Haplorhini Studios - Technologiepark 8 - 33100 Paderborn

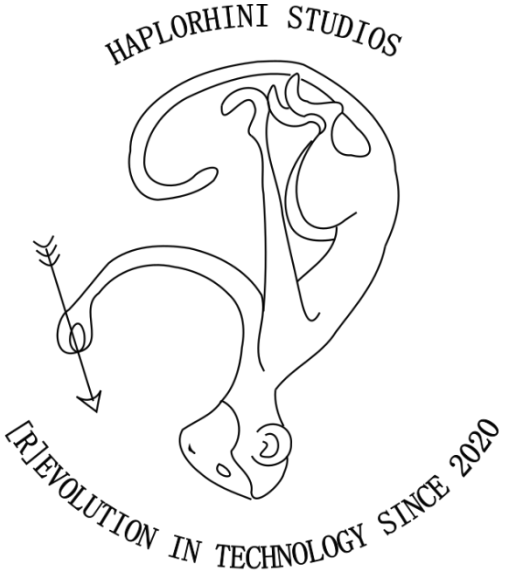
DevOps-Dokument

Softwaretechnikpraktikum 2020/21

Haplorhini Studios

Gruppe 05

“Amazonenspiel”



Changelog

Version 2 (05.02.2021):

- ⊕ [Development] UML-Klassendiagramm für die Turnier-Verwaltung hinzugefügt
- ⊕ [Development] Beschreibender Text zur Software-Architekur und möglichen Weiterentwicklungen für die Turnier-Verwaltung hinzugefügt
- ⊕ [Development] Neue Varianten des KI-Spielers wurden in das UML-Klassendiagramm und in den beschreibenden Text aufgenommen
- ⊕ [Development] Verwendete Libraries wurden an den neuesten Stand der Software angepasst
- ⊕ [Development] Namen der Komponenten korrigiert

- ⊕ [Operation] Standardparameter für den KI-Spieler und Spielserver hinzugefügt
- ⊕ [Operation] Zu Installations- und Betriebsanleitung wurden die entsprechenden Texte für die Turnierverwaltung hinzugefügt
- ⊕ [Operation] Beobachter-Screenshots auf den neusten Stand gebracht
- ⊕ [Operation] Abschnitt zu externen Tests sowie dafür verwendete Tools wurden dem DevOps-Prozess ergänzt

- ⊖ [Development] Pieces-Package entfernt, wird nicht mehr verwendet
- ⊖ [Development] Bewegungsanimationen für den Beobachter wurden aus den Weiterentwicklungsvorschlägen entfernt, da dieses Feature bereits umgesetzt wurde
- ⊖ Einige kleinere Textfehler wurden behoben

Version 1 (08.01.2021):

Vorherige Version zum Zeitpunkt der ersten Abgabe des DevOps-Dokuments

Inhaltsverzeichnis

1 Development

1.1 Produktkomponenten

Das Endprodukt besteht aus den Implementationen der verschiedenen Komponenten des Amazonen-Spiels, dem Test-Report zu der Software, einer Produkt-Website und weiteren Dokumenten, die das Produkt und den Projektablauf detaillierter beschreiben.

1.1.1 Softwareprodukt

Der Hauptbestandteil des zu entwickelnden Produkts ist die implementierte Software des Amazonen-Spiels. Diese setzt sich aus den Haupt-Komponenten KI-Spieler, Spielserver, Turnier-Verwaltung und Beobachter und aus den Hilfs-Komponenten Model, HttpsUtil und Validation zusammen. Auf den Aufbau der Gesamtarchitektur, sowie auf die Architektur der einzelnen Komponenten wird ausführlich in *1.2 Software-Architektur* und *1.3 Komponentenspezifikation* eingegangen.

1.1.2 Test-Report

Der Test-Report enthält eine Auflistung aller automatisierten und manuellen Tests, welche für die einzelnen Komponenten ausgeführt wurden. Zur KI-Spieler-Komponente wird noch einmal gesondert dargelegt, in welchem Umfang der Algorithmus der KI auf seine Spielfähigkeit getestet wurde.

1.1.3 Weitere Dokumente zum Produkt

Um die Eigenschaften, Nutzungsmöglichkeiten und mögliche Erweiterungen, sowie Verbesserungen der Software verständlich zu machen, werden die folgenden Dokumente bei der Endabgabe eingereicht.

Das **Quality-Assurance-Dokument** zur Beschreibung der Definition-of-Done und des Testplans, die zur Gewährleistung der Softwarequalität verwendet werden.

Das **Interface-Dokument,** um festzuhalten, welche Informationen wann und in welcher Form zwischen den Komponenten ausgetauscht werden.

Dieses **DevOps-Dokument**, welches neben einer Beschreibung des Aufbaus der entwickelten Software auch die Betriebs- und Installationsanleitung des Produkts beinhaltet.

1.1.4 Website

Die Website dient der Präsentation des Unternehmens und der von ihm geschaffenen Produkte. So kann der Kunde sich auf unserer Website über die Hintergründe von Haplorhini Studios informieren und herausfinden, welche Projekte aktuell in Arbeit sind. Auf der Produkte-Seite kann nachgelesen werden, wie genau die KI funktioniert und wie sie genutzt werden kann. Für die Entwicklung des Amazonenspiels kann sogar der wöchentliche Stand der Arbeiten über die Blogbeiträge mitverfolgt werden. Für weitere Rückfragen steht auch jeder Zeit die Kontakt-Seite auf der Website zur Verfügung.

Die in 1.1.3 erwähnten Dokumente werden ebenfalls auf der Website zur Verfügung gestellt.

1.2 Software-Architektur

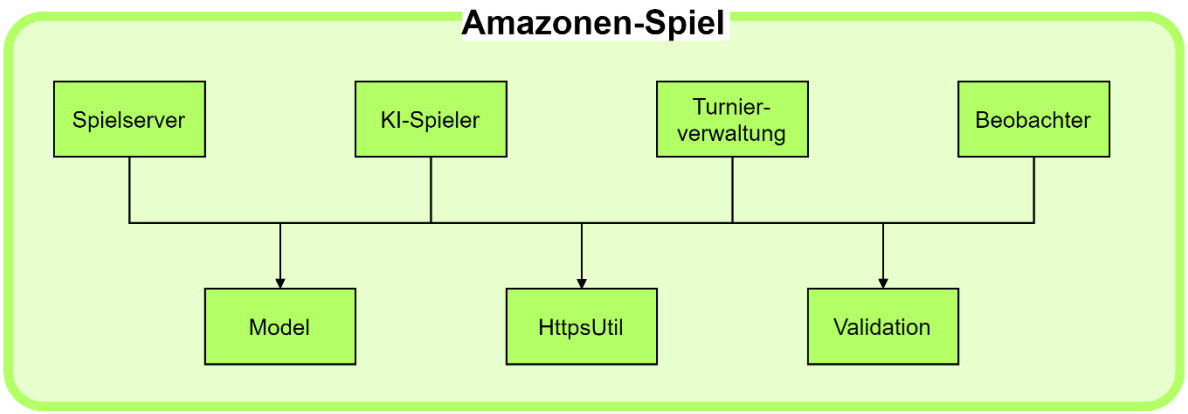


Abbildung 1: Komponentenstruktur des Projekts

Vorgegeben waren die Komponenten “Spielserver”, “KI-Spieler”, “Turnierverwaltung” und “Beobachter”. Diese kommunizieren über JSON-Objekte in String-Notation, welche über HTTPS-Requests verschickt werden. Für das Versenden von Anfragen wird dabei die HttpComponents[1] Bibliothek genutzt. Das Empfangen von Anfragen wird hierbei über die Java-eigene Komponente HttpsServer geregelt, zusätzlich wird hier die Commons[2] Bibliothek eingesetzt, um die Anfragen zu verarbeiten. Die Commons-Cli[3] Bibliothek wird genutzt um die Kommandozeilenargumente auf geordnete Weise zu parsen.

