DevOps-Dokument

-

Verzahnung von Development und Operations

Version 1.1

Inhalt

1. Zielsetzung des Dokuments

Das DevOps-Dokument beschreibt den Aufbau der für Sie entwickelten Software und erklärt grundlegende Zusammenhänge der Komponenten. Dieses Dokument soll bei der Verzahnung der Bereiche Development und Operations eine wichtige Hilfestellung bieten. Da auch während der Projektlaufzeit Änderungen an Strukturen innerhalb der Software vorgenommen werden können, behalten wir uns auch Änderungen während des gesamten Projektes am Dokument vor.

2. Änderungshistorie

Version	Datum	Änderung	Geändert von
1.0	16.11.2018	Initial	Steffen Sassalla
1.1	19.01.2019	Erweiterung KI/Android	Ella Vahle

3. Development

Die in diesem Abschnitt behandelten Themen richten sich an Entwickler. Es werden grundlegende Themen wie Architektur, Komponenten, Kommunikation der einzelnen Module und logische Zusammenhänge erklärt. Das soll auch zukünftige Entwickler in die Lage versetzen sich in der entwickelten Software zurecht zu finden und die Hintergründe der Entwicklung besser zu verstehen. Aufgrund der Ausrichtung an Entwickler werden Fachbegriffe nicht gesondert erklärt und werden als vorausgesetzt angesehen.

3.1 Gesamtarchitektur

In diesem Kapitel soll der Aufbau der Gesamtarchitektur beleuchtet und ein Einblick in die spezifischen Strukturen gegeben werden. Dabei wird zuerst auf die oberste Ebene, die Bausteine, eingegangen und Gemeinsamkeiten erläutert, um danach auf die expliziten Unterschiede der Bausteine eingehen zu können.
Wir haben uns für diese Aufteilung entschieden, um einerseits kleine Entwicklerteams bilden zu können, die auf die Anforderungen der einzelnen Bausteine spezialisiert sind und andererseits Kriterien wie Low Coupling und High Cohesion unter den Komponenten und Modulen zu erfüllen.

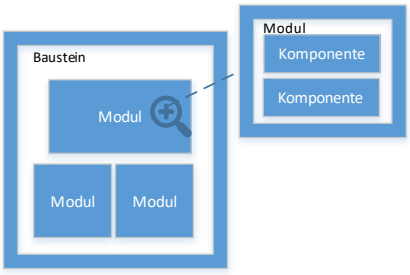

Abbildung 1 - Zusammenhang Baustein und Modul

3.2 Bausteine

Bausteine der entwickelten Software sind Spielserver, Desktop-Beobachter, Android-Beobachter und Engine-Teilnehmer (KI), welche in Einzelprojekten und in kleinen spezialisierten Teams entwickelt wurden. Jeder Baustein enthält wiederum Module.

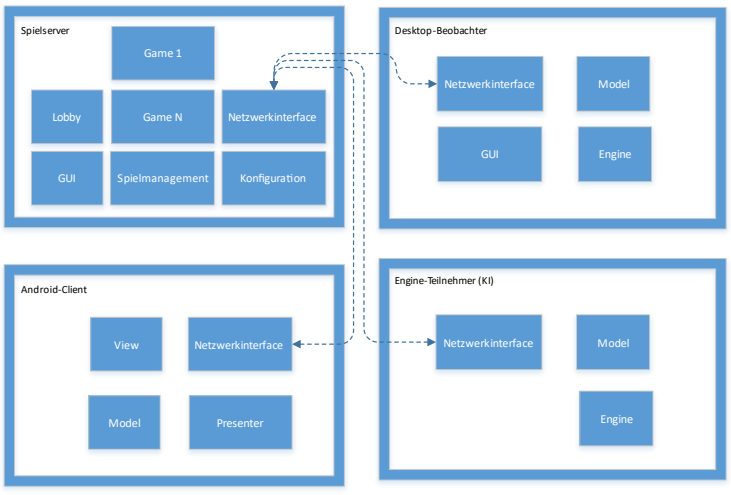

Abbildung 2 - Gesamtarchitektur

Wie in *Abbildung 2* zu sehen ist, bestehen die einzelnen Bausteine wiederum aus zusammenhängenden Modulen. Jeder Baustein enthält ein Netzwerkinterface, um die Kommunikation zwischen den Bausteinen zu garantieren. Dabei verwendet das Netzwerkinterface die im Interface-Komitee zur Verfügung gestellten Klassen. Um Redundanz zu verhindern implementiert jeder Baustein das gleiche Netzwerkinterface und verwendet die für den Baustein

wichtigen Funktionen (vgl. *Besonderheiten Netzwerkinterface*). Dadurch, dass der Spielserver eine zentrale Position durch die Verwaltung der Spiele sowie Bereitstellung und Ausführung der Spiellogik einnimmt, bildet dieser den größten Baustein mit deutlich mehr Modulen und Komponenten. Die Spiellogik besteht z.B. aus dem Validieren von Spielzügen, dem Verwalten des Spielfeldes und der Punkteberechnung.

3.3 Module

In diesem Kapitel wird genauer auf die einzelnen Module der Bausteine eingegangen.

3.3.1 Spielserver

Der Spielserver ist zusammenhängend, wie in der folgenden illustrierenden Grafik dargestellt, aufgebaut. Dabei ist es vor allem wichtig zu verstehen, wie die einzelnen Module untereinander kommunizieren und zusammenarbeiten.

