DevOps Dokument

zum Spiel „Dog“

Softwaretechnikpraktikum Gruppe 08

10. Dezember 2023

1 Development

Der Development Teil des Dokumentes erläutert die Architektur der einzelnen Komponenten und des gesamten Produktes.

1.1 Architektur

Nachfolgend werden sämtliche Komponenten im Detail vorgestellt, wobei ihre Struktur und die bestehenden Verbindungen erläutert werden. Die Funktionen der Komponenten sind dabei innerhalb UML-Klassendiagramme grafisch dargestellt.

1.2 Server

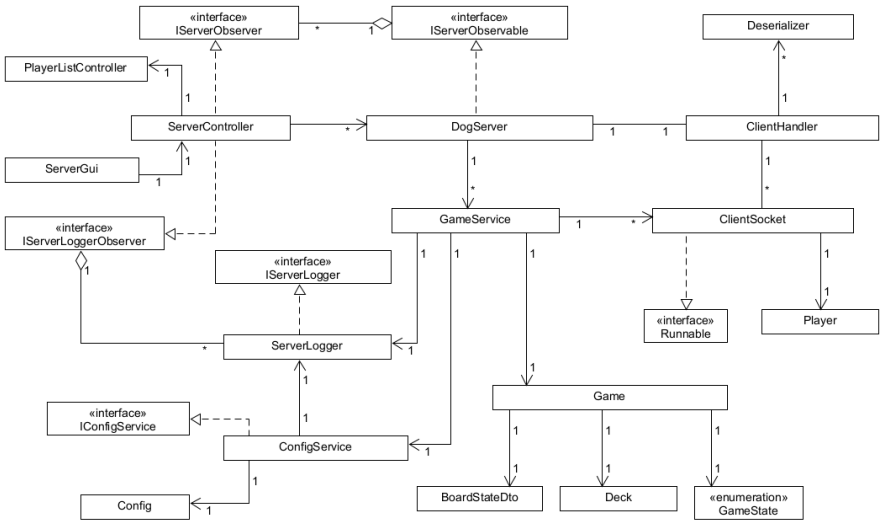

Abbildung 1: Server Architektur

1.2.1 Model

Config - Das Model Config enthält verschiedene Attribute hinsichtlich der Konfiguration eines Spiels, wie z.B. die Anzahl der Spieler oder die Anzahl der Karten eines Spielers. Weitere Informationen sind unter dem Punkt 2.1.1 Spielkonfiguration zu finden.
Game - Im Game Model werden die Informationen von den laufenden Spielen abgespeichert. Dazu gehören der Status des Spiels, das Spielbrett, die Decks, der momentane Player Index sowie die bisher verbrachte Zeit. Für den Spielstatus gibt es zusätzlich eine Aufzählungs-Klasse mit verschiedenen mögliche Gamestate, wie “paused”, “running”, “canceled” und “over”. Des Weiteren führt eine Assoziation von Game Model zu den Models BoardStateDto und Deck.
Das BoardstateDto Model ist ein Datentransferobjekt (kurz: DTO) und verwaltet sämtliche Informationen bezüglich des Spielfelds, wie den Ablagestapel. Das Deck Model hingegen kümmert sich um die Verwaltung des Ziehstapels.
Player - Das Player Model speichert spezifische Informationen über den Spieler. Das Player Attribut im ClientSocket wird zunächst auf Null gesetzt und sobald klar ist, ob es sich um einen Spieler handelt auf ein Player Objekt gesetzt. Dieses Player Model speichert Player-Informationen, die nur relevant für Spieler sind, wie z.B. die Karten, die der Spieler gerade auf der Hand hält.

1.2.2 Observer-Pattern

In der Architektur wird darüber hinaus das Observer-Pattern (s. 1.6) durch zwei Klassen implementiert.
Die Klasse ServerController beobachtet als Observer den Dogserver, welcher hierbei als Subject fungiert. Dies bedeutet konkret, dass bei Änderungen des Zustands am Dogserver dieser den Observer ServerController über mehrere update()-Methoden benachrichtigt. Dies wird realisiert, indem die ServerController-Klasse von der Interface-Klasse IServerObserver erbt und die DogServer-Klasse von der Interface-Klasse IServerObservable erbt. Des Weiteren fungiert der ServerController ebenfalls als Beobachter für die ServerLogger-Klasse. Hier ist dasselbe Prinzip wie zuvor durch Erbung mit einer Observer bzw. Observable Interface-Klasse geregelt. Somit fungiert der ServerLogger als Subject, welches den ServerController bei Zustandsänderungen, wie z.B. Fehlermeldungen und Loggen, informiert.
Mit diesem Pattern können wir schnell und effizient wichtige Informationen an den ServerController senden. Daraufhin kann der Controller die View (ServerGUI), informieren und updaten.

