JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von

Felix Stadler k12043299
Simon Ulmer k12043331
Dominik Niederberger
k12111671

Gruppe 5

Angefertigt am

Institut für
Wirtschaftsinformatik –
Software Engineering

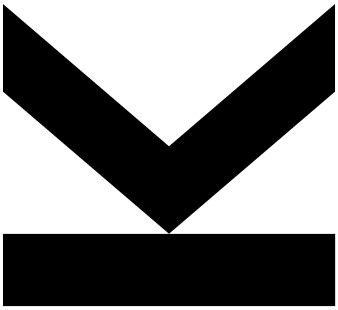
Beurteiler / Beurteilerin

Dr. Johannes Sametinger

Datum

06.2023

GOGAME APPLIKATION



Systemdokumentation

PR Software Engineering

259.999
SS 2022

JOHANNES
KEPLER

Kurzfassung

GoGame ist eine Java Applikation für 2 Spieler. Sie wurde im Rahmen der Lehrveranstaltung Praktikum Software Engineering auf der Johannes Kepler Universität Linz entwickelt.

Dieses Dokument gilt als Systemdokumentation. Darin werden die verschiedenen Komponenten des Systems erläutert, darunter die Klasseneinteilung, die zugrunde liegende Architektur des Systems und die verwendeten Entwurfsmuster und -prinzipien.

JOHANNES KEPLER UNIVERSITÄT LINZ

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Diese Systemdokumentation bietet einen umfassenden Überblick über das Go-Softwareprojekt und dient als Leitfaden für Entwickler und Wartungspersonal. Sie enthält detaillierte Informationen über die Architektur, Funktionalität und Implementierung des Systems. Es werden Kenntnisse über Java und die Verwendung von Git vorausgesetzt.

2. Architektur

Das Softwareprojekt basiert auf dem Software-Design-Pattern Model-View-Controller (MVC). Dieses Entwurfsmuster soll zu einer übersichtlichen Trennung der Strukturen im System beitragen.

In den folgenden Kapiteln werden die einzelnen Komponenten der Entwurfsmuster beschrieben. Jede Komponente wurde im Code in ein eigenes Package gegliedert.

2.1. Model (MVC)

Im Model werden die Daten und die Logik des Spiels repräsentiert. Dadurch kann die Grundeinstellung des Spielbrettes eingestellt, der derzeitige Spielstand abgefragt und Berechnungen bei neuen Spielzügen eingeleitet werden

2.2. View (MVC)

Die View ist für die Darstellung der Benutzeroberfläche (UI) verantwortlich. Sie zeigt die Daten aus dem Model an und ermöglicht dem Benutzer die Interaktion mit der Anwendung. Somit kann das Spiel graphisch dargestellt werden und z.B. Spielzüge gemacht werden.

2.3. Controller (MVC)

Der Controller steuert den Datenfluss zwischen dem Model und der View. Er behandelt die Benutzerinteraktionen und handelt entsprechend, indem er das Model aktualisiert oder die View anweist, Änderungen vorzunehmen. Im Controller werden zum Beispiel die Koordinaten eines gewünschten Spielzuges am Spielbrett in die Koordinaten für das Model umgewandelt.

2.4. Listener

Zudem wurde zusätzlich das Listener / Observer Entwurfsmuster eingesetzt, um Komponenten über diverse Änderungen im Model zu benachrichtigen. Jegliche View-Komponente implementiert das Interface GameListener, um auf die verschiedenen Spielstatus zu reagieren und dementsprechende Aktualisierungen vorzunehmen.

2.5. Zusammensetzung

Um den Zusammenhang der einzelnen Komponenten des MVC-Patterns zu vereinheitlichen, wurde über eine standardisierte Definition der Konstruktoren entschieden. Als Startpunkt der Struktur gilt das Model. Dieses wird als Übergabe-Parameter in View-Klassen verwendet.

Sollte eine View-Klasse einen Controller zur Kommunikation mit dem Model benötigen, wird dieser im Konstruktor der View erstellt. Der Controller übernimmt im Konstruktor die derzeitige View und das Model. Somit sind die Komponenten Model, View und Controller miteinander verbunden.

