

Maturarbeit PiM Jugger Videotool

Mael Pittet

April 2025

Inhaltsverzeichnis

	Seite
1 Vorwort	3
2 Einführung	4
3 Jugger	5
3.1 Was ist Jugger	5
3.2 Wichtige Jugger Begriffe	5
3.3 Spielablauf	6
3.4 Wichtige Regeln	6
4 Hilfsmittel	7
4.1 Java	7
4.2 Entwicklungsumgebung	7
4.3 Maven	7
4.4 Github	7
4.5 JavaFX und Java Scene Builder	8
4.6 KI	8
5 Code	10
5.1 Klassen	10
5.2 GUI	13
5.3 Eigenes Dateiformat	14
6 Diskussion	18
6.1 Was wurde erreicht?	18
6.2 Wo gibt es noch Verbesserungspotenzial?	18
6.3 Was bedeutet diese Applikation für den Sport Jugger?	18
6.4 Wie kann diese Applikation weiterentwickelt werden?	19
7 Reflexion	20
7.1 Was habe ich gemacht?	20
7.2 Was habe ich gelernt?	20
7.3 Was würde ich anders machen	21
8 Anhang	23
9 Selbstständigkeitserklärung	24

1 Vorwort

Diese Maturaarbeit entstand im Rahmen meines Abschlussjahres am Gymnasium Oberwil. Die Idee dazu entstand, als ich gemeinsam mit meinem Verein einen Abend lang Spielvideos analysierte. Unser Ziel war es, die Duellquoten sowie die Spielzeiten der einzelnen Spielenden zu erfassen. Ein erster Versuch mit Excel erwies sich jedoch als unpraktisch. Da ich mich sehr für Informatik interessiere, verband ich mein persönliches Interesse mit den Bedürfnissen des Vereins und entwickelte eine Software, die meine Neugier für die Programmierung stillt und die Datensammlung erheblich erleichtert. Die erfolgreiche Umsetzung dieser Arbeit wäre ohne Unterstützung nicht möglich gewesen. Mein besonderer Dank gilt meinem Betreuer, Herrn Flückiger, der mich mit seinem umfangreichen Wissen durch manche Blockade geführt hat. Ebenso danke ich Manuel und Ana, welche mir mit ihren fundierten Jugger-Wissen dabei geholfen haben, die richtigen Prioritäten bei der Datenerfassung zu setzen. Schliesslich möchte ich auch Silas danken, der die Jugger-Erklärung mit seinen Bildern anschaulich ergänzt hat.

2 Einführung

Wir leben in einem Zeitalter der Digitalisierung. Warum also nicht auch die Trainingsvorbereitungen digitalisieren? In vielen Sportarten, darunter Fussball und Tennis, ist Videomaterial eine grosse Hilfe, um zu erkennen, welche Bereiche noch trainiert werden sollten. Der Basler Verein nimmt seine Spiele zwar schon lange auf, doch bisher wurden die meisten Aufnahmen nur zur Unterhaltung oder für subjektive Verbesserungsideen genutzt. Wurde ein Spiel dennoch mit Zahlen erfasst, dauerte dies mit Excel mehrere Stunden. Deshalb habe ich mir zum Ziel gesetzt, meinem Verein und der kompletten Jugger-Gemeinschaft mit dieser Arbeit eine Anwendung zur Verfügung zu stellen, die die objektive Erfassung der Informationen in einem Jugger-Spiel vereinfacht. Dazu habe ich zunächst die Bedürfnisse innerhalb meines Vereins abgefragt, um zu erfahren, welche Informationen benötigt werden. Zu Beginn dieser Arbeit wird die Sportart Jugger vorgestellt, um den Kontext und die Besonderheiten des Spiels zu erläutern. Anschliessend werden die für die Umsetzung verwendeten Hilfsmittel beschrieben und der Aufbau des entwickelten Codes detailliert dargestellt. Darauf aufbauend folgt eine Diskussion der Ergebnisse, in der die Stärken und Grenzen der entwickelten Anwendung reflektiert werden. Abschliessend wird in einer persönlichen Reflexion auf den Entstehungsprozess sowie mögliche Weiterentwicklungen eingegangen.

3 Jugger

3.1 Was ist Jugger

Jugger ist eine moderne Mannschaftssportart, die in den späten 1980er-Jahren durch den postapokalyptischen Film „The Salute of the Jugger“ (deutscher Titel: „Die Jugger – Kampf der Besten“) inspiriert wurde. Seither hat sich der Sport weit vom Film entfernt und ein eigenständiges, strukturiertes Regelwerk entwickelt, das auf Fairness, Sicherheit und Dynamik ausgelegt ist. Innerhalb der Jugger-Community sind die spanische und die deutsche Gruppe die zwei grössten Gemeinschaften. Es lassen sich jedoch Unterschiede in verschiedenen Aspekten zwischen beiden Gruppen beobachten. Da das einzige Schweizer Team „Jugger Basiliken Basel“ in der deutschen Community aktiv ist, wird im vorliegenden Text lediglich auf das deutsche Regelwerk sowie auf die deutsche Spielpraxis eingegangen.

