Yorker* über eine elektronische Maschine, welche das Spiel „Nim“ spielen kann (Grant und Lardner 1952). Bei diesem klassischen Gesellschaftsspiel liegen vor den beiden Spielern mehrere Reihen an beliebigen Gegenständen. Die Spieler nehmen nun abwechselnd eine frei wählbare Anzahl an Gegenständen aus einer ausgewählten Reihe. Gewonnen hat die Person, die den letzten Gegenstand nimmt (Bouton 1901, S. 35). Hierbei handelt es sich also um ein Spiel mit perfekter Information, da alle Spieler alle Informationen über das Spiel besitzen. Die Maschine wurde 1951 von der britischen Firma Ferranti für die Nationalausstellung „Festival of Britain“ gebaut, und „Nimrod“ getauft wurde. Sie besaß vier Reihen an Lichtern mit dazugehörigen Knöpfen, welche die Reihen der Gegenstände repräsentierten, und ermöglichte Interessierten, so eine Partie „Nim“ gegen den Computer zu spielen (Baker 2010; Grant und Lardner 1952).

Der Übergang vom klassischen Gesellschaftsspiel zu Arcade-Klassiker wie „Pac-Man“ passierte um 1980, wo nun künstlich gesteuerte Gegenspieler dem Menschen entgegentraten, und so den Anfang einer der wichtigsten Komponente für modernen Videospiele darstellte (Quadir und Khder 2022, S. 465–466). Über die Jahre hinweg verbesserte sich die Qualität der künstlichen Intelligenz von virtuellen Gegnern und Mitspielern, und wurde zu einem umfangreichen Feld innerhalb der Spieleentwicklung. Beispiele wie das Alien in „Alien: Isolation“ zeigen, wie weit diese Technologien bereits fortgeschritten sind: Ein Monster in einem Horrorspiel, welches komplexe Verhaltensmuster aufweist und sich selbst erschließen muss, wo sich der Spieler befindet. Diese Art der intelligenten Gegner erzeugt Spannung und trägt stark zur Immersion in das Spiel bei (Thompson 2017).

Während KIs wie in „Alien: Isolation“ explizit in das Spiel programmiert sind, übernehmen andere, wie im Beispiel „Nim“, die Rolle des Spielers selbst. Interessant für den Anwendungskontext dieser Arbeit ist hier, welche Informationen die KI

bekommt, und auf welche Weise sie diese erhält. Im Beispiel „Alien: Isolation“ bekommt die KI ihre Informationen durch Geschehnisse im Spiel. Macht der Spieler beispielsweise Geräusche, während sich das Modell des Aliens in der Nähe befindet, oder läuft der Spieler am Alien vorbei, kann die KI darauf reagieren (Thompson 2017). Die Informationen, welche die KI hier bekommt, können auch Dinge umfassen, von denen der Spieler nichts weiß. Dies ist in diesem Beispiel kein großes Problem, da das Spielprinzip darauf basiert, dass der Spieler vor dem Monster flüchtet, und es als intelligent und unberechenbar wahrgenommen werden soll, jedoch kann dieses Konzept in anderen Kontexten als unfair wahrgenommen werden (Švelch 2020). KI, welche intern implementiert wird und als Ersatz für menschliche Spieler agieren soll, hat die Möglichkeit unter dem Vorwand von Schwierigkeitsstufen oder Ähnlichem die Regeln des Spiels zu umgehen. So kann es sein, dass die KI in einem Strategiespiel für ihre Einheiten nur halb so viele Ressourcen sammeln muss wie ein menschlicher Spieler, oder sie in einem Rennspiel simplifizierte Physik erfährt. Dies gibt der KI einen unfairen Vorteil und kann, wenn es zu extrem angewandt wird und es sehr offensichtlich ist, den Spielspaß ruinieren – der Spieler soll immerhin denken, dass die KI mit den gleichen Voraussetzungen wie er selbst arbeiten muss (Cass 2002, S. 42; Hladky und Bulitko 2008, S. 39).

Eine andere Art der KI sind solche, die von Grund auf in der gleichen Situation sind wie der Spieler. Das bedeutet, dass sie die gleichen Voraussetzungen haben (gleiche Kosten für Einheiten, identische Physik, ...), aber auch ausschließlich dieselbe Art von Informationen bekommen wie der Spieler. Möglich ist dies beispielsweise durch das Aufsetzen einer regulären Spielumgebung, um den Input auf dem Bildschirm als Input für die künstliche Intelligenz zu nutzen, welche dann Tastatur- oder Controllerbefehle als Output generiert. Ein Beispiel für diese Art ist die Minecraft-KI von OpenAI. Hier wurde für das Open World Sandbox-Spiel Minecraft ein Modell trainiert, welches fortgeschrittene, logische Sequenzen ausführen kann, für welche menschliche Spieler teilweise rund 20 Minuten benötigen. Dabei ist wichtig zu erwähnen, dass Umgebungen in Minecraft zufällig generiert werden, und die KI sich in diesen zufällig generierten Umgebungen selbstständig orientieren und zurechtfinden muss (Baker et al. 2022, S. 2). Bei diesem spezifischen Beispiel wurde sogenanntes „Video Pre-Training“ angewandt – eine Technik, bei welcher eine KI auf vorher aufgenommenem Videomaterial trainiert wird – als auch Reinforcement Learning, um das Modell

präziser abzustimmen. Natürlich ist dieses Beispiel sehr extrem: Hier wurden insgesamt 70.000 Stunden an Videomaterial genutzt, um das Modell zu trainieren (Baker et al. 2022, S. 2–8). Ähnliche Experimente in kleinerer Form wurden bei Google DeepMind 2013 an Atari 2600 Spielen durchgeführt. Hier wurde ein CNN und ein DQN in Kombination mit Reinforcement Learning genutzt, um Modelle nur an Hand von Videodaten zu trainieren (Mnih et al. 2013, S. 1–6). Im praktischen Teil der Arbeit wurde das selbe Konzept, also Reinforcement Learning in Kombination mit einem DQN und einem CNN zu nutzen, als Grundlage für die Entwicklung genutzt, da auch hier rein mit den Informationen gearbeitet wird, welche der Spieler vom Spiel bekommt.

Eine erwähnenswerte moralische Komponente bezogen auf das Lernen von Spielen durch KI ist die des „Cheatings“. Cheating beschreibt im Kontext von Videospielen das Eingreifen in das Spiel durch externe Tools um sich selbst, meist gegenüber anderen Spielern, einen unfairen Vorteil zu verschaffen. Dies ist nicht unbedingt auf KI beschränkt, sondern kann auch ohne trainierte Modelle erreicht werden. Dennoch sind diese Cheats ein konstantes Problem für Spielentwickler, hauptsächlich in Multiplayer-Spielen, da so auch der Spielspaß von anderen gefährdet ist. David Durst und Carly Taylor schrieben diesbezüglich ein Paper für den Spieleentwickler Activision, in welchem sie das Problem klar definieren: *„The competition between anti-cheat and cheat developers is an asymmetric, repeated cat-and-mouse cycle. Anti-cheat developers detect new behaviors indicative of cheating. Cheat developers respond by generating different behavior. Anti-cheat developers detect the new behaviors, and the cycle continues.“* (Durst und Taylor 2023). Künstliche Intelligenz befeuert diesen Kreislauf in beide Richtungen – es wäre beispielsweise denkbar, mithilfe von trainierten Modellen die einprogrammierten Verhaltensweisen der Cheat-Entwickler, aber auch der Anti-Cheat-Entwickler zu erkennen und zu umgehen oder sie einzuschränken. Ein Modell zu erstellen, welches einen Spieler imitiert, und so ein Spiel spielt, wäre nach der genannten Definition auch „cheating“, da ein Programm schneller auf Dinge reagieren kann als ein Mensch und so dem Nutzer einen Vorteil verschafft. Dies ist eine Angelegenheit, welche für jedes Spiel unterschiedlich schwer umzusetzen ist. Je mehr Informationen ein Spiel hat, und je mehr mögliche Aktionen man ausführen kann, desto schwerer die Entwicklung eines solchen Cheats. Die Umsetzung für StepMania ist dabei relativ einfach, da Input und Output eines Modells

nicht sehr komplex sind – ein Screenshot des Spielfeldes als Input und 4 mögliche Tastendrücke als Output.

Man könnte also das Argument aufzeigen, dass das fertige Ergebnis des praktischen Teils dieser Arbeit ein Cheat ist. Da es sich hier aber um ein Singleplayer-Spiel handelt, und keine anderen Spieler davon beeinträchtigt werden, ist der Begriff „Cheating“ hier Auslegungssache.

### 3.2. Künstliche Intelligenz in Unterhaltungsmedien

Im vorherigen Kapitel wurde sich mit dem Thema künstlicher Intelligenz in Videospielen beschäftigt, doch dies ist nur ein kleiner Teil von dem tatsächlichen Anwendungsbereich von KI. Im Folgenden soll sich mit dem Thema KI in der Produktion von Medien beschäftigt werden. Inspiration für diesen Bereich gab es durch verschiedene Kanäle auf der Livestream-Plattform Twitch – hier sind mehrere Accounts aufgetaucht, bei dessen Livestreams kein Mensch vor der Kamera sitzt, sondern eine KI, welche Videospiele spielt und mit den Zuschauern interagiert.

Eine der größten dieser Accounts heißt „vedal987“, auf welcher eine KI namens „Neuro-sama“ bei jedem Stream rund 4.000 Zuschauer unterhält (TwitchMetrics 2023; D'Anastasio 2023). Für die Interaktion mit dem Livestream-Chat wird, ähnlich wie bei ChatGPT, ein Large Language Model genutzt, welches die Nachrichten der Zuschauer nimmt, und daraus eine Antwort generiert. Diese Antwort wird dann in ein Text-to-Speech-Programm geleitet, welches die Wörter vertont (D'Anastasio 2023; vedal.xyz 2023). Diese Vertonung des LLM-Outputs wird ebenfalls genutzt, um auf Videos und andere Geschehnisse während des Streams zu reagieren (Tsiaoussidis 2023). Neben diesen Interaktionen spielt „Neuro-sama“ auch Videospiele, wobei nicht öffentlich bekannt ist, welche Technologien dabei verwendet werden. Die KI für das Spiel „osu!“ wurde dabei vor über 4 Jahren entwickelt, und seitdem so weit trainiert, dass das Modell nun besser als jeder menschliche Spieler ist (Neuro-sama 2019; Garcia 2023).

Während es sich bei „Neuro-sama“ um einen Livestream handelt, bei dem die KI mit anderen, realen Menschen interagiert, gibt es auch KI-Livestreams, welche klassischen



Fernsehsendungen nachempfunden sind, bei welchen nicht auf die Zuschauer eingegangen wird. Einer dieser Livestreams läuft auf dem Twitch-Kanal „watchmeforever“ und überträgt eine nichtendende Folge einer Sitcom namens „Nothing, Forever“, welcher der amerikanischen Serie „Seinfeld“ nachempfunden ist. Das Projekt wurde von der Gruppe Mismatch Media ins Leben gerufen, welche sich darauf spezialisiert hat, die sogenannte „KI-betriebene Generation von Medien“ für die breite Masse verfügbar zu machen (Diaz 2023; Mismatch Media 2023). Im Livestream sieht man wiederkehrende Orte und Charaktere, welche miteinander interagieren und in Dialog treten. Die Optik der Sitcom ist ein 3D-Pixel-Stil und auch die Stimmen klingen durch den Einsatz einer Text-to-Speech-Software sehr robotisch, was an Videospiele aus den Neunzigern erinnert (Diaz 2023). Laut einem Interview mit Skyler Hartle, einem Mitschöpfer von Mismatch Media, wird der Inhalt der Serie durch OpenAI's GPT-3 erstellt, und dann durch Microsofts Azure Cognitive Services vertont – durch die einfach zugängliche Art der KI-Tools sei es für das Team möglich, problemlos neue Shows oder Formate zu erstellen (Diaz 2023; Winslow 2023). Hartle betont dabei, dass das Ziel des Projektes und auch der Plattform darin liegt, es Personen mit limitierten Ressourcen möglich zu machen, ihre kreativen Vorstellungen umzusetzen – klassisches Fernsehen zu ersetzen sei nicht der Sinn (Winslow 2023).

Ein weiteres Beispiel dieser KI-generierten Sendungen, welche durch „Nothing, Forever“ inspiriert wurde, „ist „ai\_sponge“ – ein Twitch-Kanal, welcher ab dem 05.03.2023 für etwa einen Monat eine künstlich generierte, nicht-endende Episode der Kinderserie „Spongebob Schwammkopf“ livestreamte. Auch hier wurden Dialoge zwischen den Charakteren durch KI generiert. Die Charaktere selbst wurden allerdings nicht abgeändert, sondern direkt aus der Originalserie übernommen und mit 3D-Modellen in ihrer aus der Serie bekannten Umgebung visualisiert. Die Themen, welche in dem Livestream in den Konversationen aufkamen, unterschieden sich stark zu den gewohnten Themen der Originalsendung. Während dies einerseits ein großer Punkt in der Beliebtheit dieses Kanals war, wird andererseits auch vermutet, dass dies der Grund war, weswegen der Livestream Ende März von der Plattform gesperrt wurde (Eudaly 2023).

Beschwerden über KI, welche in der Erstellung von Medien involviert wird, werden dabei immer größer. Die Writers Guild of America (WGA) streikte Anfang Mai 2023 für fairere Budgets bei Fernsehserien und Filmen auf Streaming-Plattformen, und

argumentierte auch für eine Regelung, welche unter anderem den Einsatz von künstlicher Intelligenz für die Generierung von Quellmaterial verbietet (Writers Guild of America 2023). In einem Interview mit der Webseite „Polygon“ erklärt der Autor und Aktivist Cory Doctorow wo er das Problem mit KI in der Filmbranche sieht: *„If it becomes the case that the way you make a new Pixar movie is by typing some prompts into a keyboard, and then going down the street to the smokehouse for a three-martini lunch and then coming back, and it’s there on your hard drive [...] then all of a sudden people are like, ‘I can copy that Pixar movie and sell it to anyone I want.’ Because it was made by a bot.“* (Broderick 2023)

Problematisch wird das Konzept von KI in Film und Fernsehen auch im Bereich der Legalität. Das U.S. Copyright Office entschied am 21. Februar 2023, dass Bilder in dem Buch „Zarya of the Dawn“ der Autorin Kristina Kashtanova nicht unter das Copyrightgesetz fallen, da sie mit der Software midjourney erstellt wurden, und so nicht als Produkt von menschlicher Urheberschaft bezeichnet werden können (Brittain 2023; United States Copyright Office 2023). Dies stellt einer der ersten offiziellen Entscheidungen in Bezug auf Copyright für KI-generierte Werke dar (Brittain 2023). Inwiefern sich dies auf KI-generierte Sendungen, Streamer oder KI-gesteuerte Spiele auswirkt, ist jedoch noch unklar.

