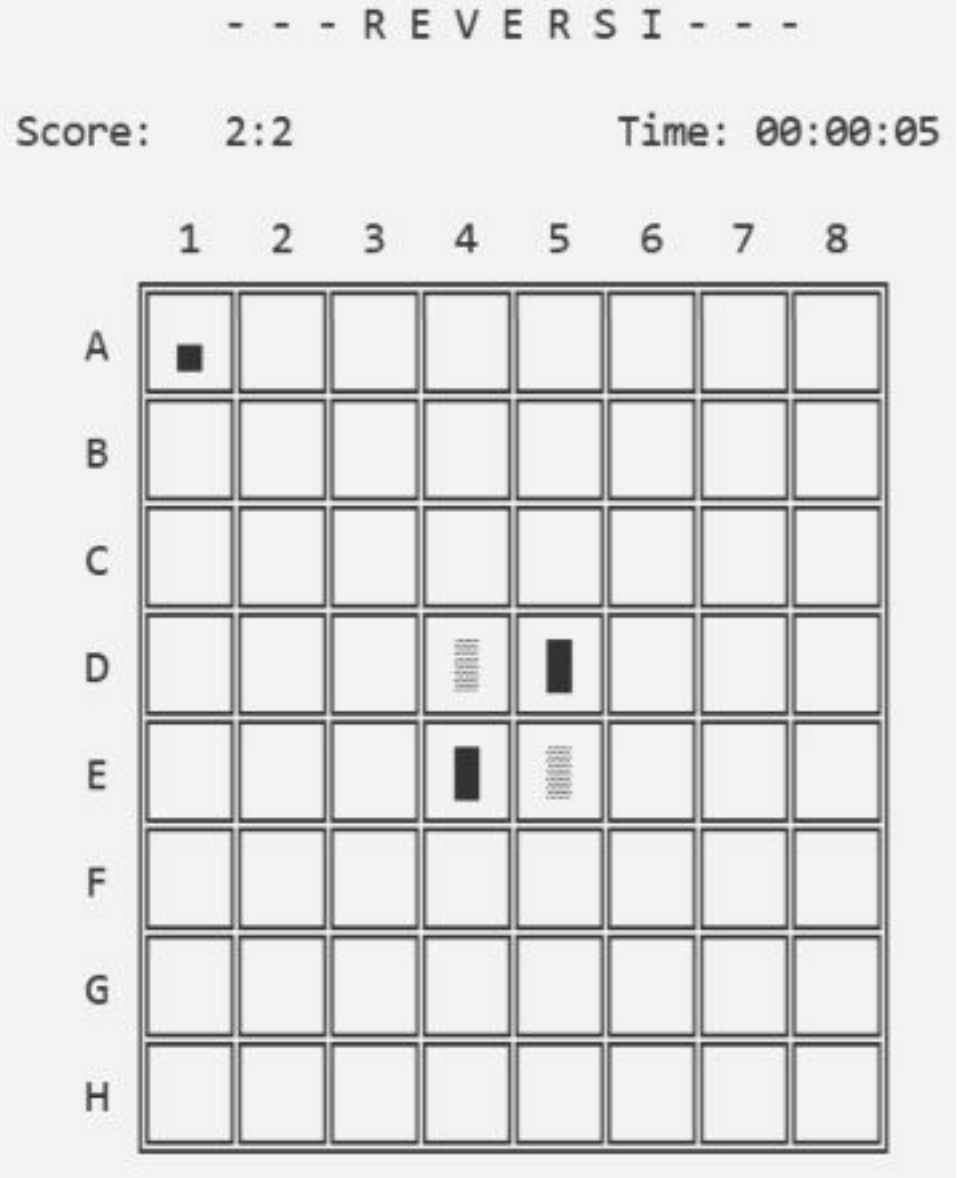


Entwicklung des Spiels „Reversi” in C

Maximilian Fickers, Fabian Heeke, Julian Görres

Inhaltsverzeichnis

1 Ausgangssituation

1.1 Projektziele und Teilaufgaben

Die folgende Projektarbeit stellt den Ablauf des Reversi[1] - Projektes dar. Ziel dieses Projektes ist eine spielbare Version des strategischen Brettspieles Reversi in der Programmiersprache C[2] zu entwickeln.

“Reversi (auch Othello genannt) ist ein strategisches Brettspiel für zwei Personen. Auf einem 8×8-Brett legen die Spieler abwechselnd Spielsteine, deren Seiten unterschiedlich (schwarz und weiß) gefärbt sind. Ein Spieler („Weiß“) legt seinen Stein immer mit der weißen Seite nach oben, der andere („Schwarz“) entsprechend mit der schwarzen. Zu Spielbeginn befinden sich vier Steine in vorgegebener Anordnung auf dem Brett. Ein Spieler muss seinen Stein auf ein leeres Feld legen, das horizontal, vertikal oder diagonal an ein bereits belegtes Feld angrenzt. Wird ein Stein gelegt, werden alle gegnerischen Steine, die sich zwischen dem neuen Spielstein und einem bereits gelegten Stein der eigenen Farbe befinden, umgedreht. Spielzüge, die zu keinem Umdrehen von gegnerischen Steinen führen, sind nicht erlaubt. Das Ziel des Spiels ist es, am Ende eine möglichst große Anzahl von Steinen der eigenen Farbe auf dem Brett zu haben. [...] Wenn die Spieler unmittelbar nacheinander passen, wenn also keiner mehr einen Stein setzen kann, ist das Spiel beendet.“[1]

1.2 Kundenanforderungen

Die Umsetzung des Projektes gestaltete sich anhand verschiedener Kundenanforderungen. Diese werden in Abbildung 2 dargestellt.

