



Entwicklungsdocumentation

Technische Dokumentation

Softwaretechnik-Projekt SoSe2024

Thema

Plattform zum Vergleich von Spiele-KIs: **KIMaster**



Technische Dokumentation

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	Schreiben	Max Bachmann
0.4	17.07.2024	3.1, 3.2, 4	Schreiben	Pascal Waldschmidt
0.5	17.07.2024	3.4, 4.3	Schreiben	Justine Buß, Thorben Jones, Maximilian Bachmann
0.6	19.07.2024	5	Schreiben	Justine Buß
0.7	20.07.2024	3	Schreiben	Pascal Waldschmidt
0.8	20.07.2024	3.1.4	Schreiben	Thorben Jones
0.9	21.07.2024	3.3, 3.4, 3.6, 4.1.3	Schreiben	Alex Roos
1.0	21.07.2024	3.1-3.3, 3.7, 5.2, 5.4	Schreiben	Sven Roman Reinhard
1.1	22.07.2024	3.7	Schreiben	Sven Roman Reinhard
1.2	23.07.2024	Alle	Nachbessern	Justine Buß
1.3	23.07.2024	7	Schreiben	Justine Buß
2.0	24.07.2024	6	Schreiben	Justine Buß

Technische Dokumentation

Herausgeber	Technische Hochschule Mittelhessen – FB06 Mathematik, Naturwissenschaften und Informatik	
Dateiname	Technische_Dokumentation_ver_2.0	
Dokumentenbezeichnung	Technische Dokumentation: Plattform zum Vergleich von Spiele-KIs: KIMaster	
Version	2.0	
Stand	Donnerstag, 25. 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
	Thorben Jones	thorben.jones@mni.thm.de
Kurzinfo	„Technische Hochschule Mittelhessen Softwaretechnik-Projekt. Technische Dokumentation zum Softwaretechnik-Projekt KIMaster“	



Technische Dokumentation

Inhaltsverzeichnis

1 Projektbeschreibung	9
1.1 Projektumfeld	9
1.2 Organisatorisches Vorhaben	9
1.3 Ziel des Projektes	9
1.4 Beteiligte	9
1.5 Dokumenteninhalt	10
2 Projektübersicht	10
3 Entwicklungsdokumentation	12
3.1 Architektur und Kommunikation	12
3.1.1 Server	12
3.1.2 GameClients	14
3.1.3 Webkomponente	15
3.1.4 Schnittstelle für externe Anbindung	16
3.1.5 Auto-Test-Komponente	17
3.1.6 Docker	19
3.2 Technologiestack	21
3.2.1 Sprachen	21
3.2.2 Frameworks	21
3.2.3 Bibliotheken	22
3.2.4 Werkzeuge und Sonstiges	23
3.2.5 Anmerkungen zur Kompatibilität	24
3.3 Code-Struktur	26
3.3.1 Namenskonventionen	26
3.3.2 Kodierungsstandards	26
3.3.3 Quellcode Referenz	26
3.3.4 Docker	27
3.3.5 Integration von Code aus externen Repositories	34
3.4 Trainingsprozess	36
3.4.1 Trainingseinstellungen	36



Technische Dokumentation

3.4.2 Trainingsdauer	37
3.4.3 Analyse der Trainingsergebnisse	38
3.5 Entwicklungsprozess	39
3.5.1 Zeit- und Aufgabenverfolgung mit Jira.....	39
3.5.2 Git-Workflows	41
3.5.3 Projektverlauf	43
3.6 Build-Prozess	45
3.6.1 Systemanforderungen	45
3.6.2 Installationsanweisungen.....	45
3.6.3 Hinweise zum Build Prozess	47
3.7 Deployment-Prozess	47
3.7.1 Einrichten des Servers.....	47
3.7.2 Code von GitHub laden	48
3.7.3 Pfad zum Docker-Socket anpassen.....	48
3.7.4 Programm in Rootles-Docker starten	48
3.7.5 Verfügbarkeit im THM-Netz prüfen	50
3.7.7 Probleme während des Deployments	50
4 Weiterentwicklungsdocumentation	52
4.1 Implementierung von AlphaZero-Spielen.....	52
4.1.1 Auswahl des Spiels	52
4.1.2 Anlegen der Projektordner und -dateien	53
4.1.3 Implementierung der Spiellogik	55
4.1.4 Erstellen des neuronalen Netzes	60
4.1.5 Trainieren des Spiels	62
4.1.6 Frontendanbindung	67
4.2 Integration von GitHub-Spielen	74
4.2.1 Auswahl des GitHub-Spiels	74
4.2.2 Mögliche Anpassung der Struktur	74
4.2.3 Anpassung der Imports und ggfs. Logik	74
4.2.4 Training und Frontend Anbindung	75



Technische Dokumentation

4.3 Funktionserweiterung unserer Plattform.....	76
4.3.1 Spielfeldkonfigurationen und Spielvariationen	76
4.3.2 Schwierigkeitsgrade und Spielmodi	77
4.3.3 Funktion Blunder	78
4.3.4 Funktion Hint	80
4.4 Verbesserung der Benutzererfahrung	80
4.4.1 Benutzeranmeldung, Datenmanagement und Datenschutz	81
4.4.2 Administratorenrollen und -rechte	81
4.4.3 Implementierung von Themes	82
4.4.4 Weitere Soundeffekte.....	83
5 Testdokumentation	85
5.1 Teststrategie/-ansatz	85
5.2 Testplanung (Testumgebungen und -daten)	85
5.2.1 Testdaten.....	85
5.2.2 Testdurchführung.....	86
5.2.3 Testunabhängigkeit	86
5.2.4 Testreihenfolge	87
5.3 Automatisierte Testfälle	88
5.4 Testergebnisse/-berichte	91
5.5 Ausführliche Fehlersuche und Bugfixing.....	92
6 Literaturverzeichnis	93
7 Glossar	96
7.1 Abkürzungsverzeichnis	96
7.2 Begriffserklärungen	97

Technische Dokumentation

Abbildungsverzeichnis

Abbildung 1: Architektur Übersicht.....	12
Abbildung 2: Serverarchitektur	12
Abbildung 3: GameClient Architektur	14
Abbildung 4: Webkomponente Architektur	15
Abbildung 5: Auto-Test Architektur	17
Abbildung 6: GameClient requirements.txt - Kompatibilität	24
Abbildung 7: Trainingszyklus	37
Abbildung 8: Kanban Board.....	39
Abbildung 9: Stundentracking	40
Abbildung 10: Worklogs	40
Abbildung 11: Sprintübersicht.....	41
Abbildung 12: GitHub-Repository.....	42
Abbildung 13: Projektverlaufsdigramm	43
Abbildung 14: GitHub klonen	46
Abbildung 15: Projektübersicht -> Games.....	53
Abbildung 16: Ordnerstruktur Spieleordner	54
Abbildung 17: Code Spielbrettimplementierung	55
Abbildung 18: Code i_game Interface	56
Abbildung 19: Code Spiellogikimplementierung.....	56
Abbildung 20: Code requirements.txt	57
Abbildung 21: Code PyGame Import in draw().....	57
Abbildung 22: Code Othello NNet.py.....	60
Abbildung 23: Code Othello OthelloNNet.py	60
Abbildung 24: Code Checkers NNet.py.....	61
Abbildung 25: Code Checkers CheckersNNet.py	61
Abbildung 26: Projektübersicht -> Trainer	62
Abbildung 27: Code Trainer/main.py.....	62
Abbildung 28: Trainer Spieleanwahl	63
Abbildung 29: Trainer Konfiguration	64
Abbildung 30: Trainingsphase	65
Abbildung 31: Trainingsmodell & -Zwischenstände	66
Abbildung 32: Checkers Trainingsmodell	66
Abbildung 33: Projektübersicht -> Frontend	67
Abbildung 34: Frontend/src/components/	67
Abbildung 35: Code Spieleanwahl	68
Abbildung 36: Code Sprachanpassung Spiel.....	68
Abbildung 37: Frontend/src/components/UI/PlayPage	69
Abbildung 38: Code Spieleimplementierung einfach.....	70

Technische Dokumentation

Abbildung 39: Code Spieleimplementierung atypisch - Mausposition	70
Abbildung 40: Code Spieleimplementierung atypisch - Schaltflächen	71
Abbildung 41: Code Spieleimplementierung atypisch - VUEX	71
Abbildung 42: Projektübersicht -> FrontendDebug	72
Abbildung 43: Frontend/Debug/src/components/WebSocketTest.vue.....	73
Abbildung 44: Code Importanpassung.....	75
Abbildung 45: Code Pit-Anpassung	78
Abbildung 46: Testergebnisse erfolgreich.....	91

Tabellenverzeichnis

Tabelle 1: Interne Beteiligung	10
Tabelle 2: Verwendete Sprachen	21
Tabelle 3: Verwendete Frameworks	21
Tabelle 4: Verwendete Bibliotheken.....	23
Tabelle 5: Verwendete Werkzeuge und sonstige Hilfsmittel.....	24
Tabelle 6: Trainingskonfigurationen und -zeiten	37
Tabelle 7: URLs.....	48
Tabelle 8: Pfadanpassung Docker-Socket	48
Tabelle 9: Testbeispiel.....	87
Tabelle 10: Testreihenfolge	87
Tabelle 11: Automatisierte Testfälle	90
Tabelle 12: Abkürzungsverzeichnis	96
Tabelle 13: Begriffs-Glossar	105



Technische Dokumentation

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 von KIMaster, einer benutzerfreundlichen Plattform, die es Nutzern ermöglicht, praxisnahe Erfahrungen mit künstlicher Intelligenz (KI) zu sammeln. KIMaster 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
Stellvertretende Projektleitung	Justine Buß	MNI/Ingenieur-Informatik	justine.buss@mni.thm.de
Teammitglied	Alexander Roos	MNI/Ingenieur-Informatik	alexander.roos@mni.thm.de
Teammitglied	Maximilian Bachmann	MNI/Informatik	maximilian.lars.bachmann@mni.thm.de
Teammitglied	Omar Karkotli	MNI/Informatik	omar.karkotli@mni.thm.de
Teammitglied	Sven Roman Reinhard	MNI/Informatik	sven.roman.reinhard@mni.thm.de
Teammitglied	Pascal Waldschmidt	MNI/Informatik	pascal.waldschmidt@mni.thm.de



Technische Dokumentation

Dozent	Prof. Dr. Frank Kammer	MNI	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 zur Verfügung gestellt, dass die Nutzung der bereitgestellten Webseite und der Schnittstelle für externe Anbindungen genauer erläutert.

2 Projektübersicht

Das gesamte Projekt [KIMaster](#) ist auf GitHub verfügbar, wo alle relevanten Dateien, einschließlich des Quellcodes, der Dokumentation und der Projektressourcen, bereitgestellt werden. Das Repository bietet Zugriff auf die aktuelle Version der Software sowie eine Übersicht über den Fortschritt und die Historie des Projekts. Hier können alle Teammitglieder und Interessierten die Entwicklung verfolgen, zur Weiterentwicklung beitragen und bei Bedarf Unterstützung leisten.

Das Repository enthält ebenfalls die ausführliche Projektdokumentation mit diesem Dokument, der technischen Dokumentation, dem Benutzerhandbuch, dem Pflichtenheft, das die detaillierten Informationen zu den funktionalen und qualitativen Anforderungen auf Basis des Lastenhefts definierte. Das Lasten- und Pflichtenheft dienten als verbindliche Grundlage für die Umsetzung des Projektes und sollten sicherstellen, dass alle Anforderungen und Ziele klar dokumentiert sind.

- Technische Dokumentation & Benutzerhandbuch: [Spezifikation/Dokumentation/](#)
- Pflichtenheft: [Spezifikation/Pflichtenheft/](#)
- Lastenheft: [Spezifikation/Lastenheft/](#)



THM

**CAMPUS
GIESSEN**

TECHNISCHE HOCHSCHULE MITTELHESSEN

MNI

Mathematik, Naturwissenschaften
und Informatik

Entwicklungsdocumentation

Entwicklungsdocumentation

Entwicklungsdocumentation

3 Entwicklungsdocumentation

3.1 Architektur und Kommunikation

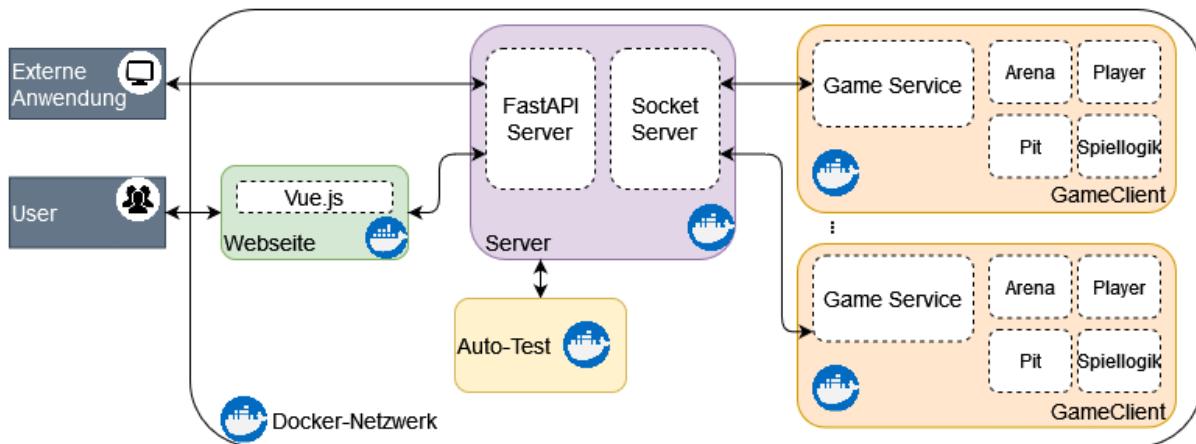


Abbildung 1: Architektur Übersicht

Die Architektur des Systems besteht aus mehreren miteinander verbundenen Komponenten, die innerhalb eines [Docker](#)-Netzwerks betrieben werden. Diese Struktur ist darauf ausgelegt, eine modulare, skalierbare und effiziente Verwaltung der verschiedenen Funktionseinheiten zu ermöglichen.

3.1.1 Server

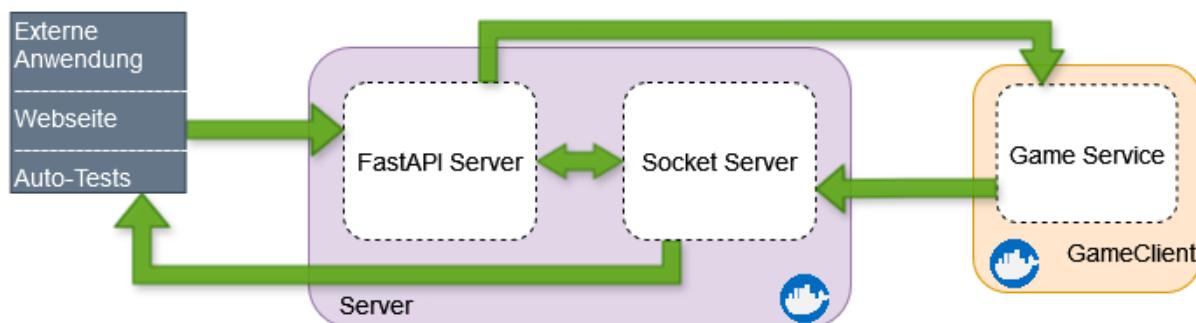


Abbildung 2: Serverarchitektur

Das Herzstück des Systems bildet der Server, der aus zwei Hauptkomponenten besteht:

Der [FastAPI](#) Server ist zuständig für die Bearbeitung von Nutzeranfragen und fungiert als zentrale Schnittstelle für die Kommunikation zwischen verschiedenen Systemkomponenten und externen Anwendungen. Obwohl FastAPI üblicherweise [HTTP](#)-basierte Anfragen unterstützt, erfolgt die

Entwicklungsdokumentation

Kommunikation in dieser Implementierung ausschließlich über [WebSockets](#). Diese Entscheidung ermöglicht eine bidirektionale Echtzeitkommunikation, die besonders wichtig ist für Anwendungen mit schnellen und kontinuierlichen Interaktionen, wie z.B. Online-Spiele. Zudem unterstützt WebSockets eine bessere Identifikation der Clients.

Der FastAPI Server hat folgende spezifische Aufgaben:

- **WebSocket-Verbindung initialisieren:** Bereitstellung einer WebSocket-Verbindung für die Kommunikation mit dem Webserver und extern angebundenen Nutzern.
- **Verbindungsmanagement:** Akzeptanz und Verwaltung neuer Nutzeranfragen in einer aktiven Verbindungsliste, sowie Aktualisierung bei Verbindungstrennungen.
- **Nachrichtenempfang und -verarbeitung:** Empfang und Dekodierung von [JSON](#)-Nachrichten. Zudem die Filterung basierend auf erlaubten Befehlen mit Verwerfung von unbekannten Befehlen.
- **Weiterverarbeitung von Kommandos:** Weiterverarbeitung der akzeptierten JSON-Nachrichten, die im FastAPI-Server gehandhabt werden können (beispielsweise die Spracheinstellungen)
- **Weiterleitung von Kommandos:** Weiterleitung von nicht im FastAPI Server behandelten Befehlen an den Socket Server.
- **Antworten senden:** Formatierung von Antwortnachrichten im JSON-Format für WebSocket-Clients und versenden der Nachrichten.

Der Socket Server ermöglicht die bidirektionale Kommunikation zwischen dem Server und den GameClients, also zwischen den Nutzern und den Lobbys, in denen die Spiele stattfinden. Die Kommunikation vom Socket Server aus findet ebenfalls über WebSockets statt.

Der Socket Server übernimmt folgende Aufgaben:

- **WebSocket-Verbindung initialisieren:** Bereitstellung einer WebSocket-Verbindung für die Kommunikation mit GameClients.
- **Nachrichtenverarbeitung:** Akzeptanz und Weiterverarbeitung von Nachrichten, die vom FastAPI Server weitergeleitet werden oder die von den GameClients zurückgesendet werden sollen.
- **Lobbymanagement:** Generierung individueller Lobby-Schlüssel, Instanziierung neuer Lobby-Instanzen, Zuweisung von Spielern zu aktiven Räumen, Verwaltung von Spielpositionen und Entfernung von leeren oder inaktiven Lobbys.
- **GameClients-Instanziierung und Bereinigung:** Erzeugung neuer GameClients bei Spielerstellung und Bereinigung von Docker-Instanzen bei Lobby-Entferungen.

Entwicklungsdokumentation

3.1.2 GameClients

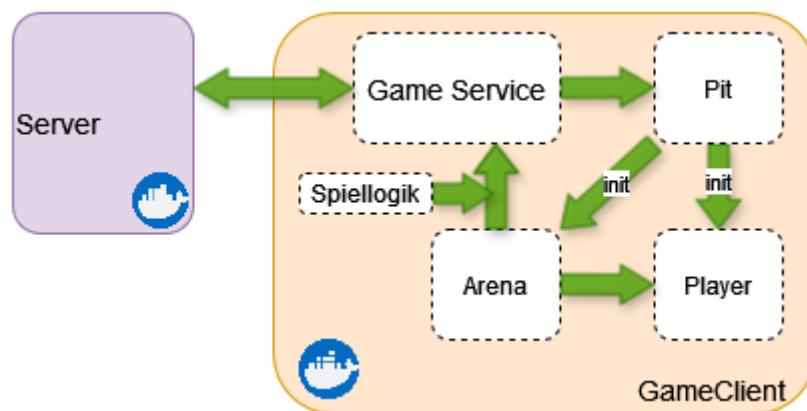


Abbildung 3: GameClient Architektur

Die GameClients bilden die Spielumgebung und bestehen aus mehreren untergeordneten Komponenten, die die eigentliche Spiellogik implementieren. Jeder GameClient umfasst die folgenden Hauptkomponenten:

Game Service: Der Game Service ist die zentrale Komponente eines GameClients, die die Koordination der verschiedenen Spielmodule übernimmt und die Kommunikation mit dem Server sicherstellt. Er hat somit verwaltende Instanz mit den anzusteuernenden WebSocket Endpunkten, als auch der anzusteuern Logik.

Pit: Pit hat ebenfalls verwaltenden Tätigkeiten. Er initialisiert den Player und die Arena, stoppt letztere auch gegebenenfalls und enthält zusätzliche Funktionalitäten, wie das Zug zurücknehmen oder die Zeitleiste.

Arena: Die Arena repräsentiert das Spielen selbst. Hier werden die Aktionen der Spieler und die Interaktionen innerhalb des Spiels verwaltet. Die Arena-Komponente interagiert direkt mit dem Game Service, um Spielaktionen zu verarbeiten und den aktuellen Spielstand zu aktualisieren, zudem fragt sie die aktiven Züge beim Player an.

Player: Der Player ist die Komponente, die den Spieler im Spiel repräsentiert. Er stellt also die Zugverwaltung dar und interagiert mit der Arena und anderen Komponenten, um die Aktionen des Spielers zu steuern und zu aktualisieren.

Spiellogik: Die Spiellogik-Komponente enthält die Regeln und Algorithmen, die das Spiel steuern. Sie ist eng mit der Arena, dem Player und der Pit verknüpft und sorgt dafür, dass das Spiel gemäß

Entwicklungsdokumentation

den definierten Regeln abläuft. Die Spiellogik stellt sicher, dass alle Spielaktionen korrekt interpretiert und ausgeführt werden.

3.1.3 Webkomponente

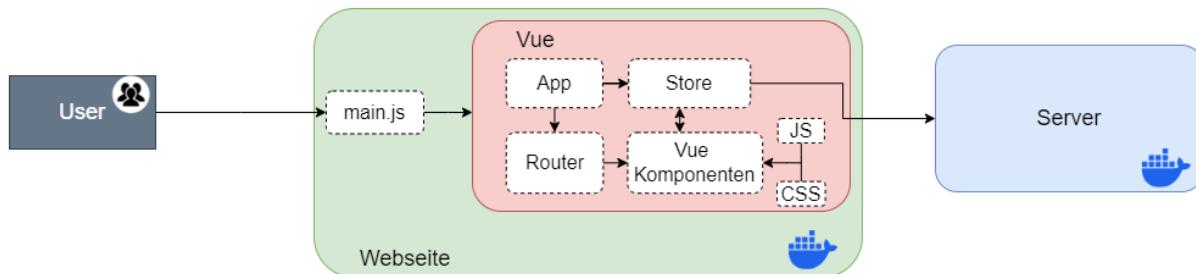


Abbildung 4: Webkomponente Architektur

Die Webseite ist als [Single Page Application \(SPA\)](#) konzipiert und bietet den Nutzern umfassenden Zugang zu den Funktionen des Projekts. In dieser Architektur fungiert die Webseite als grafische Benutzeroberfläche (GUI) für das zugrunde liegende Backend-System. Die Initialisierung der Webseite erfolgt durch eine HTTP-Anfrage im Browser, und ihre interne Struktur ist wie folgt aufgebaut:

Main.js: Dies ist der Einstiegspunkt der Anwendung. Hier wird das [Vue](#)-Projekt gestartet und mit dem [HTML](#)-Dokument verbunden. Main.js kümmert sich um die Grundkonfiguration und initialisiert die gesamte Vue-Anwendung.

App: Die Hauptkomponente der Vue-Anwendung übernimmt die zentrale Rolle, indem sie andere Komponenten initialisiert und die Sichtbarkeit von Komponenten über den Router View steuert.

Vue Komponenten: Diese sind die sichtbaren Bausteine der Webseite, die in verschiedene Kategorien unterteilt werden:

- **Seitenkomponenten:** Vollständige Seiten wie die Lobby oder die Spieleseite, die spezielle Funktionen und Inhalte darstellen.
- **Wiederverwendbare UI-Elemente:** Kleine, oft wiederverwendbare Elemente wie Pop-ups, Buttons oder Formulare, die in unterschiedlichen Teilen der Webseite erscheinen können.
- Styling und Logik sind in separaten [CSS](#)- und [JavaScript](#)-Dateien ausgelagert, um die Wartbarkeit und Lesbarkeit des Codes zu verbessern und eine klare Trennung von Präsentation und Funktionalität zu gewährleisten.



Entwicklungsdokumentation

Router: Der Router verwaltet die verschiedenen Ansichten der Anwendung, indem er die Namen der einzelnen Komponenten definiert und die Navigation zwischen diesen Ansichten ermöglicht. Dies geschieht innerhalb des Router Views, der die aktuelle Ansicht darstellt und es den Benutzern erlaubt, nahtlos zwischen den verschiedenen Seiten zu wechseln.

Store: Das State-Management-System, bekannt als [Vuex](#), verwaltet den globalen Zustand der Anwendung. Es sorgt dafür, dass der Zustand unabhängig vom aktuellen Routing konsistent bleibt und alle Komponenten auf denselben Status zugreifen können. Der Store enthält auch die WebSocket-Verbindung zum Backend-Server, die als zentrale Kommunikationsstelle fungiert und den Austausch von Daten in Echtzeit ermöglicht.

Für Entwickler steht zusätzlich eine Debug-Webseite zur Verfügung, die bei Bedarf in einem separaten Container gestartet werden kann. Diese Debug-Webseite bündelt alle Funktionalitäten in einer einzigen Vue-Komponente und verwendet weder Router noch Store. Dies erleichtert das Testen und Debuggen, da alle Funktionen an einem Ort verfügbar sind und keine komplexen Abhängigkeiten bestehen.