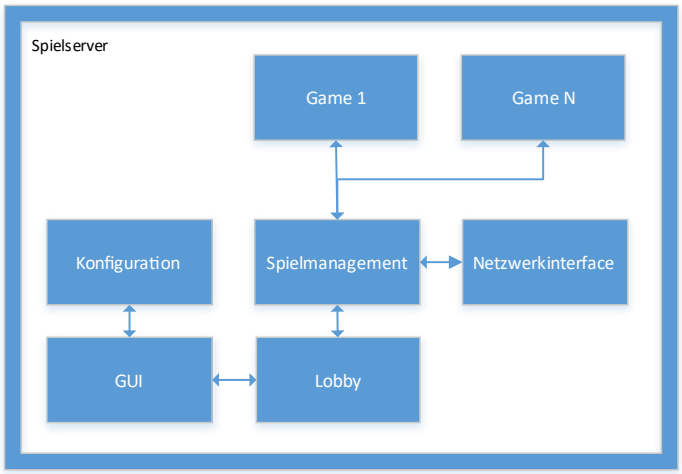

Abbildung 3 - Spielserver

Das Fundament des Spielservers ist das Spielmanagement. Es verwaltet die gesamte Verteilung der ein- und ausgehenden Messages an Games und Lobby, aber auch z.B. die Interaktion von GUI und einem bereits gestarteten Game. Wie in der Grafik bereits angedeutet, können mehrere verschiedene Spiele gleichzeitig gestartet sein, die über die GUI verwaltet werden. Wenn der Ausrichter über die GUI ein Spiel erstellt, so wird zuerst entweder eine Konfiguration geladen oder neu erstellt. Für jedes Spiel wird anschließend mit der gegebenen Konfiguration ein Thread erstellt. Zwischen den Klassen wird immer über Message-Objekte kommuniziert. Hierbei wird die Technik des Netzwerkinterfaces (vgl. *Besonderheiten Netzwerkinterface*) verwendet. Die Message-IDs und auch die Message-Klassen aus dem Interfacedokument wurden dafür erweitert (vgl. *Tabelle 1*).
Der Spielserver kann gestartet werden, ohne dass bereits ein Game gestartet ist. Um dennoch die Verbindung von Clients zu ermöglichen, wird die Lobby zur Verfügung gestellt. Verbindet sich ein Client mit dem Spielserver, so übernimmt die Lobby den Client. Dort haben sie die Möglichkeit einem Spiel über die Lobby beizutreten, oder sich zuteilen zu lassen, um je nach Typ aktiv mitspielen zu können oder ein laufendes Spiel zu verfolgen.
Die gesamte Verwaltung der einkommenden Messages über das Modul Netzwerkinterface verwaltet die Klasse *GameManagement*. Folgendes vereinfachtes Sequenzdiagramm soll die zentrale Verarbeitung der Messages verdeutlichen.

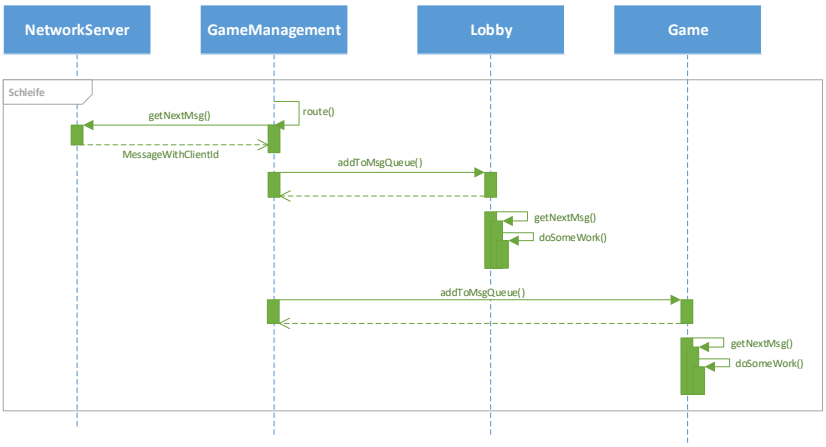

Abbildung 4 - Kommunikation über Queues

Dabei halten die Klassen *NetworkServer*, *GameManagement*, *Lobby* und *Game* jeweils eine eigene Queue in der Message-Objekte hinzugefügt und abgerufen werden können. Das entkoppelt die Komponenten voneinander und garantiert eine asynchrone Kommunikation, sodass die Threads ungestört arbeiten können.

Die GUI kommuniziert ebenfalls über Messages mit dem Spielmanagement. Möchte zum Beispiel der Ausrichter das Spiel pausieren, so sendet er ein Objekt von Typ ChangeGameState an das Spielmanagement. Dieses platziert ein neues Message-Objekt vom Typ *PauseGame* in die Queue des zugehörigen Spiels. Da das Game, wie in Abbildung 4 bereits illustriert, ebenfalls die Queue in einem gegebenen Intervall abruft, arbeitet es die Queue und somit auch das Message-Objekt ab, castet es und reagiert auf die entsprechende Anforderung. In dem Fall wird das Spiel angehalten.
In folgender Tabelle sind die neu implementierten Message-Objekte zu entnehmen:

ID	Pseudoname	Kommando
3110	ChangeGameState	Ändern des Spielstatus
1000	ClientConnect	Client verbindet sich mit Server
2120	ClientJoinGame	Client betritt Spiel
1100	Disconnect	Client trennt Verb. zum Server
2110	FinishGame	Spiel ist beendet
3140	FinishTournament	Turnier ist beendet
1400	GameStart	Starte das Spiel
3990	GameStartError	Fehler beim Starten eines Spiels
1402	GameAbort	Breche das Spiel ab
3991	GameAbortError	Fehler beim Abbrechen eines Spiels
1403	GamePause	Pausiere das Spiel
3992	GamePausedError	Fehler beim Pausieren eines Spiels
1404	GameResume	Führe das Spiel fort
3993	GameResumeError	Fehler beim Fortführen eines Spiels
3130	MoveToGame	Spieler einem Spiel hinzufügen
2130	NewGame	Neues Game erstellen
2140	NewTournament	Neues Turnier erstellen
3120	Shutdown	Beenden des Servers

Tabelle 1 - zusätzlich implementierte Message-Objekte

3.3.2 Desktop-Beobachter

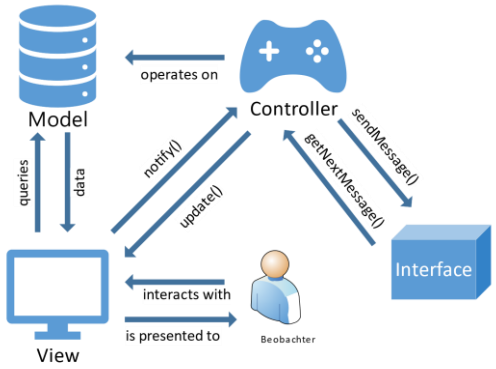

Abbildung 5 - implementiertes MVC

Der Desktop-Beobachter stellt die Anwendung zur Verfügung, die es einem Benutzer ermöglicht sich mit dem Spielserver zu verbinden und Spiele zu beobachten. Dabei basiert der Desktop-Beobachter auf dem Design-Pattern Model-View-Controller.

Für die Verarbeitung der GUI-Eingaben und deren Auswirkungen auf die Oberfläche sind die Controller Klassen zuständig. Jede Scene, gegeben durch die entsprechende FXML-Datei (vgl. *JavaFX*), besitzt einen eigenen Controller. Dieser erbt von der Klasse *AbstractSceneController*, welche die Funktionalität zum Wechsel zwischen den einzelnen Szenen *switchScene* bereitstellt. Die Instanz des konkreten Controllers wird der jeweiligen FXML-Datei zugeordnet und verwaltet die GUI-Elemente. Die Methode *initialize* ermöglicht es dem Entwickler den GUI-Elementen beim Laden der Szene bestimmte Attributwerte und Eventhandler zuzuweisen. Zur Laufzeit nimmt der Controller dann die Eingaben entgegen und führt die entsprechenden *EventHandler*, falls vorhanden, aus. Dabei agiert er sowohl mit dem Model, um benötigte Daten anzufordern/weiterzugeben, als auch mit der View um diese zu aktualisieren.