3.2 Wichtige Jugger Begriffe

- **Jugg:** ein zylindrischer Schaumstoff, welcher als Spielball verwendet wird
- **Mal:** Ein donutförmiger Gegenstand, in den der Jugg gesteckt werden muss, um einen Punkt zu ergattern
- **Pompfe:** Die Spielgeräte, mit denen die Spieler*innen sich mit anderen duellieren. Dabei gibt es:
 - **Stab:** hat eine Trefferfläche, sowie eine Blockfläche
 - **Langpompfe:** hat nur eine Trefferfläche
 - **Q-Tipp:** hat an den Enden jeweils eine Trefferfläche (Q-Tipp zu deutsch Wattestäbchen)
 - **Schild:** eine Schild und dazu eine kurze Pompfe
 - **Doppelkurz:** zwei kürzere Pompfen, welche mit jeweils einer Hand gehalten werden kann
 - **Kette:** Eine 3.2m lange Schnur mit einem Schaumstoffball am Ende, gibt mehr Strafzeit
- **Läufer*in:** Person, welche den Jugg tragen darf, dafür allerdings keine Pompfe führen darf
- **Grün / Rot:** Gängiger Begriff, um während des Spiels den eigenen Teammitgliedern mitzuteilen, ob das eigene Team (grün) den Jugg kontrolliert oder das gegnerische (rot)
- **Druckpunkt:** Eine Person, welche ihr Duell schnell ausspielen will, mit der Hoffnung, bessere Chancen auf einen Sieg zu haben und somit dem gegnerischen Team in den Rücken fallen zu können

- **Steine:** Die Messeinheit in Jugger; beträgt anderthalb Sekunden
- **Läufi / Spieli:** Eine gängige Variante, um alle Geschlechter anzusprechen, sowie die Wörter kurz zu halten, da während des Spiels lange Wörter zu lange brauchen.

3.3 Spielablauf

Um Jugger zu spielen, braucht man fünf Personen. Davon muss eine Person Läufer*in sein und maximal eine Person darf die Kette spielen. Bei Turnieren sind in der Regel vier Schiedsrichter*innen im Einsatz, wobei diese Zahl variieren kann. Einer der Schiedsrichter beginnt das Spiel und zählt im Takt der Steine von 20 herunter. Alle fünf Personen rennen zur Mitte, in der der Jugg liegt. Wer von einer Pompfe getroffen wird, muss für fünf Steine (oder sieben Sekunden) in die Knie gehen und ist somit für siebeneinhalb Sekunden aus dem Spiel. Sollte eine Kette eine Person treffen, muss diese für acht Steine (oder zwölf Sekunden) in die Knie gehen. Der Zug endet, sobald einer der beiden Läufer*innen den Jugg ins Mal steckt und somit dem Team den Punkt sichert.

3.4 Wichtige Regeln

- **Pinnen:** Wenn eine Person kniet, darf eine andere Person kommen und ihre Pompfe auf die kniende Person legen. Diese Person darf nun nicht mehr aufstehen, bis die Person die Pompfe wegnimmt. Diesen Vorgang nennt man eine Person pinnen. [1]
- **Frühstart:** Das Spiel wird mithilfe der Steine angezählt. Dabei wird von 20 heruntergezählt. Es endet mit „3, 2, 1, Jugger“. Wenn eine Person vor dem nächsten Stein nach 1 auf dem Feld ist, hat diese Person einen Frühstart gemacht. Das Spiel wird abgebrochen und beide Teams gehen hinter die Grundlinie zurück. Sollte dies ein zweites Mal geschehen, bekommt das andere Team den Jugg und darf nun damit starten.[1]

4 Hilfsmittel

4.1 Java

Für meine Maturaarbeit habe ich mich für die Programmiersprache Java entschieden. Dies ist auf verschiedene Faktoren zurückzuführen. Ich verfüge bereits über umfassende Erfahrung im Bereich Java. So habe ich zwei Universitätsvorlesungen besucht, in deren Rahmen ich bereits erste Erfahrungen in der Java-Entwicklung sammeln konnte. Darüber hinaus habe ich bereits Minecraft-Plug-ins entwickelt und Java auch am Gymnasium erlernt. Java eignet sich zudem hervorragend für die Erstellung eigenständiger Software, da es auf sämtlichen Betriebssystemen ausgeführt werden kann. Des Weiteren existiert im Bereich der Programmiersprache Java eine umfangreiche Community, die eine Vielzahl an Bibliotheken und nützlichen Anleitungen bereitstellt. Zu guter Letzt sei darauf hingewiesen, dass Java eine objektorientierte Programmiersprache ist und demzufolge objektorientiert programmiert werden kann.

4.2 Entwicklungsumgebung

Als integrierte Entwicklungsumgebung habe ich mich für IntelliJ IDEA entschieden, da ich noch eine Lizenz dieser Software habe (aufgrund meines Schülerstudiums an der Universität Basel). Hinzu kommt, dass ich bereits mit dieser IDE gearbeitet habe. Sie bietet sowohl eine gute GitHub-Integration als auch eine gute Maven-Integration. Leider ist diese Lizenz im Sommer abgelaufen, weshalb ich das Projekt auf Visual Studio Code verschieben musste. Damit hatte ich zwar etwas Erfahrung, war aber froh, dass ich das Projekt nicht mit VS Code einrichten musste, da ich das noch nie gemacht hätte. [2] [3]

4.3 Maven

Im Rahmen meiner Maturaarbeit habe ich mich für die Verwendung des Projektmanagement- und Build-Tools Maven entschieden. Maven ist ein leistungsfähiges Werkzeug, das insbesondere bei externen Bibliotheken und dem allgemeinen Aufbau von Projekten Unterstützung bietet. Es ist richtig, dass andere Lösungen wie z. B. Gradle ähnliche Funktionalitäten bieten. Nichtsdestotrotz ist Maven im Vergleich einfacher zu verstehen und wird von IntelliJ direkt unterstützt. Sollte ich mich nach Abschluss meiner Maturaarbeit dazu entschliessen, diese zu veröffentlichen, wäre eine umfassende Dokumentation von grossem Nutzen. Maven bietet in diesem Zusammenhang eine gute Unterstützung, da es die Möglichkeit gibt, ein Maven-Projekt auf verschiedene Arten zu erstellen.

