Technische Dokumentation

Softwaretechnik-Projekt SoSe2024

Thema

Plattform zum Vergleich von Spiele-KIs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Änderungshistorie** | | | | |
| **Version** | **Datum** | **Kapitel** | **Änderung** | **Name** |
| 0.1 | 30.06.2024 | Alle | Anlegen und Füllen | Justine Buß |
| 0.2 | 02.07.2024 | Alle | Anlegen | Justine Buß |
| 0.3 | 15.07.2024 | 3,4 | Füllen | Max Bachmann |

|  |  |  |
| --- | --- | --- |
| Herausgeber | [Technische Hochschule Mittelhessen – FB06 Mathematik, Naturwissenschaften und Informatik](https://www.thm.de/mni/) | |
| Dateiname | Technische\_Dokumentation\_ver\_0.3 | |
| Dokumentenbezeichnung | Technische Dokumentation: Plattform zum Vergleich von Spiele-KIs | |
| Version | 0.3 | |
| Stand | Mittwoch, 17. Juli 2024 | |
| Status | In Bearbeitung | |
| Autoren | Justine Buß, Thorben Jones, Alexander Roos, Maximilian Bachmann, Omar Karkotli, Sven Reinhard, Pascal Waldschmidt | |
| Freigegeben von |  |  |
| Ansprechpartner | Justine Buß | [justine.buss@mni.thm.de](mailto:justine.buss@mni.thm.de) |
|  | Thorben Jones | [thorben.jones@mni.thm.de](mailto:thorben.jones@mni.thm.de) |
| Kurzinfo | „Technische Hochschule Mittelhessen Softwaretechnik-Projekt. Technische Dokumentation.“ | |

Inhaltsverzeichnis

[1 Projektbeschreibung 7](#_Toc171995415)

[1.1 Projektumfeld 7](#_Toc171995416)

[1.2 Organisatorisches Vorhaben 7](#_Toc171995417)

[1.3 Ziel des Projektes 7](#_Toc171995418)

[1.4 Beteiligte 7](#_Toc171995419)

[1.5 Dokumenteninhalt 8](#_Toc171995420)

[2 Projektübersicht 8](#_Toc171995421)

[3 Entwicklungsdokumentation 9](#_Toc171995422)

[3.1 Architektur & Design (System, Komponenten-, Modul- Datenflussdiagramme) 9](#_Toc171995423)

[3.2 Technologiestack (Sprachen, Frameworks, Bibliotheken und Werzeuge) 9](#_Toc171995424)

[3.2.1 Sprachen 9](#_Toc171995425)

[3.2.2 Frameworks, Bibliotheken, Werkzeuge, Protokolle 9](#_Toc171995426)

[3.3 Code-Struktur (Verzeichnisstruktur, Hauptklassen/-module, Namenskonventionen, Kodierungsstandards) 10](#_Toc171995427)

[3.4 Entwicklungsprozess (Scrum, Git-Workflows, Build-/Deployment-Prozess) 11](#_Toc171995428)

[3.5 Deployment 11](#_Toc171995429)

[4 Weiterentwicklungsdokumentation 12](#_Toc171995430)

[4.1 Implementierung von AlphaZero-Spielen 12](#_Toc171995431)

[4.1.1 Auswahl des Spiels 12](#_Toc171995432)

[4.1.2 Anlegen der Projektordner und -dateien 12](#_Toc171995433)

[4.1.3 Implementierung der Spiellogik 14](#_Toc171995434)

[4.1.4 Erstellen des neuralen Netzes 21](#_Toc171995435)

[4.1.5 Trainieren des Spiels 23](#_Toc171995436)

[4.1.6 Frontendanbindung 27](#_Toc171995437)

[4.2 Integration von GitHub-Spielen 28](#_Toc171995438)

[4.2.1 Auswahl des GitHub-Spiels 28](#_Toc171995439)

[4.2.2 Mögliche Anpassung der Struktur 28](#_Toc171995440)

[4.2.3 Anpassung der Imports und ggf. Logik 29](#_Toc171995441)

[4.2.4 Training und Frontend Anbindung 29](#_Toc171995442)

[(4.3 Erweiterung der Spielfeldoptionen und Konfigurationen 30](#_Toc171995443)

[4.3.1 Anpassung der Spiele 30](#_Toc171995444)

[4.3.2 Anpassung der JSON-Commands 31](#_Toc171995445)

[4.3.3 Anpassung im Frontend?) 31](#_Toc171995446)

[4.4 Verbesserung der Benutzererfahrung 31](#_Toc171995447)

[4.4.1 Benutzeranmeldung, Datenmanagement und Datenschutz 32](#_Toc171995448)

[4.4.2 Administratorenrollen und -rechte 32](#_Toc171995449)

[5 Testdokumentation 33](#_Toc171995450)

[5.1 Teststrategie/-ansatz (Testmethoden) 33](#_Toc171995451)

[5.2 Testplanung (Testumgebungen und -daten) 33](#_Toc171995452)

[5.3 Testfälle/-szenarien (Beschreibung der Testfälle, erwartete Ergebnisse) 33](#_Toc171995453)

[5.4 Testergebnisse/-berichte (Zusammenfassung, Fehler und Probleme Maßnahmen zur Fehlerbehebung, Retests) 33](#_Toc171995454)

[6 Glossar (Begriffe, Verweise) 34](#_Toc171995455)

[7 Index 35](#_Toc171995456)

Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

Tabellenverzeichnis

[Tabelle 1: Interne Beteiligung 7](#_Toc170811924)

# 1 Projektbeschreibung

## 1.1 Projektumfeld

Diese technische Dokumentation beschreibt das Softwaretechnik-Projekt, das im Rahmen eines Kurses an der Technischen Hochschule Mittelhessen unter der Leitung von Prof. Dr. Frank Kammer entwickelt wurde.

## 1.2 Organisatorisches Vorhaben

Das Projekt wurde im Zeitraum vom 11. April 2024 bis 25. Juli 2024 durchgeführt und umfasste einen Rahmen von bis zu 270 Stunden pro beteiligtem Teammitglied.

## 1.3 Ziel des Projektes

Die Entwicklung einer benutzerfreundlichen Plattform, die es Nutzern ermöglicht, praxisnahe Erfahrungen mit künstlicher Intelligenz (KI) zu sammeln. Die Plattform bietet eine interaktive Umgebung, in der verschiedene KIs gegeneinander antreten können und Nutzer die Möglichkeit haben, eigene KI-Entwicklungen zu testen.

## 1.4 Beteiligte

|  |  |  |  |
| --- | --- | --- | --- |
| **Rolle** | **Name** | **Fachbereich/ Studienfach** | **Kontaktinformation** |
| Projektleitung | Thorben Jones | MNI/Ingenieur-Informatik | [thorben.jones@mni.thm.de](mailto:thorben.jones@mni.thm.de) |
| Stellvertretende Projektleitung | Justine Buß | MNI/Ingenieur-Informatik | [justine.buss@mni.thm.de](mailto:justine.buss@mni.thm.de) |
| Teammitglied | Alexander Roos | MNI/Ingenieur-Informatik | [alexander.roos@mni.thm.de](mailto:alexander.roos@mni.thm.de) |
| Teammitglied | Maximilian Bachmann | MNI/Informatik | [maximilian.lars.bachmann@mni.thm.de](mailto:maximilian.lars.bachmann@mni.thm.de) |
| Teammitglied | Omar Karkotli | MNI/Informatik | [omar.karkotli@mni.thm.de](mailto:omar.karkotli@mni.thm.de) |
| Teammitglied | Sven Reinhard | MNI/Informatik | [sven.roman.reinhard@mni.thm.de](mailto:sven.roman.reinhard@mni.thm.de) |
| Teammitglied | Pascal Waldschmidt | MNI/Informatik | [pascal.waldschmidt@mni.thm.de](mailto:pascal.waldschmidt@mni.thm.de) |
| Dozent | Prof. Dr. Frank Kammer | MNI | [frank.kammer@mni.thm.de](mailto:frank.kammer@mni.thm.de) |

Tabelle 1: Interne Beteiligung

## 1.5 Dokumenteninhalt

Die vorliegende Dokumentation umfasst alle technischen Details der entwickelten Software. Sie beinhaltet eine umfassende Entwicklungsdokumentation, die den gesamten Entwicklungsprozess beschreibt, eine Weiterentwicklungsdokumentation, die Hinweise und Anleitungen für zukünftige Erweiterungen und Verbesserungen bietet, sowie eine Testdokumentation, die die durchgeführten Tests und deren Ergebnisse detailliert darstellt.

