

# **DIPLOMARBEIT**

## **JAVACHESS, CHESSPI ANDCHESS**

**Höhere Technische Bundeslehr- und Versuchsanstalt Anichstrasse**

---

### **Abteilung**

### **Elektronik & Technische Informatik**

Ausgeführt im Schuljahr 2017/18 von:      Betreuer/Betreuerin:

Alexander Beiser 5CHEL

Ing. MSc. Markus Signitzer

Marcel Huber 5CHEL

Innsbruck, am 03.04.2018

---

Abgabevermerk:

Betreuer/in:

Datum:

## **ERKLÄRUNG DER EIGENSTÄNDIGKEIT DER ARBEIT**

### EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

---

Ort, Datum

Verfasser, Verfasserinnen

Vor- und Zunamen

---

Ort, Datum

Verfasser, Verfasserinnen

Vor- und Zunamen

## **i ZUSAMMENFASSUNG DES PROJEKTERGEBNISSES**

### **i.i Kurzfassung**

Alexander Beiser und Marcel Huber entwickelten im Zuge ihrer Diplomarbeit 2017/18 ein Schachspiel, welches auf einem Personal Computer (PC) , Android-Smartphone und Raspberry PI spielbar ist. Das Schachspiel basierte auf einem bereits von ihnen geschriebenen rohen „Gerüst“.

Dieses Spiel wurde mit der Programmiersprache Java entwickelt, weiteres war für den Raspberry PI ein Gehäuse zu designen und zu realisieren. Um das Spielvergnügen für den Raspberry PI auch unterwegs zu ermöglichen, wurde eine Akkusteuerung entworfen und realisiert.

#### **i.i.i Alexander Beiser**

Alexander Beiser war für große Teile des Backends zuständig. Ein Hauptteil bestand aus der Entwicklung eines Chess Engine, also eines Zugmechanismus, welcher speziell für die ebenso von Alexander Beiser erschaffene künstliche Intelligenz performiert wurde. Er entwickelte auch die Akkusteuerung für den Raspberry PI und designete das Gehäuse.

#### **i.i.ii Marcel Huber**

Marcel Huber war für weite Teile des Frontend Bereichs zuständig. Der Schwerpunkt lag auf dem Implementieren von JavaFX, dem Verbessern der audiovisuellen Gestaltung und dem Entwickeln eines Netzwerkspielmodus. Auch eine Andorid - Application (App.) sollte entwickelt werden.

## i.ii Abstract

Alexander Beiser and Marcel Huber developed a chess game which is playable on PC, Android-Smartphones and Raspberry Pi as their thesis in the years of 2017 and 2018. The chess game was based on an already written temporary basic version. Said game was developed with the help of the programming language java, furthermore a housing for the Raspberry Pi was designed and realised with assistance of a 3D-printer. To ensure unproblematic gaming for the Raspberry Pi outside as well, a controller for an accumulator was designed and created additionally.

## i.iii Projektergebnis

Das Projekt wurde in weiten Teilen umgesetzt. Die Desktopversion von JavaChess wurde vollständig entwickelt. Sie ist im HotSeat Modus, Netzwerk - Modus und AI - Modus spielbar und wird mithilfe des Grafikframeworks entwickelt.

Auch die Portierung auf den Raspberry Pi und der Gehäusebau für diesen waren erfolgreich.

Um die vorgegebene Richtzeit für das Projekt nicht zu überschreiten und die restliche Zeit in die Behebung einiger Fehler zu investieren, wurde auf die Portierung von JavaChess auf Android - Smartphones und die weitere Entwicklung einer Ansteuerungsschaltung für den Akku des 'ChessPi' verzichtet.

## ii LIZENZ

Das Schachprogramm wird unter der „Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License“ entwickelt.

Dies räumt jeden Menschen folgende Rechte ein:

- **Teilen:** Das Programm darf frei kopiert und weiterverteilt werden.
- **Verändern:** Das Programm darf frei verändert werden. Somit dürfen natürlich Verbesserungen implementiert werden.

Allerdings muss dies unter den folgenden Bedingungen geschehen:

- **Zuschreibung:** Man muss die Namen der Entwickler entsprechend anführen und angeben, ob Veränderungen gemacht wurden.
- **Nicht kommerziell:** Das Programm darf nicht kommerziell benutzt werden.
- **Gleiche Lizenz:** Sobald Veränderungen gemacht wurden, muss die Originallizenz weiter verwendet werden. Auch darf nicht von der obigen genannten Lizenz abgewichen werden.
- **Gesetzeskonform:** Das Programm darf nicht so verändert werden, dass die Nutzung illegal wird.

### **iii DANKSAGUNG**

Vor allem möchten wir unserem Betreuer Ing. MSc. Markus Signitzer für die Unterstützung danken.

Weiters bedanken wir uns bei Fachlehrer Strohmaier für die Unterstützung beim Bau des Gehäuses.

Auch möchten wir uns bei Elektrotechnik Beiser, Andreas Beiser, für die finanzielle Unterstützung bedanken.

Bei unserer Eltern, Brigitte und Andreas Beiser, bzw. bei Gabriele und Alexander Huber.

Die Bereitschaft unserer Beta-Tester, unser Programm auf Herz und Nieren zu prüfen und wenn nötig, sogar Blut dafür zu vergießen.

Alexander Hold und Thomas Klotz, da wir mit ihnen das Programm so weiter entwickelt haben, um es mit ihrer „Launchpad“ Diplomarbeit kompatibel zu machen.

Auch bedanken wir uns bei Raphael Meier, der uns bei dem ursprünglichen Gehäusedesign sehr geholfen hat.

# Inhaltsverzeichnis

<b>Erklärung der Eigenständigkeit der Arbeit</b>	i
<b>i ZUSAMMENFASSUNG DES PROJEKTERGEBNISSES</b>	ii
i.i Kurzfassung . . . . .	ii
i.i.i Alexander Beiser . . . . .	ii
i.i.ii Marcel Huber . . . . .	ii
i.ii Abstract . . . . .	iii
i.iii Projektergebnis . . . . .	iii
<b>ii LIZENZ</b>	iv
<b>iii DANKSAGUNG</b>	v
<b>1 Einleitung</b>	1
<b>2 Vertiefende Aufgabenstellung</b>	2
2.1 Alexander Beiser . . . . .	2
2.2 Marcel Huber . . . . .	2
<b>3 Schach, eine Erklärung</b>	3
3.1 Was ist Schach? . . . . .	3
3.2 Spielregeln . . . . .	3
3.2.1 Zugregel Bauer . . . . .	4
3.2.2 Zugregel Springer . . . . .	5
3.2.3 Zugregel Läufer . . . . .	5
3.2.4 Zugregel Turm . . . . .	5
3.2.5 Zugregel Dame . . . . .	5
3.2.6 Zugregel König . . . . .	5
3.3 Schachmaschinen . . . . .	6

<b>4 Java Chess</b>	<b>7</b>
4.1 Einführung . . . . .	7
4.2 Java Chess - Übersicht . . . . .	7
4.2.1 Blockschaltbild . . . . .	9
4.3 Package- und Klassenübersicht . . . . .	10
4.3.1 audio . . . . .	10
4.3.2 backgroundmatrix . . . . .	10
4.3.3 game . . . . .	11
4.3.4 gui . . . . .	11
4.3.5 images . . . . .	12
4.3.6 launchpad . . . . .	12
4.3.7 network . . . . .	12
4.3.8 saveload . . . . .	13
4.3.9 start . . . . .	13
4.4 Initialisierung . . . . .	14
4.5 Repräsentation der Figuren: . . . . .	14
4.6 Zugmechanik und Local-Mode . . . . .	15
4.6.1 Die Move-Klasse - Funktion . . . . .	16
4.6.2 Die Move-Klasse - Code . . . . .	18
4.7 Schach-, Schachmatt- und Patt-Abfrage . . . . .	25
4.7.1 BackgroundGrid - Klasse . . . . .	25
4.7.2 Schachmatt - Übersicht . . . . .	25
4.7.3 Schach . . . . .	25
4.7.4 Schachmatt . . . . .	26
4.7.5 Schachking . . . . .	27
4.7.6 DRAW (Patt) . . . . .	28
4.7.7 DRAW (Code) . . . . .	29
4.8 Die grafische Benutzeroberfläche . . . . .	31
4.8.1 JavaFX . . . . .	31

4.8.2 Das Darstellen des Schachfeldes . . . . .	32
4.8.3 Klickbarkeit der Schachfelder . . . . .	39
4.8.4 Drag and Drop . . . . .	40
4.8.5 Hervorheben bestimmter Felder . . . . .	40
4.8.6 Startup-Bildschirm und Informationsbildschirm . . . . .	41
4.8.7 Die Menüleiste . . . . .	43
4.8.8 Die GUI - Klasse . . . . .	44
4.8.9 Informations- und Optionspopups . . . . .	44
4.9 Sounds . . . . .	46
4.9.1 Der JavaFX - Soundplayer . . . . .	46
4.9.2 Der alternative Soundplayer . . . . .	48
4.10 LAN-Mode . . . . .	49
4.10.1 Netzwerkprogrammierung unter Java . . . . .	49
4.10.2 Netzwerkprogrammierung in JavaChess . . . . .	50
4.10.3 Verbindungsvorgang des Clients . . . . .	51
4.10.4 Verbindungsvorgang des Hosts . . . . .	53
4.10.5 Spielfluss im LAN-Modus . . . . .	56
4.10.6 Das Heartbeat - System . . . . .	58
4.11 Computer Modus (AI-Mode) . . . . .	62
4.11.1 Prinzipielle Möglichkeiten einer AI . . . . .	62
4.11.2 Verwendete Schach-AI-Funktion . . . . .	63
4.11.3 Der Min-Max-Algorithmus . . . . .	66
4.11.4 Code . . . . .	67
<b>5 ChessPI</b>	<b>74</b>
5.1 RaspberryPI . . . . .	74
5.2 Touchscreen . . . . .	74
5.3 Implementierung von JavaChess . . . . .	75
5.4 Wieso reicht die offizielle JDK nicht aus? . . . . .	78
5.5 Konfiguration des Touchscreens . . . . .	78

5.6 Verwendete JavaChess Version . . . . .	78
<b>6 Akkusteuerung</b>	<b>79</b>
6.1 Kenngrößen des benötigten Akkumulators . . . . .	79
6.2 Wählen des Akkumulators . . . . .	79
6.3 Der Li-Ion Akku . . . . .	80
6.4 Idee der ersten Akkusteuerungsschaltung . . . . .	80
6.4.1 Laden der Akkus . . . . .	81
6.4.2 Entladen der Akkus . . . . .	81
6.4.3 Seriell-Parallel-Schaltung . . . . .	82
6.4.4 Sicherheit . . . . .	83
6.4.5 Schaltung . . . . .	85
6.4.6 Messung . . . . .	86
6.5 Idee der zweiten Akkusteuerungsschaltung . . . . .	87
6.5.1 Laden der Akkus . . . . .	87
6.5.2 Entladen der Akkus . . . . .	88
6.5.3 Sicherheit . . . . .	88
6.5.4 Messung . . . . .	88
6.5.5 Schaltung . . . . .	90
6.6 Verwendete Lösung . . . . .	91
<b>7 Gehäuse</b>	<b>92</b>
7.1 Maße der Bauteile . . . . .	92
7.2 Geplantes mit dem 3D-Drucker gefertigtes Gehäuse . . . . .	92
7.3 Gehäuse - Kunststoffwerkstätte . . . . .	95
<b>8 Android</b>	<b>98</b>
<b>9 Beta</b>	<b>99</b>
9.1 AI und Spielmechanik . . . . .	99
9.2 LAN und GUI . . . . .	101

9.2.1	Josef Scharmer . . . . .	101
9.2.2	Sabrina Schiestl . . . . .	101
9.2.3	Katharina Weide . . . . .	101
9.2.4	Fabian Thurnes . . . . .	102
<b>10</b>	<b>BENUTZERDOKUMENTATION</b>	<b>103</b>
10.1	Installationsanleitung . . . . .	103
10.2	Benutzung des Spiels . . . . .	103
10.2.1	Der Hot-Seat-Modus . . . . .	103
10.2.2	Der LAN - Modus . . . . .	104
10.2.3	Der AI - Modus . . . . .	105
10.2.4	Die erweiterten Optionen . . . . .	106
10.2.5	Alle Menüpunkte . . . . .	107
10.3	Fehlermeldungen und Hinweise auf Fehlerursachen . . . . .	109
10.3.1	Fehler beim Starten des Spiels . . . . .	109
10.3.2	Fehler beim Speichern eines Spielstandes . . . . .	109
10.3.3	Fehler beim Laden eines Spielstandes . . . . .	109
10.3.4	Fehler im LAN - Modus . . . . .	109
<b>I</b>	<b>Abbildungsverzeichnis</b>	<b>I</b>
<b>II</b>	<b>TABELLENVERZEICHNIS</b>	<b>I</b>
<b>III</b>	<b>Literaturverzeichnis</b>	<b>II</b>
<b>IV</b>	<b>Abkürzungen</b>	<b>IV</b>
<b>V</b>	<b>PFlichtenheft</b>	<b>V</b>
V.I	JAVACHESS KONZEPT . . . . .	VI
V.II	Einzubauende Features . . . . .	VI
V.II.I	JavaChess (Desktop Version - Windows) . . . . .	VI
V.II.II	ChessPI (Raspberry PI 3) . . . . .	VII

V.II.III AndChess (Android) . . . . .	VII
V.III Appendix . . . . .	VII
V.III.I Vorhandene Bugs . . . . .	VII
<b>VI Datenblätter</b>	<b>IX</b>
<b>VII ZUSAMMENFASSUNG</b>	<b>XVI</b>
VII.I Alexander Beiser . . . . .	XVI
VII.II Marcel Huber . . . . .	XVII
VII.III-Schlussfolgerung / Projekterfahrung . . . . .	XVIII
VII.IV-Kostenaufstellung und Arbeitsnachweis . . . . .	XIX
VII.IV.I Kostenaufstellung . . . . .	XIX
VII.IV.II Arbeitsnachweis Diplomarbeit . . . . .	XXI

# 1 EINLEITUNG

Alexander Beiser und Marcel Huber sind beide begeisterte Schachspieler, womit die Entwicklung eines Schachspiels naheliegt. Gegen Ende des 4. Jahres der HTL (Höhere Technische Lehranstalt) treffen sie die Entscheidung, ein Schachspiel selber zu entwickeln und keine Diplomarbeit von einer Firma anzunehmen. Für die Entwicklung ihres Schachspiels werden sie von der Firma „Elektrotechnik Beiser“ unterstützt. Diese Firma übernimmt die anfallenden Kosten für die Hardwarekomponenten.

Anfang der 5. Klasse der HTL finden die Sondierungsgespräche mit ihrem Betreuer, Ing. MSc. Markus Signitzer, statt, welcher ihnen bekannt gibt, was im Zuge dieser Diplomarbeit alles erledigt werden muss.

Durch die Gespräche kommt man zum Schluss, dass für die Diplomarbeit ein Schachspiel in Java entwickelt werden soll und dieses auf einem RaspberryPI und auch auf die Android Plattform portiert werden muss. Weiteres wird eine Akkusteuerung für den RaspberryPI entwickelt, ein Gehäuse designt und realisiert.

Das Graphical-User-Interface des Spiels soll mit JavaFX erstellt werden. Das Spiel soll gegen eine selbstentwickelte künstliche Intelligenz, im Hot Seat Modus oder im Local Area Network spielbar sein.

Im Hot Seat Modus spielt man auf einem PC abwechselnd die Partien. Details werden in einem Pflichtenheft festgehalten, dieses Pflichtenheft befindet sich im Anhang (V).

Der einfacheren Lesbarkeit halber verwenden wir bei häufigen Begriffen, wie beispielsweise Spieler, usw., meistens die maskuline Form und bitten die Leserinnen unserer Arbeit, sich genauso wie die männlichen Leser angesprochen zu fühlen.

## 2 VERTIEFENDE AUFGABENSTELLUNG

### 2.1 Alexander Beiser

Überarbeitung des Schachmatt-Algorithmus, Entwicklung der Zugmechanik und Entwicklung einer künstlichen Intelligenz.

Implementierung des Schachspiels auf den Raspberry-PI, gleichzeitiges Designen des Gehäuses für den Raspberry-PI und Entwicklung der Akkuansteuерungsschaltung.

### 2.2 Marcel Huber

Entwicklung der Netzwerkfähigkeit und Implementierung von Java FX. Verbesserung und Weiterentwicklung der auditiven und visuellen Gestaltung. Entwicklung der Android-App.

# 3 SCHACH, EINE ERKLÄRUNG

## 3.1 Was ist Schach?

Um den Aufbau des Programmes nachvollziehen zu können, sollten die Grundregeln des Schachspiels geläufig sein. Hier haben wir versucht, die wichtigsten Regeln kurz zusammenzufassen.

Was ist Schach?

Schach ist ein strategisches Brettspiel, indem es darum geht, die feindliche Seite zu schlagen. Die feindliche Seite hat verloren, wenn der König im Schachmatt steht.

Der Name Schach kommt aus dem persischen „Schah“ und bedeutet so viel wie König, woher der Name „königliches Spiel“ stammt.

Ursprünglich wurde das Spiel vermutlich in Nordindien erfunden und kam im Zuge der islamischen Expansion, von 630 bis ca. 750, nach Europa [1].

## 3.2 Spielregeln

Nach der ersten Erklärung, was Schach ist, kommen wir zu den Spielregeln. Schach wird auf einem 8\*8 karierten Feld gespielt. Die Nummerierung erfolgt horizontal durch das Alphabet a bis h und vertikal durch Ziffern 1 bis 8. Zu Beginn gibt es zwei Teams, meist Weiß und Schwarz, mit jeweils 16 Figuren. Folgende Figuren sind zu Beginn am Feld:

- acht Bauern

- zwei Springer

- zwei Läufer

- zwei Türme

- eine Dame

- ein König

Das Ende des Spiels erfolgt entweder durch schachmatt, Aufgabe eines Spielers oder durch ein Remis/Patt. Schachmatt bedeutet, dass der König bedroht wird und es dem Spieler nicht mehr möglich ist, den König aus dieser Position zu befreien.

Patt Möglichkeiten:

- Ein Patt entsteht, wenn eine der Parteien keinen legalen Zug mehr ausführen kann oder
- durch ein „technisches Remis“, wenn außer den beiden Königen nur mehr ein Läufer oder Springer am Feld ist.
- Wenn 50 Züge lang keine Spielfigur geschlagen oder kein Bauer bewegt wurde und der am Zug befindliche Spieler das Remis verkündet.
- Wenn eine identische Stellung mit identischen Zugmöglichkeiten mindestens drei Mal vorkommt, kann ein Spieler ein Remis beantragen.

### 3.2.1 Zugregel Bauer

- Der Bauer darf einen Schritt nach vorne ziehen, wenn das Feld leer ist.
- Befindet sich der Bauer in der Ausgangsposition und wurde noch nicht gezogen, kann er auch wahlweise zwei Schritte vorrücken.
- Der Bauer schlägt vorwärts, diagonal ein Feld.
- Spezialzug „En Passant“. Dies kann er als einzige Spielfigur, wenn ein feindlicher Bauer zuvor einen Doppelschritt gemacht hat und somit den eigenen Bauern die Option nimmt, den gegnerischen Bauern anzugreifen. Falls er ausgeführt wird, ist der feindliche Bauer vernichtet und der eigene rückt diagonal ein Feld hinter den nicht mehr existierenden Bauern.
- Sobald ein Bauer die gegnerische „Grundreihe“ erreicht, wird ein Bauertausch durchgeführt. Hier muss der Bauer gegen eine Dame eingetauscht werden.

### 3.2.2 Zugregel Springer

- Der Springer darf auf das Feld ziehen, dass zwei Felder horizontal bzw. diagonal und eines diagonal bzw. horizontal (gegengleich) versetzt ist. z.B.: von b8 auf c6

### 3.2.3 Zugregel Läufer

- Der Läufer darf diagonal, so weit er will ziehen und schlagen, jedoch darf er nicht über eine Figur ziehen.

### 3.2.4 Zugregel Turm

- Ein Turm darf horizontal bzw. vertikal ziehen und schlagen so weit er will, jedoch nicht über Figuren hinweg.

### 3.2.5 Zugregel Dame

- Eine Dame darf horizontal, vertikal bzw. diagonal ziehen und schlagen so weit sie will, jedoch nicht über Figuren hinweg.

### 3.2.6 Zugregel König

- Der König kann horizontal, vertikal bzw. diagonal ein Feld ziehen.
- Spezialzug „Rochade“. Dabei wird der König entweder zwei Felder nach links, bzw. zwei Felder nach rechts bewegt. Der Turm bewegt sich dabei drei Felder nach rechts bzw. zwei Felder nach links. König und Turm dürfen bis zu diesem Zug noch nicht bewegt worden sein, weiteres darf keines der Felder über das sie ziehen, bedroht werden.

### 3.3 Schachmaschinen

Seitdem es die Möglichkeit gibt, einen schachspielenden Mechanismus zu bauen, hat man dies auch getan. Zu Anfang war dies noch der „Schach spielende Türke“, welcher 1769 von Wolfgang von Kempelen konstruiert wurde. [1]

Der richtige Durchbruch geschah aber erst durch die Erfindung des Computers. Die Hardware wurde immer leistungsfähiger, wodurch der Mensch als Gegner immer weiter in Bedrängung geriet. 1997 schlug der von IBM speziell entwickelte Schachcomputer „Deep Blue“ den damaligen Schachweltmeister Kasparow, wodurch die künstliche Intelligenz in diesem Bereich offiziell den Menschen überholt hat.

Heutzutage wird gegen Schachcomputer vor allem zu Trainingszwecken gespielt. Solche Schachcomputer finden sich mittlerweile auf so ziemlich jedem Gerät, egal ob Smartphone, Tablet oder PC/Laptop. Meist sind diese Programme aber proprietär und „closed source“. Wir entwickeln deshalb ein „open Source“ Schachspiel, das auf mehreren Plattformen spielbar ist.

## 4 JAVA CHESS

### 4.1 Einführung

Bevor mit der Dokumentation des Programmcodes begonnen werden kann, werden zuerst einige Möglichkeiten beschrieben, wie ein Schachprogramm prinzipiell programmiert werden kann.

Hierfür gibt es zwei Möglichkeiten:

1. Die Figuren kennen ihre Position
2. Das Brett kennt die Positionen der Figuren

Das die Figuren ihre Position kennen, klingt zuerst gar nicht so abwegig. Probleme treten aber auf, sobald schachmatt überprüft werden soll. Hierfür muss überprüft werden, ob irgendeine gegnerische Figur den König schlagen kann, wofür man aber das Objekt der gegnerische Figur benötigt. Dies ist kein programmietechnisches Problem, dadurch entstehen aber längere Wartezeiten.

Falls das Brett die Position der Figuren kennt und diese Figuren lediglich über eine Zahlenmatrix dargestellt werden, ist das Spiel nicht nur sehr viel performanter, es ergeben sich auch große Vorteile beim Entwickeln der künstlichen Intelligenz.

Wir entschieden uns für diese letztere Lösung.

### 4.2 Java Chess - Übersicht

JavaChess ist in der Programmiersprache Java geschrieben. Java ist eine objektorientierte, klassenbasierte Hochsprache der Informatik. Java hat den Vorteil, dass es nicht Hardware gebunden ist und somit ein Programm, geschrieben auf einer Distribution des Betriebssystems GNU/Linux (GNU's Not Unix (GNU)) auf (zumindest theoretisch) allen unterstützten Systemen läuft.

Somit können wir unser Spiel auch auf einem Raspberry-PI lauffähig machen.

Das von uns verwendete Graphical-User-Interface (GUI) Environment ist JavaFX.

Es wurde erstmals im Dezember 2008 den Programmierern zugänglich gemacht und soll das bis dahin als Standard verwendete Java-GUI-Environment „Swing“ersetzen. Die Unterschiede bestehen im Aufbau, wie eine GUI realisiert werden kann, bis hin zu den verbesserten grafischen Effekten, die durch JavaFX möglich sind.

Dadurch wurde die Entscheidung gefällt, Java mit JavaFX zu verwenden.

Java Chess nützt in Folge dessen einige dieser Vorteile aus, vor allem objektorientiertes Design.

#### 4.2.1 Blockschaltbild

Hier wird ein Einblick gegeben, wie JavaChess funktioniert. Dies geschieht anhand von einem Blockschaltbild:

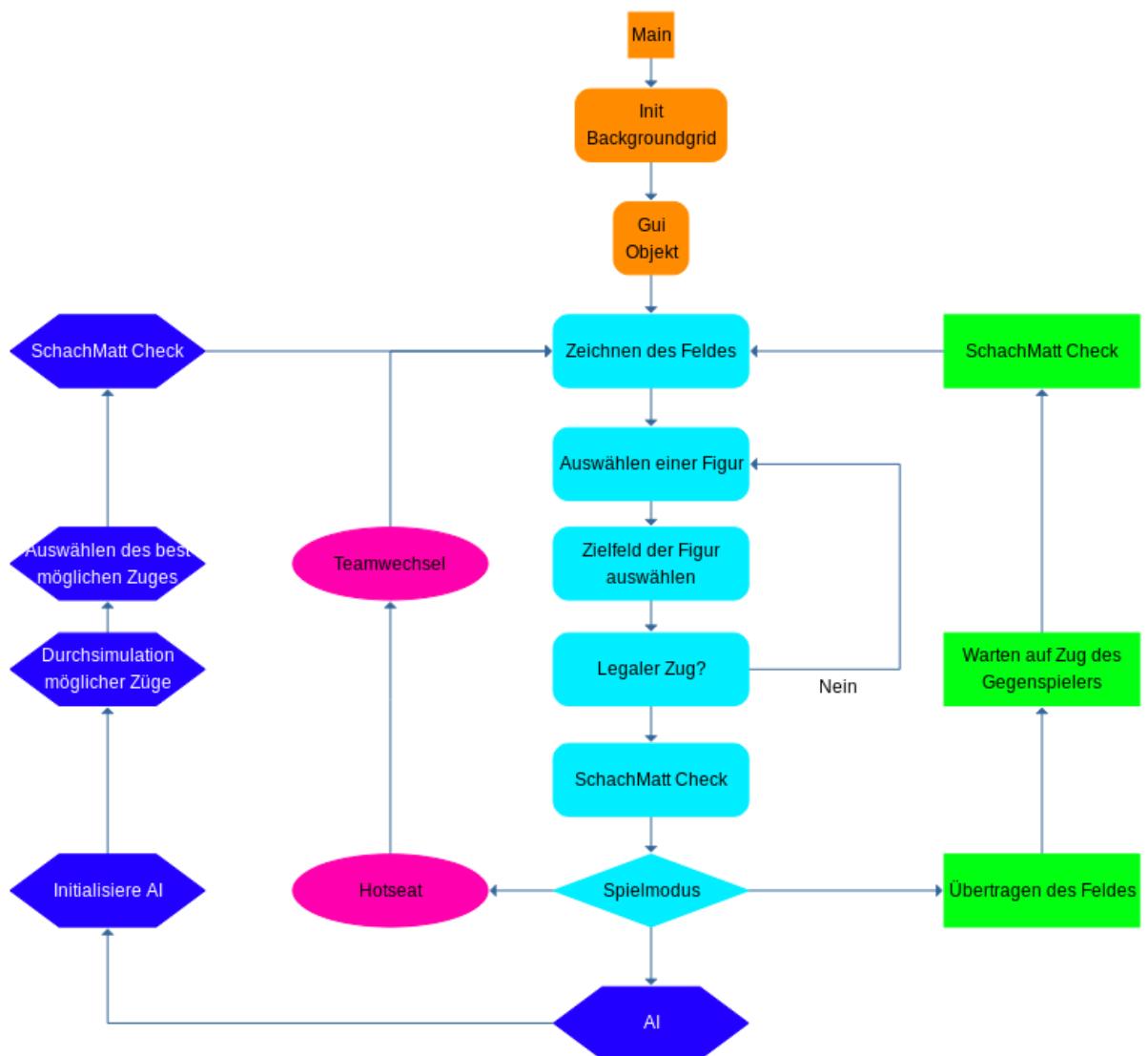


Abbildung 1: Das Blockschaltbild

## 4.3 Package- und Klassenübersicht

Im Folgenden werden die groben Inhalte und Funktionen der Packages und Klassen zusammengefasst.

### 4.3.1 audio

Dieses Package beinhaltet alle Klassen, die benötigt werden, um die Soundeffekte im Spiel abzuspielen. Zusätzlich sind die benötigten Audiofiles enthalten.

- **AudioManager.java:** Diese Klasse managt die zwei Möglichkeiten, Audiodateien im Programm abzuspielen. Warum dies notwendig ist, wird in der entsprechenden Sektion erklärt. Auch die Einstellungsvariablen für die Sounds (z.B. Lautstärke) werden hier gespeichert.
- **JavaFxAudio.java:** Mit dieser Klasse werden die Sounds für den JavaFX-Soundplayer geladen und abgespielt.
- **NativeAudio.java:** Dies Klasse wurde nachträglich hinzugefügt und ermöglicht das Abspielen von Sounds auf Systemen, welche den JavaFX-Soundplayer nicht unterstützen, wie Ubuntu Version 14.1 und später.