1.2.3 View und Controller

Die einzige View für den Server ist die ServerGUI. Bei jeder Interaktion erfolgt die Kommunikation mit dem ServerController. Auf diese Weise erhält der ServerController, ähnlich dem MVC-Pattern (s. 1.6), die Benutzerinteraktion von der View und verarbeitet diese in Verbindung mit der Spiellogik.
Ein weiterer Controller, der dem ServerController unterstellt ist, ist der PlayerListController. Mit diesem werden in einem weiteren Fenster Benutzerinteraktionen verarbeitet. Dieses zusätzliche Fenster wurde konzipiert, um Spieler zu Spielen zuzuweisen. Die Controller treten anschließend in Kommunikation mit der Spiellogik, unter Anwendung des zuvor beschriebenen Observer-Patterns. Zur Logik gehören unter anderem der DogServer, der ServerLogger, der ConfigService, der ClientHandler, die ClientSockets und der GameService.

Die **DogServer-Klasse** ist das Herz der Architektur. Hier werden sämtliche wichtige Informationen weiter gesendet und empfangen. So kriegt der DogServer jegliche Nachrichten vom Client, die über den ClientHandler verschickt werden. So werden Informationen wie jegliche Dto Objekte und das Einloggen von neuen Spieler gesendet. Daraufhin verarbeitet der DogServer diese Nachrichten und schickt diese weiter an den Controller und somit an die View. Womit das MVC-Pattern hier wieder angewandt wird. Des Weiteren werden wichtige Informationen über das Spiel von den Controllern an die GameService-Klasse geschickt. Dadurch wird das Spiel bei Änderungen geupdatet.

Die **ClientHandler-Klasse** ist die Schnittstelle zwischen Server und Client. Diese kommuniziert über ClientSockets, mithilfe von Runnables, mit den Spielern. Die Runnable-Klasse ermöglicht die Erstellung mehrerer Threads, wodurch eine simultane Kommunikation mit verschiedenen Clients möglich ist, ohne auf die Antwort eines einzelnen warten zu müssen. Die gesendeten JSON-Pakete werden dabei mit der Deserializer-Klasse durch GSON in Java Objekte umgeformt. Dies ermöglicht eine einfache Kommunikation zwischen Server und Clients.

Die **ConfigService-Klasse** fungiert als Verwalter der Config-Klasse und übermittelt wichtige Informationen an den ServerLogger. Dieser hat wiederum die Funktion, die wichtigen Ereignisse in den ServerLog anzuzeigen. Darüber hinaus erfolgt eine Kommunikation zwischen dem GameService und dem ConfigService, wodurch Anpassungen an den Spiel-Einstellungen übertragen und aktualisiert werden können.

Die **GameService-Klasse** verwaltet alle Aspekte, die direkt mit dem Spielgeschehen in Verbindung stehen, was im Grunde der gesamten Logik entspricht. So werden Änderungen am Spielgeschehen von der GameService-Klasse direkt an die Spieler übermittelt. Diese direkte Kommunikation ist möglich, da der GameService Listen von Spielern und Beobachtern verwaltet. Des Weiteren werden sämtliche Informationen und Nachrichten ebenfalls zum ServerLogger geschickt, der diese dann mit dem Observer-Pattern an den ServerController weitergeben kann. Zudem erhält der GameService Nachrichten von dem DogServer, wie die vom Interface festgelegten DTOs.

1.3 PC-Beobachter

1.3.1 Observer-Pattern

In der PC-Beobachter Architektur wird das Observer-Pattern (s. Observer-Pattern) in drei Klassen angewendet. Die Klasse PCObserverControllerMenu beobachtet die Klasse ClientGUI, die dabei als Subject fungiert. Die Klasse PCOCMenu implementiert das IClientObserverMenu-Interface. Durch die geerbten update()-Methoden des IClientObservable-Interface werden die entsprechenden handle()-Methoden der PCObserverControllerMenu-Klasse aufgerufen, um angemessen auf Übertragungen durch den Server zu reagieren.
Analog zur zuvor beschriebenen Klasse PCObserverControllerMenu operiert die Klasse PCObserverControllerGameplay. Hierbei wird jedoch das IClientObserverGameplay-Interface implementiert.

