Softwareprojekt:

Dokumentation

Gruppe B

Wintersemester 2019/20 und

Sommersemester 2020

Spielimplementierung: Dominion

Stand:

25. Februar 2021

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Dokumentation behandelt die softwareseitige Implementierung des Brettspielklassikers „Dominion“ im Softwareprojekt 2019/2020. Die Softwareentwicklung im Softwareprojekt wurde mithilfe der agilen Methode Scrum durchgeführt.

In den nachfolgenden Absätzen werden wir Ihnen einen kurzen Einblick in die verschiedenen Kapitel dieser Dokumentation geben.

Das Ziel des Softwareprojektes war es, eine digitale Spielversion des Brettspiels „Dominion“ mittels agiler Methoden zu entwickeln. Die Besonderheit in dieser Iteration des Softwareprojektes war, dass wir ein Grundgerüst gegeben hatten und auf Basis eben dieses das Spiel entwickeln mussten.

Das erste Kapitel behandelt die Anforderungsanalyse, welche wir zu Beginn des Wintersemesters durchgeführt haben. Hier gehen wir insbesondere auf die Meilensteine und eigene Anforderungen, die wir ermittelt haben, ein.

Das zweite Kapitel behandelt die Umsetzung des Softwareprojektes über die beiden Semester. Zunächst werden wir dem Leser die grundlegenden Konzepte und dann die verschiedenen Module unserer Software erläutern.

Das dritte Kapitel beinhaltet die Testdokumentation. Als Erstes gehen wir auf unser Testvorgehen und den grundlegenden Ablauf, also unserem Testschema, ein. Darauf folgend gehen wir auf die von uns entwickelten Tests sowie die Testabdeckung im gesamten Projekt ein. Wir erläutern ebenfalls, wieso bestimmte Tests nicht implementiert wurden und gehen zuletzt auf das durchgeführte Codereview ein.

Im vierten Kapitel gehen wir auf die Arbeitsorganisation innerhalb der Gruppe ein, beginnend mit der Rollenbeschreibung jedes Mitglieds. Wir stellen weiterhin vor, wie wir uns innerhalb des ersten Semesters und insbesondere während des digitalen Semesters organisiert haben. Außerdem gibt es einen Rückblick über die zehn Sprints, welche wir im Softwareprojekt durchgeführt haben. Zuletzt wird auf die verwendeten Frameworks, Bibliotheken und Tools eingegangen.

Es folgt darauf ein Ausblick, wie sich das Projekt theoretisch weiter entwickeln könnte und ein Fazit der Gruppe. Das Fazit beinhaltet einen Rückblick, welcher sich damit beschäftigt, was wir gelernt haben, über die Höhen und Tiefen und Feedback zum Softwareprojekt.

Im Anhang findet sich die Spielanleitung.

Kapitel 2

Analyse

In diesem Kapitel geht es um die Anforderungsanalyse und um die umgesetzten User Stories.

Bei der Anforderungsanalyse wird zwischen den vorgegebenen Anforderungen bezogen auf die einzelnen Meilensteine und den Anforderungen, welche die Gruppe selber an das Spiel stellt, unterschieden.

Die eigenen Anforderungen sind hierbei in mehrere Abschnitte geteilt, wie der Login-/Register-Screen, das Hauptmenü, die Lobby, das Spiel und Grundsätzliches.

2.1 Anforderungsanalyse

2.1.1 Vorgegebene Anforderungen

In den folgenden Unterabschnitten werden die vorgegbenen Anforderungen erläutert.

2.1.1.1 Meilenstein 1: Basisarchitektur mit UserManagement

Dieser Meilenstein beinhaltet das anfängliche Zurechtfinden mit der Basisarchitektur des Softwareprojektes 2019. Die Speicherung der Nutzer soll am Anfang im Hauptspeicher erfolgen, damit bei einem Neustart keine Nutzer neben den Vordefinierten vorhanden sind. Der Nutzer soll in der Lage sein, sich für das Spiel zu registrieren, ggf. mit einem Passwort und einem Nutzernamen. Nutzer, die bereits registriert sind, müssen in der Lage sein, sich mit ihren Daten anmelden zu können und nach dem Einloggen in das Hauptmenü gelangen, wo sie alle anderen eingeloggten Spieler sehen können.

2.1.1.2 Meilenstein 2: Hauptmenü, Chat, Spiel-Ansätze

Zur Bewältigung des zweiten Meilensteins musste verstanden werden, wie die Kommunikation zwischen Server und Client erfolgt. Die Nutzer sollten nach dem Anmelden in das Hauptmenü gelangen und dort alle Nutzer sehen können, die ebenfalls angemeldet sind. Hierfür sollte

kein Refresh-Button verwendet werden. Außerdem sollten die Nutzer sich untereinander Chat-Nachrichten schicken können. Dies sollte ebenfalls ohne einen Refresh-Button geschehen. Zuletzt wurde ein einfaches Modell des Spielmodells gefordert. Idealerweise sollen mithilfe der Konsole bereits Interaktionen durchgeführt werden können.

2.1.1.3 Meilenstein X "Prototyp"

Der letzte Meilenstein beinhaltet die Realisierung des UserManagements mithilfe einer relationalen Datenbank, wobei die Nutzerinformationen über Java modelliert und zugreifbar sein sollen. Der Zugriff soll grundsätzlich auf beliebige per JDBC ansprechbare Datenbanken erfolgen und mit möglichst wenig Aufwand austauschbar sein. Außerhalb der Klasse, die für den Datenbankzugriff verantwortlich ist, darf keine Person wissen, wie die Daten gespeichert werden. Das Spiel an sich muss so weit funktionstüchtig sein, dass der Nutzer ein Spiel mit sich oder einem Bot spielen kann, aber es dürfen noch Fehler vorhanden und die Bedienbarkeit noch nicht ausgereift sein. Dieser Meilenstein mag zwar der Letzte sein, aber es dürfen noch Änderungen und Erweiterungen vorgenommen werden, um Problemfelder auszubessern.

2.1.2 Eigene Anforderungen

Die eigenen Anforderungen wurden in den Gruppensitzungen besprochen und im Verlauf des Projektes ständig erweitert. Wenn wir uns nicht sofort einigen konnten, wurde per Abstimmung entschieden, welche umgesetzt werden sollen.

2.1.2.1 Login-/Register-Screen

Wenn man bereits registriert ist, kann man sich mit seinem Benutzernamen und Passwort einloggen. Dann wird geprüft, ob die Eingaben zueinander passen. Sollte dies nicht der Fall sein, wird eine Fehlermeldung angezeigt, ansonsten wird man eingeloggt und gelangt ins Hauptmenü. (Sollte man das Passwort vergessen haben, kann man auf den passenden Link klicken.) Ist man noch nicht registriert, kann man auf den dazugehörigen Link klicken, dann ändert sich die View und man kann sich über einen Button registrieren. Dafür muss ein Benutzername, eine E-Mail-Adresse, ein Passwort sowie die Wiederholung des Passworts eingegeben werden.

2.1.2.2 Hauptmenü

Im Hauptmenü soll zu sehen sein, welche Spieler gerade online sind. Diese Liste soll sich von selbst aktualisieren, wenn sich Nutzer ein- oder ausloggen.

Zudem soll es möglich sein, über einen öffentlichen Chat mit allen angemeldeten Spielern zu kommunizieren. Dabei soll der Name des Senders und der Zeitpunkt angezeigt werden. Neue Nachrichten im Chat sollen automatisch angezeigt werden. Zudem wird angezeigt, welcher Nutzer sich ein- bzw. ausloggt.

Es gibt eine funktionierende Liste, in der die aktuellen Lobbies mit ihrem Namen angezeigt werden. Leere Lobbies sollen aus der Liste entfernt werden. Zusätzlich wird noch angezeigt, wie

viele Spieler sich bereits in der Lobby befinden, wie groß die Lobby ist, ob die Lobby offen oder privat ist und ob sich die Lobby bereits im Spiel befindet. Jede Lobby in der Liste besitzt einen eigenen Beitreten-Button, über den man der Lobby beitreten kann. Sollte man sich bereits in der Lobby befinden, die maximale Anzahl an Spieler bereits erreicht sein oder sich die Lobby schon im Spiel befinden, ist der Beitreten Button deaktiviert und der Lobby kann nicht beigetreten werden. Sollte die Lobby privat sein, öffnet sich ein kleines Fenster, in dem man das zugehörige Passwort eintragen soll. Tritt man einer Lobby bei, so öffnet sich diese in einem neuem Tab, welcher auch automatisch im Fokus ist.

Über einen Button kann jeder Spieler neue Lobbies erstellen. Dann öffnet sich ein neues Fenster, in dem man den Namen der Lobby, sowie ein Passwort (optional) eintragen und über den Lobby-erstellen-Button abschicken kann. Ist der Name bereits vergeben, enthält dieser Sonderzeichen oder ist länger als 30 Zeichen, wird eine Fehlermeldung angezeigt. Wird ein Passwort bei der Lobby-Erstellung eingegeben, so ist die Lobby privat, ansonsten ist sie offen.

An der rechten Seite des Fensters soll sich eine Seitenleiste mit Buttons befinden, die in jedem Tab sichtbar und aufrufbar sein sollen. Das gilt für den Spielanleitungs-, Einstellungen- und den Logout-Button. Alle drei können im Hauptmenü, in der Lobby sowie im Spiel betätigt werden. Beim Klick auf den Spielanleitung-Button öffnet sich die Spielanleitung in einem neuen Fenster.

Beim Klick auf die Einstellungen öffnet sich ein neues Fenster, in welchem der Name, die E-Mail-Adresse und das Passwort geändert werden können, sowie der Account gelöscht oder auch der Sound an- bzw. ausgestellt werden kann. Zum Ändern seiner Daten muss einmal das aktuelle Passwort des Nutzers eingegeben werden. Wenn man seinen Account löschen will, muss man dies in einem neuen Fenster, welches beim Betätigen des Buttons erscheint, bestätigen. Bestätigt man dies, wird der Account gelöscht und der Nutzer automatisch ausgeloggt. Ihm wird nun wieder der Login-Screen angezeigt.

2.1.2.3 Lobby

In der Lobby soll es möglich sein, über einen Chat mit den anderen Lobby-Mitgliedern zu kommunizieren. Zudem soll die Lobby über einen Button sowie über das Schließen des Lobby-Tabs verlassen werden können. Mit einem Bereit-Button können die Spieler ihren Status ändern. Sollten alle bereit sein, wird das Spiel automatisch im gleichem Tab gestartet. Der Owner soll zudem noch die maximale Anzahl der Spieler über ein Drop-down-Menü einstellen können, die Anzahl geht von 2 bis zu 4 Spielern. Dies ist aber nur möglich, wenn nicht schon zu viele User in der Lobby vorhanden sind. Zudem soll er in der Lage sein, die Königreichskarten, mit denen er spielen will, auszuwählen. Diese werden dem Server übergeben und dann bei Spielstart initialisiert. Sollten keine oder weniger als 10 Karten ausgewählt worden sein, werden die restlichen zufällig gewählt. Des Weiteren kann er Botss hinzufügen. Als Owner hat er auch das Recht, andere Spieler und auch Botss wieder aus der Lobby zu entfernen. Dies geht über den entsprechenden Button neben den Namen der User in der Userliste. Die Userliste aktualisiert sich automatisch. Über einen grünen bzw. roten Punkt links neben dem Namen wird allen Spielern angezeigt, ob der Spieler bereit ist oder nicht. Die anderen Spieler sollen erkenn können, wer der Owner ist, z. B. über eine Krone neben seinem Namen.

2.1.2.4 Spiel

Im Spiel soll sich ebenfalls ein Chat befinden, in dem die Spieler sich untereinander austauschen können. Zudem soll in dem Chat auch angezeigt werden, wer gerade dran ist und wichtige Aktionen, wie das Spielen oder Kaufen einer Karte, werden ebenfalls dort angezeigt.

Unter dem Chat befindet sich noch einmal die Userliste. Dort soll neben dem eigenen Namen angezeigt werden, wie viele Siegespunkte man gerade hat.

Des Weiteren gibt es einen Button, falls man aufgeben möchte, und einen Button, um die aktuelle Phase zu überspringen. Dieser ist aber nur aktiviert, wenn man auch an der Reihe ist. Jeder Spieler hat außerdem einen Kloknopf; diesen kann er drücken, wenn das Spiel kurz pausiert werden soll. Die anderen Spieler können dann darüber abstimmen, ob sie die Pause zulassen wollen oder nicht. Ist mindestens die Hälfte aller Spieler dafür (Bots werden beim Voting nicht berücksichtigt), wird das Spiel für eine Minute pausiert. Der Anfragesteller kann die Pause aber auch frühzeitig wieder abbrechen.

Zudem hat jeder Spieler einen Nachziehstapel inklusive Anzeige, wie viele Karten sich in dem Stapel befinden, Ablagestapel, Handkarten und ein Infofeld. Auf dem Infofeld wird angezeigt, in welcher Phase man sich befindet bzw. ob man überhaupt dran ist, wie viele Aktionen und Käufe man zur Verfügung hat und wie viel Geld man ausgespielt hat. Außerdem soll jeder Spieler immer alle Karten auf dem Spielfeld angezeigt bekommen, inklusive deren Anzahl, sowie die Ablagestapel und Handkarten (von denen aber nur die Rückseite) der Mitspieler zu sehen. Spielt ein Spieler eine Karte aus, wird das den anderen angezeigt, indem die ausgespielte Karte offen über seine Handkarten gelegt wird. Alle anderen Aktionen der Gegner können die Mitspieler über passende Animationen verfolgen.

Wenn man nicht an der Reihe ist, soll man also in der Lage sein, im Chat zu schreiben, aufzugeben, die Züge der Mitspieler zu sehen und im Chat nachlesen zu können. Außerdem soll man sich die Karten auf dem Spielfeld bzw. die eigenen Karten auf der Hand in groß ansehen können, dies soll über einen Rechtsklick auf die jeweilige Karte geschehen.

Ist man am Zug und hat mindestens eine Aktionskarte auf der Hand, so kann man diese ausspielen und die entsprechende Aktion durchführen. Welche Aktion das genau ist, wird dem Spieler über das Infofeld vermittelt. Sobald man keine Aktionen mehr zur Verfügung hat oder sich keine Aktionskarten mehr auf der Hand befinden, wechselt man automatisch in die Kaufphase. Hier müssen erst alle Geldkarten über einen Button ausgespielt werden, dann kann man eine Karte mit dem entsprechenden Wert vom Spielfeld kaufen. Dieser Button ist in der Aktionsphase deaktiviert, damit man nur in der Kaufphase Karten kaufen kann. Nach dem Kauf wandert die gekaufte Karte, sowie alle zum Kauf benötigten Geldkarten direkt auf den Ablagestapel. Hat man keine weiteren Käufe mehr zur Verfügung, wird automatisch in die Clearphase gewechselt. Das bedeutet alle Karten, die sich auf der Hand oder in der Aktionszone befinden wandern auf den Ablagestapel und man bekommt fünf neue vom Nachziehstapel. Wenn der Nachziehstapel leer ist, wird der Ablagestapel neu gemischt und dem Nachziehstapel hinzugefügt, dies soll dem Spieler auch angezeigt werden.

Um das Spiel für den Spieler übersichtlicher zu gestalten, werden in der Aktionsphase die Geldkarten auf der Hand verdunkelt und in der Kaufphase die Aktionskarten.

Sollte man versuchen eine Karte zu kaufen, wenn man nicht an der Reihe ist, seine Geldkarten noch nicht ausgespielt hat oder nicht genug Geld hat, wird eine entsprechende Fehlermeldung

angezeigt.

2.1.2.5 Grundsätzliches

Die registrierten Nutzer werden auf einer Datenbank auf Keno O. Network Attached Storage (NAS) gespeichert.

Bei der Kommunikation zwischen Server und Client soll funktionieren. Es soll auch möglich sein, Nachrichten vom Server nur an ein bestimmtes Spiel/eine Lobby oder einen spezifischen Spieler zu schicken.

Alle Fenster, die sich in einem neuen Fenster statt in einem neuen Tab öffnen, sollen mittig zum jeweiligen Hauptfenster positioniert sein. In der Lobby und im Spiel könne die Einstellungen zwar geöffnet werden, aber man soll seine Daten nicht ändern und auch seinen Account nicht löschen können. Nur der Ton ist weiterhin einstellbar.

2.2 Umgesetzte Userstories

Im folgenden wurden die umgesetzten Userstories erläutert.

2.2.0.0.1 SWP2019B-7 Ein Gast muss sich registrieren können, damit er sich einloggen kann. Hier soll sich ein fremder Benutzer neu für das Spiel registrieren können. Dafür wird eine grafische Oberfläche beim Benutzer nötig sein sowie eine Verarbeitung des Servers. Dabei muss in der Datenbank geguckt werden, dass nicht zweimal der gleiche Benutzer erstellt wird. Dabei wird eine RegisterUserRequest an den Server geschickt. Wenn die Registrierung berechtigt ist, wird eine RegisterUserMessage zurückgeschickt. Dann kann der Benutzer sich einloggen.

2.2.0.0.2 SWP2019B-8 Die Nutzer sollen nach dem Einloggen in das Hauptmenü gelangen und dort alle Nutzer sehen, die auch eingeloggt sind. Bei dieser Userstory soll der Benutzer die Möglichkeit bekommen, sich einloggen zu können und somit ins Hauptmenü zu gelangen. Hierbei wird eine LoginRequest an den Server geschickt. Dieser überprüft, ob der Benutzer noch nicht eingeloggt ist und ob er schon in der Datenbank existiert. Wenn ja, dann kommt eine UserLoggedInMessage zurück. Wenn der Benutzer weitergeleitet wurde ins Hauptmenü, bekommt er eine UserListMessage. Dabei wird die Liste von Spielern, die online sind, im Hauptmenü aktualisiert.

2.2.0.0.3 SWP2019B-61 - Als Nutzer möchte ich über die Einstellungen meine persönlichen Daten (z. B. Name, Passwort) ändern können. Hierbei handelt es sich um eine Userstory, bei der ein weiteres Fenster geöffnet werden soll, in dem man sämtliche Benutzerdaten ändern kann. Dies beinhaltet sowohl Passwort als auch Nutzername und E-Mail. Die neuen Daten sollen dann an den Server geschickt und von ihm bestätigt werden.

2.2.0.0.4 SWP2019B-65 - Als Nutzer möchte ich im Hauptmenü ein Chat-Fenster angezeigt bekommen. Dabei soll im Hauptmenü ein Chat erstellt werden, der später einfach auch in Lobbys oder im Spiel implementiert werden kann. Dabei wird eine Nachricht von einem eingeloggten Benutzer an den Server gesendet, mit einer ChatID. Diese ChatID ist dafür zuständig, dass man die Nachricht auch dem richtigen Chat zuordnen kann. Wenn die NewChatMessageRequest beim Server ankommt, wird geguckt, welche Spieler ebenfalls in dem Chat mit dieser ChatID sind. Zu diesen wird dann die NewChatMessage geschickt.

2.2.0.0.5 SWP2019B-87 - Als User möchte ich einen Button haben, mit welchem ich Lobbys erstellen kann. Für diese Userstory wird im Hauptmenü ein Button erstellt, mit der man eine Lobby erstellen kann und eine Liste, bei der man alle erstellten Lobbys sehen und beitreten kann. Beim Erstellen der Lobbys muss serverseitig darauf geachtet werden, dass es keine Lobbys mit dem gleichen Namen gibt.

2.2.0.0.6 SWP2019B-113 - Als Nutzer möchte ich erst das Spiel starten können, wenn alle Spieler bereit dazu sind. Um diese Userstory umzusetzen, kriegt jeder Benutzer einen Bereit-Button, der gedrückt wird, sobald der Benutzer das Spiel starten will. Wenn alle Spieler in der Lobby bereit sind, wird das Spiel gestartet und zur Gameview gewechselt. Ebenso sollen alle Benutzer in der Lobby sehen, wie der Bereit-Status der anderen aussieht.

2.2.0.0.7 SWP2019B-209 - Als User möchte ich, dass das Spiel grafisch realisiert wird. Bei dieser Userstory müssen die Messages vom Server auf dem Client verarbeitet und grafisch umgesetzt werden. Ebenso muss es für den Spieler Möglichkeiten geben, Aktionen zu verarbeiten und an den Server weiterzuschicken.

2.2.0.0.8 SWP2019B-269 Als Spieler möchte ich, dass der Bot an einem Spiel teilnehmen kann. Hier soll es die Möglichkeit für den Spieler geben, gegen einen Bot zu spielen. Dabei braucht der Spieler zum einen in der Lobby einen Knopf, bei dem er angeben kann, dass er einen Bot hinzufügen möchte. Dabei muss eine Anfrage an den Server gestellt werden, der einen Bot erstellt. Der Bot ist immer bereit. Sobald ins Spiel gegangen wird, spielt der Bot automatisch, sobald er an der Reihe ist.

2.2.0.0.9 SWP2019B-148 Dokumentation Die Dokumentation wurde im Laufe des Projekts immer nebenher mitgeschrieben, wenn am Projekt/Code gearbeitet wurde. Jedoch fehlt zum Schluss etwas von der Dokumentation. Dafür wurde diese Userstory erstellt.

2.2.0.0.10 SWP2019B-412 Als User möchte ich ein gut getestetes Spiel haben. Während des ganzen Projekts wurden die Tests parallel zum Code geschrieben. Diese Tests sind meist nicht ausreichend oder lückenhaft. Damit diese weiter vervollständigt werden können, wurde diese Userstory erstellt.

Kapitel 3

Umsetzung

In diesem Kapitel werden die verschiedenen Konzepte und Pattern vorgestellt, welche die Gruppe B genutzt hat.

Es folgt die Erläuterung der Umsetzung des Einloggen/Registrierens, des Hauptmenüs, des Spiels, der Lobby, des Chats und der Bots. Hierbei wurde die Umsetzung einmal auf die clientseitige Umsetzung (siehe Abschnitt 3.2) und die serverseitige Umsetzung (siehe Abschnitt 3.4) aufgeteilt. Außerdem wird auf das Common-Modul eingegangen.

3.1 Konzepte und Pattern

3.1.1 Model View Presenter

Eine Vorgabe für die Umsetzung des Spiels war die Verwendung des Model-View-Presenter Pattern. Bei dem Pattern steht die Trennung zwischen der grafischen Oberfläche und die Logik im Vordergrund. Die Oberfläche selbst soll dem Benutzer eine Schnittstelle bieten, um Eingaben zu tätigen und Ausgaben sichtbar zu machen. Sie soll möglichst wenig Logik enthalten.

Die „View“ Implementierung findet mit Hilfe der JavaFX Bibliothek statt. Um das Layout und die View darzustellen, wurde mit FXML und CSS gearbeitet. Durch einen gleichnamigen Controller ist die View ansteuerbar. Die Controller stellen den „Presenter“ des Patterns dar. Durch sie werden die Eingaben der Benutzer verarbeitet und als Message-Objekt an den Server geschickt. Zudem ist der Presenter für die korrekte Darstellung der Antworten des Servers auf die View verantwortlich.

Das “Model“ des Patterns ist überwiegend auf dem Server. Er verarbeitet die Eingaben des Nutzers und antwortet anschließend mit Bestätigungen, Fehlernachrichten oder Updates, die für den jeweiligen Client relevant sind.

3.1.2 Dependency Inversion Pattern

Um für eine Austauschbarkeit zu sorgen und eine bereitstehende Grundarchitektur nutzen zu können, wurde das Depency Inversion Prinzip verwendet. Hier wird eine Implementierung vorgelegt, die eine starkes Abstrahieren von Klassen vorsieht. Serverseitig wurden für den Chat, die Lobby und das Spiel eine Managementklasse und eine Serviceklasse angelegt. Diese abstrahieren die Lobbys, den Chat und das Spiel fast vollständig.

3.1.3 EventBus

Es wird die Implementierung des EventBus Patterns von Google Guava verwendet. Der EventBus wurde für eine einfache und abstrakte Kommunikation zwischen Klassen genutzt. Die Klassen müessen nur an den EventBus angeschlossen werden. Sie haben nicht zwingend eine direkte Verbindung oder Abhänigkeit untereinannder.

3.1.4 DAO-Pattern

Das DAO-Pattern wurde genutzt, um die Datenbank möglichst von dem eigentlich Programm abzukapseln. So wird dafür gesorgt, dass die Datenbank nur mit einer Klasse (DatabasedUserStore), die als Schnittstelle dient, zu tun hat.

3.1.4.1 DAO-Pattern - Beispielbilder

```java
/**
 * Dieser UserStore benutzt eine Datenbank zum Speichern von Usern.
 * Wichtig: Dieser UserStore gibt niemals das Passwort eines Users zurück!
 *
 * @author Keno S
 * @since Sprint 9
 */
public class DatabaseBasedUserStore extends AbstractUserStore implements UserStore {

    private static final Logger LOG = LogManager.getLogger(DatabaseBasedUserStore.class);

    // TODO: Spalte username in der DB ggf. anpassen?

    private static final String SQL_SELECT_ALL_USER = "SELECT username, password, email FROM user";
    private static final String SQL_SELECT_USER = "SELECT username, password, email FROM user WHERE username = ?";
    private static final String SQL_SELECT_USER_PWD = "SELECT username, password, email FROM user WHERE username = ? AND password = PASSWORD(?)";
    private static final String SQL_DELETE_USER = "DELETE FROM user WHERE username = ?";
    private static final String SQL_UPDATE_USER = "UPDATE user SET username = ?, password = PASSWORD(?), email = ? WHERE username = ? AND password = PASSWORD(?)";
    private static final String SQL_INSERT_USER = "INSERT INTO user (username, password, email) VALUES (?, PASSWORD(?), ?)";
```

Abbildung 3.1: DAO - Beispielbild 1

```java
/**
 * Stellt eine User Abfrage an die Datenbank mit dem Namen und Passwort der Anfrage.
 * Wenn ein User gefunden wird, wird dieser ohne Passwort zurückgegeben.
 * Sonst wird ein Optional.empty() zurückgegeben.
 *
 * @author Keno S.
 * @param username Der Username
 * @param password Das Passwort des Users
 * @return Den (nicht) gefundenen User der Datenbank
 * @since Sprint 9
 */
@Override
public Optional<User> findUser(String username, String password) {

    User user = null;
    try {
        Connection conn = establishConnection();
        PreparedStatement stmt = conn.prepareStatement(SQL_SELECT_USER_PWD);
        stmt.setString( parameterIndex: 1, username);
        stmt.setString( parameterIndex: 2, password);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            user = new UserDTO(rs.getString( columnLabel: "username"),  password: "", rs.getString( columnLabel: "email"));
        }

        rs.close();
        conn.close();
        stmt.close();

    } catch (SQLException e) {
        LOG.debug(e.getMessage());
    }
```

Abbildung 3.2: DAO - Beispielbild 2

3.2 Client

Im Client-Modul findet die graphische Umsetzung des Spieles statt. Hier werden unter anderem die einzelnen Views implementiert und definiert, was bei der Interaktion mit dem User passiert. Beispielsweise wird definiert, was passiert, wenn der User einen Button anklickt. Die Implementierung hierfür erfolgt unter anderem über die verschiedenen *.fxml-Dateien und die jeweiligen dazugehörigen Presenter.

Außerdem erfolgt auf Clientseite die graphische Darstellung der Spiellogik, welche im Server-Modul stattfindet. Die nötige Kommunikation dafür erfolgt über den EventBus.

3.2.1 Hauptmenü

Das Hauptmenü wird erstellt, sobald der Benutzer die Nachricht vom Server bekommen hat, dass er sich erfolgreich eingeloggt hat. Sobald dies geschehen ist, öffnet sich das Hauptmenü-Fenster für den Benutzer. Nach dem Einloggen wird der eingeloggte Benutzer gespeichert und es wird die Liste der Spieler, welche online sind und die Lobbys, die erstellt wurden, geschickt. Im Hauptmenü-Fenster hat man die Übersicht, über den globalen Chat, die Liste der erstellten Lobbys und die Liste der eingeloggten Benutzer. Die Listen sind dann in der Klasse MainMenuPresenter gespeichert. Diese Listen ändern sich, sobald neue Messages vom Server reinkommen. Bei der Online-Spieler-Liste wird ein Spieler hinzugefügt, sobald eine UserLoggedInMessage reinkommt. Ein Spieler wird bei UserLoggedOutMessage entfernt.

Abbildung 3.3: Hauptmenü

Bei der Lobby-Liste wird bei der Erstellung einer Lobby, eine CreateLobbyMessage abgefangen und zur Liste hinzugefügt beziehungsweise die Listen „name“, „host“, etc. werden mit den Werten der dazugehörigen Lobby aufgefüllt. Im weiteren Verlauf kommen weitere LobbyMessages an, sobald sich etwas an der Lobby ändert. Diese werden abgefangen und die dazu gehörigen Werte in der Liste werden verändert. Sollte eine UpdatedInGameMessage reinkommen, setzt sich der „inGame“-Wert der Lobby auf true und der rote Punkt der Lobby färbt sich grün. Danach kann man den „Beitreten“-Knopf nicht mehr Bedienen. Der Chat wird in dem Abschnitt gleichnamigen Kapitel behandelt.

3.2.2 Lobby

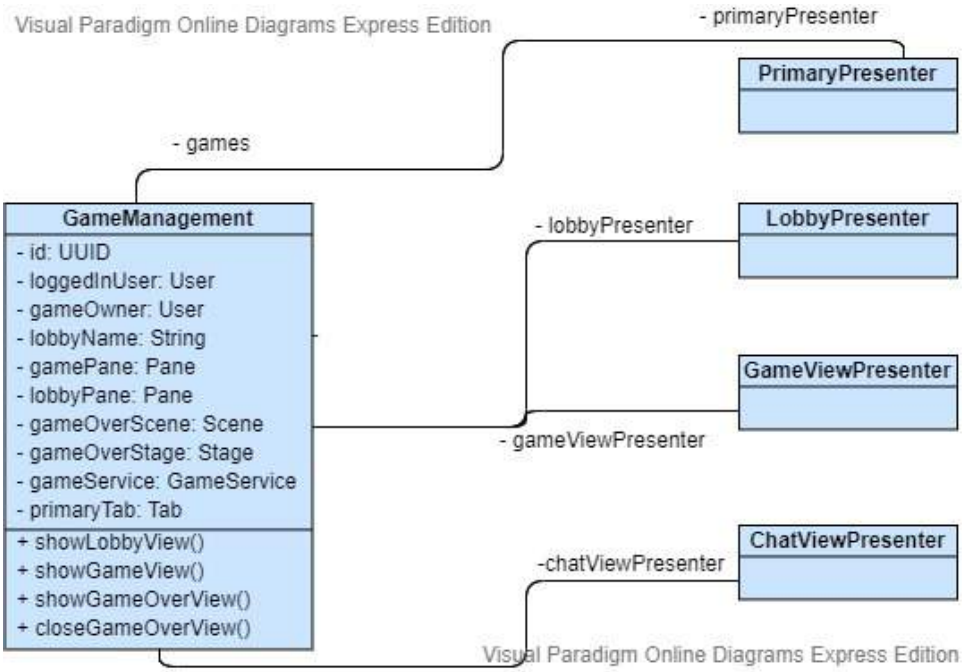

Abbildung 3.4: GameManagement

Clientseitig wird für jede Lobby eine GameManagement-Instanz erstellt. Diese enthält die ID der Lobby bzw. des Spiels, ihren Namen und den loggedInUser sowie den Owner der Lobby. Darüber hinaus besitzt jedes GameManagement einen LobbyPresenter, einen GameViewPresenter, einen ChatViewPresenter und einen PrimaryPresenter. Außerdem stellt die Klasse Methoden bereit, mit der zwischen Lobbyview (showLobbyView()) und Gameview (showGameView()) gewechselt werden kann, sowie eine Methode zum Öffnen (showGameOverView()) bzw. Schließen (closeGameOverView()) der GameOverView. Die GameManagement-Instanzen der Lobbies bzw. Spiele, in denen sich der User befindet, werden im PrimaryPresenter in einer Map<UUID, GameManagement> games gespeichert.

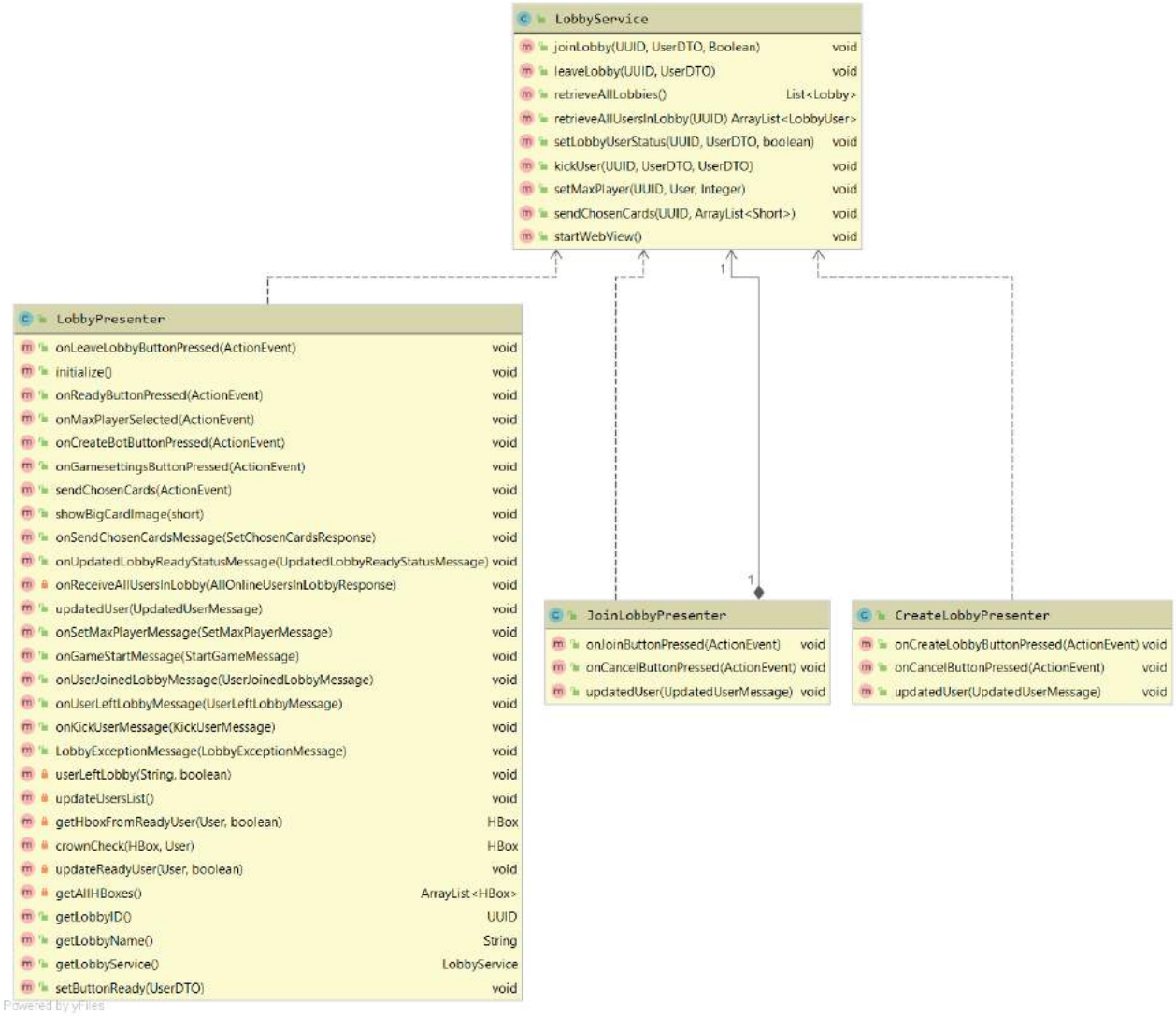

Abbildung 3.5: Lobby Package. Attribute wurden zur besseren Übersichtlichkeit ausgeblendet; die Funktionsweise der (relevanten) Methoden werden in den folgenden Absätzen beschrieben.

Abbildung 3.6: Lobby erstellen

Eine neue Lobby kann im Hauptmenü erstellt werden. Drückt der User dort den Lobby erstellen-Button erscheint ein neuer CreateLobbyPresenter und der User muss dort den Namen der Lobby sowie ihr Passwort (optional) eingeben. Seine Eingaben kann der User dann entweder bestätigen oder das Erstellen der Lobby abbrechen. Drückt er auf Lobby erstellen, wird zunächst geprüft, ob der User einen gültigen Lobbynamen eingegeben hat. Dieser darf maximal 30 Zeichen lang sein und nur Buchstaben, Zahlen und Leerzeichen enthalten, aber mit keinem Leerzeichen anfangen. Ist der eingegebene Name nicht gültig, wird dem User eine Fehlermeldung angezeigt. Ist der Name gültig, wird ein CreateLobbyRequest an den Server gesendet, welcher den User, den Lobbynamen und das Passwort (dieses ist ein leerer String, falls der User kein Passwort gesetzt hat) enthält. War das Erstellen der Lobby serverseitig erfolgreich (der Client enthält eine CreateLobbyMessage) wird eine neue GameManagement-Instanz erstellt und ein neuer Tab mit der LobbyView wird geöffnet. Der Name des Tabs entspricht dabei den Namen der Lobby, welcher durch das Anhängen von - Lobby künstlich verlängert wird, um zu verhindern, dass der User aus Versehen die Lobby durch Schließen des Tabs verlässt.

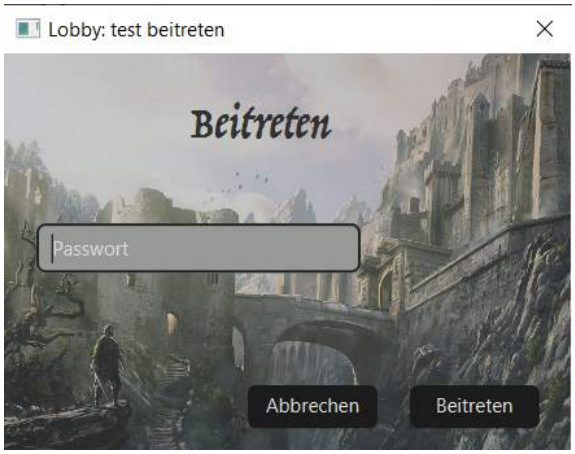

Abbildung 3.7: Einer (passwortgeschützten) Lobby beitreten

Will man keine neuen Lobby erstellen, sondern einer bestehenden beitreten, kann man - falls die Lobby noch betreten werden kann - in der Lobbytabelle im Hauptmenü bei der entsprechenden Lobby auf Beitreten klicken. Ist die Lobby privat, also durch ein Passwort geschützt, öffnet sich der JoinLobbyPresesenter, wo man das Passwort der Lobby eingeben muss. Ist dieses falsch, erscheint eine Fehlermeldung. Ist das Passwort korrekt, wird im LobbyService die Methode

joinLobby(lobbyID: UUID, user: UserDTO, isBot: boolean) aufgerufen wird und ein LobbyJoinUserRequest gesendet. Besitzt die Lobby kein Passwort, wird direkt joinLobby(lobbyID: UUID, user: UserDTO, isBot: boolean) aufgerufen. Erhält man daraufhin eine UserJoinedLobbyMessage, wird, wie bei Erhalt einer CreateLobbyMessage, eine neue GameManagement-Instanz erstellt und neuer Tab mit der LobbyView geöffnet.

Verlassen kann man eine Lobby entweder durch das Schließen des Tabs oder durch Klicken auf den Lobby Verlassen-Button. In beiden Fällen wird dann im LobbyService die Methode leaveLobby(lobbyID: UUID, user: UserDTO) aufgerufen, welche ein LobbyLeaveUserRequest an den Server gesendet und der Lobbytab wird gelöscht, falls der User ihn nicht selber geschlossen hat. Hat ein User eine Lobby verlassen, werden die Userliste in der Lobby sowie die Lobbytabelle im Hauptmenü aktualisiert.

Erstellt man selber eine Lobby, ist man ihr Owner und die LobbyView sieht wie folgt aus:

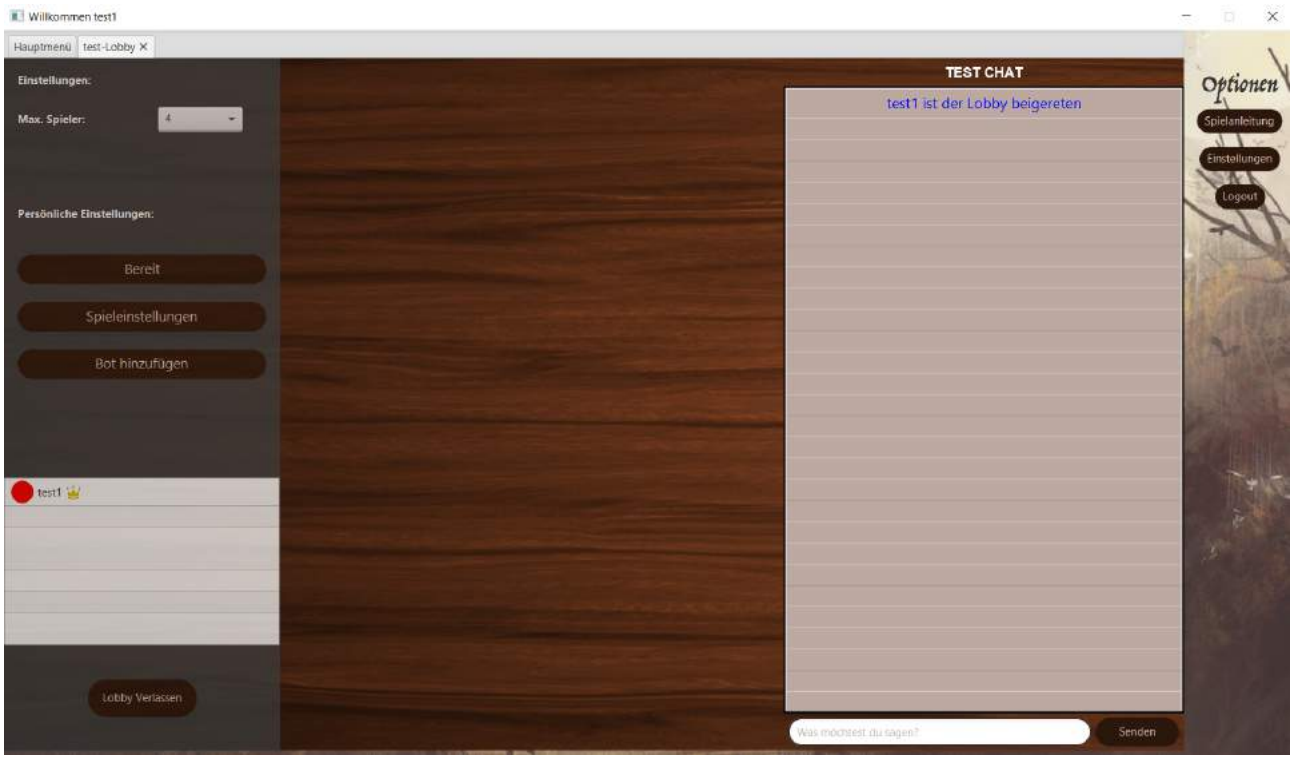

Abbildung 3.8: LobbyView (Owner)

Rechts befindet sich der Chat der Lobby. Links oben kann man in einer ComboBox die maximale Spielerzahl (2-4) verändern. Ändert sich der Wert, wird über die setMaxPlayer(lobbyID: UUID, user: UserDTO, maxPlayer: int) - Methode im LobbyService ein SetMaxPlayerRequest gesendet. War das Ändern erfolgreich, wird der Wert der ChomboBox aktualisiert; Erhält der User eine Fehlernachricht (bspw. wenn drei User in der Lobby sind und er versucht maxPlayer auf 2 zu setzen), wird ihm diese angezeigt und der alte Wert bleibt bestehen.

Drückt man auf Bereit (wenn man gerade nicht bereit ist) bzw. Nicht Bereit (wenn man gerade bereit ist), wird ein UpdatedLobbyReadyStatusRequest über die Methode setLobbyUserStatus(lobbyID: UUID, user: UserDTO, status: boolean) im LobbyService gesendet und der Bereitstatus in der Userliste sowie der Text des Buttons werden bei Erhalt einer UpdatedLobbyReadyStatusMessage aktualisiert.

