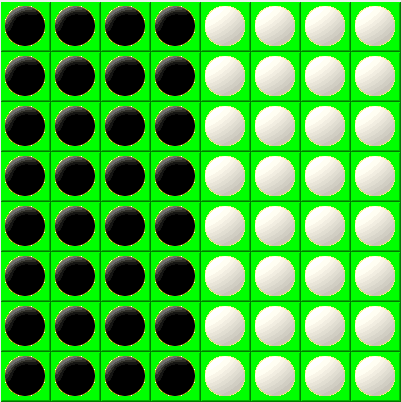
**Zürcher Hochschule für angewandte Wissenschaften**

**Diplomstudium Informatik**



**Projektarbeit 4. Semester**

**Informatikprojekt**

**JReversi**

|  |  |
| --- | --- |
| Autoren | Oliver Aeschbacher  René Kamer |
| Dozent | Jens-Christian Fischer |
| Projektstart | 26. März 2012 |
| Projektpräsentation | 15. Juni 2012 |

Inhaltsverzeichnis

[1. Abstract 4](#_Toc326963820)

[2. Detailanalyse der Aufgabenstellung 4](#_Toc326963821)

[3. Motivation 4](#_Toc326963822)

[4. Projektplanung 5](#_Toc326963823)

[a. Prinzipielles Vorgehen 5](#_Toc326963824)

[b. Iterationplan 6](#_Toc326963825)

[i. Iteration 1 6](#_Toc326963826)

[ii. Iteration 2 6](#_Toc326963827)

[iii. Iteration 3 7](#_Toc326963828)

[c. Aufteilung der Aufgaben 7](#_Toc326963829)

[d. Anpassungen im Verlauf des Projekts 7](#_Toc326963830)

[5. Umgebung des Projekts 8](#_Toc326963831)

[a. Programmwahl 8](#_Toc326963832)

[b. Tools 8](#_Toc326963833)

[6. Reversi 8](#_Toc326963834)

[a. Geschichte des Spiels 8](#_Toc326963835)

[b. Regeln / Erklärung des Spiels 8](#_Toc326963836)

[c. Aktuelle Geschehnisse 8](#_Toc326963837)

[7. Spieletheorie 8](#_Toc326963838)

[a. Spielbaum-Theorie 8](#_Toc326963839)

[b. Heuristik 8](#_Toc326963840)

[c. MiniMax Algorithmus 8](#_Toc326963841)

[d. Alpha-Beta Algorithmus 8](#_Toc326963842)

[8. Umsetzung des Projekts 9](#_Toc326963843)

[a. GUI 9](#_Toc326963844)

[i. Darstellung / Aufbau 9](#_Toc326963845)

[ii. Update Problematik 9](#_Toc326963846)

[iii. Thread Problematik 9](#_Toc326963847)

[b. Traversieren der Strukturen 9](#_Toc326963848)

[c. Auslesen der Spielsituationen 9](#_Toc326963849)

[d. Heuristik 9](#_Toc326963850)

[e. Umsetzung des Alpha-Beta Algorithmus 9](#_Toc326963851)

[9. Projektfazit 9](#_Toc326963852)

[10. Danksagungen 9](#_Toc326963853)

[11. Quellen 9](#_Toc326963854)

[12. Abbildungsverzeichnis ? 9](#_Toc326963855)

# Abstract

Wir haben als Aufgabenstellung eine Reversi Implementation ausgewählt. Dieses Brettspiel soll einerseits gegeneinander spielbar sein sowie auch gegen einen Computergegner.   
Es soll ein Schwierigkeitsgrad ausgewählt werden können, mit welcher danach der Computergegner spielt. Hierbei soll eine KI (Künstliche Intelligenz) zum Einsatz kommen, welche auf den gewählten Schwierigkeitsgrad zugeschnitten ist.  
Ebenfalls soll eine Eröffnungsbibliothek implementiert werden, welche Eröffnungszüge korrekt anzeigt. Die vorliegenden Spielinformationen sollen übersichtlich dargestellt werden, d.h. gemachte Züge, wenn ein Spieler gepasst hat, Eröffnung sowie Gewinner bzw. Verlierer.  
Es werden hohe Ansprüche an Aussehen und Animation gestellt.