4.4 Github

Für alle grösseren Projekte ist eine Versionskontrolle unerlässlich. Aufgrund meiner wiederholten Erfahrung mit GitHub und demzufolge auch mit Git war mir bereits im Vorfeld bewusst, dass auch dieses Projekt mit GitHub durchgeführt

werden würde. GitHub weist eine Vielzahl an Vorteilen auf. Zunächst entspricht es dem Industriestandard, stellt kostenfrei Repositories zur Verfügung und ist direkt mit IntelliJ kompatibel.

4.5 JavaFX und Java Scene Builder

Wenn man eine Benutzeroberfläche in Java erstellen möchte, kann man zwischen verschiedenen Bibliotheken wählen. In meinem damaligen Universitätsprojekt haben wir uns für JavaFX[4] entschieden und ich habe mich in meinem aktuellen Projekt ebenfalls für JavaFX entschieden. Einerseits, weil ich bereits Erfahrung damit habe, andererseits, weil JavaFX eine grosse und moderne Auswahl an visuellen Elementen bietet. Zudem unterstützt JavaFX Multimedia, was ich zum Abspielen von Videos benötige. Um mir die Arbeit mit JavaFX zu erleichtern, habe ich den Java Scene Builder verwendet. Mithilfe dieser Anwendung kann ich die visuellen Elemente per Drag-and-Drop anordnen und somit das GUI effizienter gestalten.

4.5.1 Overleaf - LaTeX

Diese Maturaarbeit wurde mit LaTeX erstellt. Zum einen, weil ich damit bereits Erfahrung habe, zum anderen, weil sich damit formelle Arbeiten gut umsetzen lassen. LaTeX bietet viele kleine Funktionen, die das Leben vereinfachen, z. B. beim Zitieren oder beim Erstellen des Inhaltsverzeichnisses. Für die Arbeit mit LaTeX habe ich den Online-Editor Overleaf[5] verwendet, da dieser eine kostenfreie Variante anbietet und ich bereits Erfahrung damit sammeln konnte.

4.5.2 IEEE

Jede wissenschaftliche Arbeit benötigt Quellen, die korrekt angegeben werden müssen. Da ich eine Maturaarbeit im Bereich Informatik schreibe, habe ich mich für den Zitierstil IEEE entschieden, da dieser im Bereich Computer Science weit verbreitet ist. [6]

4.6 KI

In meiner Maturaarbeit habe ich Künstliche Intelligenz als Hilfsmittel eingesetzt. Ich habe zwei verschiedene Arten von KI eingesetzt. Einerseits habe ich ChatGPT von OpenAI[7] genutzt, um meinen Code zu unterstützen und Bugs einfacher zu lösen. Dabei habe ich folgenden Prompt verwendet: „As a professional coder, you have been tasked with creating software to help people analyze sports videos. You have been working on this project for some time. However, you have encountered a problem: your code doesn't run because there is a bug. The error message is: `¡Error message!` Please provide a detailed explanation of why this error is occurring and how to fix the bug. Please cite your sources.“Damit hat mir ChatGPT geholfen, meine Bugs zu lösen, ohne dass ich lange ähnliche Probleme auf StackOverflow und ähnlichen Webseiten suchen

musste. Die zweite KI war DeepL Write[8], die mir dabei geholfen hat, meine Texte grammatikalisch zu verbessern. Dabei habe ich die Sprache auf Deutsch gestellt und den Stil auf Standard gelassen (also keinen bestimmten Stil).

5 Code

Da ich den Code hoffentlich noch einiges Überarbeiten werden, sind die Text hier weder definitiv noch vollständig.

5.1 Klassen

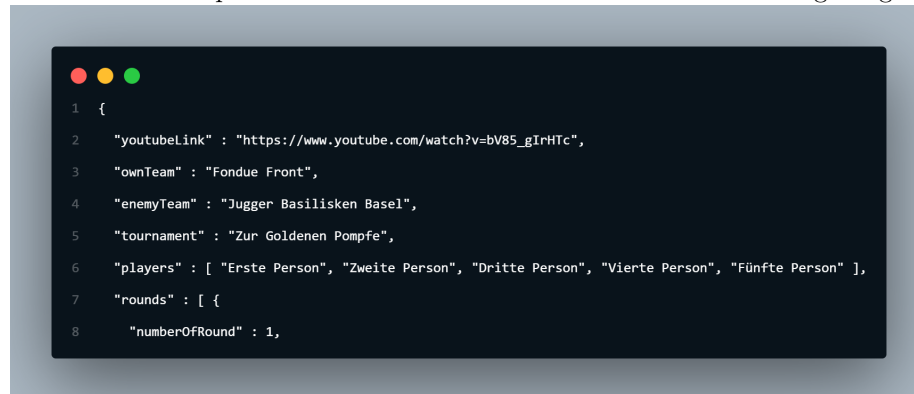
Java ist eine objektorientierte Programmiersprache. Deshalb habe ich in diesem Projekt viel mit verschiedenen Klassen gearbeitet. Diese werden im folgenden Abschnitt vorgestellt.

5.1.1 Main

Die Main-Klasse ist die Klasse, von der aus das Programm startet. In meiner Klasse ist dies also die Klasse, die das GUI startet. Darüber hinaus hat die Main-Klasse keine Funktion.

