



INTERNATIONALE
HOCHSCHULE

Projektplan – DLMCSPSE01_D Projekt: Software Engineering

Erstellt von: Dustin Lucht
Matrikelnummer: 92203340
Studiengang: Master of Science Informatik
Tutor: Markus Kleffmann
Datum: 17.12.2023

1 Ziele, Umfang und angestrebtes Ergebnis des Projekts

Das Ziel dieses Projekts ist es, ein interaktives Schachspiel in Python zu entwickeln, das die Möglichkeit bietet, gegen einen Computer oder zu zweit zu spielen. Das Spiel soll eine besondere Note durch die Integration von Power-Ups erhalten, die dem Spieler strategische Vorteile verschaffen. Zu den Power-Ups gehören: Zwei Züge hintereinander, Zufällige Zerstörung einer gegnerischen Figur, zufällige Promotion eines eigenen Bauers und der Vorschlag des eines guten nächsten Zuges durch eine Schach-Engine. Das angestrebte Ergebnis ist ein benutzerfreundliches Schachspiel mit intuitiver Bedienung und spannenden Spielmöglichkeiten dank der Power-Ups.

2 Anvisierte Zielgruppe

Die Hauptzielgruppe dieses Spiels sind Schachliebhaber und Gelegenheitsspieler, die eine besondere Herausforderung suchen. Durch die Integration von Power-Ups wird auch eine jüngere Zielgruppe angesprochen, die an einem innovativen Ansatz für das klassische Schachspiel interessiert ist.

3 Potenzielle Projektrisiken und Gegenmaßnahmen

- Komplexität der Power-Up-Implementierung:

Es könnte herausfordernd sein, die Power-Ups nahtlos in das Spiel zu integrieren.

- Gegenmaßnahme: Eine sorgfältige Planung der Architektur und regelmäßige Code-Reviews werden durchgeführt, um mögliche Komplikationen frühzeitig zu erkennen.

- Erforderte Rechenleistung:

Bei zu geringer Rechenleitung könnten Spielzüge zu lange zum Berechnen benötigen.

- Gegenmaßnahme: Durch regelmäßige Performance-Tests mit anschließenden Code-Reviews sollen rechenintensive Code-Stellen frühzeitig ausfindig gemacht und verbessert werden. Zudem soll die Stärke des Computer Gegners einstellbar sein. So kann die Rechenzeit reduziert werden.

- Benutzeroberfläche und Benutzererfahrung:

Die Gestaltung einer intuitiven Benutzeroberfläche könnte Schwierigkeiten bereiten.

- Gegenmaßnahme: Regelmäßige Usability-Tests und Anpassungen der Benutzeroberfläche basierend auf dem Feedback der Testpersonen (Bekannte aus dem Freundeskreis).

- Balancing der Power-Ups:

Es könnte schwierig sein, ein ausgewogenes Verhältnis zwischen den verschiedenen Power-Ups zu finden, um ein faires Spiel zu gewährleisten.

- Gegenmaßnahme: Ausgiebige Tests und möglicherweise die Implementierung von Anpassungsmöglichkeiten für die Power-Ups, oder die Reduzierung der Wahrscheinlichkeiten.

- Nicht-gefallen der Idee

Konservative Schachspieler könnten die innovative Spielidee nicht mögen:

- Gegenmaßnahme: Die Powerups könnten optional an- oder ausgeschaltet werden.

4 Zeitplan und Meilensteine

Wann (bis Datum)	Thema	Beschreibung
Woche 1-2 (11.10.23)	Konzept und Grundstruktur	Brainstorming und Ausarbeitung des Spielkonzepts inklusive Power-Up-Liste.
		Erstellung der Projektstruktur und Initialisierung des Versionskontrollsysteams
		Separates Testen der zu nutzenden externen Bibliotheken
Woche 3-4 (25.10.23)	Spiellogik und Grundfunktionalitäten	Implementierung einer GUI
		Implementierung der Schachregeln und der grundlegenden Spiellogik
		Einbindung von Spieler-gegen-Spieler-Funktionalitäten
Woche 5 (01.11.23)	Power-Up-Integration	Implementierung der Power-Ups (Einfrieren, Doppelzug, Teleportation etc.)
		Tests der Power-Up-Funktionalitäten
Woche 6 (08.11.23)	Feinschliff und Abschluss	UI-Design optimieren
		Abschließende Tests und Fehlerbehebung
		Benutzerdokumentation und Readme-Datei

Tabelle 1: Zeitplan und Meilensteine

Mit diesem Plan soll sicherstellen, dass das Projekt in sechs Wochen erfolgreich abgeschlossen werden kann. Regelmäßige Tests und Reviews sind fest eingeplant, um die Qualität des Spiels sicherzustellen.



INTERNATIONALE
HOCHSCHULE

Anforderungsdokument – DLMCSPSE01_D
Projekt: Software Engineering

Erstellt von: Dustin Lucht
Matrikelnummer: 92203340
Studiengang: Master of Science Informatik
Tutor: Markus Kleffmann
Datum: 17.12.2023

1 Management Summary

Dieses Projekt fokussiert sich auf die Entwicklung eines innovativen interaktiven Schachspiels, welches das klassische Schach durch die Integration von Power-Ups bereichert. Diese neuen Elemente ermöglichen es Spielern, unkonventionelle strategische Züge zu tätigen und verleihen dem Spiel eine einzigartige Dynamik.

Das Hauptziel ist es, ein ansprechendes und benutzerfreundliches Schachspiel zu schaffen, das sowohl traditionelle Schachliebhaber als auch Gelegenheitsspieler anspricht. Die Power-Ups sollen eine frische Herausforderung und Unterhaltung in das traditionelle Schachspiel bringen.

Nutzen und Vorteile

- Innovatives Spielerlebnis: Durch die Power-Ups wird eine neue Ebene der Taktik und Spannung eingeführt.
- Anpassbare Schwierigkeitsstufen: Die KI-Anpassung ermöglicht es Spielern aller Erfahrungsstufen, das Spiel zu genießen.
- Zwei Spielmodi: Ein Modus gegen den Computer (Stockfish) und ein Modus in welchen man gegen einen Freund spielen kann.

Zielgruppe

Das Spiel richtet sich an eine breite Zielgruppe – von Schachenthusiasten bis hin zu Gelegenheitsspielern, die nach einer neuen Herausforderung suchen.

Projektumfang

Das Projekt umfasst die Entwicklung der Spiellogik, die Integration der Power-Ups, das Design der Benutzeroberfläche und die Anpassung der KI.

Zeitplan und Meilensteine

- Anfangsphase (Woche 1-2): Konzeptentwicklung und Aufbau der Projektstruktur.
- Mittlere Phase (Woche 3-5): Implementierung der Spiellogik und Integration der Power-Ups.
- Endphase (Woche 6): Optimierung, Tests und Finalisierung.

Potenzielle Risiken und Herausforderungen

- Komplexität der Power-Up-Implementierung: Planung und regelmäßige Code-Reviews zur Risikominderung.
- Rechenleistung: Performance-Tests zur Optimierung.

- Balancing und Benutzererfahrung: Usability-Tests und Feedback-Einholung für ein ausgewogenes Spielerlebnis

2 Systemumfang und Kontext

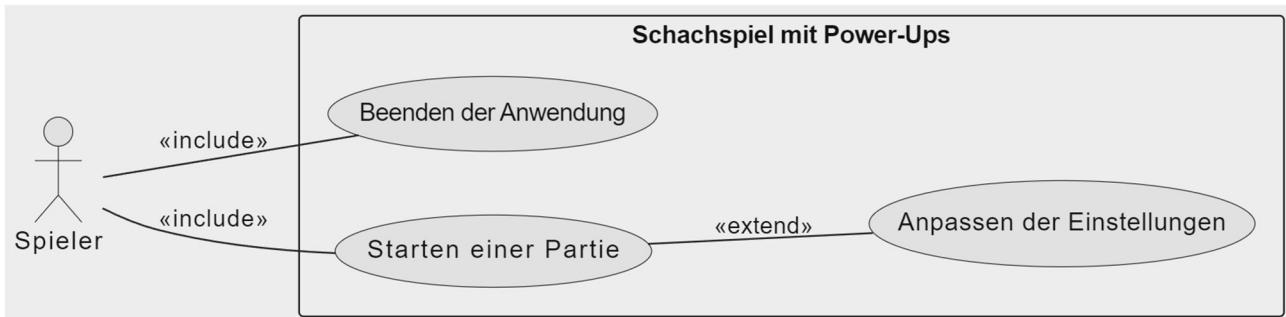


Abbildung 1: UML-Anwendungsfalldiagramm

3 Funktionale Anforderungen

Kennung:	1
Titel:	Neue Partie starten
Beschreibung:	Der Spieler möchte eine neue Partie gegen den Computer beginnen.
Aktoren:	Spieler
Vorbedingungen:	<ul style="list-style-type: none"> Die Anwendung ist gestartet und die Benutzeroberfläche ist sichtbar.
Nachbedingungen:	<ul style="list-style-type: none"> Eine neue Partie gegen den Computer hat begonnen, und der Spieler ist am Zug.
Ablauf:	<ol style="list-style-type: none"> Der Spieler wählt die Option "Start". Der Spieler gelangt ins „Pre-Game“ Menü. Der Spieler wählt „Spiele gegen Stockfish“ aus und drückt auf „Partie starten“. Das System generiert ein neues Schachbrett und platziert die Figuren gemäß den Schachregeln. Der Computer wird als Gegner ausgewählt und eine KI wird aktiviert. Der Spieler beginnt die Partie.

Tabelle 1: Use Case 1

Kennung:	2
Titel:	Neue Partie starten
Beschreibung:	Die beiden Spieler möchten eine neue Partie Gegeneinander beginnen.
Aktoren:	2 Spieler
Vorbedingungen:	<ul style="list-style-type: none"> Die Anwendung ist gestartet und die Benutzeroberfläche ist sichtbar.
Nachbedingungen:	<ul style="list-style-type: none"> Eine neue Partie für zwei Spieler hat begonnen, und der erste Spieler ist am Zug.
Ablauf:	<ol style="list-style-type: none"> Der erste Spieler wählt die Option "Neue Partie starten". Der erste Spieler gelangt ins „Pre-Game“ Menü. Der erste Spieler wählt „Spiel gegen einen Freund“ aus und drückt auf „Partie starten“. Das System generiert ein neues Schachbrett und platziert die Figuren gemäß den Schachregeln. Der erste Spieler beginnt die Partie.

Tabelle 2: Use Case 2

Kennung:	3
Titel:	Figuren bewegen und mögliche Züge anzeigen
Beschreibung:	Der Spieler möchte eine Schachfigur bewegen und benötigt eine Anzeige der möglichen Züge.
Aktoren:	Spieler
Vorbedingungen:	<ul style="list-style-type: none"> Der Spieler ist am Zug, und es wurden noch keine Züge getätigt.
Nachbedingungen:	<ul style="list-style-type: none"> Die ausgewählte Figur wurde bewegt, und die möglichen Züge wurden dem Spieler vor der Bewegung angezeigt.
Ablauf:	<ol style="list-style-type: none"> Der Spieler wählt eine Figur auf dem Schachbrett aus. Das System hebt alle möglichen Züge für die ausgewählte Figur hervor. Der Spieler wählt einen der möglichen Züge aus.

	4. Das System bewegt die Figur auf das entsprechende Feld und aktualisiert die Stellung auf dem Schachbrett.
--	--

Tabelle 3: Use Case 3

Kennung:	4
Titel:	Power-Up einsetzen
Beschreibung:	Der Spieler möchte ein Power-Up während des Spiels einsetzen, um einen strategischen Vorteil zu erlangen.
Aktoren:	Spieler
Vorbedingungen:	<ul style="list-style-type: none"> • Der Spieler ist am Zug. • Der Spieler hat mindestens ein Power-Up zur Verfügung.
Nachbedingungen:	<ul style="list-style-type: none"> • Das Power-Up wurde eingesetzt, und die Partie wird fortgesetzt.
Ablauf:	<ol style="list-style-type: none"> 1. Der Spieler wählt die Option "Power-Up einsetzen". 2. Das System zeigt dem Spieler die verfügbaren Power-Ups an. 3. Der Spieler wählt das gewünschte Power-Up aus. 4. Das System führt die entsprechende Aktion gemäß dem ausgewählten Power-Up aus.

Tabelle 4: Use Case 4

Kennung:	5
Titel:	Schachmatt erkennen
Beschreibung:	Das Spiel soll erkennen, ob eine der Könige schachmatt gesetzt wurde und die Partei somit vorbei ist.
Aktoren:	Spieler
Vorbedingungen:	<ul style="list-style-type: none"> • Es ist der Zug des Spielers. • Der Spieler hat einen möglichen Zug durchgeführt.
Nachbedingungen:	<ul style="list-style-type: none"> • Die Partie endet, und der entsprechende Spieler gewinnt.
Ablauf:	<ol style="list-style-type: none"> 1. Das Spiel erkennt, dass ein Spieler im Schachmatt steht und beendet das Spiel. 2. Es erscheint eine entsprechende Nachricht auf dem Bildschirm mit der Möglichkeit ins Hauptmenü zurückzukehren, oder die Partei erneut zu starten.

Tabelle 5: Use Case 5

Kennung:	6
Titel:	Unentschieden erkennen
Beschreibung:	Das Spiel soll erkennen, ob ein Unentschieden vorliegt.
Aktoren:	Spieler
Vorbedingungen:	<ul style="list-style-type: none"> • Es ist der Zug des Spielers. • Der Spieler hat einen möglichen Zug durchgeführt.
Nachbedingungen:	<ul style="list-style-type: none"> • Die Partie endet unentschieden.
Ablauf:	<ol style="list-style-type: none"> 1. Das Spiel erkennt, dass ein Unentschieden vorliegt (z.B. durch dreifache Stellungswiederholung oder 50-Züge-Regel). 2. Es erscheint eine entsprechende Nachricht auf dem Bildschirm mit der Möglichkeit ins Hauptmenü zurückzukehren, oder die Partei erneut zu starten.

Tabelle 6: Use Case 6

Kennung:	7
Titel:	Einstellungen ändern
Beschreibung:	Der Spieler möchte die Einstellungen des Spiels anpassen, um seine Präferenzen zu berücksichtigen.
Aktoren:	Spieler, Einstellungen
Vorbedingungen:	<ul style="list-style-type: none"> • Die Anwendung ist gestartet und die Benutzeroberfläche ist sichtbar.
Nachbedingungen:	<ul style="list-style-type: none"> • Die ausgewählten Einstellungen werden gespeichert und auf das laufende Spiel angewendet.
Ablauf:	<ol style="list-style-type: none"> 1. Der Spieler navigiert zu den Einstellungen vor den Beginn eines Spiels. 2. Das System zeigt dem Spieler verschiedene Optionen an, die angepasst werden können, wie z.B. die Schwierigkeit des Computers oder mit welcher Farbe man beginnen möchte. 3. Der Spieler wählt die gewünschten Einstellungen aus und kann eine Partie starten.

Tabelle 7: Use Case 7

Kennung:	8
-----------------	---

Titel:	Spiel beenden
Beschreibung:	Der Spieler soll das Spiel während eines Spiels und aus dem Hauptmenü heraus beenden können.
Aktoren:	Spieler
Vorbedingungen:	<ul style="list-style-type: none"> Die Anwendung ist gestartet, und entweder ist eine Partie aktiv, oder der Spieler befindet sich im Hauptmenü.
Nachbedingungen:	<ul style="list-style-type: none"> Das Spiel ist beendet, und die Anwendung ist geschlossen oder der Spieler ist zurück im Hauptmenü.
Ablauf:	<ol style="list-style-type: none"> Der Spieler wählt die Option „Spiel beenden“ aus dem Hauptmenü oder während eines laufenden Spiels aus. Wenn ein Spiel aktiv ist, fragt das System, ob der Spieler den aktuellen Fortschritt speichern möchte, bevor das Spiel beendet wird. Der Spieler bestätigt, dass das Spiel beendet und die Anwendung geschlossen werden soll, oder er wählt aus, zum Hauptmenü zurückzukehren. Das System führt die gewählte Aktion aus, beendet das Spiel und schließt die Anwendung oder kehrt zum Hauptmenü zurück.

Tabelle 8: Use Case 8

4 Nicht-funktionale Anforderungen

- Benutzeroberfläche
 - Die Benutzeroberfläche soll intuitiv und benutzerfreundlich gestaltet sein.
 - Die Anwendung soll auf verschiedenen Bildschirmgrößen und -auflösungen gut lesbar und bedienbar sein.
- Leistung
 - Das Spiel soll auf Windows-Plattformen flüssig und ohne spürbare Verzögerungen laufen (Ausnahme bildet ein hoch gesetzter Schwierigkeitsgrad des Computers).
 - Die Ladezeiten sollen minimal sein.
- Grafische Darstellung
 - Trotz fehlenden UI-Artists soll die Oberfläche ansprechend aussehen.
 - Die Figuren sollen eindeutig erkennbar und differenzierbar sein.
- Plattformkompatibilität
 - Die Anwendung soll auf den Windows-Versionen 10 und 11 reibungslos funktionieren.

5 Glossar

Begriff	Erklärung
Schach (Im Schach stehen)	Schach tritt ein, wenn der König bedroht ist, aber noch mindestens eine legale Fluchtmöglichkeit hat.
Schachmatt	Schachmatt tritt ein, wenn der König bedroht ist und keine legale Fluchtmöglichkeit hat, das Spiel ist dann für den bedrohten Spieler verloren.
Power-Ups	Power-Ups sind spezielle Fähigkeiten oder Gegenstände in einem Spiel, die dem Spieler vorübergehende Vorteile oder besondere Aktionen verleihen. Sie werden eingesetzt, um den Spielverlauf zu beeinflussen.