Zusätzlich wird ein separates Benutzerhandbuch [Link einfügen] zur Verfügung gestellt, dass die Nutzung der bereitgestellten Webseite und der Schnittstelle für externe Anbindungen genauer erläutert.

# 2 Projektübersicht

Die gesamte Beschreibung des Projekts, einschließlich der Ziele und Anforderungen an die zu entwickelnde Software, ist ausführlich im Pflichtenheft [Link einfügen] formuliert, das auf Basis des Lastenhefts [Link einfügen] erstellt wurde. Das Pflichtenheft enthält detaillierte Informationen zu den funktionalen und qualitativen Anforderungen, die die Rahmenbedingungen für die Entwicklung festlegen. Es dient als verbindliche Grundlage für die Umsetzung des Projekts und stellt sicher, dass alle definierten Anforderungen und Ziele klar dokumentiert sind.

# 3 Entwicklungsdokumentation

## 3.1 Architektur & Design (System, Komponenten-, Modul- Datenflussdiagramme)

## 3.2 Technologiestack (Sprachen, Frameworks, Bibliotheken und Werzeuge)

### 3.2.1 Sprachen

1. Python:

Sowohl das AlphaZero-Framework, auf dem wir aufbauen und welches wir erweitert haben, als auch die implementierten Spiele, als auch die Kommunikation zwischen Backend und Frontend sind komplett in Python geschrieben.

Genutzt wird Python 3.11, da mit Python 3.11 die Kompatibilität mit Bibliotheken, die für das AlphaZero-Framework benötigt werden, gewährleistet ist.

1. JS:

<Frontend>

1. CSS:

<Frontend>

1. … ?

### 3.2.2 Frameworks, Bibliotheken, Werkzeuge, Protokolle

1. Vue.js: Für die realisierung des Frontends. Als State Management System wird VUEX verwendet, für das interne Routing der vue-Router, und für die ermöglichung von mehrsprachigkeit die vue-18n Bilbiothek.

<Frontend>

1. AlphaZero-Framework:
2. Docker:
3. Websockets:

Für die Kommunikation zwischen Backend und Frontend bzw. Backend und externe Schnittstelle (KI, Spieler) benutzen wir das Websockets-Protokoll, welches über das fastAPI-Framework bezogen wird.

Websockets ermöglicht bidirektionale Kommunikation in Echtzeit. Heißt nach einem initialen TCP-Handshake wird der gesamte Overhead gespart und Daten werden ohne Weiteres hin- und hergeschickt. Zudem kann der Server den Clients ohne Request Nachrichten schicken. Das macht Sinn, da interne Spielzustände sich ändern können ohne zu vorige Anfrage durch den Client. Dann muss der Server antworten können. Methodiken wie Polling durch die Clients ersparen wir uns dadurch, wodurch die Serverlast reduziert wird.

1. JSON:

JSON wird als Datenaustauschformat genutzt, da einfach zu handhaben ist, effiziente Zugriffe auf Daten zulässt und weit verbreiteter Standard ist.

1. Pytorch:

Pytorch ist ein Framework zur Entwicklung und zum Training von neuralen Netzen und wird im Zuge des AlphaZero-Frameworks genau für dies bei uns eingesetzt.

1. Keras:

Keras setzt auf Tensorflow auf und abstrahiert viel vom Nutzer. Es ermöglicht mit wenigen Befehlen ein leichtes Bauen der neuralen Netze. Dennoch bevorzugen wir Pytorch, da mit Pytorch viel schneller trainiert werden kann.

1. Pygame:

Für das Zeichnen der Spielbretter benutzen wir Pygame, eine Python-Bibliothek. Das Spielbrett wird gezeichnet, das entstandene „surface“ wird in einen png-bytestream umgewandelt, verschickt und im Frontend wieder reinterpretiert als png.

1. Numpy:

Intern werden die Spielbretter als Numpy-Arrays gehandhabt und ausgetauscht. Dies ermöglicht effiziente Operationen darauf, bspw. beim Rotieren, Spiegeln usw.

1. Pytest:

<Sven>

Anzumerken ist des Weiteren, dass die requirements.txt so angepasst sind, dass sie plattformübergreifend auch mit der Apple Silicon Chip-Generation kompatibel sind.

Das umfasst v.a. die Module tensorflow und h5py. Bei jenen gab ein Problem.

Und auch durch die Dockerisierung ist dieses Problem nicht behoben, denn würde bspw. h5py in der Version 3.8.0 nicht am Anfang der Server/requirements.txt stehen, so würde es als Submodul über tensorflow geladen werden in anderer Version, wodurch es zu einem Fehler beim build in Docker kommt.

Heißt bestenfalls einfach so lassen wie es ist, v.a. in der Reihenfolge h5py ganz oben stehen lassen.

## 3.3 Code-Struktur (Verzeichnisstruktur, Hauptklassen/-module, Namenskonventionen, Kodierungsstandards)

## 3.4 Entwicklungsprozess (Scrum, Git-Workflows, Build-/Deployment-Prozess)

## 3.5 Deployment

# 4 Weiterentwicklungsdokumentation

## 4.1 Implementierung von AlphaZero-Spielen

Um den Spielepool zu erweitern und die Vielfalt für Spieler zu erhöhen, können neue Spiele nach Bedarf hinzugefügt werden. Aktuell basieren die vorgeschlagenen Spiele auf dem AlphaZero-Framework, das durch seine deterministische Natur, die Auslegung für zwei Spieler und die vollständige Information während des Spiels charakterisiert ist. Weitere Beispiele hierfür sind Schach, Halma, Hex, Mühle, Königsrennen, Käsekästchen, NoGo und Abalone oder Abwandlungen der bereits integrierten Spiele, beispielsweise GoBang oder 3D-Tic-Tac-Toe.

### 4.1.1 Auswahl des Spiels

Identifizierung des Spiels, das in das System integriert werden soll, basierend auf den festgelegten Kriterien und den Anforderungen des AlphaZero-Frameworks.

* **Determinismus**: Jeder Spielzug und Zustand des Spiels muss eindeutig durch die Spielregeln festgelegt sein. Dies ermöglicht es dem AlphaZero-Algorithmus, auf Basis von simulierten Spielen und Lernen optimale Spielstrategien zu entwickeln.
* **Auslegung für zwei Spieler**: AlphaZero ist primär auf Spiele ausgelegt, die zwischen zwei Spielern gespielt werden, wobei jeder Spieler abwechselnd einen Zug macht. Dies ermöglicht eine klar definierte Interaktion und Strategieentwicklung zwischen zwei Parteien.
* **Vollständige Information:** Das Spiel muss in einer Umgebung stattfinden, in der beide Spieler jederzeit Zugriff auf alle relevanten Informationen haben. Im Gegensatz dazu stehen Spiele mit verdeckten Informationen.

### 4.1.2 Anlegen der Projektordner und -dateien

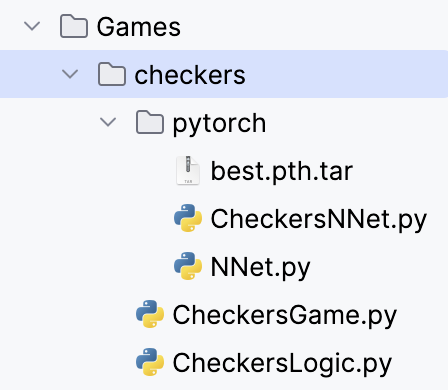
1. In der Projektübersicht in den Games/ Ordner navigieren.

Ein Bild, das Text, Screenshot, Zahl enthält.