### 4.3.2 backgroundmatrix

In diesem Package sind Klassen beheimatet, die für die Spiellogik verantwortlich sind.

- **BackgroundGrid.java:** Dies ist eine „Basisklasse“ der Spiellogik. In ihr werden Schach- und Schachmattabfragen durchgeführt. Zusätzlich werden in ihr verschiedene spielrelevante Variablen gespeichert, beispielsweise das aktuell ziehende Team.
- **Move.java:** In dieser Klasse werden die erlaubten Zugmuster für jede einzelne Figur vermerkt und ausgewertet.

#### 4.3.3 game

Die Klassen für die einzelnen Spielmodi werden in diesem Package abgelegt.

- **AI.java:** Ist quasi Schnittstelle zwischen AILogic und dem restlichem Spiel. Sie wird als neuer Thread gestartet, um das restliche Spiel nicht zu beeinträchtigen. Ruft AILogic auf.
- **AILogic.java:** Hier ist der eigentliche KI-Algorithmus. Wertet alle möglichen Züge nach dem Bestmöglichen aus.
- **AlvsAI.java:** Diese Klasse reguliert den „geheimen Spielmodus“, in dem der Computer gegen sich selbst spielt.
- **LAN.java:** In dieser Klasse werden alle Variablen für den LAN-Spielmodus gespeichert. Zudem werden die Schnittstellen zur Netzwerkkommunikation bereitgestellt.
- **Local.java:** Diese Klasse beinhaltet die Variablen für den lokalen Modus.
- **MovePos.java:** Diese Klasse stellt eine Hilfsklasse dar. Sie wird benutzt, um die momentane, zukünftige, vergangene oder Spezialposition von Figuren zu bestimmen.

#### 4.3.4 gui

In diesem Package sind alle Klassen zusammengefasst, die etwas mit der GUI, also mit dem „Graphical User Interface“ zu tun haben.

- **About.java:** Diese Klasse generiert ein Popup, welches Auskunft über die Entwickler und die Lizenz gibt.
- **BoardGui.java:** In dieser Klasse wird das Schachbrett gezeichnet. Außerdem werden hier Usereingaben mit der Maus verarbeitet und es werden graphische Effekte erzeugt. Auch wird in dieser Klasse ein Teil des Spielflusses, insbesondere das Senden und Empfangen des LAN-Modus, geregelt.

- **GUI.java:** Diese Klasse erweitert die FX - Applikationsklasse. Sie stellt alle grafischen Elemente dar und regelt deren Platzbedarf. Manche Funktionen, wie beispielsweise das Starten eines neuen Spiels oder das Laden eines Spielfeldes, sind hier enthalten
- **Help.java:** Die Helpklasse erzeugt ein Popup, in welchem die Spielanleitung angezeigt wird.
- **Menu.java:** Die Menüklasse erzeugt das Spielmenü. Alle Funktionen aus dem Menü werden hier ausgeführt oder zumindest aufgerufen. Konkret werden Spielmodi gewechselt, Felder gespeichert oder geladen, neue Spiele gestartet und es werden die Popups aufgerufen.
- **Popup.java:** Diese Klasse erzeugt ein Popup, in dem viele Spielemente konfiguriert werden können. Genauer werden die Lautstärke, AI-Schwierigkeit, und weitere kleine Einstellungen angeboten. Zusätzlich werden Informationen zum Spiel angezeigt.
- **Tile.java:** Diese Klasse repräsentiert ein Feld auf dem Schachbrett und enthält dessen Informationen.

#### 4.3.5 images

Dieses Package enthält alle Bilddateien, die benötigt werden, um das Spielfeld darzustellen.

#### 4.3.6 launchpad

In diesem Package sind alle Klassen beinhaltet, welche den Launchpad-Support ermöglichen.

#### 4.3.7 network

Dieses Package beinhaltet alle Klassen, die benötigt werden, um den Netzwerkspielmodus zu ermöglichen.

- **Heartbeat.java:** Diese Klasse stellt sicher, dass der Verlust einer Netzwerkverbindung erkannt wird.
- **hostingJob.java:** Diese Klasse wird benötigt, um das Hosten eines Netzwerkspiels zu regeln. Genaueres dazu unter (4.10.4)
- **ReadingJob.java:** Diese Klasse ermöglicht es, Objekte vom Netzwerk zu empfangen.

#### 4.3.8 saveload

- **Load.java:** Diese Klasse enthält alle notwendigen Funktionen, um Spielstände zu laden. Außerdem werden alle Fehler, die beim Ladevorgang auftreten können, hier bearbeitet.
- **Save.java:** Um Spielstände zu speichern, wird diese Klasse benötigt. Sie enthält alle notwendigen Funktionen dazu und bearbeitet Fehler, die beim Speichern auftreten können.

#### 4.3.9 start

Dieses Package enthält die Klasse „*Main.java*“, welche nur dazu dient, das Spiel zu starten.

## 4.4 Initialisierung

Als Referenz bzw. Hilfe siehe 4.2.1. Zuerst startet das Programm in der Main-Methode der Main-Klasse. Als Nächstes wird das Backgroundgrid-Objekt und das GUI-Objekt von der GUI-Klasse initialisiert.

Dieses Objekt ladet im Anschluss die BoardGui-Klasse, welche eine Abstraktion eines Canvas ist. In einem Canvas ist eine gewisse Art des Zeichnens möglich. Dadurch wird auch das Schachbrett gezeichnet und weiteres findet in dieser Klasse der „Spielfluss“ statt.

Im Default-Modus startet das Spiel im Hot-Seat Game-Mode, siehe 4.6. Hier spielt der Spieler zuerst einen Zug, woraufhin kontrolliert wird, ob der Zug legal ist. Da das Spiel im Hot-Seat-Mode startet, wechselt der Spieler und das Schachbrett wird mit der Aufstellung nach Zug eins neu gezeichnet.

## 4.5 Repräsentation der Figuren:

Die Figuren werden über eine Zahlenmatrix repräsentiert. Dabei bekommt jede Figur eine individuelle Zahl zugeteilt.

Eine solche Zahl besteht aus drei Ziffern, z.B.: 102. Diese ist der zweite weiße Bauer, die erste Ziffer gibt dabei an, ob es Team Weiß (1) oder Schwarz (2) ist. Die zweite Ziffer gibt den Figurentyp an, also Bauer, Turm, etc. Die dritte Ziffer gibt an, die wievielte Figur es ist.

Diese Matrix ist in einem Objekt von der „Background-Matrix“ gespeichert.

Zu Beginn einer jeden Partie wird einmal die Startaufstellung im „Konstruktor“ der „Background-Matrix“ initialisiert:

110	120	130	140	150	131	121	111
101	102	103	104	105	106	107	108
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
201	202	203	204	205	206	207	208
210	220	230	240	250	231	221	211

Tabelle 1: Repräsentation der Figuren

## 4.6 Zugmechanik und Local-Mode

Sobald das Spiel geladen und initialisiert ist, wird automatisch der Hot-Seat Spielmodus ausgewählt. Beim Hot-Seat Modus spielt man auf einem Device nacheinander. Dieses eigentliche Spiel geschieht in einem Objekt der „BoardGui“ Klasse. Die BoardGui Klasse ist ein Canvas Objekt, also ein Objekt, auf dem man zum Beispiel zeichnen kann. Diese Funktion wird ausgenutzt, um das Spielfeld zu zeichnen. Wie dies genau geschieht wird in 4.8.2 erläutert.

Nun ist der weiße Spieler an der Reihe. Welches Team an der Reihe ist, wird durch den Boolean „team“ bestimmt. Dieser Spielstand wird in einem Objekt der Backgroundgrid Klasse gespeichert. True bedeutet, dass der weiße Spieler am Zug ist, false, dass der schwarze am Zug ist.

Der Spieler kann nun die Figur anwählen, die er bewegen möchte, oder die linke Maustaste gedrückt halten und so die Figur via Drag & Drop aufnehmen. In dieser Position werden alle möglichen Bewegungen des Spielsteins angezeigt. Hier ist zu erwähnen, dass potenzielle Angriffe in rot und mögliche Bewegungen in grün dargestellt werden.

Nun muss der Spieler nur noch das Feld, auf das er ziehen möchte, anklicken bzw. die Figur darüber absetzen. Der Move Algorithmus berechnet nun, ob dieser Zug auch möglich ist. Falls dieser Zug erlaubt ist, wird die Hintergrundmatrix, also das Spielfeld, entsprechend mit den neuen Positionen der Figuren, umgeschrieben.

Im Hot-Seat Modus wird jetzt das Team gewechselt und die GUI neu gezeichnet, damit die Änderungen in der Matrix sichtbar werden.

#### 4.6.1 Die Move-Klasse - Funktion

Wie in 4.6 beschrieben, gibt es eine Abfrage, ob ein Zug erlaubt ist. Dafür gibt es aber mehrere Möglichkeiten:

Der erste Ansatz besteht darin, dass jede Figur ihren erlaubten Zug selber überprüft. Hierbei muss klar sein, dass wir für jede Figur ein eigenes Objekt des jeweiligen Klassentypes (z.B.: Bauer) angelegt haben. Für die Zugüberprüfung wird an die Figur die Position übergeben, wohin sie ziehen soll und das momentane Spielfeld, als 8x8 Matrix. Ein Boolean als Rückgabewert indiziert, ob dieser Zug legal ist.

Bevor eine solche Move-Abfrage aber überhaupt durchgeführt werden kann, muss diese auch aufgerufen werden und erkannt werden, welcher Spielstein ausgewählt wurde. Dies geschieht in einem Abschnitt der BoardGui-Klasse.

Der erste Ansatz ist somit nicht wirklich eine Klasse, sondern auf viele Klassen verteilt. Dies funktioniert ohne bekannte Bugs, dadurch wird der Code aber stellenweise sehr unübersichtlich. Weiteres verbraucht das Spiel teilweise mehr Ressourcen als unbedingt notwendig. Eine AI und eine Zugindikation mit diesem Ansatz zu schreiben ist, aber schlicht unvorstellbar.

Der zweite Ansatz besteht darin, alle Zugabfragen in einer Klasse zu implementieren. Dazu wurde dem Objekt dieser Klasse die aktuellste Position und die Zielposition in Form von Koordinaten wie auch das Spielfeld übergeben. Geprüft wird wieder, ob der Spielzug erlaubt ist. Die Abfrage, welcher Spielstein ausgewählt wurde, geschieht über eine separate Klasse.

Vorteile ergeben sich aus der Übersicht. Die Nachteile sind aber nach wie vor der Ressourcenverbrauch, die Schwierigkeit eine Zugindikation zu programmieren und eine KI-Kompatibilität zu schaffen.

Der dritte Ansatz beschäftigt sich mit der Vorschau der möglichen Züge. Man gibt dem Objekt der Move Klasse einfach alles, was bereits im zweiten Ansatz übergeben wurde. Nun wird aber ein Integer-2D-Array aus möglichen Zügen zurückgegeben. Dies funktioniert ohne Probleme.

Das Einzige, das als Problem deklariert werden kann, ist, dass dadurch das „DRY“-Prinzip (Don't repeat yourself) verletzt wird. In der Move Klasse gibt es nun einmal die Abfrage, ob der Zug erlaubt ist und einmal die Abfrage, welche Züge möglich sind. Für die KI ist es von Vorteil, wenn sie alle möglichen Züge eines Spielsteines bekommt. Es sollte aber auch klar sein, welcher Spielstein zuvor auf dem Feld stand, was durch diese Methode nur indirekt möglich ist.

Der vierte Ansatz nimmt sich allen diesen Problemen an, indem der dritte Ansatz ausgebaut wird und eine neue „MovePos“-Klasse eingeführt wird. Die Move-Klasse kann nun als eine Art Black-Box verstanden werden: Man sagt der Move Methode, welche Figur man ausgewählt hat und man erhält alle möglichen Züge als MovePos-ArrayList zurück. Der eigentliche Zug muss aber extern durchgeführt werden.

Das Objekt der Klasse MovePos beinhaltet die alte Position des Spielsteins, die neue Position, die ID des Spielsteins, die ID des Feldes, auf das gezogen wurde und für die Rochade bzw. den En-Passant noch zwei weitere Informationen zu den Feldern, wo und was darauf war. Die Klassen der Figuren werden nicht mehr benötigt.

Dadurch wird für die KI-Berechnung, für die Schachmatt-Methode, für die Zugüberprüfung und für das Anzeigen aller möglichen Züge die gleiche Basisstruktur der Zugberechnung verwendet. Der Unterschied besteht darin, dass die KI direkt die Zugberechnung aufruft, währenddessen die Zugüberprüfung und das Anzeigen der möglichen Züge indirekt, durch die GetMove-Methode, darauf zugreift.

#### 4.6.2 Die Move-Klasse - Code

Die folgende Dokumentation erfolgt als Pseudo-Code.

Zuerst wird die GetMove-Methode beschrieben. Diese wird für die Spielerzüge verwendet. Sobald ein Spieler auf ein Feld klickt, bzw. bereits eines angeklickt hat und nun einen Zug tätigen will, wird diese Methode ausgeführt.

---

```
import...
...
public class Move{
    ...
    public Move(){
        _bSelect = false;
        ...
        das Standardmaessig keine Figur ausgewaehlt wurde
        ...
    }
    ...
    //Methode fuer Spielerzug bzw. zum anzeigen aller moeglichen Positionen
    public int[][] GetMove(Position und ID von Spielfigur, Objekt von
        Hintergrundmatrix){
        ...
        iDiff = Differenz zwischen zuvor ausgewahlter und jetzt ausgewahlter
        Figur/ ausgewahltem Spielfeld

        if(Die Figur darf bewegt werden){
            //ArrayList von der Klasse MovePos-moegliche Zuege
            ArrayList MoveList = Bekomme alle moeglichen Zuege der Figur
                (getMoveMeeple)
            for(Durch alle moeglichen Zuege){
                if(Die Figur darf auf das ausgewahlte Feld ziehen){
```

```

    ...
    Das Feld wird mit den neuen Positionen ueberschrieben
    Das „Feld“ wird in der BackgroundGrid Klasse als
        iBackground[][] gespeichert
    Der Zug wird als getaetigter Zug in die
        zuletzt-getaetigter-Zug ArrayList geschrieben
    ...
}

}

} else {
    ...
    ...
    ...
    if(Position ist Figur){
        ArrayList MoveList = Bekomme alle moeglichen Zuege der Figur
            (getMoveMeeple)
        for(Durch alle moeglichen Zuege){
            if(Zug auf Leeres Feld){
                _MoveList.add(GezogenesFeld)
            }else{
                _HitList.add(GezogenesFeld)
            }
        }
    }
    return GetaendertesSpielfeld
}

```

---

Die getMoveMeeple-Methode gibt alle möglichen Spielzüge einer Figur zurück. Die Rückgabe erfolgt als eine MovePos-ArrayList. Ein Objekt der Klasse MovePos bein-

hält einen möglichen Zug einer Figur, z.B.: Bauer von A2 auf A3.

Die getMoveMeeple-Methode unterscheidet zuerst nach der ausgewählten Figur, sprich Bauer, Turm, Läufer, Springer, Dame oder König.

Je nach Figur wird überprüft, ob die figurspezifischen Züge möglich sind. Falls ein solcher Zug möglich ist, wird dieser in die ArrayList geschrieben, welche am Ende zurückgegeben wird.

---

```
//Herzstueck der Move Klasse - gibt alle moeglichen Zuege zurueck
public ArrayList getMoveMeeple(Spielfeld, Position von Spielfigur){

    new ArrayList MovePos...MP

    if(Bauer){

        //Zuege

        if(einfacher Zug moeglich){

            MovePos Zug...
            ...
            MP.add(Zug)
            ...
        }

        if(zweifacher Zug moeglich){

            MovePos Zug...
            ...
            MP.add(Zug)
            ...
        }
    }

    //Schlaege

    if(weisses Team){

        if(Schlag diagonal nach links moeglich){

            MovePos Zug...
            ...
            MP.add(Zug)
            ...
        }
    }
}
```

```
if(Schlag diagonal nach rechts moeglich){

    MovePos Zug...
    ...
    MP.add(Zug)
    ...
}

} else {

    if(Schlag diagonal nach links moeglich){

        MovePos Zug...
        ...
        MP.add(Zug)
        ...
    }

    if(Schlag diagonal nach rechts moeglich){

        MovePos Zug...
        ...
        MP.add(Zug)
        ...
    }

}

//EnPassant

if(min. 2. Zug){

    ...
    letzterZug = getLastMove
    ...
    if(wenn feindlicher Bauer danebensteht && ID letzter Zug == id
        Bauer daneben && Im letzten Zug 2 Felder bewegt worden sind){

        MovePos Zug...
        ...
        MP.add(Zug)
        ...
    }
}
```

```
        }

    }

}else if(Turm){

    for(i=1 bis 7){

        if(Feld in X bzw. Y Richtung Ziehbar bzw. Figur schlagbar &&

            keine Figur dazwischen) {

                MovePos Zug...

                ...

                MP.add(Zug)

                ...

            }

        }

    }

}else if(Springer){

    if(Feld auf eine von acht Arten ziehbar / schlagbar){

        MovePos Zug...

        ...

        MP.add(Zug)

        ...

    }

}else if(Lauefer){

    for(i=1 bis 7){

        if(Feld in eine von vier Richtungen schlagbar/ziehbar){

            MovePos Zug...

            ...

            MP.add(Zug)

            ...

        }

    }

}else if(Dame){

    for(i=1 bis 7){
```

```
if(Feld in eine von vier Richtungen schlagbar/ziehbar){

    MovePos Zug...
    ...
    MP.add(Zug)
    ...
}

if(Feld in X bzw. Y Richtung Ziehbar bzw. Figur schlagbar &&

    keine Figur dazwischen) {

    MovePos Zug...
    ...
    MP.add(Zug)
    ...
}

}

}else if(Koenig){

    if(Standard Zuege moeglich){

        MovePos Zug...
        ...
        MP.add(Zug)
        ...
    }

    if(Feld 4 Felder links vom Koenig leer){

        if(Check Rochade Bedingungen-alle Felder dazwischen leer-kein

            Feld ist bedroht){

            MovePos Zug...
            ...
            MP.add(Zug)
            ...
        }

    }
}
```

```
        }

        if(Feld 3 Felder rechts vom Koenig leer){

            if(Check Rochade Bedingungen-alle Felder dazwischen leer-kein

                Feld ist bedroht){

                    MovePos Zug...

                    ...

                    MP.add(Zug)

                    ...

                }

            }

        }

        return MP;

    }

    ...

    Methode Bauerntausch - falls ein Bauer die entsprechende Position

    erreicht hat, wird auf diese Position eine Dame gesetzt

    ...

    getSchach //ueberprueft ob Koenig im Schach ist (=vl. illegaler Zug)-via

        Schachmatt Methode

    getSchach2 //ueberprueft ob Koenig im Schach ist (Warnung an Spieler)-via

        Schachmatt Methode

    ...

    getter und setter Methoden fuer Private Variablen

}
```

---

## 4.7 Schach-, Schachmatt- und Patt-Abfrage

### 4.7.1 BackgroundGrid - Klasse

Die Schachmatt-Abfrage befindet sich in der BackgroundGrid-Klasse. Diese Klasse beinhaltet alle wichtigen Variablen und dient intern quasi als Speicher. So wird in der BackgroundGrid-Klasse das momentan am Zug befindliche Team und die Zughistorie abgespeichert.

Diese Klasse implementiert auch „Serializable“, womit ein Objekt dieser Klasse in der Speichern-Funktion in eine Datei geschrieben wird.

### 4.7.2 Schachmatt - Übersicht

Die Schachmatt-Abfrage teilt sich in drei Methoden innerhalb der Backgroundgrid Klasse auf:

- Schach
- Schachmatt
- Schachking

### 4.7.3 Schach

Die Schachmethode kann auf JEDE Figur angewendet werden und gibt TRUE zurück, wenn diese von einer anderen Figur angegriffen werden kann. Logisch gesehen gibt sie FALSE zurück, wenn die Figur nicht angegriffen werden kann.

Im Prinzip werden alle Figuren aufgerufen und überprüft, ob diese die ausgewählte Figur angreifen können.

**Pseudo Code:**


---

```

private boolean Schach(Spielfeld, Lokation der Spielfigur auf die Schach
angewendet werden soll){
    for(alle Figuren){

        if(Figur ist Bauer und kann Spielfigur angreifen){

            return true;

        } else if(Figur ist Turm und kann Spielfigur angreifen){

            return true;

        } else if(Figur ist Springer und kann Spielfigur angreifen){

            return true;

        } else if(Figur ist Läufer und kann Spielfigur angreifen){

            return true;

        } else if(Figur ist Dame und kann Spielfigur angreifen){

            return true;

        } else if(Figur ist König und kann Spielfigur angreifen){

            return true;

        }

    }

    return false;
}

```

---

**4.7.4 Schachmatt**

Die Schachmatt-Methode kann nur auf den König angewendet werden. Diese überprüft nacheinander alle Bedingungen, die erfüllt sein müssen, damit der König wirklich schachmatt ist. Anfangs wird überprüft, ob er dem Angreifer ausweichen kann bzw. schlagen kann. Falls dies nicht möglich ist, wird überprüft, ob der Angreifer selbst geschlagen werden kann.

Anschließend wird überprüft, ob es möglich ist, zwischen dem Angreifer und dem

König mit irgendeiner Figur zu springen.

Pseudo-Code:

---

```
private boolean Schachmatt(Spielfeld, ID und Postion des Koenigs,
    Backgroundgrid Objekt){
    for(Positionen wo Koenig hinziehen kann){
        if(Position nicht bedroht){
            return false;
        }
    }

    if(Schach Methoden auf Angreifer anwenden == TRUE){
        return false;
    }

    for(alle moeglichen Zuege des Angreifers){
        for(alle Figuren des anderen Teams){
            for(alle Zuege der Figur)
                if(Zug moeglich && dadurch Koenig nicht mehr im Schach){
                    return false;
                }
        }
    }
}
```

---

#### 4.7.5 Schachking

Die Schachking-Methode wird immer am Ende eines Zuges aufgerufen und überprüft, ob ein Team schachmatt ist. Falls dieses nicht schach, also der König im schach steht, oder schachmatt ist, wird nachgesehen, ob eine Patt-Situation vorherrscht. Dies geschieht in der Draw-Methode.

Falls ein Team schach ist, wird nachgesehen, ob dieses auch schachmatt ist.

Pseudo-Code:

---

```

public boolean Schacking(team,Spielfeld,auf welche Figur/Position die
    Abfrage gemacht werden soll, Schachmatt/Schach, simulierter Koenig){
    ID = ID des Koenigs

    Schach Abfrage auf ID

    if(wenn Schach nicht zutrifft und Schachmatt ausgefuehrt werden soll){
        Patt Situation soll ermittelt werden
    }

    if(wenn Schach zutrifft und Schachmatt ausgefuehrt werden soll){
        Schachmattabfrage
        if(Schachmatt trifft zu und weisses Team){
            weisses Team verliert
        }
        if(Schachmatt trifft zu und schwarzes Team){
            schwarzes Team verliert
        }
    }
}

```

---

#### 4.7.6 DRAW (Patt)

Die CalcDraw-Methode überprüft, ob ein Unentschieden entstanden ist. Welche Möglichkeiten es für ein Unentschieden gibt, findet man in den Spielregeln.

Zuerst wird gezählt, wie viele Figuren das jeweilige Team noch hat. Wenn nur mehr der König übrig ist, wird überprüft, ob ein Patt vorherrscht. Es werden alle möglichen Züge des Königs durchsimuliert und falls kein gültiger dabei ist, herrscht ein Patt vor. Weiters wird geprüft, ob sich ein Patt durch zu wenige Spielfiguren ergeben hat, falls dies ebenso nicht der Fall ist, werden noch Threefold-Repetition und die 50 Zug Re-

gel überprüft.

Die Methode CalcDraw ist eine Methode mit einem privaten Zugriffsmodifikator, womit nur Methoden innerhalb der BackgroundGrid Klasse darauf Zugriff haben. Die Methode ist eine Erweiterung der SchachKing-Methode, womit diese auch von dort aufgerufen wird.

Der Rückgabetyp der Methode ist ein boolean, welches true zurückgibt, wenn ein Unentschieden vorliegt und ansonsten false.

#### 4.7.7 DRAW (Code)

Folgende Dokumentation des Codes geschieht als Pseudo-Code:

---

```
private boolean CalcDraw(int iID, int[][] iBackground, int KingX, int
KingY, boolean team, BackgroundGrid BGG ){

    int iSum1 = CountMeeplesForTeamWhite;
    int iSum2 = CountMeeplesForTeamBlack;

    if(Wenn der Koenig die letzte Spielfigur eines Teams ist){
        ArrayList-Typ-MovePos KingMoves = Bekomme alle moeglichen Zuege des
        Koenigs;
        for(Durch alle moeglichen Zuege durch gehen){
            ...Mache den Zug auf der Hintergrundmatrix...
            if(Wenn bei einem Zug kein Schach vorherrscht){
                return false; //kein Draw
            }
            ..Mache den Zug rueckgaengig...
        }
    } else if(Wenn zu wenige Spielfiguren uebrig sind, um ein Schachmatt noch
    zu erreichen){
        return true; //Draw
    }
}
```

```

}

if(Die Zugrunde ist >= 50){

    iMeeples = Alle noch vorhandenen Figuren;

    for(bekomme Spielfelder der letzten 50 Zuege und gehe durch diese
        durch){

        iMeepleRef = Noch vorhandene Figuren zu Zug x

        if(iMeeples == iMeepleRef){

            count++; //Ein Zug wurde keine Figur geschlagen

        }

        if(Wenn die letzten 50 Zuege keine Figur geschlagen wurde und kein
            Bauer bewegt hat){

            return true;

        }

        if(Wenn sich in den letzten 50 Zuegen ein Bauer bewegt hat){

            return false;

        }

    }

}else if(TurnRound >= 6){

    Bekomme alle Spielfelder

    if(Wenn eine Spielposition (ganzes Brett) drei Mal im Spiel vorkommt){

        Der Spieler kann ein Unentschieden proklamieren durch „threefold“
        repetition

    }

}

```

---

## 4.8 Die grafische Benutzeroberfläche

Was wäre ein Spiel ohne grafische Benutzeroberfläche? Eine ansehnliche GUI (engl.: *Graphical User Interface*) ist ein Kernbestandteil jeden Spieles. Um unsere Oberfläche vom Jahre 1998, dem Erscheinungsjahr von Swing, dem Toolkit das zuvor von uns verwendet und auch im Unterricht erlernt wurde, in die Gegenwart zu befördern, wird JavaFX verwendet. Mit JavaFX wurde ein modernes und Plattformunabhängiges Framework ausgewählt, um die grafische Oberfläche umzusetzen.

### 4.8.1 JavaFX

JavaFX ist, wie schon erwähnt, das von uns verwendete Grafikframework für Java. Es stellt den designierten Nachfolger von Swing dar. Das Framework soll das Erstellen von plattformübergreifenden, multimedialen GUIs erleichtern und die Lücken, welche die veralteten Frameworks beinhalten, füllen. Um JavaFX zu verstehen, wird nun die grundsätzliche Funktionsweise beschrieben.

Nach dem Start, der nicht mehr mit der „main-Methode“, sondern über eine Launch-Methode erfolgt, wird eine so genannte „Stage“ angezeigt. Vereinfacht kann man sagen, dass es sich dabei um das Fenster handelt, das angezeigt wird.

Auf der Stage werden überlicherweise eine oder auch mehrere „Scenes“ dargestellt. Eine Scene kann man sich wie einen Container vorstellen, der die grafischen Objekte des Programmes enthält.

Der Szenengraph besteht weiters aus Nodes (also Knoten). Es gibt „Parent-Nodes“, die weitere Nodes als Kinder enthalten können. Eine Node ohne weitere Kinder nennt man „Leaf“. Diese Leafs stellen die sichtbaren Elemente der GUI dar, also beispielsweise Buttons, Textfenster oder Ähnliches. Die Parent-Nodes sind sozusagen „innere Knoten“, die man sich als unsichtbare strukturelle Elemente vorstellen kann. Eine Applikation muss die Root-Node für eine Scene angeben.

Die Scene stellt das Bindeglied zwischen dem aus Nodes bestehenden Szenengraphen und dem vom Betriebssystem zur Verfügung gestellten Fenster dar. Um dies

zu verstehen, nehmen wir beispielsweise an, wir wollen die Größe eines Fensters ändern. Sobald die Größe des Fensters verändert wurde, versucht die Scene, die Änderung an die Root-Node weiterzugeben. Dabei wird die Methode „isResizable“ der Root-Node aufgerufen, welche der Stage mitteilt, ob die Node auf Änderungen reagieren möchte. Ist dies der Fall, verändert die Scene die Größe der Nodes, ansonsten belässt die Scene die bestehenden Größen. Auf diese Weise vermittelt die Scene zwischen der Stage und den Nodes.

