



**Fakultät  
Informatik und Mathematik**

# Projektbericht

zum Wahlpflichtfach im SS 2021

**Implementierung von Brettspielen am Beispiel ReversiXT**



**Gruppe:** 01

**Autoren:** benedikt.halbritter@st.oth-regensburg.de  
markus1.koch@st.oth-regensburg.de  
iwan.eckert@st.oth-regensburg.de

**Leiter:** Prof. Dr. rer. nat. Carsten Kern

**Abgabedatum:** ??..2021



# Inhaltsverzeichnis



# 1 Einleitung

ReversiXT ist eine Erweiterung des Brettspiels Reversi, welches in den 1880er Jahren vom Engländer Lewis Waterman entwickelt wurde. Es handelt sich um ein 2-Personen-Spiel mit einem 8x8 großen Spielfeld. Es werden je zwei Spielsteine in unterschiedlichen Farben diagonal zueinander mittig auf dem Spielfeld platziert (siehe Abbildung ??). Der blaue Spieler beginnt und darf nun auf leere Felder setzen, die ausgehend von diesem Feld in beliebiger Richtung (senkrecht, waagerecht oder diagonal) einen oder mehrere gegnerische Steine zwischen eigenen Steinen einschließen. Beim Durchführen eines Spielzuges werden die gegnerischen Spielsteine in die eigene Farbe umgefärbt. Die möglichen Startzüge werden in der Grafik eingezeichnet. Gewonnen hat derjenige, der zum Schluss die meisten Steine besitzt.

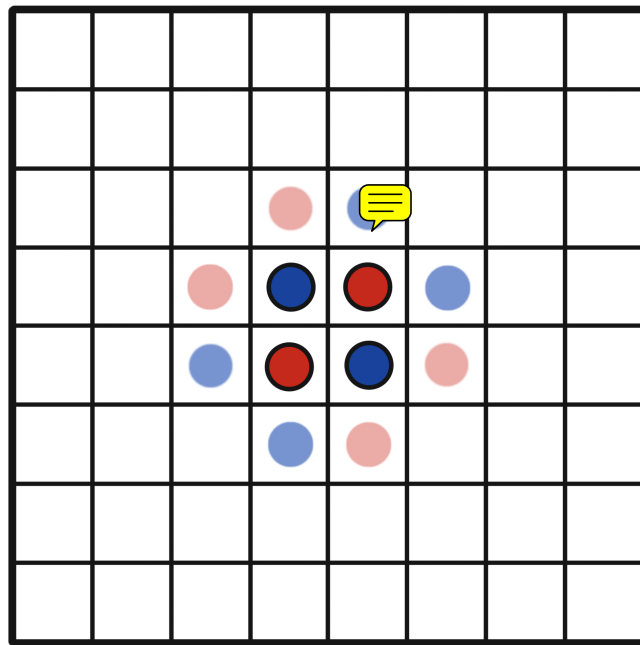


Abbildung 1: Grundspiel Reversi mit möglichen Startzügen

## 1.1 Besonderheiten von ReversiXT

Das Spiel ReversiXT, welches als **Reversi Extreme** bezeichnet wird, basiert auf den gleichen Regeln wie das Grundspiel Reversi. Es beinhaltet jedoch einige Zusatzregelungen, welche hier kurz erklärt werden. Bei ReversiXT treten bis zu acht Spieler auf einem maximal 50x50 großen Spielfeld gegeneinander an. Das Spiel besteht dabei aus zwei unterschiedlichen Spielphasen. In Phase 1 müssen Spieler solange ziehen, bis kein Zug mehr möglich ist. Im Anschluss beginnt die Phase 2, die sogenannte **Bonusphase**. Zudem gibt es mehrere Spezialfelder, die vorab fest definiert sind und jeweils nur einmal ausgelöst werden können. Mit einem **Invertfeld** werden die Farben aller Spieler um eins weiter verschoben. In einem **Choicefeld** wird ein Spieler mit einem