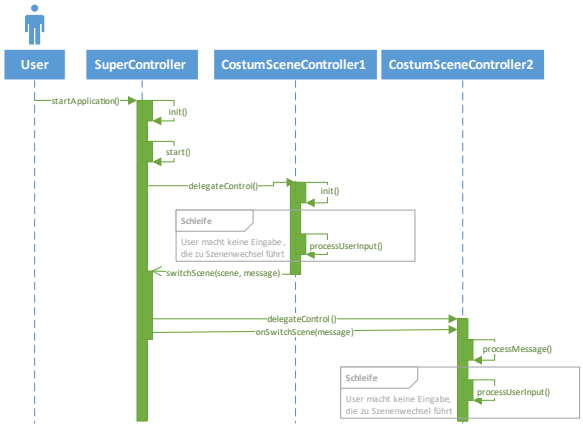

Abbildung 6 - Sequenzdiagramm Controller

Der Hauptakteur ist dabei der Supercontroller, auf welchen alle anderen Controller eine statische Referenz haben. Er kennt alle Controller, ordnet sie ihren FXML-Dateien zu und übernimmt die Aufgabe das Anzeigen und Wechseln der Szenen zu verwalten. Mittels der *DisplayedScene* kann der Supercontroller den Fokus bzw. die Kontrolle an eine Controllerinstanz, dessen Szene angezeigt werden soll, abgeben. Geschieht ein Szenenwechsel, so teilt die aktuelle Szene dies dem Supercontroller mit und gibt die Kontrolle an ihn zurück. Dieser aktualisiert die aktuelle Szene und delegiert die Kontrolle an diese. Dabei hat die aktuelle Szene die Möglichkeit eine beliebige Nachricht mitzugeben, welche der Supercontroller entgegennimmt und mittels *onSceneSwitch* an die neue Szene weitergibt, wo sie verarbeitet wird.

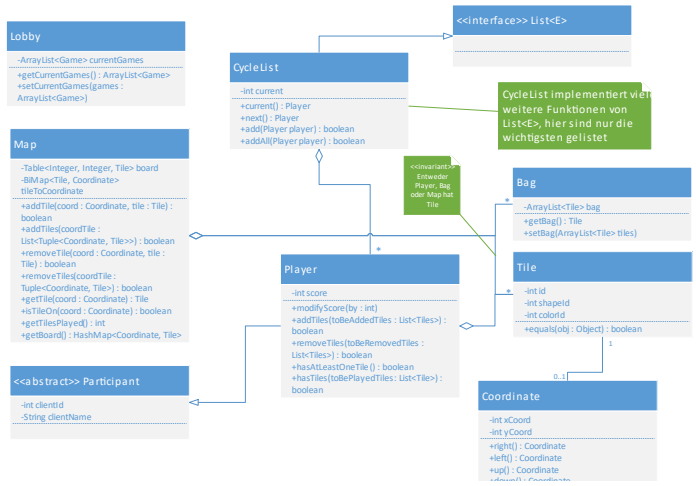

Abbildung 7 - grundlegende Klassen des Spielmodells

Beim Start des Programms initialisiert der *SuperController* die Referenzen sowie die erste Szene und zeigt diese an.

Im Verzeichnis *src/main/java* im Package Model werden alle für das Model benötigten Klassen zur

Verfügung gestellt. Die Klasse Lobby wird lediglich im *LobbySceneController* verwendet und enthält die auf dem Server verwalteten Spiele. Die CycleList implementiert das Interface List und dient dazu, alle Spieler zu verwalten und diese mit der Methode next() zyklisch zu durchlaufen. Die Map hält mithilfe einer Table die gesamten gelegten Tiles.

3.3.3 Engine-Teilnehmer (KI)

Der Engine-Teilnehmer (auch künstliche Intelligenz) ist ein Programm, das eigenständig als Spieler eingesetzt werden kann. Er analysiert das Feld sowie die Züge der anderen Spieler und führt daraufhin nach eigenen Berechnungen einen Zug durch.

Der Engine-Teilnehmer basiert auf einem abgewandelten Model-View-Controller- und dem Strategy-Pattern. Da die KI vollkommen eigenständig arbeitet, benötigt es keine Oberfläche. Die View findet man in der Implementierung aus diesem Grund nur in einer stark abgeschwächten Form. Man findet sie im weiteren Sinne als Terminal, an welchem die KI gestartet wird und dann Informationen über gemachte Züge, Punktzahlen und Spielausgang in textueller Form angezeigt werden. Hinzu kommt noch das Modul Netzwerkinterface, welches für die Kommunikation mit dem Server verantwortlich ist.

Der Parser verarbeitet die vom Server gesendeten Nachrichten und macht daraufhin entsprechende Aufrufe auf dem Controller der KI. Der Controller selbst führt keine Berechnungen durch, sondern delegiert diese an die AILogic, das Model dieses Bausteins. Die AILogic hält alle für die KI relevanten Informationen, wie das Spielfeld oder die Steine im Besitz des Engine-Teilnehmers. Auf Grundlage dieser Daten berechnet die KI dann mithilfe des im Model festgelegten Algorithmus einen möglichst optimalen Zug. Diesen sendet der Controller mithilfe des Networkclients zurück an den Server und beendet somit den Zug.

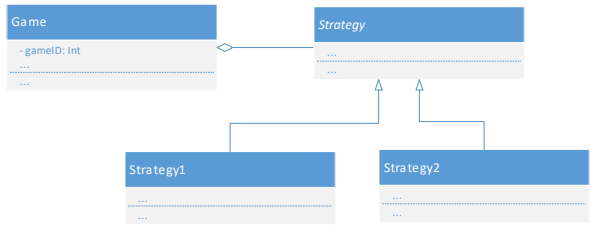

Abbildung 8 – Strategy-Pattern

Innerhalb des Models haben wir uns an dem Strategy-Pattern (siehe Abbildung TODO) orientiert. Das Pattern kommt in einem bestimmten Kontext zum Einsatz, in unserem Fall innerhalb eines Spiels. Für ein Spiel hat der Engine-Teilnehmer grundlegende Informationen und Algorithmen wie beispielsweise die reine Validierung eines möglichen Spielzugs, ohne dabei Ziele wie hohe Punktzahlen zu verfolgen. Diese sind in der abstrakten Klasse Strategy festgehalten. Die einzelnen davon erbenden Strategien sind dann Spezialisierungen davon. Eine Strategie kann beispielsweise auf Effizienz, Schnelligkeit oder hohe Punktzahlen abzielen oder sich durch ihre Vorgehensweise auszeichnen. Aktuell ist in unserer Version nur eine Strategie implementiert, was den Nutzen des Patterns in Frage stellt. Wir haben uns dennoch für das Strategy-Pattern entschieden, um eventuelle Erweiterungen (bspw. für die Turnierversion) leicht vornehmen zu können.