#### 4.8.2 Das Darstellen des Schachfeldes

Die erste Frage, die bei dem Erstellen der Gui aufgetreten ist, war wie das eigentliche Schachfeld seinen Weg auf den Bildschirm findet. Um diesen ursprünglichen Zweck zu erfüllen, wurde die Klasse „BoardGui.java“, die mittlerweile den Kern der grafischen Oberfläche, in dem das hauptsächliche Spielgeschehen stattfindet, eingeführt.

Die Klasse ist eine Erweiterung der Klasse „Canvas.java“, die mit dem JavaFX Framework Einzug in die Welt von Java gefunden hat. Das Canvas stellt eine Node dar (vgl. 4.8.1) und kann somit direkt auf einer Scene angezeigt werden.

Ein Canvas (dt. Leinwand) kann mit einer Zeichenfläche verglichen werden: Mithilfe eines sogenannten „GraphicsContext“, einer Klasse, die vom Canvas beinhaltet wird, können Methoden aufgerufen werden, die das Canvas in seiner Darstellung verändern. Somit ist es möglich, auf dem Canvas Linien, Formen, Farben und sogar Bilddateien „zu zeichnen“.

Da das Schachfeld im Spielverlauf sehr oft neu gezeichnet werden muss, wurde dieser Vorgang auf eine Methode ausgelagert. Diese nennt sich „redraw“. Was genau geschieht, wenn diese Methode aufgerufen wird, wird im Folgenden erläutert.

Als erstes wird mithilfe des GraphicsContext auf der Größe des gesamten Canvas ein braunes Rechteck erstellt, um eine Grundierung für das Spielfeld zu erhalten. Im Anschluss wird die Methode „DrawGrid(Hintergrundmatrix)“ aufgerufen, die mithilfe der Hintergrundmatrix (vgl. Abb. 1) das eigentliche Spielfeld mitsamt den Figuren

darstellt.

Dort werden zuerst Skalierungsfaktoren erstellt, die es ermöglichen, die Längeneinheiten, die verwendet werden, um Formen zu generieren, unabhängig von der aktuellen Größe des Canvas, anzugeben. Diese Faktoren ergeben sich aus der simplen Teilung der aktuellen Canvasgröße durch den Faktor 100. Dadurch erhält man einen Wert, der einem Prozent der Canvasgröße entspricht. Nun ist es komfortabel möglich, die Positionen und Längen der anzuzeigenden Objekte am Bildschirm mithilfe dieses Skalierungswertes in einer prozentuellen Form anzugeben.

Im nächsten Schritt wird begonnen, ein Raster aus zuerst vertikalen und dann horizontalen schwarzen Linien zu „zeichnen“. Diese bestehen aus dünnen Rechtecken und sollen später die eigentlichen Schachfelder voneinander separieren und für eine saubere Trennung zwischen diesen sorgen.

In einer Schleife werden Werte nach oben gezählt, mit deren Hilfe die Linien in regelmäßigen Abständen dargestellt werden können. Die Abstände, in denen sich die Linien befinden, setzen sich zusammen aus dem Wert, den der Zählstand erreicht hat, multipliziert mit dem Skalierungsfaktor und der Summe aus der Breite einer Linie und der Breite eines späteren Schachfeldes.

Im Folgenden werden die bisherigen Schritte in Form von Pseudo-Code deutlich gemacht.

---

```
SkalierungX = Aktuelle Canvasbreite/100;  
SkalierungY = Aktuelle Canvashoehe/100;  
  
/* Zeichnen eines Rechtecks:  
zeichneRechteck(PoistionX, PositionY, LaengeX, LaengeY);  
*/  
  
//vertikale Linien  
gc.setFill(Color.BLACK);
```

```

for (Zaehlen von 1 bis 9) {
    Abstand in X = (Zahlstand * (Linienbreite in Prozent + Felda
    breite in Prozent) * SkalierungX;

    zeichneRechteck(Abstand in X, 0, Linienbreite * SkalierungX,
    aktuelle Canvashoehe);

}

//horizontale Linien
for (Zaehlen von 1 bis 9)) {
    Abstand in Y = (Zahlstand * (Linienbreite in Prozent + Felda
    breite in Prozent) * SkalierungY;

    zeichneRechteck(0, Abstand in Y, aktuelle Canvasbreite, Linienbreite
    * Skalierungsfaktor);
}

```

---

Anmerkung: *Im eigentlichen Code wird jeweils noch ein „Offset“ hinzugefügt, um Platz für die Beschriftung der Schachfelder zu lassen. Aus Verständlichkeitsgründen wurde hier jedoch auf die Erwähnung von diesem verzichtet.*

Im Anschluss dazu werden die eigentlichen Schachfelder eingefärbt. Dies erfolgt in zwei verschachtelten for-Schleifen, wobei immer eine Zeile von links nach rechts eingefärbt wird, um dann in die nächste Zeile zu springen. Die Farbe der Felder wird nach dem Zählstand der zwei verschachtelten Schleifen ausgewählt: Je nachdem, ob die Feldanzahl gerade oder ungerade ist, muss auch die passende Farbe verwendet werden. Die Positionierung der Felder geschieht mithilfe einer ähnlichen Formel wie zuvor für die Trennlinien. Diese setzt sich zusammen aus dem Zählstand der Schleifen, den Breiten der Felder und Linien und dem Skalierungsfaktor. Nach dem Determinieren der Farben und Längen wird ein Objekt der Klasse „Tiele.java“ erstellt, dem für spätere Zwecke eine ID zugewiesen wird. Zusätzlich erhält

es die Positionierungs- und Farbparameter, die benötigt werden, um ein Schachfeld darzustellen. Danach werden die Felder gezeichnet. Mithilfe des folgenden Pseudo-Codes sollen auch diese Schritte veranschaulicht werden.

---

```

for (y von 1 bis 8 zaehlen) {
    for (x von 1 bis 8 zaehlen) {
        if ((y == gerade) { // Even Odd
            if ((x == gerade) {
                farbe1();
            } else {
                farbe2();
            }
        } else {
            if ((x == gerade) {
                farbe2();
            } else {
                farbe1();
            }
        }
    }
}

PositionInX = Zahelstand in X * Linienbreite + Zaehlstand in X *
Feldbreite;
PositionInY = Zahelstand in Y * Linienbreite + Zaehlstand in Y *
Feldbreite;
Tile Feld = new Tile(PositionInX, PositionInY);
... //eigenschaften des Feldes setzen (Farbe, ID etc.)
ZeichneRechteck(PositionInX * SkalierungX, PositionInY *
SkalierungY, Breite * SkalierungX, Hoehe * SkalierungY);

```

---

Anmerkung: Auch hier wurden die „Offsets“ für die Feldbeschriftungen der Einfach-

heit halber nicht erwähnt.

Die Darstellung der Figuren erfolgt über eine ähnliche Technik. Wieder handelt es sich um zwei verschachtelte Schleifen, die Zeile für Zeile abgearbeitet werden. Diesmal wird jedoch nicht unterschieden, ob die Zähler in den Schleifen gerade oder ungerade sind, sondern nach dem zugrundeliegenden Hintergrundraster unterscheiden.

Die Figuren werden nicht über vordefinierte Formen dargestellt, sondern als Bilddateien geladen und mithilfe des GraphicsContext auf die Canvas gebracht. Die Position der Figuren unterscheidet sich bis auf einen Faktor, der die Figuren in die Mitte eines Feld setzt, nicht von den Positionen der Spielfelder.

In folgendem Pseudo-Code wird auch dieser Teil des Codes veranschaulicht.

---

```
for (y von 1 bis 8 zaehlen) {  
    for (x von 1 bis 8 zaehlen) {  
        PositionInX = PositionDesFeldesInX+Figurenkorrektur;  
        PositionInY = PositionDesFeldesInX+Figurenkorrektur;  
        Figurenzahl = Hintergrundraster bei Zeile x und Spalte y;  
        //weisses team  
        if (Figurenzahl < 110 && Figurenzahl >= 100) {  
            ZeichneFigur(PositionInX, PositionInY, weisser Bauer);  
        } else if (Figurenzahl >= 110 && Figurenzahl < 120) {  
            ZeichneFigur(PositionInX, PositionInY, weisser Turm);  
        } else if (Figurenzahl >= 120 && Figurenzahl < 130) {  
            ZeichneFigur(PositionInX, PositionInY, weisser Springer);  
        } else if (Figurenzahl >= 130 && Figurenzahl < 140) {  
            ZeichneFigur(PositionInX, PositionInY, weisser Laeufer);  
        } else if (Figurenzahl >= 140 && Figurenzahl < 150) {  
            ZeichneFigur(PositionInX, PositionInY, weisse Koenigin);  
        } else if (Figurenzahl == 150) { // white king
```

```

        ZeichneFigur(PositionInX, PositionInY, weisser Koenig);

    }

    //schwarzes Team

    if (Figurenzahl < 210 && Figurenzahl >= 200) {

        ZeichneFigur(PositionInX, PositionInY, schwarzer Bauer);

    } else if (Figurenzahl >= 210 && Figurenzahl < 220) {

        ZeichneFigur(PositionInX, PositionInY, schwarzer Turm);

    } else if (Figurenzahl >= 220 && Figurenzahl < 230) {

        ZeichneFigur(PositionInX, PositionInY, schwarzer Springer);

    } else if (Figurenzahl >= 230 && Figurenzahl < 240) {

        ZeichneFigur(PositionInX, PositionInY, schwarzer Laeufer);

    } else if (Figurenzahl >= 240 && Figurenzahl < 250) {

        ZeichneFigur(PositionInX, PositionInY, schwarze Koenigin);

    } else if (Figurenzahl == 250) { // black king

        ZeichneFigur(PositionInX, PositionInY, schwarzer Koenig);

    }

}

}

```

---

Um die Erstellung des Spielfeldes abzuschließen und eine Beschriftung für das Schachfeld einzubauen, werden zwei weitere Schleifen (diesmal allerdings nicht mehr verschachtelt) eingebaut. Anhand der Zählstände der Schleifen werden die passende Position der Beschriftung und der passende Buchstabe bzw. die passende Zahl bestimmt. Bevor der Text eingefügt wird, werden noch braune Flächen generiert, die genau in den zuvor freigelassenen Offset passen. Anhand von folgendem Pseudo-Code wird auch dieser Vorgang veranschaulicht.

---

```

for (y von 0 bis 7 zaehlen) {

    gc.fillRect(BreiteEinerLinie*SkalierungX, y*Offset*SkalierungY,
                Breite*SkalierungX, BreiteEinesFeldes*SkalierungY);

    bezeichnung=y+1;
}

```

---

```
    schreibeText(y);  
}  
  
  
for (x von 0 bis 7 zaelhen) {  
    gc.fillRect(x*Offset*SkalierungX, BreiteEinerLinie*SkalierungY,  
    BreiteEinesFeldes*SkalierungX,Breite*SkalierungY );  
  
  
switch (x) {  
    case 0:  
        s = "A";  
        break;  
    case 1:  
        s = "B";  
        break;  
  
    ...  
  
    case 7:  
        s = "H";  
        break;  
}  
    schreibeText(s);  
}  
}
```

---

#### 4.8.3 Klickbarkeit der Schachfelder

Um den dargestellten Feldern nun auch eine Funktion zu geben, muss man sie dazu bringen, auf Mauseingaben zu reagieren.

Unter JavaFX können die meisten Nodes mit so genannten „ActionListener“ bzw. „EventListener“ versehen werden. Dies sind Objekte, die ab ihrem Erstellungszeitpunkt auf einen bestimmten Typus von Ereignis warten. Die beschriebenen Ereignisse können Mauseingaben, aber auch Tastatureingaben oder die Veränderung einer bestimmten Variable sein.

Auch die Klasse „BoardGui.java“ kann mit solchen Event-Handlern versehen werden, da sie ja eine Tochterklasse der unter JavaFX standardmäßig enthaltenen Canvas-Klasse ist.

Um die Schachfiguren zu bewegen, wurde die Methode „setOnMouseReleased“ der BoardGui-Klasse verwendet. Diese Methode ist ein EventListener, der seinen Code ausführt, sobald ein Mausklick aufhört. In diesem Listener wird nun jener Code ausgeführt, der die korrekte Aktion für den gerade aktiven Spielmodus ausführt.

In folgendem Pseudo-Code wird dies deutlich gemacht.

---

```
for(jedes Feld){  
  
    if(Feld ageklickt && Feld mit Figur besetzt){  
        Figur angewaehlt;  
    }else if(Feld angeklickt && Feld nicht mit Figurbesetzt && Figur  
        angewaehlt){  
        zugabfrage();  
        if(Zug ist gueltig){  
            if(Spielmodus ist lokal){  
                ... //Lokaler Modus  
            }  
            if(Spielmodus ist LAN){
```

```
...      //LAN-Modus  
}  
  
if(Spielmodus ist AI){  
...      //AI-Modus  
}  
}  
}  
}
```

---

#### 4.8.4 Drag and Drop

Um eine alternative Möglichkeit zu bieten, die Figuren zu bewegen, wurde implementiert, Züge auch mittels Ziehen und Loslassen der Figuren durchführen zu können.

Dafür wurden zwei weitere Event-Listener benötigt: „setOnMouseDragged“ und „setOnMousePressed“. Das Anwählen einer Figur wurde von der zuvor benutzten Methode „setOnMouseReleased“ in die neue Listener-Methode „setOnMousePressed“ verschoben. Diese führt ihren Code aus sobald ein Mausklick beginnt und nicht sobald er aufhört. Dies ist notwendig, da bei einem „Drag“, also beim Ziehen einer Figur, die Maus gedrückt gehalten werden muss.

Die Methode „setOnMouseDragged“ ruft immer, sobald eine Figur gezogen wird, die Redraw-Methode, also die Methode zum Zeichnen des Feldes, auf, wobei die Figurenposition der gezogenen Figur nun einfach auf die Position des Mauszeigers gesetzt wird.

Wenn das Ziehen einer Figur beendet ist, wird wieder der Code in der Methode „setOnMouseReleased“ ausgeführt (vgl. 4.8.3).

#### 4.8.5 Hervorheben bestimmter Felder

Um auch Schachanfängern ein gutes Spielerlebnis zu ermöglichen, wurde die Funktion eingebaut, Felder hervorzuheben, auf die eine Figur ziehen kann.

Es gibt drei verschiedene Fälle, in denen Felder speziell markiert werden:

- Wenn auf eine Figur geklickt wird, werden alle leeren Felder, auf die die Figur ziehen kann, mithilfe einer Methode blau umrandet. Dafür wird für jedes leere Feld ermittelt, ob die Figur darauf ziehen darf. Ist dies der Fall, so wird die Markierung durchgeführt.
- Ähnlich dem Markierungsvorgang für leere Felder werden schon besetzte Felder auch überprüft. Ist ein Zug möglich, darf die Figur auf dem Feld also geschlagen werden, so wird das entsprechende Feld, mithilfe einer eigenen Methode, rot umrandet.
- Wenn ein Zug stattgefunden hat, wird das Feld, auf dem die Figur zuvor gestanden ist und jenes, auf dem die Figur nach dem Zug steht, grün eingefärbt. Dazu werden die betreffenden Felder nach dem Zug gespeichert. Das Einfärben geschieht dann direkt in der redraw-Methode, die um diese Funktion erweitert wurde.

#### 4.8.6 Startup-Bildschirm und Informationsbildschirm

Um beim Starten des Programmes oder beim Warten auf eine Verbindung im LAN-Modus einen ansprechenden Informationsbildschirm darzustellen, wurden mehrere Methoden implementiert, die zwar unterschiedliche Schaltflächen anzeigen, jedoch im Kern gleich aufgebaut sind. Im Folgenden wird die grundsätzliche Erstellung dieser Informationsbildschirme erläutert

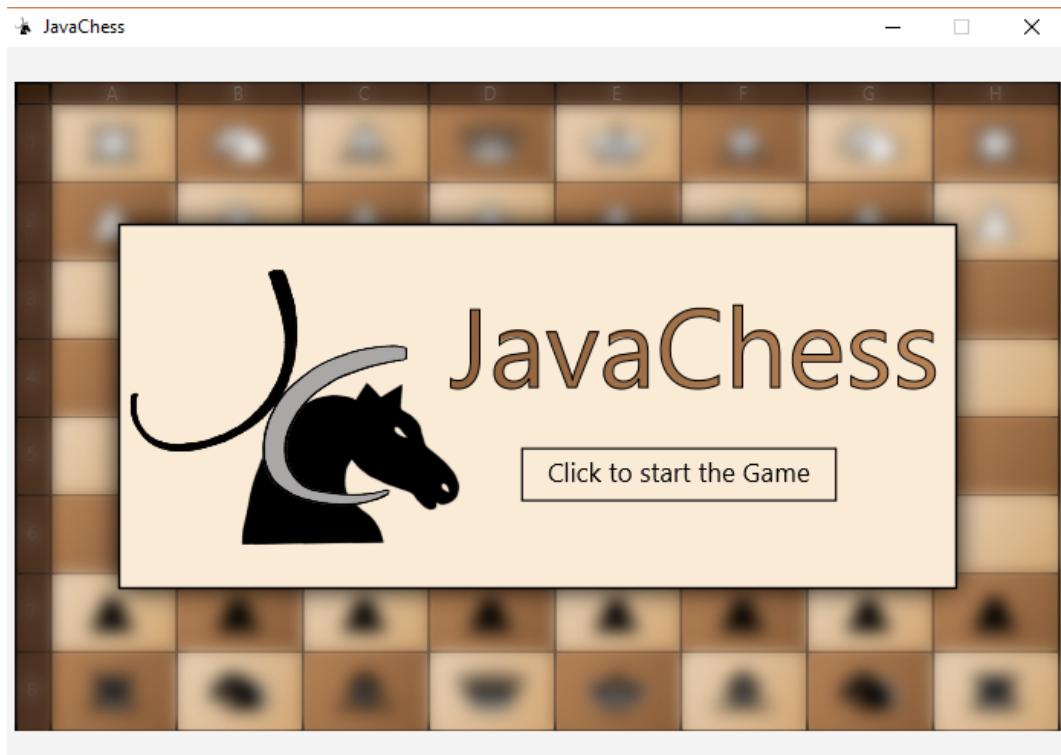


Abbildung 2: Informationsbildschirm beim Starten des Spiels

Als erstes wird die Klickbarkeit der Schachfelder mittels einer speziellen Deaktivierungsvariable entfernt. Im Anschluss wird ein Objekt der Klasse „BoxBlur“ erstellt. Dies ist ein graphischer Effekt, der standardmäßig mit JavaFx ausgeliefert wird. Man kann ihn vergleichen mit einer Art Filter, den man über graphische Objekte legen kann. Alles hinter diesem Filter erscheint dann verschwommen. Mithilfe des GraphicsContext wird dieser Effekt dann auf unsere „BoardGui“ angewandt. Im Anschluss wird ein braunes Rechteck generiert, das in die Mitte des Bildschirms gezeichnet wird. Auf diesem Rechteck wird später der Text des jeweiligen Informationsbildschirmes angezeigt. Um dieses Rechteck in den Vordergrund zu heben, wird ihm der Effekt „dropShadow“ hinzugefügt. Dies ist auch ein mit JavaFX mitgelieferter Effekt, der bewirkt, dass das dargestellte Objekt einen dezenten Schatten wirft. Der dargestellte Text wird je nach Situation ausgewählt. Beim Starten des Programmes wird das Logo und der Titel angezeigt, im LAN-Spielmodus die entsprechenden Informationen (vgl. 4.10.4) angezeigt. Zum Schluss wird noch ein „Event-Handler“ hinzugefügt, der auf Mausklicks rea-

giert. Je nach Situation bewirkt dieser Klick entweder nur ein Verschwinden des Informationsbildschirmes, oder es werden zusätzliche Befehle ausgeführt (vgl. 4.10.4).

#### 4.8.7 Die Menüleiste

Um den Spielmodus zu ändern, das Schachfeld abspeichern zu können, oder weitere Optionen und Hilfefenster aufrufen zu können, soll am oberen Rand des Bildschirmes eine Menüleiste angezeigt werden.

Diese Menüleiste wird mithilfe der in JavaFx enthaltenen „MenuBar“ - Klasse realisiert. Diese stellt eine weitere Node dar, die speziell dafür gedacht ist, weitere Nodes in Form von Menüpunkten zu beinhalten und darzustellen. Die von uns verwendete Klasse „Menu“ erbt von dieser Klasse. Im Folgenden wird beschrieben, wie die „Menu“-Klasse arbeitet.

Als erstes werden alle benötigten Untermenüs, in Form von den JavaFx bereitgestellten Untermenüs, erstellt. Im Gesamten werden vier dieser Menüs benötigt:

- **Game:** Dieses Menü beinhaltet alle Menüpunkte, die verwendet werden, um ein neues Spiel zu starten oder Spielstände zu laden und zu speichern.
- **Gamemodes:** In diesem Menü werden die verschiedenen Spielmodi ausgewählt. Zudem wird, wenn der LAN-Modus aktiv ist, eine Option angezeigt, um die Verbindung zu trennen.
- **Other:** Weitere Informationen und die Option, ein Patt auszulösen, werden in diesem Menü untergebracht.
- **Help:** In diesem Menü können Informationen zum Spiel und die Hilfeseite aufgerufen werden.

Dem jeweiligen Untermenü werden dann die entsprechenden Menüpunkte hinzugefügt. Auf jeden dieser Menüpunkte wird ein „EventListener“ aufgesetzt, der den richtigen Code beim Auswählen eines Menüpunktes ausführt. Die genaue Funktionalität der einzelnen Menüpunkte wird unter anderem in 4.8.9 beschrieben.

#### 4.8.8 Die GUI - Klasse

In der GUI-Klasse werden die Nodes zusammengeführt. Diese Klasse erweitert die „Applikation“-Klasse, also jene Klasse, die als Hauptklasse einer JavaFX Applikation angesehen werden kann. Sie wird beim Starten des Programmes initialisiert. In ihr werden die BoardGui-Klasse, die benötigt wird, um das Spielfeld zu zeichnen, und die Menu-Klasse, welche die Menüleiste mit allen ihren Untermenüs darstellt, initialisiert. Im Anschluss wird ein Objekt der Klasse „BorderPane“ erstellt. Dies ist auch eine Node, welche die Funktion hat, das Layout anderer Nodes auf dem Bildschirm zu regeln. Im Anschluss werden die Nodes „BoardGui“ und „Menu“ zu dieser BorderPane hinzugefügt. Diese BorderPane wird nun zu einer neu erstellten Scene hinzugefügt, welche wiederum zu der neu erstellten Stage hinzugefügt wird. Diese Stage wird dann am Bildschirm angezeigt. Zuletzt wird noch der Startupbildschirm (vgl. 4.8.6) angezeigt (vgl. Abb. 2).

#### 4.8.9 Informations- und Optionspopups

Um zusätzliche, nicht spielrelevante Informationen anzuzeigen, wurden die drei folgenden Klassen erstellt:

- Popup.java
- Help.java
- About.java

Die Klassen „About“ und „Help“ ähneln sich strukturell sehr. Das liegt daran, dass beide Klassen nur dazu da sind, Informationen anzuzeigen. Um diese Popups anzuzeigen, werden jeweils in der Klasse eine neue Stage und eine neue Scene erstellt, die bei Bedarf angezeigt werden können.

In der „About“-Klasse werden Informationen in Form von Labels, also Textflächen, bezüglich der Entwickler und der Lizenz des Programmes angegeben. Auch das Schachlogo wird angezeigt. Diese Nodes werden zur neu erstellten Scene hinzuge-

fügt. Die Positionierung der Nodes erfolgte nicht skalierbar, also in absoluten Positionen.

In der „Help“-Klasse werden die anzuzeigenden Objekte prinzipiell gleich erstellt wie in der „About“ - Klasse. Da die Spielanleitung, die diese Klasse beinhaltet, allerdings relativ umfangreich ist, wurde der Text zu einer „ScrollPane“ hinzugefügt. Dies ist eine Node, die sich der Länge des Textes anpasst, der ihr übergeben wird. Wenn der Text über die Größe der ScrollPane hinausgeht, wird ein Scrollbalken angezeigt, mit dem durch den Text gescrollt werden kann.

In der „Popup“-Klasse werden zusätzliche Optionen zum Spiel bereitgestellt. Diese Optionen werden mithilfe von „Slidern“, „Buttons“ und „Checkboxen“, welche alle Kontrollelemente aus der JavaFX-Bibliothek darstellen, und auch zu einer neuen Scene und Stage hinzugefügt werden, verändert. Mithilfe von Listenern kann auch hier wieder ein spezieller Code beim Betätigen eines solchen Objektes ausgeführt werden.

Zusätzlich werden in dieser Klasse Informationen über das aktuelle Spiel, genauer die Rundenanzahl und das aktuell ziehende Team, mithilfe von Labels angezeigt.

Da alle drei Klassen eine neue Stage eröffnen, können sie parallel zum Spielfenster neue Fenster anzeigen. Dies muss auch nicht manuell auf neue Threads ausgelagert werden, da die JavaFX-Applikation diesen Vorgang automatisch beim Öffnen einer neuen Stage vornimmt.

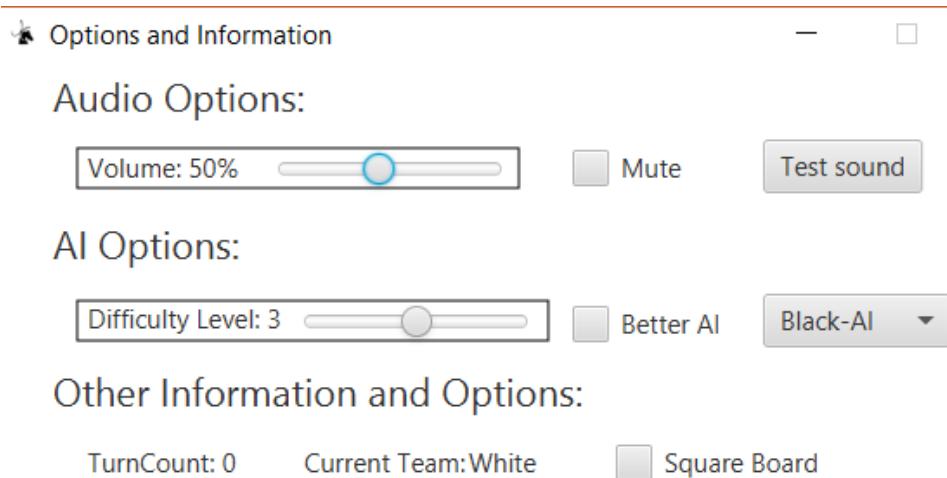


Abbildung 3: Optionspopup

## 4.9 Sounds

Um dem Spieler eine bessere Erfahrung zu bieten, wurde die Entscheidung getroffen, Soundeffekte in das Spiel einzubinden. Die Entscheidung, das durch JavaFX bereitgestellte Sound - Framework zu verwenden, war naheliegend.

### 4.9.1 Der JavaFX - Soundplayer

Der Soundplayer sollte als eigene Klasse ausgeführt werden. Um ein einfaches Implementieren der Effekte zu ermöglichen, sollten die Sounds über eine spezielle Methode ausgewählt und abgespielt werden.

Als erstes werden die Audiodateien in das Programm geladen. Danach wird für jeden Sound mithilfe dieser Dateien ein Objekt der Klasse „Media“ erstellt. Dies ist eine von JavaFX mitgelieferte Klasse und repräsentiert den Soundeffekt im Programm. Um die Effekte in der Lautstärke verändern zu können, oder sie ganz auszuschalten, werden zwei Variablen eingebaut. Diese können zur Laufzeit in den Optionen (vgl. 4.8.9) verändert werden.

Um die Sounds später im Programm abspielen zu können, wird die zuvor genannte Methode vorbereitet. Der Methode wird beim Aufrufen ein String mitgegeben, mit dem ausgewählt werden kann, welcher Sound abgespielt wird.

Zuerst wird in dieser Methode überprüft, ob der Sound ausgeschaltet ist. Im Anschluss wird anhand des Strings ausgewertet, welcher Effekt abgespielt werden soll. Nach der Auswertung wird ein Objekt der Klasse „MediaPlayer“ erstellt, welches es ermöglicht, die zuvor erstellten „Media“ - Objekte abzuspielen. Ihm wird das entsprechende „Media“ - Objekt mitgegeben. Danach wird die Lautstärke des „MediaPlayer“ - Objektes entsprechend der Variable gesetzt und der Sound wird abgespielt.

Der folgende Pseudo-Code veranschaulicht den beschriebenen Vorgang.

---

```
//Constructor  
public JavaFxAudio() {
```

```
Effekt1 = new Media(Soundfile1);
Effekt2 = new Media(Soundfile2);

...
Effekt3 = new Media(Soundfile3);

}

//Methode zum Abspielen der Sounds
public void playSound(Auswahl1) {
    if (Sound ist nicht Ausgeschaltet) {
        switch (Auswahl) {
            case "move":
                Player = new MediaPlayer(Effekt1);
                break;
            case "menu":
                Player = new MediaPlayer(Effekt1);
                break;
            case "startup":
                Player = new MediaPlayer(Effekt1);
                break;
            //Wenn kein Effekt ausgewaehlt wird
            default:
                break;
        }
        Player.setzeLautsaerke(Lautstaerke);
        Player.abspielen();
    }
}
```

#### 4.9.2 Der alternative Soundplayer

Nach dem Veranstalten interner Tests wurde festgestellt, das unter manchen Linux-Betriebssystemen die Initialisierung des Soundplayers fehlschlägt.

