

**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: 14.07.2021

Inhaltsverzeichnis

1 Einleitung	1
1.1 Besonderheiten von ReversiXT	1
1.2 Vorstellung des Wahlpflichtfaches	3
2 Allgemeine Informationen	4
2.1 Team und Kommunikation	4
2.2 Technische Daten	7
2.3 Datenstruktur	9
3 Spielfeldbewertung	10
3.1 Bestandteile der Heuristik	10
3.1.1 Gewichtung des Spielfeldes	11
3.1.1.1 Erreichbare Felder	11
3.1.1.2 Feldbewertung	11
3.1.1.3 Bewertung von Nachbarfeldern	12
3.1.1.4 Faktor der Felder	12
3.1.1.5 Bewertung von Spezialfeldern	13
3.1.2 Mobilität	13
3.1.3 Spielfeldbelegung	13
3.2 Algorithmen der Zugauswahl	15
3.2.1 Minimax Algorithmus	15
3.2.2 Alpha-Beta-Pruning	15
3.2.3 Vergleich der Algorithmen	16
3.2.4 Optimierung von Alpha-Beta-Pruning durch Zugsortierung	23
3.2.5 Erweiterte Zugsortierung	24
3.2.6 Harmlose Gegner	26
4 Spielanalyse	28
4.1 Rekonstruktion von Spielen	28
4.2 Erreichbarer Felder anzeigen	29
4.3 Detaillierter Statistiken anzeigen	30
4.4 Nutzen der Spielanalyse	30
5 Kartenanalyse	31
5.1 Ablauf eines Analyse Vorgangs	31
5.1.1 Erreichbarkeit in ReversiXT	31
5.1.2 Logisches Vorgehen	31
5.1.2.1 Suchen der Startpunkte	31
5.1.2.2 Folgen der möglichen Züge	32
5.2 Vorteile durch den MapAnalyzer	35
5.2.1 Leichtere Bewertung des Spielfortschritts	35
5.2.2 Herausrechnen von nicht erreichbaren Bonusfeldern	35
5.3 Herausforderung bei der Entwicklung	37
5.3.1 Performance Probleme	37
5.3.2 Fehlerfreies abgehen von Transitionen	38

5.3.3	Fehler bei der Berechnung	38
6	Karten für den Wettkampf	39
6.1	Comp2021 - Zweispielerkarte	39
6.2	Comp2021 - Dreispielerkarte	40
6.3	Comp2021 - Vierspielerkarte	41
6.4	Comp2021 - Achtspielerkarte	42
7	Verbesserungen für FightClub und MatchPoint	43
7.1	Direkte Konsolenausgabe	43
7.2	Export von Spielzuständen	43
7.3	Profilbasiertes Matchpoint	43
7.4	Integrierte Tooltips	44
7.5	Informationen über Fairness	44
8	Fazit	45
9	Anhang	46

1 Einleitung

ReversiXT ist eine Erweiterung des Brettspiels Reversi, welches in den 1880er Jahren vom Engländer Lewis Waterman [1] 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 1). Der rote 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 sind in der Grafik semi-transparent eingezeichnet. Sobald beide Spieler keine Spielzüge mehr haben, ist das Spiel beendet. Gewonnen hat derjenige, der die meisten Steine besitzt.

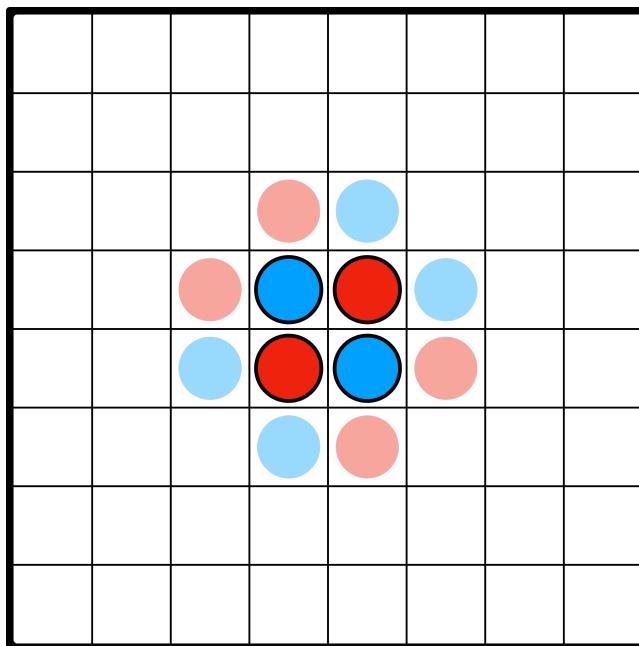


Abbildung 1: Grundspiel Reversi mit möglichen Startzügen (semi-transparent markiert)

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 *Bombenphase*. Zudem gibt es mehrere Spezialfelder (siehe Abbildung 2), die vorab fest definiert sind und jeweils nur einmal ausgelöst werden können. Mit einem

Inversionsfeld werden die Farben aller Spieler um eins weiter verschoben. Bei einem *Choicefeld* wird die Farbe von sich selbst mit einem ausgewählten Spieler getauscht, dabei 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 Notwendigkeit der Umfärbung anderer Spieler - 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 *Bonusfeld* darf der Spieler zwischen einem Überschreibstein und einer Bombe wählen, welche dem jeweiligen Spieler gutgeschrieben wird. Sobald kein Spieler mehr einen legitimen Spielzug besitzt, beginnt die zweite Phase. Hierbei müssen abwechselnd, sofern vorhanden, alle Bomben gesetzt werden. Hierzu wird ein Feld (Löcher ausgenommen) ausgewählt und ein Bereich mit festgelegtem Radius weggesprengt. Dabei werden alle betroffenen Felder zu *Löchern* umgewandelt. Löcher sind nicht bespielbare/besetzbare Felder.

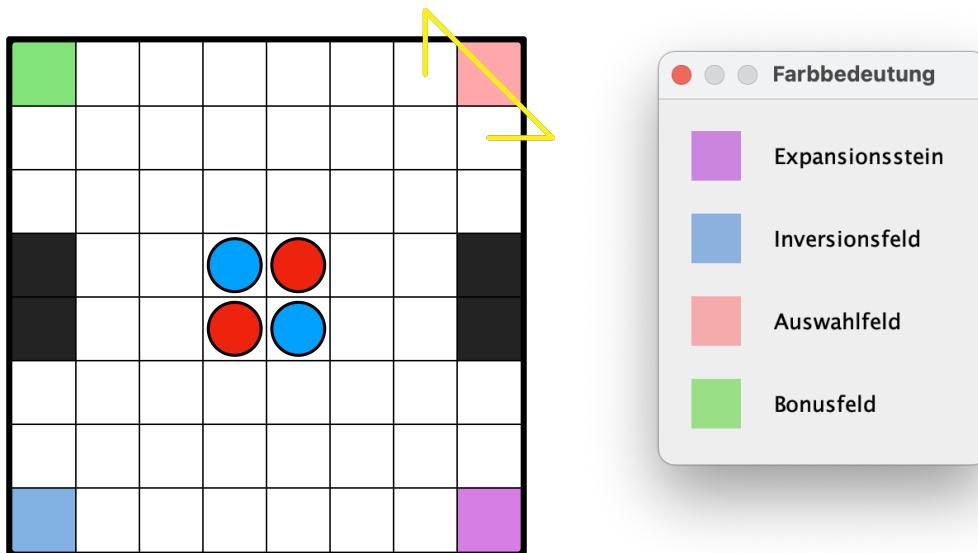


Abbildung 2: ReversiXT Karte inkl. eingezeichneter Transition

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. Befindet sich nun ein Spieler auf dem siebten Feld in der ersten Reihe (siehe Abbildung 2) und zieht nach oben, kommt er anschließend ganz rechts in der zweiten Reihe 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. ReversiXT Karten müssen zudem auch nicht zweidimensional sein, sondern können höherdimensionale Formen annehmen. Dadurch kann man sich eine solche Karte auch nicht mehr als Brettspiel vorstellen, was dazu führt, dass es nur sehr schwer nachvollziehbar ist, an welcher Stelle ein gewisser Spielzug endet. Aus diesen Gründen ist es für einen Menschen sehr schwer bis unmöglich ReversiXT erfolgreich gegen einen K.I. Client zu spielen.

1.2 Vorstellung des Wahlpflichtfaches

Die Aufgabe in diesem Modul besteht darin, einen Client mit einer künstlichen Intelligenz zur bestmöglichen Spielzugwahl zu entwickeln, da es Computern möglich ist, alle interessanten Spielzüge zu berücksichtigen und diese miteinander zu vergleichen. Wir erwarten uns von diesem Fach, dass es dem Team einen Überblick in die Entwicklung von künstlicher Intelligenz gibt. 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. Bei maschinellem Lernen werden Programme mithilfe von Daten trainiert, um dann daraus gewisse Muster und Zusammenhänge selbstständig zu verstehen. Damit nun z.B. eine Suchmaschine Katzenbilder von Hundebildern unterscheiden kann, muss sie vorab mit einer riesigen Menge solcher Bildern angelernt werden. In Bereichen, in denen es kaum Daten für ein zu lösendes Problem gibt, oder bei dem die Daten zu sehr voneinander abweichen kann maschinelles Lernen nicht eingesetzt werden. Betrachtet man nun ein Spiel wie ReversiXT und möchte hier nun maschinelles Lernen einsetzen entsteht kein Problem aufgrund fehlender Daten, sondern es ergibt sich ein ganz anderes Problem. Damit ein Computer selbstständig lernt, wie man ReversiXT spielt, muss dieser alle Möglichkeiten von Spielzügen durchgehen, um dann daraus Rückschlüsse über Züge zu erhalten, die zu einem Sieg führen. Nun gibt es aber so viele mögliche Züge, dass die Zeit nicht ausreicht um ein Spiel komplett durchsimulieren zu können und vor allem hat man dann auch nur ein einziges Spiel simuliert, was selbstverständlich alles andere als repräsentativ ist. Mit der Entwicklung einer künstlichen Intelligenz und einer Einführung über das maschinelle Lernen erhofft man sich die Vor- und Nachteile beider Innovationen erkennen und hautnah die Unterschiede verstehen zu lernen.

2 Allgemeine Informationen

Bei umfangreicherer 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 besser nachvollziehbar wie das Team Entscheidungen herbeigeführt.

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 kennengelernt und sich daraus viele Gemeinsamkeiten herausgestellt haben. Mithilfe von Signal, Zoom und Discord teilt das Team aktuelle Pläne und neue Erkenntnisse. Zudem wird mindestens einmal wöchentlich ein Meeting abgehalten, bei dem jeder Entwickler seine Änderungen dem Team vorstellt. 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 sich alle Beteiligten 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 kennengelernt, welches in diesem Projekt von großer Bedeutung ist. Alle Entwickler belegten zudem im letzten Semester das Fach Algorithmen & Datenstrukturen, in dem jeder einen umfangreichen Überblick über gängige Sortier- und Suchalgorithmen erhalten hat. Dank diesem Modul ist es für alle Mitwirkende wesentlich leichter unterschiedliche performancerelevante Stellen aufzuspüren und diese mithilfe von bekannten Algorithmen weiter zu optimieren, um eine bestmögliche Geschwindigkeit zu erreichen. Das Team schreibt alle Klassennamen, Variablen sowie Kommentare und Dokumentationen auf Englisch, 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. Zudem übernimmt er die Aufgabe, abgeschlossene Spiele zu analysieren und eventuelle schlecht Züge ausfindig zu machen. Dadurch kann das gesamte Team leichter Karten entdecken, die aktuell nicht hervorragend bespielt werden. Aus dem

oben genannten Grund, testet er auch die Karten auf fehlerhafte Züge und passt zudem die Parameter des Clients an. Benedikt kümmert sich zudem, dass regelmäßig Teambesprechungen stattfinden und teilt den Teammitgliedern die aktuellen Aufgaben der Woche mit. 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. Er versucht zudem eine gerechte Aufgabenverteilung für alle Beteiligten zu finden und schreibt diese in einen gemeinsamen Kalender. Des Weiteren übernimmt er die Fehlersuche in der Heuristik sowie deren Lösungen. Außerdem überarbeitet er die Architektur und führt größere Refactoring-Aufgaben durch. Markus Koch verfügt bereits über Erfahrungen in L^AT_EX, wodurch er den Teamkollegen bei der Einrichtung der benötigten Software hilft. Zudem übernimmt er die Aufteilung des gesamten Dokumentes um die Übersichtlichkeit zu verbessern. Er besitzt erste Kenntnisse in der Illustration von Bildern, weshalb er die Bilder für den Projektbericht gestaltet. Unter Anderem übernimmt er auch die Aufgabe, den E-Mail Verkehr zu überwachen und die Teammitglieder über E-Mails zu informieren und diese schnellstmöglich zu beantworten. Außerdem ist Markus sehr kontaktfreudig, weshalb er mit anderen Teams Gespräche über deren aktuellen Stand führt, um sich mit anderen Team besser vergleichen zu können.

Die Vorlesung findet immer dienstags statt. Im Anschluss wird im Team das neue Übungsblatt geprüft 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. (Das fertige Klassendiagramm befindet sich im Anhang.) In der nachfolgenden Tabelle sind alle zu erledigenden Aufgaben mit den entsprechenden Bearbeitern eingetragen.

