

# **Entwicklung von Multimediasystemen Belegaufgabe „Reversi“**

von

Jan Didschuneit – s0539714

Baran Özyüzücüler - s0540255

# Inhaltsverzeichnis

## Inhaltsverzeichnis

1. Projektbeschreibung.....	3
1.1 Allgemein.....	3
2.2 Eigene Anforderungen.....	3
2. Entwicklungsprozess.....	4
2.1 Projektierungsphase.....	4
2.2 Entwicklungsphase.....	4
2.2.1 Die Datenbank – SQLite.....	4
2.2.2 Die künstliche Intelligenz.....	4
2.2.3 Git als Verwaltungshilfe.....	5
2.2.4 Nicht bearbeitet Features.....	5
3. Die Applikation.....	6
3.1 Startbildschirm.....	6
3.2 Highscore.....	7
3.3 Optionen.....	8
3.4 Spieler gegen Spieler.....	9
3.5 Spieler gegen Computers.....	10
3.6 Spielfeld.....	11
4. Testing.....	12
4.1 Unittests.....	12
4.2 Multiplattform Testing.....	12
4.3 Memory Tests.....	12
5. Anmerkungen.....	13
5.1 Bugs.....	13
5.2 Letzte Wort.....	13

# 1. Projektbeschreibung

## 1.1 Allgemein

Die allgemeine Aufgabe war es, das Spiel Reversi zu entwickeln, programmieren und den Code sowie das Programm auf Funktionalität zu testen. Der rein logische Teil sollte Qt-unabhängig mit C++ entwickelt werden, die grafische Oberfläche dagegen mit Qt. Die erforderlichen Anforderungen wurden vom Dozenten in der ausgehändigten PDF festgelegt.

## 2.2 Eigene Anforderungen

Getreu den neuen Anforderungen des Sommersemesters 2015 sollte den gegebenen Anforderungen noch fünf selbst gewählte Punkte hinzugefügt werden. Wir haben uns hierfür für folgende Punkte entschieden.

1. Highscoreinträge sortiert anzeigen lassen
2. Musiklautstärke regeln oder komplett stumm schalten
3. Eigene Musik laden
4. Spiel auf Zeit als Option bei Spieler gegen Spieler
5. Mögliche Züge anzeigen ein- oder ausschaltbar

## 2. Entwicklungsprozess

### 2.1 Projektierungsphase

Im Vorfeld galt es erst einmal sich in die Eigenschaften und Gegebenheiten von Qt einzuarbeiten. Dies geschah teilweise mithilfe der Qt eigenen Tutorials. Zudem mussten wir uns erst einmal Gedanken zu allgemeinen Überlegungen des Projekts machen, wie zum Beispiel welche Architektur wir verwenden wollten, was für Klassen wir brauchen, wie diese miteinander kommunizieren oder voneinander abhängen und andere Dinge. Dennoch kam es im Laufe späterer Phasen zu neuen Erkenntnissen weswegen die Projektstruktur teilweise neu strukturiert werden musste.

Auf wohlgeformte und ausgearbeitete Projektdateien oder Diagramme (zB. UML) wurde hierbei verzichtet, das meiste ist in handschriftlicher oder skizzenartiger Form geschehen.

### 2.2 Entwicklungsphase

Wir haben uns dafür entschieden direkt von Beginn an der Entwicklungsphase mit dem QT Creator zu arbeiten um uns direkt an diese Software zu gewöhnen und auch auf einige Annehmlichkeiten dieser zurückgreifen zu können. Als Beispiel wären hier die automatisch generierten Makefiles mithilfe von qmake und der Projektdatei zu erwähnen. Auch haben wir uns dagegen entschieden, erst einmal eine reine Konsolenanwendung zu erstellen. Stattdessen wurde zuerst eine einfaches Fenster mit einer Zeichenfläche erstellt, hinter welchem nur ein Mouselistener sowie eine simple Zeichenmethode stand das Spielfeld darzustellen. Nachdem der Mausklick einwandfrei erkannt wurde, haben wir begonnen die eigentlichen Models zu entwickeln (Spieler, Spielfeld) nur um dann später den Controller und die anderen Klassen zu ergänzen.

