JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von

Felix Stadler k12043299
Simon Ulmer k12043331
Dominik Niederberger
k12111671

Gruppe 5

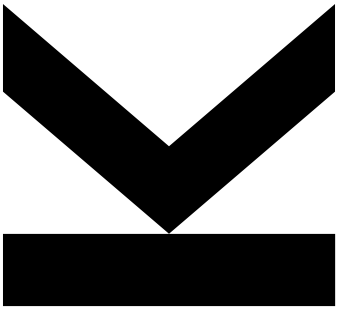
Angefertigt am

Institut für
Wirtschaftsinformatik –
Software Engineering

Beurteiler / Beurteilerin

Dr. Johannes Sametinger

GOGAME APPLIKATION



Projektdokumentation

PR Software Engineering

259.035
SS 2023

Datum

06.2023

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.

Das folgende Dokument gilt als Projektdokumentation. Darin werden die Anforderungen an das System, das Vorgehen im Entwicklungsprojekt, als auch eine Reflexion der Projektmitglieder beschrieben.

JOHANNES KEPLER UNIVERSITÄT LINZ

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Organisation

Das Softwareprojekt benötigt für eine effiziente Umsetzung mehrere organisatorische Maßnahmen. Neben der Organisation des Projektes werden mehrere Softwareprodukte genutzt, um die Implementierung des Softwaresystems zu optimieren. Alle Maßnahmen werden in den folgenden Kapiteln beschrieben.

1.1. Projektorganisation

Um einen optimalen Entwicklungsprozess zu gewährleisten sind Organisatorische Maßnahmen notwendig. Hierbei sind besonders die Kommunikation und Arbeitseinteilung im Team wichtig. Bei diesem Projekt wurde auf eine Agile Projektorganisation gesetzt wonach die Entwicklung in definierte Iteration stattgefunden hat. Mittels Kanban Board wurden die Sprints und ihre Tasks aufgelistet und eingeteilt und priorisiert.

1.2. Softwarewerkzeuge

Für das Projekt wurden diverse Softwarewerkzeuge verwendet. Nachführend werden kurz der Verwendungszweck und das zugehörige Tool aufgelistet.

Gemeinsame Datenverwaltung
Um für alle Projektmitglieder, die benötigen Dokumente bereitstellen zu können, wurde auf die Cloudlösung von Microsoft OneDrive gesetzt. Hierdurch muss man sich keine Sorgen um Versionsprobleme machen und auch Features wie die automatische Speicherung und das simultane Arbeiten auf einem Dokument werden unterstützt.

IDE
Zur Programmierung wurde auf die Java IDE von Jetbrains/IntelliJ gesetzt.

Repository
Für die Speicherung und Versionierung des Softwareprojektes wurde GitHub verwendet. Dies kann direkt in die IDE eingebunden werden, was eine korrekte Versionsverteilung enorm vereinfacht.

Zeitaufzeichnung
Um die Zeitaufzeichnung für das Projekt zu vereinfachen wurde das Tool Clockify verwendet. Dieses ermöglicht eine Zeitaufzeichnung im Browser wobei jede Aufzeichnung einem Sprint/Task zugeordnet werden kann. Auch ein Dashboard ist inkludiert, um einen Überblick über das aktuelle Projekt zu erhalten.

Kommunikation
Für den stätigen Austausch unter den Projektmitgliedern wurde Discord verwendet. Dies ermöglicht eine leichte Freigabe des eigenen Bildschirms, wodurch auch Pair-Programming mit Ortsdifferenz möglich ist.

2. Anforderungen

Um die Anforderungen des Softwaresystems korrekt planen und umsetzen zu können, müssen zuerst die Regeln des Brettspieles Go definiert werden. Die Definition der Regeln und der Anforderungen folgt demnach in den nächsten Kapiteln.

2.1. Die Regeln von Go

Bevor die Anforderungen formuliert werden können, müssen zuerst die Regeln analysiert werden.
Go ist ein strategisches Brettspiel, aus Asien. Es wird von zwei Spielern auf einem quadratischen Brett mit 19x19 Linien gespielt. Jedoch gibt es für Anfänger auch kleinere Formate wie 13x13 oder 9x9. Das Ziel des Spiels ist es, mehr Territorium auf dem Brett zu kontrollieren als der Gegner.

Jeder Spieler hat eine bestimmte Anzahl von Spielsteinen, in der Regel schwarze und weiße Steine genannt. Der Spieler mit den schwarzen Steinen beginnt das Spiel. Die Spieler setzen abwechselnd einen Stein auf eine der Kreuzungen auf dem Brett. Sobald ein Stein gesetzt wurde, kann er nicht mehr bewegt werden.

Die Spielsteine können zusammengefasst werden, um Gruppen zu bilden, die gemeinsam bewegt und geschützt werden können. Wenn eine Gruppe von Steinen von einem oder mehreren gegnerischen Steinen vollständig umzingelt wird, wird sie gefangen genommen und aus dem Spiel entfernt.

Das Spiel endet, wenn beide Spieler zustimmen (hintereinander passen) oder wenn ein Spieler aufgibt – beim Aufgeben ist immer der aufgebende Spieler der Verlierer. Der Gewinner des Spieles ist jener, der zu Spielende die meisten Punkte erzielt hat. Hierfür werden die am Spielfeld befindlichen Steine als jeweils ein Punkt und die geschlagenen Steine als jeweils ein Punkt gezählt. Zu diesen Punkten werden noch die jeweiligen Territoriums Punkte addiert. Dies sind jene Flächen, die vom Spieler umzingelt sind, also jene Flächen, die nicht vom Gegner berührt werden – dabei gilt, dass sobald sich ein Stein des Gegners innerhalb der eigenen Fläche befindet, diese nicht mehr gezählt wird (siehe Abb. 1, wenn auf B2 ein schwarzer Stein wäre, dann würde das Territorium nicht mehr für weiß zählen). Die Punkte pro Spieler sind demnach die Summe der geschlagenen Steine, plus die am Spielfeld platzierten Steine, plus die eingenommenen Territorien. Der Spieler mit der höheren Punktezahl gewinnt.