# Detailanalyse der Aufgabenstellung

Für die vorgegebene Aufgabenstellung werden verschiedenste Konzepte benötigt. Diese umfassen:

* GUI Implementierung des Spielfeldes sowie der Umgebenden Informationsanzeigen
* Korrekte Abbildung der Spielregeln des Spiels
* Algorithmen zur korrekten Traversierung des Spielfeldes
* Algorithmen zum Erkennen eines Passes (direkt ausgelöst, Spieler muss nicht einen Passknopf drücken)
* Abstrahieren der Vorliegenden Spielinformationen in entsprechende Datenstrukturen
* Spielbaumtheorie
* Aufbauen einer Datenstruktur für einen Spielbaum
* Animation des Umdrehens der Spielsteine
* Bewertungsheuristik
* KI: Spielbaumtheorie im Zusammenhang mit dem Alpha Beta Algorithmus und deren Umsetzung

Die benötigten Konzepte werden einerseits selbst erarbeitet und andererseits aus der bekannten Spieltheorie abgeleitet.   
Die selbst erarbeiteten Algorithmen müssen einen hohen Grad an Geschwindigkeit bieten, da die man dem menschlichen Spieler keine sehr langen Wartezeiten zumuten will.  
Die Datenrepräsentationen dürfen ebenfalls nur sehr wenig Platz beanspruchen da die Spielbaumtheorie eine Speicherintensive Implementation darstellt bei einem komplexen Spiel wie Reversi.  
Die Konzepte für Alpha Beta Algorithmen sind seit vielen Jahren bekannt und werden aus der Theorie direkt umgesetzt. Es wurde zuerst noch der MiniMax Algorithmus diskutiert, da jedoch die Alpha Beta Methode bis zur Hälfte an Speicher spart, ist dies die erste Wahl geworden.

# Motivation

# Projektplanung

## Prinzipielles Vorgehen

Als erstes stand die Planungsphase an. Möglichst alle zu implementierenden User Stories wurden erstellt und die dazugehörigen Tasks geschrieben. Da man am Anfang nicht alle Eventualitäten klären kann kamen im Verlauf des Projektes noch einige User Stories und Tasks hinzu.  
Prinzipiell wurde als erstes das GUI realisiert, damit man visuell arbeiten kann und schon eine Diskussionsgrundlage hatte. Danach wurden sogleich die ersten Spiellogiken implementiert (Man könnte auch sagen, das Regelwerk wurde sukzessive umgesetzt).  
Weitere Funktionen wurden hinzugefügt, so wie z.B. die Möglichkeiten, das Spiel entsprechend seinen Bedürfnissen einzustellen (Schwierigkeitsgrad, Menschlicher oder Computergegner, etc).

Die Spiele-Anzeigen (letzter gemachter Zug, Passen, Spielgewinner) wurden noch vor schwierigeren Passagen (Spieltheorie, Animation) eingebaut, da dies noch relativ einfach zu machen sein sollte.

Da wir auf dem Internet und in einschlägigen Foren praktisch keine animierten Versionen eines Reversis fanden, haben wir noch einen Animationsteil eingebaut (Zusätzliche User Story im Verlauf des Projekts), dies, um auch unseren eigenen Ansprüchen gerecht zu werden.

Um eine gute Basis nach der zweiten Iteration zu schaffen, haben wir ein grosszügiges Refactoring eingeplant. Dies, damit wir im nächsten Teil mit einer sauberen und aufgeräumten Umgebung arbeiten können.

Danach folgte die Umsetzung des eigentlichen Algorithmus, welcher den Kern des Projekts bildet. Dazu haben wir extra Zeit eingeplant, in welcher wir zuerst die gesamten Spieltheorien sowie die zugehörigen Algorithmen kennenlernen konnten. Danach konnten wir die Implementation der eben erwähnten Punkte in Angriff nehmen und das Projekt für erste externe Tests abschliessen.  
Für diesen Teil haben wir uns ein wenig mehr Zeit herausgenommen, wissend dass wir hier noch auf Probleme und Verzögerungen stossen könnten.

Am Ende wurde das Programm noch relativ ausgiebig von externen Beta Testern auf Herz und Nieren geprüft. In der letzten Phase wurden die Aussagen der Tester nochmals auf das Programm übertragen bzw. angepasst und die letzten Dokumentationslücken geschlossen.

## Iterationplan

Der Iterationsplan wurde online geführt, der Einfachheit halber ist dieser jetzt in seiner endgültigen Version in dieses Dokument übernommen worden. Die Tasks beziehen sich auf das externe Dokument User Stories - Cards and Tasks.docx bzw. PDF.