Schlussendlich, nach dem Erweitern und Aufschönen der grafischen Oberfläche, haben wir auch abschließend die Unittests eingebunden, mit welchen wir unsere Modelklassen und die Algorithmen des Controllers testen. Hier mussten wir unter anderem das Projekt neu strukturieren, da es Probleme gab die Unittests selbst in das eigentliche Projekt einzubinden. Stattdessen haben wir uns dafür entschieden, das Projekt auch noch als Library zu kompilieren, auf welche später die Unittests, welche in ein neues Unterprojekt geschoben wurden, zugreifen konnte.

#### 2.2.1 Die Datenbank – SQLite

Als Teil der Anforderung war es Aufgabe die Punktzahlen als Highscore in einer kleinen Datenbank zu speichern. Dies wurde mit SQLite realisiert und das Programm selbst erstellt diese Datei, wenn nicht vorhanden, bei erstmaligem Versuch einen Spieler einzupflegen. Hierbei ist es zu keinen nennenswerten Problemen gekommen.

#### 2.2.2 Die künstliche Intelligenz

Bei der künstlichen Intelligenz sind wir dann doch auf gewisse Probleme gestoßen. Schon von Anfang an war uns klar, dass es das Beste wäre alle möglichen nachfolgenden Züge beider Spieler in einer Baumstruktur zu speichern und diese dann später durch den Algorithmus auswerten zu

lassen. Da es aber keine vorgefertigte Baumstruktur unter C++ (wie zB. Liste oder Vektor) gibt, haben wir uns dafür entschieden auf eine externe, fertige Lösung zurückzugreifen anstatt die Struktur komplett selbst zu entwickeln. Genutzt haben wir Bibliothek „tree.hh“ (<http://tree.phisci.com/>). Mithilfe dieser Bibliothek war es uns möglich den erforderlichen Baum zu erstellen, dessen Wurzel das aktuelle Spielfeld beinhaltet und die darauf folgenden Blätter alle möglichen Spielzüge der beiden Spieler bis auf eine vorher definierte Tiefe. Um eine gute Intelligenz zu ermöglichen wäre es natürlich ratsam eine größere Zahl als Tiefe zu nehmen, doch haben wir uns selbst bei der Schwierigkeit „hard“ für eine Tiefe von 5 entschieden, damit die Wartezeit nicht zu lang wird. Testweise wurde bei einem 4x4 Spielfeld einmal mit einer Tiefe von 15 zu arbeiten, damit der Computer bis zu jedem möglichen Spielende die Züge vorhersehen und auswerten kann. Zudem Wird bei der höchsten Schwierigkeitsstufe auch gesehen, ob Eckfelder als mögliche Züge gesehen werde und diese direkt gesetzt ohne das die Funktion die gesamte Auswertung durchläuft. Bei der Schwierigkeit „easy“ wird hingegen nur eine Liste mit allen möglichen Zügen erstellt und ein zufälliger aus dieser Liste genommen. Für die Auswertung haben wir versucht die Idee des Minimax-Algorithmus umzusetzen, wobei wir erst einmal bis zu jedem Endblatt gegangen sind, dort dann die Differenz der Steine beider Spieler berechnet haben und dementsprechend dann eine Gewinnchance ermittelt haben (-1 bei Niederlage, 0 bei Unentschieden, 1 bei Gewinn) und dann hoch iteriert versucht haben je nachdem welcher Spieler an der Reihe ist den Entscheidungsprozess des Minimax-Algorithmus umzusetzen.