Für die Anbindung von Listener an das Model wird die Methode addGameListener bereitgestellt. Klassen, welche das Interface GameListener implementieren, können somit in eine Liste des Models eingetragen werden, welche bei Änderungen benachrichtigt wird.

2.6. UML-Klassendiagramm

In der folgenden Abbildung wird das UML-Klassendiagramm der entwickelten Go-Applikation. Um die Übersichtlichkeit im Diagramm zu erhöhen, werden bei den 1:1 Beziehungen die Kardinalitäten weggelassen und nur bei 1:* oder *:* dargestellt.

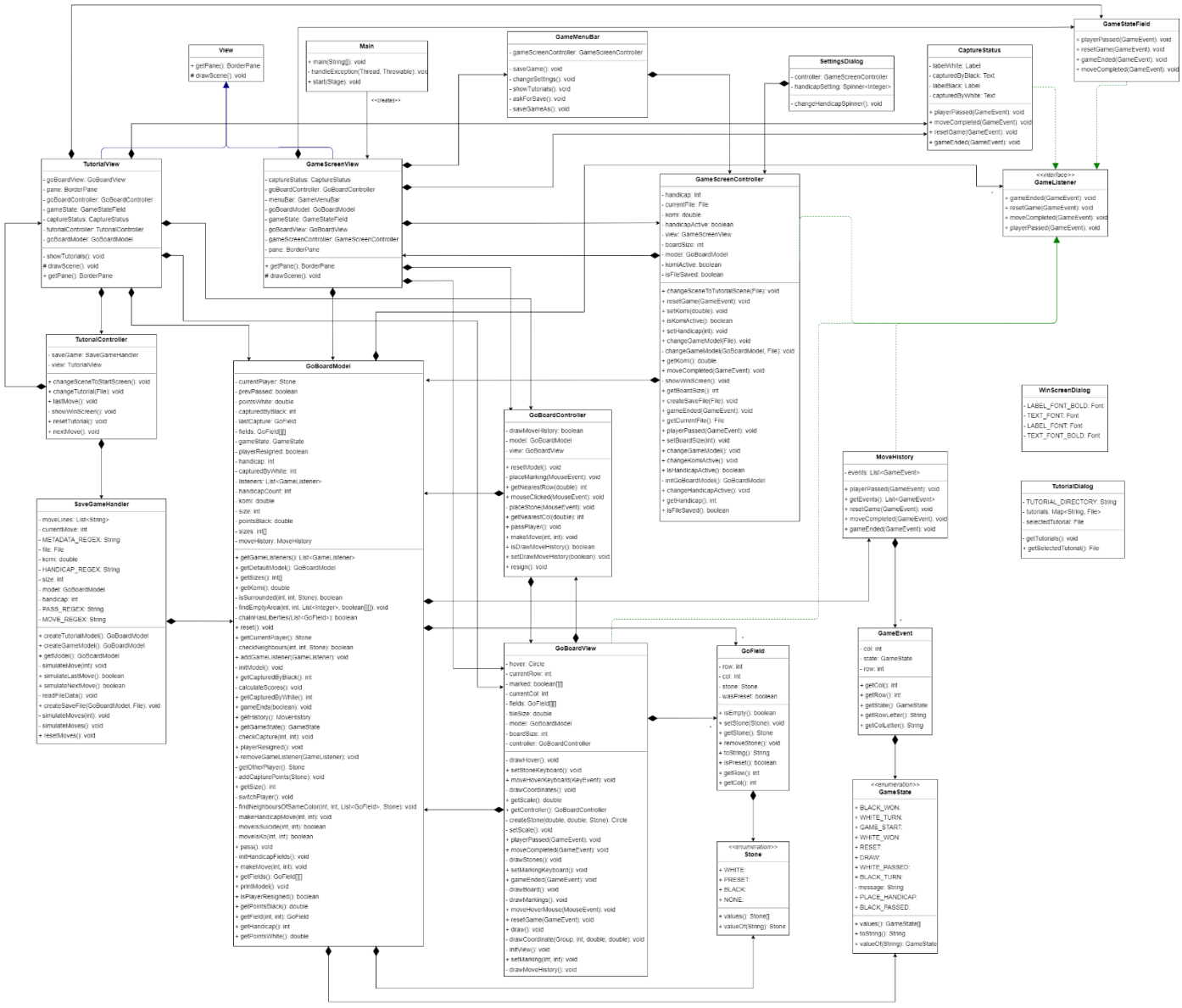


Abbildung 1: UML-Diagramm

3. JavaDoc

Der Code des Softwaresystems wurde mittels JavaDoc dokumentiert. Dies soll zu einer Code-nahen Dokumentation beitragen, was zukünftige Entwicklungen erleichtern soll. Die JavaDoc Dokumentation ist im HTML-Format beigelegt. Sämtliche weitere begleitende Dokumentation zum System ist aus diesem Dokument zu entnehmen.

In den folgenden Kapiteln werden begleitend sämtliche Klassen nach Packages gegliedert erläutert.

4. Model (Package)

Das Model besteht aus der Klasse GoBoardModel, welche die Klassen GoField und Stone verwendet. Ein GoBoardModel beinhaltet ein zweidimensionales Array aus GoField Instanzen. Jedes GoField beinhaltet eine Instanz des ENUM Stone.

4.1. GoBoardModel

Die Klasse GoBoardModel stellt die Gesamtheit des Models dar. In der Klasse werden die Einstellungen des Brettspiels gespeichert. Zudem wird der derzeitige Spielstand dargestellt und Berechnungen zu Spielzügen gemacht. Es werden öffentliche Methoden zur Interaktion mit dem Spielfeld und mehrere Getter / Setter zur Verfügung gestellt.