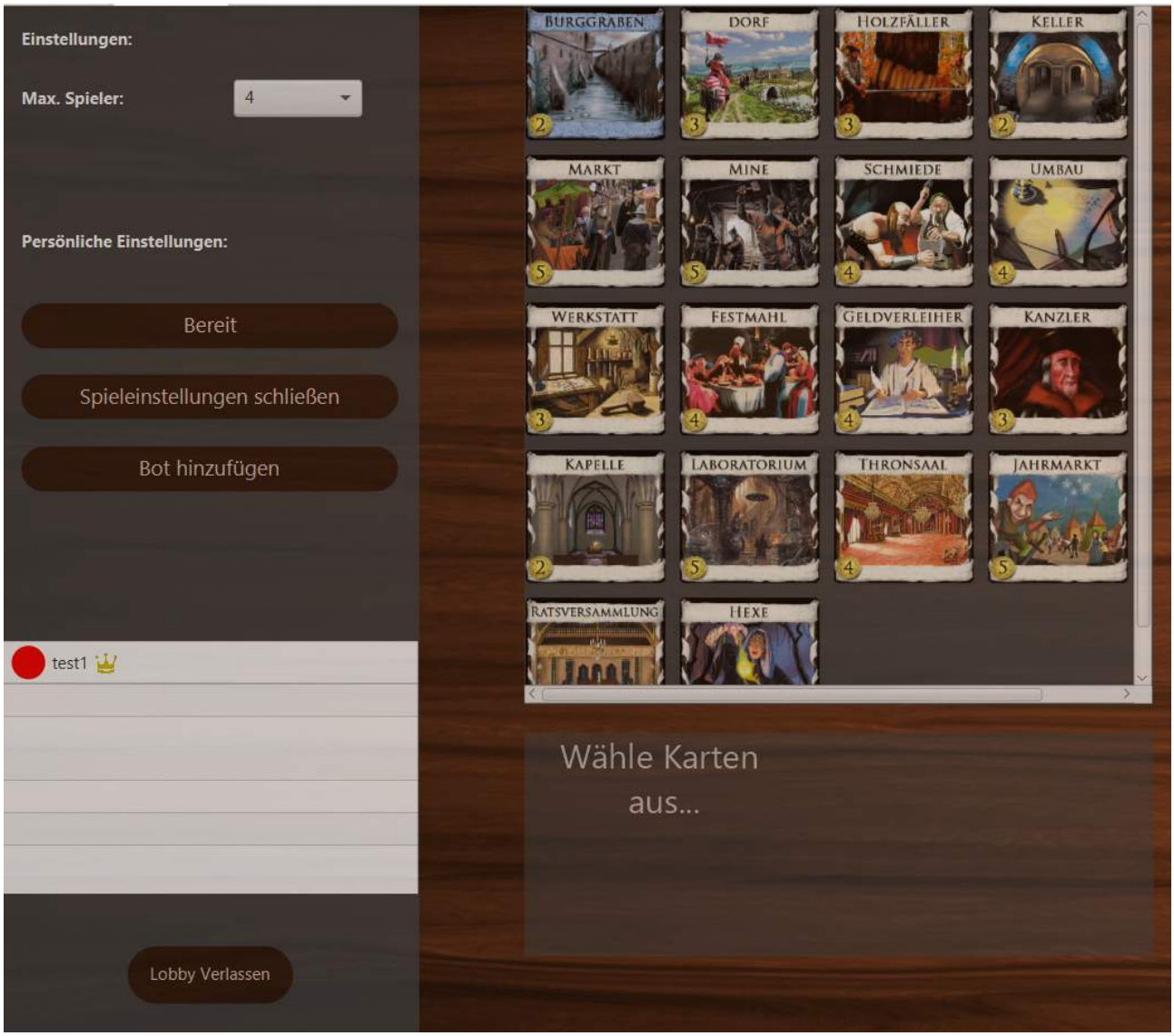

Abbildung 3.9: Kartenauswahl

Über den Spieleinstellungen - Button kann man in einer VBox, die nach Drücken des Buttons sichtbar wird, auswählen, mit welchen Aktionskarten gespielt werden soll. Die ausgewählten Karten werden dann durch Aufruf der sendChoosenCards(lobbyID: UUID, chosenCards: ArrayList<Short>) - Methode im LobbyService in einem SendChosenCardsRequest gesendet und die Auswahlbox schließt sich wieder.

Durch Drücken des Bot-hinzufügen-Buttons wird ein AddBotRequest gesendet und ein Bot wird der Lobby hinzugefügt.

In der Spielerliste sind alle User, die sich in der Lobby befinden, inklusive ihres Bereitstatus gelistet. Zusätzlich wird durch eine Krone dargestellt, welcher User der Owner der Lobby ist. Ist man Owner der Lobby, befindet sich zudem neben jedem Usernamen (außer dem eigenen) ein Button, mit dem man den User aus der Lobby entfernen kann. Das Drücken dieses Buttons ruft im LobbyService die Methode kickUser(lobbyID: UUID, gameOwner: UserDTO, userToKick: UserDTO) auf, welche ein KickUserRequest an den Server sendet.

Ist man nicht Owner der Lobby, sind die Combobox, der Spieleinstellungen und der Bot hinzufügen-Button nicht sichtbar.

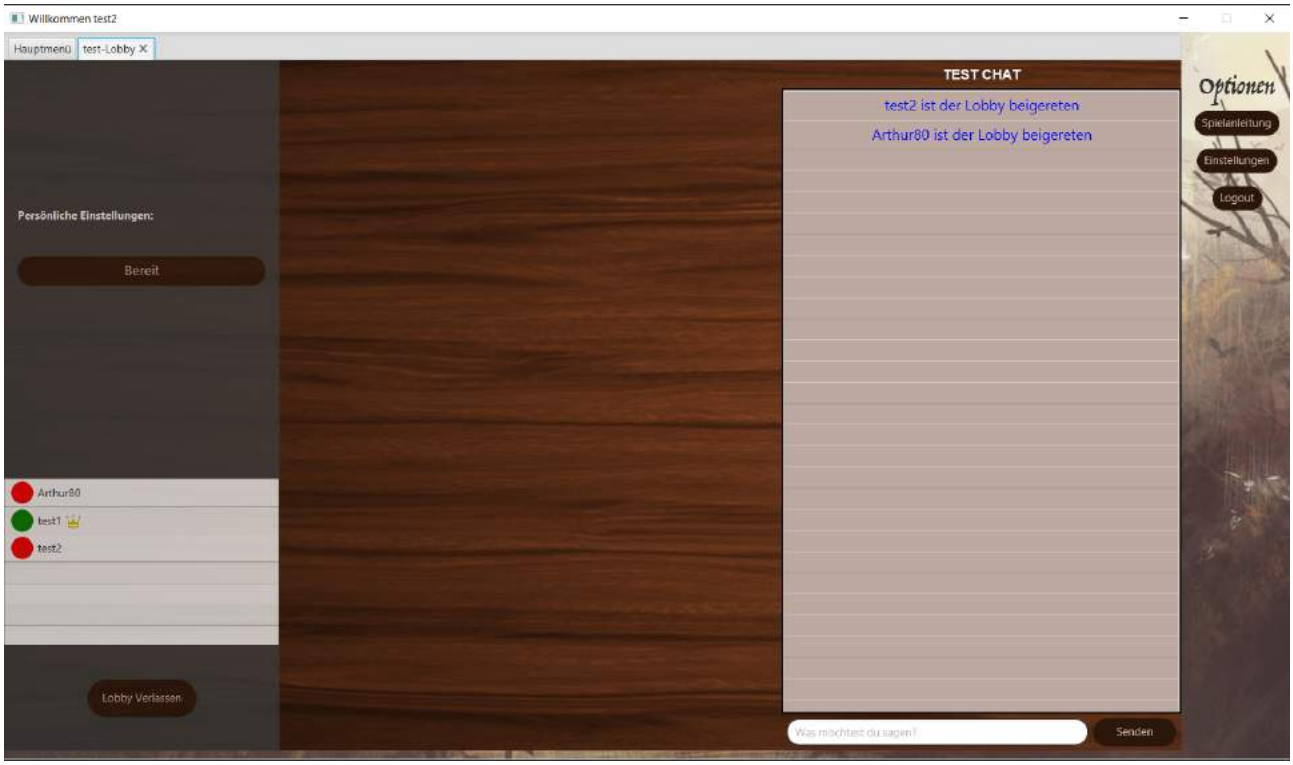

Abbildung 3.10: LobbyView (Nicht-Owner)

3.2.3 Spiel

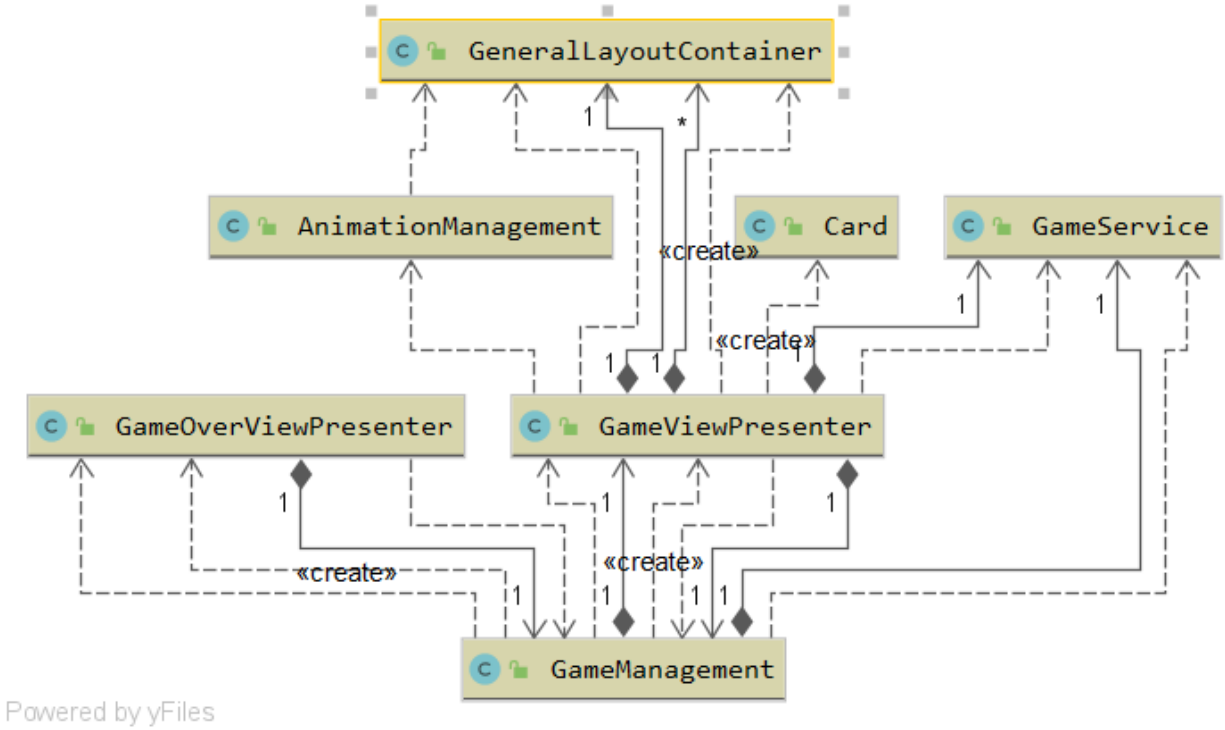

Abbildung 3.11: Package Game

Im Diagramm sind die für die clientseitige Implementierung genutzten Klassen dargestellt.

- GeneralLayoutContainer: Genereller Layout-Container für Karten als zusammenfassender Container für den DeckLayoutContainer, DiscardPileLayoutContainer, HandcardsLayout-Container und den PlayedCardLayoutContainer
- AnimationManagement: Stellt Methoden zur Ausführung von Kartenanimationen bereit
- Card: Graphische Darstellung (ImageView) einer Karte
- GameService: Klasse, von der aus Game-Requests an den Server gesendet werden
- GameViewPresenter: Graphische Darstellung des Spiels; Verarbeitung von Messages
- GameOverViewPresenter: Graphische Darstellung der GameOverView
- GameManagement: Siehe 3.4

3.2.3.1 Spielstart

Empfängt der Client eine StartGameMessage, wird im GameManagement die Methode showGameView() aufgerufen und in den entsprechenden Tab wird die GameView geladen. Diese sieht wie folgt aus:

Abbildung 3.12: GameView

Im linken Bereich befindet sich oben der Kloknopf, mit dem man eine einminütige Spielpausierung anfordern kann. Darunter befindet sich der Spielchat als auch die Spielerliste. Rechts neben der Spielerliste ist die Anzahl der Siegpunkte zu sehen. Ganz unten befindet sich ein Button, mit dem man Aufgeben und ein Button mit dem man die aktuelle Phase überspringen kann. Ist man gerade nicht am Zug, ist Letzterer deaktiviert.

Im rechten Bereich sieht man unten mittig seine Handkarten. Links daneben befindet sich der eigene Nachziehstapel inklusive Anzahl Karten auf diesem; rechts daneben der eigene Ablagestapel (im Bild nicht sichtbar, da sich noch keine Karte auf diesem befindet). Oben (beziehungsweise je nach Spielerzahl auch links und rechts) sieht man seine Gegner und deren Hände (verdeckt). Des Weiteren wird der jeweils dazugehörige Ablagestapel angezeigt.

In der Mitte befinden sich die Aktionskarten und oben rechts die Punkte- und Anwesenkarten sowie die Fluchkarte. Für jede dieser Karte wird angezeigt, wie viel Stück noch auf dem Spielfeld vorhanden sind. Außerdem kann man sich jede Karte groß anzeigen lassen. Hierzu wird die jeweilige Karte mit einem Rechtsklick angeklickt.

Über seiner eigenen Hand wird einem angezeigt, wie viele Aktionen, Käufe und (sofern die Geldkarten ausgespielt wurden) wie viel Geld zur Verfügung steht. Zudem wird dargestellt, in welcher Spielphase man sich gerade befindet.

3.2.3.2 Aktionsphase

Abbildung 3.13: Aktionsphase

Befindet man sich in der Aktionsphase werden Geldkarten auf der Hand verdunkelt und einem wird angezeigt, dass man Aktionen spielen darf. Spielt man eine Karte per Klick aus, wird ein PlayCardRequest an den Server gesendet. Enthält man daraufhin eine GameExceptionMessage,

wird einem die Fehlermeldung angezeigt. War das Ausspielen der Aktionskarte erfolgreich, so wird diese in die Aktionszone des Spielers bewegt und das Infofeld aktualisiert.

3.2.3.3 ChooseCardRequest

Bei einigen Aktionskarten muss der Spieler Karten von seiner Hand oder aus dem Spielfeld auswählen. Ist dies der Fall, erhält der jeweilige Spieler ein ChooseCardRequest und ihm wird im Infofeld angezeigt was er machen muss. Der ChooseCardRequest enthält eine Liste mit allen Karten, aus denen der Spieler auswählen darf, sowie die Kartenquelle und die Anzahl der Karten, die er entsprechend der mitgegebenen Informationen wählen darf. Kann er eine Karte vom Spielfeld wählen, werden nach den Informationen des ChooseCardRequest alle Karten verdunkelt, die er nicht wählen darf. Wählt der Spieler dann die geforderte Anzahl Karten aus oder bestätigt seine Auswahl durch Drücken des „Auswahl beenden “- Buttons (sofern er beliebig viele Karten wählen darf), werden die gewählten Karten in einer ChooseCardResponse an den Server geschickt und dort verarbeitet. Hat der Spieler keine Karten ausgewählt, wird die entsprechende Kartenaktion serverseitig abgebrochen. Die folgenden zwei Abbildungen zeigen die jeweiligen Sichten:

Abbildung 3.14: Karten von der Hand wählen

Abbildung 3.15: Karte aus dem Spielfeld wählen

3.2.3.4 OptionalActionRequest

Einige Aktionen sind optional. Kann der Spieler entscheiden, ob er eine Aktion ausführen möchte oder nicht, wird ihm ein OptionalActionRequest gesendet. Ihm wird dann die optionale Aktion im Infofeld angezeigt und er kann über eine Entscheidung festlegen, ob er diese Aktion ausführen möchte oder nicht. Seine Antwort wird dann in einer OptionalActionResponse an den Server gesendet und die Aktion wird, je nach Entscheidung des Spielers, ausgeführt oder übersprungen. Ein Beispiel ist im folgenden Bild zu sehen:

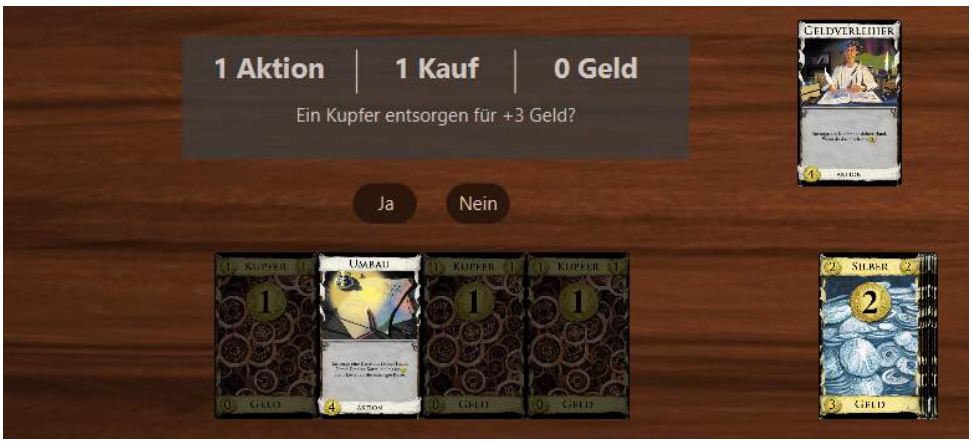

Abbildung 3.16: Optionale Aktion

3.2.3.5 Kaufphase

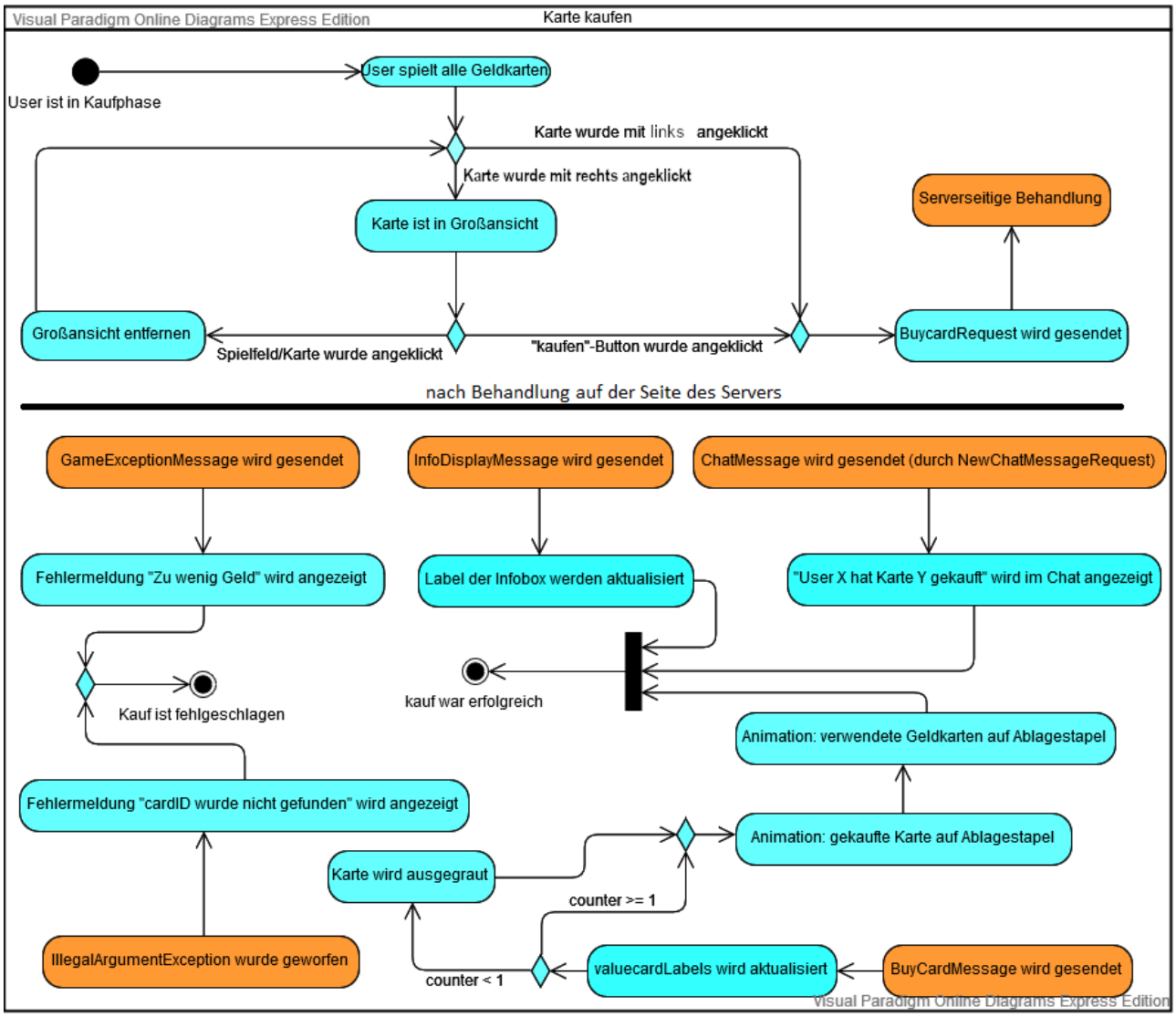

Abbildung 3.17: Kauf einer Karte

In der Kaufphase muss man zunächst durch Drücken des „Alle Geldkarten ausspielen “Buttons alle seine Geldkarten ausspielen. Diese werden dann in die Aktionszone des Spielers bewegt und ihr Wert wird dem Spieler in seinem Infofeld angezeigt. Hat man seine Geldkarten bereits ausgespielt, ist nicht an der Reihe oder befindet sich nicht in der Kaufphase, so ist der Button nicht anklickbar.

Abbildung 3.18: Geldkarten ausspielen

Eine Karte kann nun direkt gekauft werden, indem man sie mit der linken Maustaste anklickt. Daraufhin wird ein BuyCardRequest an den Server gesendet. Ebenfalls ist eine Karte kaufbar, indem man sie sich zunächst mit Rechtsklick groß anzeigen lässt und dann den Kaufen-Button drückt, woraufhin die Großansicht wieder entfernt und ein BuyCardRequest gesendet wird.

Schlägt der Kauf fehl, wird eine GameExceptionMessage gesendet und einem die entsprechende Fehlermeldung angezeigt. War der Kauf dagegen erfolgreich und es wird eine BuyCardMessage empfangen, wird im Chat angezeigt, dass User X Karte Y gekauft hat und der Infotext wird nach den in der InfoPlayDisplayMessage enthaltenen Informationen aktualisiert. Die gekaufte Karte wird dann in einer Animation auf den Ablagestapel des Spielers bewegt. Außerdem wird die Anzahl der Karte auf dem Spielfeld aktualisiert und die Karte wird ausgegraut, falls kein Exemplar mehr auf dem Spielfeld vorhanden ist.

3.2.3.6 Clearphase

Enthält der Client eine StartClearPhaseMessage, werden die Handkarten des Spielers sowie die Karten, die sich in seiner Aktionszone befinden, auf seinen Ablagestapel animiert und ihm werden seine neuen Handkarten angezeigt.

3.2.3.7 Pausen

Ein Spieler kann jederzeit durch Drücken des Kloknopfs anfragen, das Spiel für 60 Sekunden zu pausieren. Drückt er diesen, wird ein PoopBreakRequest an den Server gesendet, welcher wiederrum eine PoopBreakMessage zurücksendet. Daraufhin wird den anderen Spielern angezeigt, dass er eine Klopause machen möchte und sie können abstimmen, ob sie dafür (Okay) oder dagegen (Nope) sind. Ist mindestens die Hälfte aller Spieler dagegen (Bots werden beim Voting nicht berücksichtigt), wird eine CancelPoopBreakMessage gesendet und es findet keine Pause statt.

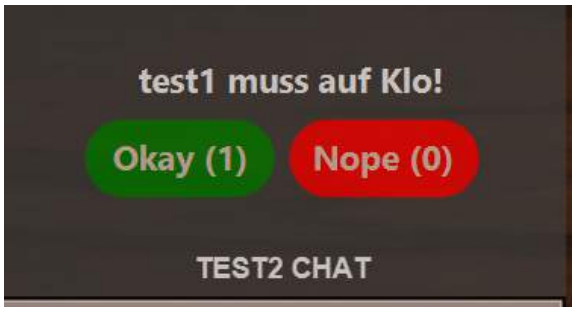

Abbildung 3.19: Klopause - Abstimmung

Stimmt die Hälfte der Spieler oder mehr für eine Pause, wird eine StartPoopBreakMessage gesendet. Der Kloknopf ist dann nicht mehr sichtbar und im Spielfeld wird ein Countdown sichtbar. Das Spiel geht dann weiter, wenn der Countdown abgelaufen ist oder der Spieler, der die Pause angefordert hat, auf Abbrechen drückt.

Abbildung 3.20: Klopause

3.2.4 Chat

Dem Spieler ist es jederzeit möglich, mit seinen Mitspielern über einen Chat zu kommunizieren. Dazu steht ihm zum einen ein globaler Chat im Hauptmenü zur Verfügung (siehe Abb. 3.22), ein Lobbychat (3.23) sowie ein Spielchat (3.24). Jeder Chat wird über einen ChatViewPresenter initialisert. Über eine ChatID als String wird zwischen den einzelnen Chats unterschieden: Handelt es sich um den öffentlichen Chat lautet die ID "global", bei einem Lobby- oder Spielchat wird eine UUID im Konstruktor übergeben und dann im ChatID-Attribut als String gespeichert. Dadurch wird sichergestellt, dass die Nachrichten immer nur in dem Chat, von welchen aus sie geschickt wurden, angezeigt werden.

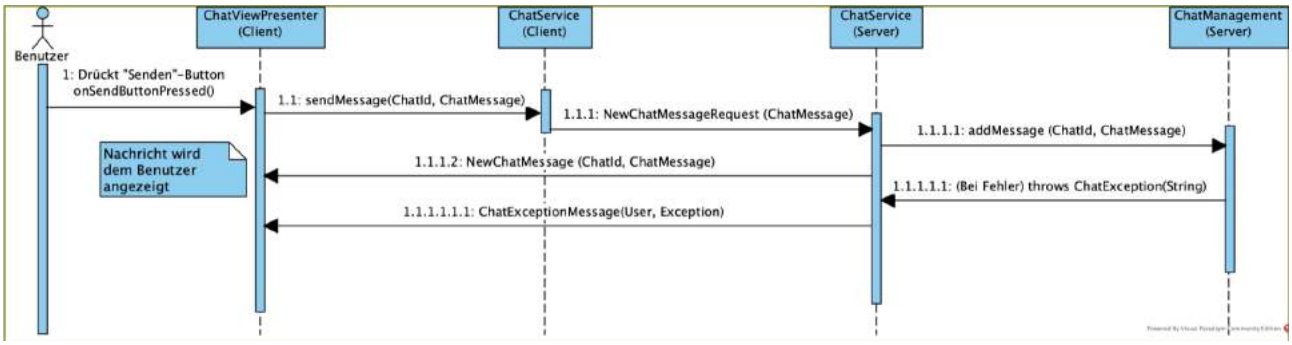

Abbildung 3.21: Sequenzdiagramm zum Senden einer Chatnachricht

Wie in der Abbildung zu sehen ist, wird, wenn ein Spieler eine Nachricht schicken will, über den ChatService zuerst ein Request an den Server geschickt. Dieser verarbeitet die Request dann und schickt eine Message zurück, welche ebenfalls den Sender und die Nachricht enthält. Diese Message wird im ChatViewPresenter abgefangen und so verarbeitet, dass den Spieler die Chatnachricht auf dem Bildschirm angezeigt wird. Bei jeder neuen Nachricht wird der Name des Senders über der Textblase angezeigt, außer ein Spieler schickt mehrere Nachrichten hintereinander, dann steht der Name nur über der ersten Nachricht. Jede Nachricht ist auf 160 Zeichen begrenzt. Die aktuelle Anzahl der noch verfügbaren Zeichen wird dem Spieler mittels eines Tooltip über seinem Eingabefeld angezeigt.

Im Hauptmenü kann er mit allen aktuell eingeloggten Spielern chatten. Zusätzlich wird ihm angezeigt, wenn ein neuer Spieler sich einloggt und wenn ein alter sich ausloggt.

Erstellt der Spieler nun eine Lobby oder tritt einer bei, ist in dem neuen Tab, der sich dann öffnet, ebenfalls ein Chat vorhanden. Hier können allerdings nur die Spieler, die sich auch in der Lobby befinden, miteinander schreiben. Außerdem wird ihm angezeigt, wenn ein neuer Spieler der Lobby beitritt bzw. wieder austritt oder entfernt wird (auch bei Bots). Zudem wird angezeigt, wenn der Owner der Lobby Aktionskarten für das Spiel ausgewählt hat.

Wenn dann das Spiel gestartet ist, wird der Spielchat initialisert. Hier können die Spieler wie gewohnt miteinander schreiben und es wird angezeigt, wenn ein Spieler aufgibt. Ansonsten werden hier noch die wichtigsten Spielzüge angezeigt, also wer gerade dran ist und welche Karten gekauft bzw. ausgespielt wurden.

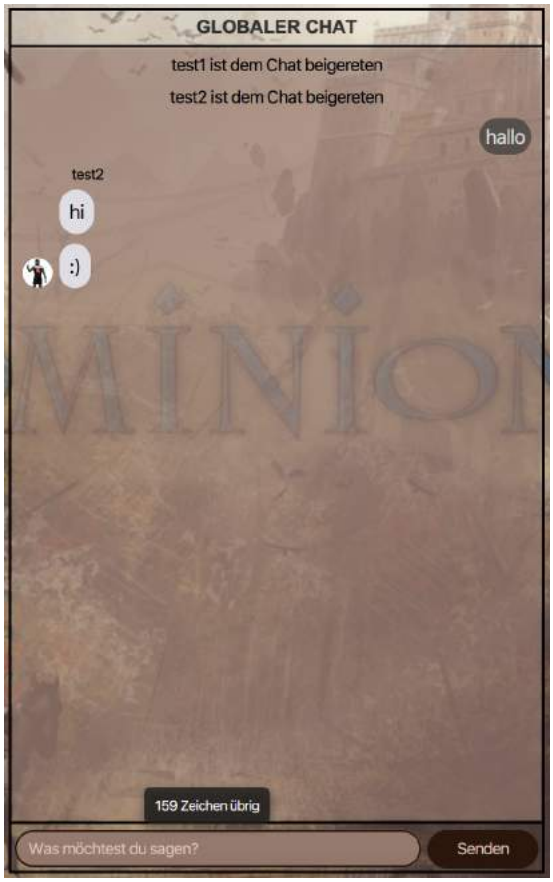

Abbildung 3.22: Der Chat im Hauptmenü

Abbildung 3.23: Der Chat in der Lobby

Abbildung 3.24: Der Chat im Spiel

3.3 Common

Das Common-Modul ist ein Modul, das durch seine Eigenschaften beiden Hauptmodulen unterstützend zur Verfügung steht. Das heißt, sowohl das Client-Modul als auch das Server-Modul

können auf die im Common-Modul verfügbaren Java Klassen, Interfaces, Requests, Messages und Responses zugreifen. Die von beiden anderen Modulen genutzten Java Klassen umfassen dabei unter anderem ein ChatService-Interface, ein LobbyService-Interface, die Java Klassen LobbyUser und UserDTO, den JSON-Card-Parser und beispielsweise eine RegisterUserRequest, LoginSuccessfulResponse oder auch eine UpdateUserMessage. Des Weiteren stehen im Common-Modul Tests zur Verfügung. Hierunter fällt beispielsweise ein Test für den JSON-Card-Parser.

3.3.1 Common-Modul - Beispielbilder

Im Folgenden sind Bilder der zur Verfügung stehenden Implementierungen zu sehen:

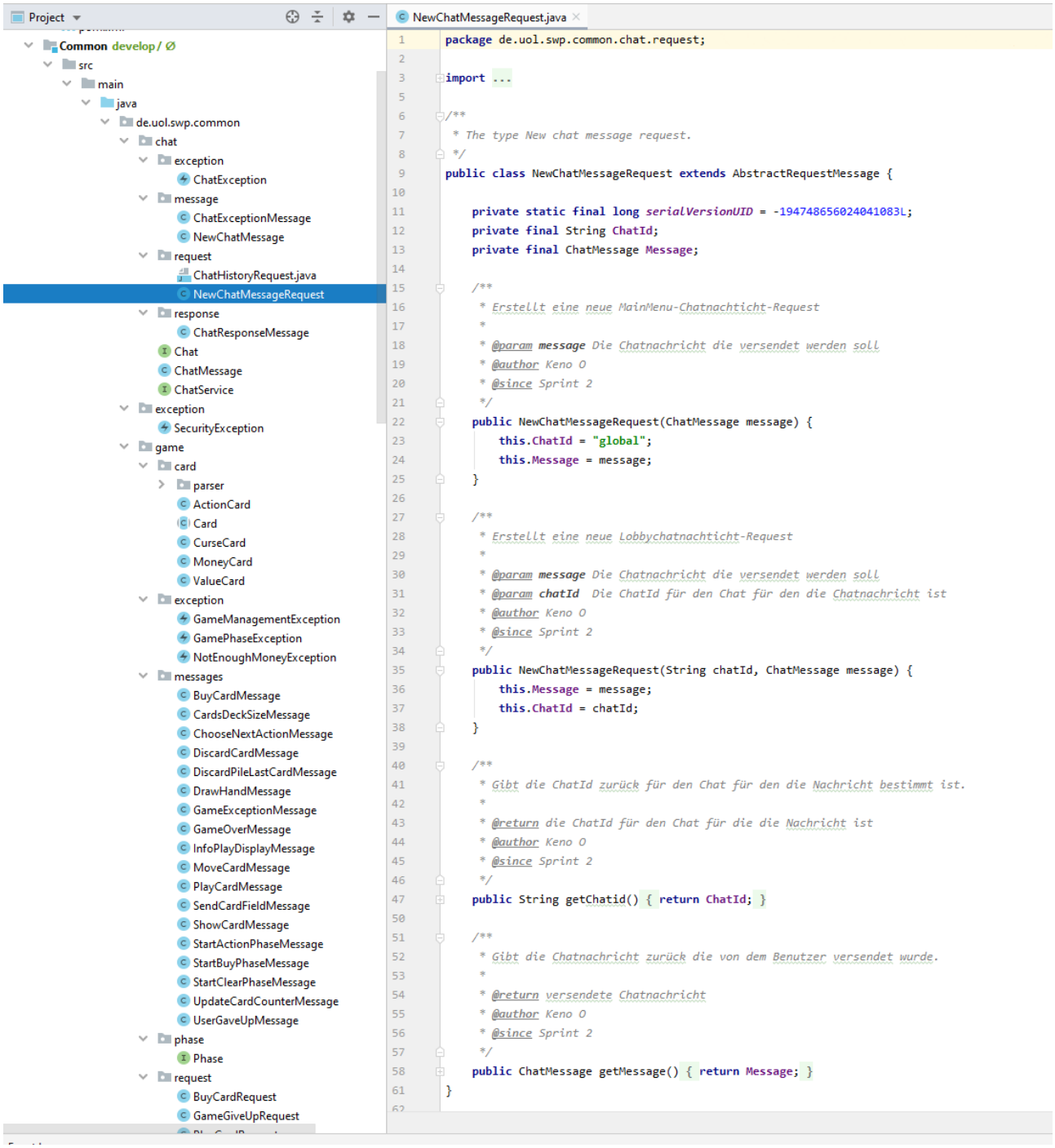

Abbildung 3.25: Common-Modul - Beispielbild 1

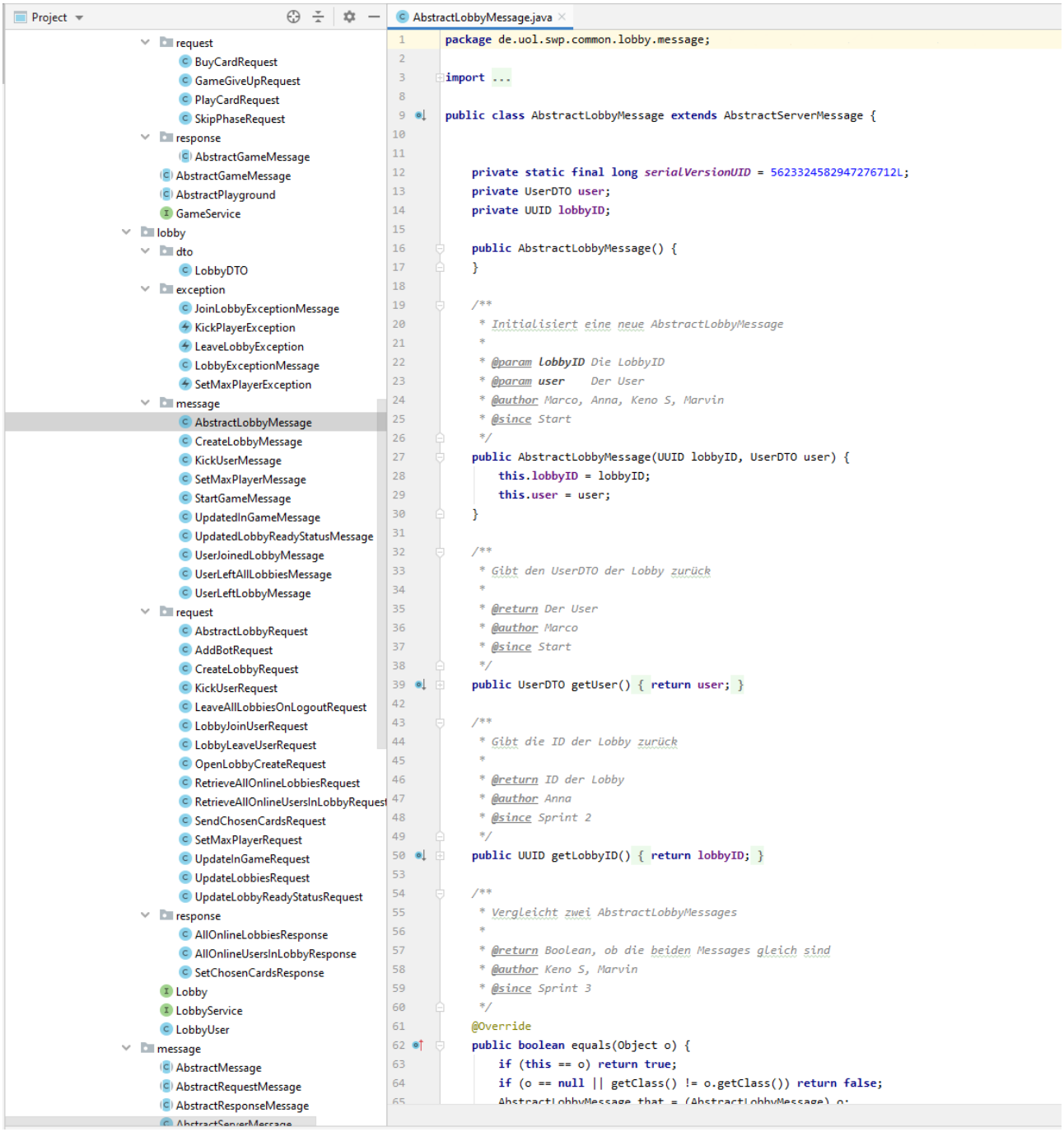

Abbildung 3.26: Common-Modul - Beispielbild 2

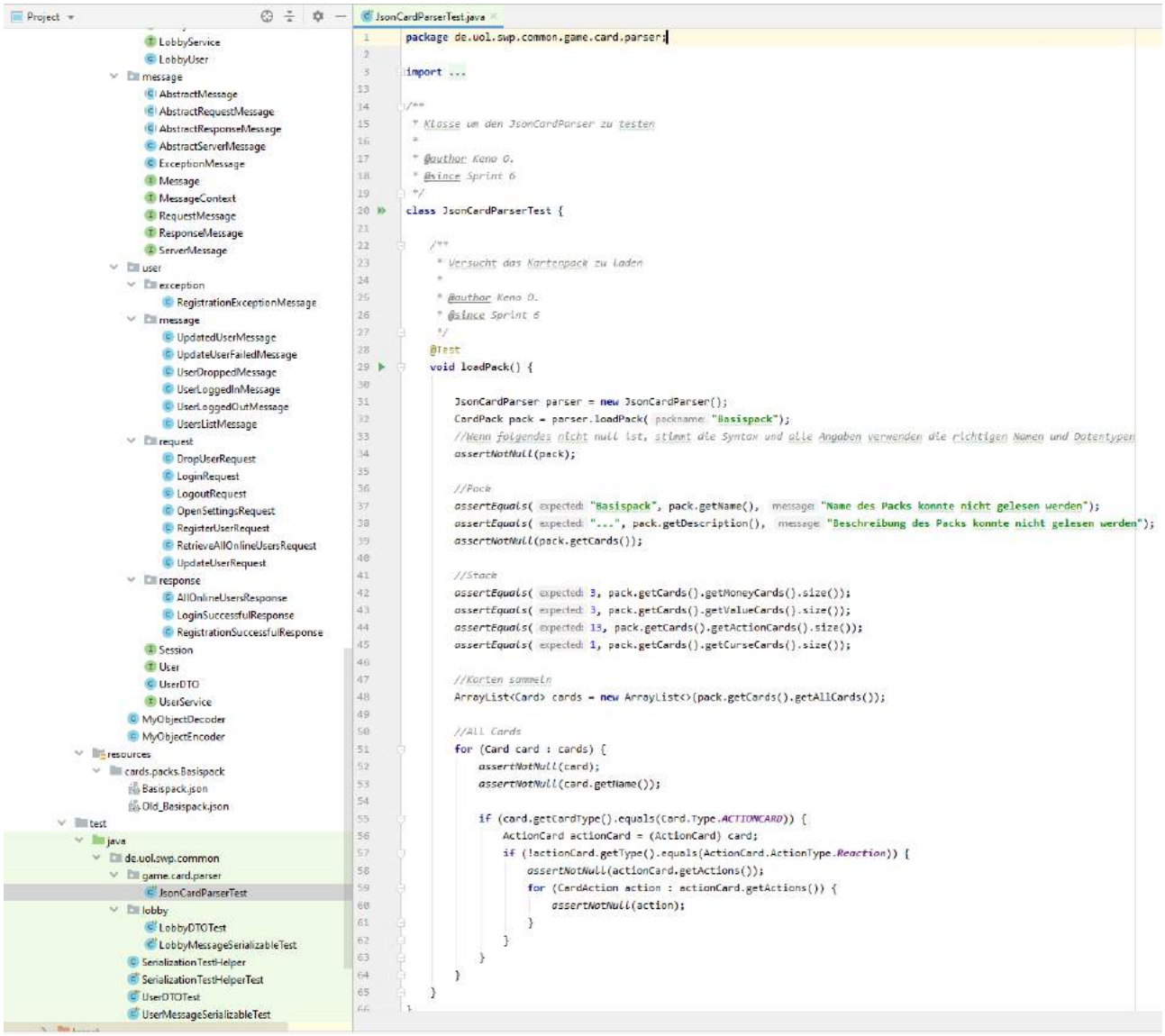

Abbildung 3.27: Common-Modul - Beispielbild 3

3.4 Server

Im Server-Modul findet die logische Umsetzung des Spiels statt. Da ein Teil der Anforderung war, dass der Client “so dumm wie möglich“ gehalten werden soll, wird der Großteil der Logik auf dem Server verarbeitet. Mit Hilfe von Google’s EventBus erhält der Server Requests der Clients, die er verarbeitet und anschließend mit einer Message an den/die Client/s zurücksendet beziehungsweise beantwortet. Ein solcher Request könnte zum Beispiel in Form einer Nutzeranmeldung, dem Erstellen einer Lobby oder mit dem Vorhaben des Spielens einer Karte im Spiel, vorliegen. Weiterhin beinhaltet der Server keine visuellen Ressourcen, welche im Client gespeichert und aufgerufen werden.