### **2.2.3 Git als Verwaltungshilfe**

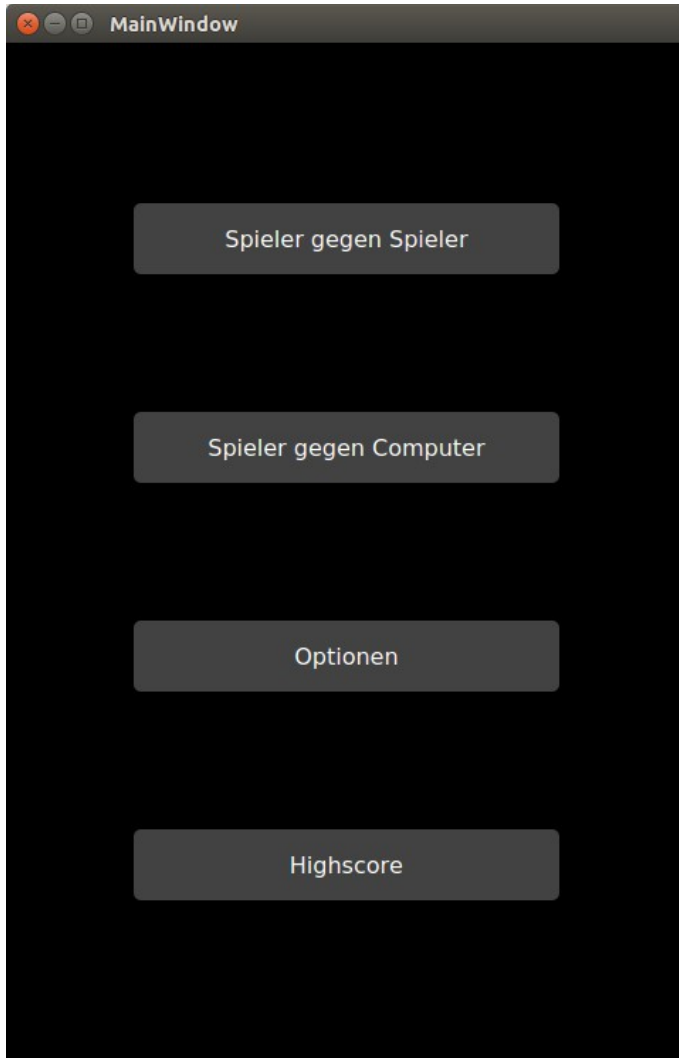
Wie in den Anforderungen gefordert haben wir Git benutzt, genauer gesagt Github, um die Software zu entwickeln und jede Änderung zu committen und zu pushen, damit man später eine gute Möglichkeit zur Versionierung hat. Dennoch war es praktischer erste Änderungen erst einmal lokal zu tätigen und mithilfe anderer Kommunikationswege (zB. Dropbox) auszutauschen. Daher wird in der Historie von Github auch nur ein Projektbearbeiter angezeigt.

### **2.2.4 Nicht bearbeitet Features**

Von uns nicht bearbeitet wurde das Netzwerkspiel, also jegliche Anforderung, die mit dem Netzwerk oder der Kommunikation über jenes zu tun haben. Zudem haben wir das Projekt oder einzelne Klassen nicht als Thread umgesetzt, weswegen es theoretisch nicht möglich ist länger dauernde Bearbeitungsschritte abubrechen. Stattdessen haben wir lieber den Ansatz gewählt, die Algorithmen so einfach wie möglich zu halten, dass besagte Bearbeitungsschritte eben nicht länger brauchen. Zu guter Letzt wurde ebenfalls das Speichern und Laden des Spiels nicht eingeplant und ebenso nicht realisiert.