1.3.2 Model, View und Controller

Die Klasse ClientGUI fungiert als View und leitet die Datenpakete des Servers an die Controller weiter (s. 1.3.1). Des Weiteren ist die Klasse DrawBoard eine weitere View, die die Darstellung aller Spielfelder, Hausfelder und Spielfiguren übernimmt. Die View PCObserverControllerInfo stellt alle relevanten Spielinformationen übersichtlich in einem eigenen Fenster dar.
Um die unterschiedlichen Zustände des Beobachters zu repräsentieren, existieren mehrere Controller. Diese Zustände umfassen den Startbildschirm, das Menü sowie das Spielfeld mit den Unterzuständen Hausfelder und Info. In den jeweiligen Controller-Klassen erfolgt eine entsprechende Verarbeitung der weitergeleiteten Informationen.
Die Klasse PCObserverControllerMenu ist für sämtliche Aspekte der Serververbindung und der Darstellung des Menüs verantwortlich. Ein weiterer Controller, die PCObserverControllerGameplay-Klasse, behandelt alle Aspekte des Spielbeobachtens, einschließlich der Spielfeldinitialisierung und der Darstellung von Veränderungen während eines Spiels. Als Hilfsklassen sind dieser die Klassen PCObserverControllerHouses, Board, DrawBoard und PieceHandler zugeordnet. Die Klasse PCObserverControllerHouses wie-

derum kümmert sich um alle Angelegenheiten im Zusammenhang mit den Hausfeldern und greift dabei auf die Models HouseBoard, Board und PieceHandler zurück, die gleichzeitig als Hilfsklassen dienen.
Das Model HouseBoard verwaltet die Koordinaten der Hausfelder, während das Model Board die Koordinaten der Spielfelder berechnet. Zusätzlich kümmert sich das Model PieceHandler um die Position der Spielfiguren.

1.4 Engine-Teilnehmer

Der Engine-Teilnehmer ist ein von uns entwickeltes Programm, der als ein nichtmenschlicher Spieler fungiert. Er wird auf einem von uns noch zu entwickelnden Algorithmus basieren, der effizient und korrekt Spielzüge ausführen wird. Der Teilnehmer strebt danach, so viele Punkte wie möglich zu erzielen und dabei gleichzeitig mit den verbleibenden Figuren möglichst weit voranzukommen. Aktuell befindet sich der Engine-Teilnehmer noch in einer frühen Entwicklungsphase und besteht bisher lediglich aus festen, hart codierten Paketen. Dies ermöglicht es uns zumindest, die Funktionalitäten des PC-Beobachters für die Messe vorzustellen.

1.5 Smartphone-Teilnehmer

Die Entwicklung des Smartphone-Teilnehmers startet unmittelbar nach dem Erwerb eines Smartphone-Beobachters auf der Messe. Das Dokument wird entsprechend bei der Endabgabe überarbeitet.

1.6 Patterns

Model-View-Controller

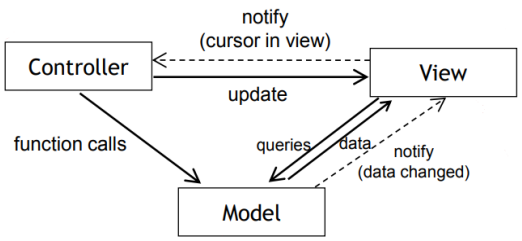

Abbildung 2: Model-View-Control Pattern

Das Model-View-Controller-Pattern ordnet die Software und ihre Dynamik den Komponenten der Architektur zu. Der Model-Teil, als unabhängige Entität von den anderen beiden Komponenten, enthält Daten, Anwendungsfunktionen und den Anwendungszustand. Seine zentrale Funktion besteht darin, Veränderungen an Controller und View zu kommunizieren.
Die View-Komponente präsentiert die visuelle Benutzeroberfläche und zeigt Daten an, die vom Model angefordert werden. Ereignisse, die sich aus der Benutzerinteraktion ergeben, werden an die entsprechenden Controller-Komponenten weitergeleitet.
Die Controller-Komponente ist für die Behandlung von User-Input wie Funktionsaufrufe, Button-Auswahl oder Scrolling zuständig. Sie empfängt Veränderungen von den Views, aktualisiert das Model und fungiert als Bindeglied zwischen den beiden Komponenten. Der Controller repräsentiert auch die Interaktion von Server und Client.

Observer Pattern

Das Observer-Pattern ist ein Entwurfsmuster, das in der Softwareentwicklung verwendet wird, um eine einseitige Abhängigkeit zwischen Objekten zu implementieren. Das Pattern ist ebenfalls für die Kommunikation zwischen Model und View zuständig. Dies hat zur Folge, dass Änderungen an einem Objekt (dem sogenannten "Subject") automatisch an alle abhängigen Objekte (den "Observers") weitergeleitet werden. Die Views können dabei als Observers gesehen werden und die Models als Subjects.

1.7 Eingesetzte Technologien

Docker - Docker ist eine Plattform, die es ermöglicht, Anwendungen isoliert in Containern zu transportieren. Container sind eigenständige, ausführbare Softwarepakete, die alle erforderlichen Abhängigkeiten, Bibliotheken und Einstellungen enthalten, um eine Anwendung unabhängig von der Umgebung auszuführen.

Java SE 11 - Java SE 11 (Java Standard Edition 11) ist eine Version der Java-Plattform, die von Oracle Corporation veröffentlicht wurde.