Je nach übergebener Größe an das Model werden verschiedene Kompositionen der GoField Objekte erstellt. Dies hat zum Beispiel eine Auswirkung auf die Platzierung der Handicap Steine und geschieht in den privaten Methoden initModel() und initHandicapFields(). Nach dem setzen eines Steines mittels makeMove() wird die Methode checkCapture() aufgerufen, um die Freiheiten der annähernden Positionen zu prüfen. Mittels den Methoden moveIsSuicide() und moveIsKo() wird außerdem die Gültigkeit eines Spielzugs geprüft. Am Ende des Spiels wird mittels der privaten Methode calculateScores() der jeweilige Punktestand der Spieler berechnet. Dies geschieht anhand von Komi, Territorium und gefangenen Steinen.

4.2. GoField

In der Klasse GoField werden Informationen zum jeweiligen Spielfeld gespeichert. Neben der Position am Spielbrett wird auch der derzeitige Stein gespeichert. Über die Schnittstelle der Klasse können Statusinformationen abgefragt und der gesetzte Stein geändert werden.

4.3. Stone

Die ENUM-Klasse Stone stellt die möglichen Steine auf dem Spielbrett dar. Es gibt Optionen für Schwarz, Weiß, Handicap-Felder und leere Felder.

5. View (Package)

Die View Komponenten wurden in drei Bereiche aufgeteilt. Das Spielbrett selbst wird durch die Klasse GoBoardView dargestellt. Die Hauptansicht der Applikation befindet sich in der Klasse GameScreenView. Für die Anzeige von Problemstellungen / Tutorials wird die Klasse TutorialView verwendet. Sämtliche View-Klassen implementieren das Interface GameListener um auf Aktualisierungen im Model zu reagieren.

5.1. View (Klasse)

Klassen, welche als Szene für JavaFX verwendet werden sollen, erweitern die abstrakte Klasse View. Die Klasse soll standardisierte Methoden zum Übergeben der Root-Node als auch zum erstmaligen Zeichnen dieser zur Verfügung stellen.