Automatisch generierte Beschreibung

1. Einrichten der erforderlichen Projektordner und -dateien, die für die Entwicklung und Integration des neuen Spiels erforderlich sind.

* Games/<game>: Ordner für den Quellcode.
* Games/<game>/pytorch/: Für die Implementierung mit PyTorch. Empfohlen aufgrund von deutlich schnelleren Trainingsphasen und stärkerer Modelle.
  + Games/<game>/pytorch/<Game>NNet.py: Wrapper-Klasse für ein neuronales Netzwerk.
  + Games/<game>/pytorch/NNet.py: Definition des neuronalen Netzwerks
* Games/<game>/keras/: Für die Implementierung mit Keras. Ist ebenfalls eine PyTorch Implementierung vorhanden mit Modell, wird diese jedoch bevorzugt vom dynamischen Importer.
  + Games/<game>/keras/<Game>NNet.py: Wrapper-Klasse für ein neuronales Netzwerk.
  + Games/<game>/keras/NNet.py: Definition des neuronalen Netzwerks
* Games/<game>/<Game>Game.py: Python Code, der das Spiel implementiert. Hier müssen die Methoden aus dem Interface Tools/i\_game.py implementiert werden.
* Games/<game>/<Game>Logic.py: Python Code, der die Spielbrettlogik und Aktionen auf dieses Spielbrett umfasst. Sie stellt Methoden bereit, um das Spiel zu implementieren.



### Implementierung der Spiellogik

Anmerkungen vorab:

* Grundsätzlich empfiehlt es sich die Spiele so zu implementieren, dass Züge eindimensional gehandhabt werden. Also nur ein Index für ein Feld, statt x und y Koordinate. Dabei wird das Spielfeld durchnummeriert und jedes Feld erhält einen Index. Zum einen ist die Kommunikation mit dem Frontend so aufgebaut, zum anderen ist die Handhabung allgemein leichter.

Falls man bspw. bereits auf eine Vorimplementierung zurückgreift, die dies nicht umsetzt, so benötigt es Hilfsmethoden, die die Züge intern umwandeln, was auch nicht das Problem darstellt.

* Spiele, die nur einen Zielindex haben als Zug (bei TicTacToe das Feld, bei 4gewinnt die Reihe), dort ist der Zug nur ein „int“. Bei Spielen mit Ziehmöglichkeit, Startfeld und Zielfeld (Dame, Schach…), dort sind Züge Tupel, also „(<start\_pos>, <ziel\_pos>)“ und werden intern so verarbeitet.

1. Implementierung des Spielbrettes und Aktionen auf dem Spielbrett befinden sich in der Games/<game>/<Game>Logic.py. Diese Klasse wird als Hilfsklasse genutzt, um die Spielbretter und die Figuren darauf darzustellen. Hier werden zudem die Züge der Figuren simuliert.

Nur Methoden der jeweiligen Games/<game>/<Game>Game.py haben Zugriff darauf.

|  |  |
| --- | --- |
| Marke 1 mit einfarbiger Füllung |  |

1. Analysieren des Tools/i\_game.py Interfaces. Es definiert die spezifische Methode, um Spielzüge auszuführen, Spielzustände zu verwalten und Spielinformationen abzurufen. Diese Schnittstelle stellt sicher, dass das Spiel gemäß den Anforderungen des AlphaZero-Frameworks korrekt integriert wird. Das Interface ist ausreichend kommentiert und gibt genau vor, welche Argumente und Rückgabewerte die entsprechenden Methoden haben müssen.

Kommen neue Funktionen und/ oder Spiele hinzu, die nicht in das vorhandene Interface passen, so lässt sich dieses allerdings auch selbstverständlich erweitern – denkbar bei Spielen, die weder einen Zugindex (Zielposition) noch zwei Zugindizes (Startposition, Zielposition) aufweisen. Derzeitig ist unser Framework nur auf jene beiden Varianten ausgelegt, da uns kein Spiel in den Sinn kommt, welches eine andere Zugstruktur aufweist.

Falls solche Spiele hinzukommen, sind Anpassungen am Interface und in der GameClient/game\_client.py nötig, die die eingehenden Züge parst und an die Spiellogiken über die GameClient/arena.py weiterreicht.

|  |  |
| --- | --- |
|  | Abzeichen mit einfarbiger Füllung |

1. In Games/<game>/<Game>Game.py gilt es die Methoden des Tools/i\_game.py Interfaces zu implementieren. Dazu muss das Interface importiert und von der Klasse erweitert werden.

|  |  |
| --- | --- |
|  | Marke 3 mit einfarbiger FüllungMarke 3 mit einfarbiger FüllungMarke 3 mit einfarbiger Füllung |

Die Methoden der jeweiligen <Game>Game.py werden an verschiedenen Stellen benutzt:

1. In der Trainer/coach.py in Kombination mit den neuralen Netzen und der Monte Carlo Tree Search (Tools/mcts.py) zum Trainieren.
2. bei der Spielzugberechnung von KIM in der Server/socketServer.py durch die neuralen Netze mithilfe der Tools/mcts.py.
3. in der GameClient/arena.py, um das Spielfeld zu initialisieren, einen Zug auszuführen oder zu überprüfen, ob das Spiel zu Ende ist.
4. in der GameClient/game\_client.py, um bspw. den Zug des Frontend um 180° zu rotieren.
   1. Dies trifft nur auf Spiele zu, bei denen das Spielfeld für einen der beiden Spieler um 180° rotiert wird, damit beide von unten nach oben spielen können - so z.B. bei unserer Implementierung des Spiels „Dame“.
   2. Bei allen anderen Spielen wird der Zug einfach so wieder zurückgegeben.

Die Methoden müssen sich an die Vorgaben des Interfaces halten.

Neben den zu implementierenden Interfacemethoden können jegliche Hilfsmethoden verwendet werden.

Sonstige Hinweise:

1. Züge, die durch die KI ausgegeben werden, sind nicht wirklich Züge, sondern Zugindizes der Züge im getValidMoves-Array.

Alle validen Züge werden als binäres Array, ob der jeweilige Zug möglich ist (1) oder nicht (0), mit getValidMoves abgefragt, und es wird lediglich der Index des besten Zugs zurückgegeben.

Heißt: diese müssen vor der Ausführung noch rückumwandelt werden, wobei eine Methode des i\_game.py Interfaces zum Tragen kommt, nämlich translate, die für das jeweilige Spiel implementiert werden muss und den Index in den tatsächlichen Zug zurückumwandelt.

Bei Spielen wie TicTacToe oder 4gewinnt, bei dem der Zug nur aus einem Wert, nämlich der Zielposition besteht und bei denen die Indizes zugleich auch 1:1 die Züge auf dem Brett sind, kann der Index in translate unverändert wieder zurückgegeben werden.

Bei allen anderen Spielen, bei denen der Zug zumeist ein Tupel ist, funktioniert dies nicht mehr und man muss translate komplexer implementieren, passend zum Spiel. Die Berechnungsmethode, die einem Zug eine Indexposition im getValidMoves-Array zuweist, wird dabei im selben Atemzug benutzt, um den Index wieder zurückzurechnen in einen Zug. Orientierung bietet hierbei das Spiel „Dame“.

1. Eine Herausforderung bei der Implementierung des Interfaces kann die Methode getSymmetries darstellen. Diese wird in Trainer/coach.py dazu benutzt, zusätzlich zu einem Spielzustand alle weiteren Symmetrien auch in die Trainingsdaten mit aufzunehmen, um einfach mehr Trainingsdaten zu haben, auf denen trainiert werden kann und um später besser generalisieren zu können.

Bei einfachen Spielen wie TicTacToe und 4gewinnt, bei der es auch keiner komplexen translate-Methode bedarf, ist der folgende Abschnitt nicht relevant, aber bei allen anderen Spielen mit Tupel-Zügen (also Start- und Zielposition eines Spielsteins), kann er das sein.

Häufige Symmetrien sind Rotation und Spiegelung oder das Vertauschen von Reihen.