Entwickler	Aufgabe
Benedikt, Iwan, Markus	Entwicklung der Datenstruktur
Benedikt, Iwan, Markus	Algorithmus zum Finden gültiger Züge
Benedikt, Iwan, Markus	Dokumentation einer Zugheuristik
Iwan, Markus	Erweitern gültiger Züge Spezialfelder
Benedikt	Verbesserung der Zugheuristik
Iwan	Implementierung der Zugheuristik
Markus	Erstellen des Projektberichtes (Kapitel 1–3)
Iwan, Markus	Implementierung der Netzwerkfunktionalität
Markus	Implementierung der Client-Parameter
Benedikt	Erstellen des Build-Files
Iwan	Implementierung der Paranoidsuche
Iwan	Implementierung von Alpha-Beta-Pruning
Markus	Erstellen der Statistik für den Projektbericht
Iwan	Implementierung abschaltbarer Zugsortierung
Iwan	Implementierung von Iterative-Deepening
Benedikt	Implementierung des MapAnalyzers
Markus	Implementierung des GameAnalyzers
Markus	Erweitern der Client-Parameter
Benedikt, Iwan, Markus	Erstellen von Anforderungen an den Fightclub
Benedikt, Markus	Überarbeitung des Projektberichtes
Benedikt, Markus	Implementierung der Bombenheuristik
Iwan	Implementierung von BRS, BRS+ und Killer Heuristik
Benedikt, Markus	Implementierung von Monte-Carlo-Tree-Search
Benedikt	Implementierung einer Spezialfeld-Berechnung
Iwan	Implementierung einer intelligenten Zugsortierung
Markus	Implementierung eines Heuristik-Controllers
Benedikt, Iwan, Markus	Überarbeitung des Projektberichtes

Tabelle 1: Übersicht der Aufgabenverteilung

2.2 Technische Daten

Als Entwicklungsumgebung wird von allen Teamkollegen das Programm IntelliJ von JetBrains [2] in der Version 2021.1 verwendet. Diese IDE bietet viele nützliche Zusatzfunktionen, wie z.B. das automatische Erstellen von UML-Klassendiagrammen, eine direkt integrierte Versionskontrolle, eine umfassende Code-Vervollständigung und ein profundes Plug-In mittels dessen man unkompliziert L^AT_EX 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, erhofft man sich weniger Kompatibilitätsprobleme bei der Entwicklung und eine 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 [3] 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 die Gruppe auf Gradle 6.3 [4]. Bei dieser Version handelt es sich bewusst um eine veraltete jedoch stabile Version da mehr Informationen als bei der neusten Version 7.0 zur Verfügung stehen und man sich dadurch zudem weniger auftretende Kompatibilitätsprobleme erhofft. Der Hauptunterschied von Gradle zu anderen Build-Systemen besteht in der Konfigurationsdatei. Gradle setzt hierbei auf Groovy, welches im Gegensatz zu XML wesentlich besser lesbar ist und auch einen *Continuous Mode* besitzt. Dadurch "[...] kann Gradle Änderungen am Quellcode überwachen und automatisch betroffene Artefakte neu bauen." [5, S. 219 ff.] Gradle erkennt somit Quellcode, der sich nicht geändert hat. Dieser muss anschließend nicht neu gebaut werden, wodurch sich eine viel kürzere Erstellungszeit ergibt. Das Projekt ist mit der Programmiersprache Java umgesetzt, da diese die gelernte Sprache aus Programmieren 2 ist und somit alle Beteiligten bereits umfangreiche Erfahrungen damit gesammelt haben und man sich somit nicht erst in eine andere Programmiersprache einarbeiten muss. Zum automatischen Testen der Softwarekomponenten wird außerdem das JUnit-Testframework [6] in Version 4 benutzt, da alle Teammitglieder 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 [7] getestet, ob das Programm unter Linux - der Zielplattform des Modules - 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
- Labor-PC
 - Prozessor: 3,70 GHz 6-Core Intel Core i7-8700K
 - Arbeitsspeicher: 32 GB 2400 MHz DDR4
 - Grafikkarte: NVIDIA GeForce RTX 2080
 - Festplatte: HDD 1TB - SATA III
 - Betriebssystem: Ubuntu 18.04.1

2.3 Datenstruktur

Das Spielfeld wird in der `Board`-Klasse in einem zweidimensionalen `char`-`Array` gespeichert. Das Team hat sich 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 der Server die Informationen über die Map als hexadezimalen Byte-Stream übermittelt, speichert der Client 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 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. Des Weiteren 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 berechnet:

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

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

Befindet sich nun eine Transition an der Position $x = 21$ und $y = 17$ in Richtung $r = 4$ (nach unten), so lautet der dazugehörige `Key` 21.174, wie er auch in Formel (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 und einer y Koordinate sowie der Richtung r . Befindet man sich auf einem Feld und möchte einen Zug in eine Richtung machen, in der sich jedoch kein weiteres Feld befindet, wird anhand dieser Informationen ein `HashKey` erzeugt und in der `HashMap` danach gesucht. Das Ergebnis ist nun ein Objekt der Klasse `Transition` in der die Position und die Richtung des Feldes enthalten ist, in die ein gewisser Zug auf dem Spielfeld transferiert wird.

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

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 kann die K.I. Spielzüge besser einstufen und gegebenenfalls das Spiel mit Spezialsteinen beeinflussen. Ein naiver Ansatz wäre der Vergleich der Anzahl an Spielsteinen jedes Spielers. Dieses Vorgehen reicht jedoch nicht für eine adäquate Bewertung der Spielsituation aus. In Abbildung 3 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.

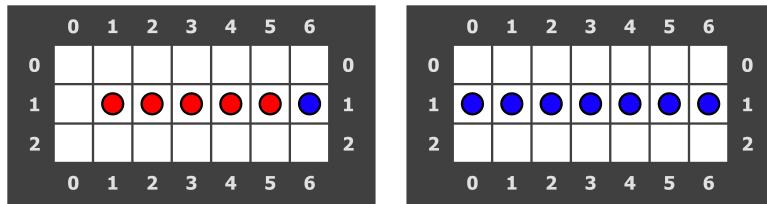


Abbildung 3: Problematik der naiven Bewertung

3.1 Bestandteile der Heuristik

Wie bereits in der Einleitung dieses Abschnitts begründet, genügt das reine Abzählen der Spielsteine nicht aus. Deshalb setzt dieser Client auf drei unterschiedliche Heuristikbestandteile, 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 Spiels jedoch nicht immer gleichbedeutend. Zu Beginn muss darauf geachtet werden, dass man sehr flexibel bleibt und Positionen erobert die später nur schwer einnehmbar sind. Hierzu zählen vor allem Ecken und Kanten. Erobert man nun ein solches Feld, wird der Gegner in seiner Zugwahl eingegrenzt, da er keine Züge wählen will, die bereits einen Zug später wieder überzogen werden. Eine solche Situation schränkt die Mobilität des Gegners drastisch ein und auch die Bewertung des Spielers aufgrund seines Kartenzwieltes leidet darunter. Je weiter man sich jedoch dem Spielende nähert, desto unwichtiger wird dieses Bewertungskriterium. Dies liegt daran, dass man das Spiel nicht aufgrund hoher Mobilität gewinnt, sondern anhand der Anzahl seiner Spielsteine. Aus diesem Grund wird das Verhältnis der Spielsteine zum Ende hin immer wichtiger. Maßgeblich ist nun den prozentualen Spielverlauf abzuschätzen, um damit die einzelnen Heuristikbestandteile akkurat gewichten zu können.

3.1.1 Gewichtung des Spielfeldes

Die Bewertung der Karte ist eine dynamisch verändernde Heuristik in ReversiXT. Diese Heuristik ist das Auge des Clients, da nur so begehrenswerte Felder wie Ecken oder Bonussteine erkannt werden können und diese enorme Vorteile während des gesamten Spielverlaufes geben. Ecken und Kanten sind dabei besonders interessant. Zwar sind sie schwächer als im originalen Reversi, da man Sie mit Überschreib-, Choice- oder Inversionsteinen immer noch einnehmen kann. Trotzdem können früh eingenommene Ecken und Kanten der Grundstein für ein dominantes Spiel sein und müssen deswegen kontrolliert werden.

3.1.1.1 Erreichbare Felder

Bevor die Karte bewertet oder irgendein Zug berechnet wird, kalkuliert der *MapAnalyzer*, welche Felder auf der Karte theoretisch erreichbar sind und welche nicht. In den nächsten Bewertungsschritten werden alle laut MapAnalyzer nicht erreichbaren Felder als Loch gesehen. Für diese Felder wird an sich kein Wert berechnet, da niemals ein Zug auf Sie stattfinden wird. Es können aber in die Bewertung anderer Felder einbezogen werden, was im nachfolgenden Kapitel beschrieben wird.

3.1.1.2 Feldbewertung

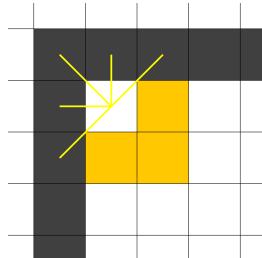


Abbildung 4: Bild einer Ecke in ReversiXT, mit 5 nicht bespielbaren Feldern.

Jedes einzelne erreichbare Feld wird bewertet und erhält einen Grundwert. Dieser Grundwert berechnet sich aus der Anzahl der nicht erreichbaren Nachbarfelder, multipliziert mit einem festen *Faktor* der die Feldbewertung nochmals anhebt. Ecken haben so zum Beispiel den Grundwert 5, da in ihrem direkten Umfeld fünf Felder nicht begehbar sind (siehe Abbildung 4). Dieser Wert wird anschließen noch mit einem festen Wert multipliziert, der sich für alle Abstufungen von Ecken ändert.

3.1.1.3 Bewertung von Nachbarfeldern

Bei der Feldbewertung werden nicht nur die Felder an sich bewertet. Die Bewertung wirkt sich auch auf benachbarte Felder aus. So kann er Client entscheiden, dass es eventuell nicht sehr schlau ist, direkt neben ein begehrtes Feld zu ziehen, und es damit dem Gegner zu ermöglichen einfach über einen drüberziehen um dieses Feld zu erobern. Um dem entgegenzuwirken, beeinflussen Felder auch ihre Nachbarfelder in alle Zugrichtungen. Das folgende Bild zeigt die Ausbreitung dieser Welle.

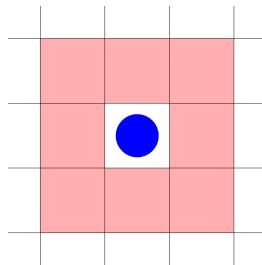


Abbildung 5: Ausbreitung eines Feldes rot markiert.

Das blau markierte Feld ist das zu analysierende Feld. Alle rot markierten Felder bekommen einen Malus auf ihren Wert. Somit sind sie für die Zugberechnung weniger attraktiv. Das blaue Feld hingegen erhält einen Bonus auf seinen Wert, da man es gerne erreichen möchte. Diese Welle breitet sich auch über Transitionen hinweg aus. Des Weiteren wird beim Ziehen auf ein Feld auch die Welle entfernt. Das ist besonders bei Spezialfeldern wichtig, da sobald dies ausgelöst wurden alle Felder im Umkreis wieder *normal* attraktiv sein müssen und keinen Bonus mehr aufgrund des Spezialfeldes benötigen. Diese Gewichtung wird durch den MapAnalyzer entfernt.

3.1.1.4 Faktor der Felder

In ReversiXT sind Ecken sehr viel Wert, doch bei einigen Karten gibt es sehr viele Ecken. Die Frage ist also, wie man diese einzelnen Ecken unterscheidet und nach Nutzen sortieren kann. Jede Ecke im Spiel hat einen Faktor der sich daraus berechnet, wie weit man von diesem Feld aus jeweils in eine Richtung gehen kann ohne auf ein Loch zu stoßen. Transitionen wird dabei natürlich gefolgt. Für jedes erreichte Feld wird nun der Faktor von 1.0 an um 0.1 erhöht. Die Folge ist, dass Ecken von denen man in verschiedenen Richtungen weit in die Karte ziehen kann höher bewertet und damit vom Client bevorzugt werden.

3.1.1.5 Bewertung von Spezialfeldern

Spezialfelder machen ReversiXT aus. Sie haben einen enormen Einfluss auf das Spiel und können Partien schnell drehen. Deshalb ist die Kontrolle und Priorität dieser Felder mit Bonussteinen extrem wichtig. Wie wichtig die einzelnen Bonussteine sind, wurde vom Team gemeinsam in den ersten Stunden beschlossen und folgende Priorität festgelegt:

$$\text{Bonusstein} > \text{Choice} > \text{Inversion} \quad (3)$$

Der Bonusstein wird periodisiert, da so in den letzten Zügen der ersten Phase nochmals sehr viele Steine umgefärbt werden können. Die Bewertung der Bonusfelder ist genau so wie bei einem *normalen* Feld. Das zu berechnende Feld bekommt je nachdem, ob es ein Choice-, Inversion- oder Bonusfeld ist, einen Bonuswert, der es bei der Zugauswahl extrem attraktiv macht. Da der Feldwert des Bonusfeldes sehr hoch ist, wirkt sich dies natürlich auf die umstehenden Felder aus, weshalb die angrenzenden Felder dementsprechend unattraktiv sind. Zusätzlich gibt es bei der Bonusfeldbewertung die Möglichkeit, Felder als aktiviert zu setzen, wodurch die Welle auf diesem Feld rückgängig gemacht wird. Dies ist nötig, um sicherzustellen, dass die K.I. nicht durch bereits aktivierte Spezialfelder beeinflusst wird.

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 entscheidend ist.

3.1.3 Spielfeldbelegung

Da es besonders gegen Ende des Spiels essenziell 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 aussagekräftig ist. Dies liegt daran, dass man besonders zum Spielende durch eine zuvor sehr gute Mobilität einen vermeintlich besseren Spieler sehr viele Steine wegnehmen kann. Allerdings muss man diese Berechnung wie Anfangs erwähnt mit einfließen lassen, da man zum Schluss einen Greedyansatz wählen muss. Denn letztlich ist für das Gewinnen des Spieles nur die Anzahl an Spielsteinen von Bedeutung.

3.2 Algorithmen der Zugauswahl

Für einen Computer ist es unmöglich den besten Spielzug zu finden, da hier jedes mögliche Spielfeld überprüft werden müsste. Bei einem Spiel wie Schach wären dies circa 10^{120} unterschiedliche Spielbrettzustände [8]. Bei ReversiXT liegt die Zahl wesentlich höher, da es zum einen viel größere Karten und zum anderen bis zu 8 Spieler gibt. Damit ein Computer trotzdem einen guten Zug abliefern kann werden spezielle Algorithmen benötigt. Ein weit verbreiteter Algorithmus für Zwei-Personen-Nullsummenspiele¹ lautet Minimax.

3.2.1 Minimax Algorithmus

Zur Veranschaulichung von Minimax soll ein kurzes Gedankenexperiment (siehe Abbildung 6) dienen. Es gibt zwei Personen. Die eine hat zwei Schubläden A & B mit je zwei Gegenständen (Wert in Euro angegeben). Die andere Person darf sich nun eine Schublade aussuchen, aus dem Sie ein Geschenk bekommt. Nun muss der Besitzer dieser Gegenstände auswählen, welches er dem Glücklichen überreichen möchte. Deshalb wird er den Gegenstand wählen, der für Ihn am wenigsten Verlust darstellt. Aus diesem Grund ergibt es für den Spieler YOU keinen Sinn die Schublade B auszuwählen, da man hier den Gegenstand für 1 € bekommt, andernfalls den für 20 €.

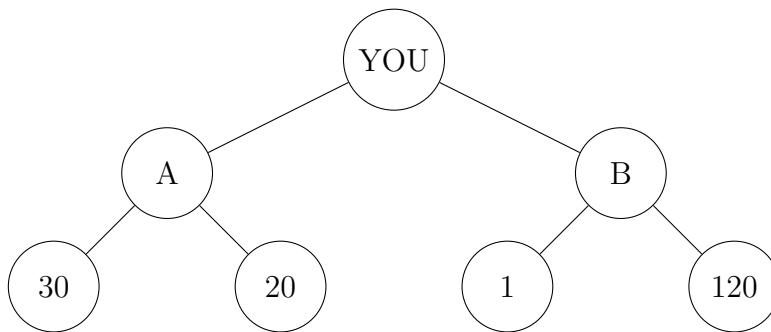


Abbildung 6: Grafik zur Veranschaulichung des Gedankenexperimentes

Ein solches Verfahren kann nun genutzt werden um unterschiedliche Spielzüge zu vergleichen und den Bestmöglichen - unter Berücksichtigung der späteren Züge des Gegners - auszuwählen. Jedoch beinhaltet dieser Algorithmus einen Nachteil - es werden auch Spielzüge betrachtet, die nicht infrage kommen, da ein Spieler diesen Zug niemals wählen wird.

