Projektbericht
Studiengang Informatik

Marcel Geirhos

Künstliche Intelligenz für das Brettspiel Mühle

Prüfer:	Prof. Dr. Ulrich Klauck
Zweitprüfer:	Prof. Dr. Rainer Werthebach
Einreichungsdatum	Oktober 2017

Inhaltsverzeichnis

Literaturverzeichnis

(IN) INGO NEUMAYER, Sabine K.: *Künstliche Intelligenz*. http://www.planet-wissen.de/technik/computer_und_roboter/kuenstliche_intelligenz/index.html),

(Kue) Wikipedia: *Künstliche Intelligenz*. https://de.wikipedia.org/wiki/Künstliche_Intelligenz,

(Lil) LILL, Felix: *Mein Zwilling, der Roboter*. http://www.tagesspiegel.de/themen/reportage/hir ishiguro-baut-menschmaschinen-sein-roboter-faehrt-fuer-ihn-zu-konferenzen/10314280-2.html,

(Min) Wikipedia: *Minimax-Algorithmus*. https://de.wikipedia.org/wiki/Minimax-Algorithmus,

(Nag) NAGELS, Philipp: *Facebook musste AI abschalten, die „Geheimsprache" entwickelt hat*. https://www.welt.de/kmpkt/article167102506/Facebook-musste-AI-abschalten-die-Geheimsprache-entwickelt-hat.html,

(Swi) Wikipedia: *Swing (Java)*. https://de.wikipedia.org/wiki/Swing_(Java),

(Tis) TISSLER, Jan: *Praktische Anwendungen für Künstliche Intelligenz – heute und morgen*. https://upload-magazin.de/blog/13645-anwendungen-beispiele-kuenstliche-intelligenz/),

(Way) Wikipedia: *Autonomes Auto von Google, 2017*. https://de.wikipedia.org/wiki/Waymo#/media/File:Waymo_self-driving_car_front_view.gk.jpg),

(Zü) ZÜRICH, ETH-Bibliothek: *Mühle*. http://www.library.ethz.ch/ms/Virtuelle-Ausstellungen/Alles-ist-Spiel!-Unterhaltungsmathematik-in-historischer-Perspektive/Muehle,

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Das Brettspiel Mühle

Das Brettspiel Mühle ist für zwei Spieler. Das Spielbrett besteht aus 24 Feldern, die über Verbindungslinien miteinander verbunden sind. Jeweils acht Felder sind in einem Quadrat angeordnet. Die daraus resultierenden drei Quadrate liegen ineinander verschachtelt und bilden das Mühle Spielbrett.

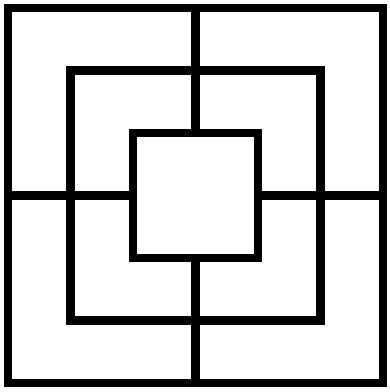

Abbildung 1.1: Mühle Spielbrett

Zu Beginn des Spiels bekommt jeder Spieler neun, gleichfarbige Spielsteine meistens in den Farben weiß und schwarz. Das Spiel ist beendet, sobald ein Spieler nur noch zwei Spielsteine auf dem Spielbrett liegen hat oder ein Spieler keinen Spielzug mehr tätigen kann. Das Mühle Spiel lässt sich in drei Phasen gliedern:

1. Setzphase
2. Verschiebenphase
3. Springenphase

Spielregeln für die Setzphase:

Bei der Setzphase darf jeder Spieler abwechselnd einen eigenen Spielstein auf ein Feld des Spielbrettes legen. Wenn beide Spieler ihre neun Spielsteine gesetzt haben beginnt die zweite Phase.

Spielregeln für die Verschiebenphase:

Jeder Spieler darf abwechselnd einen eigenen Spielstein auf ein freies, benachbartes Feld setzen. Das Feld auf das der Spielstein verschoben werden soll muss

über eine Verbindungslinie mit dem aktuellen Feld auf dem der Spielstein liegt verbunden sein. Wenn ein Spieler keinen seiner verbleibenden Spielsteine bewegen kann hat der Spieler verloren.

Spielregeln für die Springenphase:

Sobald ein Spieler nur noch drei Spielsteine auf dem Spielbrett hat befindet sich dieser Spieler in der Springenphase. In dieser Phase darf der Spieler jede Runde einen eigenen Stein auf ein beliebiges freies Feld setzen. Wenn der Spieler nur noch zwei Spielsteine auf dem Spielbrett besitzt hat er das Spiel verloren.

Eine Mühle bilden:

In jeder Phase des Spiels kann ein Spieler eine Mühle bilden. Eine Mühle besteht aus drei gleichfarbigen und benachbarten Spielsteinen die auf einer Geraden liegen. Wenn ein Spieler eine Mühle bildet darf er dem gegnerischen Spieler einen Spielstein entfernen. Der entfernte Spielstein darf nicht Teil einer Mühle sein, außer alle gegnerischen Spielsteine sind Teil einer Mühle, dann darf auch ein Spielstein von einer Mühle entfernt werden.

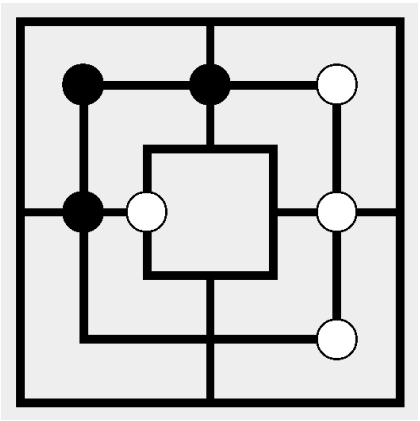

Abbildung 1.2: Gebildete weiße Mühle

Wissenswertes:

Die möglichen Stellungen bei einem Mühle Spiel beträgt etwa 1,8 * 10^{10}. Alle möglichen Spielstellungen wurden in einer 17 GByte großen Datenbank gespeichert. Dabei verbraucht eine Stellung mit Bewertung ein Byte. Jürg Nievergelt und Ralph Gasser beweisten 1994 an der ETH Zürich das eine Partie die beidseitig korrekt gespielt wird immer unentschieden endet. Um diesen Beweis zu erbringen wurden drei Jahre lang mit Unterbrechungen auf mehreren Computern und verschiedenen Programmen Berechnungen durchgeführt.

1.2 Eingesetzte Software und Programmiersprache

1.2.1 Programmiersprache Java

Java ist eine objektorientierte Programmiersprache die 1995 von Sun Microsystems entwickelt wurde. Seit 2010 ist Sun Microsystems ein Tochterunternehmen von Oracle. Dabei ist Java eine plattformunabhängige Programmiersprache das bedeutet das die geschriebene Software sowohl auf Windows, MacOS, Linux und vielen weiteren Betriebssystemen lauffähig ist.