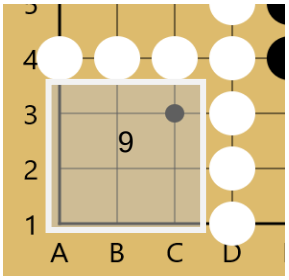


Abbildung 1: Erklärung Territorium

Auf folgende Genauigkeiten ist noch zu achten:
Ein Stein hat auf dem Spielfeld 4 Freiheitsgrade. Am Spielfeldrand 3. Sowie in einer Ecke 2. Sobald ein Stein keine Freiheitsgrade mehr besitzt, weil er von gegnerischen Steinen gefangen wurde, muss dieser vom Spielfeld genommen werden.

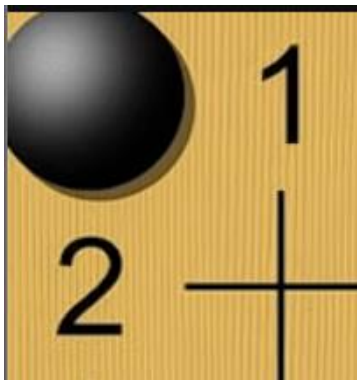


Abbildung 4: 2 Freiheitsgrade

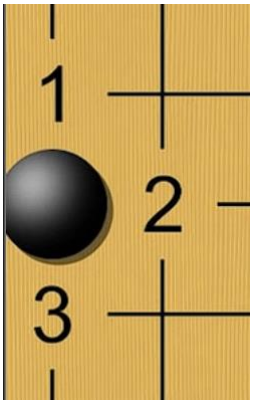


Abbildung 3: 3 Freiheitsgrade

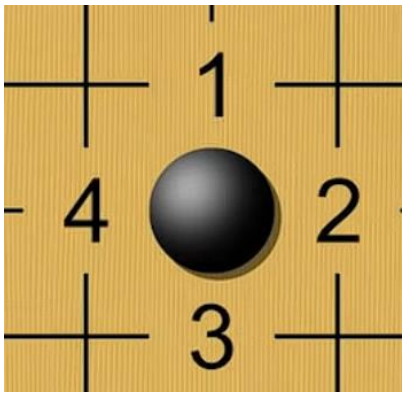


Abbildung 2: 4 Freiheitsgrade

Wie in Abbildung 5 zu sehen lassen sich diese Regeln auch auf Gruppen von Steinen anwenden. So wären diese 3 Steine gefangen wenn auf den 8 Freiheitsgraden weiße Steine platziert wären und müssten somit vom Spielfeld genommen werden.

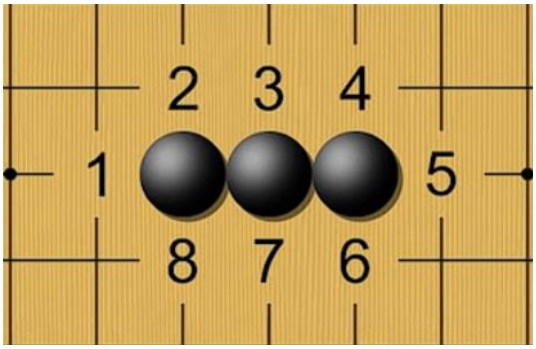


Abbildung 5: Gruppe von Steinen

Eine weitere wichtige Regel ist, dass „Selbstmord“ verboten ist. Das bedeutet, dass ein Stein nicht dort platziert werden darf, wo er keine Freiheitsgrade würde (siehe Abbildung 6 – in der Mitte der Gruppe von Steinen).

Zuletzt gibt es noch die Ko-Regel. Diese besagt, dass zwei aufeinanderfolgende Züge nicht die ursprüngliche Stellung wiederherstellen dürfen. Im Laufe des Spieles kann es zu einer

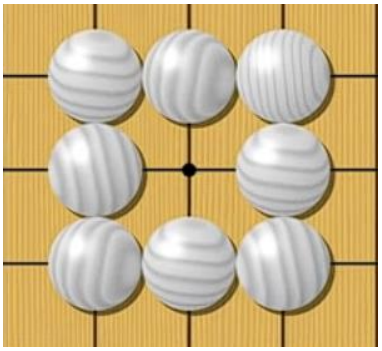


Abbildung 8: Selbstmord Regel

Pattsituation kommen. Wie in Abbildung 7 und 8 zu sehen, kann hier abwechselnd von beiden Spielern der generische Stein gefangen werden, wonach sich das Spiel in einer Endlosschleife befinden würden. Dies wird durch die Ko-Regel verhindert.

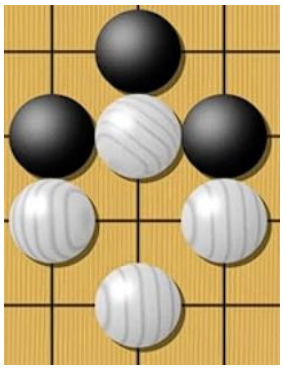


Abbildung 7: Schwarz setzt

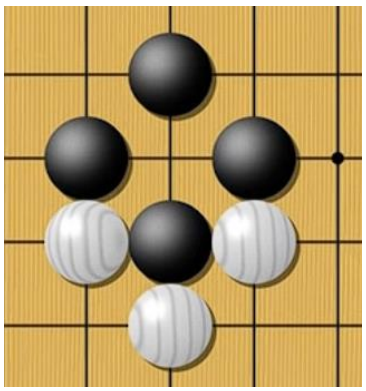


Abbildung 6: Weiß setzt

2.2. Anforderungen

2.2.1. GUI

Das Graphical User Interface (GUI) soll dem Nutzer der Software als Interaktionsschnittstelle dienen. Das GUI sollte neben den Grundlegenden Interaktionsmöglichkeiten für das Brettspiel noch folgende Anforderungen erfüllen.

Das GUI sollte auf Änderungen der Größe des Fensters reagieren und die Spieloberfläche dementsprechend aktualisieren.

Die Interaktion sollte neben einem Input durch die Maus auch mittels Tastatureingaben gesteuert werden können.

