# B 2 c iv 2. Spielzugberechnung

## 1. Alpha-Beta-Suche als allgemeines Prinzip

Zur Berechnung des nächsten Spielzuges verwendet unsere KI einen Algorithmus zur Alpha-Beta-Suche. Da diese eine Weiterentwicklung des Minimax-Algorithmus ist, soll im Folgenden erst dieser beschrieben werden. Der **Minimax-Algorithmus** ist ein Algorithmus zur Bestimmung von Spielzügen in Zwei-Personen-Spielen. Grundgedanke ist, dass davon ausgegangen wird, dass jeder Spieler den Zug auswählt, der ihm am meisten Vorzüge in der jeweiligen Situation verspricht. Dabei kalkuliert er mögliche Erwiderungen des Gegners ein. Es wird davon ausgegangen, dass dieser Gegner ebenfalls optimal spielt. Kalkuliert die KI jede mögliche Erwiderung des Gegners und die darauf folgende eigene Erwiderung etc. bis zum Ende des Spiels ein, kann sie optimal spielen und ist unabhängig von der Strategie des Gegners.

Zentraler Baustein des Minimax-Algorithmus ist die **Bewertungsfunktion**, die jeder Stellung zuordnet, wie gut sie für den eigenen Spieler ist. Ein Spielfeld, bei dem das eigene Team gewinnt, bekommt einen sehr hohen Wert zugeordnet(z.B. +100), wohingegen ein Spielfeld bei dem der Gegner gewonnen hat einen sehr niedrigen Wert zugeordnet bekommt (-100). Bietet das gegenwärtige Spielfeld für keinen der beiden Spieler einen Vorzug, sollte eine ideale Bewertungsfunktion den Wert 0 zuordnen. Der Spieler wird deshalb versuchen, einen Spielzug durchzuführen, der ein möglichst gut bewertetes Spielfeld für ihn verspricht (**Maximierungsstrategie**). Gleichfalls wird der Gegner diejenigen Spielzüge durchführen, die ihm eine möglichst niedrige Bewertung **(Minimierungsstrategie)** sichern.

Grundsätzlich kalkulieren beide Spieler die Erwiderungen des Gegners mit ein, sodass sich ein **Spielbaum** aufbaut (siehe Abbildung)

Maximierung

Zug in   
Spalte 2

Zug in   
Spalte 1

Spieler A

Zug in   
Spalte 2

Zug in   
Spalte 2

Zug in   
Spalte 1

Zug in   
Spalte 1

Minimierung

Spieler B

Maximierung

Spieler A

Abbildung: Minimax-Spielbaum der Tiefe 3

Der Aufbau dieses Spielbaumes soll kurz erläutert werden. Es wird davon ausgegangen, dass Spieler A am Zug ist und in dem Spiel grundsätzlich die Optionen „Spalte 1“ und „Spalte 2“ zur Verfügung stehen (weil beispielsweise alle anderen Spalten schon voll sind). Der Spieler probiert den Spielzug in Spalte 1 aus (Knoten 1). Er muss nun herausfinden, was der Gegner erwidern würde, um einzuschätzen, wie gut dieser Spielzug ist. Der Gegner (Spieler B) versucht seinerseits mögliche Spielzüge von A zu kalkulieren. Dabei probiert er zunächst die erneute Erwiderung auf Spalte 1 und gelangt zum Knoten 2. Da die Suchtiefe erreicht ist, kommt die Bewertungsfunktion zum Zug, die ermittelt, dass bei dieser Spielzug-Folge Spieler A gewonnen hätte (da die Bewertungsfunktion +100 zurückgibt). Spieler B prüft anschließend als möglichen eigenen Spielzug die Spalte 2 und gelangt zum Knoten 3. Auch hier wird die Bewertungsfunktion aufgerufen, die diesem Spielfeld den Wert -10 zuordnet. Dieser Wert ist wesentlich besser als der in Knoten 2 ermittelte (Ziel für B ist die Minimierung), sodass sich der Gegner in diesem Fall für den Zug in Spalte 2 entscheiden würde. Da jetzt beide Zugoptionen durchkalkuliert wurden kann dem Knoten 1 ein Wert von -10 zugeordnet werden, der dem Minimum der beiden auf der unteren Ebene berechneten Werte (+100, -10) entspricht. Spieler A weiß jetzt also, das der Zug in Spalte 0 eine Bewertung von -10 zur Folge hätte (der Gegner also etwas im Vorteil wäre, da die Bewertung kleiner als 0 ist). Anschließend kalkuliert Spieler A für seinen Knoten 0 den Zug in Spalte 2 und mögliche Erwiderungen des Gegners analog zum eben beschriebenen Verfahren durch. Da dem Knoten eine Wertigkeit von -20 zugeordnet wird, entscheidet er sich gemäß seiner Maximierungsstrategie (er will die höchstmögliche Spielfeld-Bewertung) für den Zug in Spalte 1.

