Softwaretechnikpraktikum WS 2021/22

Umsetzung eines Maulwurf Company-Spiels

DevOps Dokument

Version 1.1

(Stand: 28.01.2022)

SWTPra Gruppe 10

Changelog

Version 1.1

- Development-Teil: Gesamt-Architektur ergänzt und an Smartphone angepasst
- Development-Teil: Nummerierung geändert
- Development-Teil: 1.1 “Smartphone-Teilnehmer” ergänzt
- Development-Teil: 1.3.2 View Beschreibungen überarbeitet
- Development-Teil: 1.4 Smartphone-Teilnehmer hinzugefügt
- Development-Teil: 1.7 Weiterentwicklung Smartphone-Teilnehmer hinzugefügt
- Operation-Teil: 2.3.1 PC-Beobachter überarbeitet und erweitert
- Operation-Teil: 2.3.2 Ausrichter ausgearbeitet und erweitert
- Operation-Teil: 2.3.3 Engine Teilnehmer aktualisiert und erweitert
- Einige Rechtschreibfehler und Grammatikfehler korrigiert

Inhalt

1. Development

Im Development Abschnitt dieses Dokumentes wird die Arbeitsweise der Software aus Entwickler-Seite erklärt. Die Gesamt-Architektur beschreibt kurz die Funktionen der einzelnen Komponenten, darauf folgt die Beschreibung des Servers, des PC-Beobachters, des Engine-Teilnehmers sowie der Libraries und Frameworks und wie das Programm weiterentwickelt werden kann.

1.1 Gesamt-Architektur

Die Aufgabe der Software ist es, das klassische Brettspiel "Maulwurf Company" mit einigen Modifikationen als Anwendung für PCs und Smartphones anzubieten.
Zur Umsetzung dieser Aufgabe wurde entschieden, die Architektur der Software kann in zwei Anwendungen aufzuteilen. Die Umfangreichste Anwendung ist der der PC-Client, der sowohl den Ausrichter, den Engine-Teilnehmer als auch den PC-Beobachter beinhaltet. Die zweite Anwendung ist der Smartphone-Client, der die Funktion eines Smartphone-Beobachters und eines Smartphone-Teilnehmers beinhaltet. Zusätzlich enthalten beide Anwendungen Komponenten zur Netzwerk Kommunikation.

Die Umsetzung der Kommunikation zwischen dem Server und dem Client erfolgt über das Client-Server-System. Da der Server von dem Ausrichter verwaltet wird, hat der Ausrichter die Möglichkeit alle Spiele und auch die Verbindungen zu den Clients zu kontrollieren. Die Clients müssen sich am Server anmelden um die Dienste (Spiele) des Servers anfordern zu können.
Für den Datenaustausch zwischen Server und PC-Clients werden server- und clienteigene Klassen und Methoden benutzt, um einander dem Interface Dokument entsprechende Nachrichten im JSON-Format zuzuschicken. Zur Umsetzung des Datenaustausches zwischen Server und Smartphone-Clients wurden die server-eigenen Klassen und Methoden und die Interface Library verwendet, um Nachrichten dem Interface Dokument entsprechend zu interpretieren und zu verschicken.

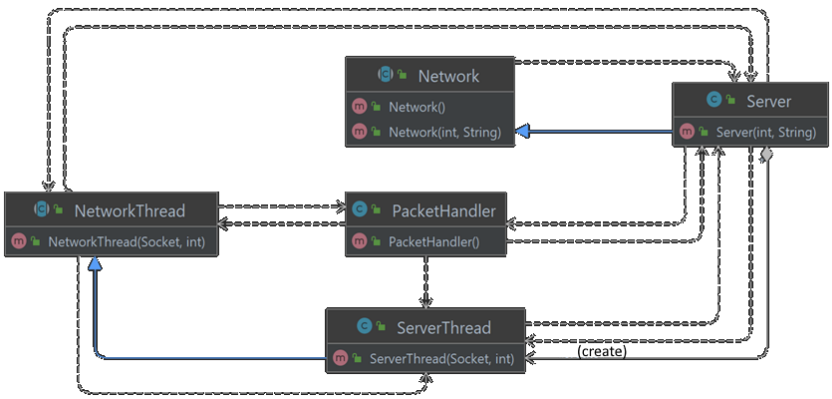
Sobald der Ausrichter gestartet wurde, ist dieser bereit, Anmeldungen von Clients zu empfangen. Zusätzlich können Spiele und Turniere über eine integrierte grafische Oberfläche mit Hilfe des Frameworks “JavaFX” erstellt werden. Erfolgt eine Anmeldung an einem Server, so werden die vom Server übertragenen Daten durch eine im PC-Beobachter integrierte graphische Benutzeroberfläche (GUI) dargestellt, die ebenfalls mit Hilfe des Frameworks “JavaFX” erstellt wird. Zusätzlich gehört zu jeder JavaFX-Datei auch eine “.fxml”-Datei, die den Großteil der darzustellenden Bilder der entsprechenden GUI speichert. Was genau dargestellt werden soll, wird von dem Client berechnet, wodurch das von uns verwendete Model View Controller (MVC)-Modell vollständig definiert ist. Dabei ist der Client der Controller, die JavaFX- und zugehörige fxml-Dateien sind die View und die Restklassen sind das Model.

Abgesehen von dem Java Code sind in beiden Clients sowie der obersten Ebene der Projekt Struktur auch Gradle Dateien vorhanden, die das Bauen der ganzen Software, also die Exekution der Software als Applikation, ermöglichen.

1.2 Server

Das Projekt verwendet eine eigens dafür entwickelte Interface API. Diese ist auf Basis des Interface Dokuments entwickelt worden. Somit ist die Kommunikation mit den von anderen Gruppen des Praktikums entwickelten Modulen möglich. Für das Aufsetzen der Entwicklungsumgebung wurde als Build und Dependency Management Tool Gradle[1] 7.3 verwendet. Die minimal unterstützte und für die Entwicklung genutzte Java Version ist 11.0.

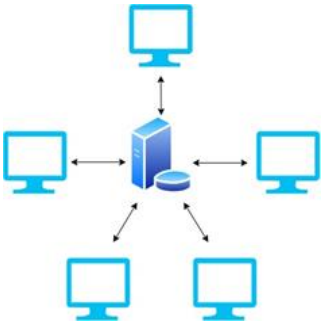
Zur Client-Server-Verbindung wurde der Standard-Port 5000 genutzt und der Serversocket läuft auf dem Localhost. Die allgemeine Kommunikation zwischen den Komponenten ist weitestgehend abstrahiert durch Vererbungen von Basisklassen, dies sorgt für eine bessere Verständlichkeit und Einhaltung der “don't-repeat-yourself”-Regel. Aus diesem Grund sind die Klassen „Client“ und „Server“ eine Erweiterung der Klasse „Network“. Das Handling der Verbindung läuft jeweils über die Klassen “Serverthread” und “ClienThread” welche die Basisklasse “Networkthread” haben. Der Server und der Client lesen empfangene Nachrichten im String-Format ein. Diese Nachrichten sind gemäß des Interface Dokuments im JSON Format. Daraufhin wird überprüft, um welche Art von Empfänger es sich handelt (Server, Client oder Künstliche Intelligenz). Dementsprechend wird der für den Empfänger passende PacketHandler mit den dazugehörigen “PacketHandler”-Methoden aufgerufen.