## 4. Entwicklung

### 4.1. Zielsetzung und Anforderungen

Ziel des Projektes ist es, eine KI zu entwickeln und zu trainieren, die es schafft, StepMania Charts zu spielen. Wichtig zu erwähnen ist dabei, dass es nicht das Ziel ist, eine „perfekte“ KI zu trainieren, die jeden beliebigen Chart mit einer einhundertprozentigen Genauigkeit spielen kann, da dafür der Trainingsprozess zu lange dauern würde. Vielmehr geht es um den Prozess der Entwicklung einer solchen KI, welche Probleme dabei auftreten, wie sie gelöst werden können und wie sich ein Modell nach kurzen Trainingseinheiten verhält. Das Ziel ist es, einen generellen Lernerfolg zu erreichen – nicht ein perfektes Modell zu erstellen.

Während der Entwicklung der KI und deren Bestandteile wurden alle Schritte und Ansätze dokumentiert. Dies ist in den folgenden Kapiteln dargestellt. Dabei sind auch Ideen und Konzepte aufgeführt, welche es schlussendlich nicht in die finale Version geschafft haben. Warum sich anfangs für diese Ansätze entschieden wurde und weshalb sie später nicht weiterverfolgt wurden, wird im Laufe der Dokumentation erklärt.

Der komplette Code ist einsehbar unter

<https://github.com/alexanderreiprich/StepMania-5-AI>.

### 4.2. Umsetzung der verschiedenen Bestandteile

#### 4.2.1. Aufsetzen eines Environments

Wie in Kapitel 2.3 bereits behandelt, bietet das Environment die Umgebung für den Agent - wie er in jedem Step agieren soll, was er an Feedback bekommt, etc. In diesem Kapitel wird der Aufbau und die Entwicklung des Environments für den Anwendungsfall StepMania 5 beschrieben werden. Als Programmiersprache wurde sich für Python 3.9.7 entschieden. Für die Umsetzung des Environments wurde das Python-Modul „gym“ in der Version 0.21.0 genutzt, welches von OpenAI entwickelt

worden ist. Die Umsetzung des Modells, auf welches später eingegangen wird, wurde mit dem Python-Modul „Stable Baselines 3“ in der Version 1.8.0 entwickelt.

Ein Environment besteht typischerweise aus mehreren Methoden.

- `init()` - Die Initialisierung des Environments.
- `step()` - Die Methode, welche wiederholt aufgerufen wird. Hier wird die Aktion des vorherigen Steps ausgeführt, eine neue Observation aufgenommen, der Reward des Steps bestimmt und geprüft, ob die derzeitige Episode vorbei ist.
- `reset()` - Die Methode, welche genutzt wird, um das Environment auf den Anfangszustand zurückzusetzen.
- `close()` - Die Methode, die das Environment schließt und beanspruchte Ressourcen wieder freigibt.
- `render()` - Die Methode, welche das Geschehen visualisiert.

Bei dem Anwendungsfall StepMania ist sowohl `render()` als auch `close()` nicht von Bedeutung, da wir weder die Aktionen visualisieren müssen noch Ressourcen verwenden, welche freigegeben werden müssen.

Um die `step()`-Methode übersichtlicher zu gestalten, wurden noch drei weitere Methoden in die Environment-Klasse hinzugefügt:

- `get_observation()` - In dieser Methode wird der Screenshot, welcher in jedem Step gemacht wird, so bearbeitet, dass er nur wichtige Informationen für die Erkennung des Gameplays beinhaltet.
- `get_reward()` - Wie der Name bereits sagt, wird in dieser Methode der aktuelle Reward berechnet.
- `get_over()` - Diese Methode wird ausgeführt, um festzustellen, ob die Episode vorbei ist.

Der Output der `step()`-Methode besteht aus vier Dingen - der Observation, dem Reward, einem Boolean, welcher aussagt, ob die Episode vorüber ist, und einem Info-Objekt, welches für die Nutzung des Environments selbst nicht relevant ist. Die drei essenziellen Rückgabewerte eines Steps werden also in den eigenen, extra hinzugefügten Methoden berechnet und zurückgegeben. Dies ermöglicht nun das einfache Anpassen der einzelnen Bestandteile, sowie das explizite Aufrufen dieser Methoden außerhalb des Environments für Testzwecke.

Wenn das Environment erstellt wird, wird die `init()`-Methode ausgeführt. Hier soll sich nur um die Initialisierung des Environments und deren Bestandteile gekümmert werden. Die Ausnahmen diesbezüglich sind das Skalieren und Verschieben des Spielfensters, was nur indirekt mit dem Environment selbst zu tun hat. Grund dafür ist, dass die Position des Fensters ausschlaggebend für die Screenshots bzw. das Zuschneiden dieser ist. Wichtiger sind hier die Attribute, in welchen sich Instanzen der vorher erstellten Klassen für Screenshots, Pattern Recognition und Input Sending befinden. Sie werden als Klassenattribute gespeichert, um später auf die Methoden dieser Klassen einfacher zugreifen zu können, da man für das Aufrufen der Methoden sonst immer wieder eine Instanz der Klasse erzeugen müsste.

Als Klassenattribute werden auch die Positionen der benötigten Bereiche, also dem Gameplaybereich, dem Scorebereich, etc., abgespeichert.

Ebenfalls essenziell sind die Attribute, welche den Action Space und den Observation Space beinhalten. Der Action Space sagt aus, was der Agent für Inputmöglichkeiten hat. In diesem Fall enthält der Action Space eine von 5 diskreten Zahlen – 0 bis 4. Während die 0 dafür genutzt wird, dem Environment mitzuteilen, dass der momentane Status der Tasten beibehalten werden soll, stehen die anderen Zahlen jeweils für eine der vier Tasten, welche der Spieler drücken kann. Wird ein Input verarbeitet, errechnet das Modell, ob keine Änderung am status quo durchgeführt werden soll, oder ob der Zustand eine der vier Tastendrücken gewechselt werden soll. Hier wird bereits ein Problem deutlich – es ist dem Modell nicht möglich, zwei Tasten gleichzeitig zu drücken, was von dem Spiel durchaus verlangt werden kann. Um dieses Problem zu umgehen, wurden mehrere Alternativen ausprobiert. Beispiele dafür beinhalteten unter anderem den Ansatz, ein Array als Action Space zu nutzen, welches die Information für jede Pfeilrichtung übermittelt, was jedoch nicht kompatibel mit dem DQN-Modell ist. Ein anderes Konzept beschrieb das Definieren von 16 diskrete Zahlen als Action Space, was jede mögliche Kombination an Tastendrücken abgedeckt hätte. Problematisch war bei diesem Ansatz die Performance, da das Modell zu viele mögliche Aktionen hatte, und das Lernen so um ein vielfaches länger gedauert hätte. Da die Performance für den Lernerfolg sehr essenziell war, und es in diesem Projekt sehr wichtig war, schnell beurteilen zu können, ob das Modell „richtig“ lernt, wurde sich dafür entschieden, bei nur einer Aktion pro Step zu bleiben.

Im Observation Space wird definiert, was der Agent als Input, also Observation, erhält. Für dieses Environment werden, wie bereits angesprochen, Screenshots von der Gameplay-Region des Spiels gemacht, welche der Agent als Observation nutzen soll. Dafür wird als Observation Space eine Box definiert - ein Raum, in welchem sich mehrere Zahlen zwischen bestimmten Wertintervallen befinden. Hier wird festgelegt, dass sich alle Werte zwischen 0 und 255 befinden, da wir in unserem Fall von der Helligkeit von Pixeln reden, sowie einer Shape von (1, 135, 100). Die Shape steht für die Dimensionen des Bildes, welches wir als Observation nutzen - eine Breite und Höhe von 100 mal 135 Pixeln in nur einem Farbkanal, da die Screenshots in Schwarz-Weiß konvertiert werden, um Daten zu sparen.

```

def __init__(self):
    super().__init__()
    # Define action space
    self.action_space = Discrete(5)

    # Observation Array
    self.observation_space = Box(
        low=0,
        high=255,
        shape=(1, 135, 100),
        dtype=np.uint8
    )

    # Define extraction parameters for the game
    self.screenshot_helper = Screenshot()
    self.input_sending_helper = SendInput()
    self.pattern_recog_helper = RecognizePattern()
    self.capture = mss()
    self.steps = 0
    self.previous_reward = 0
    self.window_location = {'top': 35, 'left': 10, 'width': 410,
'height': 230}
    self.game_location = {'top': 15, 'left': 20, 'width': 100, 'height':
185}
    self.score_location = {'top': 215, 'left': 280, 'width': 100,
'height': 25}
    self.done_location = {'top': 0, 'left': 0, 'width': 90, 'height':25}
    self.past_arrows_location = {'top': 45, 'left': 50, 'width': 140,
'height': 2}
    self.cur_held_buttons = {'a': False, 's': False, 'w': False, 'd':
False}

    self.action_map = {
        0: "no_op",
        1: 'a',
        2: 's',
        3: 'w',
        4: 'd',
    }

    # Adjust window position and size
    win = pygetwindow.getWindowsWithTitle('StepMania')[1]
    win.size = (450, 290)
    win.moveTo(0, 0)

```

*Listing 1 - Code der init-Funktion des Environments*

Nachdem das Environment initialisiert wurde, kann es genutzt werden. Dabei ist der Ablauf einfach erklärt - gestartet wird mit einem Aufruf der Reset-Methode, welche das Environment auf seinen Startzustand setzt. Die Reset-Methode gibt eine Observation des Startzustandes zurück, die dann vom Modell genutzt wird, um eine Aktion zu bestimmen, die es basierend auf der Observation für die Bestmögliche hält. Mit dieser Aktion wird die Step-Methode aufgerufen, welche die Aktion ausführt, und den Reward für diese Aktion, sowie eine neue Observation, an das Modell zurückgibt. Ebenfalls wird mitgegeben, ob die Abbruchbedingung des Environments erfüllt ist. Der Reward wird genutzt, um die Qualität der vergangenen Aktion zu evaluieren und sich selbst dementsprechend anzupassen, während mit der Observation eine neue Aktion bestimmt wird, mit welcher die Step-Methode erneut aufgerufen wird, sollte die Abbruchbedingung nicht erfüllt sein. Dieser Kreislauf wird so lange ausgeführt, bis eine vorher definierte Anzahl an Steps erreicht wurde, oder bis die Abbruchbedingung erfüllt ist.

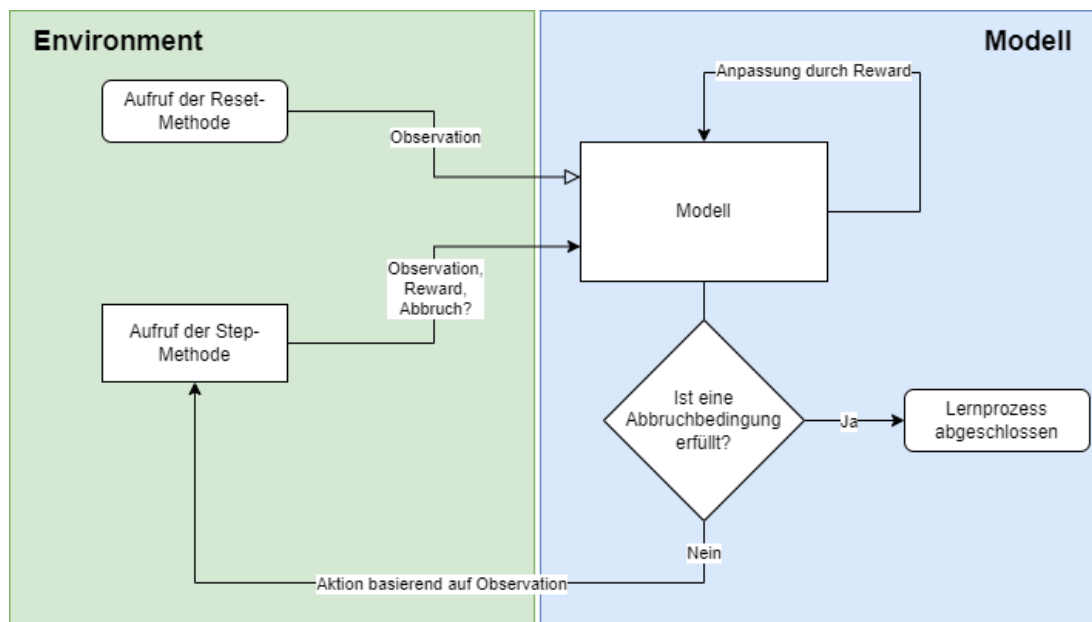


Abb. 5 - Ablaufdiagramm des Lernprozesses, Teile im Environment und Teile im Modell sind farblich unterteilt (Eigene Darstellung)

Die Step-Methode führt zuerst die Aktion aus, welche sie über den Parameter bekommen hat. Daraufhin wird ein Screenshot des gesamten Fensters gemacht, und herunterskaliert, um die Größe des Bildes für die Observation zu reduzieren. Mithilfe von Template Matching wird nun überprüft, ob die Abbruchbedingung erfüllt wurde. Wurden alle drei Methoden aufgerufen, werden ihre Ergebnisse zusammen mit einem Info-Objekt als Output der Step-Methode zurückgegeben.



```

def step(self, action):
    # Manage input based on action parameter
    if action != 0:
        if (list(self.cur_held_buttons.values())[action - 1]):
            self.cur_held_buttons[list(self.cur_held_buttons)[action - 1]] = False
            self.input_sending_helper.releaseKey(self.action_map[action])
        else:
            self.cur_held_buttons[list(self.cur_held_buttons)[action - 1]] = True
            self.input_sending_helper.holdKey(self.action_map[action])

    # Take screenshot for done, observation and reward functions
    screenshot =
np.array(self.capture.grab(self.window_location))[:, :, :1].astype(np.uint8)
    downscaled_screenshot =
self.screenshot_helper.downscaleImageBinary(screenshot, (225, 150), (1, 150,
225))
    self.steps += 1

    # Checking if the game is over
    done = self.get_over(screenshot)

    # Get the next observation
    new_observation = self.get_observation(downscaled_screenshot)

    # Use score as reward
    reward = self.get_reward(screenshot, action)
    info = {}

    return new_observation, reward, done, info

