1 Vier gewinnt

Die Spielregeln von "Vier Gewinnt" sind sehr einfach: Das Spielfeld besteht aus 7 Spalten und 6 Reihen. Jeder Spieler erhält zu Beginn des Spiels 21 Steine, ein Spieler grüne, der Andere gelbe. Die Steine können im Raster nur vertikal eingeworfen werden.

Die beiden Spieler würfeln zuerst aus, wer beginnt. Abwechselnd werfen die beiden Spieler jeweils einen Stein ihrer Farbe in eine Spalte. Die Steine dürfen in jeder Spalte eingeworfen werden, sofern diese nicht schon mit 6 Steinen voll besetzt ist.

Ziel des Spieles ist es, eine durchgehende Reihe von 4 Steinen seiner Farbe zu erhalten. Dabei ist egal, ob diese Reihe vertikal, horizontal oder diagonal ist.

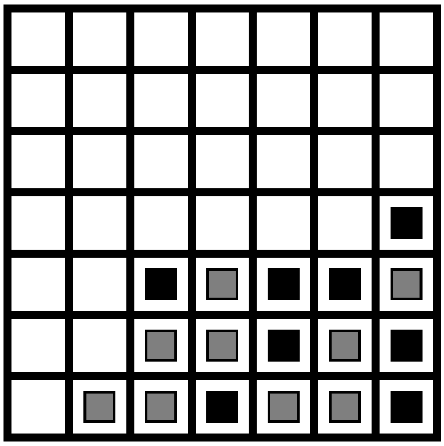

Abbildung 1: Vier gewinnt

2 Spielbäume

Mit Hilfe von Spielbäumen und den entsprechenden Algorithmen ist es Computern möglich, Strategiespiele zu spielen. Bei diesen Spielen handelt es sich um Zweipersonenspiele, bei denen jeder Spieler vollständig weiß, welche Züge bereits gemacht wurden und welche Züge noch möglich sind.

Eine Spielalgorithmus besteht aus folgenden Komponenten:

- *Zuggenerator*
  Aufgabe des Zuggenerators ist es, die möglichen Folgezüge zu ermitteln.

- *Bewertungsfunktion*
  Die Bewertungsfunktion ermittelt für eine gegebene Stellung einen Zahlenwert. Je größer der Betrag dieser Zahl ist, um so besser ist die Stellung. Das Vorzeichen gibt an, für welchen Spieler die Stellung günstig ist.

- *Suchstrategie zum Finden des "besten" Zuges*
  Ziel ist es, den "besten" Zug zu ermitteln, wobei aber möglichst wenige Stellungen bewertet werden sollen.

Die aktuelle Stellung, die möglichen Züge und die daraus resultierenden Folgestellungen werden in einem Suchbaum gespeichert. Jeder Konten im Suchbaum stellt eine mögliche Stellung dar und die Kanten die gültigen Züge, die zu den Stellungen führen. Für komplexe Spiele ist es nicht möglich, den gesamten Spielbaum aufzubauen. Für z.B. Schach wären dazu etwa 10^{120} Knoten nötig. Aus diesem Grund baut man nicht den gesamten Spielbaum auf, sondern nur bis zu einer bestimmten Tiefe und bewertet anschließend die erreichten Stellungen (Blätter). Somit hat ein Spieler das Ziel, den Zahlenwert zu maximieren, der Gegner diesen Zahlenwert zu minimieren.

3 Der MiniMax-Algorithmus

Beim MiniMax-Algorithmus wird der Spielbaum bis zu einer Tiefe t aufgebaut und jedes Blatt bewertet. Die beiden Spieler sollen hier Max und Min heißen.

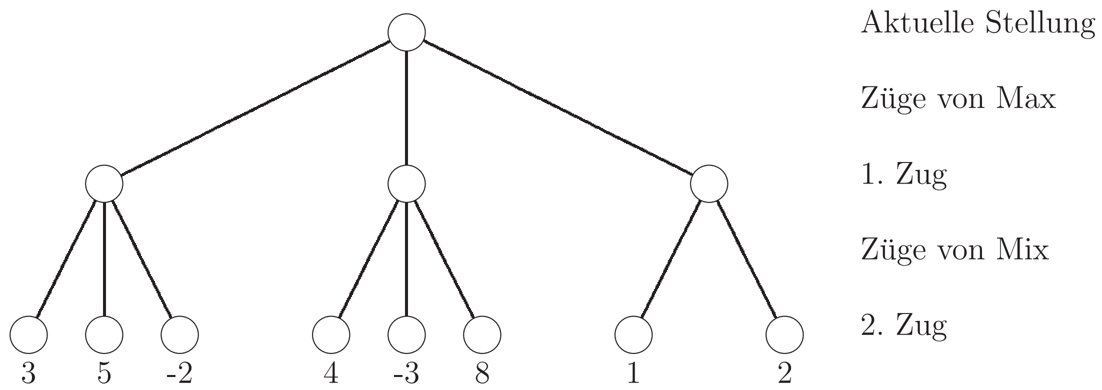

Abbildung 2: Spielbaum

Abb. 3 stellt einen Spielbaum der Tiefe 2 dar. Max ist am Zug. Max hat das Ziel, zu der Stellung zu gelangen, die die höchste Bewertung erhalten hat. Min hat das Ziel, zu der Stellung zu gelangen, die die niedrigste Bewertung erhalten hat