Dabei wird zum einen das Brett und der dazugehörige Policy-Vektor mit den Wahrscheinlichkeiten zu den validen Zügen für den jeweiligen Spielzustand gedreht, gespiegelt, getauscht.

Damit bei einem quadratischen Brett (Dame, Schach, …) der Policy-Vektor auch mittels numpy zu einem quadratischen zweidimensionalen Array reshaped (numpy.reshape) werden kann, welches mittels einem einzigen Methodenaufruf (numpy.rot90) zusammen mit dem Brett gedreht werden kann, ohne dass es zu Verzerrungen kommt, also dass nach der Drehung ein und derselbe Zug plötzlich eine andere Wahrscheinlichkeit zugewiesen bekommt, weil der reshaped Policy-Vektor nicht quadratisch ist, also bspw. 8 geschachtelte Arrays mit je 8 Wahrscheinlichkeiten, macht es Sinn, ein Lückenfüller (Padding) einzuführen.

Die Größe des Policy-Vektors leitet sich von der Methode getActionSize des Interfaces ab, heißt also, dass man die „ActionSize“, also die Menge aller möglichen Züge im gesamten Spiel so gestalten sollte, dass wenn man die quadratische Wurzel daraus zieht, eine Ganzzahl herauskommt. So wird sichergestellt, dass später durch numpy.reshape des Policy-Vektors ein quadratisches zweidimensionales Array herauskommt.

Beachtet man dies, so ist die getSymmetries-Methode in wenigen Zeilen abgefrühstückt, und es ist sichergestellt, dass es zu keinen Verzerrungen kommt, wodurch die Trainingsdaten allesamt falsch wären und schlecht trainiert werden würde.

In dem Sinne, weil von der „ActionSize“ auch die getValidMoves-Methode abhängt, muss dieses Padding auch berücksichtigt werden bei der Berechnung der Indizes der validen Züge für das getValidMoves-Array.

Inspirationsquelle liefert unsere Implementierung von „Dame“.

Dort haben wir genau diesen Ansatz verfolgt.

1. ValueError werfen, falls der Zug nicht valide ist.

Die GameClient/arena.py erhält den Zug, sei es von KIM oder von einer externen KI/ einem externen Spieler, und prüft nicht unmittelbar den Zug, ob er valide ist. Sie reicht diesen in die Spiellogik weiter mittels der Methode getNextState, welche dann in der jeweiligen <Game>Logic.py den Zug ausführen lässt. Da wir auf eine weitere „validMoves“-Methode im Interface verzichten wollten – die Bestehende liefert ja nicht wirkliche Züge, sondern ein binäres Array aus Nullen und Einsen für die Zugberechnung innerhalb des AlphaZero-Frameworks – muss die Zugüberprüfung in <Game>Logic.py in der Methode stattfinden, in der der Zug ausgeführt wird.

Konvention ist:

Falls der Zug nicht valide ist, muss ein ValueError ausgelöst werden, der in unserer GameClient/arena.py abgefangen wird, wodurch eine Nachricht an den Spieler rausgeht, dass der Zug ungültig war und auf einen nächsten Zug gewartet wird.

1. Bei allen Spielen, bei denen es per se möglich ist, unendlich oft redundante Züge zu machen, sollte in der getGameEnded-Methode, die ausgibt, ob das Spiel fertig ist und wer gewonnen hat, als Endbedingung für ein Unentschieden eingeführt werden, dass bspw. nach 30 oder 50 solcher Züge, bei denen nichts passiert, bspw. kein Stein geschlagen wird, es zu einem Unentschieden kommt.

Macht man dies nicht, so wird beim Training sehr schnell in Tools/mcts.py ein RecursionError geworfen werden, da die Rekursionstiefe von 1000 überschritten wird, wenn stets neue Züge gemacht werden, aber das Spiel nicht terminiert, weil nur hin- und herbewegt wird.

1. Bei Spielen, bei denen Aussetzen eine Möglichkeit ist – so z.B. Othello falls man temporär nicht mehr ziehen kann – ist dieser Zug aktuell im Framework als letzter Zug kodiert. Ist das Feld 6x6 groß, ist die „ActionSize“ eigentlich 6\*6 = 36, doch mit Aussetzen eben +1, also 37. Als Zug ergibt sich also der Index 36 für aussetzen. Sowohl Frontend als auch Backend folgen dieser Konvention.

Das Ganze macht auch Sinn, falls man bspw. das Spielfeld drehen will und damit auch den Policy-Vektor, so kann man ganz einfach mit Array-Slicing den letzten Index abschneiden, mit Numpy ein reshape() durchführen, drehen, und am Ende wieder den Index anfügen.

1. Weitere Hinweise, derer es hier keine weitere Erläuterung bedarf, finden sich im Tools/i\_game.py Interface.
2. Werden zusätzliche imports neben numpy benötigt, so müssen diese aufgrund der Server und GameClient Architektur in die GameClient/requirements.txt hinzugefügt werden. Man beachte, dass sich dabei die Größe der GameClients erheblich erhöhen kann.

Problematisch daher, dass jeder GameClient in einem eigenen Docker Container läuft und dadurch bei vielen parallelen Spielen sehr viel RAM aufgefressen wird. Deshalb wird die ausschließliche Handhabung mit numpy und math bevorzugt.

|  |  |
| --- | --- |
|  | Marke 4 mit einfarbiger Füllung |

Ein Sonderfall stellt die PyGame Bibliothek dar, die zum Zeichnen der Spielfelder verwendet wird. Der Import dieser Bibliothek muss explizit in der draw-Methode stattfinden. Dies ist auf die Größe der PyGame Bibliothek zurückzuführen.

|  |  |
| --- | --- |
|  | Marke 5 mit einfarbiger FüllungMarke 5 mit einfarbiger Füllung |

### 4.1.4 Erstellen des neuralen Netzes

1. Für alle Spiele, die auf einem 2-dimensionalen Spielbrett gespielt werden, können die neuralen Netzwerk Klassen von Othello für PyTorch (Games/othello/pytorch/NNet.py, Games/othello/pytorch/OthelloNNet.py) oder Tic-Tac-Toe für Keras (Games/tictactoe/keras/NNet.py, Games/tictactoe/keras/TicTacToeNNet.py) als Vorlage genommen werden. Dazu muss der Inhalt der Klassen nach Games/<game>/<framework>/NNet.py und Games/<game>/<framework>/ <Game>NNet.py kopiert werden.

|  |  |
| --- | --- |
|  | Marke 1 mit einfarbiger Füllung |
|  | |
|  | Marke 1 mit einfarbiger Füllung |

1. Nach dem Kopieren müssen lediglich alle Aufrufe der Othello- oder TicTacToe-Klasse auf das entsprechend neu zu implementierende Spiel angepasst werden.

|  |  |
| --- | --- |
|  | Abzeichen mit einfarbiger Füllung |
|  | |
|  | Abzeichen mit einfarbiger FüllungAbzeichen mit einfarbiger FüllungAbzeichen mit einfarbiger Füllung |

1. Für alle Spiele, die nicht die 2D-Bedingung erfüllen, kann ein Großteil der Struktur weiterhin übernommen werden. Natürlich müssen auch die Aufrufe der übergeordneten Klasse angepasst werden. Zusätzlich müssen Anpassungen an den entsprechenden PyTorch- oder Keras-Aufrufen und Tensoren vorgenommen werden.

### 4.1.5 Trainieren des Spiels

1. Für das Training steht eine eigene main.py zur Verfügung, die einen durch das Trainieren hindurchführt.

Ein Bild, das Text, Screenshot, Zahl enthält.

