Bearbeitungsbeginn: [01.03.2023]

Vorgelegt am: [tt.mm.jjjj]

**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*

Inhaltsverzeichnis

[Abstract 3](#_Toc140442423)

[1. Einführung 4](#_Toc140442424)

[1.1. Motivation und Ziel der Arbeit 4](#_Toc140442425)

[1.2. Aufbau der Thesis 5](#_Toc140442426)

[2. Grundlagen 6](#_Toc140442427)

[2.1. Stand der Forschung 6](#_Toc140442428)

[2.2. Computer Vision 7](#_Toc140442429)

[2.3. Machine Learning 7](#_Toc140442430)

[2.3.1. Reinforcement Learning 8](#_Toc140442431)

[2.3.2. Deep Learning 11](#_Toc140442432)

[2.3.3. Evolutionary Algorithms 13](#_Toc140442433)

[3. KIs im Bereich Unterhaltung 16](#_Toc140442434)

[3.1. Videospiele 16](#_Toc140442435)

[3.2. Livestreaming 17](#_Toc140442436)

[4. Entwicklung 17](#_Toc140442437)

[4.1. Spielprinzip, Projektablauf und Anforderungen 17](#_Toc140442438)

[4.2. Umsetzung der verschiedenen Bestandteile 19](#_Toc140442439)

[4.2.1. Aufsetzen eines Environments 19](#_Toc140442440)

[4.2.2. Jupyter Notebook 28](#_Toc140442441)

[4.2.3. Custom Theming 30](#_Toc140442442)

[4.2.4. Tesseract und Template Matching 33](#_Toc140442443)

[4.2.5. Erster Lernversuch 39](#_Toc140442444)

[4.2.6. Neues Rewardsystem 41](#_Toc140442445)

[4.2.7. Datenkomprimierung 47](#_Toc140442446)

[4.3. Bewertung und Einordnung 51](#_Toc140442447)

[5. Fazit und Ausblick 51](#_Toc140442448)

[5.1. Fazit 51](#_Toc140442449)

[5.2. Ausblick 51](#_Toc140442450)

[Literaturverzeichnis 51](#_Toc140442451)

# Abstract

Videospiele bilden einen wichtigen Aspekt der Unterhaltung in der heutigen Gesellschaft. Spieleentwickler:innen verbringen viel Zeit damit, Spiele zu entwickeln und zu testen, um dem Konsument:innen schlussendlich das ansprechendste Produkt zu liefern. Besonders bei Spielen mit einer Online-Multiplayer-Funktionalität ist ein nicht unwichtiger Teil hiervon die Entwicklung von Maßnahmen gegen die Manipulation durch andere Programme. Diese können in großem Maße das Spielerlebnis anderer beeinträchtigen, in dem sie den Nutzenden unfaire Vorteile geben.

Während ein Großteil der sogenannten Cheating-Software direkt auf die Spieldateien zugreift, gibt es auch andere Möglichkeiten, durch ein externes Programm auf das Spiel Einfluss zu nehmen. In dieser Arbeit soll die Technologie der Computer Vision zusammen mit Machine Learning im Hinblick auf das Steuern von Videospielen untersucht werden. Als Untersuchungsgegenstand liegt hierbei das 2001 erschienene Rhythmusspiel „StepMania“ zugrunde, welches extern durch eine künstliche Intelligenz gesteuert werden soll.

Die Arbeit soll in einen theoretischen sowie praktischen Teil untergliedert werden, wobei der Fokus der Arbeit auf den praktischen Teil gerichtet werden soll. Im Theorieteil wird sich mit dem generellen Konzept von Machine Learning und KI im Bezug auf Videospiele auseinandergesetzt. Hierbei soll die Funktionsweise von Lernalgorithmen behandelt werden, und wie diese Art der KI bereits in Spielen angewandt wird. Ebenso wird auf die Anwendung von KI in der Unterhaltungsbranche, spezifisch im Livestreaming-Bereich eingegangen.

Im praktischen Teil der Arbeit soll selbst mithilfe von Computer Vision und einem Lernalgorithmus ein Programm entwickelt werden, welches das bereits angesprochene Spiel „StepMania“ lernen und spielen soll. Die technische Umsetzung soll in Python mit Unterstützung durch verschiedene Libraries wie OpenCV und Ähnliche erfolgen. Das Ergebnis dieses Teils soll ein Programm sein, mit welchem man ein beliebiges Level in „StepMania“ mit 100% Genauigkeit abschließen kann.

# Einführung

## Motivation und Ziel der Arbeit

Künstliche Intelligenz ist derzeit so im Trend wie noch nie. Laut dem Artificial Intelligence Index Report 2022 der Stanford Universität 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 (Zhang et al. 2022). Durch den Hype des Chat-Bots „Chat-GPT“ Anfang 2023, als auch durch das wachsende Interesse an KI-Kunst durch Dienste wie „midjourney“ sind die Fähigkeiten von KI ein wachsendes Thema, nicht nur im Bereich der Informatik, sondern der ganzen Gesellschaft – laut Bayrischem 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).

~~Dabei beschränkt sich das Angebot von KI nicht bloß auf Wissen. Auch „Kunst“ von künstlicher Intelligenz ist mittlerweile möglich – beispielsweise durch die KI midjourney eines amerikanischen Forschungsinstituts. Nach der Eingabe, was für ein Bild erzeugt werden soll, produziert die KI vier unterschiedliche Varianten des gewünschten Motivs, wobei die Qualität stark variiert (Spehr 2023).~~ Mit dieser neuen Art der Generierung von Content kommen natürlich auch Probleme – sowohl moralische, als auch juristische. Nachdem 2022 ein Werk einer KI als Sieger eines Kunstwettbewerbes gekürt wurde, gab es Diskussionen, inwiefern Bilder, welche von KI erzeugt wurden, tatsächlich Kunst, und die Ersteller:innen tatsächlich Künstler:innen sind (Roose 2022). Ebenfalls war es unklar, ob mit KI erstellte Bilder rechtlich unter Copyright geschützt sind. Nutzer:innen behaupten, die eingegebenen Befehle, die zur Kreation des Bildes geführt haben, wären ihr eigenes Werk, was jedoch nach einem Urteil des U.S. Copyright Office nicht der Fall ist (Novak 2023).

Wird der Gedanke, dass Künstliche Intelligenz kreativ wird, weitergeführt, 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 von Reinforcement Learning, einem spezifischen Lernalgorithmus, eingegangen werden soll. Ebenso soll das Thema KIs zur Medienproduktion und -verarbeitung anhand von zwei Beispielen behandelt werden. Im praktischen Teil dieser Arbeit soll selbst eine KI zur Steuerung eines Videospiels entwickelt und trainiert werden. Dabei wird das Vorgehen dokumentiert, der Vorgang und die Herangehensweise beschrieben und erklärt, und schlussendlich retrospektiv analysiert. Während das Ziel dieses Teils zwar ein fertiges Produkt ist, soll der Fokus dabei mehr auf dem Prozess der Entwicklung liegen – welche Probleme aufkommen, wie diese gelöst werden, was es für Alternativen gibt, und wie das Projekt schlussendlich kritisch zu betrachten ist.

## Aufbau der Thesis

Im ersten Kapitel dieser Arbeit geht es um die Thesis an sich. Dabei soll im ersten Unterkapitel zuerst die Motivation dargestellt werden um anschließend den theoretischen und praktischen Teil dieser Ausarbeitung zu erläutern. Im zweiten Unterkapitel wird der Aufbau der Arbeit genauer 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 Medienproduktion und -verarbeitung einzugehen. Hier werden zwei Beispiele analysiert, welche in diesen Bereich 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 chronologisch auf den Entwicklungsprozess einzugehen. Eines dieser Unterkapitel beinhaltet dabei entweder ein Modul bzw. Funktionsweise des Programmes oder einen wichtigen Schritt in der Entwicklung. Abschließend wird das Vorgehen analysiert, bewertet und kritisiert.