Der Enginge-Teilnehmer prüft für einen Spielzug alle möglichen Felder. Dabei geht er zunächst vertikal durch das bereits belegte Spielfeld, wobei er lediglich die Felder direkt neben äußeren Steinen betrachtet, da lose Steine nicht erlaubt sind. Hat die KI (mindestens) einen Stein auf der Hand, welcher auf die Position gelegt werden kann, so probiert so möglichst weit horizontal anzulegen. Im Anschluss daran probiert sie erlaubte Kombinationen aus horizontalem und vertikalem Legen. Wurden alle Möglichkeiten für ein Feld als Ausgangspunkt für den Zug probiert so macht geht

der Algorithmus zum nächsten Feld über und wiederholt die Prozedur. Nachdem das Spielfeld vertikal durchlaufen wurde, wird es analog horizontal durchlaufen. Das Resultat der Berechnung ist der Kombination mit den meisten Punkten.

3.1.4 Android-Client

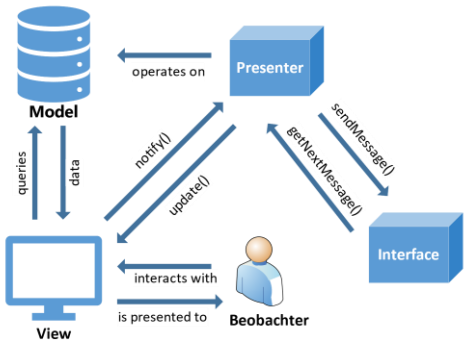

Abbildung 10 - MVP

Der Android-Client stellt die Anwendung zur Verfügung, die es einem Benutzer ermöglicht sich mit dem Spielserver zu verbinden und Spiele zu beobachten oder zu spielen. Dabei basiert der Android-Client auf dem Design-Pattern Model-View-Presenter (MVP) (siehe Abbildung 9) . Der Presenter ist dabei das Bindeglied zwischen Netzwerkkommunikation, Model und View. Die Netzwerkkommunikation ist stark an die Netzwerkkommunikation des Desktop-Clients angelehnt. So werden alle eingehenden Messages von der Netzwerkkommunikation direkt an den Presenter weitergeleitet, der aufgrund der eindeutigen ID der Message an die entsprechende aktive Activity weiterleitet. So werden auch alle gesendeten Nachrichten über den Presenter an die Netzwerkkommunikation weitergeleitet. Dabei implementiert jede View ein eigenes Controller-Interface. Schematisch soll das der Verbindungsaufbau in dem nebenstehendem Sequenzdiagramm veranschaulicht werden. In dem Controller-Interface sind die Methoden definiert, die für die Verarbeitung der einkommenden Messages benötigt werden. Alle Controller-Interfaces erben wiederum von der Klasse Controller, die Methoden für die Ausnahmebehandlung von Messages mit der ID 900, 910 oder 920 zur Verfügung stellen. Diese Methoden müssen dann in der jeweiligen View implementiert und definiert werden. So ist jede View gezwungen auch auf Ausnahmen und Fehler des Servers umzugehen. Zudem weiß der Presenter zu jeder Zeit, welche View gerade aktiv ist. Das gesamte Model ist ebenfalls stark an das Model des Desktop-Clients angelehnt.

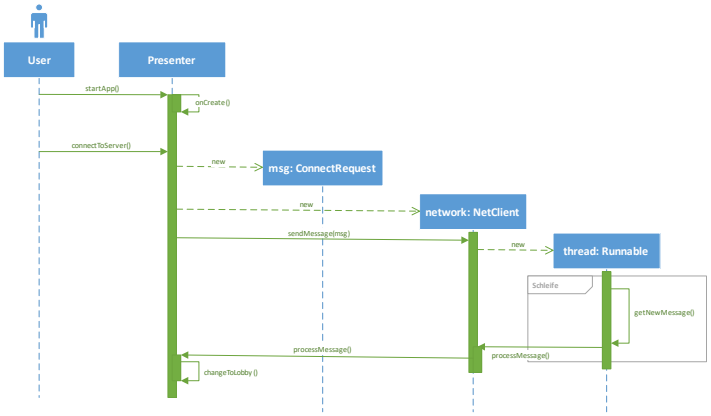

Abbildung 9 - Verbindungsaufbau im Android-Teilnehmer

Die Elemente der View repräsentieren hierbei die GUI. Eine „Anzeige“ auf dem Android enthält jeweils immer eine Activity. Eine Acitivty wiederum kann aus sogenannten Fragements bestehen. Gibt es einzelne GUI-Elemente in einer Activity, die speziell gekapselte Features implementieren, so werden diese ausgelagert in Fragments. Das ist zum Beispiel im Spielmodus oder auch im Beobachtermodus der Fall. Hier gibt es die gekapselten Features Chat (Chatten mit anderen Spielern oder Beobachtern), Board (das eigentliche Spielfeld), Bag (Anzeige der im Beutel befindlichen Steine) und Hands (Anzeige der Steine des jeweiligen Spielers im Beobachtermodus oder im Spielmodus die eigene Hand mit der Möglichkeit Steine auf das Spielfeld zu legen).