Intellij IDEA - IntelliJ IDEA ist eine integrierte Entwicklungsumgebung (IDE), die von JetBrains entwickelt und u.a. für die Java-Programmierung konzipiert wurde.

Scenebuilder - Scene Builder ist ein visuelles Layout-Werkzeug, das von Oracle für die Entwicklung von Benutzeroberflächen mit JavaFX verwendet wird.

JavaFX - JavaFX ist ein plattformübergreifendes Framework für die Entwicklung von Benutzeroberflächen (GUIs) in Java. JavaFX bietet eine Vielzahl von Funktionen für die Erstellung interaktiver Anwendungen mit verschiedenen Designs.

Gson - Gson ist eine Java-Bibliothek, die von Google entwickelt wurde und die Konvertierung zwischen JSON (JavaScript Object Notation) und Java-Objekten erleichtert. JSON ist ein leicht lesbares Datenformat, das häufig für den Datenaustausch zwischen Servern und Webanwendungen verwendet wird. Die Gson-Bibliothek ermöglicht es, Java-Objekte in JSON-Strings umzuwandeln (Serialisierung) und umgekehrt JSON-Strings in Java-Objekte zu konvertieren (Deserialisierung).

JUnit - JUnit ist ein Framework für das Testen von Java-Anwendungen. Es ermöglicht Entwicklern, automatisierte Tests für ihre Java-Programme zu erstellen und auszuführen.

1.8 Zukünftige Weiterentwicklung

Durch die Modularität der Software erreichen wir eine effizientere Entwicklung und verbesserte Erweiterbarkeit. Konkret wurde der Code in relativ abgegrenzte und wiederverwendbare Teile strukturiert. Auf diese Weise können neue Komponenten wie beispielsweise der einzukaufende Smartphone-Beobachter problemlos integriert werden. Diese Struktur ermöglicht zudem eine unkomplizierte Anpassung der Logik, wodurch die Implementierung neuer Funktionen und Spielregeln erleichtert wird. Der PC-Beobachter wird um einige kleine Funktionen erweitert und verfeinert. Ebenfalls wird der schon zuvor erwähnte

Engine-Teilnehmer (s. 1.4) weiterentwickelt zu einem gewinnbringendem System und der eingekaufte Smartphone-Beobachter (s. 1.5) zu einem Smartphone-Teilnehmer angepasst.

2 Operation

Der Operations-Teil des Dokuments umfasst ein Handbuch für den Server und PC-Beobachter sowie eine Erklärung zum Bauen, Installieren, Konfigurieren und Ausführen des Produkts. Zusätzlich bietet er Einblicke in den aktuellen Stand der DevOps-Prozesse.

2.1 Handbuch

2.1.1 Server

Installation

Die Installation und Ausführung des Servers erfordert Java Version 11. Für die Benutzeroberflächen-Entwicklung nutzen wir JavaFX Version 17.0.2 und setzen JUnit zum automatisierten Testen ein. Außerdem verwenden wir das Tool Apache Maven, welches die Anwendung aufbaut und gegebenenfalls Abhängigkeiten automatisch lädt. Jegliche Anforderungen für die Installation sind plattformunabhängig, somit werden auch alle Betriebssysteme (Windows, macOS, Linux) unterstützt.

Server Konfiguration

Zum Starten des Servers führt man zunächst die dazugehörige .jar Datei aus. Beim Öffnen der .jar Datei startet unmittelbar die Server-GUI, mit welcher man den Server starten kann. Dafür kann man oben links eine beliebige Zahl von 1024 bis 49151 als Port angeben und auf “Server starten” klicken. Da der Server lokal gehostet wird, ist die IP-Adresse durch das Endgerät bereits festgelegt. Die Konfiguration des Servers erfolgt mit Hilfe von JSON-Dateien, die alle erforderlichen Informationen speichern.

