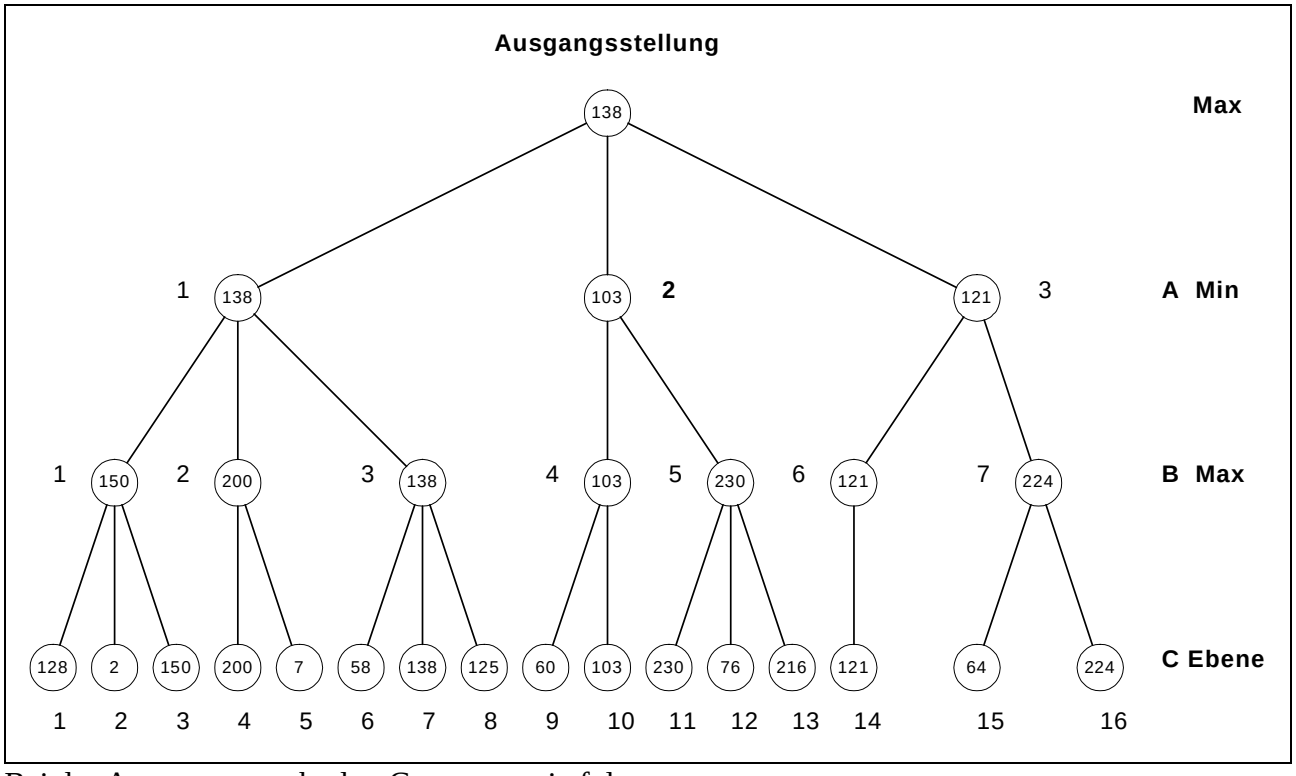
Schneller, tiefer, schlauer

Für Strategiespiele werden **Datentypen** für das Spielfeld, die Figuren und die Züge benötigt. Implementiert werden müssen die Prozeduren *Zuggenerator*, *Bewertungsfunktion* und *Zugausführung*. Die Leistungsfähigkeit des Rechners steigt mit den verwendeten Algorithmen.

Am Strategiespiel "Vier Gewinnt" sollen die leistungsfähigen Verfahren vorgestellt werden. Bei "Vier Gewinnt" gibt es für jeden Zug höchstens sieben Zugmöglichkeiten. Der Computer braucht diese Möglichkeiten nur durchzuprobieren und den Zug mit der höchsten Bewertung auswählen. Tatsächlich funktioniert diese Methode. Da aber nur ein Halbzug vorausberechnet wird, spielt ein solcher Algorithmus sehr schwach.
Erst durch Ausbau der Rechentiefe kann eine wesentliche Leistungssteigerung erzielt werden. Es sind 7^{Tiefe} Halbzüge zu generieren und zu bewerten. Leider ist es nicht ohne weiteres möglich, mehrere Züge im Voraus zu berechnen, da der Computer nicht weiß, welchen Zug der Gegner als nächsten ausführen wird. Das Programm geht daher davon aus, dass der Spieler den für sich günstigsten Zug – und damit den für den Computer ungünstigsten Zug – spielen wird. Tut der Gegner dies nicht, so war zwar die Berechnung und der daraus resultierende Zug falsch, aber dadurch, dass der Gegner einen für ihn ungünstigeren Zug gewählt hat, verschlechtert sich seine Stellung und er verschafft damit dem Computer einen Vorteil.
Dies alles wird von dem sogenannten **Minimax-Algorithmus** durchgeführt.