## Iteration 1



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Legende: | | | | |
| User Stories | Tasks | Milestones | In Progress | Abge-schlossen |
|

## Iteration 2



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Legende: | | | | |
| User Stories | Tasks | Milestones | In Progress | Abge-schlossen |
|

## Iteration 3





|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Legende: | | | | |
| User Stories | Tasks | Milestones | In Progress | Abge-schlossen |
|

## Aufteilung der Aufgaben

Die Aufgaben des Projekts wurden grösstenteils nach den Stärken der Programmierer vorgenommen. Oliver Aeschbacher hat seine Stärken im GUI Programmieren sowie im Bereitstellen guter Datenstrukturen und Threading, René Kamer hat seine Stärken eher in der Algorithmik sowie im mathematischen Teil und Heuristik.  
Absichtlich wurden jedoch gewisse Aufgaben auch direkt diesem Ansatz entgegengesetzt verteilt, damit sich auch während des Projekts Lerneffekte und Horizonterweiterungen einstellen konnten. Kleinere Anpassungen waren nach dieser Vergabe noch nötig, damit die Arbeit ungefähr gleich verteilt werden konnten auf beide Personen.

## Anpassungen im Verlauf des Projekts

Ein prinzipieller kleiner Lapsus ist uns bei der Planung passiert, was sich im Nachhinein als gut für uns erwiesen hat. In der festen Annahme, dass es drei Iterationphasen mit je drei Wochen gibt, haben wir die Planung entsprechen ausgeführt. Im Verlauf des Projekts mussten wir jedoch feststellen, dass wir vier Iterationsphasen mit je drei Wochen zur Verfügung hatten. Dies hat uns jedoch gut in die Hände gespielt, da gegen Ende des Semesters der Zeitplan wesentlich straffer wurde und wir pro Woche nicht mehr so viel Zeit in dieses Projekt investieren konnten. Ebenfalls sind in der letzten Iterationsphase noch einige Probleme aufgetaucht. Dazu aber später mehr.  
Des Weiteren haben wir den Ablauf noch angepasst, um animierte Spielsteine zu erzeugen. Dies hatten wir aus zwei Gründen getan:

* Man findet keine oder nur sehr wenige Implementationen im Netz, welche animierte Spielsteine implementiert haben. Dies wollten wir ändern.
* Wir haben selbst sehr hohe Ansprüche an Ausführung und Qualität unserer Arbeit und wollten unbedingt noch eine Animation mit reinnehmen.

# Umgebung des Projekts

## Programmwahl

---Why Java? ---

## Tools

---GC Tools, Git, etc.----

# Reversi

## Geschichte des Spiels

---Kurzer Geschichtlicher Abrisss

## Regeln / Erklärung des Spiels

---Alle Regeln auflisten (internet, mit Graphiken zur Erklärung)

## Aktuelle Geschehnisse

---Entwicklungen, Computerprogramme, Turniere, aktive Reversi Ligen (oder so)

# Spieletheorie

## Spielbaum-Theorie allgemein

Vorbemerkung: Ein Spielbaum lässt sich für beinahe jedes Spiel aufstellen, wie z.B. für Schach, Reversi, TicTacToe oder VierGewinnt.  
Wir betrachten ein sehr einfaches Spiel, in dem es in jeder Spielstellung nur zwei erlaubte Züge gibt, links (l) oder rechts (r). Befindet sich das Spiel nun in einer bestimmten Spielstellung S und Spieler A führt einen Zug aus, so sind zwei Folgestellungen möglich, Sl und Sr. Bild 1 veranschaulicht diese Situation durch einen gerichteten Graphen. Die Knoten des Graphen entsprechen den Spielstellungen die (von oben nach unten gerichteten) Kanten des Graphen entsprechen den Spielzügen. Der Graph ist ein Baum, der Spielbaum für einen Zug ausgehend von Spielstellung S.

**r**

**l**

**Sr**

**Sl**

**S**

Danach ist Spieler B am Zug. Je nachdem, ob er Spielstellung Sl oder Sr vorfindet und je nachdem, ob er Zug l oder Zug r ausführt, ergibt sich anschließend eine der vier möglichen Spielstellungen Sll, Slr, Srl, Srr. Bild 2 veranschaulicht diese Situation wieder anhand des entsprechenden Spielbaums.   
Nach einem weiteren Zug von Spieler A und einem weiteren Zug von Spieler B sind insgesamt 16 Spielstellungen möglich. Es möglich, dass unterschiedliche Zugfolgen zu der gleichen Spielstellung führen. D.h. unterschiedliche Knoten des Spielbaums können gleiche Spielstellungen repräsentieren.