Für die Umsetzung der Anforderungen haben wir die vorgegebenen Komponenten um die “Model”-Komponente, die “Validation”-Komponente sowie die „HttpsUtil“-Komponente ergänzt.

Hierbei enthält die Model-Komponente alle Klassen, welche spielrelevante Informationen halten.

Die Validation-Komponente, welche zu großen Teilen auf der Bibliothek Gson[4] basiert, kümmert sich dabei um die Validierung und die Verarbeitung der übertragenen Daten.

Die HttpsUtil-Komponente sorgt für die Konfiguration der Kommunikation über HTTPS.

1.3 Komponentenspezifikation

1.3.1 KI-Spieler

Der KI-Spieler ist ein nichtmenschlicher Spieler, der das Amazonen-Spiel spielen kann. Mit diesem Ziel stellt er einen Webservice zur Verfügung, welcher von dem Spielserver aufgerufen werden kann. Sobald der KI-Spieler vom Spielserver nach seinem nächsten Spielzug gefragt wird, nutzt dieser einen Algorithmus, um den nächsten Zug zu berechnen und teilt diesen dann dem Spielserver mit.

[1] HttpComponents Bibliothek von der Apache Software Foundation
[2] Commons Bibliothek von der Apache Software Foundation
[3] Commons-Cli Bibliothek von der Apache Software Foundation
[4] Gson Bibliothek von Google

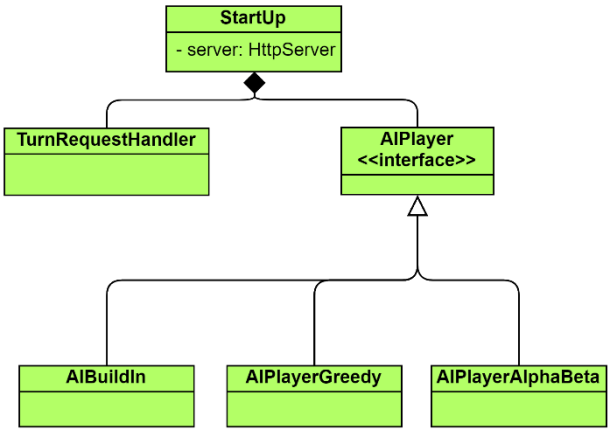


Abbildung 2: Grundlegender Aufbau der KI-Spieler-Komponente

Hier zu sehen sind die grundlegenden Klassen des KI-Spielers. Die StartUp-Klasse startet bei Start der Komponente einen Http-Server, wobei die TurnRequestHandler-Klasse herangezogen wird, um die eingehenden Anfragen zu verarbeiten. Außerdem wird eine Instanz einer Klasse, welche das AIPlayer-Interface implementiert erzeugt, wobei es sich um eine Klasse handelt, die mit Hilfe eines Algorithmus den nächsten Zug berechnet. Dieser muss mit der gegebenen Situation auf dem Brett vereinbar sein und auch möglichst zu einem späteren Sieg des Spiels beitragen.

Der Algorithmus der AIPlayer-Instanz ermittelt den zu machenden Zug, indem alle möglichen Züge bewertet werden und der beste Zug ausgewählt wird.

Die AIPlayerGreedy-Klasse berechnet einen Spielzug, indem berechnet wird, wie stark dieser die Bewegungsfreiheit des Kontrahenten im Verhältnis zur eigenen einschränkt. Als Metrik für die Bewegungsfreiheit wird die Anzahl der erreichbaren Spielfelder herangezogen. Hierbei findet eine Gewichtung statt. Spielfelder, die in weniger Schritten erreichbar sind, erhalten eine höhere Wertung. Der Grad, der beschreibt, wie wichtig eine hohe Einschränkung der Bewegungsfreiheit des Kontrahenten gesehen wird, wird als Aggressivität bezeichnet. Eine geringe Aggressivität bedeutet eine Bevorzugung einer hohen eigenen Bewegungsfreiheit, eine hohe Aggressivität bedeutet eine Bevorzugung einer hohen Einschränkung der Bewegungsfreiheit des Kontrahenten.

Die AIPlayerAlphaBeta-Klasse, ist eine weitere Klasse, welche optional statt der AIPlayerGreedy-Klasse verwendet werden kann. Diese schlägt nach einem Alpha-Beta-Pruning-Verfahren[5] den nächsten Zug vor, nutzt sonst aber den gleichen Algorithmus wie die zuvor beschriebene AIPlayerGreedy-Klasse.

Die AIBuildIn-Klasse, ist ebenso eine weitere Klasse, welche optional statt der AIPlayerGreedy-Klasse verwendet werden kann. Diese verwendet das Alpha-Beta-Pruning-Verfahren sowie ein eigenes Verfahren, welches darauf abzielt, das eigene Überleben zu sichern, in dem die eigenen Königin-Figuren sich möglichst effizient selbst einbauen sollen. Zuerst wird dabei das Alpha-Beta-Pruning-Verfahren verwendet, gewechselt wird, wenn auf dem Spielbrett eine gewisse Anzahl von freien Feldern unterschritten wurde.

Verwendet werden die Bibliotheken GSON, Commons und Commons-Cli.

Um den KI-Spieler weiterzuentwickeln, kann die Gewichtungsfunktionen und die Aggressivität der KI, nach dem in *2.3 Beschreibung des DevOps-Prozesses* für die KI beschriebenen Optimierungsverfahren, weiter verbessert werden. Außerdem wäre es möglich weitere Alternativ-Verfahren, wie es auch schon bei dem

[5] Alpha-Beta-Pruning: https://de.wikipedia.org/wiki/Alpha-Beta-Suche

Alpha-Beta-Pruning-Verfahren geschehen ist, durch weitere Klassen, die das AIPlayer-Interface implementieren, umzusetzen.

1.3.2 Spielserver

Auf dem Spielserver werden die Spiele zwischen den KI-Spielern konfiguriert und ausgetragen. Es können gleichzeitig mehrere Spiele ausgeführt werden. Der Spielserver fragt abwechselnd bei beiden Spielern ihren Zug an und prüft dann, ob der zurückgesendete Spielzug gültig ist. Dabei protokolliert er alle Nachrichten, die ausgetauscht wurden.