1.2.2 GUI-Toolkit Swing

Swing wurde ebenfalls von Sun Microsystems entwickelt und gehört seit 1998 zur Java-Runtime. Mithilfe von Swing können grafische Benutzerschnittstellen objektorientiert programmiert werden. Durch die Plattformunabhängigkeit von Swing können auch diese Anwendungen auf verschiedenen Betriebssystemen ausgeführt werden.

1.2.3 Eclipse

Eclipse ist eine quelloffene und integrierte Entwicklungsumgebung (IDE) für die Entwicklung von Software. Auch Eclipse ist plattformunabhängig einsetzbar. Für Eclipse gibt es unzählige Erweiterungen sogenannte Plugins.

1.2.4 Inkscape

Die frei verfügbare und plattformunabhängige Software Inkscape ist zur Erstellung und Bearbeitung zweidimensionaler Vektorgrafiken entwickelt worden. Vektorgrafiken haben den Vorteil das sie auf verschiedene Größen skaliert werden können ohne das diese unscharf angezeigt werden.

1.2.5 Github

Github ist ein Versionsverwaltungssystem mit deren Hilfe Software-Entwicklungsprojekte auf Servern bereitgestellt wird. Damit können ganze Software Teams an einem Projekt arbeiten das den gleichen Projektstand aufweist. Ein weiterer Vorteil ist die Wiederherstellung älterer Softwareversionen die als Backup dienen können.

1.3 Was ist künstliche Intelligenz?

Künstliche Intelligenz (KI) oder auch artifizielle Intelligenz (AI) genannt ist ein Teilgebiet der Informatik und versucht eine möglichst menschenähnliche Intelligenz zu simulieren. Daraus folgt das eine Software die eine künstliche Intelligenz simulieren soll Probleme eigenständig lösen kann. Durch die Kombination der beiden Fachgebiete Mathematik und Informatik kann intelligentes Verhalten auf computergestützten Systemen simuliert werden. Die Forschungs- und Entwicklungsergebnisse der KI nahm auch auf viele weitere Forschungsgebiete Einfluss vor allem in den Neurowissenschaften. Im Bereich der künstlichen Intelligenz wird zwischen einer starken KI und einer schwachen KI unterschieden:

Starke KI:

Durch eine starke KI soll die Intelligenz des Menschen und dessen Gedankengänge mechanisiert werden. Eine Maschine soll sich so verhalten wie ein Mensch. Weltweit wird dieses Ziel von vielen Forschungs- und Entwicklungsteams verfolgt. Gefühle wie Liebe, Trauer, Freude, Angst, ... können nur ansatzweise simuliert und nachgeahmt werden. Bisher konnte die Vision der starken KI nicht vollständig umgesetzt werden.

Schwache KI:

Bei der schwachen KI sollen verschiedenste Anwendungsprobleme des Menschen durch simulierte Intelligenz unterstützt und gelöst werden. Manche KI Systeme sind in der Lage durch künstliche neuronale Netze selbstständig zu lernen. Dies wird auch unter dem Begriff Deep Learning verstanden. Solche Systeme werden nicht mehr programmiert, sondern mit Hilfe von Daten trainiert. In den letzten Jahrzehnten gab es im Bereich der schwachen KI enorme Fortschritte.

Turing-Test:

Alan Turing ein britischer Mathematiker entwickelte im Jahr 1950 den sogenannten Turing-Test. Dabei kommuniziert ein Mensch parallel mit einem Mensch und einer Maschine zum Beispiel über ein Chat Programm. Sowohl Mensch als auch Maschine versuchen die Testperson zu überzeugen das ein realer Mensch mit ihnen kommuniziert. Wenn die Testperson am Ende der Unterhaltung nicht eindeutig bestimmen kann welcher der Gesprächspartner Mensch oder Maschine ist gilt die Maschine als intelligent. Bisher wurde der Turing-Test von keiner Maschine bestanden und KI-Experten gehen davon aus das dies in absehbarer Zeit auch nicht passieren wird.

1.4 Einsatzgebiete von künstlicher Intelligenz

1.4.1 Gesichtserkennung

Einige Gebiete der künstlichen Intelligenz wird von uns Menschen heutzutage nicht mehr als künstliche Intelligenz betrachtet, da diese in unseren Alltag integriert wurden. In dieses Gebiet zählt auch die Gesichtserkennung. Für uns Menschen ist es simpel Gesichter anderer Personen zu erkennen und bekannte Gesichter Personen zuzuordnen die wir kennen. Auch wenn diese mit unterschiedlichen Lichtverhältnissen, Gesichtsausdrücken, Sonnenbrillen, etc. auftreten. Aus Sicht einer Software ist dies aber weitaus komplexer. Konkrete Anwendung findet eine Gesichtserkennungssoftware bei Sicherheitskameras die Gesichter von Verbrechern in einer Datenbank gespeichert hat und diese mit den aufgenommenen Gesichtern abgleicht. Eine weitere Anwendung befindet sich heutzutage bei vielen Menschen in der Hosentasche. Moderne Smartphones erlauben es dem Benutzer ihr Smartphone mit ihrem Gesicht zu entsperren. Allgemein wurden im Bereich der Gesichtserkennung in den letzten Jahren große Fortschritte erzielt.

1.4.2 Autonomes Fahren

Kaum ein Thema im Bereich künstlicher Intelligenz ist so populär wie das autonome Fahren. Große Unternehmen wie beispielsweise Google, Tesla, Uber und viele weitere entwickeln selbstfahrende Fahrzeuge. Heutige Fahrzeuge besitzen Software die den Sicherheitsabstand zum Vordermann automatisch einhält oder der Straßenspur folgt. Einparkhilfen sind heutzutage in vielen Fahrzeugen integriert und erleichtern dem Fahrer das einparken oder nehmen dem Fahrer gleich die ganze Arbeit ab. Aber ein komplett autonom fahrendes Fahrzeug benötigt weitaus mehr Intelligenz, da nicht jede Situation eindeutig ist und nur mit Erfahrung gemeistert werden kann. Es gibt viele Situationen wie etwa Schneefall, Baustellen, Verkehrspolizisten, etc. die hohe künstliche Intelligenz erfordert. Die Übergangszeit zwischen menschlichen auf maschinelles Fahren wird die größte Herausforderung in sich bergen. Sobald alle Fahrzeuge autonom fahren, könnten diese auch untereinander kommunizieren und den Verkehrsfluss damit positiv beeinflussen.

Abbildung 1.3: Autonomes Fahrzeug von Google

1.4.3 Robotik

Ein weiteres großes Einsatzgebiet der künstlichen Intelligenz ist die Robotik. Schon in den 1960er Jahren kamen die ersten Roboterarme bei General Motors zum Einsatz um Aufgaben wie etwa Schweißen oder Lackieren zu übernehmen. Die Einsatzmöglichkeiten von Robotern sind vielzählig. Vor allem bei gesundheitsschädlichen oder riskanten arbeiten ersetzen Roboter immer mehr den Menschen. Im Bereich der Bombenentschärfung sind seit vielen Jahren Roboter im Einsatz. Moderne Roboter werden immer menschenähnlicher. Hiroshi Ishiguros ist ein japanischer Ingenieursprofessor aus Osaka und konstruierte sich sein eigenes Abbild als Geminoiden.

