

Schach, Dame, Mühle, Go, Vier Gewinnt oder Halmal Diese Spiele sind alle grundverschieden und doch lassen sich Gemeinsamkeiten finden, die sich für die Programmierung einer Brettspiel-KI nutzen lassen.

Wir werden uns mit der Theorie der Zugberechnung beschäftigen, die weitgehend Brettspielunabhängig ist. Das heisst, alles, was Sie hier lernen werden, können Sie ohne weiteres auf jedes beliebige Brettspiel anwenden. Die Grundstruktur ist nämlich bei jeder effizienten KI die gleiche. Lediglich die Zug- und Schlagregeln unterscheiden sich. Aber sehen Sie selbst:

Die Gemeinsamkeiten

Zunächst können wir feststellen, dass bei jedem der oben aufgeführten Brettspiele, und um solche soll es vornehmlich gehen, nur zwei Spieler gegen einander antreten. Beide ziehen abwechselnd, wobei Sie für jeden Zug n Zugmöglichkeiten haben. Jeder Spieler versucht natürlich auf Gewinn zu spielen. Mit anderen Worten: beide Spieler versuchen in jeder Situation des Spiels einen bestmöglichen Zug zu machen.

Doch was ist überhaupt ein bestmöglicher Zug? Sofern sich das Spiel nicht kurz vor dem Ende befindet, ist es uns kaum möglich alle Züge voraus zu sehen, die uns zu einem Sieg führen. Nein, vielmehr versuchen wir einen Zug zu finden, der die Brettstellung zu unserem Gunsten gestaltet. Jedesmal also, wenn wir dran sind, schauen wir uns ein paar mögliche Züge an, die wir ausführen können, schauen, was dann wohl passieren könnte, wenn wir diese Züge tätigen, und suchen uns dann den Zug aus, der uns am besten gefällt. Und ob Sie's glauben oder nicht, dieses Prinzip lässt sich auf den Computer übertragen. Die einzelnen Arbeitsschritte lassen sich wie folgt gliedern:

Die Arbeitsschritte

1. Ermittle alle möglichen Züge, die die jeweilige Farbe in dieser Stellung machen kann.
2. Führe aus der Liste aller möglichen Züge den ersten Zug aus und gehe wieder zurück zu Schritt 1.; wiederhole diesen Zyklus x -mal.

Wenn wir also für jeden möglichen Zug alle darauf folgenden Züge berechnen und aus diesen Zügen wieder alle folgenden Züge berechnen usw., dann erstellen wir im Grunde eine Art Stellungsbaum, in dem wir für eine bestimmte Tiefe alle möglichen Stellungen berechnen.

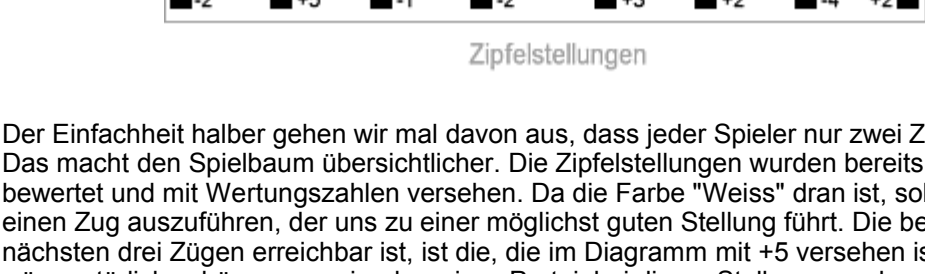
3. Betrachte jede Zipfelstellung und bewerte diese.
4. Wähle den Zug, der zur besten Stellung führt.