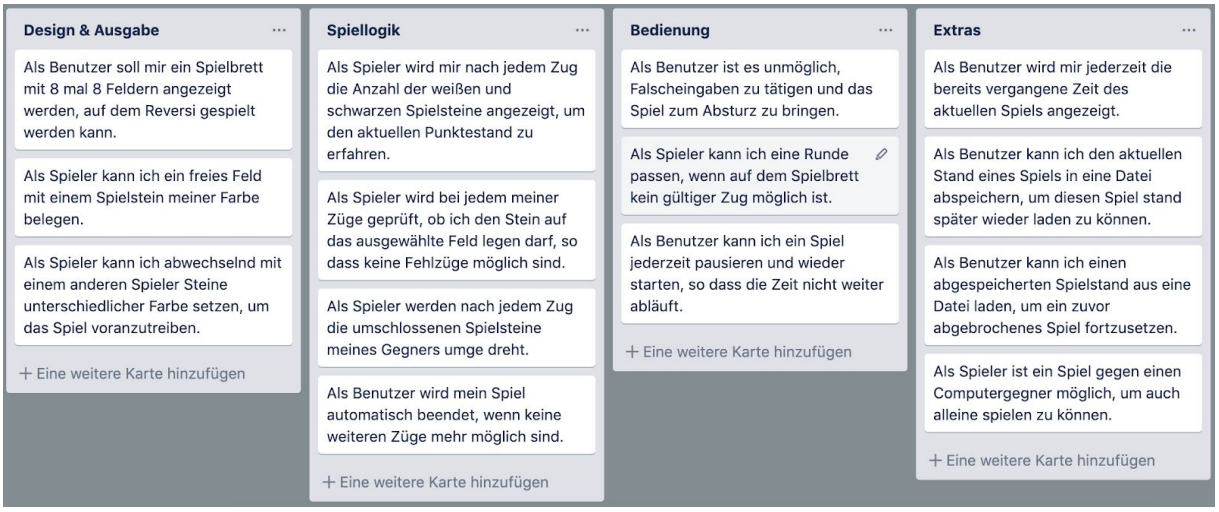


Abb. 2 - Kundenanforderungen

1.3 Projektumfeld

Das Spiel Reversi wurde als Projektarbeit im *Lernfeld 6: Entwickeln und Bereitstellen von Anwendungssystemen* realisiert.

Die Programmiersprache C und die Entwicklungsumgebung CodeBlocks[3] wurden hierbei als generelle Anforderungen an das Projekt gestellt. Die Verwendung von Bibliotheken, die nicht zu der C-Standard-Bibliothek gehören, war hierbei gestattet. Abgesehen der Vorgabe von Programmiersprache und -umgebung sollten sowohl erstellte Dateien, als auch wichtige Quellcode Abschnitte dokumentiert werden. Darüber hinaus sollte bei Umsetzung des Projektes darauf geachtet werden, dass Dateien, die Quellcode enthalten, eine gewisse Zeilenlänge nicht überschreiten.

2 Ressourcen und Ablaufplanung

2.1 Terminplanung, Ablaufplanung

Bei der Planung des Projektes wurde sich an der MVP-Methode[4] orientiert. Diese sieht vor, zunächst nur die nötigsten Funktionen des Programms zu implementieren und nach Vollendung dieser Basis das Projekt entsprechend zu erweitern. Wir entschieden uns für diesen Ansatz, da wir so nicht gezwungen waren uns von vornherein Gedanken über die Umsetzung einzelner Zusatzfeatures zu machen. Dies konnten wir stattdessen auf einen Zeitpunkt verschieben, an dem wir eine bessere Vorstellung von dem Verlauf und den Möglichkeiten/Grenzen des Projektes hatten. Die Planung des Ablaufs des Projektes wurde in einem GANTT-Diagramm[5] festgehalten, welches in Abbildung 3 dargestellt wird.