5.1.2 Reader und Writer

Ursprünglich gab es nur eine Writer-Klasse welche über drei Funktionen verfügte: eine Datei erstellen, eine Datei öffnen und in eine Datei schreiben. Die Idee war, dass, wenn am Anfang eine Datei über den File-Chooser geöffnet wird, die ArrayList von der Datei abgelesen und in den Array des Angaben-Controllers übertragen wird. Sollte eine neue Datei erstellt werden, hat diese Klasse einen Datei-Auswähler verwendet und überprüft, ob die Datei auch wirklich erstellt wurde. Anschliessend wurde das Fenster auf das Hauptfenster mit den Tabs gewechselt. Wenn eine Person nun eine Datei abspeichern wollte, gab es die Funktion „WriteFile“, die einen „BufferedWriter“ verwendete und den Array von der „Angaben-Datei“ übernahm. Da diese Methode allerdings sehr instabil und sehr kompliziert war zum bearbeiten bin ich auf JSON umgestiegen.

A screenshot of a code editor with a dark background and light-colored text. The code is a JSON string representing a game object. It is displayed within a window that has three colored window control buttons (red, yellow, green) in the top-left corner. The JSON string is as follows:

```
1 {  
2   "youtubeLink" : "https://www.youtube.com/watch?v=bV85_gIrHTc",  
3   "ownTeam" : "Fondue Front",  
4   "enemyTeam" : "Jugger Basiliken Basel",  
5   "tournament" : "Zur Goldenen Pompfe",  
6   "players" : [ "Erste Person", "Zweite Person", "Dritte Person", "Vierte Person", "Fünfte Person" ],  
7   "rounds" : [ {  
8     "numberOfRound" : 1,
```

So sieht ein JSON String aus, welche aus meinem Game-Objekt entnommen wurde. Die erste Linie ist dabei das erste Feld im Game-Objekt, youtubeLink. Das ist der Key und dann gibt es noch den Link (https://www.youtube.com/watch?v=bV85_gIrHTc) welches den Wert (Value) darstellt und dann bei der Reader Klasse herausgelesen

wird und dem Youtube-Links zugeordnet wird. Dabei musst ich natürlich die beiden Reader und Writer Klassen ändern.

Writer-Klasse Die Writer-Klasse arbeitet nun mit Jackson, d. h. mit einem ObjectMapper und einem ObjectNode. Ein ObjectMapper mappt dabei ein Objekt (z. B. „Game“) als einen String mit Key-Value-Paaren. Damit die Klasse in Zukunft noch um weitere Informationen erweitert werden kann, habe ich jeweils Funktionen definiert, die das Key-Value-Paar auf dem ObjectNode speichern. Dort bleibt es, bis alle Key-Value-Paare auf dem ObjectNode gespeichert wurden. Dann kommt der ObjectMapper und „übersetzt“ Java-Code in einen JSON-String, der auf eine Datei geschrieben wird.

Reader-Klasse Die Reader-Klasse macht grundsätzlich das Gleiche, nur in umgekehrter Reihenfolge. Sie liest mit dem ObjectMapper die Informationen vom JSON-String und „übersetzt“ sie in Java. Das ObjectNode speichert diesen Code ab und mithilfe der Keys kann man den Value herausholen und abspeichern (z. B. `game.youtubeLink = jsonNode.get("youtubeLink").asText()`). Das muss ich dann nur noch mit allen anderen Feldern machen und schon ist mein Game-Objekt wieder mit den richtigen Informationen gefüllt und ich kann damit arbeiten.

5.1.3 GUI-Controller

Die GUI-Controller (Graphical User Interface Controller) ermöglichen es den JavaFX-Klassen, Eingaben entgegenzunehmen und diese zu verarbeiten. Dies wird mithilfe sogenannter Felder ermöglicht. Diese haben einen speziellen, eindeutigen Namen, der es erlaubt, ein Eingabefeld oder einen Knopf mit dem entsprechenden Feld zu verknüpfen. Somit kann ich im Code dieses Feld speziell bearbeiten oder an andere Klassen weiterleiten.

```
@FXML
private WebView webView;
@FXML
private TextField youtubeURLField;
@FXML
private TextField zuege;
@FXML
private TextField eigenesTeam;
@FXML
private TextField gegenerischesTeam;
@FXML
private TextField turnier;
@FXML
private TextField teamPlayers;
```

Ein Beispielcode aus meiner Klasse „AngabenController“: Mit @FXML weiss Java, dass ich die Bibliothek FXML verwenden möchte, um diese Elemente als Variable abzuspeichern. Somit kann ich später im Code darauf zurückgreifen. So kann ich beispielsweise später beim Textfeld zuege überprüfen, was eingetragen wurde, und weiss dann, wie viele Züge dieses Spiel hat.

- **MainController:** Der MainController ist der GUI-Controller, der dafür zuständig ist, dass die Tabs für das GUI richtig geladen werden, immer das richtige Fenster geöffnet ist und das Fenster richtig benannt ist. Zudem

überprüft er, ob alle Angaben im ersten Tab gemacht wurden, da diese benötigt werden, um die Datei zu erstellen.