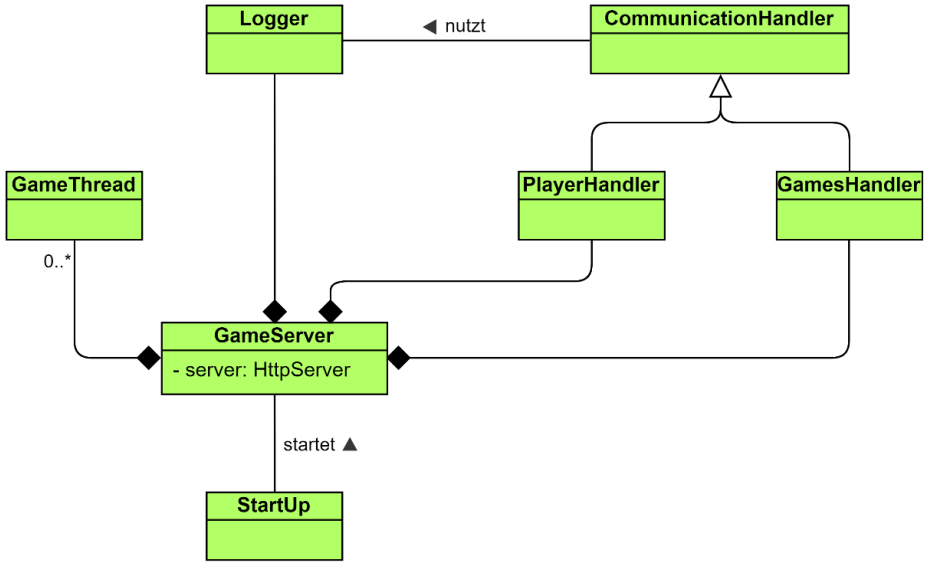


Abbildung 3: Grundlegender Aufbau der Spielserver-Komponente

Hier zu sehen sind die Klassen, welche den Grundaufbau des Spielservers bilden. Zu Beginn erzeugt die StartUp-Klasse eine Instanz der GameServer-Klasse, welche dann wiederum einen HTTPS-Server, den sie von der HttpsUtil-Komponente erhält, startet. Zur Verarbeitung ankommender Anfragen werden die Klassen PlayerHandler und GamesHandler herangezogen. Diese erben von der Klasse CommunicationHandler, die ihnen den Zugriff auf nützliche Hilfsmethoden bezüglich der Kommunikation über HTTPS und eine Instanz der Logger-Klasse gewährt. Die Logger-Klasse kümmert sich um eine formgerechte Führung eines Logs in Textform.

Für den Ablauf des sogenannten „Gameloops“[6] des Amazonenspiels ist die GameThread-Klasse zuständig. Diese erbt dabei von der Java-eigenen Klasse Thread[7] und kümmert sich um die Ausführung genau eines Gameloops. Für ausgehende Requests hier wird die HttpsUtil-Komponente herangezogen.

Verwendet werden die Bibliotheken GSON, Commons, Commons-Cli und HttpComponents.

Für den Fall, dass die API erweitert werden soll, fügt man einfach weitere Klassen, die von CommunicationHandler erben, hinzu.

1.3.3 Turnier-Verwaltung

Mit der Turnier-Verwaltung kann ein Turnier ausgerichtet werden. Dafür löscht diese vor dem Start eines Turniers alle vorhandenen Spiele, welche auf dem Spielserver vorhanden sind und prüft, ob alle

[6] Gameloop: https://de.wikipedia.org/wiki/Game_Loop
[7] Thread: https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

teilnehmenden KI-Spieler erreichbar sind. Während des Turniers protokolliert sie alle Spielergebnisse sofort in einer Datei.

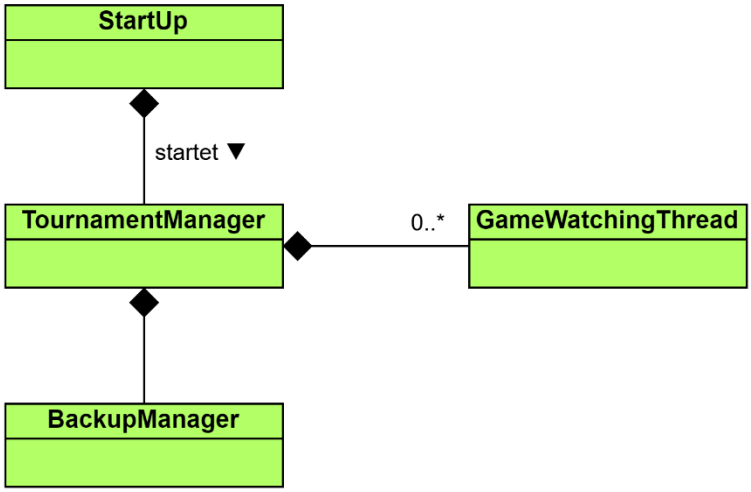


Abbildung 4: Grundlegender Aufbau der Turnier-Verwaltung-Komponente

Hier zu sehen sind die Klassen, welche den Grundaufbau der Turnier-Verwaltung bilden. Zu Beginn erzeugt die StartUp-Klasse eine Instanz der TournamentManager-Klasse, welche sich dann um das Starten der Spiele kümmert. Für die Sicherung und Wiederherstellungsfähigkeiten erzeugt die TournamentManager-Instanz eine Instanz der BackupManager-Klasse und zieht die, von dieser angebotenen, Methoden heran.

Für jedes auf dem Spielserver gestartete Spiel wird eine Instanz der Klasse GameWatchingThread erzeugt, welche den Verlauf jeweils eines Spiels auf dem Spielserver beobachtet und bei Ende des Spiels das Ergebnis an die TournamentManager-Instanz zurück gibt.

Die GameWatchingThread-Klasse erbt hierbei von der Java-eigenen Klasse Thread.

Für ausgehende Requests wird die HttpsUtil-Komponente herangezogen, für das Verarbeiten der Antworten wird die Validation-Komponente verwendet.

Verwendet werden die Bibliotheken GSON, Commons, Commons-Cli und HttpComponents.

Eine Möglichkeit zur Weiterentwicklung wäre die Erweiterung um weitere Turnier-Modi, wie zum Beispiel einen KO-Modus, in welchem zufällige Paarungen der Spieler generiert werden und der Verlierer jeweils ausscheidet.

1.3.4 Beobachter

Der Beobachter besitzt eine grafische Benutzeroberfläche, welche eine Übersicht über die Spiele, den Spielablauf eines ausgewählten Spiels oder eine Liste aller Spieler mit der Anzahl der gewonnen Spiele darstellen kann. Dafür fragt der Beobachter die entsprechenden Informationen regelmäßig vom Spielserver ab.

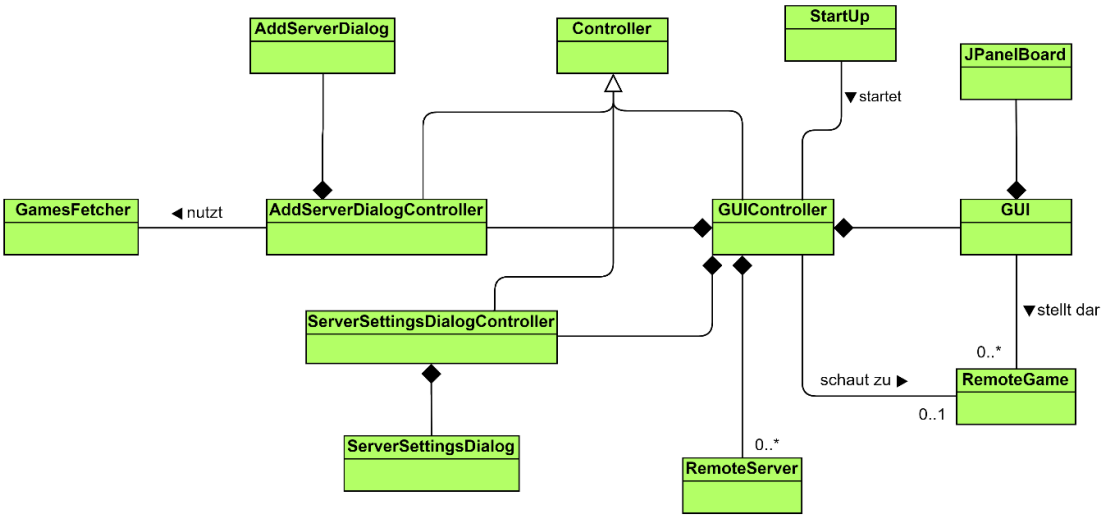


Abbildung 5: Grundlegender Aufbau der Beobachter-Komponente

In dieser Abbildung sind jene Klassen dargestellt, die den Grundaufbau des Beobachters bilden. Aus Gründen der Übersichtlichkeit wurden einige Klassen weggelassen, die für das Verständnis der Gesamt-Architektur nicht notwendig sind.

