Fachkonzept zum Programmierprojekt WS12/13

Zusammenstellung

von

Jonne Hass, Benjamin Held, Richard Pump,
Alexander Sirotin, Julian Haack

21. November 2012

Betreuer:
Michael Antemann

Version 3.0

Hochschule Hannover, Abt. Informatik

Inhaltsverzeichnis

Kapitel 1

Fachkonzept - Einleitung

1.1 Systemzweck

Der Zweck dieses Projekts ist es im Rahmen der Veranstaltung „Softwareprojekt im 3.Semester“ ein größeres Prorammierprojekt in kleinen Teams durchzuführen und erfolgreich umzusetzen. Aufgabe ist es das klassische Brettspiel „Mühle“ als Java-Applikation umzusetzen. Dabei sollen nicht nur zwei menschliche Spieler gegeneinander antreten können, sondern auch ein Spieler gegen eine künstliche Intelligenz oder zwei vom Computer gesteuerte Spieler gegeneinander spielen. Eine „Musterlösung“ für diese Aufgabe ist dabei nicht vorhanden. Viel mehr sollen die Projektteilnehmer die Aufgabe nach eigenen Vorstellungen planen und umzusetzten.
Das Softwareprojekt soll dabei einen Einblick in die spätere Arbeit an größeren Projekten bieten und dafür die Grundlagen bereitstellen, die über das Semester hinüber vermittelt werden sollen.

1.2 Ziele und Erfolgskriterien des Projekts

Die erfolgreiche Bearbeitung des Softwareprojektes ist dabei an mehrere Ziele und Kriterien geknüpft. Wir möchten ein leicht verständliches, gut bedienbares Programm erstellen. Es soll die optionale Möglichkeit haben, nicht gegen menschliche Spieler anzutreten, sondern auch gegen den Computer, der wiederum über verschiedenen Schwierigkeitsstufen verfügen soll.
Das ganze Projekt soll termingerecht zum Ende des Wintersemesters 2012/2013 fertig sein, die durch den Dozenten aufgestellten Meilensteine eingehalten werden. Da dieses Softwareprojekt im Rahmen des Studiums der angewandten Informatik abläuft, steht neben dem Erlernen der Grundfertigkeiten der Projektarbeit auch das Sammeln von Erfahrung, sowie die Erweiterung bereits vorhandener Fertigkeiten im Vordergrund.
Um diese Ziele erfolgreich umzusetzen ist es notwendig, dass alle Teilnehmer gut und effektiv zusammenarbeiten. Eigene Schwächen und Fehler werden dabei sicherlich vorkommen, sollten dann aber eingestanden und bewältigt werden. Da es sich um ein Gemeinschaftprojekt handelt, sollte man zusammen mit anderen effektive Lösungmöglichkeiten erarbeiten und nicht ausschließlich seine eigenen Lösungen verwenden (egolose Programmierung).

Die Einhaltung des vorgegebenen Zeitrahmens ist dabei unerlässlich. Die Aufgaben müssen zügig, aber qualitätiv hochwertig bearbeitet werden, damit ausreichend Zeit zur Qualitätskontrolle und Nachbesserung vorhanden ist. Um dies erfolgreich zu bewältigen ist ein gewisses Maß an Eigeninitiative notwendig, die nicht nur das machen lässt, was unbedingt erforderlich ist, sondern jeden dazu ermuntern sollte neue Ansatzpunkte und Ideen mit in des Projekt einzubringen. Dabei sollten alle ihre erworbenen Fähigkeiten einbringen.

1.3 Kurzüberblick Mühle-Regeln

Im Folgenden werden die wichtigsten Phasen und Regeln des zu programmierenden Spieles kurz erläutert. Das Spiel besteht aus 3 Phasen:

- Phase 1: Abwechselndes Setzen der Spielsteine auf das Spielfeld
- Phase 2: Abwechselndes Bewegen eines Spielsteins in ein angrenzendes, freies Feld, solange man mehr als 3 Spielsteine hat
- Phase 3: Abwechselndes Bewegen eines Spielsteins, wobei der Spieler mit 3 verbleibenden Spielsteinen auf ein beliebiges, freies Feld springen darf

Die grundlegenden Spielregeln sind folgende:

- Immer wenn ein Spieler am Zug ist (Phase 2 & 3) darf er einen seiner Spielsteine bewegen
- Schließt ein Spieler durch das Bewegen eines Spielsteins eine Mühle (3 Steine in einer Reihe), so darf er einen gegnerischen Spielstein entfernen (der zu keiner geschlossenen Mühle gehört)
- Hat der gegnerische Spieler nur Spielsteine, die eine geschlossene Mühle bilden, so darf er einen Spielstein aus einer Mühle entfernen
- Kann ein Spieler keinen seiner Steine bewegen, verliert er das Spiel
- Hat ein Spieler nur noch 2 Spielsteine, verliert er das Spiel

Kapitel 2

Anforderungen

In diesem Abschnitt werden die funktionalen und nicht-funktionalen Aspekte des Projektes dargestellt.

2.1 Use-Case Diagramme

Das Use-Case Diagramm zeigt die Akteure, die die im Folgenden beschriebenen Use-Cases ausführen können. Der Admin (meist die gleiche Person, wie der Spieler) initialisiert das Programm, indem er es startet.
Der Spielstart unterteilt sich dann in zwei Möglichkeiten: Das Konfigurieren und Starten eines neuen Spiels oder die Fortsetzung eines Spiels mittels eines zu ladenden Spielstandes.
Danach sind beide Spieler abwechselnd am Zug und führen je nach Phase des Spiels einen Spielzug aus, der dann von der Logikebene auf seine regelgerechte Ausführung überprüft wird. Ist der Spielzug gültig, wird dieser vom Spiel notiert und ausgegeben. Hat einer der Spieler gewonnen oder möchte einer der Spieler das Spiel vorzeitig beenden, so wird vom Admin die entsprechende Aktion zum Beenden des Programmes durchgeführt.

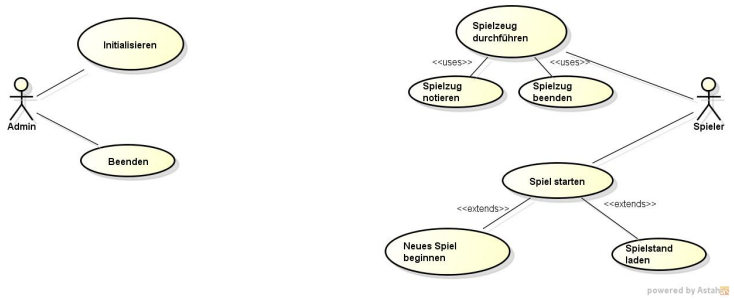

Abbildung 2.1: Use-Case Diagramm

2.2 Use-Case Beschreibungen

Das Projekt umfasst 6 Use-Cases. Use-Case 1 beschreibt die Initialisierung des Programms, während Use-Case 2 und 3 die beiden resultierenden Möglichkeiten eines neuen Spiels, sowie dem Laden eines vorhandenen Spielstandes beschreiben.
Use-Case 4 beschreibt den Ablauf eines Spielzuges, der mittels Use-Case 5, dem Notieren eines Spielzuges, geloggt wird. Use-Case 6 befasst sich abschließend mit dem Beenden des Programms durch den Benutzer.
Wie bereits beim Use-Case Diagramm beschrieben werden die Use-Cases 1 und 6 vom Admin ausgeführt, während die anderen Use-Cases durch den Spieler ausgeführt werden.

Use-Case 1: Initialisieren
Kurzbeschreibung: Use-Case zur Beschreibung der Initilisierung
Vorbedingung: Das Spiel ist gestartet, Initialmenü wird angezeigt
Szenario: Verschiedene Auswahlmethoden -Neues Spiel: Erstellen eines neuen Spiels -Spiel laden: Laden eines gespeicherten Spiels -Spiel beenden: Das Spiel beenden
Nachbedingung: Abhängig von der Auswahl wird mit dem Folgeschritt fortgefahren
Ausnahme: -

Tabelle 2.1: Use-Case 1

Use-Case 2: Neues Spiel
Kurzbeschreibung: Use-Case zum Erstellen eines neuen Spiels
Vorbedingung: Spiel gestartet, Wahl von „Neues Spiel“ in Use-Case 1
Szenario: -Eingabe der Spielernamen -Festlegung von menschlichen Spielern oder Computerspielern -Bei Auswahl eines Computerspielers wird die Denkzeit/ Schwierigkeit festgelegt
Nachbedingung: Das Spiel kann gestartet werden
Ausnahme: Die KI-Komponente kann nicht gefunden werden, eine entsprechende Fehlermeldung wird ausgegeben

Tabelle 2.2: Use-Case 2

Use-Case 3: Spiel laden
Kurzbeschreibung: Use-Case zum Laden eines gespeicherten Spielstandes
Vorbedingung: Spiel gestartet, Wahl von „Spiel laden“ in Use-Case 1
Szenario: -Datei mit gespeicherten Spielinformationen wird geladen -Das Programm ließt die Informationen ein und setzt den gespeicherten Spielzeitpunkt um
Nachbedingung: Das Spiel kann fortgesetzt werden
Ausnahme: Es kann kein gültiger Spielstand gefunden werden, eine entsprechende Fehlermeldung wird ausgegeben

Tabelle 2.3: Use-Case 3

Use-Case 4: Spielzug durchführen
Kurzbeschreibung: Use-Case zur Beschreibung des Ablaufs eines Spielzuges
Vorbedingung: Das Spiel wurde erfolgreich gestartet, es wurde erfolgreich ein Spielstand geladen
Szenario: -Der aktuelle Spieler setzt (Phase 1) oder bewegt (Phase 2/3) einen seiner Spielsteine -Das System prüft die Korrektheit des Zuges -Der aktuelle Spieler entfernt einen Spielstein, sollte er eine Mühle geschlossen haben -Das System überprüft, ob ein zulässiger Stein entfernt wurde -Das System überprüft die Siegbedingungen
Nachbedingung: Loggen des Spielzuges
Ausnahme: Ein falscher Spielzug fordert den Spieler zur Durchführung eines regelkonformen Spielzuges auf

Tabelle 2.4: Use-Case 4

Use-Case 5: Spielzug notieren (Loggen)
Kurzbeschreibung: Use-Case zum Notieren des ausgeführten Spielzuges
Vorbedingung: Regelkonformer Spielzug wurde durchgeführt
Szenario: -Das System notiert das Feld, in das ein Stein gesetzt wurde (Phase 1) -Das System notiert das Feld, aus dem und in das gezogen worden ist
Nachbedingung: Der nächste Spieler ist am Zug
Ausnahme: -

Tabelle 2.5: Use-Case 5

Use-Case 6: Spiel beenden
Kurzbeschreibung: Use-Case zum Beenden eines laufenden oder abgeschlossenen Spiels
Vorbedingung: Spiel ist im Gange
Szenario: -Der Spieler möchte das Spiel beenden, obwohl es nicht abgeschlossen ist -Er kann einen Speicherstand anlegen -Das Programm wird beendet
Nachbedingung: Gegebenenfalls wurde ein Speicherstand angelegt
Ausnahme: -

Tabelle 2.6: Use-Case 6

2.3 Testfälle

Aus den eben beschriebenen Use-Cases werden nun verschiedene Testfälle abgeleitet, die während der Ausführung des Programmes auftreten können.

2.3.1 Use-Case 1: Initialisieren

Testfall 1.1: Spiel wird gestartet.
Situation: -
Erwartetes Ergebnis: Der Startdialog wird angezeigt.

2.3.2 Use-Case 2: Neues Spiel

Testfall 2.1: Neues Spiel erfolgreich starten.
Situation: Der Benutzer erstellt ein neues Spiel.
Erwartetes Ergebnis: Nach Eingabe der relevanten Daten wird ein neues Spiel gestartet.