1 Klassendiagramm zur Struktur der an der Kommunikation beteiligten Klassen auf Seiten des Servers

Wir verwenden die Sterntopologie. Als zentraler Knoten steht der Server im Mittelpunkt und verbindet sich mit allen Clients. Die Kommunikation findet über den Server statt. Die einzelnen Clients kommunizieren ausschließlich mit dem Server und keinesfalls untereinander.

[1] https://docs.gradle.org/



2 Schematische Darstellung der Sterntopologie

Für die Netzwerkkommunikation und den Aufbau des Spiels selbst wurden bei der Entwicklung verschiedene Design-Patterns genutzt. Die Nutzung der beiden folgenden Varianten bilden, unter Berücksichtigung der Faktoren Wartbarkeit und Erweiterbarkeit, das Grundgerüst für die Software.

Observer Pattern

Für die Kommunikation zwischen Client und Server wird das Observer Pattern verwendet. Dies beschreibt ein reines IO-System, welches den über den TCP-Stream empfangenen Input analysiert, den JSON-String in ein Packet-Objekt umwandelt und nach Analyse der entsprechenden Umgebung an den dafür zuständigen “Packet-Handler” weitergibt. Dieser wird, auf Basis des Contents des entsprechenden Packets, dafür passende Handlungen ausführen und gegebenenfalls auf die empfangene Nachricht mittels Antwort-Packet dem Server oder Client antworten. Die Handlung auf das empfangene Packet sowie die entsprechende Antwort haben meistens Auswirkung auf die Darstellung oder Struktur des in der Nachricht angesprochenen Models und somit werden automatisierte Prozesse gestartet, die das Spielfeld sowie das Back-End direkt aktualisieren.

Beispielsweise empfängt der Client eine Nachricht in Form des Spielfeldes, welche das Model auf Basis der empfangenen Nachrichten verändert. Mit diesem Mechanismus lassen sich Änderungen am Spielfeld automatisch umzusetzen. Dies bezieht sich nicht nur auf das Spielfeld, sondern auch auf die entsprechenden Models welche durch die einzelnen Nachrichten im speziellen angesprochen und dann verändert werden.

Model View Controller Pattern

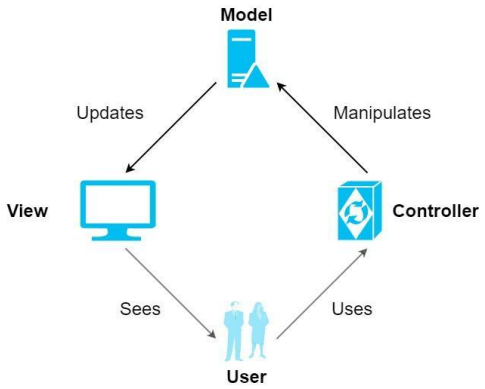
Für den Spielaufbau wird das Model View Controller Pattern (MVC-Pattern) verwendet. Diese strikte Modularisierung, unter Berücksichtigung der Zielkriterien Low Coupling und High Cohesion, ermöglichen uns eine erhöhte Wartbarkeit und eine einfache Anpassbarkeit der Software. Diese Architektur lässt sich in die drei Teilmodule Model, View und Controller unterteilen, die anhand des Interfaces zwischen Nutzer und System erläutert werden.

Controller: Die Klasse „Client“ ist die Controller Klasse und hat Zugriff auf alle benötigten Informationen. Bei Änderungen werden die Daten auf dem Server geprüft, verarbeitet und über die Klasse “Packethandler” zu “ClientPacketHandler” geschickt. Dieser leitet die Daten

an den Controller „Client“ weiter, der anschließend die Daten des Models manipuliert und in der jeweiligen View anzeigt.

Model: Jedes Model bekommt seine Informationen von der Controller-Klasse “Client”, welche die Daten des Models manipuliert. Diese Informationen werden an die zugehörige View gesendet und angezeigt.

View: Die View bezeichnet die grafische Komponente. Diese wird von dem Model aktualisiert und ist die Schnittstelle zwischen Benutzer und System. Für jede View gibt es ein passendes Model, dem die Daten bereitgestellt werden.



3 Schematische Darstellung des Model View Controller Patterns

1.3 PC-Beobachter

1.3.1 Logik

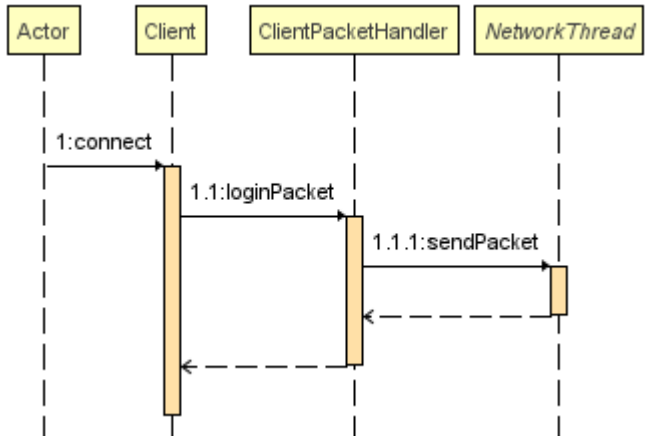
Der Client (PC-Beobachter) baut über die ClientThread-Klasse eine Verbindung zum Server auf. Die ClientThread-Klasse erbt wiederum von der NetworkThread-Klasse und letztendlich von Thread. Dabei kümmert sich diese Klasse darum, eine entsprechende Socket[2]-Verbindung zum Server herzustellen.

Das Senden und Empfangen von Paketen wird über die ClientPacketHandler-Klasse geregelt. Die ClientPacketHandler-Klasse beinhaltet eine Methode, um die GameState-Pakete des Servers zu verarbeiten und delegiert diese, je nach Paket, an eine darauf spezialisierte Methode weiter.

Die Klasse Client verwendet die oben erwähnten Netzwerk-Klassen und instanziiert jeweils eine Version der beiden Klassen. Dabei stellt diese die Schnittstelle für die Kommunikation mit dem Server für die jeweiligen UI-Klassen bereit. LoginScreen erstellt aus den eingegebenen Login Daten (Name, IP und Port) ein Objekt der Klasse Client, welches im Anschluss von den UI Klassen verwendet wird, um Informationen zu dem Server zu senden, sowie Informationen von dem Server abzufragen. Dabei wird auch direkt die connect()-

[2] https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html

Methode des Clients aufgerufen, um entsprechend des Interface Dokumentes, ein Login-Paket an den, beim Login angegebenen, Server zu schicken. War dies erfolgreich, ist der Client zum Senden und Empfangen von Paketen bereit.



4 Sequenzdiagramm: Aufruf der connect()-Methode

1.3.2 View

Die GUI des Clients ist in unterschiedliche Klassendateien für die Oberflächen des PC-Beobachters aufgeteilt. Für die Anmeldeoberfläche, Spieleauswahl, Turnierauswahl gibt es eigene Klassen, die jeweils als eigene Oberfläche dargestellt werden und sich um die Benutzereingaben kümmern. Die grafische Oberfläche wird über das JavaFX- Framework erstellt und beinhaltet abseits vom Spielfeld pro Oberfläche eine fxml-Datei. Die verwendeten Hintergründe der Oberflächen sind in der jeweiligen fxml-Datei definiert.