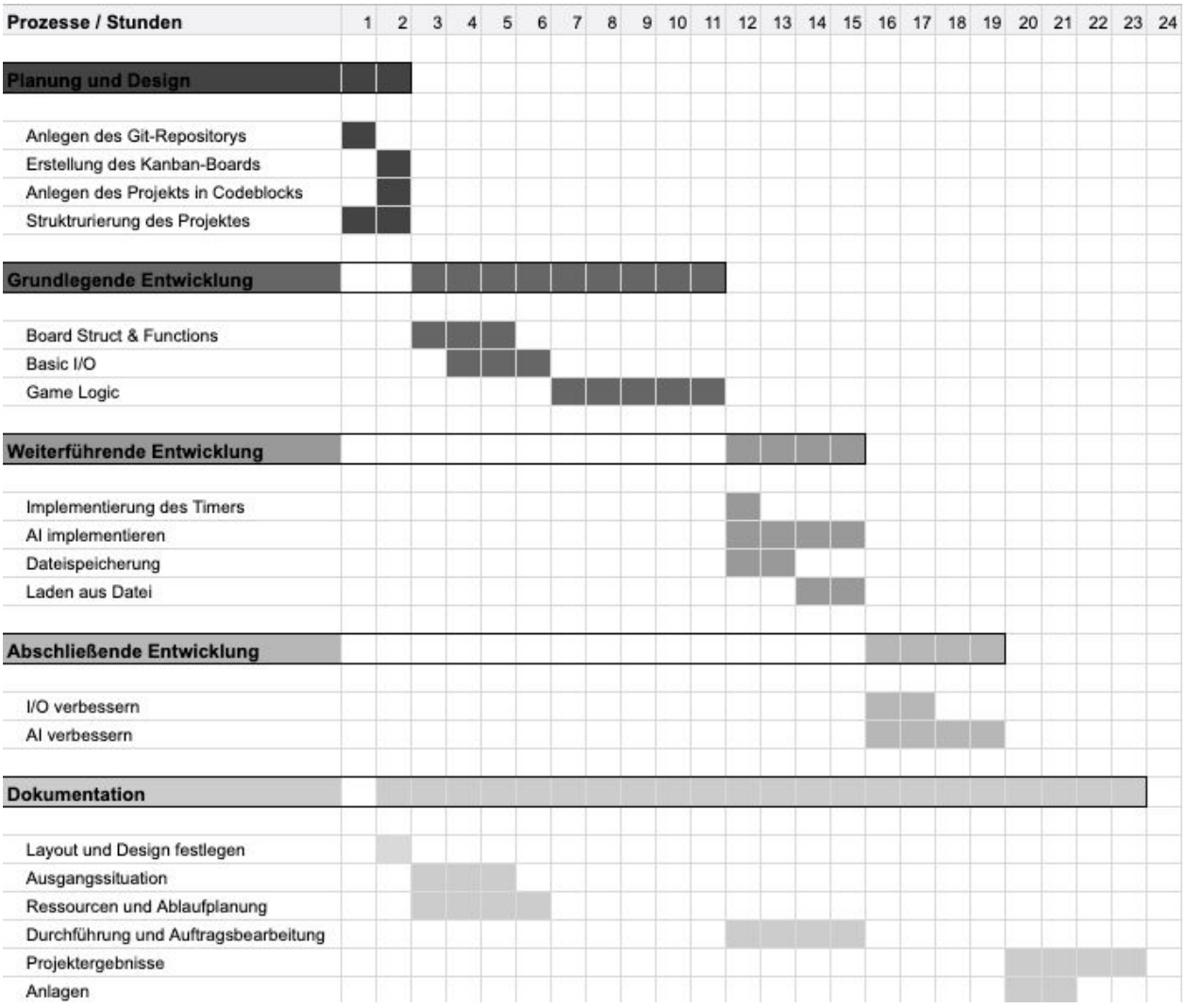


Abb. 3: GANTT-Diagramm

Außerdem entschieden wir uns dazu, einzelne Projektschritte mit Hilfe der Software-Planungsmethode Kanban zu realisieren. Die Planung dieses Projektes wurde mit der Software-Planungsmethode Kanban realisiert. Aus den Kundenanforderungen wurden einzelne Prozesse definiert und den jeweiligen Entwicklern aus dem Projekt zugewiesen.

Durch die Benutzung dieser Software-Planungsmethode erhofften wir uns stets einen guten Überblick über die Bearbeitung von anfallenden Aufgaben zu behalten. Abbildung 4 erklärt die grundlegende Funktionsweise von Kanban-Boards anhand einer entsprechenden Grafik.

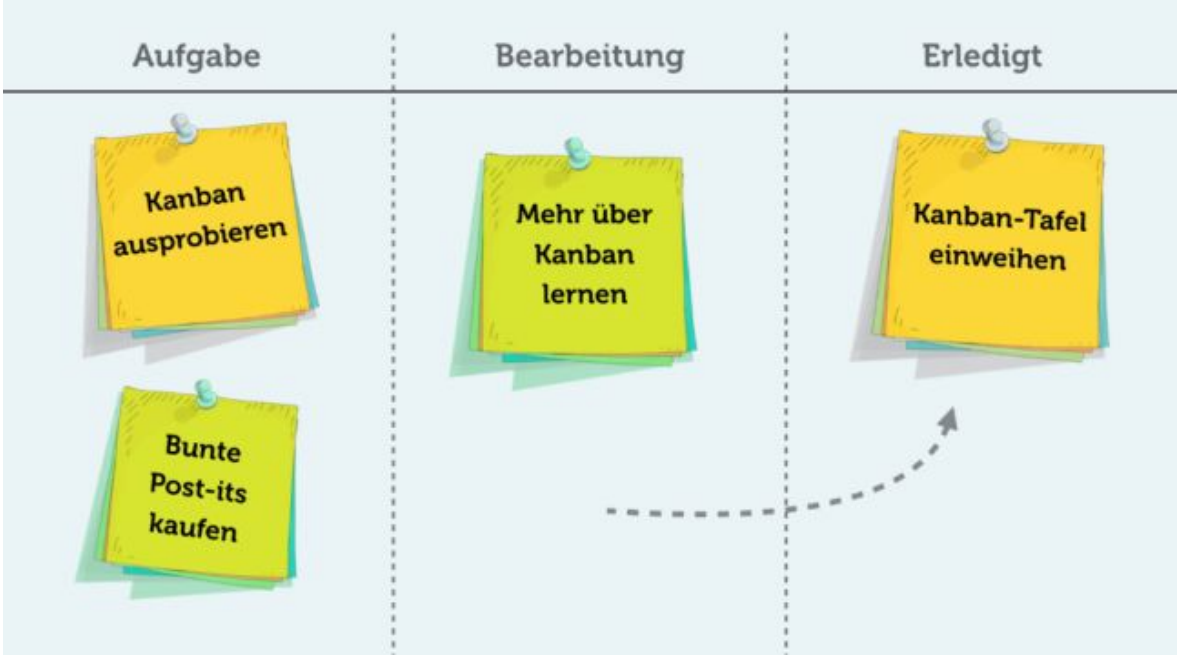


Abb. 4: Kanban-Board Beispiel [6]

2.2 Kostenplanung

Die folgende Tabelle gibt einen Überblick über die geplanten Personalkosten. Diese basieren auf der zuvor durchgeführten Ablaufplanung mithilfe des GANTT-Diagramms[6] (Abb.3).