Testfall 2.1.1: Zwei menschliche Spieler.
Situation:

- Der Admin wählt ein neues Spiel aus.
- Der Admin wählt einen menschlichen Spieler für Weiß aus.
- Der Admin wählt einen menschlichen Spieler für Schwarz aus.
- Der Admin startet das Spiel.

Erwartetes Ergebnis: Ein neues Spiel wird gestartet.

Testfall 2.1.2: Ein menschlicher Spieler, ein KI-Spieler.
Situation:

- Der Admin wählt ein neues Spiel aus.
- Der Admin wählt einen menschlichen Spieler für Weiß aus.
- Der Admin wählt einen KI-Spieler für Schwarz aus.
- Der Admin setzt eine Denkzeit für den KI-Spieler.
- Der Admin startet das Spiel.

Erwartetes Ergebnis: Ein neues Spiel wird gestartet.

Testfall 2.1.3: Zwei KI-Spieler.
Situation:

- Der Admin wählt ein neues Spiel aus.
- Der Admin wählt einen KI-Spieler für Weiß aus.
- Der Admin wählt einen KI-Spieler für Schwarz aus.
- Der Admin setzt eine Denkzeit für den ersten KI-Spieler.
- Der Admin setzt eine Denkzeit für den zweiten KI-Spieler.
- Der Admin startet das Spiel.

Erwartetes Ergebnis: Ein neues Spiel wird gestartet.

Testfall 2.2: Neues Spiel nicht erfolgreich starten.
Situation: Der Benutzer erstellt ein neues Spiel, es werden ein oder zwei KI-Spieler ausgewählt.
Erwartetes Ergebnis: Es kann keine KI-Komponente gefunden werden und es wird eine Fehlermeldung ausgegeben.

2.3.3 Use-Case 3: Spiel laden

Testfall 3.1: Einen Spielstand laden.
Situation: Der Admin lädt einen Spielstand.
Erwartetes Ergebnis: Ein Datei öffnen Dialog wird angezeigt.

Testfall 3.2: Ein gespeichertes Spiel erfolgreich laden.
Situation:

- Der Admin lädt einen Spielstand.
- Der Admin lädt eine Datei.
- Die Datei ist ein kompatibler Spielstand.

Erwartetes Ergebnis: Der Spielstand wird geladen, das Spiel kann fortgesetzt werden.

Testfall 3.3: Ein gespeichertes Spiel nicht erfolgreich laden.
Situation: Der Benutzer möchte einen existierenden Spielstand laden, es existiert allerdings kein gültiger Spielstand.
Erwartetes Ergebnis: Es kann kein gültiger Spielstand gefunden werden und es wird eine Fehlermeldung ausgegebe.

Testfall 3.3.1: Inkompatible Datei laden.
Situation:

- Der Admin lädt einen Spielstand.
- Der Admin lädt eine Datei.
- Die Datei ist ein inkompatibler Spielstand.

Erwartetes Ergebnis: Der Admin wird hierüber informiert und kann eine andere Datei auswählen.

Testfall 3.3.2: Datei ist kein Spielstand.
Situation:

- Der Admin lädt einen Spielstand.
- Der Admin lädt eine Datei.
- Die Datei ist kein Spielstand.

Erwartetes Ergebnis: Der Admin wird hierüber informiert und kann eine andere Datei auswählen.

Testfall 3.3.3: Datei existiert nicht.
Situation:

- Der Admin lädt einen Spielstand.
- Der Admin lädt eine Datei.
- Die Datei existiert nicht.

Erwartetes Ergebnis: Der Admin wird hierüber informiert und kann eine andere Datei auswählen.

Testfall 3.3.4: Datei nicht lesbar.
Situation:

- Der Admin lädt einen Spielstand.
- Der Admin lädt eine Datei.
- Die Datei kann nicht gelesen werden.

Erwartetes Ergebnis: Der Admin wird hierüber informiert und kann eine andere Datei auswählen.

Testfall 3.3.5: Ladevorgang abbrechen.
Situation:

- Der Admin lädt einen Spielstand.
- Der Admin bricht den Ladevorgang ab.

Erwartetes Ergebnis: Der Admin befindet sich wieder im Startdialog.

2.3.4 Use-Case 4: Spielzug durchführen

Testfall 4.1: ungültiger Spielzug.
Situation: Spieler ist am Zug
Erwartetes Ergebnis:

- Spielkontrolle verbietet Spielzug.
- Spielstein wird nicht gesetzt.
- Spieler wird aufgefordert einen gültigen Spielzug durchzuführen.

Testfall 4.1.1: ungültige Auswahl des Spielsteins.
Situation:

- Spielstein wird in ausgewähltes Feld gezogen.
- Eine Mühle wird dabei geschlossen.
- Ein eigener Spielstein soll entfernt.

Erwartetes Ergebnis:

- Spielkontrolle verbietet Spielzug.
- Spielstein wird nicht entfernt.
- Spieler wird aufgefordert einen gültigen Spielzug durchzuführen.

Testfall 4.2.1: Gültiger Spielzug, (n> 3 Steine, keine Mühle).
Situation:

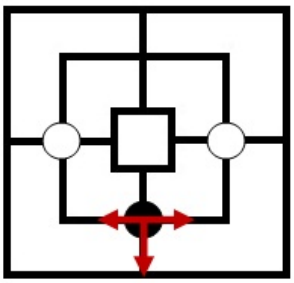

Abbildung 2.2: Testfall 4.2.1

Erwartetes Ergebnis: Spielstein wird in ausgewähltes Feld gezogen und Siegbedingung geprüft.

Testfall 4.2.2: Gültiger Spielzug, (n> 3 Steine, Mühle wird geschlossen, gegnerische Steine auch außerhalb einer Mühle vorhanden).
Situation:

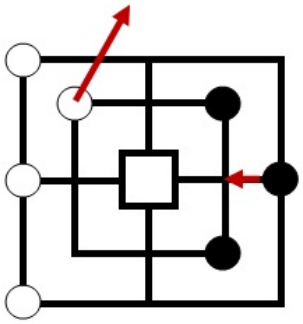

Abbildung 2.3: Testfall 4.2.2

Erwartetes Ergebnis:

- Spielstein wird in ausgewähltes Feld gezogen.
- Eine Mühle wird dabei geschlossen.
- Ein gegnerischer Spielstein, der nicht zu einer Mühle gehört, wird entfernt.
- Siegbedingung wird geprüft.

Testfall 4.2.3: Gültiger Spielzug, (n> 3 Steine, Mühle wird geschlossen, keine gegnerischen Steine außerhalb einer Mühle vorhanden).

Situation:

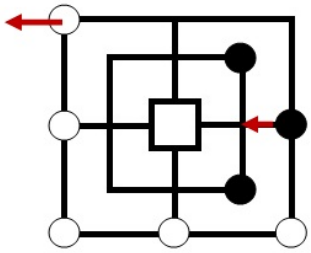

Abbildung 2.4: Testfall 4.2.3

Erwartetes Ergebnis:

- Spielstein wird in ausgewähltes Feld gezogen.
- Eine Mühle wird dabei geschlossen.
- Ein gegnerischer Spielstein wird entfernt.
- Siegbedingung wird geprüft.

Testfall 4.3.1: Gültiger Spielzug, (n= 3 Steine, keine Mühle).
Situation:

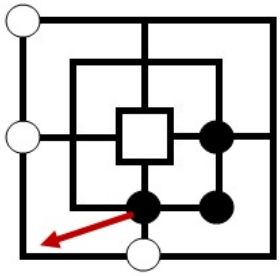

Abbildung 2.5: Testfall 4.3.1

Erwartetes Ergebnis: Spielstein springt in ein auswähltes Feld und Siegbedingung geprüft.

Testfall 4.3.2: Gültiger Spielzug, (n= 3 Steine, Mühle wird geschlossen, gegnerischen Steine außerhalb einer Mühle vorhanden).

Situation:

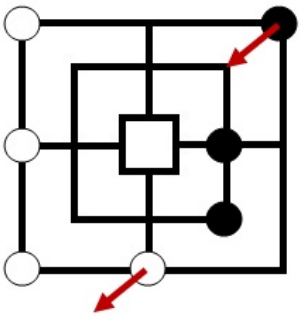

Abbildung 2.6: Testfall 4.3.2

Erwartetes Ergebnis:

- Spielstein springt in ausgewähltes Feld.
- Eine Mühle wird dabei geschlossen.
- Ein gegnerischer Spielstein, der nicht zu einer Mühle gehört, wird entfernt.
- Siegbedingung wird geprüft.

Testfall 4.3.3: Gültiger Spielzug, (n= 3 Steine, Mühle wird geschlossen, keine gegnerischen Steine außerhalb einer Mühle vorhanden).
Situation:

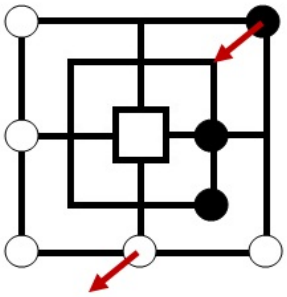

Abbildung 2.7: Testfall 4.3.3

Erwartetes Ergebnis:

- Spielstein springt in ausgewähltes Feld.
- Eine Mühle wird dabei geschlossen.
- Ein gegnerischer Spielstein wird entfernt.
- Siegbedingung wird geprüft.

2.3.5 Use-Case 5: Spielzug notieren

Testfall 5.1: Log öffnen.
Situation:

- Spiel im Startdialog.
- Der Admin öffnet das Log.

Erwartetes Ergebnis: Leeres Log wird angezeigt.

Testfall 5.1.1: Log öffnen nachdem ein Spielstand geladen wurde.
Situation:

- Der Admin lädt ein Spiel.
- Der Admin öffnet das Log.

Erwartetes Ergebnis: Das Log enthält den Verlauf des geladenen Spiels.

Testfall 5.2: Protokoll über das Spiel zweier KI-Spieler.
Situation:

- Der Admin startet ein neues Spiel zwischen zwei KI-Spielern.
- Die KI-Spieler ziehen bis einer gewinnt.
- Der Admin öffnet das Log.

Erwartetes Ergebnis: Das Log enthält N+2 Einträge, 1 über den Start des Spiels, N über die gemachten Züge, 1 über den Sieger des Spiels.

2.3.6 Use-Case 6: Spiel beenden

Testfall 6.1: Spiel beenden nach einem abgeschlossenen Spiel.
Situation: Das Spiel ist abgeschlossen, das Programm soll beendet werden.
Erwartetes Ergebnis: Das Programm wird beendet.

Testfall 6.2: Spiel beenden während das Spiel nicht abgeschlossen ist.
Situation:

- Spiel läuft.
- Der Admin beendet das Programm.
- Der Admin wählt speichern aus.

Erwartetes Ergebnis: Es erscheint ein Datei speichern Dialog.

Testfall 6.2.1: Speichern in nicht existierende Datei.
Situation:

- Spiel läuft.
- Der Admin beendet das Programm.

- Der Admin wählt speichern aus.
- Der Admin gibt einen Pfad zu einer nicht existierenden Datei an.

Erwartetes Ergebnis: Die Datei wird angelegt und mit dem aktuellen Spielstand befüllt. Das Programm ist beendet.

Testfall 6.2.2: Speichern in nicht existierenden Pfad.
Situation:

- Spiel läuft.
- Der Admin beendet das Programm.
- Der Admin wählt speichern aus.
- Der Admin gibt einen Pfad zu einer nicht existierenden Datei an.
- Das Verzeichnis des Pfades existiert nicht.

Erwartetes Ergebnis: Der Admin wird hierüber informiert und kann einen anderen Pfad auswählen oder die Datei überschreiben.

Testfall 6.2.3: Speichern in existierende Datei.
Situation:

- Spiel läuft.
- Der Admin beendet das Programm.
- Der Admin wählt speichern aus.
- Der Admin gibt einen Pfad zu einer existierenden Datei an.
- Der Admin lässt die Datei überschreiben.

Erwartetes Ergebnis: Die Datei wird geleert und mit dem aktuellen Spielstand befüllt. Das Programm ist beendet.