Zu Beginn erzeugt die StartUp-Klasse eine Instanz der GUIController-Klasse, welche dann wiederum eine Instanz der GUI-Klasse erzeugt. Die GUI-Klasse generiert dann das Programm-Fenster und instanziiert die Oberflächenelemente.

Die Interaktionen mit den Oberflächenelementen der GUI werden an die GUIController-Instanz weitergereicht.

Über den AddServerDialog, welcher vom AddServerDialogController verwaltet wird, werden im GUIController Repräsentationen aller Spielserver, mit denen sich verbunden wird, abgelegt. Die Repräsentation eines Spielservers wird über die RemoteServer-Klasse vorgenommen. Für das Laden aller Spiele eines Spielservers wird die Klasse GamesFetcher herangezogen.

Über den ServerSettingsDialog, welcher vom ServerSettingsDialogController verwaltet wird, werden die Spiele des Spielservers ausgewählt, welche zur Auswahl zum Zuschauen zur Verfügung stehen sollen.

Verwendet werden die Bibliotheken GSON, Commons und HttpComponents.

1.3.5 Model

Die Model-Komponente umfasst die meisten spielbezogenen Klassen. Diese stehen den anderen Komponenten dabei frei zur Verfügung. Das Modul besteht aus folgenden Packages:

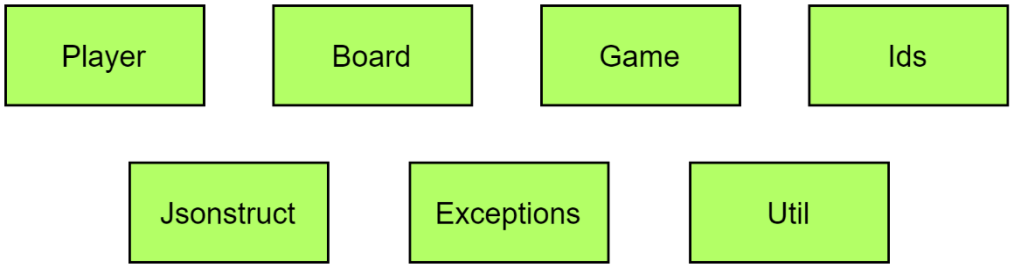


Abbildung 6: Die sieben Packages der Komponente

Die Komponente Model, umfasst die Datenstrukturen bzw. Klassen, welche sich direkt aus der Schnittstellenspezifikation ergeben. Da diese Datenstrukturen in allen bzw. mehreren Komponenten benötigt werden, wurden sie in einer extra dafür angelegten Komponente untergebracht.

- In Player befinden sich dabei die Klassen, welche für die Repräsentation eines Spielers und seinen Zügen zuständig sind.
- In Board befinden sich die Klassen, die direkt für die Repräsentation des Spielbretts zuständig sind.
- In Game befinden sich die Klassen, die für die Repräsentation eines Spielobjekts zuständig sind.
- In Ids befinden sich die Klassen, die für die Ids der Spieler, der Spiele und der Züge zuständig sind.
- In Jsonstruct befinden sich alle Klassen, die als Schemata genutzt werden, um JSON-Strings zu parsen oder zu erzeugen.
- In Exceptions befinden sich die Fehlerklassen, welche sich aus dem direkten Umgang mit den Spielen, Spielern, Zügen und Figuren ergeben. Dazu gehört beispielsweise der Fehler, dass eine Spiele-Id bereits vergeben ist.
- In Util befinden sich einige Utility-Klassen[8] die für die anderen Klassen von Nutzen sind.

1.3.6 Validation

Mit der Validation-Komponente werden eingehende Nachricht auf korrekte Formatierung und Vollständigkeit geprüft und dabei in die internen Datenstrukturen übertragen.

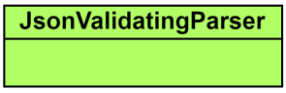


Abbildung 7: Die Hauptklasse der Validation-Komponente

Die Komponente besteht dabei aus der JsonValidatingParser-Klasse, welche mithilfe der Bibliothek Gson und weiteren Hilfsklassen und Annotations[9], JSON-Objekte parsen und gleichzeitig validieren (d.h. auf Vollständigkeit und korrekte Form prüfen) kann.

Für die Validierung werden die Klassen, welche es zu parsen gilt, mit speziellen Annotations markiert und dann dem JsonValidatingParser übergeben.

1.3.7 HttpsUtil

Die HttpsUtil-Komponente ermöglicht den anderen Komponenten das Erstellen eines konfigurierten HTTPS-Servers. Zudem wird auch eine Möglichkeit angeboten, um Anfragen über HTTPS zu verschicken.

Abbildung 8: Die zwei Klassen dieser Komponente

Hier zu sehen sind die beiden Klassen der HttpsUtil-Komponente.

Die HttpClientFactory-Klasse bietet über die getNewHttpsClient-Methode eine fertig konfigurierte Instanz der Klasse CloseableHttpClient[10] an.

[8] Utility-Klassen: https://mein-javablog.de/java-utility-klassen/
[9] Annotation: https://de.wikipedia.org/wiki/Annotation_(Java)
[10] Klasse aus der Bibliothek Commons

Die HttpsServerFactory-Klasse bietet über die getNewHttpsServer-Methode eine Instanz der Java-eigenen Klasse HttpsServer an, welche bereits für die Nutzung von HTTPS konfiguriert ist. Die Instanzen der HttpsServer-Klasse werden dann noch für jede Komponente mit den Listenern versehen, die in den Komponenten benötigt werden.

Für die HttpClientFactory-Klasse wird der Http-Client der Bibliothek HttpComponents verwendet.

Die Klasse HttpsServerFactory nutzt die Java-eigene Bibliothek zu HTTPS-Servern[11].

[11] Von Oracle Sun veröffentlichte Bibliothek H`ttpserver`

2 Operations

2.1 Betriebsanleitung

Im folgenden Abschnitt wird erläutert, wie die Software-Komponenten benutzt werden können.

2.1.1 KI-Spieler

Der KI-Spieler wird über ein Terminal durch die Ausführung seiner Jar-Datei gestartet. Hierbei können ein Hostname[12] und ein Port übergeben werden. Dies geschieht über die Prozessargumente „-hostname“ und „-port“. Standardmäßig wird der Hostname „localhost“ und der Port „33098“ genutzt. Über die Angabe von „-insecure“ kann die Verwendung von HTTPS (bzw. TLS) deaktiviert werden, es wird dann auf HTTP gewechselt.

Sobald der KI-Spieler gestartet ist läuft ein Webserver und der KI-Spieler ist damit bereit auf eingehende Nachrichten zu reagieren.

2.1.2 Spielserver

Der Spielserver wird über ein Terminal durch die Ausführung seiner Jar-Datei gestartet. Hierbei können ein Hostname und ein Port sowie ein Zugriffs-Token übergeben werden. Dies geschieht über die Prozessargumente „-hostname“ und „-port“ sowie „-token“. Standardmäßig wird der Hostname „localhost“, der Port „33100“ und das Token „31415926535897932384626433832795“ verwendet.

