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

II. Tabellenverzeichnis

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.

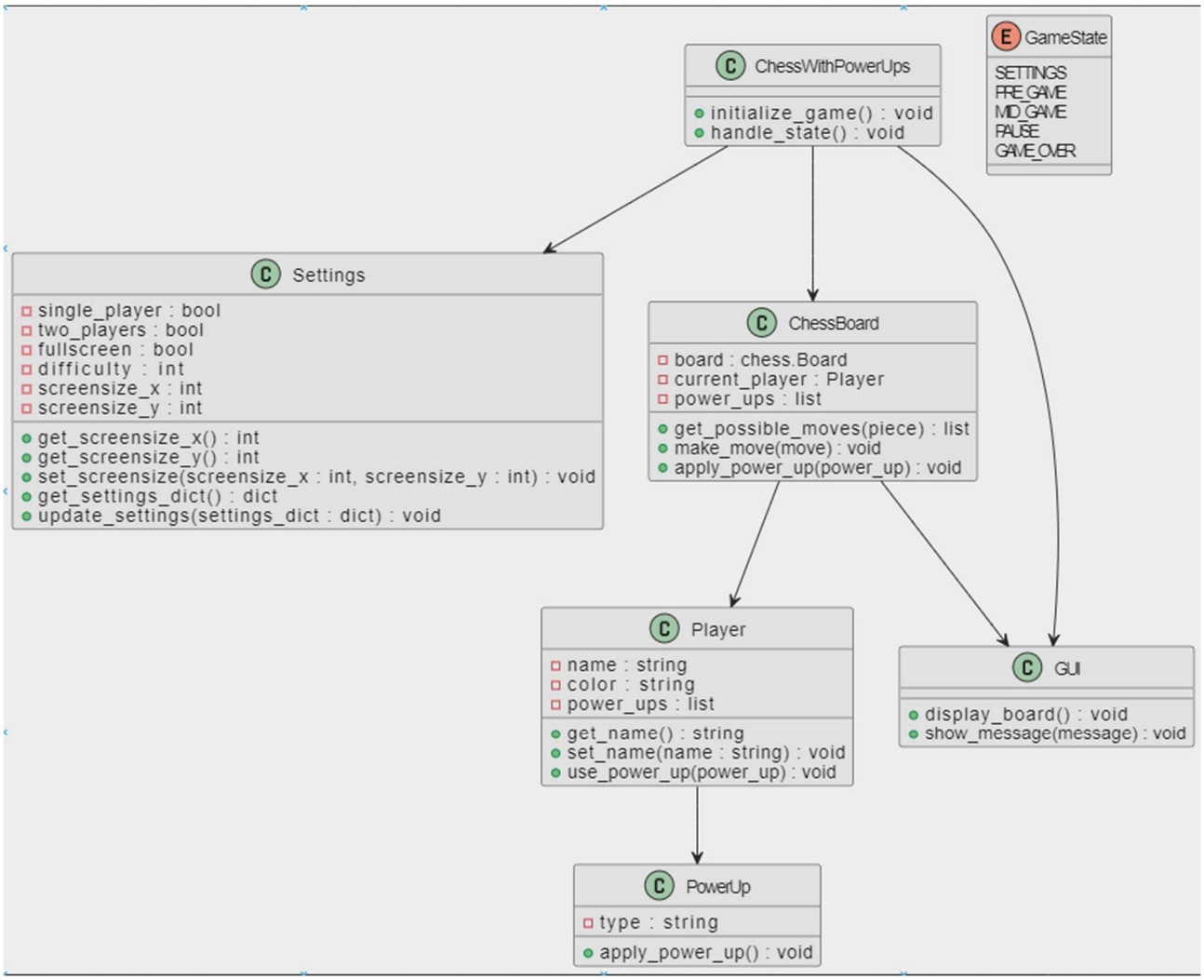

Abbildung 1: UML-Klassendiagramm zu Beginn des Projektes

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:

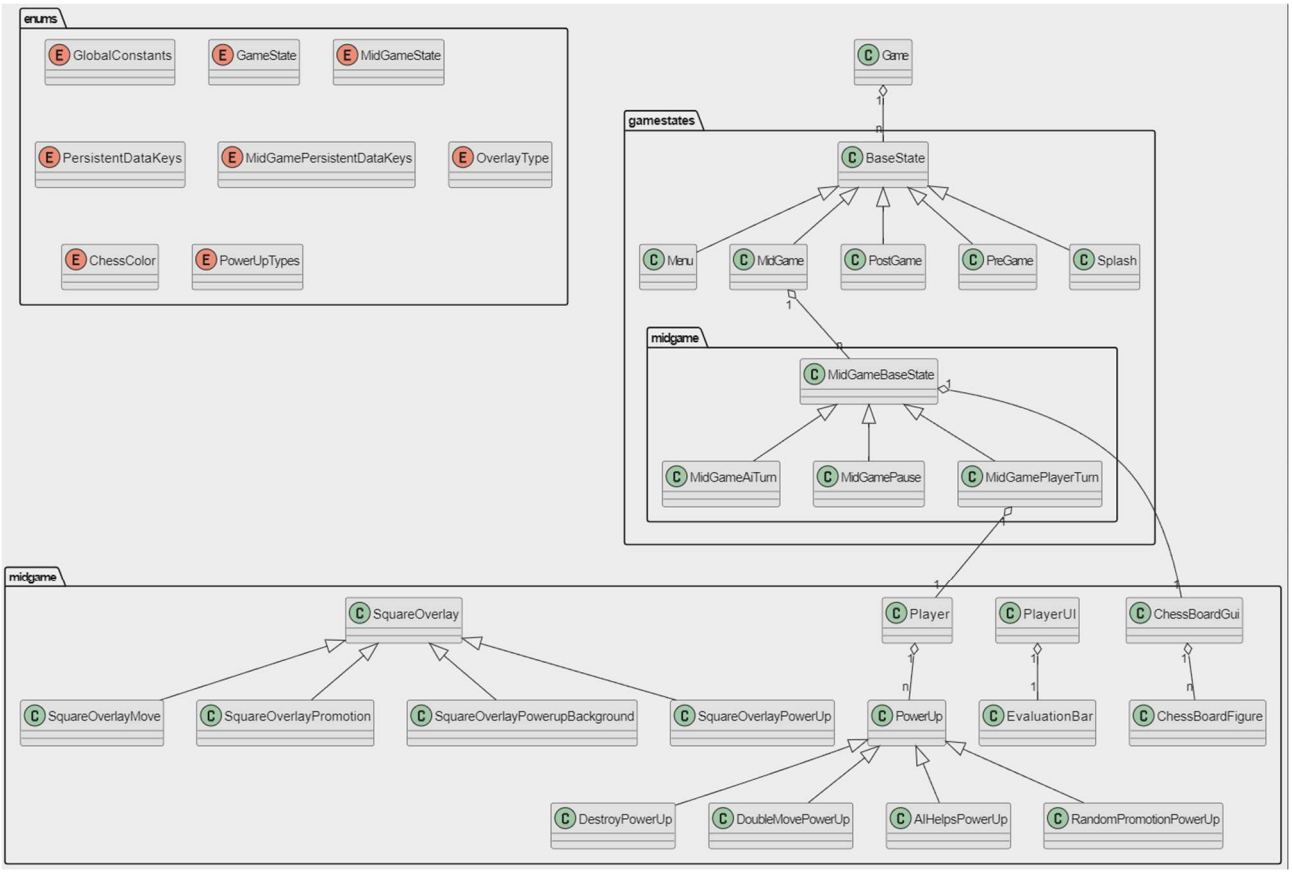

Abbildung 2: UML-Klassendiagramm am Ende des Projektes

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 *MidGames* 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:

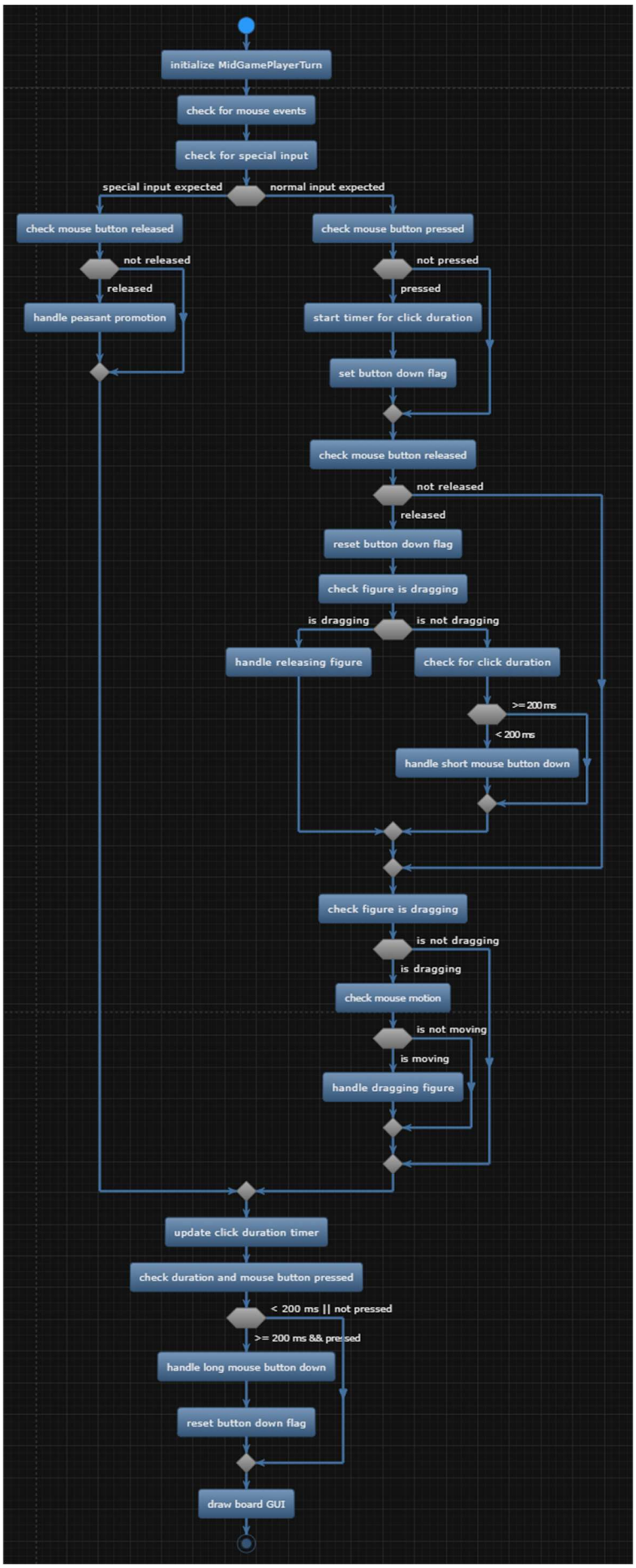

Abbildung 3: UML-Aktivitätsdiagramm

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 Funkion „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:

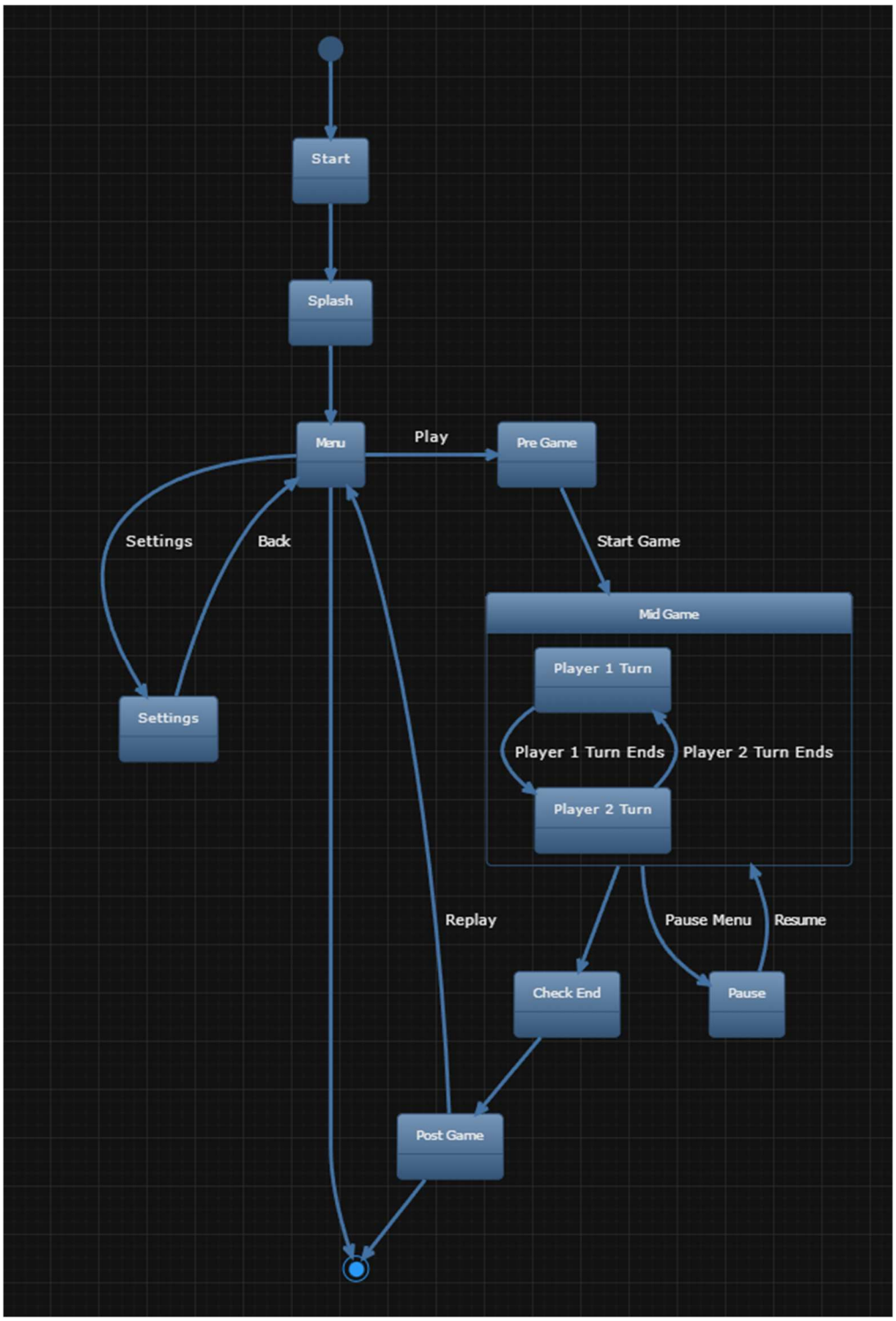

Abbildung 4: Zustandsdiagramm

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:

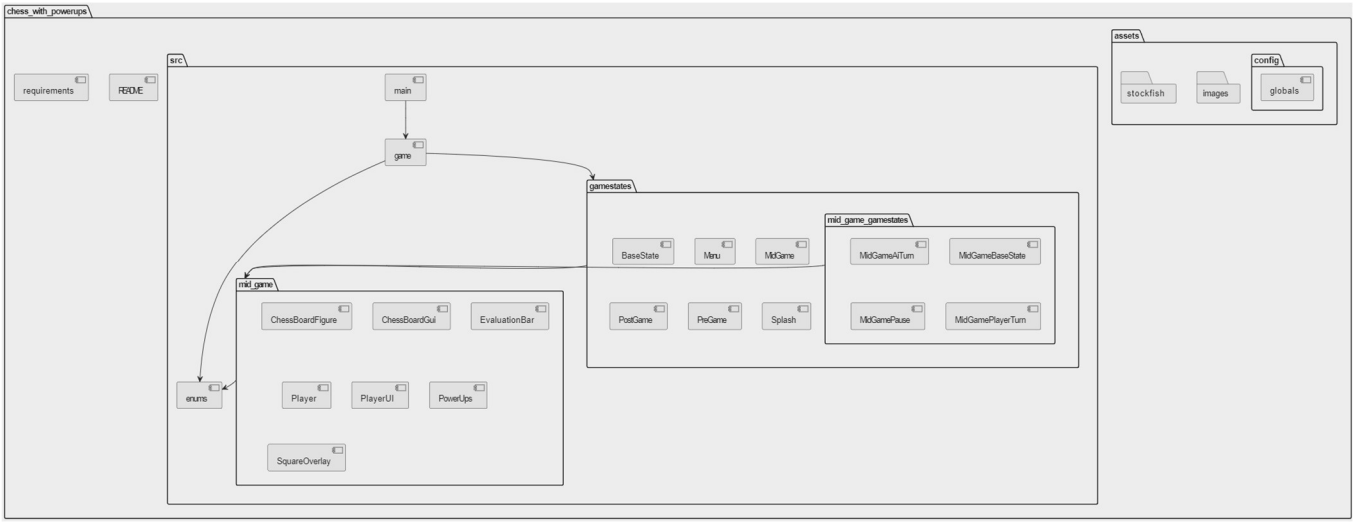

Abbildung 5: Komponentendiagramm

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“