2.2.2. Spiele lesen / schreiben

Vergangene Spiele sollten durch eine Textdatei gespeichert und wiedergegeben werden können. In der Textdatei soll jeder Spielzug in einer für Menschen lesbaren Form gespeichert werden. Somit können die eingelesenen Spiele schrittweise abgespielt werden.

2.2.3. Unterschiedliche Brettgrößen

Das Brettspiel Go wird standardmäßig auf einem 19x19 Brett gespielt. Jedoch soll vor dem Spiel die Größe des Brettes auch auf 13x13 und 9x9 eingestellt werden können. Dadurch wird auch die Anzahl der maximal platzierbaren Kompensationssteine verändert.

2.2.4. Markieren von Steinen und Positionen

Im Verlauf des Spiels ist es wichtig, potenzielle Positionen zu kennzeichnen, bei denen das Setzen eines bestimmten Spielsteins dazu führen würde, dass ein feindlicher Stein oder eine feindliche Gruppe geschlagen wird. Darüber hinaus kann der Spieler auch einzelne Steine markieren, um spezifische Spielstrategien zu verfolgen.

2.2.5. Problemstellungen / Lösungen

Das Spiel wird mit einer umfangreichen Bibliothek von vordefinierten Spielsituationen ausgestattet sein. Diese Spielsituationen dienen dazu, dass Spieler ihre Fähigkeiten und Taktiken durch Übung und Nachverfolgung verbessern können. Die Bibliothek bietet eine Vielzahl von Szenarien, die verschiedene Schwierigkeitsgrade und Herausforderungen beinhalten. Durch das Nachverfolgen und Üben dieser vorgefertigten Spielsituationen können Spieler ihre strategischen Entscheidungsfindungsfähigkeiten schärfen und ihre Spielstärke weiterentwickeln.

2.2.6. Steinvorgaben

Wenn ein Spieler stärker ist als der andere, besteht die Möglichkeit, dem schwächeren Spieler Kompensationssteine anzubieten. Diese Kompensationssteine werden auf vordefinierten Positionen auf dem Spielbrett platziert. Sie dienen als Ersatz für den ersten Zug des schwächeren Spielers. Durch das Hinzufügen dieser Kompensationssteine erhält der schwächere Spieler eine bessere Ausgangsposition. Dies ermöglicht ein faires und ausgeglichenes Spiel, selbst wenn die Spieler unterschiedliche Fähigkeiten haben.

2.2.7. Zugverzicht

Der Spieler hat die Möglichkeit auf seinen Spielzug zu verzichten und somit keinen Stein zu setzen. Sobald beide Spieler hintereinander auf ihren Spielzug verzichtet haben, wird das Spiel beendet.

2.2.8. Freiheiten von Steinen und Gruppen

Die freien Flächen neben platzierten Steinen werden Freiheiten genannt. Sie repräsentieren Plätze, an denen Steine platziert werden können. Ein Stein kann maximal vier Freiheiten besitzen.

Wenn jede Freiheit eines Steins durch einen Stein des Gegners besetzt, wird dieser Stein bzw. die Steingruppe geschlagen. Die Steine werden somit vom Spielbrett genommen und zum Punktestand des Schlagenden addiert.

2.2.9. Punktezählung nach Spielende

Sobald beide Spieler gepasst haben, wird das Spiel beendet. Der Punktestand der Spieler ergibt sich durch die am Spielfeld befindlichen Steine, das eingeschlossene Gebiet, der

gefangenen / geschlagenen Steine und dem eingestellten Komi. Der Spieler mit den meisten Punkten gewinnt das Spiel. Dieser Punktestand soll am Spielende ersichtlich sein. Dabei sollten die Punkte in Gebietspunkte und Punkte durch gefangene Steine aufgeteilt sein.

3. Marktanalyse

Das Brettspiel Go ist auf verschiedenen Plattformen verfügbar. Diese reichen von Browser-Implementationen über zahlungspflichtige Spiele-Applikationen zu Open-Source Softwaresystemen. Folgende Produkte werden aufgelistet um diverse Besonderheiten, die eine Differenzierung am Markt darstellen, aufzuzeigen.

3.1. Browser Spiele

Bei den Browser Applikationen gibt es einerseits sehr einfach gehaltene Anwendungen wie „**Cosumi**“ oder „**Playgo**“ bei denen nur ein Spiel gegen eine KI möglich ist und auch die grafische Umsetzung sehr simple gehalten wurde. Für das Spielen gegen andere Spieler gibt es Plattformen wie „**Gokgs**“ oder (die größte) „**Online-Go**“. Diese Plattformen bieten umfassende Einstellmöglichkeiten bei den Spielregeln (Komi, Freiheiten usw.) sowie ansprechende Spieloberflächen. Weiters gibt es bei „**Online-Go**“ die Möglichkeit Tutorials zu absolvieren und andere Live-Spiele anzusehen sowie sich mit anderen Go-Spielern auf der Seite zu vernetzen.

3.2. Mobile Apps (IOS)

Im Appstore wird eine große Auswahl an Go-Applikationen angeboten. Bei der Marktanalyse wurden drei Applikationen mit besonderen Features gefunden, die im Folgenden vorgestellt werden.

Die App **Go Game** hat die Besonderheiten, dass das Spielerlebnis durch umfassende Einstellmöglichkeiten an der GUI individualisiert werden kann. Weiters ist es möglich mit verschiedenen Regeln (Chinesisch und Japanisch) zu spielen und ist ein Timer für die beiden Spieler implementiert.

Tsumego Pro ist keine Anwendung, um Go zu spielen. In der App werden Problemstellungen/Tutorials zur Verfügung gestellt, um „ideale“ Spielzüge zu lernen und trainieren.

Little Go ist eine Open-Source Anwendung, die im Appstore zur Verfügung gestellt wird. Die Besonderheiten bei der Applikation sind die Möglichkeit alte Spiele zu speichern und später erneut durchzuspielen sowie die ausführlichen Statistiken, die sich zu jedem Spiel angezeigt werden können (z.B. Anzahl der Züge, Anzahl der geschlagenen Steine).