Tabelle 9: Glossar



INTERNATIONALE
HOCHSCHULE

Projektdokumentation – DLMCSPSE01_D
Projekt: Software Engineering

Erstellt von: Dustin Lucht
Matrikelnummer: 92203340
Studiengang: Master of Science Informatik
Tutor: Markus Kleffmann
Datum: 17.12.2023

Inhaltsverzeichnis

I.	Abbildungsverzeichnis	3
II.	Tabellenverzeichnis	3
1	Auswahl des Vorgehensmodells	4
2	Auswahl von zusätzlichen Tools	4
3	Klassendiagramm	5
3.1	Kurzerklärung der Klassen.....	7
3.1.1	Enums	7
3.1.2	Gamestates	7
3.1.3	Midgamestates	8
3.1.4	Midgame.....	8
4	Aktivitätsdiagramm.....	9
5	Zustand Entwurfsmuster	11
6	Komponentendiagramm.....	13

I. Abbildungsverzeichnis

Abbildung 1: UML-Klassendiagramm zu Beginn des Projektes	5
Abbildung 2: UML-Klassendiagramm am Ende des Projektes.....	6
Abbildung 7: UML-Aktivitätsdiagramm	10
Abbildung 4: Zustandsdiagramm.....	12
Abbildung 9: Komponentendiagramm	13

II. Tabellenverzeichnis

Tabelle 1: Beschreibung der Enum-Klassen	7
Tabelle 2: Beschreibung der Gamestate-Klassen	8
Tabelle 3: Beschreibung der Midgamestate-Klassen	8
Tabelle 4: Beschreibung der Midgame-Klassen	9

1 Auswahl des Vorgehensmodells

In diesem Schachspiel-Projekt, das von einer Person in sechs Wochen bewältigt werden soll, bietet sich ein modifiziertes Wasserfallmodell an, das aufgrund seiner klaren Struktur und sequenziellen Abfolge gut geeignet ist. Die Entscheidung für das V-Modell wurde getroffen, da es eine detaillierte Planung und schrittweise Umsetzung ermöglicht. Die klare Definition von Phasen, beginnend mit der Anforderungsdefinition über den Modellentwurf bis hin zu Tests auf verschiedenen Ebenen, erleichtert die Verfolgung des Fortschritts und die systematische Entwicklung. Die enge Kopplung von Entwicklung und Tests in einem diagonalen Verlauf des V-Modells ermöglicht zudem eine frühzeitige Identifizierung und Behebung von Fehlern.

2 Auswahl von zusätzlichen Tools

Um ein derartiges Programm in circa 6 Wochen programmieren zu können, werden verschiedene Tools benötigt, die einem dabei unterstützen. Teilweise vorgegeben ist dabei die Programmiersprache. Von der Liste ausgewählt wird für dieses Projekt die Programmiersprache Python, da bereits viele Erfahrungen mit dieser Sprache gemacht werden konnten. Gleiches gilt für PyCharm als IDE (Entwicklungsumgebung). Git bzw. GitHub ist vorgegeben und wird als Versionierungstool genutzt (https://github.com/DustinLucht/chess_with_powerups).

Die Entscheidung, PyGame als Spielframework zu verwenden, ergab sich aus seiner Benutzerfreundlichkeit und seiner Effizienz bei der Entwicklung von 2D-Spielen. Dieses Framework bietet eine Vielzahl von Funktionen und erleichtert die Umsetzung von Grafiken, Benutzeroberflächen und die Steuerung von Spielfluss. Zusätzlich wird die Python Chess-Bibliothek (<https://pypi.org/project/chess/>) und die Stockfish-Engine (<https://stockfishchess.org/about/>) in das Projekt integriert. Die Python Chess-Bibliothek erleichtert die Umsetzung der Schachlogik und ermöglicht es, die Regeln des Spiels zu implementieren. Sie bietet eine Vielzahl von Funktionen, die das Arbeiten mit Schachbrettern, Zügen und Partien erleichtern und ermöglicht eine direkte Integration der Stockfish-Schachengine. Dies ermöglicht die Implementierung eines intelligenten Computergegners für den Einzelspielermodus. Stockfish ist bekannt für seine hohe Spielstärke und wird oft in Schachsoftware und -plattformen eingesetzt.

Die Kombination dieser Tools bietet eine solide Basis, um das Schachspiel mit Power-Ups in der vorgegebenen Zeit erfolgreich zu entwickeln. Python und die genannten Bibliotheken ermöglichen eine effiziente Umsetzung der Spiellogik und bieten die notwendige Flexibilität, um Power-Ups, Benutzeroberfläche und Spielablauf zu gestalten.

3 Klassendiagramm

Zu Beginn des Projektes wurden folgendes Klassendiagramm für eine Grundlage einer sauberen Programmierung erstellt.

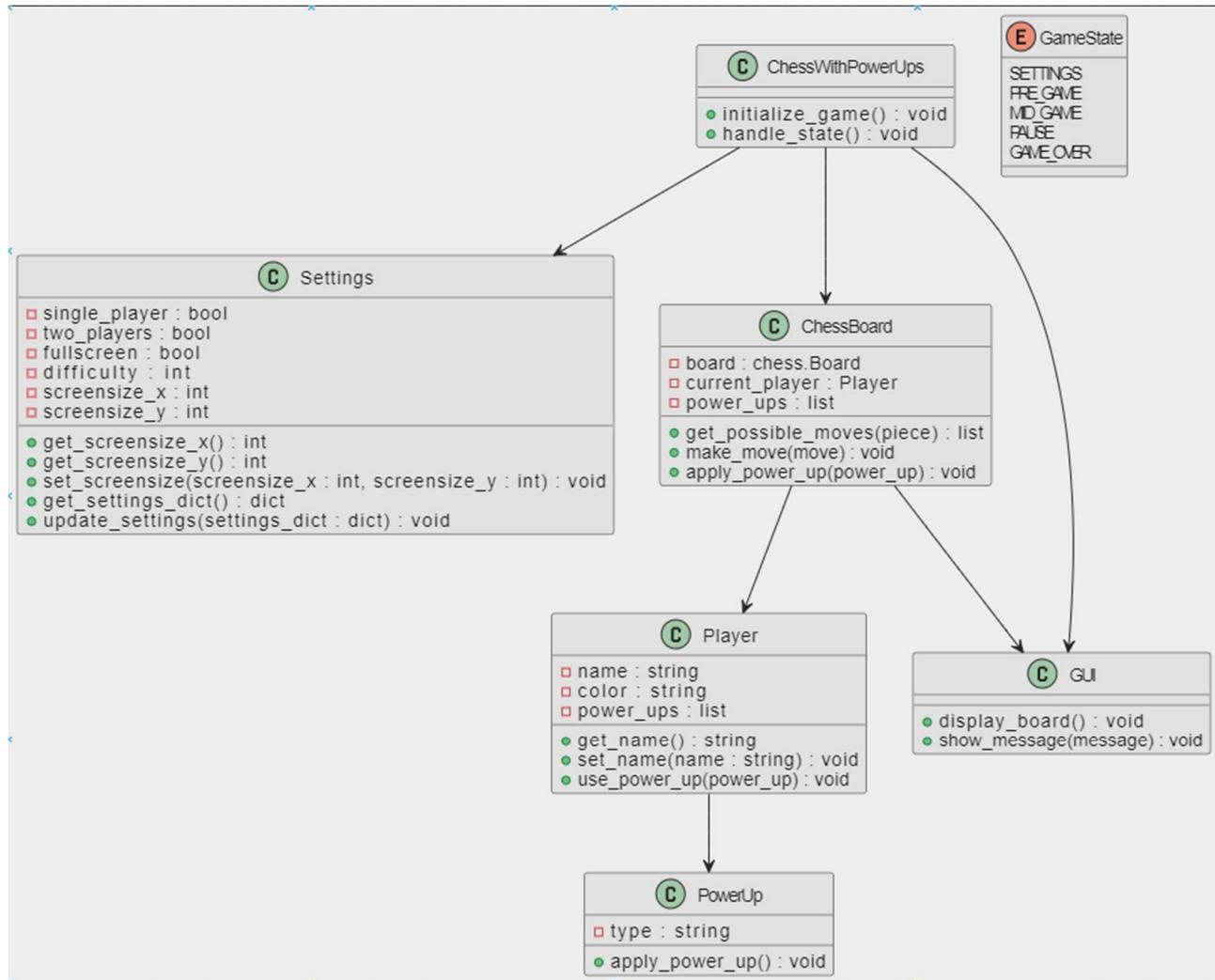


Abbildung 1: UML-Klassendiagramm zu Beginn des Projektes

Aufgrund der mangelnden Erfahrung in der Spieleentwicklung und der Entwicklung mit der Bibliothek PyGame, war der erste Entwurf des Klassendiagramms nicht ausgereift. Im Laufe der Programmierung kam es zu Erkenntnissen und neuen Erfahrungen, welche die Struktur des Codes und den Aufbau der Klassen wie folgt geändert haben:

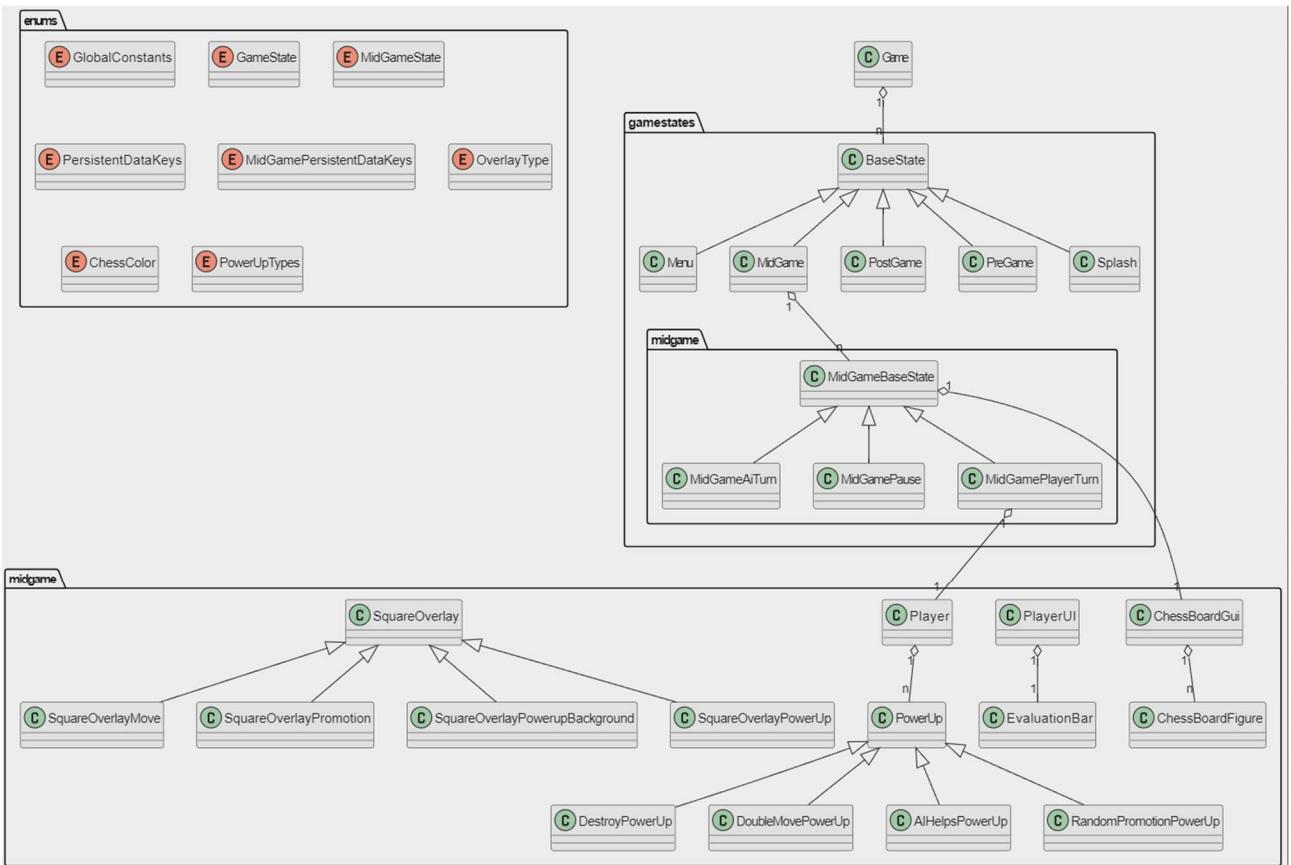


Abbildung 2: UML-Klassendiagramm am Ende des Projektes

Zum einen sind deutlich mehr Klassen während der Entwicklung ergänzt wurden. Das liegt vor allem daran, dass die Größe des Projekts und die Menge des nötigen Codes unterschätzt wurde. Zum anderen wurde die gesamte Codestruktur in vier einzelne Gruppen (Namensräume oder auch „namespaces“) unterteilt, um einen besseren Überblick zu gewährleisten.

Außen vor sind die Enum-Klassen, welche genutzt wurden, um mithilfe der Codevervollständigung arbeiten zu können. Zudem helfen diese dabei Schreibfehler zu vermeiden.

Die Spielinstanz befindet sich in der Klasse *Game* und ist als Startpunkt des gesamten Spiels außerhalb der einzelnen Namensräume anzufinden.

Die Klassen, die genutzt wurden, um die einzelnen Spielstände zu steuern, befinden sich in dem Namensraum *gamestates*. Ebenfalls darin enthalten ist der Namensraum *midgame*, welcher die Klasse *MidGame* weiter unterteilt, um besser die einzelnen Spielstände während des eigentlichen Schachspiels steuern zu können.

Der gleichnamige Namensraum, welcher sich außerhalb des *Gamestates*-Namensraum befindet, beinhaltet alle Klassen, die für die Gamelogik, sowie GUI während des Spiels benötigt wird.

3.1 Kurzerklärung der Klassen

Nachfolgend werden alle Klassen in den einzelnen Namensräumen und ihre Funktion kurz beschrieben. Die Game Klasse bildet das Herzstück des Spiels, steuert den Spielablauf und verwaltet die verschiedenen Spielzustände.

3.1.1 Enums

Enum-Klassen (oder Aufzählungsklassen) sind spezielle Datentypen, die vordefinierte Werte enthalten. Sie verbessern die Code-Lesbarkeit und helfen, Fehler zu vermeiden, indem sie klare, benannte Konstanten für die Programmierung bereitstellen.

Klassenname	Beschreibung
GlobalConstants	Beinhaltet Werte, welche im gesamten Projekt genutzt werden. Dazu gehören die gesetzte Bildschirmgröße und die Größen der Figuren. Diese Variablen wurden global festgelegt, um sie schnell ändern zu können, ohne dabei alle Stellen der Nutzung im Code zu suchen.
GameState	Beinhaltet die einzelnen Namen der Spielzustände („SPLASH“, „MENU“, „PRE_GAME“, „MID_GAME“, „POST_GAME“)
MidGameState	Beinhaltet die einzelnen Namen der Spielzustände innerhalb des eigentlichen Schachspiels („TURN_PLAYER_1“, „PAUSE“, „TURN_PLAYER_2“).
PersistentDataKeys	Beinhaltet die Schlüssel für alle persistenten Daten, welche Spielstands übergreifend benötigt werden.
MidGamePersistentDataKeys	Beinhaltet die Schlüssel für alle persistenten Daten, welche innerhalb des <i>MidGames</i> Spielstands übergreifend benötigt werden.
OverlayType	Beinhaltet alle verschiedenen GUI-Overlays, die im Spiel benutzt werden, wodurch sich die einzelnen Overlays gut unterscheiden lassen.
ChessColor	Beinhaltet die Werte „BLACK“ und „WHITE“.
PowerUpTypes	Beinhaltet alle verschiedenen Typen von Power-Ups, wodurch sich die einzelnen Power-Ups gut unterscheiden lassen.

Tabelle 1: Beschreibung der Enum-Klassen

3.1.2 Gamestates

Die Gamestate-Klassen steuern die Logik während einzelner Spielzustände. Dies ermöglicht eine effiziente Organisation und Handhabung verschiedener Spielphasen, indem sie spezifische Aktionen und Ereignisse für jeden Zustand verwalten, wie zum Beispiel Menüs, Spielverlauf oder Endbildschirme.

Klassenname	Beschreibung
BaseState	Beinhaltet Werte und Funktionen, welche die anderen GameState-Klassen erben. Beispiele für solche Funktionen sind drei bekannte Methoden aus der Spieleentwicklung: „get_event()“, „update()“, „draw()“. Dadurch kann die Game Klasse in der Hauptschleife bei allen Spielständen diese wichtigen Funktionen aufrufen. Eine weitere Funktion ist die „start_up()“-Funktion. Diese wird immer bei dem Übergang von dem einen zum anderen Spielstand aufgerufen und ermöglicht es, je nach Bedarf, auf Ereignisse von dem vorherigen Spielstand zu reagieren.

Splash	Beinhaltet die Logik, um ein kleines Intro zu Beginn des Spiels zu zeigen. Außerdem ist dies der erste Spielstand des Spiels.
Menu	Nach dem Splash-Bildschirm bekommt der Spieler das Spielmenu angezeigt. Sämtliche Logik für das Menu, befindet sich in dieser Klasse.
PreGame	Beinhaltet die Logik für das Anzeigen des Spielmenüs vor dem Spiel. Der Spieler kann zum Beispiel festlegen, ob er gegen Stockfish oder einen Freund spielen möchte.
MidGame	Beinhaltet das eigentliche Schachspiel. Zudem wird ähnlich wie in der Game-Klasse ein weiterer GameStateHandler genutzt, um die Spielstände während des Spiels weiter zu unterteilen und zu organisieren.
PostGame	Beinhaltet die Logik zum Anzeigen eines kleinen Menüs am Ende des Spiels. Angezeigt wird zudem welcher Spieler gewonnen hat.

