

Othello

— Entwicklung einer KI für das Spiel —

Patrick Müller, Max Zepnik

18. Februar 2019

Inhaltsverzeichnis

1	Einleitung	2
2	Othello	3
2.1	Spielregeln	4
2.2	Spielverlauf	4
2.3	Spielstrategien	5
2.4	Eröffnungszüge	5
3	Grundlagen	7
3.1	Spieltheorie	7
3.2	Spielstrategien	8
3.2.1	Min-Max	8
3.2.2	Alpha-Beta Pruning	8
3.2.3	Suboptimale Echtzeitentscheidungen	11
3.3	Monte Carlo Algorithmus	13
3.3.1	Algorithmus	13
3.3.2	Überlegungen zu Othello	13
4	Implementierung der KI	15
5	Evaluierung	16
6	Fazit	17

Kapitel 1

Einleitung

Computergegner ..

. test..

text1 ...

am Ende schreiben

auf Fazit beziehen?

Kapitel 2

Othello

Othello wird auf einem 8x8 Spielbrett mit zwei Spielern gespielt. Es gibt je 64 Spielsteine, welche auf einer Seite schwarz, auf der anderen weiß sind. Der Startzustand besteht aus einem leeren Spielbrett, in welchem sich in der Mitte ein 2x2 Quadrat aus abwechselnd weißen und schwarzen Steinen befindet. Anschließend beginnt der Spieler mit den schwarzen Steinen.

Die Spielfelder werden in verschiedene Kategorien eingeteilt (siehe Abbildung 2.1):

- Randfelder: äußere Felder (blaue Felder) [o.V15]
- C-Felder: Felder, welche ein Feld horizontal oder vertikal von den Ecken entfernt sind (vgl. [Ber])
- X-Felder: Felder, welche ein Feld diagonal von den Ecken entfernt sind [o.V15]
- Zentrum: innerste Felder von C3 bis F6 (grüne Felder) [o.V15]
- Zentralfelder: Felder D4 bis E5 [o.V15]
- Frontsteine: die äußersten Steine auf dem Spielbrett um das Zentrum (vgl. [Ort]).

Diese Kategorien sind für die spätere Strategie wichtig.

	A	B	C	D	E	F	G	H
1		C					C	
2	C	X					X	C
3								
4				W	S			
5				S	W			
6								
7	C	X					X	C
8		C					C	

Abbildung 2.1: Kategorien des Spielfeldes

Von Othello gibt verschiedene Varianten. Eine Variante ist Reversi. Die verschiedenen Varianten sind allerdings bis auf die Startposition gleich. Bei der Variante Reversi sind die Zentralfelder noch nicht besetzt und die Spieler setzen die vier Steine selbst, während bei Othello die Startaufstellung fest vorgegeben ist.

2.1 Spielregeln

Othello besitzt einfache Spielregeln, welche im Spielverlauf aber auch taktisches oder strategisches Geschick erfordern. Jeder Spieler legt abwechselnd einen Stein auf das Spielbrett. Dabei sind folgende Spielregeln zu beachten welche auch in Abbildung 2.2 abgebildet sind:

- Ein Stein darf nur in ein leeres Feld gelegt werden.
- Es dürfen nur Steine auf Felder gelegt werden, welche einen oder mehrere gegnerischen Steine mit einem bestehenden Stein umschließen würden. Dies ist im linken Spielbrett durch grüne Felder und im mittleren Spielbrett durch das gelbe Feld hervorgehoben. Das gelbe Feld (F5) umschließt mit dem Feld D5 (blau) einen gegnerischen Stein. Es können auch mehrere Steine umschlossen werden. Allerdings dürfen sich dazwischen keine leeren Felder befinden.
- Von dem neu gesetzten Stein in alle Richtungen ausgehend werden die umschlossenen gegnerischen Steine umgedreht, sodass alle Steine die eigene Farbe besitzen. In dem Beispiel ist das im dem rechten Spielbrett zu sehen. F5 umschließt dabei das Feld E5 (rot). Dieses Feld wird nun gedreht und wird schwarz.
- Ist für einen Spieler kein Zug möglich muss dieser aussetzen. Ein Spieler darf allerdings nicht freiwillig aussetzen wenn noch mindestens eine Zugmöglichkeit besteht.
- Ist für beide Spieler kein Zug mehr möglich, ist das Spiel beendet. Der Spieler mit den meisten Steinen seiner Farbe gewinnt das Spiel.
- Das Spiel endet auch wenn alle Felder des Spielbrettes besetzt sind. In diesem Fall gewinnt ebenfalls der Spieler mit den meisten Steinen seiner Farbe.