3.3. Desktop Applikationen

Bei den Desktop Applikationen ist die Auswahl der Go-Spiele gering. Hier wurden nur die Applikationen „**Just Go**“ und „**Gomuko Lets Go**“ gefunden. Diese beiden sind sich relativ ähnlich, da sie beide von denselben Entwicklern sind. „**Gomuko Lets Go**“ ist dabei die ältere, weniger umfangreiche und dadurch auch günstigere Anwendung. Die Besonderheiten von „**Just Go**“ sind die 3D und 4k Grafiken, der Karrieremodus, das Nachspielen von berühmten Go-Partien und der Single- sowie Multiplayermodus.

3.4. Open Source Lösungen

Bei der Markanalyse wurde auch die Open-Source Community GitHub nach vergleichbaren Java Applikation durchsucht.

Bei der Applikation von Thomas Durand et al. welche unter folgendem Link zu finden ist https://github.com/Dean151/Go-gamel handelt es sich um eine Go-Game Variante, welche mit Java und Java Swing umgesetzt wurde. Des Weiteren wurde Github und Maven als zusätzliche Softwaretools verwendet.
Besonderheiten der Implementierung sind:

- 2 Spieler spielen lokal gegeneinander abwechselnd auf der GUI
- Alle klassischen GO-Regeln wurden implementiert
- Spiele können gespeichert und geladen werden
- Es gibt die Möglichkeiten für undo/redo und skip
- Es kann zwischen 3 Spielfeldgrößen ausgewählt werden (9x9, 13x13, 19x19)
- Handicap kann vor Spielbeginn eingestellt werden
- Bei Spielende werden die finalen Punkte berechnet

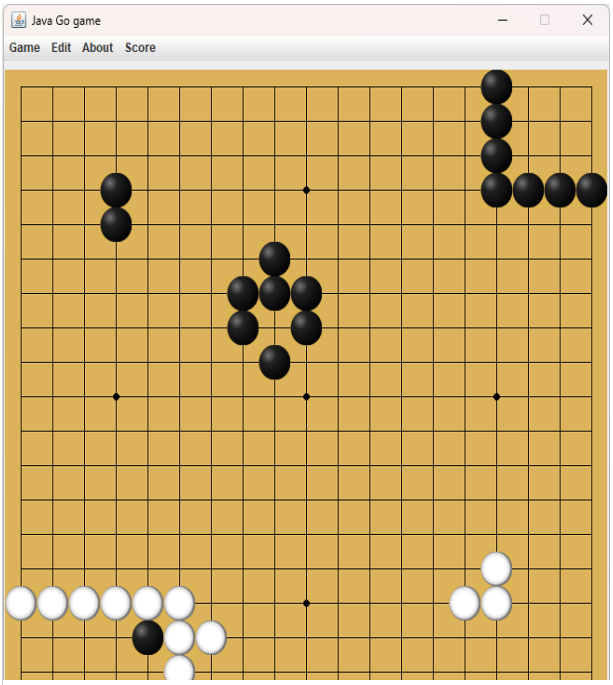


Abbildung 9: Java Go-Game

Im Vergleich hierzu ist die Applikation von Patrik und Janik Tinz, welche unter folgendem Link zu finden ist https://github.com/patricktinz/go-game ganz anders aufgebaut. Hier wird keine extra GUI erstellt, sondern die Konsolenausgabe der IDE als Spielfeld verwendet. Es wurden als zusätzliche Softwaretools ebenfalls GitHub zur Versionskontrolle und Apache Ant als Build-Tool verwendet.

Besonderheiten der Implementierung sind:

- KI kann als Gegner gewählt werden
- Gefangene Steine werden in Echtzeit angezeigt
- Undo Funktion

```
### Willkommen bei Wombat Go Version: 1.0 ##
(1) Spiel starten
(2) Beenden
Eingabe >
1

Wer moechte beginnen:
(1) Mensch
(2) Computer
Eingabe >
2

###       Zug 0      ###
### Gefangene Steine ###
Schwarz: 0    Weiss: 0

    A  B  C  D  E  F  G  H  J
 9  -  -  -  -  -  -  -  -  -
 8  -  -  -  -  -  -  -  -  -
 7  -  -  -  -  -  -  -  -  -
 6  -  -  -  -  -  -  -  -  -
 5  -  -  -  -  -  -  -  -  -
 4  -  -  -  -  -  -  -  -  -
 3  -  -  -  -  -  -  -  -  -
 2  -  -  -  -  -  -  -  -  -
 1  -  -  -  -  -  -  -  -  -

Eingabe Computer: c9

###       Zug 1      ###
### Gefangene Steine ###
Schwarz: 0    Weiss: 0

    A  B  C  D  E  F  G  H  J
 9  -  -  B  -  -  -  -  -  -
 8  -  -  -  -  -  -  -  -  -
 7  -  -  -  -  -  -  -  -  -
 6  -  -  -  -  -  -  -  -  -
 5  -  -  -  -  -  -  -  -  -
 4  -  -  -  -  -  -  -  -  -
 3  -  -  -  -  -  -  -  -  -
 2  -  -  -  -  -  -  -  -  -
 1  -  -  -  -  -  -  -  -  -
```

Abbildung 10: Wombat Go Version

4. Sprints

Im folgenden Abschnitt werden die Planungen, tatsächlichen Umsetzungen und Probleme, die innerhalb der Sprints aufgetreten sind, für die drei durchgeführten Sprints thematisiert. Vorab wurde die Anforderung aus Kapitel 2 in 3 verschiedene Prioritäten eingeteilt. Die Namensgebung hier ist sprechend für die hierarchische Priorisierung.

High Priority

- Standardspielregeln
- Freiheiten, Zugverzicht, Handicap, …
- Funktionierendes GUI
- Punktezählung nach Spielende
- UML-Diagramm

Medium Priority

- Resizable GUI
- Keyboard Input
- Spiele laden und speichern
- Lesbare Spieldateien
- Unterschiedliche Brettgrößen
- Benutzer-, System- und Projektdokumentation

Low Priority

- Markieren von Steinen / Positionen
- Setzreihenfolge anzeigen
- Problemstellungen (Tutorials)
- Codeanalyse

Diese Features wurden danach mittels Kanban Board in GitHub zu Issues konvertiert bzw. zusammengefasst und den einzelnen Sprints zugeordnet. So sind am Kanban Board die großen Arbeitspakte in kleinere Issues aufgespalten, um eine besser Arbeitsaufteilung im Team sowie Organisation zu gewährleisten. In Tabelle 1 ist die Sprintplanung mit den zugehörigen Start- sowie Endterminen festgehalten.