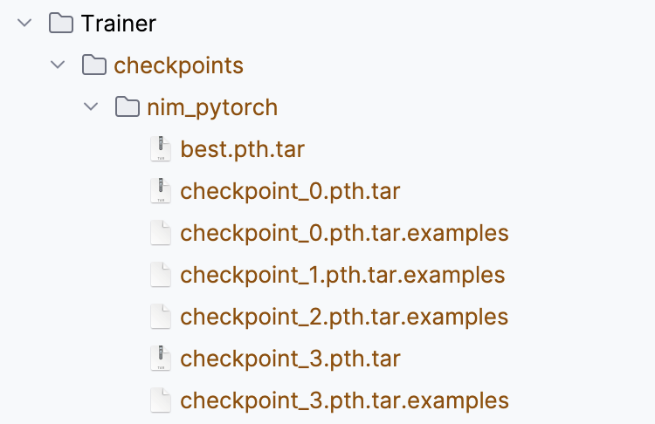
Automatisch generierte Beschreibung

1. Dazu muss in den Trainer/ Ordner navigiert werden. Dort muss die Trainer/main.py ausgeführt werden.

|  |  |
| --- | --- |
| Abzeichen mit einfarbiger Füllung |  |

1. Wenn die Ordner und Dateien richtig angelegt wurden, findet der dynamische Lader das neu angelegte Spiel und ermöglicht dessen Auswahl. Wichtig hierbei ist, dass die Dateien für das neuronale Netz und die Spiellogik den Namenskonventionen folgen und in den richtigen Ordnern liegen.
2. Dem Trainer, der mittels der main.py ausgeführt wird, ist es dabei egal, ob bereits ein trainiertes Modell vorliegt, oder ob von Neuem trainiert wird.
3. Nach der vollendeten Trainingsphase und Akzeptierung des neuen Modells, wird dieses automatisch im Ordner Trainer/<game>\_<framework>/ gespeichert. Die Datei folgt bei einem PyTorch-Training der Namensgebung best.pth.tar und bei Keras best.h5. Diese können daraufhin noch beliebig umbenannt werden, solange die Endung best.pth.tar oder best.h5 erhalten bleibt. Zugleich werden in diesem Verzeichnis die Trainingsdaten in der Form checkpoint\_<iteration>.pth.tar.examples abgelegt, anhand derer trainiert wurde. Für jede Iteration gibt es eine checkpoint\_<iteration>.pth.tar.examples - Datei.

Die best.h5 bzw. best.pth.tar beschreibt dabei das aktuell beste Modell. Des Weiteren wird dieses zugleich auch immer als checkpoint\_<iteration>.pth.tar bzw. checkpoint\_<iteration>.h5 abgelegt. Dadurch hat man alle Zwischenstadien der besten Modelle weiterhin noch erhalten, denn die best.h5 bzw. best.pth.tar überschreibt sich ja stetig, falls es ein besseres Modell künftig gibt. Macht Sinn, falls man ein Modell bspw. bereits zu stark trainiert hat, und auf Schwierigkeitsgrad „easy“ es schon unschlagbar ist.



1. Hat man sich bei der Auswahl der Iterationen ein wenig verschätzt und es dauert einem zu lange, kann man natürlich jederzeit das Training abbrechen. Das bis dato beste Modell ist in obig genanntem Verzeichnis abgespeichert mit den zugrundeliegenden Trainingsdaten.

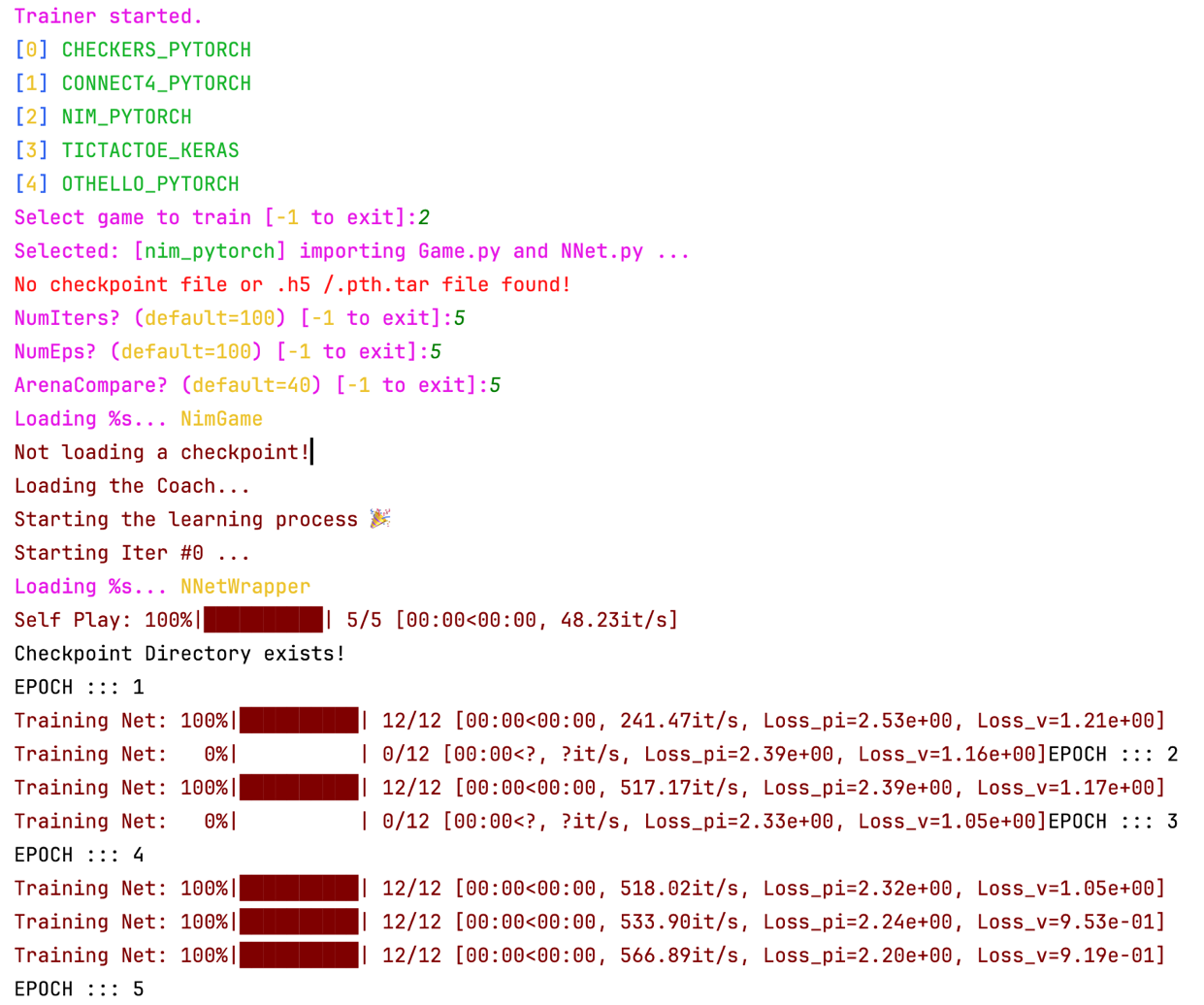
Will man dieses anschließend bspw. am Folgetag weitertrainieren, so ist dies ebenfalls möglich. Der Trainer wählt das Modell (best.h5 bzw. best.pth.tar) und die checkpoint\_<iteration>.pth.tar.examples - Datei mit der höchsten Iteration aus, die nebenbei auch bis zu einem gewissen Grad all die Trainingsdaten der vorigen Iterationen (checkpoint\_<iteration>.pth.tar.examples - Dateien) enthält.

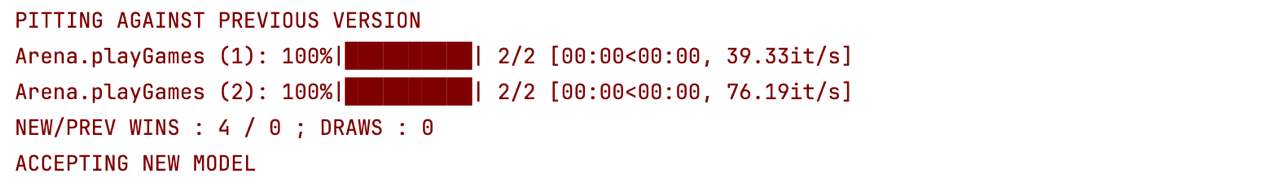
Wichtig ist, dass diese Dateien weiterhin im Verzeichnis Trainer/<game>\_<framework>/ liegen.

