Praktikum 03- Grundlagen der KI

Datum: 02.11.2025

Name: Umar Farooq

Games.01: Handsimulation: Minimax und alpha.beta-Pruning

1) Minimax- Bewertung

- 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

2) Alpha- Beta- Pruning

Wurzel A (MAX) startet mit α = −∞, β = +∞

a. Knoten B (MIN) erbt α = −∞, β = +∞
   i. 8 gesehen → β = 8
   ii. 7 gesehen → β = 7
   iii. 3 gesehen → β = 3, B liefert 3

b. zurück bei A → α = max(−∞, 3) = 3

c. Knoten C (MIN) erbt α = 3, β = +∞
   i. E (MAX) mit α = 3, β = +∞
      1. 9 gesehen → α = 9, E liefert 9
   ii. zurück bei C → β = min(+∞, 9) = 9
   iii. F (MAX) mit α = 3, β = 9
      1. 2 gesehen → α bleibt 3, F liefert 2
   iv. zurück bei C → β = min(9, 2) = 2
   v. Prüfe β ≤ α → 2 ≤ 3, daher **Pruning** von G und allen Blättern unter G
   vi. C liefert 2

d. zurück bei A → α bleibt 3

e. Knoten D (MIN) erbt α = 3, β = +∞
   i. erstes Blatt 2 → β = 2
   ii. β ≤ α → 2 ≤ 3, daher **Pruning** der restlichen beiden Blätter von D
   iii. D liefert 2

A wählt max(3, 2, 1) = 3

Geschnittene Kanten:

- C → G und alle drei Blätter unter G
- D → die letzten beiden Blätter (nach dem ersten Wert 2)

3) Bessere Ordnung für Pruning

Die Knoten können derart geordnet werden. Ordnungen müssen für MAX von gut nach schlecht und für MIN von klein nach groß sein. Beispiel unter C: F zuerst, dann sofort β = 2 und wegen α = 3 wird E und G komplett abgeschnitten. Eine mögliche Reihenfolge:

- Bei C: F, danach E, dann G
- Bei D: das Blatt mit Wert 1 zuerst, dann werden die zwei übrigen Blätter abgeschnitten

Games.02: Optimale Spiele: Minimax und alpha-Beta-Pruning

Die Implementierung finden sie unter der datei: Tic tac toe.py

Vergleich für die Knotenanzahl
Szenario 1: Startstellung. Reines Minimax durchsucht den gesamten Baum einer perfekten Partie. Alpha-Beta mit der obigen Reihenfolge (Zentrum zuerst, danach Ecken) reduziert die Zahl der besuchten Knoten sehr deutlich.
Szenario 2: Eine fast volle Stellung, in der ein sofortiger Gewinn existiert. Alpha-Beta findet den Gewinn oft nach sehr wenigen Knoten, während Minimax alle verbleibenden Züge gleichrangig betrachtet.

Games.03 - Minimax vereinfachen

Negamax mit Nullsummenannahme lassen sich in eine Funktion integrieren mit max und min.

```
def negamax(s, color): # color = +1 (MAX) oder –1 (MIN)
If terminal(s): return color * utility(s)
best = -inf
for a in actions(s):
best = max(best, -negamax(result(s, a), - color))
return best
```

Beispielbaum Tiefe 2. Blätter (aus MAX-Sicht): (+3,+5) links, (+2, -1)
Rechts unter MIN.
Minimax: max(min(3,5), min(2, -1)) = max(3, -1) = 3
Negamax liefert denselben Wert

Games.05: Minimax generalisiert

Hier wird das Max-n-Verfahren verwendet. Jeder innere Knoten wählt das Kind, das die Komponente des aktuellen Spielers maximiert. Die Blätter sind gegeben als Tripel (x1, x2, x3).

Von links nach rechts ausgewertet:

- Linker Teilbaum
  - Spieler 3 vergleicht (1,2,3) mit (4,2,1) → wählt (1,2,3)
  - Spieler 3 vergleicht (6,1,2) mit (7,4,−1) → wählt (6,1,2)
  - Spieler 2 vergleicht diese beiden → vergleicht zweite Komponente 2 vs 1 → wählt (1,2,3)
- Rechter Teilbaum
  - Spieler 3 vergleicht (5,−1,−1) mit (−1,5,2) → wählt (−1,5,2)
  - Spieler 3 vergleicht (7,7,−1) mit (5,4,5) → wählt (5,4,5)
  - Spieler 2 vergleicht zweite Komponente 5 vs 4 → wählt (−1,5,2)
- Wurzel bei Spieler 1 vergleicht erste Komponente 1 vs −1 → wählt **(1,2,3)**

Annotationen der inneren Knoten und der Wurzel:

- Linker Kindknoten der Wurzel: (1,2,3)
- Rechter Kindknoten der Wurzel: (−1,5,2)
- Wurzel: (1,2,3)

Games.04: Suchtiefe begrenzen

Beispielswerte (6 Zustände)
Drei Endzustände

X gewinnt = +1
0 gewinnt = -1
Unentschieden = 0

Drei Zwischenzustände

Leicht besser für X

- X2=0
- X1=5 (R1, R2, C1, C2, Anti-Diagonale)
- O2=0
- O1=2 (R3, C3)
  Eval = 3·0 + 5 − (3·0 + 2) = **3**

Deutlicher Vorteil für X

- X2=1 (R1)
- X1=1 (C1)
- O2=0
- O1=2 (R2, Anti-Diagonale)
  Eval = 3·1 + 1 − (0 + 2) = **2**

Ausgeglichen trotz Drohung von O

- X2=0
- X1=4 (R1, R2, C3, Hauptdiagonale)
- O2=1 (R3)
- O1=1 (C1)

Eval = 0 + 4 − (3·1 + 1) = **0**

Diese Evaluierungsfunktion ist sinnvoll, weil sie

- **Gewinnchancen** und **Verteidigungsnotwendigkeiten** richtig priorisiert,
- **schnell berechenbar** ist,
- **symmetrisch und konsistent** zwischen beiden Spielern funktioniert,
- und **qualitativ gute Schätzwerte** liefert, wenn man den Baum nicht vollständig durchsuchen kann.