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

Abbildung 2.2: valide Zugmöglichkeiten für Schwarz und ausgeführter Zug

2.2 Spielverlauf

Das Spiel wird in drei Abschnitte eingeteilt [Ort]:

- Eröffnungsphase
- Mittelspiel
- Endspiel

Diese Abschnitte sind jeweils 20 Spielzüge lang. Im Eröffnungs- und Endspiel stehen zum Mittelspiel wenige Zugmöglichkeiten zur Verfügung, da entweder nur wenige Steine auf dem Spielbrett existieren oder das Spielbrett fast gefüllt ist und nur noch einzelne Lücken übrig sind. Im Mittelspiel existieren sehr viele Möglichkeiten, da sich schon mindestens 20 Steine auf dem Spielbrett befinden und diese sehr gute Anlegemöglichkeiten bieten.

2.3 Spielstrategien

Wie in anderen Spielen gibt es auch in Othello verschiedene Strategien. Dabei kann beispielsweise offensiv gespielt werden, indem versucht wird möglichst viele Steine in einem Zug zu drehen. Es gibt auch defensive „stille“ Züge. Ein „stiller“ Zug dreht keinen Frontstein um und dreht möglichst nur wenige innere Steine um (vgl. [Ort]).

Generell ist eine häufig genutzte Strategie die eigene Mobilität zu erhöhen und die Mobilität des Gegners zu verringern. Mit dem Begriff Mobilität sind die möglichen Zugmöglichkeiten gemeint. Durch das Einschränken der gegnerischen Mobilität hat dieser weniger Zugmöglichkeiten und muss so ggf. strategisch schlechtere Züge durchführen.

Die Position der Steine auf dem Spielbrett sollte ebenfalls nicht vernachlässigt werden. beispielsweise sollen Züge auf X-Felder vermieden werden, da der Gegner dadurch Zugang zu den Ecken bekommt. Dadurch können ggf. die beiden Ränder und die Diagonale gedreht werden und in den Besitz des Gegners gelangen.

In der Eröffnungsphase sollten die Randfelder ebenfalls vermieden werden, da diese in dieser frühen Phase des Spiels noch gedreht werden können und der taktische Vorteil in einen strategischen Nachteil umgewandelt wird.

2.4 Eröffnungszüge

In der nachfolgenden Tabelle 2.1 sind verschiedene Spieleröffnungen und deren Häufigkeit in Spielen aufgelistet. Spielzüge werden in Othello durch eine Angabe der Position, auf welche der Stein gesetzt wird, dargestellt. Ein vollständiges Spiel lässt sich deshalb in einer Reihe von maximal 60 Positionen darstellen.

Name	Häufigkeit	Spielzüge
Tiger	47%	F5 D6 C3 D3 C4
Rose	13%	F5 D6 C5 F4 E3 C6 D3 F6 E6 D7
Buffalo	8%	F5 F6 E6 F4 C3
Heath	6%	F5 F6 E6 F4 G5
Inoue	5%	F5 D6 C5 F4 E3 C6 E6
Shaman	3%	F5 D6 C5 F4 E3 C6 F3

Tabelle 2.1: Liste von Othelloeröffnungen [Ort]

[Ort] gibt folgende weitere Tipps für Eröffnungen:

- Versuche weniger Steinchen zu haben als dein Gegner.
- Versuche das Zentrum zu besetzen.

- Vermeide zu viele Frontsteine umzudrehen.
- Versuche eigene Steine in einem Haufen zu sammeln statt diese zu verstreuen.
- Vermeide vor dem Mittelspiel auf die Kantfelder zu setzen.

Viele dieser Tipps können auch im späteren Spielverlauf verwendet werden.

Kapitel 3

Grundlagen

3.1 Spieltheorie

In dem folgenden Unterkapitel werden grundlegende Definitionen eingeführt. Diese sind an [RN16] angelehnt.

Definition 1 (Spiel (Game))(vgl. [RN16] S. 162)) Ein **Spiel** besteht aus einem Tupel der Form

$$G = \langle S_0, \text{player}, \text{actions}, \text{result}, \text{terminalTest}, \text{utility} \rangle$$

Definiere Q als die Menge aller Zustände im Spiel.