Beide Versionen der Webseite – die Produktionsversion und die Debug-Version – werden durch separate [Node.js](#) HTTP-Server bereitgestellt. Diese Server laufen in unabhängigen Docker-Containern, was eine hohe Flexibilität und Skalierbarkeit gewährleistet und gleichzeitig eine klare Trennung zwischen den verschiedenen Entwicklungs- und Produktionsumgebungen ermöglicht.

3.1.4 Schnittstelle für externe Anbindung

Die KIMaster WebSocket Schnittstelle wurde entwickelt, um eine umfassende und flexible Kommunikationsschnittstelle zu einem WebSocket-Server bereitzustellen. Neben der davon bereitgestellten Webseite ermöglicht dies ebenfalls den direkten Zugriff und die Steuerung des Systems über eine programmatische Schnittstelle, ohne dass die Webseite aufgerufen werden muss. Dies unterstützt eine nahtlose Interaktion und Integration mit externen Anwendungen oder Terminal-basierten Systemen.

Diese Schnittstelle bietet ebenfalls eine robuste Methode zur Verwaltung von Verbindungen, Lobbys und Spieloperationen und ermöglicht es Entwicklern, die Funktionen der Webanwendung programmatisch zu nutzen und zu erweitern. Die API ist vollständig unabhängig von der Webseite, was bedeutet, dass alle erforderlichen Operationen – wie das Erstellen und Beitreten von Lobbys, das Starten und Steuern von Spielen – direkt über die API erfolgen können. Dies ist besonders nützlich für Szenarien wie KI-gesteuerte Spiele, bei denen Interaktionen ohne direkte Benutzeroberfläche erforderlich sind.

Entwicklungsdokumentation

Durch die gewählte Architektur des FastAPI Servers in Kombination mit der WebSocket-Anbindung ist keine separate Handhabung für externe Anbindungen im Vergleich zur Anbindung über den Webserver erforderlich. Dies bedeutet, dass der Server sowohl Anfragen, die typischerweise von der Webanwendung kommen, als auch WebSocket-Verbindungen, die von externen Anwendungen oder Terminal-basierten Systemen aufgebaut werden, über eine einheitliche Schnittstelle verwaltet.

Diese Architektur minimiert den Aufwand für die Implementierung und Wartung von separaten Schnittstellen für unterschiedliche Verbindungstypen. Sowohl die Webanwendung als auch externe Clients interagieren über denselben Endpunkt, was die Integration und Erweiterung der API vereinfacht.

Dank der asynchronen Natur von FastAPI und der robusten Unterstützung für WebSockets können alle Verbindungen – ob von der Webseite oder von externen Quellen – effizient und ohne zusätzliche Komplexität in einem einheitlichen Verbindungsmanagement-System behandelt werden. Dies reduziert den Entwicklungsaufwand und gewährleistet, dass alle Verbindungen mit denselben Prozessen und Sicherheitsmechanismen überwacht und gesteuert werden.

3.1.5 Auto-Test-Komponente

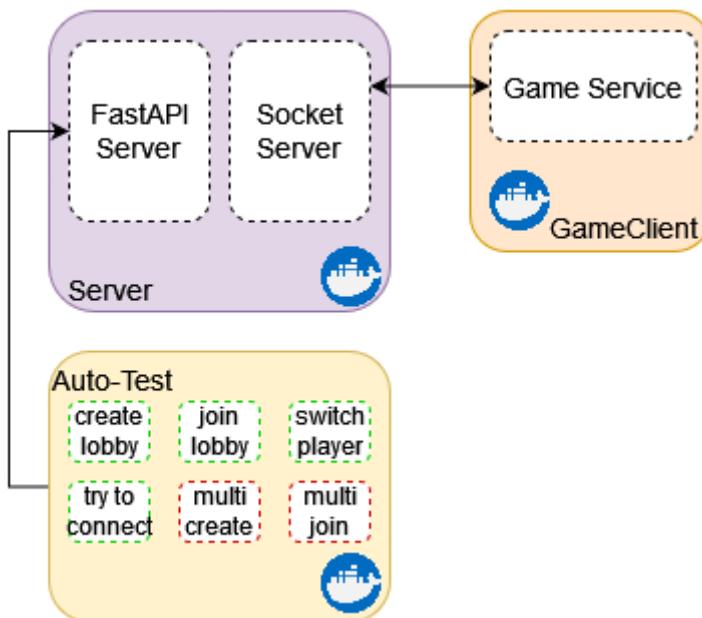


Abbildung 5: Auto-Test Architektur

Die Auto-Test-Komponente ist ein entscheidender Bestandteil der Qualitätssicherung innerhalb des KIMaster-Systems. Sie führt automatisierte Tests durch, die innerhalb des Docker-Netzwerks

Entwicklungsdokumentation

ablaufen und direkt über WebSockets mit dem Server kommunizieren. Diese automatisierten Tests sind darauf ausgelegt, die Funktionalität aller Systemkomponenten gründlich zu überprüfen. Sie stellen sicher, dass alle Teile des Systems korrekt arbeiten und die Gesamtstabilität gewahrt bleibt.

Die Tests überprüfen umfassend verschiedene Aspekte des Systems, von der Server-API über die GameClient-Instanzen. Durch die direkte Kommunikation über WebSockets wird sichergestellt, dass die Tests unter Bedingungen durchgeführt werden, die den realen Einsatz des Systems möglichst genau simulieren. Diese Echtzeit-Kommunikation ermöglicht eine präzise und schnelle Rückmeldung zu den Testergebnissen.

Die Auto-Test-Komponente wird dabei als normaler Client über die WebSocket-Verbindung angebunden. Diese Integration wird durch die Generalisierung des FastAPI Servers ermöglicht, der flexibel genutzt werden kann, um verschiedene Zwecke, einschließlich der Durchführung von Tests, zu unterstützen. Durch diese Vorgehensweise können die Tests in einer Umgebung durchgeführt werden, die realistischen Bedingungen nahekommt, was die Effektivität der Tests weiter erhöht.

Entwicklungsdokumentation

3.1.6 Docker

KIMaster ist eine Multi-Container-Anwendung, die auf Docker-Containern basiert. Die Architektur besteht aus mehreren spezialisierten Containern, die jeweils eine bestimmte Funktion innerhalb des Gesamtsystems übernehmen. Diese Container arbeiten zusammen, um die verschiedenen Anforderungen und Aufgaben der Anwendung zu erfüllen. Die Container sind:

- **swtp-server:** Stellt die zentrale Serverkomponente der Anwendung dar. Er ist verantwortlich für die Bereitstellung der API, über die externe und interne Anfragen bearbeitet werden. Darüber hinaus verwaltet der Server alle GameClient Instanzen, die im System aktiv sind. Dies umfasst das Management von Verbindungen, das Routing von Nachrichten und das Verwalten von Spielzuständen.
- **swtp-game-client:** Stellt eine einzelne Instanz eines Spielclients dar, der innerhalb einer Lobby agiert. Bei Erstellung einer neuen Lobby durch den swtp-server wird automatisch eine neue Instanz des swtp-game-clients gestartet. Jede swtp-game-client Instanz erhält vom swtp-server einen eindeutigen Namen zur Identifikation und Zuordnung zu den Nutzern in den Lobbys.
- **swtp-frontend:** Enthält den Webserver, der die Benutzeroberfläche (Webseite) für die Endanwender bereitstellt.
- **swtp-frontend-debug:** Stellt eine kompakte Weboberfläche bereit, die speziell für Entwicklungs- und Debugging-Zwecke konzipiert ist. Diese Umgebung bietet Entwicklern zusätzliche Tools und Funktionen, um die Anwendung während der Entwicklung zu überwachen und zu testen.
- **swtp-test:** Ist zuständig für die Durchführung der automatisierten Tests. Beim Start dieses Containers werden automatisch Smoke-Tests ausgeführt, um die Grundfunktionen der Anwendung auf deren Stabilität und Funktionalität zu überprüfen.

Der Orchestrierung der Container wird durch [Docker-Compose](#) umgesetzt. Docker-Compose ermöglicht die Definition und Verwaltung mehrerer Container als Teil eines gemeinsamen Projekts. Die zugehörige Konfigurationsdatei `docker-compose.yml` befindet sich im Root-Verzeichnis des KIMaster-Projekts und enthält die notwendigen Definitionen und Konfigurationen für die Container und deren Netzwerkanbindung.

- **Netzwerk:** Alle Container laufen in einer gemeinsamen Netzwerkumgebung, die in Docker-Compose als `swtp-net` definiert ist. Dieses Netzwerk ermöglicht eine reibungslose Kommunikation zwischen den verschiedenen Containern und stellt sicher, dass die einzelnen Services effizient miteinander interagieren können.



Entwicklungsdokumentation

- **Skalierung und Flexibilität:** Docker-Compose ermöglicht es auch, die Anzahl der Instanzen für bestimmte Container (z. B. mehrere GameClient-Instanzen) dynamisch zu skalieren, um den Anforderungen und der Last des Systems gerecht zu werden.

Durch den Einsatz von Docker-Containern und Docker-Compose wird sichergestellt, dass KIMaster modular, skalierbar und einfach zu verwalten ist. Das ermöglicht eine flexible Entwicklung und effiziente Bereitstellung der Anwendung in verschiedenen Umgebungen.

Entwicklungsdokumentation

3.2 Technologiestack

3.2.1 Sprachen

Backend		
Python	3.11	Implementierung der Backend-Logik, einschließlich der Verwaltung von WebSocket-Verbindungen und der Integration verschiedener Bibliotheken für maschinelles Lernen und Bildverarbeitung. Zudem die Implementierung der Spielelogiken.
Frontend		
CSS		Aussehen der Webseite wird angepasst, um das Nutzererlebnis zu verbessern.
HTML	HTML5	Gibt dem Browser an welche Elemente dargestellt werden sollen, wird dynamisch durch Vue beziehungsweise durch JavaScript geändert.
JavaScript / TypeScript		Dynamische Anpassung des HTML DOMs, Implementierung der in den Vue-Komponenten definierten Logik innerhalb des Browsers.

Tabelle 2: Verwendete Sprachen

3.2.2 Frameworks

Backend		
AlphaZero -Framework		Bildet die Grundlage für die Implementierung und Erweiterung der KI-Komponenten durch das Training von neuronalen Netzen zur Lösung komplexer Spiele.
pytest	8.2.1	Zur Erstellung und Ausführung von Testfällen und Testskripten für die Validierung und Qualitätssicherung der Kommunikation.
TensorFlow	2.14.0	Für maschinelles Lernen und neuronale Netzwerke, die für die Implementierung und das Training von KI-Modellen verwendet wird.
Frontend		
Vue.js	3.4.21	Grundlage der Webseite. Bietet die Möglichkeit die Webseite als reaktive Single Page Application (SPA) zu realisieren

Tabelle 3: Verwendete Frameworks



Entwicklungsdokumentation

3.2.3 Bibliotheken

Backend		
colorama	0.4.6	Farbige Ausgaben in der Konsole des Trainers.
docker	7.0.0	Interaktion mit der Docker-API, um Container zu erstellen, zu verwalten und zu steuern.
fastapi	0.111.0	API-Framework zur Erstellung von robusten und schnellen Web-APIs, das eng mit Python verwendet wird, um Endpunkte und die WebSocket-Verbindung zu definieren.
h5py	3.8.0	Arbeit mit HDF5-Dateien zur Speicherung und Verwaltung von Datenstrukturen in der Backend-Anwendung.
iniconfig	2.0.0	Handhabung von Konfigurationsdateien und -einstellungen.
keras		Aufbauend auf TensorFlow mit Abstraktion vieler Details für den Nutzer. Es ermöglicht das einfache Erstellen neuronaler Netze mit wenigen Befehlen.
numpy	1.26.4	Numerische Berechnungen, die für komplexe mathematische Operationen und Datenmanipulationen in der Spiellogik und -implementierung angewandt wird.
packaging	24.0	Handhabung von Verpackungsinformationen und -strukturen.
pandas		Datenmanipulation und -analyse, die für die Verarbeitung und Analyse großer Datensätze in der Spiellogik verwendet wurde. Momentan ist keine Nutzung vorgesehen.
pluggy	1.5.0	Management von Plugins und Erweiterungen, für die Erweiterungsfähigkeit und Flexibilität der Anwendung.
pygame	2.5.2	Für die Zeichnungen der Spielbretter für die spätere Nutzung auf der Webseite.
pytorch		Zur Entwicklung und zum Training von neuronalen Netzen, was im Rahmen des AlphaZero-Frameworks eingesetzt wird.
requests	<2.32.0	Senden von HTTP-Anfragen an externe Ressourcen für die Kommunikation mit externen Diensten und APIs.
starlette	0.37.2	ASGI-Toolkit, das als Basis für FastAPI dient und schnelle asynchrone Serveranwendungen ermöglicht.
torch	2.2.2	Für maschinelles Lernen und die Entwicklung und das Training von neuronalen Netzen.
torchvision	0.17.0	Torchvision ist eine Python-Bibliothek, die als Erweiterung von PyTorch vorgefertigte Datensätze, Bildtransformationen und vortrainierte Modelle für Computer Vision bereitstellt.

Entwicklungsdokumentation

tqdm	4.66.4	Anzeige von Fortschrittsbalken und Visualisierung von Prozessen im Trainer.
uvicorn	0.29.0	ASGI-Server, der mit FastAPI verwendet wird, um asynchrone Web-Anwendungen zu unterstützen.
WebSockets	12.0	Für die WebSocket-Kommunikation, die für die Echtzeitkommunikation und den Datenaustausch zwischen Frontend und Backend verwendet wird.
Frontend		
qrcode	1.5.3	Ermöglicht das optionale Darstellen des Lobby Schlüssels als QR-Code zur verbesserten Unterstützung von Mobilgeräten.
Vuex	4.1.0	State Management in Vue.js-Anwendungen, dass die Verwaltung von Zuständen (States), die über mehrere Komponenten hinweg geteilt werden müssen, erleichtert.
Vue Router	4.3.2	Routing in Vue.js-Anwendungen, dass die Navigation zwischen verschiedenen Ansichten (oder Seiten) in einer Single-Page-Anwendung (SPA) ermöglicht.
vue-i18n	9.13.1	Ermöglicht die Unterstützung mehrerer Sprachen und die Lokalisierung von Texten und Inhalten in der Anwendung. Vue-i18n bietet Mechanismen zur Übersetzung von Texten und zur dynamischen Anpassung der Anwendungssprache basierend auf Benutzereinstellungen oder anderen Kriterien.
Jsdoc-vuejs	4.0.0	Unterstützung von vue Dateien und Komponenten für JSDoc

Tabelle 4: Verwendete Bibliotheken

3.2.4 Werkzeuge und Sonstiges

Cisco Secure Client	5.0.00907	VPN-Verbindung zur Universität beim Deployment.
Docker	26.1.4	Zur Containerisierung und Verwaltung der einzelnen Softwarekomponenten.
Docker Compose	V2.27.1-desktop.1 & 1.29.2	Zur Orchestrierung und Verwaltung mehrerer Docker-Container, das für die Bereitstellung und Integration von mehreren Anwendungskomponenten und -services verwendet wird.
Docker Desktop	4.31.1	Benutzeroberfläche für die Verwaltung von Docker-Containern und -Images.
DocStrings		Verwendung zur teilweisen Dokumentation des Quellcodes.
Git/GitHub	2.45.2	Versionierung und Dateimanagement.

Entwicklungsdokumentation

HTTP-Server	10.5.0	Node.js Server, welcher die Webseite nach einer HTML-Anfrage für den Browser bereitstellt.
JavaScript Object Notation (JSON)		Datenaustauschformat, das effiziente Zugriffe auf Daten ermöglicht und ein weit verbreiteter Standard ist.
Jira		Projektmanagementtool.
PuTTY	0.81	Zur SSH-Verbindung mit dem Server beim Deployment.
Sphinx		Software-Dokumentationswerkzeug, zur Dokumentation des Quellcodes.
JSDoc	4,0,3	Dokumentationswerkzeug zur Dokumentation des Frontend Codes.
Vite	5.2.8	Kombiniert und optimiert die verschiedenen Vue Komponenten, um das deployte Produkt via Minimierung zu verbessern. Dient ebenfalls als Development Server um die Entwicklung der Frontend-Implementierung zu erleichtern.

Tabelle 5: Verwendete Werkzeuge und sonstige Hilfsmittel

3.2.5 Anmerkungen zur Kompatibilität

▼	Server	1	<code>h5py~=3.8.0</code>
	└ Dockerfile	2	<code>fastapi==0.111.0</code>
	└ connection_manager.py	3	<code>uvicorn==0.30.1</code>
	└ docker_api.py	4	<code>docker~=7.0.0</code>
	└ fastAPIServer.py	5	<code>requests < 2.32.0</code>
	└ lobby.py	6	<code>pygame==2.5.2</code>
	└ lobby_manager.py	7	<code>numpy==1.26.4</code>
	└ requirements.txt	8	<code>tensorflow~=2.14.0</code>
	└ socketServer.py	9	<code>torch~=2.2.2</code>
	└ start.py	10	<code>tqdm~=4.66.4</code>
		11	<code>pandas~=2.2.2</code>
		12	<code>torchvision~=0.17.0</code>

Abbildung 6: GameClient requirements.txt - Kompatibilität

Die `Server/requirements.txt` wurden so angepasst, dass sie plattformübergreifend auch mit der Apple Silicon Chip-Generation kompatibel ist. Dies betrifft insbesondere die Module TensorFlow und h5py, bei denen es Kompatibilitätsprobleme gab. Auch durch Dockerisierung wurde dies nicht vollständig behoben. So muss beispielsweise h5py in der Version 3.8.0 an erster Stelle in der `requirements.txt` stehen, um zu verhindern, dass es als Submodul von TensorFlow in einer anderen Version geladen wird. Dies könnte zu Fehlern beim Docker-Build



Entwicklungsdocumentation

führen. Daher sollte die Reihenfolge in der requirements.txt beibehalten werden, insbesondere h5py ganz oben.

Entwicklungsdocumentation

3.3 Code-Struktur

3.3.1 Namenskonventionen

- **Klassen:** Verwendung von CamelCase
 - z.B. AbstractConnectionManager, SocketServer
- **Methoden/Funktionen:** Verwendung von snake_case
 - z.B. create_app(), send_response()
- **Attribute:** Verwendung von snake_case
 - z.B. active_connections, msg_builder

3.3.2 Kodierungsstandards

- **Dokumentation:** Klassen und Methoden sind mit DocStrings dokumentiert, die ihre Funktion und Parameter beschreiben.
- **Typannotationen:** Verwenden Typannotationen für Parameter und Rückgabewerte, um die Lesbarkeit und Fehlervermeidung zu verbessern.
- **Imports:** Klare Trennung zwischen externen Bibliotheken und internen Modulen.
- **Fehlerbehandlung:** Umfangreiche Fehlerbehandlung, insbesondere bei WebSocket-Verbindungen und Docker-Interaktionen.

3.3.3 Quellcode Referenz

Unsere Projektdokumentation wurde erweitert, um die automatisch generierte API-Dokumentation einzubeziehen, die mit Sphinx erstellt wurde. Diese Sphinx-Dokumentation bietet eine umfassende und detaillierte Beschreibung des Quellcodes und der API-Funktionalitäten. Sie umfasst folgende Inhalte:

- **Modulübersicht:** Eine detaillierte Übersicht über alle Module, die in unserem Projekt enthalten sind.
- **Klassen- und Methodenbeschreibungen:** Ausführliche Erläuterungen zu allen Klassen und Methoden, einschließlich der Parameter und Rückgabewerte.

Entwicklungsdokumentation

- **Quellcodebeispiele:** Praktische Beispiele und Code-Snippets, die die Nutzung der API demonstrieren.
- **Zusätzliche Anmerkungen und Dokumentation:** Hintergrundinformationen und Erläuterungen, die das Verständnis der API und ihrer Verwendung erleichtern.

Backend-Dokumentation: Die Sphinx-Dokumentation ist in einem separaten HTML-Format verfügbar und über die im Repository befindliche `index.html`, zu finden im Ordner `Spezifikation/Dokumentation/Quellcode/index.html`, aufgerufen werden.

Frontend-Dokumentation: Die JSDoc-Dokumentation ist in einem separaten HTML-Format verfügbar und über die im Repository befindliche `index.html`, zu finden im Ordner `Spezifikation/Dokumentation/Frontend/index.html`, aufgerufen werden.

@FrontendFertig

3.3.4 Docker

3.3.4.1 Docker Compose

Die Interaktion der Container untereinander wird durch Docker-Compose verwaltet. Die dazu benötigte Datei `docker-compose.yml` ist im Root-Verzeichnis des KIMaster Projekts zu finden. Diese Docker-Compose-Datei definiert und konfiguriert mehrere Services, die in einer gemeinsamen Netzwerkumgebung (swtp-net) ausgeführt werden. Folgend werden die Netzwerke und Services vorgestellt:

Netzwerke

- **swtp-net:** Dies ist das Netzwerk, in dem alle Services kommunizieren. Es ermöglicht eine einfache Vernetzung der Container untereinander.

Services

1. **swtp-server:** Dies ist der Server-Service, der als Hauptkomponente fungiert.

build:

context: Gibt den Kontext des Builds an, in diesem Fall das aktuelle Verzeichnis `(. /)`. Der Kontext ist so gewählt, weil das Image des Servers auch das Verzeichnis `./Server` und `./Tools` enthalten muss, damit der Server funktionsfähig ist. Dazu muss der Kontext von Hand so konfiguriert werden, dass `./Tools` im Kontext des `swtp-server` liegt.

Entwicklungsdocumentation

dockerfile: Pfad zur Dockerfile des Servers (`./Server/Dockerfile`).

image: Name des erzeugten Docker-Images (`swtp-server-img`).

container_name: Der Name des Containers (`swtp-server`).

environment:

SERVER_HOST: Hostname des Servers (`swtp-server`).

SERVER_PORT: Port des Servers (8010).

NETWORK: Netzwerkname (`swtp-net`).

WORKER: Anzahl der Worker (1).

KEYLEN: Länge der Lobby-Schlüssel (5).

ports: Mappt den Port 8010 des Hosts auf den Port 8010 des Containers.

networks: Nutzt das benannte Netzwerk `swtp-net`.

volumes: Mappt das Docker-Socket-Volume auf `/var/run/docker.sock`. Der Pfad des Docker-Socket auf dem Hostrechner muss in der Datei `.env` konfiguriert werden (Default: `DOCKER_SOCK_PATH=/var/run/docker.sock`)

deploy:

resources:

limits:

cpus: CPU-Beschränkung auf 0. (0 = Die Leistung wird auf alle verfügbaren CPUs gleichmäßig verteilt.)

2. **swtp-game-client:** Dies ist der Game-Client-Service. Für jede Lobby erzeugt der swtp-server eine Instanz dieses Services. Bei der Erstellung wird jeder Instanz ein eindeutiger Name zugeordnet so kann der Server jeden Game-Client eindeutig identifizieren und jeder Lobby korrekt die WebSocket-Verbindungen der User zuordnen.

build:

context: Gibt den Kontext des Builds an, in diesem Fall das aktuelle Verzeichnis (`./`). Hier gilt die gleiche Begründung wie im `swtp-server`

dockerfile: Pfad zur Dockerfile des Game-Clients (`./GameClient/Dockerfile`).

Entwicklungsdocumentation

image: Name des erzeugten Docker-Images (`game-client-img`).

networks: Nutzt das benannte Netzwerk `swtp-net`.

depends_on: Startet zuerst den (`swtp-server`), falls dieser noch nicht läuft, danach `swtp-game-client`.

3. **swtp-frontend:** Dies ist der Frontend-Service, der die User-Webseite bereitstellt.

build:

context: Gibt den Kontext des Builds an, in diesem Fall das aktuelle Verzeichnis (`./`).

dockerfile: Pfad zur Dockerfile des Frontends (`./Frontend/Dockerfile`).

image: Name des erzeugten Docker-Images (`frontend-img`).

restart: Immer neustarten, sobald der Container gestoppt wurde (`always`). Dies kann zu unleserlichen Logs führen und das Debuggen erschweren, falls der Container in einem Absturz-Restart-Loop steckt. Zum Debuggen sollte diese Zeile auskommentiert werden.

container_name: Der Name des Containers (`swtp-frontend`).

networks: Nutzt das benannte Netzwerk (`swtp-net`).

depends_on: startet zuerst den (`swtp-server`), falls dieser noch nicht läuft, danach `swtp-frontend`.

ports: Mappt den Port 8086 des Hosts auf den Port 8080 des Containers. Dies bedeutet, dass alle Anfragen, die an den Port 8086 des Hosts gesendet werden, an den Port 8080 des `swtp-frontend` Containers weitergeleitet werden. Dies ermöglicht den Zugriff auf den Frontend-Service über den Host-Port 8086.

- **Was tun, wenn der Port 8086 bereits belegt ist?**

Wenn der Port 8086 auf dem Host-System bereits von einem anderen Dienst belegt ist, führt dies zu einem Portkonflikt, und Docker wird den Container nicht starten können. In diesem Fall gibt es mehrere Möglichkeiten, das Problem zu beheben.

