BBS Brinkstraße

Dokumentation

**Reversi**



Abbildung 1: Spielbrett von Reversi, Quelle: https://cdn3.volusion.com/artgw.hyvvw/v/vspfiles/photos/CB05120-4.jpg?1534152438

Verfasser: Lukas Gutknecht und Lukas Alte-Bornholt

Fach: Lernfeld 6 – Programmieren

Betreuer: Herr Wichmann

Abgabetermin: 23.05.2019

inhaltsverzeichnis

[1. Ausgangssituation 1](#_Toc9540766)

[1.1. Projektziele und Teilaufgaben 1](#_Toc9540767)

[1.2. Projektumfeld 1](#_Toc9540768)

[1.3. Kundenanforderungen 1](#_Toc9540769)

[1.3.1. Benutzerperspektive 1](#_Toc9540770)

[1.3.2. Spielerperspektive 1](#_Toc9540771)

[2. Ressourcen und Ablaufplanung 2](#_Toc9540772)

[2.1. Terminplanung, Ablaufplanung (Gantt-Diagramm) 2](#_Toc9540773)

[2.2. Kostenplanung 3](#_Toc9540774)

[3. Durchführung und Auftragsbearbeitung 3](#_Toc9540775)

[3.1. Prozessschritte, Vorgehensweise, Qualitätssicherung 3](#_Toc9540776)

[3.1.1. Vorgehensweise 3](#_Toc9540777)

[3.1.1.1. Allgemeine Vorgehensweise in der Programmierung 3](#_Toc9540778)

[3.1.1.2. Vorgehensweise im Projekt 3](#_Toc9540779)

[3.1.2. Prozessschritte 3](#_Toc9540780)

[3.1.2.1. Planung der Programmierung 3](#_Toc9540781)

[3.1.2.2. Durchführung der Programmierung 6](#_Toc9540782)

[3.1.3. Qualitätssicherung 10](#_Toc9540783)

[3.2. Abweichungen, Anpassungen, Entscheidungen 10](#_Toc9540784)

[3.2.1. Abweichungen 10](#_Toc9540785)

[3.2.1.1. Zentrale Header-datei aufsplitten 10](#_Toc9540786)

[3.2.1.2. Verwendung von einem zweiten Thread 11](#_Toc9540787)

[4. Projektergebnisse 11](#_Toc9540788)

[4.1. Soll-Ist-Vergleich, Qualitätskontrolle 11](#_Toc9540789)

[4.2. Abweichungen, Anpassungen 11](#_Toc9540790)

[5. Anlagen 12](#_Toc9540791)

[5.1. Kundendokumentation 12](#_Toc9540792)

[5.1.1. Allgemeines zum Spiel 12](#_Toc9540793)

[5.1.2. Spielablauf 12](#_Toc9540794)

[5.1.3. Ziel des Spiels 12](#_Toc9540795)

[5.2. Abbildungsverzeichnis 13](#_Toc9540796)

[5.3. Tabellenverzeichnis 13](#_Toc9540797)

# Ausgangssituation

## Projektziele und Teilaufgaben

Das Ziel in diesem Projekt ist, das strategische Brettspiel Reversi als Konsolenanwendung zu programmieren.

## Projektumfeld

Als Programmiersprache ist C vorgegeben sowie als Entwicklungsumgebung Codeblocks.

## Kundenanforderungen

### Benutzerperspektive

* Es wird mir ein 8 mal 8 Feld angezeigt auf dem Reversi gespielt werden kann
* Es ist mir unmöglich eine falsche Eingabe zu machen und das Spiel womöglich zum Absturz zu bringen
* Es wird mir jederzeit, die bereits vergangene Zeit im Spiel angezeigt
* Ich kann jederzeit das Spiel pausieren, sodass die Zeit nicht weiterläuft und ich keine weiteren Eingaben tätigen kann
* Der aktuelle Stand des Spiels kann jederzeit abgespeichert werden und zu einem späteren Zeitpunkt wieder aus einer Datei geladen werden, um ein abgebrochenes Spiel wieder fortzusetzen
* Das Spiel wird automatisch beendet, sobald keine Züge mehr möglich sind und beide Spieler unmittelbar hintereinander gepasst haben