Abbildung 1.4: Hiroshi Ishiguros (rechts) und sein Roboter (links)

1.4.4 Computerspiele

In vielen Computerspielen ist künstliche Intelligenz heute nicht mehr wegzudenken. Kaum ein modernes 3D Computerspiel kommt ohne künstliche Intelligenz aus. Vor allem in Strategiespielen oder Shootern werden Wegfindungsalgorithmen benötigt. In populären Strategiespielen wie Dota existieren künstliche Gegner die selbst gegen Profi Spieler gewinnen. In Brettspielen wie etwa Schach, Dame, Go oder Mühle gibt es künstliche Intelligenzen die dem Menschen bereits überlegen sind und gegen die höchstens ein Unentschieden gespielt werden kann. Die programmierte Software ist dabei in der Lage viele Züge vorauszudenken. Eine künstliche Intelligenz für Mühle muss dabei viele Situationen beachten wie etwa das öffnen und schließen von eigenen oder gegnerischen Mühlen, das richtige Setzen der Spielsteine, verhindern von Zwickmühlen, etc. .

2 Aufgabenstellung

In der objektorientierten Programmiersprache Java soll das Brettspiel Mühle programmiert werden. Die grafische Benutzerschnittstelle wird mit dem GUI-Toolkit Swing umgesetzt. Grafische Elemente wie das Spielbrett oder die Spielsteine werden mithilfe des Programms Inkscape als Vektorgrafiken erstellt. Dies hat den Vorteil das die Elemente sich der Bildschirmauflösung anpassen können ohne das diese unscharf wirken. Die GUI des Brettspiels Mühle soll sich der Bildschirmauflösung anpassen, damit Spielbrett, Spielsteine und andere grafische Elemente benutzerfreundlich dargestellt werden.

Der Kern der Software bildet die künstliche Intelligenz gegen die der Spieler antritt. Die künstliche Intelligenz muss in der Lage sein mehrere Züge nacheinander simulieren zu können. Eine selbstlernende künstliche Intelligenz ist nicht Teil des Programms. Das bedeutet die künstliche Intelligenz kann nicht trainiert werden und wird an keine Datenbank gekoppelt sein.

3 Anforderungsanalyse

3.1 Aufbau von Mühle

Eine Anforderung an die Software ist der grafische Aufbau des Mühle Spielbretts. Das Mühle Spiel soll über einen Startbildschirm gestartet werden und erreichbar sein. Die grafischen Elemente wie das Spielbrett und die Spielsteine werden als Vektorgrafiken erstellt. Diese werden dann mithilfe des GUI-Toolkit Swing in die Software integriert und angezeigt.

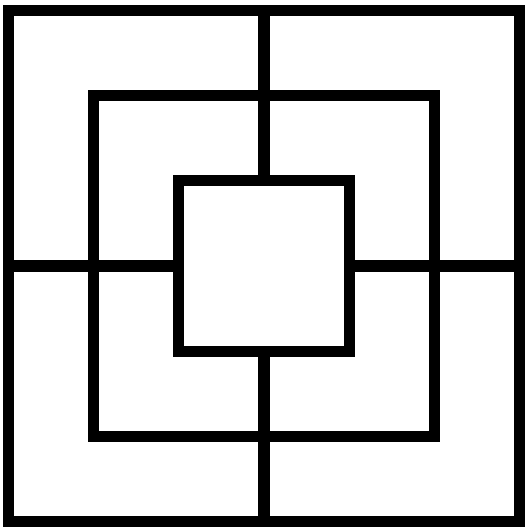

Abbildung 3.1: Aufbau Mühle Spielbrett

3.2 Grundlegende Logik

In allen drei Spielphasen kann eine Mühle gebildet werden dabei darf der Spieler einen gegnerischen Spielstein entfernen. Dieser Spielstein darf aber nicht Teil einer Mühle sein. Wenn alle Spielsteine des Gegners einer Mühle angehören darf auch ein Spielstein von einer Mühle entfernt werden. Spielinformationen wie die Anzahl der gespielten Runden, die verbleibenden Spielsteine, in welcher Spielphase befindet sich der Spieler und welcher Spieler ist aktuell am Zug sollen am rechten Rand des Spielbretts angezeigt werden.

3.2.1 Setzphase

In der 1.Phase der Setzphase muss jeder Spieler abwechselnd einen Spielstein auf ein freies Feld setzen dürfen. Spieler weiß darf den ersten Spielstein auf ein Feld setzen. Bei einem nicht zulässigen Spielzug soll nichts passieren bis ein gültiger Spielzug ausgeführt wird. Sobald beide Spieler ihre neun Spielsteine auf das Spielbrett gesetzt haben beginnt die 2.Phase des Mühle Spiels.

3.2.2 Verschiebenphase

In der Verschiebenphase dürfen beide Gegner abwechselnd einen Spielstein auf ein freies, benachbartes Feld setzen. Dabei müssen die Nachbarfelder des ausgewählten Feldes bekannt sein. Bei einem nicht zulässigen Spielzug soll nichts passieren bis ein gültiger Spielzug ausgeführt wird. Bei einem bereits ausgewählten Spielstein muss der Spieler in der Lage sein sich einen anderen Spielstein auswählen zu können und den Zug mit diesem weiter zu führen. Kann ein Spieler keinen Spielstein mehr verschieben hat er das Spiel verloren und der Game Over Bildschirm wird angezeigt und das Spiel wird beendet. Sobald ein Spieler nur noch drei verbleibende Spielsteine auf dem Spielbrett hat befindet sich dieser Spieler in der 3.Phase.

3.2.3 Springenphase

Wenn ein Spieler sich in der 3.Phase befindet darf er einen eigenen Spielstein auf ein beliebiges, freies Feld setzen. Bei einem nicht zulässigen Spielzug soll nichts passieren bis ein gültiger Spielzug ausgeführt wird. Bei einem bereits ausgewählten Spielstein muss der Spieler in der Lage sein sich einen anderen Spielstein auswählen zu können und den Zug mit diesem weiter zu führen. Sobald ein Spieler nur noch zwei Spielsteine besitzt hat dieser das Spiel verloren und der Game Over Bildschirm wird aufgerufen und das Spiel wird beendet.

3.3 Künstliche Intelligenz

Die künstliche Intelligenz des Mühle Spiels soll in der Lage sein mehrere Spielzüge nacheinander simulieren zu können. Es soll möglich sein verschiedene Schwierigkeitsstufen auswählen zu können. Die Schwierigkeit soll sich an der Anzahl vorraus gerechneter Spielzüge orientieren. Außerdem soll der Spieler auswählen können, ob weiß oder schwarz der KI-Gegner ist. Eine Verbindung zu einer Datenbank mit vordefinierten Spielzüge ist nicht vorgesehen. Die künstliche Intelligenz soll nicht trainiert werden können.

3.4 Abgrenzung

Die Software soll mit keiner Datenbank verbunden werden die, die bereits gespielten Spielzüge speichert und die künstliche Intelligenz trainieren kann. Es soll auch nicht möglich sein das aktuelle Mühle Spiel zu speichern oder ein bereits gespieltes Spiel zu laden. Ein rückgängig machen der Spielzüge soll nicht möglich sein.