## 3. Die Applikation

### 3.1 Startbildschirm

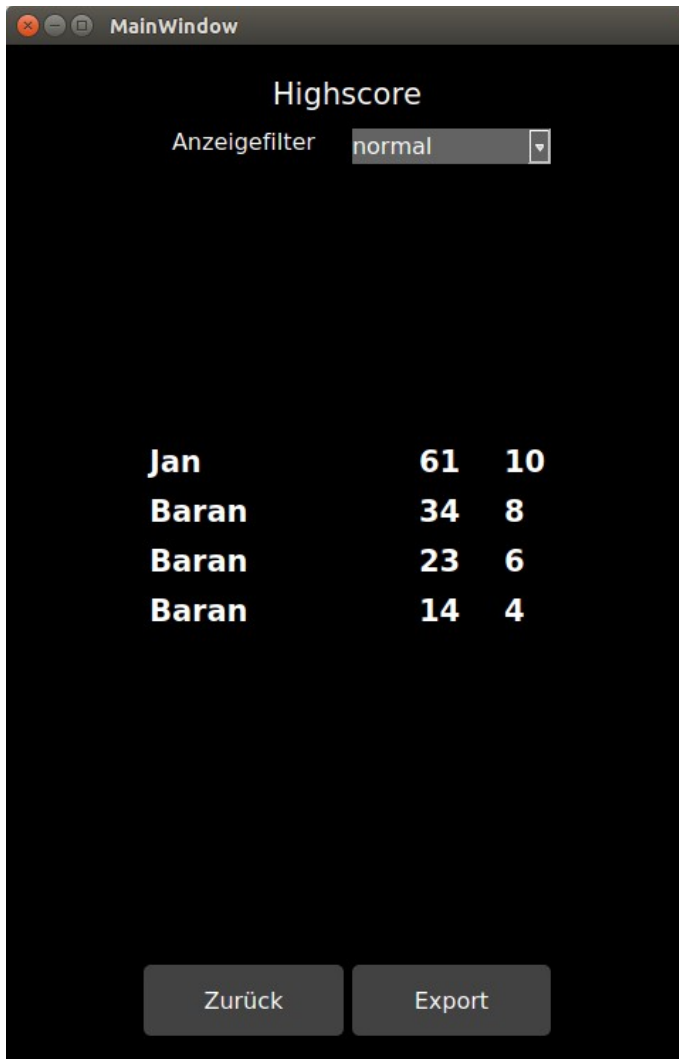


Der Startbildschirm selbst bietet Zugriff auf die wichtigsten Menüpunkte der Anwendung. Von hier kann der Benutzer ein neues Spiel (Spieler gegen Spieler oder Spieler gegen Computer) starten, auf die Optionen zugreifen (wo er zum Beispiel die Sprache ändern kann) oder die Highscore einsehen.

Der Unterpunkt Credits wurde eingespart, da man diese auch im Quellcode einsehen kann.

Zudem wurde sich bei der Gestaltung der Applikation sehr an dem Erscheinungsbild geläufiger Apps orientiert um der Anwendung möglichst eine plattformunabhängige Darstellung zu ermöglichen.