Wahlweise kann noch über die Angabe von „-debug“ erreicht werden, dass erweiterte Konsolenausgaben aktiviert werden. Über die Angabe von „-traffic“ kann diese Konsolenausgabe um einen Mitschnitt der eingehenden und ausgehenden Anfragen ergänzt werden. Über „-insecure“ kann die Verwendung von HTTPS (bzw. TLS) deaktiviert werden.

Sobald der Spielserver gestartet ist, läuft ein Webserver und der Spielserver ist damit bereit, auf eingehende Nachrichten zu reagieren.

2.1.3 Turnier-Verwaltung

Die Turnier-Verwaltung wird über ein Terminal durch die Ausführung seiner Jar-Datei gestartet. Dabei wird eine Liste von Spielern in Textform übergeben und wahlweise noch die Netzwerkadresse sowie das Zugriffs-Token des Spielerservers, sowie einen Alternativspeicherpfad für die Backupdateien. Dies geschieht über die Prozessargumente „-players“, „-address“ sowie „-token“ und „-backupdir“. Standardmäßig wird die Netzwerkadresse „https://localhost:33100“ und das Token „31415926535897932384626433832795“ verwendet.

Die Spieler werden standardmäßig aus der Datei „spieler.txt“ des Verzeichnisses, in welchem die Jar-Datei ausgeführt wird, geladen.

Die Datei mit der Liste der Spieler muss dabei nach dem folgenden Schema formatiert sein:

```
<Name 1>,<Adresse 1>
<Name 2>,<Adresse 2>
[…]
```

(Also beispielsweise könnte die erste Zeile wie folgt aussehen: Johny Boy,https://localhost:33098)

[12]Hostname: https://de.wikipedia.org/wiki/Hostname

Es wird also in jeder Zeile der Datei genau ein Spieler angegeben. Spielername und Spielernetzwerkadresse, getrennt durch ein Komma, bilden hierbei einen Spieler.

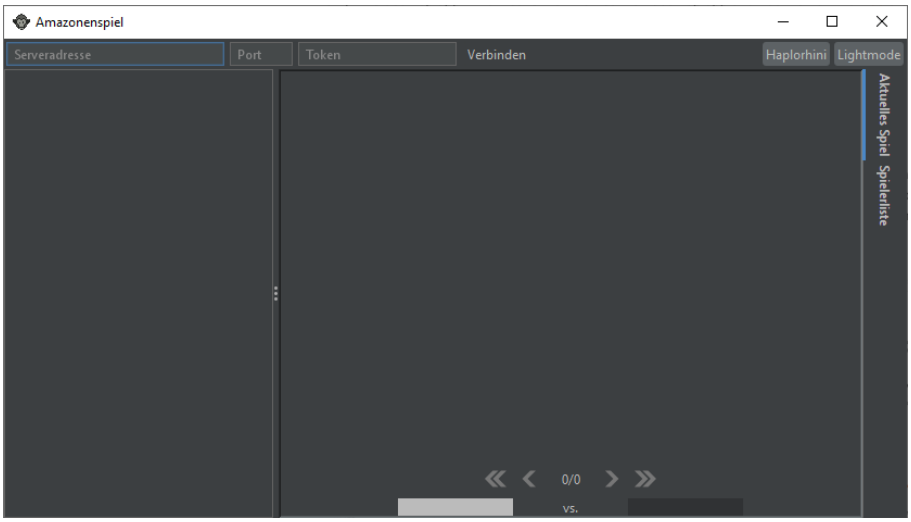
Die Backupdateien werden, falls nicht anders bestimmt, in dem Verzeichnis gespeichert, in welchem die Jar-Datei ausgeführt wird.

Sobald die Turnierverwaltung gestartet ist, wird direkt ein Turnier auf dem Spielserver veranstaltet. Ein weiteres Eingreifen durch den Endnutzer ist nicht erforderlich.

2.1.4 Beobachter

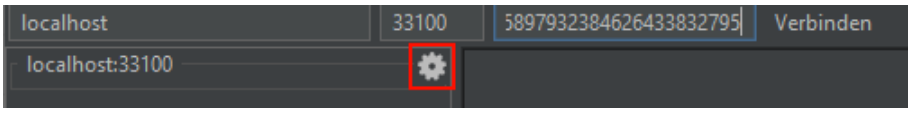
Gesamtsicht:

Der Beobachter wird durch einen Doppelklick seiner Jar-Datei gestartet. Um sich mit einem Spielserver zu verbinden, trägt man die Serveradresse, den Port und den Zugriffs-Token in die obere Leiste ein und klickt anschließend auf „Verbinden“.
Bei erfolgreichem Verbinden erscheint dann ein Server-Eintrag wie in der folgenden Abbildung zu sehen.



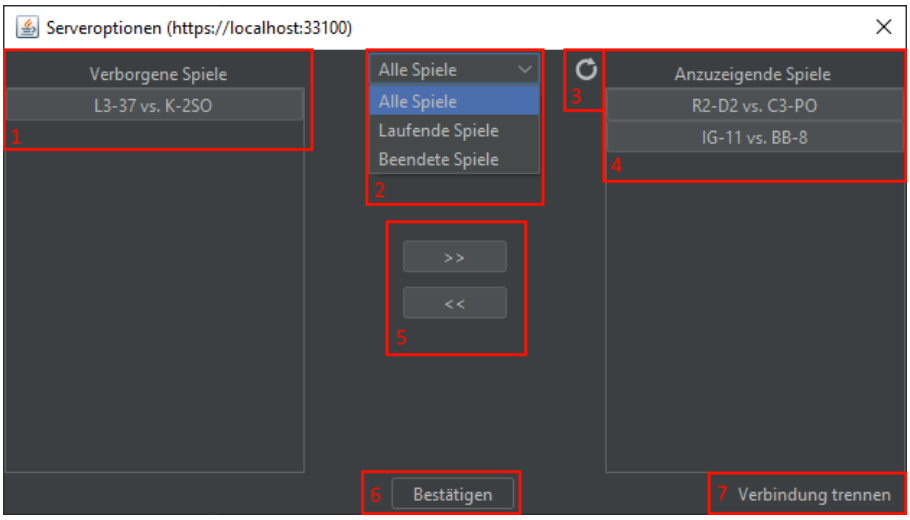
Serverkonfiguration:

Durch Klick auf das Zahnrad eines Servers lässt sich das Pop-Up öffnen, welches im Folgenden genauer erläutert wird.



Spieleselektion:

Die Spiele eines Servers sind in verborgen und sichtbar unterteilt. Es ist dabei möglich den sichtbaren Spielen zuzuschauen, verborgene Spiele bleiben versteckt.



Markierung 1:
Liste aller verborgenen Spiele. Durch Anklicken eines Spiels wechselt dieses zu den sichtbaren Spielen.

Markierung 2:
Aufklappbares Menu, erlaubt einem auszuwählen ob nur bestimmte Spiele für die Auswahl angezeigt werden sollen.

Markierung 3: Schaltfläche zum Aktualisieren der Liste von Spielen

Markierung 4: Liste aller angezeigten Spiele. Durch Anklicken eines Spiels wechselt dieses zu den verborgenen Spielen.