4 Implementierung

4.1 Datenstrukturen

4.1.1 Felder

Jedes Feld ist ein eigenes Objekt und besitzt eine eindeutige Id von 0-23. Die Felder sind wie in folgender Abbildung mit der eindeutigen Id implementiert. Zu jedem Feld gibt es eine x und y Position. Außerdem wird bei jedem Feld gespeichert von welchem Spieler ein Spielstein auf dem Feld liegt.

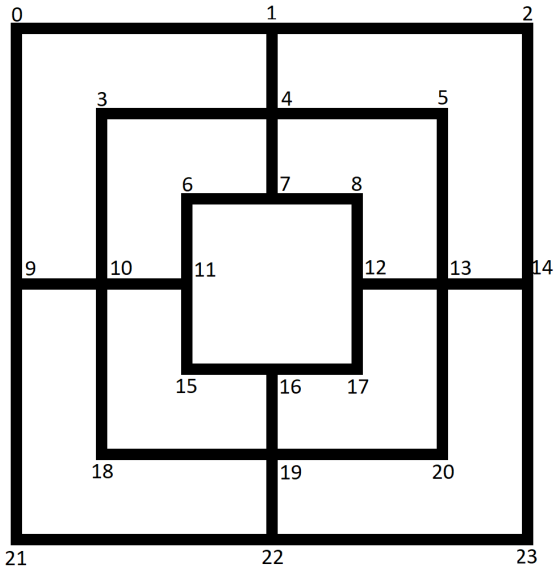

Abbildung 4.1: Datenstruktur für Felder

4.1.2 Nachbarschaftsfelder

Jedes Feld besitzt 2-4 Nachbarschaftsfelder. Dabei hat zum Beispiel das Feld mit der Id 13 die vier Nachbarschaftsfelder 5, 12, 14 und 20 (blau). Wenn ein Feld weniger als vier Nachbarschaftsfelder besitzt wie etwa das Feld mit der Id 0 werden die fehlenden Felder mit der Id -2 definiert. Daraus folgen für das Feld 0 folgende Nachbarschaftsfelder: 1, 9, -2 und -2 (grün).

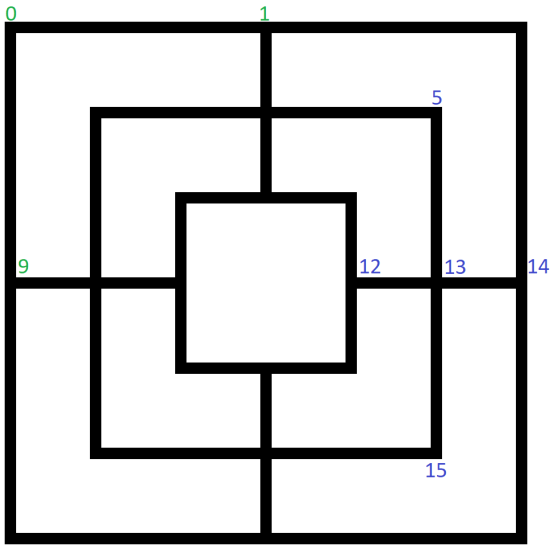

Abbildung 4.2: Datenstruktur für Nachbarschaftsfelder

4.1.3 Linien

Insgesamt gibt es auf einem Mühle Spielbrett 16 Linien. Jede Linie hat eine eindeutige Id von 0-15. Acht Linien sind horizontal ausgerichtet und sind mit den Ids 0-7 definiert. In der Abbildung sind diese mit schwarz dargestellt. Weitere acht Linien sind vertikal ausgerichtet und sind mit den Ids 8-15 definiert. In der Abbildung sind diese mit grün hervorgehoben. Jede Linie besteht aus genau drei Feldern. Die Linie mit der Id 1 besteht aus den drei Feldern mit den Ids 3, 4 und 5. Ein Feld gehört immer zu genau zwei Linien. Das Feld 11 ist Teil der beiden Linien mit den Ids 3 (schwarz) und 10 (grün).

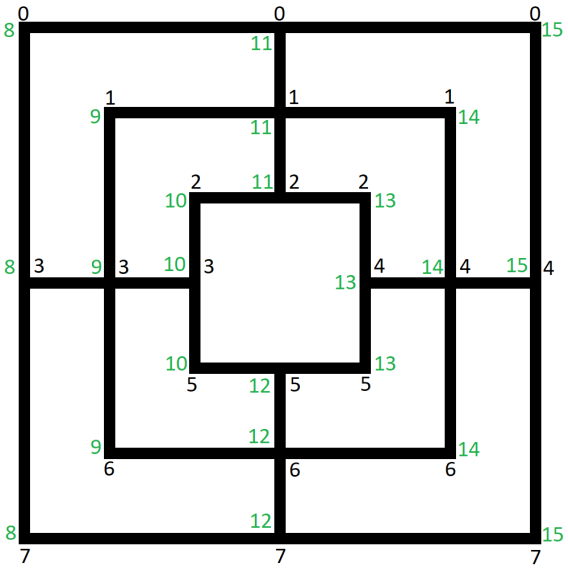

Abbildung 4.3: Datenstruktur für Linien

4.1.4 Spielsteine

Jeder Spieler hat neun gleichfarbige Spielsteine. Jeder Spielstein hat eine eindeutige Id von 0-17. Die Spielsteine mit einer geraden Id 0, 2, 4, etc. gehören zum weißen Spieler. Alle Spielsteine mit einer ungeraden Id 1, 3, 5, etc. gehören zum schwarzen Spieler. Für jeden Spielstein wird sich gemerkt, ob dieser auf dem Spielbrett liegt und ob dieser Spielstein Teil einer Mühle ist.

4.1.5 Spielzüge

Jeder Spielzug wird durch drei Variablen definiert:

1. from
2. to
3. remove

In der Variable "from"wird die Feld id gespeichert von dem der Spielstein kommt. Wenn der Spielstein noch nicht im Spiel ist und noch gesetzt werden muss ist dieser Wert -1. Die Variable "toßpeichert die Feld id auf die der Spielstein verschoben wird. Die Variable "removeßpeichert die Feld id auf dem ein Spielstein entfernt wurde, wenn eine Mühle gelegt wurde. Wenn bei einem Spielzug kein Spielstein entfernt wird ist dieser Wert als -1 definiert. Diese Werte werden für die künstliche Intelligenz benötigt um die verschiedenen Spielzüge abbilden zu können.