```

*Listing 2 - Code der step-Methode des Environments*

Nach jedem abgeschlossenen Chart wird die Reset-Methode ausgeführt. Diese enthält eine Reihe an Inputs, welche das Spiel vom Ergebnisscreen zuerst zurück in das Hauptmenü und danach in den nachfolgenden Chart navigieren. Abschließend wird ein Screenshot gemacht, welcher an das Modell weitergegeben wird.

```
def reset(self):
    # Exit to menu, select new song and start
    time.sleep(5)
    pydirectinput.press('enter')
    time.sleep(6)
    pydirectinput.press('d')
    time.sleep(2)
    pydirectinput.press('enter')

    # Edge Case - 'Roulette' is selected
    time.sleep(1.5)
    pydirectinput.press('enter')
    time.sleep(3)
    pydirectinput.press('enter')

    # Reset variables
    self.previous_reward = 0
    self.steps = 0
    self.cur_held_buttons = {'a': False, 's': False, 'w': False, 'd': False}

    # Take screenshot to pass to observation
    screenshot = np.array(self.capture.grab(self.window_location))[:, :, :-
1].astype(np.uint8)
    downscaled_screenshot = self.screenshot_helper.downscaleImage(screenshot,
(225, 150), (1, 150, 225))

    return self.get_observation(downscaled_screenshot)
```

*Listing 3 - Code der reset-Methode des Environments*

Die oben angesprochenen Methoden `get_observation()`, `get_done()` und `get_reward()` sind prinzipiell sehr ähnlich. Sie nehmen den Screenshot, welchen sie beim Aufruf der jeweiligen Methode bekommen, und schneiden ihn auf den jeweiligen Teil zu (in der Reward-Methode beispielsweise also auf den Teil, welcher die Scoreanzeige enthält). Die `get_observation()`-Methode gibt nur ihren Teil, der das Spielgeschehen zeigt, zurück, während die `get_done()`- und die `get_reward()`-Methoden unter anderem eine externe Template Matching Funktion nutzen, um ihre jeweiligen Informationen

verarbeiten zu können. Während bei der `get_done`-Methode nur ein Wahrheitswert zurückgegeben wird, muss bei der `Reward`-Methode der Score weiter interpretiert werden, um nützliches Feedback geben zu können. Wenn in den nachfolgenden Kapiteln von der Methode gesprochen wird, welche für die Reward-Verteilung zuständig ist, ist damit diese Methode gemeint. Ihre Funktionsweise hat sich im Laufe des Projektes mit am meisten verändert, worauf später noch eingegangen wird.

```
def get_observation(self, img):
    # Crop gameplay part of the window screenshot
    obs = img[:, self.game_location['top']:(self.game_location['top'] +
self.game_location['height']),
self.game_location['left']:(self.game_location['left'] +
self.game_location['width'])]

    return obs
```

*Listing 4 - Code der `get_observation`-Methode des Environments*

```
def get_over(self, img):
    # Crop done part of the window screenshot
    obs = img[self.done_location['top']:(self.done_location['top'] +
self.done_location['height']),
self.done_location['left']:(self.done_location['left'] +
self.done_location['width'])]

    return self.pattern_recog_helper.analyze_results(obs)
```

*Listing 5 - Code der `get_over`-Methode des Environments*

```

def get_reward(self, img):
    # Crop score part of the window screenshot
    obs = img[self.score_location['top']:(self.score_location['top'] +
self.score_location['height']),
self.score_location['left']:(self.score_location['left'] +
self.score_location['width'])]

    # Calculate the reward by subtracting the previous score from the current
score
    # If the song is over, or the score isn't recognized properly, set reward
to 0 to avoid accidentally returning a negative reward
    new_reward = self.pattern_recog_helper.analyze_score(obs)
    if (new_reward > self.previous_reward):
        reward = new_reward - self.previous_reward
        # Set the current reward as the previous reward for the next iteration
        self.previous_reward = new_reward
    else:
        reward = 0
    return reward

```

*Listing 6 - Code der get\_reward-Methode des Environments*

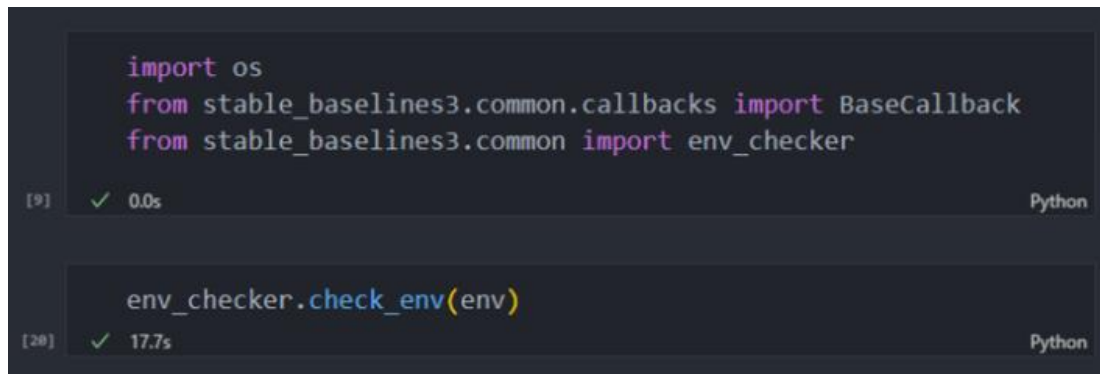
#### 4.2.2. Jupyter Notebook

Da das Programmieren von Environments häufiges Ausführen von bestimmten Codeteilen erfordert, ist eine beliebte Herangehensweise das Nutzen eines sogenannten Jupyter Notebooks. Diese basieren auf Project Jupyter und bieten eine webbasierte Grundlage zum Programmieren von Projekten z.B. in den Bereichen Data Science und Machine Learning. Sie ermöglichen es, Code, Dokumentation und Output in einer Datei zu vereinen (Project Jupyter 2023).

In diesem Fall wurde sich für ein Jupyter Notebook entschieden, da das selektive Ausführen von Code, als auch das direkte Darstellen der aufgenommenen Bereiche den Entwicklungsprozess erleichtert hat. Das Erstellen von Graphen mithilfe von dem Python-Package „matplotlib“ konnte ebenfalls schnell durchgeführt werden, was ein großer Vorteil war.

Jupyter Notebooks sind in sogenannte Code- und Markdown-Cells aufgeteilt, in welchen man entweder Code oder Text im Markdown-Format unterbringen kann. Im folgenden Beispiel kann man zwei Code-Cells sehen - die Erste enthält mehrere

Import-Aufrufe, welche nur einmal ausgeführt werden müssen, um die Module zu importieren. Die Zweite enthält einen Funktionsaufruf, welcher während des Testens des Environments mehrfach aufgerufen werden musste. Da man jede der Code-Cells einzeln ausführen kann und Imports und Variablen über die Laufzeit gespeichert werden, mussten bei jedem Funktionsaufruf nicht alle Import-Aufrufe ausgeführt werden, was Zeit und Rechenkraft spart.



```
import os
from stable_baselines3.common.callbacks import BaseCallback
from stable_baselines3.common import env_checker
```

[9] ✓ 0.0s Python

```
env_checker.check_env(env)
```

[20] ✓ 17.7s Python

Abb. 6 - Zwei Code-Cells eines Jupyter Notebooks innerhalb der Entwicklungsumgebung Visual Studio Code

Während die Entwicklung des Environments durch die Jupyter Notebooks stark vereinfacht wird, gibt es dennoch Grenzen und Nachteile.

Die Entwicklung wurde insofern eingeschränkt, dass bei jeder Änderung von externen Dateien oder Modulen, der Kernel des Notebooks neu gestartet werden muss. Jedes Notebook läuft auf einem Python Kernel, welcher einem die Möglichkeit gibt, Variablen auch über längere Zeit und außerhalb der Laufzeit aufzubewahren. Dies hat allerdings auch zur Folge, dass alte Versionen von externen Modulen noch verwendet werden, wenn der Kernel nicht neu initiiert wird. Dies ist etwas umständlich, aber nicht vermeidbar.

Ebenfalls musste das Environment zum Dokumentieren der Testläufe in einen sogenannten Monitor-Wrapper untergebracht werden. Um dies zu erreichen, musste das Environment als Klasse in einer eigenen Python-Datei in einen Unterordner des Gym-Packages verschoben werden. Dies bedeutet, dass Änderungen am Environment innerhalb der Notebook-Datei extra in diese Python-Datei kopiert werden mussten, damit das Environment im Wrapper aktuell bleibt. Hätte man eine Python-Datei und keine Notebook-Datei genutzt, wäre dieser Schritt einfacher.

#### 4.2.3. Custom Theming

Informationen auf dem Bildschirm sind ein wichtiges Thema. Diese können für den Spieler zwar relevant sein, für die KI sind sie jedoch unwichtig und verhindern ein klares Erkennen der Pfeile. Auf dem folgenden Bild sieht man, welche Elemente auf dem Spielfeld zu finden sind und wie ein einfacher Template Matching Algorithmus damit umgeht.

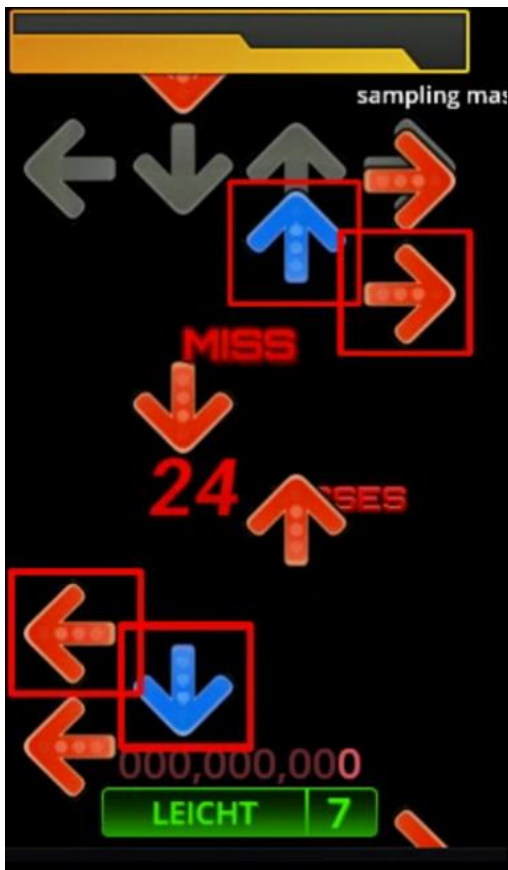


Abb. 7 - Screenshots des Gameplays, bei welchem die durch den Template Matching Algorithmus erkannten Elemente markiert wurden

Wie man sieht, kann das Programm die beiden Pfeile in der Mitte des Bildes nicht als solche erkennen, da sich dort das sogenannte „Judgement“ befindet - ein Feedback für den Spieler, ob und wie gut er die Note getroffen hat. Durch den Text und die Zahl können die Pfeile nicht gut erkannt werden.

Ähnlich sieht es auch mit dem nicht-erkannten Pfeil in der unteren linken Ecke aus. Er wird nicht erkannt, da er sich neben der Score-Anzeige befindet, welche die Punkte des Spielers zählt. Diese Anzeige ist zwar relevant, da sie für das Feedback des

Reinforcement Learning Algorithmus gebraucht wird, jedoch ist sie an dieser Position hinderlich.

Dazu kommen noch weitere Elemente, die Teile des Spielfeldes verdecken - die Anzeige am oberen Rand, welche zeigt, wie viele Lebenspunkte der Spieler noch hat, oder die Schwierigkeitsanzeige des Levels am unteren Rand.

Um diese Probleme zu umgehen, wurde ein sogenanntes Custom Theme erstellt. StepMania 5 unterstützt spieler-erstellte Veränderungen der Benutzeroberfläche. Verändern kann man diese durch das Abändern von .ini und .lua-Dateien im Spielverzeichnis. In diesem Anwendungsfall haben Änderungen in ein paar Zeilen der metrics.ini-Datei, also die Datei, die für den Großteil von Themeänderungen zuständig ist, sowie eine Änderung von zwei Variablen in einer .lua-Datei den Bereich übersichtlicher gemacht.

In der metrics.ini-Datei wurde die Score-Anzeige um einen gewissen Pixelwert nach rechts verschoben, sodass sie sichtbar bleibt, aber nicht mehr im Bereich des Gameplays liegt. Ebenfalls wurden dort alle möglichen Overlay-Elemente deaktiviert, welche durch Änderungen dieser .ini-Datei aus- oder angeschaltet werden konnten.

Da ein generelles Ausblenden des Judgements in der für den Gameplaybereich zuständigen Other.lua-Datei nicht möglich ist, wurde die Anzeige um einen hohen Wert außerhalb des dargestellten Bereichs geschoben.

```
[ScreenGameplay]
ScoreP1X=SCREEN_RIGHT-200
ScoreP1OnCommand=visible,true
SecondaryScoreP1OnCommand=visible,false
ShowCreditDisplay=false
StepsDisplayP1OnCommand=visible,false
StepsDisplayP1X=SCREEN_RIGHT+SCREEN_RIGHT
StepsDescriptionP1OnCommand=visible,false
LifeP1OnCommand=visible,false
```

Abb. 8 – Auszug der metrics.ini-Datei im StepMania 5 Ordner

```

function JudgmentTransformCommand( self, params )
    local y = -3000
    if params.bReverse then y = y * -1 end
    self:x( 0 )
    self:y( y )
end

```

Abb. 9 – Auszug der Other.lua-Datei im StepMania 5 Ordner

Hier sieht man das finale Overlay nach Anwendung der Modifikationen durch das eigen-erstellte Theme. Die einzigen Elemente auf dem Bildschirm sind der Gameplay-Bereich, sowie die Score-Anzeige, welche sich isoliert unten auf der rechten Seite befindet.