## 3.2 Highscore

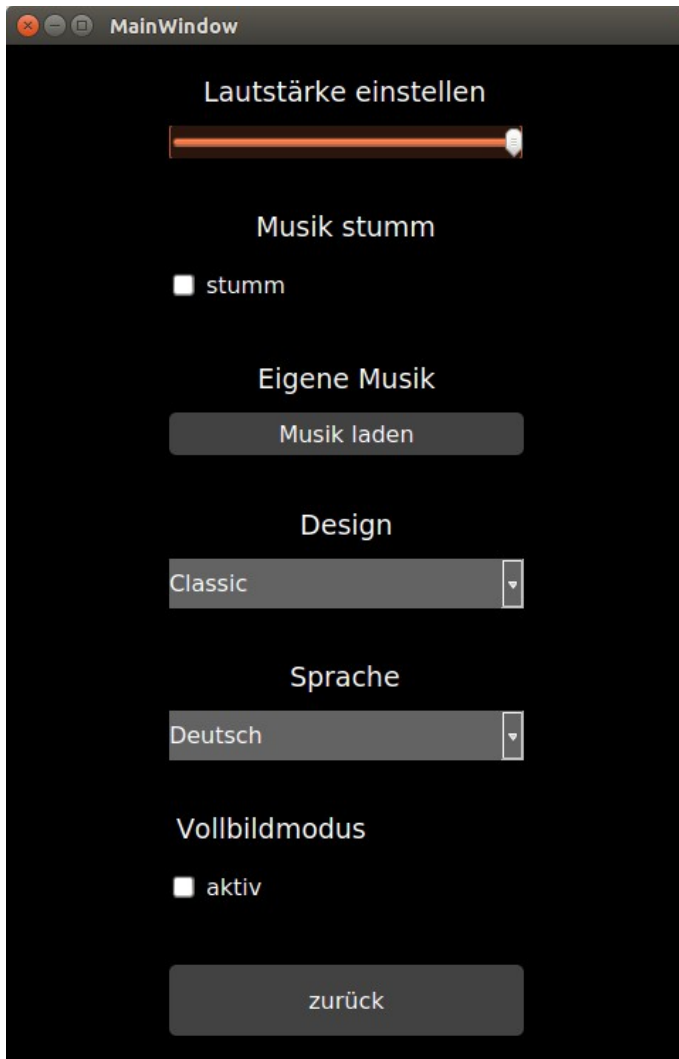


Im Screen des Highscores kann man sich die jeweiligen Punktzahlen der Sieger anzeigen lassen. In der normalen Anzeige ist die Auflistung wie folgt: Spieler – Punktzahl – Spielfeldgröße.

Mithilfe des Dropdown Menüs kann der Nutzer nun die Highscore nach Spielfeldgröße ordnen lassen, was die Anzeige vereinfacht und zudem sinnvoller erscheint, da man höhere Punktzahlen mit größerer Spielfeldgröße erreicht.

Zudem haben wir uns entschieden nur die ersten 12 Einträge der Datenbank anzeigen zu lassen, damit es keinen Umbruch und damit keine Scrollleiste gibt. Dies könnte man je nach Anforderung oder Wunsch noch ändern.

### 3.3 Optionen



Das Optionsmenü ermöglicht es dem Nutzer die Lautstärke der Hintergrundmusik einzustellen, die Musik sogar komplett stumm zu schalten, eigene Musik zu laden, sowie das Design und die Sprache zu ändern.

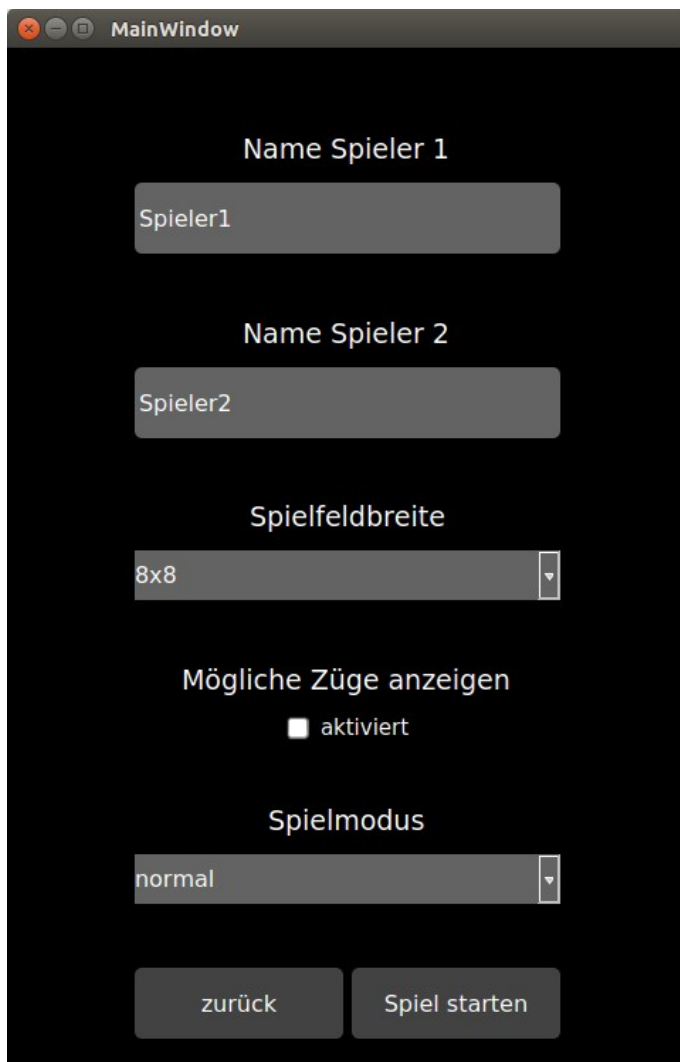
Ebenso ist es hier möglich in den Vollbildmodus zu schalten und diesen auch wieder zu deaktivieren.