LoginScreen: Im LoginScreen gibt der Beobachter seinen Namen und die IP, sowie den Port des Servers ein, um sich mit einem Server zu verbinden. Über den Login Button wird ein entsprechender Socket aus den Eingaben instanziiert, um eine Verbindung zu dem angegebenen Server aufzubauen. Hierbei werden die Eingaben vorher auch auf korrekte Syntax überprüft. Nach erfolgreicher Rückantwort des Servers wird man in das PlayerMenu weitergeleitet. Als Eingabe werden UTF-8 Zeichen bis einem maximum von 32 Zeichen erlaubt.

PlayerMenu: Dies ist das Hauptmenü des PC-Beobachters und erscheint, sobald der Client mit einem Server verbunden ist. Es bietet die Möglichkeit, sich vom Server abzumelden, eine Liste der Spiele anzuzeigen (GameSelection), sowie eine Liste der Turnierspiele (TournamentSelection). Beim Aufruf der jeweiligen Liste wird ein entsprechendes Paket zur Abfrage der Overview gesendet, um daraus die Liste der Spiele oder Tuniere zu generieren und anzuzeigen.

GameSelection: In diesem Fenster werden gestartete, laufende und pausierte Spiele des Servers angezeigt, denen der Beobachter beitreten kann. Weitere Game Overview Updates vom Server rufen automatisch aus dem ClientPacketHandler die Update Methode innerhalb der GameSelection auf, um eine aktualisierte, sortierte Liste der Spiele anzuzeigen. Wenn ein Spiel der Liste ausgewählt wird und der beobachten Button gedrückt wird, wird überprüft, in

welchem Status dieses sich befindet. Falls das Spiel vorbei ist, wird der Score des Spiels in einer entsprechenden Punkte Tabelle angezeigt (LeaderBoard), ansonsten wird überprüft ob das Spiel bereits läuft/pausiert ist oder noch nicht gestartet wurde. Bei noch nicht gestarteten Spielen wird die View an den LobbyObserverGame übergeben, bei einem gestarteten/pausierten Spiel wird ein joinGame Packet gesendet und bei welcomeGame Antwort, dass GameBoard erstellt.

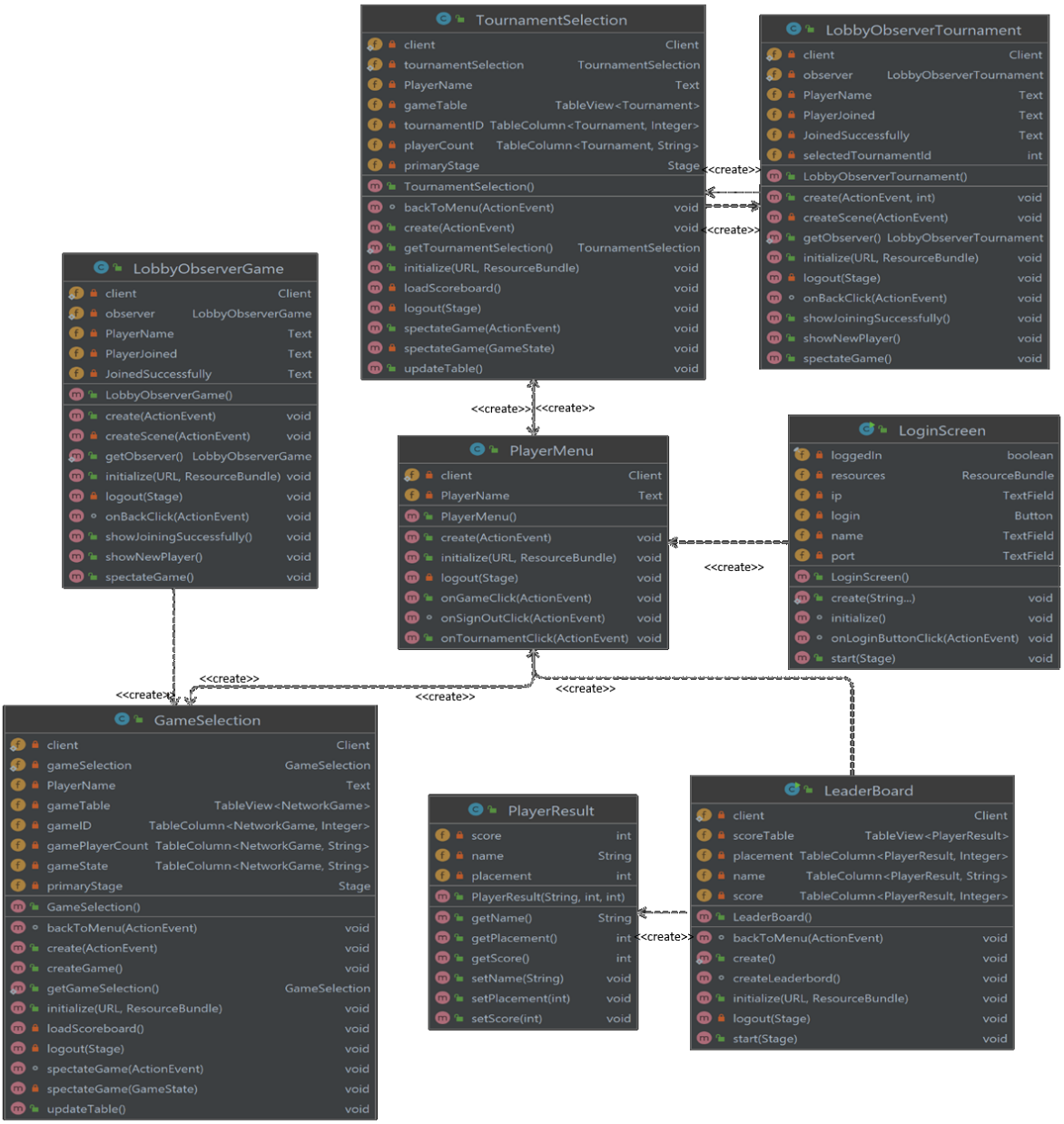
LobbyObserverGame: Hier wird die Lobby des Spiels angezeigt, dem der Beobachter beigetreten ist, inklusive der Information, ob der Beitritt erfolgreich war und welche anderen Spieler innerhalb der Lobby warten. Außerdem zeigt es die Einstellungen des Spiels an. Aus der Game Overview zieht diese Klasse die Informationen über das jeweilige Spiel, d.h. welche Einstellungen (Settings) gesetzt sind. Bei einem Beitritt eines weiteren Spielers (playerJoined) wird aus dem ClientPacketHandler direkt die entsprechende Update Methode der Spielerliste dieser Klasse aufgerufen. Hierüber aktualisiert sich die Tabelle mit den Spielern der Lobby und die Anzeige, dass ein Spieler beigetreten ist.

TournamentSelection: Für die Turnierauswahl werden die gleiche Funktion und die gleichen UI-Elemente wie für die Spielauswahl (GameSelection) für Turniere verwendet. Bei Auswahl eines Turnieres wird der aktuelle Status überprüft und zu der jeweiligen passenden View Scoreboard, Lobby oder zum Spiel) geleitet.