- **FileOeffnenController** Der FileOeffnenController hat nur zwei Funktionen. Die erste wird aufgerufen, wenn ein neues File erstellt werden soll – dann wird einfach die Writer-Klasse verwendet, um ein neues File zu erstellen. Die zweite öffnet einen FileChooser, eine Klasse aus der JavaFX-Bibliothek, die es ermöglicht, eine Datei auszuwählen. Diese Datei wird mithilfe der OpenFile-Funktion der Writer-Klasse geöffnet.
- **AngabenController** Im Verlauf der Maturaarbeit hat sich der Angaben-Controller einiges verändert. Die erste Idee war, diesen Controller zu verwenden, um die Eingaben mit den diversen Feldern abzuspeichern. Beim Wechseln auf ein anderes Tab wird überprüft, ob die Felder einen funktionierenden Inhalt haben. Anschliessend sollten sie in einem Array gespeichert werden. Dieses Array ist allen anderen Klassen zugänglich. Da dieses Array fixe Angaben darüber enthält, wo welche Informationen zu holen sind, können diese Klassen die Informationen einfach abfragen, indem sie nach dem Inhalt des richtigen Index fragen. Die Klasse hatte auch einen Getter und einen Setter für das Array, damit das Array privat bleiben konnte. Dies war allerdings ein sehr komplizierter und unlogischer Aufbau, wie beim „Eigene Dateiformat“ besprochen. Deswegen ist der Aufbau nun folgender:
- **EintragenController** Der Eintragen-Controller verfügt wie der Angaben-Controller über viele Felder, in denen die einzelnen visuellen Elemente als Variablen abgespeichert werden. Im Gegensatz dazu hatte der Eingaben-Controller allerdings eine Funktion, die am Anfang aufgerufen wird und nochmals überprüft, ob das File alle Informationen enthält. Sollte dies nicht der Fall sein, wird eine Fehlermeldung ausgegeben. Zusätzlich hatte er zwei Funktionen, mit denen die Daten zu den Zügen oder dem Array des Eingaben-Controllers hinzugefügt werden konnten. Da das Auslesen der YouTube-URL am Anfang noch nicht funktionierte, hatte er eine Funktion, die bei einem Knopfdruck ein bestimmtes Sportvideo abspielte.
- **AuswertungController** Der AuswertungController liest zunächst das File aus, damit alle Felder im Game-Objekt sicher gefüllt sind (z. B. falls die Datei von einem Freund kommt). Danach sorgt er dafür, dass die TableView die richtige Anzahl an Spalten und Zeilen hat, um den Namen und den prozentualen Anteil der Züge einer Person anzuzeigen. Anschliessend wird für jede Person ein neuer Tab hinzugefügt, um die Gewinnrate anzeigen zu können.

5.1.4 Speicherklassen

In meiner Maturaarbeit habe ich Speicherklassen verwendet. Das sind Klassen, die per se keine Funktion haben, aber über Speicherfelder verfügen, sodass ich

diese Klassen als Objekte mit Daten speichern kann. Mithilfe dieser Klassen muss ich nicht immer auf einen bestimmten Eintrag im Array zugreifen, sondern habe alle relevanten Daten in einem Objekt vereinigt. Die Klassen dienen vor allem der Organisation meines Projekts, da sie eine bessere Übersicht bieten als Eingabefelder in einem Array. In meiner Arbeit habe ich die drei Klassen Game, Round und Fight verwendet. Dabei speichert Game auch Round als Feld ab und Round speichert Fight ab. Somit reicht es, wenn ich Game bearbeiten kann, um das gesamte Spiel zu bearbeiten und richtig abzuspeichern. Um zu schreiben, muss ich der Write-Klasse nur das Game-Objekt übergeben. Genauso muss ich beim Auslesen dann nur die Informationen in das Game-Objekt abspeichern.

5.2 GUI

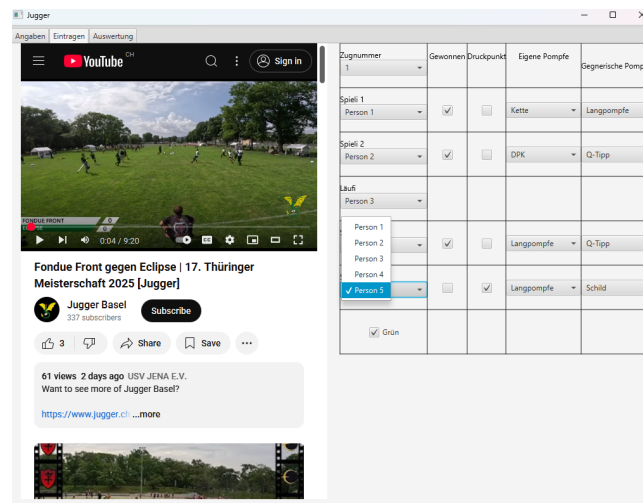
5.2.1 Aufbau

Das GUI wurde mit der JavaFX- und FXML-Bibliothek organisiert. Die grafische Benutzeroberfläche verfügt über eine Haupt-FXML-Datei, die die drei Tabs „Angaben“, „Eintragen“ und „Auswertung“ in einem Tabsystem darstellt. Dann hat das GUI diese drei FXML-Dateien wie eine FXML-Datei für den Anfang, wo man entscheidet, ob man eine Datei öffnen oder erstellen will.

5.2.2 Aussehen

Das Aussehen des GUIs ist sehr grundlegend gehalten. Es ist wahrscheinlich einer der Punkte, der nach dem Abschluss der Maturaarbeit noch viel verändert wird.