3.4.1 Hauptmenü

3.4.1.1 Einstellungen

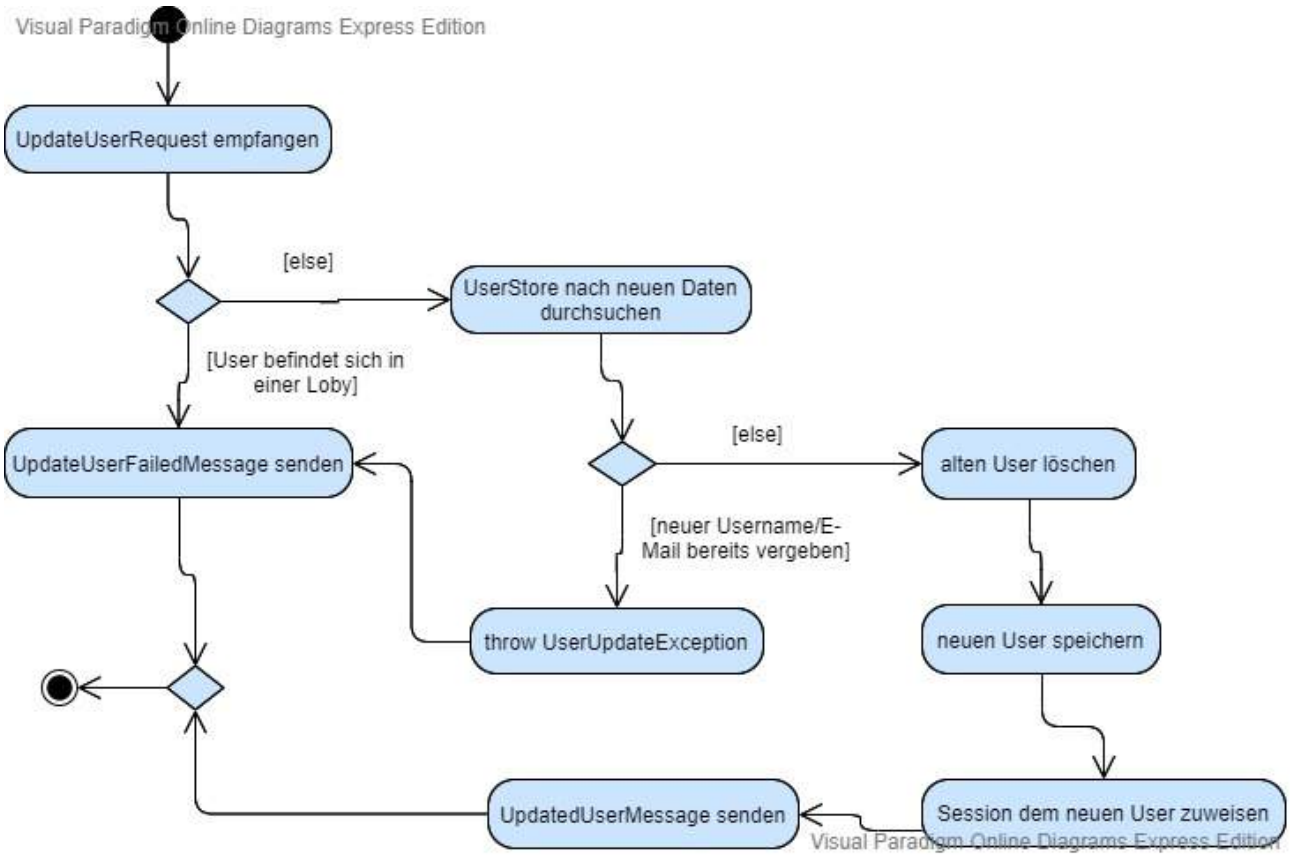

Abbildung 3.28: Userdaten ändern - serverseitiges Aktivitätsdiagramm

Wird im AuthenticationService ein UpdateUserRequest empfangen, wird als Erstes mit Hilfe der userinLobby(User user) - Methode im LobbyManagement geprüft, ob sich der User, der das Request gesendet hat, in einer Lobby befindet. Ist dies der Fall, kann er seine Daten nicht ändern und ihm wird eine UpdateUserFailedMessage gesendet. Ist dies nicht der Fall, wird der UserStore nach dem gewünschten neuen Usernamen und/oder der neuen E-Mail durchsucht. Existiert bereits ein User mit dem neuen Namen bzw. der neuen E-Mail, wird eine UpdateUserException geworfen, da diese nicht mehrfach vergeben sein dürfen. Dem User wird dann eine UpdateUSerFailedMessage gesendet. Sind die gewünschten neuen Daten noch nicht vergeben, wird der alte User aus dem Store gelöscht. Anschließend wird ein neuer User mit den neuen Daten erstellt und im Store gespeichert. Die Session des alten Users wird danach dem neuen User zugewiesen und es wird eine UpdatedUserMessage gesendet.

3.4.1.2 Lobbytabelle

Wird im LobbyService ein RetrieveAllOnlineLobbiesRequest empfangen, wird den Clients eine AllOnlineLobbiesResponse gesendet, die eine Collection mit allen im LobbyManagement gespeicherten Lobbies enthält.

3.4.1.3 Logout

Bei einem LogoutRequest wird im AuthenticationService zunächst geprüft, ob es sich um einen HardLogout handelt, also ob der User sich durch das Schließen des Fensters ausgeloggt hat. Trifft dies zu, wird für den User in jedem Game, in dem er sich befindet, ein GameGiveUpRequest gesendet. Falls es sich nicht um einen HardLogout handelt und sich der User in einem Spiel befindet, wird ihm eine Fehlernachricht gesendet.
Andernfalls wird der User über die logout(user: User) - Methode im UserManagement aus den loggedInUsers entfernt und seine Session wird entfernt. Anschließend wird ein LeaveAllLobbiesOnLogoutRequest gesendet, woraufhin der User aus allen Lobbies, in denen er sich befindet, entfernt wird. Zuletzt wird dann eine UserLoggedOutMessage gesendet.

3.4.1.4 Account löschen

Wird im AuthenticationService ein DropUserRequest empfangen, wird zunächst geprüft, ob er sich in einem Spiel befindet. Ist dies der Fall, darf er seinen Account nicht löschen und ihm wird eine Fehlernachricht gesendet. Ist dies nicht der Fall, wird eine UserDroppedMessage gesendet und der User über die dropUser(user: User) - Methode im UserManagement aus dem Store gelöscht und ausgeloggt. Seine Session wird anschließend ebenfalls gelöscht.

3.4.2 Lobby

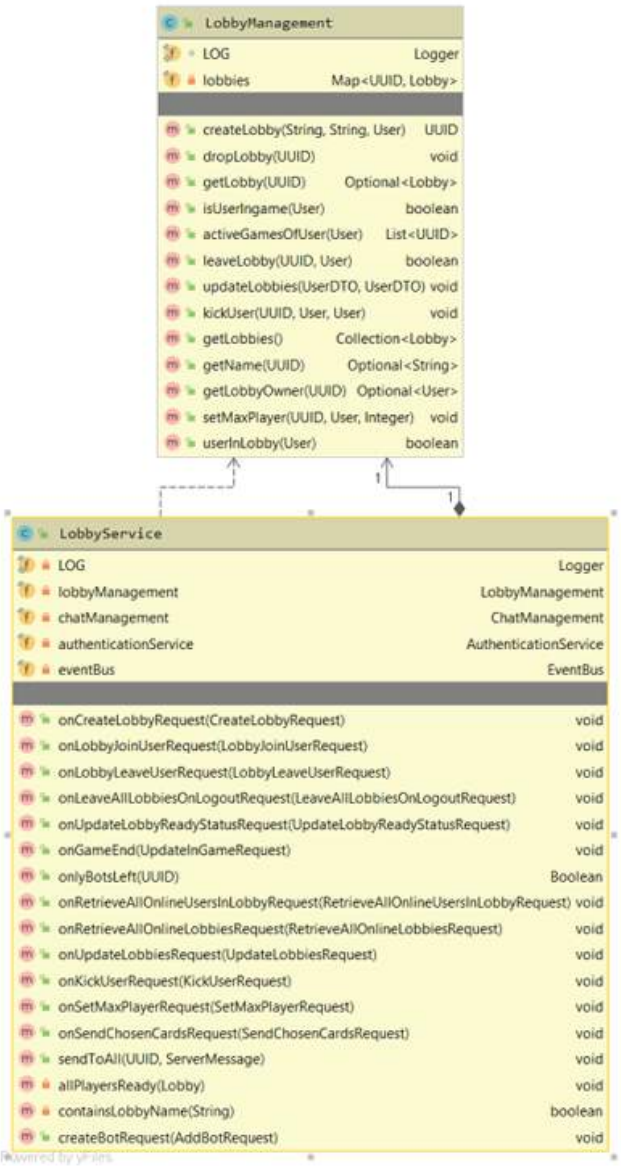

Abbildung 3.29: LobbyService und LobbyManagement. Die Methoden werden im Folgenden erläutert.

Serverseitig sind die Klassen LobbyManagement und LobbyService für die Verwaltung der Lobbies zuständig. Alle existierenden Lobbies werden im LobbyManagement in einer Map<UUID, Lobby> lobbies gespeichert.

3.4.2.1 Erstellen einer neuen Lobby

Will der User eine neue Lobby erstellen, wird hierzu ein CreateLobbyRequest gesendet, welches den User sowie den vom ihm gewählten Lobbynamen und das von ihm festgelegte Passwort enthält. Wird im LobbyService das CreateLobbyRequest empfangen, wird zunächst mit Hilfe der Methode containsLobbyName(lobbyname: String) geprüft, ob schon eine Lobby mit dem im Request gespeicherten Namen existiert. Ist dies der Fall, wird eine CreateLobbyMessage gesendet,

bei der alle Parameter bis auf User null sind. Ist dies nicht der Fall, kann die Lobby erstellt werden. Hierfür wird zuerst im LobbyManagement die Methode createLobby(name: String, lobbyPassword: String, owner: LobbyUser) aufgerufen, welche eine neue ID (randomUUID) generiert sowie eine neue Lobby mit den übergebenen Parametern erstellt und beides dann in die lobbies-Map einfügt. Anschließend wird mit der zuvor generierten LobbyID ein neuer Chat erstellt und eine CreateLobbyMessage wird an alle eingeloggten User gesendet.

3.4.2.2 Beitreten einer Lobby

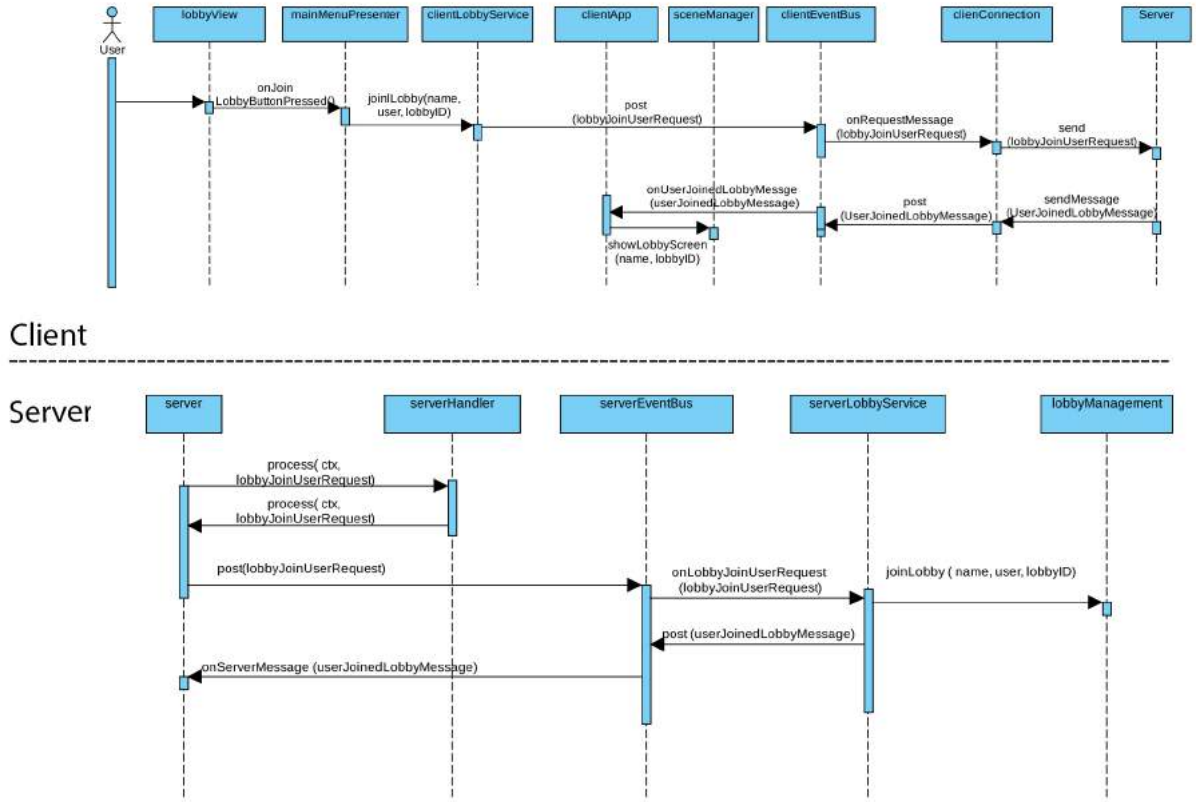

Abbildung 3.30: Lobby beitreten: Sequenzdiagramm. Die Kommunikation bei allen anderen Lobby-Requests erfolgt analog hierzu.

Drückt der User den Lobby beitreten-Button einer Lobby, wird im LobyService die Methode joinLobby(lobbyID: UUID, user: UserDTO, isBot: boolean) aufgerufen und ein LobbyJoinUserRequest gesendet. Empfängt der LobbyService ein LobbyJoinUserRequest, wird zunächst geprüft, ob der User der Lobby beitreten darf. Hierzu müssen folgende Bedingungen erfüllt sein:

- Eine Lobby mit der im Request enthaltenen ID existiert in der lobbies-Map
- Der User ist noch nicht in dieser Lobby drin
- Die maximale Spielerzahl der Lobby ist noch nicht erreicht
- In der Lobby ist kein Spiel aktiv

Ist eine dieser Bedingungen nicht erfüllt, wird dem User eine JoinLobbyExceptionMessage gesendet. Ansonsten wird aus dem User ein neuer LobbyUser erstellt und mit der Methode joinLobby(User: lobbyUser) dem Userset der Lobby hinzugefügt. Anschließend wird eine UserJoinedLobbyMessage an alle eingeloggten User gesendet. Handelt es sich bei dem User um einen Bot, wird zusätzlich der Bereitstatus des Users auf true (= bereit) gesetzt, eine UpdatedLobbyReadyStatusMessage gesendet und mit der Methode allPlayersReady(lobby: Lobby) geprüft, ob das Spiel starten kann.

3.4.2.3 Verlassen einer Lobby

Bei Erhalt eines LobbyLeaveUserRequest wird der betreffende User mit der Methode leaveLobby(lobbyID: UUID, user: User) aus dem Userset der Lobby entfernt, falls eine Lobby mit der angegebenen LobbyID existiert. War das Entfernen des Users erfolgreich, wird ggf. der Owner der Lobby aktualisiert und überprüft, ob keine User mehr oder nur noch Bots in der Lobby drin sind. Falls ja, wird diese über die Methode dropLobby(lobbyID: UUID) aus der lobbies-Map entfernt. Anschließend wird eine UserLeftLobbyMessage oder eine LobbyExceptionMessage, falls das Entfernen nicht erfolgreich war, gesendet.

Bei einem **Logout** verlässt der User über ein LeaveAllLobbiesOnLogoutRequest, welches gesendet wird, wenn der User erfolgreich ausgeloggt wurde, alle Lobbies, in denen er drin ist. Dazu wird über die lobbies-Map iteriert und der User aus jeder Lobby, in deren Userset er gespeichert ist, entfernt. Abschließend wird eine UserLeftAllLobbiesMessage gesendet.

3.4.2.4 Bereitstatus ändern

Empfängt der LobbyService ein UpdateLobbyReadyStatusRequest, wird über die Methode setReadyStatus(user: User, isReady: boolean) der entsprechende Eintrag in der readyStatus-Map der Lobby aktualisiert.

3.4.2.5 Spielstart

Ein Spiel startet in der Lobby, wenn alle Spieler bereit sind (und sich mehr als ein Spieler in der Lobby befindet). Dies wird im LobbyService über die Methode allPlayersReady(lobby: Lobby) geprüft, welche immer aufgerufen wird, wenn ein User seinen Bereitstatus ändert oder die Lobby verlässt. Kann das Spiel in der Lobby beginnen, wird das inGame-Attribut der Lobby auf true gesetzt und eine StartGameMessage sowie eine StartGameInternalMessage gesendet.

3.4.2.6 Owner-spezifische Funktionen

Neben den o. g. Aktionen gibt es weitere, die nur dem Lobbyowner zur Verfügung stehen. Darunter fallen das Verändern der maximalen Spieleranzahl, das Auswählen der Karten, mit denen gespielt werden soll, das Entfernen von Spielern und das Hinzufügen von Bots.

3.4.2.6.1 Ändern der maximalen Spieleranzahl

Bei Erhalt eines SetMaxPlayerRequest wird über die Methode setMaxPlayer(lobbyID: UUID, user: User, maxPlayerValue: int) versucht, den maxPlayer-Wert der Lobby zu aktualisieren. Gelingt dies nicht, da der User nicht der Owner der Lbby ist oder bereits drei Leute in der Lobby sind und versucht wird, maxPlayer auf zwei zu setzen, wird eine SetMaxPlayerException geworfen und eine LobbyExceptionMessage gesendet. War das Ändern erfolgreich, wird eine SetMaxPlayerMessage gesendet.

3.4.2.6.2 Kartenauswahl

Wird im LobbyService ein SendChosenCardsRequest empfangen, werden die im Request enthaltenen Karten über die Methode setChosenCards(cards: List<Short>) in der Lobby gespeichert und es wird eine SetChosenCardsResponse an den Client gesendet.

3.4.2.6.3 Hinzufügen eines Bots

Bei Erhalt eines AddBotRequest wird zunächst ein neuer BotPlayer erstellt und über ein LobbyJoinUserRequest der ensprechenden Lobby hinzugefügt. Außerdem wird ein AuthBotInternalRequest gesendet, woraufhin im AuthenticationService eine neue Session erstellt und der botSessions-Map hinzugefügt wird.

3.4.2.6.4 Kicken von Usern

Bei Erhalt eines KickUserRequest wird im LobbyManagement die Methode kickUser(id: UUID, userToKick: User, owner: User) mit der ID der Lobby, dem zu kickenden User und den User, der das Request gesendet hat, aufgerufen. In der kickUser-Methode wird geprüft, ob eine Lobby mit der angegebenen ID existiert und wenn ja, ob das Request vom Owner gesendet wurde. Falls ja, wird der zu kickende User aus der Lobby entfernt und es wird eine KickUserMessage an alle User der Lobby gesendet. Konnte der User nicht entfernt werden, wird eine KickPlayerException geworfen und dem Sender des Requests wird eine LobbyExceptionMessage gesendet.

3.4.3 Einloggen/Registrieren

Die Kommunikation des Einlogg-Vorgangs sowie der Registrierung ist bereits vom Basisprojekt abgedeckt, weshalb hier genauer auf die hinzugefügte Kommunikation und die clientseitigen Aktionen eingegangen wird.

3.4.3.1 Einloggen

Wird das Spiel gestartet und die Verbindung zum Server erfolgreich hergestellt, dann zeigt sich folgendes Fenster.

Abbildung 3.31: Loginfenster

Der Nutzer hat nun folgende Möglichkeiten:

1. Seine Logindaten einzugeben und mit einem Klick auf „Absenden“ zu bestätigen
2. Wenn das Passwort vergessen wurde, auf „Passwort vergessen“ zu klicken. Dieses hat jedoch noch keinerlei Funktion und wurde deswegen mit in den Ausblick aufgenommen.
3. Wenn noch kein Account angelegt wurde, auf „Ich habe noch keinen Account“ klicken, um einen Account zu erstellen (Siehe Registrierung)

Entscheidet sich der Nutzer dafür, seine bestehende Kombination aus Benutzername und Passwort einzugeben und dies mit einem Klick auf „Absenden“ zu bestätigen, werden die Daten mittels eines LoginRequests an den Server gesendet und folgendermaßen überprüft:

- Ist der „Benutzername“ nicht leer?
- Ist das „Passwort“ nicht leer?
- Ist der Benutzer vorhanden?
- Ist das Passwort des Benutzers korrekt?
- Ist der Benutzer bereits angemeldet?
- Ist ein anderer Fehler aufgetreten?

Ist der Benutzername und/oder das Passwort nicht gesetzt, so wird dem Nutzer folgende Fehlermeldung angezeigt:

Abbildung 3.32: Login Fehlgeschlagen - Nutzername/Passwort

Ist der Benutzer nicht in der Datenbank vorhanden oder das Passwort des vorhandenen Benutzers fehlerhaft, so wird dem Nutzer folgende Fehlermeldung angezeigt:

Abbildung 3.33: Login Fehlgeschlagen - Nutzer nicht vorhanden/ Passwort fehlerhaft

Ist der Benutzer auf dem Server bereits angemeldet, so wird dem Nutzer folgende Fehlermeldung angezeigt:

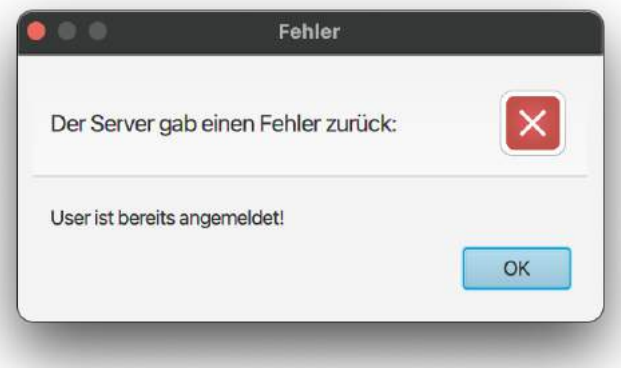

Abbildung 3.34: Login Fehlgeschlagen - Benutzer bereits angemeldet

Tritt bei der Anmeldung serverseitig ein Fehler auf, so wird dem Benutzer eine Meldung angezeigt, dass ein "Allgemeiner Fehler" bei der Authentifizierung aufgetreten ist.

War die Anmeldung erfolgreich, so wird serverseitig eine Bestätigung an den Client gesendet und dieser wechselt dann zum Hauptmenü.

3.4.3.2 Registrieren

Wenn der Nutzer auf „Ich habe noch keinen Account" klickt, so wird der „RegisterPresenter" angezeigt. Hier kann dieser nun zwischen den folgenden Möglichkeiten wählen:

1. Die für die Registierung notwendigen Daten eingeben und mit dem Klick auf „Absenden" bestätigen
2. Mit einem Klick auf „Ich habe bereits einen Account" zurück zum „LoginPresenter" gelangen

Entscheidet sich der Nutzer dafür, sich zu registrieren und seine Daten (Benutzername, E-Mail-Adresse und Passwort) einzugeben und mit einem Klick auf „Absenden" zu bestätigen, dann werden die eingegebenen Daten bereits clientseitig auf Fehler überprüft:

- Benutzername, Passwortfeld oder E-Mail-Feld ist leer
- Die eingegebenen Passwörter stimmen nicht überein
- Die angegebene Mailadresse ist nicht gültig

Wurde eines der geforderten Felder nicht ausgefüllt, erhält der Nutzer folgende Fehlermeldung:

Abbildung 3.35: Registrierung fehlgeschlagen - Ein gefordertes Feld wurde nicht ausgefüllt

Wurden unterschiedliche Passwörter eingegeben, erhält der Nutzer folgende Fehlermeldung:

Abbildung 3.36: Registrierung fehlgeschlagen - Die Passwörter stimmen nicht überein

Wurde eine ungültige Mailadresse angegeben, so erhält der Nutzer folgende Fehlermeldung:

Abbildung 3.37: Registrierung fehlgeschlagen - Die Mailadresse entspricht nicht dem geforderten Format

Die eingegebenen Werte werden bei richtiger Eingabe anschließend an den Server gesendet, der die gleichen Parameter erneut prüft und ggf. die gleichen Fehlermeldungen zurück an den Client sendet. Verlief hingegen alles fehlerfrei, so wird serverseitig der Benutzer angelegt und der Nutzer wird über die erfolgreiche Registrierung informiert.

3.4.4 Spiel

3.4.4.1 Struktur

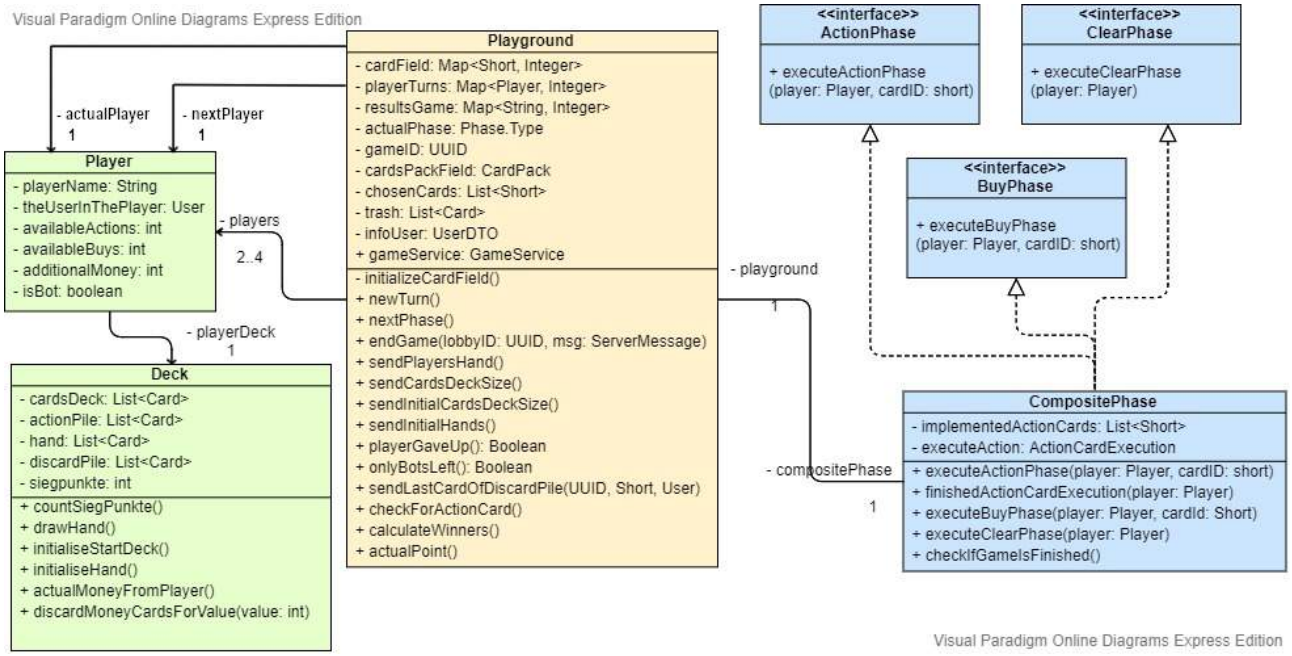

Abbildung 3.38: Spielstruktur

Auf dem Server findet die Spielverwaltung in den Klassen GameManagement und GameService statt. Dabei werden im GameManagement alle Spiele in einer Map<UUID, Game> games

gespeichert. Ein **Game** besitzt eine spezifische gameID: UUID, welche identisch zu der ID der Lobby, in der das Spiel gespielt wird, ist, einen Chat und einen Playground. (nicht im Diagramm oben dargestellt)

Im Playground werden alle **Player** in einer Liste players gespeichert. Jeder Spieler besitzt dabei ein Deck (playerDeck: Deck), eine Anzahl verfügbarer Aktionen (availableActions: int) bzw. Käufe (availableBuys: int) und zusätzliches Geld (additionalMoney: int). Es wird zudem über das Attribut isBot: boolean gespeichert, ob es sich um einen Bot handelt.

Im **Deck** sind alle Karten, die der Spieler besitzt, gespeichert. Dabei entspricht cardsDeck dem Nachziehstapel, discardPile dem Ablagestapel, actionPile der Aktionszone und hand der Hand. Es wird auch die Anzahl der Siegpunkte gespeichert. Die Klasse stellt Methoden zum Zählen der Siegpunkte, zum Initialisieren der Hand und des Startdecks bereit. Für die Durchführung eines Kaufs gibt es die Methoden actualMoneyFromPlayer(), welche bestimmt, wie viel Geld der Spieler auf der Hand hat, und discardMoneyCardsForValue(), welche genutzte Geldkarten ablegt.

Der **Playground** besitzt neben der Spielerliste eine CompositePhase sowie eine GameService-Instanz, eine ID, welche der gameID entspricht, und eine Map<String, Integer> resultsGame, welche das Endergebnis enthält, wobei der Key dem Namen des Spielers entspricht und der Value die von ihm erreichte Punktzahl ist. Da bei Punktegleichstand der Spieler mit den wenigsten Zügen gewinnt, wird die Anzahl an Zügen jedes Spielers in einer Map<Player, Integer> playerTurns gespeichert. In einer weiteren Map<Short, Integer> cardField wird gespeichert, welche und wie viele Karten noch auf dem Spielfeld vorhanden sind, wobei der Key der ID der Karte entspricht. Darüber hinaus wird gespeichert, welcher Spieler momentan am Zug ist (actualPlayer: Player), welcher als Nächstes dran ist (nextPlayer: Player) und in welcher Phase sich der Spieler momentan befindet (actualPhase: Phase.Type). Jeder Playground besitzt zudem einen Müll (trash) und einen infoUser, der Spielinfos (z.B. den Kauf einer Karte von Spieler xy) als Chatnachricht an alle Spieler sendet.

Die Klasse **CompositePhase** implementiert die Interfaces ActionPhase, BuyPhase und Clearphase und stellt die Methoden executeAction/Buy/ClearPhase bereit, welche für die Ausführung der Phasen zuständig sind. Mit der Methode checkIfGameIsFinished() wird nach jeder Clearphase geprüft, ob das Spiel zu Ende ist. In dieser Klasse werden außerdem, welche Aktionskarten spielbar sind (implementedActionCards) und die aktuelle ActionCardExecution gespeichert.

3.4.4.2 Initialisierung/Spielstart

Ein neues Spiel wird erstellt, wenn im GameService ein StartGameInternalRequest empfangen wird. Hierzu wird im GameManagement die Methode createGame(gameID: UUID) aufgerufen. Diese erstellt ein neues Game, welches die Lobby, in der das Spiel läuft, den Chat der Lobby sowie den GameService enthält. Darüber hinaus besitzt jedes Game eine Playground-Instanz, welche beim Erstellen eines Games erzeugt wird. Die Playground-Klasse stellt dabei das eigentliche Spiel dar. Wird ein neuer Playground erstellt, wird aus jedem User ein neuer Player erstellt und dessen Deck mit 7 Kupfer- und 3 Anwesenkarten initialisiert. Im Playground wird das Kartenfeld initialisiert, wobei die evtl. zuvor vom Owner gewählten Aktionskarten berücksichtigt werden. Hat dieser keine Karten oder weniger als zehn Karten ausgewählt, erfolgt die Auswahl der (restlichen) Aktionskarten, mit denen gespielt wird, per Zufall. Das erstellte Game wird dann der games-Map hinzugefügt und das Spiel wird über den Aufruf von newTurn() gestartet.

3.4.4.3 Spielablauf

Eine Runde im Spiel besteht aus drei Phasen: Der Aktionsphase, der Kaufphase und der Clearphase. Eine neue Runde wird im Playground über die Methode **newTurn()** gestartet. Wird die Methode zum ersten Mal aufgerufen, werden zunächst actual- und nextPlayer initialisiert, wobei der erste Nicht-Bot Spieler in der Spielerliste anfängt und der Spieler, der ihm in der Liste folgt, als Nächstes dran ist. Wird das Ende der Liste erreicht, wird wieder von vorne angefangen. Bevor der aktuelle Spieler wechselt, wird aber noch dem alten aktuellen Spieler seine neue Hand sowie die Anzahl der Karten auf seinem Nachziehstapel gesendet. Jedes Mal, wenn ein neuer Spieler am Zug ist, wird in der Methode die Anzahl seiner Züge in der playerTurns-Map um eins erhöht. Anschließend wird mit Hilfe der Methode checkForActionCard() geprüft, ob der Spieler eine Aktionskarte auf der Hand hat. Ist dies der Fall, wird eine StartActionPhaseMessage an alle Spieler gesendet, ansonsten wird über den Aufruf von nextPhase() die Aktionsphase übersprungen.

Der Wechsel zwischen den einzelnen Phasen eines Spiels findet über die Methode **nextPhase()** im Playground statt. Diese wird entweder automatisch aufgerufen, bspw. wenn der Spieler in der Kaufphase keine Käufe mehr zur Verfügung hat, oder durch ein SkipPhaseRequest vom Spieler selbst. Die Methode überprüft dann, in welcher Phase sich der Spieler befindet. Befindet er sich in der Aktions- oder Kaufphase, wird die aktuelle Phase aktualisiert und dem Spieler eine StartBuy- bzw. StartClearPhaseMessage gesendet. Falls der Spieler in die Clearphase gewechselt ist, wird diese dann über die Methode compositePhase.executeClearPhase(actualPlayer: Player) ausgeführt. Befindet sich der Spieler momentan in der Clearphase, wirft die Methode eine GamePhaseException, da diese nicht übersprungen werden darf.

3.4.4.4 Überspringen von Phasen

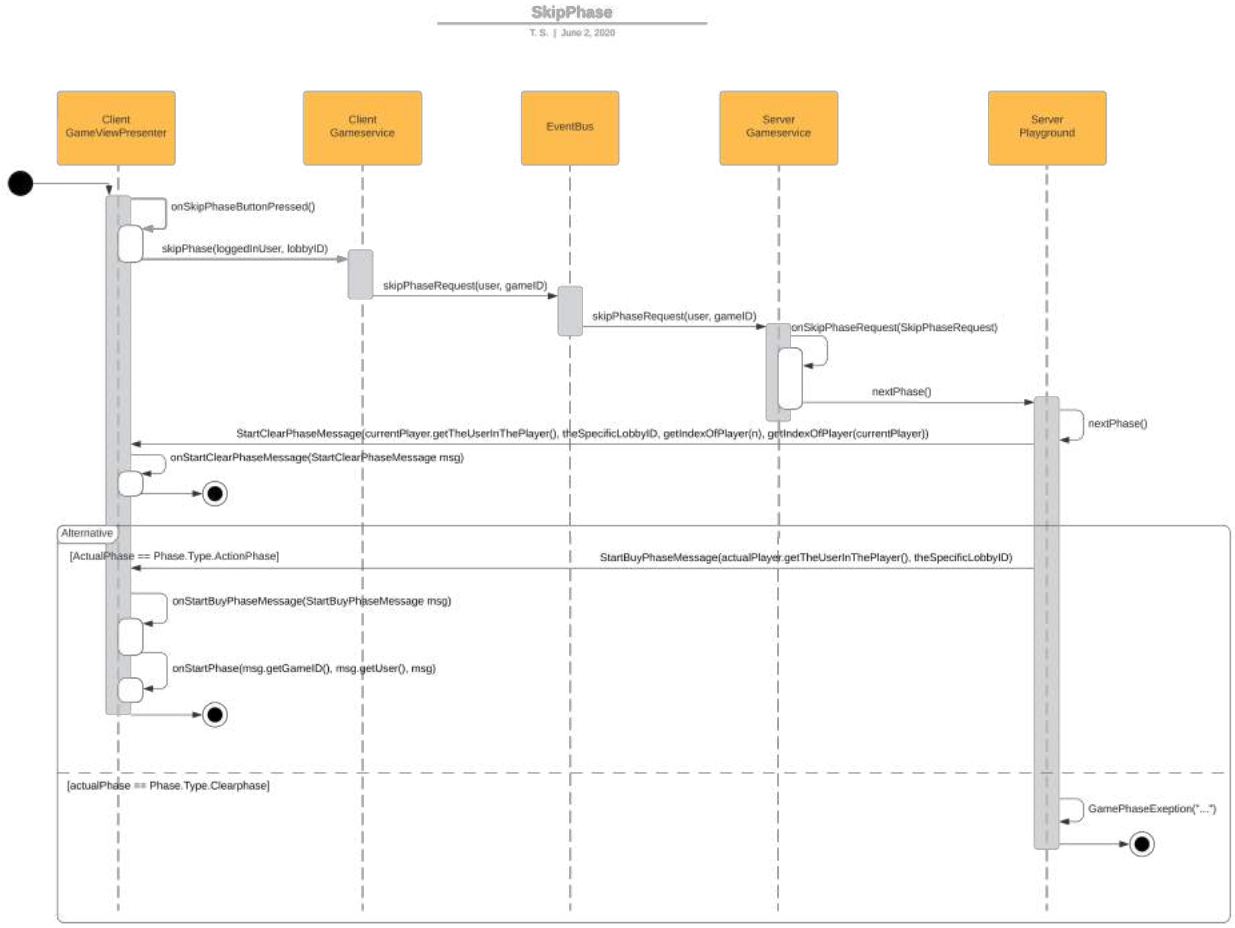

Abbildung 3.39: Überspringen einer Phase

Der Spieler ist nicht verpflichtet in der Aktionsphase eine Aktionskarte zu spielen oder in der Kaufphase eine Karte zu kaufen. Entscheidet er sich dafür, die Phase zu überspringen und drückt den SkipPhase-Button, wird ein SkipPhaseRequest gesendet. Wird dieses im GameService empfangen, wird zunächst überprüft, ob der User dem actualPlayer entspricht. Ist dies der Fall, wird über den Aufruf von nextPhase() versucht, die aktuelle Phase zu überspringen. Ist dies erfolgreich, wird den Spielern eine entsprechende Message gesendet (vgl. vorheriger Absatz). Wird eine GamePhaseException geworfen, wird dem Spieler eine GameExceptionMessage mit der Message der Exception gesendet.

3.4.4.5 Aufgeben

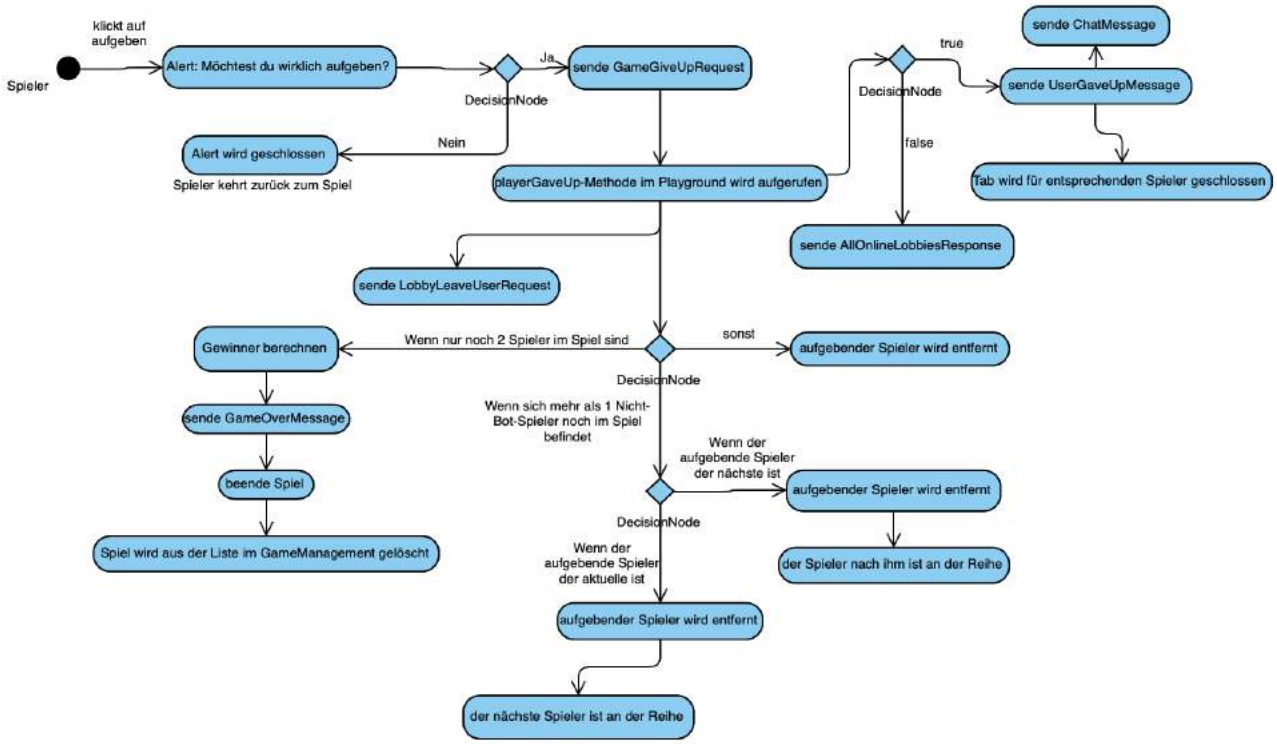

Abbildung 3.40: Aufgeben

Ein Spieler kann jederzeit aufgeben. Drückt er auf den Aufgeben-Button erscheint zunächst ein Alert. Bestätigt der Spieler dort, dass er aufgeben möchte, wird ein GameGiveUpRequest gesendet. Wird dieses vom GameService empfangen, wird im entsprechenden Playground die Methode playerGaveUp aufgerufen und es wird ein LobbyLeaveUserRequest für den aufgebenden Spieler gesendet. Die playerGaveUp-Methode prüft zunächst, wie viele Spieler sich noch im Spiel befinden. Sind nur noch zwei Spieler im Spiel, wird der Gewinner über die Methode calculateWinners() im Playground ermittelt und es wird eine GameOverMessage mit dem Spielergebnis an alle Spieler gesendet. Dann, und auch wenn sich nur noch Bots im Spiel befinden, wird das Spiel aus der games-Map im GameManagement entfernt und es wird ein UpdateInGameRequest gesendet, wodurch der inGame-Status der Lobby wieder auf false gesetzt wird.

Sind noch mindestens zwei weitere Nicht-Bot-Spieler im Spiel, wird der aufgebende Spieler aus der Spielerliste des Playgrounds entfernt. Handelt es sich dabei um den aktuellen Spieler, wird anschließend die aktuelle Phase auf Clearphase gesetzt und der nächste Spieler ist an der Reihe (Aufruf von newTurn()). Handelt es sich bei dem aufgebenden Spieler um den nextPlayer, wird dieser aktualisiert. Abschließend wird eine UserGaveUpMessage gesendet und das Aufgeben wird den anderen Spielern im Chat angezeigt und der entsprechende Tab wird für den User, der aufgegeben hat, gelöscht.

3.4.4.6 Der JSONCardParser

Der JsonCardParser ist ein Werkzeug, welches zum Laden einer eigens für das Spiel angefertigte Logiksprache im JSON-Format entwickelt wurde. Diese Logik-Sprache enthält diverse Befehle, welche wiederum Attribute und Eigenschaften enthalten, die die Ausführung des Befehls bestimmen.

3.4.4.6.1 Aufbau

Die Oberklasse des JsonCardParsers ist die Klasse JsonCardParser, welche eine öffentliche Methode 'loadpack' zur Verfügung stellt. Diese Methode versucht, mittels des von Google zu Verfügung gestellten GsonBuilders, die in JSON-Format gespeicherte Kartenlogik zu Laden und in ein Java-verständliches Card- mit CardAction-Objekt zu wandeln. Da es nicht so einfach ist, die gesamte Logik einfach vollatomatisch interpretieren zu lassen, wurden hierfür einige Hilfsklassen geschrieben, die alle nichttrivialen Interpretations-Fälle bearbeiten.

Im Wesentlichen besteht der Deserialisierer aus folgenden Komponenten:

- Der JsonCardParser: Oberklasse, der den Deserialisierer initialisiert.
- Der ValueCardDeserialisierer: Dient zum Parsen einer ValueCard bzw. Anwesenkarte.Dieser Parser muss lediglich den Namen, die ID, die Kosten und den Wert der Karte auslesen.
- Der MoneyCardDeserialisierer: Dient zum Parsen einer MoneyCard bzw. Geldkarte.Dieser Parser muss lediglich den Namen, die ID, die Kosten und den Wert der Karte auslesen.
- Der CurseCardDeserialisierer: Dient zum Parsen einer CurseCard bw. Fluchkarte. Dieser Parser muss lediglich den Namen, die ID, die Kosten und den Wert der Karte auslesen.
- Der ActionCardDeserialisierer: Dient zum Parsen einer ActionCard bzw. Aktionskarte. Dieser Parser muss nicht nur die Basiswerte einer Karte, sprich den Namen, die ID und die Kosten auslesen, sondern mitunter auch die Aktionen, die die Karte beim Ausspielen auslöst beinhalten.Um diese Aktionen nun auch auslesen zu können werden folgende weitere spezielle Deserialisierer verwendet.
- Der CardActionArrayDeserialisierer: Dient zum Parsen einer Liste an CardActions bzw. Kartenaktionen. Eine Kartenaktion kann einer der 11 Befehle sein, welche in 3.10 näher beschrieben sind. Um diese individuellen Kartenaktionen laden zu können bedarf es folgende weiterere Deserialisierer, welche speziellere Attribute einer Kartenaktion laden können.
  - Der ValueDeserialisierer: Dient zum Parsen von einer Value bzw. Wert (in diesem Falle ein Wertebereich eines Attributs einer Kartenaktion). Eine Value hat einen minimalen und einen maximalen Wert. Beide Werte sind vom Typ short und Optional in der Angabe, allerdings muss mindestens eines der beiden Werte bei Angabe dieser Eigenschaft gesetzt sein.
  - Der ConditionDeserialisierer: Dient zum Parsen einer Condition bzw. Bedingung. Eine Bedingung wird für einige Befehle verwendet, um entscheiden zu können ob diese ausgeführt wird bzw. auch wiederholt ausgeführt wird.
    Eine Bedingung besteht aus folgenden Komponenten:

Einen jeweils linken und rechten Werteparameter, welcher entweder ein Integer-Wert enthalten darf, oder eine Reihe an Aktionen, die wiederum einen Integerwert oder einen Boolean-Wert zurückgibt.

