BBS Brinkstraße

Dokumentation

**Reversi**



Verfasser: Lukas Gutknecht und Lukas Alte-Bornholt

Fach: Lernfeld 6 – Programmieren

Betreuer: Herr Wichmann

Abgabetermin: 23.05.2019

Inhalt

[1. Ausgangssituation 3](#_Toc7525992)

[1.1. Projektziele und Teilaufgaben 3](#_Toc7525993)

[1.2. Kundenanforderungen 3](#_Toc7525994)

[1.2.1. Allgemeines zum Spiel 3](#_Toc7525995)

[1.2.2. Spielablauf 3](#_Toc7525996)

[1.2.3. Ziel des Spiels 3](#_Toc7525997)

[1.3. Projektumfeld 4](#_Toc7525998)

[2. Ressourcen und Ablaufplanung 4](#_Toc7525999)

[2.1. Terminplanung, Ablaufplanung (Gantt-Diagramm) 4](#_Toc7526000)

[2.2. Kostenplanung 5](#_Toc7526001)

[3. Durchführung und Auftragsbearbeitung 5](#_Toc7526002)

[3.1. Prozessschritte, Vorgehensweise, Qualitätssicherung 5](#_Toc7526003)

[3.1.1. Planung der Programmierung 5](#_Toc7526004)

[3.1.2. Durchführung der Programmierung 6](#_Toc7526005)

[3.2. Abweichungen, Anpassungen, Entscheidungen 9](#_Toc7526006)

[4. Projektergebnisse 9](#_Toc7526007)

[4.1. Soll-Ist-Vergleich, Qualitätskontrolle 9](#_Toc7526008)

[4.2. Abweichungen, Anpassungen 9](#_Toc7526009)

# Ausgangssituation

## Projektziele und Teilaufgaben

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

## Kundenanforderungen

### 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:



Abb. 1:Reversi Anfangsposition,

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 einschließbar, 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.

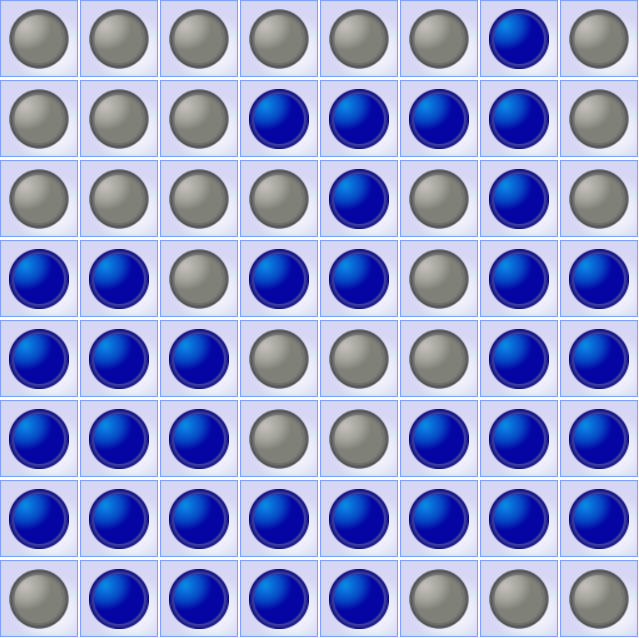


Abb. 2: Blau gewinnt: 36 – 28,

Quelle: https://www.topster.de/reversi/

## Projektumfeld

Als Programmiersprache ist C vorgegeben sowie als Entwicklungsumgebung Codeblocks.

# Ressourcen und Ablaufplanung

## Terminplanung, Ablaufplanung (Gantt-Diagramm

|  |  |  |
| --- | --- | --- |
| **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 | 8 | LG |
| Überprüfung auf ein Ende des Spiels |  |  |
| Spiel pausieren, sodass die Spielzeit anhält und man sich nicht mehr auf dem Feld bewegen und Steine setzen kann | 1 | LAB |
| Darstellung der Zeit parallel zum Spielen (Threading) | 0,5 | LAB |

## 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. Sich fragen, was im Projekt, nicht unbedingt fester Bestandteil des Spiels Reversi, fehlt
2. Gemeinsame Überlegung mit dem Projektpartner über Sinnhaftigkeit eines Features
3. Entwicklung des Features
4. 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, da es nicht sehr viele Gestaltungsmöglichkeiten gibt und dies bei einer Konsolenanwendung nicht im Vordergrund steht.