5.2. GoBoardView

In der Klasse GoBoardView wird das Spielfeld als Erweiterung der Klasse Pane dargestellt. Die Größe des Spielfeldes wird in der Methode initView() definiert. Die Klasse verfügt über die double Variable tileSize, welche die Größe eines Feldes in Pixeln beschreibt. Bei Änderungen der Fenstergröße wird diese Variable aktualisiert, um die Größe des Brettes zu ändern.

Die Klasse verwaltet auch das Zeichnen des „Hover-Steines“ welcher die nächste Position laut Maus oder Tastatur angibt. Der Hover-Stein ist in der Farbe des jeweiligen Spielers eingefärbt und wird rot eingefärbt, wenn ein gewähltes Feld nicht gültig ist.

Es können auch Positionen mittels Rechtsklick oder Tastatureingabe markiert werden. Diese Markierungen werden, so wie der Hover-Stein, nur im GoBoardView angezeigt.

Sollte die Historie der vergangenen Spielzüge angezeigt werden, zeichnet die Klasse diese mittels Nummerierung der letzten Events in auf die gesetzten Steine.

Um die Interaktionsmöglichkeiten von der Klasse zu entkoppeln, wurden keine Eventhandler eingebaut. Diese werden je nach Anwendungsfall (Spiel oder Tutorial) in der Klasse, welche den GoBoardView enthält, eingefügt.

5.3. GameScreenView

Die Klasse GameScreenView stellt den Hauptinteraktionspunkt für den Benutzer da. Sie beinhaltet eine Instanz des GoBoardView und weitere UI-Komponenten zur Visualisierung und Einstellung des Spieles. Der GameScreenView erweitert die abstrakte Klasse View und setzt die Eventhandler, welche für die Interaktion mit dem Spielbrett nötig sind.

Durch den GameScreenView werden Klassen wie GameMenuBar, GameStateField und CaptureStatus instanziiert, um Änderungen im Spiel darzustellen.

5.4. TutorialView

Die Klasse Tutorialview ist die zweite Erweiterung der Klasse View und soll dem Benutzer die Interaktion mit Problemstellungen / Tutorials ermöglichen. Sie setzt ebenfalls Eventhandler zur Interaktion mit dem Spielbrett und instanziiert auch Klassen wie GameStateField und CaptureStatus.

5.5. CaptureStatus

Die Klasse CaptureStatus stellt die Punkte durch Gefangennahme von gegnerischen Steinen Visuell dar. Sie implementiert GameListener und aktualisiert den Spielstand nach Beendigung eines Zuges.

5.6. GameMenuBar

Die Klasse GameMenuBar ist eine Menüleiste durch die diverse Interaktionsmöglichkeiten mit dem System bereitgestellt werden. Mittels der Menüleiste können Spiele gespeichert, geöffnet oder neu gestartet werden. Interaktionen mit dem Spiel wie Passen, Aufgeben oder die Anzeige der Spielhistorie sind auch möglich. Es können die Speileinstellungen geändert und Tutorials ausgewählt werden. Ebenfalls gibt es eine textuelle Repräsentation der Steuerung und ein „About us“ Fenster.

5.7. GameStateField

Das GameStateField erweitert die Klasse TextField und zeigt den Spielstatus Textuell an. Es übernimmt den Text der GameEvent Instanz und stellt je nach Spielereignis diese Informationen bereit.

5.8. SettingsDialog

Die Klasse SettingsDialog dient zur Änderung der Spieleinstellungen in Bezug auf Spielgröße, Handicap und Komi. Durch die Instanziierung wird ein Dialog dargestellt, mit dem die möglichen Spieleinstellungen getroffen werden können. Durch eine Bestätigung wird das neue Spiel geladen.

5.9. TutorialDialog

Der TutorialDialog ist ein Dialog, der es erlaubt, aus vorgespeicherten Tutorials zu wählen und diese zu Betrachten. Die Tutorials, welche sich im Ressourcen-Ordner befinden, werden angezeigt und nach einer Bestätigung geladen. Die Anzeige passiert durch einen Wechsel in den TutorialView.