Testfall 6.2.4: Speichern in nicht schreibbare Datei.
Situation:

- Spiel läuft.
- Der Admin beendet das Programm.
- Der Admin wählt speichern aus.
- Der Admin gibt einen Pfad zu einer nicht existierenden Datei an.
- Das Ziel ist nicht schreibbar.

Erwartetes Ergebnis: Der Admin wird hierüber informiert und kann einen anderen Pfad auswählen.

Testfall 6.2.5: Abbrechen des Speichervorgangs.
Situation:

- Spiel läuft.
- Der Admin beendet das Programm.
- Der Admin wählt speichern aus.
- Der Admin bricht das Speichern ab.

Erwartetes Ergebnis: Das Programm ist beendet, es wurde nichts gespeichert.

2.4 Nicht-funktionale Anforderungen

Neben einigen funktionalen Anforderungen, soll das Projekt auch mehrere nicht funktionale Anforderungen erfüllen.
Die Softwarearchitektur soll so aufgebaut sein, dass sie folgende Bedingungen erfüllt:

- graphische Benutzungsschnittstelle.
- mindestens zwei Schichtenarchitektur: (GUI- und Anwendungsschicht).
- ausführbar aus einer jar-Datei.
- Spiel-Algorithmen sind als austauschbare Komponenten implementiert.
- genutzte Javaversion: jdk6.

Das Projekt soll zudem noch einige ergonomische Komponenten berücksichtigen:

- komfortables Setzen der Steine mit der Maus.
- Spielablauf durch Spieler/ Beobachter gut verfolgbar.
- Verfolgbarkeit gewährleistet durch Loggen der Spielzüge.

Optionale Anforderungen an das Projekt:

- Akzeptabel funktionierende künstliche Intelligenz.
- Laden/ Speichern bereits begonnerer Spiele.
- Optinale Verbesserung und Refactoring am implementierten GUI und Spiel.

Kapitel 3

Domänenmodell

In diesem Abschnitt wird das Domänenmodell vorgestellt. Es besteht aus 4 Domänen, die nachfolgend beschrieben sind.

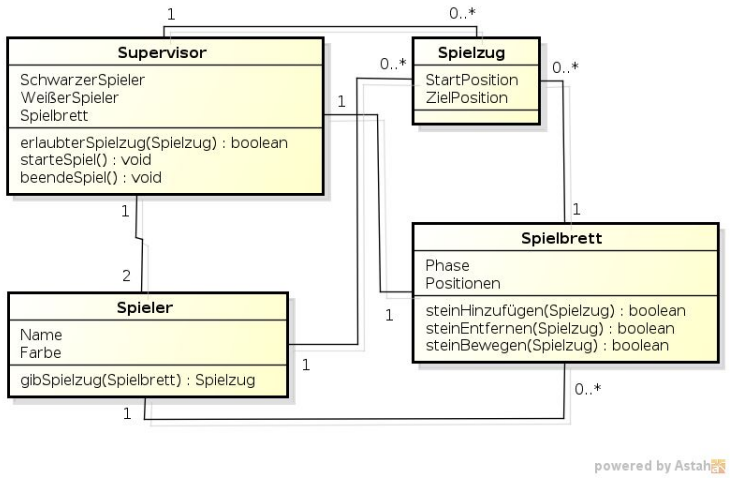

Abbildung 3.1: Domänenmodell

3.0.1 Spielzug

Der Spielzug ist der Bote zwischen den anderen Domänen, er ist eine reine Datentragende Klasse, die die Startposition und die Zielposition eines Steines definiert. Ist die Startposition ausserhalb des Feldes, so wird der Stein hinzugefügt. Ist die Zielposition außerhalb des Feldes, so wird der Stein entfernt.

3.0.2 Spielbrett

Das Spielbrett ist für den aktuellen Status des Spieles verantwortlich. Es weiß an welchen Positionen sich die Spielsteine befinden und muss über Änderungen an diesen informiert werden. Es ist ebenfalls dafür verantwortlich zu erkennen und zu speichern in welcher der drei Phasen sich das Spiel befindet.

3.0.3 Spieler

Der Spieler hat einen Namen und eine Farbe (Weiß oder Schwarz). Er ist ebenfalls dafür verantwortlich auf Anfrage neue Spielzüge zu generieren und erhält hierfür das aktuelle Spielbrett.

3.0.4 Supervisor

Der Supervisor ist dafür verantwortlich das Spiel zu initialisieren, zu beenden und zu überwachen. Er entscheidet ob ein Spielzug gültig ist und welcher Spieler aktuell einen Zug machen darf. Er ist auch der einzige der Änderungen am Spielbrett vornehmen darf. An der Phase in der sich das Spielbrett befindet erkennt er welche Aktion er als nächstes auslösen muss, also ob er das Spiel beenden soll oder den nächsten Spieler nach einem Zug fragen soll. Hierzu kennt er beide Spieler sowie das Spielbrett.

Kapitel 4

GUI-Prototyp

Dieser Abschnitt zeigt einen ersten Prototyp der graphischen Benutzeroberfläche. Es ist folgender Ablauf der einzelnen Komponenten angedacht. Der Ablauf wird zunächst unter Vorraussetzung eines menschlichen Spielers gezeigt. Die anderen Spielkombinationen sind dazu analog zu sehen. Zu Beginn öffnet sich das Startfenster mit einem leeren Spielfeld (siehe Abbildung 4.1).

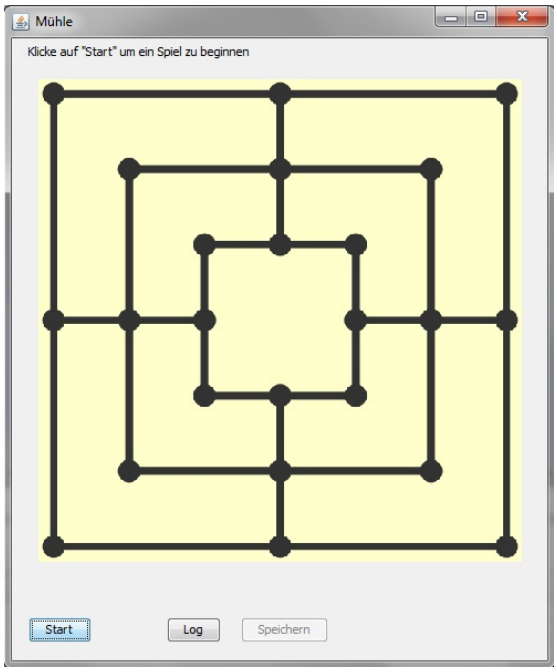

Abbildung 4.1: Spielfeld

Um ein Spiel zu starten, klickt man, wie in der Fensternachricht angegeben auf den Button „Start“, welcher ein weiteres Fenster öffnet. In diesem Fenster kann der Spieler nun festlegen, ob ein neues Spiel gestartet werden oder ein Spielstand geladen werden soll. Möchte der Spieler ein neues Spiel starten, so kann er in diesem Fenster die nötigen Einstellungen zur Spielerfarbe und KI-

Schwierigkeit einstellen. Eine optionale Eingabe für Spielernamen kann folgen (Abbildung 4.2).

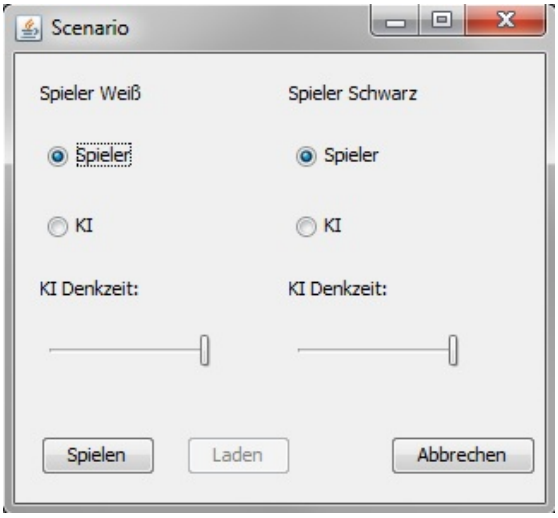

Abbildung 4.2: Fenster zum Initialisieren eines Spiels

Sind diese Einstellungen abgeschlossen kann der Spieler nun auf dem Spielfeld eine Partie Mühle spielen. Über die Schaltfläche „Log“ kann optional eine Übersicht über getätigte Züge angezeigt werden (Abbildung 4.3).

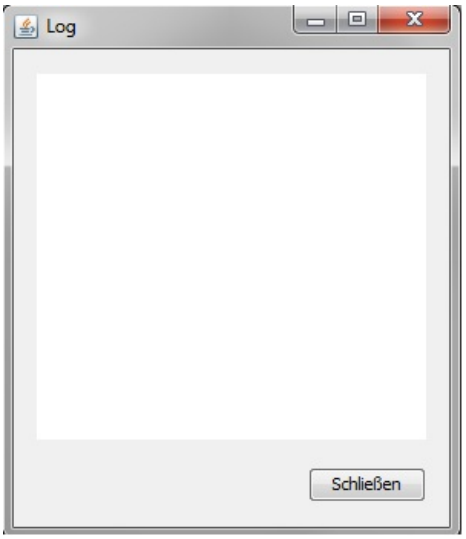

Abbildung 4.3: Logbuch für Spielzüge

Die Kombination aller 3 Fenster zum Spielstart sind in Abbildung 4.4 zu sehen:

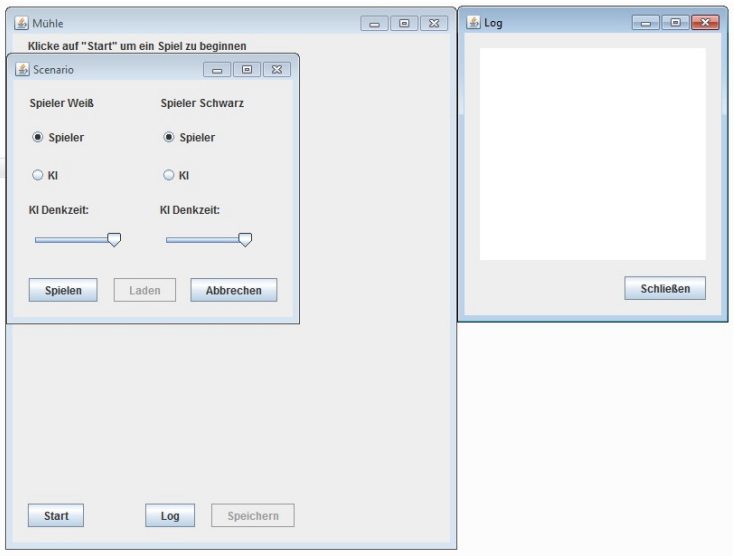

Abbildung 4.4: GUI-Prototyp zum Spielstart

Kapitel 5

Projektplan

Zunächst die Mitarbeiterübersicht:

- MA1: Jonne Haß
- MA2: Benjamin Held
- MA3: Richard Pump
- MA4: Alexander Sirotin
- MA5: Julian Haack

Zunächst die Ist-Stunden der einzelnen Mitarbeiter:

Name	Projektabschnitt	IST-Stunden
Jonne Haß	Planung	1
	Grundlegendes	38
	Qualitätssicherung	10
	Summe	**49**
Benjamin Held	Planung	1
	Grundlegendes	66
	Summe	**67**
Richard Pump	Planung	11
	Grundlegendes	42
	Projektleitung	10
	Summe	**63**
Alexander Sirotin	Planung	1
	Grundlegendes	24
	Summe	**25**
Julian Haack	Planung	1
	Grundlegendes	24
	Summe	**25**

Tabelle 5.1: IST-Stunden

Anschließend folgt der zu diesem Zeitpunkt aktuelle Projektplan:

Nr.	Arbeitspaket	Plan	Ist	Prognose	MA	Start	Ende
1	*Anforderungen an KI*	15	15	15		26.06.	08.10.
1.1	**Anforderungen**	5	5	5	Alle	26.09.	03.10.
	Ziele festlegen						
1.2	**Projektplan erstellen**	10	10	10	MA3	26.09.	05.10.
2	*Stufe 1: Grundlegendes*	273	183	289		26.09.	21.11.
2.1	**Analyse**	52	54	52		26.09.	08.10.
2.1.1	Use-Cases	12	12	12	MA2	26.09.	05.10.
2.1.2	Testfälle	16	22	16	MA1, MA2	26.09.	05.10.
2.1.3	GUI-Design	12	8	12	MA5	26.09.	05.10.
2.1.4	Domänenmodell	12	12	12	MA1	26.09.	05.10.
2.2	**Design**	60	109	99		10.10.	24.10.
2.2.1	Klassendiagramm GUI	14	15	14	MA1, MA5	10.10.	17.10.
2.2.2	Sequenzdiagramm GUI	6	6	6	MA1, MA5	17.10.	20.10.
2.2.3	Klassendiagramm	24	40	31	MA2, MA3	10.10.	17.10.
	Anwendungsschicht				MA5	17.10.	20.10
2.2.4	Sequenzdiagramm	12	44	44	MA2, MA3	17.10.	19.10.
	Anwendungsschicht				MA5		
2.2.5	Oberserver-Pattern	4	4	4	MA2	10.10.	17.10.
	jar-input Implementierung						
2.3	**Implementierung**	121	20	118		24.10.	13.11.
2.3.1	Logische Architektur/	46	17	48	MA2, MA3	24.10.	13.11.
	Kontrollklassen						
2.3.2	GUI	46	2	24	MA1, MA5	24.10.	13.11.
2.3.3	Einfache KI	24	-	44	MA4	24.10.	13.11.
2.3.4	Dokumentation pflegen	5	1	2	Alle	24.10.	13.11.
2.4	**Debugging/ Testen**	20	-	20		13.11.	16.11.
2.4.1	Testfälle durcharbeiten	8	-	8	MA1, MA2	13.11.	16.11.
2.4.2	Generelles Testen	12	-	12	Alle	13.11.	16.11.
2.5	**Puffer**	20	-	0		16.11.	21.11.
3	*Stufe 2: „Hohe Kunst“*	170	-	168		21.11	14.12.
3.1	**Analyse 2**	20	-	18		21.11.	24.11.
3.1.1	Überarbeitung von Analayse 1	16	-	14	Alle	21.11.	24.11.
3.1.2	Anforderungen an KI	4	-	4	MA3	21.11.	24.11.
3.2	**Design 2**	20	-	20		24.11.	27.11.
3.2.1	KI-Design	15	-	15	MA1, MA3	24.11.	27.11.
					MA2		
3.2.2	Schnittstellen-Integration	8	-	8	MA4, MA5	24.11.	27.11.
3.3	**Implementierung 2**	110	-	110		28.11.	17.12.
3.3.1	Implementierung der						
	Schnittstellen	48	-	48	MA4, MA5	28.11.	17.12.
3.3.2	KI	62	-	62	MA1, MA3	28.11.	17.12.
					MA2		
3.4	**Debugging/ Testen**	20	-	20		17.12.	21.12.
4	*Projektleitung*	28	3	25	MA3	26.09.	15.01.
5	*Qualitätssicherung*	28	3	25	MA1	26.09.	15.01.
6	*Puffer*	120	-	0		21.12.	15.01.
	Summen:	618	58	460			

Tabelle 5.2: Projektplanung/ Arbeitspakete

Kapitel 6

Design - Einleitung

Nach dem Fachkonzept für das Programmierprojekt WS12/13 folgt nun im nächsten Schritt das Design-Konzept mit den Klassendiagrammen und den dynamischen Modellen. Dabei soll das Projekt in zwei Schichten zerlegt werden. Einmal die Präsentationsschicht, die alle Methoden und Klassen der grafischen Benutzeroberfläche enthält und die Anwendungsschicht, die alle Servicemethoden und Attributsklassen enthält. Weiterhin wird in die Anwendungsschicht die sogenannte Fassade eingebettet, die für den Datenaustausch der beiden Schichten zuständig ist.
In den folgenden Kapiteln werden nun die beiden Schichten näher erläutert. Es werden die zugehörigen Klassendiagramme abgebildet und ausgewählte dynamische Prozesse in Prozessdiagrammen gezeigt. Des Weiteren wird jeweils das zu Grunde liegende Konzept der Methoden und der beiden Schichten näher beschrieben.
Auf die Fassade, als Verbindungselement der beiden Schichten, wird im Folgenden kurz eingegangen und ihr Konzept grob vorgestellt.

6.1 Vorgaben

Damit später die künstlichen Intelligenzen der verschiedenen Gruppen gegeneinander spielen können, müssen in der Anwendungsschicht 3 vorgegebene Interfaces implementiert sein:

- Das Interface Strategy, das einheitliche Methoden zum Durchführen eines Spielzuges enthält.
- Das Interface Slot, welches die Spielfelder auf dem Spielfeld repräsentiert.
- Das Interface Move, das einen Spielzug beschreibt.

Weiterhin soll sichergestellt werden, dass die anderen KIs mittels Reflection in das eigene Projekt eingelesen werden können.
Ein weiterer Punkt ist die Implementierung des sogenannten „Oberserver-Patterns“, welches die Unabhängigkeit der beiden Schichten sicherstellen soll. Details dazu folgen später.

6.2 Die Fassade als Verbindung der Schichten

Die Fassade bildet den zentralen Knoten zwischen der grafischen Benutzeroberfläche und der logischen Anwendungsschicht. Sie regelt die gesamte Kommunikation der beiden Schichten und gibt benötige Informationen zwischen denselben weiter. Die eigentlich Main-Methode des Projektes ist die Runner-Klasse, die

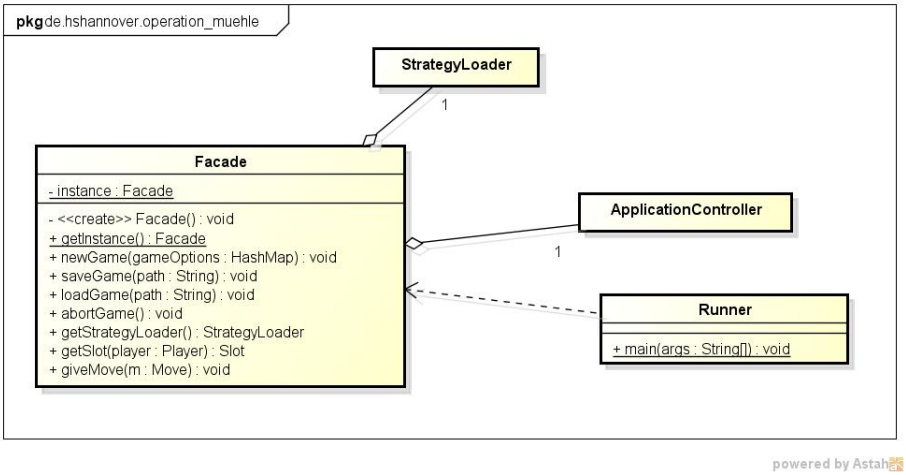

Abbildung 6.1: Modell der Fassade

die Fassade erzeugt. Die Fassade kennt nur den Anwendungscontroller, sie ist dem GUI-Controller allerdings bekannt. Sie dient dem GUI also als Oberfläche. Während die Kommunikation vom GUI über die Fassade zur Anwendungsschicht problemlos möglich ist, verläuft die Kommunikation in die entgegengesetzte Richtung anders. Für diese Richtung wird das vorgegebene „Observer-Pattern“ verwendet. Wenn sich in der Anwendung etwas verändert, wird die GUI-Schicht darüber informiert und mittels update-Methode werden die Daten an die grafische Benutzeroberfläche weitergegeben. Eine direkte Kommunikation von der Anwendung über die Fassade zur GUI ist daher im Bezug auf die Unabhängigkeit der Schichten nicht möglich.

Kapitel 7

Die Grafische Benutzeroberfläche

7.1 Klassendiagramm der GUI

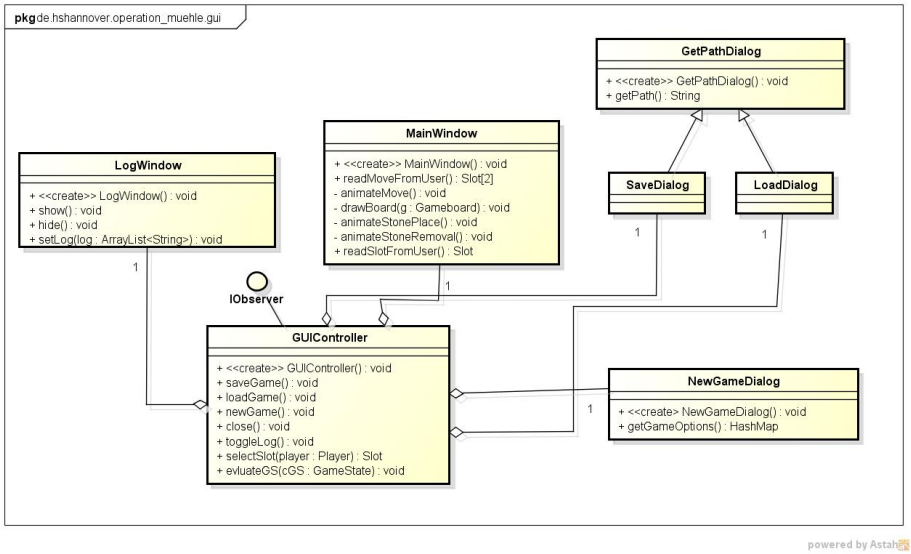

Abbildung 7.1: Die grafische Benutzeroberfläche

Das Klassendiagramm der grafischen Benutzeroberfläche ist abgetrennt von der Anwendungsschicht. Die Verbindung zur Fassade ist natürlich kenntlich. Die Fassade wird in Kapitel 6 näher beschrieben.

7.2 Konzept

In diesem Abschnitt werden die einzelnen Klassen und ihre Funktion innerhalb der GUI erläutert. Die GUI besteht aus einem Controller, der das Observer-

Interface implementiert, damit er sich beim ApplicationController als Observer registrieren kann. Ist ein Spieler an der Reihe, so erkennt der GUI-Controller dies und erstellt mithilfe des MainWindows einen Spielzug, welcher über die Facade and den ApplicationController gereicht wird. Im MainWindow wird das Spielbrett in der aktuellen Version, sowie eine Statusnachricht bezüglich des Spiels angezeigt. Zusätzlich existiert noch ein LogWindow, in welchem das Log des Spiels angezeigt wird, sofern gewünscht. Weiterhin gibt es noch Dialogfenster zum Erstellen eines neuen Spiels, Speichern eines momentan laufendem Spiels sowie Laden eines gespeicherten Spiels.

7.2.1 GUIController

Der GUIController ist der Dreh- und Angelpunkt der grafischen Benutzeroberfläche. Er ist dafür verantwortlich die Aktionen in den einzelnen Fenstern auszulösen, sie anzuzeigen, zu verstecken und Information von ihnen zu sammeln. Neben der Kommunikation mit der Fassade und den Fenstern, legt er diese beim Start des Programmes an und kennt diese daher auch. Sollte das Programm durch einen Benutzer beendet werden, sorgt der GUIController für eine saubere Beendigung der grafischen Schicht.

7.2.2 Das MainWindow

Dieses Fenster ist die Hauptinteraktionsfläche für den Benutzer. Von hier aus kann er ein neues Spiel starten, ein Bisheriges laden, im Spiel speichern, das Log aufrufen und das Programm beenden. Des Weiteren stellt es das Spielfeld dar und bietet dem Spieler die Möglichkeit seine Züge dem Spiel mitzuteilen. Hierzu wird es als Observer für das Gameboard in der Anwendungsschicht registriert und holt sich bei Änderungen die entsprechenden Informationen, um die entstandenen Änderungen grafisch umzusetzen.

