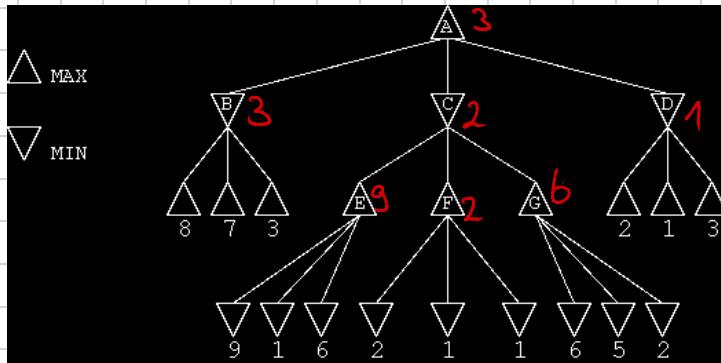



Übungsblatt : Spiele

Games 01: Handsimulation: Minimax & alpha-beta-Pruning



1. Minimax Bewertung

B (Min) : Blätter 8, 7, 3 → 3 wird genommen

E (Max) : Blätter 9, 1, 6 → 9 wird genommen

F (Max) : Blätter 2, 1, 1 → 2 wird genommen

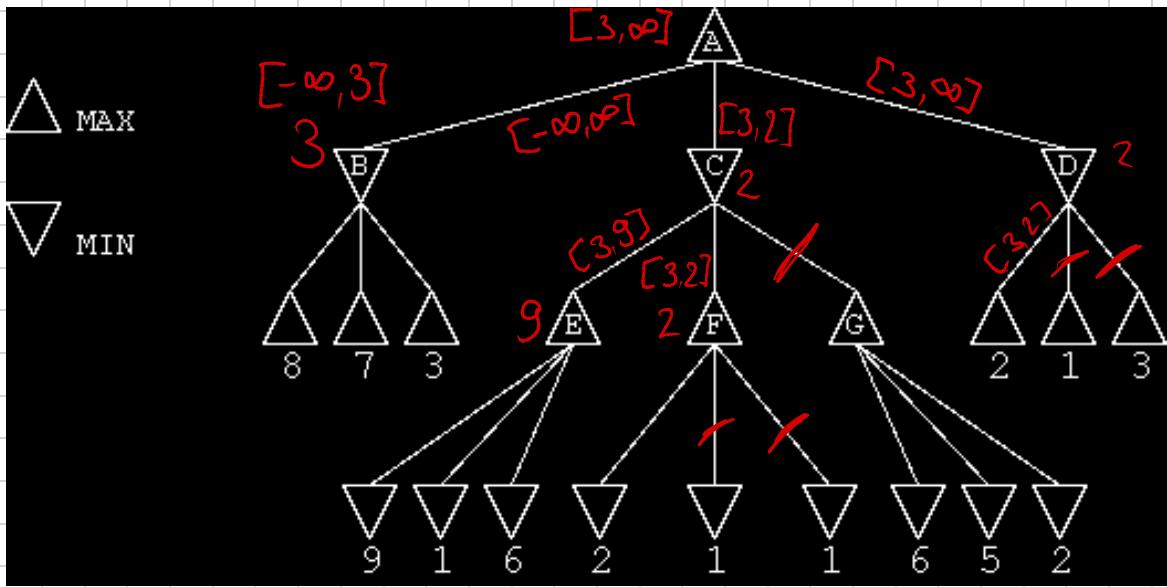
G (Max) : Blätter 6, 5, 2 → 6 wird genommen

C (Min) : Knoten 9, 2, 6 → 2 wird genommen

D (Min) : Blätter 2, 1, 3 → 1 wird genommen

A (Max) : Knoten 3, 2, 1 → 3 wird genommen

2. Alpha - Beta - Pruning



Pruning Regeln

- 1) Schneide unter Min-Knoten ab, deren $\beta \leq$ dem α des Max-Vorgängers ist.
- 2) Schneide unter Max-Knoten ab, deren $\alpha \geq$ dem β des Min-Vorgängers ist.

3. Maximieren des alpha-beta-Pruning

Das Vertauschen von D mit B sowie E mit F ändert an sich die Anzahl der beim alpha-beta-Pruning abgeschnittenen Zweige nicht. Auch das optimieren der Ordnung der Blätter bringt meiner Ansicht nichts, also Max Knoten immer zuerst die großen Werte und bei Min-Knoten immer zuerst die kleinsten.

Grund dafür ist meiner Meinung nach die flachere Struktur des Baumes und eine relativ ähnliche Werte Verteilung.