Einem mittleren Vergleichsoperator, welcher angegeben werden muss, falls es sich beim linken oder rechten Werteparameter nicht um einen Boolean-Wert als Rückgabe handelt.

3.4.4.6.2 Folgende Hierarchie verwendet der Parser:

...cards/packs/PAKETNAME/
    PAKETNAME.json
    images/
        KARTENID.png
        ...

3.4.4.6.3 Folgendes Format kann der Parser auslesen:

Attributname	Datentyp	Beschreibung
name	String	Name des Kartenpaketes (muss mit Datei und Ordnernamen übereinstimmen)
description	String	Beschreibung des Kartenpaketes (kann bei der Auswahl inGame z.B. angezeigt werden)
cards	Array	Array, welches die verschiedenen Kartenstapel eines Typs enthält

Tabelle 3.1: Kartenpack

Attributname	Datentyp	Beschreibung
(Optional)MoneyCards, ValueCards, ActionCards, CurseCards	Array	Array, welches die verschiedenen Karten eines gleichen Typs enthält

Tabelle 3.2: Kartenstapel

Attributname	Datentyp	Beschreibung
id	short	Nummer, welche jede Karte eindeutig identifiziert. (Jede ID, unabhängig vom Kartenpaket, darf nur einmal verwendet werden)
name	String	Name der Karte
(Money- & ValueCards) value	short	Kaufkraft bzw. Wertigkeit der Karte
cost	short	Preis der Karte
(ActionCards) actions	Array	Array der Aktionen, die diese Karte auslöst (Beispiele: Siehe Aktionen)

Tabelle 3.3: Karte

3.4.4.6.4 Kartenaktionen Enums

Wert	Beschreibung
NONE	Kein festgelegter Kartentyp
MoneyCard	Geldkarte
ActionCard	AktionsKarte
ValueCard	Anwesenkarte
CurseCard	FluchKarte

Tabelle 3.4: Card.Type: Typ einer Karte

Wert	Beschreibung
TRASH	Müll
HAND	Hand
BUY	Kaufzone
DRAW	Nachziehstapel
DISCARD	Ablegestapel
TEMP	Temporärer Stapel

Tabelle 3.5: Playground.ZoneType

Wert	Beschreibung
NONE	mich
ALL	Alle Spieler
OTHERS	Alle anderen Spieler
LAST	Vorangegangener Spieler
NEXT	Nächster Spieler

Tabelle 3.6: Action.ExecuteType: Aktion anwenden auf... (aus Sicht des aktuellen Spielers)

Wert	Beschreibung
REACTION	Reaktion auf eine Angriffskarte
ATTACK	Angriffskarte
ACTION	Aktionskarte

Tabelle 3.7: CardAction.Type: Typ einer Aktionskarte

Wert	Beschreibung
BUY	Mögliche Kaufaktionen
ACTION	Mögliche Aktionskarten
MONEY	Verfügbares Geld

Tabelle 3.8: AbstractPlayground.PlayerActivityValue: Repräsentiert mögliche Typen von spielerspezifischen Eigenschaften.

3.4.4.6.5 Aktionseigenschaften

Jede Aktion hat außerdem eine Reihe an Eigenschaften, die bei der Ausführung beachtet werden müssen. Ein Wert gilt als unbegrenzt bzw. beliebig, wenn dieser 255 bzw. -255 beträgt. (0-*)

Ein Wert gilt als maximaler bzw. minimaler Wert, wenn dieser 254 bzw. -254 beträgt. (Bsp.: Ein Spieler hat 4 Karten auf der Hand und eine Aktion z.B. GetCard gibt die Eigenschaft count mit 254 und cardSource mit Hand an, so sind alle Karten auf der Hand gemeint. Also 4)

Die einzelnen Aktionen einer Aktionskarte sind getrennt voneinander zu betrachten. Lediglich die Eigenschaft nextAction sorgt dafür, dass das Ergebnis der aktuellen Aktion der Aktion in nextAction übergeben wird.

Name	Wertebereich	Beschreibung
allowedCardtype	Card.Type	Gibt an, welcher Kartentyp bei der Aktion verwendet werden darf
Worth	short !=0	Gibt an, wie hoch der Wert der durch die Aktion aufgenommene/ abgelegte Karte mindestens sein muss
Cost	short !=0	Gibt an, wie hoch der Preis der durch die Aktion aufgenommene/ abgelegte Karte mindestens sein muss
executeType	Action.ExecuteType	Gibt an, auf wen die Aktion anzuwenden ist aus Sicht des aktuellen Spielers.
Value	short !=0 <= maxValue	Gibt an, wie hoch der Value-Übergabeparameter mindestens sein muss.
equalizeCards	Boolean	Gibt an, ob für jede hinzugefügte Karte eine entfernt werden muss und umgekehrt.
cardSource	Playground.ZoneType	Gibt an, woher eine Karte genommen werden muss.
cardDestination	Playground.ZoneType	Gibt an, wohin eine Karte gelegt werden muss.
executionOptional	Boolean	Gibt an, ob die Ausführung der Aktion durch den Benutzer abgebrochen werden kann
executionOptionalMessage	String	Optionale Aktion als Text, der dem Spieler angezeigt wird
removeCardAfter	Boolean	Gibt an, ob die Karte, nachdem sie gespielt wurde, Entsorgt wird.
hideCardDuringAction	Boolean	Gibt an, ob Karten, die durch diese Aktion abgelegt/ gezogen wurden, den anderen Spielern sichtbar sind.
nextAction	action	Aktion, die das ergebnis der vorrangegangenen Aktion als Eingabe erhält.

Tabelle 3.9: Aktionseigenschaften

Name	Parameter	Beschreibung
AddCapablePlayerActivity	short != 0, PlayerActivity-Value	Fügt mögliche Aktivitäten des Spielers hinzu bzw. zieht welche ab
ChooseCard	count, cardSource	Sorgt dafür, dass der Spieler eine Anzahl Karten auswählen muss
GetCard	count, cardSource	Holt Karten-Informationen von einer bestimmten Zone
If	condition, action	Prüft, ob eine vorrangegangene Aktion erfolgreich war und führt anschließend eine weitere Aktion aus
UseCard	count, Card	Benutzt eine Aktionskarte so oft wie angegeben.
ShowCard	Card	Zeigt die angegebene Menge Karten des angegebenen Spielers (cardSource benötigt)
While	condition,actions	Führt eine Aktion, solange eine Bedingung erfüllt ist, aus
Move	Card-Array	Bewegt die übergebenen Karten anhand der festgelegten Parameter. (cardSource, cardDestionation benötigt)
ChooseNextAction	Action-Array	Stellt den Spieler vor die Wahl, welche Aktion als nächstes ausgeführt werden soll.
Count	Card-Array	Gibt die Anzahl an Karten in einem Array wieder zurück
ForEach	Card-Array, Action-Array	Führt eine Reihe an Aktionen auf eine Reihe an Karten einzeln auf jede Karte an

Tabelle 3.10: Aktionen. Anmerkung: If und While wurden aus Zeitgründen nicht implementiert.

3.4.4.7 Aktionsphase

Wird ein PlayCardRequest empfangen, wird zunächst überprüft, ob der User, der das Request gesendet hat, momentan an der Reihe ist und wenn ja, ob er sich auch in der Aktionsphase befindet. Trifft dies zu, wird die Karte über den Aufruf von executeActionPhase in der CompositePhase-Klasse ausgespielt und den Spielern wird (falls das Ausspielen erfolgreich war) eine PlayCardMessage sowie eine ChatMessage, die anzeigt, dass der Spieler die Karte gespielt hat, gesendet.

In der Methode executeActionPhase wird zunächst die zu der ID gehörige Karte aus dem CardPack geholt. Existiert keine Karte mit der übergebenen ID, wird eine IllegalArgumentException geworfen. Dann wird geprüft, ob der Spieler diese Karte auf der Hand hat und ob diese ausspielbar ist. Trifft eines davon nicht zu, wird auch hier eine IllegalArgumentException geworfen. Kann die Karte gespielt werden, wird dem Attribut executeAction eine neue Instanz der ActionCardExecution-Klasse zugewiesen, welche mit new ActionCardExecution(cardId, playground, false) erstellt wird. Der letzte Parameter gibt dabei an, ob die Aktionskarte mit Hilfe einer anderen Aktionskarte (z.B. Thronsaal) gespielt wurde. Anschließend wird die neue Klasse auf dem EventBus registriert und die gespielte Karte wird aus der Hand des Spielers entfernt.

Die eigentliche Ausführung der Karte findet dann in der ActionCardExecution-Klasse statt. Diese besitzt folgende Attribute:

- theCard: ActionCard - Die Aktionskarte
- player: Player - Spieler der Aktionskarte
- playground: Playground

- nextActions: List<CardAction> - Liste aller Unteraktionen der aktuellen Aktion
- parent: ActionCardExecution - ActionCardExecution, aus der diese ActionCardExecution-Instanz erzeugt wurde (nur bei der Karte Thronsaal genutzt)
- execution: ActionCardExecution: ActionCardExecution, die für die Ausführung von useCard erzeugt wurde
- inputCard: Card - Vom Spieler gewählte Karte
- useAction: UseCard - Die aktuell ausgeführte UseCard-Aktion
- useCardExecution: boolean - Ob die Klasse zur Ausführung von UseCard dient
- executeOptionalAction: boolean - Ob eine optionale Aktion ausgeführt werden soll
- removeCardAfter: boolean - Ob die Karte nach Ausspielen entsorgt wird
- finishedExecution: boolean - Ob die Karte vollständig ausgeführt wurde
- startedNextActions: boolean - Ob mit der Ausführung der Unteraktionen begonnen wurde
- finishedNextActions: boolean - Ob alle Unteraktionen einer Aktion abgearbeitet wurden
- waitedForPlayerInput: boolean - Ob auf eine Auswahl des Spielers gewartet wird
- useCardCounter: int - Wie oft eine Karte im Zuge von UseCard schon ausgespielt wurde
- actionExecutionID: int - ID der ActionCardExecution. Diese ist 0, falls die Karte direkt ausgespielt wurde und 1, falls sie durch eine andere Aktionskarte (Thronsaal) gespielt wurde
- actualStateIndex: int - Index der aktuell ausgeführten Aktion
- nextActionIndex: int - Index der aktuell ausgeführten Unteraktion

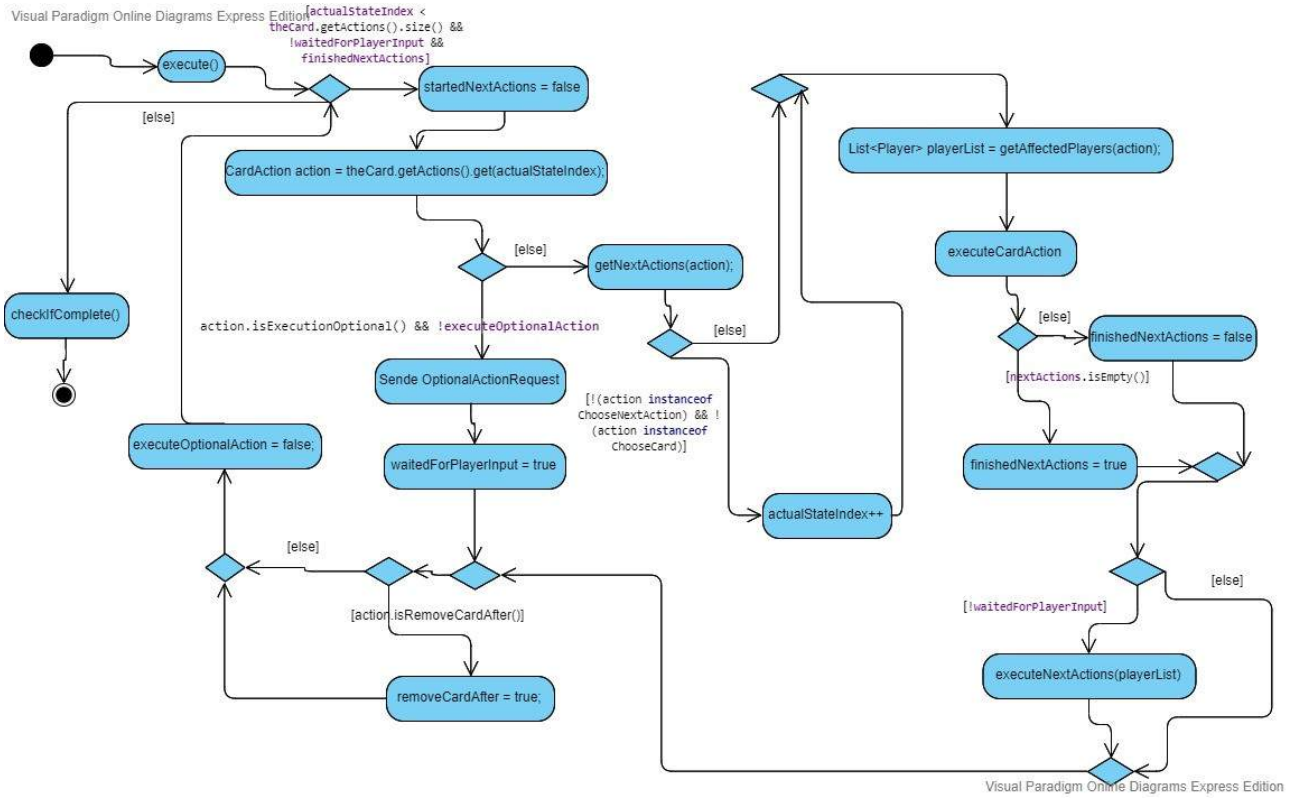

Abbildung 3.41: Ausführung einer Aktionskarte

Die Ausführung einer Aktionskarte findet in der Methode execute() statt:
In einer while-Schleife passiert Folgendes solange wie die Bedingung actualStateIndex < card.getActions().size() && !waitedForPlayerInput && finishedNextActions erfüllt ist:

1. startedNextActions wird auf false gesetzt
2. Die aktuelle CardAction wird aus card.getAction() geholt und im Attribut action gespeichert.
3. Ist action eine ComplexCardAction und das Attribut executionOptional true sowie executeOptionalAction false, wird ein OptionalActionRequest an den Spieler gesendet und waitedForPlayerInput auf true gesetzt.
   Wird eine OptionalActionResponse empfangen, wird in der Methode onOptionalActionResponse waitedForPlayerInput wieder auf false gesetzt. Hat sich der Spieler dafür entschieden, die Aktion ausführen zu sollen, wird executeOptionalAction auf true gesetzt, ansonsten wird der actualStateIndex um 1 erhöht. Dann wird wieder execute() aufgerufen.
4. Ist die Bedingung aus 3. nicht erfüllt, werden über den Methodenaufuf getNextAction(action) alle Unteraktionen aus action geholt und in nextActions gespeichert
5. Ist action weder eine Instanz von ChooseNextAction noch und eine von ChooseCard, wird der actualStateIndex um 1 erhöht.
6. Dann werden über getAffectedPlayers(action) alle Spieler, auf die die Aktion ausgeführt werden soll, ermittelt und in einer Liste (playerList) gespeichert. Handelt es sich bei der action um eine Get- oder Move-Instanz, wird executeCardAction(action, null, playerList, false) aufgerufen, andernfalls wird executeCardAction(action, playerList) aufgerufen. Beide Methoden führen dann die entsprechende Aktion aus.

7. Je nachdem ob nextActions leer ist, wird anschließend finishedNextActions auf true bzw. false gesetzt. Ist waitedForPlayerInput false, wird daraufhin executeNextActions(playerList) aufgerufen.

8. Falls action.isRemoveCardAfter() true ist, wird removeCardAfter auf true gesetzt. Abschließend wird exeuteOptionalAction auf false gsetzt.

Ist die Ausführung einer Aktion inklusive ihrer Unteraktionen beendet, wird über die Methode checkIfComplete geprüft, ob alle Aktionen vollständig abgearbeitet wurden. Falls nein, wird wieder execute() aufgerufen. Falls doch, wird (je nachdem ob removeCardAfter false oder true ist) die gespielte Karte in die Aktionszone des Spielers oder auf den Müll gelegt und in CompositePhase wird die Methode finishedActionCardExecution aufgerufen. In dieser Methode wird dann availableActions des Players um eins verringert, ihm wird eine neue InfoPlayDisplay gesendet und er wechselt über playground.nextPhase() in die Kaufphase, falls er keine Aktionen mehr spielen kann oder keine Aktionskarte mehr auf der Hand hat.

3.4.4.8 Kaufphase

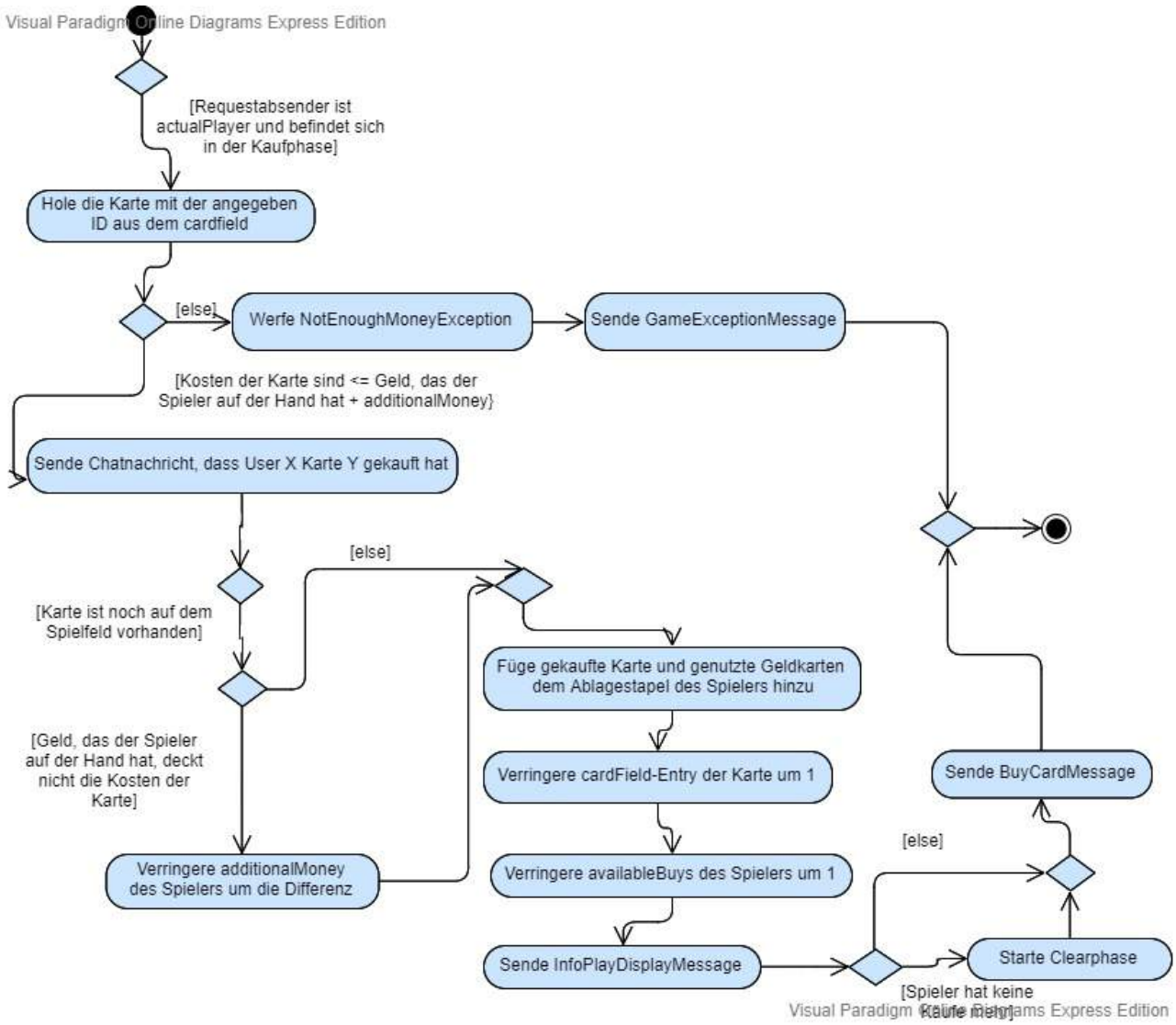

Abbildung 3.42: Kauf einer Karte

Wird ein BuyCardRequest empfangen, wird im GameService zuerst geprüft, ob der User eine Karte kaufen darf, d.h. er muss der aktuelle Spieler sein und sich in der Kaufphase befinden. Ist dies der Fall, wird anschließend geprüft, ob er genug Geld zur Verfügung hat, um die im Request angegebene Karte zu kaufen. Ist dies nicht der Fall, wird eine NotEnoughMoneyException geworfen und dem Spieler wird eine GameExceptionMessage gesendet. War der Kauf dagegen erfolgreich, wird eine BuyCardMessage gesendet.

Hat er genug Geld zur Verfügung, wird eine neue ChatMessage gesendet, welche das Kaufen der Karte im Spielchat anzeigt. Dann wird die Methode executeBuyPhase(player: Player, cardID: short) in der Klasse CompositePhase aufgerufen, wo der Kauf ausgeführt wird. Dort wird zunächst mit Hilfe der cardField-Map geprüft, ob die Karte noch auf dem Spielfeld vorhanden ist. Ist dies der Fall, wird daraufhin geprüft, ob der Spieler zum Kauf sein zusätzliches Geld nutzen muss. Falls ja, wird das additionalMoney-Attribut des Spielers um die entsprechende Differenz verringert. Die Karte wird dann dem Ablagestapel des Spielers hinzugefügt, für den Kauf genutzte Karten werden von der Hand auf den Ablagestapel verschoben und der cardField-Eintrag der Karte wird um eins verringert. Anschließend wird die Anzahl verfügbarer Käufe des Spieler um eins verringert und ihm wird eine InfoPlayDisplayMessage gesendet. Hat der Spieler keine Käufe mehr zur Verfügung, wird die Clearphase über den Aufruf von playground.nextPhase() gestartet.

3.4.4.9 Clearphase

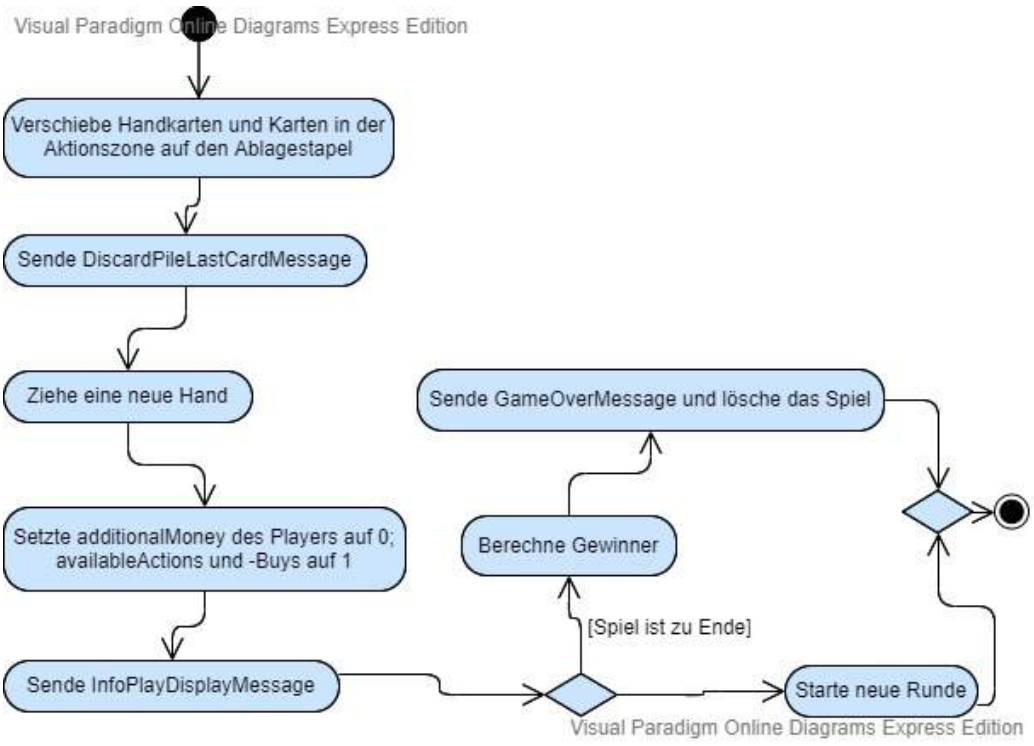

Abbildung 3.43: Clearphase

Die Clearphase wird in der Klasse CompositePhase über die Methode executeClearPhase(player: Player) ausgeführt. Dort werden zunächst alle Karten, die sich auf der Hand oder in der Aktionszone des Spielers befinden, auf seinen Ablagestapel verschoben und den Spielern wird die letzte Karte auf seinem Ablagestapel über eine DiscardPileLastCardMessage gesendet. Danach

wird über die Methode deck.drawHand() eine neue Hand für den Spieler gezogen. Sind hierfür nicht mehr genügend Karten auf seinem Nachziehstapel vorhanden, wird der Ablagestapel gemischt und zum neuen Nachziehstapel und die noch fehlenden Karten werden gezogen. Anschließend werden availableActions, availableBuys und additionalMoney des Spielers wieder auf die Standardwerte gesetzt (1 Aktion, 1 Kauf und 0 zusätzliches Geld) und ihm wird wieder eine InfoPlayDisplayMessage gesendet. Abschließend wird über die Methode checkIfGameIsFinished() überprüft, ob das Spiel zu Ende ist. Dies ist der Fall, wenn entweder der Provinzstapel oder drei beliebige andere Stapel leer sind. Ist das Spiel zu Ende, wird der Gewinner über die Methode playground.calculateWinners() ermittelt. Das Spielergebnis wird den Spielern dann in einer GameoverMessage gesendet und das Spiel wird gelöscht sowie der inGame-Status der Lobby auf false gesetzt. Ist das Spiel nicht zu Ende, wird eine neue Runde über playground.newTurn() gestartet.

3.4.4.10 Bots

Bei den Bots handelt es sich um Spieler die automatisch vom Server gesteuert werden. Diese werden vom Spieler über die Lobby hinzugefügt oder entfernt. Der Bot hat beim Erstellen immer den Status, dass er bereit ist, um in das Spiel zu starten. Ebenfalls wird ihm beim Erstellen ein zufälliger Name zugewiesen mit einer zufälligen Nummer am Ende. Beim Starten ins Spiel erhält er, so wie ein normaler Spieler, die Messages und verarbeitet diese jeweils in einem eigenen Thread. Außerdem wird vor dem Ausführen einer Aktion immer eine bestimmte Zeit abgewartet, damit das Ausspielen der Karten beim Bot organischer aussieht.

3.4.4.10.1 Kaufen von Karten

Wenn der Bot Karten kaufen soll, kriegt er die StartBuyPhaseMessage. Dabei wird erst geguckt, wie viele Käufe der Bot hat. So oft wird er durch den Entscheidungsprozess durchgehen, welche Karte er kaufen soll. Zu Anfang überprüft er, wie viele Aktionskarten er auf der Hand hat. Dann geht er durch den Entscheidungsprozess, welche Karte er kaufen will. Wenn er mindestens acht Geld hat kauft er eine Provinzkarte; wenn er schon in Runde 25 ist und mindestens 5 Geld hat, kauft er ein Herzogtum. Wenn diese beiden Sachen nicht zutreffen, dann wird geguckt, ob er weniger als 20 Prozent an Aktionskarten auf der Hand hat. Trifft dies zu, dann kauft er die teuerste mögliche Aktionskarte. Wenn das auch nicht zutrifft, dann wird geguckt, dass er die teuerste Geldkarte holt. Schließlich schickt er dann die BuyCardRequest ab und rechnet neu aus, wie viel Geld er auf der Hand hat.

3.4.4.10.2 Ausspielen von Karten

In der Aktionsphase kriegt der Bot die StartActionPhaseMessage. Dabei guckt der Bot erst einmal darauf, wie viele Aktionsphasen er hat. Wenn er mehr als null hat, dann führt er den folgenden Code aus: Es wird nach der teuersten Aktionskarte auf der Hand gesucht und diese wird dann ausgespielt. Nach dem die Karte ausgespielt wurde kriegt der Bot die PlayCardMessageMessage zurück. Dabei entnimmt er dem Wert der Karte ob noch eine Aktionsphase gespielt werden muss oder ob es zum Beispiel eine weitere Kaufphase gibt. Wenn es eine Karte ist, wo eine Karte ausgewählt werden muss, dann wird auf Folgendes geachtet:

- Keller: Es werden alle Kupfer- und alle Punktekarten abgelegt
- Mine: Es wird die Geldkarte mit dem niedrigsten Wert abgelegt

- Umbau: Es wird die Karte mit dem niedrigsten Wert entsorgt
- Kapelle: Es werden alle Kupferkarten und Fluchkarten entsorgt.

3.4.5 Chat

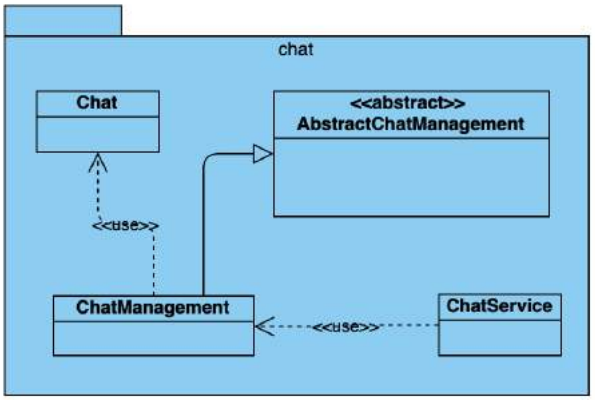

Abbildung 3.44: Klassendiagramm zum Chat

Auf dem Server gibt es die Klasse Chat. Für jeden neuen Chat wird eine Instanz dieser Klasse mit der zugehörigen ID erstellt. Hier werden die Nachrichten gespeichert. Die jeweilige Instanz der Chat-Klasse wird im ChatManagement erstellt und durch dieses dann auch verwaltet. Das ChatManagement erweitert das AbstractChatManagement, welches wiederum das ChatService-Interface aus dem Common implementiert. Die Kommunikation mit dem Client findet im ChatService statt. Hier werden die vom Client gesendeten ChatMessageRequests abgefangen und verarbeitet, also über das ChatManagement dem Chat hinzugefügt. Dann wird eine Antwort erstellt, welche an den Client zurückgeschickt wird, damit dieser dann die Nachricht dem User anzeigen kann.

Kapitel 4

Testdokumentation

Das folgende Kapitel bietet Aufschluss auf die von uns durchgeführten Testverfahren und die damit verbundene Testabdeckung. Im Einzelnen gliedert sich das Kapitel wie folgt:

- Testvorgehen
- Verfügbare Tests
- Testabdeckung
- Nicht verfügbare Tests
- Beschreibung - Codereview

4.1 Testvorgehen

4.1.1 JUnit 5

Innerhalb des Softwareprojektes wurden verschiedene Tests angelegt. Getestet wurde mit Hilfe des Tools JUnit 5. JUnit 5 ist dabei eine Art Programmiergerüst, welches das Testen von Methoden, Klassen und weiteren Implementierungen ermöglicht und ist standardgemäß in IntelliJ integriert.

4.1.2 Die drei A bei einem Test

Ein grundsätzlicher Test besteht dabei immer aus drei Teilen (3 A's). Diese lauten „Arrange“, „Act“ und „Assert“.

Innerhalb des „Arrange“ werden dabei die benötigten Objekte erzeugt und es findet eine Konfiguration statt. Dies könnte zum Beispiel die Definition und Initialisierung einer für den Test benötigten Variable sein.

Beim folgenden „Act“ interagiert der Test mit den Methoden und erzeugten Objekten. Dies kann zum Beispiel ein Methodenaufruf sein.

Anschließend wird beim „Assert“ geprüft, ob sich sämtliche Testeigenschaften wie erwartet verhalten haben. Hierbei wird geprüft, ob das erwartete Ergebnis dem durch den Test erzielten Ergebnis entspricht. Positiv verlaufene Tests werden dabei grün gekennzeichnet, fehlerhafte rot.

4.1.3 Beispieltests

Ein Beispiel für einen Test wäre die Testmethode „assertEquals(Erwartet, Variable/Methode)“ oder auch „assertTrue/False(...)“.

Beim assertEquals() findet dabei ein Vergleich zwischen zwei Objekten oder auch Variablen statt. Entspricht das Ergebnis dem Erwarteten, so ist der Test erfolgreich. Andernfalls schlägt dieser fehl. Beim assertTrue/False() wird der Programmteil innerhalb der Klammern ausgeführt beziehungsweise der Wahrheitswert ermittelt. Ist der Wahrheitswert wahr bzw. falsch, so ist der Test erfolgreich.

4.1.4 Regeln

1. Anlegen einer seperaten, eindeutigen Testklasse ist Pflicht.
2. Java Richtlinien sind einzuhalten (eindeutige Variablennamen,...)
3. Ein Test wird mit @Test gekennzeichnet
4. Ein Test muss einen eindeutigen, identifizierbaren und selbstredenden Namen haben. Zum Beispiel: „testGiveUp()“
5. Ein einzelner Test sollte der Übersicht halber nicht zu komplex sein (ein Test für eine Funktion).
6. Ein Test muss einen Zweck erfüllen. Sinnlose Tests sind nicht förderlich
7. Überprüfe den Test selbst. Logische Fehler werden nicht erkannt. Das heißt, ein logisch falscher Test kann trotzdem erfolreich sein, obwohl dieser fehlschlagen müsste.

4.1.5 Testschema - Grundsätzlicher Ablauf

Die Erstellung eines Tests verläuft in der Regel nach folgendem Schema unter Berücksichtigung der drei A und der Regeln:

1. Anlegen einer seperaten Testklasse
2. Festlegen, was getestet werden soll
3. Arrange (Definiton und intitalisierung der benötigten Objekte)
4. Optional: Anlegen einer @BeforeAll Methode, welche vor den/dem eigentlichen Test/-s ausgeführt wird.
5. Deklaration und Implementierung eines oder mehrerer Tests

6. Optional: Anlegen einer @AfterAll Methode, welche nach dem/den eigentlichen Test/-s ausgeführt wird.
7. Überprüfen des/der Tests hinsichtlicher der logischen und syntaktischen Korrektheit
8. Ausführung des/der Tests
9. Überprüfung des/der Ergebnis/-se (Erfolgreich oder Fehlgeschlagen)
10. Optional: Anpassung der/des Tests, Fehlersuche & erneute Ausführung des/der Tests

Des Weiteren sind noch weitere Möglichkeiten und Schritte vorhanden, um einen Test zu erstellen oder auch anzupassen. Zum Beispiel „@BeforeEach“, welches einen Programmteil enthalten kann, der vor jedem einzelnen Test ausgeführt wird.

4.2 Verfügbare Tests

In der folgenden Liste sind sämtliche Testklassen ohne die jeweiligen Untertests aufgeführt:

4.2.1 Client

1. AnimationTest
2. ChatServiceTest
3. DeleteAccountEventTest
4. GameServiceTest
5. LobbyServiceTest
6. RegisterEMailPatternTest
7. RegistrationErrorEventTest
8. UnableToNotifyExceptionTest
9. UserServiceTest

4.2.2 Common

1. ActionCardTest
2. BuyCardTest
3. ChatExceptionMessageTest
4. ChatExceptionTest
5. CursedCardTest

6. GameManagementExceptionTest
7. GamePhaseExceptionTest
8. JsonCardParserTest
9. KickPlayerExceptionTest
10. LeaveLobbyExceptionTest
11. LobbyDTOTest
12. LobbyMessageSerializableTest
13. MoneyCardTest
14. NewChatMessageTest
15. NotEnoughMoneyExceptionTest
16. SecurityExceptionTest
17. SerializationTestHelper
18. SerializationTestHelperTest
19. SetMaxPlayerExceptionTest
20. UserDTOTest
21. UserMessageSerializableTest
22. ValueCardTest

4.2.3 Server

1. ActionCardTest
2. ActionCardWithResponseTest
3. ActualPointMessageTest
4. AuthenticationServiceTest
5. BotPlayerTest
6. BuyCardTest
7. ChatManagementTest
8. ChatServiceTest
9. CompositePhaseTest
10. DatabaseBasedUserStoreTest
11. DeckTest

12. DrawHandMessageTest
13. GameManagementTest
14. GameServiceTest
15. GiveUpTest
16. LobbyManagementTest
17. LobbyServiceTest
18. MainMemoryBasedUserStoreTest
19. PlaygroundTest
20. PoopTest
21. UserManagementExcptionTest
22. UserManagementTest
23. UserServiceTest
24. UserUpdateExceptionTest

4.3 Testabdeckung

Die Testabdeckung lässt sich mangels Erfahrungen, wie sie in anderen Gruppen ist, schlecht im richtigen Kontext einschätzen. Wir schätzen unsere Testabdeckung im unteren Mittelfeld ein, da wir in der Gruppe von Anfang an das Schreiben von Tests vernachlässigt haben. Erst zur Mitte des Softwareprojektes stieg die Motivation, Test zu schreiben, da es hier angefangen hat, öfter Probleme zu geben. Zum Ende des Softwareprojektes wurde die Idee der Test-Driven-Softwareentwicklung viel ernster genommen und dementsprechend sind die meisten Tests erst zum Ende des Softwareprojektes entstanden. Getestet wurde hauptsächlich das Servermodul, insbesondere die Sachen, die von der Gruppe zum Basisprojekt hinzugefügt worden sind. Es wurden kaum bis keine weiteren Tests für das Basisprojekt geschrieben. Die Packages Game und Lobby haben die beste Testabdeckung im kompletten Projekt, da dort die Spiellogik stattfindet. Die Testabdeckung im Client ist kaum vorhanden, da sich das grafische Testen als sehr schwierig herauskristallisiert hat und wir hier nur manuell per Hand getestet haben. Man kann abschließend sagen, dass wir, obwohl die meisten nebenbei Softwaretechnik I gehört haben, das Softwareprojekt nicht Test-Driven-gestützt durchgeführt haben. Hätten wir dies getan, wäre unsere Testabdeckung vermutlich höher.

4.4 Nicht verfügbare Tests

Grundsätzlich sind keine Tests zu Bildern, Icons, CSS, der Spieleanleitung und der Musik mangels Testmöglichkeiten und zu hoher Komplexibilität vorhanden. Des Weiteren wurden hauptsächlich Tests für den Server geschrieben, da der Client sehr wenig eigene Logik enthält und sich das Testen von grafischen JavaFX Elementen als zu komplex gestaltet. Wie im obigen

Kapitel Testabdeckung beschrieben, wurden keine weiteren Tests zu dem uns zur Verfügung gestellten Basisprojekt durch den Dozenten Marco Grawunder erstellt.

Die Funktionalität der jeweils einzelnen Komponenten wurde dennoch durch ausführliche Reviews, Beobachtung und Codekontrolle gewährleistet. Hinzu kommt das regelmäßge Testen der einzelnen Funktionen beziehungsweise Implementierungen im Spiel selbst. Dies wurde sowohl zwischendurch durch die einzelnen Teammitglieder, als auch durch das Team bei den Teamtreffen durchgeführt. Mögliche Bugs wurden anschließend via Jira Ticket aufgenommen und behoben.

4.5 Beschreibung - Codereview

Das gruppenweite Code-Review wurde in den Gruppensitzungen am 13.01.2020 sowie am 17.01.2020 durchgeführt. Da zu diesem Zeitpunkt noch kein Code in den Master-Branch gemergt worden war, wurde hierzu ein Pull Request vom Develop-Branch in den Master-Branch erstellt, welcher in den beiden Gruppensitzungen gemeinsam überprüft wurde. Änderungsvorschläge wurden dabei an der betroffenen Stelle als Kommentar hinzugefügt, es wurde ein dazugehöriger Task erstellt und die die Person, die diesen Task bearbeiten sollte, wurde markiert.

Beim Review wurde festgestellt, dass JavaDoc in einigen Klassen komplett gefehlt hat oder unvollständig war. Auch fiel auf, dass die Dokumentation teilweise auf Deutsch und teilweise auf Englisch verfasst wurde. Um dies zu beheben, wurden alle Klassen unter den Gruppenmitgliedern gleichmäßig aufgeteilt und jeder hatte die Aufgabe, die ihm zugeteilten Klassen hinsichtlich JavaDoc zu vervollständigen und anzupassen. Außerdem wurde festgestellt, dass Code von anderen Mitgliedern teilweise nicht genutzt wurde, obwohl es teilweise möglich bzw. angebracht wäre. Auf Grund dessen wurde der Hinweis ausgesprochen, sich den Code gründlich anzuschauen und besser auf programmweite Anpassungen zu achten. Hinsichtlich des Lobbymanagements wurde bemerkt, dass viele Methoden und Requests bzw. Messages noch mit dem Lobbynamen anstatt der Lobby-ID gearbeitet haben. Da die Umstellung etwas aufwendiger sein würde, wurde hierzu kein Task sondern ein neues Ticket (SWP2019B-199 - Sprint 5) erstellt.

Zu guter Letzt wurde festgestellt, dass noch zu wenige Tests vorhanden waren.

Kapitel 5

Arbeitsorganisation

Zu Beginn dieses Kapitels erfolgt eine Erläuterung der Rollenverteilung wie sie die Gruppe B vorgenommen hat. Anschließend werden die verschiedenen Rollen kurz erklärt.

Anschließend erfolgt eine Erläuterung der Arbeitsweise. Hierbei wird im Besonderen auf die verschiedenen Tools eingegangen, die wird für die Unterstützung unserer Arbeit verwendet haben, die z.B. Discord.

Danach wird auf die Präsentationen im Rahmen der Meilensteine eingegangen. Hierbei wurde unter anderem auch auf die Rückmeldung durch den Dozenten eingegangen.

Außerdem erfolgt in diesem Kapitel noch ein kurzer Rückblick über die Sprints.

Des Weiteren werden verwendete Frameworks, Bibliotheken und weitere Tools erläutert.

Zum Schluss folgt das Projekttagebuch, welches die einzelnen Tickets des Projektes auflistet. Dabei sind die Tickets nach Sprint gruppiert.

5.1 Rollenverteilung

Zu Beginn des Projektes erfolgte eine Aufteilung verschiedener Rollen auf die Gruppenmitglieder der Gruppe B. Hierfür haben wir in einer Gruppensitzung jeweils angeben können, welche Rollen wir gerne übernehmen würden. Anschließend haben wir die Rollen auf Grundlage dessen verteilt, wobei wir uns bei der Rolle des Konfliktmanagers darauf geeignigt haben, dass die Rolle von einer weiblichen und einer männlichen Person übernommen werden sollte.

Zum Ende des Projektes haben uns leider zwei Mitglieder die Gruppe verlassen, weswegen wir die Rollen von Marvin Thate (Patternbeauftrager) und Alexander Linke (Jira und Projektplanung) neu aufteilen mussten. Für die Rolle von Marvin stellte dies kein Problem dar, da er seine Rolle zusammen mit Paulina übernommen hatte. Alexander Linkes Rolle übernahm die Scrum-Masterin Fenja Bauer, da diese mit Alexander zusammen die Projektplanung durchgeführt hatte und einen groben Überblick hatte.

Tabelle 5.1: Rollenverteilung

Rolle	**Ausgeführt durch**
Codequalitätsbeauftrager	Darian Alves
Git/Bitbucket	Ferit Askin
Jira und Projektplanung	Fenja Bauer
Scrum-Master	Fenja Bauer
Reviewbeauftrager	Julia Debkowski
Konfliktmanagement	Mahmoud Haschem & Anna Hiemenz
Dokumentations- und Backupbeauftrager	Rike Hochheiden (Helfer: Timo Siems)
Patternbeauftrager	Paulina Kowalska
Entwicklungsumgebung & Maven	Keno Oelrichs Garcia
JavaFX & ControlFX	Devin Schlüter
Testbeauftrager	Timo Siems
DB-Zugriff	Keno Schwedler

5.1.1 Codequalitätsbeauftrager

Bei dem Codequalitätsbeauftragten geht es hauptsächlich um die Wahrung der Lesbarkeit des Codes. Dabei muss darauf geachtet werden, dass grundlegende Codekonventionen eingehalten und Code-Smells vermieden werden. Wenn diese nicht eingehalten wurdern, musste darauf hingewiesen werden. Es musste dann auch am Anfang dafür gesorgt werden, dass es weitere Codekonventionen gibt als die Grundlegenden, die in den bereits bekannten Vorlesungen besprochen wurden. Darüber hinaus muss darauf geachtet werden, dass die Java-Doc Konventionen eingehalten werden.

