Road to Reversi

**23.05.2019**

Jonas Pfingst, Joshua Purwin, Sebastian Greger

Die Spannende Reise von 3 jungen Programmieren

Inhalt

[Ausgangssituation 2](#_Toc9496288)

[Projektziel 2](#_Toc9496289)

[Projektumfeld 3](#_Toc9496290)

[Ressourcen und Ablaufplanung 3](#_Toc9496291)

[Personalplanung 3](#_Toc9496292)

[Terminplanung 4](#_Toc9496293)

[Kostenplanung 4](#_Toc9496294)

[Durchführung und Auftragsbearbeitung 5](#_Toc9496295)

[Spielfeld 5](#_Toc9496296)

[Spiellogik 6](#_Toc9496297)

[Projektergebnis 6](#_Toc9496298)

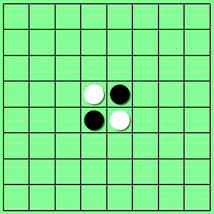
[Anlagen 7](#_Toc9496299)

[Kundendokumentation 7](#_Toc9496300)

# Ausgangssituation

## Projektziel

Zum Ende des ersten Lehrjahres haben wir uns zum Abschluss im Lernfeld 6 nochmal einem Projekt gewidmet. Das Ziel dieses Projektes war es ein funktionierendes Reversi Spiel in C zu Programmieren. Bei Reversi handelt es sich es sich um ein zwei-Spieler Strategiespiel welches auf einem 8 mal 8 großen Spielfeld gespielt wird. Zu Beginn des Spieles liegen vier Steine in vorgegebener Anordnung auf dem Spielfeld.



https://learnplaywin.net/reversi-rules/

Nun platzieren beide Spieler abwechselnd einen Stein ihrer Farbe auf dem Spielfeld und versuchen gegnerische Steine mit den eigenen Steinen einzuschließen. Befindet sich ein Stein zwischen gegnerischen Steinen wird dieser umgedreht und gehört nun zu den gegnerischen Steinen. Gewinner des Spiels ist derjenige, der am Ende die meisten Steine sein Eigen nennen kann. Wichtig zusagen ist noch, dass Steine nicht einfach willkürlich platzier werden dürfen, sondern immer horizontal, vertikal oder diagonal an ein bereits belegtes Feld angrenzen müssen. .

Unsere Mission war es nun, dieses Spiel zu Programmieren. Dabei sollte jeder Spieler abwechselnd an der Reihe sein, einen Stein zu platzieren. Der aktuelle Punktestand sowie welcher Spieler aktuell am Zug ist soll dem Benutzer jederzeit angezeigt werden.

Außerdem soll es einen Timer geben, der die Laufzeit des aktuellen Spiels anzeigt. Zudem soll es jederzeit möglich sein das Spiel zu pausieren und den Timer anzuhalten. Für den Benutzer soll es unmöglich sein Falscheingaben zu machen, zudem soll der Spieler eine Runde passen können, wenn für ihn kein Zug möglich ist. Das Spie soll automatisch beendet werden wenn kein weiterer Zug mehr möglich ist.

Der Spieler soll das Spiel jederzeit pausieren und anhalten können. Darüber hinaus soll es möglich sein das aktuelle Spiel abzuspeichern. Dieser Spielstand kann dann später wieder geladen werden und das unvollendete Spiel kann zu Ende gebracht werden.

## Projektumfeld

Wie bereits oben erwähnt ist die vorgegebene Programmiersprache für unser Projekt C. Als Entwicklungsumgebung ist CodeBlocks vorgegeben, als Compiler der MinGW-Compiler.

Des Weiteren soll das Programm sinnvoll gegliedert sein und der Quellcode sollte in einzelne Funktionen und Dateien ausgelagert werden, sodass das Programm hinterher auch durch den Quellcode nachvollziehbar ist und Änderungen einfacher zu implementieren sind. Hierzu soll auch jede einzelne Funktion einen Kommentar haben, welcher beschreibt was die Funktion genau macht, welche Übergabeparameter sie besitzt, sowie was für einen Wert/Werte sie gegebenfalls zurückgibt.