3.2.2 Alpha-Beta-Pruning

Dieses Problem kann mithilfe des Alpha-Beta-Prunings drastisch reduziert werden. Hierbei werden Hilfsvariablen (alpha und beta) durchgereicht. Diese Verbesserung muss Zugfolgen, die das

¹Duden: Spiel, bei dem die Summe der Einsätze, Verluste und Gewinne gleich null ist

Ergebnis unbeeinflusst lassen, nicht mehr kontrolliert. Beim Gedankenexperiment muss somit nach dem Gegenstand im Wert von 1 € kein weiterer Knoten untersucht werden, da der Spieler YOU diesen Pfad (im Beispiel die Schublade B) nicht wählen wird. Das liegt daran, dass für ihn die Schublade A wesentlich interessanter ist. Selbstverständlich nur unter Berücksichtigung, dass der Andere seinen Verlust minimieren möchte.

3.2.3 Vergleich der Algorithmen

Um die Bedeutung dieser Variante zu verdeutlichen, folgt nun ein Vergleich beider Verfahren. Dabei wird zuerst dieselbe Karte mit unterschiedlichen Suchbaumtiefen getestet. Es treten in den Tiefen 3 bis 8 jeweils zwei Spieler gegeneinander an². Es wird je einmal der erste Spieler und einmal der zweite Spieler mit Alpha-Beta gestartet.

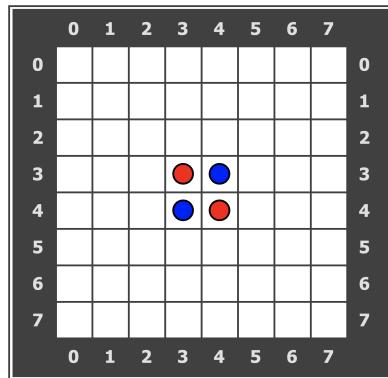


Abbildung 7: Verwendete Karte für den Vergleich von Minimax und Alpha-Beta

Es wird aus folgenden Gründen die Karte aus Abbildung 7 zu Beginn gewählt:

1. Es gibt nur zwei Spieler, womit beide Verfahren gegeneinander antreten können.
2. Es gibt keine Spezialsteine, was das Spiel einfacher gestaltet und beim Vergleich der Algorithmen vorerst nicht von Bedeutung ist.
3. Die verwendete Karte ist komplett symmetrisch, wodurch eine Fairness für beide Spieler garantiert wird.
4. Es handelt sich um eine relativ kleine Karte, damit die Berechnungen zeitlich einigermaßen eingegrenzt werden können.

²Die Suchbaumtiefe 8 wurde auf einem Labor-PC der OTH Regensburg durchgeführt. Nähere Informationen siehe Kapitel 2.2

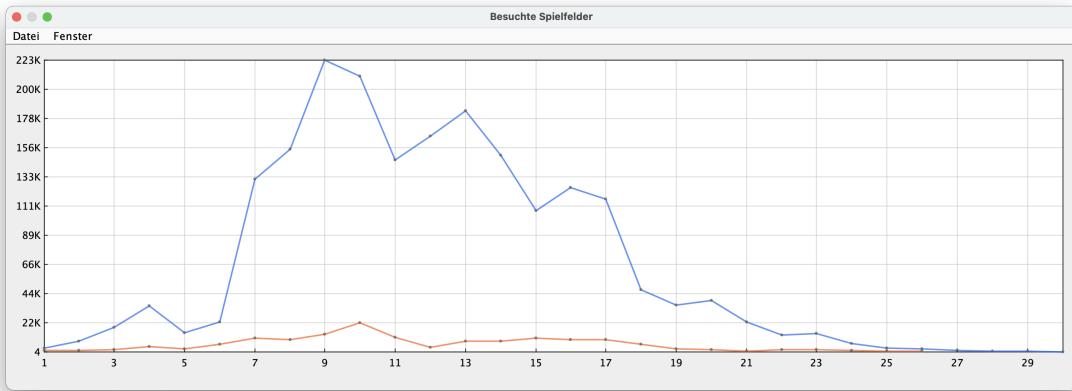


Abbildung 8: Diese Statistik wurde bei einem Spiel der Tiefe 5 aufgenommen.

In Abbildung 8 sind sowohl überprüfte Karten ohne Alpha-Beta (blaue Linie), als auch mit Alpha-Beta (orange Linie) zu sehen. Hierbei ist klar ersichtlich, dass Alpha-Beta-Pruning einen enormen Leistungsvorteil liefert. Der obige Screenshot stammt aus der selbstentwickelten Software GameAnalyzer. Mehr dazu in Kapitel 4.

Die Tabelle 2 liefert eine Übersicht über alle getesteten Suchbaumtiefen dieser Karte.

Tiefe	Alpha-Beta	Minimax
3	1.079,5	2.434,5
4	5.985,5	18.417,5
5	22.653	244.854,5
6	133.657,5	1.313.347
7	419.537	28.802.230
8	1.898.497	449.424.275

Tabelle 2: Besuchte Karten mit den Suchalgorithmen (Ergebnisse wurden geglättet)

Auf den ersten Blick wirken diese Zahlen nun ziemlich bedeutungslos. Werden Sie jedoch in ein Koordinatensystem (siehe Abbildung 9) eingetragen und eine Linie eingezeichnet, kann man den Verlauf bei weiteren Suchbaumtiefen beider Algorithmen erahnen.

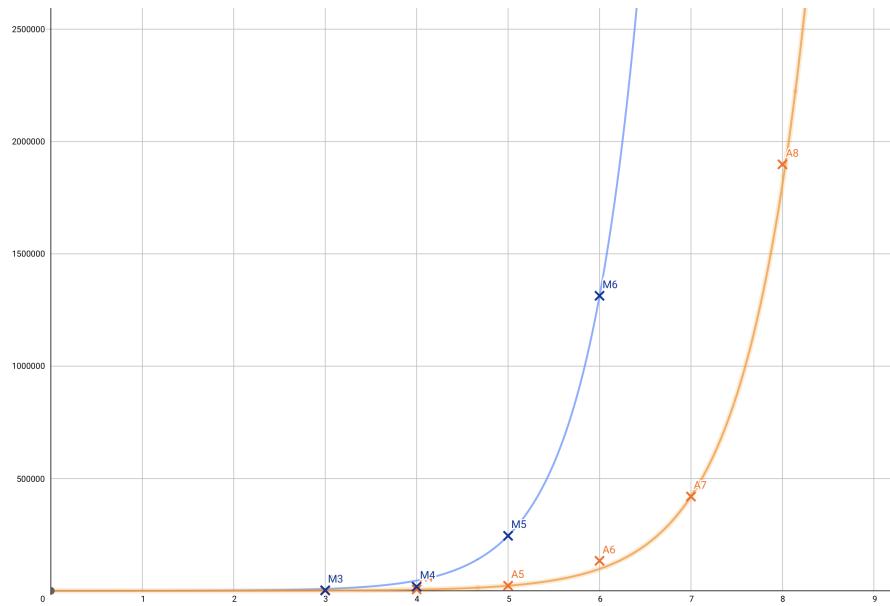


Abbildung 9: Suchbaumtiefen in ein Koordinatensystem eingetragen

Der Aufwand, alle möglichen Spielzustände zu vergleichen, wächst bei beiden Verfahren exponentiell mit der Anzahl der Suchbaumtiefe. Jedoch bewirkt das Alpha-Beta-Pruning eine gestauchtere Kurve als der Minimax-Algorithmus ohne Alpha-Beta-Pruning.

Bei gleicher Anzahl an bewerteten Feldern wird mit Alpha-Beta-Pruning der Suchbaum tiefer analysiert und dadurch mehr Züge miteinander verglichen. Das führt dazu, dass bei gleichem Zeitaufwand ($\hat{=}$ gleiche Anzahl an analysierten Spielzuständen) das beste Ergebnis einer tieferen Suchbaumebene gefunden wird.

Um diese Aussage adequate zu bestätigen sind vier weitere Karten (siehe Abbildung 14) aufgeführt, bei denen erneut Tests durchgeführt werden. Die Statistiken zum Vergleich der Algorithmen befindet sich jeweils unterhalb der textuellen Vorstellung der einzelnen Karten. Die blaue Linie stellt hierbei immer den Minimax-Algorithmus und die orangefarbene Linie das Alpha-Beta-Pruning dar.

Reversi Cuboid: Diese Map basiert auf dem Grundspiel von Reversi, jedoch sind hier vier Bereiche angeordnet. Durch diese Anordnung ist diese Map fair und es gibt einige Spezialfelder, die jedoch erst im späteren Verlauf des Spieles erreichbar sind. Durch diese Karte soll getestet werden wie sich 8-Spieler-Karten auf die beiden Algorithmen auswirkt. Diese Karte wird aufgrund vieler Spieler auf Tiefe 4 durchlaufen. Dabei wird als erster und zweiter Spieler der Client zum Testen der Algorithmen verwendet. Die anderen Spieler verwenden den triviale Client, welcher zu Beginn des Semesters jedem Team ausgehändigt wird.

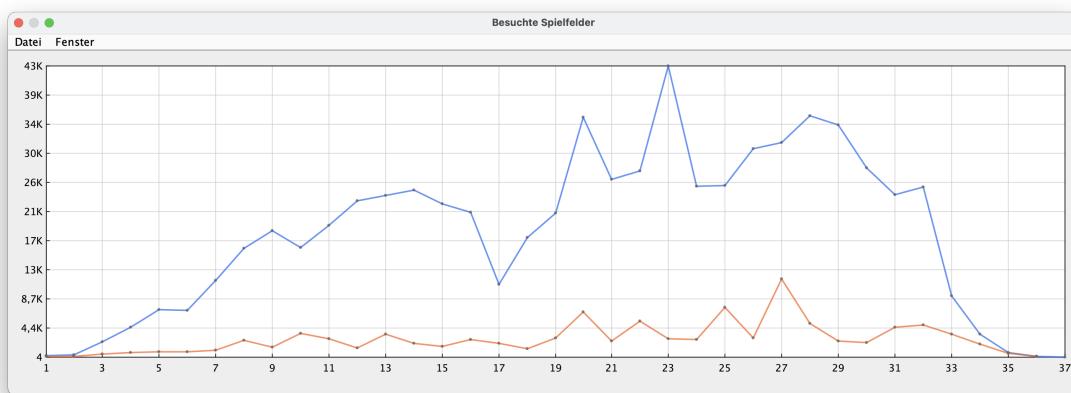


Abbildung 10: Vergleich der Algorithmen auf der Karte Reversi Cuboid

Cupcake: Diese Map hat bereits zu Beginn sehr viele unterschiedliche Startpositionen. Was dazu führt, dass der Suchbaum bereits zu Beginn sehr breit ist und somit sehr stark anwächst. Er beinhaltet jedoch keine Transitionen oder Sonstige Spezialfelder, wodurch die Anzahl damit jedoch zu einem gewissen Teil eingeschränkt werden. Die Karte wird auf Tiefe 4 durchlaufen.

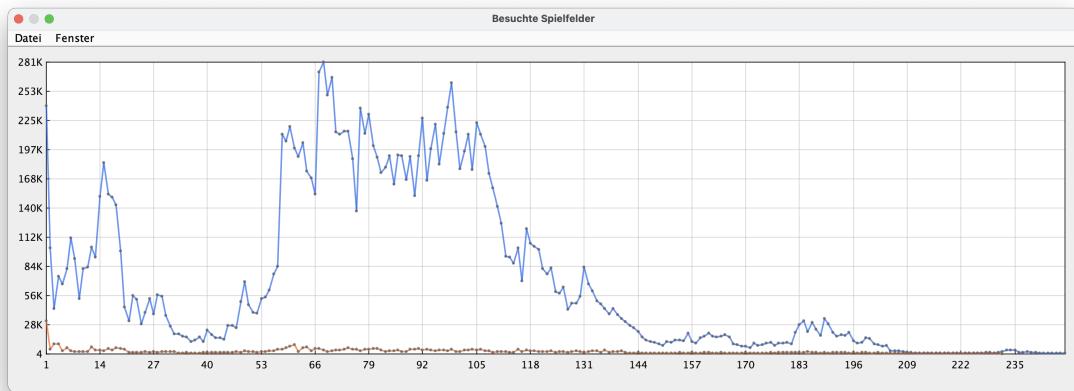


Abbildung 11: Vergleich der Algorithmen auf der Karte Cupcake

Dog Extended: Bei der Dog Extended Karte handelt es sich um die Karte, die für alle Abgaben während dem Semester verwendet wird. Aus diesem Grund wird auch die Map Dog Extended bei diesem Test überprüft. In dieser Karte sind Transitionen sowie alle möglichen Spezialsteine integriert um alle Abhängigkeiten zu testen. Diese Karte wird auf Tiefe 5 durchlaufen.

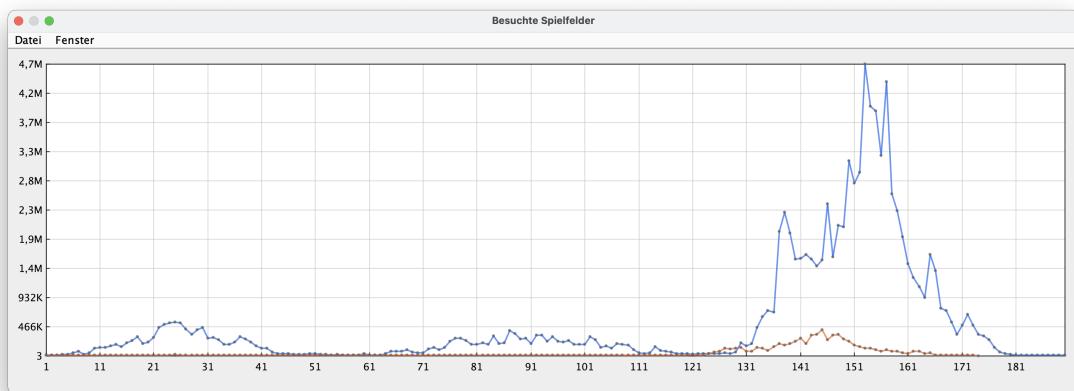


Abbildung 12: Vergleich der Algorithmen auf der Karte Dog Extended

Europa: Die Europa Karte ist eine sehr große Map bei der jedoch viele Stellen aus Löchern bestehen und es zudem eine Menge an Kanten und Ecken gibt. Dadurch wird die Spielfeldgewichtung unter Probe gestellt um zu testen ob es hierbei zu starken Abweichungen kommt, da die Gewichtung hier oft gleiche Ergebnisse liefern könnte und somit wenig weggekürzt werden kann. Diese Karte wird auf Tiefe 3 durchlaufen, da es eine sehr große Karte mit drei Spielern ist. Dabei wird als erster Spieler der triviale Client verwendet.