Tabelle 2: Beschreibung der Gamestate-Klassen

3.1.3 Midgamestates

Klassenname	Beschreibung
MidBaseState	Parallel zu der BaseState-Klasse mit kleinen Ergänzungen: Speicher des Spielbretts und anderen Schachspiel-bezogenen Daten in Variablen, um den Zugriff während jedes MidGameStates zu gewährleisten.
MidGamePause	Ein Spielstand, welcher ein kleines Menu über dem aktuellen Brett anzeigt und dem Spieler die Möglichkeit gibt das Spiel fortzusetzen, zu beenden, oder neu zu starten.
MidGamePlayerTurn	Beinhaltet Logik, um dem Spieler das Teilhaben am Schachspiel zu ermöglichen. Abseits des normalen Inputs werden in dieser Klasse auf einer höherer Abstraktionsebene auch Sonderereignisse wie eine Bauernbeförderung, oder das Einsetzen von Power-Ups bewerkstelligt.
MidGameAiTurn	Beinhaltet eine Anbindung an die Stockfish Anwendung und verarbeitet die Kommunikation mit dieser.

Tabelle 3: Beschreibung der Midgamestate-Klassen

3.1.4 Midgame

In den „midgame“-Namensraum liegen Klassen, die in den MidGameStates benötigt werden. Weiter unterteilbar sind diese Klassen in die Bereiche „Overlay“, „Spieler bezogen“ und „UI“.

Klassenname	Beschreibung
SquareOverlay	Die Basisklasse für alle Overlays. Beinhaltet Variablen, die alle Overlays aufweisen sollten (draw-Funktion etc.)
SquareOverlayMove	Overlay Klasse für die Feldmarkierungen der möglichen Züge, die ein Spieler machen kann, wenn dieser eine bestimmte Figur auswählt.
SquareOverlayPromotion	Das Overlay, welches den Spieler bei einer Bauernbeförderung die Art der Beförderung auswählen lässt.
SquareOverlayPowerUp	Das Overlay für die Anzeige der zur Verfügung stehenden Power-Ups des Spielers.
SquareOverlayPowerupBackground	Das Overlay für die vier Felder der Power-Ups, falls diese leer sind.
Player	Diese Klasse enthält grundlegende Infos über einen aktiven Spieler: Name, Farbe (schwarz, weiß), Liste von verfügbaren Power-Ups.

PowerUp	Stellt die Basis Klasse für alle Power-Ups dar.
DestroyPowerUp	Power-Up welches eine zufällige gegnerische Figur zerstört (außer den König).
DoubleMovePowerUp	Erlaubt den Spieler 2 Züge hintereinander zu machen.
AIHelpsPowerUp	Zeigt dem Spieler einen Zug, welcher Stockfish nach mehreren Sekunden Rechenzeit wählen würde.
RandomPromotionPowerUp	Ein zufälliger Bauer bekommt eine zufällige Beförderung
PlayerUI	Beinhaltet die Logik zum Anzeigen der rechten Bildschirmhälfte (Unentschieden anbieten, Anzeige der Power-Ups und der Bewertungsleiste)
EvaluationBar	Die Bewertungsleiste, die das aktuelle Spiel bewertet und zeigt welcher Spieler gerade die bessere Position hat.
ChessBoardGui	Beinhaltet sämtliche Logik zum korrekten Anzeigen des Spielbretts inklusive der Figuren und Overlays.
ChessBoardFigure	Beinhaltet die einzelnen Spielfiguren und ihre Sprites (Bilder).

Tabelle 4: Beschreibung der Midgame-Klassen

4 Aktivitätsdiagramm

In der Klasse „MidGamePlayersTurn.py“ wird unter Anderem der Input des Spielers überprüft und dementsprechend verschiedene Aktionen ausgeführt. Zu den Aktionen gehört das Auswählen einer Spielfigur, das Auswählen einer Figur für die Bauernumwandlung, oder das Ausführen eines Zuges. Der Spieler kann außerdem durch längeres gedrückt halten eine Spielfigur bewegen, um so einen Zug auszuführen. Um die einzelnen Inputs den entsprechenden Aktionen zuzuordnen existieren im Code einige Abfragen, welche durch das folgende Aktivitätsdiagramm dargestellt werden können:

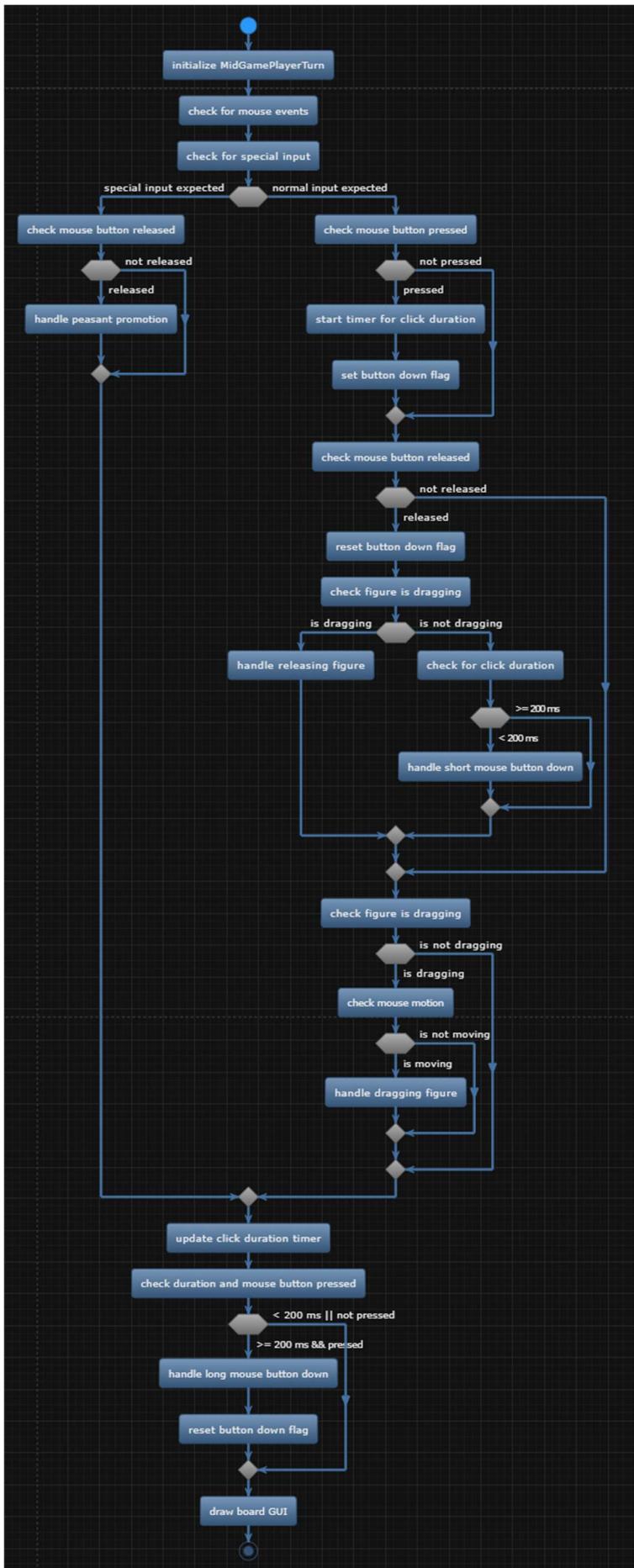


Abbildung 3: UML-Aktivitätsdiagramm

Das Ablaufdiagramm beginnt bei der Initialisierung der Klasse *MidGamePlayerTurn* und dem anschließenden wiederholten Überprüfen, ob ein neues Mouse-Event existiert. Von dort aus gibt es zwei unterschiedliche Spielstände. In dem Fall, dass der Spieler gerade dabei ist eine Beförderung eines Bauern durchzuführen, sollen alle anderen Inputs, die sich auf das normale Spiel auswirken ignoriert werden. Aus diesem Grund wird überprüft, ob diese Art vom speziellen Input von dem Spieler erwartet wird, oder nicht. Handelt es sich um diesen speziellen Input, sprich eine Beförderung eines Bauern, so wird überprüft, ob die linke Taste der Maus losgelassen wurde. Falls dies der Fall ist, wird die Funktion „handle_peasant_promotion“ aufgerufen.

Bei dem Strang für den „normalen“ Input finden alle Überprüfungen statt um die Figuren auf dem Brett zu bewegen. Um weiterführend zu unterscheiden, ob der Spieler eine Figur kurz antippt, oder diese durch längeres Gedrückt halten der Maustaste zu einem anderen Feld ziehen möchte, wird zuerst bei einem Drücken der Maustaste ein Timer zurückgesetzt und die Information gespeichert, dass die Maustaste aktuell gedrückt wird. Bei einem späteren Loslassen der Maustaste wird die Information wieder zurückgesetzt und überprüft, ob eine Figur aktuell bereits mit der Maus gezogen wird. Ist dies der Fall, wird die Funktion „handle_releasing_figure“ aufgerufen. Ist dies nicht der Fall, wird überprüft, ob der Spieler stattdessen eine Figur nur kurz angeklickt hat. Dabei wird der Timer genutzt und überprüft, ob dieser weniger als 200 Millisekunden aktiv war. Kann dies bejaht werden, wird die Funktion „handle_short_mouse_button_down“ aufgerufen.

Im nächsten Abschnitt befindet sich die Überprüfung der Mausbewegung. Wird diese bewegt und eine Figur aktiv gezogen, muss die Funktion „handle_dragging_figure“ aufgerufen werden.

Am Ende der größeren Abzweigung wird, der die Zeit des Timers aktualisiert und falls diese größer als 200 Millisekunden ist, die Funktion „handle_long_mouse_button_down“ aufgerufen und die Information, über das lange Drücken der Maus, gespeichert. Das Diagramm endet mit dem Zeichnen des Spielfelds und der Figuren.

5 Zustand Entwurfsmuster

Für eine bessere Struktur und Übersicht beim Programmieren und eine klare Trennung von Logik wurde das Spiel zustandsbasierend programmiert. Beispiele für diese Spielzustände bei Spielen können die folgenden sein: MainMenuState, PlayState, PauseState, GameOverState. In diesem Projekt können die verschiedenen States wie folgt dargestellt werden:

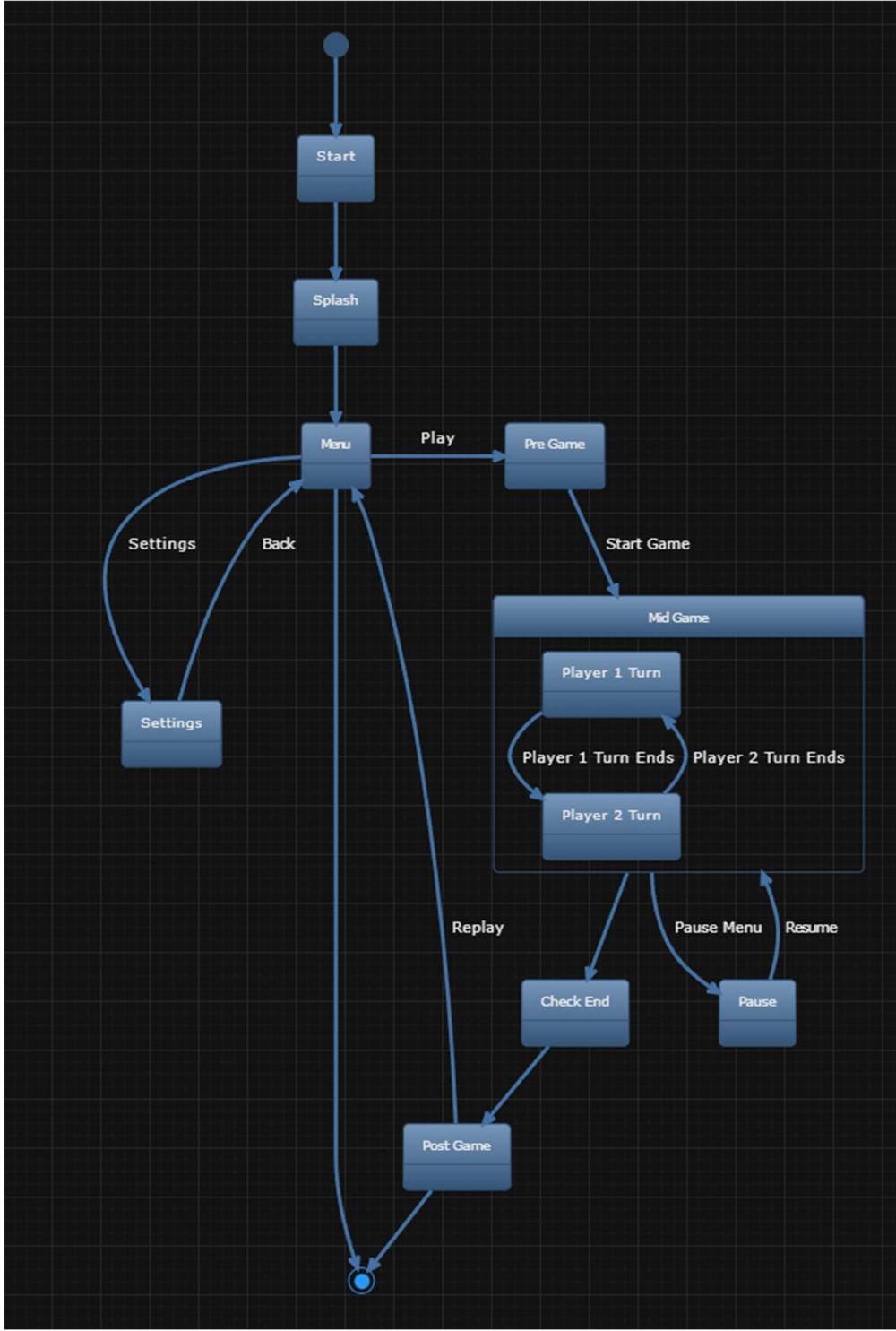


Abbildung 4: Zustandsdiagramm

Der Zustand *Mid Game* enthält eine weitere Zustandsmaschine. Grund für diese Entscheidung einer zweiten Zustandsmaschine ist die erhöhte Flexibilität: Es können weitere Zustände innerhalb von *Mid Game* hinzugefügt werden, ohne dabei die Logik außerhalb anzupassen. Des Weiteren besitzen alle Zustandsklassen ein Dictionary mit persistenten Daten, welche zwischen den einzelnen Zuständen übertragen werden können. Dabei benötigen die Zustandsklassen innerhalb der eigentlichen Spielausführung andere Informationen als die außerhalb. Die zweite Zustandsmaschine unterstützt dabei die Trennung dieser persistenten Daten.

6 Komponentendiagramm

Jegliche Komponenten des Codes befinden sich in dem Ordner `src`. In diesem Ordner befinden sich drei globale Dateien:

- Die `enum.py` definiert zentral Aufzählungen, welche in allen anderen Klassen genutzt werden können.
- Die `game.py` startet das Spiel und enthält die Haupt-Spielschleife.
- Die `main.py` ist der Startpunkt des Programms, initialisiert die Game Klasse und startet die Hauptschleife

Alle Zustandsklassen befinden sich unter `gamestates`. Innerhalb dieser Komponenten befindet sich die Komponente `mid_game_gamestates`, welche die Zustandsklassen des eigentlichen Spiels enthalten. Unter `mid_game` befinden sich die Klassen, die für die Hauptlogik des Schachspiels benötigt werden. Diese Komponenten sind im folgenden Komponentendiagramm dargestellt:

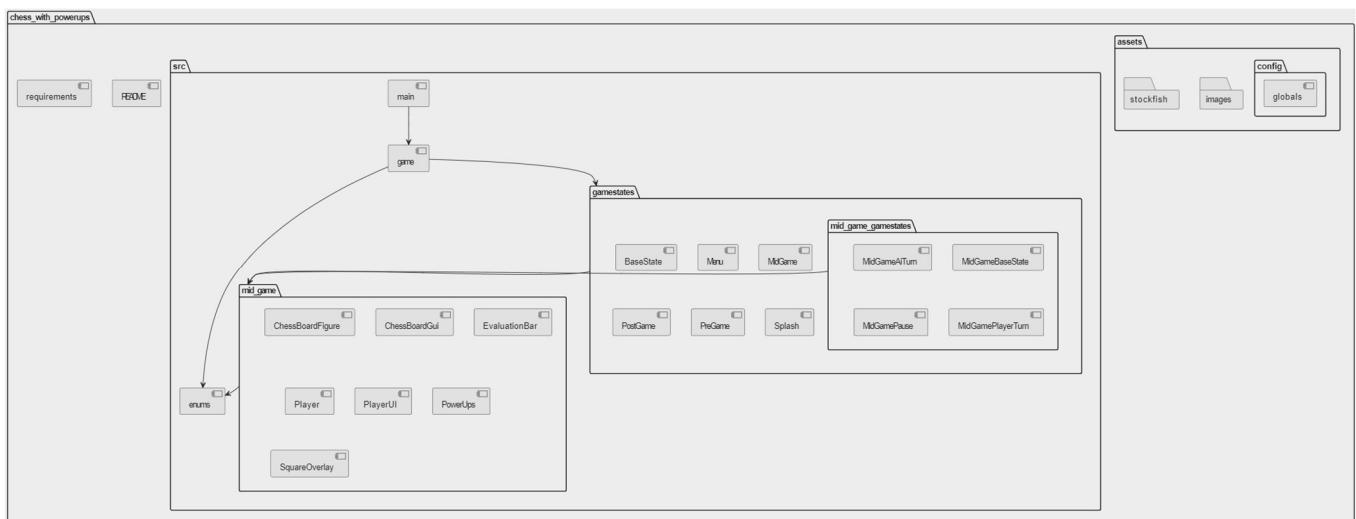


Abbildung 5: Komponentendiagramm

4 Benutzeranleitung (Windows)

Um das Spiel starten zu können bedarf es 2 Schritte und 2 Voraussetzungen.

