Blatt_3: Minimax und Alpha Beta Suche

Wilfried Namo

Aufgabe 1)

1) Minimax Werte

$$
\begin{aligned}
B &= \min(8, 7, 3) = 3 \\
E &= \max(9, 1, 6) = 9 \\
F &= \max(2, 1, 1) = 2 \\
G &= \max(6, 5, 2) = 6 \\
C &= \min(E, F, G) = \min(9, 2, 6) = 2 \\
D &= \min(2, 1, 3) = 1 \\
A &= \max(B, C, D) = \max(3, 2, 1) = 3
\end{aligned}
$$

2) Alpha Beta Suche mit Intervallen und Schnitten

Start an der Wurzel A (MAX) mit Intervall $[\alpha, \beta] = [-\infty, +\infty]$.

Schrittfolge

1) **Knoten B (MIN)** hat $[-\infty, +\infty]$.
Blätter $8, 7, 3 \Rightarrow B = 3$.
Zurück zu A: $\alpha \leftarrow \max(-\infty, 3) = 3$.
$A:\ [3, +\infty]$.

2) **Knoten C (MIN)** hat $[3, +\infty]$.

   - **Knoten E (MAX)** hat $[3, +\infty]$.
     $E = 9$. Update bei C: $\beta \leftarrow \min(+\infty, 9) = 9$.
     $C:\ [3, 9]$.
   - **Knoten F (MAX)** hat $[3, 9]$.
     $F = 2$. Update bei C: $\beta \leftarrow \min(9, 2) = 2$.
     $C:\ [3, 2]$. Da $\beta \leq \alpha$ gilt, erfolgt ein Schnitt.
     Der verbleibende Teilbaum G wird nicht mehr untersucht.
     C gibt 2 zurück.
     Zurück zu A: Intervall bleibt $[3, +\infty]$.

3) **Knoten D (MIN)** hat $[3, +\infty]$.
Erstes Blatt liefert 2, also $\beta \leftarrow 2$.
Da $\beta \leq \alpha$ gilt, erfolgt ein Schnitt.
Die zwei übrigen Blätter unter D werden nicht betrachtet.
D gibt 2 zurück.
An A: $\alpha \leftarrow \max(3, 2) = 3$.
A final: $[3, +\infty]$.

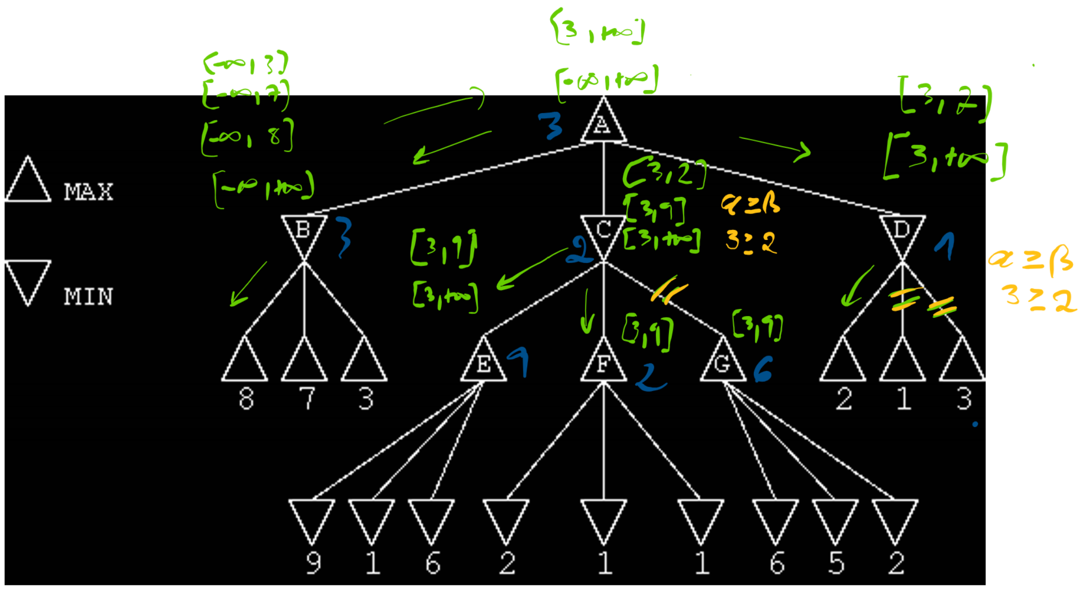
Abgeschnittene Kanten

- $C \to G$ und der gesamte Teilbaum unter G.
- Die zwei letzten Blätter unter D.