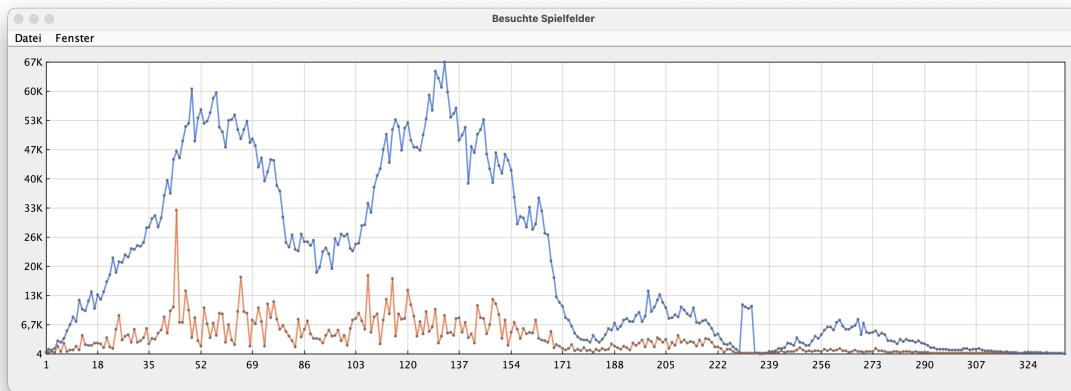
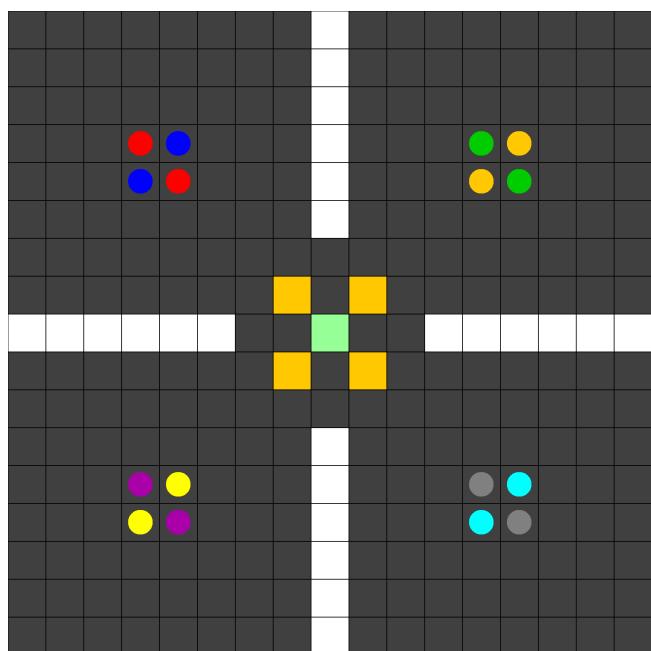


Abbildung 13: Vergleich der Algorithmen auf der Karte Europa

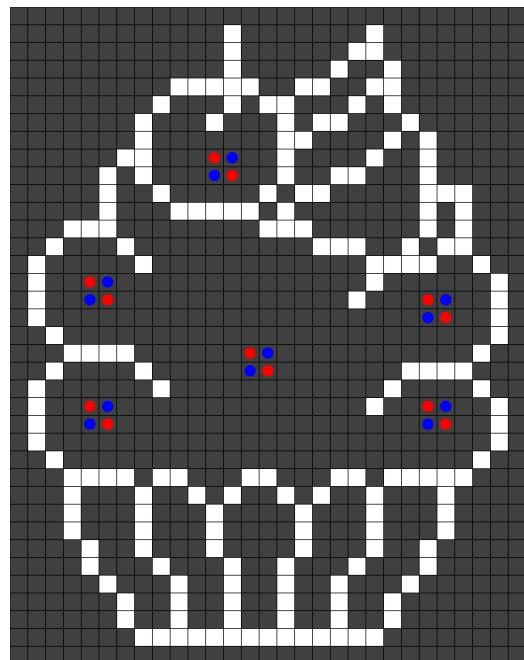
Karte	Tiefe	Alpha-Beta	Minimax
Cupcake	4	30.142	281.694
Reversi Cuboid	6	11.459	48.656
Dog Extended	5	978.310,5	4.357.785
Europa	3	34.365	64.316,5

Tabelle 3: Tabellarischer Vergleich der aufgeführten Statistiken (Ergebnisse wurden geglättet)

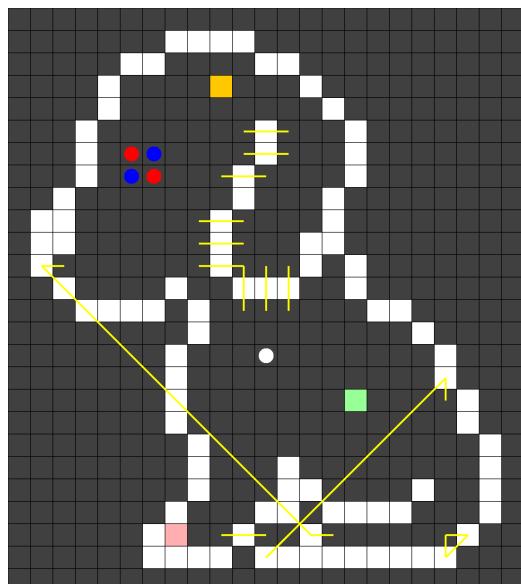
Betrachtet man nun alle aufgeführten Karten kann man zwar jeweils einen Unterschied in der Güte des Alpha-Beta Prunings feststellen, jedoch wird hier klar ersichtlich, dass sich bei jeder getesteten Karte ein gewisser Leistungsvorteil ergibt. Das Team hat sich genau aus diesen Gründen dazu entschieden, auf jeder Karte standardmäßig Alpha-Beta Pruning zu aktivieren.



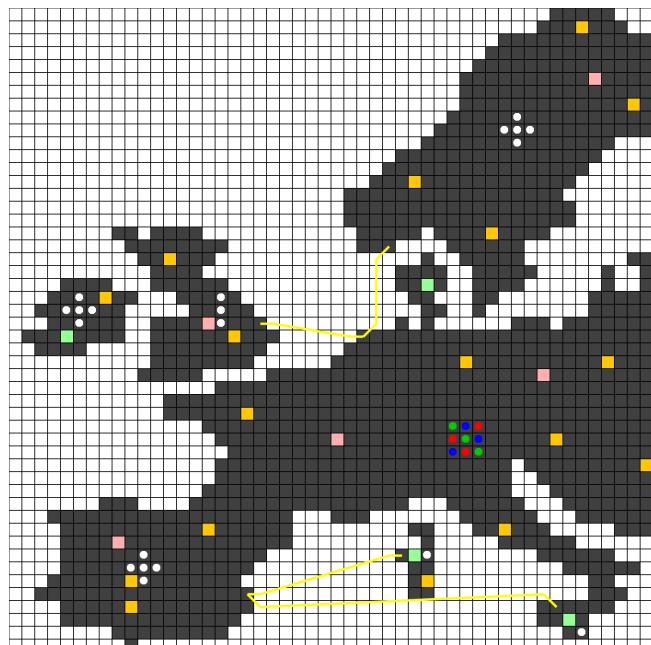
(a) Karte: Reversi Cuboid



(b) Karte: Cupcake



(c) Karte: Dog Extended



(d) Karte: Europa

Abbildung 14: Weitere Karten die für die Statistik verwendet werden

3.2.4 Optimierung von Alpha-Beta-Pruning durch Zugsortierung

Wie man im Abschnitt 3.2.2 gesehen hat, hilft Alpha-Beta-Pruning stark den Suchbaum zu verkürzen. Jedoch wird dieser Baum bei Spielen mit mehr als zwei Spielern so groß, dass man hier ebenfalls auf Grenzen stößt.

Eine Möglichkeit Alpha-Beta-Pruning zu optimieren ist, die Züge vorab zu sortieren. Die Idee besteht darin, einen vermutlich starken Zug gleich am Anfang im Suchbaum zu analysieren, um einen möglichst hohen Alpha-Wert zu bekommen und dadurch alle weiteren *schlechteren* Züge zu überspringen. Der Client sortiert also alle in seinem Spielzug möglichen Züge absteigend anhand der Heuristik und beginnt mit dem laut der Heuristik stärksten Zug.

Die Graphik 15 verdeutlicht was damit gemeint ist. Man sortiert hier als blauer Spieler die Züge so, dass der grün markierte Zug zuerstes betrachtet wird. Dieser ist nämlich besonders stark, weil man durch ihn eine Ecke gewinnt. Somit ist hier der **MapValue** (\equiv Spielfeldgewichtung) am Größten, da Ecken eine sehr hohe Bewertung besitzen. Zudem gewinnt man zwei neue Steine, wodurch die **CoinParity** (\equiv Spielfeldbelegung) steigt. Dies reicht aus, um einen hohen Alpha Wert zu bekommen und im Suchbaum die schlechten Züge zu überspringen.

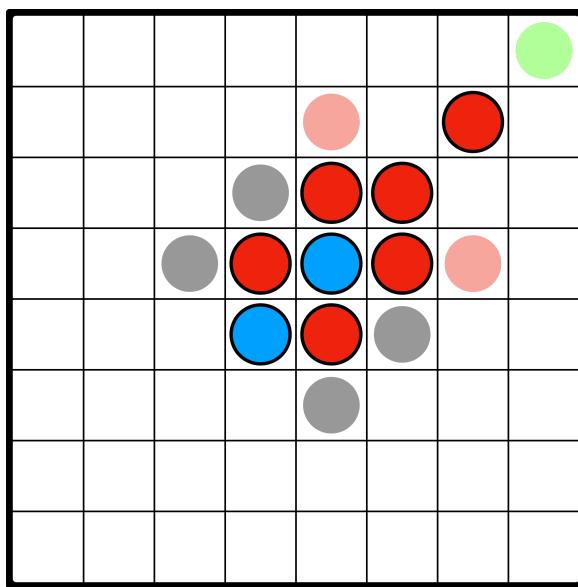


Abbildung 15: Mögliche Züge für Spieler blau sortiert in unterschiedlichen Farben.

Die zwei rot markierten Züge betrachtet man als letztes, da beide Züge genau ein Feld von der Kante entfernt sind und dadurch einen negativen MapValue haben. Dies sollte den Client davon abhalten dort hinzuziehen und das Risiko einzugehen, die Kante an den Gegner zu verlieren.

Die Zugsortierung hat jedoch auch Nachteile, die sich aus den Nachteilen der Spielfeldgewichtung aus 3.1.1 ergeben. Wenn man nämlich auf einer großen Karte spielt und es nur wenige Ecken und Kanten gibt dann sind die meisten Felder mit einer Gewichtung von 0 bewertet.

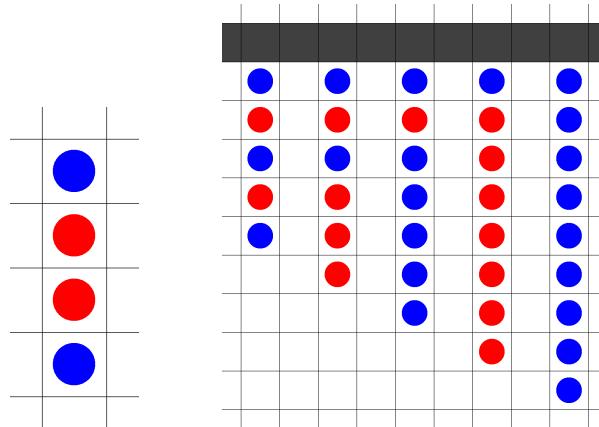


Abbildung 16: Simulation eines Spielverlaufes.

Somit sind alle Zügen vom MapValue gleich und unterscheiden sich nur durch die CoinParity und Mobility (\equiv Mobilität). Wobei hier auch oft viele Züge die gleichen Werte haben. Da dieser Fall doch häufig auftritt, sucht man oft ohne der Hilfe der Zugsortierung.

3.2.5 Erweiterte Zugsortierung

Die erweiterte Zugsortierung soll helfen das Problem der normalen Zugsortierung zu lösen und eine bessere Unterscheidung für vermutlich gute und schlechte Züge zu liefern. Deswegen wurde eine komplett neue und unabhängige Heuristik für die Zugsortierung entwickelt.

Wenn man sich die Züge in ReversiXT anschaut, dann gibt es nur zwei Zugarten. Die erste Art verdeutlicht Abbildung 16a. Der blaue Spieler kann nicht ziehen, das ist jedoch nicht schlimm, weil egal welchen Zug der Rote macht, der Blaue hat immer eine Antwort, die ihn gewinnen lässt. Der Grund dafür ist, dass der blaue Spieler hat einen Stein an beiden Enden der Spielstein-Linie. Dadurch kann er die Züge von Rot immer kontern. Der Blaue hat die volle Kontrolle über diese Art von Anordnung. Abbildung 16b zeigt eine im Grunde ähnliche Situation zu Abbildung 16a. Der Rote kann hier ebenfalls nicht gewinnen, egal wie er zieht, solange der Blaue ihn kontert. Der Grund hierfür ist wieder, dass der Blaue an beiden Enden der Linie seine Spielsteine hat. Es ist auch egal wie die anderen Spielsteine innerhalb der Linie angeordnet sind und wie sie variieren. Wenn man das Ganze aus der Sicht des roten Spielers betrachtet, dann sollte man Züge, die zu dieser Kategorie gehören vermeiden, da man in jedem Fall ein Verlust erleidet, zumindest wenn der Blaue richtig darauf reagiert.

Abbildung 17a zeigt die nächste Situation. Hier ist es stark davon abhängig wer ziehen darf. Derjenige der anfängt wird die Situation für sich entscheiden können und überführt die Anordnung zu einer wie in Abbildung 16a oder Abbildung 16b beschrieben wurde. Abbildung 17b

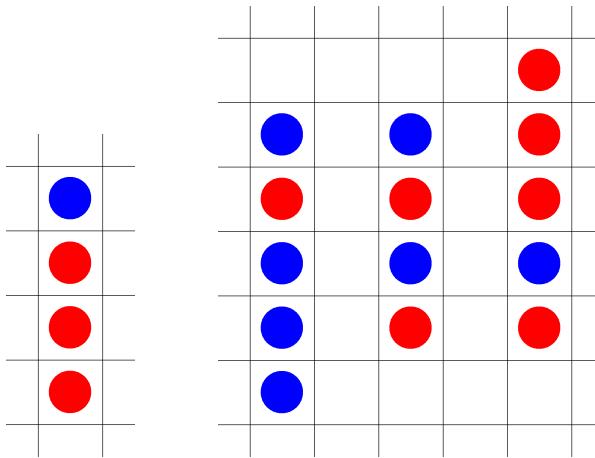


Abbildung 17: Simulation eines Spielverlaufes.

ist ebenfalls die gleiche Situation, hier ist nur wichtig das an beiden Enden der Spielstein-Linie unterschiedliche Steine sind.

Der Client teilt also die Züge vorerst in die Zwei Kategorien *sicherere* und *unsicher* Züge ein.