Datum	Planung
21.03.2023	Start Sprint 1
18.04.2023	Release Sprint 1, Start Sprint 2
23.05.2023	Release Sprint 2, Start Sprint 3
27.06.2023	Release Sprint 3, Start Final Release
18.07.2023	Final Release

Tabelle 1 Sprint Planung

4.1. Sprint 1

4.1.1. Planung

Für Sprint 1 galt es die Priorisierung auf einen funktionierenden UI-Prototyp zu legen und sich vermehrt mit JavaFX vertraut zu machen. Der Prototyp sollte jedoch schon die ersten High Priority Tasks wie teile der Standardregeln beherrschen. Simultan sollte auch die Klassenhierarchie entworfen und in einem UML-Klassendiagramm veranschaulicht werden.

4.1.2. Umsetzung

Nach Ende des ersten Sprints konnten alle geplanten Issues abgewickelt werden. In den Abbildungen 11-12 ist die erste Version des UI-Prototyps zu sehen. Hier wurde noch auf einen Startscreen gesetzt, dem der SettingsScreen folgte und erst danach das eigentliche Spiel gestartet wurde. Dieses Konzept wurde jedoch im 2 Sprint verworfen. In Abbildung 14 ist die erste Spielfeldimplementierung zu sehen, welche vom Grundprinzip bis zum Schluss verwendet wurde.

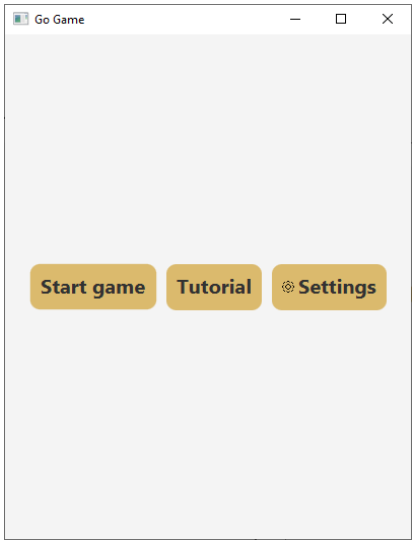


Abbildung 11: StartScreen

Abbildung 13: Settings Screen

Abbildung 12: Tutorial Screen

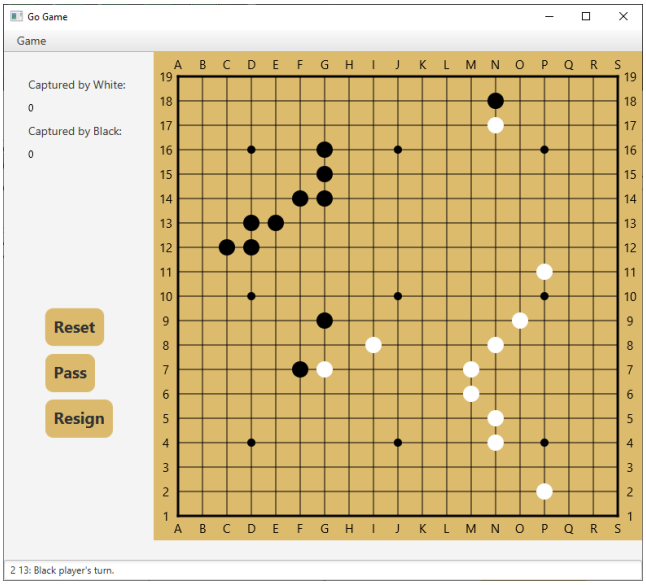


Abbildung 14: First Game Screen

Neben den lauffähigen Prototyp, welcher auch die ersten Spielregeln, wie Steine schlagen, beherrschte wurde auch zur Strukturierung des Programmes ein Ablaufdiagramm sowie ein UML-Klassendiagramm erstellt. Diese wurde jedoch im Laufe des Projektes verändert und auf die geänderten Anforderungen angepasst. Die abschließende Version ist in der Systemdokumentation zu finden, wie auch die verwendete Struktur und Design Patterns.

Da die geplanten Issues für Sprint 1 schon vor Sprintende umgesetzt werden konnten, wurde bereits mit dem Medium Priority Tasks begonnen, wie zum Beispiel die GUI resizable zu gestalten

4.1.3. Probleme

Bei diesem Sprint traten vor allem Probleme bei der Implementierung der adaptiven Anpassung der Größe des Spielfensters auf. Beim Wechseln zwischen den verschiedenen Szenen änderten sich die Größen des Fensters oftmals automatisch, wobei jedoch die Elemente im Fenster (z.B. das GoBoard) nicht mitskaliert wurden. Weiters wurde zur koordinierten Code Zusammenführung Pull Requests eingeführt, da vermehrt Probleme mit der Arbeitsaufteilung und Code Zusammenführung aufgetreten sind.

Parallel zu den technischen Problemen sind auch Grundsatzfragen aufgetaucht, da Go ein weltweit verbreitetes Spiel mit viele verschieden Regelvarianten ist, was die Implementierung eines Regelsatzes erschwerte.

4.2. Sprint 2

4.2.1. Planung

Im Rahmen von Sprint 2 stand die vollständige Implementierung der Spiellogik sowie die Weiterentwicklung der Benutzeroberfläche (GUI) im Fokus, um ein vollständig spielbares Spiel zu ermöglichen. Während des laufenden Entwicklungsprozesses sind verschiedene Ideen und Fehler/Probleme aufgetreten, die in dieser Iteration im Kanban Board als spezifische Issues erfasst wurden. Dadurch konnte ein strukturiertes Vorgehen bei der Bearbeitung dieser Themen gewährleistet werden.