3.1 Besonderheiten Netzwerkinterface

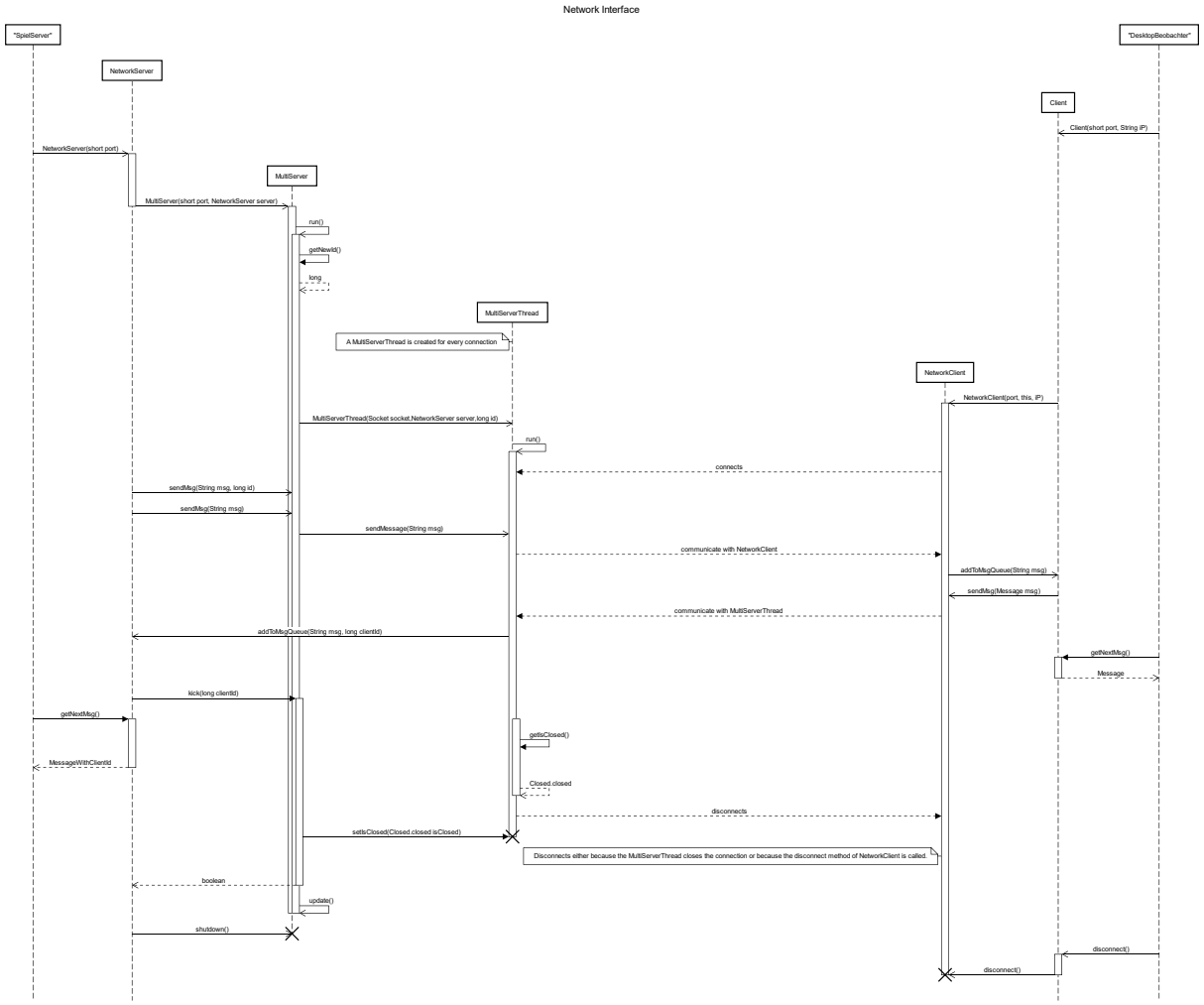

Abbildung 11 - Netzwerkkommunikation

Die Kommunikation zwischen dem Spiel Server und Desktop-Beobachter bzw. Engine-Teilnehmer erfolgt über die vom Netzwerk Interface hergestellte Netzwerkverbindung. Diese nutzt TCP, um serialisierte Message Objekte über die genutzten Sockets zu schicken. Diese Message Objekte werden vom Parser, der vom Interface-Komitee bereitgestellt wurde, vor dem Senden serialisiert und nach dem Empfangen wieder deserialisiert.

Auf Serverseite wird für jeden Client ein Thread erstellt, der die Verbindung mit dem Client verwaltet und entsprechende Nachrichten an den Client schickt. Nachrichten, welche vom Client empfangen werden, werden von diesen Threads der Message-Queue des Network Servers hinzugefügt. Von dieser werden die Nachrichten von den entsprechenden verarbeitenden Komponenten abgerufen. Damit der Server die Verarbeitung nicht stört, ist der gesamte Server in einem eigenen Thread (Klasse *MultiServer*) untergebracht.

Auf der Seite des Clients ist der Aufbau gleich, nur dass kein extra Thread die Verbindung verwaltet, da nur eine Verbindung existieren kann. Der genaue Inhalt und Ablauf der Kommunikation kann aus dem Interface-Dokument entnommen werden.

3.2 Verwendete Technologien

3.2.1 JavaFX und CSS

Zum Gestalten und Ausführen von Oberflächen wird in diesem Projekt das Framework JavaFX benutzt. In dem Framework können Oberflächen mittels Programmcode entwickelt oder durch FXML-Dateien vordefiniert werden. In diesem Projekt werden hauptsächlich FXML-Dateien benutzt, um Oberflächendesign und Programmcode trennen zu können. Abgesehen von einzelnen Teilen der Oberfläche, die zur Laufzeit verändert werden müssen, wie zum Beispiel die Größe des Spielfeldes, ist die gesamte Oberfläche durch die FXML-Dateien vordefiniert. Eine optionale Möglichkeit zum intuitiven und leichten Bearbeiten mittels Drag and Drop bietet der Java-SceneBuilder.
Um das Aussehen der Oberfläche zu verändern, benutzen wir Cascading Style Sheets (CSS). Diese dienen dazu Quellcode und GUI Style voneinander zu trennen. Die Regeln für das Aussehen stehen dazu in CSS-Dateien, welche im Programmcode der JavaFX-Scene hinzugefügt wird. Wie zum Beispiel das Rot Färben eines Inputfields nach falscher Eingabe, werden nur einzelne Regeländerungen, die zur Laufzeit geschehen sollen, im Programmcode verändert.
Allgemeine Style-Regeln, die für alle Oberflächen gelten sollen, stehen in der allgemeinen CSS-Datei (application.css), die jede Oberfläche benutzt. Für Oberflächen die mehr Style Regeln benötigen als die Regeln aus der allgemeinen CSS-Datei, wird eine nur für diese Oberfläche erstellte Datei verwendet. Diese Datei wird nach der Oberfläche benannt. In spezifischen CSS-Dateien kommen dann nur die Regeln, die für die Oberfläche gebraucht werden. Diese Aufteilung dient der besseren Übersicht und der besseren Wart- beziehungsweise Änderbarkeit des Designs der GUI.

3.2.2 Android und GUI

Im Android verwenden wir die Built-In Funktionalitäten für die Erstellung der GUIs. Dabei werden XML-Dateien in Kombination mit dem Built-In-Gestaltungstool von Android-Studio verwendet.

4. Operations

Die in diesem Abschnitt angesprochenen Kapitel richten sich hauptsächlich an die ausführende IT. Dabei wird unter anderem die Verhaltensweise der Software im Hinblick auf Interaktion mit dem Benutzer erläutert, aber auch die Arbeitsweise mit Maven wird erklärt.

