

Robert-Bosch Berufskolleg Dortmund

Blockwoche (20.01. – 24.01.2020) ITM7

Fachinformatiker für Anwendungsentwicklung

Dokumentation zum Blockwochenprojekt

**Fancy YOOLOO**

Make YOOLOO fancy again!

Abgabetermin: 24.01.2020

**Autoren**:

Philipp Gahlen

Lukas Loenser

Lukas Luzius

Falco Weihrauch



**fancy.IT**

Inhalt

[Abbildungsverzeichnis IV](#_Toc30674830)

[Tabellenverzeichnis V](#_Toc30674831)

[Abkürzungsverzeichnis VI](#_Toc30674832)

[1 Einleitung 1](#_Toc30674833)

[1.1 Projektbeschreibung 1](#_Toc30674834)

[1.2 Projektziel 1](#_Toc30674835)

[1.3 Projektumfeld und Begründung 1](#_Toc30674836)

[1.4 Projektschnittstellen 1](#_Toc30674837)

[2 Projektplanung 1](#_Toc30674838)

[2.1 Projektstruktur 1](#_Toc30674839)

[2.2 Projektablauf 1](#_Toc30674840)

[2.3 Kapazitätsplanung 2](#_Toc30674841)

[2.4 Qualitätsplanung 2](#_Toc30674842)

[3 Analysephase 2](#_Toc30674843)

[3.1 Ist-Analyse 2](#_Toc30674844)

[3.2 Lastenheft 2](#_Toc30674845)

[4 Entwurfsphase 3](#_Toc30674846)

[4.1 Zielplattform 3](#_Toc30674847)

[4.2 Architekturdesign 3](#_Toc30674848)

[4.3 Entwurf der Benutzerschnittstelle 3](#_Toc30674849)

[4.4 Pflichtenheft 3](#_Toc30674850)

[5 Implementierungsphase 3](#_Toc30674851)

[5.1 Pflichtmodul 1: Spielweise 3](#_Toc30674852)

[5.2 Pflichtmodul 2: Regeleinhaltung 3](#_Toc30674853)

[5.3 Pflichtmodul 3: Spielerkonto 4](#_Toc30674854)

[5.4 Optionalmodul 1: Bot 5](#_Toc30674855)

[5.5 Optionalmodul 2: Logger 6](#_Toc30674856)

[6 Testphase 6](#_Toc30674857)

[6.1 Test der Funktionalität 6](#_Toc30674858)

[6.2 Test der Benutzerfreundlichkeit 6](#_Toc30674859)

[7 Dokumentation 6](#_Toc30674860)

[8 Fazit 6](#_Toc30674861)

[8.1 Soll- / Ist-Vergleich 6](#_Toc30674862)

[8.2 Retroperspektive 7](#_Toc30674863)

[8.3 Probleme 7](#_Toc30674864)

[8.3.1 Spielweise 7](#_Toc30674865)

[A Anhang i](#_Toc30674866)

[A.1 Projektablauf i](#_Toc30674867)

[A.2 Verwendete Ressourcen i](#_Toc30674868)

[A.3 EPK Kartenspiel Yooloo ii](#_Toc30674869)

[A.4 Auszug Lastenheft iii](#_Toc30674870)

[A.5 Testauswertung iv](#_Toc30674871)

[A.6 Auszug Benutzerdokumentation v](#_Toc30674872)

# Abbildungsverzeichnis

[Abbildung 1: Auszug YoloServer.java 5](file:///C:\Users\gahlen\Desktop\fancyYooloo\docs\Dokumentation\ITM7_Gruppe2_DokuYooloo.docx#_Toc30675097)

[Abbildung 2: GANTT-Diagramm i](#_Toc30675098)

[Abbildung 3: EPK-Kartenspiel Yooloo ii](#_Toc30675099)

# Tabellenverzeichnis

[Tabelle 1: Modulzuteilung 1](#_Toc30674568)

[Tabelle 2: Zeitplanung 2](#_Toc30674569)

[Tabelle 3: Vor- und Nachteile Teamarbeit bei Softwareprojekten 7](#_Toc30674570)

# Abkürzungsverzeichnis

|  |  |
| --- | --- |
| EPK | Ereignisgesteuerte Prozesskette |
| JDK | Java Development Kit |
|  |  |

# Einleitung

## Projektbeschreibung

Im Rahmen der Blockwache im Frühjahr 2020 soll die Klasse ITM7 eine bestehende Implementierung des Kartenspiels Yooloo um einige Funktionen erweitern. Dazu wurde die Klasse in Gruppen zu je 2-4 Schülern eingeteilt. Anschließend sollte sich jeder Schüler eine Aufgabe aus einer Auswahl von möglichen Aufgaben aussuchen und implementieren.