5.1.2 Git/Bitbucket

Die erste Aufgabe des Git-Beauftragten lag darin, die Präsentation zu erstellen, um den anderen Projektteilnehmern Git und den Git-Arbeitsablauf zu erklären. Dies wurde im Rahmen einer PowerPoint Präsentation gestaltet, welche den Teilnehmern auch nachträglich bei Confluence zugänglich gemacht worden ist. Zu Anfang gab es öfter noch Probleme im Workflow der Teilnehmer, weshalb hier öfter ausgeholfen werden musste. Im Laufe der Zeit wurden die Probleme weniger, sodass nur noch vereinzelt geholfen werden musste.

5.1.3 Jira und Projektplanung

Die Rolle des Jira- und Projektplanungsbeauftragten musste zum Ende des Projektes neu besetzt werden, da von der Gruppe B zwei Mitglieder ausgewiesen wurden. Dadurch wurde die Scrum-Masterin zusätzlich die Aufgabe der Projektplanung zugeteilt. Zu Beginn des Projektes hatte sich die Gruppe B dazu entschieden, die beiden Aufgaben zum Teil zusammenzulegen, weswegen es ein Leichtes war, die Aufgabe der Scrum-Masterin mit der der Projektplanung und der Jira-Organisation zu vereinen.

Die Einarbeitung in die Plattform und deren Umgebung nahm zu Beginn viel Zeit in Anspruch, da man sich zuerst mit den Aufgaben vertraut machen musste. Dies wurde im Laufe der Zeit besser. Die Vorgänge, die Zeiterfassung und die korrekten Formulierungen der Taskbeschreibungen wurden auch in der Gruppe verinnerlicht. Dabei hat sich die Gruppe B dazu entschieden, einen Task nicht als solchen zu formulieren, sondern die User Story als solchen zu verstehen.

Bei dieser Rolle wurde darauf geachtet, dass bei aufgetretenen Problemen zugehörige User Stories erstellt wurden und bei den User Stories eindeutige Beschreibungen vorhanden sind. Zusätzlich wurde die Zeiterfassung stets im Blick behalten. Dies war wichtig, da sichergestellt werden sollte, dass sämtliche Gruppenmitglieder in etwa den selben Zeitaufwand für das Projekt aufwenden. Dabei wurden die Aufgaben soweit verteilt, dass die Scrum-Masterin direkt sehen konnte, wie weit die Gruppenmitglieder mit ihren Aufgaben waren. Dabei wurden die verschiedenen Gruppenmitglieder auch privat angesprochen, um zu erfahren wie weit sie gekommen sind und ob es Probleme bei der Bearbeitung gab.

Ein Problem war bis zum Ende die genaue Projektplanung zu organisieren. Viele Aufgaben, die in der Modellierung noch nicht vorgesehen waren, wurden in früheren Sprints bearbeitet. Hinzu kam, dass zu Beginn des Projektes die Projektplanung von der Gruppe B nicht genau genommen wurde. Es wurde viel Zeit von der Scrum-Masterin und dem Projektplaner investiert, um ein Entwurfsmodell vorstellen zu können, welches erst spät im Projekt umgesetzt wurde. Da die Projektplanung unter den beiden Rollen nicht gut funktionierte, änderte die Gruppe B das Vorgehen der Projektplanung und organisierte sich im Plenum. Dadurch wurden alle Gruppenmitglieder einbezogen und konnten an der Planung teilnehmen. Somit wurde später auch während der Implementierung und des Sprint das weitere Vorgehen angesprochen und modelliert.

5.1.4 Scrum-Master

Zu Beginn des Projektes musste sich die Scrum-Masterin erst einmal ausführlich mit dem Thema Scrum beschäftigen. Dieses Modell der Projektumsetzung war dieser noch nicht vertraut, weshalb die Idee erst einmal verinnerlicht werden musste. Dies dauerte seine Zeit und die Gegebenheiten von Sprint, dem Backlog, der Sprintplanung waren anfänglich schwierig umzusetzen.

Nach einer gewissen Zeit wurde das Projektmodell immer mehr verinnerlicht und somit waren die Aufgaben auch einleuchtender. Wie bei Jira und Projektplanung schon beschrieben wurde die Projektplanung im Laufe der Zeit von der Scrum-Masterin auf das Team abgestimmt. Die Kommunikation wurde hierdurch vorangetrieben und die Motivation der Gruppe B wurde stetig verbessert. Somit wurde der Prozess im Laufe der Zeit immer weiter optimiert und die letzteren Sprints wurden immer erfolgreicher.

In der Sprintplanung wurden die Ideen und Ziele gesammelt, welche User Stories umgesetzt werden sollten und wie weit wir mit dem Projekt kommen wollen. Bei der Verteilung der Aufgaben war für die Gruppe B das System First Come First Serve am besten geeignet. Hierüber haben die Gruppenmitglieder sich schnell und eigens die Aufgaben zugeteilt und diese bearbeitet. Somit musste niemand auf die Bestätigung der Gruppe warten und konnte selbstständig weiter arbeiten. Hinzu kam, dass die Gruppe B während des Sprints erlaubt hat, dass Gruppenmitglieder zusätzliche Aufgaben abarbeiten dürfen, wenn sie mit ihrer zugewiesenen Aufgabe fertig waren. Dies führte allerdings zu einem effizienteren Prozess, da dadurch niemand auf das Okay der anderen warten musste.

Zusätzlich hat die Gruppe B sich dazu entschieden, die Aufgabe des Daily-Scrum dem Sitzungsleiter zu überlassen. Das Daily-Scrum wurde an den beiden Sitzungstagen Montag und Freitag abgehalten. Dabei wurde der aktuelle Stand abgefragt und welche Aufgabe derjenige zu dem Zeitpunkt bearbeitet hat, ob es Probleme gibt und dabei wurde von der Scrum-Masterin auch Hilfestellung angeboten. Zusätzlich zum Daily-Scrum hat die Scrum-Masterin die Gruppenmitglieder privat angeschrieben und gefragt, ob Probleme auftreten oder Hilfe benötigt wird.

Des Weiteren hat die Gruppe B nach jedem Sprint, unter der Regie der Scrum-Masterin, die Retrospektive abgehalten, welche viele Lösungsmöglichkeiten beinhalteten den Scrum-Prozess effizienter zu gestalten. Anfänglich wurden die Retrospektiven in der Starfish-Methode abgehalten, doch durch Corona, dann über die Atlassian Tools, welche den Vorteil hatten, dass die Einträge anonym geschahen und dadurch wieder mehr Diskussion veranlasst wurde. Die Retrospektiven wurden in der Gruppe besprochen und im nächsten Sprint durchgeführt. Zusätzlich wurde abgefragt, ob die Gruppe mit dem jeweiligen Sprint zufrieden war. Dabei fiel auf, dass zu Beginn des Projektes die Aufgaben sehr spät bearbeitet wurden und später der Ablauf effizienter gestaltet wurde.

5.1.5 Reviewbeauftrager

5.1.5.1 Pull Requests

Bevor ein Branch in den Develop-Branch gemergt werden kann, muss dieser von mindestens zwei Leuten genehmigt werden. Zu Beginn des Projekts trat dabei zunächst das „Problem“auf, dass immer die gleichen Leute als Prüfer eingetragen wurden. Dies konnte aber mit einer kurzen Ansprache gelöst werden, sodass fortan darauf geachtet wurde, die Reviews gleichmäßig zu verteilen. Auch wurde bemerkt, dass Prüfer Pull Requests teilweise zu schnell genehmigt bzw. nicht richtig geprüft haben, sodass fehlerhafter Code in den Develop gelangt ist. Das konnte aber mit dem Hinweis an die Gruppe, bei den Reviews gründlich zu sein, behoben werden.

5.1.5.2 Gruppenweites Code Review

Weiterhin war es die Aufgabe des Reviewbeauftragten, das gruppenweite Code Review zu leiten, welches mindestens einmal durchgeführt werden musste. Ein Bericht dazu ist unter 4.5 zu finden.

5.1.6 Konfliktmanagement

Als Konfliktmanager war es unsere Aufgabe, sich im Vorfeld mit möglichen Konfliktursachen zu beschäftigen und Ansätze sowie Methoden herauszusuchen, wie mit solchen Konflikten umgegangen werden kann.

In unserer Gruppe trat nur der Konflikt auf, dass einige sich zu wenig beteiligt und so nicht genug mitgearbeitet haben. Dieses wurde dann in den Gruppensitzungen und auch teilweise beim Sprint-Review sachlich angesprochen und das führte auch dazu, dass fast alle Mitglieder wieder mehr mitgemacht haben.

5.1.7 Dokumentations- und Backupbeauftrager

5.1.7.1 Confluence

Zu Beginn des Projektes beschränkten sich die Hauptaufgaben des Dokumentations- und Backup-Beauftragen darauf, dafür Sorge zu leisten, dass Confluence übersichtlich gestaltet wurde. Hierfür wurde eine Vorlage als auch eine Übersichtsseite für Protokolle auf Confluence erstellt. Hierbei hat der Beauftragte die Übersichtsseite regelmäßig kontrolliert. Sofern die Verlinkung der Protokolle nicht korrekt war, wurde diese ggf. vom Beauftragten korrigiert oder ergänzt.

Neben den Protokollen der Sitzung wurde Confluence ebenfalls für die Einzelvorträge, die am Anfang des Projektes gehalten wurden, genutzt. Hierbei wurden verschiedene Anleitungsartikel und Präsentationen abgelegt, welche für die Vorträge erstellt wurden.

Um die erstellten Artikel übersichtlicher zu gestalten, besitzt der entsprechende Reiter eine Übersicht in Form einer Tabelle, mit den Themen, den Namen der Präsentierenden, dem Datum und der optionalen Verlinkung der erstellten Artikel.

Während des Projektes verlor Confluence jedoch an Bedeutung für die Gruppe, da für die Kommunikation Discord immer mehr an Bedeutung gewann und viele Informationen darüber der Gruppe zur Verfügung gestellt wurden.

5.1.7.2 Ablage und Erstellung der Dokumentation für die bearbeiteten Aufgaben

Zur Erstellung der Dokumentation wurde der Gruppe eine LaTeX-Vorlage sowie ein „How to LaTeX“ bereit gestellt. Die erstellten Dateien sollten von der Gruppe in einer Cloud der Universität Oldenburg hochgeladen und dort im jeweiligen Sprint-Ordner abgelegt werden, damit bei Fragen zu erstellten Methoden auf diese Dokumentation zugegriffen werden kann.

Da jedoch die Bearbeitung der Aufgaben und das fertigstellen des Prototypen für die Gruppe eine wichtigere Rolle hatte, wurde die Dokumentation zunächst teilweise etwas vernachlässigt.

5.1.7.3 Erstellung der Dokumentation für die Abgabe

Nach dem erfolgreichen Präsentieren des zweiten Meilensteins mit einem Prototypen, gewann das Erstellen der Dokumentation für die Gruppe an Bedeutung und die Gruppe einigte sich darauf mit der Dokumentation zu beginnen.

Die Gruppe beschloss Overleaf zu nutzen, da es hiermit möglich ist, gleichzeitig mit mehreren Mitgliedern an der Dokumentation zu arbeiten. Außerdem wurde für die Erstellung der Dokumentation eine Übersicht erstellt, welche die Aufgaben für die Dokumentation auflistet und angibt wer für diese Aufgaben verantwortlich ist. Die Verteilung erfolgte über Jira. Hierbei sollten die Aufgaben möglichst gleichmäßig verteilt werden, damit jeder einen Beitrag zu der Dokumentation beitragen konnte.

5.1.8 Patternbeauftrager

Als Patternbeauftragter galt es, die Gruppe über unterschiedliche Pattern zu informieren. Hierzu wurden zu Beginn der Gruppe das Prinzip von Pattern und drei Pattern: Model View Presenter, Observer Pattern und Command Pattern vorgestellt. Überwiegend wurde im Projekt das Model View Presenter angewandt. Außerdem war der Patternbeauftragte für die Wahrung der Struktur im Code zuständig.

5.1.9 Entwicklungsumgebung und Maven

Der Entwicklungsumgebungs und Maven-Beauftragte fungiert während des Projektes hauptsächlich als Ansprechpartner für Fragen und Problemlösungen rund um die verwendete IDE. In diesem Falle war dies IntelliJ. Außerdem kümmerte sich dieser um die Pflege der in dem Projekt verwendeten Frameworks und die Maven-Konfiguration.

5.1.10 JavaFX & ControlFX

Der JavaFX & ControlFX-Beauftragte hatte zunächst die Aufgabe, sich über diese Themen zu informieren und bei Fragen bezüglich dem Umgang mit den fxml-Dateien und ihrer Einbindung in den Code behilflich zu sein.

5.1.10.1 Hauptmenü

Unser Umgang mit JavaFX bezog sich am Anfang größenteils darauf, einen funktionierenden Chat für das Spiel und die Lobbyerstellung möglich zu machen. Bei der Erstellung des Chat traten anfangs Fehler auf, wie dass geschriebene Nachrichten nicht im Chat angezeigt wurden oder andere Spieler die Nachrichten nicht bekamen. Ansonsten wurde mit Listen gearbeitet, um die erstellten Lobbys und die Präsenz der Spieler, die sich gerade im Spiel befinden, darzustellen. Ein Screenshot zum Hauptmenü ist bereits in Abschnitt 3.2.1 gezeigt worden.

5.1.10.2 GameViewWIP

Die GameView stellte den größten Bereich dar, wo FX-Kenntnisse benötigt wurden. Es wurde viel über das Layout des Spieles diskutiert sowie darüber, welche FX-Mittel für welche Funktionen einsetzt werden sollen. Es wurde zum Beispiel entschieden, die Karten durch ImageViews darzustellen und das Spielfeld durch mehrere Panes in verschiedene Bereiche zu gliedern. Dies hat sich im Verlauf des Projektes insofern geändert, dass stattdessen mehr mit Regionen gearbeitet wurde, weil sie das Platzieren der Karten einfacher gemacht hatten. Eines der komplizierteren Probleme, welches beim Arbeiten mit der GameView enstanden ist, war die korrekte Speicherung der MouseEvents und die problemlose Interaktion mit den ImageViews, ohne dass andere Elemente davon betroffen sind. Bilder der GameView sieht man in Abschnitt 3.2.3.

5.1.11 Testbeauftrager

5.1.11.1 Grundsätzliches

Als Testbeauftragter hat man die Aufgabe dafür zu Sorge zu tragen, dass die zu entwickelnde Software ausreichend und im angemessenen Umfang getestet wird. Sie inkludiert die Teilaufgaben der Toolauswahl, Toolbeschreibung, der Testerstellungsüberwachung und der allgemeinen Testergebnisüberwachung. Innerhalb des Softwareprojektes hat der Testbeauftragte dafür zu sorgen, dass Tests ordnungsgemäß und rechtzeitig erstellt werden und das diese auch den richtigen Testfall abfangen und dass die einzelnen Projektmitglieder über die Testmöglichkeiten aufgeklärt sind.

5.1.11.2 Tools

Zu Anfang des Softwareprojektes wurden die beiden Testools JUnit 5 & Mocktio 3 vorgestellt. Innerhalb des Softwareprojektes kam das anschließend JUnit 5, welches standardgemäß in der verwendeten Entwicklungsumgebung IntelliJ inkludiert ist, zum Einsatz.

Damit sämtliche Mitglieder des Teams die Einbindung und Konfiguration von JUnit 5 & Mocktio 3 problemlos hinbekommen, wurde eine Kurzanleitung zur Einbindung der beiden Tools, sowie weiterführende Informationen in einer Präsentation und einer Beispiel Java-Datei auf Confluence bereit gestellt (hinterlegt in einem extra Ordner „Zusätzlich entstandene Dokumente“ → „Tools Test“ → „Tools JUnit Mockito“, „Junit Mockito“ und „IdeaProjects“).

5.1.11.3 Steuerung

Die Steuerung durch den Testbeauftragten erfolgt durch die Einweisung der einzelnen Teammitglieder und der Bereitstellung von Einführungs- beziehungsweise Hilfsmaterial. Durch gezielte Überwachung der aufkommenden Pull Requests und des Codes kann innerhalb der Arbeitsphase die Umsetzung der Tests überwacht und gegebenenfall korrigiert werden.

5.1.11.4 Umsetzung

Zum Start des Softwareprojektes wurden sämtliche Projektteammitglieder durch die Präsentation über die beiden verfügbaren Testtools informiert. Während des Projektes wurde das Team öfter darauf hingewiesen, Tests zu erstellen. Nach anfänglichen Schwierigkeiten gelang es die Anzahl an Tests kontinuierlich zu steigern. Hierbei waren die Fähigkeiten der Einzelnen und die Ticketart zu beachten. Denn nicht für jedes Ticket (z. B. beim Refactoring) waren Tests sinnvoll. Eine schlussendliche Abdeckung der Tests ist im Kapitel Testdokumentation zu finden.

5.1.12 Datenbankintegration

Zu Beginn des Projektes wurden die Gruppenmitglieder mit Hilfe einer Präsentation über die Datenbankintegration mit JDBC und grundlegende Befehle in Java informiert, damit die Gruppe

einen Überblick bekommen hat, was und wieso gewisse Codezeilen geschrieben wurden sind. Dies diente auch dazu, dass die einzelnen Gruppenmitglieder diesen Code nachvollziehen konnten. Ebenso wurde gezeigt, wie eine Verbindung zu einer Datenbank hergestellt werden kann und wie simple Abfragen durchgeführt werden können.

Zu den Aufgaben des Datenbankbeauftragten gehörten außerdem die Erstellung einer passenden SQL-basierten Datenbank, welche die Userdaten der Spieler dauerhaft speichern soll. Dies geschah zum Ende des Projektes, da der Arbeitsumfang dieser Aufgabe als nicht sonderlich groß eingeschätzt wurde. Des Weiteren sollte der bereits vorhandenen Userstore durch den Datenbank-Userstore ersetzt werden. Diese Aufgabe wurde bereits früh bearbeitet, aber wieder verworfen, da der Branch in Git gelöscht wurde. Der Fortschritt konnte jedoch zu einem späteren Zeitpunkt durch ein Backup wieder hergestellt und in einen neuen Branch eingepflegt werden.

Um die Integration zu erleichten, besteht der DatabaseBasedUserStore aus den gleichen Methoden wie der bereits gegebene MainMemoryBasedUserStore. Die Verbindung des Servermoduls mit der Datenbank wurde mit einem JDBC Treiber realisiert. Zusätzlich dazu war der Datenbankbeauftragte der Ansprechspartner, bei allgemeinen Fragen bezüglich Verbindungsproblemen mit der Datenbank oder der Integration mit dieser in IntelliJ o. Ä.

5.2 Arbeitsweise

Während des Softwareprojektes wurden zum Arbeitsorganisation bzw. Arbeitsunterstüzung mehrere gängige Tools verwendet. Hierunter fallen:

1. Discord
2. Confluence
3. Overleaf
4. Jira
5. Bitbucket

Im Folgenden wird auf die einzelnen Tools und deren Verwendung eingegangen.

5.2.1 Discord

Das Kommunikationsmittel Discord wurde innerhalb des Softwareprojektes zur Kommunikation sowie für den Datenaustausch benutzt. Der Server wurde dabei von dem Teammitglied Keno O. aufgesetzt und verwaltet. Es wurden mehrere Channels verwendet, die jeweils für unterschiedliche Zwecke benutzt wurden. Die wichtigsten davon waren:

1. Allgemein → Text - Allgemeiner Informationsaustausch
2. Organisatorisches → Text - Informationsaustausch zwecks Eigenorganisation (Gruppentreffen, ...)

3. Umfragen → Text - Channel für Umfragen
4. Neue Tickts → Text - Neue Tickets werden von einem Bot automatisiert aufgezeigt.
5. Pull-Requests → Text - Pull-Requests werden von einem Bot automatisiert aufgezeigt.
6. Pull-Request Kommentare → Text - Pull-Request Kommentare werden von einem Bot automatisiert aufgezeigt.
7. Meeting-Room → Audio - Meeting Room für die Gruppentreffen
8. Coding-Lab → Mix - Allgemeiner Channel während des Codes zum Informationsaustausch
9. Client-Lab → Mix - Allgemeiner Channel des Client Teams zum Informationsaustausch inklusive Text Fragechannel für das Server Team
10. Server-Lab → Mix - Allgemeiner Channel des Server Teams zum Informationsaustausch inklusive Text Fragechannel für das Client Team

5.2.1.1 Discord - Bots

Auf dem Discordserver wurden mehrere Bots implementiert, um automatisiert an Informationen zu kommen und den Datenaustausch zu ermöglichen. Diese wurden von Keno O. implementiert. Die Bots haben dabei unter anderem automatisiert Pull Requests, Pull Request Kommentare und neue Tickets aufgezeigt.

5.2.1.2 Discord - Beispielbilder

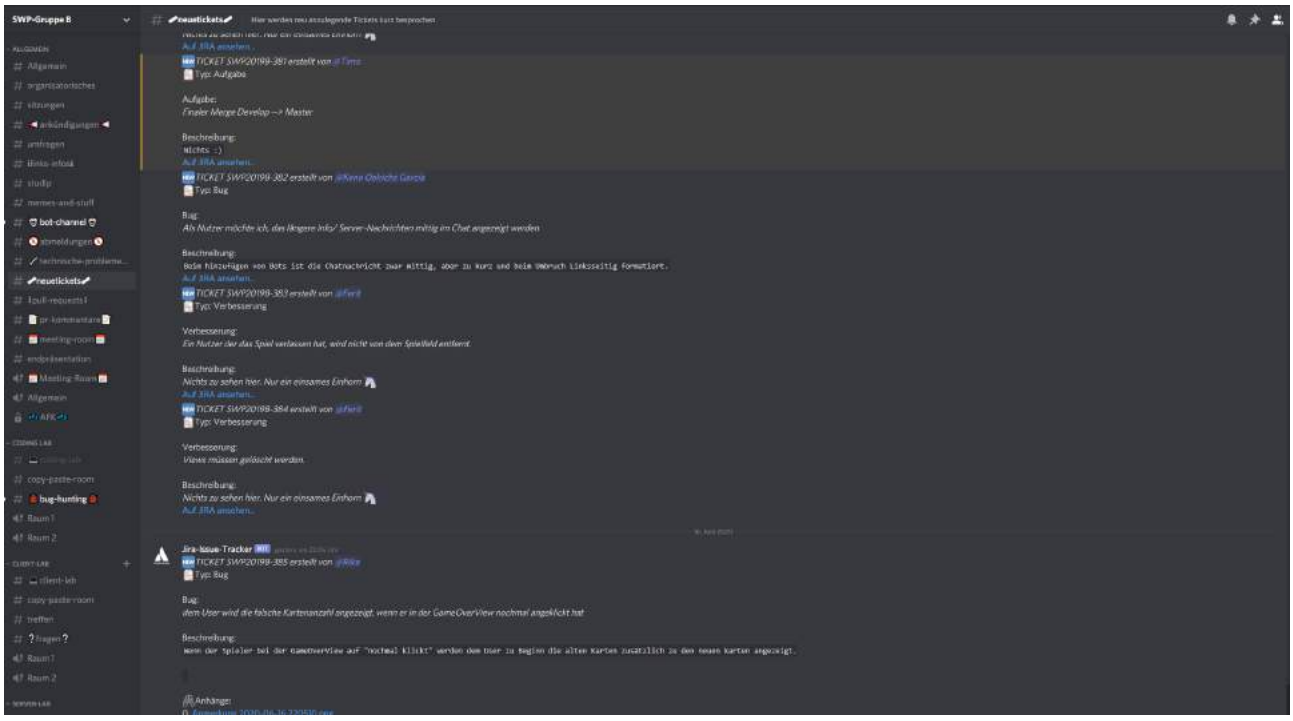

Abbildung 5.1: Discord - Beispielbild 1

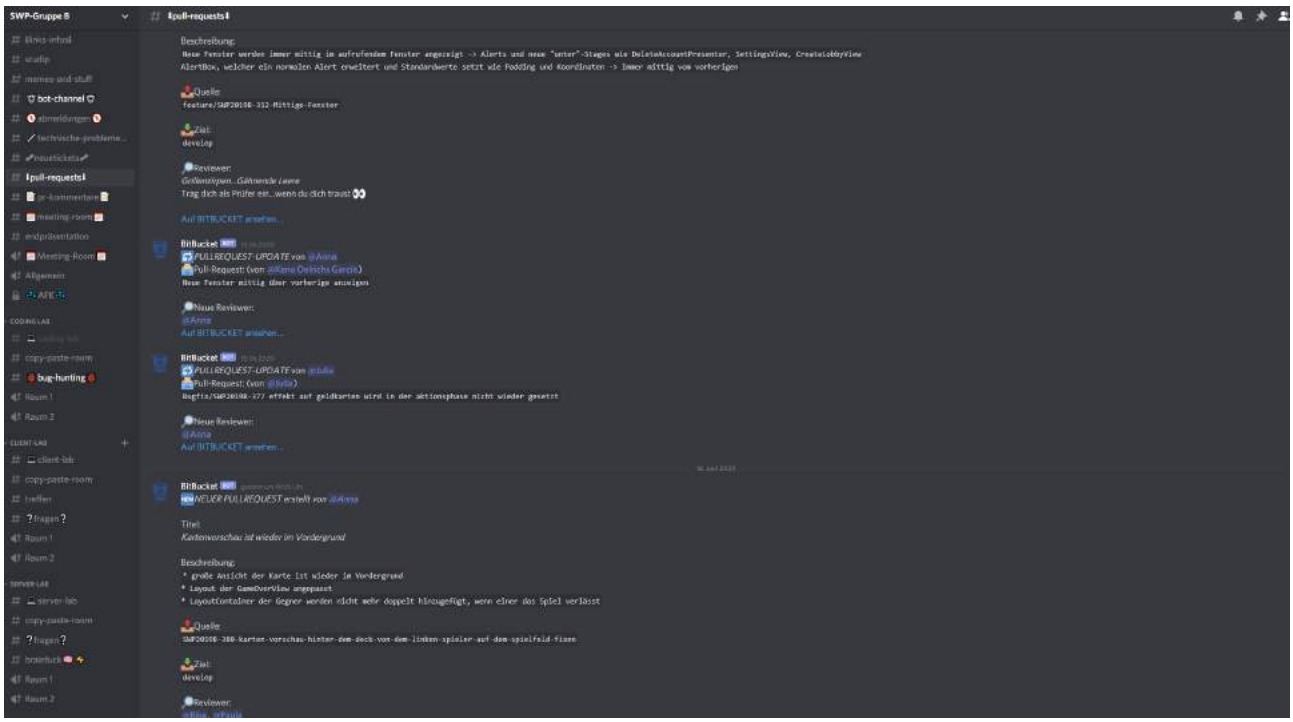

Abbildung 5.2: Discord - Beispielbild 2

5.2.2 Confluence

Confluence wurde uns zu Beginn des Projekts als zu nutzende Plattform an die Hand gegeben. Dieses Tool hat uns vor allem zu Beginn bei der Strukturierung des Projektes und der Zusammentragung von Informationen geholfen. Die beständigsten Seiten sind:

1. Einzelvorträge → Diese Seite beinhaltet eine Tabelle. In dieser werden der Rolleninhaber, dessen Themen und dazugehörige Links (zum Beispiel zur jeweiligen Präsentation) dargestellt, sodass man auf die Infos jederzeit zugreifen kann.

2. Mitglieder → Die Seite beinhaltet eine Übersicht zu den Mitgliedern unserer Gruppe. Zu diesen werden dann Mail, die Rolle und der Name auf Discord angezeigt. Anhand dieser Seite wurden wöchentlich die Rollen Moderator und Protokollant weiter gereicht.

3. Protokolle → Die wöchentliche Protokolle sollten auf dieser Seite hinzugefügt werden, sodass auch hier ein zeitunbhängiger Zugriff möglich ist.

4. Home → Auf dieser Seite ist das Schwarze Brett zu finden. Hier wurden vorläufige Tagesordnungspunkte, der Moderator und der Protokollant festgelegt. Zudem werden unter dem Punkt Inhalt alle verfügbaren Seiten verlinkt.

5.2.3 Overleaf

Bei der Webseite Overleaf handelt es sich um einen Online- LaTeX-Editor, welcher eine gemeinsame Bearbeitung eines LaTeX-Dokumentes ermöglicht. Overleaf wurde von uns dazu verwendet die Dokumentation asynchron, parallel und unabhängig voneinander schreiben zu können. Die LaTeX-Integration ermöglichte es, einfach mehrere LaTeX-Dokumente unkompliziert zusammenzufügen

und bei korrekter LaTeX-Syntax die wesentliche Formatierung ein Stück vernachlässigen zu können. Overleaf wurde dabei bis zum Ende genutzt und hat das Verfassen der Dokumentation erheblich vereinfacht.

5.2.3.1 Overleaf - Beispielbild

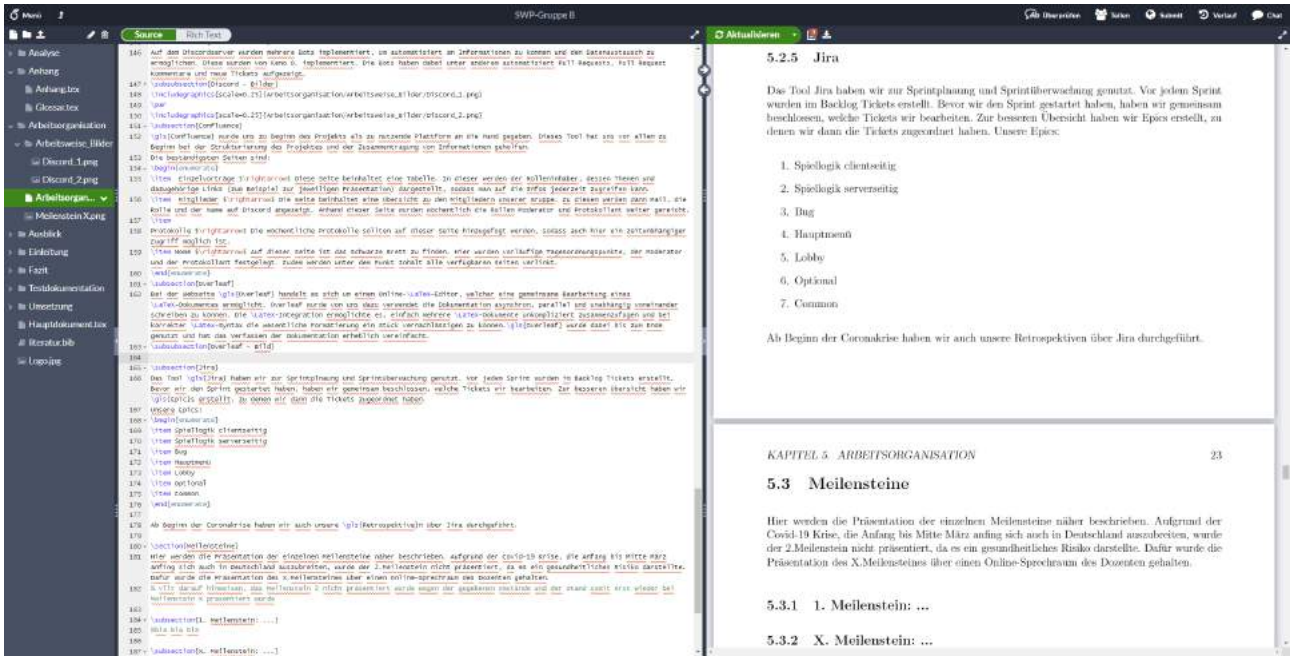

Abbildung 5.3: Overleaf - Beispielbild 1

5.2.4 Jira

Das Tool Jira haben wir zur Sprintplanung und Sprintüberwachung genutzt. Vor jedem Sprint wurden im Backlog Tickets erstellt. Bevor wir den Sprint gestartet haben, haben wir gemeinsam beschlossen, welche Tickets wir bearbeiten. Zur besseren Übersicht haben wir Epics erstellt, zu denen wir dann die Tickets zugeordnet haben. Unsere Epics:

1. Spiellogik clientseitig
2. Spiellogik serverseitig
3. Bug
4. Hauptmenü
5. Lobby
6. Optional
7. Common

Ab Beginn der Coronakrise haben wir auch unsere Retrospektiven über Jira durchgeführt.

5.2.4.1 Jira - Beispielbild

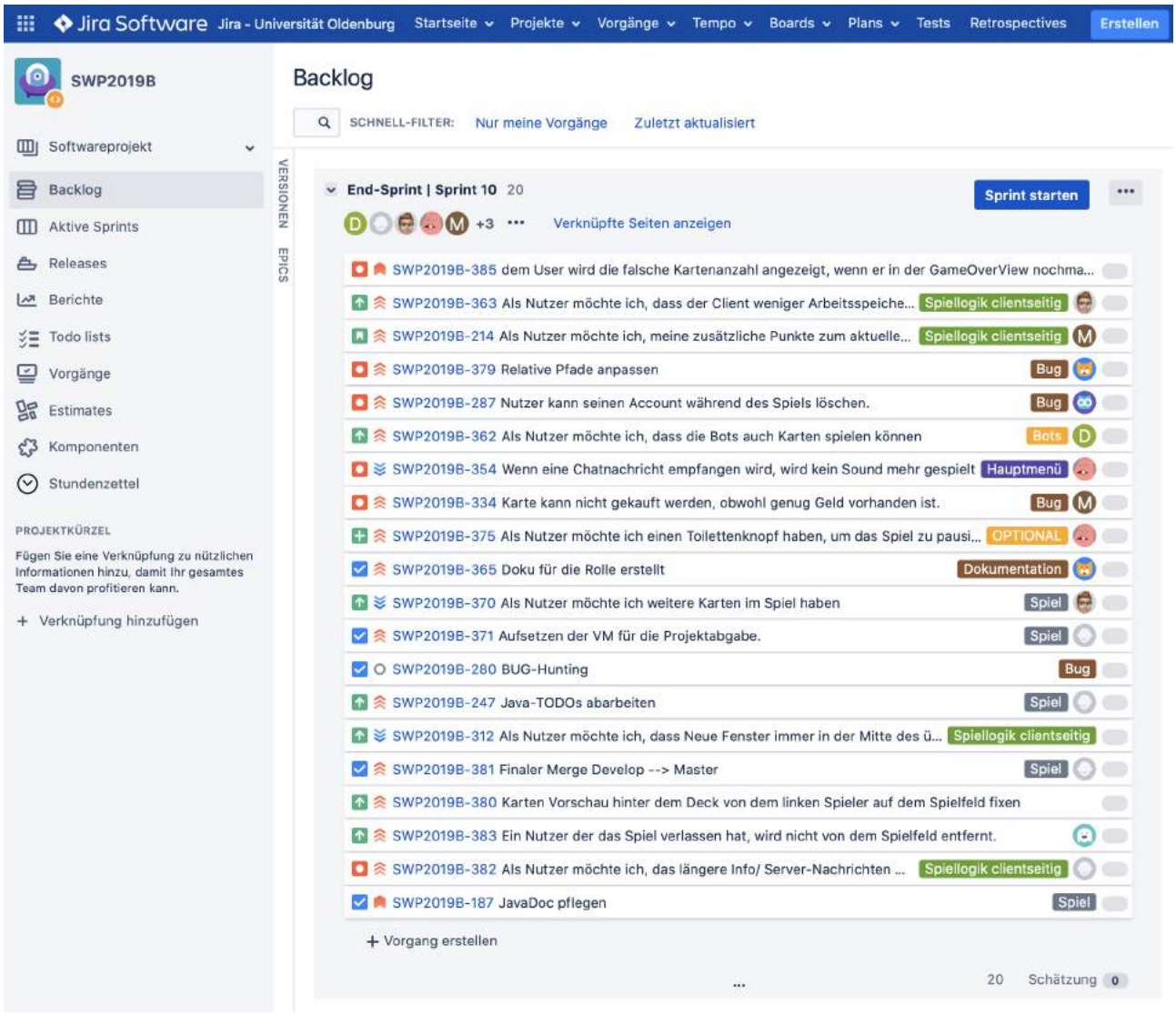

Abbildung 5.4: Jira - Beispielbild 1

5.2.5 Bitbucket

Bitbucket wurde wurde als die Plattform zum Stellen, Verwalten und Organisieren von Pull Requests und Branches innerhalb des Softwareprojektes genutzt. Das Team hat effektiv Pull Requests gestellt und kontrolliert. In regelmäßigen Abständen wurden über Bitbucket veraltete Branches gelöscht. Auch wurde Bitbucket von den Teammitgliedern genutzt, um die eigenen Statistiken (Anzahl Codezeilen, ...) einzusehen. Letztere Funktion wurde des Weiteren von unserem Begleittutor und dem Dozenten genutzt, um den Fortschritt des Einzelnen beziehungsweise der Gruppe pro Zeitraum überwachen und bewerten zu können.

5.2.5.1 Bitbucket - Beispielbild

Abbildung 5.5: Bitbucket - Beispielbild 1

5.3 Meilensteine

Hier werden die Präsentation der einzelnen Meilensteine näher beschrieben. Aufgrund der Covid-19 Krise, die Anfang bis Mitte März anfing, sich auch in Deutschland auszubreiten, wurde der 2. Meilenstein nicht präsentiert, da es ein gesundheitliches Risiko darstellte. Dafür wurde die Präsentation des X.Meilensteines über einen Online-Sprechraum des Dozenten gehalten.

5.3.1 2. Meilenstein: Hauptmenü, Chat, Spiel-Ansätze

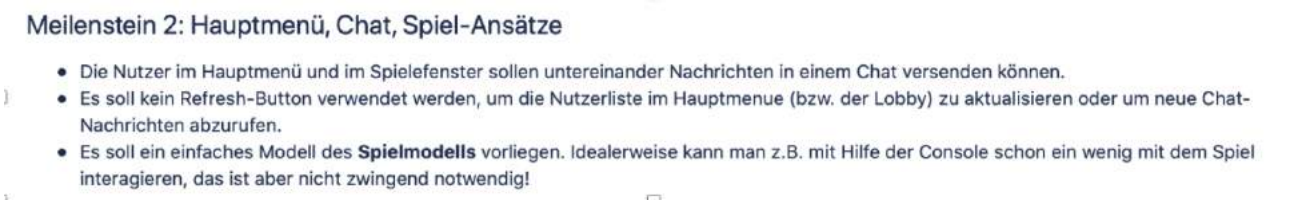

Abbildung 5.6: 2. Meilenstein

Die Präsentation zum 2. Meilenstein wurde am 16.12.2019 von Ferit Askin, Julia Debkowski und Paulina Kowalska gehalten.

Anforderungen wie der Chat und das Aktualisieren der Lobby waren bereits vollständig implementiert. Das einfache Spielmodell, welches in Form eines Klassendiagramms präsentiert wurde, war nicht zufriedenstellend. Hier wurde empfohlen dieses Modell noch mal zu überarbeiten.

Zudem wurde darauf hingewiesen, dass wir uns mehr auf das Spiel als auf kleine Details (z. B. Animationen im Anmeldebildschirm) konzentrieren sollten. Zur Präsentation an sich wurde angemerkt, dass viele der gezeigten Sequenzdiagramme den Inhalt redundant wiedergegeben haben.

Insgesamt verlief die Präsentation zufriedenstellend und der Dozent gab hilfreiche Hinweise für den weiteren Verlauf.

5.3.2 X. Meilenstein: „Prototyp“

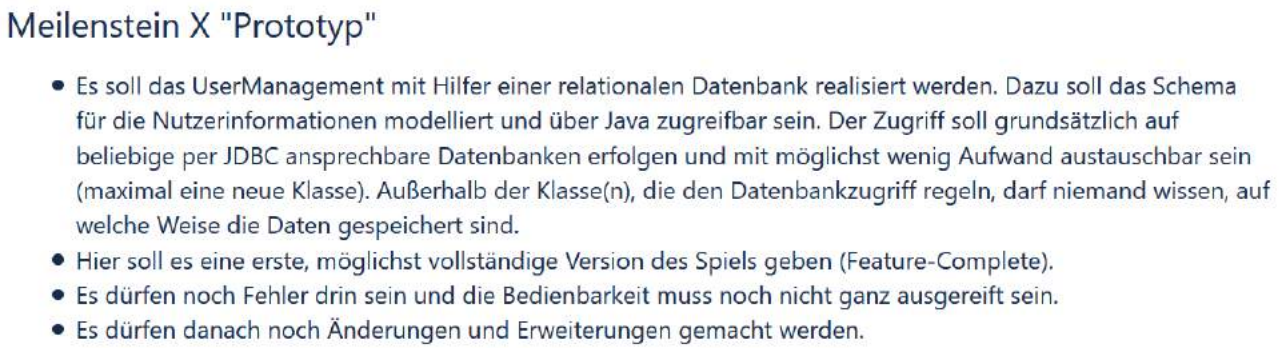

Abbildung 5.7: X. Meilenstein

Diese Präsentation wurde von Keno Oelrichs García, Fenja Bauer und Devin Schlüter am 04.06.2020 um 12.00 Uhr gehalten.

Wie in der Vorgabe des Meilensteines definiert, wurde das Spiel zum großen Teil fertig implementiert. Das bedeutet, dass der Spieler bereits zwölf Aktionskarten ausspielen kann und ihre Aktionen ausgeführt werden. Dabei wird im Spiel auch überprüft, wann ein Spieler gewonnen hat, und das Spiel wird dann beendet. Somit war es möglich, ein gesamtes Spiel normal durchzuspielen. In der Präsentation wurde darauf eingegangen, wie sich unser Projekt seit unserer letzten Präsentation positiv entwickelt hat und welche Bugs noch bestehen oder welche Verbesserungen noch implementiert werden müssen. Es wurde erwähnt, dass es möglich ist, ein Spiel mit einem Bot und Spielern zu spielen. Der Dozent hat ein kurzes Feedback gegeben, mit dem Hinweis, dass man in der Endpräsentation mehr Konzepte zeigen kann und man sich kurzfassen sollte.

5.3.3 Finale Präsentation

Die finale Präsentation wurde von der Gruppe B am 02.07.2020 gehalten.

In der finalen Präsentation hat die Gruppe B das von ihr implementierte Spiel vorgestellt. Dabei ist sie auf verschiedene Aspekte eingegangen. Zum einen wurden umgesetzte Ideen von der Gruppe vorgestellt, die sie sich selber überlegt haben. Unter anderem wurde hierbei der „Kloknopf“ vorgestellt, welchen die Gruppe für eine kurze Pause implementiert hat. Außerdem wurde auf die Architektur, die Zusammenhänge und verschiedene Software-Design-Entscheidungen eingegangen, welche im Rahmen des Projektes vorgenommen wurden. Hierbei ist die Gruppe auch auf einige Probleme eingegangen, die durch Software-Design-Entscheidungen entstanden sind. Dabei wurde hervorgehoben, wie die Gruppe das besser hätte machen können um den Lerneffekt aus diesen Fehlern hervorzuheben.

Ein weiterer Aspekt in der Präsentation waren die Technologieentscheidungen und verschiedene Einschränkungen, Anpassungen und Änderungen die beachtet bzw. im Laufe des Projektes vorgenommen worden sind. Des Weiteren wurde auch auf einige Probleme eingegangen, die während des Projektes auftraten. Im Allgemeinen hat es sich bei den meistens Problemen um Probleme gehandelt, die im Laufe des Projektes gelöst werden konnten. Lediglich die Teamgröße, mit zum Ende hin 12 Personen, das Buchern von Zeiten, sowie das Schreiben von Tests wurden als durchgehende Probleme benannt, die auch zum Ende hin noch bestanden. Da die Abgabe

des Projektes auf Grund von Corona auf Anfang August gelegt wurde, hat die Gruppe betont, das sie das Problem der Tests und der Zeitenbuchung versuchen möchte zu beheben.

Am Ende der Präsentation hat die Gruppe noch eine kurze Demo des Spieles vorgestellt. Hierbei wurde vom Dozenten darauf hingewiesen, dass komplexere Karten noch einmal im Video für die Abgabe vorgestellt werden, da dies in der Demo aus Zeitgründen nicht mehr möglich war.

Die gesamte Gruppe hat das Feedback des Dozenten als positiv aufgenommen, da kaum inhaltliche Fragen gestellt wurden. Außerdem wurde nur eine Ergänzung vom Dozenten gemacht. So wurde die Gruppe darauf hingewiesen, dass das Spiel zur Abgabe auf der Arbi laufen sollte und nicht auf dem privaten NAS eines Gruppenmitgliedes.

5.4 Rückblick über die Sprints

5.4.1 Sprint 1

Der erste Sprint, den wir hatten, lief vom 21.10.2019 bis zum 11.11.2019. Alle waren unsicher bezüglich des Projektes. Aufgaben wie das Anmelden, Abmelden und Account-Management für den Nutzer fertig zu stellen oder an jedes Gruppenmitglied eine Rolle zu vergeben und Vorträge zu organisieren waren vom Projekt vorgegebenen und mussten in diesem Sprint erledigt werden. Der Übergang zum nächsten Sprint hatte sich zeitlich hingezogen. Einer der Gründe hierfür war, dass man zu lange über mögliche Strafen oder Verhaltensregeln diskutiert hatte, die im Nachhinein nie umgesetzt wurden. Gegen Ende des Sprints wurden außerdem Ideen für mögliche User Stories gesammelt.