- Q : Menge aller Spielzustände (*States*)
- $S_0 \in \text{States}$ beschreibt den Startzustand des Spiels.
- Players : Menge aller Spieler: $\text{Players} = \{0, 1\}$
- $\text{player} : Q \rightarrow p \in \text{Players}$ ist auf der Menge der Spieler definiert und gibt den aktuellen Spieler p zurück.
- $\text{actions} : Q \rightarrow 2^Q$ gibt die validen Folgezustände eines gegebenen Zustands zurück.
- $\text{result} : Q \times Q \rightarrow Q$ definiert das Resultat einer durchgeführten Aktion a und in einem Zustand s .
- $\text{terminalTest} : Q \rightarrow \mathbb{B}$ prüft ob ein Zustand s ein Terminalzustand, also Endzustand, darstellt.
$$\text{terminalTest}(s) = \begin{cases} \text{True} & | s \in \text{TerminalStates} \\ \text{False} & | s \notin \text{TerminalStates} \end{cases}$$
- $\text{utility} : \text{TerminalStates} \times \text{Players} \rightarrow \mathbb{R}$ gibt einen Zahlenwert aus den Eingabenwerten s (Terminalzustand) und p (Spieler) zurück.

Positive Werte stellen einen Gewinn, negative Werte einen Verlust dar.

Definition
States davor

Eine spezielle Art von Spielen sind **Nullsummenspiele**.

Definition 2 (Nullsummenspiele) (vgl. [RN16] S. 161)) In einem **Nullsummenspiel** ist die Summe der utility Funktion eines Zustands über alle Spieler 0 ($\sum_{p \in \text{Players}} \text{utility}(p) = 0$).

Dies bedeutet, dass wenn ein Spieler gewinnt mindestens ein Gegenspieler verliert.

Durch den Startzustand S_0 und der Funktion action wird ein **Spielbaum** (**Spielbaum**) aufgespannt.

Definition 3 (Spielbaum (Game Tree))(vgl. [RN16] S. 162)) Ein **Spielbaum** besteht aus einer Wurzel, welche einen bestimmten Zustand (meistens S_0) darstellt. Die Kindknoten der Wurzel stellen die durch actions erzeugten Zustände dar. Die Kanten zwischen der Wurzel und den Kindknoten stellen jeweils die durchgeführte Aktion dar, die ausgeführt wurde um vom State s zum Kindknoten zu gelangen. Diese Kindknoten können wiederum weitere Knoten enthalten oder ein Endpunkt des Baumes (Blatt) darstellen.

umformulieren

Definition 4 (Suchbaum (Search Tree))(vgl. [RN16] S. 163)) Ein **Suchbaum** ist ein Teil des Spielbaums.

Überleitung
einfügen

3.2 Spielstrategien

Es gibt verschiedene Spielstrategien. Im Folgenden werden diese kurz erläutert und anschließend verglichen.

3.2.1 Min-Max

Der erste hier erläuterte Strategie ist der Min-Max Algorithmus. Dieser ist folgendermaßen definiert (siehe [RN16] S.164):

$$MinMax(s) = \begin{cases} Utility(s); & \text{wenn TerminalTest}(s) \\ \max(\{MinMax(Result(s, a)) | a \in Actions(s)\}); & \text{wenn Spieler am Zug} \\ \min(\{MinMax(Result(s, a)) | a \in Actions(s)\}); & \text{wenn Gegner am Zug} \end{cases}$$

Der Spieler sucht den bestmöglichen Zug aus **actions**, der ihm einen für seine Züge einen Vorteil schafft aber gleichzeitig nur „schlechte“ Zugmöglichkeiten für den Gegner generiert. Der Gegner kann dadurch aus allen ehemals möglichen Zügen nicht den optimalen Zug spielen, da dieser in den aktuell enthaltenen Zügen nicht vorhanden ist. Er wählt aus den verfügbaren **actions** nach den gleichen Vorgaben seinen besten Zug aus.