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

# Grundlagen

## Stand der Forschung

*Der Stand der Forschung soll kurz den generellen Stand in Bezug auf KIs in allen Bereichen behandeln, und insbesondere auf die aktuelle Entwicklung im Bereich KIs als Spieler von Videospielen eingehen.*

Die Abkürzung „KI“ steht für „künstliche Intelligenz“ – doch was genau versteht man unter diesem Begriff? Eine klare, einheitliche Definition gibt es 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 Hochrangige Expertengruppe für künstliche Intelligenz (AI HLEG) bezeichnet KI als „Systems that display intelligent behaviour by analysing their environment and taking actions – with some degree of autonomy – to achieve specific goals.“ (High-Level Expert Group on Artificial Intelligence 2019). Beide Definitionen sind relativ vage, und könnten genauso auf ein Thermostat zutreffen, welches die Temperatur misst („analysing their environment“) und reguliert („taking actions“) um die eingestellte Temperatur zu erreichen („to achieve specific goals“), wobei viele dies wahrscheinlich nicht unter KI verstehen würden (Sheikh et al. 2023, S. 16).

Durch das Thema „ChatGPT“ und die Möglichkeiten, die das Tool brachte, geriet das Thema KI stark in die Öffentlichkeit. ChatGPT ist aber nur ein Teil einer der größten Fortschritte im Bereich künstliche Intelligenz – OpenAI’s Generative Pretrained Transformer 4, kurz GPT-4, ein sogenanntes Large Language Model (LLM), welches mit einer enormen Anzahl von Daten aus dem Internet trainiert wurde. Das Modell basiert auf dem Konzept, basierend auf Inhalt und dem Kontext eines Satzes, Wörter vorherzusagen, und so Antworten zu diesen Satz zu generieren (Bubeck et al. 2023). Die Möglichkeiten scheinen dabei endlos – Aufgaben wie Gedichte verfassen, mathematische Gleichungen lösen 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). [Maybe Beweis aus dem Paper?]

## Computer Vision

*In diesem Kapitel soll das Konzept der Computer Vision erklärt werden, mit dem in der Umsetzung gearbeitet wird. Dabei werden Themen wie z.B. Pattern Recognition angesprochen.*

## Machine Learning

*Ähnlich zu dem Kapitel 2.2. wird hier Machine Learning erklärt. Es soll eine Übersicht über die Funktionsweise von Künstlicher Intelligenz geben – da es viele unterschiedliche Methoden und Herangehensweisen gibt, sollen sich zwei oder drei rausgesucht werden, welche für unterschiedliche Zwecke genutzt werden (z.B. evolutionary algorithms, reinforcement learning, …). Dabei soll stets auf den praktischen Part eingegangen werden, um so Theorie und Praxis zu verbinden und ein Beispiel zu haben, an dem sich Leser:innen orientieren können.*

Eine künstliche Intelligenz besteht aus zwei Teilen – einem Agent 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, und wie er sie erhält, kann dabei basierend auf der Art der künstlichen Intelligenz stark variieren (Russell und Norvig 2016, S. 34–35).

[BILD, vielleicht 2.1 aus Russel und Norvig 2016]

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, S. 37-38).

Dieses Grundkonzept ist stark runtergebrochen und nicht auf jede KI anwendbar – Modelle, die beispielsweise „unsupervised“, also unbeaufsichtigt, lernen, verwenden Daten ohne vorab definierte Wertung, sodass kein Feedback gegeben wird. In diesem Fall muss sich das Modell selbstständig orientieren und ohne Hilfe Muster oder Strukturen in den unbearbeiteten Datensätze erkennen (Usama et al. 2019, S. 1-2).

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 und verglichen werden. Zum einen wird das Konzept des „reinforcement learning“, also „verstärktes Lernen“ behandelt, welches im praktischen Teil dieser Arbeit genutzt wird. Verglichen wird dieses Konzept mit „evolutionary algorithms“ – Algorithmen, welche nach den biologischen Prinzipien von natürlicher Selektion und Genetik arbeiten, um so durch Abstammung Verhalten zu lernen (Li et al. 2022a, S. 2-5).

### Reinforcement Learning

Reinforcement Learning (RL) beschreibt ein Machine Learning Konzept, bei welchem der Agent durch pures Ausprobieren, „Trial and Error“, lernt. Die ausprobierten Aktionen werden durch einen Reward, also eine Belohnung, bewertet, welches 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 Rewardvergabe, welche zur Folge hat, dass eine Aktion Auswirkungen auf die Rewards der nachfolgenden Aktionen hat. RL lässt sich nicht in unbeaufsichtigtes Lernen, aber auch nicht in das Gegenteil, also 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, S. 1-3).

Eine zentrale Rolle des Reinforcement Learnings spielt der Markov Decision Process (MDP). Der MDP ist eine mathematisches bzw. stochastisches 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, bekommt der Agent also eine Information, 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 das System sich 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, S. 331-333). 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 Zeitspanne erreicht wurde (Shao et al. 2019, S. 2).

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, welche Policy das Modell zum Lernen nutzt. Bei Off-Policy Algorithmen ist die Policy, welche das Verhalten bestimmt, eine andere als die, die zum Lernen genutzt und angepasst wird. Hier wird auf eine vorherige Version zurückgegriffen, beispielsweise von einem früheren Lernversuch. On-Policy Algorithmen nutzen nur eine Policy für sowohl das Verhalten als auch das Lernen. Hier wird die Strategie, also das Verhalten des Modells während des Lernprozesses, mit den Erfahrungen der aktuellen Entscheidungen aktualisiert (Shao et al. 2019, S. 2).

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, S. 3557-3560; Luo et al. 2018, S. 1-2).

Während in model-based Ansätzen sowohl eine Wertefunktion als auch eine Policy existieren, wird innerhalb der model-free Ansätze noch zwischen value-based und policy-based RL-Ansätzen unterschieden. Der Unterschied liegt hier in der Optimierung der Entscheidungsstrategie des Agents. Während value-based Algorithmen sich auf die Optimierung der Wertefunktion, mit welcher der höchste Reward in einer Situation berechnet wird, fokussieren, wird bei einem policy-based Algorithmus die Policy direkt angepasst. Der Zwischenschritt über die Wertefunktion wird dabei übersprungen, und die Entscheidungsstrategie selbst wird geändert, was bei einem value-based Ansatz nicht passiert. Dort basieren die Entscheidungen nicht auf der Policy, sondern auf der Wertefunktion – die Policy ergibt sich nur implizit aus den Werten dieser Funktion (Karunakaran 2020a ; Shao et al. 2019, S. 3-5).

Erwähnenswert ist dabei, dass beide Konzepte kombiniert werden können, wie beispielsweise im Actor-Critic-Ansatz. Hier gibt es zwei Netze, den Actor und den Critic. Der Actor entscheidet sich für eine Aktion, welche dann von dem Critic bewertet wird. So wird intern Feedback gegeben. Dabei basiert der Actor auf einem policy-based Ansatz, während der Critic die Aktionen des Actors mit einer Wertefunktion evaluiert, wie bei einem value-based Algorithmus (Karunakaran 2020b).