Die Abbildung 18 zeigt, wie der Client für Spieler blau die Züge kategorisieren würde. Die grün markierten Züge sind somit alle *sicher* und die schwarzen *unsicher*. Im Suchbaum würde man also nur noch die drei grünen Züge anschauen und die vier schwarzen Züge komplett weglassen. Das reduziert die Breite des Baumes am stärksten, weil bereits in Tiefe 0 des Suchbaumes Knoten weggelassen werden.

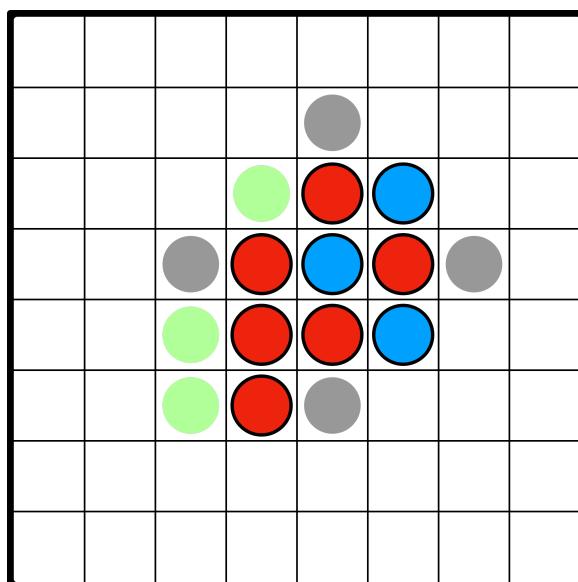


Abbildung 18: Mögliche Züge für Spieler blau sortiert in unterschiedlichen Farben.

Die Einteilung in *sicher* und *unsicher* Züge wurde jedoch noch auf folgende Kategorien weiter verfeinert:

- Eck-Züge: Alle Züge die in eine Ecke führen. Diese sind ebenfalls sehr sicher, da sie nur mit einem Überschreibstein zurückerober werden können. D.h. auch wenn am anderen Ende ein gegnerischer Spielstein ist, kontrolliert man die Situation trotzdem.
- Kanten-Züge: Alle Züge die in ein Kante führen. Züge, die entlang einer Kante gehen zählen nicht dazu. Diese Züge sind vorerst auch sicher, da sie in Zugrichtung nicht mehr erobert werden können, außer der Gegner benutzt einen Überschreibstein. Diese sind aber schwächer als Eck-Züge, da sie durch normale Spielzüge entlang der Kante erobert werden können. Das ist bei Eck-Zügen nicht möglich.
- Gefangene-Züge: Sind alle Züge die eine Situation wie in Abbildung 16a entstehen lassen. Man ist also gefangen zwischen zwei gegnerischen Steinen und sollte aus dieser Situation im nächsten Zug nicht mehr herausziehen. Der Gegner kann jedoch in der Zugrichtung nicht mehr angreifen.
- Gute-Züge: Sind alle Züge, die zu Situationen wie in Abbildung 16a oder Abbildung 16b führen. Man hat danach also die Kontrolle über den Gegner, zumindest nur in der Zugrichtung.
- Schlechte-Züge: Sind alle Züge die vorher als *unsicher* beschrieben wurden. Also Züge die vollständig gekontert werden können.

Ausnahmen bilden Züge bei denen man einen Bonus-, Choice- oder Inversionstein bekommt. Diese Züge werden unabhängig von den beschriebenen Kategorien betrachtet. Bonussteine werden immer sofort genommen, wenn das im aktuellen Zug möglich ist. Choicesteine werden ebenfalls sofort ausgelöst. Die Heuristik ermittelt dann anhand des MapValues, der CoinParity und der Mobility den besten Spieler mit dem getauscht werden soll. Inversionsteine werden nur genommen, wenn die Bewertung des getauschten Spielers mindestens zu 95% der aktuellen eigenen Bewertung entspricht.

3.2.6 Harmlose Gegner

In ReversiXT gibt es manchmal die Situationen, dass man am Anfang des Spiels erst nur gegen einen Spieler kämpft und nur im Laufe des Spiels auf andere trifft. Ein Beispiel hierfür zeigt Abbildung 19.

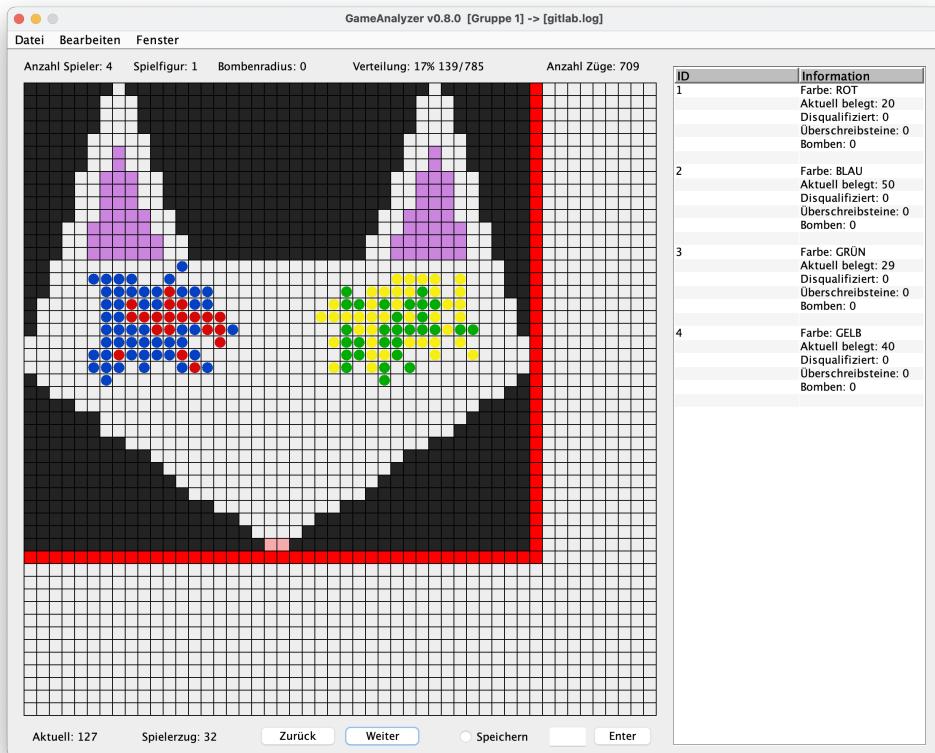


Abbildung 19: Spiel mit zwei vorübergehenden unabhängigen Spielen

Auf dieser Karte startet man in zwei Gruppen und wird erst später auf andere treffen. Für den Suchbaum macht es also keinen Sinn, alle Züge vom Gegner zu betrachten die aktuell gar nicht die eigenen Spielsteine angreifen können. In solchen Situationen kann man also die vorerst *harmlosen* Gegner im Suchbaum überspringen, in dem man von ihnen nur einen zufälligen Zug ausführt. Durch diese Taktik reduziert man quasi den Baum nur auf Gegner die einen direkt betreffen.

Diese Optimierung funktioniert jedoch nicht mehr, wenn alle Gegner einen angreifen können.

4 Spielanalyse

Um den bestmöglichen Client für ein Spiel zu entwickeln, ist es von besonderer Bedeutung Fehler zu entfernen und stetig die Qualität der Implementierung zu erhöhen. Das Team hat bereits zu Beginn festgestellt, dass es nur sehr schwer nachvollziehbar ist, abgeschlossene Spiele zu analysieren. Wird man nun durch fehlerhafte Züge disqualifiziert, oder der Code stürzt bei einer bestimmten Stelle ständig ab, ist es sehr schwer diese Situation wiederherzustellen. Zudem sind die aktuellen Informationen über die Heuristik kaum aus gespielten Spielen herauszulesen. Genau aus diesen Gründen wird die Software GameAnalyzer entwickelt.

4.1 Rekonstruktion von Spielen

Ein wesentlicher Aspekt liegt darin, ein gespieltes Spiel Schritt-für-Schritt nachspielen zu können. Dies ist auch der erste wichtige Bestandteil dieser Software. Wie man in Abbildung 20 sehen kann, wird gleich zu Beginn ein Spiel der gewählten Gruppe eingelesen und alle ausgewählten Züge nachgeahmt.

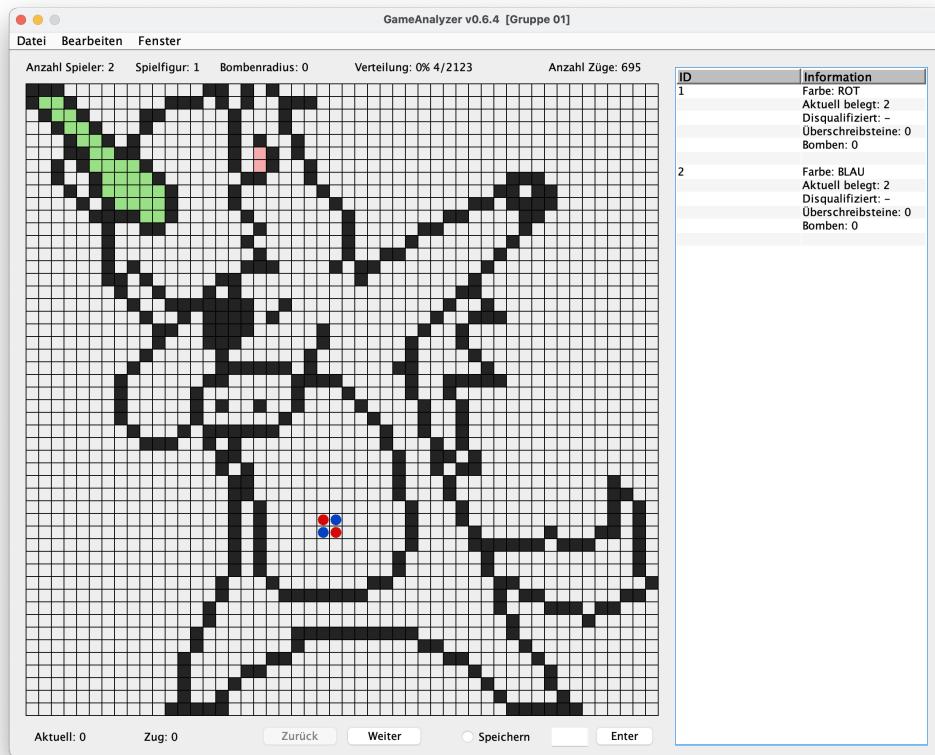


Abbildung 20: Startbildschirm GameAnalyze nach Auswahl einer Logdatei.

Mit den Tasten Weiter und Zurück kann man das Spiel in beide Richtungen durchlaufen. In der rechten Tabelle werden alle Spieler angezeigt. Zudem kann man darin nachschauen wie viele Überschreibsteine, Bomben bzw. Spielsteine sie jeweils aktuell haben. Ebenfalls ist zu sehen, welche Farbe sie haben und gegebenenfalls wann sie disqualifiziert werden. Man erhält weiterhin Informationen darüber, wie viele Spielzüge es gibt und wie viel Prozent an Spielsteinen aktuell belegt sind. Zudem kann unter Datei → Exportieren der aktuelle Stand der Karte exportiert werden, damit der Client die Karte mit dem gewünschten Spielstand wieder einlesen kann. Falls Transitionen existieren, können diese unter Bearbeiten ein- bzw. ausgeschaltet werden.

4.2 Erreichbarer Felder anzeigen

Damit keine Spielfelder betrachtet werden müssen die auf dieser Karte garnicht erreichbar sind, entwickelt unser Team einen MapAnalyzer, der zu Beginn des Spieles unerreichbare Positionen aus der Karte herausrechnet. Welche Felder erreichbar bzw. nicht erreichbar sind kann man sich im GameAnalyzer ebenfalls anzeigen lassen. Dazu geht man auf Fenster → Erreichbare Spielfelder. Nun wird ein Fenster geöffnet, wie in Abbildung 21 zu sehen ist.

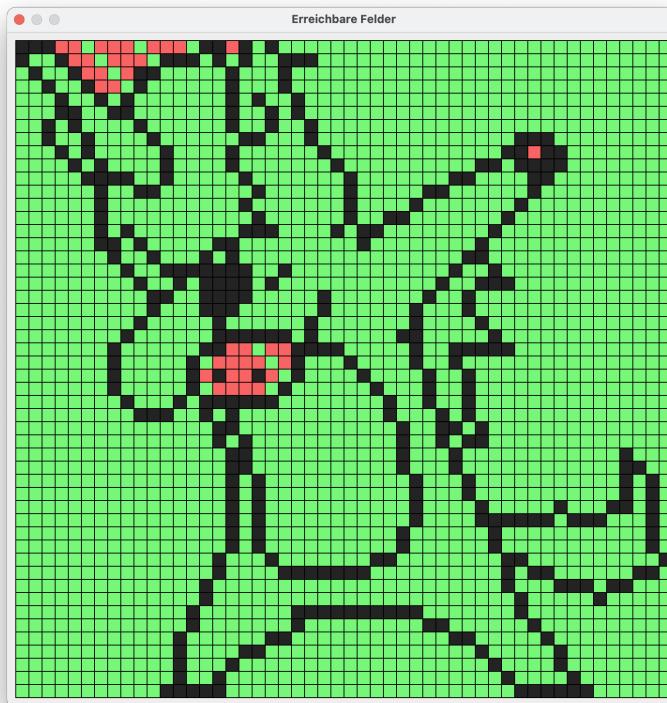


Abbildung 21: Fenster um unerreichbare Felder anzuzeigen.

Schwarze Felder bedeuten, dass es sich hier um ein Loch handelt und dieses Feld somit generell nicht zu erreichen sind. Grün erscheinen die Felder, die während des Spieles erreichbar

sind (Was jedoch nicht bedeutet, dass alle erreicht werden müssen). Rote Felder symbolisieren unerreichbare Felder während des gesamten Spieles. (Mehr Informationen siehe Kapitel ??)

4.3 Detaillierter Statistiken anzeigen

Unter dem Reiter Fenster kann man sich die unterschiedlichen Statistiken anzeigen lassen. Wie Sie in Abbildung 22 sehen können, wird damit der gesamte Zustand eines Spielers während des Spieles angezeigt. Dabei wird immer die aktuelle Implementierung verwendet.

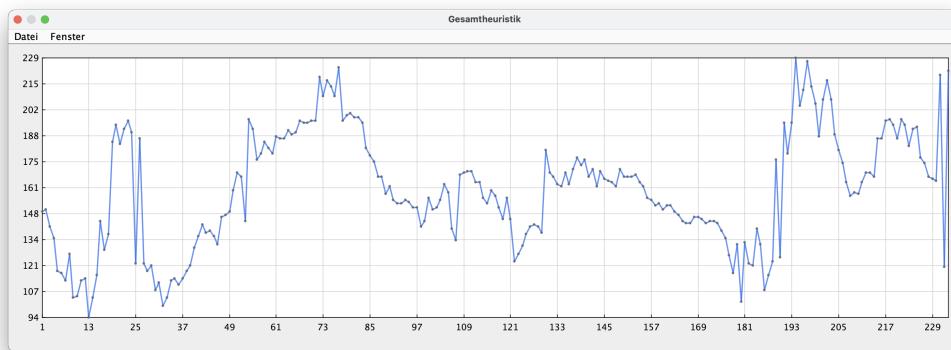


Abbildung 22: Fenster um aktuelle Heuristik anzuzeigen.