Nach umfangreichen Recherchen wurde das Problem festgestellt: Der JavaFX-Soundplayer verwendet Bibliotheken, die seit Ubuntu 14.0 nicht mehr unterstützt werden. Ohne diese Bibliotheken ist ein Abspielen des Sounds nicht möglich.

Um das Problem zu umgehen wurde entschieden, einen Soundplayer zu entwickeln, der ohne die JavaFX - Bibliotheken auskommt. Dieser verwendet das Standard - Audioframework unter Java. Das Konzept dieser Klasse ähnelt dem des ursprünglichen Soundplayers und wird deshalb nicht näher erläutert. Für Details ist der beiliegende Code zu Rate zu ziehen.

Der Nachteil, den der native Soundplayer bietet ist, dass er weniger optimiert, ist und im Vergleich zum ursprünglichen Soundplayer wesentlich mehr RAM verbraucht. Um eine optimale Nutzung zu gewährleisten wurde eine „Audiomanager“ - Klasse eingebaut, die beim Start des Spiels versucht, den „FX - Soundplayer“ zu initialisieren. Im Falle, dass dies fehlschlägt, wird stattdessen der alternative Soundplayer eingesetzt.

## 4.10 LAN-Mode

Der LAN-Modus ist jener Spielmodus, mit dem die Spieler auch auf getrennten Computern miteinander spielen können. Dabei müssen sie sich in einem LAN, also im Local Area Network, befinden. Um zu verstehen, wie der LAN-Modus arbeitet, wird eine kurze Einführung in die Netzwerkprogrammierung unter Java gegeben.

### 4.10.1 Netzwerkprogrammierung unter Java

Um unter Java eine Verbindung zwischen zwei Computern herzustellen werden so genannte „Sockets“ verwendet. Ein Socket ist ein Objekt, das die Netzwerkverbindung zwischen zwei Maschinen repräsentiert.

Unterschieden wird zwischen den Sockets der Clients, also jenen Teilnehmern, die eine Verbindung aufbauen wollen, und den Sockets der Server, also den Teilnehmern, die auf eine eingehende Verbindung warten.

Ein Objekt der Klasse `Socket` repräsentiert die Verbindung eines Clients, ein Objekt der Klasse `ServerSocket` die eines Servers.

Der große Unterschied zwischen den beiden Klassen ist, dass mit dem `ServerSocket` keine Daten gesendet oder empfangen werden. Er ist einfach nur dazu da, auf Verbindungen zu warten und diese anzunehmen. Sobald eine gültige Verbindung vorliegt, gibt der `ServerSocket` ein Objekt der Klasse `Socket` zurück, über das der Server dann mit dem Client kommunizieren kann.

Ein Problem an diesem System ist, dass der `ServerSocket` während seiner Wartezeit den weiteren Ablauf des Programmes blockiert. Es kann also nicht weiter bedient oder gar beendet werden. Die Lösung dafür ist jedoch denkbar simpel: Der `ServerSocket` wird während seiner Wartezeit auf einen anderen Thread, also einen Ausführungsstrang, ausgelagert. Dies erlaubt dem Programm, weiter auf Benutzereingaben zu antworten.

Um nun auch Daten senden zu können, muss eine Kette aus „Streams“ erstellt wer-

den. Streams sind Objekte, mit deren Hilfe Daten in ein Java-Programm eingelesen oder geschrieben werden können. Um Objekte senden und empfangen zu können, werden Object-Input-, beziehungsweise Object-Output-Streams verwendet. Dies sind Streams zur „Vorverarbeitung“, welche die Daten für einen „lowlevel-Stream“ vorbereiten. Der lowlevel-Stream kümmert sich dann um das schlussendliche Versenden der Daten.

Wenn man nun Objekte mithilfe der Sockets versenden will, so erstellt man einen Object-Stream, welchem man den lowlevel-Stream des entsprechenden Sockets als Argument mitgibt. Diesen erhält man von einer Funktion des Sockets. War der Vorgang erfolgreich können nun ganz bequem mittels Funktion des entsprechenden Streams Daten gesendet und empfangen werden.

#### **4.10.2 Netzwerkprogrammierung in JavaChess**

In JavaChess gibt es einen Host und einen Client. Diese unterscheiden sich nur beim Verbindungsvorgang. Der Host wartet auf eine eingehende Verbindung, der Client baut aktiv eine Verbindung auf. Nach dem Aufbauen einer Verbindung wird der Serversocket vom Host zu einem „normalen“ Socket umgewandelt. Ab diesem Zeitpunkt befinden sich die beiden Teilnehmer im Spielfluss und führen den selben Code aus. Das bedeutet, dass sowohl der Code für den Host als auch der Code für den Client, bis auf die Verbindungsvorgänge, absolut identisch sind.

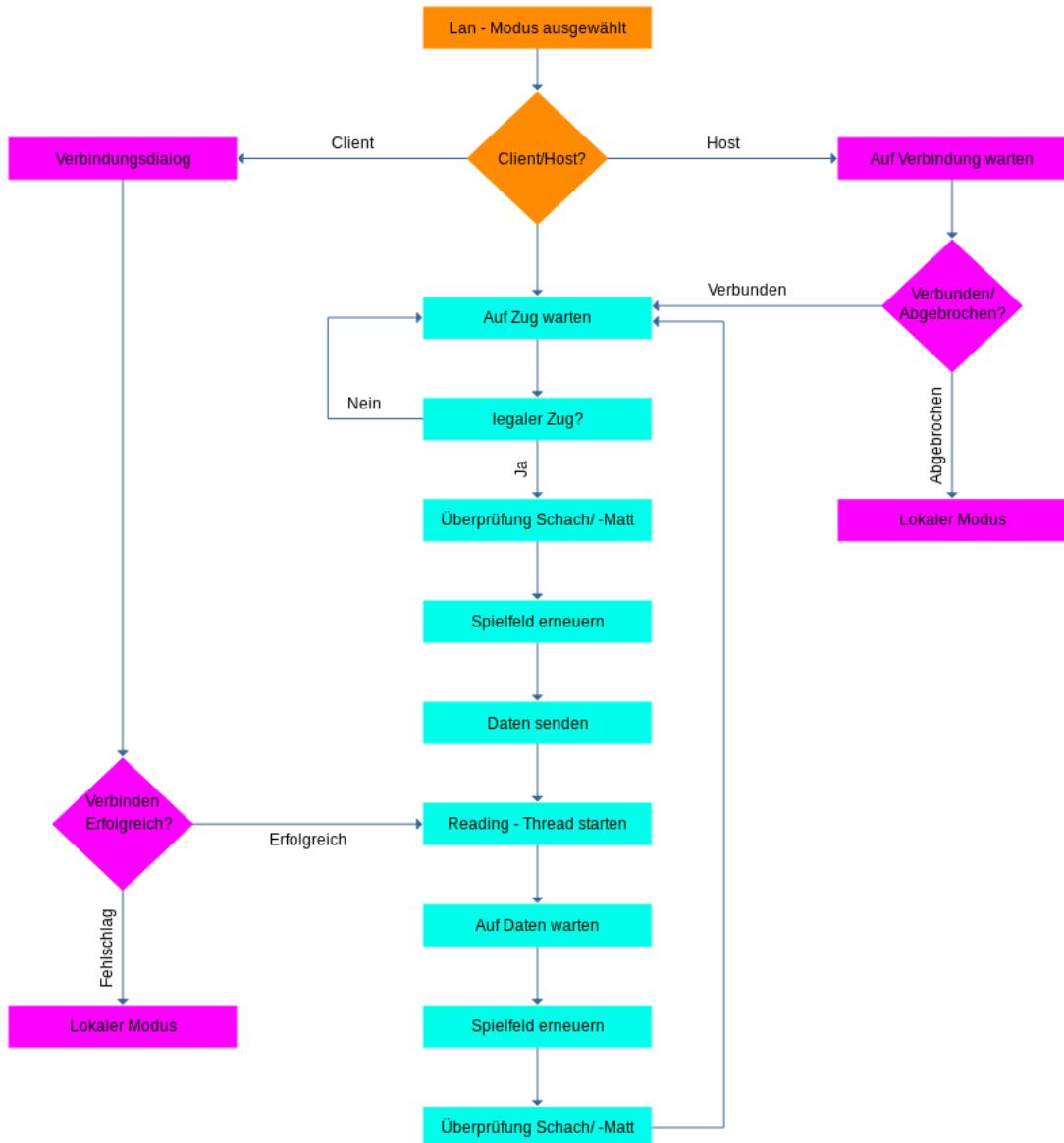


Abbildung 4: Spielzyklus im LAN-Modus

#### 4.10.3 Verbindungsvorgang des Clients

Um mit dem Client eine Verbindung aufzubauen, muss in der Menüleiste der entsprechende Punkt angewählt werden. Es öffnet sich ein Dialogfenster. Wird in diesem Dialogfenster der Punkt „Join“ angewählt, beginnt der Verbindungsvorgang.

Ein neues Dialogfenster wird angezeigt. Der Benutzer muss hier seine IP-Adresse eingeben. Ist die eingegebene Adresse korrekt, so wird der Socket der LAN-Klasse

mit dieser IP-Adresse und einem fest vorgegebenen Port initialisiert. Ist auch dies erfolgreich, so wird das Team des Clients auf Schwarz gesetzt, die Streams zur Kommunikation erstellt und es werden ein Reader Thread und ein Heartbeat Thread gestartet.

Folgender Pseudo-Code zeigt den Verbindungsverlauf:

---

```
if(join ausgewaehlt){  
  
    wechsleSpielmodus(Lan);  
    neuerDialog();  
    ...           //graphisches Design des Popups  
    Optional<String> ipResult = ipDialogue.showAndWait();  
    if (IP Adresse eingegeben){  
        try {  
            joinAdress = eingegebene Adresse();  
            SocketErstellen();  
            //Streams erstellen und Teams setzen  
            StreamsErstellen()  
            Verbindungsstatus(Verbunden);  
            Team(Schwarz);  
            //Zugverzegerung um einen Zug  
            Zugverzegerung(true);  
            SchachfelderKlickbar(false);  
            Thread rt = new Readinthread();  
            rt.start();  
            Thread th = new HeartbeatThreadThread(Heartbeat);  
            th.start();  
            //Menuepunkte entfernen und hinzufuegen
```

```

        menuFile.getItems().removeAll(Speichern und Laden);

        menuGame.getItems().removeAll(Spielmodi);

        menuGame.getItems().addAll(disconnect);

        Gui.getBoardGui().DrawGrid(Hintergrundmatrix);

        Gui.getBoardGui().redraw();

    }

} catch (UnknownHostException) {
    ...
    spielerBenachrichtigen();
    wechsleSpielmodus(Lan);

} catch (IOException e) {
    ...
    spielerBenachrichtigen();
    wechsleSpielmodus(Lan);
} catch (ClassNotFoundException e) {
    spielerBenachrichtigen();
    wechsleSpielmodus(Lan);
}

```

---

#### 4.10.4 Verbindvorgang des Hosts

Um ein Spiel zu hosten wird im Auswahldialog der entsprechende Punkt angewählt. Im Hintergrund wird ein neues Spielfeld erstellt, der Spielmodus gewechselt, und es wird der „Hosting Thread“ gestartet. Im Vordergrund wird eine Informationsfläche (vgl. 4.8.6) gezeichnet. In diesem hat der Host die Option, den Wartevorgang abzubrechen und wieder in den lokalen Modus zurückzukehren.

Der folgende Pseudo-Code zeigt, was bei dem Abbruchvorgang passiert:

```
if(Verbindvorgang abgebrochen){
```

```

SchachfelderKlickbar= true;

//wiederherstellen der Menues

menues ruecksetzen();

try{

    HeartbeatThreadStoppen();

    HostingThreadStoppen();}

} catch(Exception e){

    ...

    //Fehlerbehandlung

}

in lokalen Modus wechseln();

DrawGrid(_BGG);

}

}

```

---

Während die Schaltfläche für den Abbruch zu sehen ist, wird im Hintergrund der Hosting-Thread ausgeführt. Dieser erstellt den Serversocket, startet den Heartbeat-Thread ( vgl. 4.10.6) und wartet auf eine Verbindung.

Ist der Verbindungsvorgang erfolgreich, so erzeugt der Thread den Socket der LAN - Klasse, der zum Spielen notwendig ist. Er erzeugt die Streams um die Kommunikation zu ermöglichen und setzt den Verbindungsstatus auf „verbunden“. Anschließend muss noch die Schaltfläche zum Beenden des Wartevorganges verschwinden. Unter JavaFX kann die grafische Oberfläche allerdings nur von dem Hauptthread aus verändert werden.

Um dieses Problem zu umgehen, wird ein Objekt der Klasse Robot erstellt. Mit diesem ist es möglich, den Mauszeiger auf die Schaltfläche zu führen und einen Klick auszulösen. Dieser Klick wird im Hauptthread erkannt und das Feld wird neu gezeichnet.

Der folgende Pseudo-Code zeigt die Geschehnisse im Hosting-Thread:

```
try {

    HeaertbeatThread= new Thread(Heartbeat);
    TeamSetzen(Weiss);
    //warten auf eingehende Verbindung
    tempsock = VerbindungDesServersockets();
    bgg.getLan().Streams erstellen(); //erstellen der Streams
    bgg.getLan().setIsConnectet(true);

    //Ruecksetzen des Menues
    Point b = Position der Maus
    int x0rig, y0rig = Urspruengliche Mausposition

    try {
        Robot r = new Robot();
        r.mouseMove(Position des Mauszeigers);
        r.mouseclick(linke maustaste)
        //mauszeiger an die originale Position bewegen
        r.mouseMove(x0rig, y0rig);
    } catch (Exception e) {
        ...      //Problembehandlung
    }
} catch (Exception e) {
    ...      //Problembehandlung
}
```

---

#### 4.10.5 Spielfluss im LAN-Modus

Bis auf den Verbindungsvorgang unterscheiden sich die Mechaniken von Client und Host überhaupt nicht. Alle Variablen und Funktionen die notwendig sind, um den Spielfluss im LAN-Modus zu gewährleisten, sind in den Klassen „LAN.java“ und „BoardGui.java“ enthalten und werden sowohl vom Client als auch vom Host auf die exakt gleiche Weise verwendet. Der einzige große Unterschied ist, dass der Client zuerst mit dem Lesen, und der Host zuerst mit dem Schreiben von Daten beginnt. Der Spielzyklus, der sich ergibt, wird im Folgenden von Anfang an erklärt. Für ein besseres Verständnis kann das Flussdiagramm (Abb.: 4) zu Rate gezogen werden. Sowohl das Bewegen einer Figur, die Überprüfung, ob ein Zug erlaubt ist und die Schach/Schachmatt-Abfrage funktionieren exakt gleich wie im lokalen Spielmodus. Weiteres dazu ist im Kapitel 4.6 zu finden.

Wird ein lokaler Zug erkannt, so müssen die Daten an den zweiten Mitspieler versendet werden. Um möglichst wenige Daten zu versenden und eine einfache Implementierung zu ermöglichen wird, anstatt der ganzen Backgroundgrid - Klasse nur die repräsentative Zahlenmatrix (vgl. Abb.: 1) mit den aktuellen Figurenpositionen versendet. Dies geschieht über die, beim Verbinden der Spieler erstellte, Kette aus Streams. Direkt nach dem Versenden wird das Feld des Spielers neu geladen und dargestellt. Um den Rundenzähler aktuell zu halten, wird die Rundenzählervariable erhöht. Bevor das Programm in den Lesezustand geht, werden die Schachfelder noch unklickbar gemacht.

Um dem Spieler die Möglichkeit zu bieten, die grafische Oberfläche weiter zu benutzen und beispielsweise das Netzwerkspiel zu verlassen oder das Spiel zu beenden, wird der Lesevorgang in einen nebenläufigen Thread ausgelagert. Dies ist notwendig, da das Programm beim Lesevorgang stehen bleiben würde und nicht weiter liefe, bis der Lesevorgang abgeschlossen wäre.

Im Folgenden wird der Pseudo-Code angegeben, welcher den Schreibvorgang anschaulich darstellt:

---

```

if (Bewegung legal und Lan-Modus aktiv){

    Stream.schreiben(Hintergrundmatrix);
    zeichneSpielfeld(Hintergrundmatrix);
    erhoeheRundenzaehler();
    SpielfeldKlickbar(false);
    starteReadThread();

}

```

---

Der Lesevorgang selbst wird über eine Variable getriggert, auf die ein ChangeListener aufgesetzt wird. Der Code innerhalb eines Change-Listeners wird dann ausgeführt, wenn sich die zugehörige Variable ändert. Diese Variable wird im Reader-Thread verändert, sobald das Programm bereit ist zu empfangen. Der Code, der zum Empfangen benötigt wird, steht zwar in der BoardGui Klasse, wird aber aufgrund der Triggerung in dem zuvor erstellten Reader-Thread ausgeführt.

Beim Empfangsvorgang wird als erstes die Hintergrundmatrix neu gesetzt. Im Anschluss wird die Zuganzahl erhöht und das aktuell ziehende Team geändert. Danach wird das Feld mit der neuen Hintergrundmatrix neu gezeichnet. Zuletzt werden noch eine Schach- und Schachmattabfrage für beide Teams durchgeführt.

---

Im folgenden Pseudo-Code wird der Empfangsvorgang anschaulich dargestellt.

---

```

try {

    Hintergrundmatrix=leseStream.lesen();
    Zugnummer erhöhen

    if (Weisses Team am Zug){

        Teamwechsel(Schwarz);

```

```

        } else if (Schwarzes Team am Zug){

            Teamwechsel(Weiss);

        }

    } catch (IOException e) {

        ...

        //Problembehandlung

    }

BackGroundGrid.MatrixSetzen(Hintergrundmatrix);

SpielfeldKlickbar(true);

neuZeichnen();

Schachmattabfrage(Team Weiss);

Schachmattabfrage(Team Schwarz);

Schachabfrage(Team Weiss);

Schachabfrage(Team Schwarz);

```

---

#### 4.10.6 Das Heartbeat - System

Ein normales Trennen der Netzwerkverbindung ist einfach zu detektieren. Schwieriger wird es, wenn die Netzwerkverbindung abbricht oder eines der beiden System aufhört zu funktionieren. In diesen Fällen ist das Programm nicht mehr in der Lage, aktiv einen Befehl zum sauberen Trennen der Netzwerkverbindung mit dem Gegenüber zu versenden. Das kann dazu führen, dass ein Benutzer, der auf den Zug seines Gegners wartet, nichts von einem potentiellen Verlieren der Netzwerkverbindung oder einem Systemabsturz seines Gegenübers mitbekommt. Das führt dazu, dass der User keine Anzeige erhält, dass sein Gegenüber nicht verbunden ist, da sich das Programm im Wartezustand befindet und nur weiterläuft, sobald entsprechende Daten empfangen wurden.

Um dem Vorzubeugen, wurde ein „Heartbeat“-System implementiert. Die Idee die-

ses Systems ist simpel: In regelmäßigen zeitlichen Abständen, wie bei einem Herzschlag, werden kleine Datenpakete über das Netzwerk zum Gegenspieler versendet, weshalb auch von Heartbeat die Rede ist. Wenn die Pakete über eine längere Zeitspanne ausbleiben, wird die Netzwerkverbindung abgebrochen.

Um den Heartbeat zu realisieren, kann nicht der bestehende Socket verwendet werden, da dieser damit beschäftigt ist, die Spieldaten zu senden und zu empfangen. Das bedeutet, dass ein neuer Socket und ein neuer Serversocket für Client und Host erstellt werden müssen. Dieser Socket benötigt einen anderen Port, da die Computer, auf denen die Programme ausgeführt werden, sonst nicht unterscheiden könnten, welches Paket für welchen Thread gedacht ist.

Zusätzlich muss das ganze System auf einen zusätzlichen Thread ausgelagert werden, da es sonst die Abarbeitung anderer Teile des Programmes blockiert.

Der Thread für den Heartbeat wird jeweils zusammen mit den Verbindungsvorgängen gestartet. Auch beim Heartbeat wird zwischen Host und Client unterschieden. Wenn der Thread vom Client aus gestartet wurde, wird ein neuer Socket mit der gleichen IP-Adresse wie jener Socket, der für die Spieldaten zuständig ist, erstellt. Wenn der Thread vom Host aus gestartet wurde, wird ein Serversocket, der auf den neuen Port hört, erstellt. Dieser wartet dann auch direkt auf eingehende Verbindungen.

Ist ein Verbindungsvorgang erfolgreich, so erstellen beide Threads, also sowohl der vom Host, als auch der vom Client, ihre Input- und OutputStreams. Bei diesen handelt es sich um ObjectInput- und ObjectOutputStreams, die mit den lowlevel - Streams der jeweiligen Sockets verkettet werden.

Vor den ersten Sende- und Empfangsvorgängen wird auf jeden Socket noch ein Timer gesetzt, welcher angibt, wie lange der Thread wartet, bis er beim Ausbleiben der Pakete einen Trennungsvorgang einleitet. Dieser Timer wird nach jedem empfangenen Paket zurückgesetzt.

Im Anschluss werden die eigentlichen Sende- und Empfangsschritte eingeleitet. Damit der Host und der Client nicht gleichzeitig mit dem Schreiben beginnen, wird eine

einmalige Verzögerung beim Client eingebaut, die diesen mit dem Lesen beginnen lässt. Danach wechseln sich beide Parteien mit dem Lesen und Schreiben ab. Um das Netzwerk nicht mit „Heartbeat-Traffic“ zu fluten, wird der jeweilige Thread nach dem Senden für eine Sekunde deaktiviert. Sobald ein Disconnect von Heartbeat erkannt wurde, wird mittels einer Toggelvariable (also eine Variable mit Listener, ähnlich jener zum Empfangen der Spieldaten) ein Informationsfeld (vgl. 4.8.6) gezeichnet, welches den Spieler über die verlorene Verbindung informiert. Dieses Informationsfeld ist klickbar und bringt den Spieler zurück in den lokalen Spielmodus.

Im folgenden Pseudo-Code werden die Anweisungen im Heartbeat-Thread veranschaulicht.

---

```
try {  
    if(Client){  
  
        Socket = new Socket(Gleiche IP,neuer Port);  
        erstelleStreams(Socket);  
        Socket.TimeoutNach(15 Sekunden)  
    }  
  
    else if(Host){  
  
        Serversocket = new Serversocket(neuer Port);  
        Socket = Serversocket.warteAufVerbindung();  
        erstelleStreams(Socket);  
        Socket.TimeoutNach(15 Sekunden)  
    }  
  
    while(Endlos){
```

```
    if(Erster Lesevorgang des Clients){  
        Stream.lesen();  
    }  
  
    Stream.schreiben(Testdaten);  
    Thread.sleep(1000);  
    Stream.lesen(Testdaten);  
  
}  
  
}  
  
} catch (Exception e){  
    verbindungTrennen();  
    TriggervariableSetzen(); //triggert das Zeichnen der hinweisflaeche  
}  
}
```

---

## 4.11 Computer Modus (AI-Mode)

Im folgenden Kapitel wird erläutert, wie die AI des Schachspiels funktioniert. Dafür müssen zunächst folgende Punkte behandelt werden:

- Prinzipielle Möglichkeiten einer AI
- Verwendete Schach-AI-Funktion
- Der Min-Max-Algorithmus
- Code

### 4.11.1 Prinzipielle Möglichkeiten einer AI

Die Termini „AI“ (Artificial Intelligence), bzw. „KI“ (Künstliche Intelligenz) bedeuten das gleiche und werden anschließend abwechselnd verwendet.

Beide werden sehr oft verwendet, jedoch gibt es Unterschiede zwischen den verschiedenen Konzepten der AIs, die größer nicht sein könnten. So gibt es zum Beispiel künstliche Intelligenzen, die auf Machine-Learning-Algorithmen basieren und andere, denen ein Min-Max Prinzip zu Grunde liegt.

Das Min-Max Prinzip lässt sich aber nur bei Spielen mit perfekter Information anwenden wie Schach eines ist. Ein Spiel mit perfekter Information bedeutet, dass jeder Spieler alles weiß, so haben im Schach immer beide Spieler das gesamte Spielfeld im Blick.

Durch diese Art des Spiels können die theoretisch besten Züge ermittelt werden, um den besten möglichen Zug auszuführen. Dazu muss der Algorithmus alle möglichen Spielzüge analysieren, bewerten und vergleichen, um einen Zug zu tätigen. Der Nachteil von Min-Max ist, dass dies einen sehr hohen Rechenaufwand bedeutet. Der Vorteil von Min-Max ist, dass es in der Zeitspanne der Diplomarbeit möglich ist, eine KI zu entwickeln.

Die andere Möglichkeit ist ein Machine-Learning-Algorithmus. Dieser versucht, ein

biologisches Gehirn zu simulieren, indem er ein künstliches-neuronales-Netzwerk bildet. Diese Neuronen werden dahingehend trainiert, dass die AI aus gegebenen Datensätzen und/bzw. mit Hilfe einer Lernfunktion Rückschlüsse auf mögliche zukünftige Ereignisse schließt. Mathematisch gesehen basiert diese Art der KI auf der Wahrscheinlichkeitsrechnung.

Wichtig ist noch anzumerken, dass diese Art der KI selbstständig lernen kann, damit sie besser wird. Als Beispiel nehmen wir Schach: Die KI spielt über längere Zeit gegen sich selbst und muss somit, um sich zu verbessern, nicht unbedingt gegen andere Spieler, sowohl menschlich als auch maschinell, antreten.

Maschinelles Lernen ist ein sehr umfangreiches Thema, zu dem hier nur noch gesagt sei, dass maschinelles Lernen in letzter Zeit einige Durchbrüche erlebt hat. Um noch ein konkretes Beispiel zu nennen: Im Dezember 2017 gewann der von Google entwickelte Algorithmus „AlphaZero“ gegen die Chess-Engine „Stockfish 8“. AlphaZero basiert auf maschinellem Lernen und Stockfish 8 auf der Vorausberechnung aller möglichen Züge.[2]

#### 4.11.2 Verwendete Schach-AI-Funktion

Wie zuvor bereits beschrieben, verwendet die Java-Chess-AI eine Abwandlung des Min-Max-Algorithmus. Dieser wird „alphaBeta“ Algorithmus genannt.

Prinzipiell sucht der Algorithmus nach dem besten Spielzug. Um dies tun zu können, benötigt es einen Algorithmus, welcher alle möglichen Spielzüge bis zu einer gewissen Tiefe durchsucht und den besten Spielzug in Folge herausschreibt.

Hierfür muss geklärt werden, welcher Spieler gerade die Oberhand hat. Im entwickelten Algorithmus besteht die fundamentale „Board-Evaluation“ aus der materiellen Balance, sprich welcher Spieler hat mehr und bessere Figuren. Anschließend werden noch „Bauernformationen“, d.h. wenn die Bauern sich gegenseitig decken und sogenannte „Piece-Square-Tables“ in die Kalkulation mit eingerechnet. Piece-Square-Tables geben an, wo die Figuren statistisch gesehen am besten stehen wür-

den. Zum Beispiel sollten Türme eher in der Mitte, als am Rand stehen, da sie dadurch mehr Bewegungsfreiheit haben, was im Schachspiel eine der wichtigsten Strategien zum Sieg ist. Hierbei muss angemerkt werden, dass die Piece Square Tables von SchachmeisternInnen erstellt wurden, welche der Öffentlichkeit zugänglich gemacht worden sind.[3]

Das Durchsuchen der möglichen Spielzüge läuft folgendermaßen ab: Zunächst wird festgelegt bis zu welcher Tiefe (z.B.: 5) gesucht werden soll. Anschließend wird der erste mögliche Zug getätigt. Dann ruft sich die Methode selbst rekursiv auf, wobei die Tiefe erhöht und das Team gewechselt wird und tätigt wiederum den ersten möglichen Zug. Dies geschieht so lange, bis die gewünschte Tiefe (5) erreicht ist, bei welcher die „Board-Evaluation“ durchgeführt wird. Dieser Wert wird zwischengespeichert und der zuletzt getätigte Zug wird rückgängig gemacht.

Nun befindet sich der Algorithmus wieder in der Tiefe 4, in welcher der zweite mögliche Zug getätigt wird. Falls dieser Zug besser ist, als der vorherige, überschreibt dieser den zwischengespeicherten ersten Zug. Falls nicht, wird er von nun an nicht länger berücksichtigt.

Dies geschieht nun solange, bis alle relevanten Züge durchsucht wurden. Die nicht Relevanten werden in 4.11.4 behandelt.

Sobald der beste Zug ermittelt wurde, folgt die übliche Schachmatt-Abfrage und das Überschreiben des Schachfeldes mit den neuen Positionen. Falls die KI dadurch den eigenen König bedroht, wird ein anderer Zug genommen. Anschließend ist wie in 4.2.1 zu sehen, wieder der Spieler an der Reihe.