### Deep Learning

Die Einsatzmöglichkeiten von klassischen Machine Learning Methoden sind jedoch begrenzt. Im Bereich der Verarbeitung von natürlichen, rohen Datensätzen stoßen konventionelle Systeme, welche von Hand auf ihre Anwendungsfälle spezialisiert wurden, schnell an ihre Grenzen (LeCun et al. 2015, S. 436). Sogenanntes „Representation learning“, also das Lernen, was die einzelnen Daten repräsentieren, hilft dabei dem Programm, auch komplexere Datenstrukturen zu verstehen. Kombiniert wird dies unter anderem mit Deep Learning Methoden, um das „representation learning“ weiter zu verbessern (Bengio et al. 2013, S. 1798). Deep Learning beschreibt in diesem Kontext, dass diese Funktionen, welche die Repräsentationen erarbeiten, in Layer, also in Schichten, aufgeteilt sind. Dabei wird die Repräsentation aus dem rohen Input genommen, und eine neue, abstraktere Repräsentation erstellt. Diese wird dann an den nächsten Layer gegeben, und weiter abstrahiert. Nach genug Transformationen ist es dem Modell möglich, mit Hilfe der Abstraktion komplexe Funktionen zu erstellen, welche mit den Daten besser umgehen können. Der französische Turing-Award Gewinner Yann LeCun beschreibt das Konzept in 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, S. 436).

Die neuralen Netze, welche so aufgebaut werden, können in verschiedene Kategorien aufgeteilt werden. Relevant im Kontext der Arbeit ist das sogenannte „Convolutional neural network“ (CNN), also ein gefaltetes neurales Netz. Dieses ist inspiriert von der Biologie, in welcher das Gehirn eines Lebewesens gefaltet ist, da so in einem begrenzten Raum mehr Gehirnfläche erreicht werden kann (Kinney und Volpe 2018, S. 148).

Die Architektur von CNNs ist von der menschlichen Wahrnehmung inspiriert (Li et al. 2022b, S. 7000). 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, S. 2-3). In dem Anwendungsfall dieser Arbeit wird ein CNN genutzt, da es sich beim Input, welchen die KI erhält, um ein Bild des Spiels 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 wurde 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 eine Funktion, welche den erwarten Reward definiert, wenn in einem beliebigen Zustand eine beliebige Aktion ausgeführt wird. Dabei wird diese Funktion während des Lernprozess an die Geschehnisse angepasst, und lernt so die Zusammenhänge von Zustand, Aktion und Reward. Endresultat ist eine optimale Version dieser Q-Funktion, welche zu jedem Zustand die Aktion bestimmen kann, welche den höchsten Reward erzeugt (Roderick et al. 2017, S. 1–2).

Die Besonderheit bei einem DQN im Bezug auf die 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 bekommen hat, genauer anpassen zu können. Hier wird sogenanntes „Experience Replay“ genutzt – das Modell sammelt Erfahrungen über eine kurze Anzahl an Steps, wählt dann zufällig aus diesen Erfahrungen aus, und passt die Q-Funktion dann basierend darauf an. Dieses zufällige Auswählen wird verwendet, um die Erfahrungen extern vom Environment zu betrachten, um einen Bias zu vermeiden, welcher die Q-Funktion negativ beeinflussen könnte. Dieses Deep Q-Learning beschreibt die Exploitation Policy des DQNs. (Roderick et al. 2017, S. 2–3).

Die Exploration Policy eines DQN wird durch eine Epsilon-Greedy-Strategie realisiert. 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 die Anzahl der möglichen Aktionen, zwischen denen der Agent wählen kann, und einen vordefinierten Wert zwischen 0 und 1, die sogenannte Exploration-Rate. Die Aktion, welche rechnerisch im momentanen Zustand den höchsten Reward bringt, wird mit der Wahrscheinlichkeit ausgewählt, während die übrigen Aktionen jeweils eine Wahrscheinlichkeit von besitzen. Konkret heißt das, dass die Exploration-Rate davon abhängig ist, mit welcher Wahrscheinlichkeit das Modell Neues probiert, oder es auf das bereits gelernte zurückgreift – geht die Exploration-Rate, also Epsilon, 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).

Um die Erkenntnisse aus diesem und dem letzten Kapitel in die Praxis zu übertragen, kann man folgende Schlüsse ziehen: 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.

### Evolutionary Algorithms

Reinforcement Learning ist nicht der einzige Weg, wie eine KI trainiert werden kann. Eine von vielen Alternativen sind sogenannte Evolutionary Algorithms. Diese Algorithmen basieren auf Modellen der natürlichen, biologischen Evolution nach Darwin (Bäck 1996, S. 8).

Hier wird jedoch zwischen zwei Perspektiven von Evolution unterscheiden. 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. In dem Buch „Genetic Programming Theory and Practice XVII“ beschrieben 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 an Hand 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 Mutation bestimmt. Daraufhin beginnt der Prozess erneut und eine neue Generation wird erstellt. Dies stellt das Grundkonzept von Evolutionsalgorithmen da – es gibt unterschiedliche Varianten zu diesem Konzept, die auf diesem aufbauen (Sloss und Gustafson 2020, S. 314).

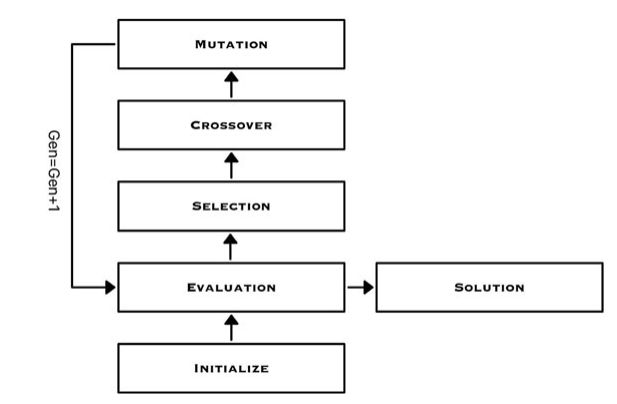


Abb. - Prozess der Evolution (Sloss und Gustafson 2020, S.314)

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. Anstatt wie 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 den menschlichen Spieler spaßigere Gegner darzustellen (Lucas und Kendall 2006, S. 11). Ein Beispiel für die Entwicklung einer KI mit einem Evolutionary Algorithms ist „Blondie24“ – eine KI, welche ohne Erklärung das Brettspiel Dame gelernt hat, und mit diesem Wissen Turniere gewonnen hat (Fogel 2001, S. 13-14; Lucas und Kendall 2006, S. 11). Hier wurde der KI nicht erklärt, wie das Spiel genau funktioniert – es wurden bloß Informationen über die Anzahl, die Position und die Art 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 zählt 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 Evolutionary Algorithms ein interessanter Anwendungsfall, wie Darse Billings et al. der Universität Alberta in ihrem 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 Brettspiele, und kombiniert mit der Komponente, in Echtzeit auf die Geschehnisse reagieren zu können, stellen damit sowohl eine größere Herausforderung für den menschlichen Spieler, als auch 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 da, die benötigt wird, um Generationen fortzuschreiten. Um dieses Problem besser dazustellen, kann man hier das Beispiel eines Schachcomputers aufführen – Sollte es pro Generation 100 Individuen geben, welche 10 Sekunden pro Spiel bekommen, und jedes Individuum einmal gegen jedes Individuum spielen sollte, käme man pro Generation auf 825 Minuten, was für 100 Generationen über 57 Tage dauern würde (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 ein Chart spielen, was, abgerundet und stark geschätzt, eine Minute dauert. 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. Dieser evolutionäre Ansatz würde also nur sinnvoll sein, wenn sich das Können der KI pro Generation so verbessert, dass es zeitlich schneller geht, als das Reinforcement Learning in der selben Zeit tut, was unwahrscheinlich ist.