4.2 Spieler vs. Spieler

4.2.1 Setzphase

In folgender Abbildung ist die Implementierung der 1. Phase (Setzphase) abgebildet. Zuerst wird in den Zeilen 160 - 161 überprüft welches Feld angeklickt wurde. Erst wenn ein Feld angeklickt wurde wird ein Spielzug ausgeführt. Anschließend wird in der Zeile 162 geprüft, ob der aktuelle Spieler einen Spielstein des Gegners entfernen darf oder ob der Spieler einen Spielstein setzen darf. Wenn der aktuelle Spieler einen Spielstein setzen darf wird in der Zeile 163 ermittelt, ob das angeklickte Feld frei ist. Bei einem freien Feld werden die Werte des Feldes gesetzt und in den Zeilen 176 - 178 wird geprüft ob mit dem gerade gelegten Spielstein eine Mühle gelegt wurde. Wenn ja darf der Spieler einen Spielstein des Gegners entfernen. Wenn nein ist der gegnerische Spieler am Zug. Sobald alle 18 Spielsteine gelegt wurden kommen beide Spieler in den Zeilen 183 - 186 in die 2. Phase (Verschiebenphase).

```
159     public void phase1(MouseEvent e) {
160         currentField = getPressedField(e);
161         if (currentField != -1) {
162             if (removeGamestone == false) {
163                 if (field[currentField].isOccupied() == false) {
164                     field[currentField].setOccupied(true);
165                     field[currentField].setWhichPlayer(game);
166                     field[currentField].setWhichGamestone(gamestone[currentGamestone].getId());
167                     gamestone[currentGamestone].setIsInGame(true);
168                     currentGamestone++;
169                     setGamestonesInMill();
170                     setPaintFieldToZero();
171                     if (field[currentField].getWhichPlayer() == 1) {
172                         field[currentField].setDrawField(3);
173                     } else if (field[currentField].getWhichPlayer() == 2) {
174                         field[currentField].setDrawField(1);
175                     }
176                     if (line[field[currentField].getLine1()].checkMill() == true ||
177                             line[field[currentField].getLine2()].checkMill() == true) {
178                         removeGamestone = true;
179                     }
180                     else {
181                         newRound();
182                     }
183                     if (game.getRoundCounter() >= 17) {
184                         playerWhite.setPlayerPhase(2);
185                     }
186                     if (game.getRoundCounter() >= 18) {
187                         playerBlack.setPlayerPhase(2);
188                     }
189                 }
190             } else {
191                 removeGamestone();
192             }
193         }
194     }
```

Abbildung 4.4: 1.Phase Setzphase Implementierung

Sobald ein Spieler eine Mühle gebildet hat darf er in jeder Phase des Spiels einen gegnerischen Spielstein entfernen. Diese Funktionalität wird mit der Methode removeGamestone() erreicht. Zuerst wird in den Zeilen 389 - 390 kontrolliert, ob auf dem Feld ein gegnerischer Spielstein liegt von dem der Spieler den Spielstein entfernen möchte. Wenn dies der Fall ist wird als nächstes untersucht, ob der gegnerische Spielstein sich in einer Mühle befindet. Bei diesem Fall darf der Spielstein nicht entfernt werden, solange nicht alle gegnerischen Spielsteine in einer Mühle liegen. In den Zeilen 394 - 404 werden dann die Werte für den entfernten Spielstein und dessen Feld neu gesetzt. Anschließend ist der Gegner am Zug.

```
388    public void removeGamestone() {
389        int currentPressedField = field[currentField].getWhichPlayer();
390        if (currentPressedField != 0 && currentPressedField != game.getCurrentPlayer()) {
391            boolean allGamestonesInMill = allGamestonesInMill(game.getCurrentPlayer());
392            if (line[field[currentField].getLine1()].checkMill() == false || allGamestonesInMill == true) {
393                if (line[field[currentField].getLine2()].checkMill() == false || allGamestonesInMill == true) {
394                    field[currentField].setOccupied(false);
395                    field[currentField].removeGamestone();
396                    gamestone[field[currentField].getWhichGamestone()].setIsInGame(false);
397                    if (game.getCurrentPlayer() == 1) {
398                        remainingGamestones(playerBlack);
399                    } else if (game.getCurrentPlayer() == 2) {
400                        remainingGamestones(playerWhite);
401                    }
402                    field[currentField].setDrawField(2);
403                    newRound();
404                    removeGamestone = false;
405                }
406            }
407        }
408    }
```

Abbildung 4.5: Remove Gamestone Methode

4.2.2 Verschiebenphase

In nachstehender Abbildung ist die Implementierung der 2. Phase (Verschiebenphase) abgebildet. In Zeile 294 - 301 wird kontrolliert, ob ein Spielstein ausgewählt wurde und es werden die Nachbarfelder des ausgewählten Feldes ermittelt. Sobald ein Spieler ein Feld ausgewählt hat kann dieser den Spielstein auf ein benachbartes Feld verschieben. Dies ist in den Zeilen 303 - 314 implementiert. Ein Spieler kann nachdem er sich für einen Spielstein entschieden hat den er verschieben möchte nochmals um entscheiden und einen anderen Spielstein auswählen. Dies wurde in den Zeilen 315 - 317 implementiert. Ab der Zeile 319 bis 330 wird entschieden ob ein Spieler in der 2. Phase bereits das Spiel verloren hat. Dies kann nur geschehen wenn der Spieler keinen Spielstein mehr bewegen kann.

Die Methode moveGamestone(int, int) bekommt als Übergabeparameter die ausgewählte Feld id und die Feld id auf das der Spielstein verschoben werden soll. In der Methode wird geprüft, ob das Feld auf das der Spielstein verschoben werden soll ein Nachbarfeld ist. Erst wenn dies erfüllt ist werden die Werte für die beiden Felder neu gesetzt. Im Anschluss wird noch untersucht, ob der verschobe-

```
293    public void phase2(MouseEvent e) {
294        currentField = getPressedField(e);
295        if (currentField != -1) {
296            if (removeGamestone == false) {
297                if (selectGamestone == false) {
298                    if (field[currentField].isOccupied() == true &&
299                            field[currentField].getWhichPlayer() == game.getCurrentPlayer()) {
300                        getNeighborFields();
301                        selectGamestone = true;
302                    }
303                } else {
304                    if (field[currentField].isOccupied() == false) {
305                        if (currentField == neighborField1 || currentField == neighborField2 ||
306                                currentField == neighborField3 || currentField == neighborField4) {
307                            moveGamestone(currentField, selectedCurrentField);
308                            setGamestonesInMill();
309                            if (removeGamestone == false) {
310                                newRound();
311                            } else {
312                                removeGamestone = true;
313                            }
314                        }
315                    } else if (field[currentField].isOccupied() == true &&
316                            field[currentField].getWhichPlayer() == game.getCurrentPlayer()) {
317                        getNeighborFields();
318                    }
```

Abbildung 4.6: 2.Phase Verschiebenphase Implementierung Teil 1

ne Spielstein eine neue Mühle gebildet hat.

```
319                     if (availableMoves(game.getCurrentPlayer()) == 0) {
320                         int opponent;
321                         if (game.getCurrentPlayer() == 1) {
322                             opponent = 2;
323                         } else {
324                             opponent = 1;
325                         }
326                         if (playerWhite.getPlayerPhase() == 2 && opponent == 2) {
327                             gameOver(1);
328                         }
329                         if (playerBlack.getPlayerPhase() == 2 && opponent == 1) {
330                             gameOver(2);
331                         }
332                     }
333                 }
334             } else {
335                 removeGamestone();
336             }
337         }
338     }
```