## Projektziel

Projektziel ist die Erweiterung der bereits bestehenden Yooloo-Implementierung um verschiedene Funktionen. Die Gruppe der Autoren hat sich neben den Pflichtmodulen (Spielweise, Regeleinhaltung und Spielerkonto) noch die beiden Optionalmodule Bot und Logger ausgesucht. Eine Ausführlichere Erklärung zu den einzelnen Modulen lässt sich Kapitel 5 entnehmen.

## Projektumfeld und Begründung

Sinn des Projektes ist es den Schülern einen Einblick in die Umsetzung von Softwareprojekten als Teamarbeit zu gewähren. Da die Umsetzung Großteils in Eigenregie durchgeführt wird, müssen auch Strukturen gebildet und Hilfsmittel genutzt werden, die Softwareentwicklung im Team vereinfacht. Ein gutes Beispiel hierfür ist der Einsatz eines Versionskontrollsystems wie z.B. git.

## Projektschnittstellen

Das Programm besitzt keine externen Schnittstellen. Der Server und Client können komplett angepasst werden, allerdings mit der Vorgabe, dass auch die Benutzung von Server oder Client anderer Gruppen möglich sein muss. Dazu muss das Protokoll, durch welches sich diese austauschen unverändert oder zumindest abwärtskompatibel bleiben.

# 2 Projektplanung

## 2.1 Projektstruktur

Um das Projekt sinnvoll bearbeiten zu können macht es Sinn die verschiedenen Aufgaben in Module zusammenzufassen und auf die Entwickler aufzuteilen. Hierzu haben sich die Entwickler vor der Umsetzung darauf geeinigt wer welche Aufgabe übernehmen will. Ein weiterer Vorteil dieses Vorgehens ist es, das nicht jeder Entwickler jeden Teilbereich des Projektes analysieren muss, um eine Einschätzung abgeben zu können, wie viel Aufwand die Umsetzung in Anspruch nimmt. Nachfolgend findet sich eine Übersicht in Tabelle 1: Modulzuteilung, welcher Entwickler welches Modul bearbeitet hat.

|  |  |  |
| --- | --- | --- |
| Art des Moduls | Name des Moduls | Entwickler |
| Pflichtmodul | Spielweise | Lukas Loenser |
| Pflichtmodul | Regeleinhaltung | Philipp Gahlen |
| Pflichtmodul | Spielerkonto | Lukas Luzius |
| Optionalmodul | Bot | Falco Weihrauch |
| Optionalmodul | Logger | Lukas Luzius |

Tabelle 1: Modulzuteilung

## 2.2 Projektablauf

Für die Umsetzung des Projektes standen der Gruppe 34 Stunden Arbeit zur Verfügung. Diese wurden vor Beginn der Umsetzung in mehrere Phasen aufgeteilt, die bei der Umsetzung von Software-Projekten typischerweise durchlaufen werden. Folgend lässt sich aus Tabelle 2: Zeitplanung entnehmen, wie viele Stunden welcher Phase des Projektes zugeordnet wurden. Anzumerken ist hier, dass es sich um Arbeitsstunden handelt, d.h. da die Gruppe aus 4 Entwicklern bestand ergibt sich eine insgesamte Zeit von 4 Entwickler \* 34 Stunden = 136 Stunden.

|  |  |
| --- | --- |
| Projektphase | Geplanter Zeitaufwand (Stunden) |
| Analysephase | 15 |
| Entwurfsphase | 20 |
| Implementierungsphase | 45 |
| Testphase | 16 |
| Dokumentation | 40 |
| Gesamt | 136 |

Tabelle 2: Zeitplanung

Außerdem lässt sich in Anhang A1: Projektablauf auf Seite i eine Übersicht des Projektablaufs als Gantt-Diagramm finden um den zeitlichen Ablauf besser nachvollziehen zu können.