5.4.2 Sprint 2

Der zweite Sprint verlief vom 11.11.2019 bis zum 25.11.2019. Wie bereits im vorherigen Sprint genannt, musste jedes Gruppenmitglied einen Vortrag zu seiner Rolle im Projekt halten. Nach dem ersten Sprint wurden viele Vorträge noch nicht gehalten, weshalb die Restlichen in diesem Sprint erledigt wurden. Aus den gesammelten User Stories hat die Gruppe mehrere Tickets erstellt. Diese wurden dann gleich nach Dringlichkeit sortiert und an einzelne Gruppenmitglieder verteilt. Am Ende des Sprints hatte die Gruppe eine simple Lobby mit Chat erstellt, wobei der Chat noch Probleme aufwies. Zum gesamten Ablauf dieses Sprints lässt sich sagen, dass die Besprechung über das genaue Lobbydesign mehr Zeit in Anspruch genommen hat als nötig gewesen wäre.

5.4.3 Sprint 3

Der dritte Sprint ging vom 25.11.2019 bis zum 16.12.2019. Es wurde weiter an der Lobby gearbeitet inklusive des serverseitigen Managements von Chat, Userlisten und den Lobbyräumen. Bugs aus dem zweiten Sprint stellten weiterhin ein Problem dar und zusätzliche Bugs mit Userlisten oder bestimmten Use-Cases tauchten auf. Um in den kommenden Sprints mit dem Spielfenster und der Spiellogik anzufangen, wurde entschieden, ein Klassendiagramm zu erstellen, das die Relation von relevanten Klassen für das Spiel serverseitig darstellt. Was hätte besser

laufen können, war das Reviewen von Pull Requests, da sich Gruppenmitglieder entweder nicht selber als Prüfer hinzufügten oder nicht oft genug nachsahen, ob sie als Prüfer hinzugefügt wurden. Dieses Problem wurde im Sprint besprochen und später durch einen Bot in Discord gelöst, der Aktivitäten in Bitbucket mit den Pull-Request und die betroffenen Personen dokumentiert beziehungsweise anpinnt hat.

5.4.4 Sprint 4

Der vierte Sprint fand zwischen dem 16.12.2019 und dem 13.01.2020 statt. Es wurde darüber abgestimmt, ob Tabs zum Managen der einzelnen Fenster benutzt werden sollen. Außerdem wurde die erste Präsentation über den Projektvortschritt gehalten. Alle Gruppenmitglieder wurden entweder einem Server-Team oder Client-Team zugeordnet, um die Arbeitsorganisation besser zu organisieren. Dies führte zu einer besseren Arbeitsqualität im weiteren Verlauf des Projektes. Wir haben zudem eine Abstimmung gehalten, ob in der vorlesungsfreien Zeit ein Sprint stattfinden sollte, mit dem Ergebnis, dass eine Mehrzahl dafür war. Die Datenbank wurde auch erfolgreich ins Projekt eingepflegt und das Spielefenster (die GameView) wurde erstellt. Dieses unterlief im Verlauf der beiden folgenden Sprints mehrere Iterationen aufgrund von Designentscheidungen mit Hinblick auf Nutzerinteraktionen.

In diesem Sprint wurde mehr fertiggestellt als in den vorherigen Sprints. Probleme, die es in diesem Sprint gab, betrafen die Kommunikation, welche zu Fehlern im Develop-Branch geführt hatte. Aufgrund dessen musste ein Teil des Sprints für die Fehlerbehebung genutzt werden.

5.4.5 Sprint 5

Der fünfte Sprint verlief vom 13.01.2019 bis zum 27.01.2020. Dieser Sprint war aufgrund von anstehenden Prüfungen relativ kurz. Die Gruppe hatte sich damit beschäftigt, Mängel in den JavaDoc Kommentaren zu beheben, das Spielefenster zu überarbeiten und sich konkrete Pläne über die Struktur und Implementierung des Spiels und die daraus resultierende Kommunikation zwischen Server und Client zu machen. Dies brachte der Gruppe konkrete Ziele für die vorlesungsfrei Zeit.

5.4.6 Sprint 6

Der sechste Sprint ging vom 27.01.2019 bis zum 27.04.2020. Dieser Sprint fand in der vorlesungsfreien Zeit statt, in der es keine verpflichteten Gruppensitzungen gab. Die jeweiligen Teams haben an ihren Tickets gearbeitet und sich über nötige Veränderungen abgesprochen. Dieser Sprint war unser bisher produktivster Sprint. Ein großer der Teil der Spielelogik wurde komplett oder teilweise fertiggestellt, darunter fällt sowohl das Kaufen und Ausspielen von Karten, als auch die Darstellung dieser auf den Händen der Spieler oder auf dem Spielfeld.

Aufgrund der vorlesungsfreien Zeit trat innerhalb dieses Sprints das Problem auf, dass einige Grupenmitglieder für längere Zeit nicht erreichbar waren. Des Weiteren kam im April 2020 die Corona-Pandemie auf, die es der Gruppe nicht ermöglichte, sich in der Uni zu treffen. Die Gruppe war sich aber darüber einig, dass die Kommunikation und allgemeine Beteiligung besser verlief als bisher.

5.4.7 Sprint 7

Der siebte Sprint fand vom 27.04.2020 bis zum 18.05.2020 statt. Innerhalb der Sprintzeit hat die Gruppe an bestehenden Aufgaben weitergearbeitet. Nebenbei wurden mehrere Bugs gefunden und ausgebessert, die für verschiedene Probleme beim Spielen verantwortlich waren.

An der Dokumentation des Projektes wurde ebenfalls gearbeitet. Diese war vor Sprintbeginn nur in geringem Maße geführt worden.

5.4.8 Sprint 8

Der achte Sprint verlief vom 18.05.2020 bis zum 05.06.2020. Große Tasks vom letzten Sprint waren noch nicht fertig und wurden in diesem Sprint weiterbearbeitet. Die vorletzte Präsentation fand in diesem Sprint statt und es wurde hauptsächlich daran gearbeitet, die Kartenlogik fertigzustellen und das Spiel komplett für maximal vier Personen spielbar zu machen. Dieses Ziel wurde zum Präsentationstermin erreicht, bis auf eine handvoll Karten, deren Fehlen das Spielen nicht beeinflusst hat. Der Sprint verlief ohne große Probleme. Das Arbeiten der Gruppenmitglieder an den jeweiligen Aufgaben war gut organisiert, die fehlende Einbindung von Tests wurde zwar angesprochen, aber auf spätere Sprints verschoben, um im internen Zeitplan zu bleiben.

5.4.9 Sprint 9

Der neunte Sprint ging vom 05.06.2020 bis zum 15.06.2020. Die Gruppe hatte sich als Ziel gesetzt, alle nötigen Vorraussetzungen für die Endpräsentation am 02.07.2020 fertigzustellen. Dies beinhaltete die Implementierung von Bots, Bugfreiheit im Code zu erreichen und die Kartenlogik für alle relevanten Karten, server- und clientseitig zu finalisieren. Unnötige Aufgaben oder Aufgaben mit niedriger Priorität wurden aus dem Backlog rausgenommen. Der Arbeitsprozess verlief sehr geordnet und stätiger Fortschrittt wurde gemacht, wodurch letztendlich alle Ziele beziehungsweise Vorraussetzungen vor dem 02.07.2020 erreicht/erfüllt wurden.

5.4.10 Sprint 10

Der zehnte und letzte Sprint fand vom 15.06.2020 bis 09.08.2020 statt. Das Spiel selbst wurde relativ früh im Lebenszyklus des Sprints fertiggestellt, sodass zur Endpräsentation am 02.07.2020 das Spiel fertig war. Die restliche Zeit des Sprints wurde damit verbacht, die Dokumentation in allen Bereichen fertig zu stellen und die Performance des Spiels zu verbessern. In dieser Phase wurden Probleme wie der hohe Verbrauch von Arbeitsspeicher oder die inkorrekte Vorschau von Karten korrigiert und andere Probleme behoben.

5.5 Verwendete Frameworks, Bibliotheken und Tools

Nachfolgend werden wir die Frameworks, Bibliotheken und Tools nennen, welche wir während des Softwareprojektes genutzt haben und kurz begründen wieso wir sie genutzt haben.

5.5.1 Frameworks

Nachfolgend werden alle von Gruppe B verwendeten Frameworks aufgelistet.

5.5.1.1 Maven

Wir haben Maven als Dependency Tool genutzt, da das Basisprojekt auf diesem aufbaut.

5.5.1.2 JUnit 5

Wir haben uns nach der Präsentation unseres Testbeauftragten dazu entschieden, JUnit für unsere Tests zu benutzten. JUnit ist wichtig, um die Funktionalität der einzelnen Programmabschnitte testen zu können und somit viele Fehler schnellstmöglich ausmerzen zu können. Da Junit/Maven automatisiert vor jedem Run des Servers sowie der Client-Module alle Tests durchläuft, konnten Fehler, die sich bei der Programmierung eingeschlichen haben und die Funktionalität gestört haben, frühzeitig behoben werden. Es wurde versucht, für jede Klasse eine eigene Testklasse zu schreiben und diese soweit es möglich war ohne „Smell Code“ zu testen.

5.5.1.3 JavaFX

Wir haben JavaFX genutzt, da dies von Marco vorgegeben wurde. JavaFX war insofern gut, da es für die Gruppe einfach zu erlernen war und man mit JavaFX relativ einfach die verschiedenen GUIs für das Spiel implementieren konnte. Der gute Support um JavaFX konnte der Gruppe in kniffligen Situationen immer helfen, Bugs oder Probleme zu lösen.

5.5.2 Libraries

Nachfolgend werden die verwendeten Bibliotheken aufgezählt.

5.5.2.1 Netty

Für die Client-Server Kommunikation wurde ebenfalls, weil es bereits im Basisprojekt eingebaut war, Netty benutzt. Anfangs fiel es der Gruppe schwer, sich in die Sender/Empfänger-Theorie einzuarbeiten, aber mit der Zeit fiel es allen immer leichter. Wir konnten so die Business-Logik des Servers von der grafischen Darstellung des Clients sauber trennen.

5.5.2.2 Google Guava

Im Basisprojekt wurde für die Kommunikation Google Guava von Marco implementiert. Diesen Ansatz haben wir weiter verfolgt und den Eventbus primär in Kombination mit Netty genutzt.

5.5.3 Tools

Nachfolgend werden alle von Gruppe B verwendeten Tools aufgelistet.

5.5.3.1 Scenebuilder

Um die gegebenen FXML Dateien visuell bearbeiten zu können, hat sich die Gruppe dazu entschlossen den Scenebuilder in IntelliJ zu nutzen. Dies hat der Gruppe enorm geholfen, da es mit diesem Tool direkt möglich war, zu sehen was die Änderung am Code bewirken. Außerdem könnte die GUI auch direkt über den Scenebuilder verändert werden. Die vorgenommenen Änderungen im Scenebuilder wurden dabei beim Speichern direkt im Code übernommen.

Somit waren alle Gruppenmitglieder,was die visuelle Bearbeitung anging, auf dem gleichen Stand. Leider generierte der Scenebuilder teilweise unnötigen Code, weshalb vorzugsweise direkt in den JavaFX-Dateien Code geändert wurde, um den diesen performanter zu gestalten und nicht notwendigen Code auszulassen.

5.5.3.2 IntelliJ

Um Komplikationen mehrerer IDEs vorbeugen zu können, hat sich die Gruppe B am Anfang des Softwareprojektes darauf geeinigt, IntelliJ als gemeinsame IDE zu verwenden. Es kam der Gruppe sehr entgegen, dass IntelliJ eine Git-Verwaltung anbietet, sodass diese auch stets genutzt wurde.

5.5.3.3 Overleaf

Overleaf wurde für die Dokumentationserstellung genutzt, da dieses Tool uns die Möglichkeit bat, mit mehreren Gruppenmitgliedern gleichzeitig an der Dokumentation zu arbeiten. Außerdem konnte der bearbeitete LaTeXCode direkt visuell dargestellt werden. Dies förderte die Effizienz der Gruppe B bei der Dokumentationserstellung. Zur besseren Strukturierung wurde für die Bearbeitung eine zusätzliche Seite auf Confluence erstellt, welche angab, wer sich mit welchem Thema beschäftigen sollte.

Dadurch übernahmen am Ende nicht nur einige wenige die Dokumentation, sondern die Themen wurden gerecht aufgeteilt, somit kam jeder zum Schreiben. Dabei hat die Gruppe B darauf geachtet, dass auf der Seite bei Confluence geschrieben wurde, ob das Thema bearbeitet wurde und wer das Thema prüfen sollte und ob dieses bereits geschehen ist (hinterlegt in einem extra Ordner „Zusätzlich entstandene Dokumente“ → „Organisation - Dokumentation“).

5.5.3.4 Discord

Zu Kommunikationszwecken verwendete die Gruppe B Discord. Dies war von Vorteil, da man im direkten Kontakt zu den Gruppenmitgliedern Probleme teilen und Fragen stellen konnten und schnell Antworten erhielt. Zudem hat die Gruppe B einen Bot auf dem Discord-Server

hinzugefügt, welcher eine Nachricht sendete, wenn ein neues Ticket erstellt wurde, ein Pull Request erstellt wurde und wenn ebendieser kommentiert wurde.

Somit musste man nicht jeden Tag auf die Atlassian Tools zurückgreifen, sondern konnte auch über den Server auf dem neuesten Stand bleiben. Dabei fiel auf, dass viele Gruppenmitglieder Discord auf ihren mobilen Geräten verwendeten und bspw. Confluence nicht.

5.6 Projekttagebuch

5.6.1 Keinem Sprint zugeordnet

Bei diesen Tickets handelt es sich in der Regel um Aufgaben, die während verschiedener Sprints bearbeitet wurden und meistens keinen direkten Bearbeitenden besaßen. Einige hiervon dienen auch der reinen Buchung von Zeiten.

- SWP2019B-1 - Gruppensitzung
- SWP2019B-56 - Hilfestellung geben
- SWP2019B-94 - Projekttagebuch
- SWP2019B-98 - Confluence & Doku
- SWP2019B-148 - Dokumentation
- SWP2019B-247 - Java-TODOs abarbeiten
- SWP2019B-187 - JavaDoc pflegen
- SWP2019B-280 - Bug-Hunting

5.6.2 Sprint 1

Der erste Sprint war für jedes Gruppenmitglied eine Einzelaufgabe. Das Vorgehen hierfür wurde durch Übungsblätter vorgegeben.

Folgende User Stories wurden bereits durch das Basisprojekt selbst erfüllt:

- *SWP2019B-7 - Ein Gast muss sich registrieren können, damit er sich einloggen kann*
- *SWP2019B-8 - Die Nutzer sollen nach dem Einloggen in das Hauptmenü gelangen und dort alle Nutzer sehen, die auch eingeloggt sind*

Folgende User Stories wurden durch die Bearbeitung des zweiten Übungsblattes erfüllt:

Ticketname	*SWP2019B-18 - Als Nutzer möchte ich meinen Account löschen können*
Typ	Story

Lösung	durch Bearbeitung der Aufgabe 2 des zweiten Übungsblattes erfüllt
Ticketname	*SWP2019B-92 - Als Benutzer möchte ich mich abmelden könne und somit wieder zum Loginbildschirm gelangen*
Typ	Story
Lösung	durch Bearbeitung der Aufgabe 3 des zweiten Übungsblattes erfüllt

5.6.3 Sprint 2

Ticketname	*SWP2019B-16 - Als Nutzer möchte ich eine Anleitung haben, um mich als Neuling in dem Spiel zurecht finden zu können.*
Typ	Story
Bearbeiter	Timo Siems
Lösung	Die Anleitung des Spieles wurde erstmals als PDF erstellt (hinterlegt in einem extra Ordner „Zusätzlich entstandene Dokumente“ → „Dominion - Anleitung - V1“).
Ticketname	*SWP2019B-60 - Als Nutzer möchte ich im Hauptmenü eine Liste der aktuell bestehenden Lobbys sehen können*
Typ	Story
Bearbeiter	Julia Debkowski, Rike Hochheiden, Keno Oelrichs Garcia
Lösung	Im Hauptmenü wurde eine Lobbytabelle hinzugefügt, die die aktuellen Lobbies anzeigt. Eine neu erstellte Lobby wurde in einem neuen Fenster geöffnet.
Unteraufgaben	• *SWP2019B-119 - Umwandlung in TableView.*
Ticketname	*SWP2019B-65 - Als Nutzer möchte ich im Hauptmenü ein Chat-Fenster angezeigt bekommen*
Typ	Story
Bearbeiter	Fenja Bauer, Keno Oelrichs Garcia
Lösung	Chat Fenster FXML kann in andere ViewPresenter eingebunden werden. Wurde in den Branch von Ticket 70 gemergt.
Ticketname	*SWP2019B-69 - Der Chat soll serverseitig implementiert sein*
Typ	Story
Bearbeiter	Keno Oelrichs Garcia
Lösung	Der ChatService wurde serverseitig implementiert, außerdem werden die Nachrichten vom Client empfangen. Wurde in Ticket 70 durch Merge in den Branch von Ticket 71 eingebunden.

Unteraufgaben

- *SWP2019B-72 - Der Server empfängt die Nachricht und gibt dem Versender eine Rückmeldung.*
- *SWP2019B-100 - Interfaces in Common erstellen.*
- *SWP2019B-103 - Eingehende Nachrichten automatisch im Chatfenster im Client angezeigt bekommen.*

Ticketname	*SWP2019B-70 - Als Nutzer will ich einen Button haben, mit dem ich Nachrichten verschicken kann.*
Typ	Story
Bearbeiter	Darian Alves, Alexander Linke
Lösung	Chatnachrichten werden bei senden (über Enter/Button) auf dem EventBus verschickt. Neue Nachrichten werden abgefragt und angezeigt, wobei längere Texte automatisch umgebrochen werden. Bei Ein-/Ausloggen eines Useres wird dieses im Chat angezeigt. Es wurde ein neues Loginfenster implementiert und der Chat designt.

Unteraufgaben

- *SWP2019B-71 - Der Client versendet die Nachricht an den Server.*
- *SWP2019B-101 - Lobby Chatfenster FXML integrieren.*

Ticketname	*SWP2019B-87 - Als User möchte ich einen Button haben, mit welchem ich Lobbys erstellen kann*
Typ	Story
Bearbeiter	Ferit Askin, Mahmoud Haschem, Paulina Kowalska
Lösung	Es wurde ein Button implementiert, mit dem der User eine neue Lobby mit Lobbynamen erstellen kann. Ist der Lobbyname leer, bekommt der User eine Fehlermeldung.

Unteraufgaben

- *SWP2019B-90 - FXML Button hinzufügen.*
- *SWP2019B-91 - Lobby erstellen.*

Ticketname	*SWP2019B-97 - Als User will ich einen Logout Button haben, um meine Session zu beenden.*
Typ	Story
Bearbeiter	Anna Hiemenz
Lösung	Das Basisprojekt wurde erweitert, damit die Logout-Funktion genutzt werden kann.

Für folgende User Stories wurde entschieden, sie nicht umzusetzen
- Lösung: **Won't do**:

- SWP2019B-15 - Als Nutzer möchte ich meine Freundesliste verwalten können.
- SWP2019B-68 - Als Nutzer möchte ich ein Lobby-Fenster sehen können
- SWP2019B-86 - Fxml Button hinzufügen.

Folgende Duplikate sind in diesem Sprint aufgetreten (mit ihren Unteraufgaben)
- Lösung: **Duplikat**:

- SWP2019B-82 - Als Nutzer möchte ich eine Lobby erstellen können
- SWP2019B-89 - Als Nutzer möchte ich einen separaten Lobbychat haben

Einige Tickets (mit ihren Unteraufgaben) wurden mit in den Sprint 3 übernommen, da ihre Bearbeitung noch nicht abgeschlossen war:

- SWP2019B-14 - Als Nutzer möchte ich über einen Chat im Lobbyfenster, um mit anderen Nutzern kommunizieren können.
  - SWP2019B-54 - Den Chat in die Lobby einbinden.
- SWP2019B-83 - Als Nutzer möchte ich vom Hauptmenü aus das Lobbyfenster erreichen können.

5.6.4 Sprint 3

Ticketname	*SWP2019B-14 - Als Nutzer möchte ich über einen Chat im Lobbyfenster, um mit anderen Nutzern kommunizieren können.*
Typ	Story
Bearbeiter	Keno Schwedler, Darian Alves
Lösung	Es wurde ein Chat in der Lobby implementiert. Betritt ein User eine Lobby, wird diese im Chat gepostet.
Unteraufgaben	• *SWP2019B-54 - Den Chat in die Lobby einbinden.*

Ticketname	*SWP2019B-66 - Als Nutzer möchte ich einer bestehenden Lobby beitreten können*
Typ	Story
Bearbeiter	Paulina Kowalska, Julia Debkowski
Lösung	in den Branch von Ticket 67 gemergt
Unteraufgaben	• *SWP2019B-130 - Button in Lobbyliste im Hauptmenü*

Ticketname	*SWP2019B-67 - Als Nutzer möchte ich eine Lobby verlassen können*
Typ	Story

Bearbeiter Paulina Kowalska, Julia Debkowski

Lösung Es wurde ein „Lobby verlassen“-Button implementiert, über den ein User eine Lobby verlassen kann.

Unteraufgaben

- *SWP2019B-128 - FXML Button zum verlassen*
- *SWP2019B-129 - Zurück zum Hauptmenü gelangen*

Ticketname *SWP2019B-102 - Als Nutzer möchte ich erst das Spiel starten können, wenn alle Spieler bereit dazu sind.*

Typ Story

Bearbeiter Keno Schwedler, Keno Oelrichs Garcia, Darian Alves

Lösung Jeder User einer Lobby besitzt einen „Bereit“-Button, der inital auf „Nicht Bereit“ steht. Ein Spiel wird erst dann gestartet, wenn alle User einer Lobby ihren „Bereit“-Button angeklickt haben.

Unteraufgaben

- *SWP2019B-115 - Als Nutzer möchte ich einen Button haben, mit welchem ich meinen Status ändern kann.*
- *SWP2019B-116 - Als Nutzer möchte ich einen Status haben, welcher im Lobbyfenser angezeigt wird.*
- *SWP2019B-134 - FXML Button, um zum Spielefenster zu gelangen.*
- *SWP2019B-143 - Lobby Mitgliederliste hinzufügen.*

Ticketname *SWP2019B-140 - Als Nutzer möchte ich mehrere Lobbyfenster parallel geöffnet haben, damit ich mehrere Spiele spielen kann.*

Typ Story

Bearbeiter Anna Hiemenz

Lösung Wenn der User mehrere Spiele gleichzeitig spielen möchte, öffnen sich diese in verschiedenen Fenstern.

Ticketname *SWP2019B-142 - Java Klassendiagramm zum einfachen Spielmodell*

Typ Aufgabe

Bearbeiter Alexander Linke, Devin Schlüter, Ferit Askin

Lösung Für die grobe Planung der Spielumsetzung wurde ein grobes Java Klassendiagramm erstellt (hinterlegt in einem extra Ordner „Zusätzlich entstandene Dokumente“ → „Dominion - Java Klassendiagramm grob“).

Ticketname *SWP2019B-156 - User wird aus GameView ausgeloggt, aber nicht aus der Lobby entfernt*

Typ Bug

Bearbeiter Julia Debkowski

Lösung Beim Logout des Users wird dieser auch aus allen Lobbies ausgeloggt.

Für folgende User Stories wurde entschieden, sie nicht umzusetzen
- Lösung: **Won't do**:

- SWP2019B-81 - Als Nutzer möchte ich mein Passwort abfragen/ändern können
- SWP2019B-141 - Im Chat wird zweimal angezeigt, wer sich einloggt.

Folgende Duplikate sind in diesem Sprint aufgetreten (mit ihren Unteraufgaben)
- Lösung: **Duplikat**:

- SWP2019B-22 - Als Nutzer will ich mehrere Spiele gleichzeit spielen können.
- SWP2019B-62 - Als Nutzer möchte ich ein neues Spiel starten können.
- SWP2019B-74 - Als Nutzer möchte ich über das Hauptmenü zu den Einstellungen gelangen können, um diese einsehen zu können
  - SWP2019B-121 - Einstellungsfenster FXML
  - SWP2019B-122 - Button um in Einstellungen zu kommen im Hauptmenü
  - SWP2019B-123 - Einstellungen werden vom Server abgerufen und angezeigt
- SWP2019B-96 - Als Nutzer möchte ich im Lobbyfenster eine Liste der in dieser Lobby angemeldeten User sehen können
- SWP2019B-109 - Als Nutzer möchte ich ein Chatfenster im Spiel haben, um mit anderen Spielern kommunizieren zu können
- SWP2019B-113 - Als Nutzer möchte ich erst das Spiel starten können, wenn alle Spieler bereit dazu sind.
  - SWP2019B-118 - FXML Button, um zum Spielefenster zu gelangen
- SWP2019B-157 - Anpassung GameView in Develop
- SWP2019B-160 - Als Nutzer möchte ich aufgeben können.
- SWP2019B-161 - Sequenzdiagramm LobbyReadyList

Folgende Tickets wurden **geschlossen**, da diese bereits (in anderen Branches) bearbeitet wurden:

- SWP2019B-77 - Als Nutzer möchte ich eine Chatnachricht mit der Enter-Taste abschicken können
- SWP2019B-83 - Als Nutzer möchte ich vom Hauptmenü aus das Lobbyfenster erreichen können.
- SWP2019B-84 - Als Nutzer möchte ich im Chat über ein bzw. ausgeloggte Benutzer informiert werden

Einige Tickets wurden mit in den Sprint 4 bzw. Sprint 6 übernommen, da ihre Bearbeitung noch nicht abgeschlossen war:

- SWP2019B-85 - Als User möchte ich mein Spiel aufgeben können. (Sprint 6)
- SWP2019B-114 - Als Nutzer möchte ich entscheiden, mit wie vielen Spielern ich spielen möchte (Sprint 4)

5.6.5 Sprint 4

Ticketname	*SWP2019B-61 - Als Nutzer möchte ich über die Einstellungen meine persönlichen Daten (z.B. Name, Passwort) ändern können.*
Typ	Story
Bearbeiter	Anna Hiemenz, Julia Debkowski
Lösung	Über den Button „Einstellungen“ wird die SettingsView geöffnet. Hier kann der User verschiedene Daten von sich ändern und auch seinen Account löschen.
Unteraufgaben	• *SWP2019B-76 - .fxml eines Einstellungfensters + SettingsPresenter Java Klasse* • *SWP2019B-79 - SettingsPresenter Java Klasse erstellen* • *SWP2019B-124 - Button im Hauptmenü, Einstellungsfenster öffnet sich wenn Button geklickt wurde* • *SWP2019B-125 - Fenster Öffnen wenn Button angeklickt wird* • *SWP2019B-126 - Einstellungen werden vom Server geladen und gespeichert* • *SWP2019B-127 - Einstellungen Serverseitig speichern* • *SWP2019B-176 - Message wird vom Client empfangen und verarbeitet* • *SWP2019B-177 - Implementierung der Buttons* • *SWP2019B-178 - Implementierung Account löschen* • *SWP2019B-182 - Einstellungsfenster visuell ansprechend gestalten*

Ticketname	*SWP2019B-88 - Als Nutzer möchte ich, dass meine Email-Adresse bei der Registrierung abgefragt wird, und bei falschen Email-Adressen ein Fehler auftritt.*
Typ	Story
Bearbeiter	Timo Siems
Lösung	Bei der Registrierung wird die angegebene Mail-Adresse auf Gültigkeit überprüft. Ist die angegebene Mail-Adresse nicht gültig, gibt es eine Fehlermeldung.

Ticketname	*SWP2019B-107 - Als Nutzer möchte ich, dass beim Klicken von Buttons passende Sound zu hören sind*
Typ	Neue Funktion
Bearbeiter	Keno Oelrichs Garcia
Lösung	Beim anklicken eines Buttons bzw. beim mit der Maus über der Button gehen ertönt ein Sound.

Ticketname	*SWP2019B-114 - Als Nutzer möchte ich entscheiden, mit wie vielen Spielern ich spielen möchte*
Typ	Story
Bearbeiter	Timo Siems, Rike Hochheiden, Keno Schwedler
Lösung	Der Owner einer Lobby hat über eine Combobox die Möglichkeit zu entscheiden wie viele Spieler maximal in der Lobby sein können.

Ticketname	*SWP2019B-153 - Als Nutzer möchte ich private Lobbys erstellen können.*
Typ	Story
Bearbeiter	Paulina Kowalska, Rike Hochheiden
Lösung	Bei Erstellung einer Lobby hat der User die Möglichkeit, ein Passwort für diese anzugeben. In der Lobbytabelle wird dies durch den Zusatz „(privat)“ bzw. „(offen)“ hinter dem Lobbynamen gekennzeichnet. Will ein anderer User eine private Lobby betreten, so muss dieses das Passwort eingeben.

Ticketname	*SWP2019B-155 - UserList in Lobby wird bei neue beigetretenen nicht aktualisiert.*
Typ	Bug
Bearbeiter	Keno Oelrichs Garcia, Paulina Kowalska
Lösung	Bug wurde gefixt.

Ticketname	*SWP2019B-158 - Spiel wird erst gestartet, wenn alle Spieler bereit sind*
Typ	Bug
Bearbeiter	Keno Oelrichs Garcia
Lösung	Bug wurde gefixt.

Ticketname	*SWP2019B-169 - Als Ersteller einer Lobby möchte ich andere Leute aus der Lobby rausschmeißen können*
Typ	Story
Bearbeiter	Darian Alves
Lösung	Der Owner kann über einen Button andere Spieler aus einer Lobby kicken, wenn das Spiel noch nicht gestartet ist.

Ticketname	*SWP2019B-170 - Als Nutzer möchte ich mich nicht bei mehreren Clients mit meinem Account anmelden können.*

Typ	Story
Bearbeiter	Keno Schwedler
Lösung	Es ist nur noch möglich sich bei einem Client mit seinem Account anzumelden.

Ticketname	*SWP2019B-172 - Das Hauptmenü soll visuell ansprechender gestaltet sein*
Typ	Verbesserung
Bearbeiter	Keno Oelrichs Garcia, Fenja Bauer
Lösung	Das Hauptmenü (MainMenuView.fxml) wurde visuell neu gestaltet.
Unteraufgaben	

- *SWP2019B-174 - Online Spieler Liste gestalten*

Ticketname	*SWP2019B-173 - Das aktualisierte Basissystem soll in das aktuelle System eingepflegt werden*
Typ	Aufgabe
Bearbeiter	Ferit Askin, Keno Schwedler
Lösung	Änderungen im Basissystem wurden in unser Projekt übernommen.

Ticketname	*SWP2019B-175 - Join Lobby Fehler*
Typ	Bug
Bearbeiter	Julia Debkowski
Lösung	Bug wurde behoben.

Ticketname	*SWP2019B-180 - Als Nutzer möchte ich sehen, wer in der Lobby der derzeitige Owner ist*
Typ	Verbesserung
Bearbeiter	Darain Alves
Lösung	In der Lobbytabelle wird der Owner einer Lobby als „Host“ angezeigt.

Ticketname	*SWP2019B-184 - Lobbytabelle-Verbesserung*
Typ	Aufgabe
Bearbeiter	Julia Debkowski
Lösung	Die Lobbytabelle aktualisiert sich bei Änderungen automatisch.

Folgende Duplikate sind in diesem Sprint aufgetreten (mit ihren Unteraufgaben)
- Lösung: **Duplikat**:

- SWP2019B-17 - Als Nutzer möchte ich gegen andere User spielen können
- SWP2019B-106 - Als Nutzer möchte ich, dass mich durch Anmeldebildschirm und Hauptmenü Musik begleitet
- SWP2019B-139 - Als Nutzer möchte ich, dass die Online-SpielerListe visuell ansprechend ist

Folgende Tickets wurden **geschlossen**, da diese bereits bearbeitet wurden:

- SWP2019B-159 - Max. Spieler Bug - Anzeigefehler + Lobbyjoinbug

5.6.6 Sprint 5

Ticketname	*SWP2019B-64 - Als Nutzer möchte ich im Spiel auf eine Spielanleitung zugreifen können.*
Typ	Story
Bearbeiter	Timo Siems
Lösung	Über einen Button kann der User in dem Spiel, sich die Spieleanleitung anzeigen lassen.

Ticketname	*SWP2019B-147 - Grob das Java Klassendiagramm zum Spielmodell in das Softwareprojekt einpflegen.*
Typ	Aufgabe
Bearbeiter	Devin Schlüter
Lösung	Das Java Klassendiagramm, welches in Sprint 3 erstellt wurde, wurde überarbeitet.

Ticketname	*SWP2019B-189 - Als Nutzer möchte ich meine Aktions- bzw. Kaufphase auch selbständig beenden können.*
Typ	Story
Bearbeiter	Julia Debkowski
Lösung	Das serverseitige Vorgehen zum Skippen einer Phase wurde implementiert. Damit hat der User die Möglichkeit eine Phase zu skippen.

Ticketname	*SWP2019B-191 - Als Nutzer möchte ich, dass ich bei Spielstart meine Startkarten sehe*
Typ	Story
Bearbeiter	Devin Schlüter
Lösung	Das clientseitige Vorgehen zum Darstellen der Starthand wurde implementiert. Karten werden der GameView hinzugefügt.

Ticketname	*SWP2019B-194 - GameManagement serverseitig*
Typ	Verbesserung
Bearbeiter	Keno Oelrichs Garcia, Julia Debkowski
Lösung	Das Serverseitige GameManagement wurde implementiert (Skizze hierfür ist hinterlegt in einem extra Ordner „Zusätzlich entstandene Dokumente“ → „Skizze-GameManagement“).

Ticketname	*SWP2019B-199 - LobbyNamen auf LobbyID umstellen*
Typ	Bug
Bearbeiter	Marvin Thate

Lösung	An einigen Stellen wurde auf den LobbyNamen verwiesen, anstatt auf die LobbyID. Dies wurde umgestellt.
Ticketname	*SWP2019B-200 - Als User möchte ich, dass ich im Spiel als Player übergeben werde.*
Typ	Story
Bearbeiter	Julia Debkowski
Lösung	Der User wird serverseitig im Spiel als Player übergeben.
Ticketname	*SWP2019B-201 - Als User möchte ich Karten im Spiel besitzen*
Typ	Story
Bearbeiter	Paulina Kowalska
Lösung	Serverseitig wurden die Karten eines Spielers, die er besitzt implementiert.
Ticketname	*SWP2019B-202 - Als User möchte ich, dass die Nachricht, welche Karten ich auf der Hand habe, an den Client gesendet wird*
Typ	Story
Bearbeiter	Ferit Askin
Lösung	Es wurde eine Nachricht erstellt, mit der die Karten auf der Hand vom Server an den Client gesendet werden.
Ticketname	*SWP2019B-204 - Als Nutzer möchte ich, dass die Reihenfolge der Spieler zufällig erfolgt*
Typ	Story
Bearbeiter	Fenja Bauer
Lösung	Spielerliste wird dem Playground hinzugefügt und zu Beginn des Spieles gemischt.
Ticketname	*SWP2019B-205 - Als User möchte ich, dass bei der Auswahl einer Karte eine Nachricht mit der KartenID an den Server gesendet wird*
Typ	Story
Bearbeiter	Paulina Kowalska
Lösung	Neben einer SelectCardRequest wurde auch eine AbstractGameMessage, die die GameID, die CardID und den Player besitzt, zur Kommunikation erstellt.
Ticketname	*SWP2019B-208 - Als User möchte ich, dass es eine Generalisierung der Java Klassen von ActionPhase und BuyPhase und ClearPhase gibt*
Typ	Story
Bearbeiter	Keno Oelrichs Garcia
Lösung	Es wurde eine Generalisierung der ActionPhase, BuyPhase und ClearPhase implementiert.

Ticketname	*SWP2019B-209 - Als User möchte ich, dass das Spiel graphisch realisiert wird*
Typ	Story
Bearbeiter	Devin Schlüter, Anna Hiemenz, Marvin Thate
Lösung	Die GameView wurde visuell angepasst.

Ticketname	*SWP2019B-210 - Als User möchte ich, dass es im Common ein JSON Objekt gibt*
Typ	Story
Bearbeiter	Devin Schlüter, Ferit Askin, Keno Oelrichs Garcia
Lösung	Informationen über die Karten werden über eine JSON-Datei zur Verfügung gestellt.

Ticketname	*SWP2019B-213 - Als Nutzer möchte ich, dass eine Kartenanimation existiert*
Typ	Story
Bearbeiter	Anna Hiemenz
Lösung	Es wurde eine Java Klasse für die Animation auf dem Client erstellt.

Ticketname	*SWP2019B-218 - Als User möchte ich wissen, ob ich eine Aktionskarte spielen kann*
Typ	Story
Bearbeiter	Julia Debkowski
Lösung	Beim Spielerwechsel wird überprüft ob der User eine Aktionskarte auf der Hand und kommt je nachdem in die Aktionsphase oder direkt in die Kaufphase.

Ticketname	*SWP2019B-219 - Als User möchte ich wissen, welcher Spieler gerade dran ist und in welcher Phase sich dieser befindet.*
Typ	Story
Bearbeiter	Paulina Kowalska
Lösung	Es wurde eine get- und eine set-Methode für die aktuelle Phase ergänzt. Die Ausgabe des aktuellen Spieler wurde durch ein anderes Ticket bereits ergänzt.

Ticketname	*SWP2019B-226 - Beim verlassen der Lobby tritt ein NullPointer Exception auf*
Typ	Bug
Bearbeiter	Darian Alves
Lösung	Fehler wurde behoben. Hat der letzte User eine die Lobby verlassen, wird diese in der Lobbyliste entfernt.

Ticketname	*SWP2019B-227 - Chat funktioniert in GameView, LobbyView und generell nicht, bzw wird nicht gefunden.*
Typ	Bug

Bearbeiter	Darian Alves
Lösung	Bug wurde behoben.

Ticketname	*SWP2019B-229 - Karten bewegen sich im Spiel nicht dahin wo sie sollten*
Typ	Bug
Bearbeiter	Anna Hiemenz
Lösung	Das AnimationManagement wurde angepasst, damit die Bewegungen wieder korrekt angezeigt werden.

Für folgende User Stories wurde entschieden, sie nicht umzusetzen
- Lösung: **Won't do**:

- SWP2019-203 - Als Nutzer möchte ich, dass dem Server eine SessionID übergeben wird, um Nachrichten nicht an alle Clients zu senden
- SWP2019-217 - Als Nutzer möchte ich, dass die Position einer Karte durch ein eigenes Positions-Objekt gespeichert wird

Einige Tickets wurden mit in den Sprint 7 übernommen, da ihre Bearbeitung noch nicht abgeschlossen war:

- SWP2019B-186 - Spielkarten vorbereiten

Folgende Tickets wurden **geschlossen**, da diese bereits durch ein anders Ticket mit bearbeitet wurden:

- SWP2019B-181 - Login Bug

5.6.7 Sprint 6

Ticketname	*SWP2019B-85 - Als User möchte ich mein Spiel aufgeben können.*
Typ	Story
Bearbeiter	Mahmoud Haschem, Marvin Thate, Fenja Bauer, Keno Oelrichs Garcia, Ferit Askin
Lösung	Der User kann über einen Button sein Spiel aufgeben. Dies wird auch im Chat, des jeweiligen Spiels angezeigt.

Unteraufgaben

- *SWP2019B-131 - FXML Spielefenster*
- *SWP2019B-132 - Chat im Spielefenster*
- *SWP2019B-133 - Button zum verlassen*
- *SWP2019B-145 - User-Liste*
- *SWP2019B-146 - Aufgeben-Button*

Ticketname *SWP2019B-188 - Als Nutzer möchte ich in der Kaufphase Karten kaufen können.*
Typ Story
Bearbeiter Paulina Kowalska, Ferit Askin
Lösung Serverseitiges Vorgehen beim Kaufen einer Karte wurde implementiert.
Unteraufgaben

- *SWP2019B-230 - BuyCardRequest*
- *SWP2019B-231 - Karte dem Deck des Spielers hinzufügen, wenn er genug Geld hat*
- *SWP2019B-232 - BuyCardMessage*

Ticketname *SWP2019B-211 - Als User möchte ich, dass mir meine Karten überall dargestellt werden können*
Typ Story
Bearbeiter Rike Hochheiden, Julia Debkowski
Lösung Für die Karten im Shop wurden ImageViews ergänzt und eine Aktion implementiert, die beim anklicken der Karte ausgeführt wird.
Unteraufgaben

- *SWP2019B-212 - Als Nutzer möchte ich, dass ich eine Karte anklicken kann und eine Aktion ausgeführt wird*

Ticketname *SWP2019B-216 - Als User möchte ich sehen, wenn sich die Spieler einer Lobby im Game befinden*
Typ Story
Bearbeiter Julia Debkowski
Lösung Die Lobbytabelle beinhaltet den Zustand eines Spieles. Ein roter Punkt signalisiert, dass das Spiel im Gange ist, ein grüner das das Spiel noch nicht gestartet ist.

Ticketname *SWP2019B-220 - Als User möchte ich, dass in jeder Clearphase überprüft wird, ob das Spiel zu Ende ist*
Typ Story
Bearbeiter Fenja Bauer

Lösung	Es wurde eine Methode implementiert die in jeder Clearphase überprüft, ob das Spiel beendet ist.
Ticketname	*SWP2019B-221 - Als User möchte ich in der Clearphase eine neue Hand bekommen*
Typ	Story
Bearbeiter	Julia Debkowski
Lösung	Die aktuelle Hand des Spielers wird in seiner Clearphase auf den Ablagestapel gelegt und eine neue Hand wird ihm zugesendet.
Ticketname	*SWP2019B-224 - Als User möchte ich, dass mir am Ende des Spiels angezeigt wird, wer gewonnen hat.*
Typ	Story
Bearbeiter	Anna Hiemenz
Lösung	Es wurde eine GameOverView ergänzt, die den Usern anzeigt, wer gewonnen hat.
Ticketname	*SWP2019B-225 - Die Anzahl der Spieler für ein Spiel kann in der Lobby nicht geändert werden.*
Typ	Bug
Bearbeiter	Darian Alves
Lösung	Bug wurde behoben.
Ticketname	*SWP2019B-233 - Als Nutzer möchte ich, das die Aktionen der Aktions- und Reaktionskarten in der JSON-Datei des Kartenpacks gespeichert werden und geladen werden können, damit diese Informationen im Spiel logisch umgesetzt werden können.*
Typ	Neue Funktion
Bearbeiter	Keno Oelrichs Garcia, Timo Siems
Lösung	Alle Aktionen werden generalisiert in der JSON gespeichert.
Ticketname	*SWP2019B-234 - Als Nutzer möchte ich, dass die Zeit in der Aktionsphase begrenzt wird. Ich möchte sehen wie viel Zeit ich noch habe.*
Typ	Story
Bearbeiter	Ferit Askin
Lösung	Die Aktionsphase wird nach einer bestimmten Zeit, in der der User nichts tut geskippt. Anmerkung: Diese Funktion wurde im weitere Projekt wieder herausgenommen und sollte zu einem späteren Zeitpunkt überarbeitet werden, da sie im weiteren Verlauf zu Problemen geführt hat.
Ticketname	*SWP2019B-237 - Als Nutzer möchte ich wissen, wieviele Karten noch auf dem Stapel verfügbar sind.*
Typ	Verbesserung
Bearbeiter	Ferit Askin

Lösung	Serverseitig wird die Anzahl der Karten auf den Kartenfeldern gesetzt.
Ticketname	*SWP2019B-238 - Die Methode zur Handkartendarstellung wird nicht durchgeführt.*
Typ	Bug
Bearbeiter	Devin Schlüter
Lösung	Bug wurde behoben.
Ticketname	*SWP2019B-239 - Als Spieler möchte ich eine Phase überspringen können, falls ich keine Handkarten kaufen möchte*
Typ	Story
Bearbeiter	Devin Schlüter
Lösung	Es wurde ein Button ergänzt, mit dem der User die Möglichkeit hat, eine Phase zu überspringen. Das serverseitige Vorgehen wurde bereits implementiert (Ticket 189 in Sprint 5)
Ticketname	*SWP2019B-242 - Die Nummern in der Kartenbezeichnungen sollen identisch mit den jeweiligen IDs in der JSON sein*
Typ	Verbesserung
Bearbeiter	Rike Hochheiden
Lösung	Ähnlich der Siegpunktkarten wird bei den Karten im Shop, sowie den Geldkarten angezeigt, wie viele Karten dieses Stapels noch zur Verfügung stehen.
Ticketname	*SWP2019B-243 - Als Nutzer möchte ich, dass sich meine Handkarten immer mittig anordnen.*
Typ	Story
Bearbeiter	Anna Hiemenz
Lösung	In der GameView wurde ein HandcarLayoutContainer hinzugefügt in dem die Handkarten eingefügt werden. Außerdem wurde das AnimationManagement angepasst.
Ticketname	*SWP2019B-244 - Der Gewinner soll ermittelt werden und an alle gesendet werden*
Typ	Verbesserung
Bearbeiter	Julia Debkowski
Lösung	Der Gewinner eines Spiels wird ermittelt und an alle User des Spiels gesendet. Das Ergebnis wird durch die bereits erstellte GameOverView (Ticket 224 im selben Sprint) kommuniziert.
Ticketname	*SWP2019B-245 - Als User möchte ich, dass die letzte Karte, die auf den Ablagestapel kommt, übergeben wird, damit sie clientseitig angezeigt werden kann.*
Typ	Verbesserung
Bearbeiter	Fenja Bauer

Lösung	Serverseitig wurde eine Message erstellt die dem Client mitgeteilt, welche Karte zuletzt abgelegt wird, damit diese auf dem Ablegestapel dar gestellt werden kann.
Ticketname	*SWP2019B-248 - inGame Status der Lobby wird bei Spielende nicht aktualisiert*
Typ	Bug
Bearbeiter	Julia Debkowski
Lösung	Der Bug wurde behoben.
Ticketname	*SWP2019B-257 - Als User möchte ich, dass die letzte Karte, die auf den Ablagestapel kommt angezeigt wird*
Typ	Story
Bearbeiter	Timo Siems
Lösung	Die vom Server gesendete letzte Karte für den Ablagestapel (siehe Ticket 245) wird clientseitig dargestellt.
Ticketname	*SWP2019B-261 - Lobbychat Hotfix*
Typ	Bug
Bearbeiter	Keno Oelrichs Garcia
Lösung	ChangeMaxPlayer Anchorpane wurde richtig in das Gridpane gelegt. Damit wurde der Bug behoben.