Von der Ausgangstellung ausgehend hat der Computer z.B. drei Möglichkeiten zu ziehen. Auf diese Züge kann der Gegner mit maximal drei Möglichkeiten je Stellung reagieren. Es entstehen die Positionen B_1 bis B_7. Der nächste Computerzug spaltet den Spielbaum dann weiter in C_1 bis C_{16} auf.

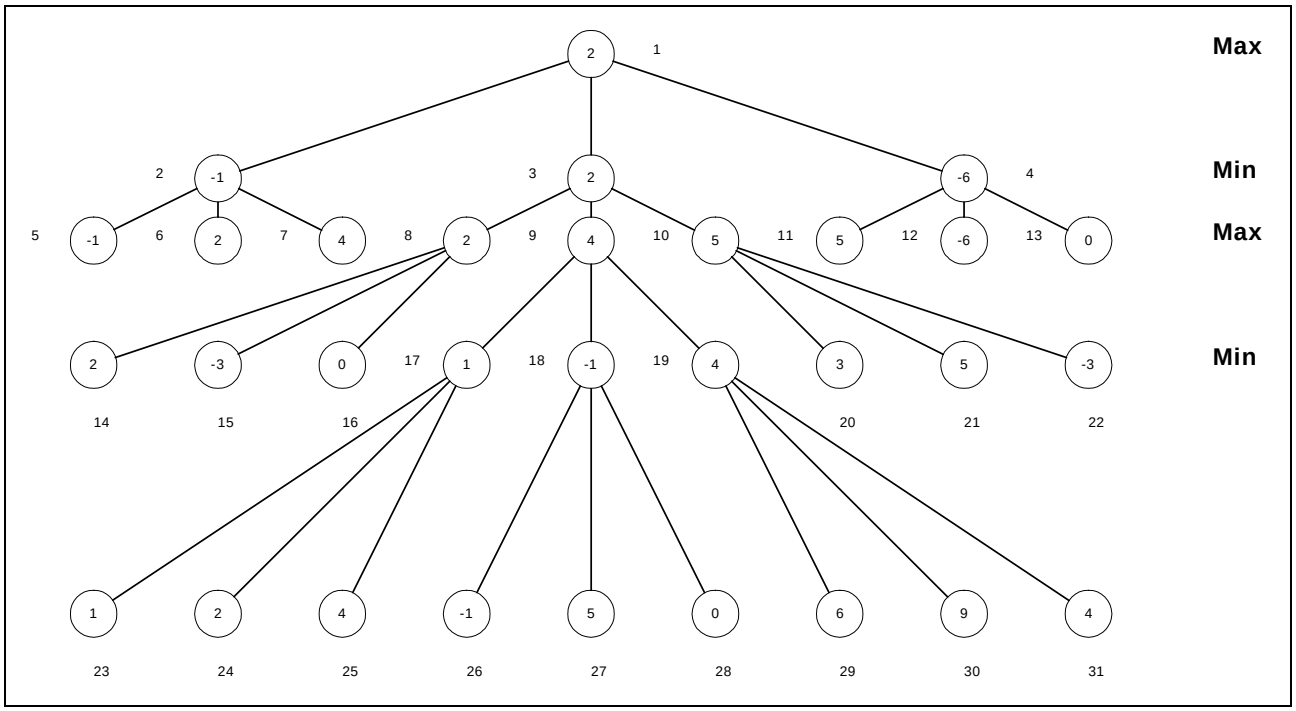


Bei der Auswertung geht der Computer wie folgt vor:
zuerst bewertet er alle Stellungen der C-Ebene. Wenn der Computer jetzt entscheiden muss welchen Zug er als nächsten machen soll (A_1 bis A_3), so kann er nicht einfach die C-Stellung nach dem größten Wert durchsuchen (C_{11}) und dann den Zug wählen, der zu dieser Stellung