4. **swtp-frontend-debug:** Dies ist der Debug-Frontend-Service. Er stellt ein Frontend bereit, dass nur für die Entwickler gedacht ist.

build:

Entwicklungsdocumentation

context: Gibt den Kontext des Builds an, in diesem Fall das aktuelle Verzeichnis (./).

dockerfile: Pfad zur Dockerfile des Debug-Frontends (./FrontendDebug/Dockerfile).

image: Name des erzeugten Docker-Images (frontend-debug-img).

restart: Immer neu starten (always).

container_name: Der Name des Containers (swtp-frontend-debug).

networks: Nutzt das benannte Netzwerk (swtp-net).

depends_on: startet zuerst den (swtp-server), falls dieser noch nicht läuft, danach swtp-frontend-debug.

ports: Mappt den Port 8087 des Hosts auf den Port 8080 des Containers. Dies bedeutet, dass alle Anfragen, die an den Port 8087 des Hosts gesendet werden, an den Port 8080 des swtp-frontend Containers weitergeleitet werden. Dies ermöglicht den Zugriff auf den Frontend-Service über den Host-Port 8087.

- **Was tun, wenn der Port 8087 bereits belegt ist?**

Wenn der Port 8087 auf dem Host-System bereits von einem anderen Dienst belegt ist, führt dies zu einem Portkonflikt, und Docker wird den Container nicht starten können. In diesem Fall gibt es mehrere Möglichkeiten, das Problem zu beheben.

5. **swtp-test:** Dies ist der Test-Service, er dient, als smoke Test.

build: Pfad zum Build-Kontext (./Test).

image: Name des erzeugten Docker-Images (swtp-test-img).

container_name: Der Name des Containers (swtp-test).

networks: Nutzt das benannte Netzwerk (swtp-net).

depends_on: startet zuerst den (swtp-server), falls dieser noch nicht läuft, danach swtp-test.



Entwicklungsdocumentation



Entwicklungsdokumentation

3.3.4.2 Relevante Dateien für Docker

Relevante Dateien für Docker: Neben Docker-Compose spielen auch die einzelnen Container eine wichtige Rolle. Der folgende Dateibaum enthält alle Dateien und Ordner die explizit zum Bauen der Docker Images benötigt werden, und darum nicht einfach entfernt oder umbenannt werden sollten. Die Dateien `start.cmd` und `start.sh` werden nicht von Docker selbst benötigt, sie dienen dazu, dass man nicht die Docker-Befehle zum Stoppen, Bauen und Starten einzutippen muss, stattdessen braucht man nur diese eine Datei auszuführen. Der In dem abgebildeten Dateibaum stellt das Verzeichnis mit dem Namen ":" das Root-Verzeichnis, welches in der Datei `docker-compose.yml` auch als Build-Kontext für alle Services außer den `swtp-test` festgelegt wurde, des Projektes dar.

```
.env
Frontend
├── .dockerignore
├── Dockerfile
└── package.json
FrontendDebug
├── .dockerignore
├── Dockerfile
└── package.json
GameClient
├── Dockerfile
├── requirements.txt
└── start.py
Games
Server
├── Dockerfile
├── requirements.txt
└── start.py
Test
├── .dockerignore
├── Dockerfile
└── requirements.txt
Tools
docker-compose.yml
start.cmd
start.sh
```

Entwicklungsdokumentation

3.3.4.3 Docker-Socket

Die .env-Datei im root Verzeichnis enthält die Umgebungsvariable DOCKER_SOCK_PATH, die den Pfad zum Docker-Socket, welcher für das Nutzen der Python-Docker API notwendig ist, speichert. Der Pfad ist standardmäßig <DOCKER_SOCK_PATH=/var/run/docker.sock> (ohne <>). Dieser Pfad erfordert jedoch Root-Rechte. Unter Rootles-Docker auf dem Deployment Server, kann man dem Pfad zum Socket des Benutzers in der bash.rc im /home Verzeichnis des Benutzers finden, falls er dort nicht zu finden ist, muss der Administrator des Servers angesprochen werden.

3.3.4.4 Tools-Verzeichnis

Das Verzeichnis Tools enthält Dateien und Funktionen die sowohl der swtp-server als auch der swtp-game-client benötigen. Aber Docker kann nicht standardmäßig beim Bauen eines Images Dateien kopieren, die nicht im Build-Kontext liegen, darum wird als Build-Context in der docker-compose.yml häufig das root Verzeichnis gewählt so liegt sowohl das Verzeichnis des jeweiligen Services als auch das Tools Verzeichnis im Kontext.

3.3.4.5 .dockerignore

Die .dockerignore Dateien im Frontend und FrontendDebug/ Verzeichnis schließen die node_modules und package-lock.json aus, weil diese beim Bau des Images automatisch generiert werden. So wird sichergestellt, dass das Image möglichst unabhängig von der Projektkonfiguration der einzelnen Entwickler ist. Außerdem wird jeweils der Dockerfile ausgeschlossen, weil innerhalb des Images nicht benötigt wird.

3.3.4.6 Anwendung mit Docker bauen und starten und Benutzen

Stoppen die laufenden Container mit:

```
docker-compose down
```

Bauen der Images:

```
docker-compose build
```

Starten des Tests-Containers:

```
docker-compose up swtp-test
```

Starten der Container für Entwickler:

```
docker-compose up -d swtp-server swtp-frontend swtp-frontend-debug
```

Entwicklungsdokumentation

Starten der Anwendung für reguläre Nutzer:

```
docker-compose up swtp-frontend
```

Nachdem alle Container erfolgreich gebaut wurden, kann man sich über die verschiedenen Schnittstellen mit den Diensten verbinden, die von unserer Plattform bereitgestellt werden.

3.3.4.7 Nutzung der Webseite

Unter der URL: <http://localhost:8086/> sollte nun lokal die Weboberfläche erreichbar sein. Im THM-Netz ist die Webseite unter <https://kimaster.mni.thm.de> erreichbar.

3.3.4.8 Nutzung der Debug-Webseite

Die Debug-Webseite ist in einem Webbrowser lokal unter <http://localhost:8087/> erreichbar. Die Debug-Webseite wird nicht im THM-Netz bereitgestellt.

3.3.4.9 Nutzung der API

Lokal ist die API ist lokal unter <ws://localhost:8010/ws> erreichbar. Im THM-Netz ist die API unter <wss://kimaster.mni.thm.de/ws> erreichbar.

Nach dem Verbinden mit der API können Messages über die Verbindung gesendet werden. Eine Liste aller spezifizierten Messages ist im Repository unter [Spezifikation/Dokumentation/Sonstiges/commands.md](#) auffindbar.

3.3.5 Integration von Code aus externen Repositories

Im Rahmen unseres Projekts haben wir verschiedene Teile des Codes aus dem externen Repository [alpha-zero-general](#) von [suragnair](#) von GitHub als Codebasis genommen und an unsere spezifischen Anforderungen angepasst. Hier ist eine detaillierte Übersicht über die vorgenommenen Änderungen:

- [Tools/utils.py](#) – Kommentare wurden hinzugefügt, um die Verständlichkeit des Codes zu verbessern.
- [GameClient/pit.py](#) – Eine eigene Version von pit.py wurde erstellt, um spezifische Anforderungen zu erfüllen.
- [Tools/neural_net.py](#) – Kommentare wurden hinzugefügt, um die Verständlichkeit des Codes zu verbessern.
- [Tools/mcts.py](#) – Neben Kommentaren wurden zusätzliche Einschränkungen (MAX) implementiert, um die Funktionalität zu erweitern.



Entwicklungsdocumentation

- `Tools/i_game.py` – Die ursprüngliche Datei hieß Game.py, wurde mit umfassenden Kommentaren erweitert und zudem um einige weiteren Funktionalitäten für die Umsetzung des Spiels Dame
- `Trainer/coach.py` – Kommentare wurden hinzugefügt, um die Verständlichkeit des Codes zu verbessern.
- `Trainer/Arena.py` – Die ursprüngliche Implementierung wurde nur für den Trainer übernommen. Für das eigentliche Spielen und die Backend Implementierung wurde eine eigene Lösung entwickelt.
- `Games/tictactoe`, `Games/othello` – Diese Spiele wurden übernommen und auf die neueste PyTorch-Version aktualisiert.

Entwicklungsdocumentation

3.4 Trainingsprozess

Während des Projekts wurden die Spiele Dame, Vier gewinnt, Nim, Othello und Tic-Tac-Toe trainiert. Das Training erfolgte mit der Datei `Trainer/main.py`. Die verwendete Hardware bestand aus einem Intel 13700K Prozessor mit 24 logischen Prozessoren, die auf 6 GHz übertaktet waren, sowie 96 GB Arbeitsspeicher. Davon waren 64 GB DDR5 6000 Arbeitsspeicher und 32 GB SSD-Auslagerungsspeicher. Eine CUDA (12.1) fähige RTX 4080 GPU mit dem Game Ready Treiber 560.70, welche ebenfalls auf 600 Watt übertaktet war, wurde für das Training verwendet. Als Hauptspeicher dienten vier 1 TB 980 Pro NVMe M.2 SSDs, die im RAID 0 Verbund betrieben wurden, um mögliche Engpässe beim Laden und Speichern von Zwischenständen zu vermeiden. Der gesamte Strom für die Trainingszeit wurde aus einem Solararray bezogen und über Nacht von einem 20 kWh Akku versorgt. Während des Projekts wurden aus Zeitgründen vier Modelle gleichzeitig trainiert. Dies führte zu Engpässen im Arbeitsspeicher, da die parallele Verarbeitung eine erhebliche Menge an RAM beanspruchte. Trotz dieser RAM-Engpässe waren sowohl die CPU als auch die GPU nicht vollständig ausgelastet und hätten Ressourcen für mindestens zwei weitere Modelle gehabt.

- **Prozessor:** Intel 13700K, 24 logische Prozessoren, 6GHz übertaktet
- **Arbeitsspeicher:** 96 GB, davon 64 GB DDR5 6000 und 32 GB SSD-Auslagerungsspeicher
- **GPU:** RTX 4080, CUDA (12.1) fähig, Game Ready Treiber 560.70, 600 Watt übertaktet
- **RAM:** 4x 1 TB 980 Pro NVMe M.2 SSDs, RAID 0 Verbund
- **Stromversorgung:** Solararray, 20 kWh Akku

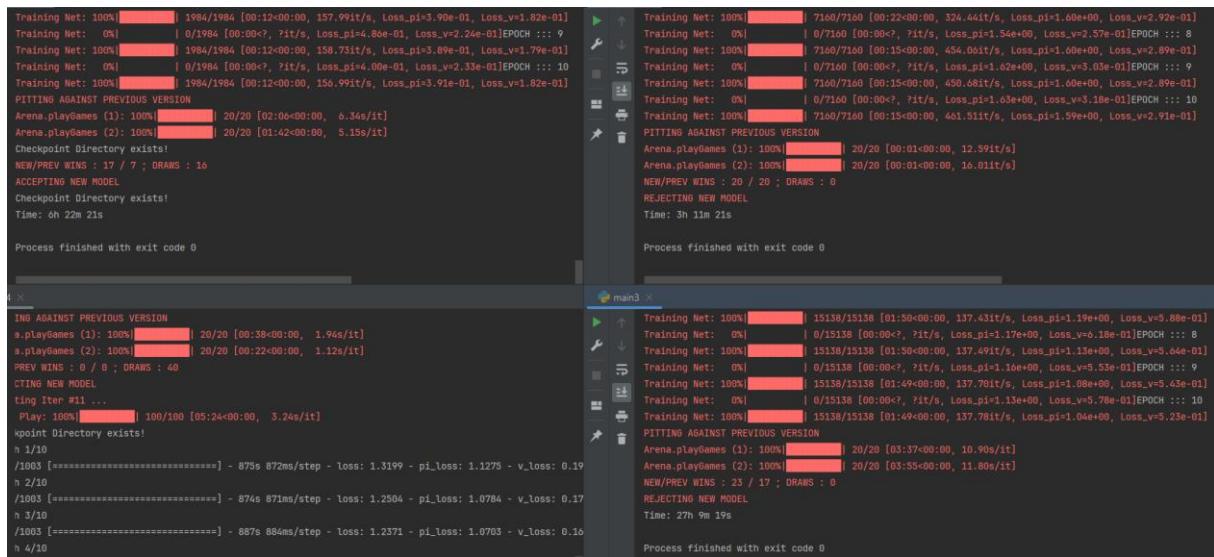
3.4.1 Trainingseinstellungen

Der Trainer ermöglichte die Anpassung der folgenden Parameter:

- **Iterationen:** Diese geben an, wie oft ein neues Modell erstellt werden soll. Mehr Iterationen erhöhen die Wahrscheinlichkeit, bessere Modelle zu finden.
- **Episoden:** Diese bestimmen die Intensität des Trainings und die Anzahl der Trainingsdaten, die zum Trainieren des neuen Modells verwendet werden.
- **ArenaCompare:** Diese geben an, wie viele Spiele das vorherige und das neue Modell gegeneinander spielen sollen, um festzustellen, ob das neue Modell besser oder schlechter ist.

Entwicklungsdocumentation

3.4.2 Trainingsdauer



```

Training Net: 100% | 1984/1984 [00:12<00:00, 157.99it/s, Loss_pi=3.90e-01, Loss_v=1.82e-01]
Training Net: 0% | 0/1984 [00:00<?, ?it/s, Loss_pi=4.00e-01, Loss_v=2.24e-01]EPOCH :::: 9
Training Net: 100% | 1984/1984 [00:12<00:00, 158.73it/s, Loss_pi=3.89e-01, Loss_v=1.79e-01]
Training Net: 0% | 0/1984 [00:00<?, ?it/s, Loss_pi=4.00e-01, Loss_v=2.33e-01]EPOCH :::: 10
Training Net: 100% | 1984/1984 [00:12<00:00, 150.99it/s, Loss_pi=3.91e-01, Loss_v=1.82e-01]
PITTING AGAINST PREVIOUS VERSION
Arena.playGames (1): 100% | 20/20 [02:06<00:00, 6.34it/s]
Arena.playGames (2): 100% | 20/20 [01:42<00:00, 5.15it/s]
Checkpoint Directory exists!
NEW/REV WINS : 17 / 7 ; DRAWS : 16
ACCEPTING NEW MODEL
Checkpoint Directory exists!
Time: 0h 22m 21s

Process finished with exit code 0

[...]
IND AGAINST PREVIOUS VERSION
s.playGames (1): 100% | 20/20 [00:38<00:00, 1.94it/s]
s.playGames (2): 100% | 20/20 [00:22<00:00, 1.12it/s]
PREV WINS : 0 / 0 ; DRAWS : 40
CTING NEW MODEL
ting Iter #1 ...
Play: 100% | 100/100 [05:24<00:00, 3.24it/s]
spoint Directory exists!
h 1/10
/1003 [=====] - 8756 872ms/step - loss: 1.3199 - pi_loss: 1.1275 - v_loss: 0.19
h 2/10
/1003 [=====] - 8746 871ms/step - loss: 1.2504 - pi_loss: 1.0784 - v_loss: 0.17
h 3/10
/1003 [=====] - 8878 884ms/step - loss: 1.2371 - pi_loss: 1.0783 - v_loss: 0.16
h 4/10

Training Net: 100% | 15138/15138 [01:50<00:00, 137.43it/s, Loss_pi=1.19e+00, Loss_v=5.08e-01]
Training Net: 0% | 0/15138 [00:00<?, ?it/s, Loss_pi=1.17e+00, Loss_v=6.18e-01]EPOCH :::: 8
Training Net: 100% | 15138/15138 [01:50<00:00, 137.49it/s, Loss_pi=1.13e+00, Loss_v=5.04e-01]
Training Net: 0% | 0/15138 [00:00<?, ?it/s, Loss_pi=1.16e+00, Loss_v=5.53e-01]EPOCH :::: 9
Training Net: 100% | 15138/15138 [01:49<00:00, 137.70it/s, Loss_pi=1.08e+00, Loss_v=5.43e-01]
Training Net: 0% | 0/15138 [00:00<?, ?it/s, Loss_pi=1.13e+00, Loss_v=5.78e-01]EPOCH :::: 10
Training Net: 100% | 15138/15138 [01:49<00:00, 137.78it/s, Loss_pi=1.04e+00, Loss_v=5.23e-01]
PITTING AGAINST PREVIOUS VERSION
Arena.playGames (1): 100% | 20/20 [03:37<00:00, 10.90it/s]
Arena.playGames (2): 100% | 20/20 [03:55<00:00, 11.80it/s]
NEW/REV WINS : 23 / 17 ; DRAWS : 6
REJECTING NEW MODEL
Time: 27h 9m 19s

Process finished with exit code 0

```

Abbildung 7: Trainingszyklus

Spiel	Bibliothek	Iterationen	Episoden	ArenaCompare	Benötigte Zeit
Checkers	PyTorch	50	100	40	27h:9m:19s
Connect4	PyTorch	100	100	40	3h:11m:21s
Nim	PyTorch	100	100	40	1h:16m:51s
Othello	PyTorch	100	100	40	6h:22m:21s
TicTacToe	Keras	11	100	40	2h:31m:33s

Tabelle 6: Trainingskonfigurationen und -zeiten



Entwicklungsdokumentation

3.4.3 Analyse der Trainingsergebnisse

Die Ergebnisse zeigen, dass die Keras-Implementierung bei unserem System deutlich langsamer war als die PyTorch-Implementierungen. Daher haben wir nach 11 Iterationen bei Tic-Tac-Toe aufgehört, da bei den 9 möglichen Spielfeldern ein weiteres Training keinen signifikanten Mehrwert mehr gebracht hätte.

Das Projekt hat gezeigt, dass PyTorch für das Training der Spiele auf unserem System effizienter war als Keras. Die umfassende Hardwarekonfiguration, einschließlich der übertakteten CPU und GPU sowie der schnellen NVMe-SSDs im RAID 0 Verbund, trug dazu bei, die Trainingszeiten zu minimieren und eine effiziente Nutzung der Ressourcen zu gewährleisten. Der Einsatz von Solarenergie und einem Akkusystem für die Stromversorgung unterstreicht das Bestreben, nachhaltige und umweltfreundliche Technologien zu nutzen.



Entwicklungsdokumentation

3.5 Entwicklungsprozess

Der Entwicklungsprozess beruhte auf der Nutzung bewährter Werkzeuge und Methoden wie [Jira](#) und dem darin verwalteten Kanban Board mit Scrum-Sprints, sowie der Programmcode Verwaltung in [Git](#), die eine kontinuierliche Verbesserung und erfolgreiche Projektdurchführung ermöglichen.

3.5.1 Zeit- und Aufgabenverfolgung mit Jira

Für die Planung, Verfolgung und Dokumentation von Aufgaben sowie zur Zeitverfolgung setzen wir auf Jira, eine leistungsfähige Projektmanagement-Software. Die Hauptmerkmale unserer Jira-Nutzung umfassen:

Aufgabenmanagement: Alle Entwicklungsaufgaben werden als Tickets in Jira erfasst. Jedes Ticket enthält Beschreibungen und Unterpunkte, die als Akzeptanzkriterien gelten. Diese Transparenz erleichtert die Nachverfolgung und Bearbeitung von Aufgaben und ermöglicht das Erkennen von Engpässen und möglichen Problemen.

The screenshot shows a Jira Kanban board with the following columns and tasks:

- ZU ERLEDIGEN 2:**
 - Benutzerhandbuch (ENTWURF, KAN-328)
 - Git publizieren (ORGANISATION, KAN-396)
 - + Vorgang erstellen
- ORGANISATION 3 MAX.: 6:**
 - Weiterentwicklungsdocumentation (ENTWURF, KAN-330)
 - Entwicklerdokumentation (ENTWURF, KAN-329)
 - Testdokumentation (ENTWURF, KAN-331)
- BACKEND 1 MAX.: 6:**
 - Python Externes Interface (IMPLEMENTIERUNG, KAN-394)
- FRONTEND 2 MAX.: 5:**
 - Frontend kommentieren (IMPLEMENTIERUNG, KAN-393)
 - Frontend fehlende Issues (IMPLEMENTIERUNG, KAN-377)
- FERTIG 16:**
 - Seite mit Vue erstellen (IMPLEMENTIERUNG, KAN-117)
 - Abschlusspräsentation (ORGANISATION, KAN-293)
 - Deployment (TESTS, KAN-269)

Abbildung 8: Kanban Board

Zeitverfolgung: Teammitglieder loggen ihre Arbeitszeit direkt in Jira über ein Jira internes Plugin namens [Worklogs](#). Diese Funktion ermöglicht den Arbeitsaufwand für einzelne Aufgaben genau zu dokumentieren und den Fortschritt im Projektverlauf zu überwachen. Dadurch konnte sichergestellt werden, dass man im Zeitplan blieb und Ressourcen effektiv genutzt wurden. Aus diesen Daten wurden wöchentliche Berichte erstellt (zu finden unter [Spezifikation/Stundentracking/](#)).

Entwicklungsdocumentation

Total time: 12h 31m

 JB	Justine Buß	Add time	Start timer		
 MLB	Maximilian Bachmann 10m	 PW	Pascal Waldschmidt 4h 14m	 OK	Omar Karkotli 8h 7m

Aktivität

Anzeigen: Time ▾ Älteste zuerst ↑

Period: Everything Member: Select... Group by: Select...

Total time: 12h 31m CSV

Date :	Time :	Member :	Comment
2024-07-09 08:15	10m	 Maximilian Bachmann	centering dialog box
2024-06-13 14:20	4h 7m	 OK Omar Karkotli	No details
2024-05-29 13:07	53m	 PW Pascal Waldschmidt	Rechere Vue best Practices ...
2024-05-29 10:36	3h 7m	 OK Omar Karkotli	einbindung von Vue und Css
2024-05-29 09:43	3h 21r	 PW Pascal Waldschmidt	Versuch Frontend mit besteh...
2024-05-29 09:40	53m	 OK Omar Karkotli	recherche wie man Vue und ...

Abbildung 9: Stundentracking

User :	Total :	April(11-30) :	May :	June :
 Alex	263h 58m	51h 58m	102h 12m	84h 36m
 Justine Buß	205h 58m	45h 53m	79h 47m	55h 44m
 Maximilian Bach...	223h 4m	54h 50m	95h 3m	37h 30m
 Omar Karkotli	183h 32m	25h 35m	46h 19m	56h 51m
 Pascal Waldsch...	194h 57m	23h 55m	59h 2m	58h 17m
 Sven Roman Rei...	154h 24m	25h 36m	73h 21m	31h 12m
 Thorben Jones	218h 58m	31h 33m	84h 12m	60h 49m
Summary ↗	1444h 51m	259h 20m	539h 56m	384h 59m

Abbildung 10: Worklogs

Kanban und Scrum: Es wurde eine hybride Methodik, die Elemente von Kanban und Scrum kombiniert, verwendet, um sowohl Flexibilität als auch Struktur im Entwicklungsprozess zu

Entwicklungsdokumentation

gewährleisten. Gearbeitet wurde in zweiwöchigen Sprints. Es gab zwei Wochenmeetings, wobei das Donnerstags-Meeting alle zwei Wochen zur Sprint-Planung und Retrospektive genutzt wurde. Zur Visualisierung der wöchentlichen Aufgaben diente das Kanban Board mit spezialisierten Spalten: Zu Erledigen, Organisation, Backend, Frontend, Docker, Meeting und Fertig. Dies ermöglichte auf einen Blick die Menge an Aufgaben pro Klein-Teams abzuschätzen und möglicherweise anzupassen.

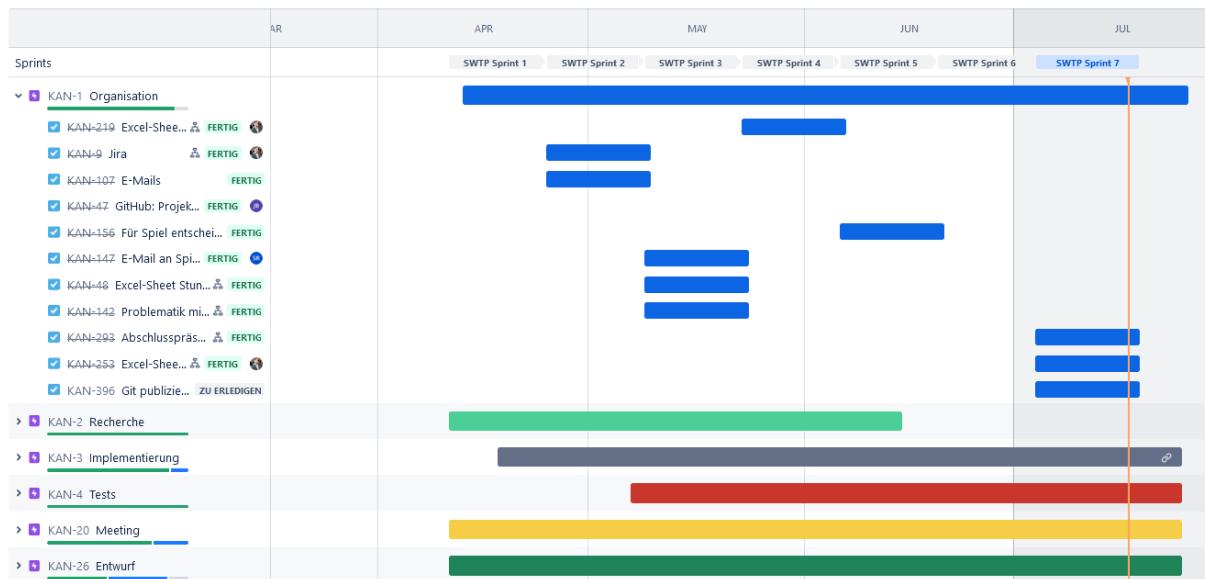


Abbildung 11: Sprintübersicht

3.5.2 Git-Workflows

Zur Versionskontrolle und für das Code-Management wird Git und [GitHub](#) eingesetzt. Der Git-Workflow ist strukturiert und darauf ausgelegt, eine klare und nachvollziehbare Versionshistorie zu gewährleisten sowie die Zusammenarbeit im Team zu erleichtern.