### Spielerperspektive

* Ich kann ein freies Feld mit meinem Symbol/meiner Farbe belegen
* Ich spiele abwechselnd mit einem anderen Spieler, um das Spiel voranzutreiben
* Wenn kein gültiger Zug möglich ist, dann muss ich passen
* Es wird mir nach jedem Zug die Anzahl der Spielsteine vom Gegner und von mir angezeigt, dies ist der Score der jeweiligen Spieler
* Es ist mir nicht möglich entgegen der Spielregeln Steine zu setzen
* Nach jedem Zug werden die umschlossenen Spielsteine des Gegners umgedreht

# Ressourcen und Ablaufplanung

## Terminplanung, Ablaufplanung (Gantt-Diagramm)

Die folgende Tabelle listet alle Schritte auf, die während der Projektphase durchgeführt wurden. Die Dauer der Stunden setzt sich zusammen aus Recherche, Problemlösung und der Programmierarbeit an sich. In der Spalte findet man entweder ein LG oder LAB. Dies sind die Initialen der Projektteilnehmer: Lukas Alte-Bornholt = LAB, Lukas Gutknecht = LG.

|  |  |  |
| --- | --- | --- |
| **Aufgabe** | **Dauer (Stunden)** | **Erledigt Von** |
| Auflistung der Projektgrundanforderungen an die Programmierung | 0.25 | LG |
| Auflistung der Projektextras (z.B. der Gegenspieler als KI) | 0.25 | LG |
| Erster Entwurf eines Spielfeldes | 1 | LG |
| Entwicklung und Programmierung eines Startmenüs + Einstellung des Modus + Auflistung der Regeln des Spiels | 4 | LAB |
| Programmierung der Grundsteuerung im Spiel (Bewegen und Setzen) | 1 | LAB |
| Score Anzeige (Anzahl der Steine der jeweiligen Spieler) | 1 | LAB |
| Zeitanzeige | 4 | LAB |
| Überprüfung auf welches Feld ein Stein gesetzt werden darf | 10 | LG |
| Überprüfung auf ein Ende des Spiels | 2 | LG |
| Spiel pausieren, sodass die Spielzeit anhält und man sich nicht mehr auf dem Feld bewegen und Steine setzen kann | 1 | LAB |
| Spiel zwischenspeichern in eine Datei und laden eines Spielstandes | 1 | LG |
| Darstellung der Zeit parallel zum Spielen (Threading) | 2 | LAB |
| Qualitätssicherung | Während der Entwicklung | LAB, LG |
| Schreiben der Dokumentation | 18 | LAB, LG |

Tabelle : Aufgabenverteilung

## Kostenplanung

|  |  |
| --- | --- |
| **Anzahl Stunden** | **Kosten in €** |
| 1 | 60 |
| Im Projekt: 45,5 | 2730 |

Tabelle : Kostenplanung

# Durchführung und Auftragsbearbeitung

## Prozessschritte, Vorgehensweise, Qualitätssicherung

### Vorgehensweise

#### Allgemeine Vorgehensweise in der Programmierung

1. Problemfeststellung
2. Code analysieren (Dauer je nach Komplexität)
3. Je nach Problem gibt es zwei Optionen:
   1. Code entfernen und durch anderen ersetzen
   2. Code erweitern
4. Abwägung möglicher Lösungen
5. Wählen der besten Lösung (nach Definition der Software-Qualität)
6. Durchführung der Problemlösung
7. Abschließende Analyse (evtl. Lösen von anderen Problemen, die durch die vorherigen Änderungen verursacht wurden)