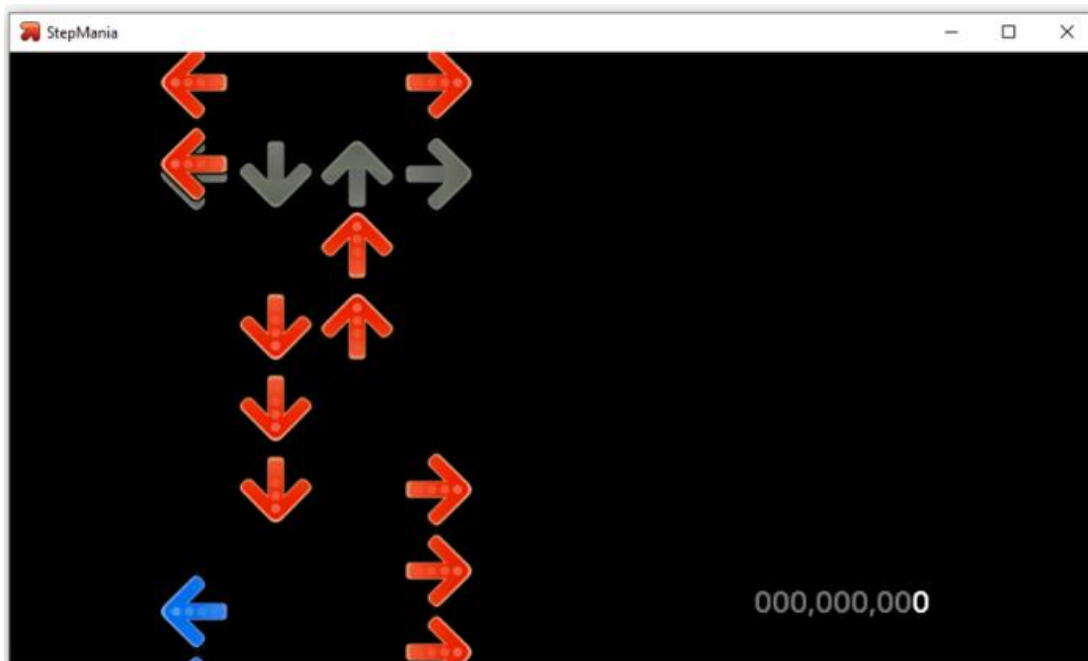


Abb. 10 - User Interface nach den Änderungen am Theme



#### 4.2.4. Tesseract und Template Matching

Für ein zuverlässiges StepMania-Environment werden Lösungen für zwei Probleme benötigt: Man braucht ein verlässliches Reward-System, welches den vorangegangenen Input bewertet, und man benötigt eine Funktion, die erkennt, ob ein Song vorüber ist, um dann den nächsten Song auszuwählen.

In der ersten Version des Environments wurde für beides Tesseract genutzt. Tesseract ist ein Open-Source-Programm, welches Optical Character Recognition, auch OCR genannt, nutzt, um Text auf Bildern zu erkennen. Tesseract konnte mit Hilfe dieser OCR-Funktionalität genutzt werden, um verschiedene Teile des Bildschirms für die oben genannten Zwecke zu interpretieren.

Für den ersten Anwendungsfall, also das Reward-System, wurde eine Funktion geschrieben, welche ein Bild als Input nimmt, und ausschließlich auf Zahlen überprüft. Als Input wurde der Teil des Bildschirms genutzt, welcher die derzeitige Punktzahl anzeigt. So wurde die gezeigte Zahl in einen verwertbaren Integer umgewandelt und konnte so als Reward für den Agent genutzt werden. Wurde nichts erkannt, beispielsweise, weil beim Start eines Songs keine Punktzahl zu sehen ist, wird 0 zurückgegeben.



Abb. 11 - Erkannte Zahl als Output (oben) mit dem Screenshot der Zahl als Input (unten)

Der zweite Anwendungsfall wird ähnlich gelöst. Hier muss erkannt werden, ob sich das Spiel derzeit auf dem Ergebnisscreen befindet, welcher angezeigt wird, wenn ein Song geendet hat. Um dies zu erkennen, wurde die gleiche Funktion von Tesseract genutzt, jedoch in einem anderen Kontext.

In diesem Bild sieht man den Ergebnisscreen eines Songs.

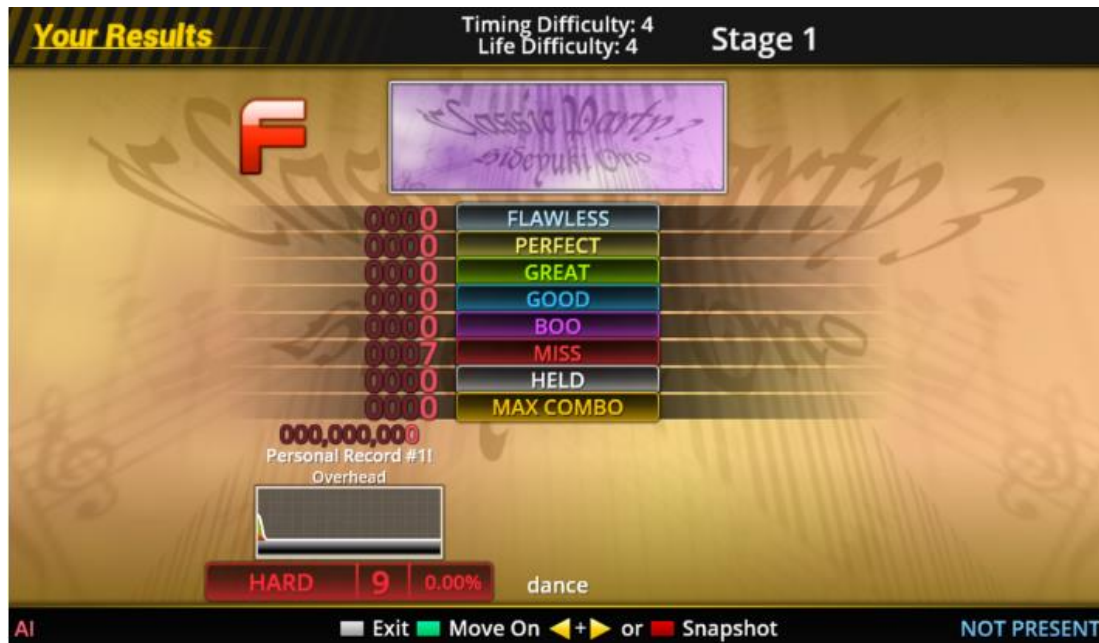


Abb. 12 - Ergebnisscreen eines Charts

Der Teil, auf den sich hier konzentriert wird, ist die obere linke Ecke, in welcher „Your Results“ steht. Tesseract hat hier diesen kleinen Ausschnitt des Bildes als Input bekommen und sollte nun die Wörter „Your Results“ erkennen. Wenn dies der Fall ist, wird die Reset-Methode ausgeführt.

Für die Erkennung wurde eine Funktion geschrieben, welche als Input sowohl das Bild als auch ein Array der zu erkennenden Zeichenfolgen und die Maximallänge der zu analysierenden Zeichen enthält.

Ein Array wurde gewählt, da Tesseract vor allem bei niedriger Auflösung manche Buchstaben als andere identifiziert. „Your Results“ wurde bei geringerer Auflösung beispielsweise als „Yow Results“ oder „Ywr Results“ erkannt. Ist dies der Fall, können mithilfe des Arrays häufig auftretende Alternativen einbezogen werden.

Die Maximallänge der Zeichen ist für diesen Anwendungsfall relevant, da Tesseract auch Leerzeichen und Zeilenumbrüche erkennt und zusammen mit dem Text ausgibt. Um eine Umformatierung des Strings zu umgehen, wurde dieser Parameter eingeführt. Diesen Parameter durch die Anzahl der Zeichen im Array selbst zu bestimmen ist in dem Fall nicht möglich gewesen, da die Einträge im Array der Zeichenfolgen unterschiedlich lang sein könnten.

Als Output wird ein Boolean zurückgegeben, welcher aussagt, ob das mitgegebene Bild eines der zu erkennenden Zeichenfolgen enthält.

Im folgenden Bild sieht man das Ergebnis des Aufrufs:

```
text_recog_helper.is_text_in_image(obs, ['Your Results'], 12).
```

*obs* bezeichnet hier die Observation, welche ebenfalls im Bild zu sehen ist.

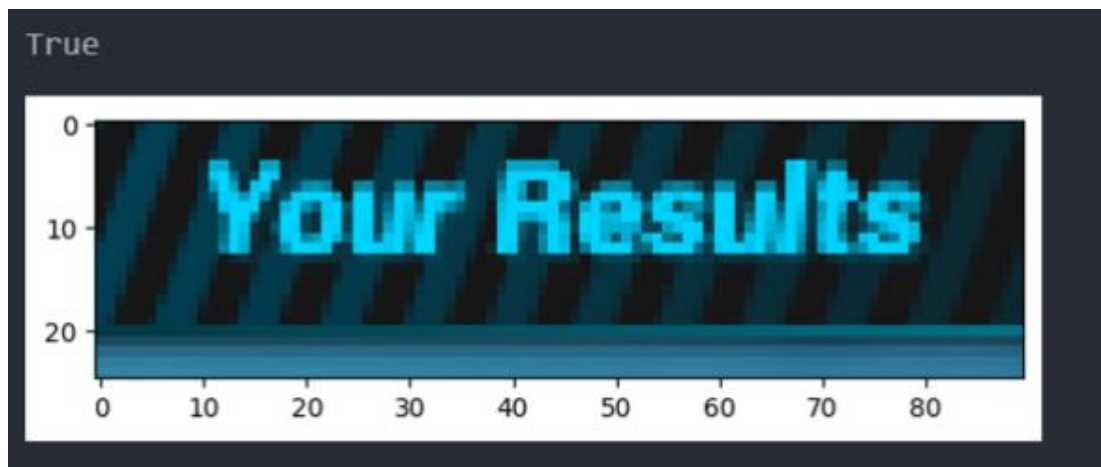


Abb. 13 - Ergebnis des genannten Funktionsaufrufs (oben) und die dazu gehörende Observation (unten)

Während die Nutzung von Tesseract sehr unkompliziert und einfach ist, hat es einen für diesen Nutzungskontext nicht unerheblichen Nachteil - die Performance. Der Aufruf der Ergebnisscreen-Analyse dauerte abgerundet etwa 0,17 Sekunden. Dies klingt nach wenig, jedoch ergibt sich dadurch, dass die Ergebnisscreen-Analyse und die Reward-Analyse beide in jedem Step ausgeführt werden, eine Zeit pro Step von etwa 0,4 Sekunden. Der Großteil der Zeit wird von eben diesen Tesseract-Aufrufen eingenommen, da diese auf ein extra installiertes Programm zugreifen müssen.

Um dieses Problem zu lösen, wurde ein Template-Matching-Ansatz eingeführt. Hier soll, ähnlich zu dem Template Matching beim Gameplay, eine Referenz mit dem Screenshot verglichen werden, um zu bestimmen, ob z.B. der Ergebnisscreen erreicht wurde.

Um die Ergebnisscreen-Erkennung zu verbessern, wurde ein Template des “Your Results”-Teils des Ergebnisscreens erstellt. Dieses Template wurde dann in jedem Step mit der derzeitigen Observation verglichen, um festzustellen, ob es sich auf dem Bildschirm befindet.

```
def analyze_results(self, img):
    threshold = 0.9
    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    res = cv.matchTemplate(img_gray, self.template_results,
cv.TM_CCOEFF_NORMED)
    if (res >= threshold).any():
        return True
    return False
```

*Listing 7 - Code der analyze\_results-Funktion im Pattern Recognition Modul*

Das gleiche Prinzip wurde nun für das Reward-System angewendet. Hier wurden Screenshots von jeder Zahl als Template geladen, und in einer Funktion mit dem Screenshot der Punktezahl verglichen. Der Teil, welcher diese Funktion von der oberen unterscheidet, ist, dass die erkannten Zahlen in der richtigen Reihenfolge angeordnet werden müssen.

Wird zum Beispiel die Zahl „4732“ erkannt, werden alle 10 Templates nacheinander, beginnend mit 0, verglichen. Gibt es eine Übereinstimmung, wird die X-Koordinate des Bildausschnittes, wo die Übereinstimmung stattgefunden hat, zusammen mit der Zahl in ein Array hinzugefügt. Dabei gibt der erste Wert die X-Koordinate an, und der zweite Wert die Zahl, welche erkannt wurde.

Zu diesem Zeitpunkt würde das Array für das Beispiel so aussehen:

```
[[40, 2], [30, 3], [10, 4], [20, 7]]
```

Dieses Array würde jetzt der Zahl „2347“ entsprechen, weshalb es nun in aufsteigender Reihenfolge nach der X-Koordinate sortiert wird, sodass das Array nicht mehr nach den erkannten Zahlen, sondern der erkannten Position von links nach rechts geordnet ist:

```
[[10, 4], [20, 7], [30, 3], [40, 2]]
```

Anschließend werden die Zahlen, also die zweiten Werte in jedem Objekt, aus dieser sortierten Reihenfolge ausgelesen und zusammengeführt, sodass das finale Ergebnis „4732“ ist.

```
def analyze_score(self, img):
    threshold = 0.82
    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    found_array = []
    index_template = 0
```

```

for index_template in range(10):
    res = cv.matchTemplate(
        img_gray,
        self.template_numbers[index_template],
        cv.TM_CCOEFF_NORMED
    )
    loc = np.where(res >= threshold)
    for pt in zip(*loc[::-1]):
        found_array.append([pt[0], index_template])

sorted_array = sorted(found_array, key=lambda x: x[0])
result = ""
for array in sorted_array:
    result += str(array[1])
if result == "":
    result = "0"
return(int(result))

```

Listing 8 - Code der `analyze_score`-Methode im Pattern Recognition Modul

Die optimierte Funktionsweise des Reward-Systems ist zwar um einiges schneller, hat allerdings auch seine Nachteile. Durch die niedrige Auflösung der Screenshots kann es vereinzelt vorkommen, dass Zahlen nicht richtig erkannt werden. Dies kann im schlimmsten Fall zu einer falschen Reward-Verteilung führen, was den Lernerfolg des Agents einschränken würde.

Durch das Anpassen des Thresholds, als auch der Auflösung des Spiels, konnte jedoch erreicht werden, dass nach 10 Tests mit je 5 Episoden alle Zahlen richtig erkannt wurden. Dieses Verfahren ersetzte danach die Tesseract-Erkennung und sorgte für eine Performancesteigerung von 2,35 auf 4,59 Bilder pro Sekunde.

```

Environment started - Using random inputs
Total Reward for episode 0 is 0
Total screenshots for episode 0 is 246
Total duration of episode 0 is 104.4965 seconds
This equals an average of 2.354146461179908 images per second

```

Abb. 14 - Ergebnis des Benchmarktests mit Tesseract

```

Environment started - Using random inputs
Total Reward for episode 0 is 0
Total screenshots for episode 0 is 480
Total duration of episode 0 is 104.4721 seconds
This quals an average of 4.594528108461494 images per second

```

Abb. 15 - Ergebnis des Benchmarktests nach beiden Änderungen

#### 4.2.5. Erster Lernversuch

Nachdem das Environment und der Agent aufgesetzt und die Performance optimiert wurde, konnte der erste Lernversuch unternommen werden. Ziel war, diese Fortschritte in mehreren Hinsichten zu analysieren. Zuerst war es wichtig zu wissen, ob es generell Fortschritte gibt. Während der Tests des Modells war noch kein Wissen vorhanden, weshalb jeder Input zufällig bestimmt und nur manchmal tatsächlich brauchbare Tastendrucke ausgegeben wurden.