Nach Abschluss des ersten Sprints zeigte sich eine mangelnde Modularität des Quellcodes, um eine effektive Wartbarkeit zu gewährleisten. Um diesem Problem entgegenzuwirken, wurde ein umfangreiches Refactoring durchgeführt, dass eine gezielte Verbesserung der Modularisierung des Quellcodes zum Ziel hatte. Diese Maßnahmen führten zu einer signifikanten Steigerung der Code-Qualität und trugen dazu bei, dass der Code nun weitaus verständlicher und besser strukturiert ist. Dadurch wurde nicht nur die Wartbarkeit erheblich verbessert, sondern auch eine solide Grundlage für zukünftige Entwicklungen geschaffen.

Im Rahmen des ersten Releases fand ein umfassendes Codereview mit dem kursleitenden Professor statt. Hier erhielten wir konstruktive Kritik für den Aufbau der GUI, welche auch in diesem Sprint umgesetzt wurde. Das Ziel war es, mehrere Applikationssichten zu entfernen und die jeweiligen Funktionen zusammenzufassen. Insbesondere wurden dabei der Homescreen und der Spielstart als Bereiche identifiziert, die überarbeitet werden sollten. Durch die gezielte Verbesserung der GUI konnte die Benutzerfreundlichkeit deutlich gesteigert werden.

4.2.2. Umsetzung

In Abbildung 15 ist die GUI am Ende von Sprint 2 zu sehen. Die Applikation hat nun nicht mehr wie im ersten Sprint einen Homescreen. Stattdessen wird direkt mit dem Spielfeld gestartet. Die Einstellungen können über die Menüleiste durchgeführt werden. Generell wurde auf ein einheitliches Erscheinungsbild geachtet.

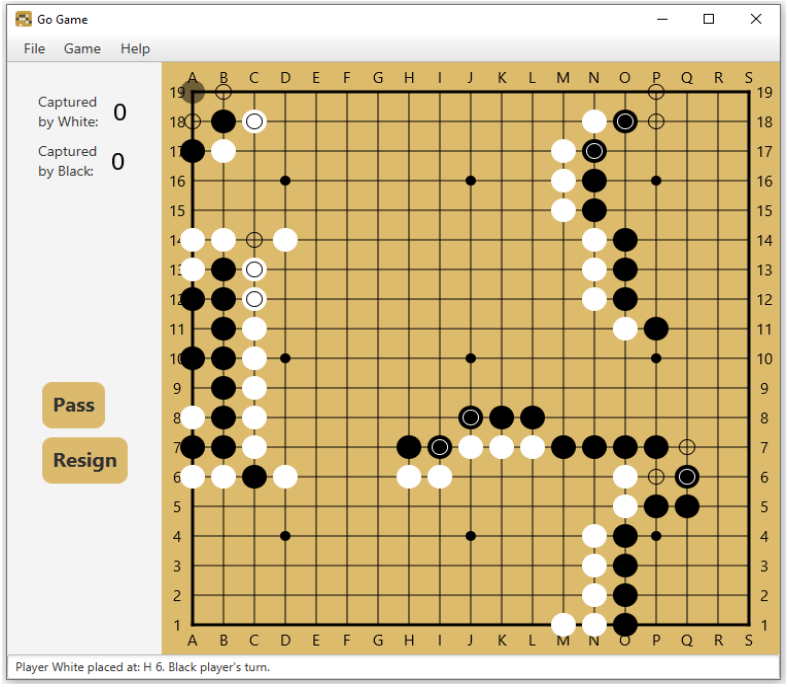


Abbildung 15: Endversion GUI

Auch die Usability der GUI wurde verbessert, da nun eine komplette Steuerung mittels Maus oder Tastatur möglich ist. So können alle Menüpunkte mittels Tastatursteuerung erreicht werden, wobei für die essenziellen Spielsteuerungen auch Shortcuts angelegt wurden.

4.2.3. Probleme

Beim Codereview wurden Vermischungen des MVC-Patterns festgestellt, wo View Komponenten in Hilfsklassen des Models verwendet wurden. Durch diesen Fehler musste die Programmstruktur angepasst werden.

Ein weiteres Problem im zweiten Sprint war, dass wenn Spielstände mit anderen Spielbrettgrößen als der gerade geöffneten Brettgröße geladen wurden, Fehler bei der Skalierung sowie beim Setzen der Steine auftraten. Die Steine wurden nicht auf die richtigen Positionen gesetzt und das Model im Hintergrund hatte eine andere Spielfeldgröße. Der Fehler wurde gelöst, indem beim Öffnen eines Spielstandes ein neues Model mit den neuen Metadaten angelegt wurde.

4.3. Sprint 3

4.3.1. Planung

Im letzten Sprint lag der Fokus darauf, die GUI sowie die Programmlogik abzuschließen, die verbliebenen Probleme und Fehler zu beheben und die Applikation als ausführbare JAR-Datei zu erstellen. Zusätzlich musste, aufgrund wesentlicher Änderungen an der Programmstruktur seit Sprint 1, das UML-Diagramm der Applikation aktualisiert werden. Ein weiterer Task war die Finalisierung der Logik zum Speichern und Laden von Spielständen.

Um einen fehlerfreien und qualitativ hochwertigen Sourcecode zu gewährleisten, sollten nach Abschluss der Implementierungen die erforderlichen Unit-Tests entwickelt werden. Die Tests sollen sicherstellen, dass das Programm den Anforderungen entspricht und zuverlässig funktioniert und sind daher von großer Bedeutung. Um die Codequalität weiter zu verbessern, sollten Werkzeuge wir das Analysetool SonarLite in Verbindung mit SonarQube und zusätzlich auch das (in der Ultimate Version) integrierte Analysetool von IntelliJ eingesetzt werden. Hierdurch soll der Quellcode gründlich analysiert werden, um mögliche Schwachstellen, Codefehler und Best Practices zu identifizieren.

Abschließend sollte eine Benutzerdokumentation, Systemdokumentation und Projektdokumentation erstellt werden. Hierbei liegt das Augenmerk auf einer umfassenden und präzisen Dokumentation, um einen klaren Überblick über die Anwendung, ihre Funktionalitäten und den Projektablauf zu geben.