#### Vorgehensweise im Projekt

1. Umsetzung sämtlicher Features, die als feste Vorgabe definiert wurden
2. Sich fragen was im Projekt, nicht unbedingt fester Bestandteil des Spiels Reversi, fehlt
3. Gemeinsame Überlegung mit dem Projektpartner über Sinnhaftigkeit eines Features
4. Entwicklung des Features
5. Dokumentierung in der Projektdokumentation

### Prozessschritte

#### Planung der Programmierung

##### Aussehen und Design

Angefangen wurde mit der Planung des Aussehens des Spiels. Hierbei hat man sich für ein schlichtes Design entschieden, dies bei einer Konsolenanwendung nicht im Vordergrund steht und man sich auf Wichtigeres als das Aussehen konzentrieren sollte, wenn dies nicht mit in die Bewertung mit einbezogen wird.

Das Spielfeld besteht aus 8 mal 8 Minus-Symbolen, welche bei einer Eingabe durch entweder „X“ für Spieler 1 oder „O“ für Spieler 2 ersetzt werden.

##### Aufbau

Damit man nicht direkt in das Spielgeschehen hineingeworfen wird, hat man sich für ein einfaches Anfangsmenü entschieden, welches sich durch Eingabe von Zahlen navigieren lässt. Folgende Optionen stehen einem nach Start des Programms zur Verfügung

* Unter dem Punkt „Spiel Starten“ findet man schließlich das 8 mal 8 Feld, eine Zeitanzeige, eine Score Anzeige und welcher Spieler gerade am Zug ist. Die Anzeigen werden nach jedem Zug aktualisiert. Hier wird das eigentliche Spiel gespielt, wo die Spieler abwechselnd Züge machen und versuchen, möglichst viele Steine auf dem Feld für sich zu gewinnen. Nachdem ein Spieler das Spiel gewonnen hat, kommt ein neuer Bildschirm, wo gesagt wird, welcher Spieler gewonnen hat und wie der finale Punktestand ist.
* Im nächsten Punkt lässt sich ein zuvor gespeicherter Spielstand laden, falls keine Datei zum Laden gefunden wurde, wird eine Fehlermeldung ausgegeben
* Unter dem Punkt Einstellungen gelangt man zu folgenden Einstellmöglichkeiten:
  + Die Auswahl zwischen zwei Spielmodi: Dem Spieler gegen Spieler- und dem Spieler gegen Computer-Modus (Im Nachhinein entfernt).
  + Regeln des Spiels, hier lassen sich kurz und knapp die Regeln des Spiels nachlesen, falls diese einem nicht vertraut sind.
  + Steuerung des Spiels, Tastenbelegungen um sich im Spiel zu bewegen, es zu pausieren oder zu speichern
  + Zurück, um zum Hauptmenü zurückzukehren
* Der letzte Hauptmenüpunkt beendet das Spiel

###### Aufbau des Startmenüs

Dieses Fenster wird direkt beim Start des Programms aufgerufen:

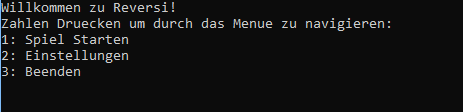


Abbildung 2: Startmenü des Programms

Abbildung : Startmenü des Programms

Wenn man Einstellungen gewählt hat, gelangt man in folgendes Fenster. Von hier aus lässt sich der Modus des Spiels einstellen (im Nachhinein aus dem Spiel entfernt), die Regeln von Reversi, sowie die Steuerung des Spiels nachlesen.

Aufbau des Spielfeldes

Sobald man ‚Spiel Starten‘ gewählt hat, erscheint dieses Fenster:

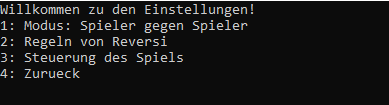


Abbildung : Einstellungen im Programm

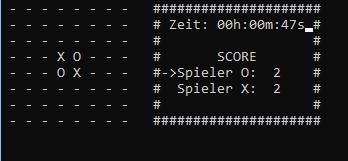


Abbildung : Spielfeld

Links abgebildet ist das Spielfeld, wo man mit den WASD-Tasten auf das gewünschte Feld navigieren kann und seinen Stein setzen kann. Auch ist zu sehen, dass jeweils zwei Steine von den Spielern diagonal zueinander bereits auf dem Spielfeld platziert sind. Dies ist die Startposition, in der jede Reversi Partie anfängt.