Die Benutzung von zusätzlichen Bibliotheken ist erlaubt. Die letzte Vorgabe ist noch, dass der Compiler beim Kompilieren keinerlei Warnungen ausspuckt.

# Ressourcen und Ablaufplanung

## Personalplanung

Um möglichst möglichst effizient voranzukommen haben wir uns in der ersten Stunde nach Projektbeginn erstmal zusammengesetzt und gemeinsam überlegt was überhaupt zu tun ist. Hier haben wir uns erstmal eine grundsätzliche Struktur erstellt nach welcher wir vorgehen. Wir haben uns entschieden das Programm in die einzelnen Dateien board, gamelogic, input und output aufzuteilen. Jonas wurde damit betraut sich ausschließlich um die gamelogic, also die eigentliche Spiellogik, zu kümmern. Er sollte sich hierzu genau mit den Regeln des Spiels vertraut machen und Algorithmen für die einzelnen Spielvorgaben entwickeln. Joshua und Sebastian sollten sich um jegliches „Drumherum“ kümmern. Dies beinhaltet mehrere Aspekte:

* Den Aufbau des Spielfeldes
* Der Ausgabe inklusive folgender Aspekte:
  + Das Spielfeld selber
  + Einem Timer mit aktueller Spielstand
  + Der aktuelle Punktestand
  + Welcher Spieler ist aktuell am Zug
* Die Eingabemöglichkeiten für den User:
  + Dem Spieler die Möglichkeit geben einen Stein auf dem Spielfeld zu platzieren
  + In der Lage sein das Spiel zu pausieren
  + Das Spiel jederzeit beenden zu können

## Terminplanung

Bei der Terminplanung hingegen haben wir versucht relativ flexibel zu bleiben. Wir haben beschlossen uns am Anfang jeder Unterrichtsstunde kurz zusammenzusetzen um den aktuellen Zwischenstand sowie die noch zu erledigenden Aufgaben zu besprechen. Daraufhin hat jeder an seinem aktuellen Thema gearbeitet und dieses gegebenfalls mit nach Hause genommen um dort noch daran weiterzuarbeiten. Zudem haben wir eine gemeinsame Whatsapp-Gruppe erstellt um uns jederzeit austauschen zu können oder den anderen Gruppenmitgliedern bei offenen Fragen jederzeit zur Hilfe kommen zu können. So standen wir permanent im gemeinsamen Austausch und konnten auch zu jeder Zeit neue Ideen oder Anregungen diskutieren.

## Kostenplanung

|  |  |  |
| --- | --- | --- |
| **Datum** | **Aufgewendete Zeit in Stunden** | |
| 29.04.2019 | 4,5 | |
| 07.05.2019 | 9 | |
| 12.05.2019 | 1 | |
| 13.05.2019 | 4,5 | |
| 21.05.2019 | 6 | |
| 22.05.2019 | 15 | |
| 23.05.2019 | 8 | |
| **Kosten pro Stunde** | | 60€ | |
| **Gesamtkosten** | | 2880€ | |

# Durchführung und Auftragsbearbeitung

## Spielfeld

Zu Beginn des Projektes haben wir uns verschiedene Möglichkeiten angeschaut, dass Spielfeld dem Benutzer darzustellen. Relativ schnell haben wir uns jedoch dazu entschieden, keine zusätzliche Bibliothek zu benutzen, sondern das gesamte Spielerlebnis auf der Konsole darzustellen. Diese Entscheidung beruhte vor allem auf dem „Keep it simple“ Prinzip. Anstatt uns noch mit einer neuen Bibliothek zu beschäftigen haben wir uns dazu entschlossen, mit dem zu Arbeiten, mit dem wir bereits vertraut waren und was wir schon aus dem Unterricht kannten. Unser Ziel war es die ganze Sache möglichst einfach zu halten.

Hierzu haben wir das Spielfeld als ein zwei-Dimensionales Array definiert. Das Array fast ein 8x8 Spielfeld. Jeder Wert im Array kann entweder null, eins oder zwei sein. Bei null ist diese Position im Spielfeld leer, bei eins liegt hier ein Stein von Spieler eins, bei zwei ein Stein von Spieler zwei..