Der erste Tab:



Der zweite Tab:

Der dritte Tab:

```

1 {
2   "youtubeLink" : "https://www.youtube.com/watch?v=bV85_g1rHfc",
3   "ownTeam" : "Fondue Front",
4   "enemyTeam" : "Jugger Basiliken Basel",
5   "tournament" : "Zur Goldenen Pompe",
6   "players" : [ "Erste Person", "Zweite Person", "Dritte Person", "Vierte Person", "Fünfte Person"
7 ],
8   "rounds" : [ {
9     "numberOfRound" : 1,
10    "gruen" : true,
11    "fights" : [ {
12      "position" : 1,
13      "name" : "Erste Person",
14      "gewonnen" : true,
15      "druckpunkt" : false,
16      "pompeTyp" : "Stab",
17      "gegnerPompeTyp" : "Langpompe"
18    }, {
19      "position" : 2,
20      "name" : "Zweite Person",
21      "gewonnen" : false,
22      "druckpunkt" : true,
23      "pompeTyp" : "Langpompe",
24      "gegnerPompeTyp" : "Kette"
25    }, {
26      "position" : 3,
27      "name" : "Dritte Person",
28      "gewonnen" : false,
29      "druckpunkt" : false,
30      "pompeTyp" : null,
31      "gegnerPompeTyp" : null
32    }, {
33      "position" : 4,
34      "name" : "Vierte Person",
35      "gewonnen" : true,
36      "druckpunkt" : true,
37      "pompeTyp" : "Kette",
38      "gegnerPompeTyp" : "Einzel-kurzpoppe"
39    }, {
40      "position" : 5,
41      "name" : "Fünfte Person",
42      "gewonnen" : false,
43      "druckpunkt" : false,
44      "pompeTyp" : "Q-Tipp",
45      "gegnerPompeTyp" : "DPK"
46    } ]
47   },
48   "howManyRounds" : 1
49 }

```

- **YoutubeLink:** Inhalt des Links (z.b <https://www.youtube.com/watch?v=rMKYSLi9Rb0>)
- **ownTeam:** Der eigene Teamname (z.b Jugger Basiliken Basel)
- **enemyTeam:** Der Name des gegnerischen Teams (z.b Seven Sins)
- **players:** Eine Arralist von allen Personen welche gespielt haben. Array-
list aus dem simplen Grund, dass ich diese List erweitern kann und muss

- **rounds:** Eine Arrayliste von allen Runden, als Arraylist da es nicht klar ist wie viele Runden ein Spiel hat. Eine Runde ist dabei folgendermassen aufgebaut:
 - **numberOfRound** Eine Zahl, welche Angibt in welcher Runde wir sind
 - **gruen:** Ein Boolean welcher angibt, ob der Jugg geholt wurde (siehe auch Jugg Begriffe: Grün 3.2
 - **fight**s Wieder eine Arrayliste von allen Fights welche im Spiel vorkommen (auch hier muss es ja dynamisch an alle Runden angepasst werden können). Dabei ist ein Fight folgendermassen aufgebaut:
 - * **position:** Um welche Position in der Linie handelt es sich (1-5)
 - * **name:** Wie heisst die Person auf dieser Position
 - * **gewonnen:** Ein Boolean, ob der Kampf gewonne wurde
 - * **druckpunkt:** Ein Boolean, ob die Person sehr aggressiv spielt und somit Druck auf die Gegner ausübt, oder ob die Person reaktiv spielt
 - * **pompfenTyp:** Mit welcher Pompfe hat die Person gespielt
 - * **gegnerPompfenTyp:** Gegen welche Pompfe wurde gespielt
- **howManyRounds** Die Anzahl an Gesamtrunden

5.3.1 Erste Idee

Die erste Idee für das eigene Dateiformat war, eine Mischung aus Array und Arraylist zu verwenden. Da ich zu jedem Zeitpunkt weiss, wie viele Eingaben ein Spiel haben kann, wollte ich dafür einen fixen Array benutzen und diesen dann einfach als Text in einer Datei abspeichern. Da ich die Anzahl Züge und das Spielende in einem Spiel nicht kenne, werden diese jeweils in einer Arraylist eingetragen. Ein Zug besteht wiederum aus einem eigenen Array, da ich hier wieder mit Sicherheit sagen kann, wie viele Eingaben es geben wird. Diese Idee hat jedoch mehrere Schwachstellen. Einerseits ist es sehr umständlich, diese Informationen in den Array zu bringen und wieder herauszuholen, andererseits führt es zu einem sehr komplizierten und unübersichtlichen Code. Da der Array in einer GUI-Controller-Klasse gespeichert wurde, musste man immer Zugriff auf eine Instanz dieser Klasse haben, was wiederum nicht so geschickt ist. Da es keine Abtrennung der einzelnen Informationen gab, war das Auslesen nahezu unmöglich.

5.3.2 Verbesserung der Idee

Die neue Idee wurde gemeinsam mit Herrn Flückiger erarbeitet. Dabei sollte CSV oder JSON verwendet werden. Da JSON das Abspeichern von Daten als Objekte erlaubt und JSON die Daten relativ einfach lesbar für Menschen abspeichert, habe ich mich für JSON entschieden. Dafür musste ich den Code so

strukturieren, dass er mithilfe von JSON die Informationen speichern kann. Um das Ganze nicht wieder über einen einzigen Array laufen zu lassen und somit die Speicherung der Objekte nutzen zu können, habe ich die Speicherklassen geschrieben. Damit habe ich nur ein einziges Game-Objekt, auf dem ich die Informationen einfach und bequem ändern/abspeichern kann. Zunächst hatte ich Probleme mit dem korrekten Speichern der ArrayList, da diese nicht als String, sondern als Objekt gespeichert werden sollte.