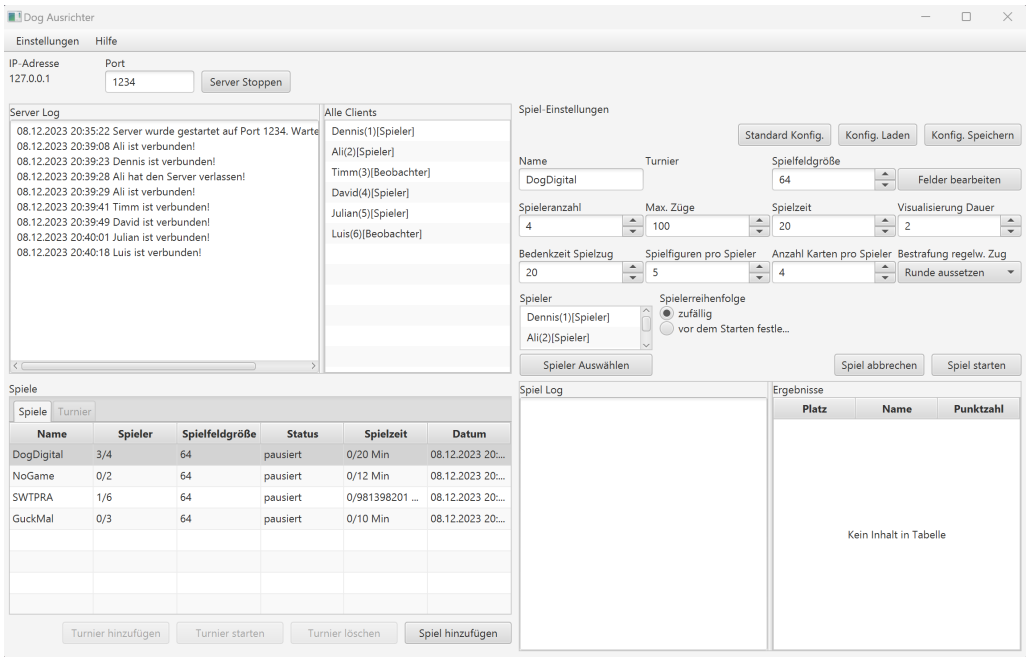

Abbildung 3: AusrichterGUI

Ausführung

Dieselbe GUI wird auch für sämtliche andere Funktionen des Ausrichters verwendet. Auf der linken Seite haben wir einen Server-Log und eine Liste von allen verbundenen Clients. Darüber in der Taskbar gibt es zudem einen Einstellungsbutton, für den Wechsel in den Dark Mode und einen Hilfe Reiter, wo man eine Anleitung und die Software Info findet.
Direkt darunter hat man einen Überblick aller laufenden Spiele und Turniere. In diesem werden der Name des Spiels/Turniers, die derzeitige und maximale Spieleranzahl, die Größe des Spielfeldes, der Status des Spiels, die derzeitige und maximale Spielzeit und das Erstellungsdatum aufgelistet. Wenn man auf das jeweilige Attribut klickt, dann wird auch nach diesem aufsteigend/absteigend sortiert.
Unten rechts wird vom ausgewählten Spiel der Spiel-Log und die Ergebnisse des Spieles/Turniers mit Platz, Name des Spieles und Punktzahl angezeigt.
Direkt unter der Spiel- /Turnierübersicht gibt es vier essenzielle Buttons. Der erste Button fügt ein Turnier hinzu, der Zweite startet ein aus der Liste ausgewähltes Turnier und der Dritte löscht ein ausgewähltes Turnier. Mit dem letzten Button lassen sich Spiele hinzufügen. Sobald man auf diesen klickt kann man das Spiel oben rechts einstellen.

Spiel Konfiguration

Die gesamte Spiel Konfiguration wird oben rechts mit drei für die Konfiguration konzipierte Hauptbuttons erstellt. Durch Betätigung des “Standard-Konfiguration”-Buttons besteht die Möglichkeit, die bereits vordefinierte Einstellung zu wählen, die das klassische Spielerlebnis widerspiegelt. Der Button “Konfiguration Laden” kann bereits gespeicherte Konfigurationen laden. Hierfür muss man lokal von dem Rechner eine passende JSON Datei auswählen. Mit dem letzten Button “Konfiguration Speichern” speichert man die ausgewählten Einstellungen als JSON Datei lokal auf dem Rechner ab.
Im Folgenden die zur Auswahl stehenden Einstellungen beginnend von oben links:

- Name: Legt den Spielnamen fest
- Turnier: Fügt das Spiel in ein auszuwählendes Turnier ein
- Spielfeldgröße: Legt die Anzahl an Spielfeldern fest
- Felder Bearbeiten: Bestimmt, wo die Karte-Ziehen Felder liegen
- Spieleranzahl: Legt eine feste Anzahl von 2-6 Spieler fest
- Max. Züge: Legt die maximale Anzahl an Zügen für das gesamte Spiel fest, bevor es automatisch beendet wird
- Spielzeit: Legt die maximale Spielzeit für das gesamte Spiel fest, bevor es automatisch beendet wird
- Visualisierungsdauer: Legt fest, wie lange die Animationen dauern (0 = keine Animationen)
- Bedenkzeit Spielzug: Legt die Zeit fest, wie viel Zeit ein Spieler für einen Spielzug hat
- Anzahl Spielfiguren: Legt die Anzahl der Spielfiguren eines Spielers fest

- Anzahl Karten pro Spieler: Legt fest, wie viele Karten ein Spieler zu Beginn jeder Runde erhält
- Bestrafung regelwidriger Zug: Legt die Bestrafung eines regelwidrigen Zuges fest.
  a) Entfernen: Spieler wird aus dem Spiel entfernt
  b) Überspringen: Spieler wird für die Runde übersprungen