Natürlich klingt das hier alles tausendmal leichter gesagt als getan, aber im wesentlichen ist dies der Aufbau, nach der jede Brettspiel-KI arbeitet. Vielleicht wird Ihnen auch nun bewusst, dass sich dieses Prinzip auf jedes beliebige Brettspiel, bei dem nur zwei Spieler gegeneinander antreten, anwenden lässt. Die Unterschiede liegen lediglich in der Zugermittlung (also in den Regeln, nach denen man einen Zug tätigen darf) und in der Stellungsbewertung. Hier liegt der Schlüssel zum Erfolg. Je besser eine Stellung auf dem Brett eingeschätzt werden kann, umso besser arbeitet die KI.

Vielleicht fragen Sie sich nun, was eine Stellungsbewertung ist. Nun, für das spätere Prinzip der richtigen Zugwahl, verwenden wir eine Stellungsbewertung, die sich als Zahl ausdrückt. Stellungen, die eher für Weiss (beim Schach zum Beispiel) besser sind, werden mit einer positiven Zahl versehen. Stellungen dagegen, mit denen eher Schwarz gewinnen kann, bekommen eine negative Zahl. Je näher die Zahl bei Null liegt, desto ausgeglichener ist das Spiel. Diese Zahl kann sich zum Beispiel aus dem Material zusammen setzen, das sich auf dem Brett befindet. Für jeden weissen Bauern beim Schach gibt es +1 Punkte für jeden schwarzen Bauern -1 Punkte. Da ein Turm mehr Wert ist, würde dieser für die weisse Seite zum Beispiel +5 Punkte wert sein und für Schwarz demzufolge -5. Weitere Kriterien, mit denen man eine Stellung bewerten kann, sind zum Beispiel die Positionen der einzelnen Figuren und ihre Bewegungsmöglichkeiten. Die Stellungsbewertung ist an das jeweilige Brettspiel individuell angepasst, die Zugermittlung im wesentlichen immer die gleiche:

Das Minimax-Prinzip

Um also einen vernünftigen Zug berechnen zu können, brauchen wir zunächst einen Stellungsbaum. Soll der Zug der weissen Partei berechnet werden, so ermitteln wir zunächst alle weissen Züge und aus diesen neuen Stellungen alle Gegenzüge (also die von schwarz), dann wieder alle weissen usw. Angenommen, wir sind nun auch in der Lage, die Zipfelstellungen vernünftig zu berechnen, dann könnte ein Stellungsbaum mit der Tiefe 3 etwa so aussehen:

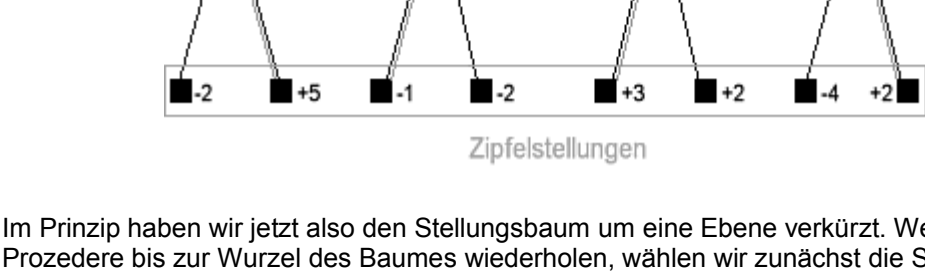


Der Einfachheit halber gehen wir mal davon aus, dass jeder Spieler nur zwei Zugmöglichkeiten hat. Das macht den Spielbaum übersichtlicher. Die Zipfelstellungen wurden bereits von einer Routine bewertet und mit Wertungszahlen versehen. Da die Farbe "Weiss" dran ist, sollten wir versuchen, einen Zug auszuführen, der uns zu einer möglichst guten Stellung führt. Die beste Stellung, die in den nächsten drei Zügen erreichbar ist, ist die, die im Diagramm mit +5 versehen ist (zweite von links). Es wäre natürlich schön, wenn wir, als weisse Partei, bei dieser Stellung raus kommen würden.