Anderen getauscht, hier ist es auch erlaubt sich selbst zu wählen. Ein **Expansionsfeld** wird als Gegner gewertet und kann nach den gleichen Regeln wie andere Spieler eingenommen werden, oder mithilfe eines **Überschreibsteins** ohne Beachtung der Spielregeln überschrieben werden. Überschreibsteine können auch genutzt werden, um einen Gegenspieler unter Beachtung der normalen Regeln zu überschreiben und dabei das Spielfeld auf gewohnte Weise einzufärben. Die Anzahl von Überschreibsteinen sowie Bomben ist zu Beginn des Spieles festgelegt. Durch ein **Bombefeld** darf der Spieler zwischen einem Überschreibstein und einer Bombe wählen, welche dem jeweiligen Spieler gutgeschrieben wird. Ist es nun keinem Spieler mehr möglich einen legitimen Spielzug durchzuführen, beginnt die zweite Phase. Hierbei müssen abwechselnd, sofern vorhanden, alle Bomben gesetzt werden. Hierzu wird ein Feld ausgewählt und ein Bereich mit festgelegtem Radius weggesprengt. Dabei werden **alle Felder** zu Löchern umgewandelt. Löcher sind nicht besetzbare Felder, **sozusagen tote Felder**.

Eine weitere Besonderheit von ReversiXT sind **Transitionen**. Eine **Transition** kann an einem Feld liegen, an dem in einer gewissen Zugrichtung kein Weiteres mehr existiert. Besitzt ein solches Feld nun eine Transition in einer Richtung, in die der Spieler durch einen Spielzug ziehen möchte, kommt sein Stein an einer anderen festgelegten Stelle des Spielfeldes heraus.

Im Grundspiel Reversi gelten Ecken und die damit eingenommenen Kanten als sichere Felder, da es nicht mehr möglich ist, diese durch einen gültigen Zug umzufärben. Im Gegensatz dazu gibt es in ReversiXT aufgrund von Überschreibsteinen keine sicheren Felder, da diese einfach überschrieben werden können. Zusätzlich gibt es **extrem viele Besonderheiten und unvorhersehbare Abhängigkeiten** aufgrund von Transitionen und den zuvor genannten Zusatzregeln. Aus diesen Gründen ist es für einen Menschen sehr schwer bis unmöglich ReversiXT erfolgreich zu spielen.

## 1.2 Vorstellung vom Wahlpflichtfach

Die Aufgabe in diesem **Fach** besteht darin, einen Client mit einer künstlichen Intelligenz zur bestmöglichen Spielzugwahl zu entwickeln, da es Computern möglich ist, alle **denkbaren** Spielzüge zu berücksichtigen und diese miteinander zu vergleichen. Wir erwarten uns von diesem **Fach** Einblicke in die Entwicklung von künstlicher Intelligenz zu erhalten. Zusätzlich ist es das erste Projekt im Studium, bei dem in einem Team an einem gemeinsamen Projekt gearbeitet wird. Wir erhoffen uns damit wichtige Erfahrungen bezüglich Softwareplanung, Teamkoordination und Arbeitsverteilung zu sammeln, die für den späteren Berufseinstieg von großem Nutzen sind. Aktuell ist zudem das Thema maschinelles Lernen in aller Munde, **wobei es nicht für jedes Projekt von Vorteil ist**. Mit der Entwicklung einer künstlichen Intelligenz und einer Einführung über das maschinelle Lernen wollen wir die Vor- und Nachteile beider Innovationen erkennen und hautnah die Unterschiede verstehen lernen.



## 2 Allgemeine Informationen


Bei umfangreicheren Softwareprojekten ist eine umfassende Vorbereitung und Planung unabdingbar. Es ist wichtig eine gute Kommunikation untereinander zu gewährleisten, um sich gegenseitig zu helfen und Informationen auszutauschen. Eine erfolgreiche Kommunikation benötigt nicht nur eine gewisse technische Grundausstattung, sondern auch ein gutes soziales Miteinander aller Teamkollegen. In diesem Abschnitt werden alle Teammitglieder sowie die verwendete Software und Hardware vorgestellt. Durch die Vorstellung ist es für den Leser nachvollziehbarer, wie die Entscheidungen im Team herbeigeführt wurden.