Prozess	Kostensatz (pro Stunde)	Dauer (in Stunden)	Gesamtkosten
Planung und Design	120€	5	600€
Entwicklung	120€	26	3.120€
Dokumentation	120€	18	2.160€
Gesamt		**49**	**5.880€**

Abb. 5: Kostenplanung

3 Durchführung und Auftragsbearbeitung

3.1 Prozessschritte, Vorgehensweise, Qualitätssicherung

Zu Beginn des Projektes wurde ein Code-Style-Guide definiert. Somit konnte gewährleistet werden, dass der Code trotz mehreren Entwicklern dennoch gleich aufgebaut und einheitlich ist. Dieser wurde als .md-Datei in das GitHub-Projekt[7] integriert, um einen permanenten Zugriff für alle Entwickler zu gewährleisten.

Die Qualität der Software wurde darüber hinaus durch ausgiebige Kontrollen schon während des Entwicklungsprozesses sichergestellt. Der von einem Entwickler geschriebene Quellcode wurde in GitHub versioniert und durch einen Pull Request[7] den beiden anderen Entwicklern zur Prüfung auf Funktion und Qualität des Quellcodes bereitgestellt. Dadurch konnte zu jeder Zeit gewährleistet werden, dass der Quellcode die nötige Qualität und Funktionalität besitzt, da ohne einen erfolgreichen Pull Request der Stand des develop branches[7] nicht verändert werden konnte. Sobald der Entwickler seinen Pull Request an die anderen Entwickler sendete, wurde der Prozess im Kanban Board auf “Kontrolle” gesetzt. Nach erfolgreicher Prüfung des neu hinzugekommenen Quellcodes wird dieser auf den develop branch gemerged[7] und der entsprechende Prozess im Kanban Board auf “Fertig” gesetzt. Falls die Prüfung nicht erfolgreich verlaufen ist wird der Prozess zurück auf “Bearbeitung” gesetzt und dem Entwickler zur Überarbeitung gegeben.

Des Weiteren wurden durch das Projektteam regelmäßige Meetings geplant. In diesen Meetings wurde gemeinsam der aktuelle Projektstand geprüft um ggf. rechtzeitig auf Änderungen reagieren zu können. Außerdem dienten diese dazu sich untereinander auszutauschen.

3.2 Abweichungen, Anpassungen

Das Projekt wurde zu großen Teilen so durchgeführt, wie ursprünglich geplant.

Letzten Endes weniger genutzt als ursprünglich geplant wurde das Kanban-Board, da alle Beteiligten durch regelmäßige Treffen während und auch außerhalb des Schulunterrichts stets darüber informiert waren, an welchen Aufgaben die anderen Gruppenmitglieder momentan arbeiten. Auch das regelmäßige Überprüfen des Codes auf Funktionalität und Qualität verlagerte sich zunehmends mehr auf die erwähnten Treffen.

4 Projektergebnisse

4.1 Soll-Ist-Vergleich

Da sich bei der Planung und Strukturierung des Projektes an den Kundenanforderungen orientiert wurde, deckt das fertige Programm diese Anforderungen zu hundert Prozent ab.

Nachdem die grundlegenden Kundenanforderungen umgesetzt worden sind, konnten auch die erweiterten Kundenanforderungen implementiert und entsprechend dem Programm hinzugefügt werden.

4.2 Abweichungen, Anpassungen

Das Projekt konnte zunächst so durchgeführt werden, wie ursprünglich geplant. Es wurde zunächst ein Prototyp erstellt, auf dessen Basis die entsprechenden Programmerweiterungen implementiert werden konnten. Ein paar Abweichungen des ursprünglichen Plans fanden jedoch trotzdem statt.

Zunächst haben wir anfangs gedacht, dass wir während des Spiels eine count-Variable für die Rundenanzahl verwenden müssen, um zu wissen, welcher Spieler an der Reihe ist. Dies hatte sich jedoch erübrigt als uns aufgefallen ist, dass Spieler auch aussetzen können, und wurde in der Folge anders gelöst.

Vereinzelt mussten wir darüber hinaus gegen Ende der Umsetzung feststellen, dass an diversen Stellen des Programms noch Verbesserungsbedarf bestand. Hierbei ist vor allem das Eingeben der einzelnen Züge zu nennen. Nachdem wir zunächst dachten, dass eine simple Eingabe der Koordinaten auf der Konsole zum Setzen der Spielfeldmarkierungen ausreichend ist, mussten wir im Verlauf des Projektes feststellen, dass diese Art der Bedienung zu umständlich ist. In diesem Zusammenhang haben wir vergleichsweise lange nach Alternativen recherchiert, bis wir uns schließlich für eine Kombination der Bibliotheken “conio.h[8]” und “windows.h[9]” entschieden haben, da diese einfach einzubinden waren und diese Kombination gewünschten Funktionen mit sich brachte (vor allem die Möglichkeit, auf dem Board navigieren zu können).