Dieser Minimax-Algorithmus kann durch die Alpha-Beta-Suche verbessert werden. Auffallend ist, dass bei oben gegebenem Spielbaum einige Zweige gar nicht erst durchsucht werden müssen, weil sie sowieso keinen Einfluss auf das Spielergebnis haben würden. Dies soll im Folgenden anhand des obigen Spielbaums erklärt werden.

Maximierung

Zug in   
Spalte 2

Zug in   
Spalte 1

Spieler A

Zug in   
Spalte 2

Zug in   
Spalte 2

Zug in   
Spalte 1

Zug in   
Spalte 1

Minimierung

Spieler B

Maximierung

Spieler A

Abbildung: Mögliches Abschneiden im Spielbaum („Cutoff“)

Wir betrachten die Situation nachdem der Algorithmus wie oben beschrieben zurück zum Ausgangsknoten gegangen ist und jetzt den Zug in Spalte 2 analysiert (rechter Teilbaum). In Knoten 4 sucht er nach möglichen Erwiderungen des Gegners, der seinerseits eine Minimierung des Spielfeldes hinsichtlich der Bewertungsfunktion versucht. Dem Algorithmus ist das Ergebnis von Knoten 1 bekannt, also ein möglicher Spielzug für A, bei dem er im schlechtes Fall mit dem Wert -10 abschneidet. Dieser Wert stellt somit eine untere Schranke dar: Findet B in einem anderen Ast eine Erwiderung, die noch schlechter für A bzw. besser für B (also kleiner als -10) ist, würde sich A auf keinen Fall für diesen Spielzug entscheiden. Genau dies passiert im Knoten 5: Der Gegner könnte so erwidern, dass das Spielfeld mit -12 bewertet wird, also schlechter als in dem Spieler A bekannten Spielzug im linken Teilbaum. Das heißt der Knoten 4 wird mit maximal -12 bewertet, da Spieler B immer den minimalen Ast auswählt. Laut Minimax-Algorithmus würde der Algorithmus jetzt noch Knoten 6 untersuchen, um die optimale Erwiderung für B im Knoten 4 zu finden. Dies ist jedoch für das Ergebnis, nämlich den besten Spielzug für A, unerheblich: Der Wert von 4 kann sich nur noch von -12 aus weiter verringern (im Beispiel auf -20) – er bleibt damit aber immer unter dem Wert des Knoten 1. Das heißt sobald Knoten 5 besucht wurde ist bekannt, das Spieler A den Knoten 4 niemals aufsuchen wird, da er bereits im Knoten 1 einen besseren Spielzug kennt. Das heißt, dass Knoten 6 nicht besucht werden muss, er kann abgeschnitten werden, was auch als „Cutoff“ bezeichnet wird. Die Schranken, zwischen denen der Zielwert liegen muss (also beispielsweise -10 als untere Schranke) werden Alpha und Beta genannt. Dies begründet auch den Namen des Algorithmus: Alpha-Beta-Suche.

## 2. Unsere Bewertungsfunktion

Zentral für die erfolgreiche Implementation der Alpha-Beta-Suche ist die Bewertungsfunktion, die jedem Spielfeld einen Wert zuordnet, der angibt, wie gut die Steine für den Spieler liegen. Eine wichtiger Teilaspekt ist die **Siegmustererkennung:** erkennt die Bewertungsfunktion vier zusammenhängende Steine des eines Spielers, muss sie einen sehr hohen Wert zurückliefern (z.B. +100), gewinnt der Gegenspieler, einen sehr niedrigen (-100). Um ein möglichst effizientes Abschneiden im Alpha-Beta-Algorithmus zu ermöglichen (viele Cutoffs, die Zeiteinsparungen bedeuten) muss es hingegen auch Zwischenstufen geben, die angeben, wie nah ein Spieler am Gewinn ist. Dazu stellt sich die Frage, welche Kombinationen für einen Spieler vorteilhaft sind.