- **Branching-Strategie:** Eine branchenbasierte Strategie wird verwendet, bei der der `main`-Branch den stabilen Code enthält, der produktionsbereit ist. Neue Funktionen und Verbesserungen werden in separaten Feature-Branches entwickelt. Sobald die Entwicklung abgeschlossen und getestet ist, werden diese Branches nach erfolgreichem Code Review in den `develop`-Branch gemerget. Zum Abschluss des Projektes gehörte die Devise das GitHub-Repository vollständig aufzuräumen und öffentlich zugänglich zu machen. Dabei wurden alle gemergten abgeschlossenen Branches entfernt und unnötige Artefakte entfernt. Das gesamte Projekt ist somit auf dem `main`-Branch zu finden.

Entwicklungsdocumentation

 Plattform-fuer-Vergleich-von-Spiele-Kls Private

 main  1 Branch  0 Tags  Go to file  Add file  Code

PascalUni	Added Comments on Frontend	45b388f · 52 minutes ago	428 Commits
External/Python	Added comments	last week	
Frontend	Added Comments on Frontend	52 minutes ago	
FrontendDebug	adding comments to dockerfiles	4 hours ago	
GameClient	adding comments to dockerfiles	4 hours ago	
Games	Go and Nogo removed from Project	2 weeks ago	
Server	Added comments	last week	
Spezifikation	moved location	4 hours ago	
Test	removing redundant test	5 days ago	
Tools	comments	last week	
Trainer	Added Code Documentation	7 hours ago	
.env	removing .from env from gitignore	2 weeks ago	
.gitignore	removing .from env from gitignore	2 weeks ago	
docker-compose.yml	Server starts automatic	6 hours ago	
start.cmd	add test to start.cmd	3 weeks ago	
start.sh	undo now for all games, frontend magic ;-)	2 weeks ago	

About
 Eine Webanwendung zum Test und Vergleich von Spiel-Kls. Benutzer können gegen andere Nutzer, vorab trainierte Kls (alpha-zero framework) antreten oder ihre eigenen Kls über eine bereitgestellte WebSocket Verbindung testen. Die Plattform bietet einen Pool an Spielen und unterstützt die Entwicklung und Verbesserung von KI-Strategien.

 Activity
 0 stars
 2 watching
 0 forks

Releases
 No releases published [Create a new release](#)

Packages
 No packages published [Publish your first package](#)

Contributors 7

Abbildung 12: GitHub-Repository

Entwicklungsdokumentation

3.5.3 Projektverlauf

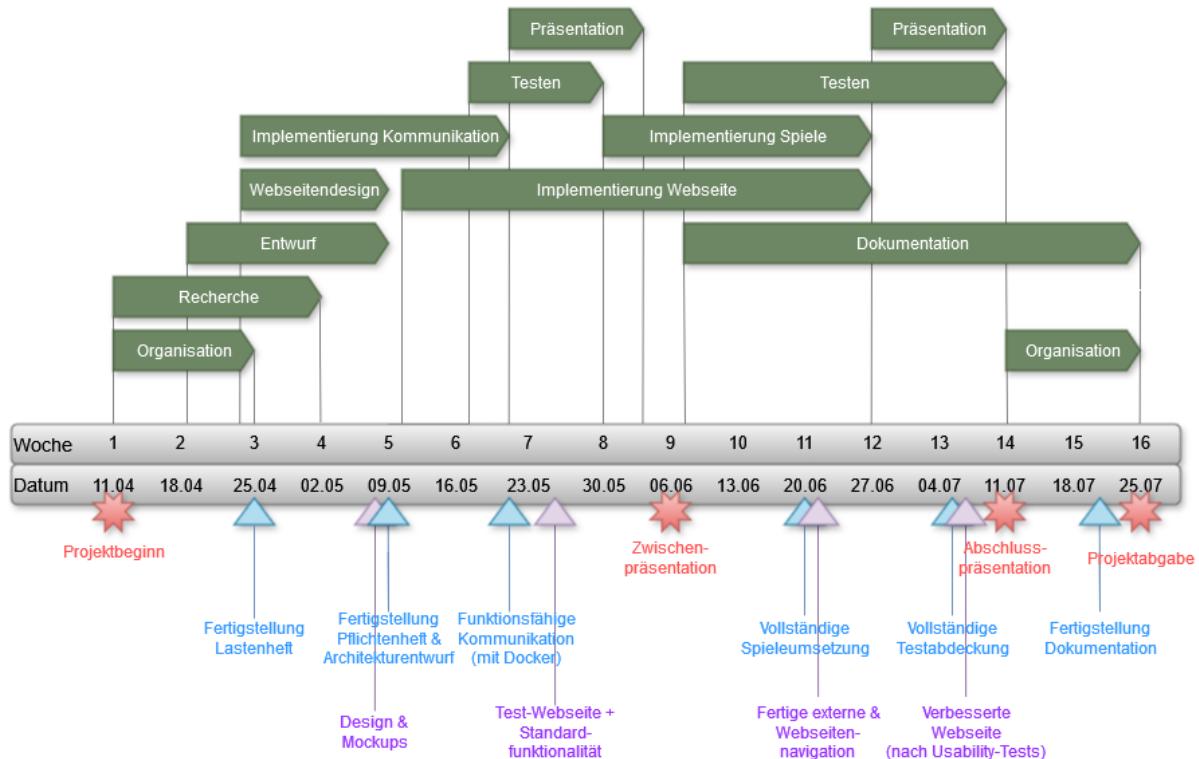


Abbildung 13: Projektverlaufsdiagramm

Das Projekt erstreckte sich über einen Zeitraum von etwa drei Monaten, von Mitte April bis Mitte bzw. Ende Juli. Der offizielle Projektstart fand am 11. April mit einem Kickoff-Meeting statt und endete mit einer abschließenden Projektpräsentation am 11. Juli und Projektabgabe am 25. Juli.

Einarbeitungs- und Entwurfsphase: In den ersten Wochen lag der Fokus auf der Einarbeitung in das Projekt und dem Entwurf der grundlegenden Projektstruktur. Bedeutende Meilensteine dieser Phase waren das Lastenheft und das Pflichtenheft. Zur Erstellung dieser Dokumente wurden erste technische Entwürfe und Implementierungen in Verbindung mit Featurebesprechungen und Recherchearbeiten durchgeführt.

Implementierungsphase: Nach Abschluss des Entwurfs wurden die Kommunikationsschnittstellen des Projekts finalisiert. Dies umfasste alle Schnittstellen, Kommunikationsprotokolle und Docker-Netzwerke. Parallel dazu wurden die Webseitendesign-Entwürfe implementiert sowie die ersten Spiele trainiert und lauffähig gemacht. Ziel war es, einen ersten Prototyp für die Zwischenpräsentation Anfang Juni zu testen und vorzuführen. Ein erster

Entwicklungsdokumentation

Webseitenentwurf stand bereit, der die Standardfunktionalitäten der Must-Haves umsetzte und ein erstes Spielen ermöglichte.

Rückmeldung und weitere Implementierung: Auf Basis des Feedbacks aus der Zwischenpräsentation lag der Fokus weiterhin auf der Implementierung. Mit einem vollständigen Server Rework wurde der Zeitplan ein wenig nach hinten verschoben, sodass die restlichen Spiele nur nacheinander umgesetzt werden konnten. Die Webseite wurde dabei ebenfalls etwas später fertiggestellt, sodass jedoch zur Abschlusspräsentation die Funktionalitäten weitestgehend standen mit einer fertigen Implementierung und spielbaren Webseite. So blieben noch kleinere Bugfixes und organisatorische Aspekte, wie das Verfassen der Dokumentation, fertig testen und Projekt zum Abschluss bringen.

Features: Einige zusätzliche Nice-To-Haves konnten dabei umgesetzt werden, wobei jedoch die Implementierung des Spiels Go nicht möglich war. Angedacht war die Einbindung eines existierenden GitHub-Repos für die Go-Logik. Aufgrund der umfangreichen Logik des Spiels und hohen benötigten Rechenleistung für das Trainieren, wäre dies die einzige Möglichkeit für die Einbindung des Spiels in der zur Verfügung stehenden Zeit. Da die gefundenen, existierenden Lösungen jedoch alle nicht lauffähig waren bzw. sich nicht an das vorgegebene Interface und die vorgegebene Struktur hielten, war es auch nach vielen investierten Stunden nicht möglich diese einzubinden (Go-Implementierungen: [liranmatcu/alpha-zero-general-with-Go-game](#), [trieu-learner/alpha-zero-general](#), [ambbber/alpha-zero-go](#))

Abschlussphase: In den letzten zwei Wochen nach der Abschlusspräsentation am 11. Juli lag der Fokus auf intensiven Prüfungen der gesamten Software, die Durchführung letzter Bugfixes und die Sicherstellung, dass alle Projektziele erreicht wurden. Die Abgabe am 25. Juli markierte das Ende des Projekts mit einer finalen Fertigstellung der Ergebnisse.

Entwicklungsdocumentation

3.6 Build-Prozess

3.6.1 Systemanforderungen

- **Für Docker Desktop (Windows)**
 - Prozessor (CPU): 64-Bit mit Second Level Address Translation (SLAT)
 - Arbeitsspeicher (RAM): 4 GB
 - Betriebssystem: Windows 10 Home, Professional oder Enterprise
 - Virtualisierung: Hardware-Virtualisierung muss im BIOS des Computers aktiviert sein
 - Hyper-V: ist für Windows Professional oder Enterprise optional
 - Windows Subsystem für Linux 2 (WSL 2): muss für Windows Home aktiv sein
- Festplattenspeicher: min. 20 GB
- Internetverbindung: zum Herunterladen von Paketen
- **Optional:**
 - Eine CUDA (12) fähige GPU
 - Für das Trainieren wird Python 3.11.1-3.11.12 benötigt.
 - Git

3.6.2 Installationsanweisungen

Server und Docker

1. **Docker installieren**
 - Besuche die [Docker Desktop Webseite](#) und lade Docker Desktop herunter.
 - Folge den Installationsanweisungen für dein Betriebssystem (Windows, macOS, Linux).
 - Nach der Installation starte Docker Desktop und stelle sicher, dass es läuft.
2. **Ordner erstellen**
 - Erstelle einen Ordner, in den du das Git-Repository klonen möchtest oder in den du die ZIP-Datei entpacken möchtest.
3. **Repository klonen oder ZIP-Datei herunterladen**
 - **Option 1: Repository klonen**
 1. Öffne ein Terminal (Command Prompt, PowerShell oder ein Unix-Terminal) und klonne das [GitHub-Repository](#) mit:

```
git clone https://github.com/12ghostrider21/KIMaster.git
```
 - **Option 2: ZIP-Datei herunterladen**
 1. Gehe zur [GitHub-Seite des Repositories](#)



Entwicklungsdocumentation

2. Klicke auf den grünen "Code" Button und wähle "Download ZIP".
3. Entpacke die heruntergeladene ZIP-Datei in den zuvor erstellten Ordner.
4. Oder: [Direct Download from REPO](#)

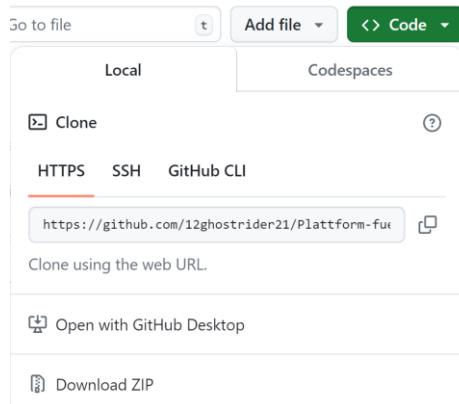


Abbildung 14: GitHub klonen

4. In das Verzeichnis wechseln

- Navigiere in das Verzeichnis des geklonten oder entpackten Repositories:

```
cd PFAD/ZU/VERZEICHNIS
```

- Ersetze PFAD/ZU/VERZEICHNIS durch den tatsächlichen Pfad zum geklonten oder entpackten Verzeichnis.

5. Docker-Container starten

▪ Windows

1. Stelle sicher, dass Docker Desktop ausgeführt wird.
2. Öffne ein Command Prompt oder PowerShell im Verzeichnis des geklonten oder entpackten Repositories.
3. Führe die `start.cmd` Datei aus:

```
./start.cmd
```

▪ Unix (Linux, macOS)

1. Öffne ein Terminal im Verzeichnis des geklonten oder entpackten Repositories.
2. Mache die `start.sh` Datei ausführbar (falls noch nicht geschehen):

```
chmod +x start.sh
```



Entwicklungsdokumentation

3. Führe die `start.sh` Datei aus:

```
./start.sh
```

3.6.3 Hinweise zum Build Prozess

Beim ersten Mal kann der Build-Prozess je nach Internetverbindung sehr lange dauern. Bei einer Download-Geschwindigkeit von 50 Mbit/s dauert der Prozess etwa 30 Minuten. Nach Abschluss des Prozesses werden automatisch alle notwendigen Container in Docker gestartet. Während der Entwicklung kam es auch schon vor, dass der Build-Prozess bis zu 2 Stunden gedauert hat. Wenn eine Änderung am Programmcode vorgenommen wird, muss neu gebaut werden. Da jedoch diesmal nur die Änderungen gebaut werden und vieles aus dem Build-Cache genutzt werden kann, dauert dieser Prozess meist nur wenige Sekunden.

3.7 Deployment-Prozess

Die Deployment-Prozedur wurde auf einem Server durchgeführt, der von der Technischen Hochschule Mittelhessen bereitgestellt wurde. Zur Verbindung mit dem Server wurde die VPN der THM und PuTTY verwendet, um eine SSH-Verbindung aufzubauen. Diese Verbindung ermöglichte den Zugriff auf den Server und die Durchführung der erforderlichen Schritte.

3.7.1 Einrichten des Servers

Auf einem Server der THM, der im internen THM-Netz und per VPN erreichbar ist, wurde von einem Administrator, [Jonas Kuche](#), ein Benutzerkonto (hier `kim-user` genannt) eingerichtet. Dieser Schritt war notwendig, um spezifische Berechtigungen und einen dedizierten Arbeitsbereich auf dem Server zu gewährleisten. Außerdem hat Jonas Kuche einen Reverse-Proxy eingerichtet, der innerhalb des THM-Netzes die zwei URLs bereitstellt, über die eine Verbindung mit der KIMaster Anwendung auf dem Server möglich ist.

Entwicklungsdokumentation

URL im THM-Netz	Weiterleitung lokal	Service
https://kimaster.mni.thm.de	http://localhost:8086/	swtp-frontend (Webseite)
wss://kimaster.mni.thm.de/ws	ws://localhost:8010/ws	swtp-server (API)

Tabelle 7: URLs

Alles, was in den Unterkapiteln 3.7.2 - 4 beschrieben wird wurde als `kim-user` durchgeführt.

3.7.2 Code von GitHub laden

Im Heimatverzeichnis von `kim-user`, wurde ein Unterverzeichnis namens `kimaster-repo/` erstellt. Es dient als Speicherort für alle relevanten Dateien und Ressourcen, die für das Projekt erforderlich sind. Innerhalb des Verzeichnisses `kimaster-repo/` wurde dann mittels `git clone` das [GitHub-Repository von KIMaster](#) geklont.

3.7.3 Pfad zum Docker-Socket anpassen

Während die Entwickler standardmäßig mit Root-Rechten entwickelten, musste das Projekt aus Sicherheitsgründen auf dem Server unter einem Benutzer ohne Root-Rechte (`kim-user`) deployt werden. Dazu musste auch die Konfiguration angepasst werden die dem `swtp-server` mitteilt wo der Docker-Socket von `kim-user`. Dazu wurde zuerst im Heimatverzeichnis von `kim-user` mittels `cat .bashrc` der Pfad zum Docker-Socket ermittelt. In der letzten Zeile der `.bashrc` (Musterbeispiel: `export DOCKER_HOST=unix:///run/user/123456/docker.sock`) stand der Pfad. Nun wurde der relevante Teil des Pfades (`/run/user/123456/docker.sock`) kopiert und mittels `nano ./kimaster-repo/KIMaster/.env` in der Datei `.env` Datei des Projektes eingefügt.

	Inhalt von <code>.env</code>
vorher	<code>DOCKER_SOCK_PATH=/var/run/docker.sock</code>
nachher	<code>DOCKER_SOCK_PATH=/run/user/123456/docker.sock</code>

Tabelle 8: Pfadanpassung Docker-Socket

3.7.4 Programm in Rootles-Docker starten

Zuerst wurde mittels `systemctl --user status docker` ermittelt das Docker gestartet werden musste. Anschließend wurde Docker mit dem Befehl `systemctl --user start`

Entwicklungsdocumentation

docker gestartet. Anschließend wurde mit systemctl --user status docker festgestellt, ob Docker tatsächlich im Rootless-Mode lief.

Nach dem Starten von Docker, wurde im Projekt-Verzeichnis die Container mit dem Befehl docker-compose build gebaut. Anschließend wurde mittelst docker-compose up -d swtp-server der swtp-server im Detached-Mode gestartet. Der Detached-Mode sorgt dafür, dass der Container im Hintergrund läuft, ohne direkt an die Konsoleneingabe gebunden zu sein.

Nach dem Starten des swtp-servers konnten nun auch der swtp-test Service mit docker-compose up swtp-test gestartet werden. Dieser wurde nicht im Detached-Mode gestartet, damit man die Testergebnisse mitlesen konnte. Nach den erfolgreichen Tests terminierte der Test-Container ohne weiteres Zutun mit Code 0.

Da alle Tests erfolgreich waren, konnte anschließend das Frontend mittels docker-compose up -d swtp-frontend gestartet werden.

Zusammengefasst:

1. Login per SSH (Putty)
2. In das Projekt Verzeichnis wechseln:

```
cd kimaster-repo/KIMaster
```

3. Verwerfen aller lokalen Änderungen

```
git stash
```

4. Update des Verzeichnisses

```
git pull
```

5. Wechsle den Pfad oder passe ihn an.

```
nano .env
```

6. Neustarten des Build-Prozesses und starten aller Server-Komponenten

```
./start.sh
```

Entwicklungsdocumentation

3.7.5 Verfügbarkeit im THM-Netz prüfen

(Dieser Schritt wurde über die VPN-Verbindung zum THM-Netz ausgeführt.)

Nach dem alle `swtp-server` und `swtp-frontend` gestartet waren, wurde im WebBrowser ([Google Chrome](#)) die URL <https://kimaster.mni.thm.de> eingegeben und das Verhalten der Funktionen auf der Webseite stichprobenartig auf unerwartetes Verhalten getestet.

Nach dem an der Webseite keine Mängel aufgefallen waren wurde nun die API mithilfe von Postman getestet. Dazu wurde eine Verbindung zu <wss://kimaster.mni.thm.de/ws> aufgebaut und die Message `{"command": "lobby", "command_key": "create"}` gesendet. Welche wie erwartet erfolgreich mit mit `response_code 100` beantwortet wurde. Da das bereits getestete Frontend die Befehle ebenfalls über die API sendet, wurden keine weiteren Tests durchgeführt und die Verbindung zur API mithilfe von Postman geschlossen.

3.7.7 Probleme während des Deployments

Es kam mehrfach das Problem auf, dass nicht ausreichend Speicher (etwa 12GB) auf dem Server zur Verfügung stand. Dies wurde im Log von docker-compose build an dem Fehler `OSSError: [Errno 28] No space left on device` bemerkbar, der zum Abbruch des Build-Vorgangs führte. Der Workaround für dieses Problem war vor jedem Build den Befehl `docker system prune -all` einzugeben, welcher alles Images und Container von Docker löscht und damit auch das über 10 GB große `swtp-server` Image. Außerdem beantragte, Jonas Kuche, einen Server mit mehr Speicher (ca. 200 Gib) bei der THM. Der Antrag wurde bewilligt, seitdem trat dieses Problem nicht mehr auf.



Weiterentwicklungsdocumentation

Weiterentwicklungsdocumentation



Weiterentwicklungsdocumentation

4 Weiterentwicklungsdocumentation

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.



Weiterentwicklungsdocumentation

4.1.2 Anlegen der Projektordner und -dateien

1. **Ordnernavigation:** In der Projektübersicht in den main-Branch navigieren und von dort in den Games / Unterordner steuern.

The screenshot shows a GitHub repository interface. At the top, there are navigation links: 'main' (selected), '2 Branches', '0 Tags', a search bar ('Go to file'), an 'Add file' button, and a 'Code' button. Below this is a list of files and folders. The 'Games' folder is highlighted with a pink background. Other visible items include 'External/Python', 'Frontend', 'FrontendDebug', 'GameClient', 'Server', 'Test', 'Tools', 'Trainer', 'docs', '.env', '.gitignore', 'docker-compose.yml', 'start.cmd', and 'start.sh'. Each item has a small icon, a name, a description, and a timestamp indicating when it was last modified.

File/Folder	Description	Last Modified
External/Python	Added comments	last week
Frontend	Logo hinzugefügt, Spielbrett optimiert, code aufgeräumt	13 hours ago
FrontendDebug	Implemented a Message for Draw, made Debug page ready ...	2 weeks ago
GameClient	fix timeline for player2 and rotation at blunder	2 weeks ago
Games	Go and Nogo removed from Project	2 weeks ago
Server	Added comments	last week
Test	removing redundant test	5 days ago
Tools	comments	last week
Trainer	Added Code Documentation	2 hours ago
docs	server.md now included in documentation	11 minutes ago
.env	removing .from env from gitignore	2 weeks ago
.gitignore	removing .from env from gitignore	2 weeks ago
docker-compose.yml	Server starts automatic	1 hour ago
start.cmd	add test to start.cmd	3 weeks ago
start.sh	undo now for all games, frontend magic ;-)	2 weeks ago

Abbildung 15: Projektübersicht -> Games

2. **Ordnerstruktur:** Einrichten der erforderlichen Projektordner und -dateien, die für die Entwicklung und Integration des neuen Spiels erforderlich sind.

Weiterentwicklungsdocumentation

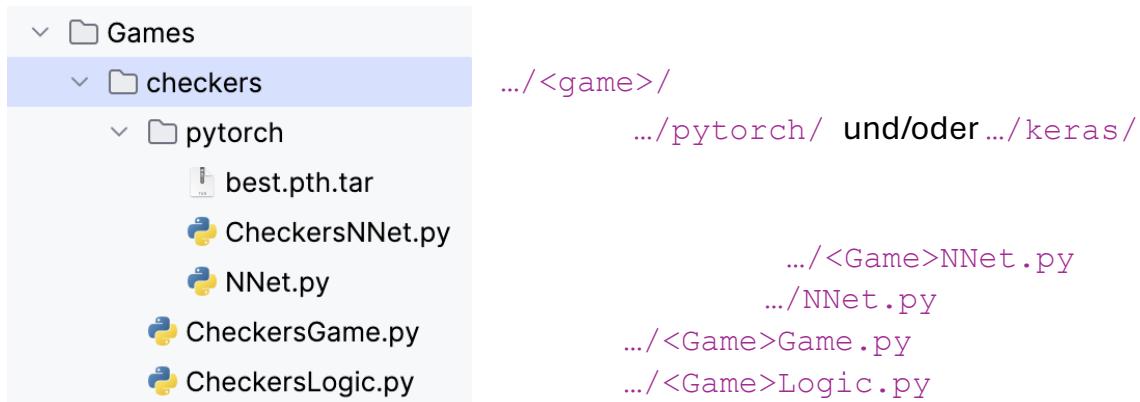


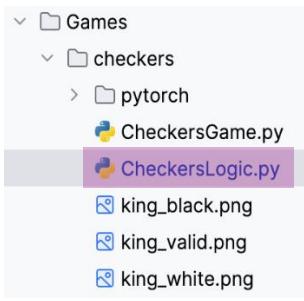
Abbildung 16: Ordnerstruktur Spieleordner

- `.../<game>`: Ordner für den Quellcode. Der Ordnername ist, der welcher vom dynamischen Importer geladen wird und dann auch als Auswahlparameter dient.
 - `.../pytorch/`: Für die Implementierung mit [PyTorch](#). Empfohlen aufgrund von deutlich schnelleren Trainingsphasen und stärkerer Modellen.
 - `.../keras/`: Für die Implementierung mit [Keras](#). Ist jedoch ebenfalls eine PyTorch Implementierung vorhanden mit dazugehörigem Modell, dann wird die PyTorch Implementation bevorzugt vom dynamischen Importer.
 - `.../<Game>NNet.py`: Wrapper-Klasse für ein neuronales Netzwerk.
 - `.../NNet.py`: Definition des neuronalen Netzwerks
- `.../<Game>Game.py`: Python Code, der das Spiel implementiert. Hier müssen die Methoden aus dem Interface `Tools/i_game.py` implementiert werden.
- `.../<Game>Logic.py`: Python Code, der die Spielbrettlogik und Aktionen auf dieses Spielbrett umfasst. Sie stellt Methoden bereit, um das Spiel zu implementieren.