führt (A_2). Würde er dies tun, so wählt der Gegner einfach B_4 und **nicht** B_5. Damit wäre der Rechner automatisch auf C_9 oder C_{10} festgelegt (60 bzw. 103 Punkte). So einfach geht es also nicht. Nehmen wir an, das Spiel befindet sich in der Position B_1. Für den nächsten Zug stehen die Stellungen C_1 bis C_3 zur Verfügung. Der Computer wählt C_3, da dieser Zug die meisten Punkte einbringt. Diesen **Maximalwert** speichert der Rechner in B_1 ab. Das gleiche wird mit den anderen B-Stellungen durchgeführt. Ausgehend von den drei Möglichkeiten in Ebene A hat der Gegner sieben Wahlmöglichkeiten. Er wird versuchen, eine Stellung mit **minimalstem** Wert zu erreichen. So erhalten A_1, A_2 und A_3 die Punktezahl 138, 103 und 121. Der Computer wählt den **maximalen** Wert A_1.
Damit hat der Algorithmus den aus Rechner Sicht besten Zug ermittelt. Bei der Wahl von A_1 kann er sicher sein, nach zwei weiteren Halbzügen eine Position zuerreichen, die mindestens 138 Punkte erzielt(bei *Fehlern* des Gegners auch ***mehr***). Da der Algorithmus abwechselnd minimale und maximale Werte berechnet, wird er als **Minimax-Algorithmus** bezeichnet.

In der Praxis reicht es allerdings nicht aus, nur die Endpositionen zu bewerten. Es muss bei jeder einzelnen Stellung überprüft werden, ob einer der Spieler gewonnen hat oder das Spiel zu ende ist. Bei großen Rechentiefen ist es möglich, Rechenzeit zu sparen, wenn der Minimax-Algorithmus mit *mehr Intelligenz* ausgestattet wird.
Nehmen wir an, der Computer hätte bereits für A_1 den Wert 138 berechnet und ist gerade damit beschäftigt für A_2 den minimalen Wert der B-Ebene zu berechnen. B_4 liefert den Wert 103, so ist es **egal** welchen Wert B_5 liefern wird, A_2 wird in jedem Fall nicht **größer** als 103 werden können, da ja der **minimalste** Wert übernommen wird. Der Wert 103 von B_4 ist bereits kleiner als der Wert 138 von A_1. Deshalb ist es ***völlig unnötig*** B_5 zu bestimmen, denn wegen B_4=103 kann A_2 gar nicht größer als 103 werden, und da A_1 bereits den Wert 138 besitzt, ist der ganze Zweig, der an A_2 hängt aus dem Rennen.
Diese Logik funktioniert auf jeder Ebene und wird als **Alpha-Beta-Optimierung** bezeichnet.

Beschneidung des Spielbaumes mittels Alpha-Beta-Algorithmus



$K_3 = Min(K_8, K_9, K_{10}) = Min(2, K_9, K_{10}) = Min(2, 4, K_{10}) \Rightarrow$ es müsste $K_{10} < 2$ sein um in die Bewertung aufgenommen zu werden.

$K_{10} = Max(K_{20}, K_{21}, K_{22}) = Max(3, K_{21}, K_{22}) \Rightarrow K_{21}, K_{22}$ werden **nicht** untersucht !

$K_9 = Max(K_{17}, K_{18}, K_{19}) = Max(1, K_{18}, K_{19})$

$K_{18} = Min(-1, K_{27}, K_{28}) \Rightarrow K_{27}, K_{28}$ werden **nicht** untersucht !

$K_1 = Max(-1, 2, K_4)$, $K_4 = Min(5, -6, K_{13}) \Rightarrow K_{13}$ wird **nicht** untersucht !

Der untergeordnete Knoten erhält Informationen über die Elternknoten durch α und β.
α wird an Max-Knoten gesetzt, β an Min-Knoten.
Max -Knoten setzen α und terminieren, wenn die aktuelle Bewertung > β wird.
Min -Knoten setzen β und terminieren, wenn die aktuelle Bewertung < α wird.

Spielfeld für Tic-Tac-Toe

7 ↘	4 ⇩	5 ⇩	6 ⇩	8 ↙
1 ⇨	**1** 3: ***1 4 7***	**2** 2: ***1 5***	**3** 3: ***1 6 8***	
2 ⇨	**4** 2: ***2 4***	**5** 4: ***2 5 7 8***	**6** 2: ***2 6***	
3 ⇨	**7** 3: ***3 4 8***	**8** 2: ***3 5***	**9** 3: ***3 6 7***	

Spielfeld bei 4 gewinnt

36	**37**	**38**	**39**	**40**	**41**	**42**
29	**30**	**31**	**32**	**33**	**34**	**35**
22	**23**	**24**	**25**	**26**	**27**	**28**
15	**16**	**17**	**18**	**19**	**20**	**21**
8	**9**	**10**	**11**	**12**	**13**	**14**
1	**2**	**3**	**4**	**5**	**6**	**7**