Allerdings müssen wir davon ausgehen, dass unser Gegner "Schwarz" natürlich auch versuchen wird, einen für sich guten Zug zu machen. Das bedeutet also, wenn wir den Zug ausführen, der uns in den linken Teilbaum führt, dann würde Schwarz als nächstes den rechten Teilbaum wählen, weil hier Zipfelstellungen liegen, die schlechter sind, als +5. Beide Parteien folgen also einfachen Regeln. Eine Partei (in diesem Fall "Weiss") versucht immer auf eine Stellung mit einem MAXimalwert zu spielen, während die andere Partei auf einen MINimalwert hinspielt. Aus dieser Logik heraus lässt sich das sogenannte Minimax-Prinzip ableiten.

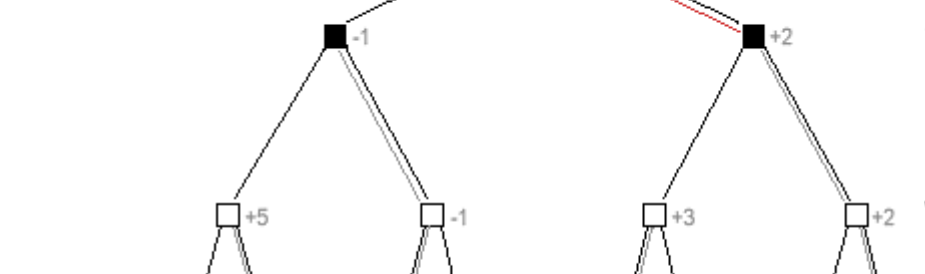
1. Bewerte alle Brettstellungen vor den Zipfelstellungen nach folgenden Regeln: ist Weiss dran, dann suche den Maximalwert, ist Schwarz dran, dann suche den Minimalwert.

In unserem Schaubild sieht das dann so aus:



Im Prinzip haben wir jetzt also den Stellungsbaum um eine Ebene verkürzt. Wenn wir dieses Prozedere bis zur Wurzel des Baumes wiederholen, wählen wir zunächst die Stellungsbewertungen aus, die Schwarz anspielen würde (und zwar die Stellungen mit den möglichst kleinen Bewertungen), und danach die, die Weiss aus der Ausgangsstellung anspielen würde. Hier das Resultat:

2. Wiederhole Schritt 1 bis zur Wurzel des Stellungsbaumes.



Das heisst also, obwohl die offensichtlich beste Stellung im linken Teilbaum liegt, wählen wir den rechten Zug, da uns dieser zu der bestmöglichen Stellung bringt, wenn man berücksichtigt, dass Schwarz versucht, genau das zu verhindern.

Der Algorithmus

Um zu gewährleisten, dass möglichst viele Leute den Minimax-Algorithmus anwenden können, verzichte ich hier auf Delphicode und stelle den Algo in Pseudo-Code dar:

```
function Minimax ( brett, farbe, tiefe ): integer
    solange tiefe > 0
        ermittle alle möglichen zuege der jeweiligen farbe
        für jeden neuen zug rufe auf:
            wert := minimax( neues_brett, not farbe, (tiefe-1) )
        wenn farbe = weiss dann
            wenn result < wert dann
                result := wert
            brett.naechsterzug := zug
        wenn farbe = schwarz dann
            wenn result > wert dann
                result := wert
            brett.naechsterzug := zug
    wenn tiefe = 0
        result := Bewerte_Stellung( brett )
ende - function
```

Sie sollten mit rekursiven Aufrufen bereits vertraut sein, um den Code verstehen zu können. Der Algorithmus macht zwei Dinge gleichzeitig: er erstellt den Stellungsbaum für eine vorgegebene Tiefe und gibt die Bewertung der Knoten zurück. Ist der Knoten ein Zipfel, so wird eine Stellungsbewertungsroutine aufgerufen, die einen Wert ermittelt. Hat dieser Knoten noch nachkommen, dann wird nach dem Rückgabewert des rekursiven Aufrufs entschieden, ob eine gute Stellung für die jeweilige Farbe gefunden wurde oder nicht.