Weiterentwicklungsdocumentation

4.1.3 Implementierung der Spiellogik

1. **Spielbrettimplementierung:** Implementierung des Spielbrettes und Aktionen auf dem Spielbrett befinden sich in der `.../<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 `.../<Game>Game.py` haben Zugriff darauf.



```

import numpy as np
import math

WHITE_NON_KING = 1
WHITE_KING = 3
BLACK_NON_KING = -1
BLACK_KING = -3
EMPTY = 0

DEFAULT_SIZE = 8

class Board:
    """
    Checkers Board.
    """

    __directions_kings = [(-1, -1), (1, -1), (-1, 1), (1, 1)]
    __directions_white = [(-1, -1), (-1, 1)]
    __directions_black = [(1, -1), (1, 1)]


def __init__(self, n: int = None, pieces: np.array = None, last_long_capture: bool = None):

```

Abbildung 17: Code Spielbrettimplementierung

2. **Interface-Methoden:** 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.

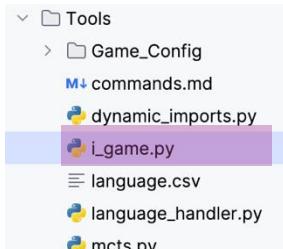
Sollten für manche Methoden externe Ressourcen benötigt werden, so muss der Pfad relativ angelegt werden. Ein Beispiel hierfür aus Checkers sind die Bilder für die Spielfiguren.

```
king_valid = pygame.image.load('..../Games/checkers/king_valid.png')
```

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

Weiterentwicklungsdocumentation

ausgelegt. 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.



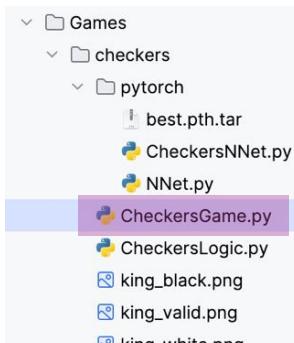
```

1  from abc import ABC, abstractmethod
2  import numpy as np
3
4
5  class IGame(ABC):
6
7      @abstractmethod
8  def getInitBoard(self) -> np.array:
9      """
10         Get the initial representation of the game board.
11
12     Returns:

```

Abbildung 18: Code `i_game` Interface

3. Spiellogik: In `.../<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. Neben den zu implementierenden Interfacemethoden können jegliche Hilfsmethoden verwendet werden.



```

1  from Tools.i_game import IGame, np
2  from Games.checkers.CheckersLogic import Board
3  import math
4
5
6  class CheckersGame(IGame):
7      """
8          Checkers Game class implementing the alpha-zero-general Game interface.
9      """
10
11     def __init__(self, n: int = None):
12         self.board = Board(n)
13         self.n = n or self.board.n
14         self.redundancy = 0
15         self.ll_capture_hist = {} # necessity of having a hist because undo function
16
17     def getInitBoard(self):
18         """
19             return initial board (numpy array) and resets data"""
20         self.resetData()
21         return np.copy(self.board.pieces)
22
23     def getBoardSize(self):
24         return self.n, self.n

```

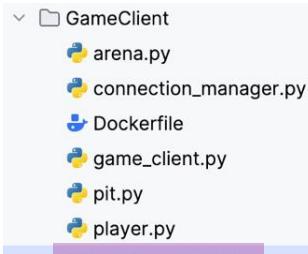
Abbildung 19: Code Spiellogikimplementierung

Die Methoden der jeweiligen `.../<Game>Game.py` werden an verschiedenen Stellen verwendet. Sie werden in der `Trainer/coach.py` in Kombination mit den neuronalen Netzen und der Monte Carlo Tree Search zum Trainieren genutzt. Zudem für die

Weiterentwicklungsdocumentation

Spielzugberechnung der KI KIM in `Server/SocketServer.py` ebenfalls unter zu Hilfenahme der neuronalen Netze mithilfe der `Tools/mcts.py`. Letztlich in `GameClient/arena.py`, um das Spielfeld zu initialisieren, einen Zug auszuführen oder zu überprüfen, ob das Spiel zu Ende ist, sowie die Nutzung für eine mögliche Rotation der Grafik im Frontend im `GameClient/game_client.py`. 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. Bei allen anderen Spielen wird der Zug einfach so wieder zurückgegeben.

4. **Imports:** 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 notwendig wird. Deshalb wird die ausschließliche Handhabung mit `numpy` und `math` bevorzugt.



```

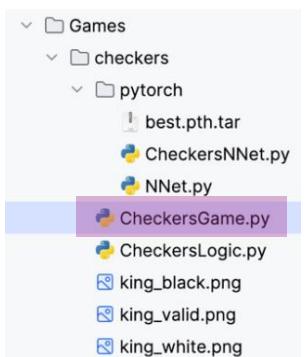
    ▾ GameClient
      ▾ arena.py
      ▾ connection_manager.py
      ▾ Dockerfile
      ▾ game_client.py
      ▾ pit.py
      ▾ player.py
      requirements.txt
  
```

```

1   websockets==12.0
2   numpy~1.26.4
  
```

Abbildung 20: Code requirements.txt

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.



```

    ▾ Games
      ▾ checkers
        ▾ pytorch
          best.pth.tar
          CheckersNNet.py
          NNet.py
          CheckersGame.py
          CheckersLogic.py
          king_black.png
          king_valid.png
          king_white.png
  
```

```

6   class CheckersGame(IGame):
191
192   def draw(self, board: np.array, valid_moves: bool, cur_player: int, '
193     import pygame
194     king_white_img = pygame.image.load('king_white.png')
195     king_black_img = pygame.image.load('king_black.png')
  
```

Abbildung 21: Code PyGame Import in draw()

Weiterentwicklungsdocumentation

Sonstige Hinweise:

- I. **Eindimensionale Handhabung der Züge:** Es wird empfohlen, Spiele so zu implementieren, dass Züge eindimensional gehandhabt werden. Das bedeutet, dass nur ein Index für ein Feld benötigt wird, statt x- und y-Koordinate. Das Spielfeld wird durchnummeriert, und jedes Feld erhält einen eindeutigen Index. Diese Vorgehensweise erleichtert die Kommunikation mit dem Frontend und die allgemeine Handhabung der Züge. Falls eine bestehende Implementierung dies nicht unterstützt, sind zusätzliche Hilfsmethoden erforderlich, die Züge intern umwandeln. Dies erfordert zusätzliche Logik und Testung. Je nach Spiel gibt es unterschiedliche Handhabungen von Zügen. Einfachere Spiele wie Tic-Tac-Toe arbeiten mit einem einfachen int als Zugindex, während komplexere Spiele wie Dame Tupel mit Start- und Zielindex verwenden. Die Implementierung muss in den Spielen konsistent sein und das zugrunde liegende Interface umsetzen.
- II. **Umwandlung von KI-Zügen:** Züge, die von der KI ausgegeben werden, sind Zugindizes im `getValidMoves`-Array. Alle validen Züge werden als binäres Array (1 für möglich, 0 für nicht möglich) abgefragt, und nur der Index des besten Zugs wird zurückgegeben. Diese Indizes müssen vor der Ausführung in tatsächliche Züge umgewandelt werden. Hierfür wird die `translate`-Methode des `Tools/i_game.py` Interfaces verwendet, die für jedes Spiel implementiert werden, muss. Bei einfachen Spielen wie Tic-Tac-Toe oder Vier gewinnt, bei denen der Zug nur aus einem Wert besteht, kann der Index unverändert zurückgegeben werden. Bei komplexeren Spielen muss die `translate`-Methode entsprechend angepasst werden, um den Index in einen tatsächlichen Zug zurückzurechnen.
- III. **getSymmetries-Methode:** Eine Herausforderung bei der Implementierung des Interfaces kann die Methode `getSymmetries` darstellen. Diese Methode wird in `Trainer/coach.py` verwendet, um alle Symmetrien eines Spielzustands in die Trainingsdaten aufzunehmen. Bei einfachen Spielen wie Tic-Tac-Toe und Vier gewinnt ist dieser Abschnitt nicht relevant, aber bei Spielen mit Tupel-Zügen (z.B. Start- und Zielposition eines Spielsteins) kann er wichtig sein. Häufige Symmetrien sind Rotation und Spiegelung. Das Brett und der zugehörige Policy-Vektor werden gedreht, gespiegelt oder vertauscht. Um Verzerrungen zu vermeiden, sollte der Policy-Vektor mittels numpy zu einem quadratischen zweidimensionalen Array reshaped werden. Dies erfordert eventuell Padding, um sicherzustellen, dass die Wurzel der ActionSize eine Ganzzahl ergibt. So können später mittels `numpy.reshape` und `numpy.rot90` Verzerrungen vermieden werden.
- IV. **Validierung der Züge:** Falls ein Zug nicht valide ist, muss ein ValueError geworfen werden. Die Methode `getNextState` in der jeweiligen `.../<Game>Logic.py` führt den Zug aus

Weiterentwicklungsdocumentation

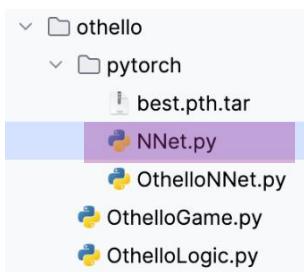
und prüft seine Validität. Die `GameClient/arena.py` fängt den `ValueError` ab und benachrichtigt den Spieler, dass der Zug ungültig war.

- V. **Endbedingungen bei endlosen Zügen:** Bei Spielen, bei denen unendlich oft redundante Züge möglich sind, sollte in der `getGameEnded`-Methode eine Endbedingung für ein Unentschieden eingeführt werden. Beispielsweise könnte das Spiel nach 30 oder 50 Zügen ohne Fortschritt (z.B. ohne Schlag eines Steins) als unentschieden gewertet werden. Dies verhindert einen `RecursionError` im Training, der durch eine Überschreitung der Rekursionstiefe verursacht werden könnte.
- VI. **Handhabung des Aussetzens:** Bei Spielen, bei denen Aussetzen möglich ist (z.B. Othello), ist dieser Zug als letzter Zug kodiert. Bei einem 8x8-Feld wäre die `ActionSize` 64, doch mit der Möglichkeit des Aussetzens 65. Der Index 64 entspricht dem Aussetzen. Diese Konvention wird sowohl im Frontend als auch im Backend befolgt.
- VII. Weitere Hinweise, derer es hier keine weitere Erläuterung bedarf, finden sich im `Tools/i_game.py` Interface. Zudem bietet das Spiel Dame für die meisten der hier vorgestellten Probleme eine Lösung oder einen Ansatz für eine Lösung in `.../checkers/CheckersGame.py`

Weiterentwicklungsdocumentation

4.1.4 Erstellen des neuronalen Netzes

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



```

import torch
import torch.optim as optim

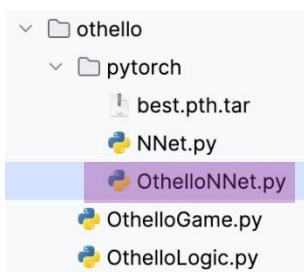
from Games.othello.pytorch.OthelloNNet import OthelloNNet as onnet

args = dotdict({
    'lr': 0.001,
    'dropout': 0.3,
    'epochs': 10,
    'batch_size': 64,
    'cuda': torch.cuda.is_available(),
    'num_channels': 512,
})

class NNetWrapper(NeuralNet):
    def __init__(self, game):
        self.net = onnet(game, args)
        self.board_x, self.board_y = game.getBoardSize()
        self.action_size = game.getActionSize()

```

Abbildung 22: Code Othello NNet.py



```

import sys
sys.path.append('..')

import torch
import torch.nn as nn
import torch.nn.functional as F

2 usages
class OthelloNNet(nn.Module):
    def __init__(self, game, args):
        # game params
        self.board_x, self.board_y = game.getBoardSize()
        self.action_size = game.getActionSize()
        self.args = args

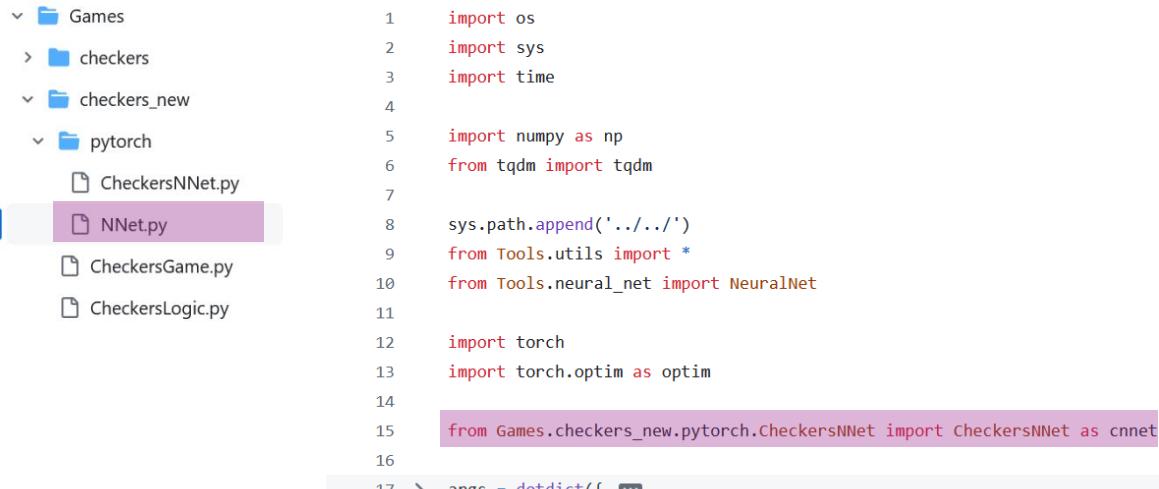
    super(OthelloNNet, self).__init__()

```

Abbildung 23: Code Othello OthelloNNet.py

Weiterentwicklungsdocumentation

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

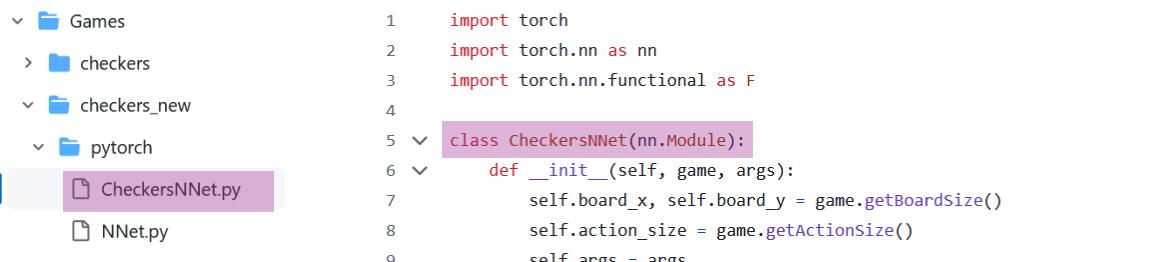


```

1 import os
2 import sys
3 import time
4
5 import numpy as np
6 from tqdm import tqdm
7
8 sys.path.append('../..')
9 from Tools.utils import *
10 from Tools.neural_net import NeuralNet
11
12 import torch
13 import torch.optim as optim
14
15 from Games.checkers_new.pytorch.CheckersNNNet import CheckersNNNet as cnet
16
17 > args = dotdict({
18     })
19
20
21 < class NNetWrapper(NeuralNet):
22     def __init__(self, game):

```

Abbildung 24: Code Checkers NNet.py



```

1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4
5 < class CheckersNNNet(nn.Module):
6     def __init__(self, game, args):
7         self.board_x, self.board_y = game.getBoardSize()
8         self.action_size = game.getActionSize()
9         self.args = args
10
11     super(CheckersNNNet, self).__init__()

```

Abbildung 25: Code Checkers CheckersNNNet.py

3. **Spiele ohne 2D Bretter:** 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.



Weiterentwicklungsdocumentation

4.1.5 Trainieren des Spiels

1. **Ordnernavigation:** Für das Training steht eine eigene `main.py` zur Verfügung, die einen durch das Trainieren hindurchführt. Dazu muss von der Projektübersicht in den `Trainer/` Ordner navigiert werden.

main		
2 Branches 0 Tags		
Go to file Add file Code		
FlyingSuricate server.md now included in documentation eac39ca · 11 minutes ago 423 Commits		
External/Python	Added comments	last week
Frontend	Logo hinzugefügt, Spielbrett optimiert, code aufgeräumt	13 hours ago
FrontendDebug	Implemented a Message for Draw, made Debug page ready ...	2 weeks ago
GameClient	fix timeline for player2 and rotation at blunder	2 weeks ago
Games	Go and Nogo removed from Project	2 weeks ago
Server	Added comments	last week
Test	removing redundant test	5 days ago
Tools	comments	last week
Trainer	Added Code Documentation	2 hours ago
docs	server.md now included in documentation	11 minutes ago
.env	removing .from env from gitignore	2 weeks ago
.gitignore	removing .from env from gitignore	2 weeks ago
docker-compose.yml	Server starts automatic	1 hour ago
start.cmd	add test to start.cmd	3 weeks ago
start.sh	undo now for all games, frontend magic ;-)	2 weeks ago

Abbildung 26: Projektübersicht -> Trainer

2. **Trainer Ausführung:** Dort muss die `Trainer/main.py` ausgeführt werden.

```
if __name__ == "__main__":
    # if multiple games training at once -> threading
    t = Trainer("../Games")
    t.run()
```

Abbildung 27: Code Trainer/main.py

3. **Spielauswahl:** 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



Weiterentwicklungsdocumentation

Dateien für das neuronale Netz und die Spiellogik den Namenskonventionen folgen und in den richtigen Ordnern liegen. 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.

```
Trainer started.  
[0] CHECKERS_PYTORCH  
[1] CONNECT4_PYTORCH  
[2] NIM_PYTORCH  
[3] TICTACTOE_KERAS  
[4] OTHELLO_PYTORCH
```

Abbildung 28: Trainer Spieleanwahl

4. **Trainingskonfiguration:** Nun lässt sich das zu trainierende Spiel, sowie die Konfiguration des Trainings auswählen. Alle Konfigurationseinstellungen beeinflussen dabei die Trainingszeit, die die Spiele benötigen. Richtwerte für einfachere Spiele liegen bei 100 Iterationen, 100 Epochen und 40 Arena Compares. Bei komplexeren Spielen sollte man mit geringeren Iterationen beginnen und gegebenenfalls nachtrainieren.
 - **NumIters:** Definiert die vollständigen Iterationen, die das Training durchläuft. Es umfasst dabei sowohl den Aufbau der neuronalen Netze durch die „Self Plays“ der KI gegen sich selbst als auch die Testspiele gegen bereits existierende Modelle.
 - **NumEps:** Definiert die Anzahl der vollständig durchgespielten „Self Plays“. Dabei werden sowohl alle Spielbrettzustände mit Policy-Vektor (Wahrscheinlichkeitsverteilungen der Züge) und zusätzlich alle Symmetrien abgespeichert, was dann in Summe die Trainingsdaten für das neuronale Netz bildet in den `checkpoint_<iteration>.pth.tar.examples`-Dateien. Dies benötigt dementsprechend während des Trainings einiges an Speicher und RAM-Nutzung.
 - **ArenaCompare:** Definiert die Anzahl der Testspiele des neu trainierten Modells gegen das bestehende Modell, nachdem das Training abgeschlossen wurde.



Weiterentwicklungsdocumentation

```
Trainer started.  
[0] CHECKERS_PYTORCH  
[1] CONNECT4_PYTORCH  
[2] NIM_PYTORCH  
[3] TICTACTOE_KERAS  
[4] OTHELLO_PYTORCH  
Select game to train [-1 to exit]:2  
Selected: [nim_pytorch] importing Game.py and NNet.py ...  
No checkpoint file or .h5 /.pth.tar file found!  
NumIters? (default=100) [-1 to exit]:5  
NumEps? (default=100) [-1 to exit]:5  
ArenaCompare? (default=40) [-1 to exit]:5
```

Abbildung 29: Trainer Konfiguration

5. **Trainingsphase:** Dann beginnt die Trainingsphase mit Validierung der neu trainierten Modelle gegen die Alten.

- **EPOCH :::: <num>:** Stellt die eigentliche Trainingsphase nach den „Self Plays“ dar. Es ist standardmäßig auf dem Wert 10 eingestellt. Dieser ließe sich im args Dictionary der .../<game>/<framework>/NNet.py des zu trainierenden Spiels anpassen.
- **PITTING AGAINST PREVIOUS VERSION:** Hier findet der Arena Vergleich statt. 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 und 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 können zudem noch einige andere Parameter verstellt werden. Die Auswahl ist jedoch für die meisten Anwendungsfälle bereits gut getroffen.

Weiterentwicklungsdocumentation

```

Loading %s... NimGame
Not loading a checkpoint!
Loading the Coach...
Starting the learning process 🎉
Starting Iter #0 ...
Loading %s... NNetWrapper
Self Play: 100%|██████████| 5/5 [00:00<00:00, 48.23it/s]
Checkpoint Directory exists!
EPOCH :::: 1
Training Net: 100%|██████████| 12/12 [00:00<00:00, 241.47it/s, Loss_pi=2.53e+00, Loss_v=1.21e+00]
Training Net:  0%|          | 0/12 [00:00<?, ?it/s, Loss_pi=2.39e+00, Loss_v=1.16e+00]EPOCH :::: 2
Training Net: 100%|██████████| 12/12 [00:00<00:00, 517.17it/s, Loss_pi=2.39e+00, Loss_v=1.17e+00]
Training Net:  0%|          | 0/12 [00:00<?, ?it/s, Loss_pi=2.33e+00, Loss_v=1.05e+00]EPOCH :::: 3
EPOCH :::: 4
Training Net: 100%|██████████| 12/12 [00:00<00:00, 518.02it/s, Loss_pi=2.32e+00, Loss_v=1.05e+00]
Training Net: 100%|██████████| 12/12 [00:00<00:00, 533.90it/s, Loss_pi=2.24e+00, Loss_v=9.53e-01]
Training Net: 100%|██████████| 12/12 [00:00<00:00, 566.89it/s, Loss_pi=2.20e+00, Loss_v=9.19e-01]
EPOCH :::: 5

PITTING AGAINST PREVIOUS VERSION
Arena.playGames (1): 100%|██████████| 2/2 [00:00<00:00, 39.33it/s]
Arena.playGames (2): 100%|██████████| 2/2 [00:00<00:00, 76.19it/s]
NEW/PREV WINS : 4 / 0 ; DRAWS : 0
ACCEPTING NEW MODEL

```

Abbildung 30: Trainingsphase

6. **Zwischenspeicherung:** Nach der vollendeten Trainingsphase und Akzeptierung des besten 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. Wodurch man alte Zwischenstadien der besten Modelle beibehalten kann, trotz Überschreibung der `best.pth.tar` bzw. `best.h5` durch bessere Modelle. Dies kann für die Nachjustierung bei Schwierigkeitsgraden sinnvoll sein.

Weiterentwicklungsdocumentation

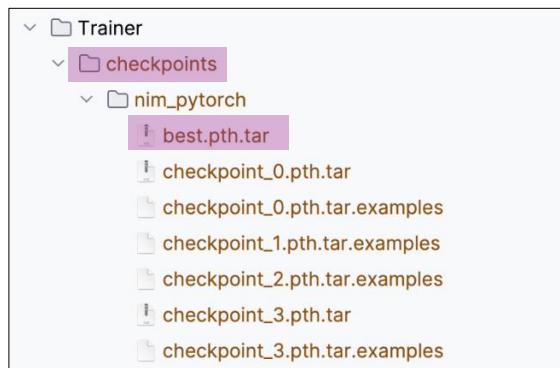


Abbildung 31: Trainingsmodell & -Zwischenstände

7. **Trainingsabbruch:** Die Trainingsphase lässt sich dabei jederzeit unterbrechen. Das bis dahin am besten trainierte Modell wird mit den zugrundeliegenden Trainingsdaten in einer neuen `checkpoint_<iteration>.pth.tar.examples` gespeichert. Wird das Training danach neu angestoßen, so lädt der Trainer den Speicherpunkt (Checkpoint) mit der höchsten Iterationsnummer aus und kann das Training fortsetzen. Wichtig ist, dass diese Dateien weiterhin im Verzeichnis `Trainer/<game>_<framework>/` liegen. Das Weitertrainieren einer Konfiguration ist ohne die ursprünglich zugrundeliegenden `checkpoint_<iteration>.pth.tar.examples`-Dateien nicht mehr möglich. Man beachte, dass diese Trainingsdaten bis zu mehrere GB groß werden können.
8. **Modellspeicherung:** Schlussendlich muss das trainierte Modell in den entsprechenden Ordner verschoben werden: `.../<game>/<framework>/`. Gut auch zum Zwischentesten, ob das Modell schon ausreichend stark trainiert wurde. Andernfalls lässt man den Trainer weiter trainieren.