LobbyObserverTournament: Hier werden die gleiche Funktion und die gleichen UI-Elemente wie bei LobbyObserverGame für Turniere verwendet. Mit Start eines Spieles wird dem Beobachter das passende GameState zugeschickt und das GameBoard wird aufgerufen.

GameBoard: Das Board (Spielfeld) zeigt das tatsächliche Spielgeschehen inklusive aller relevanten Informationen (Punkte, Reihenfolge, aktueller Spieler, verbleibende Zeit) an. Die einzelnen Teilelemente, aus denen sich das Spielfeld zusammensetzt, sind in jeweils eigene Klassen ausgelagert. Das GameBoard verwendet als einziges UI Klasse keine fxml Elemente, sondern generiert die Oberfläche automatisch aus den Spielinformationen, die im GameState und in den Updates, welche vom Server gesendet werden, enthalten sind. Dabei gibt es für die einzelnen internen Klassen (Player, Mole, Map), Abstraktionen, welche die jeweiligen UI Elemente darstellen. Das GameBoard generiert dabei zunächst aus dem GameState die Oberfläche und wartet danach auf Updates des Servers (über den ClientPacketHandler), um Spielzüge darzustellen. Hierfür gibt es entsprechend spezielle Methoden für die jeweiligen Spielzüge. Die Unterscheidung findet über den Packet Typ in dem ClientPacketHandler statt.

LeaderBoard: Das LeaderBoard zeigt in einer TableView eine sortierte Liste mit der Platzierung und den Punkten jedes Spielers an. Die Liste wird mit Hilfe von PlayerResult erstellt. Dabei wird die Liste der Spieler anhand der Punktzahl absteigend aufgebaut.



5 Klassendiagramm: Struktur der GUI Controller Klassen des Beobachters ohne Spielfeld

1.4 Smartphone-Teilnehmer

Die Klassen des Smartphone-Teilnehmer-Clients sind in die Packages board, client und ui (User Interface) gegliedert.

Im “board” package enthaltene Klassen:
Im board package liegt der Kern des Spiels. Es enthält diverse Klassen für das Generieren und das Darstellen des Spielfeldes, sowie für die Verwaltung der Spielzüge und Benutzereingaben des Teilnehmers. Außerdem enthält es die gesamte Spiellogik.

Board: Die Klasse ermöglicht Abfragen von Statusinformationen der Maulwürfe, Zugscheiben und Löchern auf dem Spielfeld. Des Weiteren wird hier überprüft, ob ein vorgenommer Zug gültig war.

GameThread: Aktualisiert in regelmäßigen Abständen das Spielfeld.

Gameview: Gameview stellt sowohl das Spielfeld, als auch die Maulwürfe, Löcher und Spielzüge dar. Bei Spielbeitritt wird hier zunächst das Spielfeld in der draw Methode generiert und die Positionen der auf dem Spielfeld befindlichen Elemente abgerufen. Außerdem wird der Hintergrund generiert. Für das Generieren des Hintergrundbildes wird durch switch-cases abhängig von der jeweiligen aktuellen Ebene des Spielfeldes ein Bild als Grundlage gewählt und anschließend bildschirmfüllend aneinandergereiht. Die Spielzüge selbst werden in der surfaceCreated Methode verarbeitet. Zunächst werden mit einem setOnTouchListener die Eingaben des Nutzers abgerufen und auf Gültigkeit überprüft. Die letzten beiden gültigen Eingaben eines Nutzers werden im Lasttwoclicks Array gespeichert. Im Falle eines Spielzuges wird makeMove im messageHandler mit den beiden im Array gespeicherten Punkten und der ausgewählten Zugscheibe aufgerufen. Außerdem hat der Spieler zu Spielbeginn über Aufruf der placeMole Methode die Möglichkeit einen Maulwurf zu platzieren.

Pos: Die Klasse enthält die Matrix des Spielfeldes.

Scoreboard: Das Scoreboard stellt alle Informationen zu den Spielständen da.

ShownMole: Ermöglicht Platzierung und ausführen von Spielzügen von Maulwürfen. Enthält Darstellung und Animation der Maulwürfe.

ShownDisc: Die Klasse enthält alle für die Logik und die Darstellung der Zugscheiben erforderliche Methoden.

Im "client" package enthaltene Klassen:
Das Client Package stellt die grundlegenden Funktionen bereit. Es stellt die Verbindung zum Server her. Des Weiteren verwaltet es das Senden und Empfangen von Paketen.
Client: Ermöglicht durch die auf dem Loginscreen eingegebenen Login Daten (Name, IP und Port) eine Verbindung des Clients mit dem Server. Der Client instanziiert sowohl den MessageHandler als auch den ServerHandler.

MessageHandler: Verwaltung von vom ServerHandler empfangenen Nachrichten.

ServerHandler: Die Klasse sendet und empfängt Packets vom Server.

Im "ui" package enthaltene Klassen:
Das ui Package enthält sowohl die Lobbys, als auch den Loginbildschirm und ermöglicht das Beitreten von Spielen. Des Weiteren enthält es Klassen mit Hilfsmethoden.

GameActivity: Bei Aufruf eines Spiels ruft die Klasse die GameView auf und falls das Spiel noch nicht gestartet sein sollte, zunächst die GameWaitLobby.

GameListAdapter: Lädt die Spielinformationen in den Listen der Lobby.

LeaveGameDialog: Bei Verlassen des Spiels wird ein Dialog angezeigt. Dialog enthält Option für Verlassen des Spiels.

LobbyActivity: Dies ist das Hauptmenü des Smartphone Teilnehmers. Es wird aufgerufen, sobald das Einloggen erfolgreich war. Es ermöglicht den Wechsel zwischen Beitritt als Teilnehmer und Beobachter. Außerdem aktualisiert es die Listenelemente. Des weiteren hat der Spieler die Möglichkeit ShowRules aufzurufen.

LoginActivity: Im Loginscreen gibt der Nutzer seinen Namen und die IP, sowie den Port des Servers ein, um sich mit einem Server zu verbinden. Nach erfolgreicher Rückantwort des Servers wird man in die LobbyActivity weitergeleitet. Hierbei werden die Eingaben vorher auch auf korrekte Syntax überprüft. Außerdem hat der Spieler die Möglichkeit ShowLoginRules aufzurufen.

ShowConfiguration: Zeigt die Regeln und die Spielkonfiguration während einem Spiel an.

ShowRules: Ermöglicht das Anzeigen der Regeln in der Lobby.

ShowLoginRules: Ermöglicht das Anzeigen der Regeln während des Logins.

GameWaitLobby: Der Wartebildschirm wird aufgerufen, falls ein Spiel bei Spielbeitritt noch nicht gestartet sein sollte, und wird bei Spielbeginn automatisch geschlossen.

TabsAdapter: Enthält die Tabs in der Lobby welche Spiele bzw. Turniere mit verschiedenen Status enthalten.

TournamentListAdapter: Ermöglicht das Anzeigen der Turnierinformationen in einer Liste.

Utils: Enthält einige Methoden für Arrays und ermöglicht Anzeige von Meldungen, welche in diversen anderen Klassen benutzt werden.