Voraussetzungen:

- Python ist in der Version 3.11 oder höher installiert (Es sollten auch ältere Versionen, bis zur Version 3.6, funktionieren, getestet wurde aber mit Python 3.11.5).
- Git ist auf dem Rechner installiert.

Schritte:

- Klonen des Git Repos mit dem Befehl (Git Bash) „git clone https://github.com/DustinLucht/chess_with_powerups.git“.
- Starten der „`chess_with_powerups/start.bat`“



INTERNATIONALE
HOCHSCHULE

Testdokument – DLMCSPSE01_D
Projekt: Software Engineering

Erstellt von: Dustin Lucht
Matrikelnummer: 92203340
Studiengang: Master of Science Informatik
Tutor: Markus Kleffmann
Datum: 17.12.2023

1 Dokumentation der Tests

Nachfolgend werden die einzelnen Tests der Anwendung beschrieben und dokumentiert. Sind einzelne Schritte bereits in vorherigen Tests vorhanden, werden diese nicht wiederholt. Stattdessen beginnt der Test bei dem ersten neuen Teilschritt.

1.1 Testfall 1: Neue Partie gegen den Computer starten (Use Case 1)

Testziel:

Überprüfen, ob der Spieler eine neue Partie gegen den Computer starten kann.

Schritte:

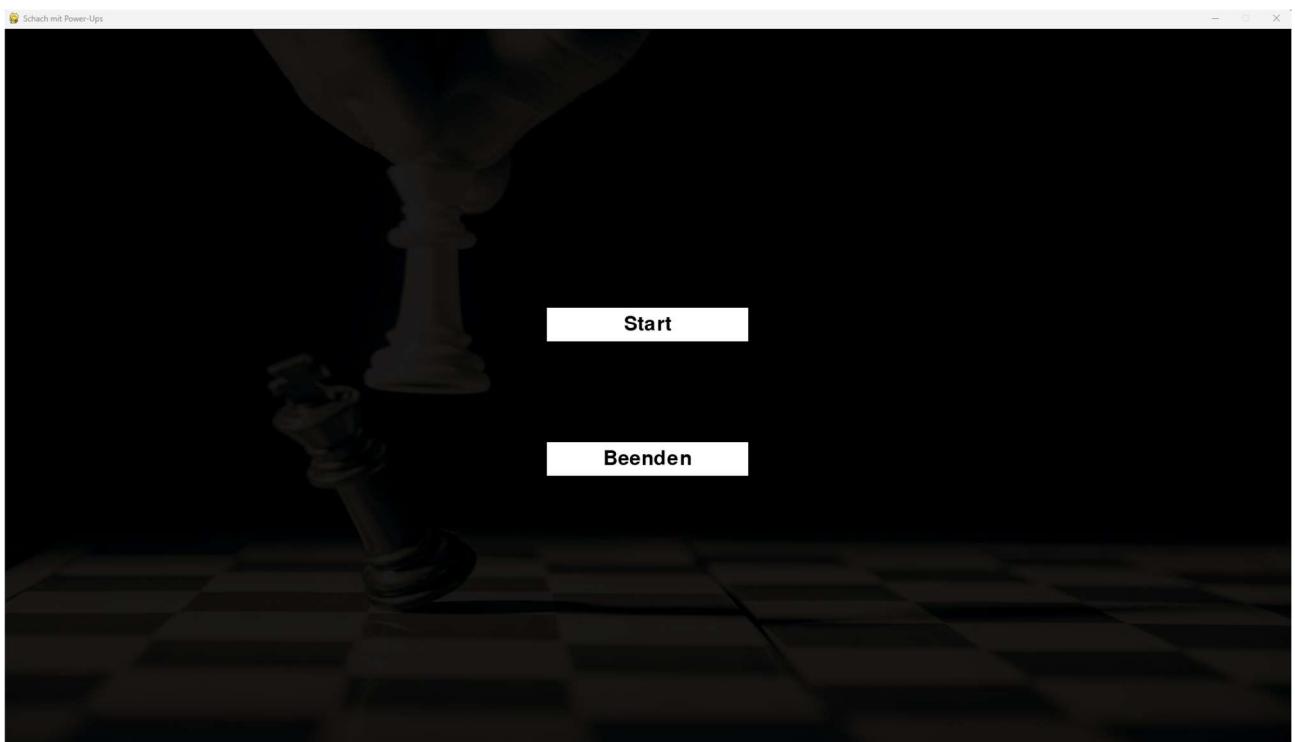
- Starte die Anwendung.
- Wähle die Option "Start".
- Im "Pre-Game" Menü wähle "Spiele gegen Stockfish" aus und drücke auf "Partie starten".

Erwartetes Ergebnis:

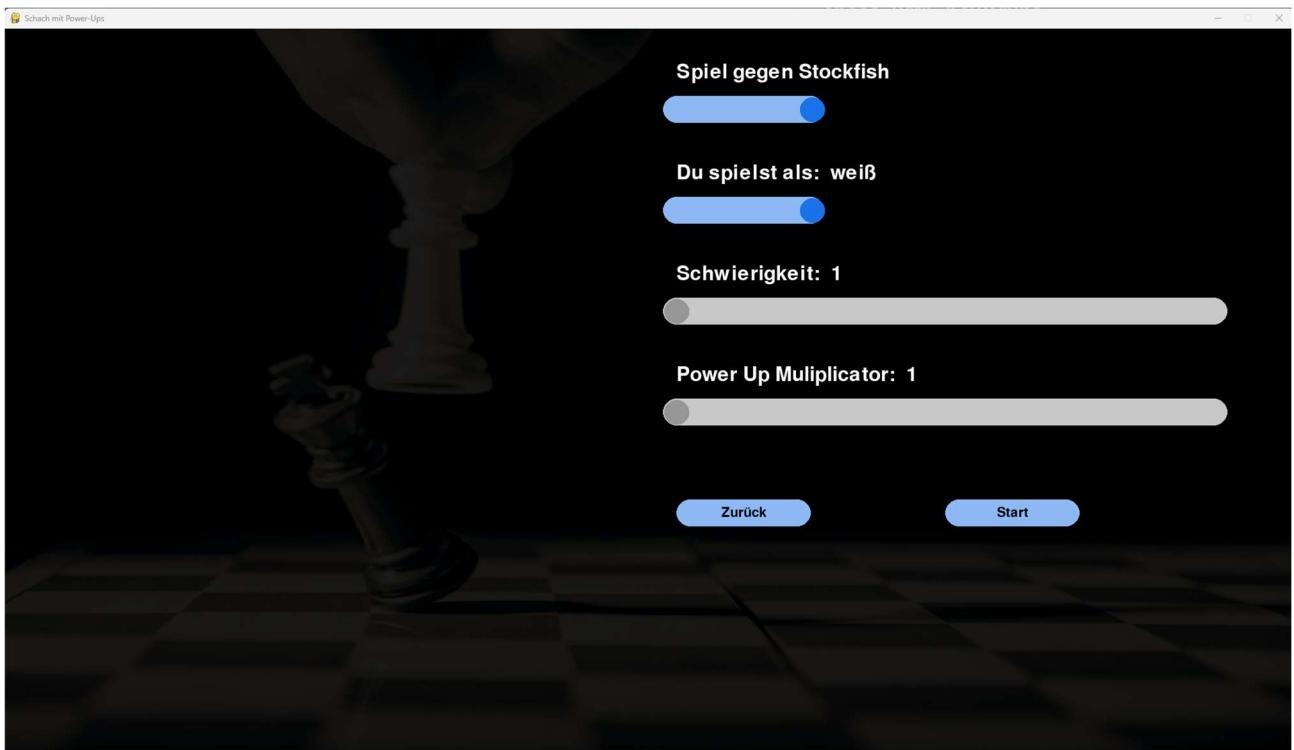
Eine neue Partie gegen den Computer hat begonnen, und der Spieler ist am Zug.

Testdurchführung:

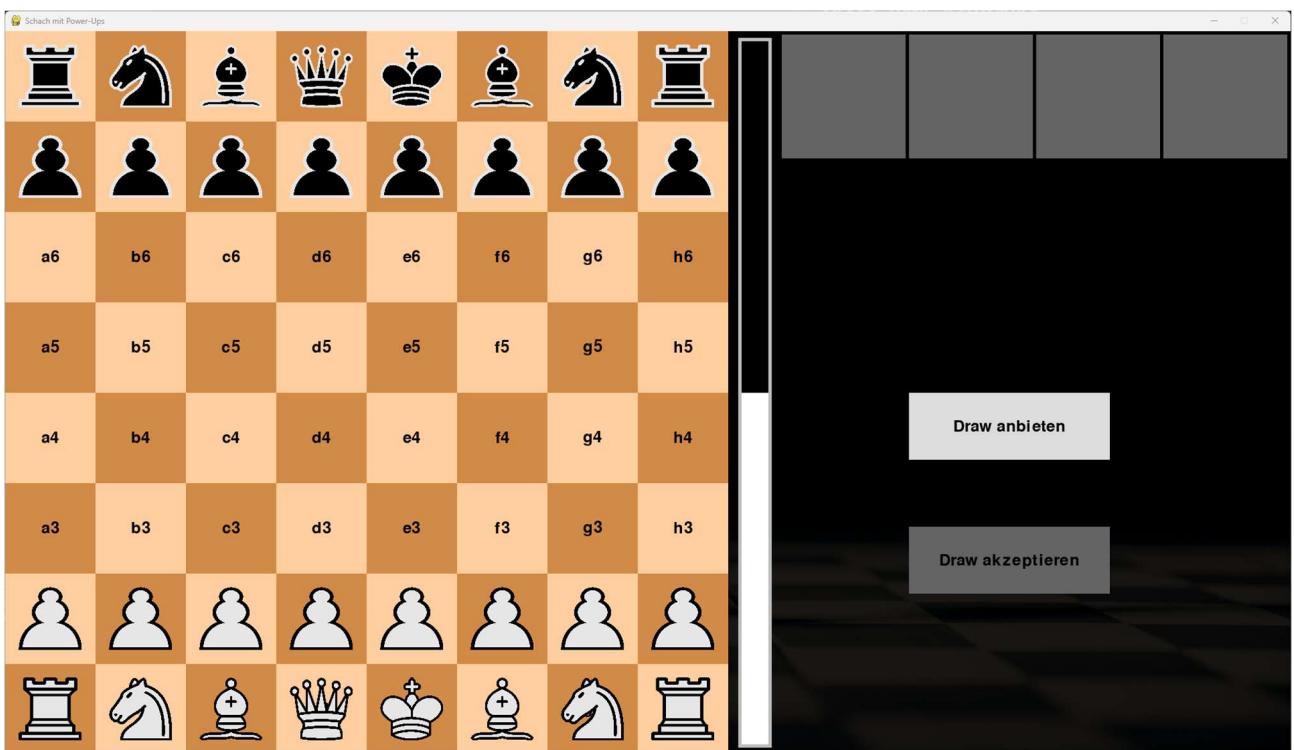
Mithilfe der „.README“ können alle nötigen Schritte erledigt werden um die Anwendung erfolgreich zu starten. Nach dem Start der Anwendung erscheint, nach einem kurzen Splash-Screen, das Hauptmenü.



Drückt man auf „Start“ erreicht man das Einstellungsmenü einer Partie.



Drückt man erneut auf Start beginnt eine neue Partie Schach.



1.2 Testfall 2: Neue Partie für zwei Spieler starten (Use Case 2)

Testziel:

Überprüfen, ob zwei Spieler eine neue Partie gegeneinander starten können.

Schritte:

- Starte die Anwendung.

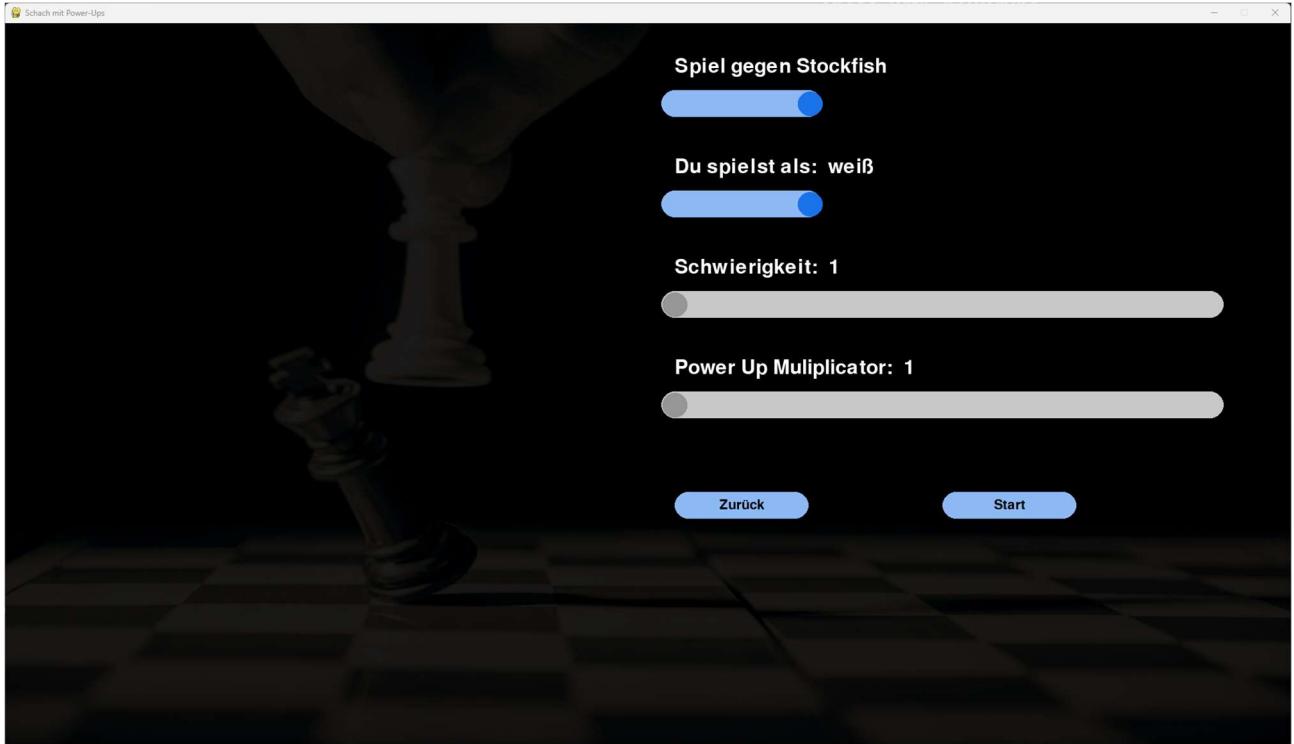
- Der erste Spieler wählt die Option "Neue Partie starten".
- Im "Pre-Game" Menü wählt der erste Spieler "Spiel gegen einen Freund" aus und drückt auf "Partie starten".

Erwartetes Ergebnis:

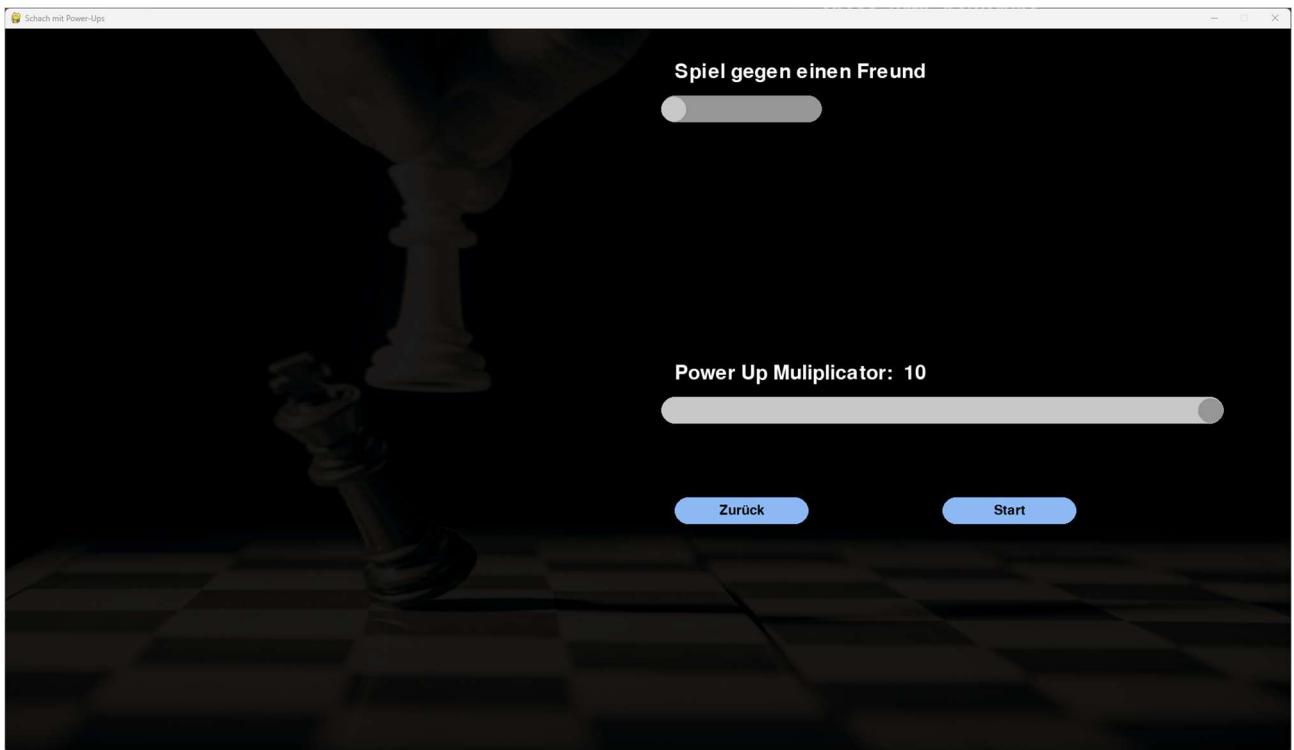
Eine neue Partie für zwei Spieler hat begonnen, und der erste Spieler ist am Zug.