4.1 Installations- und Ausführungsdetails

Die Ausführung von der entwickelten Software benötigt das Java Runtime Environment und Java Development Kit in der Version 1.8.0 oder höher, um fehlerfrei funktionieren zu können. Es bietet somit die Möglichkeit die Software auf beinahe allen Betriebsgeräten zu starten. Dies stellt die von Ihnen gewünschte Unabhängigkeit vom Betriebssystem sicher. Für den Android-Client benötigt es mindestens eine API-Level 21 oder höher (ab Android 5.0.1). Wichtig ist, dass unter Windows die Umgebungsvariable Java definiert ist. Zudem ist eine funktionierende Netzwerkverbindung notwendig, damit über das Netzwerk gespielt werden kann.
Der Desktop-Beobachter kann mittels der executable „Qwirkle.jar“ gestartet werden, der Spielserver wiederum über „Qwirkle-Ausrichter.jar“ und die KI über „Qwirkle-KI.jar“. Es bedarf keiner weiteren Konfiguration.
Für den Android-Client muss die APK auf das entsprechende Android-Gerät geladen werden. Hierbei ist zu beachten, dass für die Installation eine Installation von Software außerhalb des Google-Play-Stores erlaubt sein muss. Diese Einstellung finden Sie in der Regel unter „Einstellungen“. Für genauere Angaben schauen Sie bitte in das Handbuch Ihres Handys. Nach erfolgreicher Installation wird die Qwirkle-App unter den anderen auf Ihren Handys installierten Apps angezeigt.
Sollte die „.jar“ für den Desktop-Beobachtersich nicht per Doppelklick starten lassen, so nutzen Sie die Kommandozeile. Unter Windows nennt sich das Programm „Eingabeaufforderung“, unter Linux „Terminal“ und unter MacOS „Bash“. Dort müssen sie zunächst mit dem Befehl „cd *Dateipfad*“ (bspw. „cd C:/Users/Benutzer/Dokumente“) in das Verzeichnis wechseln, in welchem die Datei liegt und können Sie dann mittels „java –jar *Dateiname*“ starten. Für die KI ist es wichtig Argumente anzugeben. Die KI wird nach dem Muster

„java -jar -ip=<IP-Adresse Server> -port=<Port> -name=<Spielernamen>“

gestartet.

4.2Technische Voraussetzung für die Entwicklung

4.2.1 Entwicklungsumgebung

Die gesamte zu programmierende Software für den Desktop wurde mithilfe der IDE Eclipse entwickelt. Eclipse ist jedoch keine Voraussetzung für die Weiterentwicklung, da auch sonstige Java-fähigen Entwicklungsumgebungen genutzt werden können. Ausgenommen davon sind die verwendeten Plugins EclEmma und ObjectAid (vgl. *Hinweise für die Weiterentwicklung*), die nur mit Eclipse funktionieren, jedoch nicht essentiell für eine weitere Entwicklung benötigt werden.

Für den Android-Teilnehmer wurde die Entwicklungsumgebung Android Studio verwendet.

4.2.2 Maven und Gradle

Maven, wenn es nicht mittels einer davon unterstützen IDE verwendet wird, bedarf eines Downloads von der offiziellen Internetseite. Die Version muss 3.5.3 oder höher sein. Für die Entwicklung ist es wichtig, dass folgende Befehle unterstützt werden: *validate*, *compile*, *test*, *package*, *integration-test*, *verify*, *install* und *clean*. Diese Befehle können durch die Konsole oder per Oberfläche der Entwicklungsumgebung genutzt werden. Weiteres zur Funktion der Befehle ist im Build-Prozess zu

finden (vgl. *Build-Prozess*). Ein Befehl beginnt immer mit der Folge *„mvn“*, gefolgt von dem eigentlichen Befehl und dahinterstehenden Parametern.

Für den Android-Beobachter wurde hingegen Gradle genutzt. Gradle dient dem gleichen Zweck wie Maven, kommt allerdings ohne oben genannte Befehle aus. Stattdessen erstellt es beim builden automatisch ein ausführbares Package.

4.2.3 Build-Prozess mit Maven

Der Build-Prozess startet mit der Vollendung aller Code-Reviews, die dem Release zugeordnet sind. Er wird erst dann ausgeführt, wenn die für den Release notwendigen User-Stories gemäß Definition-of-Done abgeschlossen sind. **Wichtig:** Bevor der Build-Prozess gestartet wird, muss das Projekt über mvn update geupdated werden, damit alle notwendigen Resourcen erstellt werden. Anschließend werden folgende Maven-Befehle ausgeführt:

- **clean:** beseitigt erstelle Artifacts[1] aus alten Builds
- **validate**: validiert das Projekt auf Korrektheit und Verfügbarkeit aller notwendigen Informationen
- **compile**: kompiliert den Quelltext des Projektes
- **test**: testet den kompilierten Quelltext durch ein passendes *„unit testing framework“*[2], diese Tests sollten keinen *„packaged“* oder *„deployed“* Quelltext verlangen
- **package**: verpackt den kompilierten Quelltext in ein verarbeitbares Dateiformat
- **integration-test**: verarbeitet und erstellt die Packages, sofern benötigt, in eine Umgebung in der Integration-Tests ausgeführt werden können
- **verify**: führt jegliche Tests zur Validierung der Integrität des Packages und der Qualitätssicherungskriterien aus
- **install**: installiert das Package in das lokale Repository zur Nutzung als Abhängigkeit für andere lokale Projekte

Nach erfolgreichem Abschluss dieser Prozedur ist ein Build erstellt, der dann in GitLab commited wird. Sofern die Prozedur fehlschlägt, wird kein Build erstellt und die auftretenden Probleme müssen behandelt und behoben werden.

Die notwendigen Befehle für Gradle werden automatisch beim Ausführen oder Builden der App in Android-Studio durchgeführt. Es sind keine speziellen Befehle auszuführen.

4.2.4 JavaFX

Um mit JavaFx in Eclipse zu arbeiten, ist es empfehlenswert das Plugin *„e(fx)clipse“* über den integrierten Marketplace zu installieren. Zusätzlich bietet es sich an, den JavaSceneBuilder von Oracle zu verwenden, um an den Oberflächen übersichtlich arbeiten zu können. Dabei ist zu beachten, dass Version 2.0 benutzt wird, da die Versionen untereinander nicht kompatibel sind. Nach Installation muss in Eclipse der Installationspfad von JavaSceneBuilder angegeben werden. Dafür wählt man im Reiter am oberen Rand das Menü „Window“. In der Liste am linken Rand klickt man dann JavaFX an. Für das Textfeld „SceneBuilder executable“ wählt man dann den Pfad der Ausführungsdatei des

[1] Mit Artefakt sind im Zusammenhang mit Maven Arbeitsergebnisse gemeint. Maven-Projekte haben in der Regel ein Hauptergebnis-Artefakt, oft eine `jar`-, `war`- oder `ear`-Datei. Aber auch andere Ergebnisse wie Projektdokumentationen können als Artefakt bezeichnet werden. Quelle: https://www.torsten-horn.de/techdocs/maven.htm abgerufen am 08.12.2018

[2] in diesem Projekt wird das JUnit-Framework verwendet

SceneBuilders. Wurden diese Einstellungen vorgenommen kann eine FXML-Datei über einen Rechtsklick mit dem Menüpunkt „Open with SceneBuilder“ im SceneBuilder öffnen.