### 2.1 Team und Kommunikation

Alle Teammitglieder studieren Allgemeine Informatik, kennen sich seit dem Erstsemester und befinden sich aktuell im vierten Semester. Das Team belegte gemeinsam zwei allgemeinwissenschaftliche Fächer, wodurch sich alle noch näher kennenlernen konnten und sich viele Gemeinsamkeiten herausgestellt haben. Mithilfe von Signal, Zoom und Discord werden aktuelle Pläne und neue Erkenntnisse geteilt, wie mindestens einmal wöchentlich ein Meeting abgehalten, bei dem alle Änderungen dem Team vorgestellt werden. Bei diesem Meeting handelt es sich um eine Art von Code review, bei dem gegebenenfalls kleine Fehler bzw. Unklarheiten aufgespürt werden, um diese dann schnellstmöglich zu beseitigen. Durch diese Vorstellungsrunden erhoffen wir uns einen besseren Überblick über das gesamte Projekt und ein tieferes Verständnis des Programmcodes. Alle Teammitglieder haben bereits durch die Vorlesungen Programmieren 1 und Programmieren 2 die grundsätzlichen Programmierkonzepte und vor allem das objektorientierte Programmieren gelernt, welches in diesem Projekt von großer Bedeutung ist. Alle Beteiligten belegten zudem im letzten Semester das Fach Algorithmen & Datenstrukturen, in dem jeder einen umfangreichen Überblick über gängige Sortier- und Suchalgorithmen erhalten hat. Dank dieses Fachs ist es für alle Entwickler wesentlich leichter, verschiedene performancerelevante Stellen aufzuspüren und diese mithilfe von geeigneten Algorithmen weiter zu optimieren, um eine bestmögliche Geschwindigkeit heranzuholen. Das Team hat sich darauf geeinigt, die Klassennamen, Variablen sowie Kommentare und Dokumentationen auf Englisch zu schreiben, da es sich hierbei um die Sprache der Softwareentwicklung handelt. Benedikt Halbritter hat vor seinem Studium eine schulische Ausbildung als Informatiker abgeschlossen und kann aus diesem Grund gelernte Inhalte an das Team weitergeben, was sich des Öfteren als sehr positiv herausstellt. Iwan Eckert befindet sich in einem dualen Studium, seine beruflichen Erfahrungen bringen dem Team vor allem in der Konfiguration des Build-Tools enorme Zeitersparnisse, sowie Performanzoptimierungen aufgrund von Umstellungen auf andere Softwarekomponenten. Markus Koch verfügt bereits über Erfahrungen in LaTeX, wodurch er den Teamkollegen bei der Einrichtung der benötigten Software hilft. Zudem übernimmt er die Aufteilung des gesamten

Dokumentes  die Übersichtlichkeit zu verbessern.

Die Vorlesung findet immer dienstags statt. Im Anschluss wird im Team das neue Übungsblatt  durchgesehen und eine gerechte Arbeitsteilung durchgeführt. Hier wird ein Konzept der einzelnen Vorgehensweisen erstellt und zudem an einem konzeptionellen  Klassendiagramm gearbeitet, damit jedem die Schnittstellen bekannt sind und somit jeder ohne große weitere Absprachen an seinen Aufgaben arbeiten kann. In der nachfolgenden Tabelle sind alle zu erledigenden Aufgaben mit den entsprechenden Bearbeitern eingetragen.



Aufgabe	Entwickler
1.5	Benedikt, Iwan, Markus
2.1	Benedikt, Iwan, Markus
2.2	Benedikt, Iwan, Markus
2.3	Iwan, Markus
3.1	Benedikt
3.2	Iwan
3.3	Markus
4.1	Iwan, Markus
4.2	Markus
4.3	Benedikt
5.1	..
5.2	..
6.1	..
6.2	..
7.1	..
7.2	..
8.1	..
8.2	..
9.1	..
9.2	..