Testdurchführung:

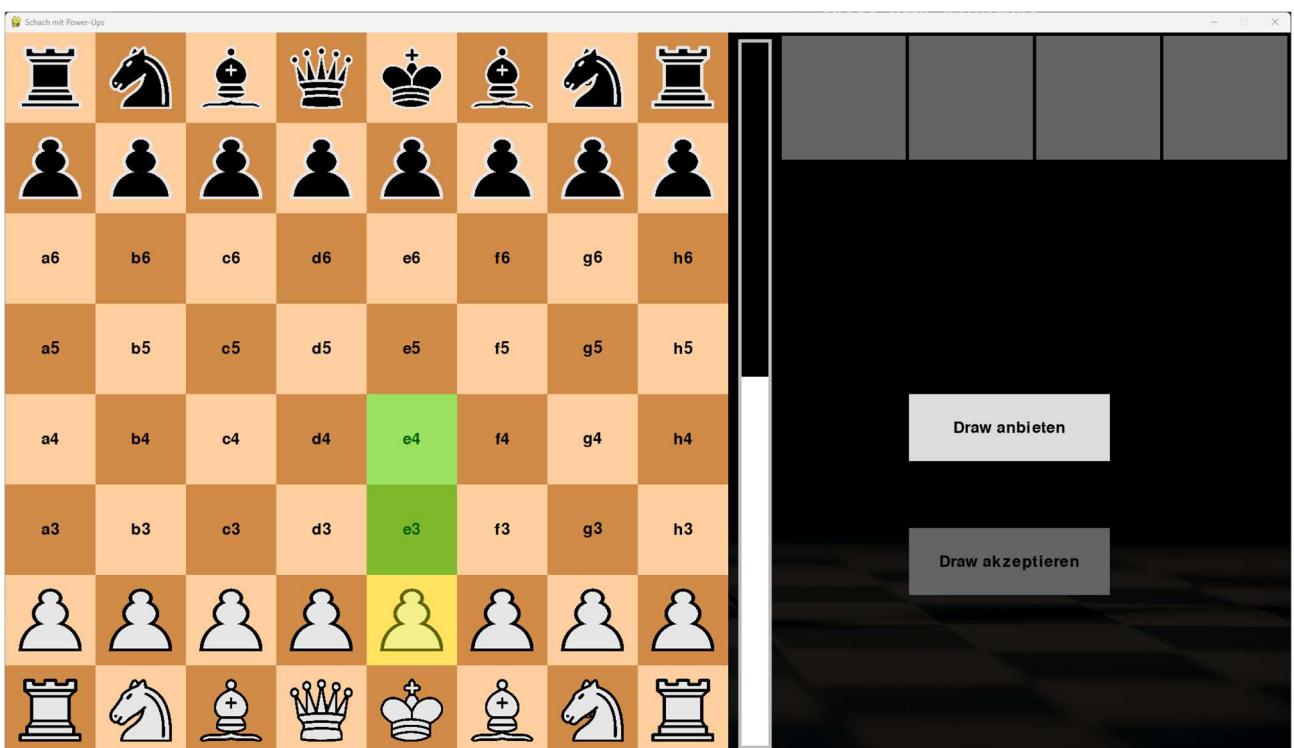
Beginn des Tests ist das Einstellungsmenü einer neuen Partie aus dem vorherigen Test.



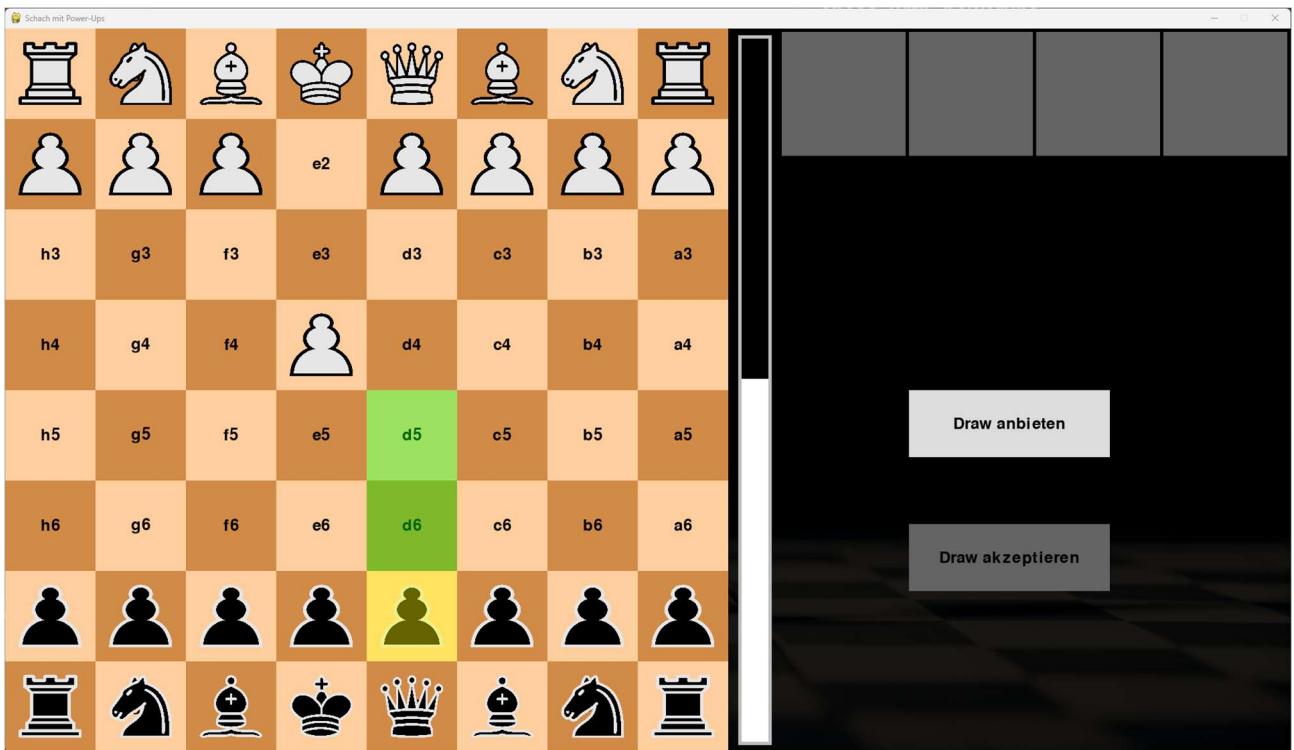
Mit einem Klick auf den Slider „Spiele gegen Stockfish“, wechselt der Slider und die Einstellung „Spiele gegen einen Freund“ ist gesetzt.



Nach dem Start der Partie beginnt man weiß.



Nach dem ersten Zug kann der zweite Spieler einen Zug mit Schwarz tätigen.



1.3 Testfall 3: Figuren bewegen und mögliche Züge anzeigen (Use Case 3)

Testziel:

Überprüfen, ob der Spieler eine Schachfigur bewegen kann und die möglichen Züge angezeigt werden.

Schritte:

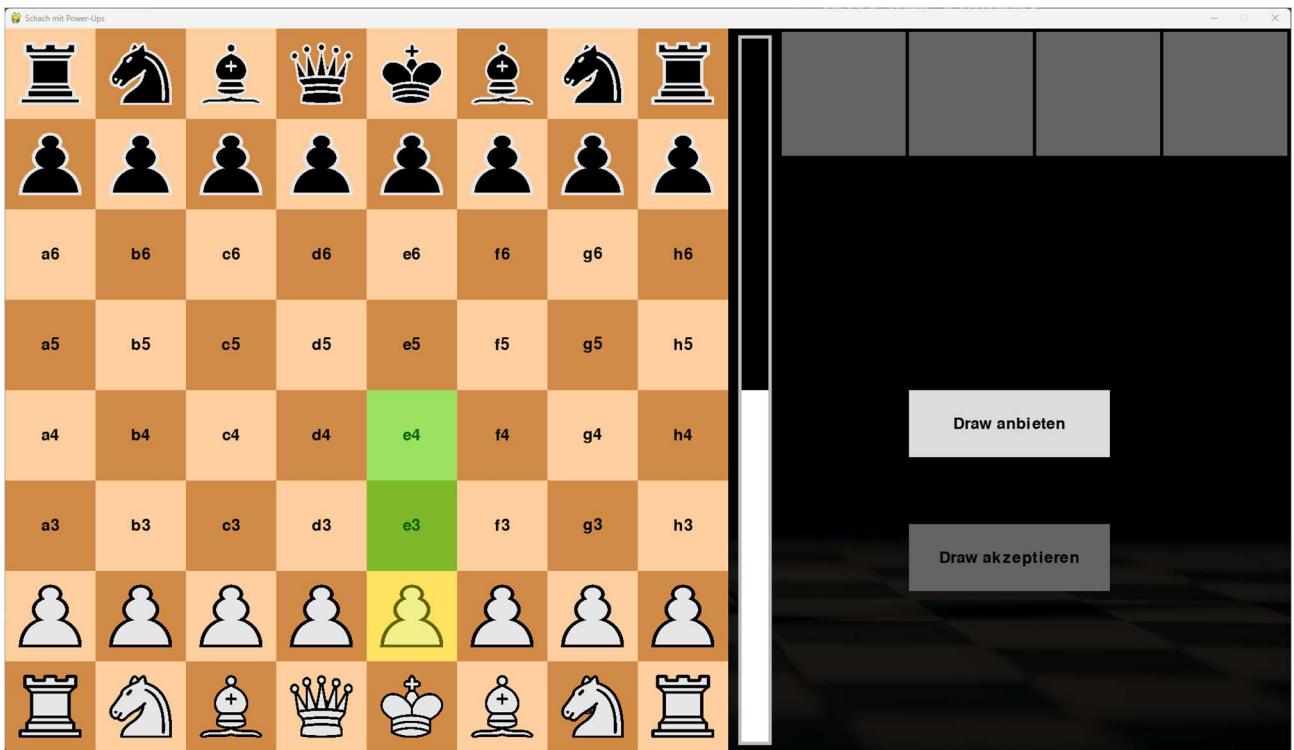
- Starte eine neue Partie.
- Der Spieler wählt eine Figur auf dem Schachbrett aus.
- Überprüfe, ob alle möglichen Züge für die ausgewählte Figur hervorgehoben werden.
- Der Spieler wählt einen der möglichen Züge aus.

Erwartetes Ergebnis:

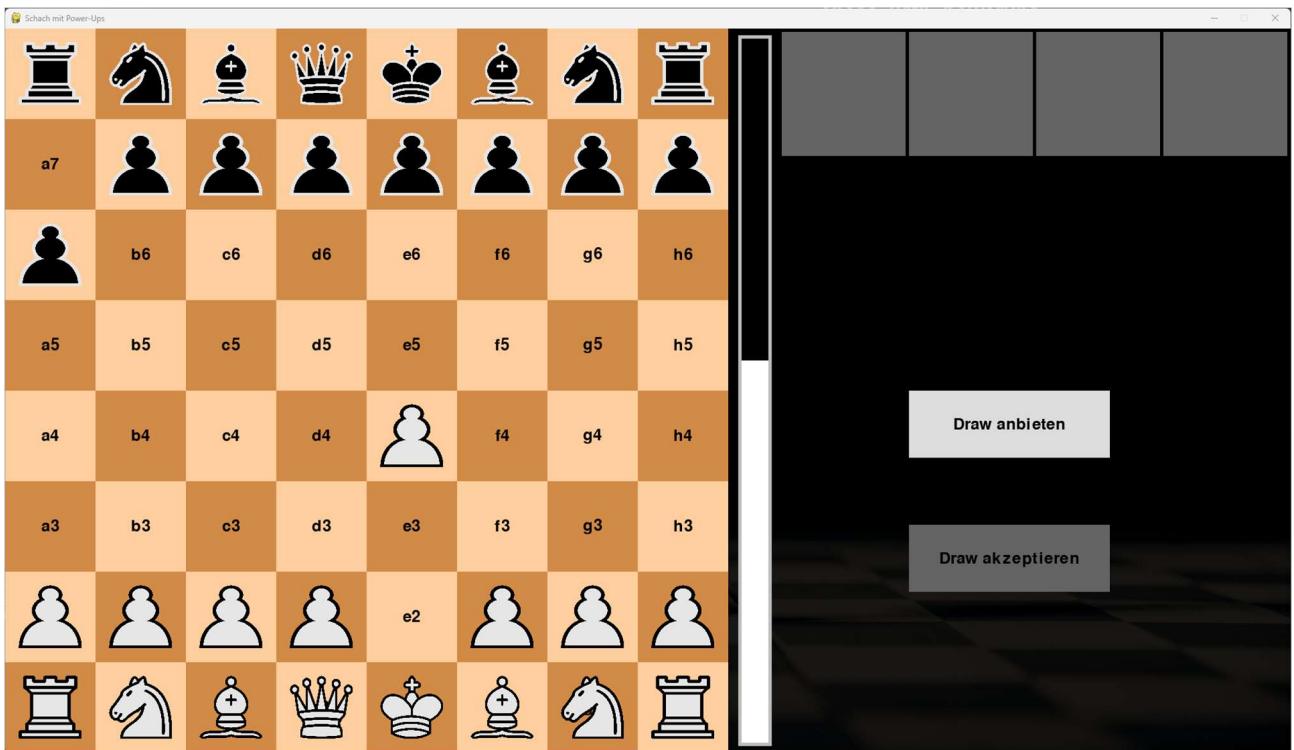
Die ausgewählte Figur wurde bewegt, und die möglichen Züge wurden dem Spieler vor der Bewegung angezeigt.

Testdurchführung:

Während einer Partie kann man eine beliebige Figur der eigenen Farbe anklicken.



Die Anwendung zeigt erfolgreich alle möglichen Züge an. Mit einem Klick auf eines der möglichen Felder, wird die Figur dorthin gesetzt und der nächste Spieler ist am Zug.



1.4 Testfall 4: Power-Up einsetzen (Use Case 4)

Testziel:

Überprüfen, ob der Spieler ein Power-Up während des Spiels einsetzen kann.

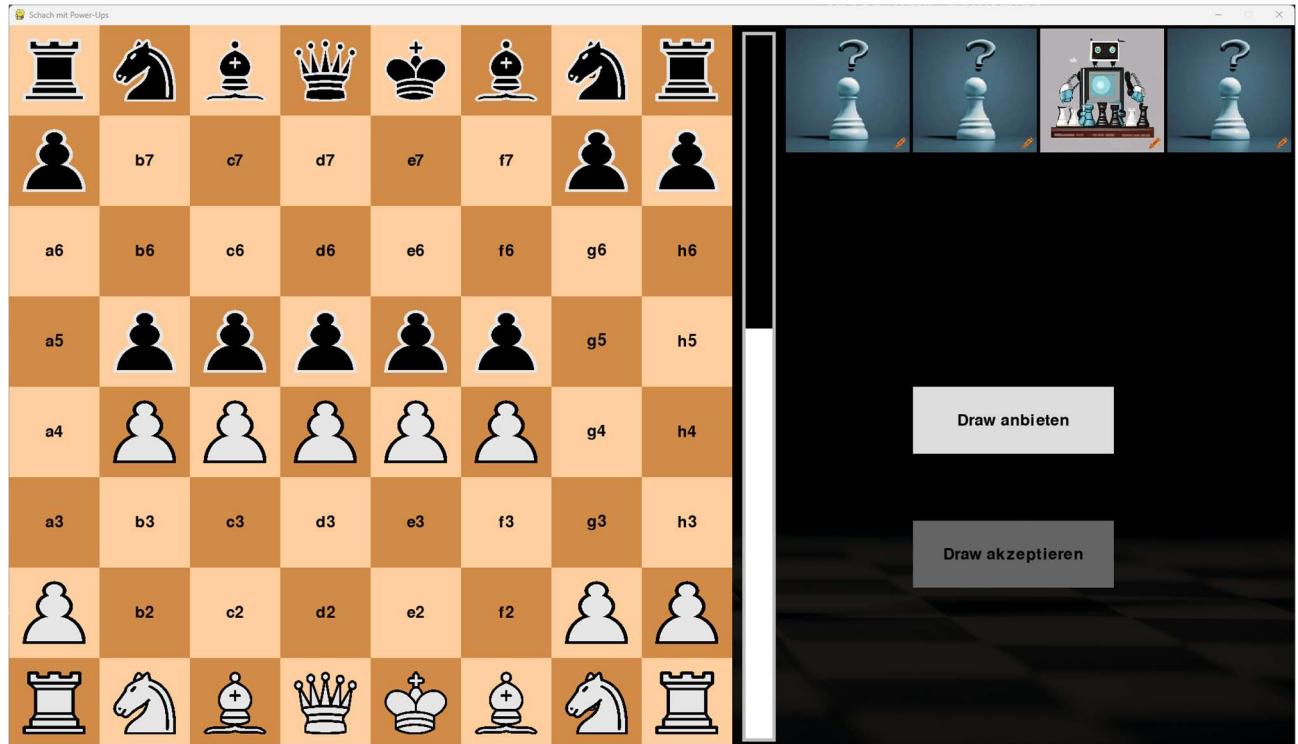
Schritte:

- Starte eine neue Partie.
- Der Spieler wählt die Option "Power-Up einsetzen".
- Überprüfe, ob dem Spieler die verfügbaren Power-Ups angezeigt werden.
- Der Spieler wählt ein Power-Up aus.