## 2.3 Kapazitätsplanung

Im Anhang A2: Verwendete Ressourcen auf Seite i findet sich eine Übersicht der verwendeten Hard- und Software sowie personelle Ressourcen. Aufgrund des geringen Budgets des Projekts wurde darauf geachtet möglichst kostenlose bzw. schon vorhandene Ressourcen zu verwenden.

## 2.4 Qualitätsplanung

Gerade bei der Bearbeitung von Aufgaben durch mehrere Entwickler, wird die Kontrolle der Qualität von Code und Ergebnis immer wichtiger. Hierzu hat sich die Gruppe der Autoren dazu entschieden die JUnit-Test-Suite zu benutzen und automatisierte Tests zu schreiben. Dabei ist es wünschenswert eine hohe Code-Coverage bei neuen Funktionalitäten zu erreichen, um die Funktionalität gewährleisten zu können.

# 3 Analysephase

## 3.1 Ist-Analyse

Vor der Umsetzung des Projektes bestand bereits eine Implementierung, die um genannte Module ergänzt werden sollte. Dieser Implementierung fehlen allerdings einige wichtige Funktionalitäten wie einen Mechanismus der das Umgehen von Regeln, wie doppeltes Ausspielen einer Karte, verhindert. Außerdem gibt die bisherige Implementierung dem Spieler keine Möglichkeit gegen einen computergesteuerten Gegner zu spielen. Um das Spielprinzip von Yooloo zu verdeutlichen lässt sich in Anhang A.3: EPK Yooloo Kartenspiel auf Seite ii der Ablauf einer Runde Yooloo entnehmen.

## 3.2 Lastenheft

Die Erstellung des Lastenheftes bildet den Abschluss der Analysephase. Hierzu werden die gesammelten Anforderungen an das Projektes verschriftlicht. Im Anhang A.4: Auszug Lastenheft auf Seite iii befindet sich ein Auszug aus dem Lastenheft des Auftraggebers, dass die Kernanforderungen an das Projekt und die gewählten Module umfasst.

# 4 Entwurfsphase

## 4.1 Zielplattform

Die Zielplattform der Anwendung sind die Rechner des Robert-Bosch Berufskollegs in Dortmund. Diese verwenden vorwiegend das Betriebssystem Ubuntu Linux. Da die Anwendung allerdings in Java implementiert wurde, ist eine Ausführung auf anderen Plattformen wie Windows oder Mac OS auch möglich und wird während der Implementierung (aufgrund der verschiedenen Entwicklersysteme) auch getestet.

## 4.2 Architekturdesign

Das Spiel läuft auf dem Client-Server Prinzip. Bei diesem Prinzip verbinden sich alle Client-Programme über eine Netzwerkschnittstelle mit dem Server-Programm und tauschen darüber Informationen zu ihrem Ablauf aus. Vorteil dieser Architektur ist die Zentralisierung von Spieldaten und eine Vereinfachung von Kontrollmechanismen (wie z.B. der Regeleinhaltung).

## 4.3 Entwurf der Benutzerschnittstelle

Der Entwurf einer speziellen Benutzerschnittstelle für den Benutzer ist nicht vorgesehen. Auf die Implementierung einer GUI wird also an dieser Stelle verzichtet. Die Informationen über den Spielablauf erfolgt über die Konsolenausgabe. An dieser Stelle sein auch angemerkt, dass eine Implementierung einer grafischen Oberfläche innerhalb des kurzen zeitlichen Fensters schwierig geworden wäre, da die grafische Gestaltung von Spielen sehr aufwändig werden kann.

## 4.4 Pflichtenheft

Am Ende der Entwurfsphase wird normalerweise ein Pflichtenheft erstellt, dass die genauen Spezifikationen des fertigen Programmes beschreibt. Da dies allerdings den Rahmen dieses Projektes sprengen würde, haben die Autoren an dieser Stelle darauf verzichtet ein Pflichtenheft zu erstellen. An dieser Stelle möchten die Autoren auch noch einmal auf den Auszug aus dem Lastenheft verweisen, dass die Kernanforderungen an die Software beschreibt, welches im Anhang A.4: Auszug Lastenheft auf S. iv zu finden ist.