4.3.2. Umsetzung

Im letzten Sprint wurden alle in Kapitel 4 festgelegten Prioritäten (High, Medium und Low) erfolgreich umgesetzt. Die Applikation verfügt nun über sämtliche geforderten Features und kann als ausführbare JAR-Datei auf Endgeräten mit der entsprechenden Java-Version verwendet werden.

Um die Codequalität weiter zu verbessern, haben wir wie geplant das Analysetool SonarLite in Verbindung mit SonarQube, FindBugs sowie das Analysetool von IntelliJ eingesetzt. Dadurch konnten wir potenzielle Probleme im Code frühzeitig identifizieren und beheben, was zu einer insgesamt höheren Codequalität geführt hat. Für mehr Informationen siehe Abschnitt 5 Codeanalyse.

Darüber hinaus wurden alle erforderlichen Dokumentationen innerhalb des letzten Sprints fertiggestellt. Dies bedeutet, dass der Sprint 3 erfolgreich abgeschlossen werden kann, da alle geplanten Ziele erreicht wurden und die geforderten Ergebnisse vorliegen.

4.3.3. Probleme

Aufgrund des Feedbacks während des Release-2-Termins war eine Umstrukturierung der Programmarchitektur erforderlich. Dabei wurde jedoch die ordnungsgemäße Integration der Funktionen zum Öffnen und Speichern von Spielständen versäumt, was zu vereinzelten Bugs und Fehlern führte. Um diese Probleme zu beheben, musste die Funktionalität umgebaut werden.

Während der Codeanalyse mithilfe von Analysetools wurden zusätzlich einige Fehler im Code entdeckt, die behoben werden mussten. Diese Fehler führten auch zu Anpassungen der Unit-Tests, um sicherzustellen, dass die korrekte Funktionalität des Codes gewährleistet wird. Weiters wurden bei der Codeanalyse nur sinnvolle Vorschläge umgesetzt, da die empfohlenen Änderungen oftmals zu einer wesentlichen Verschlechterung der Lesbarkeit/Verständlichkeit der Applikation geführt haben (für weitere Informationen siehe Abschnitt Codeanalyse). In einem Fall wurde ein Teil der Funktionsweise der Software durch den Vorschlag eines Analysetools entfernt. Vor allem in Bezug auf CSS funktionierten die Analysetools nicht sehr zuverlässig, da wir in unserer Applikation den Komponenten nicht direkt die CSS-Klassen zuweisen, sondern das gesamte Stylesheet.

Das Erstellen der JAR-Datei für die Applikation für verschiedene Betriebssysteme stellte sich als Problemstellung heraus. Hierbei wurden mehrere Versionen auf MacOS probiert bis der Fehler gefunden werden konnte. Die JAR-Datei funktionierte auf Windows wie geplant jedoch auf MacOS konnte die JAR-Datei aufgrund fehlender Berechtigungen nicht auf die Tutorial-Dateien, die in einem extra Ordner gespeichert sind, zugreifen. Durch das Ausführen der JAR-Datei mit dem Administratorbefehl sudo konnte dieses Problem behoben werden.

5. Codeanalyse

Um die Codequalität des Projektes zu gewährleisten, wurden Codeanalysen mit SonarLint (+ Verbindung mit Sonarqube), FindBugs und dem Codeanalysetool von IntelliJ durchgeführt. Alle vorgeschlagenen Änderungen durch FindBugs wurden umgesetzt, bei den Ergebnissen von SonarLint wurden einige Änderungen nicht umgesetzt.

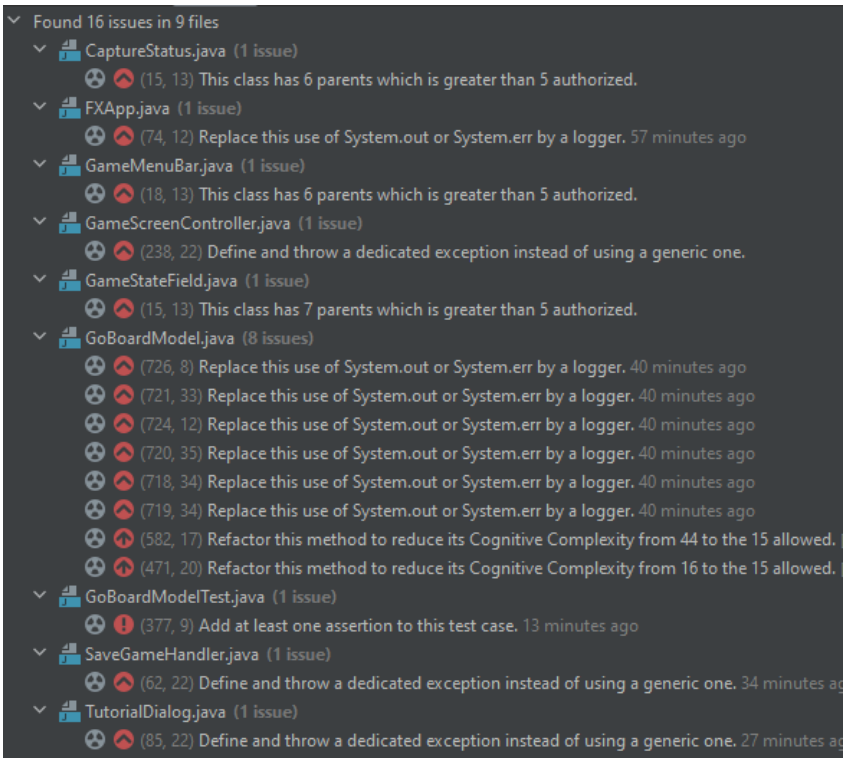


Abbildung 16: Ergebnisse SonarLint

- Replace this use of System.out or System.err by a logger.

Diese Änderungsvorschläge sind bei der Methode printModel(), die nur die Funktionalität umsetzt, das Spielbrett auf der Konsole auszugeben und daher wurde dies nicht umgesetzt.

- This class has x parents which is greater than 5 authorized.

Der Änderungsvorschlag wurde nicht umgesetzt, da das Vererbungsmodell in der Applikation bewusst so gewählt wurde und dieser erhöhten Komplexität durch eine ausführliche Erklärung in der Dokumentation sowie in der JavaDoc entgegengewirkt wird.