Folgende Duplikate sind in diesem Sprint aufgetreten (mit ihren Unteraufgaben)
- Lösung: **Duplikat**:

- SWP2019B-163 - Als Nutzer möchte ich, dass ich Karten auf das Spielfeld platzieren kann, um mit dem Spiel zu interagieren.
- SWP2019B-164 - Als Nutzer möchte ich Karten im Spiel sehen können, um die Karten voneinander zu unterscheiden. Anzeigelogik
- SWP2019B-165 - Als Nuter möchte ich, ein Deckstapel haben, von dem ich Karten ziehen kann, um Karten zu haben, mit denen ich interagieren kann.
- SWP2019B-166 - Als Nutzer möchte ich sehen können, wie viel Geld und Siegespunkte ich habe, um über meinem Spielzustand bescheid zu wissen.
- SWP2019B-167 - Als Nutzer möchte ich angezeigt bekommen, welcher Spieler gerade am Zug ist, um es zu wissen.
- SWP2019B-197 - Als Nutzer möchte ich eine Karte auswählen können (DOPPELT)
- SWP2019B-250 - Umstellung von ImageView auf ID bei der onBuyCardMessage und onPlayCardMessage - GameViewPresenter

Einige Tickets (mit ihren Unteraufgaben) wurden mit in den Sprint 7 übernommen, da ihre Bearbeitung noch nicht abgeschlossen war:

- SWP2019B-214 - Als Nutzer möchte ich, meine zusätzliche Punkte zum aktuellen Spiel angezeigt bekommen
- SWP2019B-215 - Als Nutzer möchte ich meine Gegner sehen können
- SWP2019B-236 - Als Nutzer möchte ich wissen, wer der erste Spieler ist.
- WP2019B-241 - eine Blockade einbauen, dass man sich nicht ausloggen/seinen Account löschen kann, wenn man im Spiel ist.
- SWP2019B-246 - Grafische Darstellung der Clearphase
- SWP2019B-251 - Als Nutzer möchte ich meine Karte ausspielen können, damit sie ihre Aktion ausführen kann.
- SWP2019B-252 - Kartenlogik, Serverseitige implementierung
- SWP2019B-253 - Als User möchte ich, dass mir der Kauf einer Karte auch visuell
  - SWP2019B-254 - Stapel in der Mitte: Bild entfernen, falls keine Karte mehr vorhanden ist
  - SWP2019B-255 - Bei erfolgreichen Kauf, müssen die Geldkarten des Spielers abgezogen werden
  - SWP2019B-256 - Fehlermeldung, wenn der Spieler nicht genug Geld hat
- SWP2019B-263 - NullPointerException im EventBus
- SWP2019B-267 - Als Nutzer möchte ich, dass ich Benachrichtigungen zum Spiel ausschalten kann

5.6.8 Sprint 7

Ticketname	*SWP2019B-138 - Als Nutzer möchte ich, dass man vom Hauptmenü aus mit einem Tab zu der LobbyView gelangt und somit Buttons nicht wiederholt werden*
Typ	Verbesserung
Bearbeiter	Fenja Bauer, Keno Oelrichs Garcia
Lösung	Es wurde eine PrimaryView hinzugefügt und Tabs implementiert.

Ticketname	*SWP2019B-186 - Spielkarten vorbereiten*
Typ	Aufgabe
Bearbeiter	Alexander Linke
Lösung	Karten und ihre Thumbnails, sowie die zugehörige JSON-Datei wurden erstellt.

Ticketname	*SWP2019B-236 - Als Nutzer möchte ich wissen, wer der erste Spieler ist.*
Typ	Verbesserung
Bearbeiter	Marvin Thate

Lösung	Der User, der das Spiel beginnt, wird über eine Nachricht im Chat angegeben.
Ticketname	*SWP2019B-241 - eine Blockade einbauen, dass man sich nicht ausloggen/seinen Account löschen kann, wenn man im Spiel ist.*
Typ	Verbesserung
Bearbeiter	Darian Alves
Lösung	Der User kann sich nicht mehr ausloggen bzw. seinen Account löschen, wenn er sich in einem Spiel befindet.
Ticketname	*SWP2019B-246 - Grafische Darstellung der Clearphase*
Typ	Verbesserung
Bearbeiter	Darian Alves
Lösung	Karten werden in der Clearphase auf den Ablagestapel gelegt. Dies wird animiert dargestellt.
Ticketname	*SWP2019B-251 - Als Nutzer möchte ich meine Karte ausspielen können, damit sie ihre Aktion ausführen kann.*
Typ	Story
Bearbeiter	Devin Schlüter
Lösung	Beim Anklicken einer Handkarte wird diese groß dargestellt und der User hat die Möglichkeit eine Aktionskarte über einen Button zu spielen. Die Großansicht der Karte, kann für alle Handkarten durch „zurück“ beendet werden.
Ticketname	*SWP2019B-253 - Als User möchte ich, dass mir der Kauf einer Karte auch visuell angezeigt wird*
Typ	Story
Bearbeiter	Anna Hiemenz
Lösung	Bei erfolgreichem kaufen einer Karte, werden die Geldkarten ausgespielt und die gekaufte Karte dem Ablagestapel hinzugefügt. War die gekaufte Karte, die letzte zu kaufende Karte ihrer Art, wird diese verdunkelt. War der Kauf erfolglos, erscheint eine Fehlermeldung.
Unteraufgaben	• *SWP2019B-254 - Stapel in der Mitte: Bild entfernen, falls keine Karte mehr vorhanden ist* • *SWP2019B-255 - Bei erfolgreichen Kauf, müssen die Geldkarten des Spielers abgezogen werden* • *SWP2019B-256 - Fehlermeldung, wenn der Spieler nicht genug Geld hat*
Ticketname	*SWP2019B-262 - Entfernen von Verweisen auf /css/swp.css*
Typ	Bug
Bearbeiter	Timo Siems

Lösung	Bug wurde durch Umstelltung auf „global.css“ behoben.
Ticketname	*SWP2019B-266 - Als User muss ich meinen Ablagestapel bei Bedarf auch als Nachziehstapel nutzen können.*
Typ	Neue Funktion
Bearbeiter	Fenja Bauer, Anna Hiemenz
Lösung	Über einen Kreis mit einem Label wird dem User die Anzahl der Karten im Nachziehstapel angezeigt. Ist der Nachziehstapel leer, wird der Ablagestapel geleert und der Nachziehstapel mit diesen Karten wieder befüllt.
Ticketname	*SWP2019B-272 - Bereitbutton in der Lobby wird beim Verlassen der Lobby nicht aktualisiert*
Typ	Bug
Bearbeiter	Julia Debkowski
Lösung	Der Bug wurde behoben. Der User kann nun nach dem Verlassen und wieder betreten einer Lobby auf „bereit“ klicken.
Ticketname	*SWP2019B-273 - Als Nutzer möchte ich ein grafisch ansprechendes Lobbyfenster haben*
Typ	Verbesserung
Bearbeiter	Fenja Bauer, Ferit Askin
Lösung	Die graphische Darstellung des Lobbyfensters wurde überarbeitet.
Ticketname	*SWP2019B-276 - Als Nutzer möchte ich beim klicken auf den Lobby erstellen-Button, das das sich öffnene Fenster im Focus bleibt, bis es wieder geschlossen wird*
Typ	Verbesserung
Bearbeiter	Paulina Kowalska
Lösung	Das „Lobby erstellen“- und „Lobby beitreten“-Fenster wurden in JavaFX umgewandelt und neu gestaltet. Dieses Fenster bleibt im Fokus und schließt sich nur durch eine Eingabe, durch „Zurück“ oder das X oben rechts.
Ticketname	*SWP2019B-278 - Als User möchte ich, dass serverseitig die Anzahl der Karten auf dem Nachziehstapel an den Client übergeben wird*
Typ	Story
Bearbeiter	Julia Debkowski
Lösung	Vom Server wird eine Nachricht über die aktuelle Anzahl an Karten auf dem Nachziehstapel an den Client geschickt.
Ticketname	*SWP2019B-282 - Als Host im Spiel möchte ich, dass ich die Karten für das Spiel in der Lobby auswählen kann, wenn ich keine aussuche, werden diese automatisch verteilt*
Typ	Story

Bearbeiter	Anna Hiemenz, Fenja Bauer
Lösung	Als Host einer Lobby, hat der User über einen Button die Möglichkeit, zehn Königreichkarten für das Spiel auszuwählen. Wählt er weniger als 10 bzw. gar keine aus, werden die (fehlenden) Königreichkarten per Zufall für das Spiel ausgewählt.

Ticketname	*SWP2019B-286 - Einstellungen Bugfixes*
Typ	Bug
Bearbeiter	Julia Debkowski
Lösung	DeleteAccountView wurde angepasst. Es wird keine Exception mehr geworfen, wenn der User in einer Lobby ist und seine Daten ändert.

Ticketname	*SWP2019B-289 - HTML Dateien der Spielanleitung erneuern*
Typ	Bug
Bearbeiter	Timo Siems
Lösung	Die Formatierung der Spieleanleitung wurde erneuert, da diese etwas verschoben war.

Ticketname	*SWP2019B-291 - Exception bei der Stummschaltung der Musik*
Typ	Bug
Bearbeiter	Timo Siems
Lösung	Bug wurde behoben.

Ticketname	*SWP2019B-296 - Kartenbilder ergänzen und Bilder korrigieren*
Typ	Aufgabe
Bearbeiter	Alexander Linke
Lösung	Es wurden einige Kartenbilder ergänzt und die falschen Bilder durch richtige ersetzt.

Ticketname	*SWP2019B-297 - WARNING: Resource „css/RegistrationViewPresenter.css“ not found*
Typ	Bug
Bearbeiter	Timo Siems
Lösung	Der Verweis auf die RegistrationViewPresenter.css wurde im RegistrationPresenter, SceneManager und in der RegistrationView.fxml gelöscht, da dies zu einer Exception geführt hat.

Folgende Duplikate sind in diesem Sprint aufgetreten
- Lösung: **Duplikat**:

- SWP2019B-264 - Als User möchte ich den jeweiligen Ablagestapel anderer Spieler sehen können.
- SWP2019B-271 - Als Nutzer möchte ich, dass die ClearPhase umgesetzt wird, damit die Karten in den Ablagestapel kommen

- SWP2019B-275 - Als Spieler will ich angezeigt bekommen wie viel Geld ich in der Kaufphase zur Verfügung habe

Folgende Tickets wurden **geschlossen**, da diese unter anderem bereits (in anderen Branches) bearbeitet wurden:

- SWP2019B-263 - NullPointerException im EventBus
- SWP2019B-270 - Als Nutzer möchte ich, dass sich die Phasen automatisch ändern, damit der Spielablauf schneller ist.

Einige Tickets (mit ihren Unteraufgaben) wurden mit in den Sprint 8 übernommen, da ihre Bearbeitung noch nicht abgeschlossen war:

- SWP2019B-154 - Als Nutzer möchte ich den Chat stummschalten können.
- SWP2019B-214 - Als Nutzer möchte ich, meine zusätzliche Punkte zum aktuellen Spiel angezeigt bekommen
- SWP2019B-215 - Als Nutzer möchte ich meine Gegner sehen können
- SWP2019B-252 - Kartenlogik, Serverseitige implementierung
- SWP2019B-265 - Als User möchte ich die Hände und den Ablagestapel aller meiner Gegner sehen.
- SWP2019B-267 - Als Nutzer möchte ich, dass ich Benachrichtigungen zum Spiel ausschalten kann
- SWP2019B-274 - Als Spieler will ich das Geldkarten und Aktionskarten je nach Phase unterschiedlich behandelt werden
- SWP2019B-279 - Als Spieler möchte ich, dass ich angezeigt bekomme, wie viele Aktionen, Käufe, Geld ich noch habe
  - SWP2019B-223 - Als Nutzer möchte ich wissen in welcher Phase ich mich derzeit befinde
- SWP2019B-292 - Shortcut für Buttons im Game die außerhalb des Bildschirms landen
- SWP2019B-300 - Als User möchte ich eine einheitliche Sprache innerhalb des Spiels haben
- SWP2019B-303 - In der Lobby wird die MaxSpieler - Combobox falsch gesetzt
- SWP2019B-304 - Als User möchte ich, dass die Karten, die man sich im Spiel groß anschaut nicht aufeinander gelegt werden, sondern das ‚untere'sich automatisch schließt
- SWP2019B-306 - Lobby verlassen NullPointerException

5.6.9 Sprint 8

Ticketname	*SWP2019B-215 - Als Nutzer möchte ich meine Gegner sehen können*
Typ	Story
Bearbeiter	Alexander Linke
Lösung	Die gegnerischen Spieler werden in einer Listview angezeigt. Diese wird beim Verlassen eines Mitspielers aktualisiert. Außerdem wird ein Bild des Mitspielers, sowie sein Name, auf dem Spielfeld angezeigt.

Ticketname	*SWP2019B-252 - Kartenlogik, Serverseitige implementierung*
Typ	Aufgabe
Bearbeiter	Ferit Askin, Julia Debkowski, Keno Oelrichs Garcia
Lösung	Serverseitig wurde die Aktionskartenlogik aus der JSON implementiert. Noch nicht implementierte Karten wurden aus der Basispack.json in eine backup.json verschoben.

Ticketname	*SWP2019B-265 - Als User möchte ich die Hände und den Ablagestapel aller meiner Gegner sehen.*
Typ	Neue Funktion
Bearbeiter	Devin Schlüter, Anna Hiemenz
Lösung	Die Hände und Ablagestapel der Gegner werdem dem User auf dem Spielfeld angezeigt. Außerdem wird die Clearphase und das Auspielen einer Karte für alle Spieler dargestellt.

Ticketname	*SWP2019B-267 - Als Nutzer möchte ich, dass ich Benachrichtigungen zum Spiel ausschalten kann*
Typ	Verbesserung
Bearbeiter	Keno Schwedler
Lösung	Nutzer können die Benachichtigungen zum Spiel ausschalten.

Ticketname	*SWP2019B-268 - Als Nutzer möchte ich den Spielverlauf im Chat angezeigt bekommen.*
Typ	Story
Bearbeiter	Keno Oelrichs Garcia, Fenja Bauer, Timo Siems
Lösung	Im Chat wird dem User der Spielverlauf angezeigt.

Ticketname	*SWP2019B-279 - Als Spieler möchte ich, dass ich angezeigt bekomme, wie viele Aktionen, Käufe, Geld ich noch habe*
Typ	Story
Bearbeiter	Rike Hochheiden
Lösung	Es wurde ähnlich zu dem Online-Spiel eine Anzeige eingefügt, die Informationen zu der Anzahl der Aktionen, der Käufe, zum Geld und zu der Phase enthält.

Unteraufgaben

- *SWP2019B-223 - Als Nutzer möchte ich wissen in welcher Phase ich mich derzeit befinde*

Ticketname	*SWP2019B-288 - Bereitstatus wird zurückgesetzt, wenn ein anderer Spieler die Lobby verlässt*
Typ	Bug
Bearbeiter	Marvin Thate
Lösung	Bug wurde behoben.

Ticketname	*SWP2019B-292 - Shortcut für Buttons im Game die außerhalb des Bildschirms landen*
Typ	Neue Funktion
Bearbeiter	Marvin Thate
Lösung	Es wurden Shortcuts für „Lobby erstellen“, „Aufgeben“ und „Skippen“ eingefügt.

Ticketname	*SWP2019B-298 - KOMFORT: Sound beim Spielstart*
Typ	Aufgabe
Bearbeiter	Timo Siems
Lösung	Sound zu Spielbeginn wurde durch ein Ticket ausgestellt. Dies wurde behoben und der Sound wird nun wieder bei Spielbeginn abgespielt.

Ticketname	*SWP2019B-300 - Als User möchte ich eine einheitliche Sprache innerhalb des Spiels haben*
Typ	Story
Bearbeiter	Timo Siems
Lösung	Fehlermeldungen und LOG-Nachrichten wurden auf deutsch umgestellt, wenn sie noch auf Englisch waren.

Ticketname	*SWP2019B-302 - Als Nutzer möchte ich, dass der Lobby erstellen Button verschwindet, sobald ich dieser Lobby beigetreten bin*
Typ	Story
Bearbeiter	Rike Hochheiden
Lösung	„Lobby beitreten“-Button wird auf nicht anklickbar und ausgegraut gesetzt, wenn der User in der Lobby ist, die Lobby im Spiel ist oder die sich bereits die maximale Anzahl an Spielern in der Lobby befindet.

Ticketname	*SWP2019B-303 - In der Lobby wird die MaxSpieler - Combobox falsch gesetzt*
Typ	Bug
Bearbeiter	Ferit Askin
Lösung	Nur noch der Owner sieht die MaxPlayer-ComboBox und kann diese setzen. Der Owner bekommt eine fehlermeldung, wenn er die MaxPlayer niedriger als die momentane Spieleranzahl, in der Lobby, setzen will.

Ticketname	*SWP2019B-304 - Als User möchte ich, dass die Karten, die man sich im Spiel groß anschaut nicht aufeinander gelegt werden, sondern das ‚untere‘sich automatisch schließt*

Typ	Verbesserung
Bearbeiter	Anna Hiemenz, Fenja Bauer
Lösung	Karten aus dem Shop werden mit einem Rechtsklick vergrößert und mit einem Linksklick gekauft. Wird der Hintergrund angeklickt, wird die Großansicht wieder entfernt.

Ticketname	*SWP2019B-306 - Lobby verlassen NullPointerException*
Typ	Bug
Bearbeiter	Darian Alves
Lösung	Bug wurde behoben, in dem die LobbyLeaveRequest nur noch einmal versendet wird. Diese wurde zuvor zweimal gesendet und führte zum Fehler.

Ticketname	*SWP2019B-307 - Passwordüberprüfung bei privaten Lobbys schlägt fehl*
Typ	Bug
Bearbeiter	Paulina Kowalska, Fenja Bauer, Keno Oelrichs Garcia
Lösung	Der Bug wurde behoben.

Ticketname	*SWP2019B-311 - Als Nutzer möchte ich, dass wenn ich im Spiel nicht an der Reihe bin, der „Skip Phase“-Button nicht drückbar ist*
Typ	Verbesserung
Bearbeiter	Timo Siems
Lösung	„Skip Phase“-Button wird ausgeblendet, wenn der User nicht selber an der Reihe ist.

Ticketname	*SWP2019B-317 - Clearphase hat keine Animation und die Karten auf der Hand werden zwei mal gezogen*
Typ	Bug
Bearbeiter	Darian Alves
Lösung	Die Animation wurde angepasst.

Ticketname	*SWP2019B-321 - Als Nutzer möchte ich keine Nachrichten erhalten, wenn ich nicht angemeldet bin.*
Typ	Neue Funktion
Bearbeiter	Keno Schwedler
Lösung	Wenn der User nicht angemeldet ist, bekommt er keine Nachrichten mehr zugesendet.

Ticketname	*SWP2019B-324 - Der User wird nicht ausgeloggt wenn man im Spiel den Client schließt*
Typ	Bug
Bearbeiter	Marvin Thate
Lösung	User wird durch ein „hardLogout“ ausgeloggt, wenn er das Fenster über das X schließt.

Ticketname	*SWP2019B-325 - Dem User in der Clearphase wird eine falsche Kartenanzahl beim nächsten user angezeigt*
Typ	Bug
Bearbeiter	Rike Hochheiden
Lösung	Fehler konnte durch eine Abfrage behoben werden bei der geprüft wird, ob die gesendete Hand die initale Hand ist oder nicht.

Ticketname	*SWP2019B-328 - Als Nutzer möchte ich, das ich bestätigen muss, wenn ich die Lobby über das X in der TabView verlasse*
Typ	Verbesserung
Bearbeiter	Marvin Thate
Lösung	Beim Schließen eines Lobby-/GameTabs wird der User über ein Fenster gefragt, ob er den Tab wirklich schließen will. Er muss dieses bestätigen, damit der Tab geschlossen wird.

Ticketname	*SWP2019B-331 - Aktionszähler muss bei dem Spielen einer Aktion reduziert werden*
Typ	Bug
Bearbeiter	Julia Debkowski
Lösung	Hat der User keine Aktionen mehr, wechselt er automatisch in die Kaufphase. Bei einer Play-/BuycardRequest wird überprüft, ob der User noch Aktionen bzw. Käufe hat.

Ticketname	*SWP2019B-342 - In der Aktionsphase sollen keine Karten gekauft werden können & in der Kaufphase keine Karten gespielt*
Typ	Story
Bearbeiter	Julia Debkowski
Lösung	Bei einer Buy-/PlayCardRequest wird geprüft, ob sich der User in der Kauf- bzw. Aktionsphase befindet.

Ticketname	*SWP2019B-344 - Clientseitige Unterscheidung Aktions- und Kaufphase*
Typ	Verbesserung
Bearbeiter	Anna Hiemenz, Paulina Kowalska
Lösung	Aktions- bzw. Geldkarten werden ausgegraut wenn der User sich nicht in der passenden Phase (Aktions- bzw. Kaufphase) befindet

Für folgende User Stories wurde entschieden, sie nicht umzusetzen
- Lösung: **Won't do**:

- SWP2019B-154 - Als Nutzer möchte ich den Chat stummschalten können.
- SWP2019B-274 - Als Spieler will ich das Geldkarten und Aktionskarten je nach Phase unterschiedlich behandelt werden
- SWP2019B-295 - Als Nutzer möchte ich, das beim erhalt einer ChooseCardRequest Karten auswählen kann und eine Antwort an den Server geschickt wird

- SWP2019B-308 - Die Änderung des Usernames funktioniert nicht ordnungsgemäß
- SWP2019B-309 - Implementierung der restlichen Spielkarten
- SWP2019B-343 - Einzelne Geldkarten sollen nicht mehr ausgespielt werden können

Folgende Tickets wurden **geschlossen**, da diese unter anderem bereits bearbeitet wurden:

- SWP2019B-310 - Fehlermeldung beim Erstellen einer Lobby mit gleichen Namen taucht bei allen Usern auf

Einige Tickets (mit ihren Unteraufgaben) wurden mit in den Sprint 9 übernommen, da ihre Bearbeitung noch nicht abgeschlossen war:

- SWP2019B-59 - Als Nutzer möchte ich die Möglichkeit haben einen Bot in der Lobby hinzuzufügen
- SWP2019B-185 - DAO-Pattern & SQL Umstellung
- SWP2019B-214 - Als Nutzer möchte ich, meine zusätzliche Punkte zum aktuellen Spiel angezeigt bekommen
- SWP2019B-259 - Als User möchte ich, dass ein Bot eine Spezialisierung eines Spielers ist
- SWP2019B-269 - Als Spieler möchte ich, dass der Bot an einem Spiel teilnehmen kann.
- SWP2019B-287 - Nutzer kann seinen Account während des Spiels löschen.
- SWP2019B-299 - Als User möchte ich beim Erhalt einer OptionalActionRequest entscheiden können, ob die Aktion ausführt werden soll.
- SWP2019B-315 - Als Nutzer möchte ich die Aktionen der Karten Clientseitig nachvollziehen, damit ich das Spiel spielen kann.
  - SWP2019B-295 - Als Nutzer möchte ich, das beim erhalt einer ChooseCardRequest Karten auswählen kann und eine Antwort an den Server geschickt wird
  - SWP2019B-335 - Als Nutzer möchte ich Karten zum abwerfen auswählen können, damit Karten wie Kapelle, Keller und Festmahl für mich funktionieren
  - SWP2019B-336 - Als User möchte ich, dass bei Erhalt einer RemoveActionCardMessage die angegebene Karte aus der Aktionszone des Spielers entfernt wird
- SWP2019B-326 - Als User möchte ich Karten im Aktionsfeld groß angezeigt bekommen können
- SWP2019B-339 - Als Nutzer will ich die Einstellungen nur im Hauptmenü öffnen können
- SWP2019B-341 - Messages sollen nur an die zugehören Nutzer gesendet werden
- SWP2019B-347 - Gegner werden im Spiel nicht mehr angezeigt

5.6.10 Sprint 9

Ticketname	*SWP2019B-185 - DAO-Pattern & SQL Umstellung*
Typ	Verbesserung
Bearbeiter	Keno Schwedler, Ferit Askin
Lösung	Das DAO-Pattern wurde für die User, die Lobby und das Game implementiert. SQL-Umstellung ist erfolgt.

Ticketname	*SWP2019B-259 - Als User möchte ich, dass ein Bot eine Spezialisierung eines Spielers ist*
Typ	Story
Bearbeiter	Fenja Bauer, Ferit Askin
Lösung	Über einen Butten kann der Host einer Lobby Bots hinzufügen. Diese können auch wieder entfernt werden. Der Bot bekommt einen zufälligen Namen und ist in der Lage seine Phase zu skippen.

Ticketname	*SWP2019B-260 - Als Spieler möchte ich die Spielzüge meiner Gegner sehen*
Typ	Story
Bearbeiter	Anna Hiemenz
Lösung	Animation wurde angepasst, sodass die Spielzüge der Gegner dem User angezeigt werden.

Ticketname	*SWP2019B-299 - Als User möchte ich beim Erhalt einer OptionalActionRequest entscheiden können, ob die Aktion ausführt werden soll.*
Typ	Story
Bearbeiter	Darian Alves
Lösung	Der User kann bei Erhalten einer OptionalActionRequest eine Entscheidung über einen „Ja“- bzw. „Nein“-Button treffen. Diese wird an den Server zurückgegeben.

Ticketname	*SWP2019B-315 - Als Nutzer möchte ich die Aktionen der Karten Clientseitig nachvollziehen, damit ich das Spiel spielen kann.*
Typ	Story
Bearbeiter	Anna Hiemenz, Fenja Bauer
Lösung	Bei erhalten einer ChooseCardRequest werden die Karten durch einen Effekt verdunkelt, welche vom User nicht gewählt werden können. Wird eine auswählbare Karte gewählt, wird die Karte an den Server gesendet und der Effekt der nicht auswählbaren Karten entfernt.

Unteraufgaben

- *SWP2019B-295 - Als Nutzer möchte ich, das beim erhalt einer ChooseCardRequest Karten auswählen kann und eine Antwort an den Server geschickt wird*
- *SWP2019B-335 - Als Nutzer möchte ich Karten zum abwerfen auswählen können, damit Karten wie Kapelle, Keller und Festmahl für mich funktionieren*
- *SWP2019B-336 - Als User möchte ich, dass bei Erhalt einer RemoveActionCardMessage die angegebene Karte aus der Aktionszone des Spielers entfernt wird*

Ticketname	*SWP2019B-326 - Als User möchte ich Karten im Aktionsfeld groß angezeigt bekommen können*
Typ	Verbesserung
Bearbeiter	Alexander Linke
Lösung	Das entfernen des Klick-events wurde gelöscht und dafür eine Abfrage hinzugefügt, welche überprüft, ob sich eine Aktionskarte noch auf der Hand befindet. Somit können die Karten in der Aktionszone mit rechts angeklickt werden.

Ticketname	*SWP2019B-328 - Als Nutzer möchte ich, das ich bestätigen muss, wenn ich die Lobby über das X in der TabView verlasse*
Typ	Verbesserung
Bearbeiter	Marvin Thate
Lösung	Es wurde eine Abfrage beim Schließen des Tabs über das X im Tab eingeführt.

Ticketname	*SWP2019B-329 - Beschriftung des Lobbychats ist schwer lesbar*
Typ	Bug
Bearbeiter	Keno Oelrichs Garcia
Lösung	Die Beschriftung des Lobbychats wurde optisch angepasst, damit sie besser zu lesen ist.

Ticketname	*SWP2019B-339 - Als Nutzer will ich die Einstellungen nur im Hauptmenü öffnen können*
Typ	Story
Bearbeiter	Julia Debkowski
Lösung	Es ging um die Problematik, das es zu Fehlern kam, wenn der User seinen Namen geändert hat, wenn er in einem Spiel war. Es wurde beschlossen, dass das Einstellungsfenster sich weiterhin außerhalb des Hauptmenüs öffnen lässt, jedoch nicht mehr die Möglichkeit besteht seinen Usernamen zu ändern, wenn man in einer Lobby oder einem Spiel ist.

Ticketname	*SWP2019B-340 - Als User möchte ich, dass bei einer UpdateCardCounterMessage die Anzahl verfügbarer Karten aktualisiert bzw. die Karte(n) auf dem Spielfeld ausgegraut wird, falls der Wert 0 ist.*
Typ	Story
Bearbeiter	Paula Kowalska
Lösung	Bei Erhalten einer UpdateCardCounterMessage werden Karten auf dem Spielfeld ausgegraut, wenn sie nicht mehr vorhanden sind.

Ticketname	*SWP2019B-341 - Messages sollen nur an die zugehören Nutzer gesendet werden*
Typ	Verbesserung
Bearbeiter	Keno Schwedler
Lösung	Nachrichten werden an die richtigen User gesendet. Nicht angemeldete User erhalten keine Nachrichten mehr.

Ticketname	*SWP2019B-345 - Als GameOwner will ich die aktuelle Max Player Value sehen, wenn ich eine ungültige eingabe getätigt habe.*
Typ	Aufgabe
Bearbeiter	Ferit Askin
Lösung	Wurde vom Owner eine ungültige Eingabe der maxPlayer gemacht, wird diese wieder zurückgesetzt, auf den usrprünglichen Wert.

Ticketname	*SWP2019B-349 - Als Nutzer möchte ich, das Kartenbewegungen der MoveCardMessage auf dem Spielfeld dargestellt werden.*
Typ	Neue Funktion
Bearbeiter	Anna Hiemenz
Lösung	Verhalten bei Erhalten einer MoveCardMessage wurde implementiert. Die Karten bewegen sich entsprechend der gewollten Animationen. Animation für den Müll ergänzt.

Ticketname	*SWP2019B-350 - Spielerliste im Spiel wird nicht angezeigt*
Typ	Bug
Bearbeiter	Timo Siems, Rike Hochheiden
Lösung	Die Spielerliste im Spiel wird wieder korrekt angezeigt. Eine evtl. geworfene NullPointerException wird abgefangen. Zur Unteraufgabe: Es hat sich heraus gestellt, das die Datei defekt ist. Durch ersetzen der Datei konnte der Fehler behoben werden.
Unteraufgaben	• *SWP2019B-301 - Laden der Trattatello.ttf schlägt OS abhängig fehl*

Ticketname	*SWP2019B-351 - in der Aktionsphase kann ich den Button alle Geldkarten spielen drücken*
Typ	Bug
Bearbeiter	Paulina Kowalska

Lösung	In der Aktionsphase können keine Geldkarten über den „Alle Geldkarten spieln“ ausgespielt werden.
Ticketname	*SWP2019B-352 - Der Chat wird im Hauptmenü nicht auf der vollen Höhe angezeigt*
Typ	Bug
Bearbeiter	Keno Oelrichs Garcia
Lösung	Der Bug wurde behoben.
Ticketname	*SWP2019B-353 - Der Sound kann nicht im Hauptmenü eingestellt werden, sondern nur im Login-View*
Typ	Bug
Bearbeiter	Rike Hochheiden
Lösung	Das Einstellungsfenster hat ein SoundIcon bekommen, über das der User den Sound an und aus stellen kann. Das SoundIcon in der Login-View und RegistrationView wurde entfernt.
Ticketname	*SWP2019B-355 - Einstellungsfenster hat keinen Rahmen und lässt sich nicht verschieben*
Typ	Bug
Bearbeiter	Timo Siems
Lösung	Der Bug wurde behoben.
Ticketname	*SWP2019B-356 - Das Account-löschen-Fenster enthält zu kleine Schrift und kein Hintergrund*
Typ	Bug
Bearbeiter	Timo Siems
Lösung	Die Schriftgröße und der Hintergrund wurden angepasst.
Ticketname	*SWP2019B-358 - Java Klassen zusammen fassen*
Typ	Verbesserung
Bearbeiter	Timo Siems
Lösung	Es wurden drei Java Klassen (DeckLayoutContainer, DiscardPileLayoutContainer und HandcardsLayoutContainer) zusammen gefasst, die fast identisch waren.
Ticketname	*SWP2019B-361 - Beim Aufgeben erscheint eine Exception, dass die View nicht geladen werden konnte*
Typ	Bug
Bearbeiter	Keno Schwedler
Lösung	Der Bug wurde behoben.
Ticketname	*SWP2019B-368 - Als Nutzer möchte ich random Profilbilder angezeigt bekommen.*
Typ	Verbesserung

Bearbeiter	Ferit Askin
Lösung	Es wurden Bilder hinzugefügt und die Variationen für die Zuweisung implementiert.

Ticketname	*SWP2019B-373 - Als User möchte ich, dass die Anzahl der Karten auf dem Spielfeld ähnlich zu den Siegpunktkarten angezeigt wird*
Typ	Verbesserung
Bearbeiter	Rike Hochheiden
Lösung	Die Anzahl der Karten aus dem Shop, sowie der Geldkarten werden nun genauso angezeigt, wie es bei den Siegpunkten bereits der Fall war.

Ticketname	*SWP2019B-376 - JavaDoc bei Tests*
Typ	Aufgabe
Bearbeiter	Timo Siems
Lösung	JavaDoc wurde bei einigen Test ergänzt, bei denen es gefehlt hat.

Ticketname	*SWP2019B-377 - Effekt auf Geldkarten wird in der Aktionsphase nicht wieder gesetzt*
Typ	Bug
Bearbeiter	Julia Debkowski
Lösung	Nach dem Senden einer ChooseCardReponse wird der Effekt (nicht auswählbare Karten sind verdunkelt) wieder entfernt und stattdessen werden, je nach Phase, entweder die Aktions- oder die Geldkarten wieder verdunkelt.

Folgende Tickets wurden **geschlossen**, da diese teilweise bereits bearbeitet wurden oder unwichtig geworden sind:

- SWP2019B-59 - Als Nutzer möchte ich die Möglichkeit haben einen Bot in der Lobby hinzuzufügen
- SWP2019B-269 - Als Spieler möchte ich, dass der Bot an einem Spiel teilnehmen kann.
- SWP2019B-319 - Fehlermeldung beim Lobby erstellen überarbeiten
- SWP2019B-347 - Gegner werden im Spiel nicht mehr angezeigt
- SWP2019B-359 - NullPointer bei der onBuyCardMessage
- SWP2019B-363 - Als Nutzer möchte ich, dass der Client weniger Arbeitsspeicher beansprucht
- SWP2019B-366 - Serverseitiges Refactoring

Einige Tickets wurden mit in den Sprint 10 übernommen, da ihre Bearbeitung noch nicht abgeschlossen war:

- SWP2019B-214 - Als Nutzer möchte ich, meine zusätzliche Punkte zum aktuellen Spiel angezeigt bekommen
- SWP2019B-287 - Nutzer kann seinen Account während des Spiels löschen.
- SWP2019B-334 - Karte kann nicht gekauft werden, obwohl genug Geld vorhanden ist.
- SWP2019B-354 - Wenn eine Chatnachricht empfangen wird, wird kein Sound mehr gespielt
- SWP2019B-362 - Als Nutzer möchte ich, dass die Bots auch Karten spielen können
- SWP2019B-379 - Relative Pfade anpassen

5.6.11 Sprint 10

Ticketname	*SWP2019B-214 - Als Nutzer möchte ich, meine zusätzliche Punkte zum aktuellen Spiel angezeigt bekommen*
Typ	Story
Bearbeiter	Mahmoud Haschem, Ferit Askin
Lösung	Dem User wird seine momentane Punkteanzahl angezeigt.
Ticketname	*SWP2019B-287 - Nutzer kann seinen Account während des Spiels löschen.*
Typ	Bug
Bearbeiter	Alexander Linke, Timo Siems, Paulina Kowalska
Lösung	Der User kann während eines Spiels seinen Account nicht mehr löschen.
Ticketname	*SWP2019B-312 - Als Nutzer möchte ich, dass Neue Fenster immer in der Mitte des übergeordneten Fensters angezeigt werden, um einen besseren Überblick zu behalten und schnellere Interaktionen zu ermöglichen*
Typ	Verbesserung
Bearbeiter	Keno Oelrichs Garcia
Lösung	Neue Fenster werden mittig zu ihrem übergeordneten Fenster angezeigt.
Ticketname	*SWP2019B-354 - Wenn eine Chatnachricht empfangen wird, wird kein Sound mehr gespielt*
Typ	Bug
Bearbeiter	Keno Schwedler
Lösung	Bug wurde behoben.
Ticketname	*SWP2019B-362 - Als Nutzer möchte ich, dass die Bots auch Karten spielen können*
Typ	Verbesserung
Bearbeiter	Darian Alves
Lösung	Der Bot kann Karten kaufen und Aktionen spielen. Außerdem wartet der Bot bei den Aktionen etwas.

Ticketname	*SWP2019B-365 - Doku für die Rolle erstellt*
Typ	Aufgabe
Bearbeiter	Rike Hochheiden
Lösung	Rollenbeschreibung wurde von den jeweiligen Rolleninhaber erstellt.

Ticketname	*SWP2019B-370 - Als Nutzer möchte ich weitere Karten im Spiel haben*
Typ	Verbesserung
Bearbeiter	Julia Debkowski
Lösung	Fünf weitere Karten (Ratsversammlung, Hexe, Burggraben, Geldverleiher, Thronsaal) wurden implementiert.

Ticketname	*SWP2019B-371 - Aufsetzen der VM für die Projektabgabe.*
Typ	Aufgabe
Bearbeiter	Keno Oelrichs Garcia
Lösung	Es wurde eine VM für die Projektabgabe aufgesetzt.

Ticketname	*SWP2019B-374 - Dokumentation JavaFX & ControlFX und Spielanleitung*
Typ	Aufgabe
Bearbeiter	Devin Schlüter
Lösung	Die Dokumentation für JavaFX & ControlFX sowie eine Spieleanleitung wurden erstellt.

Ticketname	*SWP2019B-375 - Als Nutzer möchte ich einen Toilettenknopf haben, um das Spiel zu pausieren.*
Typ	Neue Funktion
Bearbeiter	Keno Schwedler
Lösung	Der Toilettenknopf startet eine Umfrage. Wird die Umfrage angenommen, gibt es eine 1-minütige Pause, die nur vom Anfragesteller frühzeitig abgebrochen werden kann.

Ticketname	*SWP2019B-378 - Glossar ergänzt*
Typ	Aufgabe
Bearbeiter	Rike Hochheiden
Lösung	Für die Dokumentation wurde ein Glossar angelegt. Begriffe wurden selbstständig von den Gruppenmitgliedern angelegt. Der Bearbeitende hat die Aufgabe, das Verweise auf das Glossar sowie das anfertigen von Glossareinträgen zu kontrollieren.

Ticketname	*SWP2019B-379 - Relative Pfade anpassen*
Typ	Bug
Bearbeiter	Rike Hochheiden
Lösung	Die Bildpfade wurden dahingehend angepasst, das sie relativ sind und es keine Probleme auf anderen OS bei der Anzeige mehr gibt.

Ticketname	*SWP2019B-380 - Karten Vorschau hinter dem Deck von dem linken Spieler auf dem Spielfeld fixen*
Typ	Verbesserung
Bearbeiter	Anna Hiemenz
Lösung	Kartenvorschau ist wieder im Vordergrund und das Layout vom Spielergebnis-Fenster wurde angepasst.

Ticketname	*SWP2019B-381 - Finaler Merge Develop → Master*
Typ	Aufgabe
Bearbeiter	Ferit Askin
Lösung	Der Develop-Branch wurde für die Abgabe in den Master-Branch gemergt.

Ticketname	*SWP2019B-382 - Als Nutzer möchte ich, das längere Info/ Server-Nachrichten mittig im Chat angezeigt werden*
Typ	Bug
Bearbeiter	Ferit Askin
Lösung	Namen der Bots wurde gekürzt. Nachrichten werden zentriert dargestellt.

Ticketname	*SWP2019B-383 - Ein Nutzer der das Spiel verlassen hat, wird nicht von dem Spielfeld entfernt.*
Typ	Verbesserung
Bearbeiter	Ferit Askin, Fenja Bauer, Paulina Kowalska
Lösung	Wenn ein Spieler aufgibt oder das Spiel verlässt, werden sein Icon und seine Karten nicht mehr angezeigt.

Ticketname	*SWP2019B-385 - dem User wird die falsche Kartenanzahl angezeigt, wenn er in der GameOverView nochmal angeklickt hat*
Typ	Bug
Bearbeiter	Rike Hochheiden
Lösung	Bei der Initalisierung der Karten auf dem Feld zu Spielbegin, wird geprüft ob sich in einem der Container für die Hände/Ablagestapel des Spieles Karten befinden und ggf. entfernt.

Ticketname	*SWP2019B-388 - MainMemoryBasedDataStore löschen und DatabaseBasedUserStore benutzen*
Typ	Verbesserung
Bearbeiter	Keno Schwedler
Lösung	Ticket wurde umgesetzt.

Ticketname	*SWP2019B-390 - Warnings isPresent checks*
Typ	Verbesserung
Bearbeiter	Timo Siems

Lösung	Mögliche get()-Fehler werden abgefangen. Es wurden Supress Warnings hinzugefügt.
Ticketname	*SWP2019B-391 - Chatnachrichten von der Länge begrenzen - Twitter Style*
Typ	Verbesserung
Bearbeiter	Ferit Askin, Rike Hochheiden
Lösung	Die Länge der erlaubten Chatnachrichten wurde begrenzt. Sind die Nachrichten zu lang, gibt es eine Fehlermeldung. Die Länge wurde nach Absprache auf 1.000 gesetzt, da es noch keine Möglichkeit gab, die Anzahl der eingegebenen Zeichen anzuzeigen. Nachdem die Möglichkeit der Anzeige durch Ticket 402 möglich gemacht wurde, wurde die Zeichenanzahl auf 160 Zeichen gesetzt.
Ticketname	*SWP2019B-394 - Bekommt ein Spieler durch eine Aktionskarte Karten auf die Hand, sind diese für die Gegner nicht verdeckt*
Typ	Bug
Bearbeiter	Anna Hiemenz
Lösung	Karten die ein Gegner durch eine Aktionskarte bekommt, werden der Hand verdeckt hinzugefügt. Es kann kein Bot mehr mit dem Spiel beginnen.
Ticketname	*SWP2019B-396 - Lobbyname auf 30 zeichen begrenzen*
Typ	Verbesserung
Bearbeiter	Ferit Askin
Lösung	Lobbynamen können nur noch 30 Zeichen enthalten. Beim Erstellen einer Lobby mit mehr Zeichen, gibt es eine Fehlermeldung.
Ticketname	*SWP2019B-399 - Fluchkarte wird nicht richtig gekauft.*
Typ	Verbesserung
Bearbeiter	Ferit Askin
Lösung	Fluchkarte wurde serverseitig bei der Kaufmethode nicht berücksichtigt/geladen. Dies wurde behoben, indem die Fluchkarte in der entsprechenden Methode zurück gegeben wurde.
Ticketname	*SWP2019B-401 - Punktzahl Spiel Optisch verschönern*
Typ	Verbesserung
Bearbeiter	Ferit Askin
Lösung	Die Anzeige der eigenen Punkte wurde optisch angepasst.
Ticketname	*SWP2019B-402 - Als Nutzer möchte ich, das mir die verbleibende Zeichenanzahl bei meiner Chatnachricht angezeigt wird*
Typ	Neue Funktion
Bearbeiter	Keno Oelrichs Garcia

Lösung	Dem User wird während des Tippens angezeigt, wie viele Zeichen er noch zur Verfügung hat. Überschreitet die Länge der nachricht, diese Anzahl, so wird dem User mitgeteilt, das die Nachricht zu lang ist.
Ticketname	*SWP2019B-404 - Spielanleitung aktualisieren*
Typ	Aufgabe
Bearbeiter	Timo Siems
Lösung	Shortcuts und die Anpassungen von Burggraben und Thronsaal wurden mit in die Anleitung aufgenommen.
Ticketname	*SWP2019B-405 - Registrierungsfenster - Schriftart und Positionierung der Schrift*
Typ	Bug
Bearbeiter	Timo Siems
Lösung	Bug wurde behoben
Ticketname	*SWP2019B-406 - Programm muss über Arbi laufen*
Typ	Aufgabe
Bearbeiter	Keno Oelrichs Gracia
Lösung	Das Projekt befindet sich auf der Arbi. Dort kann der Server des Spieles gestartet werden.
Ticketname	*SWP2019B-408 - Tests - Exception - Client, Common, Server*
Typ	Verbesserung
Bearbeiter	Timo Siems
Lösung	Tests wurden erstellt
Ticketname	*SWP2019B-410 - Test Deck und GameManagement*
Typ	Aufgabe
Bearbeiter	Julia Debowski
Lösung	Tests wurden erstellt
Ticketname	*SWP2019B-411 - Tests für Chat- und GameService (Client)*
Typ	Aufgabe
Bearbeiter	Anna Hiemenz
Lösung	Tests wurden erstellt
Ticketname	*SWP2019B-412 - Als User möchte ich ein gut getestetes Spiel haben*
Typ	Story
Bearbeiter	Fenja Bauer
Lösung	„PoopTest“ wurde hinzugefügt.
Ticketname	*SWP2019B-413 - Geldanzeige zurücksetzen, wenn Spieler nochmal klickt*

Typ	Bug
Bearbeiter	Rike Hochheiden
Lösung	Bug wurde behoben.