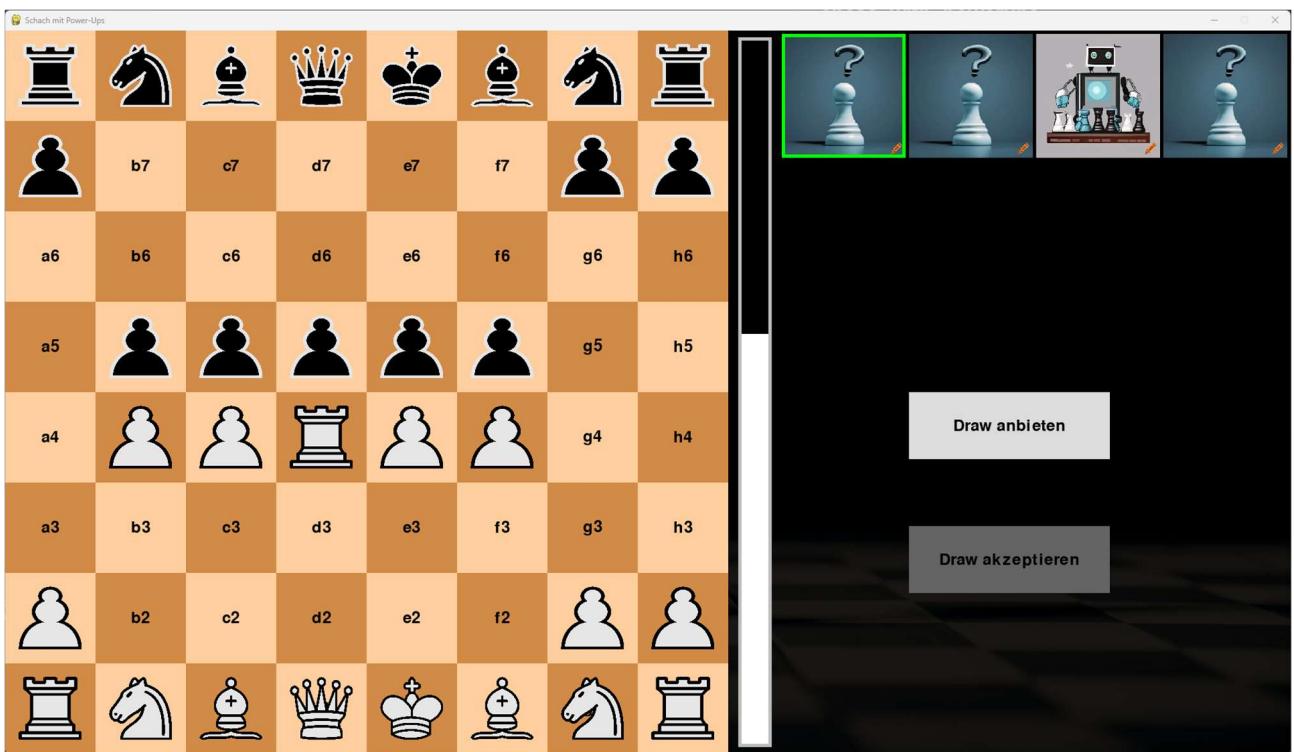
Erwartetes Ergebnis:

Das ausgewählte Power-Up wurde eingesetzt, und die Partie wird fortgesetzt.

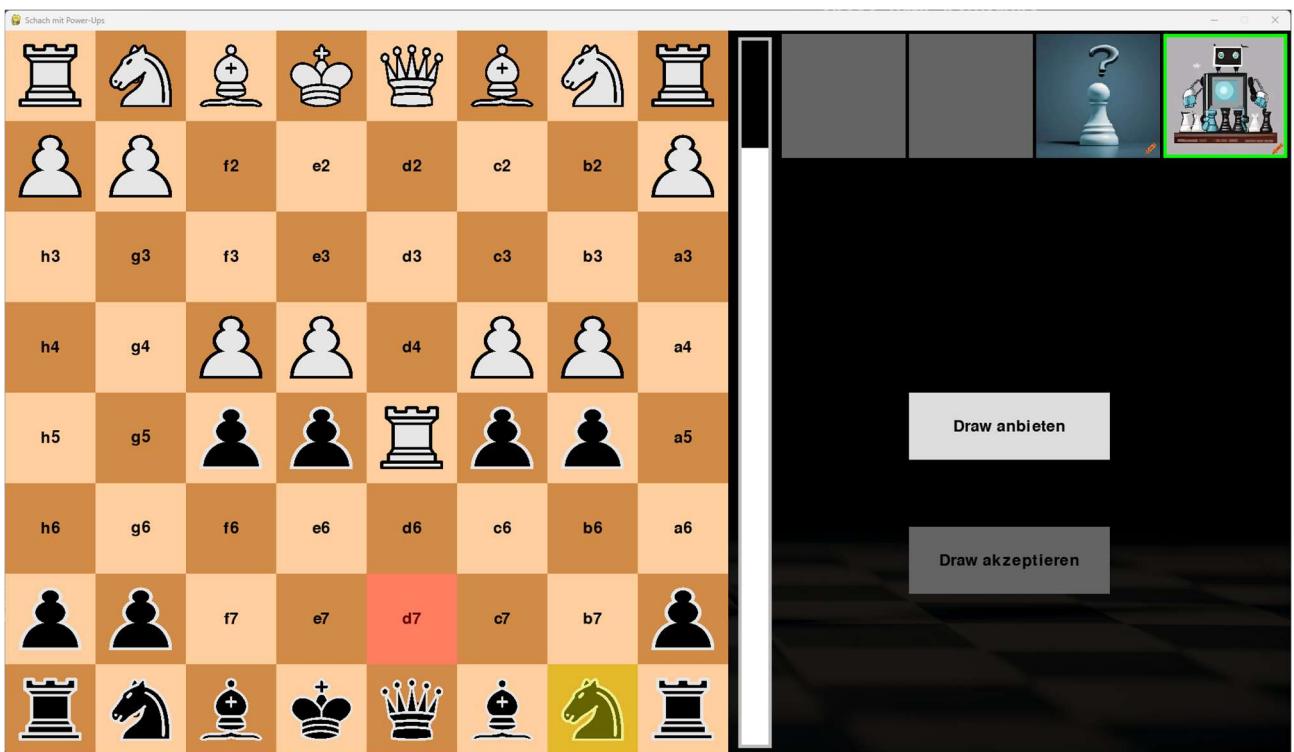
Testdurchführung:



Mit einem hohen Power-Up-Multiplikator (In diesem Fall 10), erscheinen nach wenigen Spielzügen zufällig die Power-Ups am oberen rechten Bildschirmrand. Wird die zufällige Bauernbeförderung ausgewählt, erkennt man das ein zufälliger Bauer nun ein Turm ist.



Bei dem folgenden Beispiel wird die Hilfe von Stockfish ausgewählt. Diese zeigt nach 5 Sekunden einen sehr guten Zug an.



1.5 Testfall 5: Schachmatt erkennen (Use Case 5)

Testziel:

Überprüfen, ob das Spiel erkennt, wenn ein Spieler im Schachmatt steht.

Schritte:

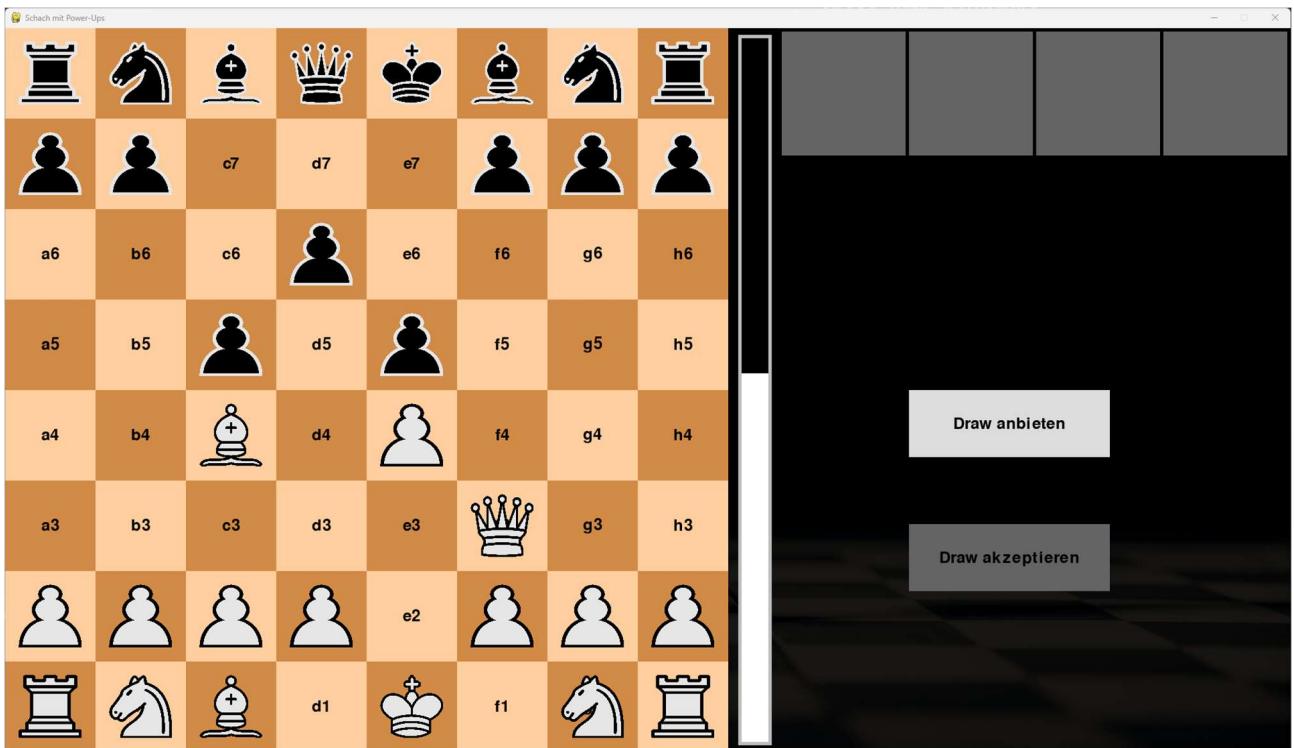
- Spiele eine Partie bis zu einem Schachmatt-Szenario.
 - Überprüfe, ob das Spiel das Schachmatt erkennt und die Partie beendet.

Erwartetes Ergebnis:

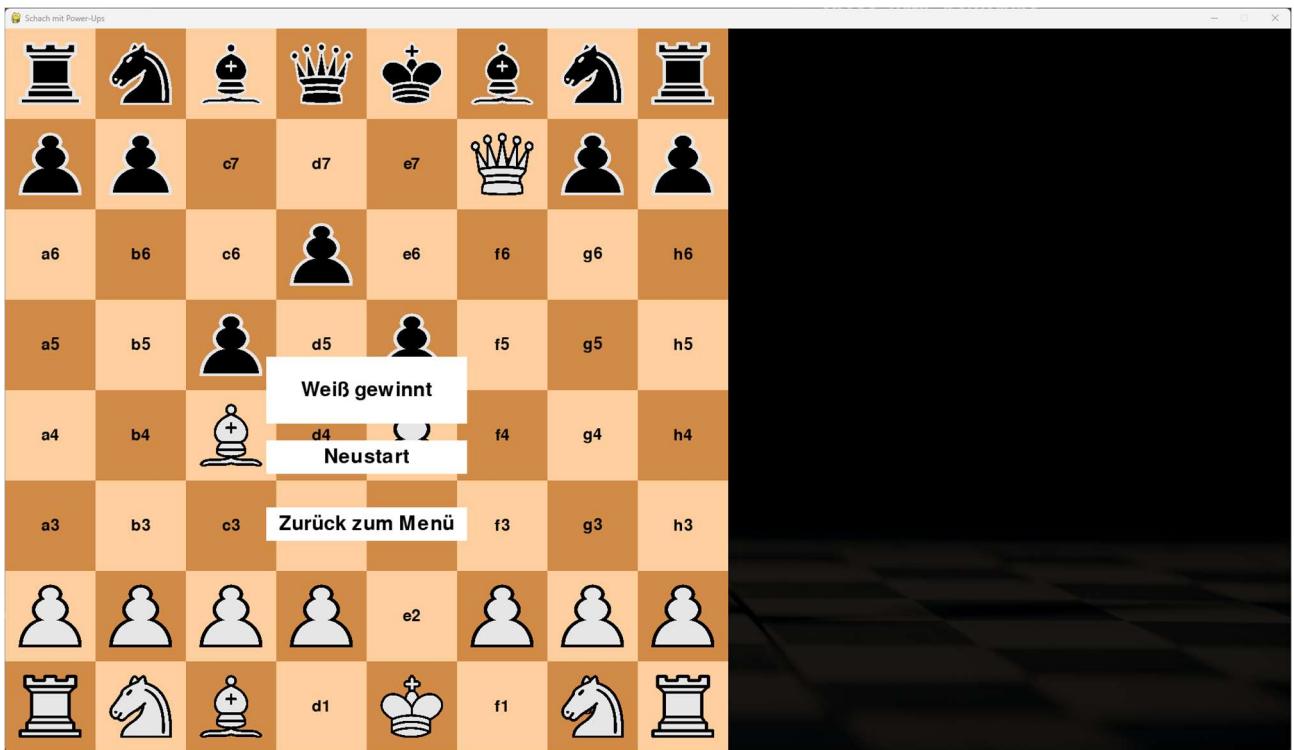
Die Partie endet, und der entsprechende Spieler gewinnt.

Testdurchführung:

Eine Partie steht kurz vor einem Schach-Matt



Sobald der letzte Zug getatigt wurde, erkennt das Spiel, dass ein Schach-Matt vorliegt und zeigt folgende Meldung an.



1.6 Testfall 6: Unentschieden erkennen (Use Case 6)

Testziel:

Die Partie endet, und der entsprechende Spieler gewinnt.

Schritte:

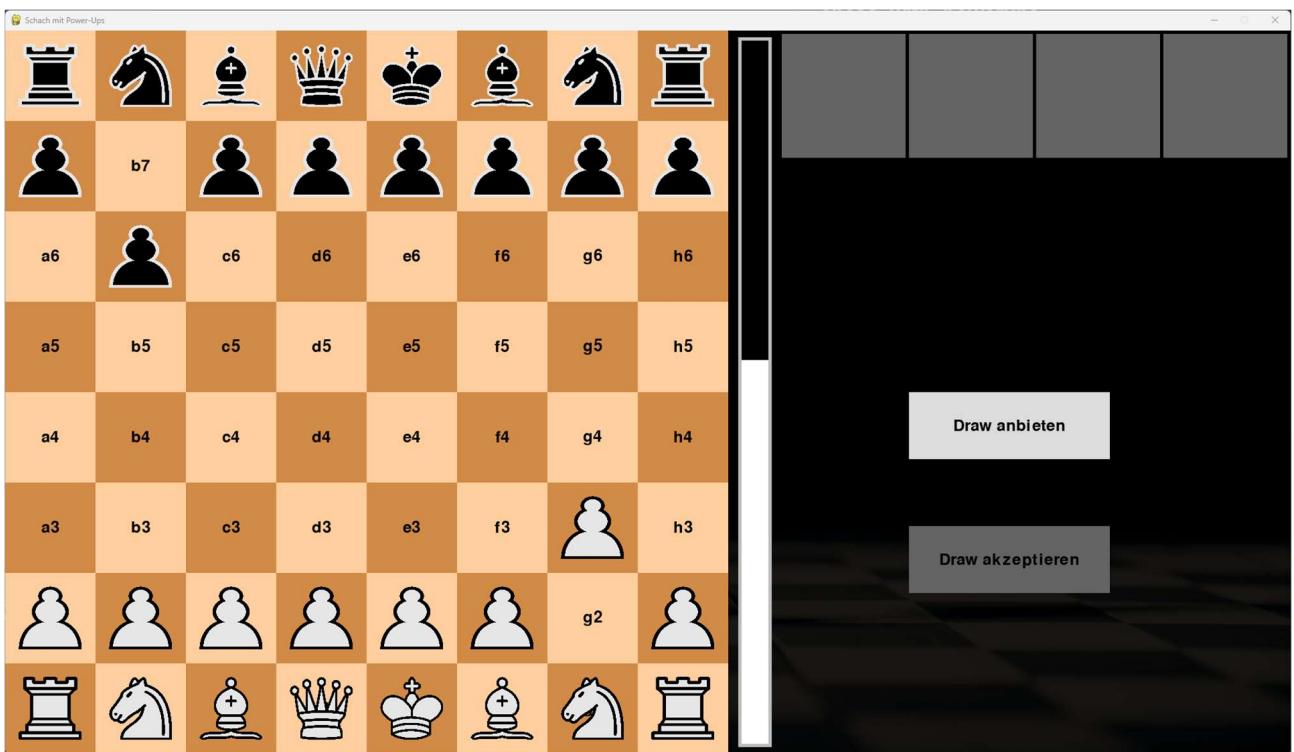
- Spiele eine Partie bis zu einem Unentschieden-Szenario (z.B. dreifache Stellungswiederholung oder 50-Züge-Regel).
 - Überprüfe, ob das Spiel das Unentschieden erkennt und die Partie beendet.