Auf der rechten Hälfte befindet sich die Spielanzeige. Sie beinhaltet die Zeitanzeige, die Scoreanzeige und welcher Spieler gerade am Zug ist (Durch den kleinen Pfeil sichtbar).

##### Steuerung

Die Steuerung wird im Allgemeinen durch die getch()-Funktion gelöst und lässt direkte Tastatureingaben zu, ohne ENTER drücken zu müssen. Die getch()-Funktion gibt im Gegensatz zur getche()-Funktion kein Echo auf der Konsole zurück und ist sehr hilfreich für das Spiel.

###### Im Menü

Man gelangt bei Programmstart in das Hauptmenü, welches durch Eingabe von vorgegebenen Zahlen sich steuern lässt. Sobald man eine aufgelistete Zahl gedrückt hat, wird direkt das nächste Menü angezeigt. Dieses Prinzip ist vergleichbar mit einem Klick auf einen Button und die schnellste Methode, um in einer Konsolenanwendung durch Menüs zu gehen oder Spiele zu spielen.

###### Im Spiel

Die Steuerung wurde ebenfalls durch eine Funktion gelöst, um es dem Benutzer so einfach wie möglich zu machen das Spiel zu spielen. Wenn man in dem Spiel ist, lässt sich der Cursor durch die W-A-S-D Tasten steuern.

* W für nach oben
* A für nach links
* S für nach unten
* D für nach rechts.

Die Tasten sind ähnlich wie die Pfeiltasten auf der Tastatur angeordnet und eine typische Belegung für viele andere Spiele.

Um nun seine Eingabe bestätigen zu können, muss der Cursor über der gewünschten Position blinken und die „y“-Taste (für yes) muss gedrückt werden. Dann ist die Eingabe getätigt und es erscheint auf der gewählten Position, je nach Spieler, ein „O“ oder „X“.

Mit der Taste P lässt sich das Spiel pausieren, sodass die Zeit nicht mehr weiterläuft und man keine Steine mehr setzen kann.

Mit der Taste O lässt sich das Spiel in eine Datei speichern, dass man es später wieder in genau diesem Zustand weiterspielen kann. Das schließt die Daten im Spielfeld, die vergangene Zeit und wer am Zug war, ein.

#### Durchführung der Programmierung

##### Struktur

Im gesamten Programm wird mit einem struct gearbeitet, welches durch einzelne Funktionen aufgerufen und bearbeitet wird. Dies erleichtert die Programmierung enorm und man hat alle wichtigen Daten in einer Variable. Zu den Daten des structs gehören:

* Gamefield (2d integer Array): speichert das gesamte Spielfeld:
  + Speichert Einsen oder Zweien, je nach Spielstein; 0 für unbesetzte Felder
* Time: (integer): speichert die gestoppte Zeit, die seit Spielstart gelaufen ist
  + Vergangene Zeit in Sekunden
* Turn (integer): speichert den jeweiligen Spieler, der gerade am Zug ist
  + 1 für Spieler Eins, 2 für Spieler Zwei
* Player1Passed, Player2Passed (integer): speichert, wie viele Male der jeweilige Spieler gepasst hat

Das Struct wird zu Beginn des Programms einmal weggespeichert und nach jedem Zug wird die Veränderung dazugeschrieben.

##### Startmenü

Das Startmenü des Programms ist dazu da, um als Benutzer Orientierung in das Programm zu bekommen. Die Zahlen vor den Menüpunkten sagen einem, welche Zahl man drücken muss, um die dahinter aufgeführte Funktion aufzurufen.