Anhand des Spielbaumes kann Max erkennen, daß er mit dem mittleren Zug eine Stellung mit Bewertung 8 erreichen kann. Ist dieser Zug aber auch der beste Zug für Max? Falls Max diesen Zug ausführt kann anschließend Min eine Stellung erreichen, die mit -3 bewertet ist, d.h. falls Max den mittleren Zug ausführt, wird Min nicht den Zug mit Bewertung 8 sondern den Zug mit Bewertung -3 ausführen. Der beste Zug den Max in dieser Situation ausführen kann, ist der rechte Zug. Falls Max diesen Zug macht, kann Min nur eine Stellung mit Bewertung 1 erreichen.

Der MiniMax-Algorithmus beginnt die Analyse des Spielbaumes eine Ebene über den Blättern. Ist diese Ebene eine Max-Ebene, so weist er dem Knoten das Maximum der Bewertungen der Kinder zu. Ist diese Ebene eine Min-Ebene, so weist er dem Knoten das Minimum der Bewertungen der Kinder zu. Der Wurzelkonten erhält somit die Bewertung, die Max bestenfalls erreichen kann, falls Min immer den für sich besten Zug wählt.

function MiniMax(v)
    if(v ist Blatt)
        return Bewertung von v
    else
        if(v ist MAX-Knoten)
            Bewertung von $v = \max\{$Bewertung von $c \mid c \in children(v)\}$
        else // v ist MIN-Knoten
            Bewertung von $v = \min\{$Bewertung von $c \mid c \in children(v)\}$
        fi
    fi

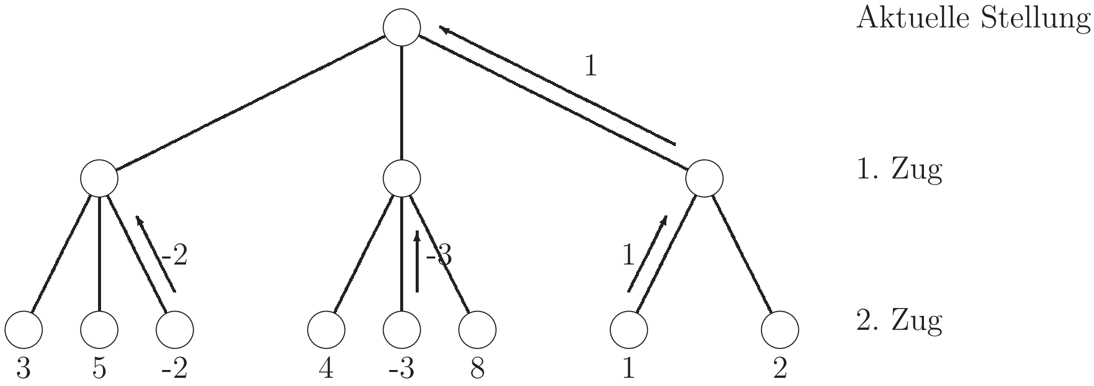

Abbildung 3: MiniMax-Spielbaum

Da das Bewerten einer Stellung sehr rechenaufwendig ist, sollte man sich überlegen, ob die Bewertungsfunktion für jede Stellung aufgerufen werden muss.

4 Das Alpha-Beta-Verfahren

Das Alpha-Beta-Verfahren ist eine Verbesserung des MiniMax-Verfahrens. Betrachten wir den Spielbaum in Abb. 4.