Außerdem hatten wir anfangs lediglich berücksichtigt das Spiel zu pausieren, nicht aber auch ein Pausenmenü einzufügen, um während der Pause leicht die gewünschten Funktionen ausführen zu können. Dieses mussten wir gegen Ende noch einfügen, wodurch verschiedene Render-Funktionen geändert werden mussten und ein vergleichsweise hoher Zeitaufwand entstand.

Als positive Überraschung stellten wir fest, dass sich das Schreiben des Computer-Gegners als einfacher herausstellte, als ursprünglich angenommen. Hier profitieren wir von dem modularem Aufbau des Projektes, sodass bei der Implementierung der KI auf bestehende Funktionen zurückgegriffen werden konnte.

5 Anlagen

5.1 Kundendokumentation

Starten des Programms

Voraussetzungen für das Starten des Spiels sind ein Windows-Betriebssystem und die Installation des Programms Codeblocks.

In dem übergebenen Projektordner muss zunächst die Datei “ifa82-reversi.cpb” per Doppelklick ausgeführt werden.

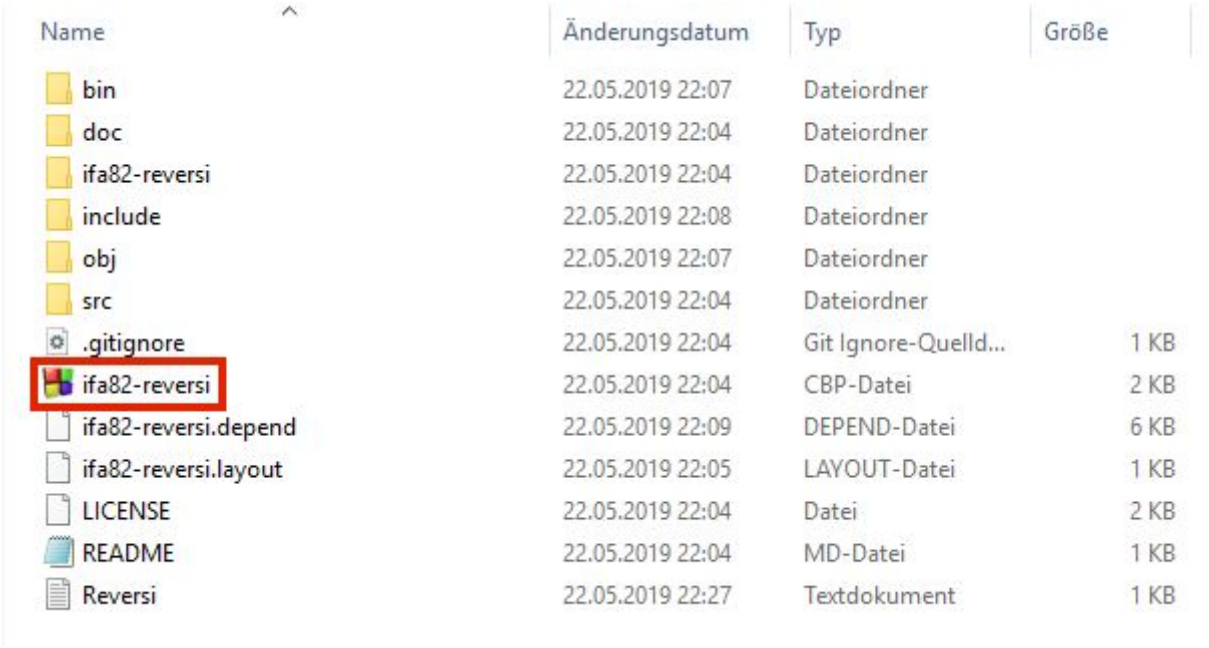


Abb. 6: Projektordner

In dem sich nun öffnenden Codeblocks-Fenster muss ein initialer Build des Programms erstellt werden, damit das Programm gestartet werden kann. Dies geschieht durch das klicken des Buttons “build and run” in der Taskleiste des Programms. Die Konsole öffnet sich und das Programm wird ausgeführt. Wurde bereits ein Build des Programms erstellt, reicht es aus den links anliegenden “Run”-Knopf zu klicken.

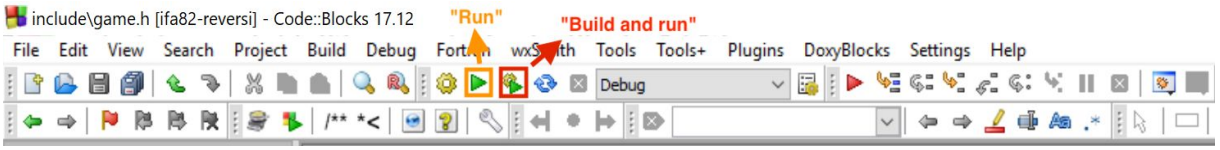


Abb. 7: Codeblocks-Toolbar

Hauptmenü und Spielstart

Nach dem Programmstart erscheint das Hauptmenü. Dieses kann durch drücken der Tasten [1], [2] oder [3] bedient werden.