Erwartetes Ergebnis:

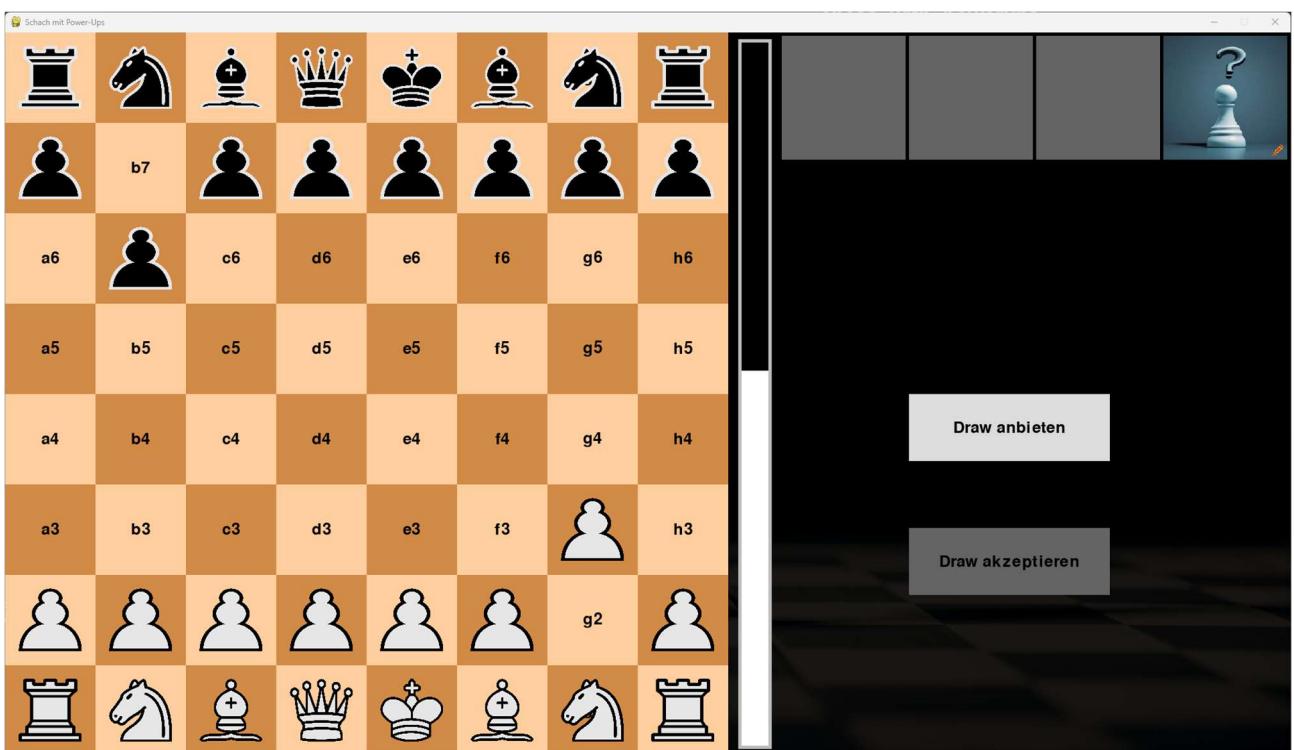
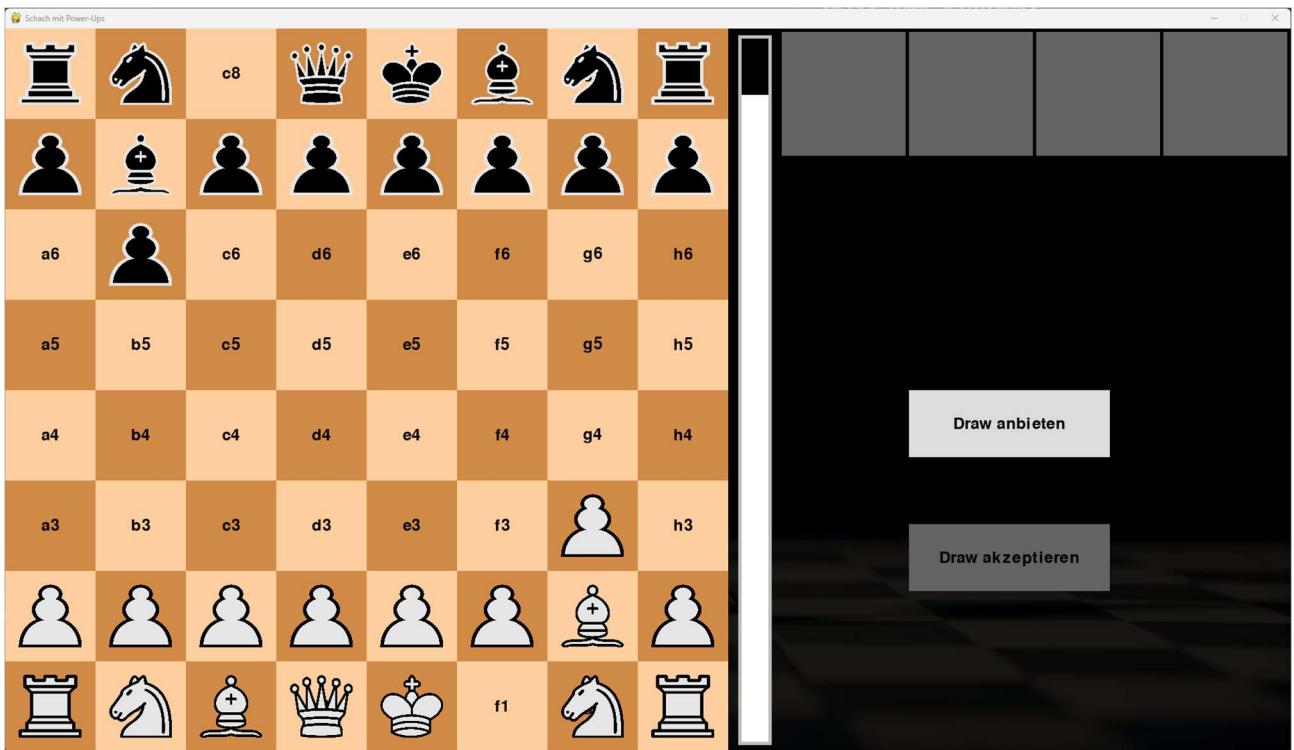
Die Partie endet unentschieden.

Testdurchführung:

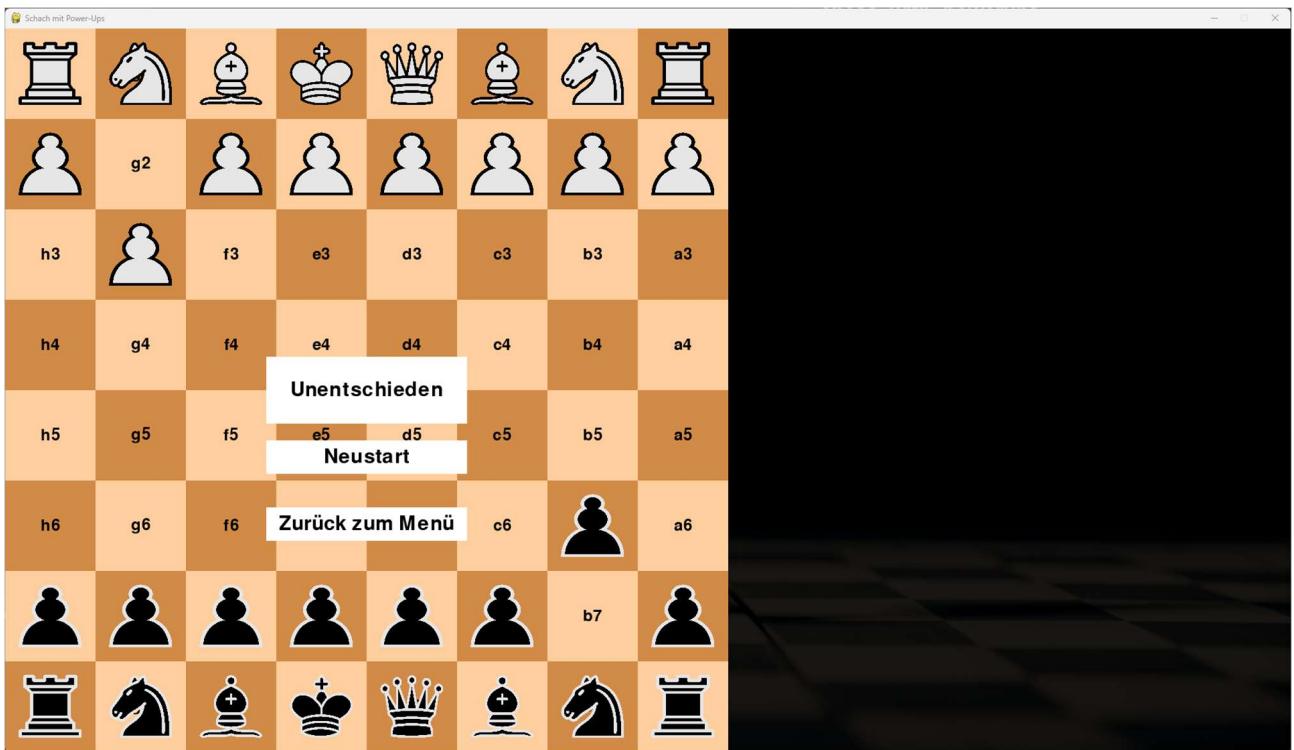
Eine Partie befindet sich im folgenden Zustand.



Beide Spieler t  igen die wiederholt den gleichen Zug (Springer vor und wieder z  ck ziehen)



Nach einer dreifachen Stellungswiederholung endet das Spiel in einem Unentschieden.



1.7 Testfall 7: Einstellungen ändern (Use Case 7)

Testziel:

Überprüfen, ob der Spieler die Einstellungen des Spiels erfolgreich ändern kann.

Schritte:

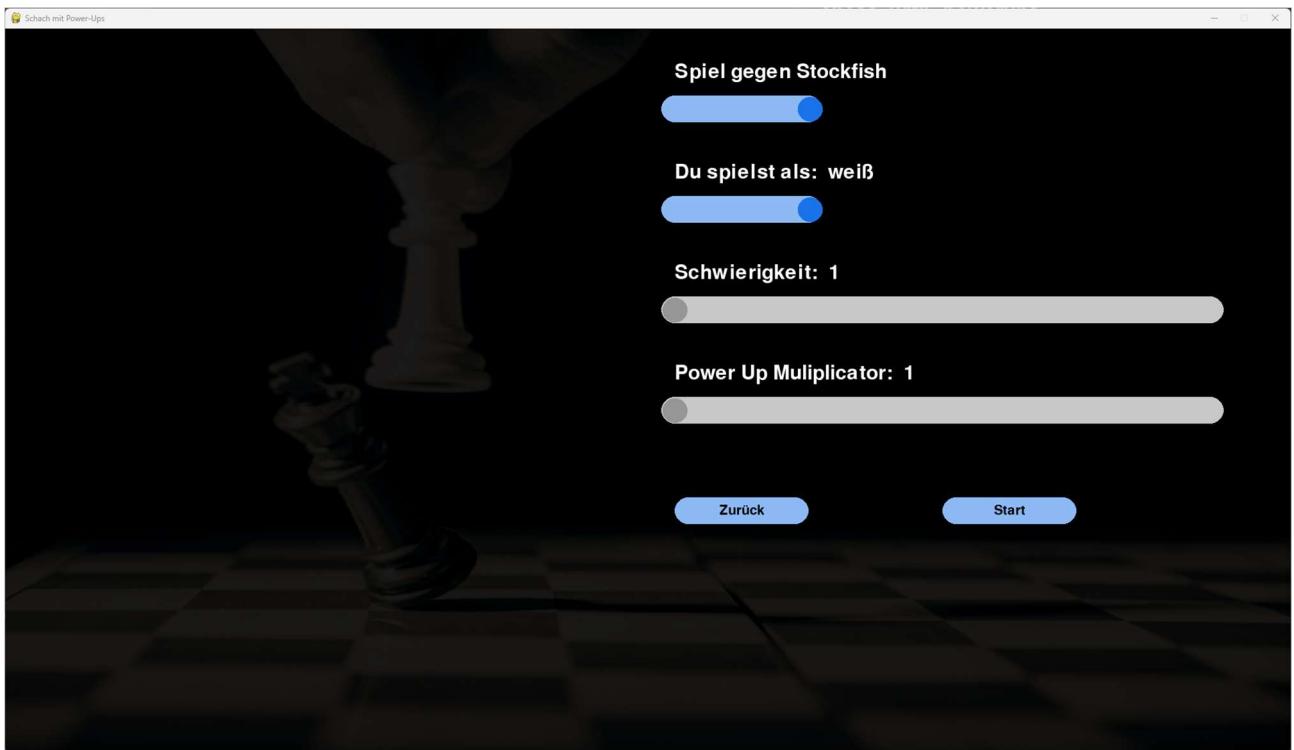
- Starte die Anwendung.
- Der Spieler navigiert zu den Einstellungen des Spiels.
- Wähle verschiedene Optionen aus (z.B. die Schwierigkeit des Computers oder mit welcher Farbe man beginnen möchte) und starte eine Partie.

Erwartetes Ergebnis:

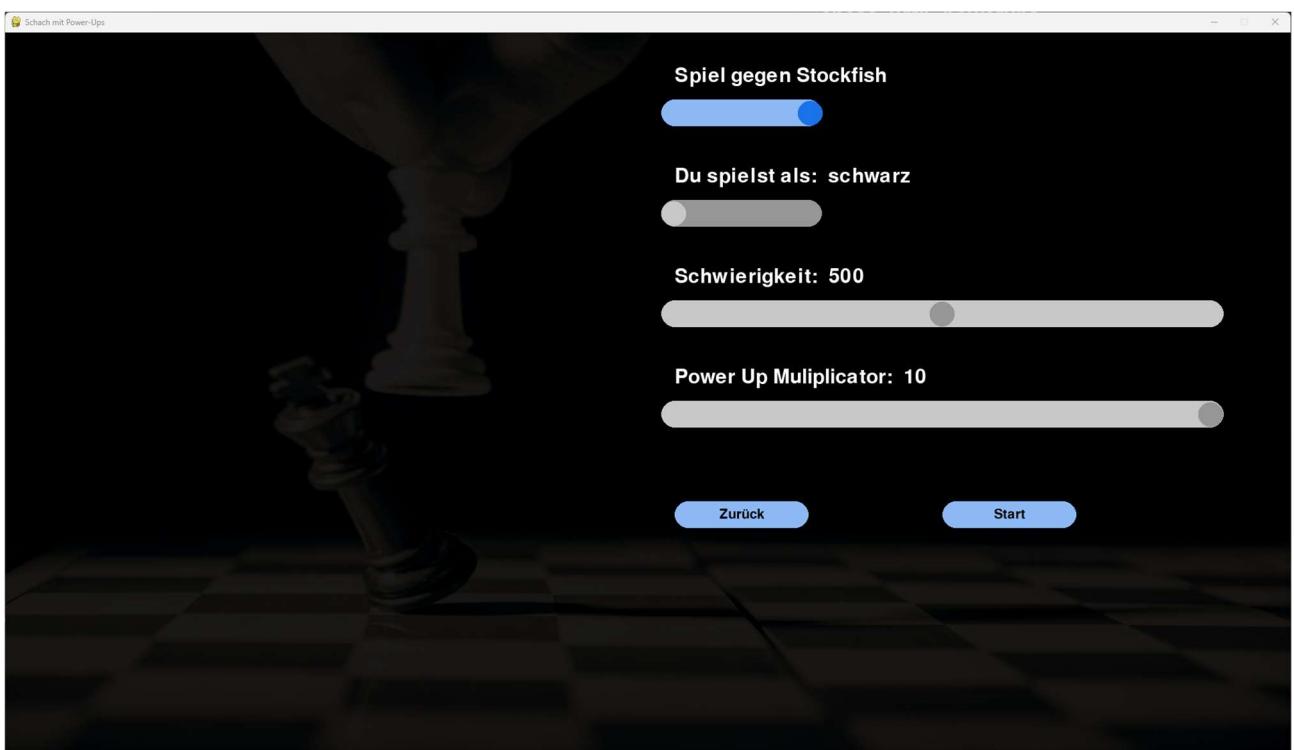
Die ausgewählten Einstellungen zu Start der Partie berücksichtigt und auf das laufende Spiel angewendet.

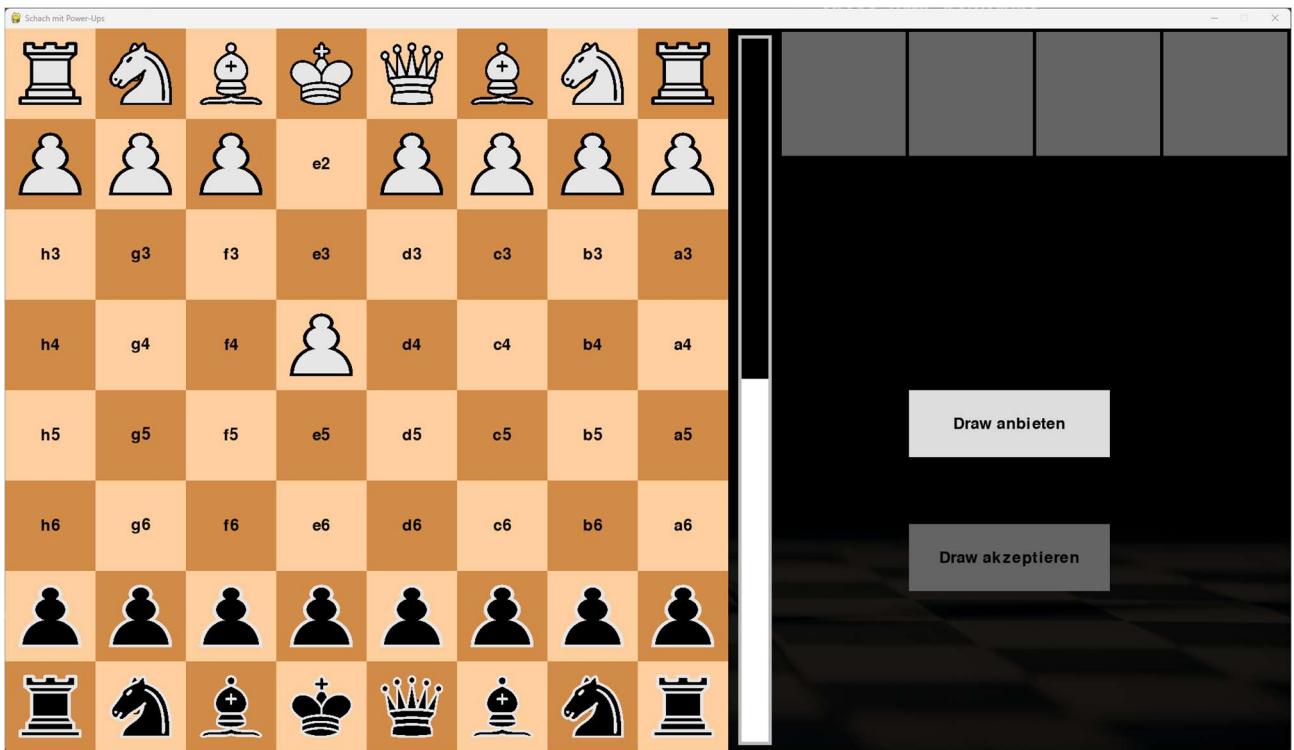
Testdurchführung:

Der Test beginnt in dem Einstellungsmenü vor einer Partie mit folgendem Bild.