1.5 Engine-Teilnehmer

Der Engine Teilnehmer, ab sofort als AI referenziert, nutzt ein konsolenbasiertes Modell. Dies Bedeutet, dass die AI keine Grafiken o.ä. anzeigt. Die gesamte Kommunikation der AI erfolgt über die Konsole mittels Textausgaben. Dies geschieht, wenn der Debug-Modus eingeschaltet ist. Sobald das Spiel gestartet ist, wird die AI aktiv. Die AI wartet auf das “playersTurnPacket” oder “playerPlacesMolePacket” vom Server mit dem Inhalt, dass sie selbst nun am Zug sei. Entsprechend setzt die AI einen Maulwurf auf ein Feld des Spielfeldes, welches kein Loch ist und nicht vorher belegt wurde. Dieses platzieren passiert zufällig. Falls entsprechende Maulwürfe platziert werden, aktualisiert die AI den aktuellen Spielstand intern.

In dem Fall, dass die AI einen Maulwurf bewegen muss, sortiert sie Ihre platzierten Maulwürfe in “In-Loch Maulwürfe” und “Freie Maulwürfe”. Es werden die Freien Maulwürfe priorisiert. Die AI geht nun alle freien, also nicht in einem Loch stehenden, Maulwürfe durch und schaut, ob es möglich ist, mit einer der vorhanden Pull-Discs einen Maulwurf in ein naheliegendes Loch

zu bewegen. Ob dieser Zug legal ist, wird durch eine Logik überprüft. Ist dies der Fall so wird der Zug ausgeführt. Ist ein solcher Zug illegal werden nach und nach alle Pull-Discs abgearbeitet dann alle freien Maulwürfe. Ist kein Maulwurf nah an einem Loch oder es ist nicht möglich mit den Pull-Discs und einem legalen Zug in ein Loch zu kommen, so wird ein random Maulwurf ausgesucht, es werden für diesen theoretisch alle Pull-Discs, sowie mögliche Bewegungsrichtungen durchgegangen und geschaut, welcher der erstmögliche legale Zug ist (es wird also ein random legaler Zug ausgeführt).

1.6 Libraries & Framework

Kommunikation: Die für die Kommunikation wichtigsten Libraries, welche sowohl vom Server als auch vom Client verwendet werden, sind Gson[3] und JsonObject[4]. Diese werden in den PacketHandler-Klassen vom Client und Server verwendet, um Java Objekte in eine JSON-Darstellung und umgekehrt JSON-Zeichenfolgen in Java-Objekte zu konvertieren. Dies ist erforderlich, da der Austausch zwischen Client und Server über das JSON-Datenformat abläuft, während die Informationen, welche ausgetauscht werden sollen, wiederum nur als Java-Objekte in den Klassen zur Verfügung stehen, weshalb Gson für die Umwandlung notwendig ist.

Für die Verbindung zwischen Client und Server sind vor allem die Libraries Socket[5] und ServerSocket[6] wichtig. Socket stellt die Verbindung zwischen Client und Server her ServerSocket erstellt auf dem Server ein Server-Socket, welcher auf einem definierten Port (hier 5000) lauscht und Pakete entgegennimmt.

Die grafischen Oberflächen von PC-Beobachter, Smartphone-Teilnehmer und Ausrichter werden, wie oben beschrieben, mit dem JavaFX-Framework erstellt. Dieses verwendet fxml-Dateien, welche sowohl die Elemente der grafischen Oberfläche selbst, als auch deren Anordnung im Fenster definieren.

Automatische Tests werden über das JUnit 5[7] Framework ausgeführt. Dafür müssen die einzelnen Tests zunächst geschrieben und können anschließend eingesetzt werden, um die Funktionalität einer Methode oder von ganzen Klassen über die aktuelle und zukünftige Entwicklung sicher zu stellen. Wird in Zukunft die Kartengenerierung der Map-Klasse angepasst, stellt ein Unit-Test sicher, dass die generierte Karte weiterhin die definierten Standards des Interface-Dokuments einhält. Als unterstützendes Framework wird hier zusätzlich Mockito[8] eingesetzt, welches ermöglicht, Klassen zu imitieren und z.B. explizit Rückgabewerte für Methoden anzugeben. Dies ist vor allem dann nützlich, wenn eine Methode eine Abhängigkeit auf eine sehr komplexe Klasse oder eine Klasse mit vielen Abhängigkeiten (z.B. wenn eine Netzwerkkommunikation erforderlich ist) hat. So kann der

[3] *com.google.gson.Gson:* https://sites.google.com/site/gson/gson-user-guide
[4] *com.google.gson.JsonObject:* https://www.javadoc.io/doc/com.google.code.gson/gson/2.6.2/com/google/gson/JsonObject.html
[5] *java.net.Socket:* https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html
[6] *java.net.ServerSocket:* https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html
[7] https://junit.org/junit5/docs/current/user-guide/
[8] https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html

Test schneller und einfacher implementiert werden, da dieser nicht auf eine funktionierende Netzwerkkommunikation angewiesen ist, welche die Komplexität des Tests erhöht.

1.7 Weiterentwicklung

1.7.1 PC-Beobachter

Um die zukünftige Weiterentwicklung so einfach wie möglich zu gestalten, ist das Projekt in jeweils kleinere Teilbereiche mit entsprechender Verzeichnisstruktur, mit Fokus auf Vererbung, aufgeteilt, um Redundanzen zu sparen. So gibt es jeweils Klassen für die einzelnen Elemente des Spiels (Map, Mole, Player, Settings usw.), die wiederum auch von spezialisierten Klassen Gebrauch machen. Mole speichert seine Position in einem Objekt der Klasse Field, welches die Koordinaten des Feldes, inklusive der Feldattribute beinhaltet. Möchte man nun ein weiteres Feld Attribut hinzufügen, kann man dieses in der Field-Klasse definieren und muss im Anschluss lediglich die neue gewünschte Spiellogik hinzufügen.

1.7.2 Smartphone Teilnehmer

Ähnlich wie der PC-Beobachter ist auch der Smartphone-Teilnehmer in kleinere Teilbereiche mit entsprechender Verzeichnisstruktur aufgebaut. Insbesondere die in der Utils Klasse enthaltenen Methoden sind darauf ausgerichtet, in vielen verschiedenen Klassen verwendet werden zu können, um Redundanzen zu vermeiden.
Mögliche Ideen für eine Weiterentwicklung wären beispielsweise:

- Hinzufügen einer AI Möglichkeit, so dass falls der Spieler keinen Zug ausführt, automatisch vor Ablauf der Zeit ein regelkonformer Zug ausgeführt wird.
- Hinzufügen der Option des Beitritts eines zufälligen noch nicht gestarteten Spiels.

2. Operation

Im Operation Abschnitt dieses Dokumentes wird die Anwendung der Software aus Benutzer-Sicht erklärt. Die Installation und Konfiguration beschreibt die Vorbereitungen, die zum Ausführen und Benutzen des Programms erforderlich sind, Ausführung beschreibt, wie der PC-Beobachter, der Ausrichter und die KI benutzt werden können und in Testing wird beschrieben wie die Software getestet und gebaut wird.

2.1 Installation

Das Programm wird mit einer jar Datei geliefert, die der Nutzer einfach ausführen kann. Eine vorherige Installation ist nicht notwendig. Allerdings wird ein System mit installierter Java SE 11 vorausgesetzt, damit das Programm genutzt werden kann. Außerdem wird das Programm

mit entsprechenden Dateien für die einfache Ausführung über die Benutzeroberfläche ausgeliefert.