Ein Modell weiter zu trainieren ohne die zugrundeliegenden checkpoint\_<iteration>.pth.tar.examples - Dateien, ist nicht möglich. Von daher empfiehlt es sich, die Trainingsdaten erst einmal zu behalten, auch wenn diese mehrere GB groß werden können bei komplexen Spielen.

1. Schlussendlich muss das trainierte Modell in den entsprechenden Ordner verschoben werden: Games/<game>/<framework>/. Gut auch zum Zwischentesten, ob das Modell schon ausreichend stark trainiert wurde. Andernfalls lässt man einfach weiter trainieren.

So sieht das Ganze dann aus, wenn das Training in der main.py angestoßen wurde:





In diesem Fall existierte entweder nicht das Modell oder keine Trainingsdaten oder eben beides nicht. Heißt es wird von vorn trainiert.

* „NumEps?“: Steht für wie viele „Self Plays“ gemacht werden, also wie viele Spiele einmal komplett durchgespielt werden, wobei jeder Spielbrettzustand mit Policy-Vektor (Wahrscheinlichkeitsverteilungen der Züge) inkl. aller Symmetrien abgespeichert wird, was dann in Summe die Trainingsdaten für das neuronale Netz bildet in den checkpoint\_<iteration>.pth.tar.examples - Dateien.
* EPOCH ::: <num> : Steht dafür, dass die Phase des „Self Plays“ abgeschlossen ist und nun das eigentliche Training eingeläutet wird. <num> kann im Trainer nicht verstellt werden, da es standardmäßig schon auf einem guten Wert steht (10). Falls man diesen Wert verändern möchte, dann im args Dictionary der Games/<game>/<framework>/NNet.py des zu trainierenden Spiels.
* „ArenaCompare?“: Steht für wie viele Testspiele des neu trainierten Modells gegen das bestehende Modell nach dem Training durchgespielt werden sollen. Das ArenaCompare knüpft an die Trainingsphase (EPOCH) direkt an

(s. „Pitting against previous version“).

Stimmt das Verhältnis von gewonnenen zu verlorenen Spielen, wird das neue Modell angenommen. Falls nicht, so wird das Alte beibehalten und in der nächsten Iteration wieder aufgegriffen, weitertrainiert. Die aktuelle Schwelle liegt bei 0.6, heißt wenn 6 von 10 Spielen gewonnen worden sind vom neuen Modell, so wird dieses angenommen. Verstellt werden kann dieser Parameter im args Dictionary in der main.py. Dort kann man auch noch weitere Parameter verstellen, auch wenn es sich unserer Meinung nicht empfiehlt, da diese schon gut gewählt sind vom AlphaZero-Framework aus.

* „NumIters?“: Steht für wie viele Iterationen Training durchlaufen werden. Also komplette Zyklen bestehend aus „Self Play“, Training (EPOCH) und Testspiele altes Modell gegen neues.
* Jeder der 3 Parameter, die man im Trainer festlegen kann („NumIters“, „NumEps“, „ArenaCompare“) beeinflusst, wie lang das Training dauert. Dort gilt es je Spiel herumzuprobieren und gute Werte zu finden. Bei wenig komplexen Spielen macht 100 (It.), 100 (Ep.) und 40 (ArenaCompare) durchaus Sinn. Bei komplexen Spielen wie Dame schraubt man den ersten Parameter, nämlich die Iterationen besser erstmal herunter auf 20. Anknüpfend weitertrainieren kann man immer noch.

### 4.1.6 Frontendanbindung

Ideen Max@Pascal:

<Anpassung width und height cursor, twoTurnGame …>

< ggf. dass für jedes neue Spiel kreative neue Möglichkeiten gesucht werden müssen, wie man die Spielzüge eingibt und übergibt => bspw. Nim, was ja mit der bisherigen Konvention brach…aber dass per se Deine Logik mit dem width und height und dem cursor dennoch anpassbar ist und dynamisch funktioniert>

<oder neue Gegebenheiten wie Aussetzlogik falls vom neuen Spiel gefordert>

1. Hinzufügen des Spiels:

-Unter Frontend/components/enums.js in der games Konstante ein neues key-value Paar anlegen, in welcher das value dem Namen des neuhinzugefügte Model entspricht.

-In der i18n.js Datei den Namen des Spiels für jede Sprache hinterlegen in der Form <Name des Models>:”<Name des Spiels in der Sprache>”.

2. Implentierung der Spielsteurung:  
Navigieren zu PlayPage.js  
2.1.Implementierung einfacher Spiele:

Handelt es sich um ein einfaches Spiel welche folgenden Eigenschaften erfüllt:  
-Spielfeld ist ein Rechteck bestehend aus quadratischen Feldern.  
-Ein Zug besteht aus der Auswahl eines einzelnen Feldes (zum Beispiel bei Tic Tac Toe), oder besteht aus der auswahl von 2 Feldern (ein von Position zu Position Zug, wie zum Beispiel bei Schach)  
Kann das Spiel ohne weiteres implementiert werden, indem man den switch case innerhalb des mounted() blocks um den case des Spiels erweitert. Hierbei muss man lediglich die Breite und Höhe des Feldes angeben, also in wie viele Teile das Feld horizontal oder vertikal unterteilt werden soll, sowie die angabe ob es sich um ein einzügiges oder zweizügiges Spiel handelt.

2.2. Implementierung komplexer beziehungsweise atypischer Spiele  
  
Für spiele welche atypischen Spielfelder und oder Zug Informationen besitzen, müssen diese je nach Spiel angepasst werden, solange das Spielfeld selbst interaktiv sein soll, muss man die trackMousePosition methode anpassen, in welcher man im switch ein neues Case für dieses Spiel anlegt. Dort kann man dann gegebenenfalls eine neue Methode anlegen welches das besondere Verhalten realisieren kann. Dabei muss darauf geachtet werden, dass die gesendeten züge sich mit den Zügen decken, auf welchem das KI-Modell arbeitet.  
  
Sollten zusätzliche Schaltflächen für das Spielen notwendig sein, so kann man diese als neues Element in die PlayPage.vue einfügen, die Komponente muss dann mit einem v-if=”ENUMS.games.<Name des Spiels>” versehen werden.   
  
Bei nicht rechteckigen Spielfeldern funktioniert das highlighting nicht, weswegen es sinnvoll ist, die klasse “highlight-cell” für diese spiele ebenfalls via v-if auszublenden, und idealerweise implementiert man alternative Highlightings welche besser auf das atypische Feld angepasst sind.

Für den Fall, dass das Spiel dynamisch auf besondere Spielzustände reagieren soll, so muss der VUEX-Store angepasst werden, da dort sämtliche Kommunikation mit dem Backend stattfindet. Als Beispiel hier, das man in Othello einen Zug überspringen kann, aber nur wenn kein anderer Zug für einen möglich ist. Da das Frontend selbst die Spielogik nicht kennt, wird auf die response von valid\_moves gewartet, und wenn diese nur den Zug 64 zurückgibt (also ein Zug außerhalb des gültigen 8x8 Feldes) so wird automatisch eine Variable geändert, auf welche über Watch in der PlayPage.js reagiert wird, sodass das Kommand für das Überspringen des Zuges gesendet werden kann.

## 4.2 Integration von GitHub-Spielen

Die Integration fertiger GitHub-Spiellösungen bedarf einiger Vorbereitungen und Formalia, um eine reibungslose Integration in das System zu ermöglichen.