##### Spielfeld

Die Darstellung des Spielfeldes erfolgt getrennt von den Werten im 2d Array Gamefield. In diesem Array werden zahlen gespeichert, die für den jeweiligen Spieler stehen. In der Funktion ‚drawGameBoard‘, wo drin das Spielfeld gezeichnet wird, entscheidet eine Switch Abfrage, welches Symbol gemalt wird, abhängig von den Werten im Array.

##### Spielzug

Nachdem das Spiel gestartet wurde, werden Steine abwechselnd auf das Spielfeld gesetzt. Bei jedem Zug gibt es einen festen Ablauf, den man jedes Mal durchläuft:

1. Durch W-A-S-D-Tasten die gewünschte Position wählen
2. Mit der Y-Taste die Position einloggen
3. Die Stone-Management-Datei checkt, ob die gewählte Position valide ist.
4. Wenn diese valide ist, wird der Stein gesetzt.
5. Wenn nicht, wird eine Fehlernachricht mit einem entsprechenden Text ausgegeben.
6. Danach ist der andere Spieler dran und durchläuft denselben Ablauf.

###### Mögliche felder zum setzen Markieren

Während des gesamten Spielens von Reversi wird überprüft, auf welche Positionen man seinen Stein setzen kann. Anschließend werden diese Möglichkeiten an Zügen dem Spieler durch ein Fragezeichen kenntlich gemacht. Er muss so, falls er nicht so vertraut mit den Spielregeln ist, nicht jeden Zug ausprobieren, ob dieser möglich ist, sondern bekommt dies direkt angezeigt.

###### Darstellung der Zeit

Zur Darstellung der Zeit wird eine Variable alle 1000 Millisekunden mithilfe eines Delays hochgezählt. Wenn diese Variable 60 erreicht hat, ist eine Minute vergangen, also zählt sich die Variable für die Minute um einen hoch. Danach wird die Variable für die Sekunden wieder auf 0 gesetzt. Das Gleiche gilt, wenn die Minuten bei 60 sind. Dann zählt sich logischerweise die Variable für die Stunden um einen hoch.

Die maximal mögliche Spielzeit liegt bei 99 Stunden, 59 Minuten und 59 Sekunden. Wenn diese Zeit erreicht wurde, wird die Funktion abgebrochen und eine Fehlermeldung ausgegeben, dass die maximale Spielzeit erreicht wurde.

Um die Zeit live zum Spiel anzuzeigen, wird ein neuer Thread eröffnet. Um auf die entsprechenden Befehle zugreifen zu können, wird die „Process.h“ Bibliothek benötigt. Der Befehl „\_beginThread()“ eröffnet dann einen neuen Thread. Er benötigt drei Argumente:

1. Die Funktion, die aufgerufen werden soll
2. Die Anzahl der Bytes für die Stackgröße
3. Argumente der Funktion, die aufgerufen wird

Der Thread gilt als beendet, entweder wenn die Funktion zu Ende ist oder der Thread von außen manuell mit „\_exitThread()“ beendet wird.

###### Passen im Spiel

Ein Spieler muss seinen Zug passen, wenn er keine Möglichkeit hat, einen Stein zu setzen. Wenn er zwei Züge direkt hintereinander passt, hat der andere Spieler automatisch gewonnen. Das Spiel ist auch vorbei, wenn beide Spieler direkt hintereinander gepasst haben. Anschließend wird ein Sieger festgestellt, indem gezählt wird, wie viele Steine die beiden Spieler haben.

##### Darstellung der Steinanzahl der jeweiligen Spieler (Score)

Der Ablauf der Funktion zur Darstellung der Scoreanzeige wird durch folgendes Struktogramm dargestellt:

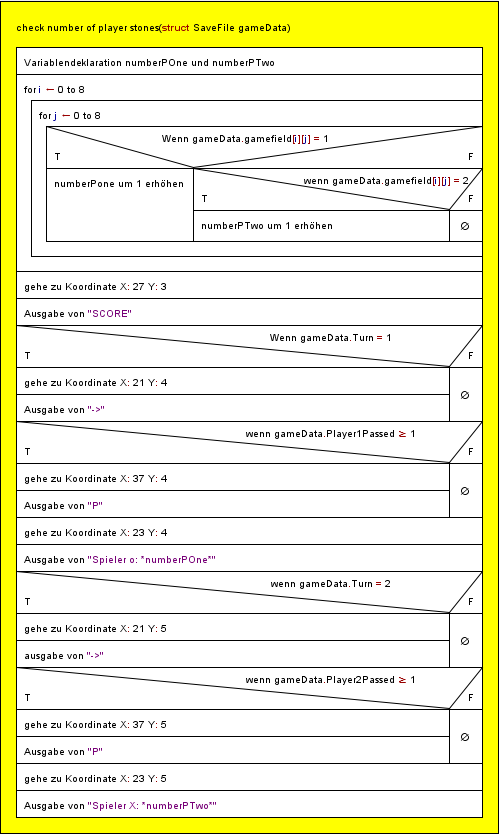


Abbildung 5: Struktogramm, der Funktionsweise der Überprüfung der Anzahl der Spielsteine des jeweiligen Spielers

##### Lesen und Schreiben eines Spielstandes aus/in eine/einer Datei

Ziel dieses Features ist es, den gesamten Spielstand in eine Datei zu speichern, um diesen wieder später laden zu können. Da man zum Zeitpunkt der Planung der Programmierung dieses Feature einbauen wollte, hat man mit einem zentralen struct sich das Speichern erheblich vereinfacht. Alle Variablen, sowie das Array, welches die Daten für das Spielfeld gespeichert hat, werden geordnet in eine Textdatei geschrieben, damit sie später reibungslos ausgelesen werden können. Dieses Feature lässt sich gerade durch das struct sehr einfach umsetzen.

##### Ende des Spiels

Das Spiel ist vorbei, wenn ein Spieler gewonnen hat oder keiner mehr einen Stein setzen kann oder wenn beide Spieler gleich große Punktzahlen haben. In allen Fällen wird eine Nachricht auf der Konsole ausgegeben. Wenn ein Spieler gewonnen hat, wird der entsprechende Spieler beglückwünscht, dass er gewonnen hat. Bei einem Unentschieden wird lediglich dargestellt, dass das Spiel unentschieden ausgegangen ist. Anschließend werden die Daten im struct zurückgesetzt. Man wird gefragt, ob man das Spiel neustarten möchte, wenn man ja sagt, gelangt man zurück in das Hauptmenü, sonst wird das Programm beendet.

### Qualitätssicherung

Während der Projektphase wird ständig über neue Änderungen kommuniziert, um möglichst viel Transparenz bei Ideen und Überlegungen zu haben. Falls ein Problem auftritt, bespricht man dies und überlegt gemeinsam eine Lösung. Danach werden die Aufgaben neu verteilt und die Programmierung geht weiter. Durch die Versionskontrollwebseite Github kann jedes Projektmitglied jederzeit auf die gesamte Commit-Historie zugreifen und jeden kleinen Entwicklungsschritt nachvollziehen. So kann optimal programmiert und entwickelt werden.

## Abweichungen, Anpassungen, Entscheidungen

### Abweichungen

Dieses Kapitel beschreibt die Abweichungen von dem Plan, wie man das Projekt konkret umsetzt, bis hin zur Feststellung, welche Sachen doch besser anders durchgeführt werden sollen, sowie die Begründung weshalb man sich zu dieser Abweichung entschieden hat.

#### Zentrale Header-datei aufsplitten

Es war anfangs geplant, dass man für alle Dateien eine Header-Datei baut, die alle Bibliotheken, die für das Projekt gebraucht werden und Funktionsprototypen der C-Dateien beinhaltet. Allerdings ist es viel übersichtlicher für jede C-Datei eine zugehörige Header-Datei zu schreiben, da man bei dem Funktionsumfang, wie er in diesem Projekt vorliegt schnell die Übersicht verliert, welche Funktion zu welcher Datei gehört und ob eine Funktion noch gebraucht wird.