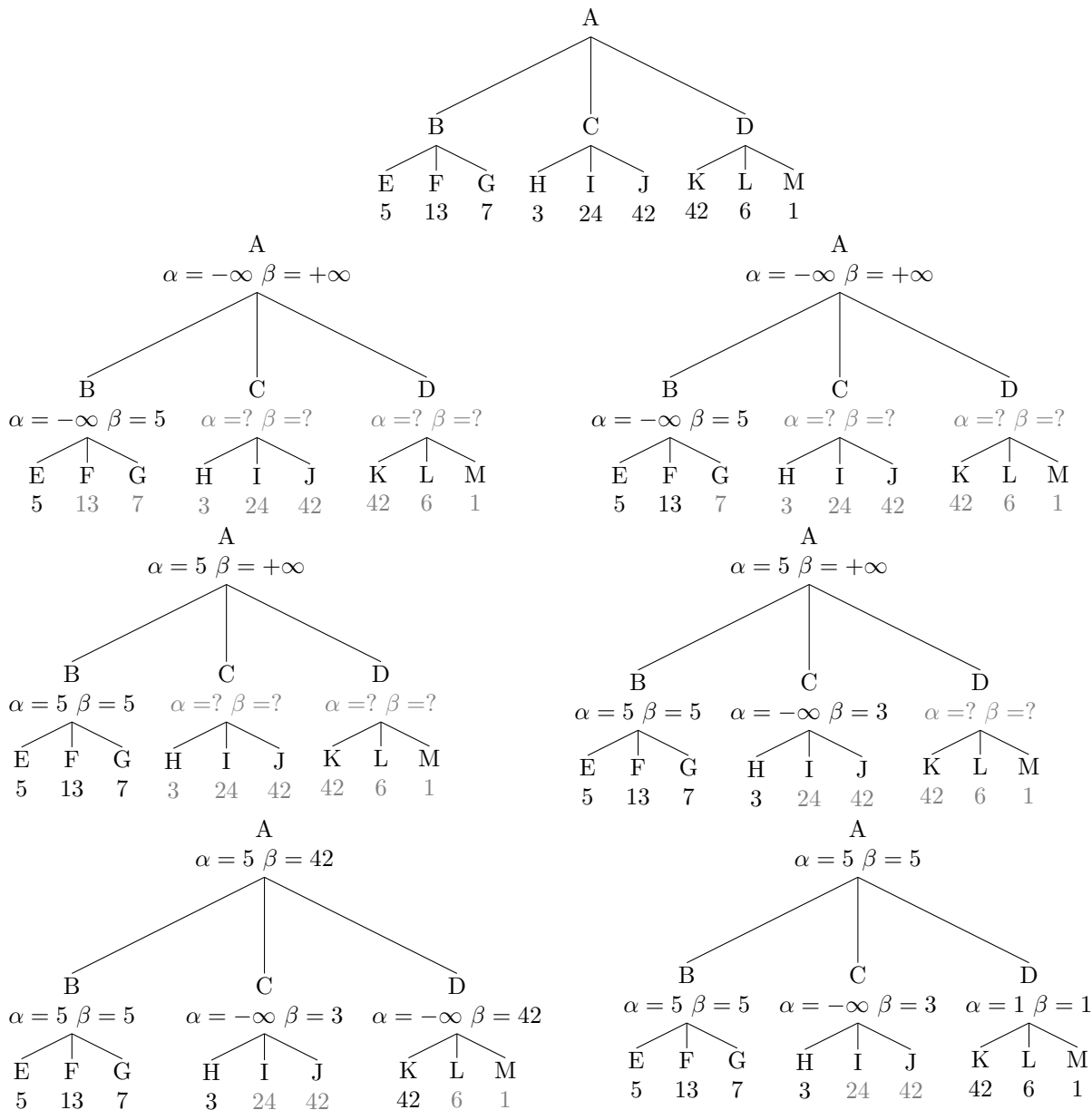
Die Strategie ist eine Tiefensuche und erkundet jeden Knoten zuerst bis zu den einzelnen Blättern bevor ein Nachbarknoten ausgewählt wird. Dies setzt das mindestens einmalige Durchlaufen des gesamten Search Trees voraus. Bei einem durchschnittlichen Verzweigungsfaktor von f bei einer Tiefe von d resultiert daraus eine Komplexität von $O(d^f)$. Bei einem einmaligen Erkunden der Knoten können die Werte aus den Blättern rekursiv von den Blättern zu den Knoten aktualisiert werden. Dadurch muss im nächsten Zug nur das Minimum aus **actions** ermittelt werden, da alle Kindknoten schon evaluiert wurden. Für übliche Spiele kann die Min-Max-Strategie allerdings nicht verwendet werden, da die Komplexität zu hoch für eine akzeptable Antwortzeit ist und der benötigte Speicherplatz für die berechneten Zustände sehr schnell wächst.

3.2.2 Alpha-Beta Pruning

Der Min-Max Algorithmus berechnet nach dem Prinzip „depth-first“ stets den kompletten Spielbaum. Bei der Betrachtung des Entscheidungsverhaltens des Algorithmus fällt jedoch schnell auf, dass ein nicht unerheblicher Teil aller möglichen Züge gar nicht erst in Betracht gezogen wird. Dies geschieht aufgrund der Tatsache, dass diese Züge in einem schlechteren Ergebnis resultieren würden als die letztendlich ausgewählten.

Dem Alpha-Beta Pruning Algorithmus liegt der Gedanke zugrunde, dass die Zustände, die in einem realen Spiel nie auftreten würden auch nicht berechnet werden müssen. Damit steht die dafür regulär erforderliche Rechenzeit und der entsprechende Speicher dafür zur Verfügung andere, vielversprechendere Zweige zu verfolgen.