- Define and throw a dedicated exception instead of using a generic one.

Der Änderungsvorschlag wurde nicht umgesetzt, da an diesen Stellen selbst definierte Exceptions nicht notwendig sind, um die Funktionalität zu gewährleisten.

- Refactor this method to reduce its Cognitive Complexity from x to the 15 allowed.

Dieser Änderungsvorschlag für die Methoden moveIsKo() und calculateScores() wurde nicht umgesetzt, da durch die umfassende Dokumentation im Code und aufgrund der Übersichtlichkeit (die teilweise verloren gehen würde, falls man die Method aufteilen würde) das Umschreiben der Methoden als nicht sinnvoll erachtet wird.

Insgesamt war dabei das Analysetool von IntelliJ genauer und konnte mehr potentielle Fehlerquellen finden als SonarLite und FindBugs, wobei generelle Sicherheitsrisiken der Applikation nur von SonarQube erfasst werden können.

6. Zeitaufzeichnung

In diesem Kapitel ist die Dokumentation der Zeitaufzeichnung des Projekts zu finden. Eine detaillierte Aufteilung der Zeitbuchungen ist direkt in Clockify vorhanden. Aufzeichnungen zu Code-Commits sind in GitHub zu finden.

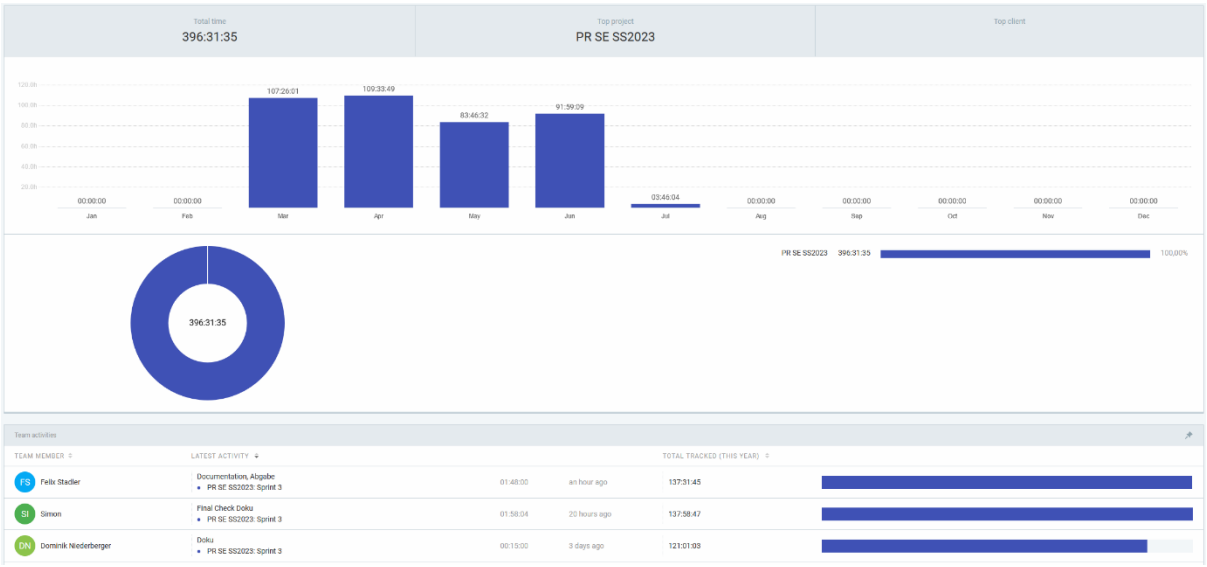


Abbildung 17: Clockify Zeitaufzeichnung

7. Reflexion

In diesem Kapitel sind die eigenen Erfahrungen der Projektteilnehmer zu finden

7.1. Felix Stadler

Mir hat das Praktikum Software Engineering gut gefallen. Die Aufteilung der Implementierung in drei Sprints gab genügend Möglichkeiten, Design-Entscheidungen zu präsentieren und über Unklarheiten zu diskutieren. Die gewählten Werkzeuge (GitHub & Clockify) waren auch hilfreich für das Projekt und sehr praxisnah. Jedoch fand ich die Wahl des Themas nicht optimal. Das Spiel Go mag zwar viele Möglichkeiten bieten, verschiedene Methoden des Software Engineering anzuwenden, trotzdem hätte zumindest über ein fixes Regelwerk entschieden werden sollen, damit die Ergebnisse der Gruppen vergleichbar sind. Aus der Sicht des Lerneffektes hätte mir eine Anforderung mit einer neuen Programmiersprache (Python, JavaScript,…) oder im Web-Applikations-Umfeld besser gefallen, da dies in der Praxis auch nicht unüblich wäre.

7.2. Simon Ulmer

Durch das Projekt konnte, das in vorherigen Lehrveranstaltungen erworbene Wissen in einem praxisnahen Beispiel angewendet werden. Dabei funktionierte die Kommunikation im Team sowie die Entwicklung mit Git über GitHub als Versionierungstool sehr gut – wie auch das Anlegen bzw. Verwalten der durchzuführenden Tasks als Issues in GitHub. Meiner Meinung nach war das Projekt eine lehrreiche Erfahrung wobei ich allerdings das Spiel GO

mit den „unklaren“ und verschiedenen Regelsets als nicht perfekt passendes Projektthema sehe.

7.3. Dominik Niederberger

Für das Projekt in Software Engineering gab es nur sehr wenig konkrete Vorgaben. Dies ließ einem viel Freiraum in der Gestaltung des ganzen Projektzeitraumes. Es konnte somit im Projektteam eine eigenständige Herangehensweise für die Problemstellung angewandt werden. Mir persönlich hat der Aufbau der Veranstaltung sehr gut gefallen, da auch in Projekten in der Industrie oft wenig Vorgaben vorhanden sind und man sich gezwungenermaßen oft selbst organisieren muss. Als konstruktive Kritik würde ich noch anfügen, dass anstelle eines Spieles vielleicht für zukünftige Projekte eher auf praxisrelevante Projektthemen gesetzt wird.