Das Zeitproblem

Prinzipiell reicht dieser Algorithmus völlig aus, um den besten Zug in einer Brettstellung zu ermitteln. Allerdings ist lediglich, wie realistisch ihre Bewertungs-Routine die Zipfelstellungen einschätzt. Allerdings gibt es ein anderweitiges Problem: in Spielen, wie Schach oder Dame haben Sie nicht nur zwei Möglichkeiten zur Auswahl, sondern manchmal sogar 20. Auf jeden Zug müssen 20 weitere Züge berechnet werden, auf diese 20 Züge 20 weitere Züge und so weiter. Hier findet also eine Potenzierung statt, die enorm ist. Der zeitliche Rechenaufwand wird schon bei geringen Tiefen nicht mehr vertretbar.

Ein einfaches Beispiel: Gehen wir einmal davon aus, dass unser Computer durchschnittlich 0,00001 Sekunden braucht, um einen Knoten zu berechnen (was unter optimierten Bedingungen durchaus realistisch ist). Da wir nicht immer 20 Möglichkeiten haben (speziell am Anfang des Spiels nicht) nehmen wir ferner an, dass wir durchschnittlich 15 Möglichkeiten pro Zug haben. Hieraus ergibt sich folgende Rechnung:

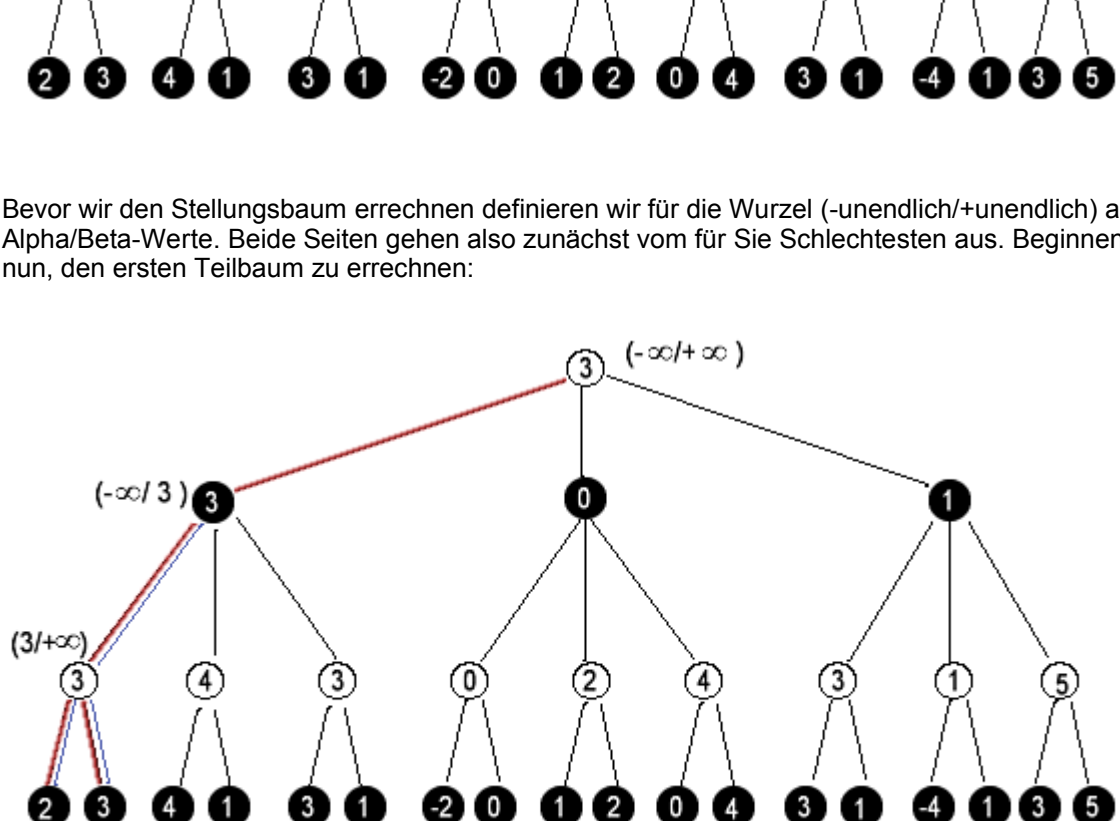
t	Zipfel (15 ^t)	Knoten insgesamt 15 ^t *(t-0)...15 ^t *(t-1)	Zeit 0,00001 * Knoten insgesamt
1	15	16	0,00016 Sek
2	225	241	0,00241 Sek
3	3375	3616	0,03616 Sek
4	50625	54241	0,54241 Sek
5	759375	813616	8,14 Sek
6	11390625	12204241	122 Sek
7	170859375	183063616	30,5 min
8	2562890625	2745954241	7,6 Stunden