Möchte man Zustände der Heuristik für später speichern, kann man sich diese selbstverständlich exportieren lassen und mit Grafiken in der Zukunft vergleichen. Ein solcher Vergleich kann natürlich auch dazu genutzt werden, um unterschiedliche Spieler bzw. Varianten miteinander in Relation zu setzen. Dies haben Sie bereits in Kapitel 3.2.3 gesehen.

4.4 Nutzen der Spielanalyse

Die Entwicklung der Spielanalyse ist ein zeitintensiver Prozess. Es werden des Öfteren Spiele gegen andere Teams über die Plattform Matchpoint gespielt. Im Anschluss werden die Logdateien an die Gruppen verteilt, damit man eventuelle Abstürze finden oder auch generell den Spielverlauf anzeigen und verbessern kann. Ein solches Turnier liefert oft mehrere Hunderte Logfiles, die man sich am Besten alle genauestens anschauen sollte. Diese Software trägt wesentlich dazu bei, dass dies für die Beteiligten in überschaubarer Zeit und noch dazu sehr detailliert möglich ist. Es ist ebenfalls möglich eine Karte einzulesen, damit man nicht extra ein Spiel mit zugehöriger Logdatei erhält. Man kann auch unter Bearbeiten → Manuelle Spielführung händisch das Spiel so kontrollieren wie man möchte. Zudem bietet dieses Fenster die Möglichkeit das manuell bearbeitete Spiel zu exportieren, um anschließend vom Server gelesen zu werden.

5 Kartenanalyse

Neben der Bewertung des Spielfeldes übernimmt der MapAnalyzer auch weitere Aufgaben. Eine der Wichtigsten davon ist es herauszufinden, welche Felder erreichbar sind und welche nicht.

5.1 Ablauf eines Analyse Vorgangs

5.1.1 Erreichbarkeit in ReversiXT

In ReversiXT ist eine Reihe immer dann einnehmbar, wenn sich zwei Steine direkt nacheinander befinden. Diese Erreichbarkeit breitet sich auch über Transitionen hinweg aus und geht so lange, bis sie auf ein Loch oder sich selbst in gleicher Richtung stößt. Als Richtung versteht man dabei die Richtung, in die ein Feld verlassen wird, um in das nächste Feld zu gelangen. Kleine Veränderungen spielen bei der Erreichbarkeit eine große Rolle. So reicht es aus, einen Expansionsstein zu platzieren, um die gesamten erreichbaren Felder auf einer Karte zu verändern. Dies kann man im Abschnitt 5.1.2.2 nachvollziehen, bei dem durch einen Expansionsstein deutlich mehr Felder erreichbar werden.

5.1.2 Logisches Vorgehen

Im Kapitel 5.1.1 wird beschrieben, dass ein einzelner Stein sehr große Auswirkungen darauf hat, ob es möglich ist, eine ganze Karte zu bespielen oder nicht. Um aber mit Sicherheit sagen zu können, welche Bereiche erreichbar und welche unerreichbar sind, muss von jedem Feld aus nach neuen möglichen Zügen gesucht werden. Im Folgenden wird der Vorgang der Kartenanalyse gezeigt.

5.1.2.1 Suchen der Startpunkte

Dieser Schritt wirkt trivial, ist aber unabdingbar für eine erfolgreiche Analyse. Indem man für jeden Spieler und Expansionsfeld in direkter Nähe nach möglichen anderen Steinen zum überziehen sucht, stellt man für alle Spieler die möglichen Richtungen an Zügen fest. Dadurch können zum Beispiel wie in der Abbildung 23 sogenannte **Inseln** ignoriert werden. Diese können während eines Spieles nie betreten werden, da sie weder Transitionen noch Expansionssteine besitzen.

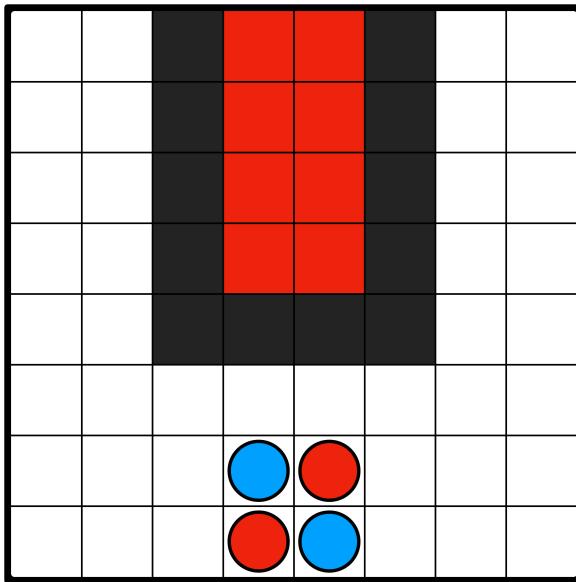


Abbildung 23: Der rote Bereich ist vom MapAnalyzer als nicht erreichbar markiert.

5.1.2.2 Folgen der möglichen Züge

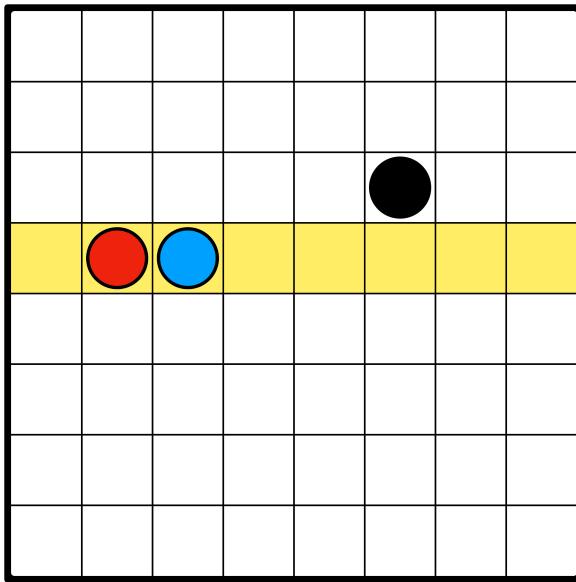


Abbildung 24: Erste markierte Feldlinie in die der MapAnalyzer schaut.

Zu Beginn wird ein Stein gewählt, und in allen acht Richtungen nach möglichen Zügen gesucht. Es wird also jedes direkt angrenzende Felde durchlaufen. Falls sich nun in direkter Nähe ein anderer Spielerstein oder ein Expansionsstein befindet, ist in dieser, sowie in der entgegengesetzten Richtung ein Zug möglich. Somit kann die komplette Reihe erreicht werden. Wieso Felder markiert werden müssen wird im Kapitel 5.3.1 genauer erklärt. Markierte Felder sind

hierbei Felder, die in einer späteren Iteration sicher noch erreicht werden. Das folgende Bild zeigt die soeben erklärten Vorgänge.

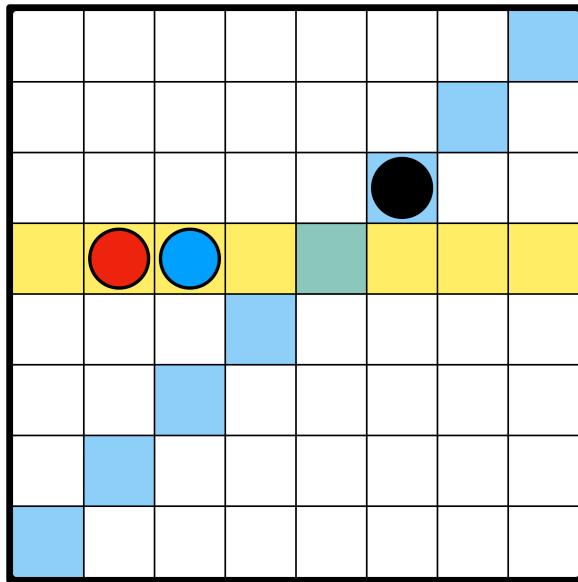


Abbildung 25: Zweite markierte Feldlinie in die der MapAnalyzer schaut.

Sobald alle Felder markiert sind, wird weiter in die übrigen Richtungen geschaut. In diesem Beispiel existiert aber kein neuer Spielstein und damit gibt es keine neuen Züge. Die gleiche Logik gilt nun auch für die nächsten Steine in der aktuellen Reihe.

Bei den darauffolgenden Feldern werden nun wieder alle anliegenden Felder auf mögliche Steine und damit mögliche Züge geprüft. Hier ist zu beachten, dass nicht auf Steine in der aktuellen Reihe geachtet wird, da diese später noch abgearbeitet werden. Zudem verhindert man damit unerwünschte Endlosschleifen. Da in der darauffolgenden Iteration das Selbe wie eben passiert, wird diese übersprungen.

In Abbildung 25 wird bei der Analyse der benachbarten Felder ein Expansionsstein gefunden. Demzufolge sind alle Felder in beide Richtungen erreichbar und werden markiert.

Nun wird dem neuen Pfad gefolgt und erneut in alle Richtungen nach Spielern gesucht. Der mögliche Weg *nach unten* wird noch nicht erkannt, da markierte Felder nicht als möglicher Zug zählen. Anschließend wird dem blauen Pfad gefolgt. In dieser Richtung stößt man jedoch auf keine anliegenden Spieler, wodurch auch diese Iteration übersprungen wird.

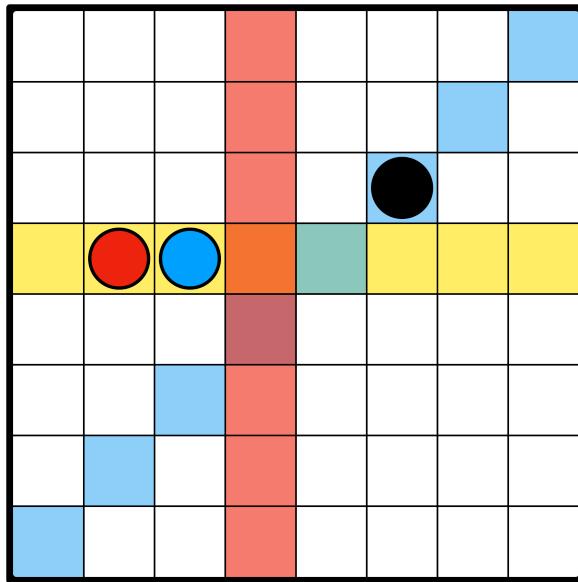


Abbildung 26: Dritte markierte Feldlinie in die der MapAnalyzer schaut.

Wenn das eine Ende des blauen Pfades erreicht ist, wird nun die andere Hälfte geprüft. Hierbei entsteht ein neuer Pfad, da das aktuelle Feld (Schnittstelle zwischen blau und rot) selbst noch nicht abgeschlossen ist, aber in direkter Nähe ein bereits fertig abgearbeitetes Feld (Schnittstelle zwischen rot und gelb) besitzt. Im nächsten Schritten wird nun dem roten Pfad nach den gleichen Regeln gefolgt. Wiederholt man dieses Vorgehen bis der Algorithmus terminiert, erhält man zum Schluss eine Karte mit erreichbaren Felder, wie Abbildung zeigt.

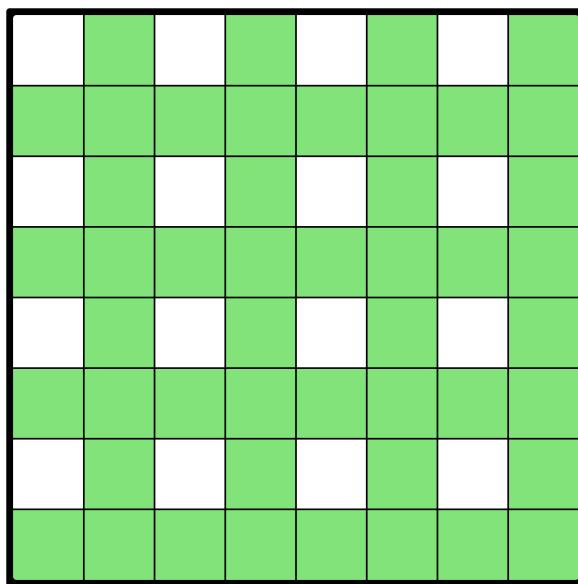


Abbildung 27: Nach Terminierung erhält man die erreichbaren Felder Karte (Grün markiert)

5.2 Vorteile durch den MapAnalyzer

Es gibt verschiedene Arten von Vorteilen, die eine genaue Kartenberechnung mit sich bringt. So erhält man zum Beispiel eine adequate Berechnung, wie viele Felder der Karte bereits belegt sind, da man unerreichbare Bereiche der Karte nicht berücksichtigt werden. Des Weiteren kann man durch das Herausrechnen auch verhindern, dass die Kartenbewertung durch Spezialsteine beeinflusst werden, die eigentlich gar nicht erreichbar sind. Im Folgenden werden nun die beiden Vorteile genauer erläutert.

5.2.1 Leichtere Bewertung des Spielfortschritts

Wenn man sich die Karte aus dem vorherigen Beispiel ansieht, kann man naiv mit der folgenden Formel die Anzahl der unbelegten Felder berechnen.

Bei dem Beispiel aus Abbildung 27 ergibt sich folgende Berechnung:

$$(8 \cdot 8) - 3 = 61 \text{ Freie Felder} \quad (4)$$

Für diese Bewertung ist das Spiel zu 5% abgeschlossen. In vielen Fällen werden die 100% nie erreicht da oft größere Teile nicht erreichbar sind oder das Spiel durch Eliminierung oder Disqualifikation vorzeitig beendet wird.

Im Gegensatz dazu liefert der MapAnalyzer, dass bei diesem Feld maximal 48 Felder bespielbar sind. Wenn man die beiden bereits belegten abzieht, bleiben noch 45 unbelegte Felder übrig. Das Spiel ist hier also bereits zu ca. 7% abgeschlossen. Natürlich kann es immer noch passieren, dass Felder während eines Spiels nicht erreicht werden, da zum Beispiel zuvor keine Züge mehr möglich sind, doch trotzdem bietet dieses Verfahren die Möglichkeit den aktuellen Spielfortschritt besser einzuschätzen und somit die Heuristik der K.I. exakter anzupassen.

5.2.2 Herausrechnen von nicht erreichbaren Bonusfeldern

Wie oben bereits aufgezeigt, sind bei einigen Karten bestimmte Felder nicht erreichbar, dieses Wissen kann für bestimmte Maps ausgenutzt werden. Sieht man sich die erreichbaren Felder der Karte in Abbildung 28 an, sieht es so aus, als wäre nur eine Reihe im linken Kartenteil erreichbar.