Im fertigen Schachspiel gibt es zwei verschiedene „Modi“ der KI:

- Normal AI
- Better AI

Die Einstellungen hierfür finden sich im Setup.

Der oben beschriebene Algorithmus ist die „Better AI“. Die „Normal AI“ unterscheidet sich in Details von der „Better AI“, so wird hier nicht nur jeder Zug gespeichert,

der besser ist als der zuletzt errechnete, sondern auch Züge die gleich gut sind wie der Letzte.

In weiterer Folge gibt es mehrere gleich „gute“ Züge, welche die KI nehmen kann. Von diesen Zügen wird durch eine Zufallszahl einer ausgewählt.

Die Schwierigkeit der „Better AI“ ist höher, da immer jene Züge die besten sind, die als letzte die vorherigen Züge überschreiben. Da durch die „Normal AI“ keine Züge überschrieben werden, können auch schlechtere Züge ausgewählt werden, welche für die KI aber gleich gut wirken. In 4.11.4 wird aber rein auf die „Better AI“ eingegangen, um das Verständnis dieser zu vertiefen. Der Code für die „Normal AI“ befindet sich in den Source Files, Package game, Klasse AI und AILogic.

Eines der größten Probleme des Algorithmus, technisch gesehen unterscheidet sich der Basisalgorithmus zwischen „Normal“ und „Better AI“ nicht, ist die Performance. Dazu ein simples Gedankenexperiment: Beim ersten Spielzug sind 20 mögliche Züge des weißen Spielers möglich und eben soviele beim darauffolgenden Zug des schwarzen Spielers. Daraus resultieren aus den beiden Spielzügen  $20 \cdot 20 = 400$  verschiedene Stellungen.

Gemittelt gibt es im Schach 28 mögliche Züge pro Spielzug. Nach fünf Zügen ergeben sich daraus schon 3.200.000 mögliche Stellungen, nach sechs wären es 64.000.000. Um zu einer Entscheidung zu kommen, muss der Computer alle diese Züge analysieren und bewerten. Dies benötigt Rechenleistung, weshalb ein solcher Schachalgorithmus sehr abhängig von der verwendeten Hardware ist.

In der theoretischen Informatik gibt es zu obigen Beispiel die O()-Notation (Siehe [10] - Seite 178), welche die voraussichtliche Laufzeit eines Algorithmus angibt. Diese kann je nach verwendetem Algorithmus konstant, logarithmisch  $\log(n)$ , linear, logarithmisch  $(n \cdot \log(n))$ , quadratisch, kubisch oder exponentiell sein, wobei exponentielle Algorithmen die rechenaufwendigsten Algorithmen sind.

Min-Max ist ein exponentieller Logarithmus. Die Anzahl der verwendeten Rechenschritte hängt, wie oben beschrieben, von der zu berechnenden Zugtiefe ab. Mathematisch können die zu berechnenden Fälle folgendermaßen ermittelt werden, wobei

$n$  die Zugtiefen ist und  $Schritte(n)$  die möglichen Züge:

$$Schritte(n) = 28^n$$

Im weiteren Sinne ist dies der Grund, warum auf Min-Max basierende künstliche Intelligenzen gegen künstliche neuronale Netzwerke verlieren (siehe 4.11.1). Die Min-Max-Algorithmen können nicht ausreichend optimiert werden, bzw. moderne Hardware besitzt einfach nicht genügend Rechenleistung, um in diesem Wettlauf mithalten zu können.

#### 4.11.3 Der Min-Max-Algorithmus

In den Kapiteln 4.11.1 und 4.11.2 wurde erwähnt, dass die JavaChess AI nach dem Min-Max Prinzip funktioniert, im Folgenden wird darauf näher eingegangen.

Min-Max ist nichts anderes, als das Erhalten des bestmöglichen Ergebnisses für einen Spieler, wenn die Züge des Gegenspielers mit eingerechnet werden. In diesem Beispiel haben wir zwei Teams, weiß und schwarz. Diese sind durch die Blasen erkenntlich. Weiteres gilt für ein kompetitives Spiel wie Schach, dass das weiße Team immer sein bestes Ergebnis herausholen möchte und das schwarze auch seines. Die Zahlen repräsentieren die Günstigkeit der Stellung für das weiße Team, wobei 10 die bestmögliche Stellung ist und 1 die schlechteste. Für das schwarze Team ist dies umgekehrt, für sie ist 10 das schlechteste Ergebnis und 1 das Beste. Somit wird das schwarze Team immer das kleinere Ergebnis nehmen, also das MINIMUM herausholen (=Min) und die Weißen immer das höchstmögliche, also das MAXIMUM (=Max). Daher kommt auch der Name, Min-Max.

Im oben genannten Beispiel kann Schwarz nun einmal wählen zwischen den Zahlen 5 und 3, 10 und 2 und 7 und 8. Da die Schwarzen immer die niedrigere Zahl nehmen, kann die niedrigere Zahl in die schwarzen Bubbles geschrieben werden (3, 2 & 7).

Somit muss sich der weiße Spieler nur noch zwischen drei Zahlen entscheiden, bei denen er die höchste nimmt, also 7. Dies ist das bestmögliche Ergebnis für das weiße Team.

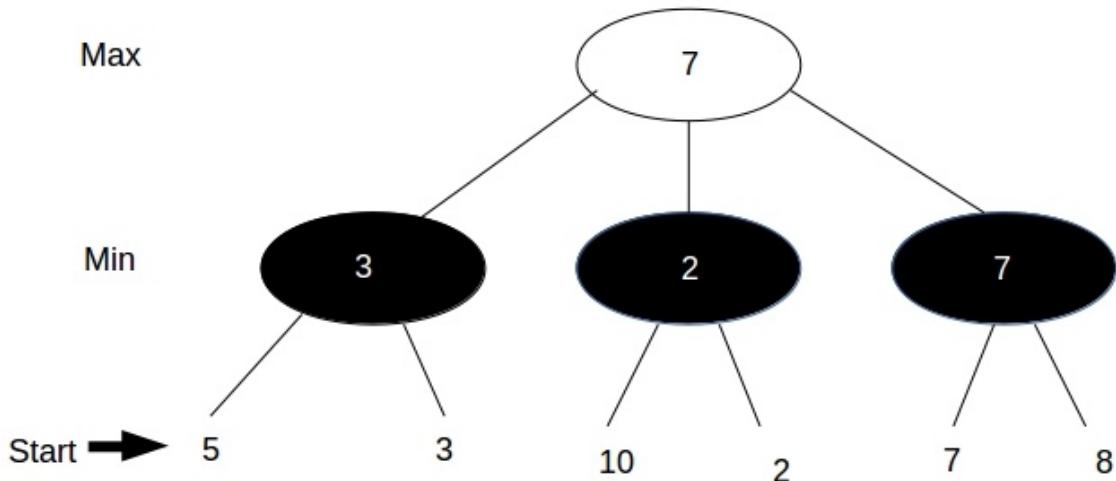


Abbildung 5: Min-Max

#### 4.11.4 Code

Die theoretischen Grundlagen zum Verständnis des Algorithmus sollten nun geklärt sein.

Die KI besteht aus zwei Klassen, AI und AILogic. AI wird als neuer Thread ausgeführt.

Der nachfolgende Code stellt die AI als Pseudo-Code-Dokumentation dar.

Die Klasse AI bekommt eine ArrayList der besten Züge (von AILogic) und führt den letzten Eintrag der Liste aus. Falls sich herausstellen sollte, dass die KI dadurch den eigenen König bedroht, nimmt sie den vorletzten Zug. Wenn dieser Zug wiederum den König in eine bedrohte Situation bringt, wird der drittletzte Eintrag ausgeführt. Dies geht solange weiter, bis alle Einträge der ArrayList aufgebraucht sind. Siehe Pseudo-Code:

---

```

public class AI extends Thread{
    public void run(){
        AILogic AIL = initialisiere AILogic()

```

AIL.alphaBeta() - initialisiert die AI-Berechnung in welcher der bestmoegliche Zug festgestellt wird

Move = bekomme alle moeglichen Zuege der AI

**for**(alle berechneten Zuege - beginnend beim letzten){

...

Der Zug wird getaetigt

...

**if**(Wenn der AI-Koenig nicht im Schach steht){

**break**;

}

...

Der Zug wird rueckgaengig gemacht

...

}

Schachmatt ueberpruefen

}

}

---

**Klasse AI-Logic:** Die Klasse AI-Logic führt die KI-Berechnung durch. Eingeteilt wird diese in die Methoden alphaBeta, aplphaBetaHelper, boardEvaluation und die verschiedenen SquareRootTables.

AlphaBeta bekommt die Informationen bis zu welcher Tiefe eine Suche stattfinden soll und für welches Team. Dies gibt AlphaBeta der Methode AlphaBetaHelper weiter.

Da der AlphaBeta-Algorithmus durch ein sogenanntes „Iterative-Deepening“ modifiziert wurde, wird die weitergegebene Tiefe schrittweise erhöht.

AlphaBetaHelper gibt die Günstigkeit eines Zuges als „Float“ zurück. Dies wird über die Methode boardEvaluation ermittelt. Falls die tiefste Ebene erreicht wurde, wird dies sofort zurückgegeben und der Rest des Codes nicht weiter ausgeführt.

Andernfalls werden alle Figuren des Teams ausgewählt und nacheinander die Spielzüge simuliert. Sobald die erste Figur des Teams simuliert wurde, ruft sich der Algorithmus rekursiv mit erhöhter Tiefe und geändertem Team auf, bis die maximale Tiefe erreicht wurde (siehe 4.11.3).

Falls ein Spielzug in der geringsten Tiefe besonders günstig ausfällt, wird dieser in eine ArrayList geschrieben, basierend auf dieser ArrayList wird eine Figur in der AI-Klasse bewegt.

---

```
public class AILogic{  
  
    int MaximaleTiefe  
  
    public float alphaBeta(tiefe, Hintergrundmatrix, Team){  
        MaximaleTiefe = tiefe  
  
        //Diese beiden Werte werden als Worst- und Best-Case Szenario  
        //verwendet.  
        alpha = 10000  
        beta = -10000  
        //Fuer iterative deepening
```

```
for(i=0;i<MaximaleTiefe,i++){  
    //Bekommt den Wert des besten Zuges  
    beta = alphaBetaHelper(starte bei Tiefe 0, Hintergrundmatrix,  
                           alpha,beta)  
}  
  
}  
  
//der AlphaBeta Algorithmus - zur Zugevaluation  
public float alphaBetaHelper(tiefe, Hintergrundmatrix, Team, alpha,  
                             beta){  
    Sum = boardEvaluation(Hintergrundmatrix, Team)  
  
    if(Sum bedeutet das feindlicher Koenig geschlagen wird){  
        return 20000  
    }  
  
    if(tiefe >= MaximaleTiefe){  
        return Sum  
    }  
  
    for(X und Y Positionen des Spielfeldes){  
        if(Spielfigur an Position X und Y){  
            Zuege = AlleMoeglichenZuegeDerSpielfigur  
            for(alle Moeglichen Zuege der Figur){  
  
                MovePos A = Zug der Figur  
                ...  
                Den Zug auf das Spielfeld uebertragen  
                Spezialzuege werden hier speziell behandelt  
                ...  
            }  
        }  
    }  
}
```

```

        Sum1 = -alphaBetaHelper(tiefe+1, BackgroundGrid,
                                Teamwechsel, -alpha, -beta)

        ...

        Den Zug r ueckgaengig machen
        Spezialzuege werden hier speziell behandelt
        ...

        //groesser fuer den „Better-AI“ Mode, groesser gleich fuer
        den normalen Modus (hier aus Uebersichtsgruenden nicht
        angefuehrt)

        if(Better-AI && Sum1 > beta){

            beta = Sum1

            if(Sum1 >= alpha){

                return alpha
            }

            if(tiefe == 0){

                ZuListeGuterZuegeHinzufuegen(A)
            }
        }

        }

        }

        return beta
    }

}

public float boardEvaluation(Hintergrundmatrix, Team){

    //Fuer die Material Balance

    for(X und Y Positionen des Spielfeldes){

        if(Bauer weisses Team){

            100 Punkte zum weissen Team dazu zaehlen

            Punkte entsprechend der Bauerntabelle hinzuzzaehlen
    
```

```
Punkte nach Bauerntabelle hinzuziehen  
} else if(Turm weisses Team){  
    500 Punkte zum weissen Team dazu zaehlen  
    Punkte entsprechend der Turmtabelle hinzuziehen  
} else if(Springer weisses Team){  
    325 Punkte zum weissen Team dazu zaehlen  
    Punkte entsprechend der Springertabelle hinzuziehen  
} else if(Lauefer weisses Team){  
    300 Punkte zum weissen Team dazu zaehlen  
    Punkte entsprechend der Laufertabelle hinzuziehen  
} else if(Dame weisses Team){  
    900 Punkte zum weissen Team dazu zaehlen  
    Punkte entsprechend der Damentabelle hinzuziehen  
} else if(Koenig weisses Team){  
    10000 Punkte zum weissen Team dazu zaehlen  
    Punkte entsprechend der Koenigstabelle hinzuziehen  
}  
  
if(Bauer schwarzes Team){  
    100 Punkte zum schwarzen Team dazu zaehlen  
    Punkte entsprechend der Bauerntabelle hinzuziehen  
    Punkte nach Bauerntabelle hinzuziehen  
} else if(Turm schwarzes Team){  
    500 Punkte zum schwarzen Team dazu zaehlen  
    Punkte entsprechend der Turmtabelle hinzuziehen  
} else if(Springer schwarzes Team){  
    325 Punkte zum schwarzen Team dazu zaehlen  
    Punkte entsprechend der Springertabelle hinzuziehen  
} else if(Lauefer schwarzes Team){  
    300 Punkte zum schwarzen Team dazu zaehlen  
    Punkte entsprechend der Laufertabelle hinzuziehen
```

```
    } else if(Dame schwarzes Team){  
        900 Punkte zum schwarzen Team dazu zaehlen  
        Punkte entsprechend der Damentabelle hinzuzzaehlen  
    } else if(Koenig schwarzes Team){  
        10000 Punkte zum schwarzen Team dazu zaehlen  
        Punkte entsprechend der Koenigstabelle hinzuzzaehlen  
    }  
}
```

Hier folgen die Tabellen fuer die Spielfiguren.

Diese geben an, wo sich die Figuren am liebsten aufhalten.

}

---

## 5 CHESSPI

ChessPI ist die Implementierung von JavaChess auf dem Raspberry PI.

Wir haben uns die Aufgabe gemacht, das Schachprogramm auf einen Mikro-Computer, wie den RaspberryPI 3b, zu portieren.

Die Benutzereingabe soll mittels eines Touchscreens erfolgen, weshalb am Raspber-ryPI ein Touchscreen angeschlossen wird. Weiteres soll für den ChessPI ein Gehäu- se designt und eine Akkusteuering entworfen werden.

### 5.1 RaspberryPI

Der Raspberry PI ist ein vollwertiger Computer, welcher mit einem Linux/GNU OS läuft. Am häufigsten wird die Distribution Raspbian verwendet.

Von uns wurde der RaspberryPI 3 Modell B verwendet. Dieser verfügt über ei- ne Quad-Core-1.2 GHz Broadcom-BCM2837-CPU, 1 Giga Byte Random Access Memory.[4] Dies ist eine deutliche Steigerung gegenüber den vorherigen Modellen, womit das Problem der notwendigen Leistung gelöst wird.

### 5.2 Touchscreen

Als Touchscreen wird das offizielle 7" Touchscreen Display verwendet. Dies hat eine Auflösung von 800x480 Pixel. [5]

Zusätzlich zum Display gibt es auch eine Adapterplatine, mit welcher der Touchs- creen versorgt wird.

Halterungen für die Adapterplatine und den Raspberry PI gibt es auf der Rückseite des Touchscreens.

### 5.3 Implementierung von JavaChess

Alle benötigten Schritte beziehen sich lediglich auf die Software- und Stromversorgungs-Implementation, nicht auf die Powerbank- und Gehäuse-Implementation.

#### Vorbereitung:

Es wird ein RaspberryPI 3, eine Mikro SD-Karte mit Raspbian, eine Stromversorgung bzw. eine Powerbank mit einem maximalen Strom von 2,5 A, der 7"Touchscreen, die Adapterplatine, eine Internetverbindung, idealerweise über ein LAN-Kabel und eine USB-Tastatur benötigt.

1. Die SD-Karte mit Raspbian wird in den Raspberry PI gesteckt.
2. Die Adapterplatine und der Raspberry PI wird auf den Touchscreen geschraubt.
3. Die Stromversorgung für das Display (VCC & GND Pin, rotes und blaues Kabel, siehe [5]) wird sichergestellt. Das Flachbandkabel / Datenkabel wird zwischen Raspberry PI und Adapterplatine angebracht.
4. Der Raspberry PI wird an die Stromversorgung angeschlossen, dadurch sollte dieser nun booten und das Display automatisch erkennen.
5. Sobald Raspbian gebootet hat, wird das LAN-Kabel angeschlossen.
6. Nun sollten folgende Befehle in der BASH ausgeführt werden:
  - (a) sudo apt-get update
  - (b) sudo apt-get upgrade
  - (c) sudo apt-get install oracle-java8-jdk
  - (d) reboot
7. Nach der Java Installation muss noch JavaFX „dazuinstalliert“ werden, da dies nicht in der Java-Embedded-JDK Serie enthalten ist.

8. Es muss OpenJFX gedownloaded werden. URL: <https://chriswhocodes.com/>
9. Hier ist die Version für den Raspberry PI zu downloaden (ARMv6)
10. Die gedownloadete OpenJFX-Zip Datei muss im Installationsverzeichnis von Java-JDK8 entpackt werden.
11. In der Commandline wird nun folgender Befehl ausgeführt: unzip openjfx-sdk-overlay-linux-armhf.zip -d /<installations-verzeichnis-von-Java  
(z.B.: /home/pi/jdk1.8.0\_92)>
12. Das aktuelle Schachspiel wird nun gedownloaded. Achtung: Es muss die Chess-PI Version verwendet werden, siehe 5.6.
13. Das Schachspiel wird in ein Verzeichnis nach Wahl abgelegt.
14. Nun kann das Schachspiel gestartet werden, dazu muss der Touchscreen aber noch konfiguriert werden, da sonst ein „interessanter“ Offset geschieht. Siehe 5.5, bzw. in der Auflistung Punkt 17.
15. Dazu muss zuerst der Touchscreen identifiziert werden: cat /sys/class/input/event1/uevent
16. Bei dem Versuchs-Raspberry PI war dieses Input Device: 0/0/0/0
17. Um das Schachspiel bequem zu öffnen, wird empfohlen ein BASH-Skript zu erstellen. Bei diesem Startvorgang muss noch angegeben werden, an welcher Position OpenJFX die Bildschirmeingaben verwendet werden sollen. Dazu wurden die besten Werte empirisch ermittelt:
18. 

```
#!/bin/bash
sudo java -Dmonocle.input.0/0/0/0.minX=0 -Dmonocle.input.0/0/0/0.minY=0
-Dmonocle.input.0/0/0/0 maxX=800 -Dmonocle.input.0/0/0/0 maxY=500
-jar chess.jar
```
19. Das Skript muss noch als ausführbar markiert werden: sudo chmod 777 <skript>

20. Nun das Skript öffnen: sudo ./<skript>
21. Das Schachspiel sollte sich nun öffnen.
22. Damit sich das Spiel direkt nach dem Bootvorgang öffnet, muss ein Skript in dem Ordner /etc/init.d/ erstellt werden. Dieses muss das bereits erstellte Skript ausführen.
23. `#!/bin/bash`  
`/bin/bash <Pfad zum Skript>/<Skript.sh>`

## 5.4 Wieso reicht die offizielle JDK nicht aus?

Oracle (Entwickler von Java) hat 2015 die Unterstützung für JavaFX auf Advanced RISC Machine (ARM) Plattformen gestrichen. Die OpenJDK besitzt, je nach Version, eine unvollständige bzw. ebenfalls keine JavaFX Unterstützung für ARM.

Dies hat zur Folge, dass JavaChess auf einem RaspberryPI eigentlich gar nicht funktionieren sollte.

Aufgrund einiger Recherchen, konnte das Spiel dennoch lauffähig gemacht werden, da zusätzlich zu der offiziellen JavaJDK von Oracle, das OpenJFX Paket installiert wurde.

## 5.5 Konfiguration des Touchscreens

Der Touchscreen muss konfiguriert werden, da OpenJFX nicht weiß, wie groß das Display ist. Dies muss konfiguriert werden, indem beim Starten der .jar diverse Startvariablen angegeben werden. Dies wird in Punkt 5.3, Auflistungspunkt 17, behandelt.

## 5.6 Verwendete JavaChess Version

Die ChessPI-Version unterscheidet sich leicht von der JavaChess-Version. Diese beinhaltet keine Einstellungsmöglichkeiten und weiters kein Speichern/Laden des Spiels.

Die KI befindet sich standardmäßig im „Better-AI“-Modus auf Schwierigkeitsstufe drei. Bei höheren Schwierigkeitsgraden wäre die erforderliche Performance nicht gegeben.

Die ChessPI-Version befindet sich aus organisatorischen Gründen nicht auf GitHub. Für die ChessPI-Version bitte Alexander Beiser anschreiben (E-Mail: alex@itbeiser.at, GitHub: alexl4123).

## 6 AKKUSTEUERUNG

Ziel ist es, eine Akkusteuering zu entwerfen. Diese soll es ermöglichen, den ChessPI auch unterwegs verwenden zu können.

Die mobile Spieldauer soll größer als eine Stunde sein.

Für eine Akkusteuering gibt es prinzipiell mehrere Möglichkeiten, angefangen bei der Powerbank, über bestimmte ICs bis hin zu einer selbst entwickelten Akkusteuering. Als Ziel ist eine selbst entwickelte Akkusteuering definiert.

### 6.1 Kenngrößen des benötigten Akkumulators

Um einen Akku auswählen zu können, muss zuerst einmal definiert werden, was gebraucht wird.

- Minimale Betriebsdauer  $t_{min} = 1h$
- Stromaufnahme RaspberryPI-Max:  $I = 2,5A$  [6] - zu beachten ist hierbei, dass die Stromaufnahme bei angeschlossenem Bildschirm ca. zwei Ampere beträgt, falls noch weitere Peripheriegeräte angeschlossen werden, erhöht sich die Stromaufnahme entsprechend.
- Versorgungsspannung RaspberryPI:  $U_V = 5V$

### 6.2 Wählen des Akkumulators

Akkumulatoren-Typen gibt es viele, für den Einsatz als RaspberryPI-Versorgung schieden alle aus, bis auf folgende Akkumulatortypen: Lithium-Ionen (Li-Ion), Nickel-Cadmium (NiCd), Lithium-Polymer (LiPo) und Lithium-Eisenphosphat ( $LiFePO_4$ ). Jeder dieser Akkutypen bietet verschiedene Vorteile, so bietet LiPo relativ hohe Entlasteströme und Li-Ion eine ziemlich hohe Energiedichte [7].

Für den Einsatz als Versorgung für den RaspberryPI sollte ein Akku möglichst an die 5V Versorgungsspannung herankommen. Dies in Betracht ziehend, scheiden NiCd (1,2V) und *LiFePO<sub>4</sub>* (3,3V) ebenfalls aus.

So bleiben entweder der LiPo (3,7V) oder der Li-Ion (ca. 3,6V) Akkumulator übrig. Von beiden wurden gleichwertige Akkus verglichen, aufgrund der niedrigeren Kosten eines *Li – Ion*<sub>[8]</sub> Akkumulators gegenüber eines *LiPo*<sub>[9]</sub> Akkumulators, ist die Entscheidung für den Li-Ion Akku gefallen.

### 6.3 Der Li-Ion Akku

Der Lithium-Ionen Akkumulator wird durch das Konstantspannungs-Ladeverfahren geladen. Hierbei wird eine Spannungsquelle mit konstanter Spannung an den Akku angeschlossen. Eingestellt wird die Spannung auf die Standardladespannung des Akkus. Bei 3,7V durchschnittlicher Spannung ist die Ladespannung typ. 4,2 Volt. Die minimal benötigte Kapazität berechnet sich wie folgt, wenn eine Stromaufnahmen von 2,5 Ampere angenommen wird.:

$$Q_{min} = t * I = 1h * 2,5A = 2,5Ah = 2500mAh$$

Bei der Verkäuferseite Reichelt wurde ein Akku mit diesen Vorgaben gefunden: [8] Dieser Akku hat eine maximale Kapazität von 2600 mAh, einen maximalen Entladestrom von 2,5 A und eine Ladespannung von 4,2V.

Das Datenblatt zu allen technischen Eigenschaften befindet sich auf der Seite von Reichelt [8] bzw. im Anhang (Siehe Abbildung 16 auf Seite XV).

### 6.4 Idee der ersten Akkusteuerungsschaltung

Die ursprüngliche Idee war es, die Akkus über eine Micro-USB Buchse zu laden und damit den Raspberry PI zu betreiben. Da USB eine Spannung von 5V aufweist, muss diese zum Laden der Akkus auf 4,2 V reduziert werden. Für den Betrieb des Raspberry PIs muss diese Spannung wieder auf 5V erhöht werden.

Geladen werden die Akkus parallel und entladen seriell, so muss eine Schaltung entwickelt werden, die eine Umschaltung von seriell auf parallel und umgekehrt, ermöglicht.

Weiteres muss die Schaltung für den sicheren Betrieb der Akkus sorgen, so muss diese bei Unter- oder Überspannung abschalten und falls ein Kurzschluss auftritt Schlimmeres verhindern.

Eine Akkuladestandanzeige wird mittels zwei LEDs realisiert, die eine schaltet bei 1/3 VCC und die andere bei 2/3 VCC.

#### 6.4.1 Laden der Akkus

Durch die Eingangsschaltung der Akkus wird die Versorgungsspannung auf unter 4,2 V gesenkt. Dies geschieht durch zwei in Serie liegende Dioden, eine Si-Diode mit einer Durchschaltspannung von ca. 0,7V und eine Shottky-Diode mit einer Durchschaltspannung von ca. 0,2V.

Der Ladestrom darf maximal 2,5A betragen, sollte aber geringer sein, um den Akku nicht zu beschädigen. Als Begrenzung kommt ein Leistungsvorwiderstand zum Einsatz. Dieser hat  $4\Omega$  und begrenzt den Strom auf 1,25 Ampere.

Da die Akkus parallel geladen werden, muss dies für jeden Akku einzeln aufgebaut werden.

In Punkt 6.4.5 befindet sich die Schaltung des ersten Versuches. Folgende Bauteile dieser Schaltung werden für das Laden verwendet:

$D_1, R_{sen1}, R_{sen2}, D_{s8}, D_{s2}, S_{I1}, S_{I2}, Akku1, Akku2$

#### 6.4.2 Entladen der Akkus

Das Entladen der Akkus erfolgt durch die Seriellschaltung der Akkus (siehe Seriell-Parallel-Schaltung). Seriell geschalten addiert sich die Spannung der beiden Akkus „normal“ auf 7.4 V, maximal auf 8.4V. Diese Spannung wird mittels Step-Down-Converter (LM2596) auf 5 V herunter geregelt. Dies kann als Versorgung für den

Raspberry PI verwendet werden.

Die Beschaltung erfolgt gemäß des Datenblattes (Siehe Abbildung 13 auf Seite XII). Die Berechnung der beiden Widerstände erfolgt ebenfalls gemäß Datenblatt.

$$R_2 = R_1 \cdot \left( \frac{V_{out}}{V_{ref}} - 1 \right)$$

$$V_{ref} = 1.25V - R_1 = 1k\Omega - V_{out} = 5V$$

$$R_2 = 1000 * (4 - 1) = 3k\Omega$$

Zum Ein- und Ausschalten des Raspberry PIs wird noch ein Schalter verbaut.

In Punkt 6.4.5 ist die Schaltung des ersten Versuches ersichtlich. Folgende Bauteile dieser Schaltung werden für das Entladen verwendet:

*LM2596, C<sub>1</sub>, D<sub>s1</sub>, L<sub>1</sub>, R<sub>1</sub>, R<sub>2</sub>, C<sub>2</sub>, Schalter, Akku1, Akku2*

#### 6.4.3 Seriell-Parallel-Schaltung

Die Seriell-Parallel-Umschaltung ermöglicht das Wechseln zwischen dem parallelen Laden und dem seriellen Entladen. Es wird erkannt, ob eine Versorgung angeschlossen ist. Wenn diese angeschlossen ist, schalten die Transistoren so, dass der ohmsche Widerstand zwischen den beiden Akkus groß ist, um einen Einfluss auf die Schaltung zu haben.

Weiteres wird das Potential des negativen Anschlusses des zweiten Akkus auf Ground gesetzt, um das Laden zu ermöglichen. Die Versorgung wird direkt vor dem Schalter des Ausganges angeschlossen, womit der Akku überbrückt wird, damit er sich nicht gleichzeitig entladen und laden muss.

Sobald das Micro-USB-Kabel ausgesteckt wurde, wird VCC mit GND leitend verbunden (Micro USB nicht vorhanden - also Potential undefiniert). Die Verbindung zwischen dem Pluspol von Akku1 und dem Minuspol des zweiten Akkus wird wiederhergestellt. Dabei wird die Verbindung zwischen dem Minuspol des zweiten Ak-

kus und GND aufgehoben, um einen Kurzschluss zu verhindern.