7.2.3 Der „New Game“-Dialog

Dieser Dialog dient zum Konfigurieren eines neuen Spiels. Er fragt die verschiedenen Einstellungsmöglichkeiten ab und liefert diese dann an den Controller zurück, der über die Fassade ein neues Spiel in der Anwendungsschicht initialisiert. Die wichtigsten sind dabei:

- Name und Spielerart eines Spielers
- Schwierigkeitsgrad wenn eine AI verwendet werden soll

7.2.4 Der GetPathDialog und seine Kinder

Der SaveDialog und LoadDialog haben die gleiche Grundaufgabe: Es gilt jeweils einen Pfad abzufragen, an dem ein gespeicherter Spielstand vorhanden ist oder an den ein aktueller Spielstand gespeichert werden soll. In den Kindern wird also nur die Präsentation für den Benutzer entsprechend angepasst, um so die beiden Auswahlmöglichkeiten richtig darzustellen.

7.2.5 Das LogWindow

Das LogWindow ist ein einfaches Textfeld welches das Spiellog anzeigt. Hierzu wird in der Fassade das Logger-Objekt observierbar gemacht und das LogWindow als Observer dafür registriert. Wurden dem Logger nach einem erfolgreich durchgeführten Spielzug Informationen weitergegeben, wird das LogFenster darüber informiert und erhält diese Informationen zum Ausgeben an den Benutzer.

Kapitel 8

Die Anwendungsschicht

8.1 Klassendiagramm der Anwendungsschicht

Das Klassendiagramm der Anwendungsschicht ist abgetrennt von der Fassade und der grafischen Benutzeroberfläche dargestellt. Es zeigt die Abhängigkeit zur Fassade. Der genaue Aufbau der Fassade wird allerdings in Kapitel 6 näher beschrieben. Daher ist die Fassade im Klassendiagramm als einfache Klasse dargestellt.
Die Klasse AI der künstlichen Intelligenz ist ebenfalls nur als einfache Klasse dargestellt und zeigt hier lediglich die Abhängigkeit vom Strategy-Interface. Da für die Implementierung der künstlichen Intelligenz weitere Verweise und Klassen notwendig sind, ist für diese Klasse ein eigener Unterpunkt mit separatem Klassendiagramm im Abschnitt 8.4 zu finden.

8.2 Konzept

In diesem Abschnitt werden die einzelnen Klassen vorgestellt und ihre Funktionalität innerhalb der Anwendungsschicht beschrieben.
Die Anwendungsschicht besteht aus einem ApplicationController, der alle für das Spiel wichtigen Operationen ausführt. Er „besitzt“ zwei Objekte des Typs Player, bei denen es sich um Repräsentationen der am Spiel beteiligten Spieler handelt. Ein Player kann eine KI sein, die als Attribut des Players eingebunden wird. Außerdem besitzt der ApplicationController noch ein Gameboard, welches wiederum aus 24 Slots besteht und das Spielbrett wiederspiegelt. Auf dem GameBoard werden nur gültige Züge ausgeführt und abgespeichert, sodass es sich hierbei um die aktuellste Version des Spielbrettes handelt. Ein Logger speichert zusätzlich die Züge aller Spieler ab. Aus dem GameBoard, dem Logger und den Player-Objekten wird vom ApplicationController regelmäßig ein GameState-Objekt erstellt, welches für die Benutzeroberfläche bereitgehalten wird. Soll das Spiel gespeichert werden, wird die Struktur des GameStates benutzt und durch ein SaveState, welcher zusätzlich beide Spieler enthält, erweitert. Diese Repräsentation wird dann gespeichert und geladen.

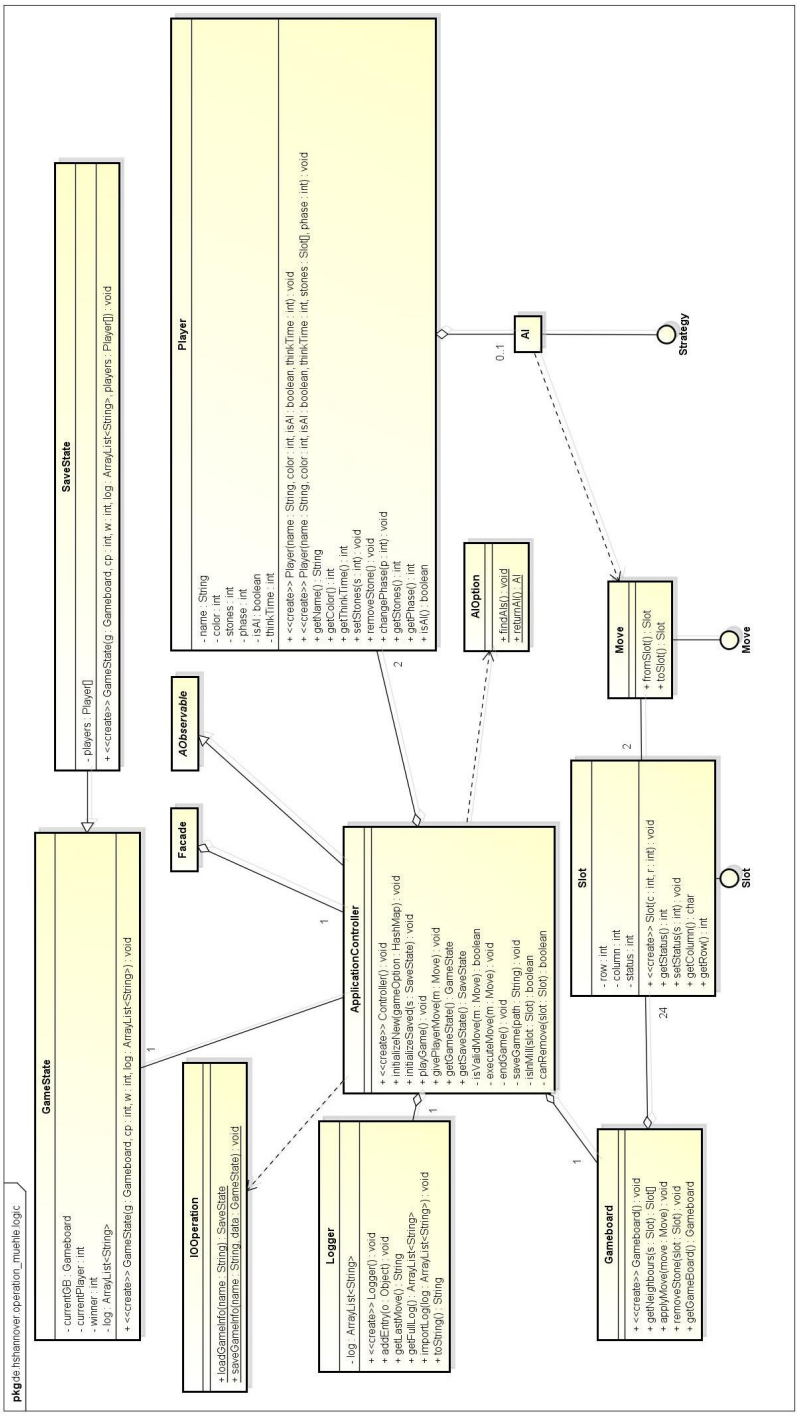

Abbildung 8.1: Klassendiagramm der Anwendungsschicht

8.2.1 ApplicationController

Die Klasse ApplicationController ist die zentrale Steuerklasse der Anwendungsschicht. Sie erzeugt alle notwendigen Objekte, die für das Ausführen eines Spiels notwendig sind. Sie ist dafür zuständig, je nach Eingabe die Vorbereitungen für ein neues Spiel anzulegen oder einen gespeicherten Spielstatus herzustellen. Weiterhin überwacht sie die Spielzüge der beiden Spieler, prüft diese auf ihre Regelkorrektheit und sammelt bzw. verteilt alle notwendigen Informationen, die benötigt werden. Die Fassade implementiert ein ApplicationController-Objekt der Anwendungsschicht.
Sie unterscheidet, ob es sich um menschliche oder computergesteuerte Spieler handelt, und besorgt bei menschlichen Spieler die relevanten Zugdaten aus der grafischen Oberfläche über die Fassade.

8.2.2 Player

Die Klasse Player ist als datentragende Klasse zur Repräsentation eines Spielers gedacht. Sie enthält die relevanten Daten, wie Name, Farbe und Spielsteine, sowie Informationen über die Spielphase, in der sich der Spieler befinden und ob es sich um einen Menschen oder um einen Computerspieler handelt. Ist das Player-Objekt ein Computerspieler implementiert die Player-Klasse ein entsprechendes Strategy-Objekt, welches die künstliche Intelligenz enthält, die die Spielzüge durchführt.

8.2.3 Logger

Die Klasse Logger ist für das Notieren der Spielzüge zuständig. Sie enthält eine Arraylist mit String-Objekten, die jeweils einen notierten Zug repräsentieren. Die weiteren Methoden sind zum notieren eines Zuges oder der Rückgabe des Zuglogs und zur Ausgabe in eine Datei oder dem Importieren aus einer Datei. Der Controller versorgt die Logger-Klasse mit den einzutragenden Spielzüge und gibt die Informationen zur Ausgabe im Logfenster weiter.

8.2.4 IOOperations

Die statische Klasse IOOperations enthält die Methoden, zum Einlesen eines Spielstandes bzw. eines Logs aus einer Datei und zum Ausgeben derselben Daten in eine Datei. Es handelt sich hier um eine reine Hilfsklasse für die Datenverwaltung.

8.2.5 Gameboard

Das Klasse Gameboard repräsentiert das Spielfeld. Sie besitzt eine Hashmap, die die Koordinatentupel (char, int) einem Spielfeld zuordnet, die ein Feld auf dem Spielfeld darstellen. Das Spielfeld kann zu einem gegebenen Slot die passenden Nachbarn suchen und zurückgeben.

8.2.6 Slot

Die Klasse Slot repräsentiert ein Spielfeld auf dem Spielbrett. Da die Spielfelder auf einem Raster ausgerichtet sind, enthält ein Slot als Koordiaten einen

Zeilen- und einen Spaltenindex. Die Klasse Slot implementiert das vorgegebene Interface Slot, welches get-Methoden für die beiden Indizes enthält. Als drittes Attribut enthält die Klasse einen Status, der angibt, ob sich ein Spielstein auf diesem Feld befindet und wenn ja, welche Farbe dieser hat. Die Klasse Slot implementiert für das Statusattribut eine get- und eine set-Methode.

8.2.7 Move

Die Klasse Move beschreibt einen Spielzug, der grundsätzlich aus zwei Slot besteht. Ein Slot, aus dem man einen Spielstein und einen weiteren Slot, in den der Stein gezogen wird. Die Korrektheit des Zuges ist in diesem Fall noch nicht relevant und wird vom Controller entsprechend überprüft. Die Klasse Move implementiert das vorgegebene Interface Move.