In den Spiel-Einstellungen unten gibt es in dem “Spiel Fenster” eine Übersicht über die Spieler, die sich bis dato in dem ausgewählten Spiel befinden.
Um Spieler hinzuzufügen, klicken Sie auf den Button “Spieler Auswählen”. Daraufhin öffnet sich ein Fenster, in dem Sie durch Gedrückthalten der Strg-Taste und Rechtsklick manuell mehrere Spieler auswählen können. Zur Auswahl stehen außerdem zwei weitere Buttons. Zum einen “Erste Spieler”, der die obersten Spieler auswählt und zum anderen der Button “Zufällige Spieler”, welcher Spieler zufällig aus der Liste der Spieler auswählt. Der dritte Button “Auswahl aufheben” entfernt einen ausgewählten Spieler. Um die Auswahl zu speichern, klicken Sie unten rechts auf “Übernehmen”. Für das Abbrechen verwenden Sie den entsprechenden Button “Abbrechen”.

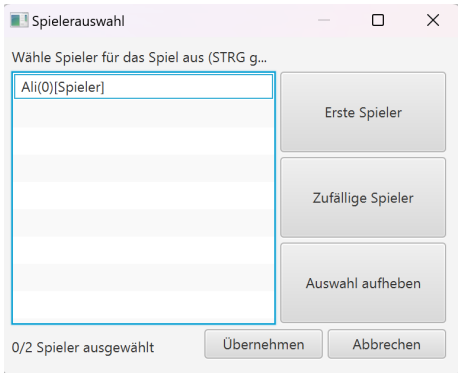

Abbildung 4: Spielerauswahl Menü

Rechts daneben kann man die Spielerreihenfolge einstellen. Zur Auswahl zufällig oder vor dem Starten des Spiels auswählen.
Um das Spiel nun zu starten, muss man auf den unten rechts stehenden Knopf “Spiel starten” klicken. Für das Abbrechen eines Spieles muss der Knopf “Spiel Abbrechen” links daneben geklickt werden.

2.1.2 PC-Beobachter

Hier in diesem Teil wird die Bedienung des PC-Beobachter und die jeweiligen Funktionen erklärt. Hierfür werden jegliche Benutzeroberflächen vorgestellt und detailliert erläutert.

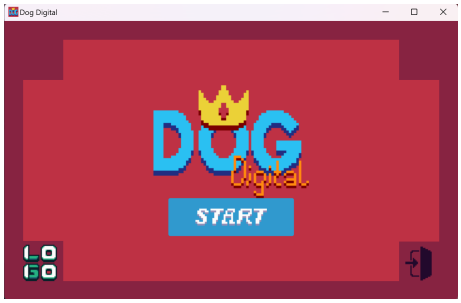

Abbildung 5: Startbildschirm

Login

Beim Starten der .jar Datei startet der Titelbildschirm. Zum Öffnen des Hauptmenüs klickt man den “Starten”-Button und zum Verlassen des Spiels unten rechts die Tür Icon. Um als Beobachter einem Server beizutreten, ist es erforderlich, einen beliebigen Benutzernamen einzugeben, sowie den entsprechenden Port und die IP-Adresse des Servers. Um sich als Beobachter zu verbinden, ist es erforderlich, das Kästchen bei “Beobachter” zu markieren und anschließend auf “Verbinden” zu klicken. Direkt darüber, in der Taskleiste, gibt es einen Hilfe-Button, wo einzelne Elemente kurz erläutert werden und rechts daneben ist der Client-Log zu sehen.