Es sollte darauf geachtet werden, dass keine Styles fest über den JavaSceneBuilder angelegt werden, sondern dass diese ausschließlich in CSS-Dateien ausgelagert werden.

4.2.5 GUI in Android Studio

Android-Studio bietet die komfortable Möglichkeit über die Built-In-Funkitionalität GUIs zu gestalten um diese anschließend direkt verwenden zu können. Hierbei ist keine weitere Konfiguration notwendig.

4.3 Hinweise für die Weiterentwicklung

Eclipse ist keine Voraussetzung zur Weiterentwicklung, da auch sonstige Java-fähigen Entwicklungsumgebungen genutzt werden können. Ausgenommen dafür sind die verwendeten Plugins EclEmma und ObjectAid, welche nur in Eclipse funktionieren, jedoch nicht essentiell für eine weitere Entwicklung benötigt werden.

EclEmma ist ein Coverage-Tool und ermöglicht die Erstellung von Ausführungs-Nutzungsstatistiken, z.B. wie häufig ein Code Segment ausgeführt wird. In Eclipse kann durch die Funktion „Install New Software” unter dem Reiter „Help” das Plugin hinzugefügt werden. Dafür muss lediglich der Link *http://update.eclemma.org/* in das „work with”-Feld eingegeben werden. Der Rest wird von Eclipse nach Benutzung des Feldes „Add” automatisch installiert und konfiguriert.

EclEmma ist auch bereits in Android-Studio integriert und kann für Code-Test-Coverage-Reports verwendet werden.

Zusätzlich wurde das Tool ObjectAid verwendet, um bei der Visualisierung von Quellcode zu unterstützen. Das Tool wird, wie bei EclEmma bereits erwähnt, ebenfalls über Eclipse installiert. Hierzu wird folgender Link verwendet: *http://www.objectaid.com/update/current*. Um die Installation abzuschließen, ist es notwendig einen Account auf der oben genannten Webseite zu erstellen, um dort eine Lizenz generieren zu können. Diese Lizenz wird anschließend im Eclipse unter *Preferences* zu ObjectAid hinzugefügt. Danach muss Eclipse neugestartet werden.

4.4 Hinweise für die Benutzung der Software

4.4.1 Spielserver

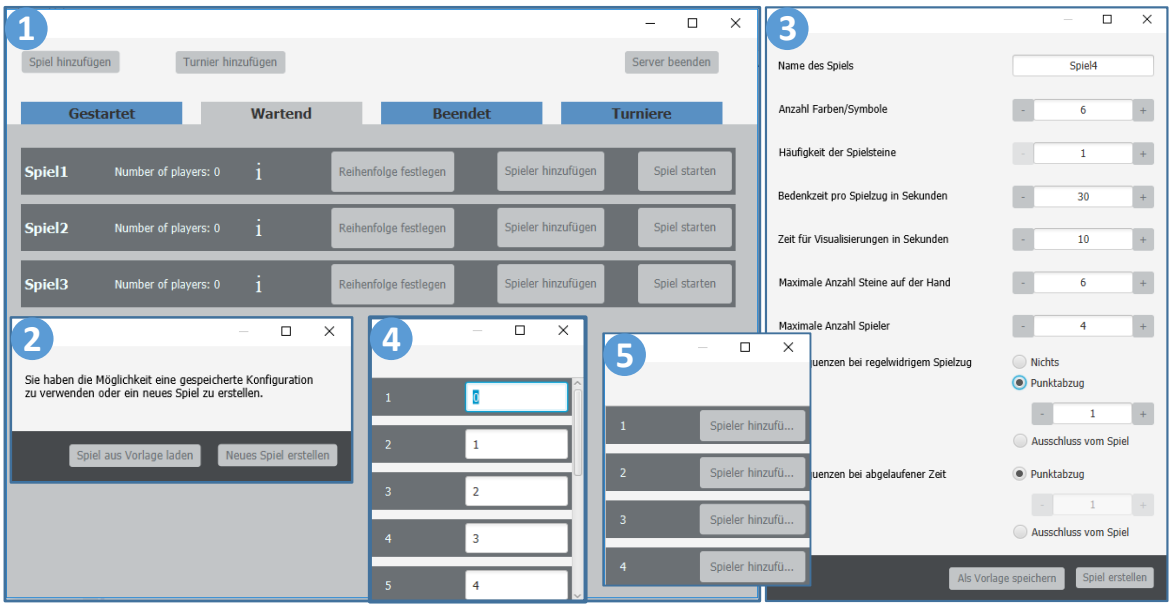

Abbildung 12 - GUI Server/Ausrichter

Wird ein Server gestartet so befindet man sich zunächst in der Lobby (1). Hier sieht der Ausrichter alle Spiele unterteilt in Laufend, Wartend, Beendet und Turniere. Es besteht die Möglichkeit ein neues Spiel oder Turnier hinzuzufügen. Hierfür öffnen sich jeweils gesonderte Fenster, in denen die nötigen Einstellungen vorgenommen werden können. Möchte man ein Spiel hinzufügen, so kann man aus einer Vorlage wählen oder ein komplett neues Spiel erzeugen (2). Im „Spiel erzeugen"-Dialog (3) können über Buttons als auch Textfelder die gewünschten Eingaben gemacht werden. Zur Verfügung stehen die in der Product Vision und im Interface-Dokument spezifizierten Optionen. Dabei werden Eingaben automatisch überprüft, sodass auf Fehler hingewiesen wird und das Speichern sowie Erstellen eines Spiels mit falschen Eingaben nicht möglich ist. Möchte man die vorgenommenen Einstellungen speichern, so gelangt man mittels „Vorlage speichern" in einen dem Betriebssystem entsprechenden „Datei speichern" Dialog. Mit „Spiel erstellen" wird ein Spiel mit entsprechenden Werten erzeugt und der Nutzer gelangt wieder in die Lobby, wo das Spiel in der Kategorie „Wartend" aufgeführt wird. In der Lobby kann ein erstelltes Spiel verwaltet werden, bevor es gestartet wird (5 und 6).
Wenn Sie mittels des „Turnier hinzufügen" Buttons ein Turnier hinzufügen möchten, haben Sie die gleichen Konfigurationsmöglichkeiten wie bei einem Spiel, mit dem Unterschied, dass jedes Turnierspiel aus zwei Spielern besteht und man stattdessen die maximale Anzahl an Teilnehmern des gesamten Turniers einstellen kann.
Die Adresse des Spielservers ist die im LAN vom i.d.R. DHCP vergebene Adresse, sprich die Adresse des Computers. Falls Verbindungen aus dem WAN möglich sein sollen, so muss der Router entsprechend mit Firewall- und Routingregeln versehen werden. Die IP des Spielservers ist somit die öffentliche IP-Adresse im Internet. Der Spielserver ist aktuell fest auf Port 33100 eingestellt. Im Laufe der Entwicklung wird eine Konfigurationsmöglichkeit implementiert.
Für detailliertere Informationen schauen Sie in unser Benutzerhandbuch.