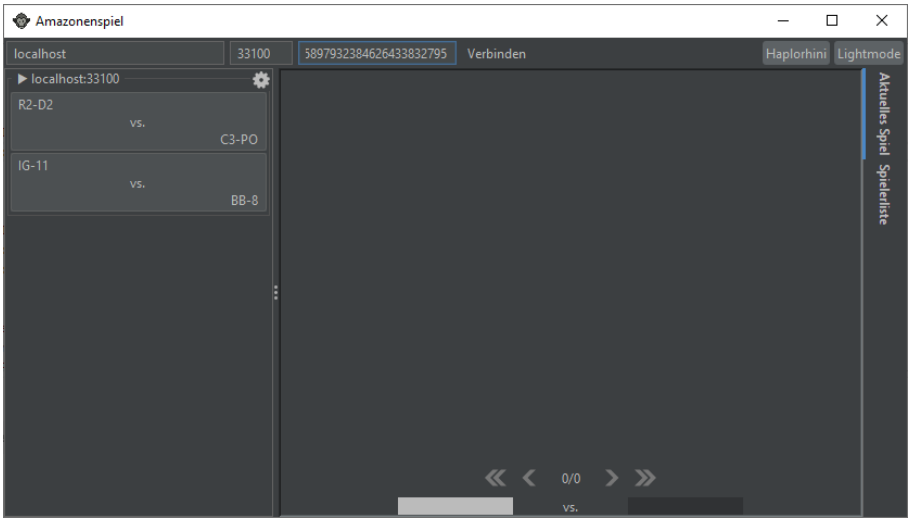
Markierung 5: Mit diesen Schalt-flächen ist es möglich alle Spiele auf eine Seite zu bewegen.

Markierung 6: Sind alle Spiele wie gewünscht ausgewählt schließt man das Fenster mit dem X oder mit einem Klick auf „Bestätigen“.

Markierung 7: Über einen Klick auf „Verbindung trennen“ kann der Server wieder aus der Liste der verbundenen Server entfernt werden.

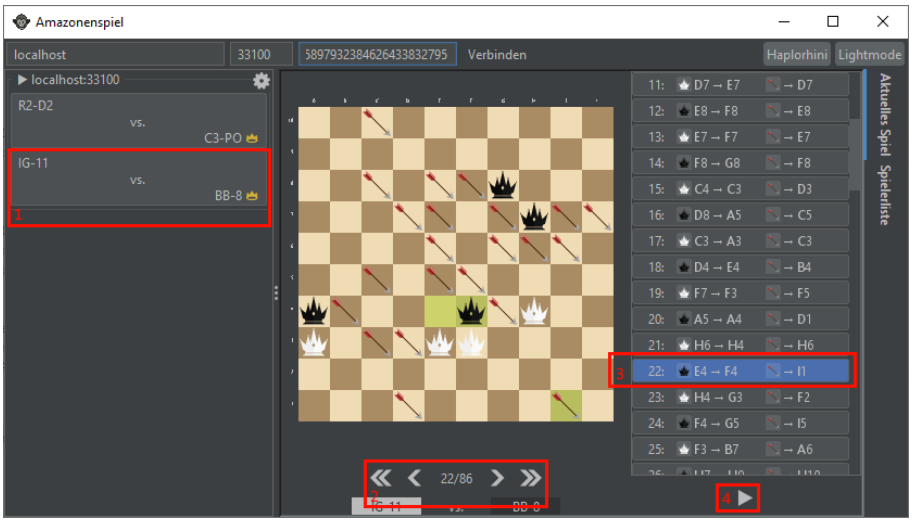
Geöffnete Spieleliste:

Durch einen Klick auf einen Server ist es möglich die Liste aller aktiven Spiele dieses Servers auszuklappen.
Durch einen erneuten Klick auf den Server wird die Liste aller aktiven Spiele des Servers wieder eingeklappt.
Ein Dreieck-Symbol vor dem Servernamen zeigt an, ob die Spielliste ausgeklappt (▶) oder eingeklappt (▼) ist.



Spielansicht:

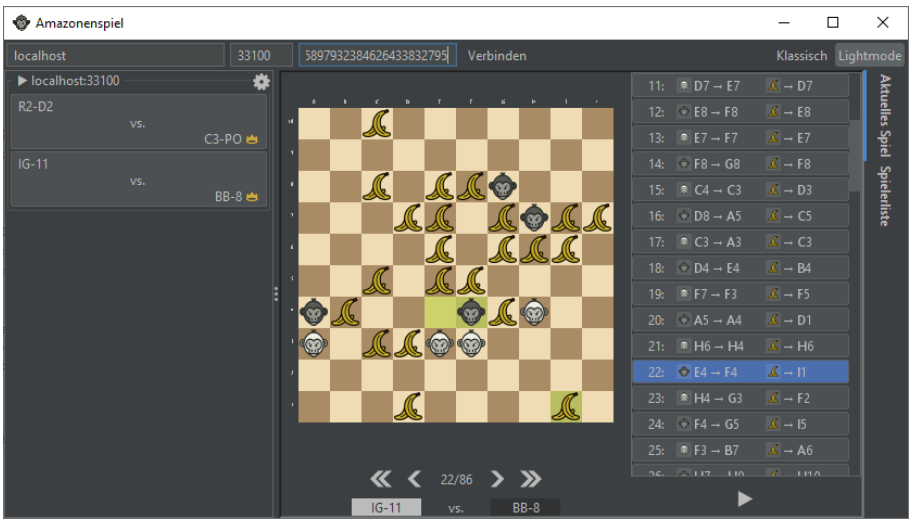
Durch einen Klick auf ein Spiel **(1)** in der Spielliste öffnet sich nun das Spielbrett des Spiels. Hinweis: Die goldene Krone markiert den Sieger eines Spieles.



Mithilfe der Schaltflächen in **(2)** lässt sich in der Historie des Spielbretts umherspringen. Die äußeren Pfeile stehen für den Sprung zum ersten bzw. zum aktuellen Brett. Die Pfeile in der Mitte stehen für einen Schritt vor bzw. zurück.
Außerdem ist es möglich in der Folge der Schritte einen einzelnen Schritt auszuwählen **(3).** Dann wird zu diesem direkt gesprungen und das Spielbrett zu diesem Zeitpunkt wird angezeigt.
Über **(4)** ist es möglich, die Folge von den Spielzügen bis zum Ende abzuspielen und nachzuvollziehen.

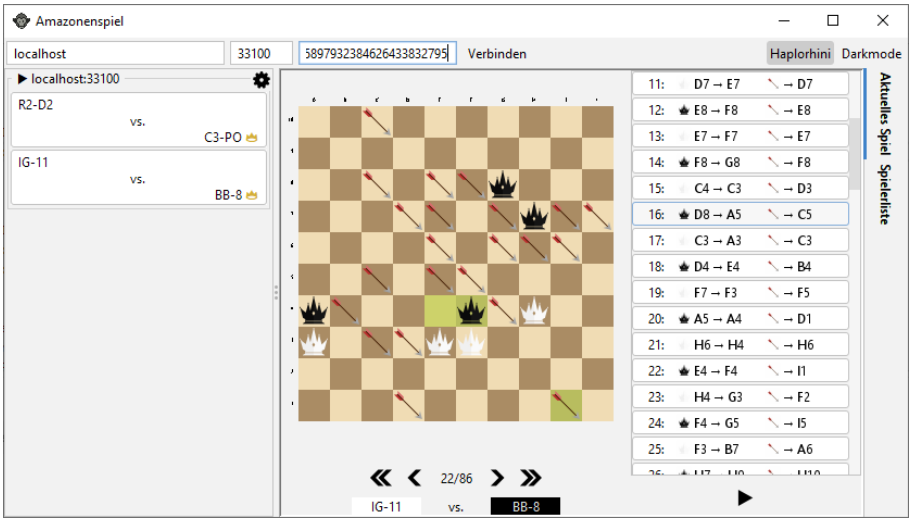
Haplorhini-Brettdesign:

Durch einen Klick auf die Schaltfläche, in der Haplorhini steht (oben rechts), wird zu dem Haplorhini-Brettdesign gewechselt. In diesem werden die Damen durch Affen und die Pfeile durch Bananenschalen dargestellt. Der Text in der Schaltfläche hat sich dabei zu Klassisch geändert.
Durch einen erneuten Klick auf die Schaltfläche, in welcher nun der Text Klassisch steht, wird zu dem vorherigen Klassisch-Brettdesign zurückgewechselt.