5.10. WinScreenDialog

Die Klasse WinScreenDialog dient zur Anzeige des Endstandes des Spiels. Die Punkte werden je Spieler in Capture-, Territoriums- und Komi-Punkte aufgelistet. Wen ein Spieler aufgibt, ist der andere Spieler der Sieger.

5.11. Ablaufdiagramm

Die folgende Abbildung stellt den möglichen Ablauf zwischen den verschiedenen Szenen (Panes und Dialogs) im Spiel dar. View-Komponenten werden mit einer durchgehenden Linie angezeigt, während Dialoge mit strichlierten Linien dargestellt werden. Die Pfeilrichtungen stellen die möglichen Interaktionswege des Benutzers zwischen den jeweiligen Komponenten dar.

In der Abbildung wir dabei auch das Zusammenspielen der Komponenten des Spielbretts dargestellt. Das Spielbrett (GoBoardView) ist als eigene Komponente jeweils in der übergeordneten View Instanz integriert. Je nach Spielmodus – Tutorial (TuorialView) oder Nomales Spiel (GameScreenView) – ist die Interaktion mit dem Spielbrett unterschiedlich.

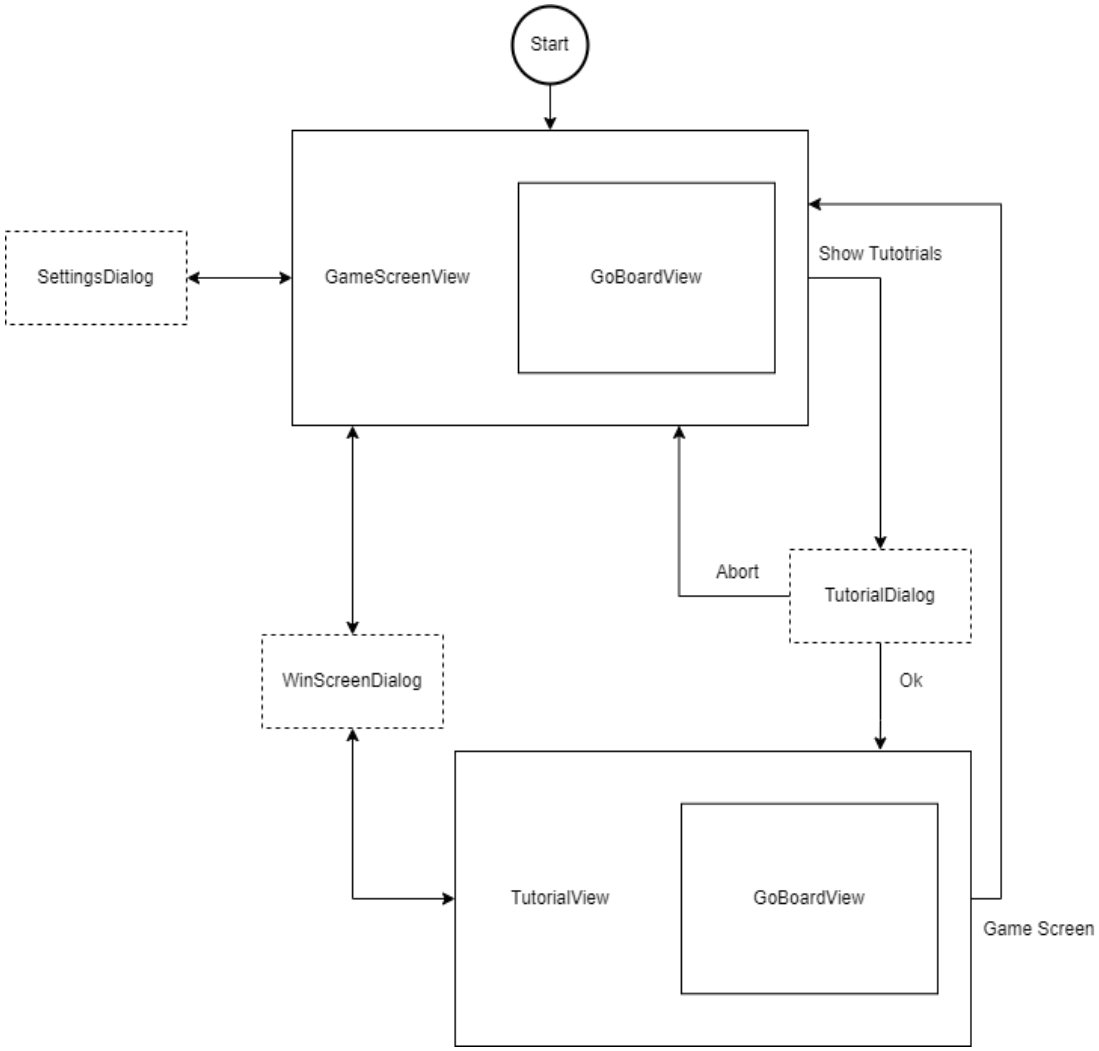


Abbildung 2: Ablaufdiagramm

6. Controller (Package)

Im Softwaresystem werden drei Controller eingesetzt. Ein Controller für den GoBoardView und ein Controller jeweils für den GameScreenView und den TutorialView.

6.1. GoBoardController

Der GoBoardController rechnet Events des GoBoards von Pixelgrößen in die Array-Indizes des GoBoardModels um. Dabei werden Klassen bereitgestellt, die den nächsten Index zur derzeitigen Mausposition ermitteln. So können Spielzüge an das Model weitergeleitet und zum Beispiel Markierungen und Hover-Steine dargestellt werden.

6.2. GameScreenController

Der GameScreenController verwaltet die Funktionen, die außerhalb des Spielbrettes getätigt werden können. Im Controller wird vermerkt, ob das derzeitige Spiel gespeichert ist, was Auswirkungen auf die dementsprechenden Optionen wie „Save“ oder „Change Settings“ hat.