Es gilt aber zu beachten, dass die Sprachumstellung nur im normalen Optionsmenü funktioniert, nicht aber im „Im-Spiel-Optionsmenü“.

Seltsamerweise hat die grafische Zeichenfläche einen Fehler, wenn man dort die Labels in jener ändern würde.



## 3.4 Spieler gegen Spieler



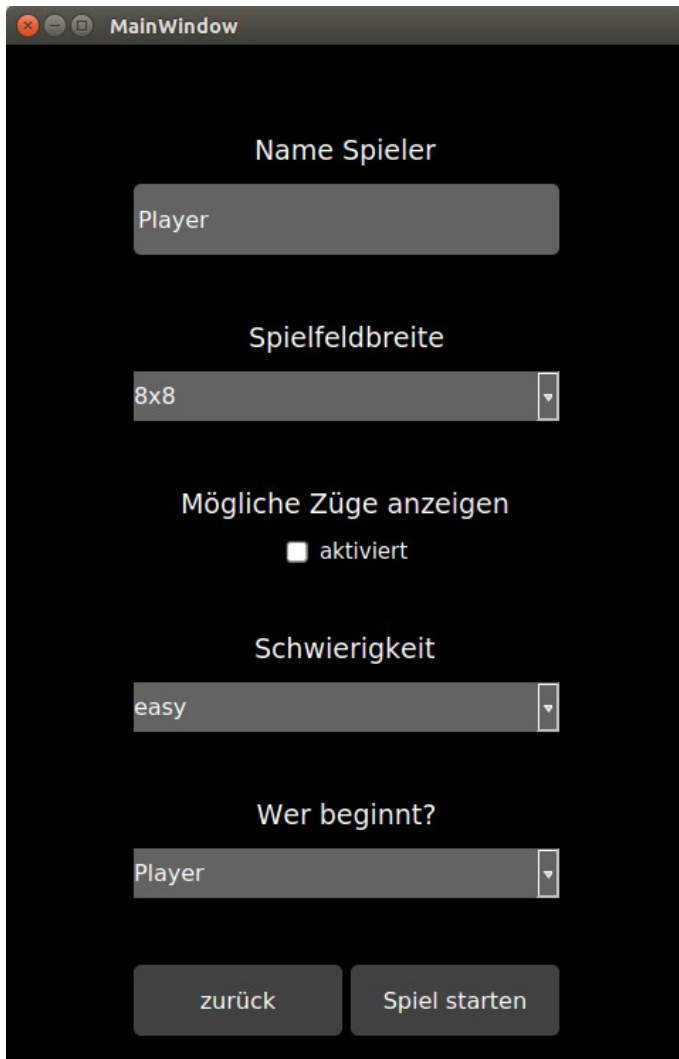
The screenshot shows a dark-themed window titled 'MainWindow'. It contains the following elements:

- Name Spieler 1:** A text input field containing 'Spieler1'.
- Name Spieler 2:** A text input field containing 'Spieler2'.
- Spielfeldbreite:** A dropdown menu showing '8x8'.
- Mögliche Züge anzeigen:** A checkbox labeled 'aktiviert' which is currently checked.
- Spielmodus:** A dropdown menu showing 'normal'.
- Buttons:** Two buttons at the bottom, 'zurück' and 'Spiel starten'.