Abbildung 3.1: Beispielhafter Spielbaum



Demonstration an einem Beispiel

Um den Algorithmus zu verdeutlichen betrachten wir das, an [RN16] angelehnte, folgende Beispiel. Das dargestellte Spiel besteht aus lediglich zwei Zügen, die abwechselnd durch die Spieler gewählt werden. An den Knoten der untersten Ebene des Spielbaum werden die Werte der Zustände gemäß der **utility** Funktion angegeben. Die Werte α und β geben den schlecht möglichsten bzw. den bestmöglichen Spielausgang für einen Zweig, immer aus der Sicht des beginnenden Spielers, an. Die ausgegrauten Knoten wurden noch nicht betrachtet.

Betrachten wir nun den linken Baum in der zweiten Zeile: Der Algorithmus beginnt damit alle möglichen Folgezustände bei der Wahl von B als Folgezustand zu evaluieren. Dabei wird zunächst der Knoten E betrachtet und damit der Wert 5 ermittelt. Dies ist der bisher beste Wert. Er wird als β gespeichert. Eine Aussage über

Warum

den schlechtesten Wert kann noch nicht getroffen werden.

Im nachfolgenden Spielbaum wird der nächste Schritt verdeutlicht. Es wird der Knoten F betrachtet. Dieser hat einen Wert von 13. Am Zuge ist jedoch der zweite Spieler. Dieser wird, geht man davon aus, dass er ideal spielt, jedoch keinen Zug wählen der ein besseres Ergebnis für den Gegner bringt als unbedingt nötig. Der bestmögliche Wert für den ersten Spieler bleibt damit 5.

Nach der Auswertung des Knotens G steht fest, dass es keinen besseren und keinen schlechteren Wert aus Sicht des ersten Spielers gibt. Daraufhin wird die 5 auch als schlechtester Wert in α gespeichert. Ausgehend von A ist der schlechteste Wert damit 5 ggf. kann jedoch noch ein besseres Ergebnis herbeigeführt werden. α wird entsprechend gesetzt und β verbleibt undefiniert.

Nun werden die Kindknoten von C betrachtet. Mit einem Wert von 3 wäre der Knoten H das bisher beste Ergebnis für die Wahl von C. Der Wert wird entsprechend gespeichert. Würde C gewählt gäbe man dem Gegenspieler die Chance ein im Vergleich zu der Wahl des Knotens B schlechteres Ergebnis herbeizuführen. Da Ziel des Spielers jedoch ist die eigenen Punkte zu maximieren gilt es diese Chance gar nicht erst zu gewähren. Entsprechend werden die Auswertung der weiteren Knoten abgebrochen.

Der Kindknoten K des Knotens D ist mit einem Wert von 42 vielversprechend und wird in β gespeichert. Da dieser Wert größer ist als die gespeicherten 5 wird auch der entsprechende Wert von A aktualisiert. Der anschließend ausgewertete Knoten L ermöglicht nun ein schlechteres Ergebnis von 6 β muss also aktualisiert werden. Der Knoten M liefert schließlich den schlechtesten Wert von 1. Da der Gegenspieler im Zweifel diesen Wert wählen würde bleibt der bisher beste Wert das Ergebnis in E. In A wird der Spieler daher B auswählen

Dieses einfache Beispiel zeigt bereits recht gut wie die Auswertung von weiteren Zweigen vermieden werden kann. In der Praktischen Anwendung befinden sich die wegfallenden Zustände häufig nicht nur in den Blättern des Baumes sondern auch auf höheren Ebenen. Der eingesparte Aufwand wird dadurch häufig noch größer.

Implementierung

Nachfolgend wird eine Pseudointerpretation des um Alpha-Beta Pruning erweiterten MinMax Algorithmus angegeben (siehe Listing 3.1):

Listing 3.1: Pseudointerpretation von Alpha-Beta Pruning

```

1 global Suchtiefe
2 int minMax(State AktuellerZustand, int Spieler, int Tiefe, int alpha, int beta) {
3     if (Tiefe == 0) {
4         return Utility(AktuellerZustand, Spieler);
5     }
6     int bisherigerMaximalWert = alpha
7     Zuege = mengeDerFolgezuege(aktuellerZustand);
8     for (Zug z in Zuege) {
9         State NeuerZustand = waehleZug(Aktueller Zustand, z)
10        wert = -minMax(NeuerZustand, anderer(Spieler), Tiefe-1, -beta, -bisherigerMaximalWert)
11        if (wert > bisherigerMaximalWert) {
12            bisherigerMaximalWert = wert
13            if (bisherigerMaximalWert >= beta) {
14                break;
15            }
16            if (Tiefe = Suchtiefe) {
17                speichereZug(z)
18            }

```

```

19     }
20 }
21 return bisherigerMaximalwert;
22 }

```

Es handelt sich um eine rekursive Implementierung. Im Basisfall ist der Spielbaum bereits bis in die angegebene Suchtiefe erforscht (Zeile 3). In diesem Fall wird der Wert der Utility Funktion für den aktuellen Spieler bei dem aktuellen Zustand zurückgegeben (Zeile 4).

