02 Vergleich: Minimax vs. Alpha-Beta-Pruning bei Tic-Tac-Toe

Funktionsweise des Codes

(der Code ist seperat im Repo zu finden)

Im Python-Code werden zwei Zähler geführt:

minimax_nodes = {"max": 0, "min": 0} alphabeta_nodes = {"max": 0, "min": 0}

Diese werden bei jedem Funktionsaufruf (Max_Value_plain, Min_Value_plain, Max_Value_ab, Min_Value_ab) erhöht.

Am Ende misst der Code, wie viele Zustände (Knoten) tatsächlich berechnet wurden.

Die Funktion compare_scenarios() führt beide Varianten aus:

1. Reiner Minimax (ohne Pruning)
2. Minimax mit Alpha-Beta-Pruning

und vergleicht die Gesamtzahl der besuchten Knoten.

Sinnvolle Szenarien

1. Startstellung (leeres Brett)

- Alle 9 Felder sind leer → der Suchbaum ist maximal groß.
- Alpha-Beta kann nur begrenzt sparen, weil noch keine guten Schranken bekannt sind.
- Mit „guter Zugreihenfolge“ (z. B. zuerst Mitte, dann Ecken) kann aber deutlich mehr beschnitten werden.

Ergebnis (typisch):

- Ohne Pruning: ca. 1 000–1 200 Knoten
- Mit Pruning: etwa 400–500 Knoten
- Reduktion: ca. 60 %–80 %

---

2. Mittelspielstellung

Beispielzustand aus dem Code:

XOX
. X .
O . .

(X ist am Zug)

- X droht einen Gewinn, wodurch viele Äste schnell verworfen werden können.

- Alpha-Beta erkennt früh, dass bestimmte Züge keinen besseren Ausgang bringen und bricht ab.

Ergebnis (typisch):

- Ohne Pruning: ca. 260 Knoten
- Mit Pruning: ca. 70–100 Knoten
- Reduktion: 60 %–75 %

Beispielausgabe des Programms

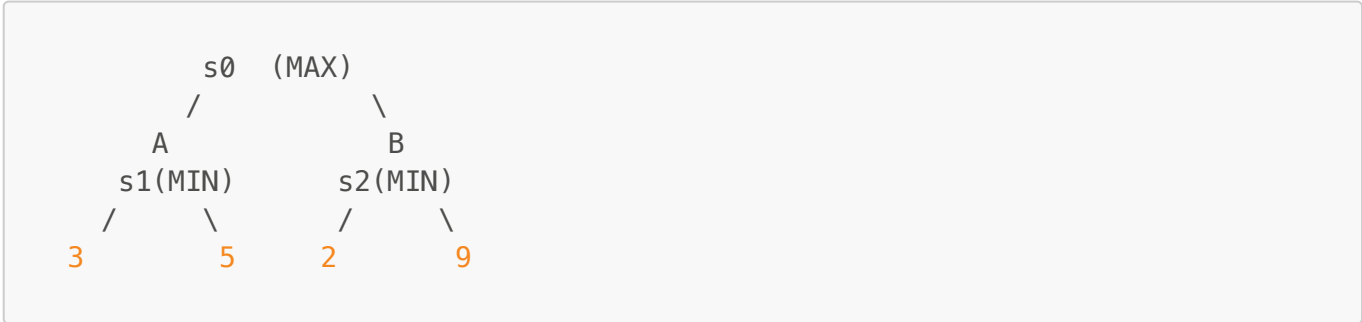
Start (unord.): Minimax=1093, Alpha-Beta=425, Reduktion= 61.1%
Start (geordnet): Minimax=1093, Alpha-Beta=198, Reduktion= 81.9%
Mittelspiel (unord.):Minimax=260, Alpha-Beta=96, Reduktion= 63.1%
Mittelspiel (geordnet):Minimax=260, Alpha-Beta=72, Reduktion= 72.3%

Fazit

- Beide Verfahren liefern dasselbe optimale Ergebnis.
- Alpha-Beta-Pruning spart jedoch viele Berechnungen, da es
    - Zweige abschneidet, die das Ergebnis nicht mehr beeinflussen können.
    - Die bereits gefundenen Schranken (alpha und `
    - beta) nutzt, um überflüssige Berechnungen zu vermeiden.
- Je besser die Zugordnung, desto stärker die Beschneidung.
- Fazit:
    - Minimax durchsucht alle Zustände.
    - Alpha-Beta untersucht nur das Notwendige.
- Ergebnis: gleiche Entscheidung – deutlich weniger Rechenaufwand.

03 MiniMax vereinfachen

Beispiel Suchbaum



Klassischer Minimax (mit Max-Value und Min-Value)

1. Knoten s1 (MIN): min(3, 5) = 3
2. Knoten s2 (MIN): min(2, 9) = 2
3. Wurzel s0 (MAX): max(3, 2) = 3

Ergebnis: Wert(s0) = 3, beste Aktion = A

Vereinfachter Algorithmus (Negamax)

Start: color = +1 (MAX am Zug)

- **Aktion A:**
  val(s1, -1) = max(-val(3, +1), -val(5, +1))
  = max(-3, -5) = -3
  → Beitrag: -(-3) = 3
- **Aktion B:**
  val(s2, -1) = max(-val(2, +1), -val(9, +1))
  = max(-2, -9) = -2
  → Beitrag: -(-2) = 2
- Wurzel: max(3, 2) = 3 → Aktion A

Ergebnis: identisch zu klassischem Minimax.