Abbildung 4.7: 2.Phase Verschiebenphase Implementierung Teil 2

```
519     public void moveGamestone(int currentField, int markedField) {
520         if (currentField == neighborField1 || currentField == neighborField2 ||
521                 currentField == neighborField3 || currentField == neighborField4) {
522             setPaintFieldToZero();
523             if (field[markedField].getWhichPlayer() == 1) {
524                 field[markedField].setDrawField(3);
525             } else if (field[markedField].getWhichPlayer() == 2) {
526                 field[markedField].setDrawField(1);
527             }
528             field[markedField].removeGamestone();
529             field[markedField].setOccupied(false);
530             int temp = field[markedField].getWhichGamestone();
531             field[markedField].setWhichGamestone(-1);
532             field[currentField].setWhichPlayer(game);
533             field[currentField].setOccupied(true);
534             field[currentField].setWhichGamestone(temp);
535             if (field[currentField].getWhichPlayer() == 1) {
536                 field[currentField].setDrawField(3);
537             } else if (field[currentField].getWhichPlayer() == 2) {
538                 field[currentField].setDrawField(1);
539             }
540             // Es wird überprüft ob der neu gesetzte Spielstein eine neue
541             // Mühle gebildet hat.
542             // ... wenn ja darf der aktuelle Spieler dem Gegner einen
543             // Spielstein entfernen.
544             if (line[field[currentField].getLine1()].checkMill() == true ||
545                     line[field[currentField].getLine2()].checkMill() == true) {
546                 removeGamestone = true;
547             }
548             selectGamestone = false;
549         }
550     }
```

Abbildung 4.8: Move Gamestone Methode

4.2.3 Springenphase

Die Implementierung der 3. Phase (Springenphase) ist in folgender Abbildung dargestellt. Diese Spielphase ist mit geringen Veränderungen äquivalent zu der 2. Phase. Da der Spieler sobald er in der 3. Phase ist auf ein beliebiges, freies Feld springen darf werden hier keine Nachbarfelder abgefragt. Die Methode jumpGamestone(int, int) die in Zeile 356 aufgerufen wird ist analog zu der Methode moveGamestone(int, int) mit der Ausnahme das die Nachbarfelder nicht abgefragt werden.

```
344    public void phase3(MouseEvent e) {
345        currentField = getPressedField(e);
346        if (currentField != -1) {
347            if (removeGamestone == false) {
348                if (selectGamestone == false) {
349                    if (field[currentField].isOccupied() == true &&
350                            field[currentField].getWhichPlayer() == game.getCurrentPlayer()) {
351                        selectedCurrentField = currentField;
352                        selectGamestone = true;
353                    }
354                } else {
355                    if (field[currentField].isOccupied() == false) {
356                        jumpGamestone(currentField, selectedCurrentField);
357                        setGamestonesInMill();
358                        if (removeGamestone == false) {
359                            newRound();
360                        } else {
361                            removeGamestone = true;
362                        }
363                    } else if (field[currentField].isOccupied() == true &&
364                            field[currentField].getWhichPlayer() == game.getCurrentPlayer()) {
365                        selectedCurrentField = currentField;
366                    }
367                }
368            } else {
369                removeGamestone();
370            }
371        }
372    }
```

Abbildung 4.9: 3.Phase Springenphase Implementierung

4.3 Minimax-Algorithmus

4.3.1 Einführung

Der Minimax Algorithmus ermittelt eine optimale Spielstrategie für Zwei-Personen-Nullsummenspiele mit vollständiger Information. Zu solchen Spielen zählen zum Beispiel Brettspiele wie Schach, Go, Dame oder Mühle. Der Minimax Algorithmus bildet das Kernelement der Mühle Software und der dahinter liegenden künstlichen Intelligenz. Durch die steigende Rechenleistung von Computern können diese immer mehr mögliche Züge berechnen was dazu geführt hat das die meisten Menschen ohne Mühe von Computern geschlagen werden können.

4.3.2 Bewertungsfunktion

Die Bewertungsfunktion ordnet dem Spielzug einen Wert von +1 zu, wenn Spieler A durch diesen Zug gewinnt, einen Wert von -1 wenn Spieler A verliert und 0 bei einem unentschieden. Wenn der vollständige Suchbaum bis zu seiner maximalen Tiefe aufgebaut wird spielt der Algorithmus ein perfektes Spiel. Dies ist aber nur bei einfacheren Spielen wie etwa Tic-Tac-Toe möglich. Bei komplexeren Spielen wie etwa Mühle wird der Suchbaum nur bis zu einer festgelegten Tiefe aufgebaut und anschließend ermittelt welcher Spielzug ausgeführt wird. Dabei wird die Bewertungsfunktion so modifiziert das gute Spielzüge einen hohen, positiven Wert bekommen und schlechte Spielzüge einen großen, negativen Wert. Umso größer die festgelegte Tiefe des Suchbaums ist umso schwieriger ist die künstliche Intelligenz zu schlagen. Den vollständigen Suchbaum aufzubauen würde bei komplexen Spielen zu viel Rechenleistung benötigen.

4.3.3 Suchbaum Beispiel

In folgender Abbildung ist ein Suchbaum mit der Tiefe 3 dargestellt. Der Knoten der Ebene 0 wird auch als Wurzelknoten bezeichnet und repräsentiert die aktuelle Spielsituation. Die Knoten in der tiefsten Ebene in der Abbildung also Knoten der Ebene 3 werden auch als Blattknoten bezeichnet. In Ebene 0 und 2 werden jeweils die maximalen Werte der darunterliegenden Knoten vererbt. Dabei wird die max-Methode angewendet. In Ebene 1 werden jeweils die minimalen Werte der darunterliegenden Ebene an die Knoten vererbt. Es wird die min-Methode angewendet. In folgender Abbildung wird der nächste Spielzug mit dem Wert 5 durchgeführt, da dieser den höchsten Wert aufweist.

4.3.4 Max Methode

In der max Methode des Minimax Algorithmus wird in den Zeilen 108 - 110 geprüft, ob es einen Spielzug gibt mit dem die KI in den nächsten Spielrunden sicher

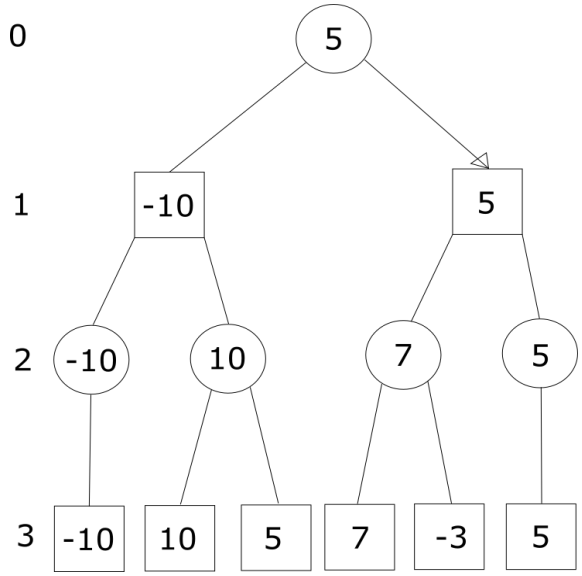

Abbildung 4.10: Suchbaum Beispiel

gewinnen kann. Wenn dies der Fall ist wird der Wert -1000 als Gewicht angenommen und an den Wurzelknoten zurückgegeben. In den Zeilen 111 - 114 werden die Spielzüge in der tiefsten Ebene bewertet und das Gewicht zurückgegeben. Die Zeilen 116 - 119 sind zur Ermittlung der verfügbaren Felder implementiert die in einer Liste eingetragen werden. Wenn diese Liste leer ist, ist kein weiterer Spielzug möglich und es wird wieder das Gewicht -1000 zurückgegeben. Ab der Zeile 120 werden dann die möglichen Spielzüge mithilfe der Funktion placeAMove(move, player) berechnet. In den Zeilen 123 - 129 wird dann das maximale Gewichte ermittelt und an die oberen Ebenen des Spielbaumes weiter gegeben bis dieser im Wurzelknoten angekommen ist. In der Funktion replaceAMove(move, player) werden dann die gesetzten Werte wieder zurückgesetzt und das Spielbrett zurückgebaut um den nächsten Spielzug berechnen zu können.