Handelt es sich nicht um einen solchen Fall werden alle möglichen Folgezüge berechnet (Zeile 7) und diese dann durch sequenzielle Ausführung bewertet (Zeile 8f). Zuvor wird dazu jedoch der bisherige Maximalwert gespeichert (Zeile 6). Um den Wert des Zuges zu bestimmen wird rekursiv die minMax-Methode erneut aufgerufen. Dabei wird entsprechend der Neue Zustand, der andere Spieler und eine um die um eins verringerte Tiefe übergeben. Der beste Wert für den anderen Spieler ist der schlechteste Wert für den ersten Spieler. Daher wird der bisherige Wert von beta als alpha übergeben. Der bisher beste Wert ist aus Sicht des anderen Spielers der schlechteste daher wird dieser als neues beta übergeben. Da die Utility Funktion so implementiert ist, dass die Summe der Wertigkeiten eines Zustandes Null ergibt muss noch das Vorzeichen geändert werden (Zeile 9).

Ist der neue Wert größer als der bisherige Maximalwert (Zeile 11) so wird dieser aktualisiert (Zeile 12). Da der zweite Spieler versucht die Punktzahl des Gegners zu maximieren bricht dieser die Auswertung aller Zweige ab, bei denen ein Ergebnis, welches besser ist als das bisher schlechteste Ergebnis, möglich wird (Zeile 13f). Abschließend wird der ausgewertete Zug gespeichert um ihn später ausführen zu können (Zeile 16).

Ordnung der Züge

Wie in obigen Beispiel an den Zweigen unter dem Knoten C zu sehen war kann, je nach der Reihenfolge in der die Folgezüge untersucht werden, die Auswertung eines Folgezustandes früher oder später abgebrochen werden. Optimalerweise werden die besten Züge, also jene Züge die einen möglichst frühen Abbruch der Betrachtung eines Knotens herbeiführen zuerst betrachtet. Um dies Abschätzen zu können bedient man sich in der Praxis einer Heuristik die Aussagen über die Güte eines Zuges im Vergleich zu den übrigen Zügen zulässt. Anhand dieser Heuristik kann dann die Reihenfolge der Auswertung einzelner Folgezustände dynamisch angepasst werden.

3.2.3 Suboptimale Echtzeitentscheidungen

Selbst die gezeigten Verbesserung des MinMax-Algorithmus besitzt noch einen wesentlichen Nachteil. Da es sich um einen „depth-first“ Algorithmus handelt muss jeder Pfad bis zu einem Endzustand betrachtet werden um eine Aussage über den Wert des Zuges treffen zu können. Dem steht jedoch die Tatsache entgegen, dass in der Praxis eine Entscheidung möglichst schnell, idealer Weise innerhalb weniger Minuten, getroffen werden soll. Hinzu kommt, dass je nach der verwendeten Datenstruktur für ein Spiel bei entsprechend hohem Verzweigungsfaktor und einer großen Anzahl von Zügen der Hauptspeicher eines handelsüblichen Computers nicht mehr ausreicht um diese zu fassen

Es gilt also eine Möglichkeit zu finden, die Auswertung des kompletten Baumes zu vermeiden.

Heuristiken

Dieses Problem lösen sogenannte Heuristiken. Dabei handelt es sich um eine Funktion die den Wert eines Spielzustandes annähert.

Die Nutzung der Heuristik wird vereinfacht, wenn Sie so definiert ist, dass sie, sofern es sich um einen Endzustand

handelt den Wert der Utility Funktion zurückgibt. Der Vorteil dieses Verhaltens wird im nächsten Abschnitt betrachtet.

Kommt eine Heuristik zur Anwendung so ist die Genauigkeit mit der diese den tatsächlichen Wert approximiert der wesentliche Aspekt der die Qualität des Spiel-Algorithmus ausmacht. Um zu verhindern, dass versehentlich die besten Züge nicht betrachtet werden, ist es essentiell, dass eine Heuristik den tatsächlichen Wert eines Zustandes nie überschätzt. Das unterschätzen des Wertes hingegen ist möglich darf im Sinne der Genauigkeit der Heuristik jedoch nicht allzu ungleichmäßig Auftreten.

Abschnittskriterium der Suche

Gibt die Heuristik im Falle eines Endzustandes den Wert der Utility Funktion zurück, so kann die oben gezeigte Implementierung so angepasst werden, dass statt der Utility Funktion einfach die Heuristik ausgewertet wird. Dadurch muss nicht mehr der Vollständige Zweig durchsucht werden und das Abbrechen nach einer gewissen Suchtiefe wird möglich.