Tabelle 1: Übersicht der Aufgabenverteilung

## 2.2 Technische Daten

Als Entwicklungsumgebung wurde von allen Teamkollegen das Programm IntelliJ von JetBrains in der Version 2020.3.3 verwendet. Diese IDE bietet viele nützliche Zusatzfunktionen und beinhaltet dabei eine integrierte Entwicklungsmöglichkeit mittels der man LaTeX Dokumente erstellen kann. Das Programm eignet sich aus diesen Gründen hervorragend für die Entwicklung umfangreicher Projekte und steht Studenten zusätzlich auch in der kommerziellen Version kostenlos zur Verfügung. Da alle Teammitglieder die gleiche IDE verwenden, erhoffen wir uns weniger Kompatibilitätsprobleme bei der Entwicklung und bessere gegenseitige Unterstützung bei Fragen zur IDE. Benedikt und Iwan verwenden als Betriebssystem Windows 10, Markus entwickelt auf macOS 11. Auf allen Systemen wird zur Versionskontrolle Git in der Version 2.31.1 eingesetzt. Auf das empfohlene Tool TortoiseGit wird aufgrund der hervorragenden Integration seitens JetBrains meist verzichtet. Zum Erstellen von automatisch generierten lauffähigen Programmen setzt unsere Gruppe auf Gradle 6.3. Bei dieser Version handelt es sich bewusst um eine veraltete jedoch stabile Version, mehr Informationen als bei der neusten Version 7.0 zur Verfügung stehen und wir auf weniger auftretende Kompatibilitätsprobleme hoffen. Es handelt sich hier um ein einfach zu benutzendes Build-Tool, was auf eine eigene Sprache namens Groovy setzt und dadurch gewisse Leistungsvorteile gegenüber dem weit bekannten aber veralteten Tool Maven hat. Das Team entschied sich, das Projekt mit der Programmiersprache Java umzusetzen, da diese die gelernte Sprache aus Programmieren 2 ist und somit alle Beteiligten bereits umfangreiche Erfahrungen damit gesammelt haben und wir uns somit nicht erst in eine andere Programmiersprache einarbeiten müssen. Zum automatischen Testen unserer Softwarekomponenten benutzen wir außerdem das JUnit Testframework, da wir alle bereits damit Erfahrungen gesammelt haben und es sich um ein weit verbreitetes Framework zum Testen unter Java handelt.

Die Hardware, die zur Entwicklung und zum Durchführen von Tests eingesetzt wird, ist im Anschluss detailliert aufgeführt. Aus Testzwecken wird zudem in einer virtuellen Maschine mit Ubuntu 20.04 LTS getestet, ob das Programm erfolgreich gebaut werden kann.

- Benedikt Halbritter
  - Prozessor: Intel Core i7 - 7490K
  - Arbeitsspeicher: 16 GB 2800 MHz DDR4
  - Grafikkarte: NVIDIA GeForce GTX 1070 Ti
  - Festplatte: SSD 1TB - M.2
  - Betriebssystem: Windows 10
- Iwan Eckert
  - Prozessor: Intel Core i7 - 4790K
  - Arbeitsspeicher: 8 GB 1866 MHz DDR3
  - Grafikkarte: NVIDIA GeForce GTX 780 Ti
  - Festplatte: SSD 512 GB - SATA III
  - Betriebssystem: Windows 10
- Markus Koch
  - Prozessor: 2,6 GHz 6-Core Intel Core i7
  - Arbeitsspeicher: 32 GB 2400 MHz DDR4
  - Grafikkarte: Radeon Pro 560X 4 GB
  - Festplatte: APPLE SSD 512GB - PCI-Express
  - Betriebssystem: macOS 11.2.3



## 2.3 Datenstruktur

Das Spielfeld wird in der `Board`-Klasse in einem zweidimensionalen `char`-Array gespeichert. Wir haben uns für ein Array statt einer Liste entschieden, da sich die Größe des Spielfeldes während des gesamten Spielverlaufes nicht mehr ändert und man so mit konstanter Zeit auf ein bestimmtes Feld zugreifen kann. Da uns der Server die Informationen über die Map als hexadezimalen Byte-Stream übermittelt, speichern wir diese als `char` ab, um einen möglichst geringen Speicheraufwand von einem Byte zu erzielen. Die Verwendung eines `Strings` würde im Gegensatz dazu ganze 4 Byte pro Feld belegen.

Zusätzlich werden in der `Board`-Klasse alle möglichen Spielzüge eines Spielers berechnet, ausgewählte Spielzüge durchgeführt und damit das Spielfeld aktualisiert. Hierbei wird ein Spielzug durch die Klasse `Move` repräsentiert, welche sämtliche Felder beinhaltet, die beim Ausführen des Spielzuges umgefärbt werden müssen. Außerdem beinhaltet die Klasse die Information darüber, ob es sich bei dem Spielzug um ein Bonus-, Inversions-, Expansions- oder Choicefeld handelt. Desweiteren speichert die `Board`-Klasse die Spieleranzahl, den Bombenradius und eine Liste aller Transitionen ab.