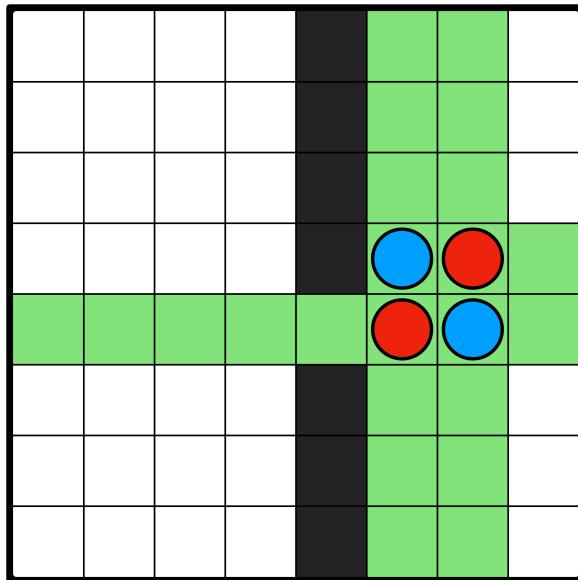


Abbildung 28: Felder die auf den ersten Blick erreichbar sind. (Grün eingezeichnet)

Wenn man jedoch die nächsten Möglichkeiten einzeichnet (siehe Abbildung 29), wird schnell ersichtlich, dass mehr Felder erreichbar sind.

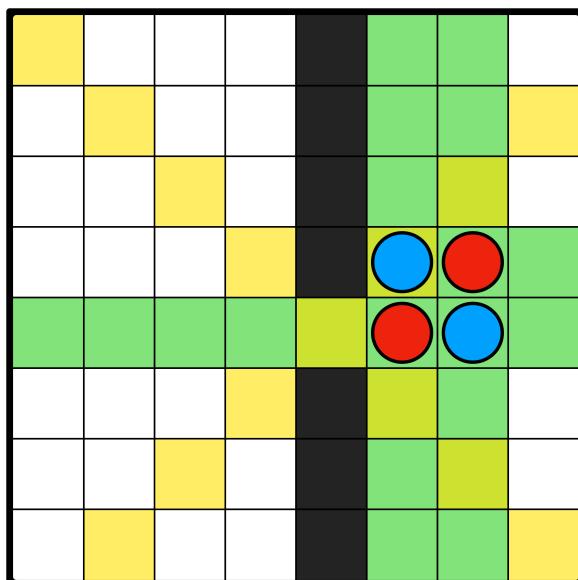


Abbildung 29: Felder die auf den zweiten Blick erreichbar sind. (Grün eingezeichnet)

Der Algorithmus wird in der Abbildung 29 nach dem gleichen Prinzip fortgeführt. Dadurch wird ersichtlich, dass in der linken Spielfeldhälfte die Felder nur in einem ganz bestimmtem Muster erreichbar sind.

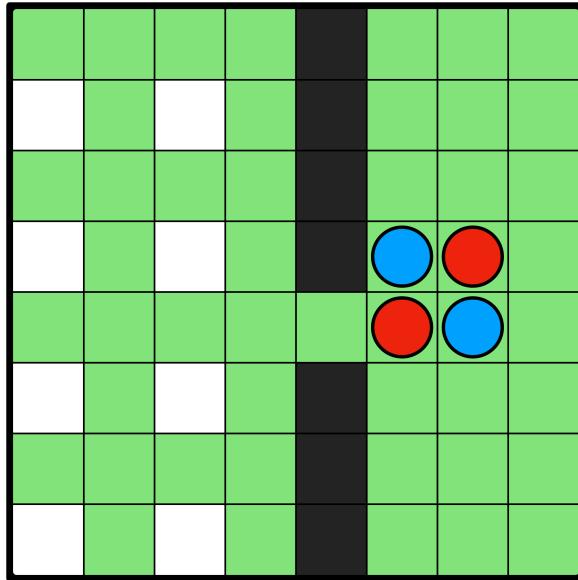


Abbildung 30: Nach Terminierung erhält man die erreichbaren Felder. (Grün markiert)

Die Strategie ist nun in diese Felder gezielt Spezialsteine zu legen. Diese Spezialfelder werden durch den MapAnalyzer noch bevor der erste Zug berechnet wird herausgerechnet und bei der Bewertung der Karte unberücksichtigt gelassen. Dadurch lässt sich der Client nicht von *falschen* Bonusfeldern verwirren und hat somit einen Vorteil gegen Clients die auf ein Herausrechnen verzichten.

5.3 Herausforderung bei der Entwicklung

Bei der Entwicklung sind zwei größere Probleme deutlich geworden. Zum einen war ein Problem die Performance, da man natürlich die meiste Zeit in die Berechnung der Züge und nicht in die Analyse der Karte investieren will. Zum anderen muss die Analyse der Karte auch mit vielen Transitionen fehlerfrei funktionieren.

5.3.1 Performance Probleme

Die ersten Versionen waren rein rekursiv, was bei größeren Karten schnell zu einem Überlauf des Speichers geführt hat. Um dieses Problem zu beheben, wurde der Analysevorgang von einem rekursiven Algorithmus zu einem teilweise iterativen Algorithmus umgebaut. Jedoch ist dies bei größeren Karten, die zusätzlich sehr viele Transitionen bzw. Expansionssteine haben, nach wie vor vorhanden problematisch. Um dieses Problem zu beheben, muss man verstehen, wie der Map Analyzer bei der Analyse vorgeht. Wenn sämtliche Steine der Karte mit Expansionssteinen belegt sind, führt das, weil die Mapanalyse im Uhrzeigersinn abläuft, zu einem Zickzack des Algorithmus durch die Karte. Der iterative Ansatz reduziert die Performance kaum, da die ganze Zeit neue rekursive Aufrufe für jede neue Richtungen entstehen. Dieses Problem wurde

durch das Vormarkieren von Feldern gelöst. Wenn ein gültiger Zug erkannt wird, werden die Felder in dieser Richtung und der entgegengesetzten Richtung markiert. Markierte Felder sind hier also Felder die in einer späteren Iteration noch von der iterativen Funktion durchlaufen werden und deshalb keinen eigenen rekursiven Aufruf benötigen. Dadurch kann man die Anzahl der rekursiven Aufrufe in einer Karte mit vielen Startsteinen drastisch reduzieren.

5.3.2 Fehlerfreies abgehen von Transitionen

Transitionen erhöhen allgemein die Komplexität von ReversiXT enorm und sie stellen während der gesamten Entwicklung eine große Herausforderung da. Das Problem beim Analysieren der Karte im Zusammenhang mit Transitionen ist das Erkennen von Schleifen die über Transitionen entstehen können. Wie eine *Transitionenschleife* aussehen kann wird in der folgenden Abbildung gezeigt:

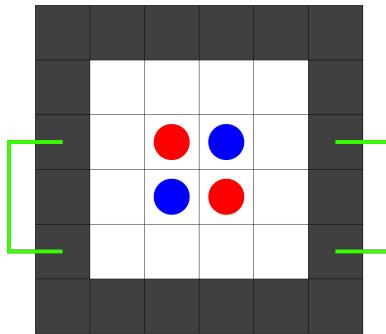


Abbildung 31: Darstellung einer Transitionenschleife.

Es kann also passieren, dass der Algorithmus, welcher die Map analysiert endlos der Transition folgt. Um das zu verhindern, merkt sich der Algorithmus welche Transitionen bereits abgelaufen wurden und benutzt diese kein zweites mal. Damit können Endlosschleifen verhindert werden. Dadurch erhöht sich die Performance, da in späteren Iterationen alten Transitionen nicht erneut gefolgt werden muss.

5.3.3 Fehler bei der Berechnung

Der MapAnalyzer berechnet alle theoretisch erreichbaren Felder. Da es aber zum Erreichen neuer Felder immer nötig ist, gegnerische Steine oder Expansionssteine zu überziehen, ist in der Realität meistens weniger begehbar als eigentlich möglich ist. Der MapAnalyzer ist bei der Analyse optimistisch, das heißt es wird lieber angenommen, dass ein Feld welches nicht erreichbar ist, als erreichbar angezeigt wird, als das Felder die erreichbar sind als nicht erreichbar eingestuft werden. Dies würde für die Zugberechnung fatale Folgen haben, da dadurch mögliche Züge nicht erkannt werden.

6 Karten für den Wettkampf

Zum Ende des Semesters treten alle Clients in einem großen Turnier gegeneinander an, um die Güte der Clients festzustellen. Zu diesem Turnier muss jedes Team vier Karten anfertigen, die speziell für ihren Client optimiert sind. Dabei muss eine Zwei-, Drei-, Vier- und Achtspielerkarte eingereicht werden. Bei den nachfolgend vorgestellten Karten handelt es sich jeweils um Maps, annähernd der Größe 50x50. Dies liegt daran, dass es sehr gute Algorithmen benötigt um auf solchen Karten gute Züge herauszusuchen. Damit kann die Güte der einzelnen Clients besser überprüft werden. Die Karten sind alle symmetrisch aufgebaut, wodurch eine gewisse Fairness gegeben ist.

6.1 Comp2021 - Zweispielerkarte

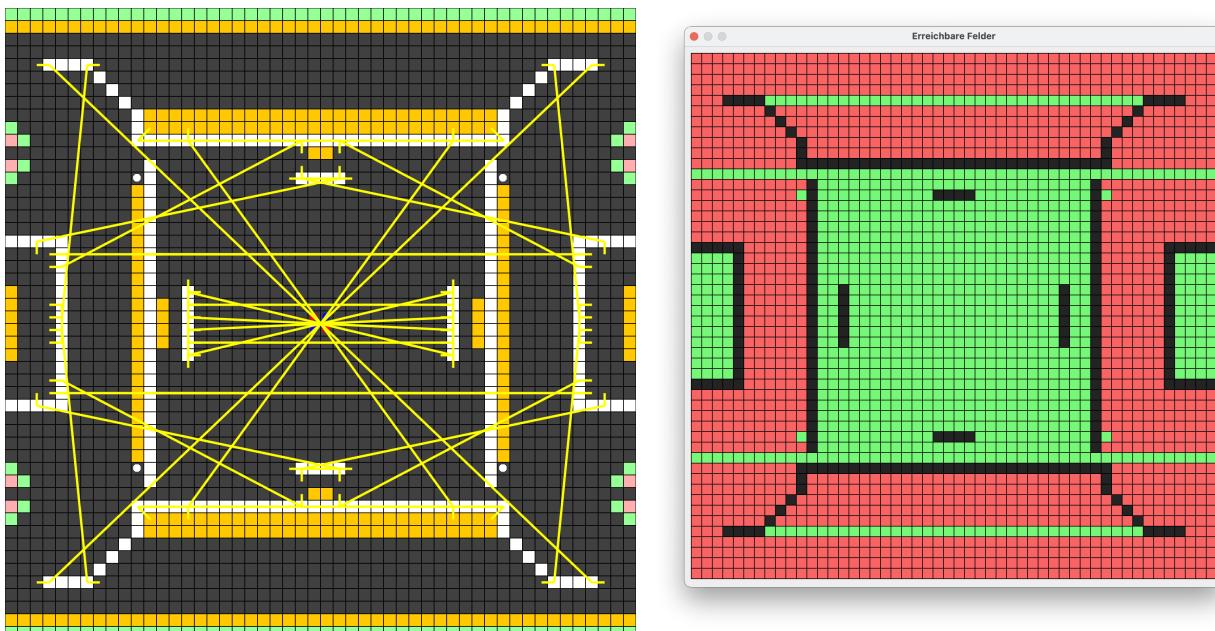


Abbildung 32: Zweispielerkarte inkl. erreichbare Felder in Grün (rechts)

Bei dieser Karte handelt es sich um eine sehr große Karte bezogen auf die Spielerzahl. Genau hier wird jedoch auch der Vorteil des MapAnalyzers erkennbar. Bei Karten, die sehr große Teile nicht bespielbarer Felder haben, nützt der MapAnalyzer extrem viel. Es werden hier nicht nur Felder herausgerechnet, sondern auch Spezialsteine die nicht erreichbar sind für die spätere Analyse unkenntlich gemacht. Man erhält somit eine adequate Aussage über den prozentualen Spielverlauf und eine realitätsnahe Vorstellung der Karte, wie diese bespielt werden kann. Zudem gibt es Transitionen die nicht gezogen werden können, da die dazu benötigten Spielfelder nicht erreichbar sind. Dieser Client kann mithilfe von Hashtabellen sehr effizient damit umgehen,

wodurch keine Performanceeinbußen entstehen.

6.2 Comp2021 - Dreispielerkarte

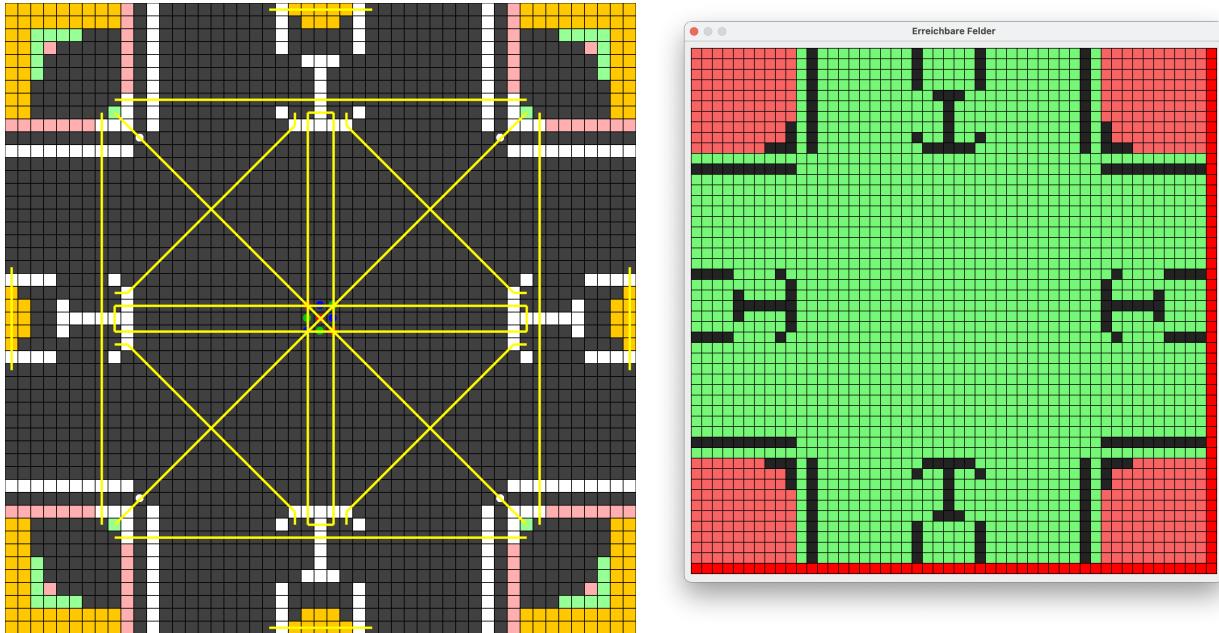


Abbildung 33: Dreispielerkarte inkl. erreichbare Felder in Grün (rechts)

Diese Dreispielerkarte ist nach dem gleichen Prinzip wie die vorherige Karte aufgebaut. Jedoch kann man auf dieser Karte 32 zusätzliche Bonussteine erreichen. Die Besonderheit liegt jedoch darin, dass sie jeweils nur aus zwei Richtungen erreichbar sind. Somit werden diese Felder erst sehr spät im Spielverlauf erobert, wodurch sich das Spiel am Ende drastisch verändern kann. Der Client verfügt über eine Möglichkeit erreichbare Spezialsteine zu finden und bei der Zugauswahl explizit nach diesen zu filtern.