4.3.5 Min Methode

Die min Methode des Minimax Algorithmus ist fast äquivalent zu der max Methode implementiert. Hier werden positive Gewichte weitergegeben, da die min Methode die gegnerischen Spielzuggewichte ausführt. Die max Methode führt die eigenen Spielzüge durch. Hier werden ab der Zeile 83 die möglichen Spielzüge berechnet und das minimale Gewichte ermittelt und an die oberen Ebenen des Spielbaumes weiter gegeben.

```
107    public int max(int depth, int player) {
108        if (gameOver == true) {
109            return -1000;
110        }
111        if (depth == 0) {
112            int weight = evaluateMove(player);
113            return weight;
114        }
115        int max = Integer.MIN_VALUE;
116        List<Move> availableFields = getAvailableFields(player);
117        if (availableFields.isEmpty()) {
118            return -1000;
119        }
120        for (int i = 0; i < availableFields.size(); i++) {
121            Move move = availableFields.get(i);
122            placeAMove(move, player);
123            int currentScore = min(depth - 1, changePlayer(player));
124            max = Math.max(currentScore, max);
125            if (currentScore >= max) {
126                max = currentScore;
127                if (depth == desiredDepth) {
128                    kiMove = move;
129                }
130            }
131            replaceAMove(move, player);
132        }
133        return max;
134    }
```

Abbildung 4.11: Max Methode Implementierung

```
74    public int min(int depth, int player) {
75        if (gameOver == true) {
76            return 1000;
77        }
78        int min = Integer.MAX_VALUE;
79        List<Move> availableFields = getAvailableFields(player);
80        if (availableFields.isEmpty()) {
81            return 1000;
82        }
83        for (int i = 0; i < availableFields.size(); i++) {
84            Move move = availableFields.get(i);
85            placeAMove(move, player);
86            int currentScore = max(depth - 1, changePlayer(player));
87            min = Math.min(currentScore, min);
88            if (currentScore <= min) {
89                min = currentScore;
90            }
91            replaceAMove(move, player);
92        }
93        return min;
94    }
```

Abbildung 4.12: Min Methode Implementierung

4.3.6 Mühle Bewertungsfunktion

Jeder Spielzugknoten der in der untersten Ebene des Spielbaumes liegt wird durch verschiedene Spielfaktoren ausgewertet. Da der Spielbaum bei Mühle nicht komplett aufgebaut und nur bis zu einer bestimmten Tiefe erstellt werden kann reicht es nicht aus die Spielzüge durch +1 (Gewonnen), 0 (Unentschieden) und -1 (Verloren) auszuwerten. Dies kann nur bei einfacheren Brettspielen wie zum Beispiel Tic-Tac-Toe realisiert werden. Der Spielausgang ist in den meisten Berechnungen im Spielbaum des Brettspiels Mühle nicht vorher abzusehen. Deshalb erfordert es eine umfangreichere Bewertungsfunktion die verschiedene Faktoren des Mühle Spiels berücksichtigt. Die folgende Tabelle stellt die Gewichtung der implementierten Faktoren dar.

Nummer	Faktor	Gewichtung
1	eigene Mühle	+15
2	geg. Mühle	-10
3	mögliche Züge	mögliche Züge * 2
4	geg. mögliche Züge	geg. mögliche Züge * (-2)
5	eigene verbleibende Spielsteine	verbleibende Spielsteine * 7
6	geg. verbleibende Spielsteine	geg. verbleibende Spielsteine * (-7)

Tabelle 4.1: Faktorgewichtungen

4.3.7 Rechenaufwand

Je tiefer der Spielbaum erstellt wird umso mehr Spielbretter werden mithilfe des Minimax Algorithmus aufgebaut. Ab der Tiefe 5 wird deutlich das der Rechenaufwand zum Aufbau des Spielbaums deutlich steigt und die Anwendung reagiert nicht mehr in Echtzeit und benötigt eine gewisse Zeit zur Berechnung. Die folgende Tabelle zeigt die Anzahl der erstellten Spielbretter beim ersten Spielzug der KI. Die Anzahl der erstellten Spielbretter wird unter anderem durch die unterschiedlichen Faktoren des Mühle Spiels beeinflusst.

Tiefe	Anzahl erstellter Spielbretter	Spielmodus
1	24	
2	576	Leicht
3	12720	
4	267744	Normal
5	5.368.224	
6	102.277.344	Schwer
7	1.846.641.504	

Tabelle 4.2: Rechenaufwand

5 Bedienungsanleitung

5.1 Programm Ausführung

Die Mühle Software wurde mit der Programmiersprache Java entwickelt. Zum ausführen der Software wird deshalb eine Java Runtime Enviroment benötigt. Diese kann zum Beispiel unter "http://java.sun.com"heruntergeladen und installiert werden. Nähere Informationen zur Installation von Java befinden sich ebenfalls auf dieser Seite.

Die eigentliche Software ist in ausführbarer .jar Form auf der CD. Durch Klick auf diese Datei wird der Startbildschirm des Mühle Programms gestartet. Die Quelldateien und Bilddateien befinden sich ebenfalls auf der CD in einem separaten Ordner. Die Quelldateien können in Eclipse importiert und dort eingesehen werden. Dafür muss der komplette Projektordner in Eclipse importiert werden.

5.2 Startbildschirm

Im Startbildschirm kann ausgesucht werden ob der weiße oder schwarze Spieler die KI spielt. Wenn keine der beiden ausgewählt wurde können zwei reale Spieler gegeneinander spielen. Der weiße Spieler fängt immer an. Es gibt drei unterschiedliche Schwierigkeitsgrade:

1. Leicht
2. Normal
3. Schwer

Beim Schwierigkeitsgrad Leicht rechnet der KI Gegner 2 Spielzüge vorraus. Bei Normal kann der KI Gegner 4 Spielzüge vorraus rechnen und bei Schwer 6 Spielzüge. Wenn der Schwierigkeitsgrad Schwer ausgewählt wird braucht die KI mehr Zeit um den nächsten Spielzug zu berechnen und die Anwendung läuft nicht mehr in Echtzeit. Deshalb kann es dort zu Verzögerungen kommen und der Spieler muss warten bis er seinen nächsten Spielzug tätigen kann. Wenn auf den Button Spiel beenden geklickt wird, wird die komplette Software beendet. Mit einem Klick auf den Button Spiel starten kann ein neues Spiel mit den ausgewählten Einstellungen gestartet werden.