8.2.8 AOberservable

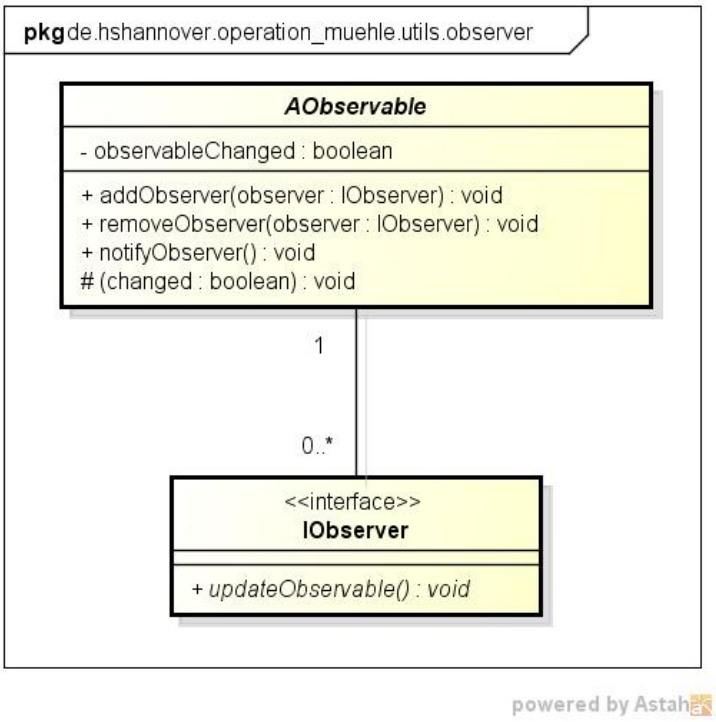

Abbildung 8.2: Diagramm: Observer

Die abstrakte Klasse AOberservable ist Teil des Observer-Patterns, welches zur Aktualisierung der Komponenten in der grafischen Benutzerschicht verwendet wird. Die Controllerklasse implementiert diese abstrakte Klasse, damit sie den GUI-Controller über Änderungen am Log, dem Spielfeld oder Informationen über einen Spielzug informieren kann und diese dann an die grafische Benutzeroberfläche weiterleiten kann.

8.2.9 AIOption

Diese Klasse implementiert Methoden, die das vorgegebene System der Reflection umsetzen lässt. Später soll das Projekt auch KIs andere Gruppen verwenden können bzw. gegen die eigene KI spielen können. Dazu muss der entsprechende Klassenpfad gesucht und eine Prüfung auf Kompatibilität durchgeführt werden. Danach kann die andere KI mittels des Strategy-Interfaces im eigenen Projekt genutzt werden.
Der Class-Loader sucht dabei nach den entsprechenden Intelligenzen und sammelt den Pfad, um damit vorhandene AIs laden zu können.

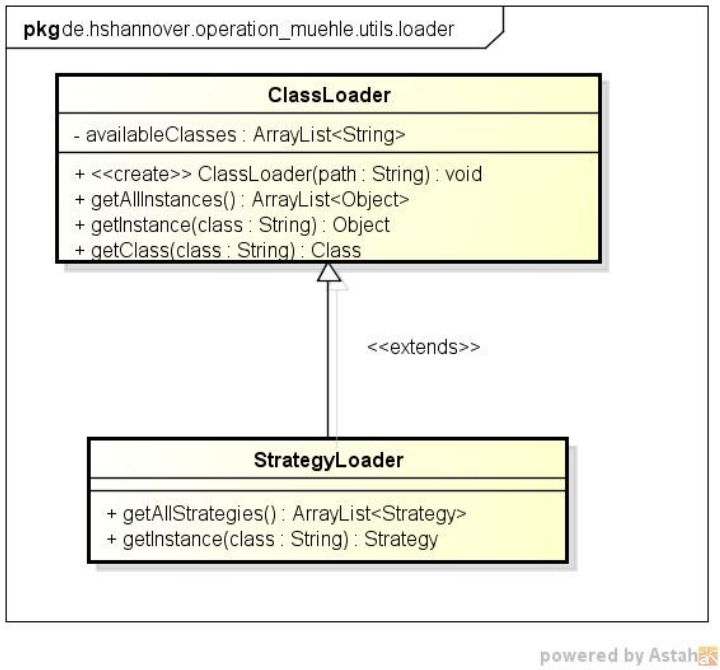

Abbildung 8.3: Diagramm: Laden einer Strategie

8.3 Die Klasse AI

Die KI besteht aus einer Hauptklasse AI, die das Strategy-Interface implementiert. Um das Spielgeschehen zu verfolgen, besitzt sie ein eigenes Gameboard, auf dem sie die Züge, die wirklich durchgeführt wurden, abspeichert. Um Spiele zu erstellen, besitzt sie einen MoveGenerator, der wiederum einen MoveEvaluator besitzt. Aufgabe des Evaluatores ist, aus der, auf den Zug folgenden Spielsituation, einen Wert zu berechnen, der wiederspiegelt, wie „gut“ dieser ist. Der MoveGenerator erstellt möglichst viele Züge, lässt diese evaluieren und gibt diese an die KI weiter, die dann diese auf Korrektheit überprüft und durchführt (siehe Sequenzdiagramme zu der künstlichen Intelligenz).

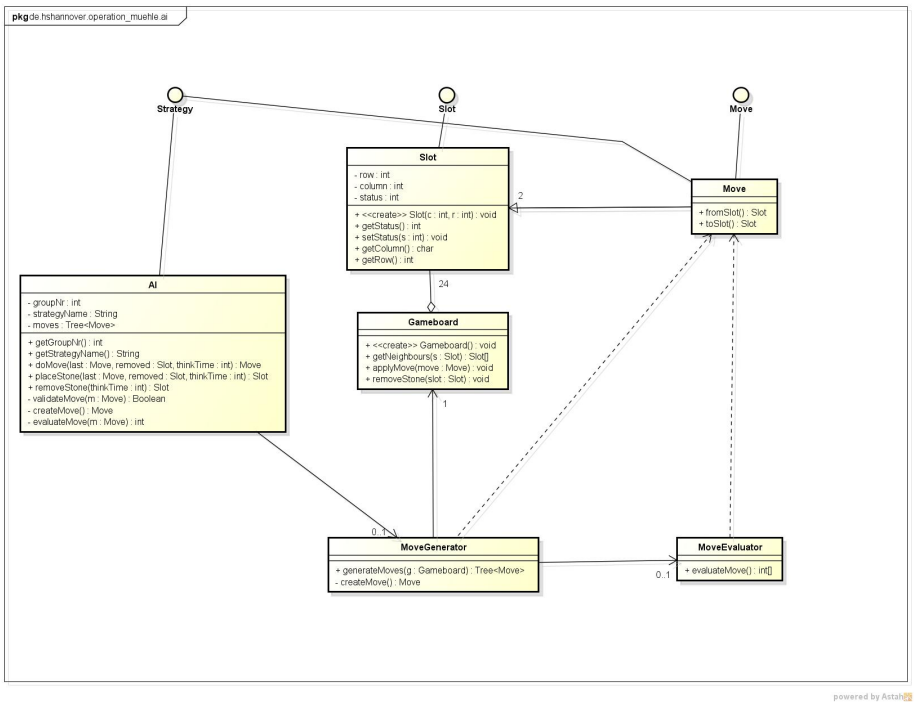

Abbildung 8.4: AI-Klasse und ihre Abhängigkeiten

Kapitel 9

Dynamische Modelle

In diesem Abschnitt werden die Sequenzdiagramme für spezielle Fälle der Use-Cases abgebildet. Es wird einmal die Initialisierung des Programms gezeigt, das sich dann aufteilt in das Anlegen eines neuen Spiels und dem Laden eines Spielstandes. Dann wird ein Spielzug abgebildet, bei dem durch den Spielzug eine Mühle geschlossen wird. Der Spielzug wird dann im Logger notiert, den Abschluss der an den Use-Cases orientierten Diagramme bildet dann das Schließen eines Spiels. Hierauf folgen zur weiteren Verdeutlichung Diagramme zur Initialisierung der KI sowie das Erstellen eines Moves durch diese.

9.1 Die Initialisierung

Die Initialisierung des Programmes beginnt im Runner, der eine Instanz der Fassade anlegt, welche wiederum den ApplicationController anlegt. Ausserdem erzeugt sie noch einen StrategyLoader. Ist dies geschehen, so erzeugt der Runner einen GUIController, der ein MainWindow und ein LogWindow anlegt, nachdem er sich beim ApplicationController als Observer registriert.

9.2 Anlegen eines neues Spiels

Um ein neues Spiel zu starten, meldet sich das GUI bei der Fassade, welche dem ApplicationController meldet, er solle ein neues Spiel starten (in From eines Aufrufes von initializeNew()). Dieser legt nun einen Logger, ein GameBoard (welches wiedrum 24 Slots anlegt) und zwei Spieler an. Handelt es sich bei einem Spieler um eine KI, so wird deren Name an den StrategyLoader geleitet, welcher die KI lädt und dem Spieler als Attribut liefert. Ist all dies geschehen, so wird dem Log der Eintrag „Game started“ übergeben, welcher den Startpunkt eines jeden Spiellogs darstellt. Zum Schluss ruft die Fassade die Methode playGame() auf, wodurch das eigentliche Spiel gestartet wird.

9.3 Laden eines Spiels

Ist ein gespeichertes Spiel zu Laden, so wird der in der Benutzeroberfläche angegebene Pfad vom GUI-Controller über die Facade weitergeleitet und der IO-

Operation übergeben, welche dann ein SaveState-Objekt aus einer Datei liest und an die Fassade gibt. Diese gibt es an den ApplicationController, welcher anhand des SaveStates die Spielsituation wiederherstellt.

9.4 Speichern eines Spiels

Soll das Spiel gespeichert werden, so benachrichtigt der GUI-Controller die Fassade, welche sich dann vom ApplicationController ein SaveState-Objekt erzeugen und übergeben lässt. Dieses Objekt wird dann an die IOOperation übergeben, welche dieses dann unter dem ‚aus dem GUI-Controller stammenden, Pfad abspeichert.

9.5 Darstellung eines Spielzuges

Soll ein Spielzug durchgeführt werden, so wird zuerst geprüft, ob es sich beim momentanen Spieler um eine KI handelt. Ist der aktuelle Spieler eine KI, so wird einfach doMove() auf aufgerufen, um einen Spielzug zu erstellen. Ist dies nicht der Fall, so ist die Spielzugerstellung etwas komplexer. Zuerst wird der Status des Booleans „Changed“ auf „true“ gesetzt, und der Observer benachrichtigt. Dabei handelt es sich um den GUI-Controller, welcher darauf den ApplicationController ein Gamestate-Objekt erstellen lässt, anhand dessen der GUI-Controller feststellt, was zu tun ist. Ist ein Spielzug gefordert, wird zuerst das dargestellte Spielbrett und Log aktualisiert. Danach wird ein Spielzug vom Benutzer über das MainWindow eingelesen, welcher dann durch die Fassade an den ApplicationController gereicht wird. Dieser überprüft dann, ob dieser gültig ist (wenn nicht wird ein neuer Spielzug angefordert) und ob eine Mühle geschlossen wurde. Sollte eine Mühle geschlossen werden, wird ähnlich wie schon zuvor, erneut die GUI benachrichtigt und statt eines Moves ein Slot mit der Position des zu entfernenden Steines zurückgeliefert. Ist all dies geschehen, wird das Gameboard des ApplicationControllers aktualisiert und die durchgeführten Aktionen werden in das Log geschrieben. Danach ist der nächste Spieler an der Reihe.

9.6 Initialisieren einer künstlichen Intelligenz

Soll die KI initialisiert werden, so erfolgt ein Aufruf des Standardkonstruktors über den StrategyLoader. Im Beispiel zu sehen ist die Initialisierung unserer KI, die stellvertretend für eine beliebige Andere, das Strategy-Interface implementierende KI, steht. Wird die Hauptklasse AI initialisiert, legt diese ein MoveGenerator-Objekt an. Dieses erzeugt ein MoveEvaluator-Objekt. Danach legt die KI ein Gameboard mit dessen 24 Slots an.

9.7 Spielzug der künstlichen Intelligenz

Um einen Spielzug zu erstellen, wird von der KI durch den MoveGenerator ein Baum mit Zügen erstellt. Jeder dieser Züge wird nach einer Heuristik durch den MoveEvaluator bewertet, sodass die KI den, für sich, Besten schnell findet.

Dieser Zug wird dann noch auf seine Korrektheit überprüft und anschließend zurückgegeben.

9.8 Spiel beenden

Um das Spiel zu beenden wird auf dem GUI-Controller, der Facade und dem ApplicationController die entsprechende Methode aufgerufen, die die Threads schließt.