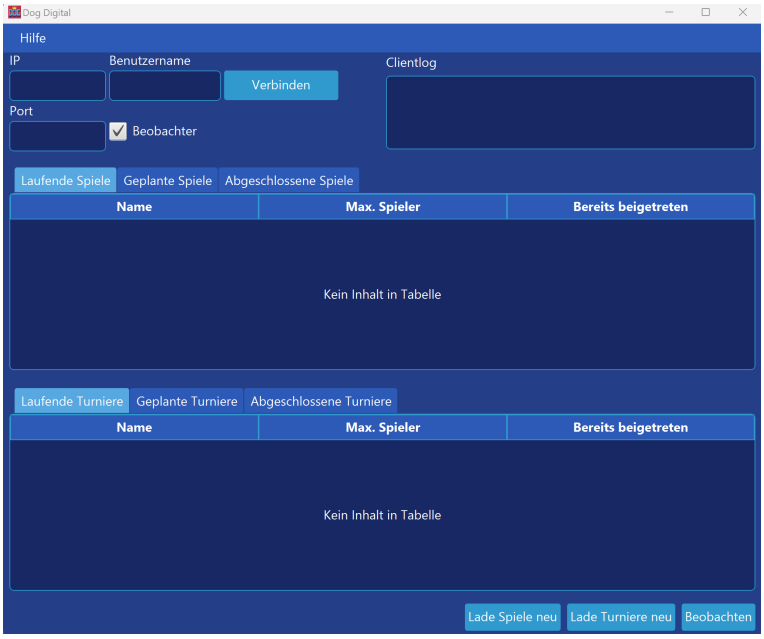

Abbildung 6: BeobachterGUI

Spiel- /Turnierübersicht und Beitreten

Die Spielübersicht in der Mitte ist in drei Reiter aufgeteilt: “Laufende Spiele”, “Geplante Spiele” und “Abgeschlossene Spiele”. Jeder einzelne Reiter listet wiederum drei Attribute auf: Name des Spiels, die maximale Anzahl an Spielern und die bereits beigetretenen Spieler. Man kann jedes Attribut durch einen Klick aufsteigend/absteigend sortieren.
Die Turnierübersicht im unteren Fenster ist ebenfalls in den gleichen drei Reitern aufgeteilt. Unten rechts gibt es die Buttons “Lade Spiele neu” und “Lade Turniere neu”, welche die Spiele- bzw. Turnierliste neu laden. Um einem Spiel beizutreten klickt man auf das gewünschte Spiel und dann auf den “Beobachten”-Button unten rechts.

Spielfeld

Das Spielfeld öffnet sich unmittelbar nach Betätigung des “Beobachten”-Buttons. Auf der linken Seite des Fensters sind die Spieler-Icons mit ihren entsprechenden Kartenanzahlen und der Anzahl der unbehausten Figuren zu sehen. Unten gibt es eine Zeitleiste, welche die noch vorhandene Zeit des aktuellen Spielers für den Spielzug anzeigt. In der oberen rechten Ecke werden die Gesamtspielzeit sowie die Anzahl der bisherigen Züge (links vom Schrägstrich) und die vorher festgelegte Gesamtanzahl der Spielzüge (rechts vom Schrägstrich) angezeigt. Mittig oben liegt der Ablagestapel mit der zuletzt gespielten Karte. In

der oberen linken Ecke stehen fünf zusätzliche Buttons zur Verfügung.
Der “Info”-Button öffnet ein separates Fenster, das die Spielregeln, die aktuelle Spielkonfiguration und die Semantik jeder einzelnen Karte erläutert.
Der “Ansicht Reset”-Button bringt den Benutzer zurück in die Mitte des Spielfeldes.
Um zwischen Vollbild- und Standardmodus zu wechseln, genügt es, einen Haken bei “Vollbild” zu setzen.
Der “Zeige Häuser”-Button öffnet ein neues Fenster, das anzeigt, welche Häuser des jeweiligen Spielers durch Spielfiguren belegt sind.
Das Spiel kann über den “Spiel verlassen”-Button beendet werden.
Das Spielfeld selbst ist in drei Kategorien unterteilt: Karte-Ziehen-Felder, Haus-Felder und normale Spielfelder. Eine Erläuterung zu diesen Kategorien befindet sich ebenfalls im Info-Fenster.

