

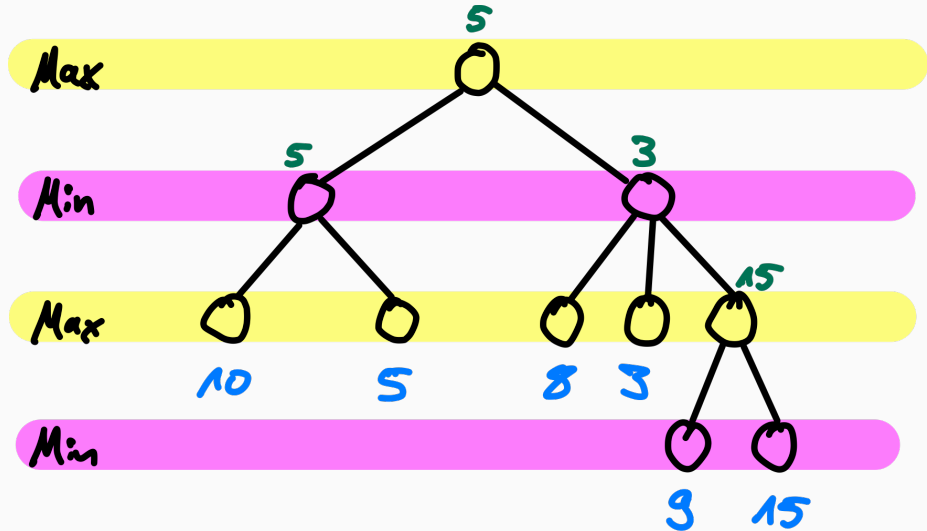
# Alpha-Beta-Pruning

---

Carsten Gips (HSBI)

Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

# Verbesserung Minimax-Algorithmus



=> Minimax-Baum: Verbesserungen möglich?

# Alpha-beta-Pruning

Minimax-Algorithmus mit zusätzlichen Informationen:

- $\alpha$ : bisher bester Wert für MAX (höchster Wert)
- $\beta$ : bisher bester Wert für MIN (kleinster Wert)

# Alpha-beta-Pruning

Minimax-Algorithmus mit zusätzlichen Informationen:

- $\alpha$ : bisher bester Wert für MAX (höchster Wert)
- $\beta$ : bisher bester Wert für MIN (kleinster Wert)

=> Beobachtungen:

1.  $\alpha$  für MAX-Knoten wird nie kleiner
2.  $\beta$  für MIN-Knoten wird nie größer

Tafelbeispiel: Beste Werte einzeichnen

# Pruning-Regeln

1. Schneide (unter) MIN-Knoten ab, deren  $\beta \leq$  dem  $\alpha$  des MAX-Vorgängers ist.
2. Schneide (unter) MAX-Knoten ab, deren  $\alpha \geq$  dem  $\beta$  des MIN-Vorgängers ist.

Abbruch, wenn kein Platz  
mehr zwischen Alpha und Beta

# Alpha-beta-Pruning – Der Algorithmus (Skizze)

```
def Max-Value(state, alpha, beta):  
    if Terminal-Test(state): return Utility(state)  
  
    v = -INF  
    for (a, s) in Successors(state):  
        v = MAX(v, Min-Value(s, alpha, beta))  
        if (v >= beta): return v  
        alpha = MAX(alpha, v)  
    return v
```

Initialer Aufruf von `Max-Value()` mit  $\alpha = -\infty$  und  $\beta = +\infty$

# Alpha-beta-Pruning – Eigenschaften

1. Pruning beeinflusst nicht das Endergebnis!
2. Sortierung der Nachfolger spielt große Rolle
3. Perfekte Sortierung:  $O(b^{d/2}) \Rightarrow$  Verdopplung der Suchtiefe möglich

Für Schach immer noch zu aufwändig ...

# Verbesserungen für Alpha-beta-Pruning

- “Killer-Move”: Maximale Effizienz nur wenn **optimaler Zug immer zuerst**  
=> Zu untersuchende Züge **sortieren/priorisieren**, zb. Schach:
  - a) Figuren schlagen
  - b) Drohen
  - c) Vorwärts ziehen
  - d) Rückwärts ziehen
- Verändern der Suchtiefe nach Spielsituation
- Bewertungsfunktion **Eval**:
  - Datenbanken mit Spielsituationen und Expertenbewertung:
    - Eröffnungsspiele (besonders viele Verzweigungen)
    - Endspiele
  - Lernen der optimalen Gewichte für **Eval**-Funktion
  - Berücksichtigung von Symmetrien



# Beispiel DeepBlue (IBM, 1997)

- Alpha-beta-Pruning mit Tiefenbeschränkung: ca. 14 Halbzüge
- Dynamische Tiefenbeschränkung (stellungsabhängig, max. ca. 40 Züge)
- Heuristische Stellungsbewertung **Eval**:
  - mehr als 8.000 Features
  - ca. 4.000 Eröffnungsstellungen
  - ca. 700.000 Spielsituationen (von Experten bewertet)
  - Endspiel-Datenbank: alle Spiele mit 5 Steinen, viele mit 6 Steinen

Quelle: (Russell und Norvig 2014, p. 185)

# Beispiel AlphaGo (Google, 2016)

- Beschränkung der Suchtiefe: Bewertung der Stellung durch *“Value Network”*
- Beschränkung der Verzweigungsbreite: Bestimmung von Zugkandidaten durch *“Policy Network”*
- Training dieser *“Deep Neural Networks”*:
  - Überwachtes Lernen: “Analyse” von Spiel-Datenbanken
  - Reinforcement-Lernen: Self-Play, Bewertung am Ende
    - Züge mit Monte-Carlo-Baumsuche ausgewählt

Quelle: (Silver u. a. 2016), siehe auch [deepmind.com/research/alphago/](https://deepmind.com/research/alphago/)

- Alpha-beta-Pruning:
  - Mitführen der bisher besten Werte für MAX und MIN:  $\alpha$  und  $\beta$
  - Abschneiden von Pfaden, die Verschlechterung bewirken würden
  - Endergebnis bleibt erhalten
  - Effizienzsteigerung abhängig von Sortierung der Nachfolger
- Viele Verbesserungen denkbar:
  - Zu untersuchende Züge “richtig” sortieren (Heuristik)
  - Suchtiefe begrenzen und Bewertungsfunktion (statt Nutzenfunktion)
  - Positionen mit Datenbank abgleichen



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.