3) Reihenfolge zur Maximierung der Schnitte

Bei C zuerst F auswerten. Dann liefert C sofort den Wert 2 und schneidet die übrigen Kinder E und G ab.
Bei D bringt eine andere Reihenfolge keinen Zusatzgewinn, da bereits das erste Blatt zum Schnitt führt.

Endwert an der Wurzel $A = 3$.



Aufgabe 2)

1. Minimax Implementierung

Listing 1: Minimax

```python
from math import inf

def minimax(state, player):
    term, u = terminal(state)
    if term:
        return u, None
    best, best_move = (-inf, None) if player == 'X' else (inf, None)
    for m in legal_moves(state):
        s2 = play(state, m, player)
        v, _ = minimax(s2, 'O' if player == 'X' else 'X')
        if (player == 'X' and v > best) or (player == 'O' and v < best):
            best, best_move = v, m
    return best, best_move
```

2. Alpha Beta Pruning ergänzung

Listing 2: Aplha-Beta-Prunning

```
from math import inf as INF

def alphabeta(state, player, alpha=-INF, beta=INF):
    term, u = terminal(state)
    if term:
        return u, None

    best_move = None
    if player == 'X':  # MAX
        best = -INF
        for m in legal_moves(state):
            s2 = play(state, m, player)
            v, _ = alphabeta(s2, 'O', alpha, beta)
            if v > best:
                best, best_move = v, m
            if best > alpha:
                alpha = best
            if alpha >= beta:  # beta-cut
                break
        return best, best_move
    else:                 # MIN
        best = INF
        for m in legal_moves(state):
            s2 = play(state, m, player)
            v, _ = alphabeta(s2, 'X', alpha, beta)
            if v < best:
                best, best_move = v, m
            if best < beta:
                beta = best
            if alpha >= beta:  # alpha-cut
                break
        return best, best_move
```

3. Vergleich der berechneten Knoten

Für den Knotenvergleich verwenden wir Tic Tac Toe mit X als MAX, O als MIN und Terminalbewertung Sieg $X = +1$, Sieg $O = -1$, Remis $= 0$. In einer typischen Mittelspielstellung A mit X am Zug (Brett: $X\,.\,O/.\,X\,./.\,.\,O$) ergeben Zähler-Wrapper für die rekursiven Funktionen: Minimax expandiert 186 Knoten und wählt Feld 5 bei Wert 0; Alpha–Beta erreicht denselben Wert 0 mit demselben besten Zug, expandiert aber nur 88 Knoten und erzeugt 26 Schnitte. In einer zweiten Mittelspielstellung B mit O am Zug (Brett: $X\,O\,X/X\,O\,./\ldots$) liefert Minimax 38 Knoten und besten Zug 7 bei Wert -1, während Alpha–Beta beim Wert -1 und besten Zug 7 bleibt, jedoch auf 28 Knoten reduziert und 7 Schnitte verzeichnet. Damit zeigt sich: Alpha–Beta hält den Minimax-Wert konstant, verringert aber die Anzahl der berechneten Knoten deutlich; der Nutzen des Prunings ist bereits in kleinen Tic-Tac-Toe-Szenarien klar sichtbar.

Aufgabe 3)

1. Vereinfachter Minimax-Algorithmus

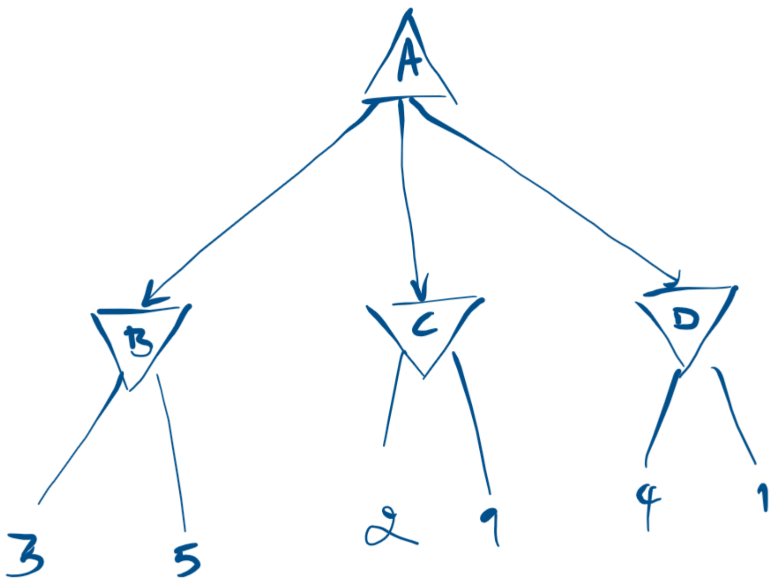
Listing 3: vereinfachter Minimax

```

# sign = +1 pour den Spieler am Zug (MAX-Sicht),
# sign = -1 pour den Gegenspieler
def Value(state, sign):
    if Terminal-Test(state):
        return sign * Utility(state)

    v = -INF
    for (a, s) in Successors(state):
        v = MAX(v, -Value(s, -sign))
    return v
```