2.2 Konfiguration

Der PC-Beobachter ist ohne Konfiguration auszuführen, da alle Informationen für den Betrieb über die grafische Oberfläche eingegeben werden.

Der Ausrichter kann ebenso komplett über die grafische Oberfläche verwendet werden, bietet allerdings auch die Möglichkeit, Spielkonfigurationen als Dateien zu importieren und so entsprechende Spiele zu konfigurieren. Für den Betrieb des Servers selbst ist es erforderlich, dass dieser ein Netzwerk-Interface mit IP hat, welche von den Clients aus erreichbar ist. Außerdem muss der Port (5000) von den Clients erreichbar sein. Hierfür muss gegebenenfalls die Firewall-Konfiguration des Servers oder der Netzwerkinfrastruktur, über welcher der Server angebunden ist, angepasst werden.

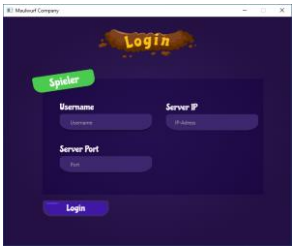
Engine-Teilnehmer können direkt über die Kommandozeile gestartet werden. Hierbei muss lediglich die IP, der Port und optional die Game ID angegeben werden.

2.3 Handbuch

In diesem Handbuch wird erklärt, wie der PC-Beobachter, der Ausrichter und der Engine-Teilnehmer verwendet werden können.

2.3.1 PC-Beobachter

Der PC-Beobachter wird über den Beobachter-Client gestartet. Der Client lässt sich durch Aufruf der mitgelieferten jar Datei und dem Kommandozeilenparameter “p” starten. Um ein Starten über die Desktop Oberfläche der jeweiligen Betriebssysteme zu unterstützen kann bei Windows die Beobachter.bat, bei MacOS die Beobachter.command, sowie bei Linux die Beobachter.sh/Beobachter-wayland.sh (je nachdem welches Display Server Protokoll von dem System verwendet wird) und entsprechende .desktop Dateien, die mit einer kompilierten jar im Verzeichnis /opt/ arbeiten, verwendet werden.



Nachdem das Programm gestartet wurde, erscheint zunächst der Login-Screen für den Beobachter. In diesem Fenster muss in dem Textfeld unter “Username” ein beliebiger Benutzername, im Textfeld unter “Server IP” die gültige IP-Adresse des Spielservers und im Textfeld unter “Server Port” der gültige Port des Spielservers eingetragen werden.Wenn alle Felder mit korrekten Eingaben gefüllt wurden, kann über den Button “Login” das Spiel-Menü geöffnet werden.

Im Menü wird oben links der Name des angemeldeten Spielers angezeigt sowie mittig die Buttons “Spiele”, “Turniere” und “Abmelden”. Um in die Spieleübersicht zu gelangen, muss der Button “Spiele” betätigt werden. Über den “Turniere”-Button gelangt der Nutzer in das Turnierauswahl-Fenster und über den “Abmelden”-Button kann der aktuelle Nutzer vom Spielserver abgemeldet werden.

Im Spiele-Fenster erscheint eine Liste mit laufenden, in Kürze startenden und vor Kurzem beendeten Spielen mit der jeweiligen Spiel ID, der Anzahl der beigetretenen Spielern und dem Status des Spiels. Der Status “OPEN” heißt, das Spiel wurde noch nicht gestartet. “STARTED” bedeutet, dass das Spiel gestartet wurde und aktuell läuft. “PAUSED” sagt aus, dass ein gestartetes Spiel pausiert wurde und “OVER” kennzeichnet ein beendetes Spiel.
Ein ausgewähltes Spiel wird grün angezeigt und kann über den “Beobachten”-Button betreten werden.

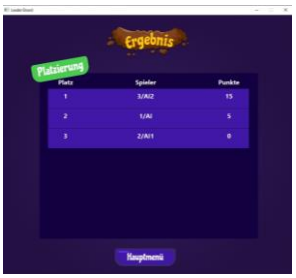
Im Lobby-Fenster werden in der linken Liste alle teilnehmenden Spieler angezeigt. Rechts gibt es einen Überblick mit allen Konfigurationen des ausgewählten Spiels. Falls ein Spieler dem Spiel beitritt erscheint die Meldung “Ein weiterer Spieler ist beigetreten”.
Sobald das Spiel gestartet ist, wird die Lobby geschlossen und das Spielfeld-Fenster öffnet sich. Durch Klick auf den “Zurück”-Button kann wieder ins Hauptmenü zurückgekehrt werden .

Im Turnier-Fenster erscheint eine Liste aller laufenden Turniere mit der jeweiligen Turnier ID und der Anzahl der teilnehmenden Spieler.
Ein ausgewähltes Turnier wird grün angezeigt und kann mit einem Klick auf den “Beobachten”-Button betreten werden.

Im darauffolgenden Lobby-Fenster wird mit der Meldung “Beitritt zum Turnier war erfolgreich! Bitte warten.” bestätigt, dass der Beitritt zum Turnier erfolgreich war.
Falls ein Spieler dem Turnier beitritt, erscheint die Meldung “Ein weiterer Spieler ist beigetreten”.
Sobald das Turnier gestartet ist, wird die Lobby geschlossen und das Spielfeld-Fenster öffnet sich. Durch Klick auf den “Zurück”-Button kann wieder ins Hauptmenü zurückgekehrt werden .

Sobald das Spiel gestartet ist, öffnet sich ein Fenster mit dem Spielfeld. Auf dem Spielfeld werden Ziehe erneut felder mit einem gelben kreis markiert. Der aktuell platzierte/bewegte Maulwurf ist mit einem Gelben Pfeilsymbol gekennzeichnet. Zudem werden in der Statusanzeige unten Informationen über das Spiel, z.B. welcher Spieler am Zug ist und um was für einen Zug es sich handelt, oben in der Mitte die verbleibende Bedenkzeit des aktuellen Spielers als Zahl und in der Tabelle rechts die Punkte mit Platzierung und die Bewegungskarten aller Spieler angezeigt. Über den Spiel verlassen Button unten rechts kann ins Hauptmenü zurückgekehrt werden.

Sobald das Spiel beendet oder abgebrochen wurde, erscheint ein Fenster mit dem Leaderboard, auf dem die Platzierung und die erreichten Punkte aller Spieler nach ihren Punkten sortiert angezeigt werden. Über den Button “Hauptmenü” wird das Leaderboard geschlossen und das Spiel-Menü geöffnet.



2.3.2 Ausrichter

Der Ausrichter wird über die Konsole gestartet. Hierzu muss beim Aufruf des Programms das Kürzel “s” als Parameter angegeben werden. Die Ausführung erfolgt gleich wie bei dem PC-Beobachter, hierfür heißen die entsprechenden Dateien Ausrichter statt Beobachter.