### 4.2.1 Auswahl des GitHub-Spiels

1. Zuerst einmal muss auch die ausgewählte GitHub Lösung die [Anforderungen des AlphaZero-Frameworks](#_4.1.1_Auswahl_des) erfüllen.
2. Zudem müssen ja einige formale Aspekte bezüglich der Ordnerstruktur und des Interfaces eingehalten werden. Das Original-GitHub, an dem sich die AlphaZero-Implementierung und das Interface orientieren, ist das [suragnair/alpha-zero-general](https://github.com/suragnair/alpha-zero-general). Implementierungen, die ebenfalls darauf aufbauen, sind zumeist fast nahtlos übernehmbar. Einige Beispiele dafür sind:
   * Go: [liranmatcu/alpha-zero-general-with-Go-game](https://github.com/liranmatcu/alpha-zero-general-with-Go-game/tree/master)
   * Halma: [davidschulte/alpha-thesis](https://github.com/davidschulte/alpha-thesis)
   * Chinesisches Schach: [NeymarL/ChineseChess-AlphaZero](https://github.com/NeymarL/ChineseChess-AlphaZero)
   * Hex, Gobang & Tic-Tac-Toe-3D: [MengYao-astro/alpha-zero-genral](https://github.com/MengYao-astro/alpha-zero-general)
   * Käsekästchen & Santorini: [alexis/alpha-zero-general](https://github.com/alexisbouley/alpha-zero-general)
   * Schach: [vdelale/AlphaZeroChess](https://github.com/vdelale/AlphaZeroChess)
   * Hex: [acebrahimian/alpha-zero-hex](https://github.com/aebrahimian/alpha-zero-hex)
   * Racing Kings: [AndrewSpano/Alpha-Zero-Racing-Kings](https://github.com/AndrewSpano/Alpha-Zero-Racing-Kings)
   * Splendor, Minivilles,… [cestpasphoto/alpha-zero-general](https://github.com/cestpasphoto/alpha-zero-general)
3. Klonen des GitHub-Repositories

### 4.2.2 Mögliche Anpassung der Struktur

1. Die Ordnerstruktur muss exakt so aufgebaut sein, wie in [4.1 Implementierung von Alpha-Zero-Spielen](#_4.1_Implementierung_von) beschrieben.
   * Das Spiel wird in Games/<game> verschoben
   * Darin befindet sich ein Keras und/oder PyTorch Ordner mit mindestens der NNet.py und der <Game>NNet.py. Zusätzliche genutzte Netzwerke sind möglich.
   * Wenn das Modell bereits trainiert ist, dann kann das best.pth.tar oder best.h5 Modell in den entsprechenden PyTorch oder Keras Ordner abgelegt werden.
   * Die Spieldateien müssen beide unter Games/ liegen: <game>Logic.py und <game>Game.py

### Anpassung der Imports und ggf. Logik

1. Je nach vorheriger Ordnerstruktur kann es sein, dass die Imports nicht mehr korrekt sind. Die Angabe der Dateipfade findet als absolute Pfade statt.

|  |  |
| --- | --- |
| Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  Automatisch generierte Beschreibung | Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.  Automatisch generierte Beschreibung |

1. Die Games/<game>/<Game>Game.py muss die Interface Bestimmungen von Tools/i\_game genau erfüllen. Dies muss überprüft werden. V. a. muss darauf geachtet werden, dass das Brett, welches in der <game>Game.py gehandhabt wird, ein Numpy-Array ist, nicht eine Instanz des Bretts aus der <game>Logic.py.

### 4.2.4 Training und Frontend Anbindung

1. Sollte noch kein trainiertes Modell vorliegen, dieses nicht zufriedenstellend agieren oder die Logik ans Interface angepasst worden sein, so muss nach den Schritten aus [4.1.5 Trainieren des Spiels](#_4.1.5_Trainieren_des_1) neu trainiert werden.
2. Die Frontendanbindung erfolgt nach selbigen Schritten wie in [4.1.6 Frontendanbindung](#_4.1.6_Frontendanbindung).

## Funktionserweiterung unserer Plattform

### Blunder

Die Heuristik, ob ein Zug ein „Blunder“ war oder nicht, basiert derweil darauf, dass unser trainiertes neuronales Netz herangezogen wird, auf dem Spielzustand zu allen validen Zügen ein Wahrscheinlichkeitsverteilung erzeugt. Der beste Zug hat die höchste Wahrscheinlichkeit, der schlechteste die niedrigste bzw. gar keine (= 0), falls die Kante im Spielbaum nicht abgelaufen wurde, weil sie nach unserem neuronalen Netz zu schlecht ist.

Aktuell wird aus all den Wahrscheinlichkeiten für den jeweiligen Spielzustand der Mittelwert genommen, und bei allen validen Zügen geschaut, ob die Wahrscheinlichkeit unterhalb dieses liegt. Falls ja, so ist es „Blunder“, falls nein, nicht.

Liegt nun der ausgeführte Zug des Spielers bzw. dessen KI innerhalb der Züge, die als „Blunder“ abgelegt sind, so wird dieser Zug in eine Liste aufgenommen.

Diese Liste wird dann an den Spieler geschickt, sobald die Berechnung für jeden seiner Spielzüge fertig ist.

Optimal ist die Heuristik noch nicht.

Und zwar haben sich im Laufe der Zeit folgende Punkte herauskristallisiert:

* + Natürlich hängt die Evaluation davon ab, wie gut die KI trainiert ist. Ist sie nur mittelmäßig trainiert, fällt auch die Evaluation schlechter aus. Gerade wenn man in der Lage ist als Spieler die KI öfters zu schlagen, und dann sich „Blunder“ anschaut, so wird einem deutlich: Erstens, ich bin in der Lage die KI zu schlagen, wie soll sie dann gut evaluieren können, ob mein Spielzug gut oder schlecht ist?!
  + Die Evaluation von „Blunder“ findet bei uns derweil erst nach Spielende statt, damit nicht zusätzliche Berechnungszeit während des Spiels geraubt wird. Gerade auf Spielstufe „hard“ ist das wichtig, da dort die Berechnungszeiten ohnehin an der Grenze sind.
  + Aus vorherigem Punkt ergab sich, dass wir die Suchtiefe nicht zu hoch eingestellt haben im Spielbaum, damit die Evaluation bei Anfrage nicht zu lange dauert – gerade bei Spielen wie Dame, wo teils 80-100 Spielzüge in einem Spiel stattfanden. Dadurch ist die Evaluation ohnehin schon ineffizienter.
  + Die KI spielt ggf. nach einem Muster. Nur weil ein Muster gut funktioniert und gewinnbringend ist, heißt es ja nicht, dass andere Spielmuster und -strategien nicht auch gewinnbringend oder gar gewinnbringender sind. „Blunder“ markert also Spielzüge an, die vielleicht brillant waren.

Hat man sich die AlphaGo-Dokumentation angeschaut, so hat der herausgeforderte Go-Champ Lee Sedol ja damals in dem einzigen Spiel, dass er gewann auch einige Spielzüge gemacht, die erstmal nicht klug wirkten, anders, als man konventionell die Züge gespielt hätte, sehr kreativ, was sich im Endeffekt als genial herausstellte. Nach unserer Heuristik wäre diese Art von Zügen damals also auch angemarkert worden, auch wenn sie zum Sieg führten, zum einzigen.

Was man künftig also noch umsetzen könnte:

1. Einerseits die Berechnung asynchron neben das aktiven Spiels laufen lassen, sodass wir auf höherer Tiefe im Monte Carlo Tree suchen können.
2. Eine andere Art von Heuristik (entweder komplett ablösend oder ergänzend)
   1. Bspw. die Spielzustände miteinander vergleicht, und schaut, binnen wie vielen Zügen auf einen eigenen Zug wie viele eigene bzw. gegnerische Spielsteine wegfallen.
   2. Diese muss jedoch generalisiert für alle Spiele gelten

### Hint

Um als Spieler dazuzulernen, seine eigene KI verbessern zu können oder die Heuristik seines nicht KI-basierten Algorithmus (Programmierung interaktiver Systeme) zu verbessern, könnte man eine weitere Funktion einbinden, und zwar „Hint“.

Wir haben nun „Blunder“ in unserem Repertoire. „Blunder“ zeigt einem allerdings nur verbesserungswürdige Züge an, weißt also nur auf „Probleme“ hin, zeigt aber keine Problemlösungen. Der Spielende bzw. Entwickler wird ein wenig allein gelassen bei der Evaluation, was denn stattdessen ein guter Zug wäre. So richtig gut lernt man oft ja erst durch Abschauen.