Der Lernversuch selbst umfasste rund zweieinhalb Stunden, was nach der optimierten Performance etwas mehr als 50.000 Steps entsprach. Das Ergebnis fiel jedoch schlechter aus als erwartet.

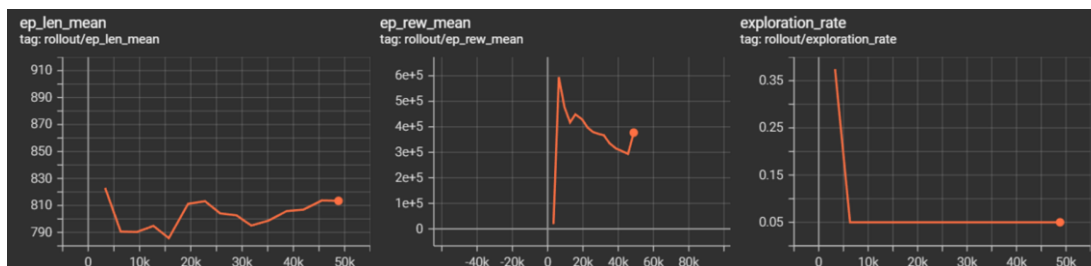


Abb. 16 - Screenshot von Tensorboard - Darstellung der Mittelwerte der Episodenlänge, des Rewards, sowie der Wert der Exploration Rate des ersten Lernversuchs über die Lerndauer

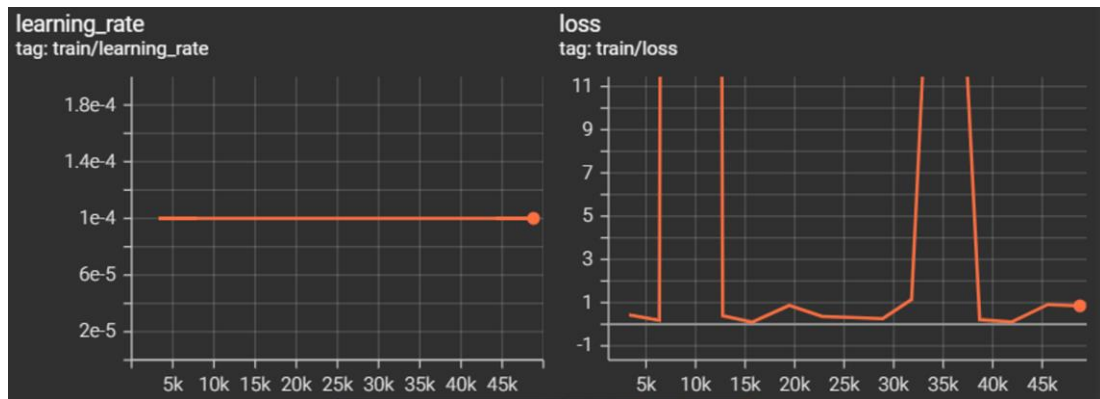


Abb. 17 - Screenshot von Tensorboard - Darstellung der Learning Rate und der Loss Rate des ersten Lernversuchs über die Lerndauer

Auf den Grafiken, welche mithilfe von Tensorboard erstellt wurden, konnte man sehen, dass der Mittelwert der Rewards tendenziell über die Zeit sank. Dies bedeutet im Kontext der Anwendung, dass mit längerem Lernen, weniger Punkte erzielt worden sind.

An dieser Tabelle kann man erkennen, welche Tasten das Modell wie oft nach 10 Minuten Spielzeit gedrückt hat.

Taste	Anzahl der Inputs
Oben	5921
Rechts	0
Unten	0
Links	3

Tabelle 1 - Anzahl Tastendrucke für jede Taste nach 10 Minuten

Man sieht, dass es eindeutige Probleme bei dem Lernverhalten der KI gibt, da die Fixierung auf eine Taste keine optimale Verhaltensweise darstellt. Die KI hat hier gelernt, dass es stetig eine gewisse Anzahl an positivem Feedback erhält, wenn eine Taste durchgängig gedrückt, direkt wieder losgelassen, und erneut gedrückt wird. Dies liegt daran, dass der Reward äquivalent zur Punktezahl ist. Dieser hohe Reward direkt am Anfang des Lernprozesses scheint der Grund gewesen zu sein, dass sich die KI nur auf diese eine Taste fokussierte, und alle anderen ignorierte.

Um zu testen, ob dieses Problem tatsächlich bereits nach kurzer Lerndauer auftrat oder es ein stetiger Prozess war, wurde die gleiche Grafik für zwei neu-trainierte Modelle nach 5.000, 10.000 und 25.000 Steps angefertigt.

<b>Taste</b>	<b>Anzahl der Inputs, 5.000 Steps</b>	<b>Anzahl der Inputs, 10.000 Steps</b>	<b>Anzahl der Inputs, 25.000 Steps</b>
Oben	4872	9629	24.091
Rechts	3	0	0
Unten	0	0	0
Links	0	0	0

*Tabelle 2 - Anzahl Tastendrucke pro Taste, nach 5.000, 10.000 und 25.000 Schritten*

Man kann erkennen, dass dieses Problem also fundamental an der Umsetzung des Reward-Systems liegt. Da die hier genutzten Parameter für die Exploration Rate und die Learning Rate die Standardwerte für DQN-Modelle unter Stable Baselines 3 sind, sollte man jedoch auch anmerken, dass diese sich eventuell nicht für diesen Anwendungsfall eignen. Genauer gesagt kann es sein, dass die Exploration Rate zu niedrig und die Learning Rate zu hoch ist. Hier wird sich zu sehr und zu früh auf das bereits gelernte fokussiert, was in diesem Fall bedeutet, dass sobald ein Ansatz für konstanten Reward gefunden wurde, nur dieser Ansatz verfolgt wird, und zu wenige Alternativen ausprobiert werden.

Für den nächsten Lernversuch galt es also, das Reward-System entsprechend anzupassen. Sollte das Problem weiter bestehen, können Änderungen an den Parametern des Modells in Betracht gezogen werden.

#### 4.2.6. Neues Reward-System

Nachdem im letzten Kapitel festgestellt wurde, dass das Reward-System nicht korrekt funktioniert, galt es nun eine Lösung zu finden, wie man dem Modell Feedback gibt, ohne es auf die falsche Bahn zu leiten.

Zuerst muss das Problem genauer untersucht werden. Da das Modell scheinbar ausgelernt hat und sich damit zufriedengegeben hat, nur eine Taste zu drücken, gab es zu viel positiven Reward für die einzelnen Aktionen. Bisher wurde der Reward so berechnet, dass der Reward des aktuellen Steps mit dem des vorherigen Steps subtrahiert wurde. Ein richtiger Tastendruck konnte so potenziell mehrere Tausende Punkte bedeuten, und somit auch mehrere Tausende Punkte an Reward an das Modell



geben. Dieses Prinzip musste also herunterskaliert werden, um den Reward zu normieren.

Ebenso gibt es im momentanen Environment keine Möglichkeit, dem Modell zu sagen, dass es etwas schlecht gemacht hat, ihm also negativen Reward zu geben. Der Score ist immer gleichbleibend und es werden keine Punkte abgezogen, wenn eine Note nicht getroffen wird. Auch existiert keine Erkennung, wenn eine Note verpasst wurde. Dies bedeutet, dass man dem Modell für schlechte Entscheidungen kein negatives Feedback geben kann - es gibt nur neutrales oder positives Feedback. Das Environment musste also dementsprechend angepasst werden.

Das Runterskalieren des Reward-Systems ist nur eine kleine Änderung. Prinzipiell ändert sich nichts, man vergleicht immer noch den Score des vorherigen Steps mit dem des aktuellen Steps, gibt nun aber nicht mehr die Differenz wieder. Hat sich der Score verbessert, wird ein Reward von 1 vergeben. Hat er sich nicht verändert, beträgt der Reward 0. Dies bedeutet, dass das Modell für jede getroffene Note belohnt wird. Die Genauigkeit wird in dem Fall außer Acht gelassen, da das jetzige Ziel ein generell funktionierendes Modell ist, und kein besonders genaues. Das Modell kann also verbessert werden, wenn mehr Reward für genauere Tastendrucke vergeben werden, dies würde jedoch den Umfang des Projektes überschreiten.

Für das zweite Problem, das Vergeben von negativem Feedback, gibt es zwei Lösungsansätze. Zuerst kann man prüfen, ob Noten nicht getroffen wurden. Wurden Noten nicht getroffen, bewegen sie sich weiter nach oben, bis sie sich außerhalb des sichtbaren Bereichs befinden, während getroffene Noten verschwinden. Man könnte nun also prüfen, ob sich über dem Judgement-Bereich, also der statischen Reihe an Pfeilen am oberen Ende des Bildschirms, Noten befinden, und dementsprechend Abzug geben. Problematisch ist hier, dass die Aktion, die dafür verantwortlich ist, dass diese Noten nicht getroffen wurden, schon lange vergangen ist. Man würde hier also Reward entweder zurückhalten bis geklärt wurde, ob Noten verfehlt wurden, oder man vergibt negativen Reward in Steps, die eventuell korrekt ausgeführt wurden, und kein schlechtes Feedback verdient haben. Zwar wäre es möglich mit „delayed Rewards“ zu arbeiten, was allerdings eine zu komplizierte Lösung für ein relativ simples Problem ist.

Sinnvoller ist es, zu prüfen, welche Tasten im momentanen Schritt gedrückt werden sollen – man analysiert also, ob sich derzeit eine Note im Judgement-Bereich befindet. Man arbeitet hier in der Gegenwart und nicht in der Vergangenheit, wie es bei den anderen beiden Ansätzen der Fall war. Dies eliminiert das Problem, Feedback für bereits vergangene Aktionen geben zu müssen. Der Ablauf wäre jetzt wie folgt: In jedem Screenshot wird geschaut, ob farbige Pixel in einem bestimmten Pixelbereich vorkommen. Diese farbigen Pixel lassen auf eine Note schließen, welche sich derzeit in dem Judgement-Bereich befinden. Dies wird für alle vier Notenspalten einmal ausgeführt. Man hat nun also vier Wahrheitswerte, welche aussagen, ob sich derzeit eine Note in einer der vier Judgement-Bereiche befindet. Das Modell sollte hier also einen Tastendruck zurückgeben. Anschließend wird geprüft, ob die Ausgabe des Modells mit der Analyse des Judgement-Bereichs übereinstimmt - die Note also richtig erkannt und gedrückt wurde. Ist dies der Fall, wird dem Modell positiver Reward, nach dem Herunterskalieren also der Wert +1, zurückgegeben. Wurde die Aktion nicht ausgeführt, die Note also verpasst, wird negativer Reward (-1) an das Modell gegeben. Wird keine Note erkannt und auch keine Aktion ausgeführt, wird +1 zurückgegeben, da dies in dieser Situation das richtige Verhalten ist. So soll kontinuierliches und unnötiges Tastendrücken abgewöhnt/verhindert werden.

```

def get_reward(self, img, action):
    # Crop score part of the window screenshot and top of the gameplay section
    score_img = img[self.score_location['top']:(self.score_location['top'] +
self.score_location['height']),
self.score_location['left']:(self.score_location['left'] +
self.score_location['width'])]
    past_arrows_img =
img[env.past_arrows_location['top']:(env.past_arrows_location['top'] +
env.past_arrows_location['height']),
env.past_arrows_location['left']:(env.past_arrows_location['left'] +
env.past_arrows_location['width'])]

    # If no input should have occurred, give negative reward
    if (self.pattern_recog_helper.input_expected(past_arrows_img, action) ==
False):
        return -1

    # If the score increased, give positive reward
    new_reward = self.pattern_recog_helper.analyze_score(score_img)
    if (new_reward > self.previous_reward):
        # Set the current reward as the previous reward for the next iteration
        self.previous_reward = new_reward
        return 5

    # If the score didn't change, and no action has taken place, give a
    neutral reward
    else:
        return 0

```

*Listing 9 – Code der überarbeiteten get\_reward-Methode des Environments*

```

def input_expected(self, img, action):
    hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
    left_arrow = hsv[:,5:35,:]
    down_arrow = hsv[:,40:70,:]
    up_arrow = hsv[:,70:100,:]
    right_arrow = hsv[:,105:135,:]

    # Checks for left, down, up and right input
    if (action == 1):
        return self.check_for_color(left_arrow)
    elif (action == 2):
        return self.check_for_color(down_arrow)
    elif (action == 3):
        return self.check_for_color(up_arrow)
    elif (action == 4):
        return self.check_for_color(right_arrow)
    # If no action took place, return True
    else:
        return True

def check_for_color(self, img):
    lower_val = np.array([0, 150, 150], dtype='uint8')
    upper_val = np.array([255, 255, 255], dtype='uint8')
    mask = cv.inRange(img, lower_val, upper_val)
    hasColor = np.sum(mask)
    if (hasColor > 0):
        return True
    else:
        return False

```

*Listing 10 – Code der input\_expected- und check\_for\_color-Methoden des Pattern Recognition Moduls*

Wichtig ist, diese Art der Erkennung nur zum Validieren des Inputs zu nutzen. Um ein „perfektes“ Modell zu erzeugen, könnte man natürlich beim Erkennen eines farbigen Pixels direkt einen Tastendruck auslösen – dies hat allerdings nichts mit künstlicher Intelligenz oder einem Lernprozess zu tun.

Nachdem die Anpassungen erfolgten, wurden mehrere Tests durchgeführt, welche in der Länge variierten. Der längste von ihnen war 150.000 Schritte lang. In der folgenden Abbildung 18 wird dieser Lernprozess gezeigt.

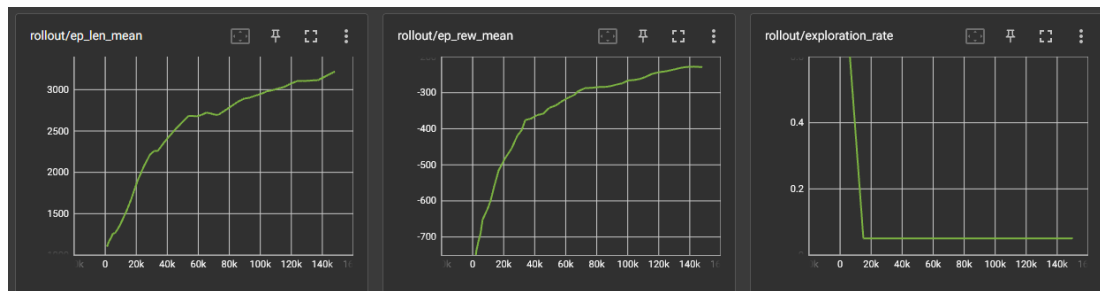


Abb. 18 - Screenshot von Tensorboard - Darstellung der Mittelwerte der Episodenlänge, des Rewards, sowie der Wert der Exploration Rate nachdem das Reward-System überarbeitet wurde

Man sieht, dass der Mittelwert der Rewards pro Episode kontinuierlich steigt, was Lernfortschritt signalisiert. Der Wert flacht gegen Ende des Lernprozesses ab, was allerdings auch an der kurzen Lerndauer liegt – 150.000 Schritte reichen bei einem komplexen Spiel wie StepMania nicht, um ein gutes Modell zu erzeugen.