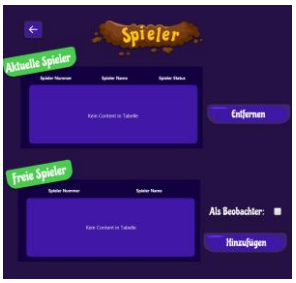
Sobald der Server gestartet wurde, erscheint das “Lobby”-Fenster des Ausrichters. Hier kann zunächst zwischen Spielen und Turnieren gewählt werden. Um in die Spieleübersicht zu gelangen, muss der Button “Spiele” betätigt werden. Und über den “Turniere”-Button gelangt der Nutzer in die Turnierübersicht.

In der Spieleübersicht kann über den “Spiel erstellen”-Button ein Fenster zum Erstellen eines neuen Spiels geöffnet, über den “Spieler hinzufügen”-Button ein Fenster zum Organisieren der Spieler in einem Spiel geöffnet und über den “Spiel starten”-Button ein gewähltes Spiel gestartet werden. Ausgewählte Spiele können zudem über die Buttons im unteren Bereich des Fensters unterbrochen, fortgesetzt oder beendet werden.

Beim Erstellen oder Bearbeiten eines neuen Spiels öffnet sich zunächst das “Settings”-Fenster. Hier kann über den Button “Konfig laden” eine gespeicherte Konfiguration geladen/wiederhergestellt oder eine neue Konfiguration manuell bestimmt werden. Für eine manuelle Konfiguration kann die Zahl der *Maulwürfe*, die erlaubte *Bedenkzeit* jedes Spielers pro Zug und die *Visualisierungszeit* der Spielzüge im jeweiligen Textfeld als Zahl eingegeben werden. Zudem kann die *Strafe* für einen regelwidrigen Spielzug aus einem Dropdown Menü ausgewählt werden. Nachdem eine Konfiguration vollständig erstellt oder geladen wurde kann mit dem Button “Spiel Erstellen” das Spiel geladen werden. Links kann der *Radius* des Spielfeldes, die Anzahl der *Spieler* und die *Zugscheiben* angegeben werden. Zudem lassen sich über den Button “Zieh-erneut Felder” das Fenster zum Bestimmen der Zieh-Erneut-Felder konfigurieren öffnen und über “Löcher Konfigurieren” ein Fenster zum Bestimmen der Löcher pro Ebene öffnen.

Wird der Button “Konfig Laden” betätigt, öffnet sich ein Fenster mit einer Übersicht über gespeicherte Konfigurationen. Hier kann eine vorhandene Konfiguration ausgewählt und über den “Laden”-Button übernommen werden oder eine im vorherigen Fenster definierte Konfiguration über den “Speichern”-Button gespeichert werden.

Im Fenster zum Bestimmen der Zieh-Erneut-Felder gibt es in der oberen Anzeige eine Übersicht mit allen Ebenen und im unteren Bereich eine Liste mit den “Zieh erneut”-Feldern der oben gewählten Ebene. Hier müssen dabei die (x,y)-Koordinaten für neue Zieh-Erneut-Felder und die Punktzahl pro Ebene angegeben werden.



Im Fenster zum Bestimmen der Löcher gibt es in der oberen Anzeige eine Übersicht mit allen Ebenen und im unteren Bereich eine Liste mit den Löchern der oben gewählten Ebene. Hier müssen dabei die (x,y)-Koordinaten für neue Löcher und die Punktzahl pro Ebene angegeben werden.

Beim “Spieler hinzufügen”-Button in der Spieleübersicht öffnet sich ein Fenster zum Verwalten von Mitspielern des gewählten Spiels. “Aktuelle Spieler” sind alle Spieler und Beobachter, die an dem Spiel teilnehmen. Über den “Entfernen” können ausgewählte Spieler oder Beobachter aus dieser Liste entfernt werden. “freie Spieler” zeigt alle Spieler und Beobachter an, die auf dem Server noch verfügbar sind. Über den “Hinzufügen”-Button können ausgewählte Spieler dem aktuellen Spiel hinzugefügt werden, wobei durch die Auswahl von “als Beobachter” bestimmt werden kann, ob dieser als Spieler oder nur Beobachter agiert.

In der Turnier-Übersicht wird über den “Turnier erstellen”-Button ein neues Turnier erstellt. Dieses kann mit Klick auf den “Turnier bearbeiten”-Button in einem neuen Fenster bearbeitet werden.

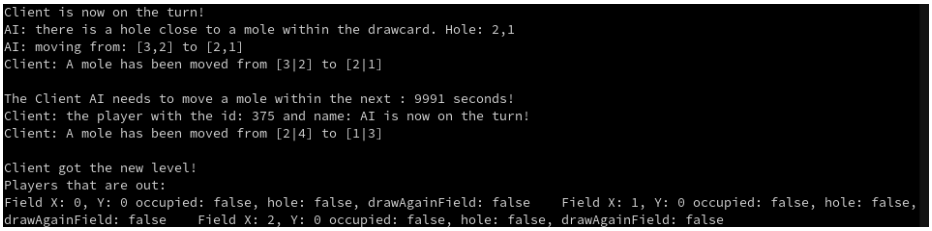
Im Fenster zum Bearbeiten eines Turniers können im linken Auswahlbereich aus vorhandenen Spielen gewählt werden, diese Spiele über den “Spiel entfernen”-Button gelöscht, über den “Spiel starten”-Button ausgewählte Spiele gestartet und über den “Spieler verwalten”-Button Spieler hinzugefügt werden. Über “Spiel hinzufügen” öffnet sich ein neues Fenster, in dem man Spiele zu dem Turnier hinzufügen kann. Ausgewählte Spiele können zudem über die Buttons im unteren Bereich des Fensters unterbrochen, fortgesetzt oder beendet werden.

In dem Fenster zum Hinzufügen von Spielen können nun schon bereits erstellte Spiele, durch Auswahl in der Tabelle, hinzugefügt werden, oder neue Spiele erstellt werden, die danach ebenfalls durch Auswahl hinzugefügt werden können.

2.3.3 Engine-Teilnehmer

Um einen Computergesteuerten Teilnehmer zu starten, muss zunächst das Terminal in dem Verzeichnis geöffnet werden in dem sich die SwtPra10-pc-1.3.2-all.jar Datei befindet.
Im Terminal kann nun über den Befehl „java -jar SwtPra10-pc-1.3.2-all.jar -a *IP/DOMAINNAME PORT GAMEID*“ ein Engine-Teilnehmer gestartet werden. Für „IP/DOMAINNAME“ muss hier die Server-IP und für „PORT“ der entsprechende Server-Port eingetragen werden und GAMEID muss durch die ID des Spiels ersetzt werden, an dem der Engine-Teilnehmer mitspielen soll. Die Angabe der GAMEID ist hierbei optional und die AI kann stattdessen auch von dem jeweiligen Server, mit dem diese verbunden ist, einem Spiel zugewiesen werden. Außerdem ist es auch möglich, den Engine-Teilnehmer über die mitgelieferte Engine.(sh für Linux und Mac/bat für Windows) zu starten. Hierbei verbindet sich die AI direkt mit der IP 127.0.0.1 und dem Port 33100.

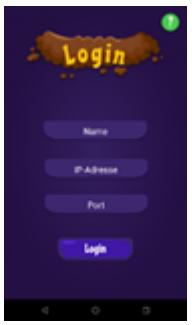
Falls der Engine Teilnehmer im Debug-Modus ausgeführt wird und an einem aktiven Spiel teilnimmt, werden alle Züge und Aktivitäten, die er ausführt, als Zeile mit allen Informationen im Terminal ausgegeben.