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 schlichtes Anfangsmenü entschieden, welches sich durch Eingabe von Zahlen navigieren lässt.

Unter dem Punkt‚ „Spiel Starten“ findet man schließlich das 8 mal 8 Feld, eine Score Anzeige, eine Zeitanzeige und welcher Spieler gerade am Zug ist. Die Anzeigen werden nach jedem Zug aktualisiert. Nachdem man das Spiel gewonnen hat, kommt ein neuer Bildschirm, wo gesagt wird, welcher Spieler gewonnen hat und wie der finale Punktestand ist.

Unter dem Punkt Einstellungen hat man die Auswahl zwischen zwei Spielmodi: Dem Spieler gegen Spieler- und dem Spieler gegen Computer-Modus. Zusätzlich kann man sich eine kurze Anleitung des Spiels durchlesen, falls man mit den Regeln nicht vertraut ist.

###### 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 ist 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 und 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“.

#### 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 1en oder 2en, je nach Spielstein; 0 für unbesetzte Felder
* Time: (integer): speichert die gestoppte Zeit die seit Spielstart gelaufen ist
  + Vergangene Zeit in Sekunden
* Turn (integer): speichertden jeweiligen Spieler, der gerade am Zug ist
  + 1 für Spieler Eins, 2 für Spieler Zwei
* Mode (integer): speichert den Modus in dem gespielt wird
  + 0 für Spieler gegen Spieler, 1 für Spieler gegen Computer

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

##### Startmenü

Das Startmenü wird durch einfache printf() Befehle realisiert. Es wird in einer do-while Schleife festgestellt, welche Taste gedrückt wurde. Falls eine nicht gewollte Taste gedrückt wird, wird der Bildschirm gelöscht und wieder aufgebaut, allerdings so schnell, dass man dies selten bemerkt. Falls eine der Zahlen von eins bis drei gedrückt wird, geschieht das, was rechts neben der Zahl steht, z.B. das Spiel beenden.

###### Aufbau des Startmenüs

Dieses Fenster wird direkt beim Start des Programms aufgerufen:

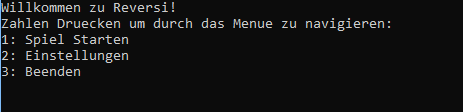


Abb. 3: Startmenü des Programms

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

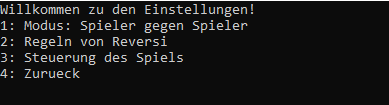


Abb. 3: Einstellungen des Programms

##### 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 StoneManagement-Datei checkt, ob die gewählte Position valide ist.
4. Wenn sie 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.

###### Überprüfung beim setzen des Steins

Das Überprüfen der Position des Steins beim Setzen ist eines der wichtigsten Dinge im Programm. Sie bestimmt, wo der Spieler als nächstes seinen Stein hinsetzen darf ohne dabei logischerweise die Regeln zu brechen. Es wird zum Beispiel überprüft, ob auf der ausgewählten Position sich schon ein Stein befindet, auf einem realen, physischen Spielfeld ist so etwas natürlich klar, dennoch muss man im Programm dies überprüfen, um ein versehentliches, falsches Setzen zu vermeiden. Danach wird überprüft, ob sich ein gegnerischer Stein in irgendeiner Richtung direkt neben dem gesetzten Stein befindet. Wenn dies erfüllt ist, wird nach einem eigenen Stein in der Richtung von der aus sich der gegnerische Stein befindet gesucht. Wenn all diese Bedingungen erfüllt wurden, dann wird der Stein schließlich gesetzt und der andere Spieler ist wieder dran.

###### 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.

##### Ende des Spiels

Das Spiel ist vorbei, wenn ein Spieler gewonnen hat oder keiner mehr einen Stein setzen kann und beide Spieler gleich große Punktzahlen haben. In beiden 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.

## Abweichungen, Anpassungen, Entscheidungen

# Projektergebnisse

## Soll-Ist-Vergleich, Qualitätskontrolle

## Abweichungen, Anpassungen