Games. 03: Minimax vereinfachen

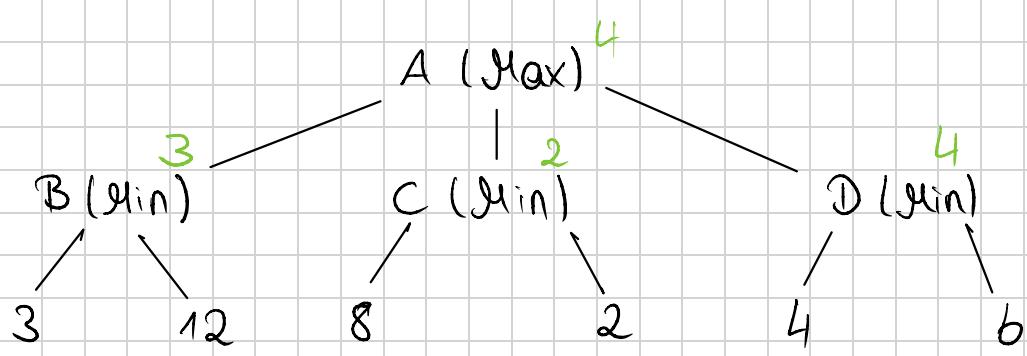
Im klassischen Minimax-Algorithmus werden zwei Funktionen verwendet: `max_value()` steht für den Max Spieler und `min_value()` für den Spieler Min. Max ist der Spieler, welcher immer den höchsten Wert anstrebt und Min der den niedrigsten anstrebt. Da TicTacToe zu den Nullsummenspielen gehört, gilt hierfür der Gewinn des einen ist der Verlust des anderen. ($U_{\text{Max}}(s) = -U_{\text{Min}}(s)$)

Dadurch besteht die Möglichkeit den Algorithmus in eine gemeinsame Funktion zu packen. Dieser wird der Negamax-Algorithmus genannt.

Die Idee dahinter ist einen Parameter "color" einzuführen, welcher angibt, welcher Spieler am Zug ist. $+1 \rightarrow \text{Max}$, $-1 \rightarrow \text{Min}$

Beim Spielerwechsel wird das Vorzeichen von "color" umgedreht.

```
def negamax(state, color):  
    if terminal(state):  
        return color * utility(state)  
    best = -INF  
    for a in actions(state):  
        v = -negamax(result(state, a), -color)  
        best = max(best, v)  
    return best
```



Minimax

B (Min) : Blätter 3, 12 → 3 wird genommen

C (Min) : Blätter 8, 2 → 2 wird genommen

D (Min) : Blätter 4, 6 → 4 wird genommen

A (Max) : Blätter 3, 2, 4 → 4 wird genommen

Negamax

Start bei A (Max) → color = +1

→ Werte der Kinder = -Negamax (child, -1)

Für B (Min, color = -1):

- Kinder: 3, 12

- Negamax = $\max(-3, -12) = -3$

↪ Wert zurück an A = $-(-3) = 3$

Für Knoten C ($\text{Min}, \text{color} = -1$)

- Kinder : 8, 2
- Negamax = $\max(-8, -2) = -2$
↳ Wert zurück an A = $-(-2) = 2$

Für Knoten D ($\text{Min}, \text{color} = -1$)

- Kinder : 4, 6
- Negamax = $\max(-4, -6) = -4$
↳ Wert zurück an A = $-(-4) = \underline{\underline{4}}$

Games 04 : Suchtiefe begrenzen

Da bei einer beschränkten Suchtiefe der Minimax-Algorithmus oft keine Endzustände mehr erreicht, wird eine Evaluierungsfunktion benötigt, um Zwischenzustände sinnvoll bewerten zu können.

Für das Spiel Tic-Tac-Toe kann man so zum Beispiel die Funktion:

$$\text{Eval}(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$$

verwenden.

Dabei beschreibt X_n die Anzahl der Reihen, Spalten oder Diagonalen mit $n X$ und keinem O .

O_n beschreibt dann die Anzahl der Reihen, Spalten oder Diagonalen mit $n O$ und keinem X .

Die Funktion bewertet also wie gut eine Stellung für X oder O ist.

1. X gewinnt

X	X	X
○	○	

$$x_3 = 1$$

$$o_3 = 0$$

$$\text{Eval} = 1'$$

2. O gewinnt

X		X
○	○	○

$$x_3 = 0$$

$$o_3 = 1$$

$$\text{Eval} = -1$$

3. Unentschieden

X	○	X
X	○	○
○	X	X

$$x_3 = 0$$

$$o_3 = 0$$

$$\text{Eval} = 0$$

4. Frühes Spiel

X		
0		

$$x_1 = 3$$

$$0_1 = 2$$

$$\text{Eval} = 3 \times 0 + 3 - (3 \times 0 + 2) = 1$$

X → Vorteil

5. Mittleres Spiel

0	0	
x	x	

$$x_2 = 1$$

$$0_2 = 1$$

$$\text{Eval} = 3 \times 1 + 0 - (3 \times 1 + 0) = 0$$

Ausgeglichen

6. 0 kurz vor dem Sieg

0	0	
x	x	

$$x_1 = 2$$

$$0_2 = 1$$

$$\text{Eval} = 3 \times 0 + 2 - (3 \times 1 + 0) = -1$$

0 → Vorteil

Games 05: Minimax generalisiert