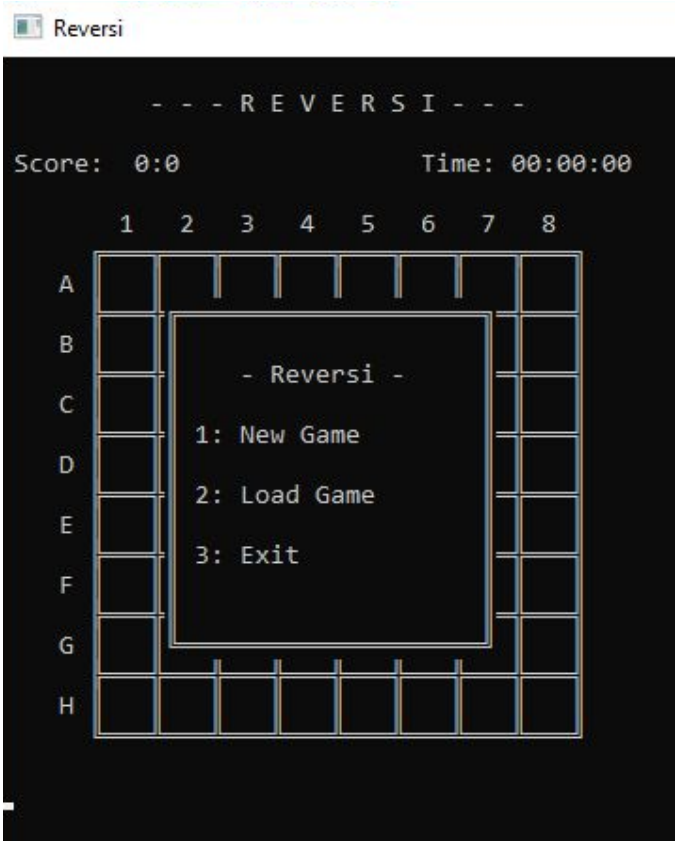


Abb. 8: Hauptmenü

Taste [1] startet ein neues Spiel.

Taste [2] lädt den letzten gespeicherten Spielstand.

Taste [3] beendet das Spiel.

Wurde ein neues Spiel gestartet gelangt der Spieler zunächst in ein zweites Menü. Dieses kann durch das drücken der Tasten [1] und [2] bedient werden.

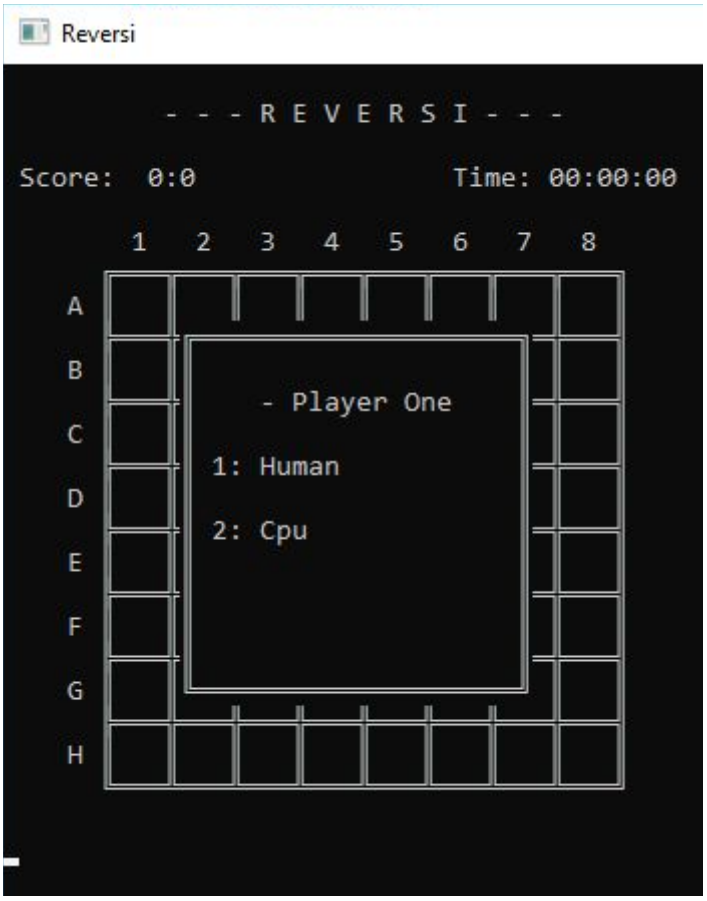


Abb. 9: Spielerwahl

Taste [1] startet ein Spiel gegen einen realen Gegner.

Teste [2] startet ein Spiel gegen den den Computer.

Es erscheint ein entsprechendes Menü für Spieler 2, welches analog zu dem oberen aufgebaut ist. Mit dem erneuten Auswählen einer der Optionen, startet das Spiel.

Spielverlauf

Es kann durch das bedienen der Pfeiltasten über das Spielfeld navigiert werden. Befindet sich der angezeigte Marker auf dem gewünschten Feld, kann durch drücken der [Enter]-Taste eine Markierung auf dem Feld hinterlassen werden. Alle Felder, die einen gültigen Zug für den aktuellen Spieler darstellen, sind mit einem Punkt gekennzeichnet.