**Slr**

**Srl**

**Sll**

**Srr**

**r**

**l**

**Sr**

**Sl**

**S**

**r**

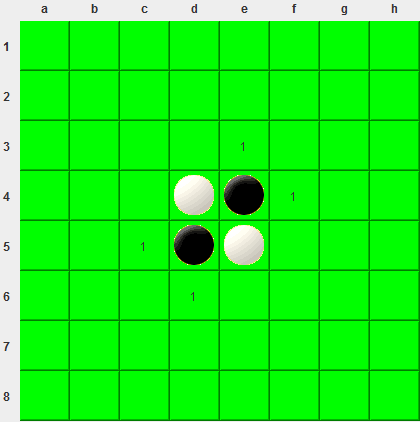
**l**

**r**

**l**

## Spielbaum-Theorie Reversi

Übertragen auf das Spiel Reversi kann man nun sagen, dass es nicht nur zwei erlaubte Züge geben kann, sondern sehr viele (um herauszufinden, was das Maximum an möglichen Spielzügen ist, müsste man alle möglichen Spielsituationen durchrechnen). Dargestellt aus der Startsituation stellt sich der Spielbaum beim Reversi so dar:



**Sc5**

**Sd6**

**Sf4**

**Se3**

**c5**

**e3**

**S**

**d6**

**f4**

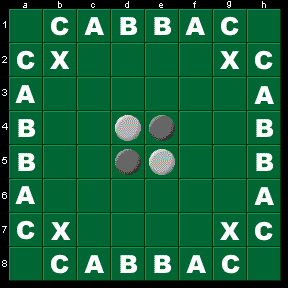
Entsprechend verläuft auch die Fortsetzung wesentlich breiter, auf eine entsprechende Grafik wird hier verzichtet.  
Wie man sich vorstellen kann, wird ein Reversi-Spielbaum mit zunehmender Spieldauer sehr breit werden (12-13 mögliche Züge sind keine Seltenheit in Spielsituationen) und die Datenrepräsentation wird entsprechend sehr viel Speicher benötigen.

## Heuristik allgemein

Das Ergebnis der folgenden Algorithmen hängt massgeblich von der Bewertungsfunktion, welche in den meisten Fällen eine Heuristik ist, ab, daher werfen wir hier einen genaueren Blick darauf.  
Eine Bewertungsfunktion ordnet jedem Zustand p einen Wert x zu über die Funktion f(p), wobei der Zustand den jeweiligen Brettsituationen in den Blättern des Baumes entspricht. Dies kann eine sehr einfache oder auch sehr komplexe Heuristik bzw. Funktion sein.   
Eine ideale Bewertungsfunktion ordnet einer Stellung den Wert +1 zu, wenn Spieler A gewinnt und den Wert -1, wenn B gewinnt. Bei Unentschieden entsprechend den Wert 0.  
Kann man den Spielbaum vollständig aufbauen, spielt der Algorithmus ein perfektes Spiel, was in der Realität jedoch nur bei sehr einfachen Spielen wie z.B. TicTacToe möglich ist.  
Bei allen anderen Spielen modifiziert man die Bewertungsfunktion, aber immer noch so, dass Spieler A bei positiver Aussicht eine positive Bewertung hat und umgekehrt.

## Heuristik Reversi

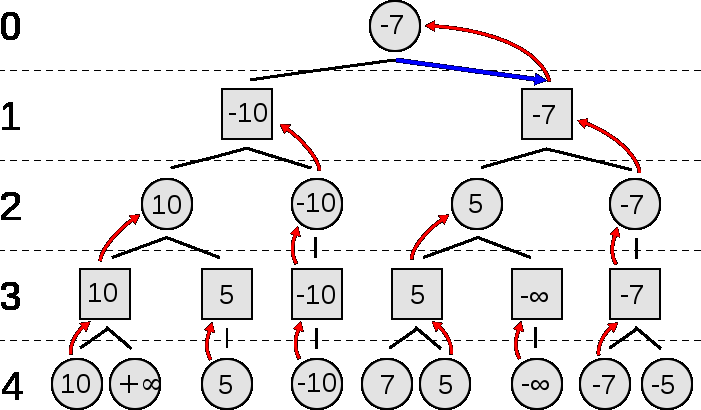
Bei Reversi sind verschiedene Bewertungsfunktionen möglich. Es hat sich aber eine allgemeine, auch für Anfänger verständliche Heuristik durchgesetzt, welche hier im Detail anhand einer Graphik erklärt wird. Diese Erklärung bezieht sich hauptsächlich auf das Mittelspiel, da die Erklärungen zu Eröffnung und Endspiel den Rahmen dieser Abhandlung sprengen würde.