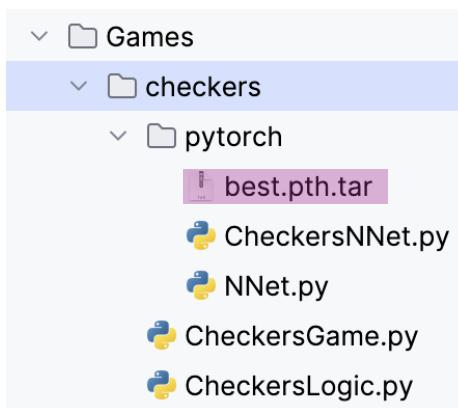


Abbildung 32: Checkers Trainingsmodell



Weiterentwicklungsdocumentation

4.1.6 Frontendanbindung

1. **Ordnernavigation:** In der Projektübersicht in den main-Branch navigieren und von dort in den `Frontend/src/components/` Unterordner steuern.

main			2 Branches	0 Tags	Go to file	Add file	Code
	FlyingSuricate	server.md now included in documentation			eac39ca · 11 minutes ago		423 Commits
	External/Python	Added comments					last week
	Frontend	Logo hinzugefügt, Spielbrett optimiert, code aufgeräumt					13 hours ago
	FrontendDebug	Implemented a Message for Draw, made Debug page ready ...					2 weeks ago
	GameClient	fix timeline for player2 and rotation at blunder					2 weeks ago
	Games	Go and Nogo removed from Project					2 weeks ago
	Server	Added comments					last week
	Test	removing redundant test					5 days ago
	Tools	comments					last week
	Trainer	Added Code Documentation					2 hours ago
	docs	server.md now included in documentation					11 minutes ago
	.env	removing .from env from gitignore					2 weeks ago
	.gitignore	removing .from env from gitignore					2 weeks ago
	docker-compose.yml	Server starts automatic					1 hour ago
	start.cmd	add test to start.cmd					3 weeks ago
	start.sh	undo now for all games, frontend magic ;-)					2 weeks ago

Abbildung 33: Projektübersicht -> Frontend

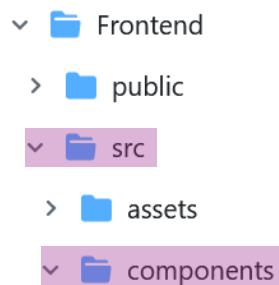
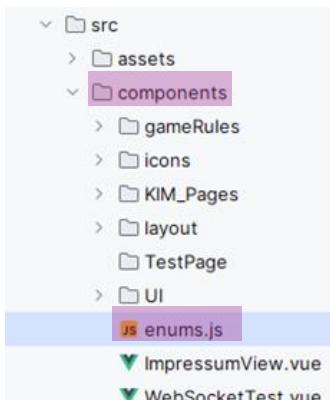


Abbildung 34: Frontend/src/components/

Weiterentwicklungsdocumentation

2. **Key-Value-Anpassung:** Um das Spiel auch ins Frontend einbinden zu können, muss dieses in das verwendete Enum eingespeist werden. Dazu muss in `.../components/enums.js` in der games Konstante ein neues Key-Value-Pair hinzugefügt werden, wobei der Wert den Namen des neu hinzugefügten Modells repräsentiert.



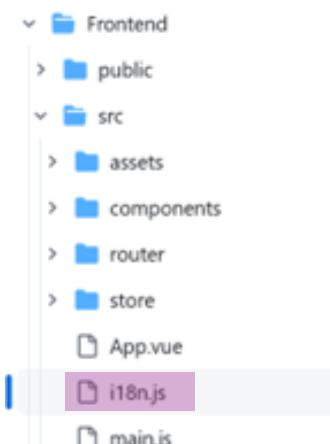
```

17     export const games = Object.freeze({
18         CONNECT4: 'connect4',
19         TICTACTOE: 'tictactoe',
20         OTHELLO: 'othello',
21         NIM: 'nim',
22         DAME: 'checkers',
23     })
24

```

Abbildung 35: Code Spieleanwahl

3. **Sprachanbindung:** Um auch die Sprachanbindung korrekt umzustellen, muss der Name sowie die Regeln des Spiels ebenfalls im Sprachmodul hinterlegt werden. Dazu muss in dieses navigiert werden, dass sich unter `.../i18n.js` finden lässt. Für jede einzelne Sprache (aktuell Deutsch, Englisch, Französisch und Spanisch) muss der Name des Spiels in dem Format: <Name des Models>:<Name des Spiels in der Sprache> implementiert werden.



```

450     chess: "Schach",
451     connect4: "Vier Gewinnt",
452     tictactoe: "Tic Tac Toe",
453     othello: "Othello",
454     nim: "Nim",
455     checkers: "Dame",

```

Abbildung 36: Code Sprachanpassung Spiel

4. **Implementierung der Spielsteuerung:** Navigieren zu `.../components/UI/PlayPage.js`

Weiterentwicklungsdocumentation

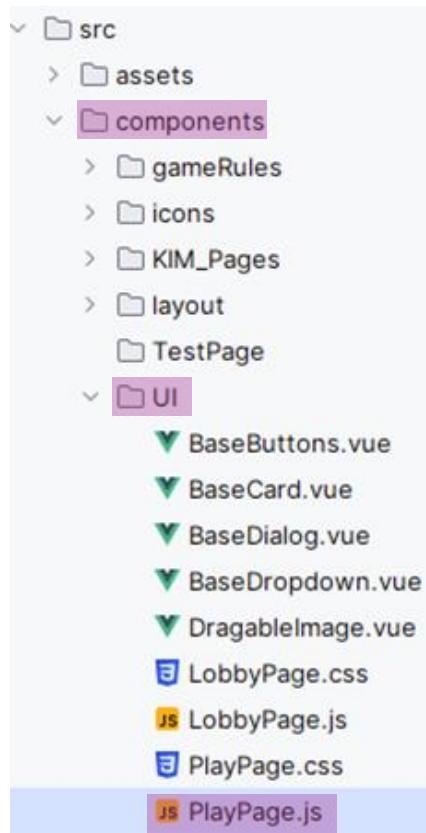
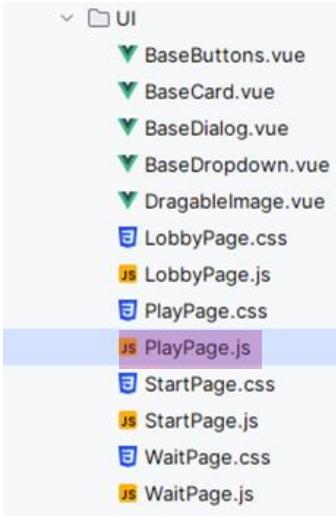


Abbildung 37: Frontend/src/components/UI/PlayPage

- i. **Implementierung einfacher Spiele:** Handelt es sich um ein einfaches Spiel welches folgenden Eigenschaften erfüllt, 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.
 - 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 Dame)

Weiterentwicklungsdocumentation



```

80  mounted() {
81    switch (this.game) { /*Define
82      case "chess":
83        this.boardWidth = 8;
84        this.boardHeight = 8;
85        this.twoTurnGame = true;
86        break;
87      case "checkers":
88        this.boardWidth = 8;
89        this.boardHeight = 8;
90        this.twoTurnGame = true;
91        break;
92      case "connect4":
93        this.boardWidth = 7;
94        this.boardHeight = 7;
95        this.twoTurnGame = false;
96        break;

```

Abbildung 38: Code Spieleimplementierung einfach

- ii. **Implementierung komplexer/atypischer Spiele:** Für Spiele, welche atypische Spielfelder und/oder mehr Zuginformationen 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 ggfs. eine neue Methode anlegen, welche das besondere Verhalten realisieren kann. Dabei muss darauf geachtet werden, dass die gesendeten Züge sich mit den Zügen decken, auf denen das KI-Modell arbeitet.

```

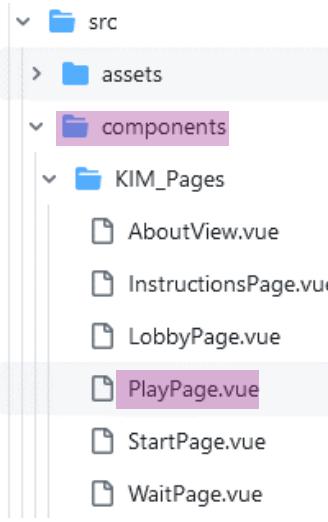
234  track.mousePosition(event) { /*Executes the actual Move on the GameBoard */
235    this.mousePosition = event.clientX;
236    this.clientY = event.clientY;
237    const imageRect = this.$refs.imageRef.getBoundingClientRect();
238    this.mousePosition = this.mousePosition - imageRect.left;
239    this.clientY = this.clientY - imageRect.top;
240    this.mousePosition = Math.ceil(
241      this.mousePosition / (this.$refs.imageRef.offsetWidth / this.boardWidth)
242    );
243    ...
244    switch (this.game) { /*Defines whether Special Rules apply for what moves the player can make */
245      case ENUMS.games.NIM: /*Executes special behaviour for the Nim Game */
246        this.nimMove(this.clientY-1)
247        break;
248      case ENUMS.games.CONNECT4: /*Only makes move in the Top rows, no matter where the player moves */
249        this.toPos= this.mousePosition-1;
250        this.playMakeMove();

```

Abbildung 39: Code Spieleimplementierung atypisch - Mausposition

Weiterentwicklungsdocumentation

Sollten zusätzliche Schaltflächen für das Spielen notwendig sein, so kann man diese als neues Element in die `../components/KIM_Pages/PlayPage.vue` einfügen, die Komponente muss dann mit einem `v-if="ENUMS.games.<Name des Spiels>"` versehen werden.

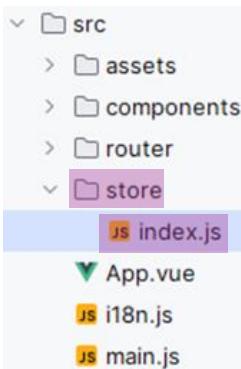


```

1   <template>
2
3   <!-- Game Controls -->
4   <base-card>
5     <div class="ButtonUp">
6       <base-button @click="showRules">{{ $t('message.show_rules') }}</base-button>
7       <base-button v-if="!this.gameOver && position !== 'sp'" @click="surrenderGame()">
8         {{ $t('message.surrender') }}
9       </base-button>
10      <base-button v-if="this.gameOver && position !== 'sp'" @click="returnLobby()">
11        {{ $t('message.lobby') }}
12      </base-button>
13      <base-button v-if="this.gameOver && position === 'sp'" @click="quitGame()">
```

Abbildung 40: Code Spieleimplementierung atypisch - Schaltflächen

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 alternative Highlightings, welche besser auf das atypische Feld angepasst sind, umzusetzen. Hierzu müssen sowohl Elemente in der `../components/KIM_Pages/PlayPage.vue` sowie in der `../components/UI/PlayPage.css` verändert werden. 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. Diesen findet man in der `../store/index.js` Datei.



```

329 case 208:
330   //commit('setIsValidMoveImage', true);
331   if (state.game==Enums.games.OTHELLO){
332     if (receivedJSONObject.moves.includes("64")|| receivedJSONObject.moves.includes(64)){
333       commit('setSkipMove',true);
334       commit('setNotif',Enums.notifStatus.SKIPMOVE);
335     } else commit('setSkipMove',false); //For Special case no available Move on Othello Board
336   };
337 break;
```

Abbildung 41: Code Spieleimplementierung atypisch - VUEX



Weiterentwicklungsdocumentation

Ein Beispiel dafür wäre, dass man in Othello einen Zug überspringen kann, wenn kein anderer Zug für einen möglich ist. Da das Frontend selbst die Spiellogik nicht kennt, wird auf die Antwort 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 `.../components/UI/PlayPage.js` reagiert wird, sodass das Kommando für das Überspringen des Zuges gesendet werden kann.

5. **Debug Seite:** Sämtliche Änderungen an der Debug Seite werden zentral in der `FrontendDebug/src/components/WebSocketTest.vue`-Datei unternommen. Hierfür müssen die neuen Spiele als neue Auswahl Option innerhalb der Game Select Komponente hinzugefügt werden. Ebenso müssen etwaige Änderungen in der Spielsteuerung angepasst werden. Dafür bietet es sich an einen weiteren Switch hinzuzufügen, welcher die Steuerung vom vorherigen Standard zur neuen Methode wechselt.

main			2 Branches	0 Tags	Go to file	Add file	Code
	FlyingSuricate	server.md now included in documentation	eac39ca · 11 minutes ago		423 Commits		
	External/Python	Added comments			last week		
	Frontend	Logo hinzugefügt, Spielbrett optimiert, code aufgeräumt			13 hours ago		
	FrontendDebug	Implemented a Message for Draw, made Debug page ready ...			2 weeks ago		
	GameClient	fix timeline for player2 and rotation at blunder			2 weeks ago		
	Games	Go and Nogo removed from Project			2 weeks ago		
	Server	Added comments			last week		
	Test	removing redundant test			5 days ago		
	Tools	comments			last week		
	Trainer	Added Code Documentation			2 hours ago		
	docs	server.md now included in documentation			11 minutes ago		
	.env	removing .from env from gitignore			2 weeks ago		
	.gitignore	removing .from env from gitignore			2 weeks ago		
	docker-compose.yml	Server starts automatic			1 hour ago		
	start.cmd	add test to start.cmd			3 weeks ago		
	start.sh	undo now for all games, frontend magic ;-)			2 weeks ago		

Abbildung 42: Projektübersicht -> FrontendDebug



Weiterentwicklungsdocumentation

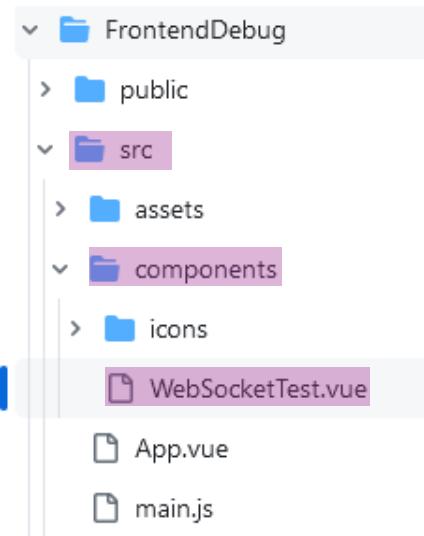


Abbildung 43: Frontend/Debug/src/components/WebSocketTest.vue

Weiterentwicklungsdocumentation

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. **Anforderungen:** Zuerst einmal muss auch die ausgewählte GitHub Lösung die [Anforderungen des AlphaZero-Frameworks](#) erfüllen.
2. **Interface-Kompatibilität:** 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](#). Implementierungen, die ebenfalls darauf aufbauen, sind zumeist fast nahtlos übernehmbar und lassen sich gut in den [Forks des Urspurungs-Repositories](#) finden. Einige Beispiele dafür sind:
 - Schach: [vinoo999/alpha-zero-general](#)
 - Hex: [visualisation/alpha-zero-general-hex](#)
 - Mühle: [LittleH0rst/alpha-zero-general](#)
 - Splendor, Minivilles, ...: [t-rekttt/alpha-zero-general](#)
3. **Klonen:** Klonen des GitHub-Repositories

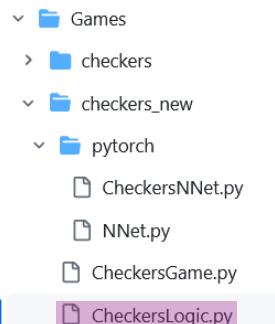
4.2.2 Mögliche Anpassung der Struktur

1. **Überprüfung der Ordnerstruktur:** Die Ordnerstruktur muss exakt so aufgebaut sein, wie in [4.1 Implementierung von Alpha-Zero-Spielen](#) beschrieben.
 - Das Spiel wird in `Games/<game>` verschoben
 - Darin befindet sich ein Keras und/oder PyTorch Ordner mit mindestens der `NNNet.py` und der `<Game>NNNet.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 Spielfile müssen beide unter `.../<game>` direkt liegen: `<Game>Logic.py` und `<Game>Game.py`

4.2.3 Anpassung der Imports und ggfs. Logik

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

Weiterentwicklungsdocumentation



```

1  from Tools.i_game import IGame, np
2  from Games.checkers_new.CheckersLogic import Board
3  import math
4
5
6  class CheckersGame(IGame):
7      """
8          Checkers Game class implementing the alpha-zero-general Game interface.
9      """
10
11 def __init__(self, n: int = None):
12     self.board = Board(n)
13     self.n = n or self.board.n
14     self.turn = 1 # reason: CheckersLogic get_legal_moves
15     self.board_history = []
16     self.board_history.append(self.board.pieces)
17     self.turn_history = []
18     self.turn_history.append(self.turn)
19     self.redundant = 0
20
21 def getInitBoard(self):
22     """return initial board (numpy array)"""
23     b = Board(self.n)
24     return b.pieces

```

Abbildung 44: Code Importanpassung

2. **Interface-Erfüllung:** Die `.../<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. **Training:** 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](#) neu trainiert werden.
2. **Frontend:** Die Frontendanbindung erfolgt nach den gleichen Schritten wie in [4.1.6 Frontendanbindung](#).

Weiterentwicklungsdocumentation

4.3 Funktionserweiterung unserer Plattform

4.3.1 Spielfeldkonfigurationen und Spielvariationen

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. Eine dynamische Anpassung der Spielfeldgröße könnte sowohl einfache Einstiegsvarianten für Anfänger als auch komplexere Szenarien für fortgeschrittene Spieler bieten. 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.

Zudem wäre es möglich verschiedene Spielvariationen in den Spielregeln umzusetzen. Dadurch erhalten die Nutzer die Gelegenheit, sich mit einer breiteren Palette an strategischen Herausforderungen auseinanderzusetzen. Diese Flexibilität unterstützt ebenfalls das tiefere Verständnis der Nutzer für ihre eigenen KI-Implementierungen. Indem die Spieler gezwungen werden, sich an unterschiedliche Spielregeln und Spielvariationen, sowie die damit verbundene Spielstrategien anzupassen. Dabei werden sie dazu angeregt, ihre Algorithmen und Lösungsansätze kritisch zu hinterfragen und zu optimieren.

- Anpassung der Spiele:** Hinzufügen eines weiteren Spieleordners mit kopiertem Inhalt, und Anpassung des Spielfeldgrößen-Übergabeparameters oder der Spiellogik in `.../<game>/<Game>Game.py` und `.../<game>/<Game>Logik.py`. Es ist dabei zwingend erforderlich, dass der Ordner für die verschiedenen Spielvariationen einzeln pro Spielversion vorliegt und unterschiedlich benannt ist, für eine Differenzierung der Spielvarianten im dynamischen Lader.
- Training:** Die neue Spielvariation muss ebenfalls trainiert werden und benötigt aufgrund der Änderung der Spielweise oder Spielfeldgröße eine eigene `best.pth.tar` bzw. `best.h5` im `.../<game>/<framework>/` Ordner liegend. Es kann den Schritten aus [4.1.5 Trainieren des Spiels](#) gefolgt werden.
- Frontend:** Es muss eine weitere Konfigurationsmöglichkeit in den Lobbys für die Spielfeldgröße oder die Variante zur Verfügung stehen. Um zusätzliche Optionen anbietbar zu machen, also neue Spielfeldkonfigurationen oder neue Spielvariationen, muss man lediglich die jeweiligen Selects innerhalb der `Frontend/src/components/KIM_Pages/Lobbypage.vue` um die neue Option anpassen, dafür sollte man außerdem neue Einträge innerhalb der

Weiterentwicklungsdocumentation

Frontend/src/i18n.js anlegen, sodass diese neuen Optionen korrekt in allen Sprachen angezeigt werden. Für das Hinzufügen von komplett neuen Optionen, wie beispielsweise Boardgröße muss man ein neues Auswahl Element in die LobbyPage einfügen, idealerweise durch ein neues Select. Zudem muss in der Frontend/src/components/UI/LobbyPage.js Datei eine neue Variable i data () angelegt werden, deren Inhalt wie V-model an die Auswahl im Select gebunden ist. Danach muss in der playCreate () -Methode die zweite data Konstante angepasst werden, sodass diese das korrekte Key-Value-Paar enthält. Man sollte dabei allgemein aufpassen, dass das hinzugefügte Auswahl-Element nur dann angezeigt wird, wenn das Spiel auch andere Konfigurationen anbietet, und versteckt wird, sollte es nicht aufrufbar sein. Zudem sollte man darauf achten, dass die neue Variable nur als Key-Value-Paar hinzugefügt wird, wenn dies für das Spiel sinnvoll ist.

4.3.2 Schwierigkeitsgrade und Spielmodi

Es besteht die Möglichkeit die von uns bereitgestellten Schwierigkeitsgrade und Spielmodi anzupassen und zu erweitern. Dies kann weitere Spielmöglichkeiten hinzufügen oder den Spielspaß weiter erweitern.

Dazu benötigt es ein wenig Vorbereitung und Anpassung, sowohl im Backend als auch im Frontend.

1. **Enum Erweiterung:** Je nach vorzunehmender Anpassung, müssen die Enum-Klassen in Tools/Game_Config/ in .../difficulty.py oder .../mode.py erweitert werden. Das weitere Handling der Parameter findet größtenteils dynamisch anhand des Enums statt.
2. **Pit Anpassung:** Die genaue Umsetzung eines neuen Spielmodi muss genauer analysiert werden. Als Ansatzpunkt kann jedoch die GameClient/pit.py dienen dort werden für die Modi die Spielereinstellungen vorgenommen.

Weiterentwicklungsdocumentation

```

      < GameClient      59      try:
>   Games          60      mcts = self.init_nn(game, network_class, h5_folder, h5_file, difficulty)
>   pretrained_models 61      match mode.value:
>     Dockerfile    62      case "player_vs_player":
>     StartClient.py 63      play1 = self.player1.play
>     arena.py       64      play2 = self.player2.play
>     game_client.py 65      case "player_vs_ai":
>     pit.py        66      play1 = self.player1.play
>     player.py      67      play2 = lambda x: mcts.getActionProb(x, temp=0)
>     requirements.txt 68      case "playerai_vs_ai":
>     requirements.txt 69      play1 = self.player1.play
>     requirements.txt 70      play2 = lambda x: mcts.getActionProb(x, temp=0)
>     requirements.txt 71      case "playerai_vs_playerai":
>     requirements.txt 72      play1 = self.player1.play
>     requirements.txt 73      play2 = self.player2.play
>     requirements.txt 74      case _: # Game mode does not exist!
>     requirements.txt 75      return
>     requirements.txt 76      except AttributeError: # Game mode does not exist!
>     requirements.txt 77      return

```

Abbildung 45: Code Pit-Anpassung

3. **Frontend:** In den Lobbys muss es zudem möglich sein, neue oder andere Konfigurationen anzuwählen. Um zusätzliche Optionen anbietbar zu machen, also neue Modi oder neue Schwierigkeitsgrade, muss man lediglich die jeweiligen Selects innerhalb der `Frontend/src/components/KIM_Pages/Lobbypage.vue` um die neue Option anpassen, dafür sollte man außerdem neue Einträge innerhalb der `Frontend/src/i18n.js` anlegen, sodass diese neuen Optionen korrekt in allen Sprachen angezeigt werden. Für das Hinzufügen von komplett neuen Optionen, wie beispielsweise Boardgröße muss man ein neues Auswahl Element in die `LobbyPage` einfügen, idealerweise durch ein neues Select. Zudem muss in der `Frontend/src/components/UI/LobbyPage.js` Datei eine neue Variable `i` `data()` angelegt werden, deren Inhalt wie V-model an die Auswahl im Select gebunden ist. Danach muss in der `playCreate()`-Methode die zweite `data` Konstante angepasst werden, sodass diese das korrekte Key-Value-Paar enthält. Man sollte dabei allgemein aufpassen, dass das hinzugefügte Auswahl-Element nur dann angezeigt wird, wenn das Spiel auch andere Konfigurationen anbietet, und versteckt wird, sollte es nicht aufrufbar sein. Zudem sollte man darauf achten, dass die neue Variable nur als Key-Value-Paar hinzugefügt wird, wenn dies für das Spiel sinnvoll ist.

4.3.3 Funktion 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 eine Wahrscheinlichkeitsverteilung erzeugt. Der beste Zug hat die höchste Wahrscheinlichkeit, der schlechteste die niedrigste bzw. gar keine (= 0), falls die Kante im Spielbaum nicht abgelaufen

Weiterentwicklungsdocumentation

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 dessen 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.

Die Heuristik ist dabei noch nicht ganz optimal. Im Laufe der Zeit haben sich 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 etwas deutlich: 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 ergibt sich, dass die Suchtiefe im Suchbaum nicht zu hoch eingestellt werden darf, 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 Spieldaten und -strategien nicht auch gewinnbringend oder gar gewinnbringender sind. „Blunder“ markiert also Spielzüge an, die vielleicht brillant waren. Diese Erkenntnis ergibt sich auch auf anderem Wege. Die Grundlage des AlphaZero-Frameworks ist eine Implementierung für das Spiel Go – AlphaGo von Google Deepmind. Mit dieser KI traten sie im März 2016 in einem internationalen Go-Turnier gegen einen der führenden Weltmeister des Go-Spiels, Lee Seodol, an. In den einen Spiel, das Seodol gegen AlphaGo gewann, nutzte er einige unkonventionelle Züge, die anfänglich unklug wirkten, sich aber im Endeffekt als sehr genial herausstellten und die KI schlagen konnten. Nach unserer Heuristik wäre diese Art von Zügen damals auch als fehlerhaft angemerkert worden, auch wenn sie zum Sieg führten und vielleicht der einzigen Chancen, die KI zu schlagen.



Weiterentwicklungsdocumentation

Folgende Aspekte könnten dementsprechend noch umgesetzt werden:

1. **Asynchronität:** Einerseits die Berechnung asynchron neben das aktiven Spiels laufen lassen, sodass wir auf höherer Tiefe im Monte Carlo Tree (`Tools/mcts.py`) suchen können.
2. **Andere Lösung:** Eine andere oder erweiternde Art der Heuristik, bei der beispielsweise die Spielzustände miteinander verglichen werden und evaluiert wird, binnien wie vielen eigenen Zügen, wie viele eigene oder gegnerische Spielsteine wegfallen. Eine Generalisierung für alle Spiele muss jedoch gewährleistet werden.