Transitionen werden in einer `HashMap` gespeichert, um einen konstant schnellen Abruf gewährleisten zu können. Der `Key` in der `HashMap` ist ein `Integer`, welcher sich wie folgt zusammensetzt:

$$\text{HashKey} = x \cdot 1000 + y \cdot 10 + r \quad (1)$$

$$21.174 = 21 \cdot 1000 + 17 \cdot 10 + 4 \quad (2)$$

Befindet sich nun eine Transition an der Position  $x_1$  und  $y_1$  in Richtung  $r_1$  (nach unten), so lautet der dazugehörige `Key` 21174, wie er auch in Zeile 2 berechnet wird. Die Kombination aus Koordinaten und der jeweiligen Richtung ist mit dieser Berechnung immer eindeutig, wodurch die Singularität des Schlüssels gewährleistet ist. Als `Value` wird der `HashMap` ein Objekt der Klasse `Transition` übergeben. Eine Transition besteht aus einer  $x_2$ ,  $y_2$  und einer  $r_2$  Koordinate sowie den Richtungen `r1` und `r2`. Hiermit kann die komplette Transition abgefragt und somit den Methoden genau mitgeteilt werden, an welcher Stelle und in welcher Richtung ein gewisser Zug auf dem Spielfeld transferiert wird.

Sämtliche Informationen, wie die Anzahl an Bomben, Überschreibsteinen und die Spielernummer werden in einer `Player`-Klasse abgespeichert. Diese Werte müssen für jeden Spieler unabhängig voneinander gesichert werden. Zudem kann dort ein disqualifizierter Spieler markiert werden.

Die `Player`- und `Board`-Klasse wird von einer übergeordneten Klasse `Game` koordiniert, die sämtliche Spielabläufe steuert.

### 3 Spielfeldbewertung

Ein wesentlicher Aspekt der K.I. ist es die aktuelle Spielfeldsituation der unterschiedlichen Spieler zu bewerten. Dadurch ist es der K.I. möglich, Spielzüge besser einzustufen und gegebenenfalls das Spiel mit Spezialsteinen zu beeinflussen. Ein naiver Ansatz wäre der Vergleich der aktuellen Spielsteine jeden Spielers. Dieses Vorgehen reicht jedoch nicht für eine adäquate Bewertung der Spielsituation aus. In Abbildung ?? würde die naive Vorgehensweise den roten Spieler besser einstufen, jedoch hat er hier keinerlei mögliche Spielzüge. Der rote Spieler kann trotz dieser Überlegenheit nicht mehr gewinnen, da der blaue Spieler im nächsten Spielzug über alle roten Steine hinwegziehen kann. Genau aus diesem Grund reicht eine naive Spielfeldbewertung nicht für ein aussagekräftiges Resultat aus.

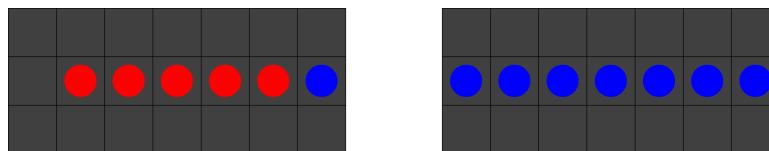


Abbildung 2: Problematik der naiven Bewertung

#### 3.1 Bestandteile und Implementierung

Wie bereits in der Einleitung gezeigt, genügt das reine Abzählen der Spielsteine nicht aus. Deshalb setzen wir auf drei unterschiedliche Heuristiken, um die aktuelle Spielsituation zu bewerten. Unter diesen drei Ansätzen werden die Mobilität der Spieler, das Verhältnis der Spielsteine sowie die aktuelle Bewertung der Karte miteinbezogen. Diese einzelnen Bestandteile sind während des Spiel jedoch nicht immer gleichbedeutend, da zum Beispiel im frühen Spielverlauf die Mobilität und das Besetzen von wichtigen Feldern von Vorteil ist. Im späteren Spielverlauf müssen die einzelnen Bestandteile anders gewertet werden. Hierbei muss darauf geachtet werden, seine Anzahl an Steinen zu erhöhen.