Die Verbindung zwischen dem zweiten Akku und dem LM2596 wird aufgetrennt, um ein Entladen des Akkus zu verhindern.

In Punkt 6.4.5 ist die Schaltung des ersten Versuches ersichtlich. Folgende Bauteile dieser Schaltung werden für das Entladen verwendet:

$T_1, R_{T1}, T_3, R_{T3}, T_6, R_{T6}, T_{m3}, T_{m4}, Akku1, Akku2$

#### 6.4.4 Sicherheit

Unter Sicherheit werden alle Sicherheitsmaßnahmen verstanden, die eine Fehlfunktion des Akkus verhindern.

Dafür gibt es einen Spannungsteiler, welcher alle kritischen Spannungen für Komperatoren zur Verfügung stellt.

Die Spannungen sind:

- Minimale Abschaltspannung:  $U_{ref4} = 3V$
- Akku zu 33% geladen:  $U_{ref3} = 3,3V$
- Akku zu 66% geladen:  $U_{ref2} = 3,7V$
- Maximale Ladespannung erreicht:  $U_{ref1} = 4,1V$

Wird die minimale Spannung unterschritten, wird die Verbindung zwischen dem Ausgang und den Akkus gekappt.

Wird die maximale Ladespannung erreicht, wird die Verbindung zwischen VCC und den Akkus getrennt. Diese beiden Schutzmechanismen werden für jeden Akku verbaut.

In der Schaltung sind drei Vier-Ampere-Sicherungen verbaut, diese sorgen für die nötige Kurzschlusssicherheit.

Die Spannungen  $U_{ref3} = 3,3V$  und  $U_{ref2} = 3,7V$  geben den Ladestand des ersten Akkus an. Zwei Komperatoren vergleichen fortlaufend die Referenzspannungen mit der Akkuspannung und geben mit Hilfe von zwei LEDs den Akkustand wieder. Da

beide Akkus immer den gleichen Ladestand haben sollten, braucht man keine zusätzliche Schaltung für den zweiten Akku. In Punkt 6.4.5 ist die Schaltung des ersten Versuches ersichtlich. Folgende Bauteile dieser Schaltung werden für das Entladen verwendet:

$$OPV_1, OPV_2, T_2, T_4, T_5, LED_1, LED_2, R_{LED1}, R_{LED2}, R_{T2}, R_{T4}, T_{T5}$$

### 6.4.5 Schaltung

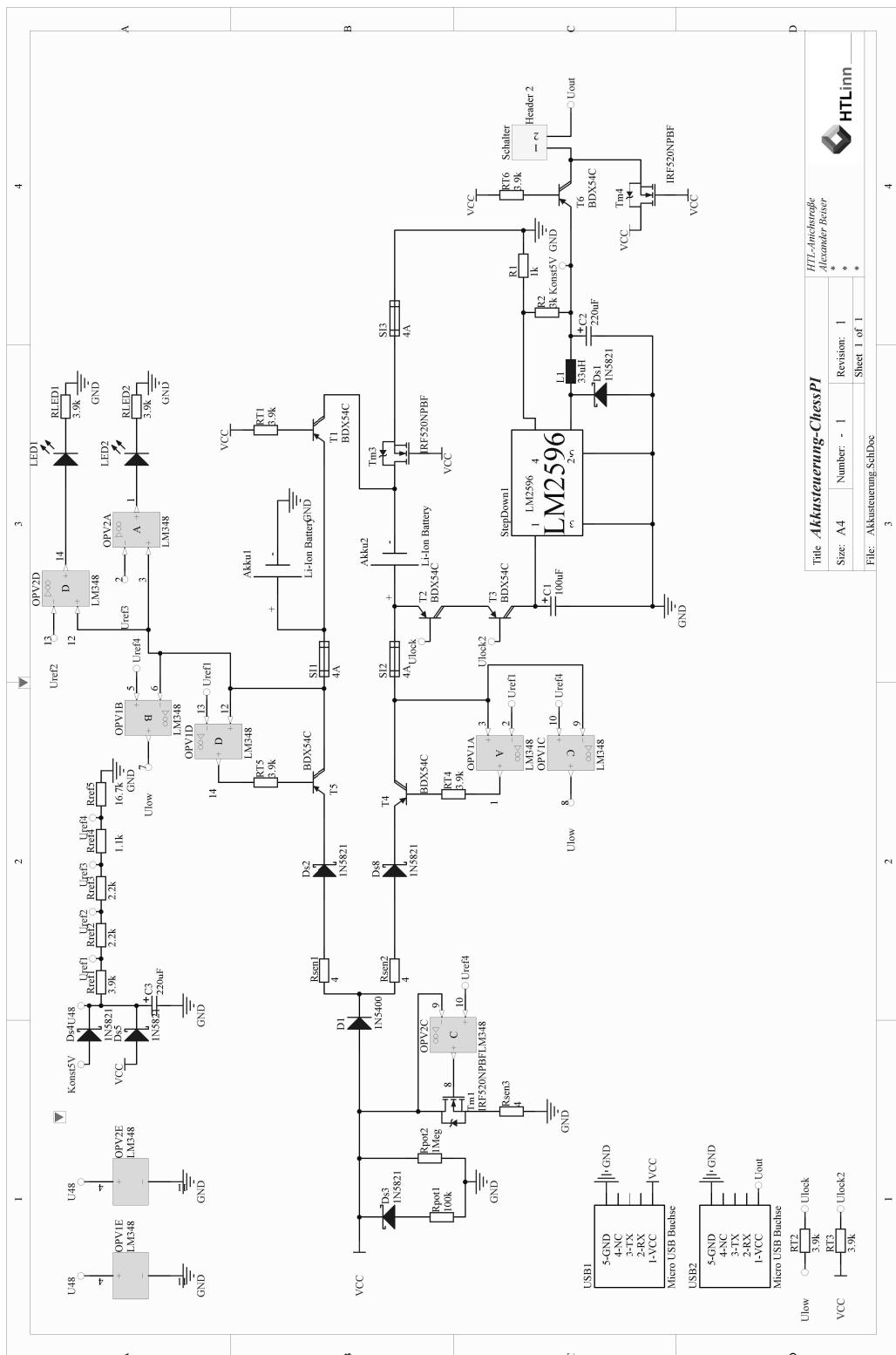


Abbildung 6: Die 1. Schaltung

#### 6.4.6 Messung

Alle Messungen werden auf mehreren Steckbrettern aufgebaut.

Die Messung wird in mehrere Phasen gegliedert:

1. Messung der Ausgangsschaltung
2. Messung der Eingangsschaltung
3. Umschalten zwischen den Schaltungen
4. Gesamtschaltungsmessung

**Messung der Ausgangsschaltung:** Es werden ausschließlich die in 6.4.2 erwähnten Bauteile verwendet. Damit die Akkus nicht gefährdet werden, wird die Messung mit einem Labornetzteil durchgeführt.

Dieses wird auf 8,4V (max Akkuspannung) gestellt, anschließend wird der Ausgang gemessen.

Die Ausgangsspannung beträgt, wie gefordert, 5V. Der maximale Ausgangsstrom beträgt mit  $T_6$  lediglich 2,0 Ampere, ansonsten 3,2 Ampere.  $T_6$  wird hierbei voll ausgesteuert. Zwei Ampere reichen für den RaspberryPI ohne Peripherie, aber mit Display aus.

Anschließend wird die Messung mit den Akkus wiederholt, wobei sich das Ergebnis nicht wesentlich ändert.

**Messung der Eingangsschaltung:** Es werden ausschließlich die in 6.4.1 erwähnten Bauteile verwendet. Die Akkus werden sofort eingebaut. Als VCC wird ein Labornetzteil verwendet, welches auf VCC=5V eingestellt wird.

Die Messung der Spannungspunkte ergibt das jeweils gewünschte Ergebnis.

Der Ladestrom ist aber viel zu niedrig, 2 mA. Zu Testzwecken werden die Sicherheitstransistoren  $T_4$  und  $T_5$  ausgebaut. Ohne diese beiden Transistoren kommt man auf einen maximalen Eingangsstrom von etwa 25 mA, was ebenfalls zu niedrig ist. Es werden noch mehrere Versuche unternommen, die Schaltung zu retten, was aber

nicht gelingt. Somit wird eine komplett neue Eingangsstufe entworfen, siehe hierfür auch die Idee der zweiten Akkusteuerungsschaltung (6.5).

## 6.5 Idee der zweiten Akkusteuerungsschaltung

Die zweite Akkusteuerungsschaltung unterscheidet sich primär von der ersten in der Eingangsschaltung bzw. der Akkuladeschaltung. Diesmal sind die Akkus ständig in Serie geschalten. Die Sicherheitsvorkehrungen wurden reduziert, sind aber immer noch ausreichend.

### 6.5.1 Laden der Akkus

Das Micro-USB-Kabel hat wiederum 5V Versorgungsspannung. Die beiden Akkus sind in Serie geschaltet, weshalb die Ladespannung 8,2V ( $2 \cdot 4,1\text{V}$ ) betragen muss. Diese Spannung wird durch einen StepUp Converter vom Typ LT1370 erreicht. Dieser ist ein verstellbarer Step-Up-Down-Converter.

Die Beschaltung des ICs und die Formel zur Berechnung der Widerstände wird wiederum aus dem Datenblatt entnommen (Siehe Abbildung 15 auf Seite XIV).

$$V_{out} = 8,2\text{V}; V_{ref} = 1,245\text{V}; R_{SU2} = 10\text{k}\Omega$$

$$V_{out} = V_{ref} \cdot \left(1 + \frac{R_{SU1}}{R_{SU2}}\right)$$

$$R_{SU1} = R_{SU2} \cdot \left(\frac{V_{out}}{V_{ref}} - 1\right) = 55,8\text{k}\Omega \rightarrow E - 12 \rightarrow R_{SU1} = 56\text{k}\Omega$$

Dieses Widerstandsverhältnis transformiert die Spannung von 5V auf 8,2V, wodurch die in Serie geschalteten Akkus geladen werden können.

In Punkt 6.5.5 ist die Schaltung des zweiten Versuches ersichtlich. Folgende Bauteile dieser Schaltung werden ausschließlich für das Laden verwendet:

*LT1370, C<sub>SU1</sub>, C<sub>SU2</sub>, C<sub>SU3</sub>, L<sub>SU1</sub>, R<sub>SU1</sub>, R<sub>SU2</sub>, R<sub>SU3</sub>, D<sub>SU1</sub>*

### 6.5.2 Entladen der Akkus

Die Entladeschaltung der zweiten Schaltung entspricht exakt der Entladeschaltung des ersten Versuches (siehe: Entladen der Akkus (6.4.2)).

In Punkt 6.5.5 ist die Schaltung des zweiten Versuches ersichtlich. Folgende Bauteile dieser Schaltung werden für das Laden verwendet:

$LM2596, D_{S1}, L_{SD1}, C_{SD2}, R_{SD1}, R_{SD2}$ .

### 6.5.3 Sicherheit

Als Sicherheit wird wiederum ein Über- und Unterladeschutz verwendet. So gibt es eine automatische Abschaltung, wenn der Akku vollgeladen oder entladen ist. Die Überwachung greift für jeden Akku einzeln. Dies geschieht über Komparatoren, aber nur einen Transistor. Dieser sperrt, wenn der Schutz greift und ist geöffnet, wenn der Akku geladen oder verwendet werden darf.

Um den Schutz beider Akkus zu ermöglichen, ist in der Serienschaltung eine Subtrahiererschaltung eingebaut. Die Differenz dieser Schaltung ergibt den momentanen Ladezustand des zweiten Akkus.

Die Referenzspannungen der Komparatoren ergeben sich aus einem Spannungsteiler zu:

- $V_{41} = 4,1V$
- $V_3 = 3V$

### 6.5.4 Messung

Die Ausgangsschaltung wurde schon in 6.4.6 geprüft und kann somit ausgelassen werden.

Das Hauptaugenmerk liegt auf der Eingangsstufe, welche die erste Schaltung nutzlos gemacht hat.

**Eingangsschaltung:** Die Messung der Eingangsschaltung erfolgt auf einem Steckbrett. VCC wird mittels eines Labornetzteils auf 5V eingestellt.

Die Spannungslevel stimmen bei allen Messpunkten. Der Eingangsstrom ist aber wiederum viel zu niedrig (10 mA mit  $T_1$ , ohne 50 mA).

**Sicherheit:** Es wird überprüft, ob die Sicherheitsabschaltung funktioniert. Dazu wird anstelle der Akkus ein Labornetzteil geschalten, welches einmal auf 8V  $U_{A2}$  und einmal auf 2,5V  $U_{A2}$  eingestellt wird. Durch diese Einstellung wird der Transistor  $T_1$  hochohmig.

Die Schaltung kann somit theoretisch als sichere Ladeschaltung verwendet werden. Das Laden eines Akkus dauert allerdings eine Woche. Da das Entwickeln einer neu-erlichen Schaltung aufgrund zeitlicher Beschränkungen nicht mehr möglich ist, wird eine Einigung mit Prof. Signitzer getroffen.

In dieser wird statt eines Akkumulators und einer Schaltung eine Powerbank ver-wendet. Für technische Daten siehe 6.6.

### 6.5.5 Schaltung

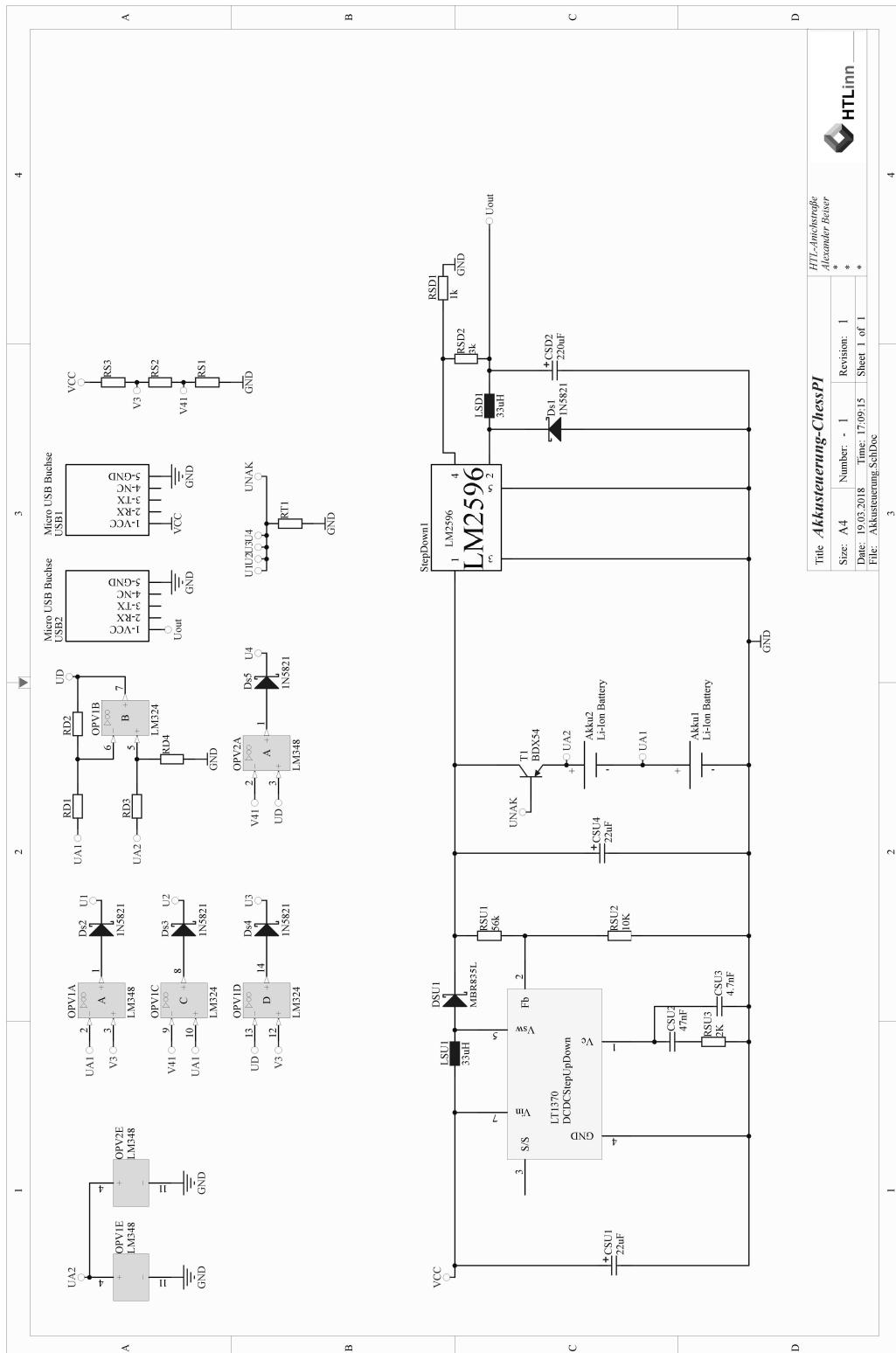


Abbildung 7: Die 2. Schaltung

## 6.6 Verwendete Lösung

Um einen ChessPI-Prototypen rechtzeitig fertigzustellen, wird auf eine selbst entwickelte Schaltung innerhalb des ChessPI verzichtet. Stattdessen soll eine Powerbank verwendet werden.

Diese muss folgende Aufgaben erfüllen:

- Min. Entladestrom von 2 A
- Spielzeit von >1h

Als Powerbank gibt es zwei zur Auswahl stehende Modelle:

- Das Modell RP-PB17 von RAVPower: Dieses erfüllt alle Anforderungen, welche die Powerbank erfüllen muss. Sie hat einen maximalen Ausgangsstrom von 2,4 Ampere und eine Kapazität von 5600 mAh, womit ein Betrieb von bis zu 2,5h möglich ist. Die Größe ist kleiner als der RaspberryPI, womit die Powerbank einfach in das Gehäuse integriert werden kann.

Datenblattreferenz für die Powerbank siehe Abbildung 10 auf Seite IX.

- Das Modell FREEPMULTI10000 von Cellularline - Dieses hat den Vorteil einer erhöhten Kapazität von 10.000 mAh gegenüber den 5600 mAh des anderen Modells. Dafür sind die Abmessungen der FREEPMULTI10000 Powerbank größer, womit ein Einbau in das Gehäuse nur schwer möglich ist.

Schlussendlich wird das erste Modell (RP-PB17) verbaut, da das zweite Modell nicht im Gehäuse Platz hat.

## 7 GEHÄUSE

Aufgabe war es, ein Gehäuse für den RaspberryPI mit Bildschirm und Akkusteuerung zu designen. Anstatt der Akkusteuerung wird nach 6.6 eine Powerbank verwendet. Für diese sollen noch Ein- und Ausschalter und eine Mikro-USB Buchse für die Versorgung eingebaut werden.

Das Gehäuse soll mittels 3D-Drucker erstellt werden, bzw. falls dies nicht in entsprechender Qualität geschehen kann, soll eines in der Kunststoffwerkstätte produziert werden.

### 7.1 Maße der Bauteile

Die Maße für den Bildschirm, den Akku und den RaspberryPI können im Anhang Datenblätter (VI) gefunden werden.

### 7.2 Geplantes mit dem 3D-Drucker gefertigtes Gehäuse

Das Gehäuse wird nach den vorherigen Kriterien entwickelt. Als Akkumodell wird das größere verwendet (FREEPMULTI10000).

Da das Gehäuse mit einem 3D-Drucker gefertigt werden soll, unterliegt die Fertigung gewissen Beschränkungen. Unter anderem ist ein Gehäuse in der ursprünglich gewünschten Größe nicht möglich, da der Drucker kein Gehäuse von 180x110x50mm, in einer realistischen Zeit und mit wirtschaftlichen Materialverbrauch drucken kann.

Dies führt zu einer Aufteilung des geplanten Gehäuses in drei Teile:

Die Basis soll auf den RaspberryPI geschraubt werden, darauf soll die Akkuhalterung geklebt werden. Da die Akkuhalterung oben offen ist, wird für diese auch ein Deckel geplant, welcher ebenfalls aufgeklebt wird.

Das Gehäuse kann produziert werden, allerdings muss das Aussehen vor einem eventuellen kommerziellen Vertrieb noch auf ein ansprechendes Design geprüft

werden. Diese Prüfung hat in unserem Fall ergeben, dass es für Endverbraucher mit großer Wahrscheinlichkeit nicht ansprechend wäre, weshalb wir eine verbesserte Version gefertigt haben (Siehe 7.3).

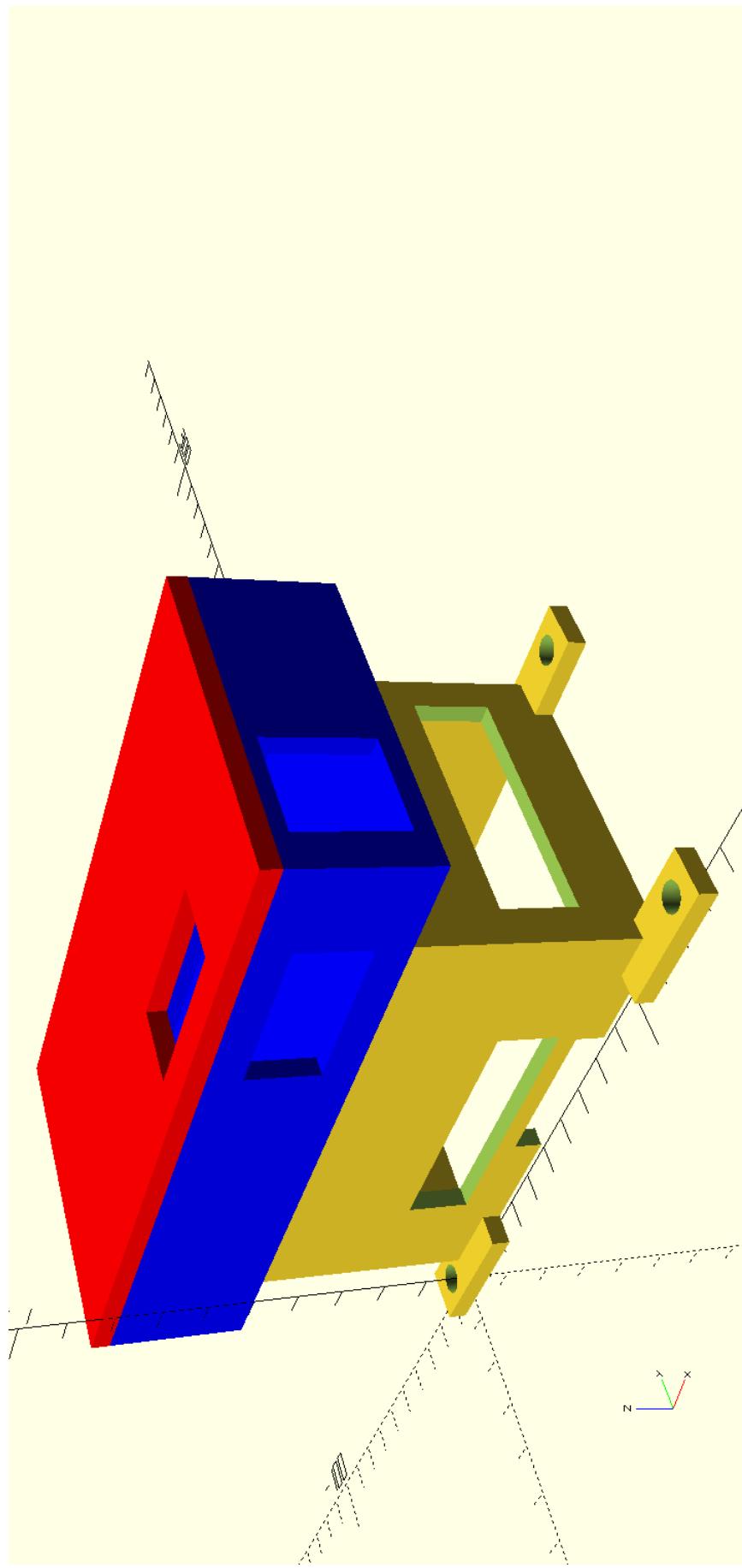


Abbildung 8: Das Gehäuse in den drei Teilen

### 7.3 Gehäuse - Kunststoffwerkstätte

Als Alternativlösung wird das Gehäuse in der Kunststoffwerkstätte gefertigt. Dieses soll aus poliertem weißem Kunststoff bestehen.

Dazu soll noch eine Ladebuchse und ein Schalter verbaut werden.

Nach Absprache mit Fachlehrer Strohmaier, welcher mich hier dankenswerter Weise unterstützt hat, wurde das Gehäuse in folgenden Schritten produziert:

- Die Bodenplatte und die beiden Seitenteile werden aus weißem Kunststoff, durch einen Lasercutter, herausgeschnitten.
- Die Bodenplatte wird gebogen und mit den beiden Seitenteilen verklebt.
- Es werden vier Löcher in das Gehäuse, für die Befestigungsschrauben gebohrt.
- Die Platte für den Bildschirm wird ausgeschnitten und die Senkung für den Bildschirm hineingefräst.
- Die Abstandshalter (innen) werden ausgeschnitten und in beide werden Löcher für die Schrauben gebohrt.
- Die Löcher für die Mikro-USB-Buchse und für den Ein-Ausschalter werden hineingeschnitten (Lasercutter).
- Es werden Schrauben für das Verschließen des Displays gesucht. Da es keine >50mm langen M3 Schrauben gibt, wird eine Gewindestange abgeschnitten und das Gehäuse mit einer Mutter befestigt.
- Damit die Mutter nicht gesehen wird, werden vier Standfüße besorgt, in denen die Mutter Platz hat.
- Der Akku wird mittels Kabelbindern und selbsthaftenden Pads befestigt.
- Von diesem geht ein aufgezwicktes USB-Kabel zum Schalter, um das Ein- und Ausschalten zu ermöglichen.

Das Laden des Akkus erfolgt mittels Mikro-USB Kabel, von der Mikro-USB Buchse zum Akku.

Ein Bild des fertigen ChessPI befindet sich auf der nächsten Seite.

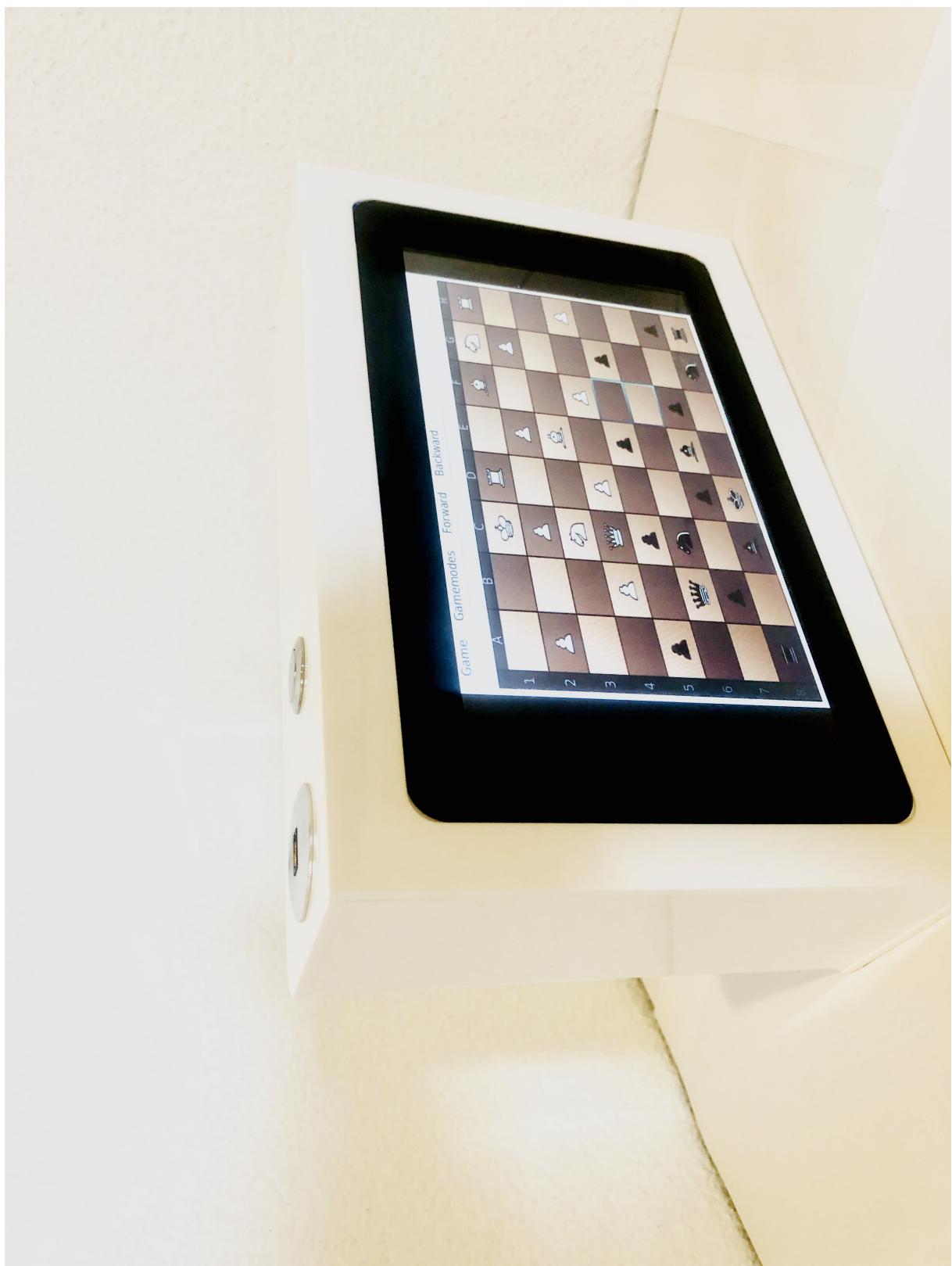


Abbildung 9: Der ChessPI

## 8 ANDROID

Um das Schachspiel überall spielen zu können, wird geplant, eine Version für die Plattform „Android“ zu entwickeln.