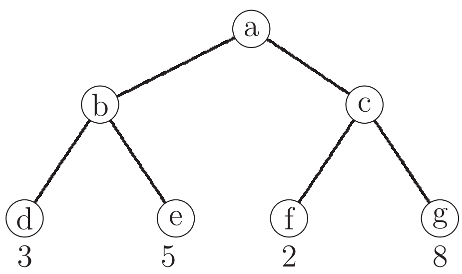

Abbildung 4: Spielbaum

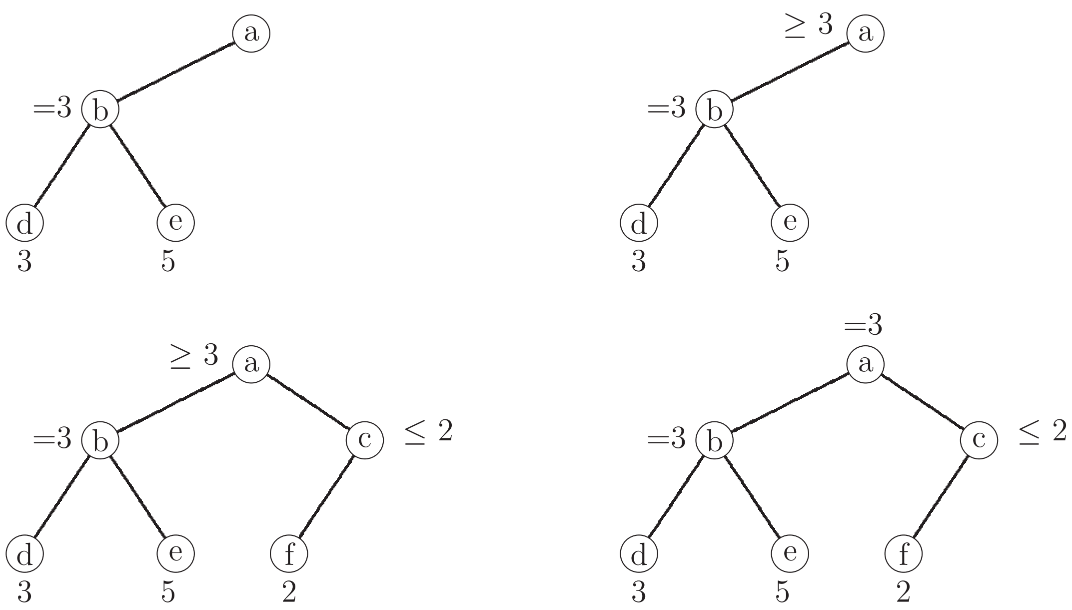

Abbildung 5: Ablauf Alpha-Beta-Suche

Die Knoten erhalten hierbei die in Abb. 5 angegebenen Bewertungen:

Als erstes wird der Knoten b mit 3 bewertet. Das bedeutet für die Wurzel a (Max-Knoten), daß ihre Bewertung nicht kleiner sein kann als 3. Betrachten wir die Bewertung von Knoten c, so reicht es aus, nur die Stellung, die durch Knoten f repräsentiert wird zu berechnen. Hier kann Min eine Stellung mit 2 erreichen, das bedeutet, die Bewertung von Knoten c (Min-Knoten) ist ≤ 2. Da die Wurzel a ein Max-Konten ist, muß der Knoten g nicht mehr betrachtet werden, da die Wurzel schon eine Bewertung von ≥ 3 (> 2) erhalten kann.

Mit Hilfe des Alpha-Beta-Verfahrens werden nur die Teile des Spielbaumes aufgebaut, die für eine Bewertung nötig sind. Knuth und Moore haben gezeigt [3], daß im "besten" Fall dadurch die Anzahl der zu betrachtenden Knoten auf $\sqrt{n}$ sinkt. Der "beste" Fall tritt dann ein, wenn der Zuggenerator immer als erstes den "besten" Zug ermittelt.

Für die Implementierung führt man zwei Hilfsvariablen ein: α und β. Diese beiden Variablen besitzen folgende Eigenschaften:

- Die Bewertung eines Konten liegt immer zwischen α und β
- Der α-Wert eines Knotens wird nie verringert
- Der β-Wert eines Knotens wird nie erhöht
- Ist ein Knoten in einer Max-Ebene abgearbeitet, so entspricht seine Bewertung dem α-Wert
- Ist ein Knoten in einer Min-Ebene abgearbeitet, so entspricht seine Bewertung dem β-Wert

```
function AlphaBeta(v, a, b)
    α = a
    β = b

    if(v ist Blatt)
        return Bewertung von v
    else
        if(v ist MAX-Knoten)
            foreach child c_i of v do
                α = max{α, AlphaBeta(c_i, α, β)}
                if(α ≥ β)
                    break
                fi
            od
            return α

        else // v ist MIN-Knoten
            foreach child c_i of v do
                β = min{β, AlphaBeta(c_i, α, β)}
                if(α ≥ β)
                    break
                fi
            od
            return β
            fi
        fi
```

Der Aufruf AlphaBeta(a, $-\infty$, $+\infty$) berechnet für einen Spielbaum mit Wuzel a den MiniMax-Wert der Wurzel.

5 Optimierungen

Bei vielen Spielen haben die Gegner nicht beliebig lange Zeit, um sich für den optimalen Zug zu entscheiden. Meist ist eine bestimmte Zeit für einen oder mehrere Züge festgelegt. Wird diese Zeitschranke überschritten, gewinnt automatisch der Gegner. Eine vorgegebene Zeitschranke stellt für obige Algorithmen aber ein Problem dar. Wählt man die Tiefe des Suchbaums zu groß, kann der Computer innerhalb der vorgegebenen Zeit evtl. keinen Zug bestimmen, ist die Tiefe zu klein gewählt, wird wertvolle Rechenzeit "verschenkt".