# 5 Implementierungsphase

## 5.1 Pflichtmodul 1: Spielweise

Wie auch wenn ein Mensch sich eine Strategie überlegt sollte jeder Client eine Strategie benutzen. Zum Punkt dieser Dokumentation wurden drei Strategien implementiert: 1. Von klein nach groß sortiert, 2. Von groß nach klein sortiert und 3. Alle Karten sind zufällig sortiert. Die Sortierung wird einmalig vor dem Spielen der Karten in der Methode „YoolooClient.fancySortierung(String property)“ ausgeführt. Für die Implementierung wurde der Sortieralgorithmus aus dem YoolooSpieler in den YoolooClient ausgelagert. Dies passierte aus dem Grund um den Vorgang mit einem Logger zu überwachen, was jedoch ein einer Klasse welche von Serializable erbt nicht möglich ist. Um zu gewährleisten das man die Strategie leicht ändern kann wurden Properties-Dateien angelegt in denen man eine Strategie in Form einer Zahl hinterlegen kann. Wenn keine Zahl hinterlegt wurde, wird immer die Zufallsstrategie benutzt.

Der Entwickler welcher sich mit der Spielweise auseinander gesetzt hat wollte noch eine weitere Spielweise implementieren. Für diese Strategie sollten alle Ergebnisse eines Spiels gespeichert und Ausgewertet werden. Mehr wird dazu in der Retroperspektive erwähnt.

## 5.2 Pflichtmodul 2: Regeleinhaltung

Für die Implementierung der Regeleinhaltung ist es erforderlich, dass der Server bei jeder gesendeten Karte prüft, ob der Spieler die Karte schon einmal gesendet hat. Dafür wird in der Methode „YoolooSession.spieleKarteAus()“ geprüft ob in dem Spielplan bereits ein Stich existiert, bei dem der Spieler die Karte ausgespielt hat.

Ist dies nicht der Fall, geht das Spiel wie gewohnt weiter und nachdem alle Spieler ihre Karten gesendet haben, wird der Stich ausgewertet.

Sollte die Karte bereits vorhanden sein, wird die Methode „YoolooServer.kickeAlleSpieler()“ aufgerufen, die dann für jeden verbundenen Client die Methode „YoolooClientHandler.kickClient()“ aufgerufen, die dem Client signalisiert, dass die Verbindung zum Server getrennt wird. Somit wird die Runde abgebrochen sobald ein Spieler die Regeln nicht eingehalten hat und die Clients müssen sich erneut verbinden um eine neue Runde spielen zu können.

## 5.3 Pflichtmodul 3: Spielerkonto

Das Pflichtmodul Spielerkonto wird in zwei Unteraufgaben aufgeteilt. Die erste Teilaufgabe besteht darin den Namen eines Spielers wiederzuerkennen und seine zuletzt gespielte Reihenfolge der Karten als initialen Zustand zu setzen. Die zweite Teilaufgabe besteht darin sicherzugehen, dass ein Spieler nicht zweimal im gleichen Spiel vorkommt.

Für die Sicherung der Kartenreihenfolgen wurden die Klassen „*YoolooUsers*“, „*YoolooPersistance*“ und „*YoolooFileWriter*“ erstellt. Die Klasse „*YoolooPersistance*“ ist eine simple Datenhaltungsklasse die serialisierbar ist und eine Zuordnung von Spielername zu Kartenreihenfolge enthält. Der „*YoolooFileWriter*“ liest und schreibt ein Objekt der Klasse „YoolooPersistance“ in eine feste Datei im Root des Projektes. Dadurch wird eine Datenhaltung über die Serverlaufzeit hinaus gewährleistet.

Als Schnittstelle zum Rest des Projektes dient die Klasse „*YoolooUsers*“. Diese stellt unter anderem die Methode „*setUserCardOrder*“ zur Verfügung, die einen String und eine Liste von Integer entgegennimmt. Diese Werte werden als Spielername und Kartenreihenfolge persistiert. Des Weiteren wird die Methode „*setUserCardOrder*“ bereitgestellt, die ebenfalls einen „*YoolooSpieler*“ entgegennimmt. Falls der Spieler bekannt ist und auch einen validen Datensatz hinterlegt hat wird ein Array von „*YoolooKarte*“ erstellt. Dieses hat die gespeicherte Reihenfolge und die Farbe der am Spieler hinterlegten Karten. Ist für diesen Spieler kein Speicherstand verfügbar wird die Kartenreihenfolge, die bereits am Spieler gesetzt ist, zurückgegeben. Da auf diese Methoden aus mehreren Threads gleichzeitig zugegriffen wird, wurden sie mit dem Schlüsselwort „synchronized“ versehen.