Forward pruning

Forward pruning durchsucht nicht den kompletten Spielbaum, sondern durchsucht nur einen Teil. Eine Möglichkeit ist eine Strahlensuche, welche nur die „besten“ Züge durchsucht (vgl. [RN16] S. 175). Die Züge mit einer geringen Erfolgswahrscheinlichkeit werden abgeschnitten und nicht bis zum Blattknoten evaluiert. Durch die Wahl des jeweils wahrscheinlichsten Zuges können aber auch sehr gute bzw. schlechte Züge nicht berücksichtigt werden, wenn diese eine geringe Wahrscheinlichkeit besitzen. Durch das Abschneiden von Teilen des Spielbaum wird die Suchgeschwindigkeit deutlich erhöht. Der in dem Othello-Programm „Logistello“ verwendete „Probcut“ erzielt außerdem eine Gewinnwahrscheinlichkeit von 64% gegenüber der ursprünglichen Version ohne Forward pruning (vgl. [RN16] S. 175).

Search versus lookup

Viele Spiele kann man in 3 Haupt-Spielabschnitte einteilen:

- Eröffnungsphase
- Mittelspiel
- Endphase

In der Eröffnungsphase und in der Endphase gibt es im Vergleich zum Mittelspiel wenige Zugmöglichkeiten. Dadurch sinkt der Verzweigungsfaktor und die generelle Anzahl der Folgezustände. In diesen Phasen können die optimalen Spielzüge einfacher berechnet werden. Eine weitere Möglichkeit besteht aus dem Nachschlagen des Spielzustands aus einer Lookup-Tabelle.

Dies ist sinnvoll, da gewöhnlicherweise sehr viel Literatur über die Spieleröffnung des jeweiligen Spiels existiert. Das Mittelspiel jedoch hat zu viele Zugmöglichkeiten, um eine Tabelle der möglichen Spielzüge bis zum Spielende aufstellen zu können. In dem Kapitel 2.4 werden die bekanntesten Eröffnungsstrategien aufgelistet. Viele Spielstrategien wie beispielsweise die Min-Max-Strategie setzen den kompletten oder wenigstens einen großen Teil des Spielbaums voraus. Dieser kann entweder berechnet werden oder aus einer Lookup-Tabelle gelesen werden. Je nach Verzweigungsfaktor der einzelnen Spielzüge kann diese allerdings sehr groß sein. Selbst im späten Spielverlauf gibt es verschiedene Spiele, welche einen großen Spielbaum besitzen.

Beispielsweise existieren für das Endspiel in Schach mit einem König, Läufer und Springer gegen einen König 3.494.568 mögliche Positionen (vgl. [RN16] S.176).

Dies sind zu viele Möglichkeiten um alle speichern zu können, da noch sehr viel mehr Endspiel-Kombinationen als diese existieren.

Anstatt die Spielzustände also zu speichern können auch die verbleibenden Spielzustände berechnet werden. Othello besitzt gegenüber Schach den Vorteil, dass die Anzahl der Spielzüge auf 60 bzw. 64 Züge begrenzt sind. Dadurch kann in der Endphase des Spiel ggf. der komplette verbleibende Spielbaum berechnet werden, da die Anzahl der möglichen Zugmöglichkeiten eingeschränkt wird.

Bei der Berechnung der Spielzüge sind die Suchtiefe und der Verzweigungsfaktor entscheidend für die Berechnungsdauer. Aus diesem Grund können im Mittelspiel keine Min-Max-Algorithmen bis zu den Blattknoten des Spielbaumes ausgeführt werden, da die Menge des benötigten Speicherplatzes außerhalb jeglicher Grenzen eines Arbeits- oder Gamingscomputers liegen.

3.3 Monte Carlo Algorithmus

Im Gegensatz zu den bisher gezeigten Algorithmen verwendet der Monte Carlo Algorithmus einen Stochastischen Ansatz um einen Zug auszuwählen. Im nachfolgenden Abschnitt wird die Funktionsweise des Monte Carlo Algorithmus erklärt. Daran angeschlossen folgen Möglichkeiten Strategische Überlegungen zum Spiel Othello einzubringen. Dabei sind die nachfolgenden Ausführungen stark angelehnt an jene von [Nij07].