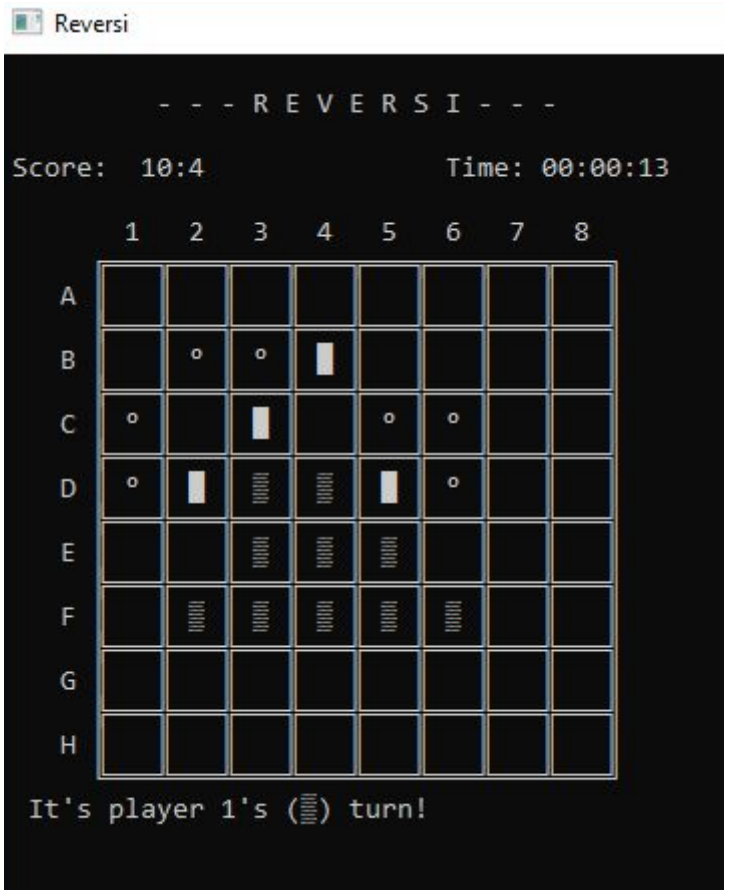


Abb. 10: Spielverlauf

Wurde eine Markierung gesetzt ist der andere Spieler an der Reihe. Dies wird wiederholt, bis das Spiel beendet ist. Die Konsole kann im Anschluss geschlossen werden.

Wenn ein Spieler keinen gültigen Zug machen kann, muss er passen. In diesem Fall wird eine entsprechende Nachricht angezeigt, die mit einer beliebigen Taste zur Kenntnis genommen werden kann, wodurch der andere Spieler wieder an der Reihe ist.

Während des Spiels haben beide Spieler die Möglichkeit das Spiel durch drücken der [P]-Taste zu pausieren.

Das sich öffnende Pausen-Menü kann durch die Tasten [1], [2] oder [3] bedient werden.

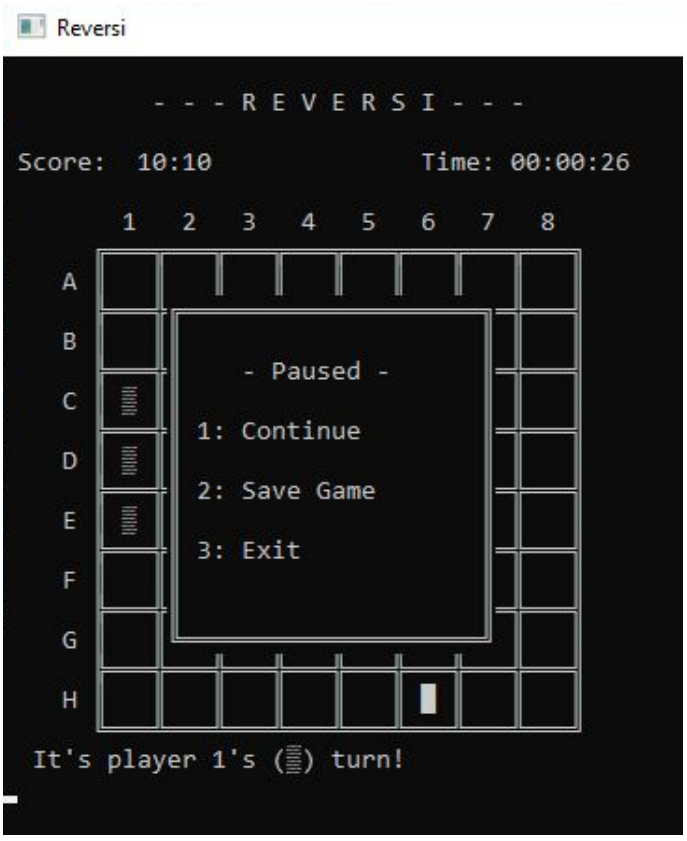


Abb. 11 Pausenmenü

Taste [1] führt das pausierte Spiel fort.

Taste [2] speichert den momentanen Stand des Spiels. Bei erfolgreichem Speichern des Spielstands erscheint eine entsprechende Nachricht auf dem Bildschirm.

Taste [3] beendet das Spiel.

5.2 Relevante technische Anmerkungen

5.2.1 Code Style Guide

Allgemein

- Einrückungen werden mit 4 Leerzeichen dargestellt.

Funktionen

- Funktionsbezeichnung im camel_case
- Private Funktionen durch Unterstrich gekennzeichnet
- Keine Leerzeichen vor oder nach den Klammern
- Geschweifte Klammern in eigenen Zeilen

Verzweigungen

- Leerzeichen nach dem if
- Geschweifte Klammern in der Zeile der Verzweigung
- Prüfung auf true oder false findet implizit statt

Schleifen

- Leerzeichen nach dem for/while
- Geschweifte Klammern in der Zeile der Schleife
- Bei for-Schleifen: Leerzeichen nur nach den Semikolons