Um den letzten Spielstand zu laden wurde die Methode „*registriereSpielerInSession*“ der Klasse „*YoolooClientHandler*“ erweitert. Als Gegenstück wurde die Methode „*run*“ derselben Klasse erweitert, um den letzten Spielstand zu speichern.

Für die zweite Teilaufgabe wurde der „*YoolooClientHandler*“ um die Methode „*playerAlreadyInSession*“ erweitert, die Anhand eines Übergebenen Strings prüft ob ein Spielername bereits registriert ist. Falls der Spieler noch nicht vorkommt wird an dieser Stelle wie vorher implementiert vorgegangen. Ist der Spieler bereits vorhanden so wird die Verbindung zum Client abgebrochen.

## 5.4 Optionalmodul 1: Bot

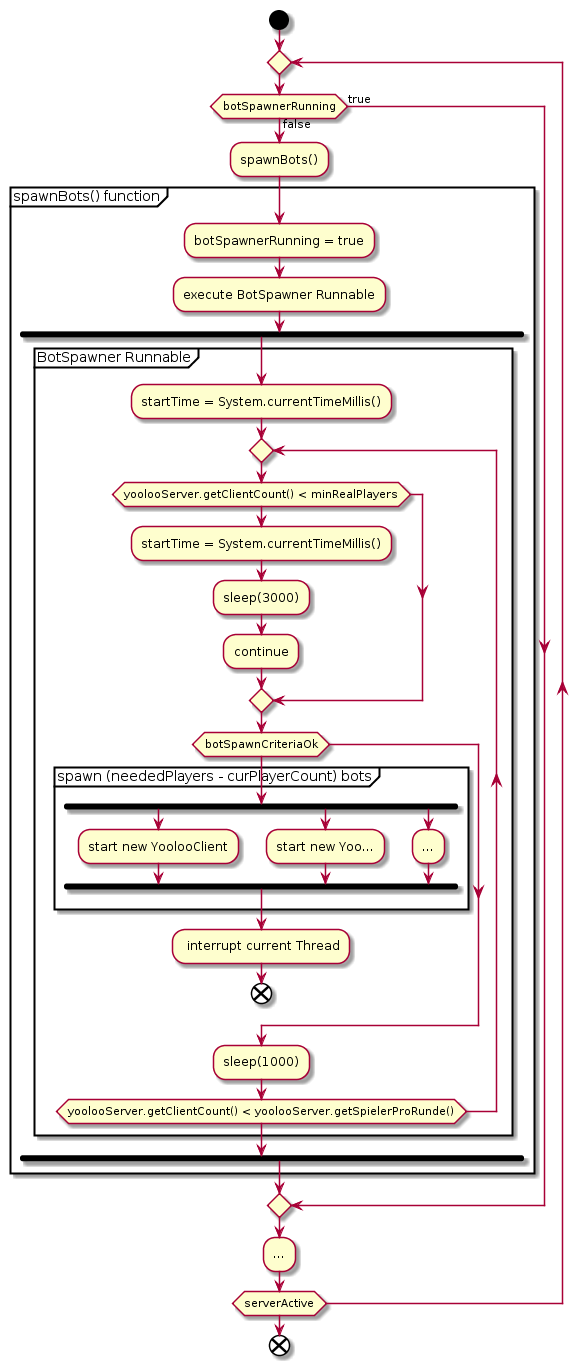
Das Bot-Feature wurde auf Basis des bestehenden Clients implementiert und direkt im Server verwurzelt. Das nebenstehende Diagramm zeigt die Integration des BotSpawn-Prozesses in die *startServer()*-Methode.

Abbildung 1: Auszug YoloServer.java

Der Ablauf:   
Zunächst wird geprüft, ob bereits ein BotSpawn-Prozess aktiv ist.  
Ist dies nicht der Fall, dann wird die Methode *spawnBots()* aufgerufen. Sie startet ein Runnable mit dem eigentlichen BotSpawner und stellt vorher sicher, dass das *botSpawnerRunning-*Flag gesetzt ist.