4.3.4 Funktion Hint

Um den Spielern zu ermöglichen, sich weiterzuentwickeln, ihre eigene KI zu verbessern oder die Heuristik ihres nicht KI-basierten Algorithmus (für die Programmierung interaktiver Systeme) zu optimieren, könnte eine zusätzliche Funktion integriert werden: „Hint“.

Derzeit verfügen wir bereits über die „Blunder“-Funktion. Diese weist lediglich auf Verbesserungswürdige Züge hin und identifiziert somit nur die Probleme, ohne jedoch Lösungen anzubieten. Spieler und Entwickler sind bei der Evaluation eines besseren Zuges weitgehend auf sich allein gestellt. Hier könnte die „Hint“-Funktion ansetzen, indem sie dem Spieler, den vom neuronalen Netzwerk berechneten besten Zug anzeigt.

Im Frontend könnte dies durch einen zusätzlichen Button im „Blunder“-Fenster umgesetzt werden, der sichtbar wird, wenn man auf „Blunder“ klickt. Die entsprechenden Draw-Methoden müssten so angepasst werden, dass das empfohlene Feld markiert wird. In der Konsole würde der Zug einfach als Nachricht angezeigt und ausgegeben.

Da die Funktion dann ohnehin vorhanden ist, könnte sie auch in ein laufendes Spiel integriert werden. Wenn der Spieler nicht weiß, welchen Zug er machen soll, kann er „Hint“ konsultieren und das Spielfeld wird entsprechend markiert oder der empfohlene Zug wird in der Konsole als String angezeigt.

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.

Weiterentwicklungsdocumentation

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



Weiterentwicklungsdocumentation

Änderungen vornehmen können. Dies trägt nicht nur zur Sicherheit und Integrität der Plattform bei, sondern ermöglicht auch eine effiziente und strukturierte Verwaltung des Systems.

4.4.3 Implementierung von Themes

Viele moderne Plattformen bieten den Nutzern zunehmend die Möglichkeit, das Aussehen der Benutzeroberfläche nach ihren persönlichen Vorlieben anzupassen. Diese Flexibilität in der Gestaltung trägt nicht nur zur Benutzerfreundlichkeit bei, sondern kann auch die Zugänglichkeit und das visuelle Erlebnis erheblich verbessern. Ein häufiges Beispiel für solche Anpassungen ist die Möglichkeit, zwischen einem "Light Mode" und einem "Dark Mode" zu wählen.

Neben dem Light- und Dark Mode könnte es auch sinnvoll sein, ein "High Contrast Theme" zu implementieren. Dieses spezielle Design ist auf hohe Sichtbarkeit und bessere Lesbarkeit optimiert, indem es starke Farbkontraste zwischen Hintergrund und Text bietet. Ein High Contrast Theme ist besonders nützlich für Menschen mit Sehbehinderungen, da es die Lesbarkeit von Texten und die Sichtbarkeit von Bedienelementen verbessert.

Um diese Modifikationen umzusetzen, müssen die CSS-Dateien der Webseite entsprechend angepasst werden. Für jeden Designmodus wird typischerweise ein separates Stylesheet verwendet, das die Farben, Schriftarten und andere visuelle Aspekte definiert. Durch das Laden des jeweiligen Stylesheets kann die Plattform dynamisch zwischen den verschiedenen Designs umschalten.

Nach der Implementierung solcher Designänderungen in der Frontend-Gestaltung ist es erforderlich, auch das Backend, entsprechend anzupassen. Aufgrund der PyGame-Verwendung zum Zeichnen der Spielbretter, ist eine visuelle Anpassungen zur besseren Integration mit dem ausgewählten Design erforderlich.

Um sicherzustellen, dass die gerenderten Spielfelder optisch zum gewählten Theme passen, könnte man die PyGame-Draw-Methode (Zeichenmethode) entsprechend anpassen. Dies kann durch einen Mechanismus geschehen, der ähnlich der Sprachauswahl funktioniert. Bei der Sprachauswahl wird in der Regel eine Konfigurationsdatei oder ein Kommando verwendet, um die Sprache der Anwendung zu ändern. Ähnlich könnte man einen Mechanismus einführen, bei dem die Theme-Einstellungen zentral gesteuert werden. Der Mechanismus könnte ein einzelnes Kommando nutzen, um die visuelle Darstellung der Spielfelder gemäß dem ausgewählten Theme zu aktualisieren.



Weiterentwicklungsdocumentation

4.4.4 Weitere Soundeffekte

Derzeit verwenden alle Spiele in unserer Plattform dieselben Soundeffekte, was möglicherweise zu einer gewissen Monotonie und einem weniger immersiven Spielerlebnis führt. Um das Spielerlebnis zu verbessern und den Spielern ein noch besseres und individuelleres Erlebnis zu bieten, wäre es sinnvoll, die Soundeffekte an die spezifischen Merkmale jedes Spiels anzupassen. Dies könnte insbesondere bei Spielen ohne physische Spielsteine von großem Vorteil sein.

Soundeffekte sind ein entscheidender Bestandteil der Spielerfahrung. Sie tragen wesentlich zur Atmosphäre und zum Spielspaß bei, indem sie das Spielerlebnis lebendiger und realistischer gestalten. Einheitliche Soundeffekte für alle Spiele können jedoch dazu führen, dass der Klang nicht optimal zur Spielumgebung oder zum Thema des Spiels passt. Um die Soundeffekte besser an das jeweilige Spiel anzupassen, könnten verschiedene Ansätze verfolgt werden. Eine Möglichkeit besteht darin, für jedes Spiel spezifische Soundbibliotheken zu erstellen.



Testdokumentation

Testdokumentation

Testdokumentation

5 Testdokumentation

5.1 Teststrategie/-ansatz

Die Teststrategie zielt darauf ab, die Funktionalität und Robustheit der WebSocket-Kommunikation sowie die verschiedenen Lobby-Management-Funktionen zu gewährleisten. Dabei kommen folgende Testmethoden zum Einsatz:

- **Unit Tests:** Einzelne Tests werden erstellt, um spezifische Funktionen und Logik innerhalb der WebSocket-Kommunikation zu überprüfen. Dies umfasst das Testen der einzelnen Kommandos wie das Erstellen, Beitreten, Verlassen von Lobbys sowie das Tauschen von Positionen.
- **Integrations Tests:** Diese Tests überprüfen die Interaktion zwischen verschiedenen Komponenten des Systems, z.B. die Kommunikation zwischen dem Client und dem Server über WebSockets. Hierbei wird überprüft, ob die WebSocket-Nachrichten korrekt gesendet und empfangen werden und ob die Systemantworten den Spezifikationen entsprechen.
- **Performance Tests:** Hierbei wird die Leistungsfähigkeit des Systems unter Belastung getestet, um sicherzustellen, dass die WebSocket-Verbindungen und -Antworten innerhalb der akzeptablen Zeitlimits liegen. Es wird auch überprüft, ob die Systemantwortzeiten den definierten Maximalwerten entsprechen.
- **Smoke-Test:** Die Gesamtheit aller Automatisierten Tests bildet einen Smoke-Test. Dieser Überprüft ob die wichtigsten Bedingungen erfüllt sind, damit die Software überhaupt funktionieren kann.

5.2 Testplanung (Testumgebungen und -daten)

Für die Durchführung der Tests wurde eine speziell konfigurierte Testumgebung eingerichtet. Zur Sicherstellung einer konsistenten und isolierten Testumgebung wurde ein Docker-Container verwendet. Dieser Container ist so konfiguriert, dass er sich wie ein Client mit dem Server verbindet und die entsprechenden WebSocket-Anfragen sendet. Diese Konfiguration ermöglichte es, Tests unter realistischen Bedingungen durchzuführen, ohne dass externe Variablen die Ergebnisse beeinflussen konnten.

5.2.1 Testdaten

Die Testdaten wurden in den Docker-Container integriert und umfassen:

Testdokumentation

- **Test-Nachrichten:** Verschiedene JSON-Nachrichten wurden vorbereitet, um verschiedene Befehle und Szenarien zu testen. Dazu gehören Nachrichten zum Erstellen und Beitreten von Lobbys, Verlassen von Lobbys und andere spezifische Testszenarien.
- **Konfigurationsparameter:** Parameter wie die maximale Antwortzeit wurden in den Tests konfiguriert, um sicherzustellen, dass alle Tests unter den vorgesehenen Bedingungen durchgeführt wurden.

5.2.2 Testdurchführung

Der Test-Docker-Container führte folgende Schritte durch:

1. **Verbindung zum Server:** Der Docker-Container stellte über die konfigurierte WebSocket-URI eine Verbindung zum Server her. Diese Verbindung simulierte das Verhalten eines echten Clients und ermöglichte es, die Kommunikation zwischen Client und Server genau zu testen.
2. **Senden von Anfragen:** Nach dem Herstellen der Verbindung sendete der Docker-Container die vorbereiteten Testnachrichten an den Server. Jede Nachricht wurde verwendet, um spezifische Funktionen und Szenarien zu überprüfen, wie z.B. das Erstellen von Lobbys, das Beitreten oder Verlassen von Lobbys und das Prüfen von Statusabfragen.
3. **Empfangen und Auswerten von Antworten:** Der Docker-Container empfing die Antworten vom Server und überprüfte diese auf Korrektheit. Die Antworten wurden gegen die erwarteten Ergebnisse überprüft, um sicherzustellen, dass der Server die Anfragen korrekt verarbeitet und die richtigen Statuscodes zurückgibt.
4. **Fehlerprotokollierung und Berichterstattung:** Alle erhaltenen Antworten, einschließlich möglicher Fehler oder unerwarteter Ergebnisse, wurden dokumentiert und analysiert. Dies ermöglichte eine detaillierte Untersuchung und Nachverfolgung von Problemen.

5.2.3 Testunabhängigkeit

Alle Tests sind unabhängig voneinander ausführbar, d.h. jeder Test kann einzeln und in beliebiger Reihenfolge gestartet werden. Es ist nicht notwendig Test A zu starten, damit anschließend Test B gestartet werden kann. Test A ist zu jedem Zeitpunkt ausführbar und Test B ist zu jedem anderen Zeitpunkt ausführbar. Außerdem rufen sich Tests niemals gegenseitig auf.

Um die Unabhängigkeit zu gewährleisten, wird innerhalb eines Tests alles vorbereitet und ausgeführt wird, was von der zu testende Funktionalität vorausgesetzt wird.

Testdokumentation

Beispiel:

Test:	Testet, ob es möglich ist eine Lobby zu verlassen
Voraussetzungen:	7. WebSocket-Verbindung muss bestehen 8. Lobby muss erstellt sein 9. Spieler muss in der Lobby sein
Tests:	Beim Senden der 1. leave_lobby_msg muss das der response-Code erfolgreich sein. Beim Senden der 2. leave_lobby_msg muss der response-Code ein Fehler sein, weil der Spieler nicht mehr in der Lobby ist.
Finally:	Verbindung zum WebSocket schließen

Tabelle 9: Testbeispiel

5.2.4 Testreihenfolge

Aus dem obigen Beispiel geht hervor, dass die Voraussetzungen nicht innerhalb des Tests überprüft werden. Dennoch muss auch sichergestellt sein, dass die Voraussetzungen erfüllt sind, darum werden zuerst die Tests für die Voraussetzungen geschrieben und dann erst die Tests für die Funktion selbst. Nach dem obigen Beispiel muss es also vor dem leave_lobby-Test schon mindestens zwei weitere Tests geben, der erste stellt sicher, dass die Verbindung zum WebSocket möglich ist und der zweite Test stellt sicher, dass das Erstellen und Beitreten zu einer Lobby erfolgreich sind. Die Reihenfolge der Tests ist also wie folgt:

Reihenfolge	Testet das ...	Voraussetzung
1.	Verbinden mit WebSocket	keine
2.	Erstellen einer Lobby	Verbunden mit WebSocket (test 1)
3.	Verlassen der Lobby	Verbunden mit WebSocket (test 1); Lobby erfolgreich erstellt (test 2)

Tabelle 10: Testreihenfolge

Die Anordnung der Tests in dieser Reihenfolge führt dazu, dass Unit-Tests weiter oben stehen und dann weiter unten die Integrationstests und Funktionstests kommen.



Testdokumentation

5.3 Automatisierte Testfälle

Testfall	Beschreibung	Erwartetes Ergebnis
WebSocket_connection	Überprüft, ob eine funktionierende WebSocket-Verbindung zum Server hergestellt werden kann.	Die Verbindung wird erfolgreich hergestellt, die Antwortzeit (Ping) liegt unter der maximalen Antwortzeit von 3 Sekunden, und die Verbindung wird ordnungsgemäß geschlossen.
lobby_creation	Testet die erfolgreiche Erstellung einer Lobby.	Die Antwort enthält einen response_code von 100 und einen gültigen Lobby-Schlüssel (key).
joining_existing_lobby	Überprüft, ob ein zweiter Spieler erfolgreich einer bereits erstellten Lobby beitreten kann.	Der Benutzer sollte erfolgreich der Lobby beitreten können. Die Antwort sollte den gleichen key wie beim Erstellen der Lobby enthalten und den response_code 101 zurückgeben.
joining_non_existing_lobby	Testet, ob der Beitrittsversuch zu einer nicht existierenden Lobby korrekt abgelehnt wird.	Der Beitrittsversuch des Spielers sollte nicht erfolgreich sein, und einen response_code von 151 zurückgeben.
creating_multiple_lobbies	Überprüft, ob mehrere Lobbies nacheinander erstellt werden können.	Alle Lobby-Erstellungsvorgang geben einen response_code von 150 zurück.
joining_multiple_lobbies	Testet das wiederholte Beitreten derselben Lobby durch einen Spieler.	Der Benutzer kann nur einmal der Lobby beitreten. Ein zweiter Versuch sollte den response_code 150 zurückgeben.



Testdokumentation

<code>leaving_current_lobby</code>	Überprüft das Verlassen der aktuellen Lobby und das Verhalten bei wiederholtem Verlassen.	Der Benutzer sollte die Lobby erfolgreich verlassen können. Der <code>response_code</code> 102 sollte zurückgeben werden. Ein zweiter Versuch sollte den <code>response_code</code> 153 zurückgeben, da der Benutzer nicht mehr in der Lobby ist.
<code>leaving_non_existing_lobby</code>	Testet den Verlass-Versuch einer nicht existierenden Lobby.	Der Benutzer sollte eine Fehlermeldung erhalten und einen <code>response_code</code> von 153 zurückgeben.
<code>swap_player</code>	Überprüft den erfolgreichen Swap eines Spielers in der Lobby, wenn nur ein Spieler in der Lobby ist.	Der Benutzer sollte erfolgreich wechseln können. Die Antwort sollte den <code>response_code</code> 103 zurückgeben.
<code>swap_player_when_occupied</code>	Testet das Tauschen der Spieler-Position, wenn die Position bereits belegt ist.	Der Benutzer sollte eine Fehlermeldung erhalten. Die Antwort sollte den <code>response_code</code> 155 enthalten.
<code>swap_player_while_not_in_lobby</code>	Testet das Tauschen der Spieler-Position, ohne dass der Benutzer in einer Lobby ist.	Der Benutzer sollte eine Fehlermeldung erhalten. Die Antwort sollte den <code>response_code</code> 153 enthalten.
<code>swap_player_to_undefined_pos</code>	Testet das Tauschen der Spieler-Position zu einer undefinierten/unsinnigen/None Position.	Der Benutzer sollte eine Fehlermeldung erhalten. Die Antwort sollte den <code>response_code</code> 154 enthalten.
<code>swap_player_to_non_sense_pos</code>		

Testdokumentation

<code>swap_player_to_none_pos</code>		
<code>get_pos_of_lobby_creator</code>	Überprüfen der Position des Erstellers einer Lobby.	Die Antwort enthält einen response_code von 104 und die Position ist „p1“
<code>get_pos_of_p2</code>	Überprüfen der Position des zweiten Spielers in einer Lobby.	Die Antwort enthält einen response_code von 104 und die Position ist „p2“
<code>get_pos_of_sp</code>	Überprüfen der Position eines Zuschauers (Spectators) in einer Lobby.	Die Antwort enthält einen response_code von 104 und die Position ist „sp“
<code>get_pos_with_no_lobby</code>	Testet den Abruf der Position, wenn der Spieler sich in keiner Lobby befindet.	Der Benutzer sollte eine Fehlermeldung erhalten. Die Antwort sollte den response_code 153 enthalten.
<code>lobby_status</code>	Testet das Abfragen des Status einer Lobby.	Die Antwort sollte den response_code 105 enthalten und den aktuellen status der Lobby anzeigen, einschließlich der Positionen der Spieler und der Anzahl der Zuschauer.
<code>none_lobby_command_key</code>	Testet das Verhalten des Systems beim Senden einer leeren Lobby-Nachricht.	Der Benutzer sollte eine Fehlermeldung erhalten. Die Antwort sollte den response_code 50 enthalten.
<code>non_sense_lobby_command_key</code>	Überprüft die Reaktion des Servers auf eine Nachricht mit einem ungültigen command_key.	Der Benutzer sollte eine Fehlermeldung erhalten. Die Antwort sollte den response_code 50 enthalten.

Tabelle 11: Automatisierte Testfälle

Testdokumentation

5.4 Testergebnisse/-berichte

Die durchgeführten Tests haben sich auf die Kernfunktionen des Lobby-Management-Systems konzentriert, einschließlich der Verbindung zu WebSocket, der Verwaltung von Lobbys und der Interaktion zwischen verschiedenen Benutzern. Alle grundlegenden Funktionalitäten wurden erfolgreich getestet, und die Tests verliefen gemäß den Erwartungen.

```

swtp-test ===== test session starts =====
swtp-test platform linux -- Python 3.11.9, pytest-8.2.1, pluggy-1.5.0 -- /usr/local/bin/python
swtp-test cachedir: .pytest_cache
swtp-test rootdir: /app/tests
swtp-test plugins: asyncio-0.23.7
swtp-test asyncio: mode=Mode.STRICT
swtp-test collecting ... collected 21 items
swtp-test
swtp-test | api_lobby_test.py::test_websocket_connection PASSED [ 4%]
swtp-test | api_lobby_test.py::test_lobby_creation PASSED [ 9%]
swtp-test | api_lobby_test.py::test_joining_existing_lobby PASSED [ 14%]
swtp-test | api_lobby_test.py::test_joining_non_existing_lobby PASSED [ 19%]
swtp-test | api_lobby_test.py::test_creating_multiple_lobbies PASSED [ 23%]
swtp-test | api_lobby_test.py::test_joining_multiple_lobbies PASSED [ 28%]
swtp-test | api_lobby_test.py::test_leaving_current_lobby PASSED [ 33%]
swtp-test | api_lobby_test.py::test_leaving_non_existing_lobby PASSED [ 38%]
swtp-test | api_lobby_test.py::test_swap_player PASSED [ 42%]
swtp-test | api_lobby_test.py::test_swap_player_when_occupied PASSED [ 47%]
swtp-test | api_lobby_test.py::test_swap_player_while_not_in_lobby PASSED [ 52%]
swtp-test | api_lobby_test.py::test_swap_player_to_undefined_pos PASSED [ 57%]
swtp-test | api_lobby_test.py::test_swap_player_to_non_sense_pos PASSED [ 61%]
swtp-test | api_lobby_test.py::test_swap_player_to_none_pos PASSED [ 66%]
swtp-test | api_lobby_test.py::test_get_pos_of_lobby_creator PASSED [ 71%]
swtp-test | api_lobby_test.py::test_get_pos_of_p2 PASSED [ 76%]
swtp-test | api_lobby_test.py::test_get_pos_of_sp PASSED [ 80%]
swtp-test | api_lobby_test.py::test_get_pos_with_no_lobby PASSED [ 85%]
swtp-test | api_lobby_test.py::test_lobby_status PASSED [ 90%]
swtp-test | api_lobby_test.py::test_none_lobby_command_key PASSED [ 95%]
swtp-test | api_lobby_test.py::test_non_sense_lobby_command_key PASSED [100%]
swtp-test ===== 21 passed in 13.75s =====
swtp-test exited with code 0

```

Abbildung 46: Testergebnisse erfolgreich

Unerwartetes Verhalten von Messages: WebSocket erlaubt das Senden und Empfangen von Nachrichten. Alle Nachrichten die über die Verbindung empfangen werden, landen in einer Queue und werden mit Hilfe des `recv()` Befehls gelesen und aus der Queue entfernt. Manchmal kann es sein, dass der Server eine Message an einen oder mehrere Clients sendet, ohne dass er auf eine Request antwortet, z.B. sendet der Server einen Broadcast, sobald sich ein neuer Spieler verbindet, dann liegt der Broadcast ganz vorne in der Queue und man muss sie erst mit `recv()`



Testdokumentation

auslesen, damit man die nachfolgenden Messages lesen kann. Das passiert beispielsweise im Test: `test_swap_player_when_occupied`.

5.5 Ausführliche Fehlersuche und Bugfixing

Im Rahmen der Testdurchführung wurde auch eine gezielte und umfassende Fehlersuche auf der Webseite durchgeführt. Die Teststrategien wurden durch gezielte Versuche ergänzt, um verschiedene Bugs zu provozieren und deren Verhalten systematisch zu untersuchen. Dies beinhaltete unter anderem:

- **Proaktive Fehlersuche:** Durch das absichtliche Senden fehlerhafter oder unvollständiger WebSocket-Nachrichten, sowie das Testen von Grenzfällen und ungewöhnlichen Nutzungsszenarien wurden potenzielle Schwachstellen im System aufgedeckt.
- **Provozierte Bugs:** Durch aktives provozieren von Fehlern konnten einige Schwachstellen und noch fehlerhafte Verhaltensweisen aufgedeckt und behoben werden. Dies beinhaltete vor allem das Verhalten der Webseite und in den Spielen und Lobbys angebotenen Funktionen.
- **Simulierte Fehlerszenarien:** Um die Robustheit der Webseite zu prüfen, wurden Szenarien nachgestellt, die möglicherweise zu Fehlfunktionen führen könnten. Dazu gehörte das Einbringen von ungültigen oder inkonsistenten Daten in die WebSocket-Nachrichten und der Versuch verbotene Aktionen auszuführen.
- **Fehleranalyse und Behebung:** Bei der Identifizierung von Fehlern wurden die Ursachen systematisch analysiert. Die Fehler wurden dokumentiert, und die notwendigen Anpassungen im Code wurden vorgenommen, um die gefundenen Probleme zu beheben. Besondere Aufmerksamkeit galt der Stabilität und Fehlertoleranz des Systems.
- **Retests:** Nach jeder Bugfix-Maßnahme wurden gezielte Retests durchgeführt, um sicherzustellen, dass die vorgenommenen Änderungen die beabsichtigte Wirkung hatten und keine neuen Fehler eingeführt wurden. Diese Retests bestätigten, dass die behobenen Fehler nun zuverlässig nicht mehr auftreten.



Technische Dokumentation

6 Literaturverzeichnis

1. THM. Prof. Dr. Frank Kammer. [Online]. Verfügbar unter: <https://www.thm.de/mni/frank-kammer> [Zugriff am: 24. Juli 2024].
2. THM. THM - Technische Hochschule Mittelhessen. [Online]. Verfügbar unter: <https://www.thm.de/site/> [Zugriff am: 24. Juli 2024].
3. 12ghostrider21. KIMaster. [Online]. Verfügbar unter: <https://github.com/12ghostrider21/KIMaster> [Zugriff am: 24. Juli 2024].
4. Docker, Inc. Docker. [Online]. Verfügbar unter: <https://www.docker.com/> [Zugriff am: 24. Juli 2024].
5. Tiangolo. FastAPI. [Online]. Verfügbar unter: <https://fastapi.tiangolo.com/> [Zugriff am: 24. Juli 2024].
6. Mozilla Developer Network. HTTP - MDN Web Docs. [Online]. Verfügbar unter: <https://developer.mozilla.org/en-US/docs/Web/HTTP> [Zugriff am: 24. Juli 2024].
7. Mozilla Developer Network. WebSockets API - MDN Web Docs. [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API [Zugriff am: 24. Juli 2024].
8. JSON. JSON - JSON. [Online]. Verfügbar unter: <https://www.json.org/json-de.html> [Zugriff am: 24. Juli 2024].
9. Mozilla Developer Network. SPA - MDN Web Docs. [Online]. Verfügbar unter: <https://developer.mozilla.org/en-US/docs/Glossary/SPA> [Zugriff am: 24. Juli 2024].
10. Vue.js. Vue.js. [Online]. Verfügbar unter: <https://vuejs.org/> [Zugriff am: 24. Juli 2024].
11. Mozilla Developer Network. HTML - MDN Web Docs. [Online]. Verfügbar unter: <https://developer.mozilla.org/en-US/docs/Web/HTML> [Zugriff am: 24. Juli 2024].
12. CSS.de. CSS. [Online]. Verfügbar unter: <https://www.css.de/> [Zugriff am: 24. Juli 2024].
13. Mozilla Developer Network. JavaScript - MDN Web Docs. [Online]. Verfügbar unter: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> [Zugriff am: 24. Juli 2024].
14. Vuex. Vuex. [Online]. Verfügbar unter: <https://vuex.vuejs.org/> [Zugriff am: 24. Juli 2024].
15. Node.js. Node.js. [Online]. Verfügbar unter: <https://nodejs.org/en> [Zugriff am: 24. Juli 2024].
16. Docker, Inc. Docker Compose Documentation. [Online]. Verfügbar unter: <https://docs.docker.com/compose/> [Zugriff am: 24. Juli 2024].
17. Python Software Foundation. Python. [Online]. Verfügbar unter: <https://www.python.org/> [Zugriff am: 24. Juli 2024].
18. TypeScript Team. TypeScript. [Online]. Verfügbar unter: <https://www.typescriptlang.org/> [Zugriff am: 24. Juli 2024].