6 Diskussion

6.1 Was wurde erreicht?

Wie in der Einleitung angesprochen, war es mein Ziel, dem Verein „Jugger Basiliken Basel“ sowie der gesamten Jugger-Gemeinschaft eine Anwendung zur Verfügung zu stellen, die eine objektive Erfassung von Informationen eines Jugger-Spiels ermöglicht. In dieser Arbeit ist mir dies gelungen. Eine Person kann die Linienduelle, die gegnerische Pompfe und die Druckpunkte eingeben. Da diese Arbeit mit einem eigenen Dateiformat arbeitet, das beliebig erweitert werden kann, kann sie in Zukunft an weitere Bedürfnisse angepasst werden. Da das Programm mit Funktionen arbeitet, die beliebig oft aufgerufen werden können, und beim Lesen nur die Schleife einmal mehr laufen gelassen werden muss, ist der Aufwand, das Dateiformat zu erweitern, gering. Zudem können bei einer Anpassung des Dateiformats alte Dateien weiterhin verwendet werden, da die neuen Informationen einfach leer gelassen werden können.

6.2 Wo gibt es noch Verbesserungspotenzial?

Obwohl vieles in dieser Arbeit gut geklappt hat, hat diese Anwendung auch ihre Grenzen. So ist das Herunterladen von GitHub für nicht technisch versierte Personen eine Herausforderung. Zudem ist, wie bei jeder Anwendung, eine Einarbeitungszeit erforderlich. Excel ist hingegen vielen Personen aus ihrem Beruf oder zumindest aus der Schule bekannt. Das Eintragen der einzelnen Werte ist ausserdem sehr zeitintensiv. Aus programmiertechnischer Sicht muss noch eine Lösung für die unterschiedlichen Tabs her. Das Wechseln zwischen den Tabs sieht zwar elegant aus, allerdings kann das Fenster nicht an die neue Benutzeroberfläche angepasst werden (ohne Neustart). Ein Neustart wäre vor allem bei längeren Öffnungszeiten des Programms nicht benutzerfreundlich, da sich das Programm plötzlich schliesst und erst nach ein paar Sekunden wieder öffnet – das führt häufig zu Frustration. Zudem ist das Überprüfen der Antworten auf einen Tabwechsel beschränkt, was das Anzeigen von Fehlern erschwert und bisher noch dazu führt, dass der Tab doppelt angezeigt wird. Zudem scheint es keinen Weg zu geben in JavaFX eine Youtube-Video mit anderer Qualität als 360p anzeigen zu lassen.

6.3 Was bedeutet diese Applikation für den Sport Jugger?

In der Jugger-Gemeinschaft wird schon länger darüber debattiert, wie Personen objektiver beurteilt werden können. Dabei wurde die Duellquote als Vorschlag oft genannt. Die Idee dahinter ist, dass eine Person effektiver in der Linie eingesetzt werden kann, wenn man objektiv weiss, gegen welche Pompfe sie die besten Quoten hat. Bisher wurde vor allem Excel als Tabelle verwendet, in die man die Linie und die Gewinner der Duelle eingegeben hat. Dies ist jedoch sehr aufwändig und lässt sich schlecht weitergeben und vergleichen, da der Aufbau dieser Tabellen in der Gemeinschaft nicht standardisiert ist. Mit dieser Anwen-

dung können die Dateien weitergegeben werden und das Programm weiss, was diese Daten bedeuten.

6.4 Wie kann diese Applikation weiterentwickelt werden?

Das Ende dieser Maturaarbeit bedeutet nicht automatisch auch das Ende der Anwendung. Mein Ziel ist es, das Projekt weiterzuführen und dabei möglichst Unterstützung aus der Jugger-Gemeinschaft zu gewinnen. Ein erster Schritt besteht darin, Feedback von Anwenderinnen und Anwendern zu sammeln und die Rückmeldungen in die Weiterentwicklung einzubeziehen. Sollten keine dringenden Anpassungen erforderlich sein, möchte ich mich zunächst dem bestehenden Tab-System widmen, da sich hier vermutlich noch elegantere Lösungen finden lassen. Ein weiterer Punkt auf der To-do-Liste ist die Ergänzung einer Funktion, die für jede Spielerin und jeden Spieler anzeigt, welche Position besonders gut funktioniert. Schliesslich muss auch das Problem der niedrigen Videoauflösung angegangen werden, da dies das Erkennen von Treffern erschwert.

Darüber hinaus wäre es sinnvoll, einfache Exportmöglichkeiten (z. B. als CSV- oder PDF-Datei) einzubauen, damit die Ergebnisse auch ausserhalb der Applikation genutzt werden können. Auch eine kleine Automatisierung, etwa durch Tastenkombinationen zum schnelleren Eintragen von Duellen, könnte den Arbeitsaufwand deutlich reduzieren und die Benutzerfreundlichkeit verbessern.

7 Reflexion

7.1 Was habe ich gemacht?

Bei meiner Maturaarbeit handelte es sich um die Entwicklung einer Applikation, die dabei helfen sollte, bei Jugger-Videos Daten besser sammeln zu können. Die Arbeit daran dauerte insgesamt acht Monate und begann mit einer Idee. Auf Basis dieser Idee und des Inputs von Personen aus meinem Jugger-Verein habe ich die Anforderungen an die Anwendung formuliert. Anschließend habe ich diese Anforderungen implementiert und die Anwendung iterativ immer wieder verbessert, sobald ich vor einer Hürde stand. Zentrale Bestandteile der Arbeit waren dabei das eigene Dateiformat mithilfe von JSON sowie die grafische Benutzeroberfläche mit JavaFX. Zudem war die Größe des Programms ein wichtiger Faktor, aufgrund dessen die Verschachtelung der Klassen angepasst werden musste.