Im BotSpawner-Runnable wird die Anfangszeit gesetzt. Anschließend wird geprüft, ob genug „echte“ Spieler dem Spiel beigetreten sind. Ist dies nicht der Fall, dann wird die Anfangszeit neu gesetzt und 3 Sekunden gewartet.  
Sollten genug „echte“ Spieler eingeloggt sein, dann wird mit der Funktion *botSpawnCriteriaOk()* geprüft, ob die BotSpawn-Kriterien erfüllt sind. Diese sind:

* Genug „echte“ Spieler vorhanden
* Lange genug auf weitere Spieler gewartet
* Spiel nicht komplett gefüllt mit Spielern

Sollten die Kriterien nicht erfüllt werden, wartet der BotSpawner eine Sekunde und prüft erneut.  
Bei Erfüllung hingegen füllt er das Spiel mit der fehlenden Anzahl an Spielern auf, indem er eigene Instanzen des YoolooClients in Runnables startet und sich dann selbst beendet.

## 5.5 Optionalmodul 2: Logger

Um dieses Modul umzusetzen wurde der Java-eigene Logger verwendet. Es wurden die Levels „*Fine*“, „*Warning*“ und „*Severe*“ verwendet, um diverse Debug-Nachrichten abzubilden. Neben dem Logging in den Erweiterungen der einzelnen Module wurden auch die bereits vorhandenen Systemausgaben ersetzt. Dabei wurde unterschieden ob es sich bei einer Nachricht um eine Debug Information handelt oder um Spielstandausgaben. Zweiteres wird weiterhin durch „*System.out.println*“ ausgeschrieben. Konfiguriert wird das Logging durch die neue Datei „*logging.properties*“.

# 6 Testphase

## 6.1 Test der Funktionalität

Der Test der Funktionalität erfolgt wie bereits erwähnt durch sogenannte Unit-Tests. Hierzu wurde die Test-Suite JUnit verwendet, die bereits in dem JDK enthalten ist. In Anhang A.5: Testauswertung auf Seite iv befindet sich ein Screenshot der Auswertung der Tests. Da alle Tests erfolgreich abgeschlossen wurden, kann die Funktionalität des Programms garantiert werden.

## 6.2 Test der Benutzerfreundlichkeit

Für den Test der Benutzerfreundlichkeit bietet es sich bei den meisten Projekten an externe Anwender oder evtl. sogar einen Kunden das Produkt testen zu lassen. Da allerdings jede Gruppe der Klasse der Autoren dasselbe Projekt bearbeitet hat und sich an dem Aufruf und der Bedienung der Programme nichts geändert wurde, ließ sich kein Tester finden bzw. war ein Test auch nicht notwendig.

# 7 Dokumentation

Um die Anwendung Benutzern und auch Entwicklern zugänglicher zu machen empfiehlt es sich sowohl eine Benutzerdokumentation, als auch eine Entwicklerdokumentation zu verfassen. Wie der Name schon sagt, ist die Benutzerdokumentation für die Gruppe der Benutzer gedacht und beschreibt den Umgang mit dem Programm. Sie umfasst oft auch die Soft- und Hardwareanforderungen, die Installationsanleitung, die Steuerung und noch mehr Hinweise, die für den Benutzer hilfreich sein könnten.

Einen Auszug aus der Benutzerdokumentation befindet sich im Anhang A.6: Auszug Benutzerdokumentation auf S. v. Die Entwicklerdokumentation beschreibt grob gesagt, genau das was dieses Dokument darstellt. Die Architektur und der Aufbau des Programms werden mit Hilfe von fachgerechten Diagrammen etc. dargestellt. Da dieses Dokument große Teile der Entwicklerdokumentation darstellt und der zeitliche Rahmen es nicht zulässt, hat der Autor auf eine separate Entwicklerdokumentation verzichtet.

# 8 Fazit

## 8.1 Soll- / Ist-Vergleich

Vor dem Soll- / Ist-Vergleich sollte der neue Ist-Zustand genau erfasst und analysiert werden. Hierzu bietet es sich an das erstellte Pflichtenheft mit den implementierten Funktionalitäten zu vergleichen. Dies stellt gleichzeitig auch einen Vergleich zwischen dem Soll- und Ist-Zustand dar. Da die Autoren allerdings der Zeit geschuldet auf ein Pflichtenheft verzichten mussten, wurde der Vergleich anhand des Lastenhefts geführt, dass in Anhang A.4: Lastenheft auf Seite iv zu finden ist. Da durch den Soll-/Ist-Vergleich festgestellt wurde, dass alle Funktionalitäten umgesetzt wurden, war das Projekt an dieser Stelle beendet.