Der Spieler ändert einiges Einstellungen.





Zu erkennen ist, dass man wie angegeben mit Schwarz spielt und Stockfish nun wie angegeben für 0.5 Sekunden rechnet (= Schwierigkeit 500).

1.8 Testfall 8: Spiel beenden (Use Case 8)

Testziel:

Überprüfen, ob der Spieler das Spiel erfolgreich beenden kann.

Schritte:

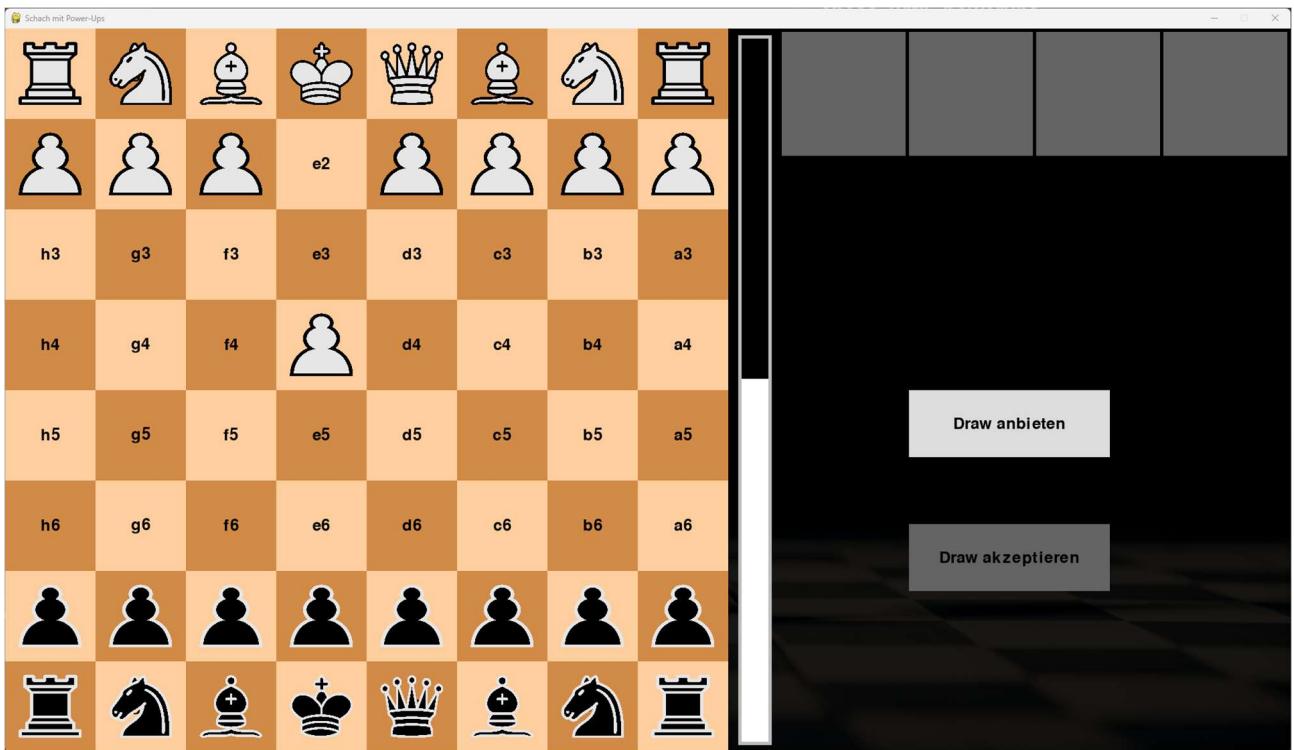
- Starte eine neue Partie oder gehe zum Hauptmenü.
- Der Spieler wählt die Option "Spiel beenden".
- Falls ein Spiel aktiv ist, überprüfe, ob das System nach dem aktuellen Fortschritt fragt.
- Der Spieler bestätigt das Beenden des Spiels.

Erwartetes Ergebnis:

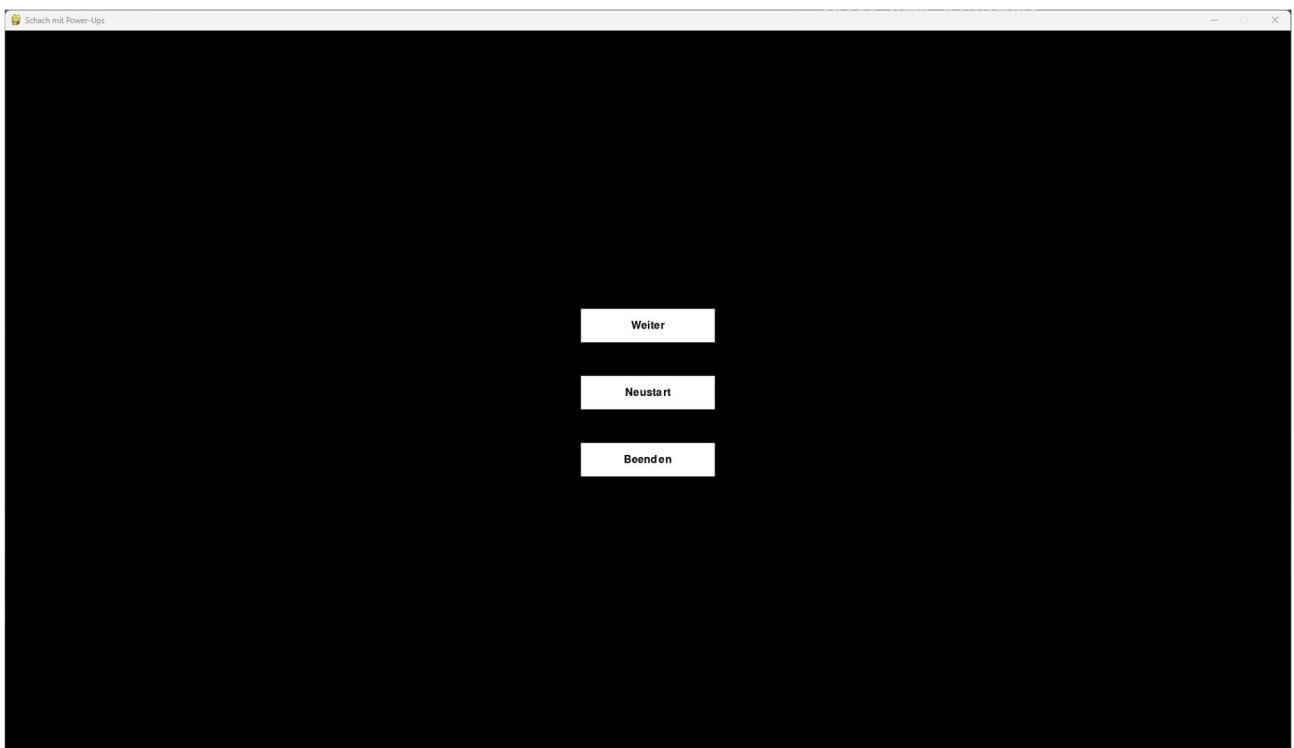
Das Spiel ist beendet, und die Anwendung ist geschlossen oder der Spieler ist zurück im Hauptmenü.

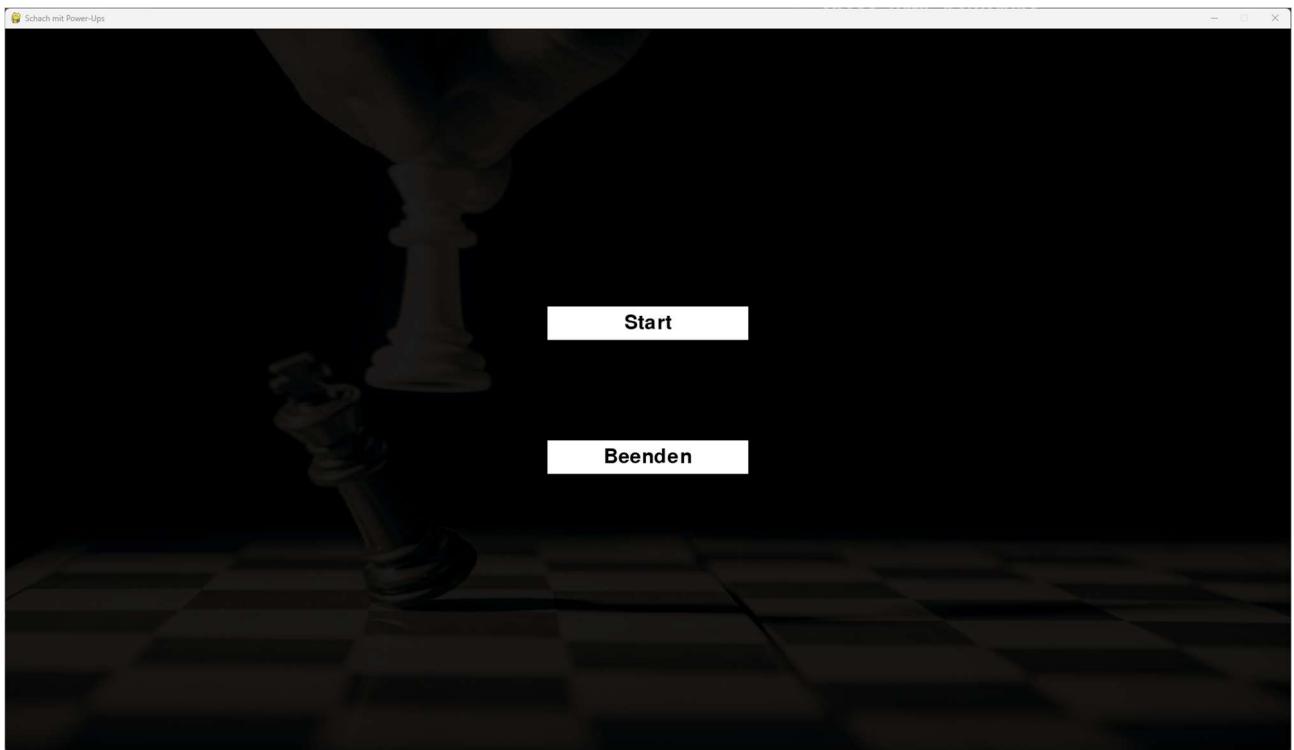
Testdurchführung:

Man befindet sich in einer Partie.



Durch Klicken der Escape-Taste, erscheint das Pausenmenü.





Beim Klick auf „Beenden“ kommt der Spieler ins Hauptmenü. Beim erneuten Drücken des „Beenden“-Buttons im Hauptmenü, hat sich die Anwendung beendet.



INTERNATIONALE
HOCHSCHULE

Abstract – DLMCSPSE01_D
Projekt: Software Engineering

Erstellt von: Dustin Lucht
Matrikelnummer: 92203340
Studiengang: Master of Science Informatik
Tutor: Markus Kleffmann
Datum: 17.12.2023

1 Einführung und Hintergrund

Das Projekt zielt darauf ab, ein innovatives Schachspiel mit Power-Ups zu entwickeln, das traditionelle Schachliebhaber und Gelegenheitsspieler anspricht. Die Integration von Power-Ups soll dem Spiel eine neue Dynamik verleihen.

2 Projektziele und -rahmen

Ziel ist die Entwicklung eines soliden Spielflusses, ansprechende UI-Designs, die Integration von Power-Ups und eine Möglichkeit die Schwierigkeit der KI anzupassen. Zwei Spielmodi sind geplant: Einzelspieler gegen den Computer (Stockfish) und ein Mehrspielermodus für das Spielen gegen einen Freund.

3 Durchführung des Projekts

Das Projekt begann mit der Einarbeitung in *pygame*. Nach der Implementierung der ersten Schaltflächen folgte die Integration einer Zustandsmaschine, um das Spiel in verschiedene Phasen zu gliedern. Dabei wurden der Splash-Screen, das Hauptmenü und das Einstellungsmenü vor dem Spiel erstellt. Die zentrale Programmierung des Schachspiels, vom Zeichnen des Bretts bis zum Implementieren von Drag-and-Drop der Figuren, sowie die Implementierung aller Schachregeln und Sonderstriche, beanspruchte den Großteil der Arbeit. Um Einarbeitungszeiten nach Pausen zu minimieren, wurde versucht täglich oder mit kurzen Unterbrechungen zu arbeiten. Gegen Ende des Projekts konnten die Power-Ups mühelos, aufgrund der bereits gesammelten Erfahrungen, integriert werden. Nach Tests und Bugfixes wurde das Spiel nach 12 Wochen erfolgreich abgeschlossen.

4 Einhaltung des Zeitplans

Die Zeitüberschreitung bei der Fertigstellung des Projekts von geplanten sechs Wochen auf tatsächlich 12 Wochen resultierte aus persönlichen Umständen, die während des Projektablaufs auftraten. Diese individuellen Herausforderungen beeinflussten die Arbeitsgeschwindigkeit und trugen zu der Verzögerung bei.



Die Grafik der auf GitHub hochgeladenen Commits spiegelt deutlich die Pausen wider, die während der Arbeit am Projekt eingelegt wurden.

5 Entwicklungsrückblick: Erfolge und Herausforderungen

Die größten Erfolge wie auch Herausforderungen traten zu Beginn des Projekts auf. Es war sehr herausfordernd das erste Mal in die Spieleentwicklung einzutauchen und zurechtzufinden. Das

erfolgreiche Bewältigen dieser Aufgabe war der erste Erfolg. Nach vier Tagen stand bereits die Zustandsmaschine inklusive des Splash-Screens, des Hauptmenüs und ersten Einstellungsmöglichkeiten. Zudem konnten bereits Spielfiguren bewegt werden und das Brett rotierte, wenn der nächste Spieler am Zug war. Eine weitere Herausforderung bestand in der zunehmenden Komplexität des Codes und der einhergehenden schlechteren Übersicht. Aus diesem Grund wurde im späteren Verlauf eine zweite Zustandsmaschine nur für das Mid-Game erstellt. Ein besonders großer Erfolg stellt die nahtlose Integration der Schach-Engine Stockfish, sowie der Power-Ups dar. Nicht nur einmal, wurde mit einem einfachen Test begonnen und anschließend eine gesamte Partie bis zum Ende gespielt, obwohl der Test bereits abgeschlossen war.

6 Kritische Reflektion

Kritische zu betrachten ist allerdings die eindeutige Unterschätzung des Aufwands für dieses Projekt und dem Erstellen eines Schachspiels. Dies ist gut erkennbar, wenn man das Klassendiagramm, welches zu dem Beginn des Projektes erstellt wurde mit dem Ist-Zustand zum Ende des Projekts vergleicht (siehe Projektdokumentation). Ebenfalls unterschätzt wurde der schriftliche Aufwand, sprich die Erstellung aller nötigen Dokumente (Projektplan, Anforderungsdokument etc.). Der Zeitverzug des Projektes kann ebenfalls kritisch betrachtet werden. Aufgrund der nicht vorhandenen Frist war es aber die richtige Entscheidung die Zeit in dem Projekt im richtigen Moment zu verringern, als diese anderweitig benötigt wurde.

7 Wichtige Erkenntnisse aus dem Projekt

Folgende Erkenntnisse wurden in diesem Projekt gewonnen und für die Zukunft mitgenommen:

- Plane Pausen (bzw. einen Puffer) direkt bei dem Projektplan mit ein.
- Gehe die möglichen Entwicklungen vorab genauer durch und schreibe alle Sachen auf, die erledigt werden müssen, um das Ziel zu erreichen. Mehr Vorbereitung bedeutet weniger Umdenken während des Projekts.
- Unterschätze nicht den Aufwand für Dokumentation und Tests. Plane diese mit ein. Ein Projekt besteht i.d.R. nicht nur aus Programmieren.

8 Schlussbetrachtung

Als gelegentlicher Schachspieler hat es mir sehr viel Freude bereitet tiefer in das Schachspiel und der Programmierung eines Schachspiels einzusteigen. Dieser Tatsache geschuldet, inklusive der gewonnenen Erkenntnisse in der Planung und Entwicklung, bereue ich es nicht, dass ich deutlich mehr Zeit in dieses Modul investiert habe als in andere. Ich habe viel über die Projektplanung und Durchführung, das Entwickeln von Spielen im Allgemeinen und den Umgang mit einer Schach-Engine gelernt. Zudem hatte ich einige spannende Spiele gegen Stockfish und konnte es mithilfe meiner Power-Ups gelegentlich besiegen. Die Durchführung des Moduls und des Projekts war somit zusammenfassend ein lehrreicher Erfolg.