Da könnte „Hint“ ansetzen, welches dem Spielenden den Zug ausgibt bzw. anzeigt, welches unser neuronales Netzes als den Besten berechnet hat. Frontendseitig mit einem zusätzlichen Button im „Blunder-Fenster“, welches aufklappt, wenn man auf „Blunder“ klickt. Die Draw-Methoden müssten dahingehend angepasst werden, dass sie dieses Feld dann markieren. In der Konsole würde stattdessen der Zug dann einfach als Nachricht ankommen und ausgeprintet werden.

Da die Funktion ohnehin dann schon da ist, könnte man sie auch in ein laufendes Spiel einbinden, sodass wenn man keine Ahnung hat, wo man hinsetzen soll, man einfach „Hint“ befragen kann und das Spielfeld markiert / auf der Konsole als String angezeigt bekommt.

## (4.3 Erweiterung der Spielfeldoptionen und Konfigurationen

Die Plattform könnte erheblich von einer möglichen Konfiguration der Spielfeldgröße profitieren. Diese Anpassungsfähigkeit erweitert die Vielfalt der Spielvariationen und ermöglicht es, die typische Benutzererfahrung zu bereichern. Neben den klassischen 3x3 Tic-Tac-Toe oder 8x8 Othello könnten so auch kleinere oder größere Varianten implementiert werden. Dadurch erhalten die Nutzer die Gelegenheit, sich mit einer breiteren Palette an strategischen Herausforderungen auseinanderzusetzen. Eine dynamische Anpassung der Spielfeldgröße könnte sowohl einfachere Einstiegsvarianten für Anfänger als auch komplexere Szenarien für fortgeschrittene Spieler bieten.

Diese Flexibilität unterstützt ebenfalls das tiefere Verständnis der Nutzer für ihre eigenen KI-Implementierungen. Indem die Spieler gezwungen werden, sich an unterschiedliche Spielfeldgrößen und damit verbundene Spielstrategien anzupassen, werden sie dazu angeregt, ihre Algorithmen und Lösungsansätze kritisch zu hinterfragen und zu optimieren. Beispielsweise könnten bei einem größeren Spielfeld neue Optimierungsmethoden erforderlich sein, um die erhöhte Komplexität und die Vielzahl möglicher Spielzüge effizient zu bewältigen. Umgekehrt könnten kleinere Spielfelder neue, schnellere Lösungsansätze erfordern, die den Spielverlauf beschleunigen.

### 4.3.1 Anpassung der Spiele

Um die Vorteile der konfigurierbaren Spielfeldgrößen voll ausschöpfen zu können, müssen neue Spiele und bereits bestehende Lösungen entsprechend dynamisch angepasst werden. Dies bedeutet, dass die Spielalgorithmen und KI-Implementierungen so gestaltet sein müssen, dass sie flexibel auf verschiedene Spielfeldgrößen reagieren können.

1. Dynamische Spielbrettlogik implementieren. Eine Vorlage dafür ist in Games/connect4/Connect4Logic.py zu finden. Die \_\_init\_\_-Methode bekommt als Übergabe Parameter die Höhe (height) und Breite (width) des Spielfelds übergeben und kann so angepasst werden. Zusätzlich liegt bei fehlender Konfiguration ein Default Wert für beide Skalierungen vor (DEFAULT\_HEIGHT, DEFAULT\_WIDTH).
2. Dynamische Spiellogik implementieren. Die Games/<game>/<Game>Game.py muss ebenfalls die Möglichkeit haben, diese Größenparameter übergeben zu bekommen. Ein Beispiel ist ebenfalls bei Vier gewinnt unter Games/connect4/Connect4Game.py zu sehen.

### 4.3.2 Anpassung der JSON-Commands

Bei der Erstellung des Spiels muss ein zusätzlicher Parameter für den Command play mit dem Command-Key create hinzugefügt werden. Die Anpassung muss an verschiedenen Stellen stattfinden.

1. Es muss ein neuer Eintrag in Tools/Game\_Config vorgenommen werden. Hier benötigt man eine neue Python-Datei, die ein Enum implementiert und so die verschiedenen Spielfeldgrößen definiert.
2. Dieses Enum muss in den Fast-API-Server importiert und verwendet werden: Server/fastAPIServer.py.
3. Im Fast-Api-Server muss zudem das Kommando in der Methode handle\_play\_command akzeptiert und ausgelesen werden.

### 4.3.3 Anpassung im Frontend?)

## 4.4 Verbesserung der Benutzererfahrung

Die Einführung einer Anzeigetafel zur Darstellung erspielter Punkte oder das Integrieren eines Achievement-Systems mit Rängen und Erfolgen könnten das Nutzererlebnis erheblich verbessern.

Eine Punktetafel bietet Spielern eine Motivation in den direkten Vergleich mit anderen Teilnehmern zu treten und schafft somit einen zusätzlichen Anreiz, sich zu verbessern und höhere Punktzahlen zu erreichen.

Darüber hinaus könnte ein Achievement-System, das verschiedene Ränge und Erfolge beinhaltet, das Engagement der Nutzer weiter steigern. Durch das Erreichen bestimmter Meilensteine und das Freischalten von Belohnungen fühlen sich die Spieler für ihre Anstrengungen und ihre Fähigkeiten anerkannt. Dies könnte beispielsweise durch das Erreichen bestimmter Punktzahlen, das erfolgreiche Abschließen von herausfordernden Aufgaben oder durch kontinuierliches Spielen über einen längeren Zeitraum erfolgen. Solche Errungenschaften könnten mit speziellen Titeln, Abzeichen oder anderen virtuellen Belohnungen versehen werden.

### 4.4.1 Benutzeranmeldung, Datenmanagement und Datenschutz

Um die beschriebenen Funktionen effektiv umzusetzen, wäre die Einführung eines Benutzeranmelde- und Registrierungssystems erforderlich. Dadurch könnten individuelle Fortschritte und Erfolge jedem Nutzer eindeutig zugeordnet und dauerhaft gespeichert werden. Eine solche Lösung würde es ermöglichen, dass die Spieler ihre Punktestände und Achievements auch über mehrere Sitzungen hinweg beibehalten und von verschiedenen Geräten aus auf ihren Fortschritt zugreifen können.

Dabei ist es unerlässlich, ein robustes Datenverwaltungssystem zu integrieren, das nicht nur die Erfassung und Speicherung der Nutzerdaten, sondern auch deren Schutz gewährleistet. Datenschutzaspekte müssen dabei eine hohe Priorität haben, um die Privatsphäre der Spieler zu schützen und den gesetzlichen Vorgaben zu entsprechen. Dies könnte durch die Implementierung moderner Sicherheitsprotokolle, Verschlüsselungstechnologien und transparenter Datenschutzrichtlinien sichergestellt werden. Zudem sollten die Nutzer klar und verständlich über die Verwendung ihrer Daten informiert und ihnen Kontrollmöglichkeiten über ihre persönlichen Informationen eingeräumt werden.

### 4.4.2 Administratorenrollen und -rechte

Mit der Einführung eines Benutzeranmelde- und Registrierungssystems wird auch die Notwendigkeit entstehen, verschiedene Administratorenrollen zu definieren und entsprechende Rechte zu vergeben. Administratoren könnten verantwortlich sein für die Verwaltung der Nutzerdaten, das Überwachen und Moderieren von Inhalten sowie das Sicherstellen der Einhaltung von Richtlinien und Datenschutzbestimmungen. Durch differenzierte Zugriffsrechte kann sichergestellt werden, dass nur befugte Personen auf sensible Daten zugreifen oder Änderungen vornehmen können. Dies trägt nicht nur zur Sicherheit und Integrität der Plattform bei, sondern ermöglicht auch eine effiziente und strukturierte Verwaltung des Systems.

# 5 Testdokumentation

## 5.1 Teststrategie/-ansatz (Testmethoden)

## 5.2 Testplanung (Testumgebungen und -daten)

## 5.3 Testfälle/-szenarien (Beschreibung der Testfälle, erwartete Ergebnisse)

## 5.4 Testergebnisse/-berichte (Zusammenfassung, Fehler und Probleme Maßnahmen zur Fehlerbehebung, Retests)

# 6 Glossar (Begriffe, Verweise)

# 7 Index