4.4.2 Desktop-Beobachter

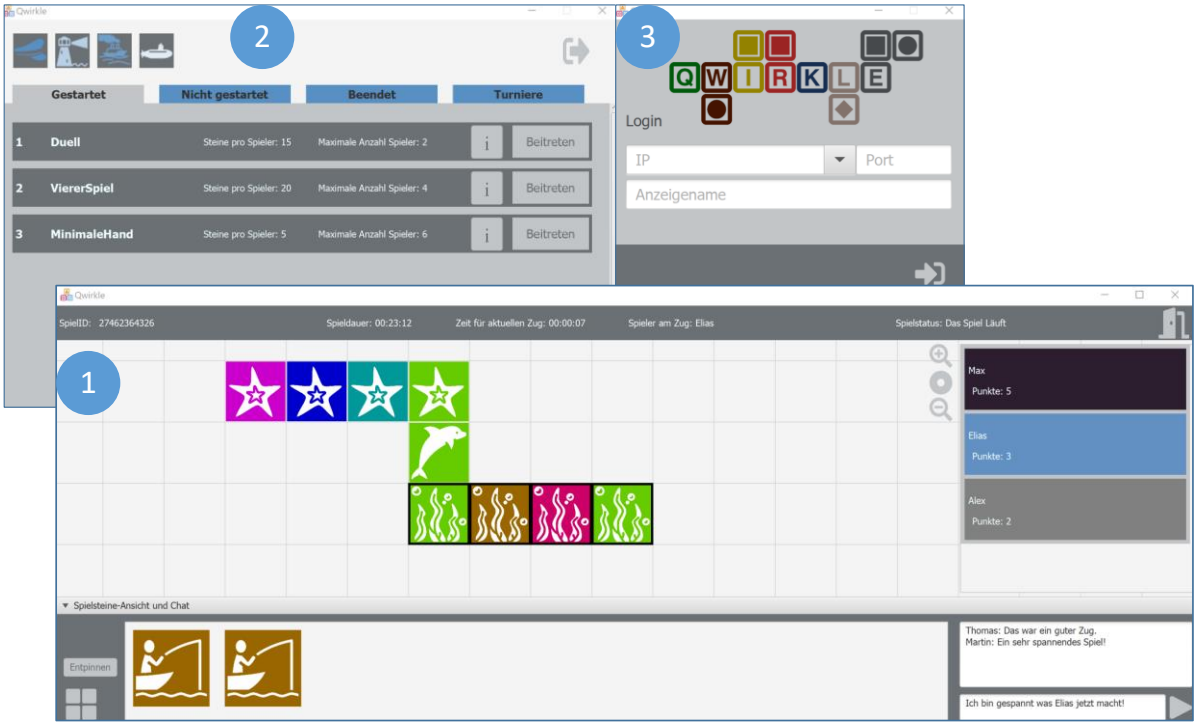

Abbildung 13 - GUI Desktopbeobachter

Wird der Desktop-Beobachter gestartet, so wird zuerst das Login-Fenster (3) geöffnet. Hier kann die gewünschte IP entweder ausgewählt oder eingetippt werden, wobei die zuletzt eingegebenen Adressen über ein Drop-Down-Menü zur Verfügung stehen. Kann man sich nicht mit dem Server verbinden, so wird eine Fehlermeldung ausgegeben.
Nach erfolgreichem Starten gelangt der Client in die Lobby (2), wo er die vom Server gehosteten Spiele angezeigt bekommt. Die Konfiguration eines Spiels kann über den Button „i“ in einem separaten Fenster angeschaut werden. Über den Button „Beitreten“ kann der Client einem Spiel als Beobachter beitreten und wird zum Spielfeld (1) weitergeleitet. Hier sieht er dann die am Spiel teilnehmenden Spieler, die bereits gelegten Steine sowie die Hand eines beliebigen Spielers und den Beutel. Weiterhin kann der Spieler sich die verbleibende Bedenkzeit anzeigen lassen und über den Chat mit seinen Beobachtern chatten.
Für detailliertere Informationen schauen Sie in unser Benutzerhandbuch.

4.4.3 Android-Teilnehmer

Wird der Android-Teilnehmer gestartet, so wird zuerst das Login-Fenster geöffnet. Hier kann die gewünschte IP eingegeben werden. Kann man sich nicht mit dem Server verbinden, so wird eine Fehlermeldung ausgegeben.
Nach erfolgreichem Starten gelangt der Client in die Lobby, wo er die vom Server gehosteten Spiele angezeigt bekommt. Die Konfiguration eines Spiels kann über den Button „Config“ Spiel als Beobachter beitreten und wird zum Spielfeld weitergeleitet. Hier sieht er dann die am Spiel teilnehmenden Spieler, die bereits gelegten Steine sowie die

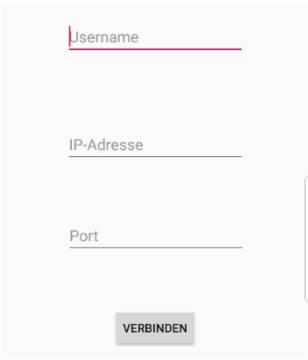

Abbildung 14 - Login

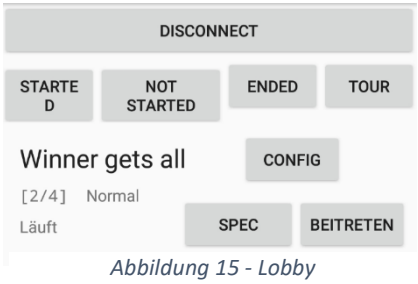

Abbildung 15 - Lobby

Hand eines beliebigen Spielers und den Beutel. Weiterhin kann der Spieler sich die verbleibende Bedenkzeit anzeigen lassen und über den Chat mit seinen Mitspielern chatten.
Für detailliertere Informationen schauen Sie in unser Benutzerhandbuch. Über den Button „Beitreten“ kann der Client selbst als Spieler beitreten. Für weitere Informationen kann das Benutzerhandbuch zur Hilfe genommen werden.

4.5 Ansprechpartner

Name	**E-Mail**	**Zuständigkeit**
Steffen Sassalla	sassalla@mail.upb.de	DevOps und Projektleitung
Dennis Suermann	deniss2@mail.upb.de	GUI und Design
Tim Dahm	dahmt@mail.upb.de	IDE, Maven und Buildprozess
Linus Jungemann	linusjun@mail.upb.de	Koordination Spielserver, Netzwerkkommunikation
Artem Burchanow	artemb@mail.upb.de	Koordination Desktop-Beobachter

Tabelle 2 - Ansprechpartner