Um alles möglichst kompakt zu halten und immer auf alle relevanten Daten zugreifen zu können, haben wir beschlossen alle für das Spielfeld und das Drumherum wichtige Daten in ein gemeinsames Struct zu packen. Dieses Struct beinhaltet neben dem oben erwähnten zwei-Dimensionalem Array auch die folgen Werte:

* Welcher Spieler aktuell am Zug ist
* Die aktuelle Punktzahl beider Spieler
* Die Startzeit des Spiels
* Die Laufzeit des aktuellen Spieles
* Den Status ob das Spiel aktuell pausiert ist oder nicht
* Die Dauer der Pause
* Die aktuell ausgewählte Reihe und Spalte

Um das Spielfeld nun darzustellen, wird dieses jedes Mal geprintet, wenn sich entweder das aktuell ausgewählte Spielfeld ändert, oder der Timer nicht mehr aktuell ist. Das Spielfeld wird also jedes Mal ausgegeben wenn entweder eine Sekunde vorbei ist, oder der Benutzer eine Eingabe getätigt hat.

Bei der Eingabe war unser erster Ansatz dem Spieler die Möglichkeit zu geben, zwei Koordinaten vom Spielfeld einzutippen und diese dann zu verarbeiten. Dies zeigte sich aber als extrem ungünstig, da so die Ausgabe so lange verzögert war, bis der Benutzer tatsächlich eine Eingabe tätigte. Also entschlossen wir uns dazu, die Eingabe mithilfe der Navigation der Pfeiltasten zu realisieren. Dies ist ein wesentlich angenehmeres Spielerlebnis für den Benutzer und gleichzeitig auch wesentlich besser für die Ausgabe.

Um den Timer zur Anzeige der aktuellen Spielzeit haben wir die Bibliothek „time.h“ importiert. Wir speichern den aktuellen Zeitpunkt wenn das Spiel beginnt in unserem Struct ab. Daraufhin berechnen wir einfach nur die Differenz zwischen dem Startzeitpunkt und dem aktuellen Zeitpunkt. Für den Fall , dass das Spiel pausiert wird, messen wir die Zeit wie lange die Spielpause dauert und ziehen diese dann bei unserer Berechnung von der aktuellen Spiellaufzeit ab.

Das nächste was wir uns vorgenommen hatten war, dem Benutzer die Möglichkeit zu geben das aktuelle Spiel zu speichern, um diese zu einem späteren Zeitpunkt fortsetzen zu können. Da wir alle relevanten Spieldaten wie bereits oben erwähnt in einem Struct speichern, hat sich das Abspeichern des aktuellen Spielstandes als relativ einfach erwiesen. Wir schreiben hierzu einfach alle aktuellen Werte aus unserem Struct in eine Textdatei. Wenn wir ein Spiel dann wieder starten wollen, können wir einfach die Textdatei auslesen und die Werte wieder in unser Struct übernehmen und das Spiel fortsetzen.

## Spiellogik

# Projektergebnis

Nach knapp drei Wochen kommt unser Projekt zu einem Ende. Das Projekt kann hierbei definitiv als erfolgreich abgeschlossen bezeichnet werden. Wir haben es geschafft das Strategiespiel in C umzusetzen. Das Spiel ist mit zwei Spielern die sich abwechseln spielbar. Zu Beginn kommt der Benutzer in ein Hauptmenü indem er die Möglichkeit hat, dass Programm zu beenden oder das Spiel zu starten. Wird das Spiel gestartet startet der Timer, der permanent die Laufzeit des aktuellen Spieles anzeigt. Der Spieler/die Spieler erhalten aber nicht nur Information über die aktuelle Laufzeit des Spiels, sondern Information darüber, welcher Spieler im Moment am Zug ist, sowie die aktuelle Anzahl an Steinen die jeder Spieler sein Eigen nennen darf.