Im Menü „Spieler gegen Spieler“ haben beide Spieler die Möglichkeit ihre Namen zu ändern. Wenn der Name allerdings zu klein ist (weniger als 3 Zeichen) wird automatisch „SpielerX“ als Name genommen.

Spielfeldgröße ist genauso einstellbar wie die Anzeige möglicher Züge und der Spielmodus.

## 3.5 Spieler gegen Computers



The screenshot shows a window titled 'MainWindow' with a dark background. It contains several settings for a game against a computer:

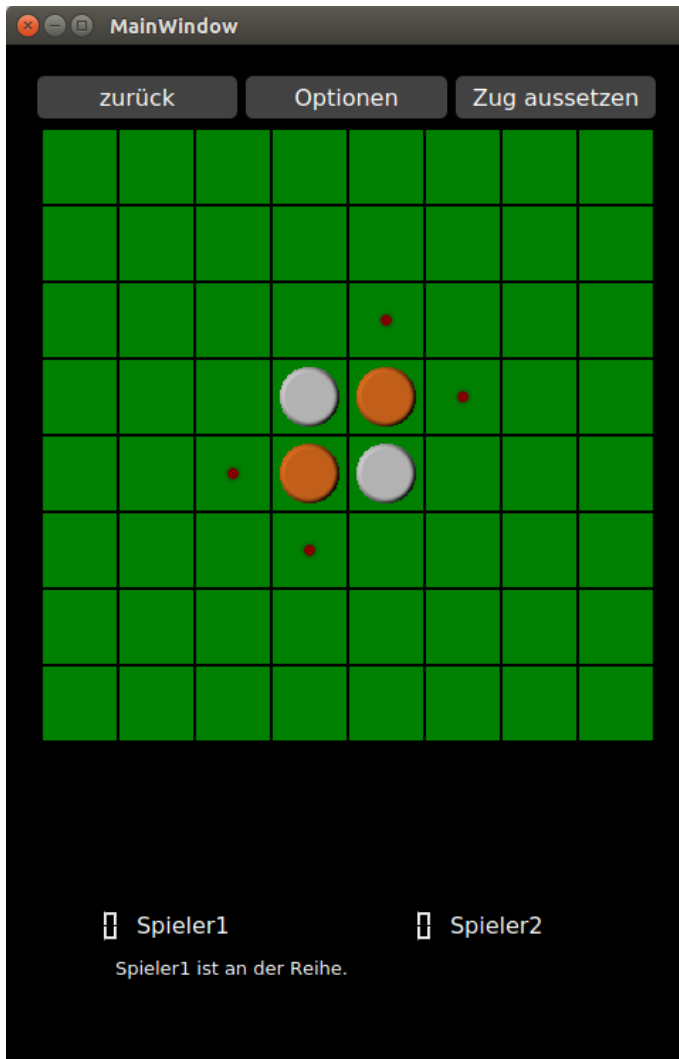
- Name Spieler:** A text input field containing 'Player'.
- Spielfeldbreite:** A dropdown menu showing '8x8'.
- Mögliche Züge anzeigen:** A checkbox labeled 'aktiviert' which is checked.
- Schwierigkeit:** A dropdown menu showing 'easy'.
- Wer beginnt?:** A dropdown menu showing 'Player'.
- Buttons:** At the bottom, there are two buttons: 'zurück' (back) and 'Spiel starten' (start game).

Das Menü „Spieler gegen Computer“ ist ähnlich wie das Menü „Spieler gegen Spieler“ aufgebaut.

Hier kann der Spieler seinen Namen angeben, sowie die Größe des Spielfeldes. Auch die Anzeige möglicher Züge ist gleich.