Diese soll eine vollständige Portierung von JavaChess darstellen und alle diese Funktionen beinhalten. Die Menüführung soll auf Benutzung mittels Wischgesten optimiert werden.

Da die Entwicklung der App eine komplett neue Implementierung der graphischen Benutzeroberfläche erfordern würde, und die Entwicklung der App zu einem relativ späten Zeitpunkt geplant ist, zudem die Desktop-Variante noch große Probleme beinhaltet, wird in Rücksprache mit dem Betreuer beschlossen, die Portierung der Android - App in der Frühphase abzubrechen.

## 9 BETA

Der letzte Meilenstein war die Beta-Phase. In dieser sollten alle Funktionen des Spiels durch ausgewählte Tester noch einmal auf Herz und Nieren geprüft werden. Die Beta Tester sind deshalb aus verschiedenen Hintergründen ausgewählt worden, zum Beispiel eine professionelle Schachspielerin oder Software-EntwicklerInnen. Dies hat den Vorteil unterschiedliche Nutzerverhalten zu testen.

Sowohl Alexander Beiser (9.1) als auch Marcel Huber (9.2) hatten fünf Tester zu den Punkten:

- I Generelles Spielgefühl
- II Eventuell gefundene Bugs bzw. Verbesserungsvorschläge

### 9.1 AI und Spielmechanik

#### **Chiara Polterauer (Schach-Staatsmeisterin)**

- I Findet das Spiel eigentlich ganz in Ordnung, hat aber ein paar Verbesserungsvorschläge für eine eventuelle Weiterentwicklung des Spiels.
- II Hat die KI im „Better-AI“ Modus auf dem maximalen Level geschlagen (bis jetzt einzige TesterIn) und wünscht sich somit eine bessere KI.

#### **Jonas Bangratz**

- I Um hier Jonas Bangratz wörtlich zu zitieren (22.03.2018): „Neben einigen Unregelmäßigkeiten, welche die Beta-Version zu diesem Zeitpunkt enthielt, war es für die Software dennoch möglich mir ein relativ gutes Spielgefühl zu vermitteln.“
- II Im Computermodus auf der niedrigsten Schwierigkeitsstufe, war es möglich den feindlichen König zu schlagen. Genauer betrachtet ist der feindliche König auf ein Feld gefahren, dass bedroht wird, wodurch dieser im nächsten Zug geschlagen werden konnte. Dies kann eigentlich gar nicht geschehen, da die KI nie den

eigenen König bedrohen würde. Da die KI auf der niedersten Schwierigkeitsstufe aber keine Züge voraus berechnet, weiß diese nicht, dass der König durch diesen Zug bedroht werden kann. Aus diesem Grund wurde die KI mit einem menschlichem Spieler in der Spiellogik vollständig gleichgestellt und darf diesen Zug nun nicht mehr ausführen. Dies hat Performance-Einbußen zur Folge, welche aber verschmerzbar sind.

### **Nadja Nicolussi**

- I Findet das Design ansprechend und die verschiedenen Spielmodi vernünftig.
- II Annahme: Ein Team hat als letzte Figur nur mehr den König. Dieser ist aber nicht ins Schach zu setzen, er weicht aus. Dies geschieht daher, dass es in der DRAW()-Methode eine Abfrage gibt, ob eine Patt Situation entstanden ist. Diese Abfrage wird erst gestartet, wenn der König die letzte verbleibende Figur ist. Diese Abfrage hatte einen Bug, welcher den König „ausweichen“ lies.

### **Moritz Schnell**

- I Findet das Spielgefühl ziemlich intuitiv.
- II Kompletter Spielabsturz durch einen Logikfehler in der AI Klasse. Die GUI war nicht mehr anklickbar, da das Überschreiben des aktuellen Spielstandes nicht funktioniert, wodurch ein Weiterspielen unmöglich war. Dieser Fehler ist durch „Race Conditions“ aufgetreten und wurde behoben.

### **Alina Schärmer**

- I Hier wird Alina Schärmer wörtlich zitiert: „Ich finde das Spiel echt gut gelungen.“
- II Annahme: Der Spieler des weißen Teams bewegt einen Bauern im Hot-Seat-Modus, anschließend wird in den KI-Modus (Standard-Einstellungen) gewechselt. Nach neuerlichem Anklicken des Spielfeldes, sollte die KI einen Zug tätigen.

Die KI tätigt aber zwei Züge, was nicht erlaubt ist. Dieser Fehler trat durch Race-Conditions beim Setzen des Boolean Team auf und ist mittlerweile behoben.

## 9.2 LAN und GUI

### 9.2.1 Josef Scharmer

1. Findet das Spiel unterhaltsam und ansprechend.
2. Joseph Scharmer fand in Zusammenarbeit mit Stefan Pichler einen Bug, der im LAN-Modus verhinderte, dass die Informationspopups bezüglich Schach und Schachmatt angezeigt werden. Dies lag an einer nicht ausgelösten Schach- bzw. Schachmattabfrage beim Empfangen des neuen Spielfeldes. Um das Problem zu lösen, wurden zusätzliche Schach- und Schachmattabfragen beim Empfangen eines Spielfeldes eingebaut.

### 9.2.2 Sabrina Schiestl

1. Findet das Spielgefühl intuitiv und gelungen.
2. Sabrina Schiestl entdeckte Fehler beim Verändern der Größe des Spielfeldes. In manchen Spielzuständen wurde die graphische Oberfläche beim Verändern der Größe des Fensters nicht richtig mitverändert. Diese Fehler wurden größten Teils behoben. Bei manchen Gelegenheiten wird das Verändern der Fenstergröße deaktiviert, um die Fehler zu vermeiden.

### 9.2.3 Katharina Weide

1. Nach den anfänglichen Abstürzen in der Betaphase gefiel ihr das Spielgefühl gut.
2. Katharina Weide stellte bei ihrem ersten Spielstart fest, dass das Spiel direkt abstürzt. Dies lag an einem Crash des Soundplayers, der auf ihrem Betriebssystem nicht verfügbar war. Die Lösung dieses Problems wurde im Kapitel 4.9.2 behandelt.

#### **9.2.4 Fabian Thurnes**

1. Lieferte folgendes Statement zum Programm ab: „Bei Casual Games ist mir ein besonders intuitives Nutzererlebnis wichtig. Ich freue mich, zum Erfolg dieses Projektes beigetragen zu haben.“
2. Fabian Thrunes war zwar weniger an der Bugfindung beteiligt, half aber sehr bei der Bewertung der Nutzerfreundlichkeit mit und machte Vorschläge diese zu verbessern. Konkret unterbreitete er den Vorschlag für die neue Menüführung.

# 10 BENUTZERDOKUMENTATION

Im folgenden Kapitel ist eine Bedienungsanleitung für JavaChess angegeben. Auch mögliche Fehlermeldungen, die das Programm ausgeben kann, werden im Folgendem abgehandelt.

## 10.1 Installationsanleitung

Grundsätzlich ist keine Installation von JavaChess notwendig. Zur Ausführung wird allerdings die aktuelle Version der „Java Runtime Environment“ oder kurz JRE, benötigt.

Um diese zu installieren, öffnet man einen Webbrowser und navigiert zur Homepage von Oracle. Dort wählt man im Menü den Punkt „Downloads“ aus. In den darauf folgenden Submenüs navigiert man weiter zum Punkt „Java“, und dann zu „Java RuntimeEnvironment“ (es kann auch dieser Link verwendet werden: <https://java.com/de/download/>). Dort klickt man auf den „Download“ Button und wählt die gewünschte Version zum Download aus. Die heruntergeladene Datei muss ausgeführt werden. Um die Installation abzuschließen, muss den Anweisungen im Wizard gefolgt werden.

Nach der erfolgreichen Installation der JRE kann JavaChess einfach ausgeführt werden.

## 10.2 Benutzung des Spiels

### 10.2.1 Der Hot-Seat-Modus

Um ein Spiel zu starten, muss einfach im Titelbildschirm mit der Maus geklickt werden. Das Programm wird automatisch in den „Hot-Seat“ - Spielmodus springen.

Im „Hot-Seat“ - Modus, oder auch lokaler Modus, können zwei Spieler am selben Computer gegeneinander spielen.

Um eine Figur zu bewegen, gibt es 2 Möglichkeiten:

- Eine Figur kann durch Klicken bewegt werden. Mit dem ersten Klick wird eine Figur ausgewählt. Daraufhin erscheinen Hilfen, die anzeigen, wohin die Figur bewegt werden kann. Leere Felder, auf die sich die Figur bewegen kann, werden blau umrandet. Felder, auf denen sich eine Figur befindet, die geschlagen werden kann, werden rot umrandet. Mit dem zweiten Klick auf eines der markierten Felder bewegt sich die Figur.
- Die zweite Möglichkeit, eine Figur zu bewegen, funktioniert mittels „Drag and Drop“. Die zu bewegende Figur wird angeklickt, die Maustaste wird gehalten. Daraufhin erscheinen wieder die bekannten Hilfen, die im vorhergehenden Punkt erwähnt wurden. Um die Figur nun zu bewegen, wird sie über das gewünschte Feld gezogen und losgelassen.

Wenn ein Spieler in den Zustand „Schach“ gerät, wird er mithilfe eines Popups darüber benachrichtigt. Dieses kann mit der vorgesehenen Schaltfläche geschlossen werden.

Erreicht ein Spieler den „Schachmatt“ - Zustand, so wird er ebenfalls mit einem Popup darüber informiert. Betätigt er auch hier die Schaltfläche, so wird das Popup geschlossen und ein neues Spiel wird gestartet.

Um von einem anderen Spielmodus in den Hot-Seat - Modus zurückzukehren, muss der Punkt „Hot-Seat“ im Menü „Gamemodes“ ausgewählt werden.

### 10.2.2 Der LAN - Modus

Mithilfe des LAN-Modus, auch Netzwerkmodus oder Onlinemode genannt, können zwei Spieler auf unterschiedlichen Computern miteinander spielen.

Voraussetzung dafür ist die Konnektivität mit einem Netzwerk, in dem sich beide Computer befinden.

Unterschieden wird in diesem Modus zwischen „Host“ und „Client“. Der Host wartet auf eine eingehende Verbindung, der Client initiiert diese.

Um im LAN-Modus zu spielen, wird der Punkt „Network“ im Menü „Gamemodes“ ausgewählt.

Nach dem Auswählen dieses Spielmodus öffnet sich ein Dialogfenster. In diesem gibt es 3 Möglichkeiten zur Auswahl: „host“, „join“ und „abort“.

Wird die Schaltfläche „host“ ausgewählt, so erscheint ein Wartebildschirm. Auf diesem wird unter anderem die IP-Adresse angezeigt, die der Client zum Verbindungsauftakt benötigt. Während der Wartebildschirm angezeigt wird, wartet das Programm im Hintergrund auf eingehende Verbindungen. Mittels Mausklick kann der Wartevorgang abgebrochen werden. Dies versetzt das Programm zurück in den lokalen Modus. Verschwindet der Wartebildschirm automatisch, so war der Verbindungsvorgang erfolgreich.

Wird die Schaltfläche „join“ im Dialogfenster ausgewählt, so erscheint ein weiteres Popup. In diesem Popup befindet sich ein Eingabefeld und zwei Schaltflächen.

Die Schaltfläche „abort“ versetzt das Programm zurück in den Hot-Seat - Modus.

Wird die Schaltfläche „Ok“ betätigt, so startet der Verbindungsvorgang zu der im Eingabefeld beschriebenen IP-Adresse.

Sobald eine erfolgreiche Verbindung besteht, kann das Spiel beginnen. Standardmäßig bewegt der Host die weißen Figuren und der Client die schwarzen Figuren. Die Steuerung des Spiels verläuft äquivalent zum Hot-Seat - Modus. Das bedeutet, die Figuren lassen sich sowohl durch Klicks, als auch durch Klicken und Ziehen bewegen. Auch die Informationspopups, die auf Schach- und Schachmattzustände hinweisen, sind im LAN - Modus vorhanden.

Um eine bestehende Verbindung zu trennen, muss im Menü „Gamemodes“ der Menüpunkt „disconnect“ ausgewählt werden. Anschließend befindet sich das Spiel wieder im Hot-Seat - Modus.

### 10.2.3 Der AI - Modus

Der AI - Modus, auch KI - Modus oder Computermodus genannt, ermöglicht es, alleine gegen einen Computergegner zu spielen.

Um in diesem Modus zu spielen, muss im Menü „Gamemodes“ der Menüpunkt „Computer“ ausgewählt werden. Ohne weitere Umwege startet der Modus.

Standardmäßig übernimmt der Computer die schwarzen Figuren.

Die Steuerung des Spiels verläuft auch hier äquivalent zum Hot-Seat - Modus. Das bedeutet, dass die Figuren auch in diesem Modus durch Klicken oder Drag and Drop bewegt werden können. Auch die bekannten Informationspopups treten in diesem Spielmodus auf.

Nachdem ein Zug getätigigt wurde, beginnt der Computer, seinen Gegenzug zu berechnen. **Achtung:** Dieser Berechnungsvorgang kann je nach Rechenleistung des Computers, auf dem das Spiel gestartet wurde, und der Schwierigkeitsstufe der AI, einen längeren Zeitraum in Anspruch nehmen. Um ein optimales Spielgefühl zu gewährleisten wird empfohlen, auf schwacher Hardware eine der niedrigeren Schwierigkeitsstufen auszuwählen.

Um den Computermodus wieder zu verlassen, muss im Menü „Gamemodes“ einer der anderen Spielmodi, also der Punkt „Network“ oder „HotSeat“ ausgewählt werden.

#### 10.2.4 Die erweiterten Optionen

Im Menü „other“ unter dem Menüpunkt „setup“ sind die erweiterten Einstellungen und Anzeigen zu finden. Diese werden in Form eines Popups zugänglich gemacht und sollen dazu dienen, mehr Informationen über das aktuelle Spiel zu erhalten und allgemeine Einstellungen zu verändern.

Grob gesagt unterteilt sich das Popup in drei Sektionen:

- **Audio Options:** In dieser Sektion werden Möglichkeiten gegeben, die Lautstärke des Spiels mithilfe eines Schiebereglers zu verändern oder den Sound mithilfe einer Checkbox aus- oder einzuschalten. Zusätzlich wird neben dem Schieberegler die aktuelle Lautstärke angezeigt und es wird eine Schaltfläche angezeigt, mit deren Hilfe ein Testgeräusch abgespielt werden kann.
- **AI Options:** Diese Sektion bietet die Möglichkeit, die Schwierigkeit des Computergegners anzupassen. Auch wird eine Einstellung zur Verfügung gestellt,

mit deren Hilfe die Farbe, welche der Computergegner übernimmt, angepasst werden kann. Mit der Checkbox „Better-AI“ kann ein alternativer AI-Algorithmus aktiviert werden, der einen noch schwierigeren Gegner bietet.

- **Other Information and Options:** Unter dieser Sektion sind Informationen über den aktuellen Rundenstand zu finden und das aktuell ziehende Team wird angezeigt. Aufgrund der Wünsche unserer Betatester wurde die Checkbox „Square Board“ eingebaut. Diese hat die Funktion, das Schachfeld auf ein quadratisches Seitenverhältnis zu beschränken.

#### 10.2.5 Alle Menüpunkte

- **Game:**
  - **New:** Mit diesem Menüpunkt kann ein neues Spielfeld erzeugt werden. Im LAN-Modus steht diese Funktion nicht zur Verfügung.
  - **Save:** Dieser Menüpunkt ermöglicht es, das aktuelle Schachfeld für einen späteren Zeitpunkt zu speichern. Wenn er ausgewählt wird, öffnet sich je nach laufendem Betriebssystem der entsprechende File - Explorer. Diese Funktion steht im LAN-Modus nicht zur Verfügung.
  - **Load:** Um die gespeicherten Spielfelder wieder laden zu können, wird dieser Menüpunkt verwendet. Auch hier öffnet sich nach dem Auswählen des Punktes der File - Explorer des Betriebssystems. Auch diese Funktion steht im LAN-Modus nicht zur Verfügung.
  - **Exit:** Dieser Menüpunkt stellt eine Möglichkeit dar, das Programm zu beenden. Als Alternative kann der Betriebssystemspezifische „Close - Button“ verwendet werden.
- **Gamemodes:**
  - **Hot-Seat:** Dieser Menüpunkt erlaubt es, den Hot-Seat - Spielmodus auszuwählen (vgl. 10.2.1).

- **Network:** Mit dem Auswählen dieses Menüpunktes beginnt die Konfiguration des LAN - Spielmodus (vgl. 10.2.2).
  - **Computer:** Durch das Auswählen dieses Menüpunktes startet der AI - Modus (vgl. 10.2.3).
  - **Launchpad:** Solange der Launchpad - Modus aktiv ist, ist es möglich, die Schachfiguren mithilfe des „Launchpad“ zu verschieben. Er funktioniert nur, wenn ein Launchpad verbunden ist. Dieser Modus ist durch eine Kooperation mit der Diplomarbeit „Launchpad“, von Alexander Hold und Thomas Klotz, entstanden. Weitere Informationen über das Launchpad sind in deren Arbeit enthalten (siehe GitHub: DarkSunHD - STM32F1 MIDI Serial Bridge).
- **Other:**
    - **Setup:** Mit dem Anwählen dieses Menüpunktes öffnen sich die erweiterten Optionen und Informationen. Mehr dazu unter 10.2.4.
    - **Draw:** Dieser Menüpunkt wird verwendet, um im Falle eines Patts das Spiel neu zu starten.
  - **Help:**
    - **About:** Wenn dieser Menüpunkt ausgewählt wird, öffnet sich ein Popup, in dem Informationen über die Entwickler, die verwendete Lizenz und die GitHub - Repository und das Logo des Spiels angeführt sind.
    - **Help:** Beim Auswählen dieses Menüpunktes öffnet sich ein weiteres Popup, in dem eine Kurzfassung dieser Anleitung zu finden ist.
  - **undo:** Mit dieser Schaltfläche ist es möglich, den letzten Zug ungeschehen zu machen. Diese Funktion ist nicht im LAN-Modus verfügbar.
  - **redo:** Um die Aktion des „Undo“ Menüpunktes umzukehren, wird dieser Menüpunkt verwendet.

## 10.3 Fehlermeldungen und Hinweise auf Fehlerursachen

### 10.3.1 Fehler beim Starten des Spiels

Wenn die „.jar - File“ nicht ausgeführt werden kann, überprüfen Sie bitte, ob eine funktionierende Version der „Java Runtime Environment“ auf Ihrem System installiert ist, und ob Sie über die aktuelle Version von JavaChess verfügen.

### 10.3.2 Fehler beim Speichern eines Spielstandes

Wenn beim Speichern eines Spieles ein Fehler auftritt, erscheint ein Popup mit der Beschriftung „File could not be saved!“. Dies kann folgende Ursachen haben:

- In dem Dateinamen, unter dem das Feld gespeichert werden sollte, ist ein Sonderzeichen enthalten, das nicht vom Dateisystem unterstützt wird. In diesem Fall muss der Name der zu speichernden Datei angepasst werden.
- Das Programm hat keine Schreibrechte. In diesem Fall muss dem Programm die Berechtigung zum Schreiben gegeben werden, bzw. muss das Programm als Administrator/Superuser gestartet werden.

### 10.3.3 Fehler beim Laden eines Spielstandes

Wenn beim Laden eines Spieles ein Fehler auftritt, erscheint ein Popup mit der Beschriftung „File could not be loaded!“. Dies kann folgende Ursachen haben:

- Die Datei, die versucht wird zu laden, ist keine .sav Datei, die vom Programm JavaChess gespeichert wurde. Stellen Sie sicher, dass Sie die richtige Datei ausgewählt haben.
- Die zu ladende Datei wurde verändert und deshalb beschädigt. In diesem Fall ist der Spielstand leider verloren und nicht wiederherzustellen.

### 10.3.4 Fehler im LAN - Modus

Folgende Fehlermeldungen könne im LAN - Modus auftreten:

- Wenn eine bestehende Netzwerkverbindung verloren geht, wird ein Informationsbildschirm mit der Meldung „The other player disconnected“ angezeigt. Diese Meldung können Sie mit einem Mausklick bestätigen. Im Anschluss kann das Spiel im HotSeat - Modus oder Computer - Modus fortgesetzt werden.
- Wenn beim Verbinden zu einem Host die Fehlermeldung „Connection Failed - the desired Host is not reachable“ auftritt, so ist die eingegebene IP-Adresse zwar gültig, jedoch antwortet der Host nicht. In diesem Fall sollten beide Spieler überprüfen, ob sie sich im selben Netzwerk befinden.
- Wenn beim Verbinden zu einem Host die Fehlermeldung „Connection Failed - It seems that something is wrong with the IP address!“ erscheint, so ist die eingegebene IP-Adresse falsch. Bitte überprüfen Sie, ob der Host im Netzwerk existiert und achten Sie darauf, ob die eingegebene Adresse dem Muster „127.0.0.0“ entspricht.

# ANHANG

## I ABBILDUNGSVERZEICHNIS

1	Das Blockschaltbild . . . . .	9
2	Informationsbildschirm beim Starten des Spiels . . . . .	42
3	Optionspopup . . . . .	45
4	Spielzyklus im LAN-Modus . . . . .	51
5	Min-Max . . . . .	67
6	Die 1. Schaltung . . . . .	85
7	Die 2. Schaltung . . . . .	90
8	Das Gehäuse in den drei Teilen . . . . .	94
9	Der ChessPI . . . . .	97
10	Datenblatt Powerbank . . . . .	IX
11	Datenblatt RaspberryPI Bildschirm . . . . .	X
12	Datenblatt LM2596-Seite 1 . . . . .	XI
13	Datenblatt LM2596-Seite 7 (Berechnung) . . . . .	XII
14	Datenblatt LT1370-Seite 1 . . . . .	XIII
15	Datenblatt LT1370-Seite 7 (Berechnung) . . . . .	XIV
16	Datenblatt Akku für Akkusteuerung . . . . .	XV

## II TABellenverzeichnis

1	Repräsentation der Figuren . . . . .	15
2	Arbeitsaufstellung nach Meilensteinen . . . . .	XXI

### III LITERATURVERZEICHNIS

- [1] Author = "Wikipedia contributors", Titel = "Chess — Wikipedia, The Free Encyclopedia", Jahr = "2018", URL = "<https://en.wikipedia.org/w/index.php?title=Chess&oldid=829981577>", Notiz = "[Online; accessed 19-March-2018]"
- [2] Author = Mike Klein, Titel = Google's AlphaZero Destroys Stockfish in 100-Game Match, URL = <https://www.chess.com/news/view/google-s-alphazero-destroys-stockfish-in-100-game-match>
- [3] Author = „Abarent”, Titel = Piece Square Table, URL = <http://www.chessbin.com/post/Piece-Square-Table>
- [4] Author = The MagPi Magazine, Titel = RASPBERRY PI 3 IS OUT NOW! SPECS, BENCHMARKS & MORE, URL = <https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/>
- [5] Author = „Gordon”, Titel = THE EAGERLY AWAITED RASPBERRY PI DISPLAY, URL = <https://www.raspberrypi.org/blog/the-eagerly-awaited-raspberry-pi-display/>
- [6] Author: The RaspberryPI Foundation, Titel: Power Supply, URL:<https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md>, Notiz = Online; Stand 19. März 2018
- [7] Author = "Wikipedia", Titel = „Akkumulator — Wikipedia, Die freie Enzyklopädie”, Jahr = "2018", URL = "<https://de.wikipedia.org/w/index.php?title=Akkumulator&oldid=175107886>", Notiz = "[Online; Stand 19. März 2018]"
- [8] Author = RS-Components, Titel = Ansmann Lithium-Akkupacks 3.7V/2600mAh, mit Drahtanschluss, URL = <https://at.rs-online.com/web/p/lithium-akkus/7760853/?searchTerm=776-0853>, Notiz = Online; Stand 19. März 2018, RS-Best. Nr.: 776-0853, Herst. Teile-Nr.: 2347-3003

[9] Author = RS-Components, Titel = RS Pro Li-Po Akku 3.7V, 2000 mAh, mit Drahtanschluss, URL = <https://at.rs-online.com/web/c/batterien/akkus/lithium-akkus/?applied-dimensions=4294830943>, Notiz = Online; Stand 19.März 2018, RS-Best Nr.: 125-1266

[10] Author 1 = Andrew Hunt, Author 2 = David Thomas, Titel = The Pragmatic Programmer, Untertitel = from journeyman to master, Verlag = Addison-Wesley, Verlagsort = Crawfordsville, Indiana, United States of America, Jahr = 2000

## IV ABKÜRZUNGEN

<b>PC</b>	Personal Computer
<b>HTL</b>	Höhere Technische Lehranstalt
<b>GUI</b>	Graphical-User-Interface
<b>GNU</b>	GNU's Not Unix
<b>LAN</b>	Local Area Network
<b>AI</b>	Artificial-Intelligence
<b>vgl.</b>	vergleiche
<b>dt.</b>	zu Deutsch
<b>engl.</b>	englisch
<b>App.</b>	Application
<b>ARM</b>	Advanced RISC Machine

# V Pflichtenheft

**Pflichtenheft für die Diplomarbeit  
JavaChess, ChessPI AndChess**

## V.I JAVAChess KONZEPT

Das Programm ist als 2D Schachspiel konzipiert. Basierend auf Java und JavaFX wird das Spiel entwickelt.

Das Spiel soll online spielbar gemacht werden, aber auch über einen Einzelspielermodus verfügen. Dieser Einzelspielermodus beinhaltet zudem eine selbst entwickelte künstliche Intelligenz.

Die grafische Oberfläche setzt sich aus einem selbst designten Schachbrett, Bedienflächen und aus Opensource-Quellen stammenden Ressourcen für die Figuren zusammen.

Selbstverständlich sollen alle möglichen Züge implementiert- und ein passender Schachmatt-Algorithmus entwickelt werden.

Auf eine ressourcenschonende Zugberechnung soll besonderer Wert gelegt werden, da die künstliche Intelligenz auf denselben Zugmechanismus zugreift, wie der menschliche Spieler.

Die Bedienung des Programms soll auf mehrere Arten möglich sein: mittels Click to Click\* und Drag and Drop.

\*Click-to-Click: Spieler A klickt auf Feld d7 – dann auf Feld d6 – Figur bewegt sich von d7 nach d6.

## V.II Einzubauende Features

### V.II.I JavaChess (Desktop Version - Windows)

#### 1. GUI

- (a) 2D Darstellung des Schachbretts
- (b) Bedienflächen, in Menüs geordnet
- (c) Spielstand Indikatoren

#### 2. Programmlogik

- (a) Zugfunktion

- (b) Schachmatt-Algorithmus
  - i. Es soll auf die Übersichtlichkeit des Programmcodes geachtet werden
  - ii. Die Kommentare des Programmcodes sollen auf Englisch erfolgen

### 3. Künstliche Intelligenz

- (a) Im Einzelspielermodus ist es möglich, gegen eine selbst entwickelte künstliche Intelligenz zu spielen.
- (b) Die Evaluierung des besten Zuges der künstlichen Intelligenz erfolgt mithilfe eines abgeänderten Min-Max-Algorithmus.

### 4. Netzwerk

- (a) Austausch der Spieldaten über ein Local Area Network.

## V.II.II ChessPI (Raspberry PI 3)

- 1. Für einen 7“ Touchscreen optimierte Spielerfahrung
- 2. Optimierung der künstlichen Intelligenz für Niedrigleistung
- 3. Eigenentwickeltes Gehäuse, welches mithilfe eines 3D-Druckers gebaut wird
- 4. Selber entwickelte Akkuansteuerungsschaltung und Einbau des Akkus innerhalb des Gehäuses

## V.II.III AndChess (Android)

- 1. Vollständig portierter Programmcode für Android basierte mobile Geräte
- 2. Angepasste GUI für eine bessere Bedienung auf mobilen Geräten

## V.III Appendix

### V.III.I Vorhandene Bugs

Hier werden alle bekannten Bugs gelistet, welche behoben werden sollen:

- Schachmatt-Algorithmus gibt, außer in einem bestimmten Fall, nur Schach aus. Dieser bestimmte Fall tritt ein, wenn zwei gegnerische Figuren den König bedrohen.
- Die Züge Rochade, En Passant, Bauerntausch und der Doppelzug des Bauern, falls dieser sich noch nicht bewegt hat, sind nicht eingebaut.
- Die Züge des Läufers und der Dame haben den Fehler, falls diese diagonal ziehen, sind auch illegale Züge erlaubt. Z.B.: Lc8 -> Le6
- Es gibt keine Feststellung, ob Team1 oder Team2 am Zug ist, beide funktionieren immer.
- Falls es zu einer „Schachsituation“ kommt, kann unter besonderen Bedingungen eine Endlosschleife entstehen, die das Programm abstürzen lassen kann.