Konstanten

- UPPER_CASE mit Unterstrichen

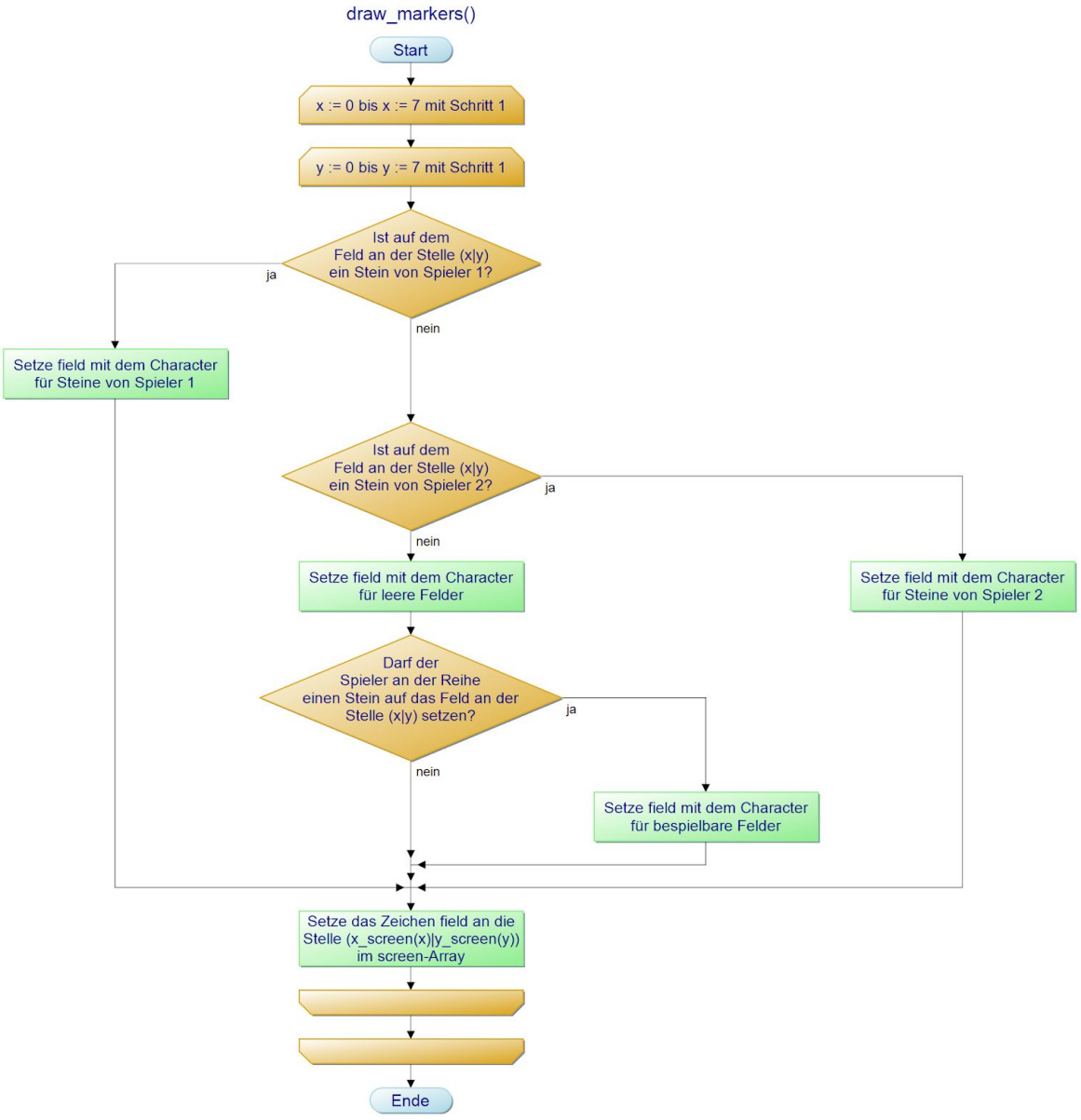
Variablen

- camel_case
- Deklaration immer mit Default-Wert
- Jede Variable in einer eigenen Zeile
- Structs und enums in PascalCase

Kommentare

- Zwei Sternchen in der ersten Zeile, danach maximal ein
- Jede Funktion und Klasse muss einen Kommentar besitzen
- Englisch

5.2.2 - Programmablaufplan zur Funktion draw_markers



5.3 Abbildungsverzeichnis

5.4 Literaturverzeichnis

[1] Wikipedia, Die freie Enzyklopädie: Seite „Othello (Spiel)“. - Online unter: https://de.wikipedia.org/w/index.php?title=Othello_(Spiel)&oldid=181670037 [10. Mai 2019]

[2] Wikipedia, Die freie Enzyklopädie: Seite C (Programmiersprache) - Online unter: https://de.wikipedia.org/wiki/C_(Programmiersprache) [14. Mai 2019]

[3] Code::Blocks: „The open source, cross platform, free C, C++ and Fortran IDE.“ - Online unter: http://www.codeblocks.org/ [15. Mai 2019]

[4] Startplatz.de:Minimum Viable Product - Online unter: https://www.startplatz.de/startup-wiki/mvp/ [23. Mai 2019]

[5] Gantt.com: Was ist ein Gantt Diagramm? - Online unter: https://www.gantt.com/ge/ [18. Mai 2019]

[6] Karrierefibel: “Kanban Board: Tipps und Definition”. - Online unter: https://karrierebibel.de/kanban/ [23. Mai 2019]

[7] GitHub.com: About GitHub - Online unter: https://help.github.com [18. Mai 2019]

[8] Wikipedia, The free Encyclopedia: conio.h - Online unter: https://en.wikipedia.org/wiki/Conio.h [19. Mai 2019]

[9] Wikipedia, The free Encyclopedia: windows.h - Online unter: https://en.wikipedia.org/wiki/Windows.h [19. Mai 2019]