##### 3.1.1 Gewichtung des Spielfeldes

Es gibt gewisse Positionen, die für einen Spieler wertvoller sind als andere. Zu diesen Positionen zählen unter anderem Kanten und Ecken, da es wesentlich schwieriger, bis gar nicht möglich ist, diese einzunehmen. Eine Ausnahme stellt hier das Einnehmen mithilfe von Überschreib-, bzw. Spezialsteinen dar. Insbesondere sind Felder, die zwei Felder oder mehr von einem Bonusfeld in direkter Richtung entfernt sind, höher gewichtet, da sie die Möglichkeit geben einen solchen Bonusstein einzunehmen, falls ein Gegner auf ein direktes Nachbarfeld des Spezialfeldes zieht.

3	2	2	2	2	6	2	6	2	6	3
2	1	1	1	1	1	1	1	1	1	2
2	1	1	1	1	4	1	10	1	4	2
2	1	1	1	1	1	1	1	1	1	2
3	2	2	2	2	6	2	6	2	6	3

Abbildung 3: Spielfeldpositionen mit Gewichtungen

Der Score eines Spielers setzt sich dann aus der Summe der belegten Felder mit der entsprechenden Gewichtung zusammen.

Dieses Bewertungsverfahren bietet Vor- und Nachteile. Positiv daran ist, dass bei Beginn des Spieles jedem Feld eine Gewichtung zugeteilt wird und diese nur noch durch Änderungen von Spezialfeldern geringfügig geändert wird. Negativ ist jedoch, dass dieses Bewertungsverfahren besonders bei großen rechteckigen Spielfeldern mit wenig Spezialfeldern annähernd wie die naive Variante funktioniert.

### 3.1.2 Mobilität


Ein weiterer Bestandteil der Analyse besteht darin, Spielsituationen anhand der Beweglichkeit der Spieler einzustufen. Hierbei wird die Anzahl an Spielzügen eines jeden Spielers bestimmt und miteinander verglichen. Wie bereits in der Einleitung gezeigt kann ein Spieler seine positive Stellung nur halten, solange er weiterhin spielfähig bleibt. Aus diesem Grund wird in diesem Ansatz bestimmt wie beweglich ein Spieler gegenüber den Anderen ist.

Ein Vorteil dieser Bewertungsmethode besteht darin, dass ein Spieler mit wenig Steinen aber einer hohen Mobilität bei diesem Verfahren nicht negativ bewertet wird. Ein Problem daran ist jedoch, dass zum Ende des Spieles dieser Ansatz an Relevanz verliert, da zum Schluss nur die Anzahl an eigenen Steinen wichtig ist.

### 3.1.3 Spielfeldbelegung

Da es besonders gegen Ende des Spiels wichtig ist, viele Steine zu besitzen, muss man dies ebenfalls in die Bewertung miteinbeziehen. Anstatt aber nur einfach die Anzahl der Spielsteine zu zählen, wird hier das prozentuale Verhältnis gegenüber allen existierenden Spielsteinen genommen.

Der Vorteil dieses Verfahrens ist hier, dass man vor allem im späteren Spielverlauf feststellen kann, wie der aktuelle Stand des Spiels ist und welche Spieler es anzugreifen gilt. Der Nachteil hierbei ist aber, dass dieses Verfahren im anfänglichen und mittleren Spielverlauf nicht aussage-

gekräftigt ist. Allerdings muss man diese Berechnung wie Anfangs erwähnt mit einfließen lassen, da man zum chluss einen Greedyansatz wählen muss. Denn letztlich ist für das Gewinnen des Spieles nur die Anzahl an Spielsteinen von Bedeutung.



## 4 Fazit

Text..

## 5 Anhang

<b>Player</b>
- overrideStone: int - number: char - bomb: int
+ hasBomb(): boolean + hasOverrideStone(): boolean

<b>Board</b>
- allTransitions: HashMap«Integer, Transition» - bombRadius: int - field: char[][]
- choice(): void - bonus(player: Player): void - inversion(): void - colorizeMove(moves: List«Move», position: int[], player: Player) + getLegalMoves(Player, boolean): List«Move» + executeMove(x: int, y: int, player: Player, overrideMoves: boolean): void