Die überarbeiteten Konzepte funktionieren, wie man bei den Resultaten sehen konnte, ziemlich gut. Problematisch wird es allerdings, wenn es in dem Song mehr als nur die Standard-Pfeile gibt. StepMania hat neben den normalen Pfeilen, welche man nur einmal antippen muss, auch noch andere Pfeilarten. Ebenfalls vorkommen können die bereits angesprochenen “Holds”, also Noten, welche für einen längeren Zeitraum gedrückt gehalten werden müssen, “Mines”, welche gar nicht gedrückt werden dürfen, “Rolls”, welche wiederholend gedrückt werden müssen, und “Lifts”, welche umgekehrt wie normale Pfeile funktionieren - hier muss die Taste vorher gedrückt werden, und im richtigen Moment losgelassen werden.

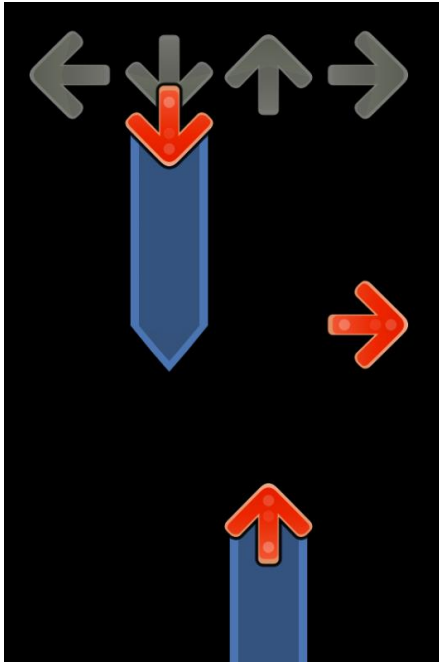


Abb. 19 - Screenshot von zwei "Holds"

Während "Rolls" und "Lifts" erst in höheren Schwierigkeitsgraden relevant werden, können "Holds" und "Mines" in jeder Art von Schwierigkeit vertreten sein. Dies ist insofern problematisch, da bei der Prüfung für negatives Feedback nicht zwischen den einzelnen Pfeilarten unterschieden wird. Eine "Mine" beispielsweise wird im momentanen Zustand der Funktion als Pfeil identifiziert und das Modell bekommt negatives Feedback, wenn es keine Taste drückt - also genau das Gegenteil von dem, was eigentlich passieren sollte. Da „Mines“ jedoch selten auftauchen, wurde dieses Problem ignoriert. Selbst wenn die Erkennung funktionieren würde, sind sie so selten in den Charts vertreten, dass das Lernen des richtigen Verhaltens in dieser Situation sehr lange dauert würde. Dass „Holds“ negativen Reward geben, musste in Anbetracht des Umfangs der Arbeit zurückgestellt werden.

#### 4.2.7. Datenkomprimierung

Momentan wird das Bild, welches der KI als Observation gegeben wird, kleiner skaliert und in Schwarz-Weiß umgewandelt, möglichst wenig Daten übermittelt werden müssen. Doch inwiefern kann man diesen Prozess noch weiter optimieren?

Eine Möglichkeit ist die Umwandlung der Bilder von Schwarz-Weiß in Binärbilder. Während in der Schwarz-Weiß-Version unterschiedliche Graustufen existieren, welche mit den Werten 0 bis 255 gespeichert werden, gibt es in Binärbildern nur Schwarz oder Weiß. Ein Pixel ist dementsprechend an (1) oder aus (0). Da die Werte nun nur von 0 bis 1 reichen, ist die Datengröße der Bilder auch kleiner.

Hierfür wurde der bereits bestehende Code, der die Bilder komprimiert, um eine Zeile erweitert, in welcher eine Threshold-Methode ausgeführt wird, die alle Pixel unter dem Wert 60 zu 0 und alle Pixel über 60 zu 1 ändert.

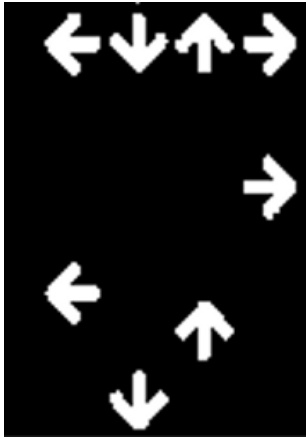
```
def downscaleImage(self, img, size, shape):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    black_white = cv2.threshold(gray, 60, 255, cv2.THRESH_BINARY)
    resized = cv2.resize(black_white, size)
    channel = np.reshape(resized, shape)
    return black_white
```

*Listing 11 - Code der downscaleImage-Methode im Image Analysis Modul*

Der Unterschied ist erkennbar:



*Abb. 20 - Bildschirmaufnahmen des Gameplays, in Graustufen*

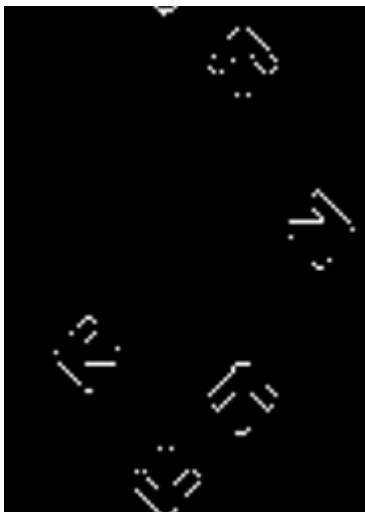


*Abb. 21 - Bildschirmaufnahmen des Gameplays, als Binärbild*

Die Größe der Datei ist dabei von 2,05 Kilobyte in der Graustufen-Version zu 782 Bytes in der binären Version gesunken.

Während Binärbilder schon eine starke Form der Dateneinsparung ermöglichen, ist es möglich, auf diesem Konzept aufzubauen und weiter zu optimieren. Eine Möglichkeit dafür ist es, ein Vergleichsbild von der letzten Observierung und der aktuellen anzufertigen. In diesem Bild sollen nur die Pixel zu sehen sein, welche im zweiten Bild einen anderen Wert haben als im ersten Bild. So beschränken sich die Informationen, welche im Bild übertragen werden, nur auf die sich bewegendenden Pfeile. Die statische Reihe an Pfeilen, die sich immer am oberen Bildschirmrand befindet, wird so komplett ignoriert.

Ein Vergleichsbild von zwei Observationen sieht so aus:



*Abb. 22 - Bildschirmaufnahme des Gameplays, als Vergleichsbild*



Die Größe der Datei ist weiter gesunken – sie beträgt nun nur noch 524 Bytes.

Das Konzept, zwei Bilder zu vergleichen, ist jedoch nicht fehlerfrei. Die bereits angesprochenen Arten von Pfeilen sind auf den Vergleichsbildern nur schlecht zu erkennen. Im folgenden Beispiel sieht man zwei Hold-Pfeile, also Pfeile, die länger gedrückt gehalten werden müssen.

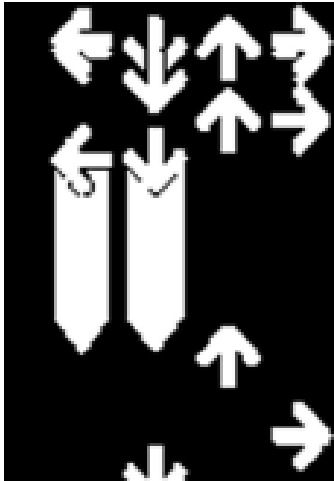


Abb. 23 - Bildschirmaufnahme des Gameplays, auf welchem u.a. zwei „Hold“-Pfeile zu sehen sind, als Binärbild

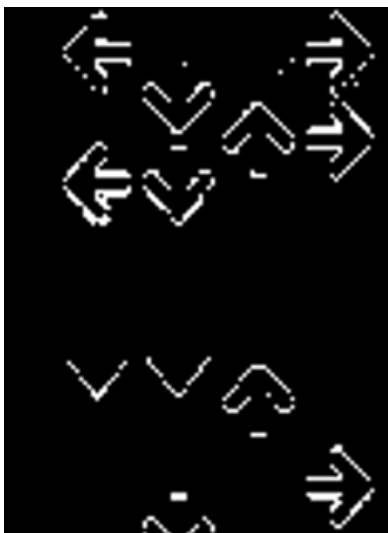
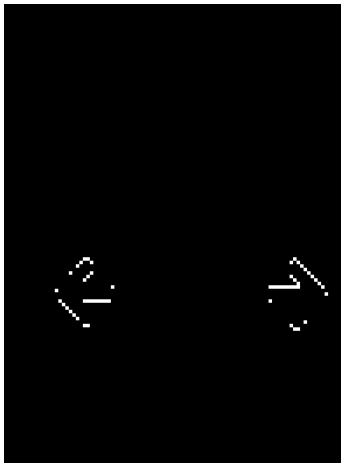


Abb. 24 - Bildschirmaufnahme des Gameplays, Vergleichsbild von Abb. 21

Da zwei Binärbilder verglichen werden, ist der Großteil des Hold-Pfeils im Vergleich nicht zu sehen, weil der innere Teil in beiden Bildern weiß ist und sich so nur der Pfeil selbst und die untere Kante tatsächlich “bewegt”. Ein Hold-Pfeil ist so von einem regulären Pfeil nicht zu unterscheiden. Während die Hold-Pfeile in diesem Beispiel durch die untere Kante noch identifiziert werden können, ist dies nicht immer der Fall: Im folgenden Bild kann man nicht erkennen, dass beide Pfeile gedrückt gehalten werden müssen, da sich die untere Kante außerhalb des Bildschirms befindet.



*Abb. 25 - Weitere Bildschirmaufnahme des Gameplays, als Vergleichsbild*

Da dieses Konzept nicht genug Informationen für die KI bereitstellt und dies das Lernen negativ beeinflusst, wurde sich trotz der Ersparnisse in Daten gegen das Konzept der Vergleichsbilder entschieden und weiter mit Binärbildern gearbeitet.

#### 4.2.8. Abschließender Trainingslauf

Um den praktischen Teil zu beenden und ein Resümee ziehen zu können, wurde ein Trainingslauf mit 1.4 Millionen Schritten durchgeführt. Der Vorgang dauerte insgesamt knapp 15 Stunden und wurde in drei separaten Trainingsepisoden, farblich im Bild differenziert, durchgeführt. Der Graph, welcher den durchschnittlichen Reward pro Episode zeigt, sah nach Abschluss des Trainings wie folgt aus:

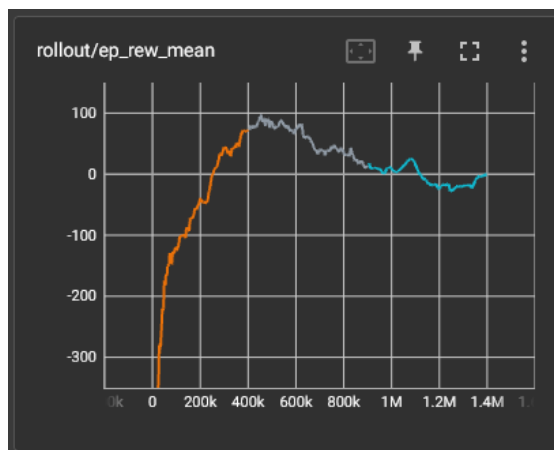


Abb. 26 - Graph in TensorBoard, welcher den durchschnittlichen Reward pro Episode über 1,4 Millionen Schritte zeigt.

Dieser Graph kann repräsentativ für den Fortschritt des Modells betrachtet werden. Bei knapp 500.000 Schritten erreicht der Wert seinen Höhepunkt, wonach er abflacht und sich zum Ende des Trainings bei 0 befindet. Warum dies so ist, kann nur anhand dieser Daten nicht beantwortet werden, da mehrere Gründe in Frage kommen.

Zum einen kann es an der Programmierung des Environments, wie beispielsweise einer nicht-optimierten Rewardfunktion liegen. Diese wurde zwar in einem vorherigen Kapitel angepasst, jedoch kann der Lernerfolg immer noch durch präzises Anpassen der Funktionsvariablen oder ähnlichem stark beeinflusst werden.

Ebenfalls können die Hyperparameter in Frage kommen, also beispielsweise wie sehr sich das Modell auf Gelerntes verlässt, oder wie oft es neues probieren soll. Diese Parameter spiegeln die in Kapitel 2.3.2 aufgezeigte Epsilon-Greedy-Strategie wieder. In der Praxis ist das „Finetuning“, also das genaue Abstimmen dieser Parameter auf den Anwendungsfall, leider sehr zeitintensiv und würde weitere Trainingsläufe von ähnlicher Dauer für Vergleichszwecke voraussetzen. Dies ist in dieser Arbeit nicht möglich gewesen. Dennoch ist davon auszugehen, dass mit diesem Finetuning der

Werte das Modell Verbesserungen aufzeigen würde, da in dem durchgeführten Trainingslauf alle Parameter auf ihrem nicht-optimierten Ursprungswert gelassen wurden.

Eine weitere mögliche Erklärung für die Performance ist das genutzte Modell selbst. Während das DQN ein adäquates Modell für den Anwendungsfall darstellt, kann davon ausgegangen werden, dass andere Modelle bessere Ergebnisse erzielt hätten. Speziell sollen dabei Modelle gemeint sein, welche auf dem Konzept des DQN aufbauen, und dies mit weiteren Techniken verbessern. Hier soll explizit auf das Paper „Rainbow: Combining Improvements in Deep Reinforcement Learning“ der Autoren Matteo Hessel et al. von Google DeepMind verwiesen werden. In diesem Paper werden sechs Verbesserungen und Erweiterungen am Konzept des DQN vorgestellt und erläutert, um alle schlussendlich in einem Algorithmus zusammenzufassen, welcher „Rainbow“ genannt wird (Hessel et al. 2017).