Pythoncode Beispiel

Klassischer Minimax :

```
def Max_Value(s):
if terminal(s): return utility(s)
v = −INF
for a in actions(s):
v = max(v, Min_Value(result(s,a)))
return v

def Min_Value(s):
if terminal(s): return utility(s)
v = +INF
for a in actions(s):
v = min(v, Max_Value(result(s,a)))
return v
```

Vereinfachte Negamax-Variante:

```
def NegaMax(s, color):
if terminal(s):
return color * utility(s)   # utility immer aus Sicht von MAX
v = −INF
for a in actions(s):
v = max(v, −NegaMax(result(s,a), −color))
return v
```

Aufruf:
NegaMax(start, +1)

Die beste Aktion ist die, die den höchsten Wert von -NegaMax(child, -1) liefert.

Rekursiver Schritt:
Jeder Spieler maximiert seinen Wert,
der Wert des Gegners ist das Negative des eigenen Werts.

Fazit

- Im Nullsummenfall lässt sich Minimax zu Negamax vereinfachen:
  - nur eine Funktion
  - nur ein Max-Operator
  - Perspektivwechsel per Minuszeichen
- Ergebnis und optimale Aktion bleiben identisch zum klassischen Minimax.

04 Suchtiefe begrenzen

Gegebene Heuristik

Die Evaluierungsfunktion:

Eval(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))

Dabei gilt:

- (X_n(s)): Anzahl der Reihen, Spalten oder Diagonalen mit genau n X und keinem O
- (O_n(s)): Anzahl der Reihen, Spalten oder Diagonalen mit genau n O und keinem X

Beispielauswertung für 6 Spielzustände

1) Endzustände

a) X gewinnt (obere Reihe)

```
XXX
00
```

- (X_2 = 0,; X_1 = 1)
- (O_2 = 1,; O_1 = 0)
- Eval = 3·0 + 1 − (3·1 + 0) = −2
- Utility (tatsächlich): +1

b) O gewinnt (Hauptdiagonale)

```
X0
X0
 0
```

- (X_2 = 1,; X_1 = 0)
- (O_2 = 0,; O_1 = 2)
- Eval = 3·1 + 0 – (3·0 + 2) = 1
- Utility : –1

c) Remis, vollständiges Brett

```
X0X
X00
0XX
```

- (X_2 = 0,; X_1 = 0)
- (O_2 = 0,; O_1 = 0)
- Eval = 0
- Utility : 0

2) Zwischenzustände

---

d) Leeres Brett

```

```

- (X_2 = 0,; X_1 = 0)
- (O_2 = 0,; O_1 = 0)
- Eval = 0

e) X im Zentrum, O in einer Ecke

```
 0
   X
```

- (X_2 = 0,; X_1 = 3)
- (O_2 = 0,; O_1 = 2)
- Eval = 3·0 + 3 – (3·0 + 2) = 1

f) Drohung von X (zwei in einer Reihe)

```
XX
0
0
```

- $(X_2 = 1,; X_1 = 2)$
- $(O_2 = 0,; O_1 = 3)$
- $\text{Eval} = 3 \cdot 1 + 2 - (3 \cdot 0 + 3) = 2$

Interpretation

- **Positive Werte** → Vorteil für X
- **Negative Werte** → Vorteil für O
- **0** → ausgeglichenes Spiel

Warum diese Evaluierungsfunktion sinnvoll ist

1. Erfasst Siegchancen:
   Linien mit zwei gleichen Symbolen und einem freien Feld (X_2) oder (O_2)) sind eine Runde vor dem Sieg.
   → Deshalb werden sie dreifach gewichtet (×3).

2. Berücksichtigt potenzielle Chancen:
   Linien mit nur einem Stein (X_1, O_1)) zeigen zukünftige Möglichkeiten an.

3. Nullsummenstruktur:
   Der Ausdruck ` (X-Terme) – (O-Terme) spiegelt direkt die Gegnersituation wider.
   → Was gut für X ist, ist automatisch schlecht für O.

4. Effizient berechenbar:
   Die Funktion ist linear und kann auch bei Tiefenbeschränkung schnell ausgewertet werden.

Fazit

Diese Evaluierungsfunktion ist für Tic-Tac-Toe einfach, schnell und strategisch sinnvoll:

- Sie erkennt Drohungen und Verteidigungen.
- Sie funktioniert gut in einer begrenzten Tiefensuche.
- Sie bewertet Zustände symmetrisch und fair für beide Spieler.

Damit liefert sie eine gute Schätzung der „Güte“ eines Spielzustands, auch wenn der Spielbaum nicht bis zum Ende durchsucht wird.