# KIs im Bereich Unterhaltung

## Videospiele

*Hier soll hauptsächlich auf die Neuro-Sama KI eingegangen werden, sowie weitere Projekte, welche mithilfe von Lernalgorithmen Videospiele steuern. Dabei soll gezeigt werden, welche Technik dahinter steckt – das Wissen aus Kapitel 2.3. soll hier wieder aufgegriffen werden.*

## Livestreaming

*Während in Kapitel 3.1. nur auf den „Spiel-Teil“ von Neuro-Sama eingegangen werden soll, wird hier nochmal explizit darauf eingegangen, dass Neuro-Sama auf Nachrichten der Zuschauer eingeht, und ebenfalls auf die Geschehnisse auf dem Bildschirm reagiert. Ebenso soll auf das Projekt „Nothing, Forever“ eingegangen werden – ein Twitch-Kanal, welcher kontinuierlich eine fiktive Episode der amerikanischen Sitcom „Seinfeld“ streamt, die komplett KI generiert ist.*

# Entwicklung

*In der Umsetzung soll ein Computerprogramm konzipiert und umgesetzt werden, welches mit Hilfe eines Lernalgorithmus ein simples Rhythmusspiel erlernt und spielt.*

## Spielprinzip, Projektablauf und Anforderungen

*Hier sollen die Anforderungen an das fertige Produkt dargestellt werden. Ebenso soll das Spielprinzip erklärt werden, und welche Herausforderungen sich dabei präsentieren.*

StepMania 5 ist sowohl ein Open-Source Rythmusspiel, 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. [Quelle hinzufügen]

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 von unten nach oben scrollen 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, je nach Genauigkeit des Treffzeitpunktes, Punkte. Hat er den Chart zu Ende gespielt, folgt der Resultscreen, wo der Spieler eine Note bekommt, welche auf der Anzahl und der Genauigkeit der getroffenen Pfeile basiert.



Abb. - Screenshot vom Gameplay von StepMania 5

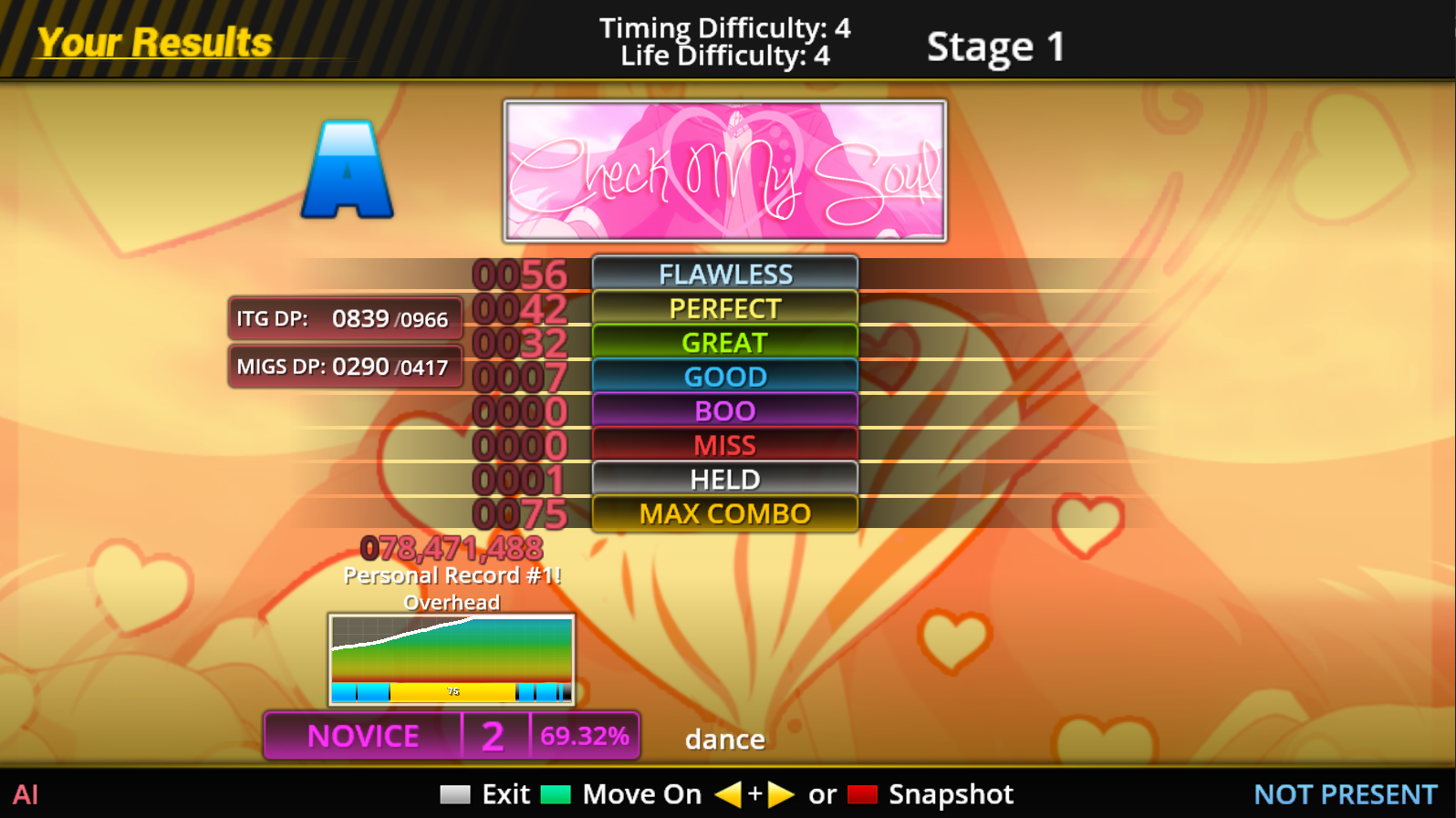


Abb. - Screenshot des Resultscreens nach Abschluss eines Charts

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, und dies den Umfang dieser Thesis übersteigt. Vielmehr geht es um den Prozess des Entwickelns einer solchen KI, welche Probleme dabei auftreten, wie sie gelöst werden können, und wie sich ein Modell nach kurzen Trainingseinheiten verhält. Ein genereller Lernerfolg ist das Ziel, kein perfektes Modell.

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. Die Kapitel sind in chronologischer Reihenfolge aufgebaut, sodass es sein kann, dass Elemente in einem Kapitel eingeführt, aber in einem späteren Kapitel überarbeitet oder entfernt werden.

## Umsetzung der verschiedenen Bestandteile

### 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 soll der Aufbau und die Entwicklung des Environments für diesen Anwendungsfall beschrieben werden. Für die Umsetzung des Environments wurde das Python-Modul „gym“ genutzt, welches von OpenAI entwickelt worden ist.

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 essentiellen Rückgabewerte eines Steps werden also 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 hätte erzeugen müssen.

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

Ebenfalls essentiell 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 16 diskreten Zahlen – 0 bis 15. Jede Zahl steht dabei für eine Kombination an Tastendrücken. Um diese Tastendrücke für Menschen verständlicher zu machen, wurden die Zahlen in das Binärsystem umgewandelt. Zahlen, welche normalerweise mit weniger als 4 Ziffern im Binärsystem dargestellt werden, wurden von links mit Nullen auf 4 Ziffern aufgestockt. Dies hat den Sinn, dass nun die vier Ziffern den Zustand der vier Pfeilrichtungen darstellen können. So wurde beispielsweise aus der Zahl 5 im Base10-System die Zahl 0101. Im Hinblick auf das Spiel kann man nun die Ziffern von links nach rechts einer Pfeilrichtung zuordnen, wie sie auch im Spiel vorkommen. Ein Input von 0101 würde bedeuten, dass der zweite und der vierte Pfeil gedrückt werden, der erste und dritte jedoch nicht. Dieses Problem wäre einfacher mit einem Array umzusetzen, wodurch man sich die Umrechnung in das Binärsystem hätte sparen können – hier hätte man allerdings den Action Space als MultiDiscrete definieren müssen, was nicht in Kombination mit dem DQN-Modell funktioniert. Den Ansatz, ein anderes Modell mit einem anderen Action Space zu nutzen, wäre eine Option, wurde im Rahmen dieser Arbeit aber nicht weiter verfolgt.