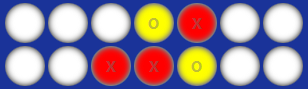
Dazu sei der Begriff der „Falle“ erklärt: Eine Falle ist ein Verbund von Steinen desselben Spielers, die zu vier zusammenhängenden Steinen vervollständigt werden kann. Zwei solcher Fallen sind in folgender Abbildung gegeben:

Abbildung: Mögliche horizontale Fallen (grün und gelb umrandet)

Die grüne Umrahmung zeigt dabei eine 2er-Falle für rot – vier Steine in einer Reihe sind möglich. Darüber befindet sich eine Falle der Größe 1 für Gelb. Der graue Kasten zeigt dabei **keine** Falle: Eine Erweiterung auf vier zusammenhängende Steine ist nicht mehr möglich. Natürlich sind größere Fallen, also Fallen bei denen möglichst schon drei Steine zusammenhängen, besser als kleinere Fallen. Wir haben uns entschieden jede Falle wie folgt zu bewerten:

|  |  |
| --- | --- |
| Fallen-Größe | Punkte |
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |

Diese Punkte werden jeweils zur Bewertung hinzugezählt oder abgezogen. Eigene Fallen führen zu einer höheren Punktzahl (Punkte werden hinzugerechnet), Fallen des Gegners führen zu einer Verminderung der Punktzahl um die jeweilige Zahl. Die Zählweise soll abschließend an einem gefüllten Feld gezeigt werden:

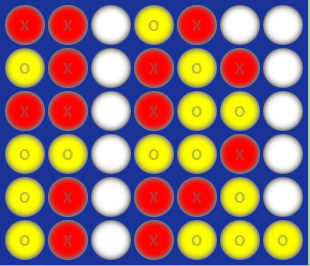


Abbildung: Spielfeld mit markierten Fallen

Punktzahl-Ermittlung (wir sind der gelbe Spieler):

|  |  |
| --- | --- |
| Falle | Punktzahl |
| Gelb |  |
| Horizontal Größe 3 | +9 |
| Diagonal Größe 3 | +9 |
| Vertikal Größe 1 | +1 |
| Rot |  |
| Horizontal Größe 3 | -9 |
| Horizontal Größe 3 | -9 |
| Diagonal Größe 3 | -9 |
|  | **Summe -8** |

Dieses Spielfeld ist also vorteilhaft für den roten Spieler.

## 3. Programmtechnische Umsetzung

Alle zur KI benötigten Klassen befinden sich im Subpackage „KI“ innerhalb von Utilities. Die Hauptklasse, die die Schnittstelle nach außen bildet, ist die **KI.java** mit ihrer Methode **calculateNextMove**. Diese Methode startet einen eigenen Thread, der sich mit der Spielzugsuche mittels Alpha-Beta-Suche beschäftigt (KIThread.java). Dieses Konstrukt dient der Überwachung des Timeouts: Von außen wurde der KI im Konstruktor übergeben, wie viel Zeit für die Spielzugberechnung zur Verfügung steht. Diese Zeit läuft der eigene KI-Thread und schreibt währenddessen die Zwischenergebnisse in eine Variable. Ist die Berechnungszeit abgelaufen, beendet die calculateNextMove-Methode innerhalb der Hauptklasse den Berechnungsthread und gibt das aktuelle Ergebnis der Berechnung an den Aufrufer der calculateNextMove-Methode zurück.

Die Alpha-Beta-Suche ist innerhalb der Java-Klasse **KIThread.java** implementiert. Diesem Thread werden im Konstruktor die aktuellen Parameter zum Spielfeld mitgegeben. Den Rahmen bildet die run-Methode des Threads, welche die Alpha-Beta-Suche startet. Die Methoden Min und Max dienen der entsprechenden Spielzugkalkulation für Spieler A und B. Die Methode Rating dient der Berechnung der Spielfeld-Bewertung wie oben beschrieben.