#### Verwendung von einem zweiten Thread

Während der Entwicklung von der Darstellung der Zeit gab es das Problem, dass diese nur aktualisiert wurde, wenn man sich auf dem Spielfeld bewegt hat. Durch die Verwendung von Threads lassen sich zwei Prozesse im Programm parallel durchführen. Nun wird unabhängig von dem Spielgeschehen die Zeit live aktualisiert.

Um einen Thread einzubinden benötigt man zunächst die Bibliothek „process.h“. Der benötigte Befehl damit ein neuer Thread eröffnet wird lautet: „\_beginThread()“. Er beinhaltet drei Argumente, einmal die Funktion, die aufgerufen werden soll, dann die Stackgröße, und zuletzt die Argumente, die der Funktion, die aufgerufen werden soll, mit übergeben werden.

# Projektergebnisse

## Soll-Ist-Vergleich, Qualitätskontrolle

In der Entwicklungsphase wurde immer wieder darauf geachtet, dass man alle Features, die feste Vorgabe waren, auch einbaut. Deshalb hat man sich im Vorfeld einen Plan über mögliche Lösungen gemacht und hat sich auf ein globales Struct geeinigt, welches die gesamten Daten, die im Spiel vorhanden sind und noch entstehen, speichert.

Dies hat den großen Vorteil, dass man mit mehreren Dateien arbeiten kann, um eine viel bessere Übersicht von dem Quellcode zu bekommen. Dies unterstützt einen enorm in der Entwicklung, weil man so oft den Fehler lokalisieren kann und nicht die ganze Zeit mit hin und her Scrollen beschäftigt ist.

Gegen Ende der Entwicklungsphase wurde mit der Story-Map und der Vorgabenliste verglichen, was alles erledigt ist und ob eventuell Features nicht umgesetzt wurden. Nicht umgesetzte Features werden im nächsten Punkt beschrieben.

## Abweichungen, Anpassungen

Die grundsätzliche Funktionalität ist im Programm gegeben. Ein Extra (laut Story-Map) wurde nicht umgesetzt:

* Alleine gegen den Computer spielen (gegen eine KI)

Das bedeutet, dass zu diesem Feature der Menüpunkt Spielmodus in den Einstellungen beibehalten wird, allerdings die Funktionalität nicht vorhanden ist. Im Klartext heißt das, dass man den Modus einfach nicht mehr umstellen kann. Der Modus wird somit auch nicht mehr im Struct abgespeichert. Die beschriebene Umstellung des Spielmodus in den vorherigen Punkten wird beibehalten, um zu zeigen, dass dieses Feature grundsätzlich ein Teil der Programmierung war.

Es wurden anstelle für die Spielsteine Symbole verwendet. Für Spieler 1 das ‚O‘ und für Spieler 2 das ‚X‘.

# Anlagen

## Kundendokumentation

### Allgemeines zum Spiel

Das Spiel wird auf einem 8x8 großen Brettspiel mit 2 Spielern gespielt. Man spielt entweder Spieler gegen Spieler oder Spieler gegen den Computer. Die Spielsteine sind auf der einen Seite schwarz und auf der anderen weiß gefärbt. Am Anfang stehen genau in der Mitte jeweils zwei diagonal zueinander liegende Spielsteine wie in der folgenden Abbildung:



Abbildung : Startposition im Spiel, Quelle: https://www.topster.de/reversi/

### Spielablauf

Die Spieler setzen abwechselnd jeweils einen Stein. Es wird versucht die Steine des Gegners entweder waagerecht, senkrecht oder diagonal mit den eigenen Steinen einzuschließen. Die nach dem Zug eingeschlossenen Steine werden dann zu den eigenen.