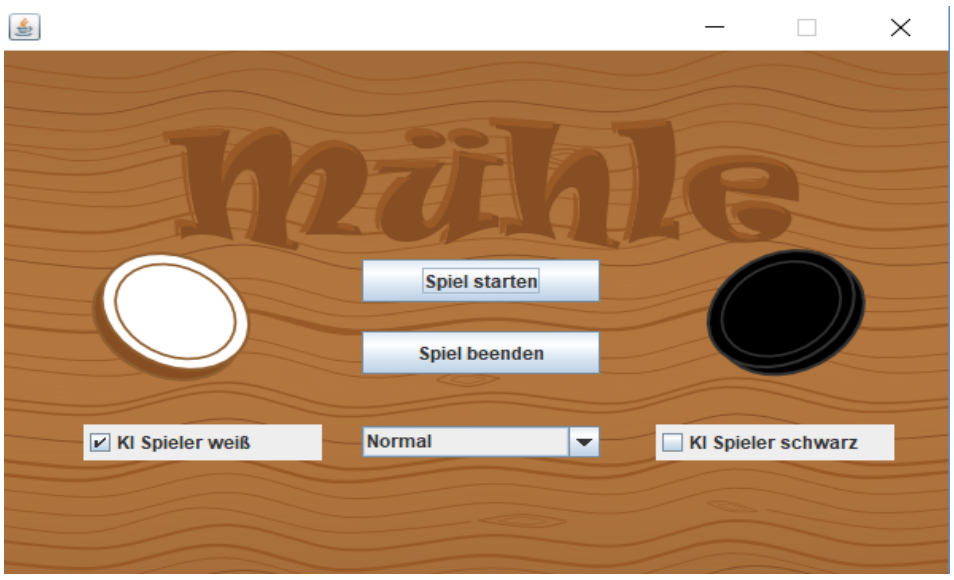

Abbildung 5.1: Mühle Startbildschirm

5.3 Mühle Spielbewegungen

Um einen Spielstein zu setzen muss auf das entsprechende Feld geklickt werden auf dem der Spielstein gesetzt werden soll. Ein schwarzer Rahmen zeigt den letzten Spielzug des schwarzen Spielers an. Ein grüner Rahmen zeigt den letzten Spielzug des weißen Spielers an. Eine rote Umrandung zeigt an das auf diesem Feld

ein Spielstein entfernt wurde. In der Verschiebenphase muss zuerst der Spielstein angeklickt werden mit dem gefahren wird und danach das Zielfeld auf das der Spielstein gesetzt werden soll. In der Springenphase ist dies äquivalent und es kann auf jedes freie Feld gesprungen werden.

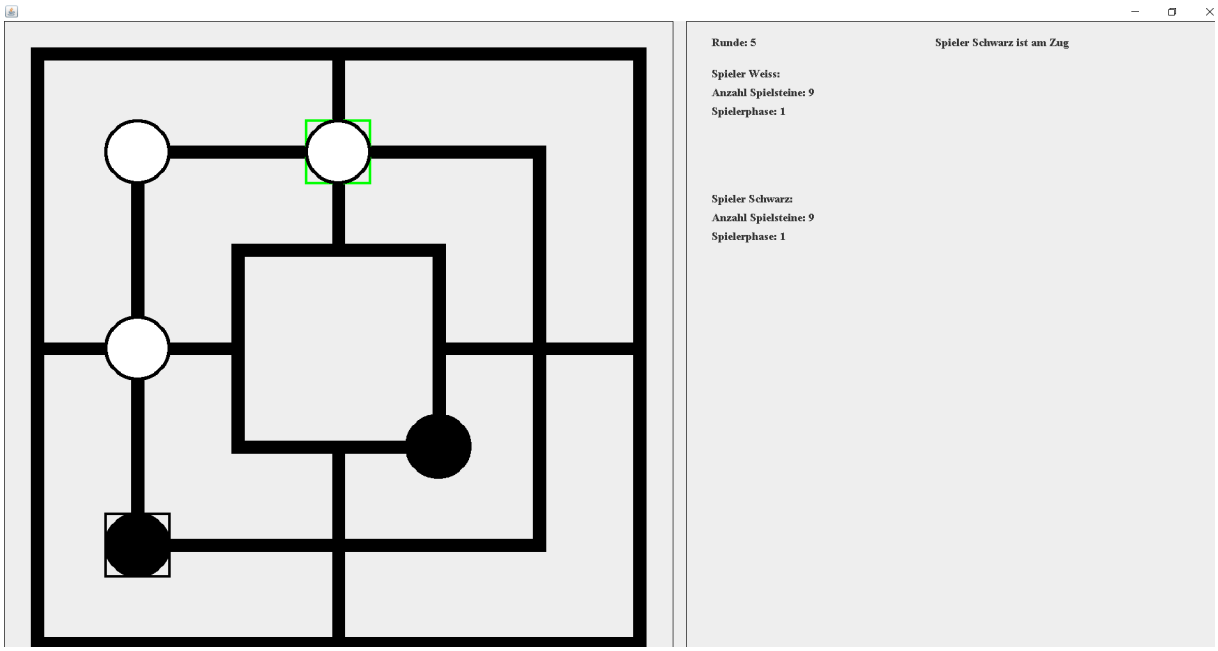

Abbildung 5.2: Mühle Spielbildschirm

Sobald ein Spieler gewonnen hat, weil der Gegner nur noch zwei Spielsteine hat oder mit keinem Spielstein mehr fahren kann wird ein Fenster mit einem Siegertext geöffnet. Mit einem Klick auf den Hauptmenü Button wird wieder der Startbildschirm geöffnet.

Abbildung 5.3: Mühle Endbildschirm

6 Fazit und Ausblick

Das Fazit meiner Projektarbeit ist das die Berechnung des Minimax Algorithmus für Zwei-Personen Brettspiele wie etwa Mühle positiv ausfällt. Die Rekursion des Algorithmus ist auch in tieferen Ebenen effizient und gibt mit der passenden Bewertungsfunktion gute und sinnvolle Spielzüge zurück. Mit steigender Tiefe des Spielbaums werden die Spielzüge des KI-Gegners immer ausgereifter und stellen selbst geübte Mühle Spieler vor einige Schwierigkeiten.

Künstliche Intelligenz wird in Zukunft in unserem alltäglichen Leben immer mehr unbewusst Platz einnehmen. Schon heute nehmen wir manche künstliche Intelligenz nicht mehr als solche wahr. Viele Forschungs- und Entwicklerteams versuchen künstliche Intelligenzen in Alltagsgegenstände zu integrieren wie etwa Autos um den Traum vom autonomen Fahren zu ermöglichen. Diese Entwicklungen sind wichtig und haben auch ihren Stellenwert. Dennoch darf nicht vergessen werden das die Technologie von Menschen kontrollierbar bleiben muss. Erst dieses Jahr musste ein Forschungsteam von Facebook zwei Bots eines KI-Systems abschalten, da diese ihre eigene Geheimsprache entwickelten. An solchen Entwicklungen wird das Potenzial künstlicher Intelligenz sichtbar und das die Forschungen in diesem Bereich noch lange nicht zuende sind.