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 bestimmen 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\_\_()  
 # Defined 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)

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 Startzustands zurück, welche dann vom Model genutzt wird, um eine Aktion zu bestimmen, die es basierend auf der Obseration 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 Model 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 demensprechend 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.

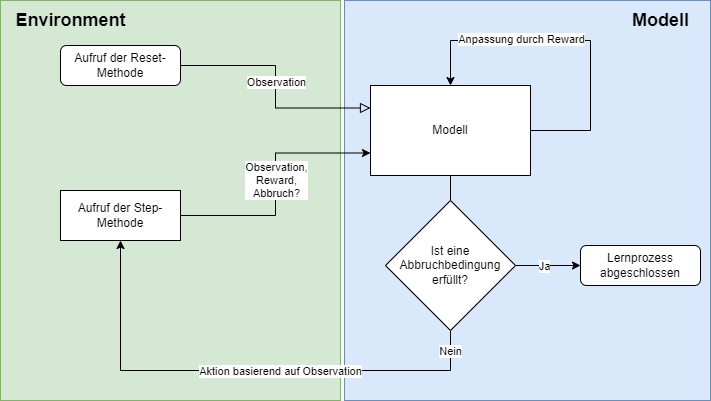


Abb. - Ablaufdiagramm des Lernprozesses, Teile im Environment und Teile im Modell sind farblich unterteilt

Die Step-Methode führt zuerst die Aktion aus, welche sie als Parameter bekommen hat. Daraufhin wird ein Screenshot des gesamten Fensters gemacht, und runterskaliert, um die Größe des Bildes für die Observation zu reduzieren. Mit Hilfe 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 and send 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.downscaleImage(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

Nach jedem abgeschlossenen Song 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 Song navigieren. Abschließend wird ein Screenshot gemacht, welcher an das Model 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)

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, welcher 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 Rewardverteilung 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

**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)

**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 accidently 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

### 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 und sparen so Zeit. [Quelle hinzufügen]

In diesem Fall wurde sich für ein Jupyter Notebook entschieden, da das selektive Ausführen von Code, als auch das direkte Darstellen der aufgenommenen Bereiche und das Erstellen von Graphen mit Hilfe von matplotlib den Entwicklungsprozess erleichtert hat.

Jupyter Notebooks sind in sogenannte Code- oder 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. Dieses Beispiel zeigt nur einen Anwendungsfall, welcher des Beispiels wegen stark runtergebrochen wurde.

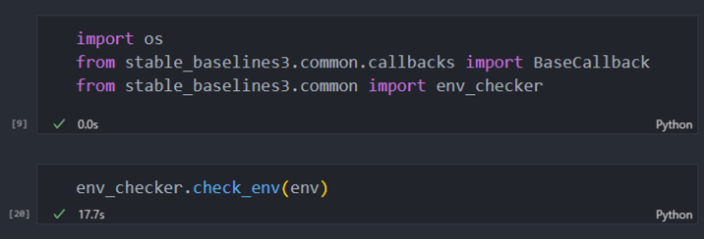


Abb. - Zwei Code-Cells eines Jupyter Notebooks innerhalb der Entwicklungsumgebung Visual Studio Code

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 eigene Python-Datei genutzt, wäre dieser Schritt einfacher.

### Custom Theming

Informationen auf dem Bildschirm sind ein wichtiges Thema. Diese können für den Spieler zwar wichtig 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 die Pattern Recognition damit umgeht.

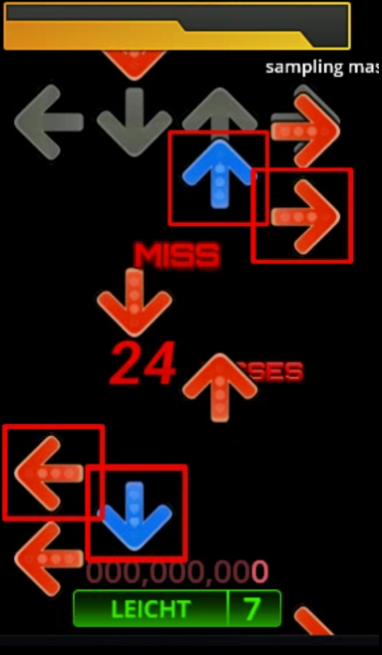


Abb. - Screenshots des Gameplays, bei welchem die erkannten Elemente markiert wurden

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 nichterkannten 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, welche 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 spielererstellte Veränderungen der Benutzeroberfläche. Verändern kann man diese durch das Hinzufügen und Abändern von .ini und .lua-Dateien im Spielverzeichnis, wobei in diesem Anwendungsfall ein paar Zeilen in der metrics.ini-Datei, die Datei, die für den Großteil von Themeänderungen zuständig ist, sowie eine Änderung von zwei Variablen in der entsprechenden .lua-Datei den gewünschten Effekt erzeugte.

In der metrics.ini-Datei wurde die Score-Anzeige um einen gewissen Pixelwert nach rechts verschoben, sodass sie 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.

Eine Veränderung der .lua-Dateien wird vom Entwickler nicht empfohlen, jedoch konnte das bereits angesprochene Judgement anders nicht verändert werden. [Quelle hinzufügen] Da ein generelles Ausblenden nicht möglich ist, wurde die Anzeige um einen hohen Wert außerhalb des dargestellten Bereichs geschoben.

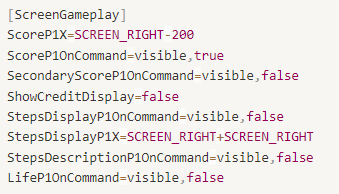


Abb. - metrics.ini - Hier wurde alles auf unsichtbar gestellt, was nicht benötigt wird. Ebenfalls wurde die Score-Anzeige an den rechten Rand des Fensters gesetzt und mit einem Offset in die Mitte des leeren Bereiches rechts neben dem Spielfeld geschoben

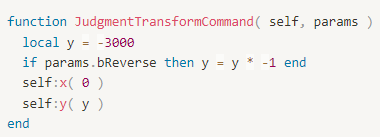


Abb. - Other.lua - Der Y-Wert wurde so verändert, dass sich das Judgement außerhalb des dargestellten Bereichs befindet

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

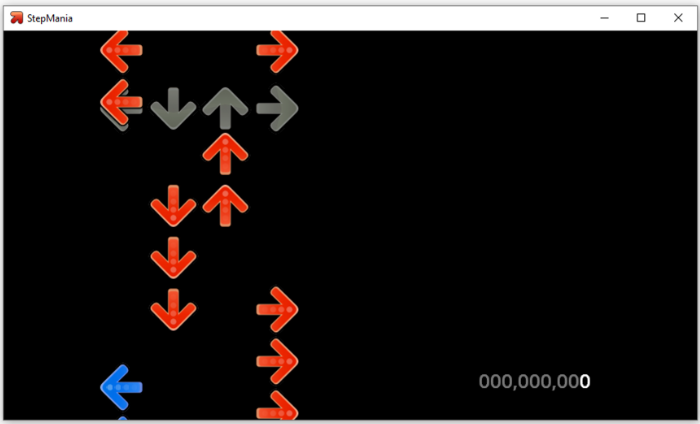


Abb. - User Interface nach den Änderungen am Theme

### Tesseract und Template Matching

Wie in Kapitel 4.2.x. bereits angesprochen, werden für ein zuverlässiges StepMania-Environment Lösungen für zwei Probleme benötigt - Man braucht ein verlässliches Reward-System, welches den vorangegangenen Input bewertet, und man benötigt eine Funktion, welche erkennt, ob ein Song vorüber ist, welche dann den nächsten Song auswählt um das Spielen fortzuführen.