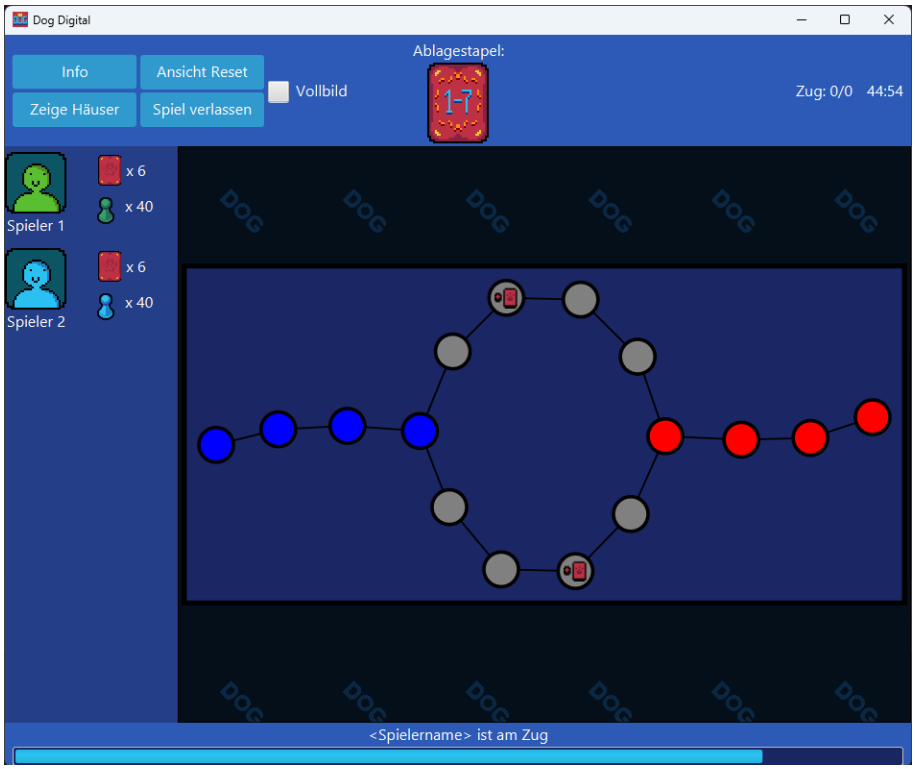

Abbildung 7: Spielfeld

2.2 Betrieb des Engine-Teilnehmers

Nutzung

Für die zukünftige Nutzung des Engine-Teilnehmers muss der Nutzer zuerst die zugehörige .jar Datei öffnen. Daraufhin gibt man den Port und IP-Adresse des Servers an und der Engine-Teilnehmer wählt aus einer Liste von bereits gegebenen Namen einen Benutzernamen aus. Der Ausrichter kann nun den Engine-Teilnehmer zu einem Spiel hinzufügen. Derzeit verfügt der Engine-Teilnehmer über keine komplexe Logik, da er sich noch in einer frühen Entwicklungsphase befindet. Er besteht bisher lediglich aus festen Code-Paket. Dies ermöglicht es uns die Funktionalitäten des PC-Beobachters für die Messe vorzustellen.

2.3 Derzeitiger Entwicklungsprozess

Für die Softwareentwicklung benutzen wir den Scrum-Prozess, womit wir agiler, flexibler und effizienter arbeiten können. Der derzeitige Entwicklungsprozess zum Bauen und Testen der Software wird mit Hilfe von GitLab CI bewältigt. Die Umsetzung der Features erfolgt anhand unserer erstellten User-Storys. Hierfür verwenden wir unser zuvor erstelltes Issue Board auf GitLab, welches wir nach Priorisierung sortiert haben. Zudem besitzt jede User Story eine Aufwandsschätzung, damit man sofort weiß, wie viel Zeit die Task beansprucht und wie viele Entwickler für ein Issue benötigt werden. Dabei gilt ein Issue als erledigt, wenn es die Bedingungen der im QA-Dokument definierten Definition-of-Done erfüllt. Dafür testen wir die Funktionen automatisch mit Hilfe von JUnit, um sicherzustellen, dass die Implementierung einwandfrei funktioniert. Darüber hinaus setzen wir GitLab CI ein, um automatische Tests über Pipelines durchzuführen und Fehler bei neuen Commits oder Merges zu überwachen. Die Pipeline nutzt für den Projektbau Docker-Container und wird von einem Runner ausgeführt. Für das Aufbauen des Projektes verwenden wir Maven und für das Senden bzw. Empfangen von Nachrichten-Paketen verwenden wir JSON. Zur Serialisierung bzw. Deserialisierung verwenden wir Gson als externe Library.

2.4 Entwicklungstools

GitLab - Als dezentrales Versionskontrollsystem wird Git genutzt, um das kooperative Arbeiten an dem Projekt zu ermöglichen. Es eignet sich zum effizienten Speichern von Datenzuständen, da Duplikate nicht genutzt werden.

GitLab CI - Als Teil des GitLab-Tools ermöglicht GitLab CI (Continuous Integration) den Entwicklungsprozess zu automatisieren. Durch die Definition von Pipelines, die bei Veränderungen im Repository automatisch ausgelöst werden, können Builds und Tests in einer vordefinierten Reihenfolge ausgeführt werden. Damit wird die Qualität der Software sichergestellt und potenzielle Probleme frühzeitig erkannt. GitLab CI ist außerdem kompatibel mit Docker, sodass Entwickler Docker-Container für ihre Builds und Deployments verwenden können.

Maven - Maven als ein für Java-Anwendungen konzipiertes Build-Management-Tool ermöglicht es, den Bauprozess zu unterstützen. Weitere Aufgaben wie die Organisation von Abhängigkeiten und das Testen von Code können mit Maven umgesetzt werden. Die Struktur und Übersichtlichkeit des Projekts werden durch die Verwendung von Maven-Konventionen sichergestellt.

UMLet - UMLet ist ein Werkzeug, mit dem UML-Diagramme entwickelt werden können. Mit Hilfe dieser bildlichen Darstellung diverser Diagramme, die der standardisierten Modellierungssprache (UML) untergeordnet sind, kann u.a. die Struktur und der Ablauf der Komponenten dargestellt werden.

Miro - Mithilfe von Miro als digitaler Kollaborationsplattform können Entwickler durch vielfältige Tools und Funktionen digitale Whiteboards erstellen, die besonders für die anfängliche Konzeption von Benutzeroberflächen geeignet sind.