Ebenfalls verwaltet der Controller das Wechseln der JavaFX Scene bei neuen Spielen oder dem Start eines Tutorials. Bei Beendigung des Spiels ruft der GameScreenController zudem den WinScreenDialog auf.

6.3. TutorialController

Die Klasse Tutorialcontroller verwaltet die Interaktion mit Tutorial-Dateien. Je nach Interaktion wird der nächste oder letzte Zug simuliert. Mit Hilfe des Controllers können auch neue Tutorials gewählt oder die JavaFX Scene zurück zum GameScreenview gewechselt werden.

7. Listener (Package)

Das Package Listener besteht aus dem Interface GameListener, der Klasse GameEvent und dem ENUM GameState. Diese verwalten das Handling von Aktualisierungen des Models.

7.1. GameListener

Das Interface GameListener ist eine Erweiterung des Interface EventListener und soll eingesetzt werden, um View-Komponenten je nach Veränderung des Models zu aktualisieren. Dafür werden vier Methoden deklariert:

7.1.1. moveCompleted()

Wird verwendet, wenn ein Spielzug erfolgreich beendet wurde.

7.1.2. resetGame()

Wird aufgerufen, wenn das Model zurückgesetzt wird.

7.1.3. playerPassed()

Wird aufgerufen, wenn ein Spieler passt.

7.1.4. gameEnded()

Wenn das Spiel beendet ist (zwei Mal passen oder Resignation), wird diese Methode aufgerufen.

7.2. GameEvent

Bei jeder Methode des GameListener-Interfaces wird eine Instanz des GameEvents übergeben. Das GameEvent stellt den jeweiligen Status des Spiels und Zusatz-Informationen dar. In jedem GameEvent wird die Referenz zum Model mitgegeben. Es kann auch dazu verwendet werden, die Indizes des letzten Spielzuges zu erhalten und gegebenenfalls in einem Lesbaren Format auszugeben.

7.3. GameState

Die ENUM Klasse GameState stellt alle möglichen Zustände des Models dar. Jeder ENUM-Wert beinhaltet auch eine textuelle Repräsentation des Zustandes. Das Model aktualisiert seinen Zustand in der variable gameState nach Bedarf. Der GameState wird auch bei der Konstruktion des GameEvents übergeben.