## 8.2 Retroperspektive

Im Rückblick auf die Woche haben die Autoren festgestellt, dass die Zusammenarbeit an Softwareprojekten einige Vor- und Nachteile bietet, dazu haben Sie im Folgenden eine Tabelle mit Vor- und Nachteilen zur Teamarbeit an Softwareprojekten ausgearbeitet:

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| * Schnelles abarbeiten von Aufgaben * Gegenseitige Unterstützung | * Mergekonflikte |

Tabelle 3: Vor- und Nachteile Teamarbeit bei Softwareprojekten

## 8.3 Probleme

In diesem Abschnitt können die Entwickler Probleme schildern die während der Bearbeitung des Projektes aufgetreten sind. Hierzu können nicht implementierte Funktionen, nicht behobene Fehler oder Probleme in der Zusammenarbeit oder Ähnlichem zuzählen.

### 8.3.1 Spielweise

Wie in der Implementierungsphase schon erwähnt, wurde eine Spielstrategie nicht zu Ende gebracht. Diese Strategie sollte auf einer statistischen Analyse aufbauen und damit die beste Chance auf den Gewinn garantieren. Dazu wurde eine stats.json-Datei erstellt in welcher die Siegesrate einer jeder Karte für jeden Stich steht. Diese Datei sollte nach jeder Runde aktualisiert werden und mit mehreren Durchgängen aufgepumpt werden. Sobald die Datei auswertbare Daten enthält soll der Client beim Sortieren seiner Karten auf diese Datei aufbauen.

Leider konnte dieses Feature nicht zu Ende implementiert werden, da es für den Entwickler ein großer Zeitaufwand gewesen wäre, welcher für eine Optionale Aufgabe nicht vertretbar wäre. Außerdem war der Entwickler nicht gut genug mit dem Themen Threads und Dateien zu beschreiben vertraut um diese Funktion in solch kurzer Zeit zu implementieren.

# A Anhang

## A.1 Projektablauf



Abbildung 2: GANTT-Diagramm

## A.2 Verwendete Ressourcen

**Hardware**:

* 4 Arbeitsplätze (1x Desktop-PC, 3x Notebook)

**Software**:

* 3x Windows 10
* 1x Mac OS X
* IntelliJ IDE (Education License)
* Oracle JDK 11
* draw.io – Web-Anwendung zur Erstellung von Diagrammen
* PlantUML
* Microsoft Word – Erstellung der Dokumentation
* Microsoft Powerpoint – Erstellung der Präsentation
* Github.com – Versionsverwaltung

**Personal**:

* 4 Entwickler – Umsetzung des Projekts
* Tester – Test der Benutzerfreundlichkeit