7.2 Was habe ich gelernt?

Es gab viele Momente während der acht Monate, in denen ich dazugelernt habe. Der größte Moment war sicherlich der Wechsel von einem einzelnen Array zu einem System aus verschachtelten Klassen. Dabei musste ich lernen, dass nicht alle Lösungen gleich skalierbar sind und dass Übersicht ab einer gewissen Größe des Projekts eine höhere Stellung als Effizienz und Shortcuts innerhalb des Codes hat. Was bei kleinen Programmen aufgrund der Übersichtlichkeit noch durchgeht, wird bei Projekten mit mehreren Klassen und mehreren hundert Zeilen Programmcode viel stärker bestraft. Zu Beginn habe ich zu viel Code innerhalb einer Klasse gelassen, um nicht jedes Mal eine Klasseninstanz zu benötigen oder damit die Funktionen nicht alle statisch sein müssen. Auch dies rächt sich durch Unübersichtlichkeit und ein allgemeines Chaos. Aus dem gleichen Grund habe ich irgendwann auch damit begonnen, mithilfe von Kommentaren in meinem eigenen Programm zu arbeiten, und auch Linien aus Bindestrichen zu verwenden, um gewisse Bereiche in größeren Klassen übersichtlicher zu gestalten. Durch diese Lernmöglichkeiten fühle ich mich nun jedoch um einiges gewappneter, dieses Projekt weiterzuführen, auf einem soliden Fundament aufzubauen oder ein neues großes Projekt zu beginnen. Auch der Einsatz von KI war für mich eine Lernkurve, da ich mich zuvor strikt geweigert hatte, KI zu verwenden. Zunächst wollte ich mithilfe von Copilot arbeiten, allerdings hat sich schnell herausgestellt, dass ich die Vorschläge häufig als nervig empfinde. Copilot schlägt immer einen Code vor, der mithilfe der Tab-Taste eingefügt wird. Ich bin es allerdings gewohnt, dass diese Taste für das automatische Ergänzen einzelner Wörter dient, sodass ich häufig acht unnütze Zeilen erhalte. Dadurch verliere ich mehr Zeit, als ich gewinne, und lerne noch weniger. Deshalb habe ich mich bei der Bugsuche auf CHatGPT beschränkt. Ein Beispiel ist eine Linie in meinem JsonFileWriter. Ich hatte lange das Problem, dass die Runden mehrmals geschrieben wurden, sobald in der ChoiceBox eine neue Runde ausgewählt wurde. Ich kam auf keine Lösung und auch StackOverflow konnte mir leider nicht weiterhelfen. Nach

einigem Hin und Her mit ChatGPT konnte der Fehler schließlich gefunden werden. Ich hatte nicht richtig überlegt und vergessen, `ObjectNode jsonNode = objectMapper.createObjectNode()` zu schreiben. `jsonNode.removeAll()` vor dem Aufrufen der Funktion zu schreiben. Dadurch wird das alte Objekt geleert und es werden nur noch die neuen Ergänzungen hinzugefügt.

7.3 Was würde ich anders machen

Wie bei jeder Arbeit gab es auch bei mir gewisse Dinge, die zu Schwierigkeiten geführt haben. Zum Glück ist man beim Programmieren daran gewöhnt, auf Fehler zu stoßen. Deshalb werde ich nicht jeden Bug einzeln aufzählen. Es gab jedoch ein paar übergreifende Dinge, die besser laufen könnten: Einerseits habe ich Dinge häufiger mal ausprobiert und geschaut, dass es für den Moment funktioniert, ohne viel darüber nachzudenken, wie es in der Zukunft aussehen wird. Darauf würde ich in Zukunft mehr Gewicht legen, vor allem, wenn ich weiß, dass es wieder ein größeres Projekt wird. Andererseits hatte ich Probleme mit dem Texten. Ich programmiere gerne und viel und hatte keine Motivationsprobleme – es hat mir zu viel Spaß gemacht. Aber das Schreiben der Maturaarbeit gehörte leider nicht zu meinen Lieblingsdingen. So habe ich es ab und zu nicht direkt beim Programmieren geschrieben, obwohl dies weniger Arbeit für mich bedeutet hätte. Da ich allerdings einen straffen Zeitplan mit regelmäßigen Checks beim Schreiben der Maturaarbeit eingerichtet habe, konnte ich mich motivieren, am Text zu arbeiten. Deswegen würde ich mich bei der Arbeit noch mehr forcieren, den Text direkt mit dem Recherchieren und dem Coden zu verbinden und den Text immer nach dem Coden auf den neuesten Stand zu bringen.

Literatur

- [1] J. League, *Downloads für Regelwerk*, <https://www.jugger.org/downloads>, Accessed: 2025-04-30, 2023.
- [2] JetBrains, *IntelliJ IDEA*, <https://www.jetbrains.com/idea/>, Accessed: 2025-08-27, 2025.
- [3] Microsoft, *Visual Studio Code*, <https://code.visualstudio.com/>, Accessed: 2025-08-27, 2025.
- [4] Ayaan07, *Java AWT vs Java Swing vs Java FX*, <https://www.geeksforgeeks.org/java-awt-vs-java-swing-vs-java-fx/>, Accessed: 2025-05-29, 2024.
- [5] Overleaf, *Overleaf*, <https://www.overleaf.com>, 2025.
- [6] 1.-S. GmbH, *IEEE Zitierweise – Leitfaden für technische Studiengänge*, <https://www.1a-studi.de/ieee/zitierweise>, Accessed: 2025-05-30, 2025.
- [7] OpenAI, *ChatGPT*, <https://chatgpt.com>, 2025.
- [8] DeepL, *DeepL-Write*, <https://www.deepl.com/de/write>, 2025.

8 Anhang

9 Selbstständigkeitserklärung