Unterschiedlich sind die Optionen um die Schwierigkeitsstufe des Computers einzustellen und wer beginnen soll ob der Spieler selbst oder der Computer.

## 3.6 Spielfeld



Das Spielfeld selbst ist relativ einfach aufgebaut, oben gibt es diverse Buttons um dem Spieler benötigte Funktionalitäten zu ermöglichen.

Mittig ist das Spielfeld gelagert, welches bei einem Resize-Event neu berechnet und gezeichnet wird.

Unten sind die wichtigen Ausgabeinformationen für den Nutzer zu finden.

## 4. Testing

### 4.1 Unittests

Wie bereits beschrieben musste aufgrund der Einbindung der Unittests die Struktur des gesamten Projekts noch einmal geändert werden. Zuvor als normales Qt-Projekt geplant, wurde der Beleg in ein Multiprojekt erstellt, in welchem mehrere Unterprojekte Platz finden können. So ist der Beleg selbst ein Projekt, die Erstellung der Library aus jenem eines und die Unittests ein drittes.

Die Entscheidung das Projekt als Bibliothek bereit zu stellen haben wir getroffen, da es uns sonst nicht möglich war die Unittests ordentlich auszuführen und es ebenso sinnvoll sein soll die Tests getrennt vom eigentlichen Projekt zu halten.

Die Unittests werden mit Erstellen des Projekts ausgeführt und haben bei uns keinerlei Fehler verursacht.

### 4.2 Multiplattform Testing

Laut den Anforderungen wurde auch verlangt, das Projekt nicht nur unter Ubuntu zu testen und für dieses System Buildartefakte zu erstellen sondern die Anwendung auch unter Windows, mithilfe von Docker und Android, mithilfe von Android SDK(NDK) zu testen.

Hierfür haben wir uns in die wichtigen Schritte eingelese und haben den Erfolg beider Methoden dokumentiert, in schriftlicher Form und als Screenshots und diese in den Ordner „Dokumentation“ gepackt.

### 4.3 Memory Tests

Hierfür haben wir das Tool Valgrind benutzt, welches auch schon direkt durch QT zur Verfügung gestellt wird. Auf der Ubuntu Maschine musste man einzig das Package „valgrind“ installieren, damit dies funktionierte.

Zuerst hatten wir einige Speicherlecks, da wir vergessen hatten, die Zeiger im Dekonstruktor aufzuräumen, doch nachdem dies nachgeholt wurde, kam es zu keinen Fehlern mehr.

Einzig ein angebliches Speicherleck bei der Variable „applause“ im Controller des Feldes war bei den letzten Tests zu finden, obwohl der Zeiger im Dekonstruktor aufgeräumt wird.

## 5. Anmerkungen

### 5.1 Bugs

Es kann zu Fehlern in der Darstellung kommen. Generell scheint die GUI von Qt etwas fehleranfällig, siehe hierzu den Bug, dass das Ändern des Text eines Labels die Zerstörung des gesamten Layouts nach sich ziehen kann.

Zudem ist es ebenso seltsam, dass die Schriftfarbe einer Combobox nur richtig angezeigt wird, wenn man in ihrem Stylesheet ein „padding“ setzt, egal ob dieses 0 Pixel hat oder nicht.

Hin und wieder trat der Fehler auf, dass der Siegestext nicht richtig platziert angezeigt wurde, da dies aber nicht reproduzierbar war, wussten wir nicht was genau diesen Fehler hervor rief.

### 5.2 Letzte Wort

Der Beleg allgemein war recht interessant, wenn gleich auch Qt nicht gerade Einsteigerfreundlich war und der Umfang des Beleges (die Anzahl an Anforderungen) vielleicht ein wenig überdimensioniert erschien.

Zudem war unsere Lösung mit dem gemeinsamen Arbeiten und das anschließende Pushen über einen Account auf Github auch nicht wirklich optimal, auch wenn es gut umzusetzen war.