Wenn man bedenkt, dass der Computer eigentlich erst ab einer Tiefe von 4 beginnt einigermaßen intelligente Züge zu machen und erst ab Tiefe 6 den Laien unter den Spielern überlegen ist, dann kann man sich vorstellen, dass eine Partie gegen den Rechner mit dem bisherigen Algorithmus ziemlich lange dauern kann. Am Anfang ist der Rechner vielleicht noch schnell, weil es nicht so viele Zugmöglichkeiten gibt. Aber gehen Sie mal eine Schachstellung im Endspiel durch. Allein die Dame hat, wenn Sie in der Mitte des Brettes steht, 27 Zugmöglichkeiten. Rechnen wir noch sämtliche anderen Figuren und deren Möglichkeiten zusammen, kommen wir im Endspiel beim Schach locker auf über 40! Bei einer Suchtiefe von 6 würde die Berechnung über 11 Stunden in Anspruch nehmen. Bei einer Tiefe von 8 sogar über 2 Jahre.

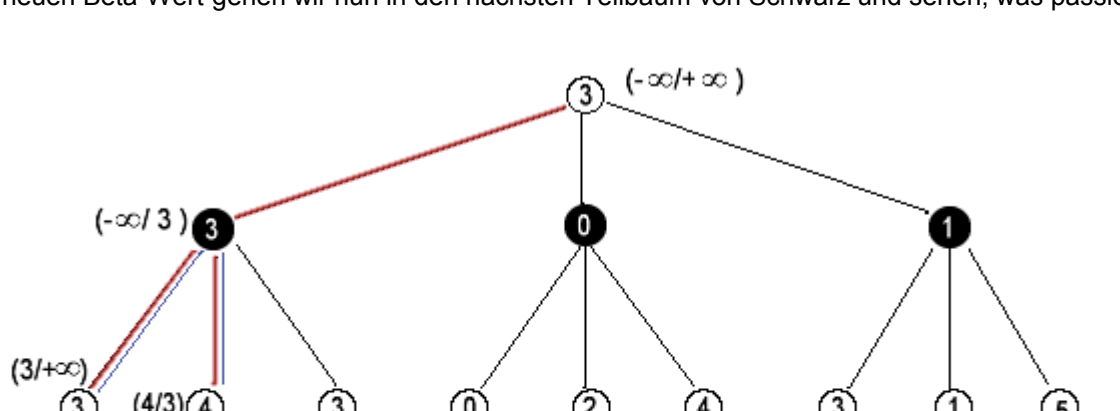
Der Alpha- Beta-Cutoff

Das Minimax-Verfahren ist die sicherste Methode den besten Zug innerhalb einer bestimmten Tiefe zu finden. Doch, wie es oben deutlich wird, ist der Zeitaufwand so hoch, dass eine praktische Anwendung des Algorithmus in Brettspielen kaum denkbar war. Obwohl dieser Algorithmus schon in den 1930er Jahren auf den ersten Papiermaschinen zum Einsatz kam, sollten noch ein paar Jahre ins Land gehen, bis die Wissenschaftler Alan Newell, John Shaw und Herbert Simon 1958 an der Carnegie-Mellon Universität entdeckten, dass man ganze Teile des Baumes weglassen kann und trotzdem, durch reine Logik, den bestmöglichen Zug bestimmen kann. Die Rede ist von dem sogenannten Alpha-Beta-Cutoff, der den Suchalgorithmus extrem beschleunigt. Dieses Verfahren brachte den Durchbruch in der Brettspiel-KI und machte die Computer von damals und heute so extrem leistungsfähig.