2. Beispielbaum



Minimax Schritt für Schritt

$$B = \min(3,5) = 3$$
$$C = \min(2,9) = 2$$
$$D = \min(4,1) = 1$$
$$A = \max(B,C,D) = \max(3,2,1) = 3$$

Ergebnis an der Wurzel: 3, bester Zug von A geht zu B.

Vereinfachter Algorithmus (eine Funktion $\mathrm{Value}(state, sign)$**)** Definition im Terminal: $\mathrm{Value}(state, sign) = sign \cdot \mathrm{Utility}(state)$. Rekurrenz: $\mathrm{Value}(state, sign) = \max_{(a,s')\in \mathrm{Successors}(state)} \big(-\mathrm{Value}(s', -$
Start an der Wurzel mit $sign = +1$.

- Kind B mit $sign = -1$: Blätter liefern -3 und -5, Maximum ist -3. Kandidat an A: $-(-3)) = 3$.
- Kind C mit $sign = -1$: Blätter -2 und -9, Maximum -2. Kandidat: $-(-2)) = 2$.
- Kind D mit $sign = -1$: Blätter -4 und -1, Maximum -1. Kandidat: $-(-1)) = 1$.
- Wurzel A: $\max\{3, 2, 1\} = 3$.

Beide Verfahren liefern an der Wurzel denselben Wert 3 und dieselbe Entscheidung zugunsten des Zuges nach B. Die vereinfachte Variante benötigt nur eine rekursive Funktion und nutzt das Vorzeichen zum Spielerwechsel.

Aufgabe 4)

Bewertungsfunktion

Für Tic Tac Toe seien X_n die Anzahl der Reihen, Spalten oder Diagonalen mit genau n X und keinem O. Analog ist O_n definiert.

- Terminale Nutzenfunktion: $+1$ falls $X_3 = 1$, -1 falls $O_3 = 1$, sonst 0.
- Nicht terminale Bewertung:

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

Drei Endzustände

Zustand	Beschreibung	Utility
T1	X gewinnt in der oberen Reihe	$+1$
T2	O gewinnt auf der Hauptdiagonale	-1
T3	Vollständiges Brett ohne Dreier (Remis)	0

Drei Zwischenzustände

Koordinaten sind (Zeile, Spalte) mit Werten in $\{1, 2, 3\}$. Gezählt wird über alle acht Gewinnlinien (drei Reihen, drei Spalten, zwei Diagonalen).

Zustand	Brettbeschreibung	X_2	X_1	O_2	O_1	Eval
Z1	X auf $(1, 1)$, O auf $(2, 2)$	0	2	0	3	-1
Z2	X auf $(1, 1)$, $(1, 2)$, O auf $(2, 2)$	1	1	0	2	$+2$
Z3	X auf $(1, 1)$, $(3, 2)$, O auf $(2, 1)$, $(2, 2)$	0	2	1	1	-2

Begründung der Sinnhaftigkeit

- Die Funktion spiegelt die Nullsummenstruktur wider, da Beiträge von X und O symmetrisch gegeneinander verrechnet werden.
- Unmittelbare Drohungen werden priorisiert, weil Linien mit zwei gleichen Steinen ohne Gegenspieler dreifach gewichtet werden.

- Die lineare Form ist rechnerisch sehr günstig und daher gut geeignet für Suchtiefenbegrenzung mit vielen zu bewertenden Knoten.
- Invarianz gegenüber Brettsymmetrien, da nur die Anzahl der Kandidatenlinien zählt und nicht deren Lage.
- Gute Muster für X erhöhen den Wert, gute Muster für O senken ihn. Das korreliert mit den realen Gewinnchancen.

Aufgabe 5)