5.1 Vorsortieren der zulässigen Züge

Anhand des AlphaBeta-Algorithmus erkennt man, dass es sinnvoll sein kann, die mögliche Folgezüge nicht in beliebiger Reihenfolge zu bewerten, sondern einen gewinnbringenden Zug zuerst zu bewerten. Mit Hilfe von entsprechenden Heuristiken werden mögliche Folgezüge sortiert und in dieser Reihenfolge im AlphaBeta-Algorithmus betrachtet. Die günstigste Reihenfolge ergibt sich, wenn immer jeder Spieler zuerst den für sich optimalen Zug wählt.

5.2 Hashing von Stellungen

Da beim Aufbau des Spielbaumes mehrmals die selbe Stellung erreicht werden kann, kann man die Bewertung einer Stellung in einer Hashtabelle abspeichern. Wird diese Stellung im Spielbaum nochmals erreicht, so muß diese nicht mehr aufwendig bewertet werden, sondern kann gleich aus der Hashtabelle ausgelesen werden.

5.3 Progressiv Deepening

Beim Progressiv-Deepening analysiert man zuerst den Spielbaum der Tiefe 1, dann der Tiefe 2, usw. Nachdem die vorgegebene Zeit abgelaufen ist, bricht man die Berechnung ab. Der beste bekannte Zug ist der Zug, der in der noch komplett berechneten vorletzten Ebene ermittelt wurde.

Bei der Analyse des zusätzlichen Aufwands geht man davon aus, daß das Bewerten einer Stellung die meiste Rechenzeit kostet. Aus diesem Grund kann man die Analyse der Laufzeit mit der Anzahl der zu bewertenden Stellungen (Knoten) gleichsetzen. Ein vollständiger Baum mit Verzweigungsgrad d hat in Tiefe t d^t Knoten. Die Anzahl der Knoten im gesamten Baum beträgt

$$\sum_{i=0}^{t} d^i = \frac{d^{t+1} - 1}{d - 1}$$

Das Verhältnis der Anzahl der Blätter in der untersten Ebene und aller inneren Knoten des Spielbaumes beträgt:

$$\frac{d^t - 1}{d^t(d - 1)} \approx \frac{1}{d - 1}$$

Anhand dieser Formel kann man erkennen, daß der Aufwand, um einen Baum mit Grad $d = 7$ bis Tiefe $t - 1$ zu bewerten nur $\frac{1}{6}$ des Aufwandes beträgt, der nötig ist, um die unterste Ebene zu bewerten. Aus diesem Grund hält sich der Gesamt-Aufwand für Progressiv Deepening in Grenzen.

5.4 Eröffnungs- und Endspielbibliotheken

Eine Eröffnungsbibliothek ist schon bis zu einer großen Tiefe vorausberechnet. Die besten Eröffnungskombinationen werden gespeichert und können nachgeschlagen werden. Analog sind in Endspielbibliotheken bestimmte Endspielsituationen vorausberechnet, bzw. Zugkombinationen gespeichert (Matt mit Läufer und Springer, Matt mit Turm und Bauer, ...).

6 Bewertungsfunktion für Vier gewinnt

6.1 Vorsortieren der Züge

Folgende Heuristiken bieten sich an:

- Zuerst Steine in die mittleren Spalten
- Anzahl der gleichfarbigen benachbarten Steine

6.2 Bewertung der Stellungen

Eine Stellung ist um gewinnträchtiger, je mehr Chancen bestehen, noch eine Gewinnstellung erreichen zu können. Da bedeutet, je mehr erreichbare Gewinnstellungen für Max existieren, desto besser, je mehr erreichbare Gewinnstellungen für Min existieren, desto schlechter ist diese Stellung.
Als Bewertung einer Stellung bietet sich hier an: Anzahl der möglichen Gewinnstellungen für Max - Anzahl der möglichen Gewinnstellungen für Min.

Desweiteren kann man bewerten, wie weit diese Gewinnstellungen schon erreicht wurden, d.h. eine mögliche Viererkombination, die schon mit 3 Steinen belegt ist wird besser bewertet als eine mögliche Viererkombination, die nur mit 2 Steinen belegt ist, usw.

Literatur

[1] J. Nils Nilsson. Principles of Artificial Intelligence. 1982. Springer-Verlag.

[2] Patrick Henry Winston. Artificial Intelligence. 1993. Addison-Wesley Verlag.

[3] D.E. Knuth and R.W. Moore An analysis of Alpha-Beta pruning. *Artificial Intelligence*. 6: 293–326, 1975.