## A.3 EPK Kartenspiel Yooloo

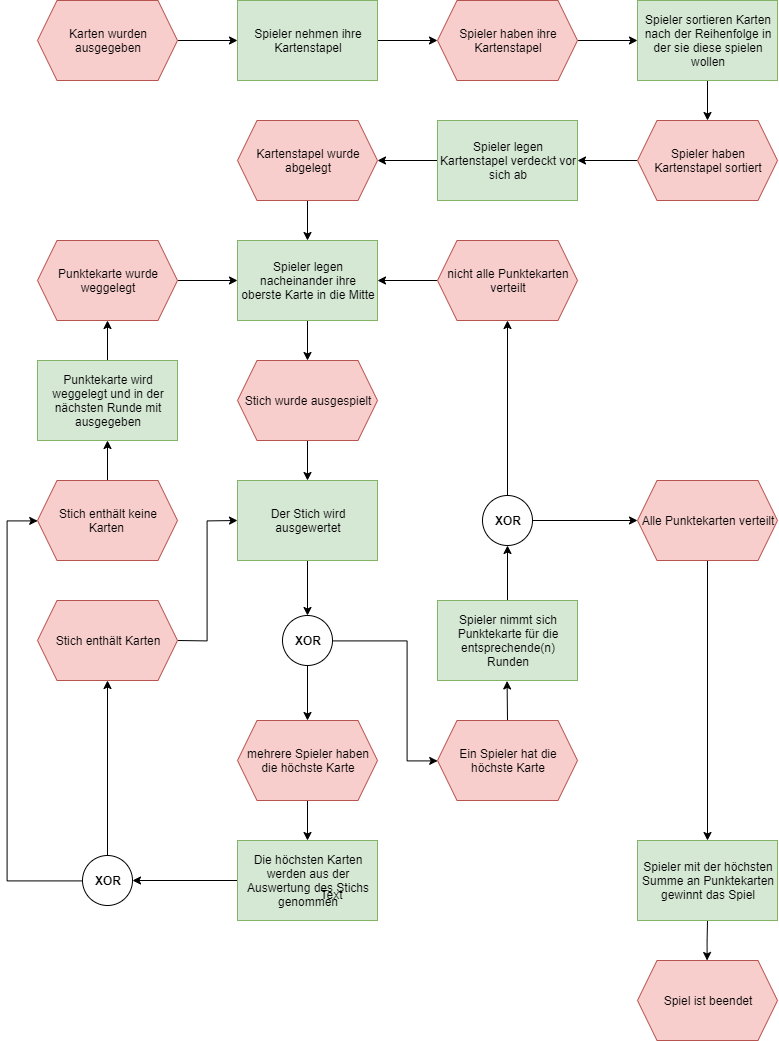


Abbildung 3: EPK-Kartenspiel Yooloo

## A.4 Auszug Lastenheft

Das Projekt umfasst drei Pflichtmodule und mehrere optionale Module.

**Pflichtmodule**

**Spielweise**

Unter der Spielweise versteht man die Strategie die verflogt wird um das Spiel zu spielen. Diese Strategie beinhaltet die Sortierung der Karten die ein Spieler spielt um einen Stich zu gewinnen. Die Anforderung zu diesem Modul ist es, den Clients unterschiedliche Strategien zugeben die sie in einem Spiel benutzten können.

**Regeleinhaltung**

In der Regeleinhaltung soll überprüft werden, dass jeder Spieler bzw. Client sich nach den Regeln verhält. Die einzige Regel die beachtet werden muss, ist das ein Client jede Karte nur einmal ausspielen darf.

**Spielerkonto**

Mit dem Spielerkonto soll gewährleistet werden, dass ein Client nur einmal in einer Sitzung vorkommt, also nicht gegen sich selbst spielt. Außerdem soll er über den Namen eindeutig identifiziert werden.

**Optionale Module**

**R2D2 Botmode**

Die Bots die über dieses Modul implementiert wurden sollen Clients simulieren die von dem Server gestartet werden sollen, wenn nicht genug Spieler für eine Runde Yooloo vorhanden sind.

**Logger**

Der Logger soll für jedes Modul Ausgaben in der Konsole anzeigen die die Abläufe und möglichen Fehler dem Nutzer darlegen. Es soll der Standartlogger von Java benutzt werden.

## A.5 Testauswertung

**YoolooSessionTests**:

|  |  |
| --- | --- |
| Test |  |
| testDuplicateCard |  |
| testDifferentColorCard |  |

**YoolooFileWriterTest**:

|  |  |
| --- | --- |
| Test |  |
| testFileWriter |  |

**DoubleUserNameCheckTest**:

|  |  |
| --- | --- |
| Test |  |
| userAlreadyInSessionTest |  |
| maxClientWatcher |  |

**YoolooUsersTest**:

|  |  |
| --- | --- |
| Test |  |
| testUserNotExistantTest |  |

**YoolooServerTest**:

|  |  |
| --- | --- |
| Test |  |
| clientConnectionTest |  |
| botSpawnTest |  |

## A.6 Auszug Benutzerdokumentation

**Build**

Just checkout the project and make a mvn clean install in the Code-directory

**Startup**

**Server**

java -jar yooloo-server.jar

**Client**

java -jar yooloo-client.jar

**Server-Configuration (server.properties)**

Just place a server.properties-File next to the yooloo-server.jar.

**Server**

Defines the used server-port:

* server.port=44137

**Game**

Defines the total nuber of Players in a Game:

* game.size=3

Defines the minimal count of real Players in a Game:

* game.min.players=3

Defines the time the server schould wait for more Players, before some bots spawn (Seconds):

* game.bot.wait=100

Prevent duplicated players (deactivate for legacy-clients):

game.nameCheck=false

* true: on
* false: off

**Bot**

Sets gamemode of spawned bots:

bot.mode=RANDOM

* RANDOM: random-deck

Das weitere Benutzerhandbuch lässt sich der readme.md entnehmen