In der ersten Version des Environments wurde dafür 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 das Netz genutzt werden. Wurde nichts erkannt, beispielsweise, weil beim Start eines Songs keine Punktzahl zu sehen ist, wird 0 zurückgegeben.

**def** **get\_number\_from\_image**(self, img):  
 analysed\_string = pytesseract.image\_to\_string(img, config='-c tessedit\_char\_whitelist=0123456789')  
 **if** analysed\_string == '':  
 analysed\_string = 0  
 **return** int(analysed\_string)



Abb. - 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 image\_to\_string()-Funktion von Tesseract genutzt, jedoch in einem anderen Kontext.

In diesem Bild sieht man den Ergebnisscreen eines Songs.

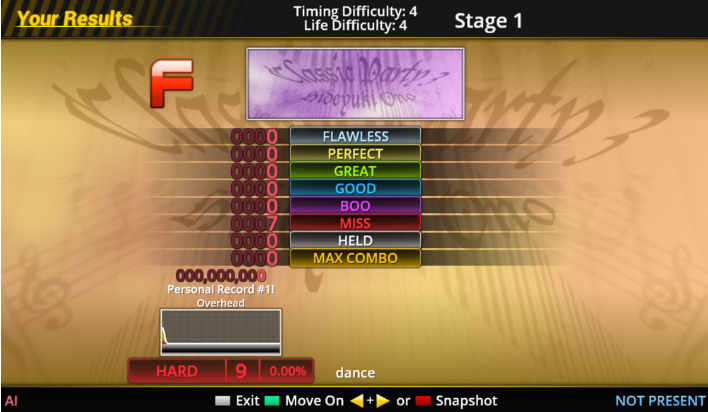


Abb. - Ergebnisscreen eines Charts

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 an Input bekommen, und sollte nun die Wörter “Your Results” erkennen. Wenn dies der Fall ist, wird eine Reihe von Inputs ausgeführt, welche den Screen verlassen und den nächsten Song auswählen.

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 ein Umformatieren 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.

**def** **is\_text\_in\_image**(self, img, text, chars):  
 done = False  
 res = pytesseract.image\_to\_string(img, lang='eng', config='--psm 7')[:chars]  
 **if** res **in** text:  
 done = True  
 **return** done

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.

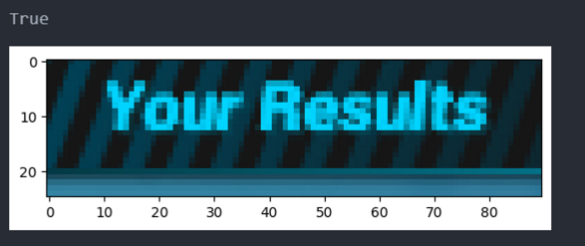


Abb. - Ergebnis des genannten Funktionsaufrufs (oben) und die dazu gehörende Observation (unten)

Während das Nutzen 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 den beiden Tesseract-Aufrufen eingenommen, da diese auf ein zusätzliches, großes 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):  
 img\_gray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY)  
 threshold = 0.9  
 res = cv.matchTemplate(img\_gray, self.template\_results, cv.TM\_CCOEFF\_NORMED)  
 **if** (res >= threshold).any():  
 **return** True  
 **return** False

Diese Änderung führte dazu, dass sich die durchschnittlichen Bilder pro Sekunde von 2,35 auf 3,73 verbessert haben.

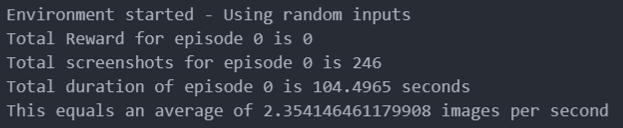


Abb. - Ergebnis des Benchmarktests mit Tesseract

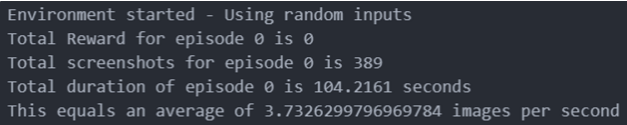


Abb. - Ergebnis des Benchmarktests mit Template Matching

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))

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 Rewardverteilung führen, was den Lernerfolg des Agents einschränken würde.

Durch das Anpassen des Thresholds, als auch der Auflösung des Spiels, konnte man jedoch erreichen, dass nach 10 Tests mit je 5 Episoden alle Zahlen richtig erkannt wurden. Dieses Verfahren ersetzte danach die Tesseract-Erkennung, und sorgte für eine weitere Performancesteigerung von 3.73 auf 4.59 Bilder pro Sekunde.

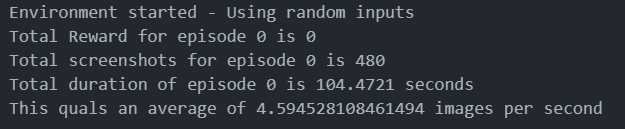


Abb. - Ergebnis des Benchmarktests nach beiden Änderungen

### Erster Lernversuch

Nachdem das Environment und der Agent aufgesetzt und die Performance optimiert wurde, wurde der erste Lernversuch unternommen. Ziel war, diese Fortschritte in mehreren Hinsichten zu analysieren. Zuerst war es wichtig zu wissen, ob es generell Fortschritt gibt. Während des Austestens des Modells war noch kein Wissen vorhanden, weshalb jeder Input zufällig bestimmt, und nur manchmal tatsächlich brauchbare Tastendrücke ausgegeben wurden. Falls nach dieser, für diese Aufgabe doch recht kurze, Zeitspanne bereits kleine Veränderungen zu sehen waren, heißt dies, dass nun nur weiter trainiert werden muss, um das Endziel zu erreichen. Ebenfalls wurde diese Trainingseinheit dafür genutzt, um zu schauen, wie sich die Hyperparameter über einen längeren Zeitraum entwickeln, und wie man die Ergebnisse aus dem Trainingsdurchlauf nutzen kann, um diese Parameter effizienter zu gestalten.

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.

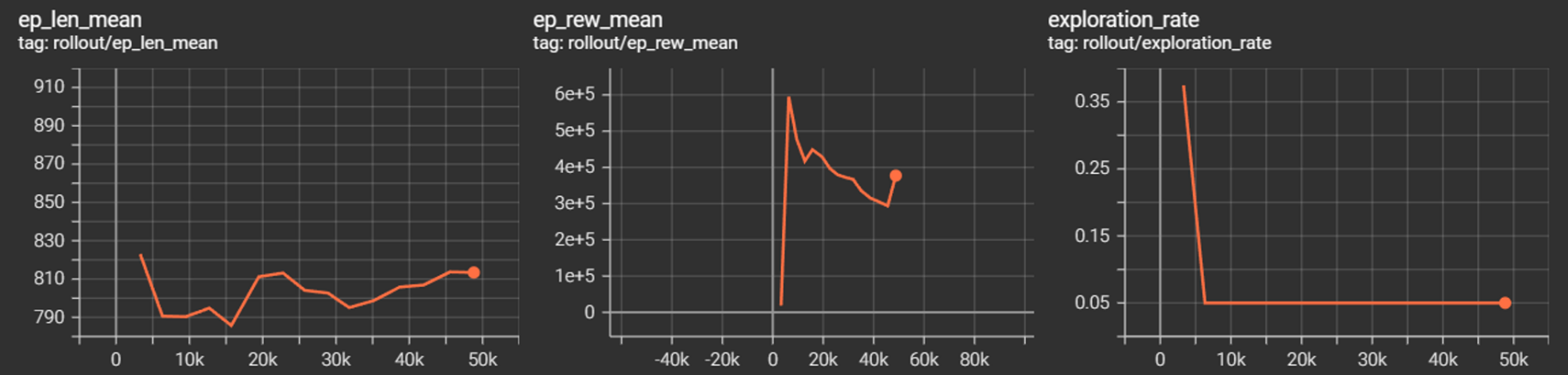


Abb. - Screenshot von Tensorboard - Darstellung der Mittelwerte der Episodenlänge, des Rewards, sowie der Wert der Exploration Rate des ersten Lernversuchs über die Lerndauer