6.3 Comp2021 - Vierspielerkarte

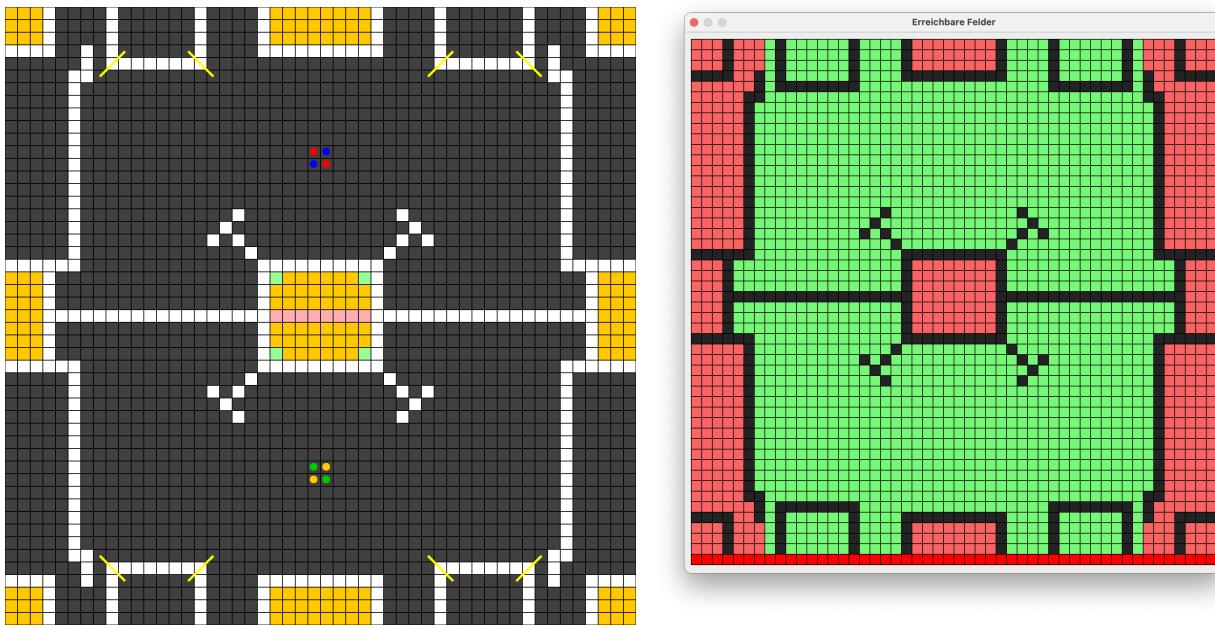


Abbildung 34: Vierspielerkarte inkl. erreichbare Felder in Grün (rechts)

Auf dieser Karte gibt es eine große Menge von Spezialsteinen, welche jedoch alle nicht erreichbar sind. Wenn ein Client speziell in die Richtung von Spezialsteinen ziehen möchte, wird dieser damit verwirrt. Die Besonderheit dieser Map liegt zudem darin, dass die linken und rechten Seiten nicht erreichbar sind. Dies liegt daran, dass man nur aus einer Richtung in diese Bereiche ziehen kann und damit diese Felder nicht einnehmbar sind. Dadurch wird der prozentuale Spielfortschritt ohne Berücksichtigung unerreichter Spielfelder stark verfälscht. Dieser Client erkennt somit die prozentuale Belegung wesentlich genauer als andere Clients. Zudem können sich auf dieser Karte die zwei Zweiergruppen nicht gegenseitig beeinflussen, da sie nur in einer Hälfte spielen.

6.4 Comp2021 - Achtspielerkarte

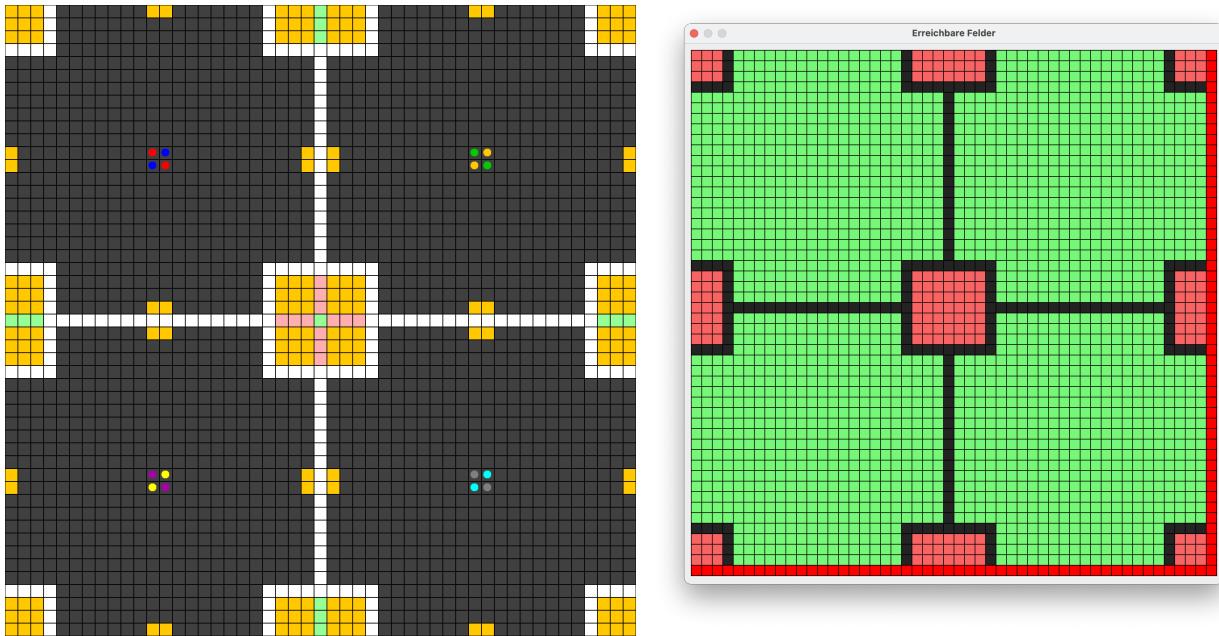


Abbildung 35: Achtspielerkarte inkl. erreichbare Felder in Grün (rechts)

Die Achtspielerkarte ist ähnlich wie die vorherige Karte aufgebaut. Es ist nur ein Bruchteil der Spezialsteine erreichbar. Zudem spielen hier vier Zweiergruppen unabhängig gegeneinander. Die Heuristik dieses Clients verfügt über die Möglichkeit, Spieler die einen selbst nicht erreichen können zu überspringen. Dabei wird ein gültiger und guter Zug ausgewählt und ähnlich wie bei BFS+ mit diesem Zug im Suchbaum weitergearbeitet. Dadurch wird der Suchbaum nicht unnötig aufgebäht, wodurch in größere Tiefen gesucht werden kann. Mithilfe dieser Reduktion des Suchbaumes werden unnötige Berechnungen vermieden und somit auch bei Achtspielerkarten eine Suchbaumtiefe von 10 und größer ermöglicht.

7 Verbesserungen für FightClub und MatchPoint

Ein Turnier wird mithilfe der Software *Matchpoint* gestartet und visuell angezeigt. Hier treten die unterschiedlichen Teams gegeneinander an um Ihre Stärken beweisen zu können. Die Abgaben der einzelnen Teams werden in *Fightclub* eingetragen, wo man zudem Maps erstellen und andere Karten herunterladen kann. Für beide Plattformen gibt es einige Verbesserungsvorschläge, damit es zukünftige Semester leichter haben einen hervorragenden K.I. Client zu entwickeln.

7.1 Direkte Konsolenausgabe

Es werden zu jedem Spiel alle Konsolenausgaben gespeichert und können im Anschluss heruntergeladen werden. Hier erhält man detaillierte Informationen über seinen Client und eventuelle Abstürze. Es könnte jedoch darauf erweitert werden, die Konsolenausgaben in Echtzeit im Browser anzuzeigen und auch im Nachhinein alles zur Verfügung stehen zu lassen. Dies könnte mit einer Datenbank und einer Kommunikation mittels Socket realisiert werden. Nutzer müssten somit nicht alle Logdateien gebündelt herunterladen. Es wird Ihnen dadurch die Möglichkeit geboten nur einzelne Dateien herunterzuladen und von überall aus live auf die Ausgaben Zugriff zu erhalten.

7.2 Export von Spielzuständen

Durch Matchpoint 2 ist es den Gruppen möglich beendete Spiele in beide Richtungen durchzulaufen. Ist man disqualifiziert worden, muss man diese Situation wiederherstellen und seine Software darauf testen. Die Herstellung dieser Situation ist jedoch sehr aufwendig und könnte durch einen einfachen Export des aktuellen Spielbrettzustandes inklusiver Transitionen deutlich erleichtert werden. Der Vorteil liegt eindeutig in dem verringerten Zeitaufwand Spiele nachträglich herunterladen, wieder einlesen und debuggen zu können.

7.3 Profilbasiertes Matchpoint

Aktuell werden ununterbrochen Turniere ausgeführt um damit die Güte der einzelnen Clients zu erhalten. Man könnte ein profilbasiertes Matchpoint entwickeln, bei dem sich die einzelnen Gruppen anmelden können um ein Spiel mit einer ausgewählten Karte gegen eine ausgeählte Gruppe zu starten. Hier könnten dann nur den beteiligten Gruppe diese Informationen über ein solches Match erteilen. Dadurch würde man Änderungen am Code leichter überprüfen, sowie eventuelle Spezialisierungen seines Clients in echten Matches testen können .

7.4 Integrierte Tooltips

Gerade zu Beginn sind einige Elemente und deren Bedeutung von Matchpoint für den Nutzer unklar. Durch die Integration von Tooltips würden anfängliche Startprobleme beseitigt werden und Matchpoint damit leichter verständlich werden.

7.5 Informationen über Fairness

Beim Erstellen einer Karte sollte speziell auf die Fairness dieser Map geachtet werden. Eine willkommene Neuerung für den Mapeditor in Fightclub wäre eine Aussage über die Fairness dieser erstellen Karte zu geben. Zudem könnte man automatisch faire Startpositionen selektieren und auf die Karte setzen. Durch diese Neuerung würde die Fairness aller Karten drastisch verbessert werden, was im Interessen aller Beteiligten sein sollte.

8 Fazit

Die Vorfreude auf dieses Wahlpflichtmodul war bereits vor dem Vorlesungsstart sehr groß. Vor allem der Gedanke ein eigenes Projekt von Anfang bis Ende zu planen, entwickeln, testen und lauffähig spielen zu sehen trug maßgeblich dazu bei.

ZOCK ist ein sehr zeitintensives, jedoch auch unglaublich spannendes Modul. Man sammelt hier in allen erdenklichen Bereichen neue Erfahrungen, wie zum Beispiel die Planung eines Softwareprojekts, verfassen eines wissenschaftlichen Berichtes, Entwicklung von Algorithmen sowie die Zusammenarbeit in einem Team. Vor allem, wenn gravierende Fehler entstehen oder der Client unerwartet x-mal disqualifiziert wird, merkt man, dass man ein Team ist und gemeinsam die Probleme angehen und beheben muss. Hier wird nicht nur Erfahrung im Bereich Softwareentwicklung, sondern auch im Zwischenmenschlichen gesammelt.

Die einzelnen Beteiligten, haben eine so große Begeisterung für diesen Kurs entwickelt, dass nicht nur die notwendigen Aufgaben erfüllt wurden, sondern viele weitere Ideen in dieses Projekt geflossen sind. Es existiert aufgrund dieser Freude an ZOCK eine neue Möglichkeit, Spiele und Karten detaillierter zu analysieren. Diese Möglichkeit entstand bezüglich der Entwicklung des MapAnalyzers, wodurch erreichbare Felder erkannt werden. Der GameAnalyzer steht in Zukunft anderen Gruppen zu Verfügung, damit diese die gleichen Vorteile haben, um Spiele nachträglich zu analysieren, exportieren, modifizieren und Statistiken aufzurufen. Es gibt zudem eine intelligente Zugsortierung, die die Probleme der naiven Zugsortierung behebt und damit extreme Leistungsopimierungen bietet.

Vor allem aber am eigentlichen Entwicklungsfortschritt sieht man, dass man Projekte nur sehr schwer durchplanen kann und man des Öfteren ein Refactoring betreiben, neuere Erkenntnisse einarbeiten oder komplett andere Funktionalitäten hinzufügen muss. Durch diese stetigen Veränderungen und Anpassungen hat das Team wertvolle Erfahrungen bezüglich Softwareentwicklung und Softwareplanung sammeln können. Besonders hervorzuheben ist, dass alle vorherigen Module in diesem Kurs Verwendung finden, sei es PG1 oder PG2 um ordentlichen Code zu produzieren, oder AD um leistungsstarke Algorithmen zu verstehen und zu entwickeln.

Im Allgemeinen ist der Aufbau dieser Lehrveranstaltung sehr gut durchdacht und strukturiert aufgebaut. Der Professor fordert sehr viel von seinen Studenten, ist dafür aber auch bereit sehr viel zu opfern. Man erhält einen fundierten Umfang über die Entwicklung einer künstlichen Intelligenz und wie diese schrittweise verbessert wird. Man lernt zudem mit Rückschlägen umzugehen, da man des Öfteren an der Spitze kratzen kann und kurze Zeit später wieder von anderen Clients vernichtend geschlagen wird. Auf all diese Erfahrungen möchte niemand aus dem Team verzichten. Die Erwartungen waren groß, welche jedoch weit überstiegen wurden. Wer sich dieses Modul und den Arbeitsaufwand zutraut, sollte es definitiv belegen.

9 Anhang



Abbildung 36: UML Klassendiagramm des Projektes.

Literaturverzeichnis

- [1] Hartnell, Tim; Charlton, Graham. Reversi/Othello. <https://archive.org/details/reversi.qb64>, 1984. (abgerufen am: 14.05.2021).
- [2] JetBrains. IntelliJ IDEA - Homepage. <https://www.jetbrains.com/idea/>.
- [3] Chacon, Scott. Git - Homepage. <https://git-scm.com>.
- [4] Gradle. Gradle - Homepage. <https://gradle.org>.
- [5] Post, Uwe. *Besser coden*. Rheinwerk Computing, 1 edition, 2018.
- [6] JUnit. JUnit4 - Homepage. <https://junit.org/junit4/>.
- [7] Ubuntu. Ubuntu - Homepage. <https://ubuntu.com>.
- [8] Hermann, Sven. Schach in Zahlen. <https://schachlich.de/schach-in-zahlen/>. (abgerufen am: 09.05.2021).