Damit man seine Steine nicht einfach irgendwo hinsetzt, sind Züge nur dann möglich, wenn Steine vom Gegner eingeschlossen werden, dabei darf zwischen den eigenen Steinen und den eingeschlossenen gegnerischen Steinen kein Freiraum sein. Die eingeschlossenen Steine werden dann zu den eigenen Steinen. In einem Zug sind auch mehrere Waagerechten, Senkrechten oder Diagonalen möglich einzuschließen, solange die oben genannten Bedingungen erfüllt werden.

### Ziel des Spiels

Es wird während des gesamten Spiels die Anzahl der Spielsteine der jeweiligen Spieler auf dem Feld angezeigt.

Ein Spieler hat gewonnen, wenn keine Züge mehr möglich sind und ein Spieler mehr Steine auf dem Feld hat als der andere. Es sind keine Züge mehr möglich, wenn das Spiel vollständig mit Steinen gefüllt ist oder beide Spieler unmittelbar hintereinander gepasst haben, da sie keine Steine mehr umschließen können. Es ist auch möglich den anderen Spieler so auszuspielen, dass keine Steine mehr auf dem Feld von ihm übrig sind.

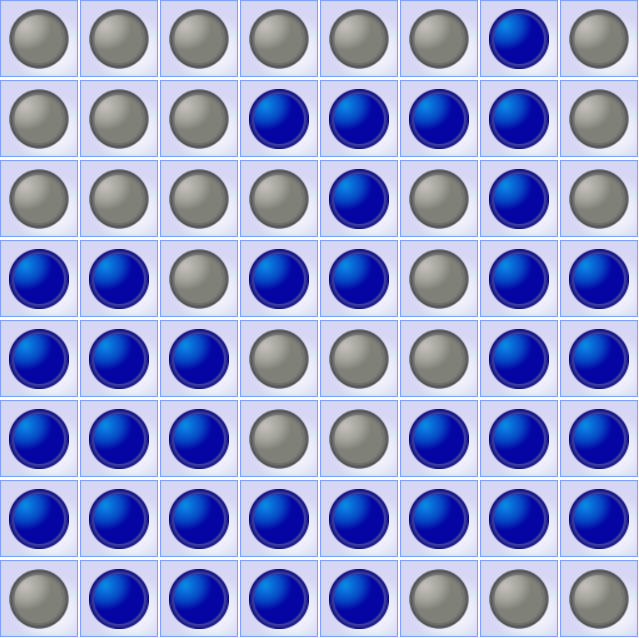


Abbildung : Ende des Spiels, Blau gewinnt, Quelle: https://www.topster.de/reversi

## Abbildungsverzeichnis

[Abbildung 1: Spielbrett von Reversi, Quelle: https://cdn3.volusion.com/artgw.hyvvw/v/vspfiles/photos/CB05120-4.jpg?1534152438 1](#_Toc9509510)

[Abbildung 2: Startmenü des Programms 4](file:///C:\Users\laltebor\Documents\Projekt\Reversi\docs\Dokumentation-Reversi.docx#_Toc9509511)

[Abbildung 3: Einstellungen im Programm 5](file:///C:\Users\laltebor\Documents\Projekt\Reversi\docs\Dokumentation-Reversi.docx#_Toc9509512)

[Abbildung 4: Spielfeld 5](file:///C:\Users\laltebor\Documents\Projekt\Reversi\docs\Dokumentation-Reversi.docx#_Toc9509513)

[Abbildung 5: Struktogramm, der Funktionsweise der Überprüfung der Anzahl der Spielsteine des jeweiligen Spielers 9](#_Toc9509514)

[Abbildung 6: Startposition im Spiel, Quelle: https://www.topster.de/reversi/ 12](file:///C:\Users\laltebor\Documents\Projekt\Reversi\docs\Dokumentation-Reversi.docx#_Toc9509515)

[Abbildung 7: Ende des Spiels, Blau gewinnt, Quelle: https://www.topster.de/reversi 13](#_Toc9509516)

## Tabellenverzeichnis

[Tabelle 1: Aufgabenverteilung 2](#_Toc9509241)

[Tabelle 2: Kostenplanung 3](#_Toc9509242)