Lightmode-Hintergrunddesign:

Durch einen Klick auf die Schaltfläche, in der Lightmode steht (oben rechts), wird in den Lightmode-Hintergrunddesign gewechselt. In diesem ändert sich die Farben der Hintergrund-flächen, von vorher dunklen, zu weißen Farbtönen.
Der Text in der Schaltfläche hat sich dabei zu Darkmode geändert.

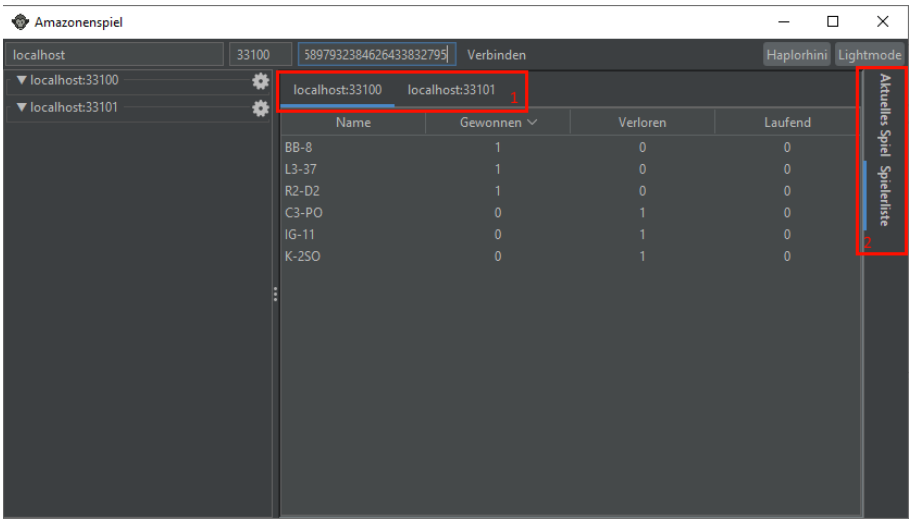


Durch einen erneuten Klick auf die Schaltfläche, in welcher nun der Text Darkmode steht, wird zu dem vorherigen Darkmode-Hintergrund-Design zurückgewechselt.

Spielerliste-Ansicht:

Durch Klick auf Spielerliste **(2)** wechselt der Beobachter zu einer Ansicht, in der die momentanen Spielerplatzierungen betrachtet werden können.

In **(1)** kann man zwischen den Spielservern wechseln, mit welchem der Beobachter ver-bunden ist. Zu Illustrations-zwecken ist hier ein weiterer Spielserver künstlich der Ansicht hinzugefügt worden.



2.2 Installationsanleitung

Für die Installation der Komponenten ist zuerst eine Installation von Gradle notwendig. Anschließend muss das AmazonenSpiel-Projekt durch Gradle importiert werden. Durch den Gradle-Task „jar“ können

dann ausführbare Jar-Dateien vom Spielserver, Beobachter und KI-Spieler generiert werden. Auch die Turnier-Verwaltungs-Komponente wird voraussichtlich so installiert werden können.

Um die SSL-Zertifikate der Komponenten anzupassen bzw. zu ersetzen muss dem mit dem Tool „Keytool[13]“ eine sogenannte Keystore-Datei erstellen. Hierfür kann der folgende Befehl genutzt werden:

```
keytool -genkeypair -keyalg RSA -alias self_signed -keypass <Passwort>
 -keystore <Dateiname> -storepass <Passwort>
```

Zu beachten ist, dass die Felder „Passwort“ und „Dateiname“ (ohne < und >) hier mit den Daten aus der Klasse HttpsServerFactory aus der HttpsUtil-Komponente übereinstimmen müssen. Standardmäßig wird das Passwort “securepassword” und der Dateiname “amazonenspiel.keystore” verwendet. Die Keystore-Datei muss dann in den jeweiligen “Resource”-Ordnern innerhalb der Komponenten-Verzeichnisse kopiert werden.

Ausgeführt werden können die Komponenten über ein Starten der generierten Jar-Dateien. Ein Anpassen der Parameter für den KI-Teil des KI-Spielers kann in seiner StartUp-Klasse vorgenommen werden.

2.3 Beschreibung des DevOps-Prozesses

In diesem Textabschnitt folgt eine textuelle Erläuterung bezüglich des bisherigen DevOps-Prozesses zum Testen und Bauen des Produkts. Insbesondere wird auf die Nutzung von GitLab CI/CD und das Projektverwaltungstool Gradle, sowie das verwendete Testframework JUnit eingegangen. Anschließend folgt eine Beschreibung des Test- und Optimierungsverfahrens der KI-Komponente.

Für die Durchführung von automatischen Tests im Projekt wird die GitLab CI/CD Funktion verwendet. Individuell für jeden Branch wurden speziell eingerichtete Pipelines angelegt, da die verschiedenen Komponenten auf unterschiedlichen Branches entwickelt werden. Die Pipelines werden nach jedem Push aktiviert und führen anschließend eine Reihe von aufeinander aufbauenden “Jobs” mit einem Runner auf der gestellten Virtual Machine der Universität aus, sodass unerwünschte Änderungen im Verhalten der Software mit dem Feedback der Pipeline direkt ermittelt werden können.

Bei den ausgelösten Jobs handelt es sich um festgelegte Gradle Tasks, welche unter Nutzung des Management-Automatisierungstool Gradle durchgeführt werden. Die Gradle Tasks umfassen unter anderem den Build-Prozess der einzelnen Komponenten unter Beachtung der Klassen und Package-Dependencies, sowie die Ausführung und Auswertung der angelegten JUnit Tests mit detaillierten lokalen HTML-Reports.

Auf dem Branch, der alle Produktkomponenten vereint, folgen aktuell zum Beispiel, abhängig vom Build-Prozess des Servers, Tests zu den Packages Server, Validation, Model und KI-Spieler. Speziell hierfür liegen für beinahe jede Klasse ausführliche JUnit Tests vor, die die Funktionalität aller ihrer relevanten Methoden untersuchen. Idealerweise prüft ein Entwickler jedoch vor jedem Push auch selbst, ob die vorliegenden Tests

[13] Keytool: https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

erfolgreich sind, denn das Durchführen aller Tests mit Gradle ist in den meisten Java Entwicklungsumgebungen mithilfe von “run configurations” leicht reproduzierbar.

Nach der Ausführung aller Gradle Tasks generiert die Pipeline einsehbare Test-reports, sogenannte “artifacts”, mitsamt Testergebnissen und Laufzeiten. Sofern ein Fehler aufgetreten ist, lässt sich auch das Stacktrace der Konsole für Debugging Zwecke auf GitLab anzeigen lassen.

Neben der Testautomatisierung wird der “build” Befehl von Gradle auch für die Generierung ausführbarer jar-Dateien genutzt.