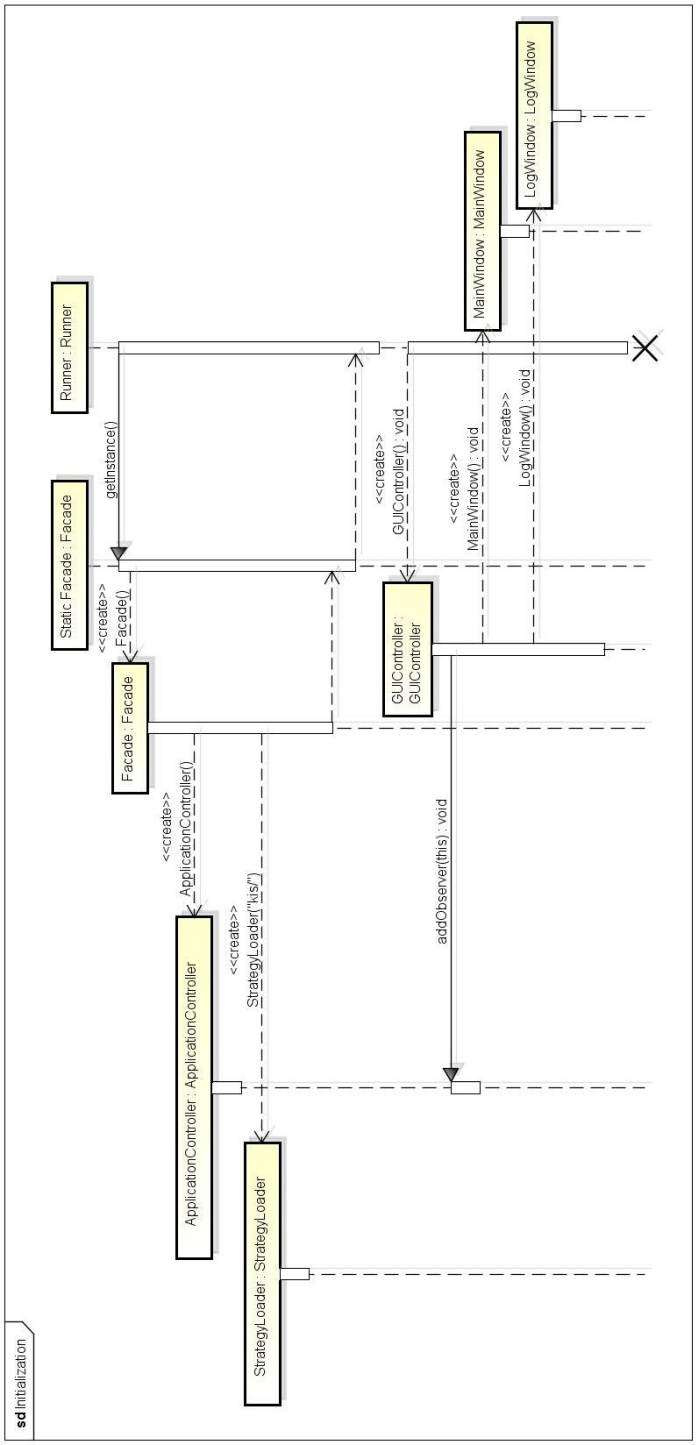

Abbildung 9.1: Sequenzdiagramm: Initialisierung

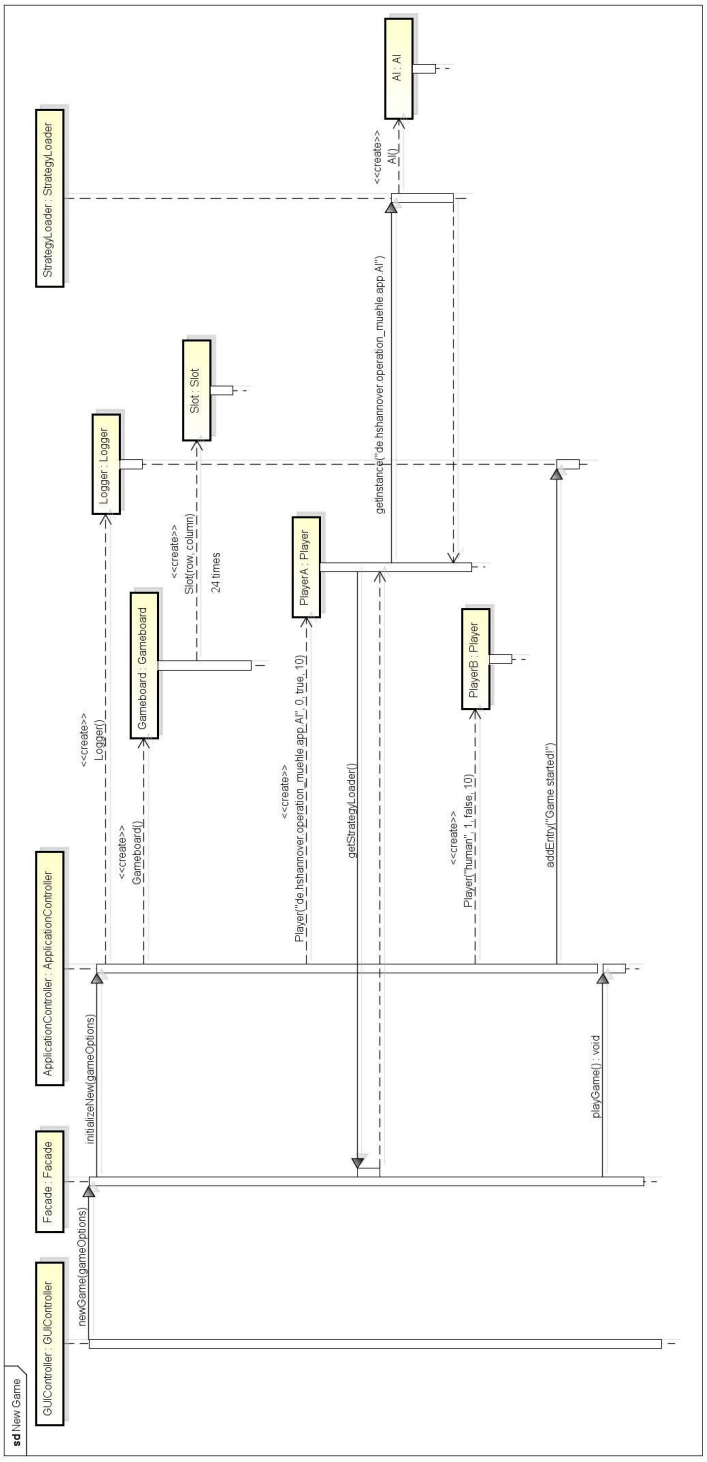

Abbildung 9.2: Sequenzdiagramm: Neues Spiel

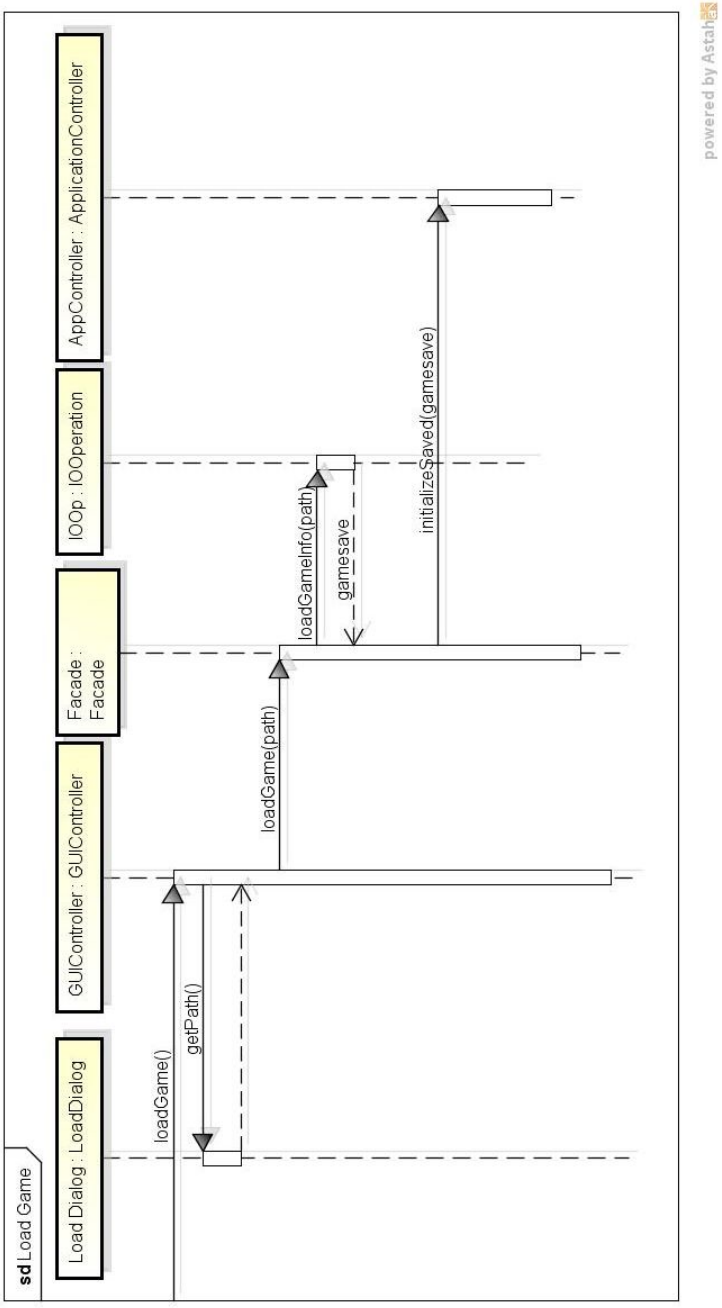

Abbildung 9.3: Sequenzdiagramm: Spiel laden

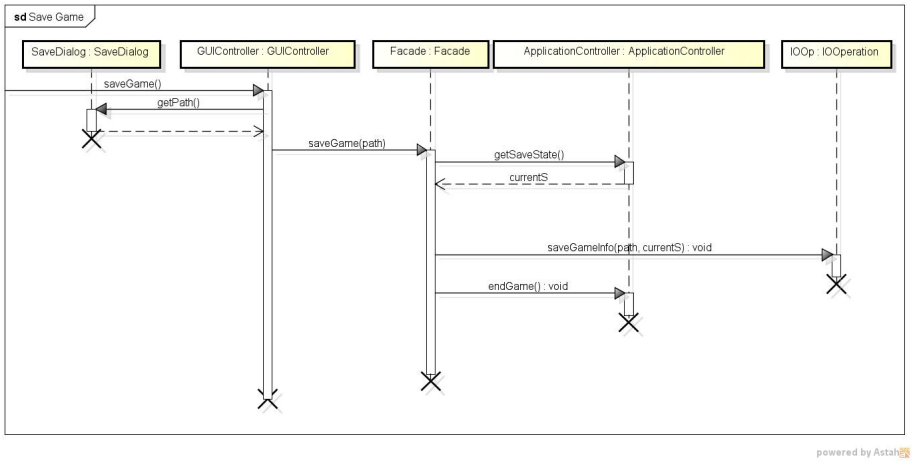

Abbildung 9.4: Sequenzdiagramm: Spiel Speichern

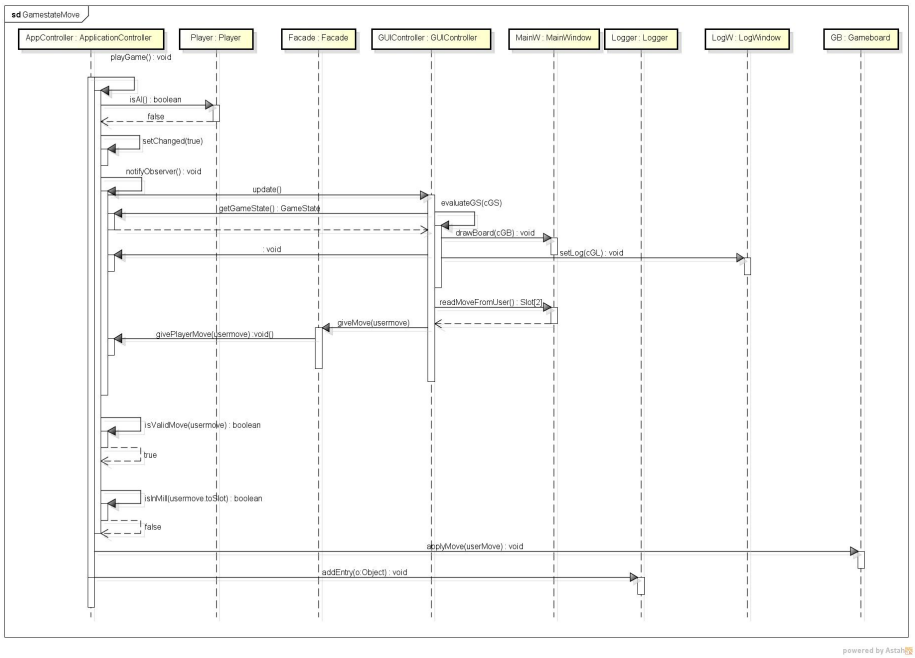

Abbildung 9.5: Sequenzdiagramm: Spielzug

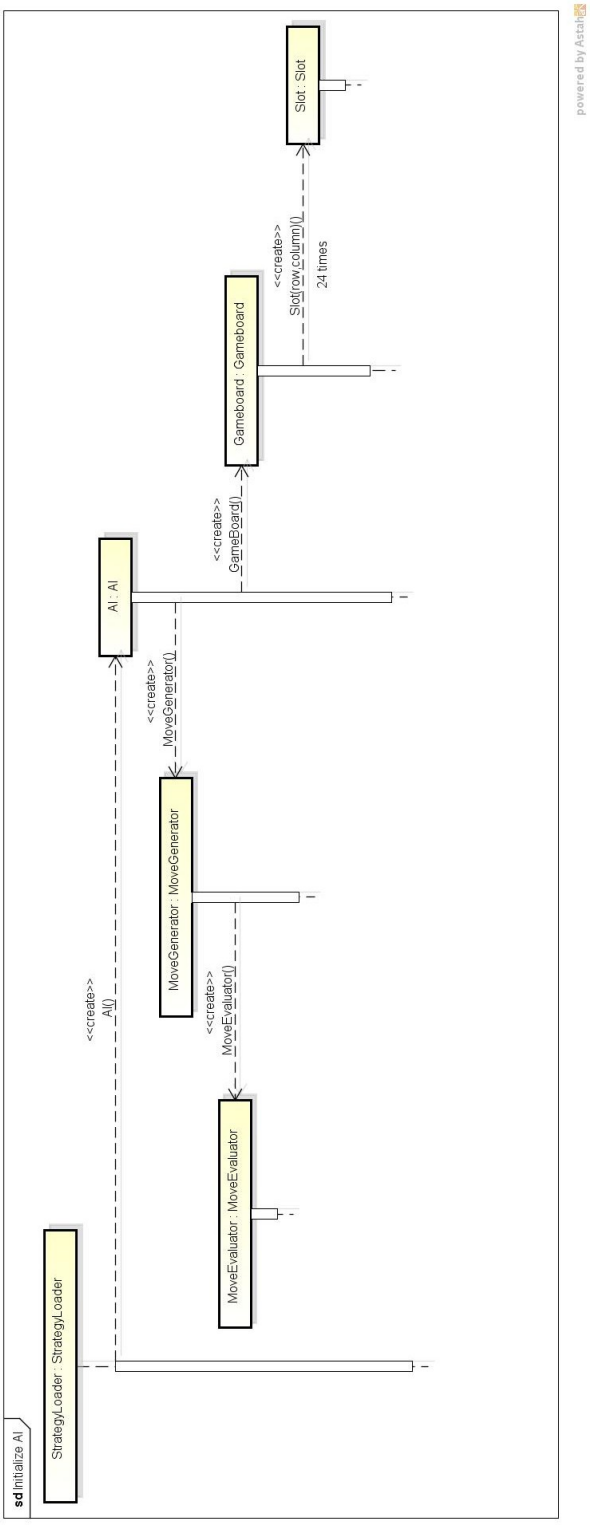

Abbildung 9.6: Sequenzdiagramm: Initialisieren der KI

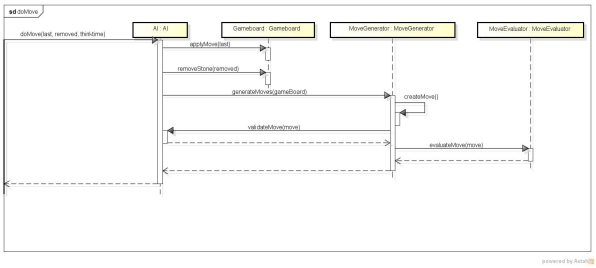

Abbildung 9.7: Sequenzdiagramm: Spielzugerstellung durch KI

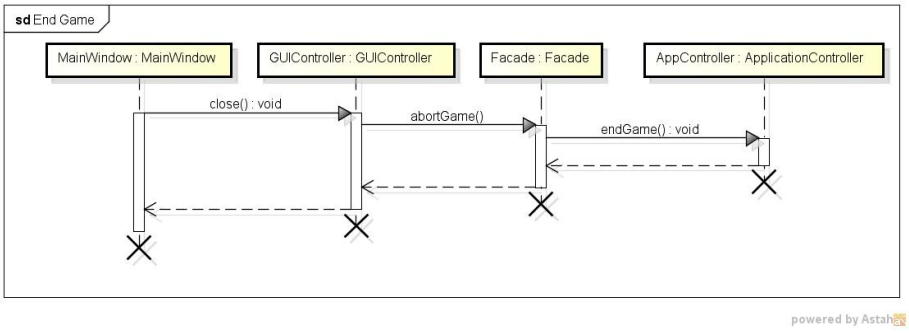

Abbildung 9.8: Sequenzdiagramm: Spiel beenden

Kapitel 10

Implementierung I

10.1 Einleitung

Nach der Anfertigung des Fach- und Designkonzeptes folgt nun die Implementierung der erstellten Ideen. Dazu werden die Klassen- und Sequenzdiagramme in Programmiercode umgesetzt und getestet. Die Tests orientieren sich an den Testfällen, die bereits in der Designphase zusammengestellt wurden. Die Konzepte aus der Designphase werden mit der Implementierung schrittweise aktualisiert, um die Änderungen während der Implementierung zu berücksichtigen. Dabei werden die Änderungen direkt in der dazugehörenden Kapiteln 7-9 umgesetzt. Die Dokumentation der Testfälle folgt im weiteren Verlauf. Die Aktualisierung des Projektplans erfolgt dagegen in Kapitel 5.

10.2 Ziele des ersten Implementierungsphase

In der ersten Implementierungsphase sollen folgenden Ziele erreicht werden:

- Einfaches, lauffähiges Programm
- Einfach künstliche Intelligenz
- Laden anderer KIs über die Schnittstelle
- Die eigene KI spiel gegen sich selbst oder eine andere eigene KI

10.3 Prüfen der Testfälle

In diesem Abschnitt werden die Ergebnisse des Tests festgehalten. Die Testfälle sind in Kaptitel 2.3 zu finden und werden an dieser Stelle nur knapp mit Referenznummer und Kurzbeschreibung übernommen!

10.4 Testfälle

Aus den eben beschriebenen Use-Cases werden nun verschiedene Testfälle abgeleitet, die während der Ausführung des Programmes auftreten können.

10.4.1 Use-Case 1: Initialisieren

Testfall 1.1: Spiel wird gestartet.
Testergebnis: noch nicht getestet.

10.4.2 Use-Case 2: Neues Spiel

Testfall 2.1: Neues Spiel erfolgreich starten.
Testergebnis: noch nicht getestet.

Testfall 2.1.1: Zwei menschliche Spieler.
Testergebnis: noch nicht getestet.

Testfall 2.1.2: Ein menschlicher Spieler, ein KI-Spieler.
Testergebnis: noch nicht getestet.

Testfall 2.1.3: Zwei KI-Spieler.
Testergebnis: noch nicht getestet.

Testfall 2.2: Neues Spiel nicht erfolgreich starten.
Testergebnis: noch nicht getestet.