Ticketname	*SWP2019B-414 - Testabdeckung verbessern*
Typ	Verbesserung
Bearbeiter	Rike Hochheiden, Paulina Kowalska
Lösung	Test wurden erstellt für: DeleteAccountEvent, ChatExceptionMessage, NewChatMessage, Card, MoneyCard, ValueCard, CurseCard, BuyCard, BuyCardRequest, BuyCardMessage und GameService

Ticketname	*SWP2019B-415 - Der Bot spielt nur Kupferkarten*
Typ	Bug
Bearbeiter	Darian Alves
Lösung	Der Bot spielt nun auch andere Karten.

Folgende Tickets wurden aus Zeitgründen, weil sie bereits durch andere Tickets behoben wurden, es sich um Duplikate gehandelt hat oder der Fehler nicht reproduzierbar war geschlossen:

- SWP2019B-334 - Karte kann nicht gekauft werden, obwohl genug Geld vorhanden ist.
- SWP2019B-363 - Als Nutzer möchte ich, dass der Client weniger Arbeitsspeicher beansprucht
- SWP2019B-384 - Views müssen gelöscht werden.
- SWP2019B-392 - Zwei/Drei Bots spielen miteinander
- SWP2019B-393 - Als User möchte ich dass im Einstellungsfenster Chatmute und Soundmute angeordnet sind.
- SWP2019B-395 - Bugfix: Karten werden in neuem Game angezeigt und altem laufenden Spiel nicht mehr
- SWP2019B-397 - ClientDisconnectMessage einbauen/nutzen
- SWP2019B-407 - Chatzeichen auf 160 Zeichen runter setzen

In diesem Sprint wurde ebenfalls die Endpräsentation vorbereitet. Hierfür wurde der Ticket *SWP2019B-403 - Endpräsentation - Vorbereitung* angelegt. Dieser wurde von den vorstellenden Gruppenmitgliedern bearbeitet.

Kapitel 6

Ausblick

Auf Grund der zeitlichen Begrenzung des Projektes war es der Gruppe nicht möglich, sämtliche User Stories umzusetzen, die im Rahmen des Projektes möglich gewesen wären. Im folgenden Kapitel erfolgt die Erläuterung dieser.

Bei der Auflistung der User Stories wurde auf eine vernünftige Reihenfolge dieser geachtet. So ist unter anderem die Funktion von Icons wichtiger als die Individualisierung der Userprofile. Dabei wurden User Stories, die einem ähnlichen Gebiet angehören, in Abschnitten zusammengefasst.

6.1 Funktion von Icons

Zuallererst gibt es noch einige Funktionen, die im Spiel umgesetzt werden können, denn es gibt noch zwei Icons/Labels die der User anklicken kann, derzeit jedoch noch keine Funktion besitzen.

6.1.1 Mit der „Passwort vergessen“-Funktion in der LoginView.fxml soll der User sein Passwort zurücksetzen können.

Derzeit hat die Auswahlmöglichkeit „Passwort vergessen“ in der LoginView.fxml noch keinerlei Funktion. Sollte der User sein Passwort vergessen haben, ist er gezwungen sein Passwort zu erraten oder muss sich einen neuen Account anlegen.

6.1.2 Der User soll in der Lage sein, den Chatsound über die Einstellungen an- bzw. ausstellen zu können.

Der User hat momentan nach Einloggen nur die Möglichkeit, den gesamten Sound des Spieles auszuschalten. Das „SoundIcon“ besitzt lediglich die Funktion sein Aussehen zu ändern, es kann jedoch noch nicht die Chatbenachrichtigungen ausschalten.

6.2 Usability

Auch die Benutzerfreundlichkeit kann noch weiter verbessert werden.

6.2.1 Die Zeit des „Kloknopfes“ soll realisitischer (3 Minuten) sein.

Die Zeit für die „Kloknopf“ ist mit einer Minuten zum jetzigen Zeitpunkt sehr knapp bemessen und könnte noch angepasst werden, da man in der Regel eine längere Pause benötigt.

6.2.2 Der User soll in der Lage sein, die Größe des Spielfensters anzupassen.

Derzeit besteht noch die Problematik mit der Bildschirmgröße. Diese besteht daraus, dass die Größen fest gecodet sind. Um dies zu lösen, könnten die Fenstergrößen variabel gestaltet werden, sodass der User die Fenstergröße individuell anpassen kann und alle User ein optimales Spieleerlebnis haben können.

6.3 Spieleerlebnis

6.3.1 Weitere Karten sollen für das Spiel implementiert werden.

Um das Spieleerlebnis noch mehr zu verbessern, könnten noch weitere Spielkarten für das Spiel implementiert werden. Derzeit ist nur eine Auswahl von Spielkarten implementiert.

6.3.2 Der Host soll die Möglichkeit haben, ein Pack für das Spiel auszuwählen.

In Rahmen der Implementierung von weiteren Karten wäre auch eine Packauswahl eine Überlegung wert. Hierbei hätte der Host eines Spieles die Möglichkeit nicht nur einzelne Karten für das Spiel, sondern auch ein bereits zusammengestelltes Pack auszuwählen.

6.4 Filter

6.4.1 Spam soll im Chat eingedämmt werden.

In einigen Chats kann es zu Spam von Usern kommen. Um dies zu verhindern ist die Einführung eines Spamfilter von Nöten. Der Filter soll verhindern, dass der Chat zugespamt wird. Dabei könnte er überprüfen, ob ein User eine Nachricht mehrmals hintereinander absenden will und dies ggf. verhindern. Außerdem könnte mit einem Spamfilter verhindert werden, dass ein User

innerhalb kürzester Zeit mehrere Nachrichten im Sekundentakt absendet. Es sind auch noch weitere Spamfilter möglich.

6.4.2 User sollen in der Lage sein, Beleidigungen im Chat zu melden.

Da es leider auch vorkommen kann, dass ein User einen anderen User beleidigt, sollen die User in der Lage sein, Nachrichten, die sie als Beleidigung auffassen, als solche zu melden. Der Admin hat dadurch die Möglichkeit, User zu verwarnen oder bei wiederholter Beleidigung im Chat auch (zeitlich) zu sperren.

6.5 Individualisierung

6.5.1 Der User soll in der Lage sein, ein eigenes Profilbild auszuwählen.

Derzeit bekommen die Spieler einer Lobby im Spiel dem Zufall nach ein Profilbild zugeordnet. Dies könnte angepasst werden, indem der User bei der Registrierung ein Profilbild hochladen oder eins der bereitgestellten Profilbilder auswählen kann. Dies könnte der User zu einem späteren Zeitpunkt über die Einstellungen ändern.

6.5.2 Der User soll in der Lage sein eine eigene Freundesliste anzulegen.

Über eine Freundesliste hätte der User die direkte Möglichkeit zu sehen, ob Mitspieler, die er kennt, online sind. Dies könnte über eine Freundesliste realisiert werden, in der der Status der Freunde angezeigt wird. Für die Nutzung dieser Freundesliste muss der User die Möglichkeit haben, anderen Usern eine Freundesanfrage zu schicken, die akzeptiert werden muss, bevor der entsprechende User in der Freundesliste auftaucht.

6.6 Privater Chat

6.6.1 Der User soll neben dem globalen Chat und dem Lobbychat auch einen privaten Chat nutzen können.

Möchte ein User sich mit einem anderen User unterhalten, soll der User die Möglichkeit besitzen, dies über einen privaten Chat zu tun, sodass die anderen User nicht mitlesen können.

6.6.2 Der User soll die Möglichkeit besitzen, einen anderen User zu blocken.

Es kann unter Umständen vorkommen, dass ein User einen anderen User über einen privaten Chat stört. Dabei muss es sich nicht unbedingt um Beleidigungen handeln. Der angeschriebende User soll daher die Möglichkeit haben, den anschreibenden User zu blocken, falls dieser auch nach Bitte nicht aufhört zu schreiben.

Kapitel 7

Fazit

Zum Schluss der schriftlichen Dokumentation der Gruppe B erfolgt in diesem Kapitel noch ein Fazit. Hierbei wird das Softwareprojekt rückblickend betrachtet. Dabei werden auch die Höhen und Tiefen im Projekt mit einbezogen. Ebenfalls gibt es in diesem Kapitel ein kurzes Feedback zum Softwareprojekt.

7.1 Rückblick - Was hat man gelernt?

Schlussendlich kann man sagen, dass das Softwareprojekt ein großer Erfolg war. Die Projektgruppe konnte einen umfassenden Eindruck gewinnen über die praktische Erstellung einer Software. Die dabei vorher angeeigneten Lehrinhalte konnten während dieses Projektes angemessen angewendet werden.

Beim Start des Projektes wurde uns über eine begleitende Vorlesung die Problemstellung erläutert. Dabei wurden uns verschiedene Werkzeuge an die Hand gelegt, die uns während des ganzen Projektes begleitet haben.

Parallel zu der Vorlesung gab es die ersten Gruppentreffen, bei der sich manche Projektteilnehmer das erste Mal kennen lernten. Zu Anfang gab es keine großen Schwierigkeiten und die Gruppe fügte sich gut zusammen. Der am Anfang erstellte Strafenkatalog kam, bis auf ein paar Ausnahmen, auch nur selten zum Einsatz. Somit wurden die Pflicht- und freiwilligen Treffen immer ernst genommen. Im weiteren Verlauf hat sich die Gruppe weiterentwickelt. Zum einen in den Themen, in denen die Gruppe neu eingeführt wurde. Die wenigsten haben vorher mit Scrum oder Git gearbeitet. Dabei konnten die Gruppenmitglieder sich untereinander immer helfen, wenn Probleme aufgetaucht sind. Es konnten immer Lösungen gefunden werden. Häufiges Anwenden führte schließlich dazu, dass alle vertraut wurden mit den Methoden, die wir verwendeten, sowie mit der Software.

Im weiteren Verlauf des Projektes trat das Problem auf, dass es keine persönlichen Treffen mehr gab. Dabei hat die Gruppe gut adaptiert und konnte ohne Hindernisse weiterarbeiten. Gerade selbständiges Arbeiten an dem Projekt war zu dieser Zeit wichtig. Dabei sind sowohl die Fähigkeiten wie das selbständige Arbeiten, aber aber auch die Team-Kompetenzen gewachsen, da gerade neue Ideen entwickelt werden mussten, um das Projekt normal weiterzuführen.

7.2 Höhen und Tiefen im Projekt

Innerhalb des Softwareprojektes gab es im Arbeitsprozess sowohl verschiedene gute, als auch schlecht verlaufende Aspekte. Diese betrafen den Arbeitsfluss an sich, aber auch die Kommunikation der Mitglieder untereinander. Auch die Arbeitsmoral war zwischenzeitig ein Gesprächsthema.

Anfänglich befand sich das Team in der Findungsphase. Hierbei traten neben den Kommunikationsproblemen ebenfalls auch Probleme mit den zu verwendenten Tools auf. Jira, Confluence und Bitbucket wurden viel zu wenig verwendet. Auch war die Bedienung und Rollenverteilung unklar.

Im Laufe des Projektes konnten die anfänglichen Probleme durch eine verbesserte Kommunikation und Aufklärung der einzelnen Tools untereinander nach und nach behoben werden. So haben sich mit der Zeit „Experten“ in den einzelnen Kategorien gebildet welche als Wissensvermittler auftraten. Weiterhin wurden durch Hilfestellungen wie das Pair Programming auch Teammitglieder motiviert und unterstützt, deren Stärken im Regelfall in den jeweils anderen Kategorien lagen. Hierdurch wurde die Arbeitsmoral und die Ticketabarbeitung im Zusammenhang mit der Codequalität maßgeblich verbessert.

Die Bearbeitung der Aufgaben verlief bis zum Ende hin immer konsequenter und zuverlässiger. Das Teamgefühl wuchs und es kam bis zum Schluss des Softwareprojektes eine Art „Arbeitsroutine“ auf. Somit konnte das Projekt zuverlässig und problemfrei zum Abschluss gebracht werden.

7.3 Feedback zum Softwareprojekt - SWP Retroperspektive

Zum Schluss wurde noch eine Retrospektive über das ganze Softwareprojekt durchgeführt. Dabei wurden noch mal alle Eindrücke von den Projektteilnehmern gesammelt. Diese Eindrücke wurden nach gut und schlecht sortiert und gruppiert. Zum Schluss wurden dann noch Vorschläge gemacht, wie die Probleme beim nächsten Mal nicht auftauchen.

Dabei ist positiv die Kommunikation im Verlauf des Projekts aufgefallen. Es wurde nie beleidigt und man hat sich immer höflich verhalten. Selbst bei Auseinandersetzungen wurde keine Person ausfallend. Ebenfalls hat man sich meist gut verstanden und konnte sich immer gut mitteilen.

Des Weiteren ist der Programmierteil des Projekts größtenteils für gut befunden worden. So haben Sachen wie z. B. Pair Programming oder Hilfestellungen von Teammitgliedern, die auch etwas mit Kommunikation zu tun haben, gut funktioniert. Aber auch die Erweiterung der Programmierkenntnisse wurden positiv aufgenommen. Was jedoch nicht so gut aufgenommen wurde, ist die Qualität des Codes an manchen Stellen. Der Code ist an manchen stellen nicht effizient oder es gibt noch ungenutzten Code. Um das zu vermeiden, sollte man nochmal seinen Code durchgehen. Ein weiterer negativer Punkt ist, dass die Pull Requests meistens einfach nur durchgewunken werden, anstatt dass diese wirklich mal überprüft werden. Dazu kommt dann auch noch das Vernachlässigen der Testabdeckung und der Dokumentation während des Projektes. Die einfachste Lösung wäre es, nach dem Schreiben der Methoden direkt Tests zu erstellen und die Doku zu schreiben.

Schließlich gibt es auch noch Punkte, die negativ aufgefallen sind, die nicht direkt mit dem Arbeiten der Gruppe zu tun haben. Dabei war ein großer Negativpunkt, dass die Gruppe zu groß für das Arbeiten mit Scrum ist. Auch das Daily Scrum war schwierig, wenn man sich nur ein bis zweimal die Woche sieht. Ein Lösungsvorschlag wäre dafür, dass man das Daily-Scrum per Discord abhalten könnte.

Im Allgemeinen kann man zum Schluss sagen, dass viele Probleme umgehbar gewesen wären, wenn von Anfang an gründlicher gearbeitet worden wäre. Dabei hätten alle Gruppenmitglieder mehr darauf achten müssen, dass die von Anfang an erstellten Konventionen eingehalten werden.

Kapitel 8

Anhang

8.1 Spielanleitung

Das Spiel beginnt damit, dass jeder Spieler fünf Karten auf die Hand bekommt. Der Spieler der gerade am Zug ist, fängt in der Aktionsphase an. In welcher Phase sich der Spieler befindet und wie viele Ausspiel-\Kaufaktionen er hat, wird ihm anhand einer Anzeige vor ihm dargestellt. Hat der Spieler keine Aktionskarte in der Hand oder keine Aktionen zum Ausspielen mehr übrig, wechselt er in die Kaufphase über. Ansonsten werden alle Geldkarten auf seiner Hand ausgegraut und er kann mit einem Linksklick eine Aktionskarte aus seiner Hand ausspielen.

Abbildung 8.1: Aktionsphase-Beispiel

Wenn die Karte weitere Spielerinteraktion benötigt, wird dies in der Anzeige angezeigt. Beispielsweise, dass der Spieler Karten von seiner Hand auswählen kann. Dabei werden alle Handkarten dunkel angezeigt und die gewählten Karten in normaler Helligkeit.

In der Kaufphase muss der Spieler als erstes über den Button „Alle Geldkarten ausspielen“ seine Geldkarten ausspielen, um mit Linksklick eine Karte kaufen zu können. Sollte der Spieler keine Kaufaktionen mehr haben, geht er in die Clearphase über, die all seine Handkarten und ausgespielten Karten auf den Ablagestapel legt und fünf neue Karten von dem Nachziestapel zieht. Sollte der Nachziehstapel keine Karten mehr enthalten, dann wird der Ablagestapel dem Nachziehstapel hinzugefügt. Diese Phase benötigt keine Spielerinteraktion und nachdem sie durchgeführt wurde, ist der nächste Spieler am Zug.

Im Chat auf der linken Seite des Spielfeldes, werden die Tätigkeiten jedes Spielers festgehalten und hier findet die Interaktion mit den anderen Spielern statt. Während ein Spieler am Zug ist, können die anderen Spieler keine Karten ausspielen oder andere Buttons anklicken außer den Aufgebe- und Logoutbutton. Sollten alle Siegespunktekarten gekauft worden sein, dann wird das Spiel beendet und der Spieler, der die meisten Siegespunkte hat, bekommt eine Gewinnbenachrichtigung. Er kann sich dann entscheiden noch einmal zu spielen oder aufzuhören.

8.2 Vollständige Klassendiagramme

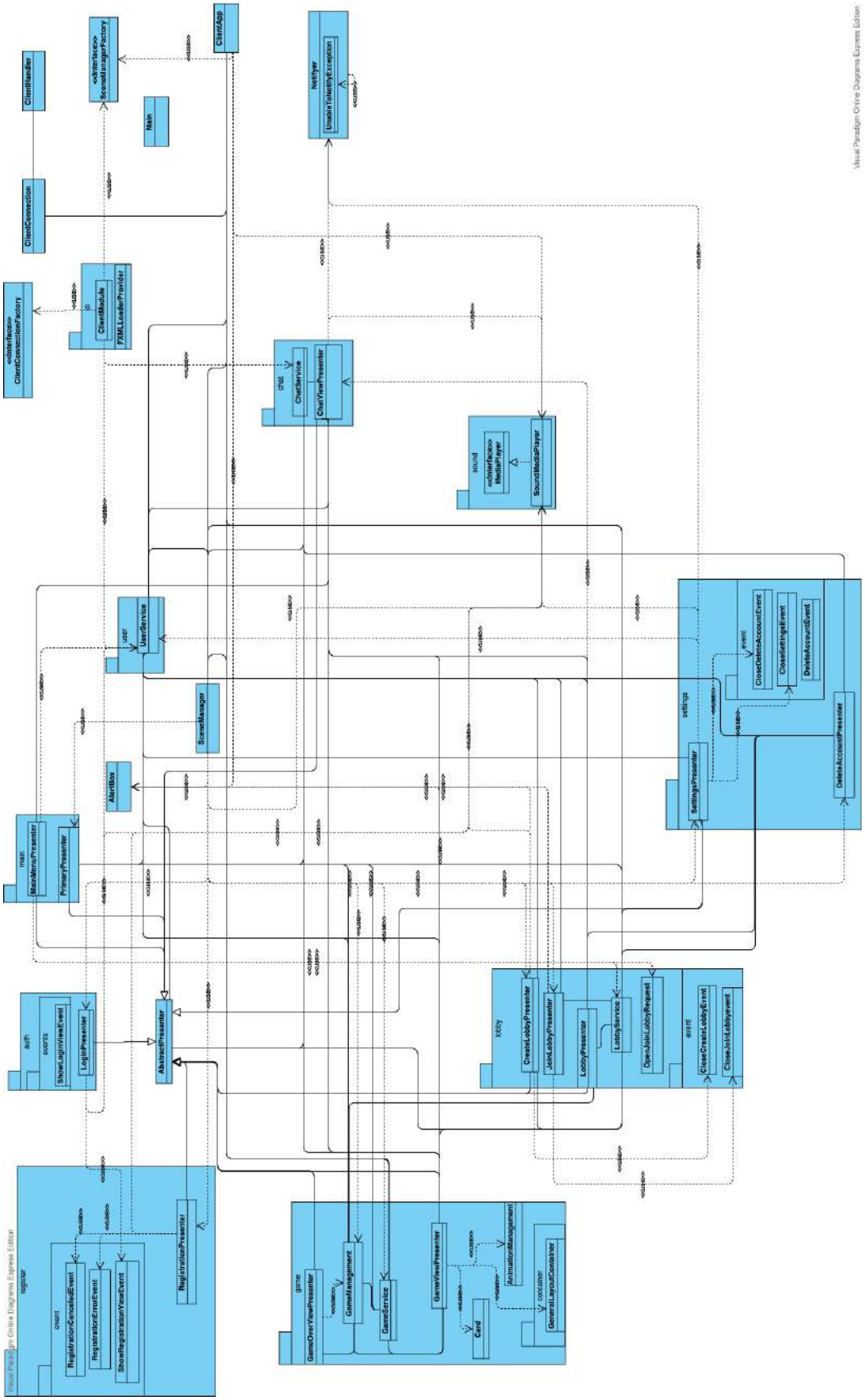

Abbildung 8.2: vollständiges Klassendigramm des Clients

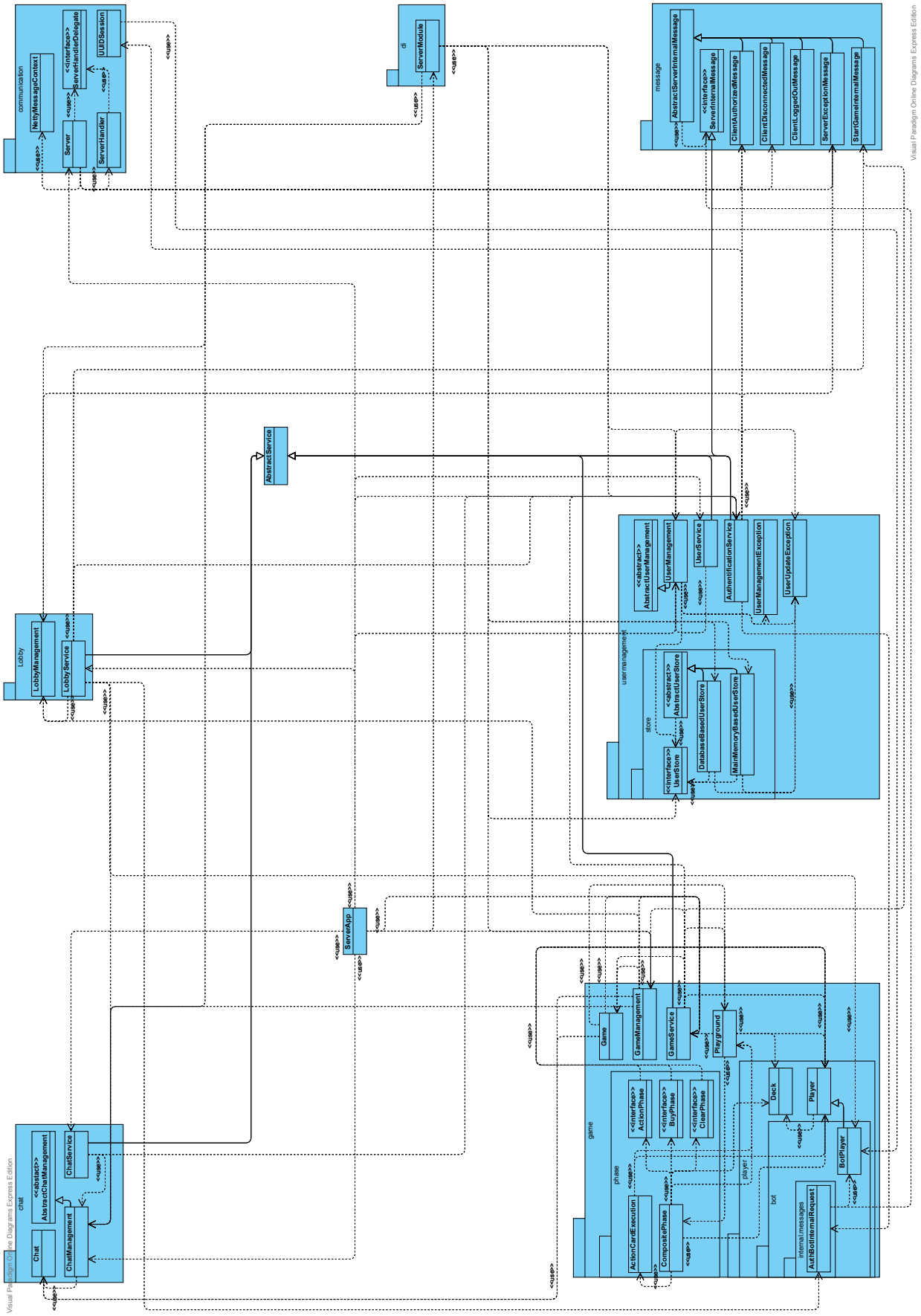

Abbildung 8.3: vollständiges Klassendigramm des Servers

8.3 Installation und Konfiguration der Virtuellen Maschine

Der Start des Spiels ist an einige Vorraussetzungen geknüft. Diese werden hier im einzelnen Behandelt. Im Wesentlichen sind zwei Möglichkeiten vorgesehen, dass Spiel zu starten.

- Mittels einer vorkonfigurierten Virtuellen Menschine (Im weiteren Verlauf VM genannt)
- Auf einem Windows-, Linux- oder macOS-System mittels der frei verfügbaren JAVA-Runtime

Die notwendige Installation und Konfiguration zum starten des Spielclients und -servers wird im Folgenden genauer beschrieben.

8.3.1 Einrichtung und Nutzung der vorkonfigurierten VM

Um die vorkonfigurierte VM nutzen zu können, benötigen wir im Vorraus folgende Dateien und Software dritter:

- Die Virtualisierungssoftware Virtualbox (zu finden auf Virtualbox.org)
- Die enthaltene Virtuelle Festplatten-Datei „SWP1920-B.vdi“

Nachdem die benötigte Software installiert, als auch die Dateien auf den Computer kopiert worden sind, kann nun die VM eingerichtet werden. Hierfür wird zuallererst die Software Virtualbox gestartet. Folgendes Fenster sollte nun zu sehen sein:

Abbildung 8.4: Virtualbox (Version 6.1)

Um nun eine neue VM zu erstellen, klicken wir nun auf die Schaltfläche „Neu“ und tragen dort alle nötigen Werte ein wie z.B.

- Den Namen der VM
- Wo die VM abgespeichert werden soll
- Um welchen Typ es sich handelt
- Um welche Betriebssystemversion es sich handelt
- Wie viel Abeitsspeicher die VM zur Verfügung haben soll (Min. 4GB)
- Das eine vorhandene Festplatte verwendet werden soll (Enthaltene Datei “SWP1920-B.vdi“ auswählen)

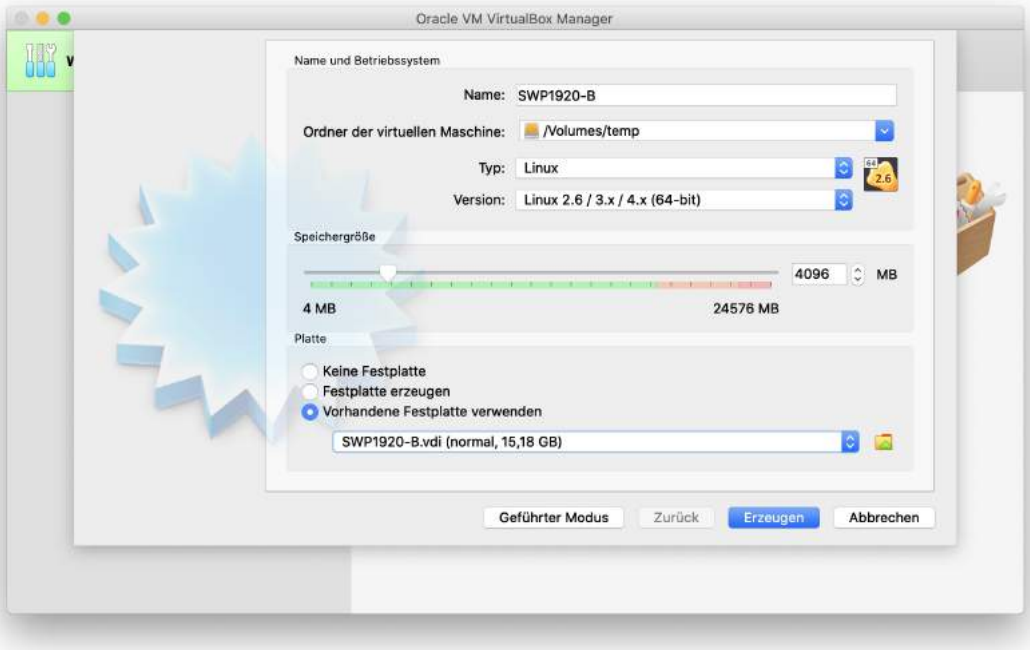

Abbildung 8.5: Erstellen einer neuen VM

Als nächstes kann die VM erstellt werden. Hierfür genügt ein Klick auf die Schaltfläche “Erzeugen“.

Um die Performance der VM zu verbessern, werden wir noch einige Einstellungen verändern. Hierfür klicken wir auf die Schaltfläche „Ändern“. Folgende Werte sollten angepasst werden:

- Unter „System/Prozessor/Prozessoren“ die Anzahl der zu verwendenen Kerne auf den höchstmöglichen Wert im grünen Bereich einstellen
- Unter „Anzeige/Bildschirm/Grafikspeicher“ die Größe des Videospeichers auf den höchstmöglichen Wert im grünen Bereich festlegen

Anschließend ist die Installation und Konfiguration auch schon abgeschlossen. Die VM kann nun über die Schaltfläche „Starten“ gestartet werden. Der Standardbenutzer sollte sich automatisch anmelden. Sollte dennoch mal das ROOT-Password benötigt werden: „dominion“.

Auf dem Desktop finden wir nun folgende vorgefertigte Skripte:

- CLIENT.sh - Startet einen Spielclient
- SERVER.sh - Startet den Spielserver

Um mehrere Spielclients zu starten, kann das entsprechende Skript einfach erneut gestartet werden.

Hinweis: Der Spielserver muss vor den Clients gestartet werden.

Glossar

Backlog Backlog ist eine Liste von Aufgaben bzw. Anforderungen, die abgearbeitet bzw. realisiert werden sollen. 69, 83

Bitbucket Bitbucket ist ein Tool von Atlassian zur gemeinsamen Bearbeitung von Software-Entwicklungsprojekte, welches Git unterstützt. 68, 74, 78, 82, 129

Bot Ein Bot ist ein Computerprogramm, das automatisch sich wiederholende Aufgaben abarbeitet. Ein Bot ist dabei nicht unbedingt auf die Interaktion mit einem Menschen angewiesen. 10, 11, 30, 80, 83, 85, 112, 113, 117, 118, 120, 121, 123

Branch Ein Branch ist eine Abzweigung eines bestimmten Codezweiges auf dem unabhängig von dem Primärcodezweig eine Funktion implementiert oder Bug gefixt werden kann. Ist das Programmieren abgeschlossen, so kann ein Pull Request erstellt werden um den Codezweig in den Primärzweig zu integrieren. 66, 74, 78, 87, 89, 91, 107, 120, 140

Bug Als Bug wird das Fehlverhalten eines Computerprogramms bezeichnet, der durch Programm- oder Softwarefehler auftritt. 80, 84, 86, 93–95, 98, 100–102, 105, 106, 109, 110, 116, 118, 122, 123

Channels Ein Channel beschreibt einen für sich selbst stehenden Kommunikationskanal. 74

Client Ein Client ist ein Programm, das mit einem Server kommuniziert und von diesem Daten und spezielle Dienste in Anspruch nimmt.. 87, 88, 97, 98, 105, 117, 123

Command Pattern Das Command Pattern ist ein Entwurfsmuster in der Softwareentwicklung. 72

Confluence Confluence ist eine Wiki-Software, welche von Atlassian entwickelt wurde. 71, 73, 74, 76, 85, 86, 129

ControlFX ControlFX ist eine Bibliothek von UserInterface Elementen und nützlichen Programmierschnittstellen für JavaFX 8.0 und darüber. 68, 72, 119

Daily-Scrum Das Daily-Scrum ist ein Scrum-Ereigniss. Es findet normalerweise täglich statt. Bei diesem Ereignis gibt jedes Mitglied darüber Auskunft: Was hat es seit dem letzen Mal gemacht? Was will es machen? Wo bestehen Hindernisse?. 70, 130, 141

DAO-Pattern Das DAO-Pattern ist ein Entwurfsmuster, das den Zugriff auf unterschiedliche Arten von Datenquellen so kapselt, dass die angesprochene Datenquelle ausgetauscht werden kann, ohne dass der aufrufende Code geändert werden muss. . 15

Discord Bei Discord handelt es sich um einen kostenlosen Onlinedienst, welcher für Instant Messaging, Chats, Sprachkonferenzen und Videokonferenzen genutzt wird. 71, 74, 82, 85, 86, 130

Epic Ein Epic ist eine große Storie, bei der die Details noch nicht ganz klar sind. Aus einem Epic werden i.d.R. mehrere User-Stories. 77

EventBus Der EventBus dient dazu die Interaktion zwischen mehreren Klassen zu erleichtern, da dieser die Nachrichten vom Sender an die registrierten Empfänger sendet. 15, 16, 34, 88, 103, 107

Framework Ein Framework ist ein Programmiergerüst, das in der Softwaretechnik, insbesondere im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungsansätzen, verwendet wird. Im allgemeineren Sinne bezeichnet man mit Framework auch einen Ordnungsrahmen. 3, 67, 72, 83, 84

Git Git ist ein Open-Source-Tool zur verteilten Versionskontrolle von Software. 68, 74, 128

Google Guava Google Guava ist eine freie Sammlung von Programmbibliotheken für die Programmiersprache Java. Die Sammlung ist als Ergänzung zur Funktionalität der Java-Klassenbibliothek gedacht und erweitert diese beispielsweise bezüglich Collections, I/O-Unterstützung oder Stringmanipulationen. 84

GUI Graphical User Interface (GUI) ist eine Form der Benutzerschnittstelle zwischen dem Computer und dem User. Ihre Aufgabe ist es eine Anwendungssoftware graphisch darzustellen.. 84, 85, 139

IDE Eine integrierte Entwicklungsumgebung (IDE, von englisch "integrated development environment") beschreibt eine Sammlung von Software, mit denen die Aufgaben der Softwareentwicklung (SWE) möglichst ohne Medienbrüche bearbeitet werden können. 72, 85

IntelliJ IntelliJ IDEA ist integrierte Entwicklungsumgebung des Softwareunternehmens JetBrains. Es unterstützt die Programmiersprachen Java, Kotlin, Groovy und Scala. 72, 73, 85

Interface Ein Java Interface (Schnittstelle) ist ein abstrakter Typ, der ein Verhalten definiert welches durch implementierende Klassen konkretisiert werden muss. 32, 46, 88

Java Klasse Eine JAVA Klasse ist ein Konstrukt, welches Methoden, Attribute und weitere Eigenschaftenh eines Objektes definiert. 32, 90, 92, 95–97, 116

JavaDoc Javadoc ist ein Software-Dokumentationswerkzeug, das aus Java-Quelltexten automatisch HTML-Dokumentationsdateien erstellt. Javadoc wurde ebenso wie Java von Sun Microsystems entwickelt und ist ab Version 2 ein Bestandteil des Java Development Kits.. 66, 86, 117

JavaFX JavaFX ist ein Framework zur Erstellung plattformübergreifender Java-Applikationen. Es ist eine Java-Spezifikation von Oracle und setzt sich zum Ziel, das professionelle Erstellen und Verteilen von interaktiven, multimedialen Inhalten und grafischen Benutzeroberflächen (GUIs) über sämtliche Java-Plattformen hinweg zu erleichtern. 68, 72, 84, 85, 105, 119

JDBC Java Database Connectivity (JDBC) ist eine Datenbankschnittstelle der Java-Plattform, die eine einheitliche Schnittstelle zu Datenbanken verschiedener Hersteller bietet und speziell auf relationale Datenbanken ausgerichtet ist. Zu den Aufgaben von JDBC gehört es, Datenbankverbindungen aufzubauen und zu verwalten, SQL-Anfragen an die Datenbank weiterzuleiten und die Ergebnisse in eine für Java nutzbare Form umzuwandeln und dem Programm zur Verfügung zu stellen. Für jede spezifische Datenbank sind eigene Treiber erforderlich, die die JDBC-Spezifikation implementieren. Diese Treiber werden meist vom Hersteller des Datenbank-Systems geliefert.. 73, 74

Jira Jira ist eine Webanwendung zur Fehlerverwaltung, Problembehandlung und operativem Projektmanagement, die von Atlassian entwickelt wurde. Jira wird auch in nichttechnischen Bereichen für das Aufgabenmanagement, primär aber in der Softwareentwicklung eingesetzt. Dort unterstützt es das Anforderungsmanagement, die Statusverfolgung und später den Fehlerbehebungsprozess. Jira ist durch seine Funktionen zur Ablauforganisation ("'Workflow-Management"') verwendbar für Prozessmanagement und Prozessverbesserung.. 66–69, 71, 74, 77, 129

JUnit 5 JUnit ist ein Framework zum Testen von Java-Programmen, das besonders für das automatisierte Testen von Klassen und Methoden geeignet ist. 61, 73, 84

Maven Maven ist ein Build-Management-Tool der Apache Software Foundation und basiert auf Java. Mit ihm kann man insbesondere Java-Programme standardisiert erstellen und verwalten. 68, 72, 84

Medienbrüche Ein Medienbruch entsteht in der Informationsverarbeitung, wenn in der Übertragungskette eines Prozesses Daten/Informationen von einem auf ein weiteres/anderes Informationsmedium übertragen werden müssen.. 139

Merge Merge ist der Vorgang des Abgleichens mehrerer Änderungen, die an verschiedenen Versionen derselben Datei getätigt wurden. In dem Projekt bezeichnet der Begriff, dass die Änderungen von einem Branch in einen anderen gespeichert werden. 87

Message Eine Message (dt. Nachricht) ist eine Art „Brief“ mit dem ein Inhalt (zum Beispiel ein Objekt) übergeben werden kann. 16, 17, 19, 20, 23, 24, 28, 29, 32, 34–36, 39, 40, 47–49, 54, 58, 59, 66, 92, 102, 112, 115

Mocktio 3 Mockito ist ein Framework zu Erstellung von so genannten „Mock-Objekten“. Mock-Objekte sind dabei Platzhalter für Objekte oder Programmteile. Es kann auch von „Dummys“ gesprochen werden. Das Framework Mockito arbeitet hierbei eng mit JUnit zusammen und ermöglichst es dadurch „Mock-Objekte“ innerhalb von Tests zu verwenden.. 73

Model View Presenter Model View Pattern (MVP) ist ein Entwurfsmuster in der Softwareentwicklung. 72

Netty Netty ist ein nicht-blockierendes Client-Server-Framework für die Entwicklung von Java-Anwendungen im Netzwerk, wie Protokollserver und Netzwerk-Clients. Das asynchrone, ereignisgesteuerte Framework für Anwendungen im Netzwerk vereinfacht die Arbeit des Entwicklers an Protokollen wie TCP und UDP sowie ihren Socket-Servern. Netty beinhaltet auch eine Implementation des Reactor-Entwurfsmusters. 84

Objekt Ein Objekt ist eine Instanz beziehungsweise einzelne Ausprägung einer Klasse und ist eindeutig. 97, 98, 139–141

Observer Pattern Das Observer Pattern ist ein Entwurfsmuster in der Softwareentwicklung. 72

Overleaf Overleaf ist ein Online-LaTeX-Editor. Für die Nutzung ist keine Installation notwendig und es können mehrere Personen gleichzeitg an dem LaTeX-Dokument arbeiten. 71, 74, 76, 77, 85

Pair Programming Pair Programming beschreibt das gemeinsame Programmieren von Codeabschnitten. Hierbei ist es am Besten, wenn einer den Code schreibt, welcher der Andere erklärt. Somit erhalten Beide einen Lerneffekt. 129

Pattern Pattern sind Entwurfsmuster mit bewährten Lösungen für wiederkehrende Entwurfsprobleme. 1, 3, 14, 67, 68, 72, 112, 113

Pull Request Ein Pull Request ist eine Anfrage die gestellt wird, damit Änderungen die in einem eigenen Branch vorgenommen wurden in einem (Ziel-)Branch übernommen werden. Ein Pull Request muss in der Regel von mindestens zwei Personen genehmigt werden. 66, 70, 73, 78, 82, 86, 129, 138

Request Mit einem Request ist eine Anfrage gemeint. So zum Beispiel könnte mit einem „UserKickRequest“ beim Server angefragt werden einen bestimmten User aus der Lobby beziehungsweise dem Spiel zu kicken. Der Request würde in dem speziellen Fall noch das User-Objekt als entsprechendes Attribut mitführen. 19–21, 23–26, 28, 29, 32, 34–40, 46–49, 54, 56, 58, 59, 66, 141

Response Eine Response ist eine Antwort auf ein Request. 25, 26, 32, 35, 40, 56

Retrospektive Bei der Retrospektive wird der vergangene Sprint rückblickend betrachtet und die bisherige Arbeitsweise geprüft. 70, 77, 129, 141

Review Beim Review werden Ergebnisse in der Softwareentwicklung manuell geprüft, indem die mitarbeitenden Personen die Ergebnisse durchsehen. 66, 68, 70

Scenebuilder Der Scenebuilder wird von Gluon betreut und ist ein Tool zur visuellen Darstellung von FXML-Dateien. Es handelt sich hierbei um Open Source Software. Es lässt den Entwickler visuell via Drag-and-Drop die Oberfläche seines Programmes erstellen. 85

Scrum Scrum ist ein Vorgehensmodell, welches für das Projekt- und Produktmanagement genutzt wird, insbesondere für das agile Projektmanagement. 68, 69, 128, 138, 141

Smell Code Unter Code-Smell, kurz Smell (deutsch ”’[schlechter] Geruch”’) oder deutsch übelriechender Code versteht man in der Programmierung ein Konstrukt, das eine Überarbeitung des Programm-Quelltextes nahelegt. Dem Vernehmen nach stammt die Metapher Smell von Kent Beck und erlangte weite Verbreitung durch das Buch Refactoring von Martin Fowler. Unter dem Begriff sollten handfestere Kriterien für Refactoring beschrieben werden, als das durch den vagen Hinweis auf Programmästhetik geschehen würde. 84

Sprint Als Sprint wird ein zeitlicher Abschnitt von etwa 2-4 Wochen bezeichnet, der wähhrend eines Scrum-Projektes stattfindet. Er beinhaltet verschiedene Scrum-Ereignisse wie das Daily-Scrum oder die Retrospektive. 4, 66, 67, 69–71, 86, 87, 89, 91, 92, 94, 95, 98, 101–103, 106, 107, 112, 117, 118, 123, 141

Task Ein Task ist ein einzelnes Aufgabenpaket, welches zu einer User Story gehören kann. 66, 69, 83

User Story Eine User Storie ist eine Anforderung an das (zu entwickelnde) System, die in der Sprache des Kunden beschrieben ist und die einen konkreten und sichtbaren Mehrwert für den Nutzer liefert. 8, 69, 86, 88, 91, 98, 111, 124, 142

Akronyme

dt. auf Deutsch. 140

GUI Graphical User Interface. 139

MVP Model View Pattern. 140

NAS Network Attached Storage. 12, 81

Abbildungsverzeichnis

Tabellenverzeichnis