Zum Testen und Optimieren des KI-Spielers, wurde ein 16-Kern Google Server, angeboten von Google Cloud Plattform, für beschränkte Zeitfenster angemietet[14]. Dafür wurde ein externes Turnierskript angelegt, welches mehrere KI-Spieler mit unterschiedlichen Gewichtungsfunktionen und Aggressivitäten in mehreren tausenden Spielen gegeneinander antreten lässt und daraus eine übersichtliche Zusammenfassung der relevanten Spielwerte generiert (die Gewinnrate als Weiß, Schwarz und deren Mittelwert, die maximale, minimale und durchschnittliche Zuganzahl eines gewonnenen Spiels und die durchschnittliche Bedenkzeit eines Zuges). Jeder Testreport wird direkt auf GitLab hochgeladen.

Das Optimierungsverfahren verläuft nach dem folgenden Schema: Von den KI-Spielern mit unterschiedlichen Parametern werden die mit der höchsten Gewinnrate ausgewählt. Anschließend werden leicht abgewandelte Variationen (also leicht andere Gewichtungsfunktion und/oder Aggressivität) im Turniersystem auf ihre Stärke überprüft. Durch dieses Verfahren lässt sich phasenweise das stärkste Gewichtungs- und Aggressivitätsverhältnis ermitteln. Mithilfe der errechneten Spielwerte wurden so auch starke Kombinationen an KI-Spielern aus der optimalen Gewinnrate für weiße und schwarze Damen entwickelt.

Ergänzend zu den automatischen Unit-Testprozessen sind externe Tests angelegt worden, mit dem Ziel sicherzustellen, dass die Netzwerkkommunikation ordnungsgemäß und vorhersehbar stattfindet. Somit wird abgedeckt ob der Server und der KI-Spieler auch in der Praxis API-konform handeln und nicht nur die theoretischen Unit-Tests bestehen, falls vorhanden. In diesen Tests wird die Korrektheit der Rest-API hinsichtlich auf folgende Gesichtspunkte geprüft: Verwerfen von ungültigen Nachrichten, Annahme und Exekution gültiger Nachrichten, Ausgabe passender Errorcodes und dem generellen Aufbau einer Verbindung. Der Programmcode hierfür wurde in Rust geschrieben, da sich, unter anderem durch persönliche Präferenzen geprägt, ein Umgang mit den sogenannten crates (libraries) als besonders unkompliziert im Gegensatz zu den entsprechenden Java Libraries erwies. Ursprünglich sollten diese Tests auf der Pipeline liegen. Dieser Vorschlag wurde aufgrund von Zeitmangel und dementsprechender Prioritätenzuordnung nicht umgesetzt.

2.4 Verwendete Tools für die Entwicklung

2.4.1 Projektmanagement

BigBlueButton: Für die wöchentlichen Online-Meetings der Teammitglieder mit dem Betreuer wurde BigBlueButton genutzt. Die Presenter-Funktion spielt hierbei eine besondere Rolle, da sie die Präsentation und Erläuterung von Ideen und neuen Entwicklungen, über einen vom Teammitglied geteilten Bildschirm, visuell ergänzt.

[14] https://cloud.google.com/free/docs/gcp-free-tier

Discord: Ein für das Projekt erstellter Discord-Server wurde für die weitere Kommunikation zwischen den Teammitgliedern verwendet. Gesonderte Channels sorgen dabei für eine klare Trennung von Aufgaben, Informationen und Prioritäten. Die verschiedenen virtuellen Sprechräume wurden für individuelle Gruppenaufgaben auch außerhalb der geplanten Meetings genutzt.

Kanboard: Das Product Backlog wurde über die Online-Plattform Kanboard angelegt. Alle Teammitglieder haben jederzeit Zugriff auf die Informationen des digitalen Kanban-Boards. Der Entwicklungsstand wird im Laufe aller Projektphasen und Sprints regelmäßig auf dem Kanban-Board festgehalten und aktualisiert.

Lucidchart: Zur Organisation der gemeinsamen Entwicklung ist das Web-Tool Lucidchart verwendet worden, mit dem verschiedenste (UML) Diagramme parallel von mehreren Nutzern erstellt und angepasst werden können.

2.4.2 Softwareentwicklung

Eclipse IDE und IntelliJ IDEA: Als Entwicklungsumgebung zum Implementieren, Testen und Debuggen von Sourcecode werden diese beiden IDEs verwendet. So werden zum Beispiel lokale JUnit Tests über die IDE durchgeführt. Die Entscheidung für eine der beiden Entwicklungstools obliegt den persönlichen Präferenzen des einzelnen Nutzers.

WindowBuilder: Hierbei handelt es sich um ein PlugIn für die Erstellung der GUI.

GitLab: Auf dem universitätseigenen GitLab-Webserver werden alle projektbezogenen Daten gesammelt und Versionen verwaltet. Dazu gehören Quellcode, Abgabedokumente, Test- und Konfigurationsdateien. Die Nutzung von GitLab birgt das Ziel, die gemeinsame Softwareentwicklung weitestgehend zu parallelisieren und übersichtlich zu halten.

GitLab CI/CD: Die Continuous-Integration&Deployment Funktion von GitLab wurde insbesondere für die Durchführung automatischer Tests im Projekt eingerichtet.

Gradle: Das bekannte Management-Automatisierungstool Gradle wird unter anderem zum lokalen Ausführen von JUnit-Tests, der Generierung von detaillierten HTML-Reports und lauffähigen jar-Dateien verwendet.

JUnit: Das verwendete Testframework für alle Modultests ist JUnit.

Virtual Maschine der Universität: Zum Ausführen der „Jobs“ in den Pipelines, welche nach einem Push ausgelöst werden, wird eine Virtual Maschine der Universität verwendet.

Google Cloud Plattform: Zum Testen und Optimieren der KI wurde ein 16-Kern Google Server für ein beschränktes Zeitfenster angemietet und für ein Turniersystem genutzt.

Python (programming language): Zur schnellen Entwicklung eines funktionierenden Prototyps und einer starken Bewertungsfunktion wurde zunächst in Python programmiert und später in Java übersetzt.

Rust (programming language): Aus Gründen der Effizienz und Vorerfahrung im Umgang mit Rust und dessen libraries (crates) wurden die externen Tests in Rust programmiert.

Reqwest, serde, serde_json: Bei diesen Tools handelt es sich um Libraries aus der Rust Umgebung. Reqwest wurde beispielsweise in Zusammenhang mit dem HTTP-Client der externen Tests genutzt, serde und serde_json dienten der JSON-Serialisierung und Deserialisierung.

2.4.3 Dokumentation & weitere Tools

Word (Microsoft Office 365): Dieses Textverarbeitungsprogramm wird für die Erstellung der Dokumente, die neben dem Hauptprodukt erstellt werden, verwendet.

PowerPoint (Microsoft Office 365): Ist ein Präsentationsprogramm, das verwendet wurde, um die Präsentation für die Messe zu erstellen.

Visual Paradigm Online: Es handelt sich hierbei um ein Web-Tool für die Erstellung von (UML) Diagrammen, welche für die zu erstellenden Dokumente benötigt werden.

GanttProject: Dieses ist ein Programm, das zur Erstellung eines Gantt-Diagramms, welches für eine zeitliche Übersicht im Rahmen der Aufwandsschätzung und des Sprint-Planning für das abgegebene Angebot, benötigt wurde.

InkScape: Das ist die genutzte Software zur Erstellung von Icons und Logos im SVG-Format.

Wordpress: Das Hosting der Projekt-Website läuft über Wordpress.