# VI DATENBLÄTTER

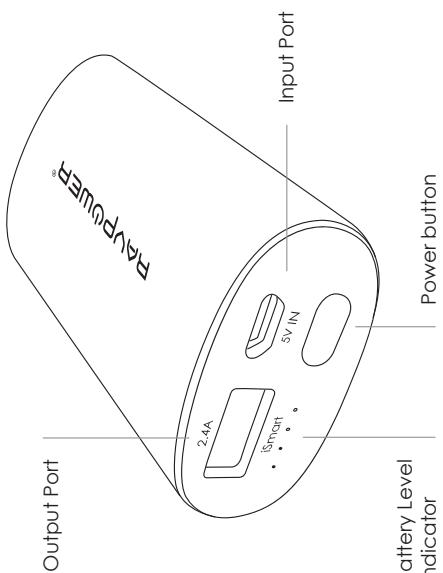
## What's In the Box

- 1 x RAVPower Power Bank (Model: RP-PB17)
- 1 x USB Charging Cable
- 1 x User Manual

## Specifications

Model	RP-PB17
Capacity	5200 mAh / 19.2 Wh
Input	DC 5 V / 2 A
Output	DC 5 V / 2.4 A
Dimensions	92mm x 46mm x 22.5mm 3.62" x 1.81" x 0.89"
Net Weight	135g/4.76oz

## Product Diagram



Thank you for choosing the RAVPOWER Luster 5200 mAh Power Bank. Please read this manual and keep it as reference. If you need any further assistance, please contact us or email RAVPOWER support at [support@ravpower.com](mailto:support@ravpower.com).

US UK CA

Abbildung 10: Datenblatt Powerbank

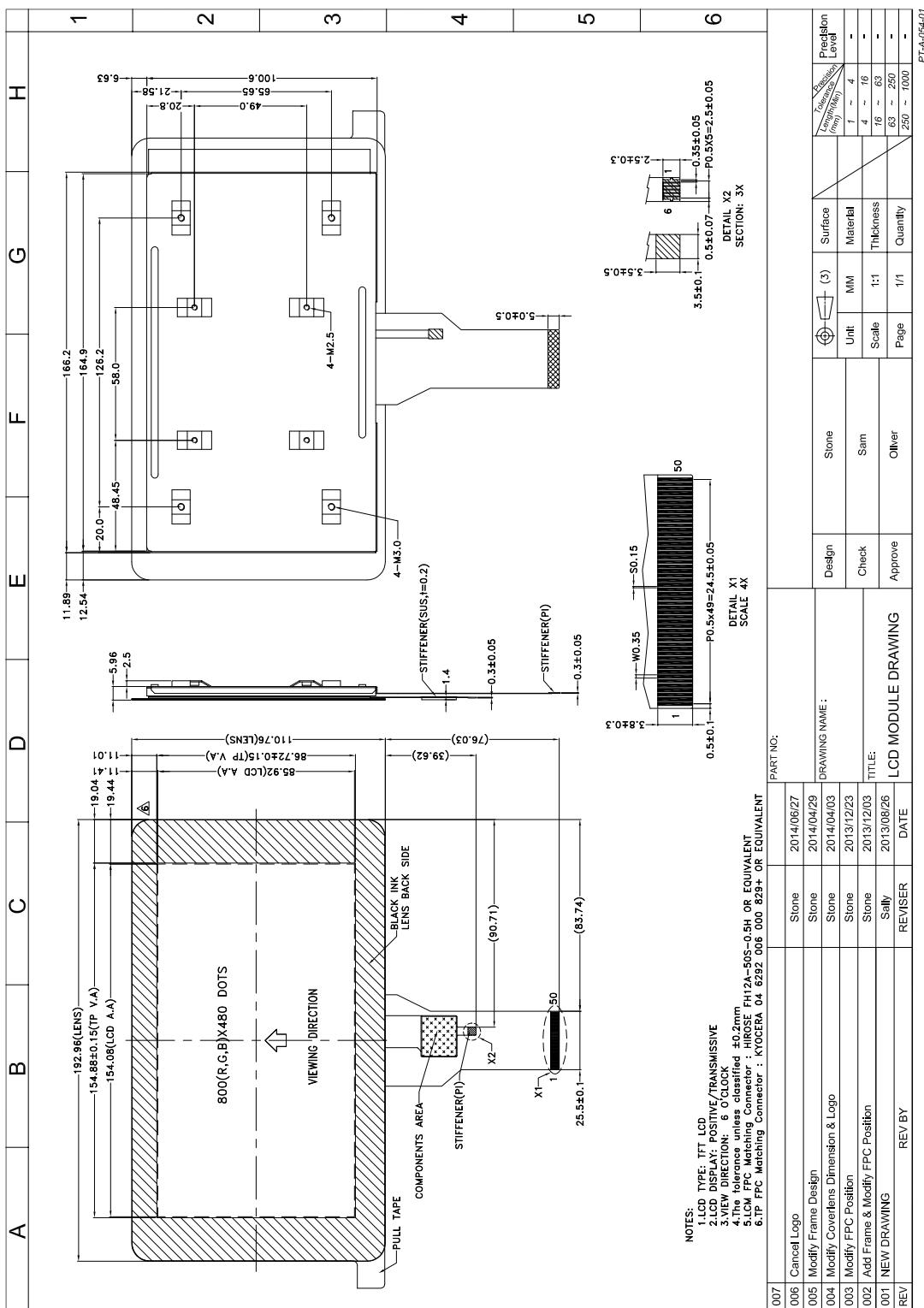


Abbildung 11: Datenblatt RaspberryPI Bildschirm

## LM2596

### 3.0 A, Step-Down Switching Regulator

The LM2596 regulator is monolithic integrated circuit ideally suited for easy and convenient design of a step-down switching regulator (buck converter). It is capable of driving a 3.0 A load with excellent line and load regulation. This device is available in adjustable output version and it is internally compensated to minimize the number of external components to simplify the power supply design.

Since LM2596 converter is a switch-mode power supply, its efficiency is significantly higher in comparison with popular three-terminal linear regulators, especially with higher input voltages.

The LM2596 operates at a switching frequency of 150 kHz thus allowing smaller sized filter components than what would be needed with lower frequency switching regulators. Available in a standard 5-lead TO-220 package with several different lead bend options, and D<sup>2</sup>PAK surface mount package.

The other features include a guaranteed  $\pm 4\%$  tolerance on output voltage within specified input voltages and output load conditions, and  $\pm 15\%$  on the oscillator frequency. External shutdown is included, featuring 80  $\mu$ A (typical) standby current. Self protection features include switch cycle-by-cycle current limit for the output switch, as well as thermal shutdown for complete protection under fault conditions.

#### Features

- Adjustable Output Voltage Range 1.23 V – 37 V
- Guaranteed 3.0 A Output Load Current
- Wide Input Voltage Range up to 40 V
- 150 kHz Fixed Frequency Internal Oscillator
- TTL Shutdown Capability
- Low Power Standby Mode, typ 80  $\mu$ A
- Thermal Shutdown and Current Limit Protection
- Internal Loop Compensation
- Moisture Sensitivity Level (MSL) Equals 1
- Pb-Free Packages are Available

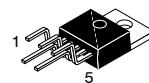
#### Applications

- Simple High-Efficiency Step-Down (Buck) Regulator
- Efficient Pre-Regulator for Linear Regulators
- On-Card Switching Regulators
- Positive to Negative Converter (Buck-Boost)
- Negative Step-Up Converters
- Power Supply for Battery Chargers



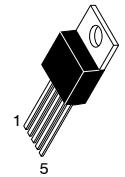
**ON Semiconductor®**

<http://onsemi.com>



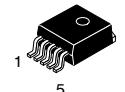
TO-220  
TV SUFFIX  
CASE 314B

Heatsink surface connected to Pin 3



TO-220  
T SUFFIX  
CASE 314D

- Pin    1.  $V_{in}$   
      2. Output  
      3. Ground  
      4. Feedback  
      5. ON/OFF



D<sup>2</sup>PAK  
D2T SUFFIX  
CASE 936A

Heatsink surface (shown as terminal 6 in case outline drawing) is connected to Pin 3

#### ORDERING INFORMATION

See detailed ordering and shipping information in the package dimensions section on page 23 of this data sheet.

#### DEVICE MARKING INFORMATION

See general marking information in the device marking section on page 23 of this data sheet.

**LM2596**

TYPICAL PERFORMANCE CHARACTERISTICS (Circuit of Figure 15)

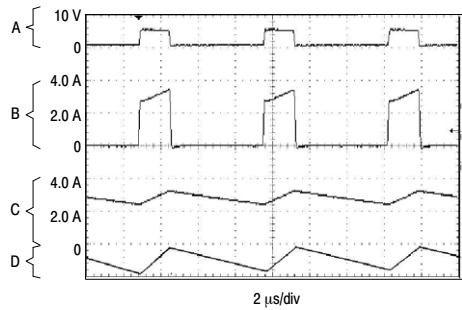


Figure 13. Switching Waveforms

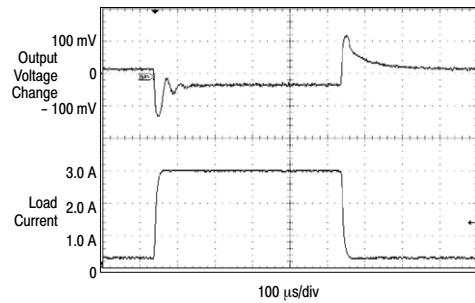


Figure 14. Load Transient Response

V<sub>out</sub> = 5 V  
 A: Output Pin Voltage, 10 V/div  
 B: Switch Current, 2.0 A/div  
 C: Inductor Current, 2.0 A/div, AC-Coupled  
 D: Output Ripple Voltage, 50 mV/div, AC-Coupled  
 Horizontal Time Base: 5.0 μs/div

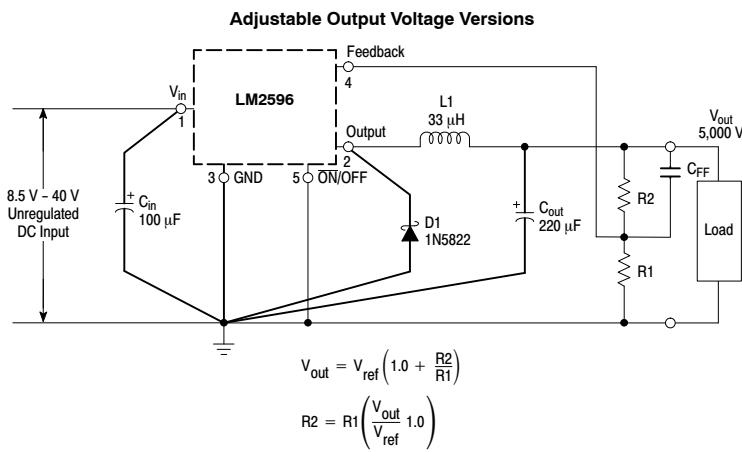


Figure 15. Typical Test Circuit

Abbildung 13: Datenblatt LM2596-Seite 7 (Berechnung)



## FEATURES

- Faster Switching with Increased Efficiency
- Uses Small Inductors:  $4.7\mu\text{H}$
- All Surface Mount Components
- Low Minimum Supply Voltage: 2.7V
- Quiescent Current: 4.5mA Typ
- Current Limited Power Switch: 6A
- Regulates Positive or Negative Outputs
- Shutdown Supply Current:  $12\mu\text{A}$  Typ
- Easy External Synchronization
- Switch Resistance:  $0.065\Omega$  Typ

## APPLICATIONS

- Boost Regulators
- Laptop Computer Supplies
- Multiple Output Flyback Supplies
- Inverting Supplies

## DESCRIPTION

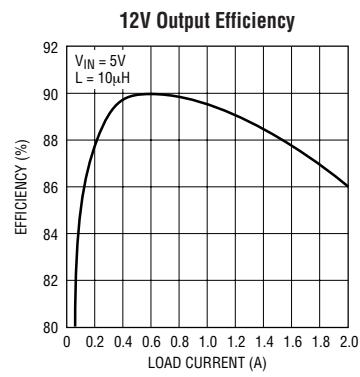
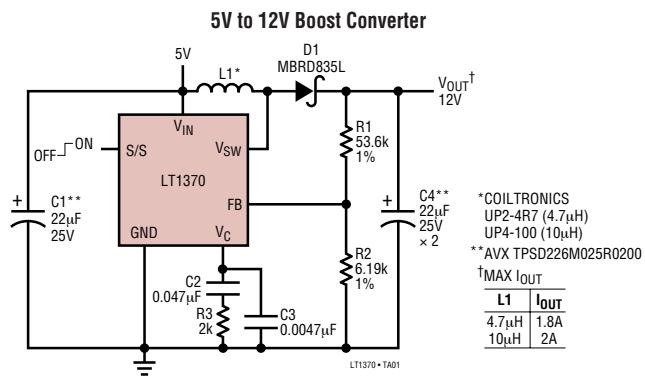
The LT<sup>®</sup>1370 is a monolithic high frequency current mode switching regulator. It can be operated in all standard switching configurations including boost, buck, flyback, forward, inverting and "Cuk." A 6A high efficiency switch is included on the die, along with all oscillator, control and protection circuitry.

The LT1370 typically consumes only 4.5mA quiescent current and has higher efficiency than previous parts. High frequency switching allows for very small inductors to be used.

New design techniques increase flexibility and maintain ease of use. Switching is easily synchronized to an external logic level source. A logic low on the Shutdown pin reduces supply current to  $12\mu\text{A}$ . Unique error amplifier circuitry can regulate positive or negative output voltage while maintaining simple frequency compensation techniques. Nonlinear error amplifier transconductance reduces output overshoot on start-up or overload recovery. Oscillator frequency shifting protects external components during overload conditions.

LT, LTC and LT are registered trademarks of Linear Technology Corporation.

## TYPICAL APPLICATION



sn1370\_1370fs



1

Abbildung 14: Datenblatt LT1370-Seite 1

## OPERATION

Unique error amplifier circuitry allows the LT1370 to directly regulate negative output voltages. The negative feedback amplifier's 100k source resistor is brought out for negative output voltage sensing. The NFB pin regulates at  $-2.48V$  while the amplifier output internally drives the FB pin to  $1.245V$ . This architecture, which uses the same main error amplifier, prevents duplicating functions and maintains ease of use. Consult LTC Marketing for units that can regulate down to  $-1.25V$ .

The error signal developed at the amplifier output is brought out externally. This pin ( $V_C$ ) has three different

functions. It is used for frequency compensation, current limit adjustment and soft starting. During normal regulator operation this pin sits at a voltage between  $1V$  (low output current) and  $1.9V$  (high output current). The error amplifier is a current output ( $g_m$ ) type, so this voltage can be externally clamped for lowering current limit. Likewise, a capacitor coupled external clamp will provide soft start. Switch duty cycle goes to zero if the  $V_C$  pin is pulled below the control pin threshold, placing the LT1370 in an idle mode.

## APPLICATIONS INFORMATION

### Positive Output Voltage Setting

The LT1370 develops a  $1.245V$  reference ( $V_{REF}$ ) from the FB pin to ground. Output voltage is set by connecting the FB pin to an output resistor divider (Figure 1). The FB pin bias current represents a small error and can usually be ignored for values of  $R_2$  up to  $7k$ . The suggested value for  $R_2$  is  $6.19k$ . The NFB pin is normally left open for positive output applications. Positive fixed voltage versions are available (consult LTC Marketing).

### Negative Output Voltage Setting

The LT1370 develops a  $-2.48V$  reference ( $V_{NFR}$ ) from the NFB pin to ground. Output voltage is set by connecting the NFB pin to an output resistor divider (Figure 2). The  $-30\mu A$  NFB pin bias current ( $I_{NFB}$ ) can cause output voltage errors and should not be ignored. This has been accounted for in the formula in Figure 2. The suggested value for  $R_2$  is  $2.49k$ . The FB pin is normally left open for negative output applications.

### Dual Polarity Output Voltage Sensing

Certain applications benefit from sensing both positive and negative output voltages. One example is the "Dual Output Flyback Converter with Ovvoltage Protection" circuit shown in the Typical Applications section. Each output voltage resistor divider is individually set as described above. When both the FB and NFB pins are used,

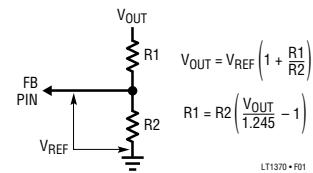


Figure 1. Positive Output Resistor Divider

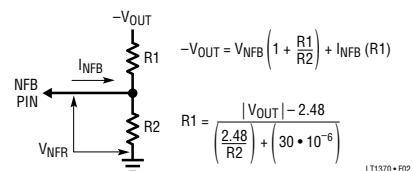


Figure 2. Negative Output Resistor Divider

the LT1370 acts to prevent either output from going beyond its set output voltage. For example, in this application if the positive output were more heavily loaded than the negative, the negative output would be greater and would regulate at the desired set-point voltage. The positive output would sag slightly below its set-point voltage. This technique prevents either output from going unregulated high at no load.

Abbildung 15: Datenblatt LT1370-Seite 7 (Berechnung)

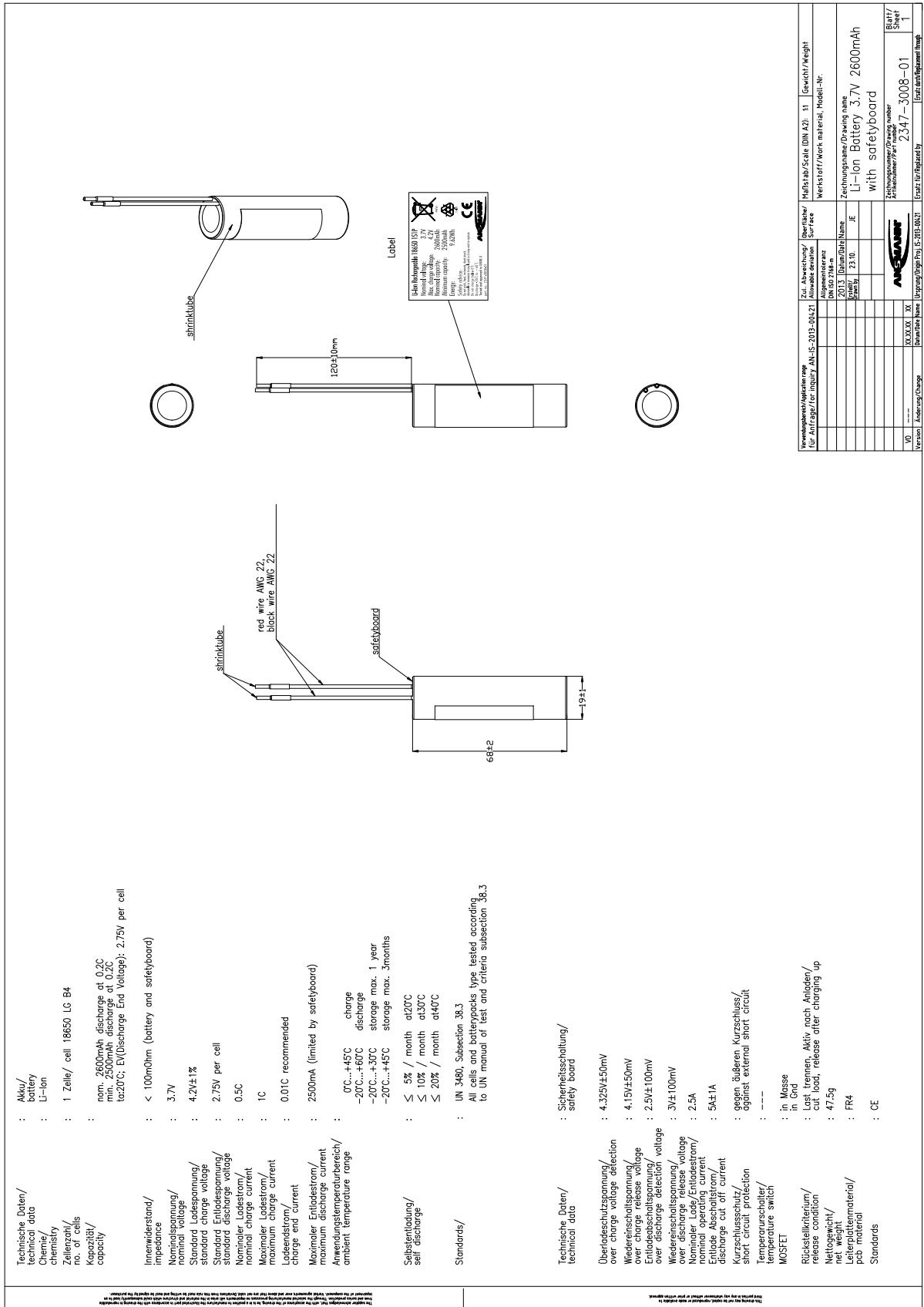


Abbildung 16: Datenblatt Akku für Akkusteuerung

# VII ZUSAMMENFASSUNG

## VII.I Alexander Beiser

Alexander Beiser war für einen Großteil der Spiellogik zuständig. Somit fallen darunter unter anderem:

- Logik des Ziehens der Spielfiguren,
- Schachmatt und Schach Abfrage,
- Speichern und Laden,
- und der Entwicklung des Computers (künstliche Intelligenz).

Außerhalb des JavaChess Bereiches hat er, trotz Hindernissen, das Spiel auf den RaspberryPI portiert. Dazu hat er ein Gehäuse entwickelt und das Ganze mit einem Akku abgerundet, wodurch ein mobiles Spielen mit dem sogenannten „ChessPI“ möglich ist.

Dies ist das abgelieferte Ergebnis. Zusätzliche Aufgabe war es, eine Akkusteuerung selber zu entwerfen. Technisch gesehen sind zwei fertige Entwürfe dieser Akkusteuerung entwickelt worden, in der Messung der jeweiligen Schaltung stellte sich heraus, dass jedoch ein Laden mit diesen Schaltungen sehr lange dauern würde. Dadurch verfehlten diese Schaltungen ihren eigentlichen praktischen Nutzen. Die Weiterentwicklung dieser Schaltung, um dadurch ein Laden in entsprechender Zeit zu ermöglichen, schlugen fehl und somit wurde aus Zeitmangel heraus eine Powerbank verwendet.

## VII.II Marcel Huber

Marcel Huber war für große Teile des „Frontends“ zuständig. Er entwickelte die grafische Benutzeroberfläche und gestaltete diese in Rücksprache mit den Testern so, dass sie ein optimales Spielerlebnis bietet.

Er implementierte weitere Optionen und sorgte für eine passende Untermalung des Spielerlebnisses mithilfe von Soundeffekten.

Zudem wurde ein Spielmodus entwickelt, der es ermöglicht, zwei Spieler über ein Netzwerk spielen zu lassen.

Die Entwicklung der Android - Version wurde aufgrund von massivem Zeitmangel in der Frühphase eingestellt.

## VII.III -Schlussfolgerung / Projekterfahrung

Das Projekt ist trotz einiger Rückschläge in weiten Teilen erfolgreich. Die künstliche Intelligenz ist stark genug um menschliche Spieler zu schlagen und das Spielen im Online Modus funktioniert tadellos.

Beide Kandidaten hatten in einigen Punkten mit massiven Problemen zu kämpfen. Vor allem drei Punkte stellten sich als einen größeren Zeitaufwand heraus, als geplant:

1. Implementierung von JavaFX

2. Das Portieren auf den RaspberryPI

3. Die Entwicklung der Akkusteuerungsschaltung

1) - Das Verwenden von JavaFx bereitete anfänglich Probleme. Besonders im Bereich des Multithreading, also das Auslagern von gewissen Aufgaben auf andere Arbeitsstränge sorgte im Zusammenhang mit JavaFx für Schwierigkeiten. Auch die mangelnde Ausbildung bezüglich JavaFx im Unterricht und die daraus folgende selbstständige Einarbeitung stellten eine besondere Herausforderung dar.

2) - Das Portieren auf den RaspberryPI stellte sich als einen sehr viel größeren Zeitaufwand heraus, als geplant. Oracle, also die Entwickler von Java, haben die Unterstützung von JavaFX für „Embedded-Processors“ eingestellt. Dadurch musste eine funktionierende Alternative gefunden werden. Nachdem diese Alternative gefunden wurde, stellte sich heraus, dass der Bildschirm in Verbindung mit Java einen gewissen „Offset“ besitzt, also der gedrückte Punkt stimmt nicht mit dem angezeigten überein. Dieser Fehler konnte ebenfalls unter entsprechendem Zeitaufwand behoben werden.

3) - Die Entwicklung der Akkusteuerung stellte sich als ebenfalls größeres Problem als gedacht dar, da der Ladestrom viel zu klein war, um eine praktische Verwendung zu ermöglichen.

Die wichtigste Projekterfahrung ist jedenfalls, dass bevor ein Projekt begonnen wird,

eine Machbarkeitsstudie durchgeführt werden sollte. Dadurch können potentielle „Zeit fressende Singularitäten“ vermieden werden. Weiteres sollte man mehr Zeit für eventuelle Fehler einplanen, da der Zeitplan sonst unmöglich einzuhalten ist. In zukünftigen Arbeiten im Team sollte man sich mehr absprechen und ständig in Kontakt über den momentanen Stand der Teammitglieder informiert sein, um die Effizienz und Effektivität zu steigern.

## **VII.IV -Kostenaufstellung und Arbeitsnachweis**

### **VII.IV.I Kostenaufstellung**

Hier erfolgt die Aufstellung der Projektkosten. Die Aufstellung erfolgt ohne Miteinberechnung der Versandkosten.

WAS	WIE VIEL	PREIS(€)
N-Mosfet	7	3.78
4 Ohm Widerstand	5 (Packung)	1.51
1N5408	10 (Packung)	5.2
PNP-Transistor	10 (Packung)	2.99
R-680 Ω	10 (Packung)	0.26
R-3.9kΩ	10 (Packung)	0.45
1N5821	50 (Packung)	15.25
Akku	2	35.94
LM348	10 (Packung)	3.34
R-1.8 kΩ	10 (Packung)	0.08
LM2596	1	3.87
C-100µF	5 (Packung)	1.23
L-33 µH	1	2.89
C-220µF	5 (Packung)	0.59
Sicherung 4A	10 (Packung)	0.73
Raspberry 3	1	61.41
7" Touchscreen	1	57.7
Micro-USB Buchse	1	20.99
Schalter	1	12.99
M-3-Stange	1	0.89
USB-2.0 Kabel	1	9.99
Mikro-USB-Verlängerung	1	9.99
LT-1370-SMD	1	11.41
LT-1370-TO220	1	21.99
Powerbank-RP-PB17	1	17.99
$\Sigma$		316.31

**VII.IV.II Arbeitsnachweis Diplomarbeit**

<b>Arbeitsaufstellung</b>				
<u>Was</u>	<u>Person</u>	<u>Stunden</u>	<u>Von</u>	<u>Bis</u>
<i>Beginn der Diplomarbeit - 14.09.2017</i>				
Schachmatt Algorithmus und Hotseat Modus	Beiser	30	14.09.2017	02.10.2017
Implementierung der Netzwerkfähigkeit	Huber	30	14.09.2017	02.10.2017
<i>Meilenstein 1 - 02.10.2017</i>				
Entwicklung der künstlichen Intelligenz	Beiser	30	02.10.2017	06.11.2017
Implementierung von JavaFX	Huber	40	02.10.2017	06.11.2017
<i>Meilenstein 2 - 06.11.2017</i>				
Raspberry-PI Implementierung und Design des Gehäuses	Beiser	35	06.11.2017	18.12.2017
Komplettüberarbeitung der GUI für eine bessere Benutzer Erfahrung	Huber	34	06.11.2017	18.12.2017
<i>Meilenstein 3 - 18.12.2017</i>				
Entwicklung der Akku- Steuerschaltung und Einbau dieser	Beiser	34	18.12.2017	04.02.2018
Portierung auf Android	Huber	15	18.12.2017	04.02.2018
<i>Meilenstein 4 - 04.02.2018</i>				
Durchführung einer Beta-Phase und Fehlerbehebungen	Beiser	30	04.02.2018	05.03.2018
Durchführung einer Beta-Phase und Fehlerbehebungen	Huber	42	04.02.2018	05.03.2018
<i>Meilenstein Dokumentation</i>				
Dokumentation	Beiser	21	14.09.2017	04.04.2018
Dokumentation	Huber	19	14.09.2017	04.04.2018

Tabelle 2: Arbeitsaufstellung nach Meilensteinen