10.4.3 Use-Case 3: Spiel laden

Testfall 3.1: Einen Spielstand laden.
Testergebnis: noch nicht getestet.

Testfall 3.2: Ein gespeichertes Spiel erfolgreich laden.
Testergebnis: noch nicht getestet.

Testfall 3.3: Ein gespeichertes Spiel nicht erfolgreich laden.
Testergebnis: noch nicht getestet.

Testfall 3.3.1: Inkompatible Datei laden.
Testergebnis: noch nicht getestet.

Testfall 3.3.2: Datei ist kein Spielstand.
Testergebnis: noch nicht getestet.

Testfall 3.3.3: Datei existiert nicht.
Testergebnis: noch nicht getestet.

Testfall 3.3.5: Ladevorgang abbrechen.
Testergebnis: noch nicht getestet.

10.4.4 Use-Case 4: Spielzug durchführen

Testfall 4.1: ungültiger Spielzug.
Testergebnis: noch nicht getestet.

Testfall 4.1.1: ungültige Auswahl des Spielsteins.
Testergebnis: noch nicht getestet.

Testfall 4.2.1: Gültiger Spielzug, (n> 3 Steine, keine Mühle).
Testergebnis: noch nicht getestet.

Testfall 4.2.2: Gültiger Spielzug, (n> 3 Steine, Mühle wird geschlossen, gegnerische Steine auch außerhalb einer Mühle vorhanden).
Testergebnis: noch nicht getestet.

Testfall 4.2.3: Gültiger Spielzug, (n> 3 Steine, Mühle wird geschlossen, keine gegnerischen Steine außerhalb einer Mühle vorhanden).
Testergebnis: noch nicht getestet.

Testfall 4.3.1: Gültiger Spielzug, (n= 3 Steine, keine Mühle).
Testergebnis: noch nicht getestet.

Testfall 4.3.2: Gültiger Spielzug, (n= 3 Steine, Mühle wird geschlossen, gegnerischen Steine außerhalb einer Mühle vorhanden).
Testergebnis: noch nicht getestet.

Testfall 4.3.3: Gültiger Spielzug, (n= 3 Steine, Mühle wird geschlossen, keine gegnerischen Steine außerhalb einer Mühle vorhanden).
Testergebnis: noch nicht getestet.

10.4.5 Use-Case 5: Spielzug notieren

Testfall 5.1: Log öffnen.
Testergebnis: noch nicht getestet.

Testfall 5.1.1: Log öffnen nachdem ein Spielstand geladen wurde.
Testergebnis: noch nicht getestet.

Testfall 5.2: Protokoll über das Spiel zweier KI-Spieler.
Testergebnis: noch nicht getestet.

10.4.6 Use-Case 6: Spiel beenden

Testfall 6.1: Spiel beenden nach einem abgeschlossenen Spiel.
Testergebnis: noch nicht getestet.

Testfall 6.2: Spiel beenden während das Spiel nicht abgeschlossen ist.
Testergebnis: noch nicht getestet.

Testfall 6.2.1: Speichern in nicht existierende Datei.
Testergebnis: noch nicht getestet.

Testfall 6.2.2: Speichern in nicht existierenden Pfad.
Testergebnis: noch nicht getestet.

Testfall 6.2.3: Speichern in existierende Datei.
Testergebnis: noch nicht getestet.

Testfall 6.2.4: Speichern in nicht schreibbare Datei.
Testergebnis: noch nicht getestet.

Testfall 6.2.5: Abbrechen des Speichervorgangs.
Testergebnis: noch nicht getestet.

Abbildungsverzeichnis

Tabellenverzeichnis