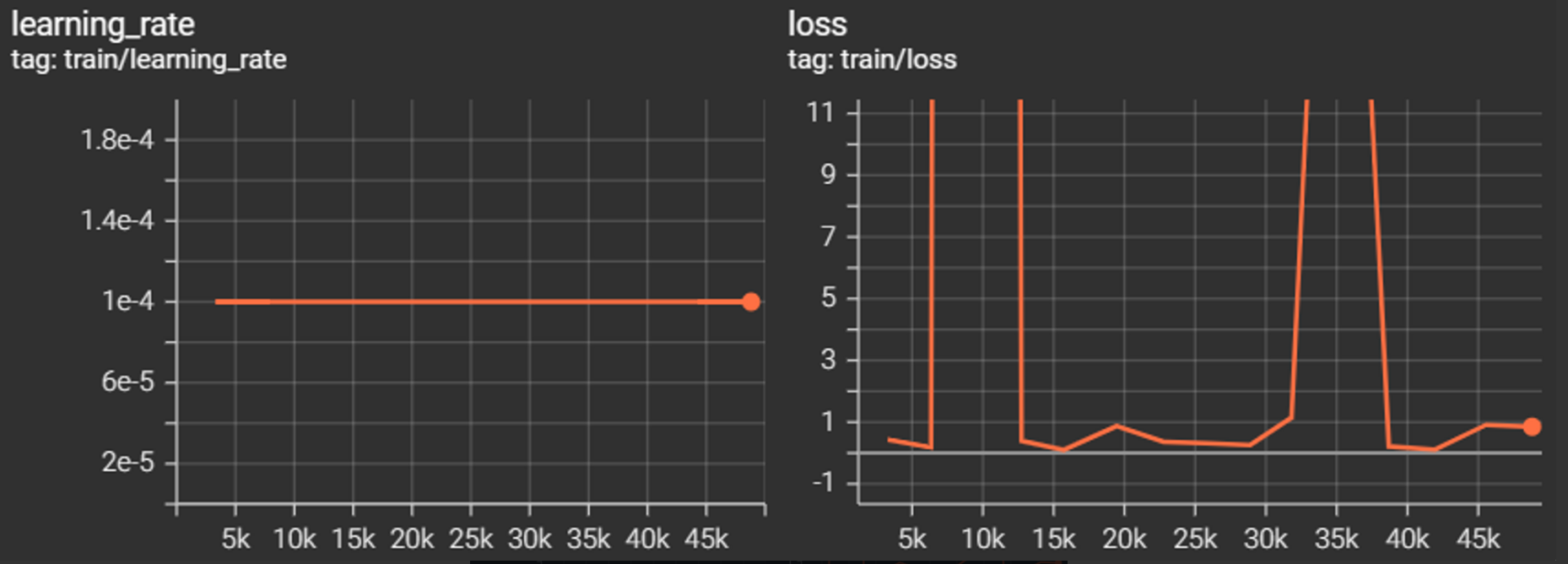


Abb. - Screenshot von Tensorboard - Darstellung der Learning Rate und der Loss Rate des ersten Lernversuchs über die Lerndauer

Auf den Grafiken, welche mit Hilfe 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. Gleichzeitig sieht man, dass die Exploration Rate nach 6.000 Steps bei 0.05 blieb, die Learning Rate sich konstant bei 0.005 befand, und die Loss-Funktion zwar niedrig war, sich vom Wert her jedoch wenig veränderte, bis auf zwei große Ausreißer. Auf die ersten beiden Punkte komme ich gleich zurück, zuerst möchte ich jedoch zeigen, wie das trainierte Model nach 50.000 Steps das Spiel spielte.

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

|  |  |
| --- | --- |
| **Taste** | **Anzahl der Inputs** |
| Oben | 5921 |
| Rechts | 0 |
| Unten | 0 |
| Links | 3 |

Man muss hier nicht weiter erklären, dass es eindeutige Probleme in dem Lernverhalten der KI gibt, da der Fokus auf eine Taste nicht das Ziel dieses Vorhabens ist. 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. Das erste Mal, wo eine Punktzahl durch rechtzeitiges Drücken der richtigen Taste erreicht wurde, scheint Grund genug gewesen ist, sich auf diese eine Taste zu versteifen, und alle anderen zu ignorieren.

Um zu testen, ob dieses Problem tatsächlich bereits nach kurzer Lerndauer auftrat, oder ob es sich stetig darauf hingearbeitet hat, habe ich die gleiche Grafik für zwei neu-trainierte Models nach 5.000, 10.000 und 25.000 Steps angefertigt.

|  |  |  |  |
| --- | --- | --- | --- |
| **Taste** | **Anzahl der Inputs, 5.000 Steps** | **Anzahl der Inputs, 10.000 Steps** | **Anzahl der Inputs, 25.000 Steps** |
| Oben | 4872 | 9629 | 24.091 |
| Rechts | 3 | 0 | 0 |
| Unten | 0 | 0 | 0 |
| Links | 0 | 0 | 0 |

Man kann erkennen, dass dieses Problem also fundamental an der Umsetzung des Reward-Systems liegt. Da die hier genutzten Hyperparameter für die Exploration Rate und die Learning Rate die Standardwerte für DQN-Models 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, sowohl das Reward-System, als auch die Hyperparameter entsprechend anzupassen, damit die KI nicht dazu verleitet wird, nur eine Taste zu drücken.

### Neues Rewardsystem

Nachdem im letzten Kapitel festgestellt wurde, dass das Rewardsystem, wie es momentan besteht, nicht 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 potentiell mehrere Tausend Punkte bedeuten, und somit auch mehrere Tausend Punkte an Reward an das Modell geben. Dieses Prinzip musste also runterskaliert 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. Das Grundprinzip bleibt gleich, man vergleicht also 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 Tastendrücke 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 aus dem sichtbaren Bereich, während Noten die getroffen wurden, 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 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, Reward für bereits vergangene Aktionen geben zu müssen. Der Ablauf wäre nun 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-Bereichen befindet, und somit jetzt eine passende Aktion ausgeführt werden soll. Das Modell sollte hier also nun einen Tastendruck zurückgeben. Nun 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 Runterskalieren also der Wert +1, zurückgegeben. Wurde die Aktion nicht ausgeführt, die Note also verpasst, wird negativer Reward, -1, an das Modell gegeben. Wird nichts erkannt und auch keine Aktion ausgeführt, wird ebenfalls +1 zurückgegeben, da dies in dieser Situation das richtige Verhalten ist. So soll kontinuierliches 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 occured, 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** 1  
   
 # If the score didn't change, and no action has taken place, give a neutral reward  
 **else**:  
 **return** 0

**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

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 triggern - dies hat allerdings nichts mit künstlicher Intelligenz zu tun.