* Generell empfiehlt es sich, so wenige Spielsteine wie möglich umzudrehen. Dies hat den Hintergrund, dass je weniger Steine man hat, umso weniger kann der Gegner schlagen. Des Weiteren erhöht eine kleine Anzahl von Steinen die Zugmöglichkeiten enorm (Fachbegriff: Mobility & Agility)
* Solange kein triftiger Grund besteht, sollte man in den „Sweet Sixteen“ bleiben (die 16 Felder im Zentrum des Spielfeldes). Der Grund: sobald man diese Verlässt, betritt man ein sogenannt „heisses“ Feld.
* Wenn man als erster einen Spielstein an den Rändern platziert, ist es im generellen besser auf ein A oder B Feld zu spielen als auf ein C Feld. Der Grund ist, dass man beim C-Feld, wenn es geschlagen wird, die Ecke verliert (Ecken können nicht geschlagen werden und haben enorme Macht)
* Man sollte verhindern, eine „Wand“ zu bilden (eine zusammenhängende Linie von Spielsteinen am Rand der Spielsteine, welche nicht am Rand des Feldes liegen). Hintergrund: diese Spielsteine bergen grosse Gefahr und haben geringe Mobilität)
* Wenn der Gegner eine „Wand“ bildet, sollte man solange wie möglich NICHT durch diese hindurchbrechen. Hintergrund: Aufgeschoben ist nicht aufgehoben.
* Es ist immer besser, wenn man die Kontrolle über Spielsteine hat, wenn diese quadratisch in der Mitte einer Gruppe von Steinen liegt als Grenzsteine (am Rand der bestehenden Spielsteine) zu haben. Der Grund liegt abermals in der Mobilität und Agilität.

Wenn man diese Regeln in der Heuristik abbilden kann (was wirklich nicht einfach ist), wird die Heuristik relativ gute Resultate liefern.

## MiniMax Algorithmus

Mit der Theorie der Spielbäume und dazugehörend der Theorie der Bewertungsfunktionen können wir nun eine Vorversion des Alpha-Beta Algorithmus betrachten: den MiniMax Algorithmus. Dieser ist anwendbar (wie der Alpha-Beta Algorithmus auch) auf Spielbäume aller Art.  
In untenstehender Grafik ist ein solcher beliebiger Spielbaum aufgezeigt, von der zugrundeliegenden Spielsituation 4 Züge in die Tiefe (entspricht logischerweise der Suchtiefe 4)  
In der Wurzel (Kopfknoten) ist A am Zug, A spielt also die Züge 0 und 2. Hier wird jeweils der untergeordnete Zug maximiert, also der für Spieler A günstigste Zug ausgewählt.  
Die Knoten bei 1 und 3 entsprechen Situationen, in denen B am Zug ist, hier wählt B den für sich günstigsten Zug. Da wir jedoch in diesem Beispiel für A rechnen, müssen diese Knoten minimiert werden.  
Angefangen bei den Blättern arbeitet sich der Algorithmus so hoch bis zu Wurzel. Der Wurzel wird also von seinen untergeordneten Knoten der grösste Wert zugewiesen und dieser Knoten wird dann auch gespielt (in der Graphik der Wert -7, was einem spezifischen Zug entspricht)



## Alpha-Beta Algorithmus

---Erklärung des Alpha-Beta Algorithmus mit ganz vielen Grafiken und Beispielen

# Umsetzung des Projekts

## GUI

## Darstellung / Aufbau

## Update Problematik

## Thread Problematik

## Traversieren der Strukturen

---Aufzeigen der selbst erarbeiteten Algorithmen

## Auslesen der Spielsituationen

---Wieso klonen für die Heuristik, Struktur des Boards

## Heuristik

---Erklären der Bewertungsfunktionen. Unterpunkte: Easy, Medium, Hard. Und wie die Heuristiken dazu verwendet werden. Evt Probleme ansprechen

## Umsetzung des Alpha-Beta Algorithmus

--- Prinzipieller Aufbau und Erklärung. Probleme bei der Umsetzung ansprechen

# Projektfazit

# Danksagungen

---Beta-Tester, Helfer, etc.

# Quellen

<http://www.iti.fh-flensburg.de/lang/algorithmen/graph/spielbaum.htm#footnote>  
E. Horowitz, S. Sahni: Algorithmen. Springer (1981)  
<http://www.iicm.tugraz.at/Teaching/theses/2000/_idb9e_/greif/node7.html>  
<http://www.informatik.uni-kiel.de/fileadmin/arbeitsgruppen/realtime_embedded/teaching/ss08/p-soft/p-soft05.pdf>  
<http://de.wikipedia.org/wiki/Minimax-Algorithmus#Bewertungsfunktion> (primär Grafiken)  
<http://games.yahoo.com/help/rv>  
<http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo19.php>

# Abbildungsverzeichnis ?