8. SaveGame (Package)

Das Package SaveGame behandelt die Interaktion und das Konvertieren von Spielständen in TXT-Dateien. Es enthält die Klassen MoveHistory und SaveGameHandler.

8.1. MoveHistory

Die Klasse MoveHistory verfolgt die Spielereignisse und ermöglicht den Zugriff auf die Liste der Ereignisse. Sie implementiert das Interface GameListener, um auf Spielereignisse zu warten und die Historie entsprechend zu aktualisieren. Die MoveHistory wird bei Erstellung eines GoBoardModels initiiert.

8.2. SaveGameHandler

Die Klasse SaveGameHandler ermöglicht das Laden und Speichern von Spieldaten in Dateien. Zur Konstruktion wird ein File benötigt, in dem die Informationen gespeichert oder geladen werden sollen. Der SaveGameHanlder kann daraufhin ein Model erstellen, welches entweder für die Interaktion im Spiel oder als Tutorial verwendet werden kann. Dafür werden die Methoden createGameModel() und createTutorialModel() angeboten. Zum Konvertieren der Datei in ein Model werden die einzelnen Zeilen, wie in Kapitel 8.2.1. beschrieben, gelesen und in eine Liste gespeichert.

Die Klasse verfügt auch über Methoden, welche das erstellte Model mit simulierten Spielzügen füllt. Dies kann mittels dem gespeicherten Zeilenindex bis zur letzten Zeile geschehen.

Die Klasse verfügt auch über eine statische Methode, welche ein GoBoardModel und ein File entgegennimmt und die Spielzüge des Models in das File speichert. Dazu wird die Klasse MoveHistory verwendet und die einzelnen Einträge im Format der RegEx gespeichert.

8.2.1. Dateiformat

Spielstände werden im TXT-Format gespeichert erlauben somit die Nachvollziehbarkeit durch ein durch Menschen lesbares Format. In der ersten Zeile der Datei werden die Meta-Informationen zum Spiel angegeben. Jede darauffolgende Zeile stellt einen Spielzug dar. Mittels RegEx werden die jeweiligen Zeilen interpretiert und evaluiert.

Die Erste Zeile ist wie folgt aufgebaut:
Spielfeldgröße (Integer)**;** Handicap (Integer)**;** Komi (Double)
Beispiel: 19;6;0.5 → Spielfeldgröße = 19, Handicap = 6, Komi = 0.5

Zeilen, welche das Setzen eines Steines beinhalten, sind wie folgt aufgebaut:
Reihe (Integer)**;** Spalte (Integer)**-** GameState
Beispiel: 8;3- White player's turn. → Reihe = 8, Spalte = 3, GameState = WHITE_TURN

Spielzüge, in denen kein Stein platziert wurde, beinhalten nur den GameState.
Beispiel: Black player passed. → GameState = BLACK_PASSED

9. Unit Tests

Um die Korrektheit der Methoden und Klassen zu gewährleisten, wurden Unit Tests zu den Klassen der Pakete Model und SaveGame erstellt. Dabei wurden jegliche Methoden dieser Pakete anhand verschiedener Szenarien getestet und in relevanten Testfällen mit der Grenzwertanalyse vorgegangen. Die Testklassen wurden in die Jeweiligen Java-Klassen des Systems aufgeteilt. Zum Beispiel wurde für die Klasse GoBoardModel somit die Testklasse GoBoardModelTest angelegt.

Die Coverage beträgt für die Packages Model und SaveGame jeweils 100% bei den Methoden und über 95% bei den Code-Zeilen.

10. Ressourcen

Im Package resources werden benötigte Ressourcen des Systems gespeichert. Diese beinhalten das Icon der Software, Bilder für Knopfe im TutorialView und eine CSS-Datei zum Styling der einzelnen Interaktionselemente.

Auf der Ebene über dem Package befindet sich ein Ordner mit vordefinierten Problemstellungen / Tutorials. Diese Dateien sind nicht in einer unteren Ebene, um den Dateipfad von der ausführbaren Jar-Datei aus leichter zu erreichen.