Nachdem das Environment angepasst wurde, wurden mehrere Tests durchgeführt, welche in der Länge variierten. Der längste von ihnen war 150.000 Schritte lang. In der folgenden Grafik sieht man die Vergleiche zwischen den einzelnen Lernprozessen. 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.

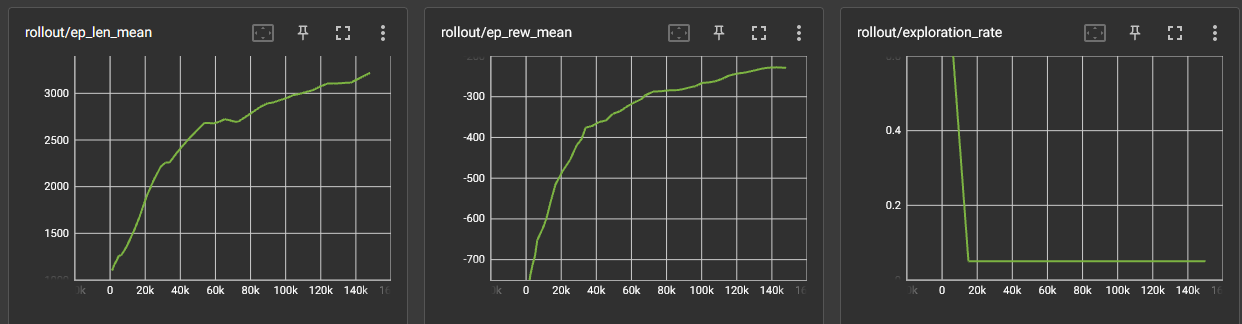


Abb. - Screenshot von Tensorboard - Darstellung der Mittelwerte der Episodenlänge, des Rewards, sowie der Wert der Exploration Rate nachdem das Rewardsystem überarbeitet wurde

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 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.

[Bild der unterschiedlichen Notenarten einfügen]

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 dem Prüfen 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. Ein “Hold” wird von der Funktion für die gesamte Dauer des “Holds” als Pfeil identifiziert. Das Modell soll also permanent diese Taste drücken, was allerdings nicht mit der Funktionsweise des Tastendrücken im Environment funktioniert. Ein Gedrückthalten einer Taste ist auf Grund technischer Limitierungen nur als Toggle umzusetzen - wird auf Grund einer Aktion eine Taste gedrückt, wird sie so lange heruntergedrückt, bis dieselbe Aktion die Taste wieder loslässt. Um einen “Hold” richtig auszuführen, müsste also erst eine Aktion eine Taste runterdrücken, dann warten und keine Aktion mit dieser Taste ausführen bis der “Hold” vorbei ist, um dann mit derselben Aktion die Taste wieder loszulassen. Da aber der “Hold” durchgehend als Pfeil identifiziert wird, erwartet die Funktion, dass in jedem Step diese Taste gedrückt wird. So wird ebenfalls falsches Feedback vermittelt

### Datenkomprimierung

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

Eine Möglichkeit ist das Umwandeln 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

Der Unterschied ist erkennbar:



Abb. - Bildschirmaufnahmen des Gameplays, in Graustufen

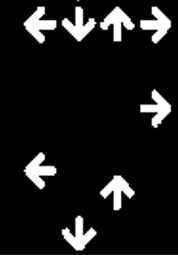


Abb. - Bildschirmaufnahmen des Gameplays, als Binärbild

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 anzufertigen, von der letzten Observierung und der aktuellen. 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 wird, nur auf die sich bewegenden 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:

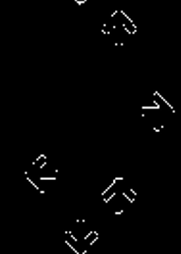


Abb. - Bildschirmaufnahme des Gameplays, als Vergleichsbild

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.

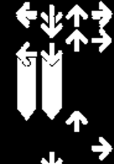


Abb. - Bildschirmaufnahme des Gameplays, auf welchem u.a. zwei „Hold“-Pfeile zu sehen sind, als Binärbild

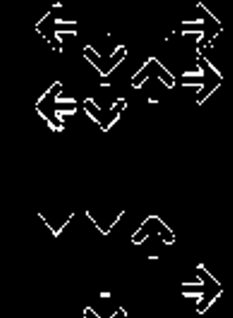


Abb. - Bildschirmaufnahme des Gameplays, Vergleichsbild von Abb. 21

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 die untere Kante “bewegt”. Ein „Hold“-Pfeil ist so von einem regulären Pfeil nicht zu unterschieden. 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.

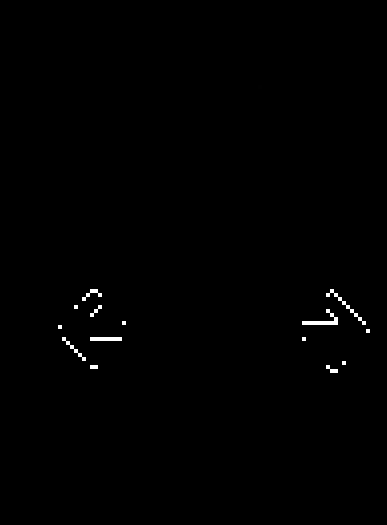


Abb. - Weitere Bildschirmaufnahme des Gameplays, als Vergleichsbild

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

## Bewertung und Einordnung

Die Erkenntnisse aus der eigenen Umsetzung sollen in diesem Kapitel retrospektiv betrachtet werden. Dabei soll analysiert werden, inwiefern andere Formen der Umsetzung andere Ergebnisse geliefert hätten, wo Schwierigkeiten mit den genutzten Technologien auftraten, ob und wie diese behoben hätten werden können, und inwiefern man die Ergebnisse auf andere Spiele oder Anwendungsbereiche übertragen kann.

# Fazit und Ausblick

Im letzten Kapitel sollen die Erkenntnisse aus der Arbeit kurz zusammengefasst werden. Zusätzlich soll in die nahe Zukunft geschaut werden, wie sich das Thema KIs in den kommenden Jahren weiter entwickelt, und inwiefern es den Sprung und die Integrierung in die Gesellschaft/den Mainstream schafft.

## Fazit

## Ausblick

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.

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.

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.

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.

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.

High-Level Expert Group on Artificial Intelligence (2019): A Definition of AI: Main Capabilities and Disciplines. Definition developed for the purpose of the AI HLEG's deliverables. European Commission. Brussels. Online verfügbar unter https://digital-strategy.ec.europa.eu/en/library/definition-artificial-intelligence-main-capabilities-and-scientific-disciplines, zuletzt geprüft am 15.07.2023.

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.

Kinney, Hannah C.; Volpe, Joseph J. (2018): Organizational Events. In: Volpe's Neurology of the Newborn: Elsevier, 145-175.e9.

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, S. M.; Kendall, G. (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.

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.

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.

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.

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.

Spehr, Michael (2023): Darth Vader küsst Marilyn. Frankfurter Allgemeine Zeitung. Online verfügbar unter https://www.faz.net/aktuell/technik-motor/digital/midjourney-bildgeneratoren-mit-ki-erzeugen-fotos-die-es-nicht-gibt-18753895.html, zuletzt aktualisiert am 20.03.2023, zuletzt geprüft am 31.05.2023.

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

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.

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.

Zhang, Daniel; Maslej, Nestor; Brynjolfsson, Erik; Etchemendy, John; Lyons, Terah; Manyika, James et al. (2022): The AI Index 2022 Annual Report. Hg. v. AI Index Steering Committee, Stanford Institute for Human-Centered AI, Stanford University. Online verfügbar unter https://aiindex.stanford.edu/wp-content/uploads/2022/03/2022-AI-Index-Report\_Master.pdf, zuletzt geprüft am 28.03.2023.