3.3.1 Algorithmus

Als Ausgangspunkt legt der Monte Carlo Algorithmus die Menge der Züge zugrunde, die ein Spieler unter Wahrung der Spielregeln wählen kann. Diese Züge seien nachfolgend Zug Kandidaten genannt. Enthält die Menge keine Züge, so bleibt dem Spieler nichts anderes übrig als auszusetzen. Enthält die Menge nur einen Zug, so muss der Spieler diesen ausführen. Per Definition ist dies dann der best mögliche Zug. Enthält die Menge hingegen mindestens zwei mögliche Züge, so gilt es den besten unter ihnen auszuwählen. Um den besten Zug zu ermitteln, wird das Spiel N_P mal bis zum Ende simuliert. Während der Simulation wird jeder mögliche Zug gleich häufig gewählt. Der Rest des simulierten Spiels wird dann zufällig zuende gespielt. Sobald alle Durchgänge erfolgt sind, wird das Durchschnittliche Ergebniss für jeden möglichen Zug berechnet. Dabei gibt es zwei Möglichkeiten dieses Ergebnis zu berechnen:

Wahlweise kann die Durchschnittliche Punktzahl eines Zuges oder die durchschnittliche Anzahl an gewonnenen Spielen herangezogen werden. Jener Zug, der nun das beste Ergebnis verspricht wird gespielt.

3.3.2 Überlegungen zu Othello

Bisher spielt der Algorithmus auf gut Glück ohne sich jeglicher Informationen des Spiels zu bedienen. In der Hoffnung das Spiel des Algorithmus zu verbessern, werden nun weitere Informationen zu Othello herangezogen. Hier sein zwei Möglichkeiten beschrieben um dies zu erreichen:

Vorverarbeitung Wie in entsprechenden Kapitel gezeigt, gibt es strategisch gute und strategisch eher schlechte Züge. In seiner Reinform betrachtet der Monte Carlo Algorithmus jedoch beide Arten von Zügen gleich stark. Die Idee der Methode der Vorverarbeitung besteht darin, schlechte Züge in einem Vorverarbeitungsschritt auszuschließen um diese in den Simulationen gar nicht erst zu spielen. Um zu entscheiden, welche Züge ausge-

geschlossen werden, werden die einzelnen Spielzustände nach Ausführung des Zuges bewertet. Dazu werden die im entsprechenden Kapitel beschriebenen Kategorien von Spielsteinen herangezogen und mit einem entsprechenden Punktwert belegt. Für die Entscheidung an sich kann man zwischen zwei Strategien gewählt werden:

Entweder kann mit N_S eine feste Anzahl an best bewerteten Zügen ausgewählt werden oder alternativ eine variable Anzahl. Dies geschieht in dem der Durchschnitt aller Bewertungen bestimmt wird und nur jene Züge ausgewählt werden die eine Bewertung von p_s Prozent des Durchschnittswertes haben.

Pseudozufällige Zugauswahl In der Standardversion des Monte Carlo-Algorithmus werden die simulierten Spiele nach der Wahl des ersten Zuges zufällig zu Ende gespielt. Diesem Ansatz liegt die Idee zu Grunde auch in dieser Phase der Simulation einige Züge anderen gegenüber zu bevorzugen. Dies geschieht nach dem gleichen Prinzip wie im Abschnitt zur Vorverarbeitung beschrieben. Da es jedoch sehr zeitaufwändig ist, die Bewertung jedes einzelnen Spielzustandes innerhalb der Simulationen vorzunehmen, erfolgt dies nur bis zu einer Tiefe N_d .

Kapitel 4

Implementierung der KI







Kapitel 5

Evaluierung

Kapitel 6

Fazit

Notes

 am Ende schreiben	2
 auf Fazit beziehen?	2
 Definition States davor	7
 umformulieren	8
 Überleitung einfügen	8
 Warum	9

Literaturverzeichnis

- [Ber] Berg, Matthias. Strategieführer. <http://berg.earthlingz.de/ocd/strategy2.php>. [Online; accessed 20-January-2019].
- [Nij07] J. A. M. Nijssen. Playing othello using monte carlo, Jun 2007.
- [Ort] Ortiz, George and Berg, Matthias. Eröffnungsstrategie. <http://berg.earthlingz.de/ocd/strategy3.php>. [Online; accessed 20-January-2019].
- [o.V15] o.V. Spiele: Othello. https://de.wikibooks.org/wiki/Spiele:_Othello, 2015. [Online; accessed 20-January-2019].
- [RN16] Stuart J. Russell and Peter Norvig. *Artificial intelligence: A modern approach*. Always learning. Pearson, Boston and Columbus and Indianapolis and New York and San Francisco and Upper Saddle River and Amsterdam, Cape Town and Dubai and London and Madrid and Milan and Munich and Paris and Montreal and Toronto and Delhi and Mexico City and Sao Paulo and Sydney and Hong Kong and Seoul and Singapore and Taipei and Tokyo, third edition, global edition edition, 2016.