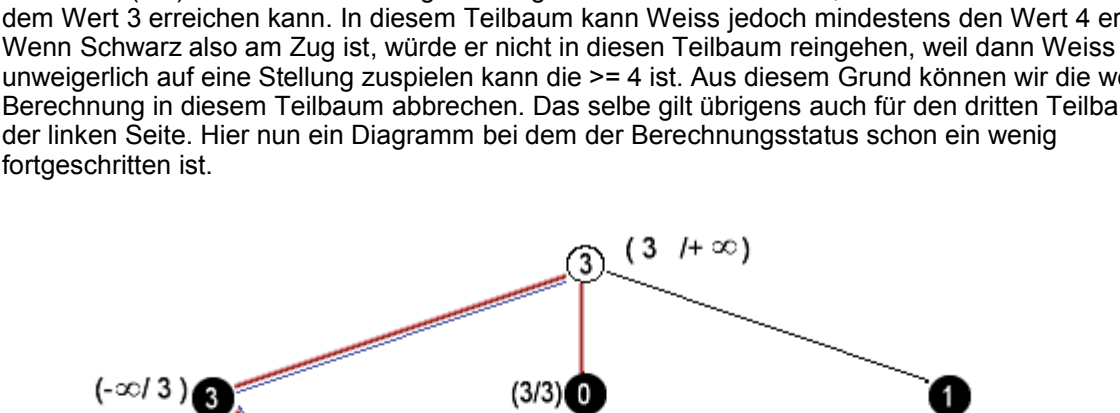
Das besondere beim Alpha-Beta-Cutoff ist, dass man sich für jeden Knoten im Baum immer den besten Wert für Weiss und für Schwarz merkt, den die jeweilige Farbe erreichen kann. Dies sind die sogenannte Alpha- und Beta-Werte. Alpha bezeichnet die beste Wertung, die Weiss erreichen kann und Beta die beste Wertung, die Schwarz erreichen kann. Betrachten Sie hierzu folgendes Stellungsdiagramm:



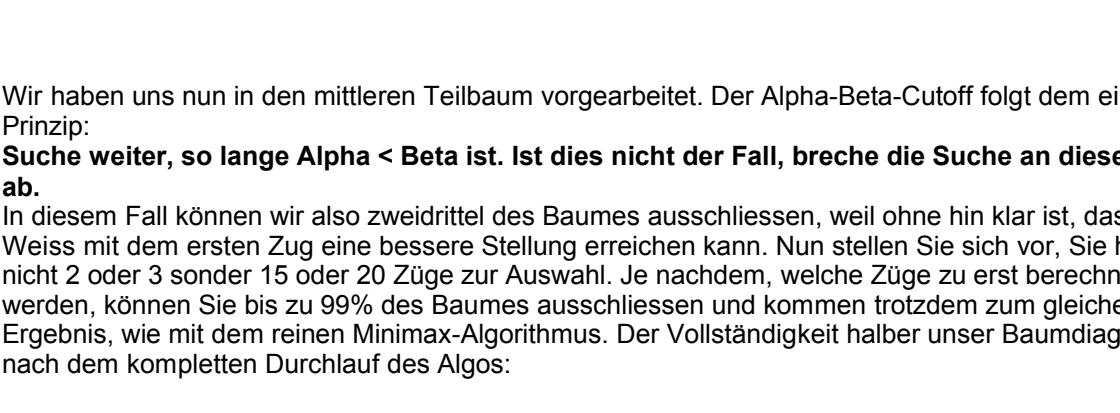
Bevor wir den Stellungsbaum errechnen definieren wir für die Wurzel (-unendlich/+unendlich) als Alpha/Beta-Werte. Und diesen gehen also zunächst vor für Sie Schlechtesten aus. Beginnen wir nun, den ersten Teilbaum zu errechnen:



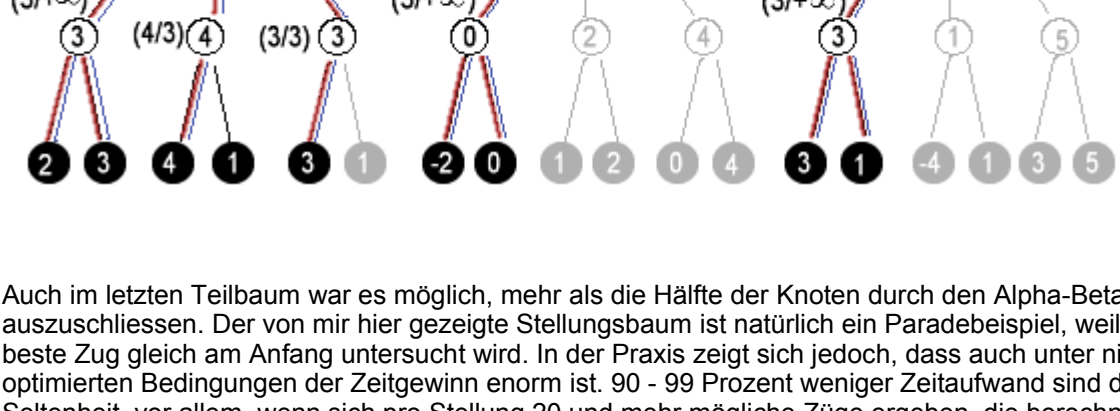
Mit dem Aufruf eines Unterknoten, werden die Alpha-Beta-Werte weitergegeben. Das heisst, bis zum Zipfel haben alle Knoten den Alpha-Beta-Wert (-unendlich, +unendlich). Im oberen Beispiel haben wir inzwischen die Zipfel mit den Wertungen 2 und 3 berechnet. Der beste Wert für Weiss ist natürlich 3. Da 3 grösser ist als (-unendlich), das ist ja der Alpha-Wert von Weiss, wird dieser Wert durch den neuen ersetzt. Der weisse Alpha-Wert wird an einen schwarzen Knoten zurückgegeben. Schwarz ist, wie wir wissen, immer darum bemüht, einen möglichst kleinen Wert zu erreichen. Da der Beta-Wert von Schwarz zur Zeit (+unendlich) ist, ist 3 natürlich geringer und wird zum neuen Beta-Wert. Mit dem neuen Beta-Wert gehen wir nun in den nächsten Teilbaum von Schwarz und sehen, was passiert.



Die beste Stellung, die Schwarz also erreichen kann, hat den Wert 3. Der nächste Zipfel, den wir untersuchen hat den Wert 4 (der dritte von links). Der Alpha-Beta-Wert von dem weissen Unterknoten lautet also (4/3). Hier können wir folgende Logik anwenden: wir wissen, das Schwarz eine Stellung mit dem Wert 3 erreichen kann. In diesem Teilbaum kann Weiss jedoch mindestens den Wert 4 erreichen. Wenn Schwarz also am Zug ist, würde er nicht in diesen Teilbaum reingehen, weil dann Weiss nämlich überlegen auf eine Stellung zuspielen kann die ≥ 4 ist. Aus diesem Grund können wir die weitere Berechnung in diesem Teilbaum abbrechen. Das selbe gilt übrigens auch für den dritten Teilbaum auf der linken Seite. Hier nun ein Diagramm bei dem der Berechnungsstatus schon ein wenig fortgeschritten ist:



Wir haben uns nun in den mittleren Teilbaum vorgearbeiten. Der Alpha-Beta-Cutoff folgt dem einfache Prinzip:
Suche weiter, so lange Alpha < Beta ist. Ist dies nicht der Fall, breche die Suche an dieser Stelle ab.
 In diesem Fall können wir also zweidrittel des Baumes ausschliessen, weil ohne sie hin klar ist, dass Weiss mit dem ersten Zug eine bessere Stellung erreichen kann. Nun stellen Sie sich vor, Sie haben nicht 2 sondern 15 oder 20 Züge zur Auswahl. Je nachdem, welche Züge zu erst berechnet werden, können Sie bis zu 99% des Baumes ausschliessen und kommen trotzdem zum gleichen Ergebnis, wie mit dem reinen Minimax-Algorithmus. Der Vollständigkeit halber unser Baumdiagramm neben dem, mit dem reinen Minimax-Algorithmus. Der Vollständigkeit halber unser Baumdiagramm neben dem, mit dem reinen Minimax-Algorithmus.



Auch im letzten Teilbaum war es möglich, mehr als die Hälfte der Knoten durch den Alpha-Beta-Cutoff auszuschliessen. Der von mir hier gezeigte Stellungsbaum ist natürlich ein Paradebeispiel, weil der beste Zug gleich am Anfang untersucht wird. In der Praxis zeigt sich jedoch, dass auch unter nicht optimalen Bedingungen der Zeitgewinn enorm ist. 90 - 99 Prozent weniger Zeitaufwand sind da keine Seltenheit, vor allem, wenn sich pro Stellung 20 und mehr mögliche Züge ergeben, die berechnet werden müssten.

Es gibt natürlich noch viele andere Optimierungsmethoden. Diese sind aber zumeist auf das jeweilige Spiel spezialisiert. Der Minimax-Algorithmus und der Alpha-Beta-Cutoff sind aber die Grundsteine jeder guten Brettspiel-KI und die richtige Stellungsbewertung der Schlüssel zum Erfolg.

Die Stellungsbewertung

Um eine komplexe Stellung durch eine einfache Wertungszahl auszudrücken, gibt es viele Möglichkeiten. Grade im Schach hat sich dazu eine eigene Quasi-Wissenschaft entwickelt. Ein guter Einstieg ist, jeder möglichen Spielfigur einen Wert zu geben. Wird eine Stellung bewertet, addieren Sie für jeden weissen Stein seinen jeweiligen Wert der Stellungsbewertung hinzu. Ist der Stein schwarz, subtrahieren Sie. Diese Art von Bewertung hat schonmal zur Folge, dass sich Ihre KI nicht ohne weiteres von Ihnen schlagen lässt. Und wenn sich die Möglichkeit ergibt, wird Ihre KI ausserdem die Möglichkeit wahrnehmen, Ihr Material zu verringern. Jedoch werden Sie ein aktives Spiel Ihres Computergegners vermissen. Basiert die Stellungsbewertung auf rein materiellen Werten, werden die Züge der KI planlos und langweilig sein. Er wird nicht wirklich versuchen Sie anzugreifen und so schnell wie möglich auf Gewinnen zu spielen. Lösen lässt sich dieses Problem durch diverse konkretere Abfragen, vor allem, wie dem Stellung und die Nähe zum Gegner betreffen. Kriterien für eine gute Stellung lassen sich am besten im Selbstversuch entdecken. Viele Denkeweisen, die der Mensch im Spiel anwendet, lassen sich am Computer übertragen.

Und ich hoffe, diese groben Ausführungen haben Ihnen den ein oder anderen Denkanstoss gegeben und vielleicht dazu motiviert, sich an eine eigene Brettspiel-KI zu wagen. Ich kann Ihnen nur sagen: es lohnt sich! Wer man so ein Programm schreibt, und wenn sich die Möglichkeit ergibt, wird Ihre KI ausserdem ein eigenes Monster erschaffen hat. Und im Gegensatz zu Frankenstein werden Sie sich darüber freuen, wenn Ihr Monster Sie plötzlich schlägt! :-)