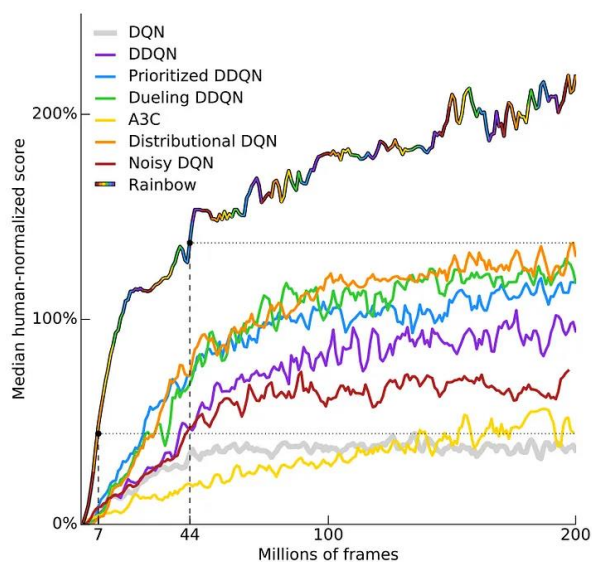


Abb. 27 - Performance der unterschiedlichen DQN-Variationen über 57 Atari-Spiele (aus: Hessel et al. 2017, S.1, Figure 1)

In Abbildung 27 kann man sehen, dass diese Kombination aus verschiedenen DQN-Variationen beim Fortschritt einen enormen Abstand zu anderen DQN-Konzepten aufweist. Aber auch wenn man jede einzelne Variation mit dem ursprünglichen DQN-Verfahren vergleicht, wird ersichtlich, dass diese besser sind. Insofern kann davon ausgegangen werden, dass unter der Nutzung eines optimierten Environments und abgestimmten Hyperparametern die Nutzung einer DQN-Variation, oder der Kombination aus mehreren dieser Variationen, ein besseres Ergebnis erzielt hätte werden können.

### 4.3. Bewertung und Einordnung

Die generelle Umsetzung einer künstlichen Intelligenz für das Spiel StepMania 5 stellte sich aus programmiertechnischer Perspektive als relativ simpel heraus. Durch die Python Packages Gym und Stable Baselines 3 ist es dem Entwickler möglich, in nur wenigen Zeilen Code eine KI aufzusetzen, welche positiven Lernfortschritt aufzeigt. Dabei liefern beide Packages genug Möglichkeiten, um das Grundkonstrukt weiter anzupassen und zu spezifizieren und so verschiedene Konzepte und Ideen umzusetzen, um so den Lernprozess weiter zu optimieren, was der zeitlich aufwändigste Teil des Prozesses ist. Im Rahmen dieser Arbeit wurden weitere Ideen und Ansätze ausprobiert, welche keinen signifikanten Fortschritt erzeugten, um sie detailliert zu erwähnen. Dieses freie Ausprobieren, als auch die generelle Umsetzung, stellte sich durch die angebotenen Funktionen der Packages als sehr intuitiv heraus und verlief durch die ausführliche Dokumentation meist problemlos.

Während das erreichte Ergebnis für den Zeitraum dieser Arbeit dem entspricht, was erwartet wurde, ist das Modell nicht optimiert und kann durch Änderungen an Parametern, dem Environment oder des verwendeten Modells weiter verbessert werden. Mehr Zeit in das Finetuning auf das Spiel zu investieren, als es hier gemacht wurde, würde das Modell weiter verbessern und optimieren, sodass schwierigere Charts gespielt werden können. Inwiefern und wie bald dann ein Plateau im Fortschritt erreicht werden würde, oder ob ein Modell unter diesen Umständen mit genug Training „perfekt“ sein würde, lässt sich aus der momentanen Perspektive nicht sagen.

Die Erkenntnisse aus diesem Projekt kann man auf andere, ähnliche Spiele übertragen. Das Prinzip, nur den visuellen Input als Observation für die KI zu nutzen, und dadurch Tastendrücke zu bestimmen, ist universell genug, dass es auch in anderen Rhythmusspielen eingesetzt werden kann. Wie der Erfolg in diesen Spielen ausfällt, kann man nicht universell sagen, da dies von der Eindeutigkeit der Informationen, der Komplexität des Gameplays und dem generellen Aufbau des Spiels abhängt.

Einen anderen Ansatz für den Lernprozess zu nutzen, wäre ein interessanter Vergleich zu diesem Projekt. Während Evolutionary Algorithms ein spannendes Konzept darstellen, sind diese, wie in Kapitel 2.2.3 beschrieben, für Rhythmusspiele leider nicht geeignet. Andere Lernmethoden, wie z.B. eine speziell angepasste Variante des

Supervised Learning, können Alternativen zum Reinforcement Learning darstellen. Inwiefern diese im Anwendungsfall StepMania 5 besser sind, als das genutzte Reinforcement Learning, bedarf jedoch einem ausführlichen Vergleich. Im Kontext von anderen Spielen ist die Wahl zwischen Reinforcement Learning und anderen Lernalgorithmen schwerer, da hier die technischen und inhaltlichen Gegebenheiten der Spiele eine große Rolle spielen.

## 5. Fazit und Ausblick

Künstliche Intelligenz erlebt derzeit einen großen Aufschwung. Fast täglich kommen neue Webseiten oder Tools zum Vorschein, welche die Möglichkeiten von KI in einer neuen Art und Weise nutzen, und so sowohl Nutzer gewinnen als auch andere Entwickler zu Neuem inspirieren. Wie in Kapitel 3 bei den Streiks der WGA angesprochen wurde, wird KI mittlerweile aktiv im Bereich der Konzeption von Filmen und Serien eingesetzt und zwingt so Angestellte dazu, gegen den Einsatz von künstlicher Intelligenz zu demonstrieren, damit sie ihren Arbeitsplatz behalten.

Dabei beschränkt sich der Einsatz von KI in der Medienproduktion nicht nur auf die Konzeption. Adobe veröffentlichte Ende Mai das „Generative Fill“-Feature in der Bildbearbeitungssoftware Photoshop – dieses ermöglicht dem Nutzer, einen Bereich des Bildes zu markieren, und zu beschreiben, inwiefern der Bereich verändert werden soll. In einem Artikel von „The Guardian“ beschreibt Chandra Sinnathamby, der Director of Digital Media and Strategy von Adobe Asia-Pacific, das Tool eher als „Co-Pilot“, um die Arbeit zu beschleunigen, als einen finalen Ersatz für Grafikdesigner darzustellen (Taylor 2023). Während so ein Feature für Kleinunternehmen ohne Kapazitäten für Grafiker ein hilfreiches Tool ist, werden damit auch Fake-Images, also künstlich bearbeitete Bilder, welche glaubhaft erscheinen sollen, immer einfacher zu erstellen. Man steht durch den Fortschritt in KI-Technologien also gesellschaftlich vor einem ganz neuen Problem, welches insbesondere in politischen Kontexten schwere Auswirkungen haben könnte. Neben dem in den vergangenen Jahren als „Fake-News“ bezeichnetem wachsendem Aufkommen von Falschinformationen können nun also leichter als jemals zuvor angebliche Bildbeweise für irreführenden Behauptungen erstellt werden. Um dem vorzubeugen, steht also die Aufgabe im Raum, einen Weg zu finden, solche Bilder verlässlich auf ihre Echtheit überprüfen zu können.

Komplett künstlich generierte Unterhaltung, wie der Neuro-sama-Livestream oder die „Nothing, Forever“-Serie, zeigen ein interessantes Subgenre der Bewegtbildproduktion auf. Während das Erstellen einzelner Folgen weniger Personal braucht als klassische Serienproduktionen, hängt die Qualität und damit die Attraktivität der Serien auf die menschlichen Rezipienten von der KI ab. Da die KI, wie in Kapitel 3 angesprochen, durch ihre Themenauswahl stark in ihrer Qualität

variiert, ist davon auszugehen, dass komplett KI-generierte Unterhaltung noch nicht unmittelbar vor der Tür steht. Dafür spricht ebenfalls, dass sich noch kein großes Medienunternehmen öffentlich zu so einer Art der Produktion geäußert hat. Dennoch ist nicht auszuschließen, dass der Fortschritt im generellen Bereich der künstlichen Intelligenz auch für diesen Anwendungsfall große Veränderungen bedeuten könnte.

Solch ein Fortschritt würde auch bedeuten, dass künstliche Intelligenz in Videospielen ebenfalls besser wird. Wir haben bereits jetzt KI als Gegenspieler, welche auf viele Dinge reagieren kann, selbstständig Entscheidungen trifft oder aus dem Verhalten des Spielers lernt. Fortschritt in dem Bereich bedeutet, dass Computergegner in der Zukunft nur noch schwer von realen Spielern zu unterscheiden sind.

Selbiges gilt für KI, welche selbstständig Videospiele lernt und spielt, ähnlich zu dem selbst entwickelten Modell, beschrieben in Kapitel 4. Wie im Kapitel 1.1 definiert, bestand das Hauptziel des praktischen Teils der Arbeit aus der Entwicklung solch einer KI für das Spiel StepMania 5. Dieses Ziel wurde soweit erreicht. Dabei wurde in Kapitel 4 gezeigt, wie diese KI den visuellen Input nutzt, verarbeitet und in einen Output/Tastendruck umwandelt. Alternative Methoden, welche innerhalb der Entwicklung ausprobiert und schlussendlich ersetzt wurden, sind beschrieben, teils ausprobiert und evaluiert worden. Die Entwicklung des Modells kann dabei noch weitergeführt und weiter ausgearbeitet werden und bietet Optionen für daran anschließende Forschungsarbeiten oder Ähnliches.

Bildet sich die Technologie in diesem Bereich weiter aus, können komplexe Spiele schneller, und simple Spiele effizienter erlernt werden. Dies kann zum Beispiel im Bereich Game-Testing/Qualitätssicherung eingesetzt werden, etwa um Strategien zu finden, welche andere Arten das Spiels zu spielen, überschatten. Mit den daraus gewonnenen Erkenntnissen kann das Spiel ausbalanciert werden, sodass dem Spieler eine größere Anzahl an möglichen Spielstilen angeboten wird, welche im Spiel gut funktionieren. Führt man diesen Gedanken weiter, kann man auch dieses Ausbalancieren automatisieren und durch eine KI realisieren lassen, um so Game Designer zu unterstützen oder sie sogar zu ersetzen.

Man sieht, die Zukunft von künstlicher Intelligenz in Medien, und insbesondere in Videospielen, ist sowohl vielversprechend als auch sehr ungewiss. Man kann jedoch mit ziemlicher Sicherheit sagen, dass die Forschung in der künstlichen Intelligenz



großen Einfluss auf viele, wenn nicht alle Medien haben wird, die derzeit eine tragende Rolle in unserem Alltag spielen.



## Literaturverzeichnis

Atkeson, C. G.; Santamaria, J. C. (1997): A comparison of direct and model-based reinforcement learning. In: Proceedings of International Conference on Robotics and Automation. International Conference on Robotics and Automation. Albuquerque, NM, USA, 20-25 April 1997: IEEE, S. 3557–3564.

Bäck, Thomas (1996): Evolutionary algorithms in theory and practice. Evolution strategies, evolutionary programming, genetic algorithms. New York: Oxford Univ. Press.

Baker, Bowen; Akkaya, Ilge; Zhokhov, Peter; Huizinga, Joost; Tang, Jie; Ecoffet, Adrien et al. (2022): Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos. Online verfügbar unter <http://arxiv.org/pdf/2206.11795v1>.

Baker, Chris (2010): Nimrod, the World's First Gaming Computer. Hg. v. Wired. Online verfügbar unter <https://www.wired.com/2010/06/replay/>, zuletzt geprüft am 18.07.2023.

Barthel, Julia; Ciesielski, Rebecca (2023): Regeln zu ChatGPT an Unis oft unklar. tagesschau. Online verfügbar unter <https://www.tagesschau.de/wissen/technologie/ki-chatgpt-uni-wissenschaft-101.html>, zuletzt aktualisiert am 15.05.2023, zuletzt geprüft am 31.05.2023.

Bengio, Yoshua; Courville, Aaron; Vincent, Pascal (2013): Representation learning: a review and new perspectives. In: *IEEE transactions on pattern analysis and machine intelligence* 35 (8), S. 1798–1828. DOI: 10.1109/TPAMI.2013.50.

Billings, Darse; Davidson, Aaron; Schaeffer, Jonathan; Szafron, Duane (2002): The challenge of poker. In: *Artificial Intelligence* 134 (1-2), S. 201–240. DOI: 10.1016/S0004-3702(01)00130-8.

Bitkom Research (2023): Die Zukunft der Consumer Technology 2022. Online verfügbar unter [https://www.bitkom.org/sites/main/files/2022-08/220823\\_CT\\_Studie\\_2022.pdf](https://www.bitkom.org/sites/main/files/2022-08/220823_CT_Studie_2022.pdf), zuletzt geprüft am 10.08.2023.

Bouton, Charles L. (1901): Nim, A Game with a Complete Mathematical Theory. In: *The Annals of Mathematics* 3 (1/4), S. 35. DOI: 10.2307/1967631.

Brittain, Blake (2023): AI-created images lose U.S. copyrights in test for new technology. Hg. v. Reuters. Online verfügbar unter <https://www.reuters.com/legal/ai->

created-images-lose-us-copyrights-test-new-technology-2023-02-22/, zuletzt aktualisiert am 23.02.2023, zuletzt geprüft am 19.07.2023.

Broderick, Ryan (2023): AI can't replace humans yet — but if the WGA writers don't win, it might not matter. Hg. v. polygon.com. Online verfügbar unter <https://www.polygon.com/23742770/ai-writers-strike-chat-gpt-explained>, zuletzt aktualisiert am 31.05.2023, zuletzt geprüft am 19.07.2023.

Bubeck, Sébastien; Chandrasekaran, Varun; Eldan, Ronen; Gehrke, Johannes; Horvitz, Eric; Kamar, Ece et al. (2023): Sparks of Artificial General Intelligence: Early experiments with GPT-4. Online verfügbar unter <https://arxiv.org/pdf/2303.12712>, zuletzt geprüft am 08.06.2023.

Cass, Stephen (2002): Mind games [computer game AI]. In: *IEEE Spectr.* 39 (12), S. 40–44. DOI: 10.1109/MSPEC.2002.1088444.

D'Anastasio, Cecilia (2023): Meet Neuro-sama, the AI Twitch Streamer Who Plays Minecraft, Sings Karaoke, Loves Art. Hg. v. Bloomberg.com. Online verfügbar unter <https://www.bloomberg.com/news/newsletters/2023-06-16/neuro-sama-an-ai-twitch-influencer-plays-minecraft-sings-karaoke-loves-art>, zuletzt aktualisiert am 16.06.2023, zuletzt geprüft am 18.07.2023.