Technische Dokumentation

19. DeepMind. *AlphaZero: Shedding New Light on Chess, Shogi, and Go.* [Online]. Verfügbar unter: <https://deepmind.google/discover/blog/alphazero-shedding-new-light-on-chess-shogi-and-go/> [Zugriff am: 24. Juli 2024].
20. pytest Development Team. *pytest Documentation.* [Online]. Verfügbar unter: <https://docs.pytest.org/en/stable/> [Zugriff am: 24. Juli 2024].
21. TensorFlow Team. *TensorFlow.* [Online]. Verfügbar unter: <https://www.tensorflow.org/> [Zugriff am: 24. Juli 2024].
22. Python Package Index. *PyPI.* [Online]. Verfügbar unter: <https://pypi.org/> [Zugriff am: 24. Juli 2024].
23. Vue.js Team. *Vue Router.* [Online]. Verfügbar unter: <https://router.vuejs.org/> [Zugriff am: 24. Juli 2024].
24. Cisco. *Secure Client.* [Online]. Verfügbar unter: <https://www.cisco.com/site/de/de/products/security/secure-client/index.html> [Zugriff am: 24. Juli 2024].
25. Docker, Inc. *Docker Desktop.* [Online]. Verfügbar unter: <https://www.docker.com/products/docker-desktop/> [Zugriff am: 24. Juli 2024].
26. Git SCM. *Git.* [Online]. Verfügbar unter: <https://git-scm.com> [Zugriff am: 24. Juli 2024].
27. GitHub, Inc. *GitHub.* [Online]. Verfügbar unter: <https://github.com/> [Zugriff am: 24. Juli 2024].
28. Atlassian. *Jira.* [Online]. Verfügbar unter: <https://www.atlassian.com/de/software/jira> [Zugriff am: 24. Juli 2024].
29. Simon Tatham. *PutTY.* [Online]. Verfügbar unter: <https://www.putty.org/> [Zugriff am: 24. Juli 2024].
30. Sphinx Team. *Sphinx Documentation.* [Online]. Verfügbar unter: <https://www.sphinx-doc.org/en/master/> [Zugriff am: 24. Juli 2024].
31. JSDoc Team. *JSDoc.* [Online]. Verfügbar unter: <https://jsdoc.app/> [Zugriff am: 24. Juli 2024].
32. Vite Team. *Vite.* [Online]. Verfügbar unter: <https://vitejs.dev/> [Zugriff am: 24. Juli 2024].
33. THM. *Jonas-Ian Küche.* [Online]. Verfügbar unter: <https://www.thm.de/mni/jonas-ian-kueche> [Zugriff am: 24. Juli 2024].
34. Google. *Google Chrome.* [Online]. Verfügbar unter: https://www.google.com/intl/de_de/chrome/ [Zugriff am: 24. Juli 2024].
35. PyTorch Team. *PyTorch.* [Online]. Verfügbar unter: <https://pytorch.org/> [Zugriff am: 24. Juli 2024].
36. Keras Team. *Keras.* [Online]. Verfügbar unter: <https://keras.io/> [Zugriff am: 24. Juli 2024].
37. NumPy Team. *NumPy.* [Online]. Verfügbar unter: <https://numpy.org/> [Zugriff am: 24. Juli 2024].

Technische Dokumentation

38. Pygame Team. *Pygame News*. [Online]. Verfügbar unter: <https://www.pygame.org/news> [Zugriff am: 24. Juli 2024].
39. Suragnair. *alpha-zero-general*. [Online]. Verfügbar unter: <https://github.com/suragnair/alpha-zero-general> [Zugriff am: 24. Juli 2024].
40. Suragnair. *alpha-zero-general* - Forks. [Online]. Verfügbar unter: https://github.com/suragnair/alpha-zero-general/forks?include=active&page=1&period=&sort_by=stargazer_counts [Zugriff am: 24. Juli 2024].
41. Vinoo999. *alpha-zero-general*. [Online]. Verfügbar unter: <https://github.com/vinoo999/alpha-zero-general> [Zugriff am: 24. Juli 2024].
42. Visuallization. *alpha-zero-general-hex*. [Online]. Verfügbar unter: <https://github.com/visuallization/alpha-zero-general-hex> [Zugriff am: 24. Juli 2024].
43. t-rekttt. *alpha-zero-general*. [Online]. Verfügbar unter: <https://github.com/t-rekttt/alpha-zero-general> [Zugriff am: 24. Juli 2024].
44. LittleH0rst. *alpha-zero-general*. [Online]. Verfügbar unter: <https://github.com/LittleH0rst/alpha-zero-general> [Zugriff am: 24. Juli 2024].

Technische Dokumentation

7 Glossar

7.1 Abkürzungsverzeichnis

Abkürzung	Beschreibung
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
BIOS	Basic Input/Output System
CPU	Central Processing Unit
CSS	Cascading Styles Sheets
CUDA	Compute Unified Device Architecture
DOM	Document Object Model
GB	Gigabyte
GiB	Gibibyte
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP(s)	Hypertext Transfer Protocol (Secure)
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz
NVMe	Nonvolatile Memory Express
QR-Code	Quick Response Code
RAID	Redundant Array of Independent Disks
RAM	Random-Access Memory
SLAT	Secon Level Adress Translation
SPA	Single Page Application
SSD	Solid-State Disk
SSH	Secure Shell
UI	User Interface
VPN	Virtual Private Network

Tabelle 12: Abkürzungsverzeichnis



Technische Dokumentation

7.2 Begriffserklärungen

Begriff	Definition
2-dimensionales Spielbrett	Ein Spielfeld, das auf zwei Dimensionen (x- und y-Achse) dargestellt wird, typischerweise in einem Rasterformat. Beispiele sind Schach und Tic-Tac-Toe.
Administratorenrollen	Verschiedene Rollen und Rechte für Administratoren zur Verwaltung der Nutzerdaten, Moderation von Inhalten und Sicherstellung der Einhaltung von Richtlinien.
AlphaZero-Framework	Ein Framework zur Implementierung und Erweiterung von KI-Komponenten durch Training von neuronalen Netzen, insbesondere für komplexe Spiele.
API-Dokumentation	Eine detaillierte Beschreibung der API, einschließlich Modulübersicht, Klassen- und Methodenbeschreibungen, und Quellcodebeispiele, die mit Sphinx erstellt werden.
ArenaCompare	Die Anzahl der Testspiele, in denen das neu trainierte Modell gegen ein bestehendes Modell getestet wird, um dessen Leistungsfähigkeit zu vergleichen.
Asynchronität	Modus der Kommunikation, bei dem das Senden und Empfangen von Daten zeitlich versetzt und ohne Blockieren des Prozesses durch bspw. Warten auf die Antwort des Empfängers (wie bei synchroner Kommunikation der Fall) stattfindet.
Automatisierte Testfälle	Vordefinierte Tests, die automatisch ausgeführt werden, um spezifische Funktionalitäten zu überprüfen.
Auto-Test-Komponente	Ein Bestandteil des Systems, der automatisierte Tests durchführt, um die Funktionalität der verschiedenen Systemkomponenten zu überprüfen und sicherzustellen, dass sie korrekt arbeiten.
Benutzeranmeldung	Einführung eines Systems zur Anmeldung und Registrierung von Benutzern, um individuelle Fortschritte und Erfolge zu speichern.
Benutzerhandbuch	Ein Dokument, das die Nutzung der Software oder Plattform für Endnutzer erklärt, einschließlich Anleitungen und Tipps zur Verwendung der verschiedenen Funktionen.
Branching-Strategie	Eine Methode zur Versionskontrolle in Git, bei der separate Branches für verschiedene Funktionen und Entwicklungen verwendet werden. Der main-Branch enthält den stabilen, produktionsbereiten Code.



Technische Dokumentation

Build-Cache	Ein Speicherbereich, der Zwischenresultate von vorherigen Builds speichert, um wiederholte Berechnungen zu vermeiden und die Build-Zeit zu verkürzen.
Build-Prozess	Der Vorgang, bei dem der Quellcode einer Software kompiliert und alle Abhängigkeiten und Konfigurationen integriert werden, um ein ausführbares Programm oder ein Container-Image zu erstellen.
CamelCase	Eine Namenskonvention, bei der jedes Wort innerhalb eines Begriffs mit einem Großbuchstaben beginnt.
Cisco Secure Client	VPN-Client zur sicheren Verbindung zur Universität beim Deployment der Anwendung.
CUDA	Compute Unified Device Architecture, eine parallele Rechenarchitektur von NVIDIA, die die Nutzung von GPUs für allgemeine Berechnungen ermöglicht.
Datenmanagement und Datenschutz	Verwaltung und Schutz der Benutzerdaten durch Implementierung von Sicherheitsprotokollen, Verschlüsselung und klaren Datenschutzrichtlinien.
Debug Seite	Eine spezielle Seite zur Fehlerbehebung und zum Testen der Spiele im Frontend. Hier werden neue Spiele als Auswahloptionen hinzugefügt, und die Steuerung wird angepasst, um neue Methoden zu unterstützen.
Debug-Webseite	Eine vereinfachte Version der Hauptwebseite, die für Debugging-Zwecke verwendet wird und keine komplexen Routing- oder State-Management-Komponenten enthält.
Deployment	Der Prozess, bei dem eine Software auf einem Server oder in einer Produktionsumgebung bereitgestellt wird, sodass sie von Nutzern verwendet werden kann.
Determinismus	Ein Spielmechanismus, bei dem jede mögliche Entscheidung und der Zustand des Spiels eindeutig durch die Regeln des Spiels festgelegt sind, was die Berechnung und Simulation durch AlphaZero ermöglicht.
Docker	Eine Plattform zur Containerisierung von Anwendungen, die es ermöglicht, Software in isolierten Umgebungen (Containern) auszuführen.
Docker Compose	Ein Werkzeug zur Definition und Verwaltung mehrerer Docker-Container, das Orchestrierung und Integration von containerbasierten Anwendungen ermöglicht.



Technische Dokumentation

Docker Desktop	Eine Benutzeroberfläche zur Verwaltung von Docker-Containern und -Images auf dem lokalen Rechner.
Docker-Container	Isolierte Umgebungen, die Software und ihre Abhängigkeiten enthalten und die Ausführung von Anwendungen in verschiedenen Umgebungen ermöglichen.
Docker-Netzwerk	Eine Netzwerkarchitektur, die in der Docker-Umgebung verwendet wird, um Container miteinander zu verbinden und zu kommunizieren.
DocStrings	Dokumentationsstrings im Quellcode, die zur Beschreibung der Funktionalität und Nutzung von Code-Elementen verwendet werden.
Dokumentation	Eine Sammlung von Dokumenten, die technische Details, Anleitungen und Beschreibungen eines Projekts enthalten. Sie dient dazu, die Softwareentwicklung und -nutzung zu erklären und zu unterstützen.
Eindimensionale Handhabung	Eine Methode zur Darstellung von Spielzügen als einzelne Indizes statt als x- und y-Koordinaten. Dies vereinfacht die Kommunikation und die Implementierung im Frontend.
Endbedingungen	Bedingungen, die bestimmen, wann ein Spiel endet, z.B. nach einer bestimmten Anzahl von Zügen ohne Fortschritt, um ein Unentschieden zu vermeiden oder endlose Schleifen zu verhindern.
Entwicklungs-dokumentation	Dokumentation, die den gesamten Entwicklungsprozess eines Projekts beschreibt, einschließlich der Designentscheidungen, der Implementierung und der verwendeten Technologien.
Enum	Eine Datenstruktur in der Programmierung, die eine feste Menge von Werten definiert. In diesem Kontext wird ein Enum verwendet, um verschiedene Spiele in der Anwendung zu repräsentieren.
EPOCH	Eine einzelne Durchlaufzeit des Trainingsprozesses, in der alle Trainingsdaten einmal durch das Modell laufen. Der Wert gibt an, wie viele Epochen das Modell durchläuft, um trainiert zu werden.
FastAPI Server	Ein Server, der mit FastAPI entwickelt wurde und als zentrale Schnittstelle für die Kommunikation zwischen verschiedenen Systemkomponenten und externen Anwendungen dient. Er unterstützt WebSockets für bidirektionale Echtzeitkommunikation.



Technische Dokumentation

Fehlerbehandlung	Techniken zur Handhabung von Fehlern und Ausnahmen, um die Stabilität und Benutzerfreundlichkeit von Anwendungen zu verbessern.
Fehlerprotokollierung	Dokumentation aller Fehler und unerwarteten Ergebnisse während der Tests.
Fehlersuche	Prozess zur Identifizierung und Behebung von Fehlern im System.
Funktion Blunder	Eine Funktion zur Identifizierung von suboptimalen Zügen durch Vergleich der Wahrscheinlichkeitsverteilung des neuronalen Netzes für jeden möglichen Zug im Spiel.
Funktion Hint	Eine Funktion, die den Spielern den besten Zug gemäß der Berechnung des neuronalen Netzes zeigt, um die Entscheidungsfindung zu unterstützen.
Game Service	Die zentrale Komponente eines GameClients, die die Koordination der verschiedenen Spielmodule übernimmt und die Kommunikation mit dem Server sicherstellt.
GameClients	Komponenten, die die Spielumgebung repräsentieren und die Spiellogik implementieren. Sie bestehen aus dem Game Service, Pit, Arena und Player.
Git	Ein verteiltes Versionskontrollsystem, das die Verwaltung von Quellcode ermöglicht und Änderungen im Code nachvollziehbar macht.
GitHub	Eine Plattform für die Versionierung und Verwaltung von Quellcodeprojekten. Sie ermöglicht es, den Quellcode und alle relevanten Projektdateien zu speichern, zu versionieren und gemeinsam mit anderen Entwicklern daran zu arbeiten.
GitHub	Eine Plattform für die Verwaltung von Git-Repositories, die kollaboratives Arbeiten an Codeprojekten ermöglicht.
High Contrast Theme	Ein Designmodus, der hohe Farbkontraste bietet, um die Lesbarkeit und Sichtbarkeit für Menschen mit Sehbehinderungen zu verbessern.
HTML	Hypertext Markup Language, eine Sprache zur Strukturierung und Darstellung von Inhalten im Webbrowser.
Imports	Die Praxis des Einfügens von externen Bibliotheken und Modulen in den Quellcode, wobei externe und interne Module klar getrennt werden.
Integrations Tests	Tests zur Überprüfung der Interaktion zwischen verschiedenen Komponenten des Systems.



Technische Dokumentation

Interface-Methoden	Methoden, die in einem Interface definiert sind und von konkreten Klassen implementiert werden müssen. Diese Methoden spezifizieren die notwendige Funktionalität für die Integration von Spielen in das AlphaZero-Framework.
JavaScript	Eine Programmiersprache, die für die Entwicklung dynamischer Inhalte in Webanwendungen verwendet wird.
Jira	Ein Projektmanagement-Tool, das zur Planung, Verfolgung und Verwaltung von Aufgaben und Projekten verwendet wird. Es bietet Funktionen für Zeitverfolgung und Aufgabenmanagement.
Kanban Board	Ein Werkzeug zur Visualisierung des Arbeitsflusses, das in Jira verwendet wird, um Aufgaben in verschiedenen Phasen darzustellen.
Keras	Eine benutzerfreundliche API für neuronale Netzwerke, die auf TensorFlow aufbaut und die Erstellung und das Training von Modellen vereinfacht.
KIMaster	Benutzerfreundliche Plattform, die Nutzern ermöglicht, praxisnahe Erfahrungen mit künstlicher Intelligenz (KI) zu sammeln, indem sie verschiedene KIs gegeneinander antreten lassen und eigene KI-Entwicklungen testen können.
Lastenheft	Ein Dokument, das die Anforderungen und Ziele eines Projekts aus der Sicht des Auftraggebers beschreibt. Es bildet die Grundlage für das Pflichtenheft.
Modulübersicht	Ein Überblick über alle Module eines Projekts, der deren Funktionalität und Struktur beschreibt.
Neural Network	Ein Modell, das durch das Training auf Daten Muster und Zusammenhänge erkennen kann, und oft in der künstlichen Intelligenz verwendet wird.
Neuronales Netz	Ein Modell des maschinellen Lernens, das auf dem biologischen Modell des Gehirns basiert. Es besteht aus Schichten von Neuronen, die Eingabedaten verarbeiten und Muster oder Zusammenhänge lernen.
Node.js HTTP-Server	Ein Server, der mit Node.js entwickelt wurde und HTTP-Anfragen bearbeitet, verwendet für die Bereitstellung von Produktions- und Debug-Versionen der Webseite.
NumEps (Number of Episodes)	Die Anzahl der vollständigen Spiele, die während des Trainings gespielt werden, um Daten für das Training des neuronalen Netzes



Technische Dokumentation

	zu sammeln. Diese Daten umfassen die Spielbrettzustände und die zugehörigen Policy-Vektoren.
NumIters (Number of Iterations)	Die Anzahl der vollständigen Trainingsdurchläufe, die das Modell durchläuft. Jeder Durchlauf umfasst Selbstspiele sowie Testspiele gegen andere Modelle.
Performance Tests	Tests zur Überprüfung der Leistungsfähigkeit eines Systems unter Belastung.
Pflichtenheft	Ein Dokument, das die detaillierten funktionalen und qualitativen Anforderungen eines Projekts beschreibt. Es dient als verbindliche Grundlage für die Umsetzung des Projekts.
Provozierte Bugs	Fehler, die durch absichtliche Fehler oder ungewöhnliche Nutzungsszenarien im System verursacht werden.
PutTY	Ein beliebtes Programm für die Erstellung von SSH-Verbindungen zu entfernten Servern, um auf diese zugreifen und Befehle ausführen zu können.
Python	Eine Programmiersprache, die für Backend-Entwicklung, maschinelles Lernen und Datenverarbeitung verwendet wird.
Python Typannotationen	Ein Feature in Python, das zur Angabe von Datentypen für Variablen, Parameter und Rückgabewerte verwendet wird, um die Lesbarkeit und Fehlervermeidung zu verbessern.
Pytorch	Eine populäre Open-Source-Bibliothek für maschinelles Lernen, die sich durch hohe Flexibilität und Geschwindigkeit beim Training von neuronalen Netzwerken auszeichnet.
Quellcode	Der Code, der die Programmierung der Software beschreibt und die Anweisungen enthält, die vom Computer ausgeführt werden.
RAID 0	Eine Speicher-Array-Konfiguration, bei der Daten auf mehrere Laufwerke verteilt werden, um die Leistung zu verbessern.
Retests	Wiederholte Tests nach der Behebung von Fehlern, um sicherzustellen, dass die Änderungen wirksam waren und keine neuen Fehler entstanden sind.
Router	Eine Komponente in Vue.js, die für die Verwaltung und Navigation zwischen verschiedenen Ansichten oder Seiten innerhalb einer Single Page Application verantwortlich ist.
Schwierigkeitsgrade	Unterschiedliche Level, die die Komplexität und Herausforderung eines Spiels anpassen.



Technische Dokumentation

Scrum	Ein agiles Projektmanagement-Framework, das auf regelmäßigen, zeitlich begrenzten Sprints basiert, um Projekte iterativ voranzutreiben.
Selbstspiele (Self Plays)	Spiele, die von der KI gegen sich selbst gespielt werden, um Erfahrungen zu sammeln und das Modell zu trainieren. Diese Selbstspiele werden genutzt, um Trainingsdaten zu generieren.
Simulierte Fehlerszenarien	Nachstellung möglicher Fehlersituationen, um die Robustheit des Systems zu testen.
Single Page Application (SPA)	Eine Webanwendung, die als einzige HTML-Seite geladen wird und dynamisch Inhalte nachlädt, um ein nahtloses Benutzererlebnis zu bieten.
Smoke-Test	Überprüfung, ob die grundlegenden Funktionen der Software funktionieren; eine Art erster Schnelltest.
snake_case	Eine Namenskonvention, bei der Worte durch Unterstriche getrennt und alle Buchstaben klein geschrieben werden.
Socket Server	Ein Server, der die bidirektionale Kommunikation über WebSockets ermöglicht und Aufgaben wie Lobbymanagement und Nachrichtenverarbeitung übernimmt.
Solararray	Eine Anordnung von Solarzellen zur Erzeugung von elektrischer Energie aus Sonnenlicht.
Soundbibliotheken	Sammlung von spezifischen Soundeffekten für jedes Spiel, um die Audioerfahrung besser an die jeweilige Spielumgebung anzupassen.
Sphinx	Ein Dokumentationswerkzeug für Software, das zur Dokumentation von Quellcode verwendet wird.
Spielfeldkonfigurationen	Anpassung der Größe und Form des Spielbretts, um unterschiedliche Spielvariationen und -schwierigkeiten zu ermöglichen.
Spielmodi	Verschiedene Modi oder Spielarten, die alternative Spielbedingungen oder -regeln anbieten.
Spielvariationen	Verschiedene Regelsets oder Versionen eines Spiels, die durch Anpassungen im Spielablauf oder der Spielfeldgröße entstehen.
Sprachanbindung	Die Integration eines Spiels in die Sprachmodule einer Anwendung, um die Mehrsprachigkeit zu unterstützen. Hierbei wird der Name und die Regeln des Spiels in verschiedenen Sprachen definiert.



Technische Dokumentation

SSH (Secure Shell)	Ein Netzwerkprotokoll zur sicheren Übertragung von Daten zwischen einem Client und einem Server, das häufig für Remote-Zugriffe und -Verwaltung verwendet wird.
Store (Vuex)	Ein State-Management-System in Vue.js, das den globalen Zustand der Anwendung verwaltet und für konsistente Datenzugriffe sorgt.
Testdaten	Daten, die verwendet werden, um Tests durchzuführen, einschließlich Nachrichten und Konfigurationsparameter.
Testdokumentation	Dokumentation, die die durchgeführten Tests der Software beschreibt, einschließlich der Testmethoden, Testfälle und Testergebnisse.
Testdurchführung	Der Prozess des Testens, einschließlich Verbindungsherstellung, Nachrichtensenden, Antwortempfang und Fehlerprotokollierung.
Testreihenfolge	Die festgelegte Reihenfolge, in der Tests durchgeführt werden sollten, um die korrekte Ausführung sicherzustellen.
Teststrategie/-ansatz	Geplante Vorgehensweise zur Überprüfung der Funktionalität und Robustheit des Systems, einschließlich spezifischer Tests.
Testumgebung	Die spezielle Umgebung, in der Tests durchgeführt werden, um reale Bedingungen zu simulieren.
Testunabhängigkeit	Eigenschaft, dass Tests unabhängig voneinander durchgeführt werden können, ohne dass die Ergebnisse anderer Tests beeinflusst werden.
Training	Der Prozess, bei dem das Modell für eine neue Spielvariante mit den spezifischen Daten trainiert wird, um ein neues Modell zu erstellen, das im Spielordner gespeichert wird.
Trainingseinstellungen	Parameter, die angepasst werden können, um das Verhalten und die Effizienz des Trainingsprozesses zu steuern, wie Iterationen und Episoden.
Unit Tests	Tests, die einzelne Funktionen oder Logik innerhalb eines Moduls oder einer Komponente überprüfen.
ValueError	Ein Fehler, der ausgelöst wird, wenn ein ungültiger Zug gemacht wird.
Vite	Ein modernes Build-Tool für die Frontend-Entwicklung, das auch als Development Server dient.
Vollständige Information	Ein Spiel, bei dem beide Spieler jederzeit alle relevanten Informationen über den Zustand des Spiels haben, im Gegensatz zu Spielen mit verdeckten Informationen.

Technische Dokumentation

VPN (Virtual Private Network)	Eine Technologie, die eine sichere, verschlüsselte Verbindung über ein öffentliches Netzwerk wie das Internet ermöglicht und den Zugriff auf private Netzwerke wie das der Hochschule erlaubt.
Vue Komponenten	Einzelne, wiederverwendbare Bausteine in einer Vue.js-Anwendung, die jeweils eine bestimmte Funktionalität oder Ansicht bieten.
Vue.js	Ein JavaScript-Framework, das für die Entwicklung von Single Page Applications verwendet wird und eine reaktive Benutzeroberfläche ermöglicht.
WebSocket	Ein Kommunikationsprotokoll, das eine bidirektionale, kontinuierliche Verbindung zwischen Client und Server ermöglicht, ideal für Echtzeitanwendungen.
Weiterentwicklungs-dokumentation	Dokumentation, die Hinweise und Anleitungen für zukünftige Erweiterungen und Verbesserungen der Software enthält.
Worklogs	Ein Jira-Plugin zur Zeiterfassung, mit dem Teammitglieder ihre Arbeitszeit auf bestimmte Aufgaben und Tickets buchen können.
Zwischenspeicherung (Checkpoint)	Die Speicherung des Modells und der Trainingsdaten zu einem bestimmten Zeitpunkt während des Trainings. Diese Daten ermöglichen es, das Training später fortzusetzen oder den Fortschritt nachzuvollziehen.

Tabelle 13: Begriffs-Glossar