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 |

|  |  |  |
| --- | --- | --- |
| Herausgeber | [Technische Hochschule Mittelhessen – FB06 Mathematik, Naturwissenschaften und Informatik](https://www.thm.de/mni/) | |
| Dateiname | Technische\_Dokumentation\_ver\_0.1 | |
| Dokumentenbezeichnung | Technische Dokumentation: Plattform zum Vergleich von Spiele-KIs | |
| Version | 0.1 | |
| Stand | Sonntag, 30. Juni 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 6](#_Toc170646738)

[1.1 Projektumfeld 6](#_Toc170646739)

[1.2 Organisatorisches Vorhaben 6](#_Toc170646740)

[1.3 Ziel des Projektes 6](#_Toc170646741)

[1.4 Beteiligte 6](#_Toc170646742)

[1.5 Dokumenteninhalt 7](#_Toc170646743)

[2 Projektübersicht 7](#_Toc170646744)

[3 Entwicklungsdokumentation 8](#_Toc170646745)

[4 Weiterentwicklungsdokumentation 9](#_Toc170646746)

[4.1 Modularität und Erweiterbarkeit 9](#_Toc170646747)

[5 Testdokumentation 10](#_Toc170646748)

Abbildungssverzeichnis

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

Tabellenverzeichnis

[Tabelle 1: Interne Beteiligung 7](#_Toc170646757)

# 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

# 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 Alpha Zero 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 und Abalone oder Abwandlungen der bereits integrierten Spiele, wie NoGo, GoBang oder 3D-Tic-Tac-Toe.

### 4.1.1 Auswahl des Spiels

Identifizierung des Spiels, das in das System integrieren 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 Selbstspiel 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 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 wird diese jedoch bevorzugt.
  + 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.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

### 4.1.3 Implementierung der Spiellogik

1. Implementierung des Spielbrettes und Aktionen auf das Spielbrett 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.

|  |  |
| --- | --- |
| 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 Rückgabewerte die entsprechenden Methoden haben müssen.

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

1. In Games/<game>/<Game>Game.py die Methoden des Tools/i\_game.py Interfaces implementieren. Diese Methoden werden von den Neuralen Netzen zum Trainieren verwendet und dienen zudem für die tatsächliche Spielausführung. Die Methoden müssen sich strikt an die Vorgaben des Interfaces halten. Dabei dürfen jegliche Hilfsmethoden verwendet werden. Dazu muss das Interface importiert und als Übergabeparameter an die Klasse weitergereicht werden.

|  |  |
| --- | --- |
| Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  Automatisch generierte Beschreibung | Marke 3 mit einfarbiger FüllungMarke 3 mit einfarbiger FüllungMarke 3 mit einfarbiger FüllungEin Bild, das Text, Screenshot, Schrift, Dokument enthält.  Automatisch generierte Beschreibung |

1. 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. Deshalb wird die ausschließliche Handhabung mit numpy und math bevorzugt.

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

1. 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.

|  |  |
| --- | --- |
| Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  Automatisch generierte Beschreibung | 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/man.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.
2. Sollte noch kein trainiertes Modell vorhanden sein, so kann ein Neues mit anzugebenden Parametern trainiert werden.
3. Nach der vollendeten Trainingsphase und Akzeptierung des neuen Modells, wird dieses automatisch in einen temporären Ordner Trainer/<game>\_<train\_model>/ gespeichert. Die Datei hat nach einem PyTorch Training die 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.
4. Schlussendlich muss das trainierte Modell in den entsprechenden Ordner verschoben werden: Games/<game>/<framework>/.

### 4.1.6 Frontend?

## 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 des Interfaces

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.

### 4.2.4 Training und Frontend Anbindung

1. Sollte noch kein trainiertes Modell vorliegen, diese nicht zufriedenstellend agieren oder die Logik geändert worden sein, so muss nach den Schritten aus [4.1.5 Trainieren des Spiels](#_4.1.5_Trainieren_des) neu trainiert werden.
2. Frontend?

## 4.3 Erweiterung der Spielfeldoptionen und Konfigurationen

Die Plattform könnte von einer möglichen Konfiguration der Spielfeldgröße profitieren. Sie erweitert die verschiedenen Spielvariationen und ermöglicht so die typische Benutzererfahrung zu erweitern. Neben den klassischen 3x3 Tic-Tac-Toe oder 8x8 Othello könnten so auch kleinere oder größere Varianten implementiert werden. Dies unterstützt ebenfalls das tiefere Verständnis der Nutzer für ihre eigenen KI-Implementierung, da Lösungsansätze und möglicherweise neue Optimierungsmethoden gefunden werden müssen.

### 4.3.1

## 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 verbessern und Belohnungen für gute Spiele und KI-Implementierungen ermöglichen.

# 5 Testdokumentation

1. Entwicklungsdokumentation
   1. Architektur & Design (System, Komponenten-, Modul- Datenflussdiagramme)
   2. Technologiestack (Sprachen, Frameworks, Bibliotheken, Datenbanken)
   3. Code-Struktur (Verzeichnisstruktur, Hauptklassen/-module, Namenskonventionen, Kodierungsstandards)
   4. Entwicklungsprozess (Scrum, Git-Workflows, Build-/Deployement-Prozess)
2. Weiterentwicklungsdokumentation
   1. Zukunft-Features
   2. Einbindung von neuen Features (mit Beispiel) -> Spiele (Backend und Frontend)
   3. Deployement
3. Testdokumentation
   1. Teststrategie/-ansatz (Testmethoden)
   2. Testplanung (Testumgebungen und -daten)
   3. Testfälle/-szenarien (Beschreibung der Testfälle, erwartete Ergebnisse)
   4. Testergebnisse/-berichte (Zusammenfassung, Fehler und Probleme Maßnahmen zur Fehlerbehebung, Retests)
4. Glossar (Begriffe, Verweise)
5. Index