David, Eli; van den Herik, H. Jaap; Koppel, Moshe; Netanyahu, Nathan S. (2014): Genetic Algorithms for Evolving Computer Chess Programs. In: *IEEE Trans. Evol. Computat.* 18 (5), S. 779–789. DOI: 10.1109/TEVC.2013.2285111.

Diaz, Ana (2023): AI Seinfeld is taking over Twitch. polygon.com. Online verfügbar unter <https://www.polygon.com/23582937/ai-seinfeld-twitch-stream>, zuletzt geprüft am 28.03.2023.

Durst, David; Taylor, Carly (2023): Reversing Anti-Cheat's Detection-Generation Cycle With Configurable Hallucinations. Hg. v. Activision, zuletzt aktualisiert am 30.03.2023, zuletzt geprüft am 10.08.2023.

Eudaly, Zack (2023): What is AI SpongeBob on Twitch? Parody stream explored. Hg. v. Sportskeeda. Online verfügbar unter <https://www.sportskeeda.com/esports/what-ai-spongebob-twitch-parody-stream-explored>, zuletzt aktualisiert am 05.06.2023, zuletzt geprüft am 19.07.2023.

Europäische Kommission (25.04.2018): MITTEILUNG DER KOMMISSION AN DAS EUROPÄISCHE PARLAMENT, DEN EUROPÄISCHEN RAT, DEN RAT, DEN EUROPÄISCHEN WIRTSCHAFTS- UND SOZIALAUSSCHUSS UND DEN AUSSCHUSS DER REGIONEN. Online verfügbar unter <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=COM%3A2018%3A237%3AFIN>, zuletzt geprüft am 09.08.2023.

Fogel, David B. (2001): *Blondie24. The amazing story of how a computer taught herself to win at checkers*. San Francisco, Calif., London: Morgan Kaufmann; International Thomson.

Garcia, Jayzle (2023): AI VTuber Neuro Sama Beats Top Players in Osu. Hg. v. ClutchPoints. Online verfügbar unter <https://clutchpoints.com/ai-vtuber-neuro-sama-beats-top-players-in-osu>, zuletzt aktualisiert am 06.02.2023, zuletzt geprüft am 18.07.2023.

García, Javier; Fernández, Fernando (2015): A comprehensive survey on safe reinforcement learning. In: JMLR.org (Hg.): *The Journal of Machine Learning Research* Volume 16. Issue 1, S. 1437–1480. Online verfügbar unter <https://dl.acm.org/doi/10.5555/2789272.2886795>, zuletzt geprüft am 19.07.2023.

Grant, Eugene; Lardner, Rex (1952): The Talk of the Town - It. In: *The New Yorker* 1952, 02.08.1952, S. 18. Online verfügbar unter <https://www.newyorker.com/magazine/1952/08/02/it>, zuletzt geprüft am 17.07.2023.

Hessel, Matteo; Modayil, Joseph; van Hasselt, Hado; Schaul, Tom; Ostrovski, Georg; Dabney, Will et al. (2017): Rainbow: Combining Improvements in Deep Reinforcement Learning. Online verfügbar unter <http://arxiv.org/pdf/1710.02298v1>.

Hladky, Stephen; Bulitko, Vadim (2008): An evaluation of models for predicting opponent positions in first-person shooter video games. In: 2008 IEEE Symposium On Computational Intelligence and Games. 2008 IEEE Symposium On Computational Intelligence and Games (CIG). Perth, Australia, 15.12.2008 - 18.12.2008: IEEE, S. 39–46.

Karunakaran, Dhanoop (2020a): Q-learning: a value-based reinforcement learning algorithm. medium.com. Online verfügbar unter <https://medium.com/intro-to-artificial-intelligence/q-learning-a-value-based-reinforcement-learning-algorithm-272706d835cf>, zuletzt aktualisiert am 17.09.2020, zuletzt geprüft am 10.06.2023.

Karunakaran, Dhanoop (2020b): The Actor-Critic Reinforcement Learning algorithm. medium.com. Online verfügbar unter <https://medium.com/intro-to-artificial-intelligence/the-actor-critic-reinforcement-learning-algorithm-c8095a655c14>, zuletzt aktualisiert am 30.09.2020, zuletzt geprüft am 10.06.2023.

LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey (2015): Deep learning. In: *Nature* 521 (7553), S. 436–444. DOI: 10.1038/nature14539.

Li, Nan; Ma, Lianbo; Yu, Guo; Xue, Bing; Zhang, Mengjie; Jin, Yaochu (2022a): Survey on Evolutionary Deep Learning: Principles, Algorithms, Applications and Open Issues. Online verfügbar unter <http://arxiv.org/pdf/2208.10658v1>.

Li, Zewen; Liu, Fan; Yang, Wenjie; Peng, Shouheng; Zhou, Jun (2022b): A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. In: *IEEE transactions on neural networks and learning systems* 33 (12), S. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827.

Lucas, Simon M.; Kendall, Graham (2006): Evolutionary computation and games. In: *IEEE Comput. Intell. Mag.* 1 (1), S. 10–18. DOI: 10.1109/MCI.2006.1597057.

Luo, Yuping; Xu, Huazhe; Li, Yuanzhi; Tian, Yuandong; Darrell, Trevor; Ma, Tengyu (2018): Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees. Online verfügbar unter <http://arxiv.org/pdf/1807.03858v5>.

Mismatch Media (2023): Startseite von mismatchmedia.com. Hg. v. Mismatch Media. Online verfügbar unter <https://www.mismatchmedia.com/>, zuletzt geprüft am 19.07.2023.

Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Graves, Alex; Antonoglou, Ioannis; Wierstra, Daan; Riedmiller, Martin (2013): Playing Atari with Deep Reinforcement Learning. Online verfügbar unter <https://arxiv.org/pdf/1312.5602>.

Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Rusu, Andrei A.; Veness, Joel; Bellemare, Marc G. et al. (2015): Human-level control through deep reinforcement learning. In: *Nature* 518 (7540), S. 529–533. DOI: 10.1038/nature14236.

Nettleton, David John (1994): Evolutionary algorithms in artificial intelligence: a comparative study through applications. Durham University, Durham. Online verfügbar unter <http://etheses.dur.ac.uk/5951/>, zuletzt geprüft am 16.07.2023.

Neuro-sama (2019): Osu Neural Network after 5 minutes of training. YouTube.com. Online verfügbar unter <https://www.youtube.com/watch?v=nSBqlJu7kYU>, zuletzt aktualisiert am 06.05.2019, zuletzt geprüft am 18.07.2023.

Nilsson, Nils J. (2013): The quest for artificial intelligence. A history of ideas and achievements. Cambridge: Cambridge Univ. Press.

Novak, Matt (2023): AI-Created Images Aren't Protected By Copyright Law According To U.S. Copyright Office. Forbes. Online verfügbar unter <https://www.forbes.com/sites/mattnovak/2023/02/22/ai-created-images-in-new-comic-book-arent-protected-by-copyright-law-according-to-us-copyright-office/>, zuletzt aktualisiert am 22.02.2023, zuletzt geprüft am 31.05.2023.

O'Shea, Keiron; Nash, Ryan (2015): An Introduction to Convolutional Neural Networks. Online verfügbar unter <http://arxiv.org/pdf/1511.08458v2>.

Project Jupyter (2023): Jupyter Notebook Documentation. Hg. v. Project Jupyter. Online verfügbar unter <https://jupyter-notebook.readthedocs.io/en/stable/>, zuletzt aktualisiert am 31.05.2023, zuletzt geprüft am 06.08.2023.

Puterman, Martin L. (1990): Chapter 8 Markov decision processes. In: Stochastic Models, Bd. 2: Elsevier (Handbooks in Operations Research and Management Science), S. 331–434.

Quadir, Abdullah Muhammad; Khder, Moaiad Ahmad (2022): Exploring the Potential of A.I.-Driven Opponents in Video Games. In: 2022 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETISIS). 2022 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETISIS). Manama, Bahrain, 22.06.2022 - 23.06.2022: IEEE, S. 464–469.

Roderick, Melrose; MacGlashan, James; Tellex, Stefanie (2017): Implementing the Deep Q-Network. Online verfügbar unter <https://arxiv.org/pdf/1711.07478>.

Roose, Kevin (2022): An A.I.-Generated Picture Won an Art Prize. Artists Aren't Happy. New York Times. Online verfügbar unter

<https://www.nytimes.com/2022/09/02/technology/ai-artificial-intelligence-artists.html>, zuletzt aktualisiert am 02.09.2022, zuletzt geprüft am 31.05.2023.

Russell, Stuart J.; Norvig, Peter (2016): Artificial intelligence. A modern approach. 3. edition. Global edition. Upper Saddle River: Pearson (Prentice Hall Series in Artificial Intelligence).

Shao, Kun; Tang, Zhentao; Zhu, Yuanheng; Li, Nannan; Zhao, Dongbin (2019): A Survey of Deep Reinforcement Learning in Video Games. Online verfügbar unter <http://arxiv.org/pdf/1912.10944v2>.

Sheikh, Haroon; Prins, Corien; Schrijvers, Erik (2023): Mission AI. The new system technology. Cham, Switzerland: Springer (Research for policy, studies by the Netherlands Council for Government Policy).

Sloss, Andrew N.; Gustafson, Steven (2020): 2019 Evolutionary Algorithms Review. In: Wolfgang Banzhaf, Erik Goodman, Leigh Sheneman, Leonardo Trujillo und Bill Worzel (Hg.): Genetic Programming Theory and Practice XVII. Cham: Springer International Publishing (Genetic and Evolutionary Computation), S. 307–344.

StepMania (2020): StepMania Wiki. Hg. v. [github.com](https://github.com). Online verfügbar unter <https://github.com/stepmania/stepmania/wiki>, zuletzt aktualisiert am 19.06.2020, zuletzt geprüft am 31.07.2023.

Sutton, Richard S.; Barto, Andrew (2020): Reinforcement learning. An introduction. Second edition. Cambridge, Massachusetts, London, England: The MIT Press (Adaptive computation and machine learning).

Švelch, Jaroslav (2020): Should the Monster Play Fair?: Reception of Artificial Intelligence in Alien: Isolation. In: Game Studies (Hg.): Game Studies Volume 20, Issue 2, June 2020. Online verfügbar unter [https://gamestudies.org/2002/articles/jaroslav\\_svelch](https://gamestudies.org/2002/articles/jaroslav_svelch), zuletzt geprüft am 17.07.2023.

Taylor, Josh (2023): Adobe to integrate AI into Photoshop amid fears of job losses and mass faking of images. Hg. v. The Guardian. Online verfügbar unter <https://www.theguardian.com/technology/2023/may/23/adobe-to-integrate-ai-into-photoshop-amid-fears-of-job-losses-and-mass-faking-of-images>, zuletzt aktualisiert am 23.05.2023, zuletzt geprüft am 04.08.2023.



Thompson, Tommy (2017): The Perfect Organism: The AI of Alien: Isolation. Hg. v. Game Developer. Informa PLC Informa UK Limited. Online verfügbar unter <https://www.gamedeveloper.com/design/the-perfect-organism-the-ai-of-alien-isolation>, zuletzt aktualisiert am 31.10.2017, zuletzt geprüft am 17.07.2023.

Tsiaoussidis, Alex (2023): Twitch's leading AI streamer has become so advanced she's actually hosting react streams now. Hg. v. Dot Esports. Online verfügbar unter <https://dotesports.com/streaming/news/twitch-ai-streamer-so-advanced-shes-hosting-react-streams>, zuletzt aktualisiert am 03.02.2023, zuletzt geprüft am 19.07.2023.

TÜV Verband (2021): Sicherheit und Künstliche Intelligenz. Erwartungen, Hoffnungen, Risiken. Hg. v. TÜV Verband. Online verfügbar unter [https://www.tuev-verband.de/?tx\\_epxelo\\_file\[id\]=856779&cHash=1af8a3f0e6c845423fdd637c8dbcd080](https://www.tuev-verband.de/?tx_epxelo_file[id]=856779&cHash=1af8a3f0e6c845423fdd637c8dbcd080), zuletzt geprüft am 10.08.2023.

TwitchMetrics (2023): Info-Seite des Twitch-Accounts "vedal987". Online verfügbar unter <https://www.twitchmetrics.net/c/85498365-vedal987>, zuletzt geprüft am 18.07.2023.

United States Copyright Office (2023): Re: Zarya of the Dawn. Washington DC, 21.02.2023. Brief an Van Lindberg.

Usama, Muhammad; Qadir, Junaid; Raza, Aunn; Arif, Hunain; Yau, Kok-lim Alvin; Elkhatib, Yehia et al. (2019): Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges. In: *IEEE Access* 7, S. 65579–65615. DOI: 10.1109/ACCESS.2019.2916648.

vedal.xyz (2023): "Advice"-Seite des "Neuro-sama"-Entwicklers. Hg. v. Vedal. Online verfügbar unter <https://vedal.xyz/advice/>, zuletzt geprüft am 18.07.2023.

Winslow, Levi (2023): AI-Generated Seinfeld-Like Twitch 'TV Show' Is Peak Absurdity. Hg. v. Kotaku. Online verfügbar unter <https://kotaku.com/twitch-nothing-forever-ai-generated-seinfeld-dalle-show-1850061075>, zuletzt aktualisiert am 01.02.2023, zuletzt geprüft am 19.07.2023.

Writers Guild of America (2023): WGA Negotiations—Status as of May 1, 2023. Hg. v. Writers Guild of America. Online verfügbar unter

[https://www.wgacontract2023.org/uploadedfiles/members/member\\_info/contract-2023/wga\\_proposals.pdf](https://www.wgacontract2023.org/uploadedfiles/members/member_info/contract-2023/wga_proposals.pdf), zuletzt geprüft am 19.07.2023.

Wunder, Michael; Littman, Michael; Babes-Vroman, Monica (2010): Classes of Multiagent Q-learning Dynamics with epsilon-greedy Exploration. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), S. 1167–1174.

Xiang, Chloe (2023): This Virtual Twitch Streamer is Controlled Entirely By AI. VICE. Online verfügbar unter <https://www.vice.com/en/article/pkg98v/this-virtual-twitch-streamer-is-controlled-entirely-by-ai>, zuletzt aktualisiert am 04.01.2023, zuletzt geprüft am 31.05.2023.

## Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Thesis selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel sind angegeben.

Der Einsatz von KI-Anwendungen ist dem betreffenden Thesisteil, der Art sowie dem Umfang nach detailliert benannt.

---

Alexander Reiprich

Furtwangen im Schwarzwald, den \_\_\_\_\_