1 Beispiel-Auszug des Terminals im Debug-Modus

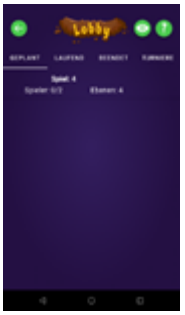
2.4 Smartphone-Beobachter und -Teilnehmer

Nachdem das Programm gestartet wurde, erscheint zunächst der Login-Screen. In diesem Fenster kann in dem Textfeld “Name” ein beliebiger Benutzername, im Textfeld muss “IP-Adresse” die gültige IP-Adresse des Spielservers und im Textfeld “Port” der gültige Port des Spielservers eingetragen werden. Wenn alle Felder mit korrekten Eingaben gefüllt wurden, kann über den Button “Login” das Spiel-Menü geöffnet werden. Um als Spieler an einem Spiel teilnehmen zu können muss auch ein Benutzername eingetragen werden. Über den “?”-Button kann das Regel-Fenster für den Login geöffnet werden – dies gilt auch für das Lobby-Fenster und das Spielfeld-Fenster.

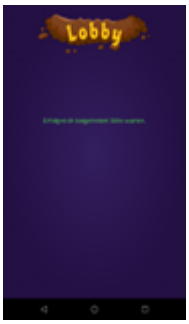


Im jeweiligen Regel-Fenster gibt es eine Übersicht über alle Regeln und Konfigurationen, die für das vorherige Fenster wichtig sind. Über den grünen Pfeil-Button kann in das vorherige Fenster zurückgekehrt werden.

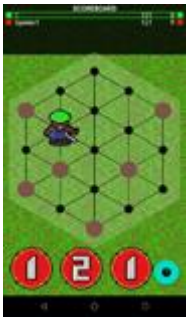
In der Lobby kann über den Button mit dem Augensymbol die Rolle zwischen Spieler und Beobachter gewechselt werden. Zudem wird über eine Reiteransicht eine Übersicht mit allen geplanten, laufenden und beendeten Spielen, sowie eine Turnierübersicht angezeigt. Zu jedem Spiel wird die Spiel ID angezeigt, aus wie vielen Ebenen das Spiel besteht und wie viele Spieler sich in dem Spiel befinden. Im “Laufend”-Reiter ist zudem angegeben, wann das Spiel gestartet hat und im “Beendet”- Reiter wird angezeigt, wann das Spiel beendet wurde und wer der Gewinner des Spiels ist. Im “Turnier”- Reiter erscheint eine Liste mit allen Turnieren, inklusive der Angabe der Turnier-ID und der Anzahl der Spieler, die an dem Turnier teilnehmen.



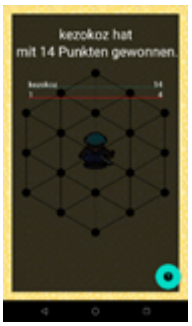
Wird ein noch nicht gestartetes Spiel gewählt, öffnet sich das Fenster mit der Wartelobby. Falls ein Spieler dem Spiel beitritt, erscheint die Meldung “Ein weiterer Spieler ist beigetreten”. Sobald das Spiel gestartet ist, wird die Wartelobby geschlossen und das Spielfeld-Fenster öffnet sich.



Auf dem Spielfeld werden Ziehe-erneut-Felder mit einem gelben kreis markiert. In der Statusanzeige oben wird über den ablaufenden Balken in der Farbe des Spielers der aktuelle Spieler und die verbleibende Bedenkzeit des aktuellen Spielers sowie die Punkte angezeigt.
Als Spieler werden im Unteren Teil des Bildschirms zusätzlich die eigenen Bewegungskarten aller Spieler angezeigt. Hier kann als aktiver Spieler nun eine Zugkarte ausgewählt und die Spielfigur bewegt werden.



Sobald das Spiel beendet oder abgebrochen wurde, erscheint ein Fenster mit dem Leaderboard, auf dem die Platzierung und die erreichten Punkte aller Spieler nach ihren Punkten sortiert angezeigt werden.
Über den Zurück-Button des Smartphones wird das Leaderboard geschlossen und das Lobby-Menü geöffnet.



2.5 Testing

Um Tests automatisiert zu starten wird GitLab CI eingesetzt. Dieses wird über die .gitlab-ci.yml Datei im Repository Root konfiguriert. GitLab CI startet bei jedem Commit eine Pipeline, welche den Build-Prozess, Unit-Tests und die Test-Code-Coverage in jeweils einem eigenen Docker Container startet. Die Docker Container für Build und Test werden, falls nicht anders definiert, direkt aus Docker Hub[9], identifiziert über den Imagenamen, heruntergeladen.

Als Build- sowie Dependency Management Tool wird Gradle eingesetzt, welches ebenfalls zum Starten der Tests und Test Coverage verwendet wird.

Für das automatische und kontinuierliche Testen des Codes wird das JUnit 5 Framework verwendet. Dabei liegen die definierten Tests unter src/test/*. Die Testmethoden müssen zunächst für die jeweilige Klasse definiert und geschrieben und ab dann bei jedem Commit mit Änderungen am Quellcode oder Build System ausgeführt werden. Um schneller schlichte Tests zu schreiben, speziell für Klassen, die auf eine Netzwerkverbindung angewiesen sind, wird Mockito als Framework verwendet, um Mock Klassen zu definieren, die echte Klassen des Projektes imitieren.

Zur Bewertung der Test Coverage, das heißt wie viel Prozent und welche Zeilen des Programmcodes mit Tests abgedeckt sind, wird JaCoCo verwendet. Hierbei wird von der GitLab CI neben dem Anstoßen der Unit Tests auch ein JaCoCo Test Report angestoßen, welcher eine XML-Datei generiert, die wiederum im Anschluss in ein von GitLab lesbares Format umgewandelt wird (cobertura XML). Somit kann in jedem Merge Request in den File Changes eingesehen werden, ob eine Zeile des Codes von einem Test abgedeckt ist oder nicht. Außerdem wird in jedem Durchlauf der Test Stage einer Pipeline die Test Coverage in

[9] https://hub.docker.com/

Prozent als Metrik angehängt. Der Docker Container hierfür kommt nicht, wie bei den anderen Pipeline Stufen, aus Docker Hub, sondern wird aus der GitLab Container Registry heruntergeladen und eingebunden. Das Image wird hier von einem GitLab Core Mitglied gepflegt[10].

Die Reports, die bei den Durchläufen der GitLab CI generiert werden, werden außerdem als Artefakte hochgeladen und können so eingesehen werden. Dabei handelt es sich um HTML Dokumente, die eine Übersicht über durchlaufene Tests, sowie deren Ergebnis liefern und angeben, wie viel Prozent des Codes in welchen Klassen von Tests abgedeckt wird.

Die GitLab CI Pipelines werden lediglich bei Änderungen in den entsprechenden Source Codes, Test-Verzeichnissen oder gradle Dateien angestoßen. D.h. eine Änderung von z.B. der README Datei des Projektes führt zu keinem Pipeline Durchlauf. So werden unnötige Pipeline Durchläufe vermieden.

[10] https://gitlab.com/haynes/jacoco2cobertura