Desweiteren ist der Spieler in der Lage das Spiel jederzeit durch das betätigen der „P“ Taste auf der Tastatur anzuhalten. Hier wird das Spiel angehalten und solange pausiert, bis die „P“ Taste erneut betätigt wird. Sollte plötzlich etwas wichtiger als eine Partie Reversi dazwischen kommen, ist es dem aktuellen Spieler auch möglich, den aktuellen Spielstand abzuspeichern. Die aktuelle Partie kann also gespeichert werden und an einem späteren Zeitpunkt fortgesetzt. Das aktuelle Spielfeld und die aktuelle Spielzeit bleiben dabei erhalten.

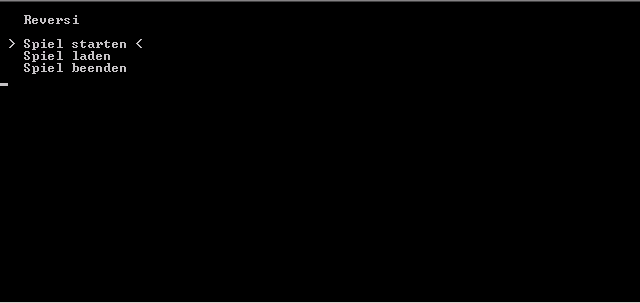
Für den Benutzer ist es unmöglich irgendwelche Falscheingaben, das Programm reagiert in keiner Weise auf irgendwelche falschen Tastatureingaben und wir konnten es auch nach ausführlichem Testen nicht einmal zu einem Absturz bringen. Darüber hinaus ist es für den Spieler auch unmöglich, einen Spielstein an einer nach den Regeln ungültigen Position abzulegen, da auch diese Eingaben vorher abgefangen und nicht vom Programm zugelassen werden.

Zusammengefasst lässt sich also sagen, dass wir die Ziele die wir uns zu Anfang des Projektes gesetzt haben, erreicht haben. Wir haben es geschafft, den Quellcode und Programm selber so simpel und übersichtlich wie möglich zu halten. Wir haben uns an alle Vorgaben gehalten und sind zu einem Ergebnis gekommen, mit dem wir in Anbetracht der zur Verfügung stehenden Zeit sehr zufrieden sind.

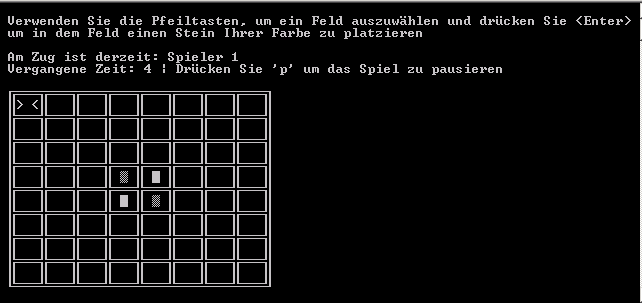
# Anlagen

## Kundendokumentation

Beim öffnen des Programmes kommt der Benutzer zuerst einmal in ein Hauptmenü. Hier hat er die Optionen ein Spiel zu starten, einen Spielstand von einem älteren Spiel zu laden, oder das Programm zu beenden.



Das navigieren ist mit Hilfe der Pfeiltasten möglich, mit der Enter Taste kann eine Auswahl getroffen werden. Wird „Programm beenden“ ausgewählt, wird die Anwendung beendet. Wird „Spielstand laden“ ausgewählt, wird der Benutzer aufgefordert, einen Pfad zu einem zuvor gespeicherten Spiel anzugeben. Über diesen Menüpunkt ist es also möglich ein früheres Spiel fortzusetzen. Mit „Spiel starten“ wird das eigentliche Spiel gestartet, dem Benutzer wird das Spielfeld angezeigt.



Das Spiel hat begonnen. Von hier an ist jeder Spieler abwechselnd in der Lage mit Enter- und Pfeiltasten seine Steine zu platzieren und Reversi zu spielen. Durch betätigen der „P“ Taste wird das aktuelle Spiel pausiert und der Spieler hat die Möglichkeiten, das Spiel fortzusetzen, das Spiel zu speichern, oder ins Hauptmenü zurückzukehren. Viel Spaß!