

Grundlagen der Künstlichen Intelligenz

5 Spiele = Adversiale Suche

Suchstrategien für Spiele, Spiele mit Zufall, Stand der Kunst

Volker Steinhage

Inhalt

- Brettspiele
- Minimax-Suche
- Alpha-Beta-Suche
- Spiele mit Zufallsereignissen
- Stand der Kunst

Spiele?

Kapitel 2 stellte *Multiagenten-Umgebungen* vor, in denen mehrere Agenten ihre Wechselwirkungen berücksichtigen müssen. *Spiele* sind *konkurrierende* Multiagenten-Umgebungen. Insbes. Brettspiele sind eines der ältesten Teilgebiete der KI (Shannon und Turing schon 1950).

Warum?

- Brettspiele sind eine *abstrakte und reine Form des Wettbewerbs* zwischen zwei *Gegnern* (lat. *Adversius*) und erfordern „offensichtlich“ eine Form von Intelligenz.
- Die *Zustände* eines Spiels sind i. A. *relativ einfach darstellbar*.
- Die möglichen *Aktionen* der Spieler *sind wohl definiert*.

-
- Realisierung des Spielens als Suchproblem (sog. *adversiale Suche*)
 - häufig mit *deterministischen* und *vollständig beobachtbare Umgebungen*
 - aber *Kontingenzprobleme*,
weil die Züge des Gegners im voraus nicht bekannt sind.

Bei Backgammon
stochastisch und
vollständig beobachtbar

Probleme

Brettspiele sind herausfordernd, weil es eben **Kontingenzprobleme** sind,
und weil die **Suchbäume sehr groß** werden.

Ein Halbzug ist ein einzelner Zug für jede Seite. Z.B. der Bauernzug von E2 nach E4 ist ein Halbzug

Beispiele:

- Schach: ca. 35 mögliche Aktionen in jeder Position und 100 Halbzüge pro Spiel
→ 35^{100} Knoten im Suchbaum (bei „nur“ ca. 10^{40} legalen Schachpositionen).
- Go: durchschnittlich 200 mögliche Aktionen bei ca. 300 Halbzügen
→ 200^{300} Knoten.
- Dame: vollständiger Suchbaum umfasst ca. 10^{40} Knoten
→ ca. 10^{21} Jahrhunderte für Baumaufbau bei 1/3 ns pro Knotengenerierung.

Gute Spielprogramme zeichnen sich dadurch aus, dass sie

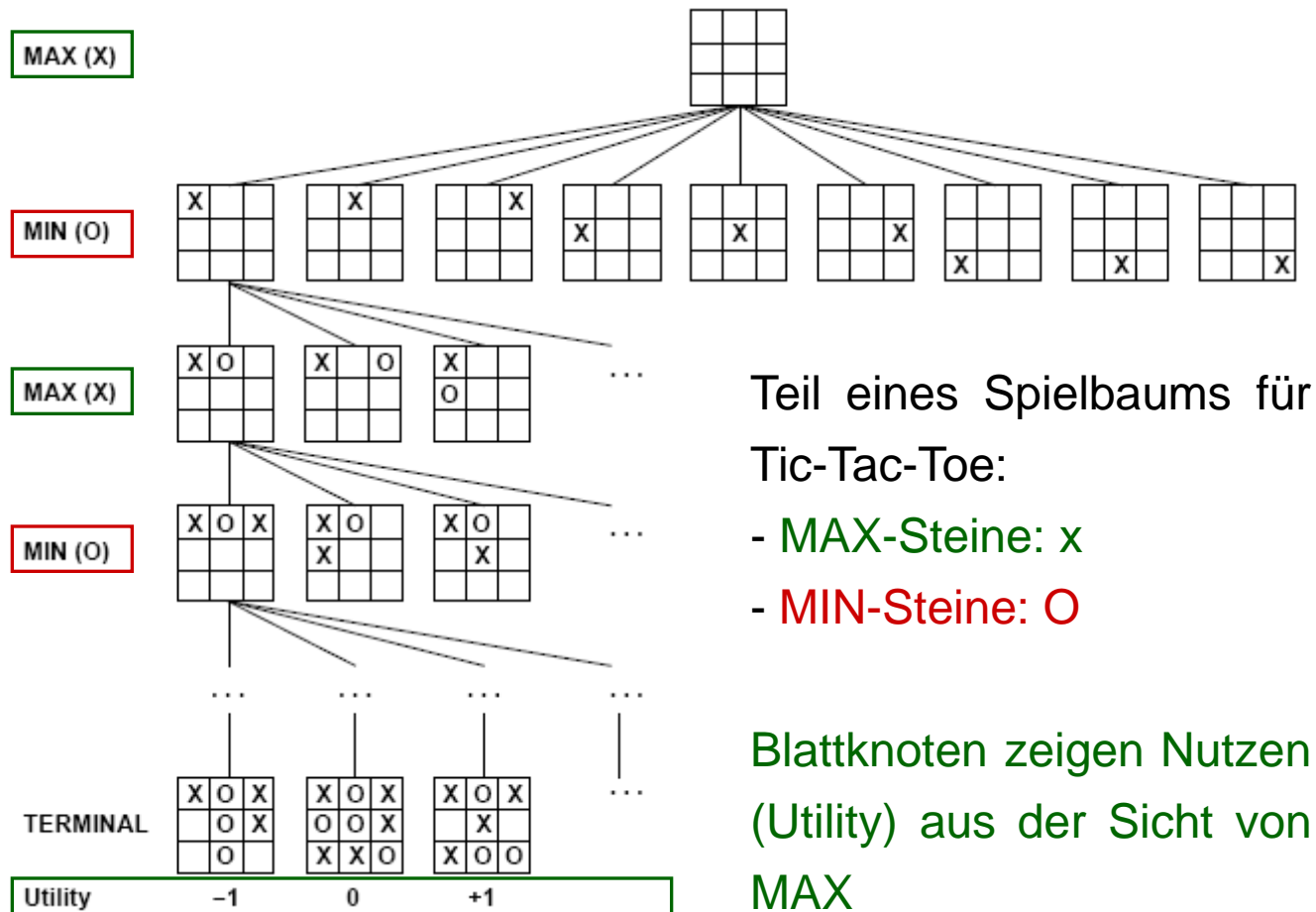
- **irrelevante Äste** im Spielbaum **abschneiden** (engl. *pruning*),
- **gute Evaluierungsfunktion für Zwischenzustände** benutzen und
- möglichst viele Halbzüge **vorausschauen** (engl. *look ahead*).

Terminologie bei Zweipersonen-Brettspielen

- **Spieler:** *MAX* und *MIN*
→ *MAX* steht für das Spielprogramm und beginnt (weil am Zug)!
- **Anfangszustand:** gültige Start- bzw. Ausgangspositionen des jew. Spiels
- **Operatoren** = legale Züge
- **Terminierungstest:** wann ist ein Spiel beendet?
→ **Terminalzustand** = Spielende
- **Nutzenfunktion** = Bewertung des *Spielausgangs*
 - ggf. einfach: +1 (Sieg), -1 (Niederlage), 0 (unentschieden)
 - ggf. differenziert: bei Backgammon Werte zwischen +192 und -192
- **Strategie:**
 - bei regulärer Suche: Lösung = Pfad von Anfangs- zu Zielzustand.
 - hier: *Siegstrategie für MAX* → Ziel: Gewinnmaximierung
→ Strategie, die *unabhängig von MINs Zügen* zum Gewinn führt!
→ korrekte Reaktionen auf *alle Züge von MIN*!

Beispiel Tic-Tac-Toe

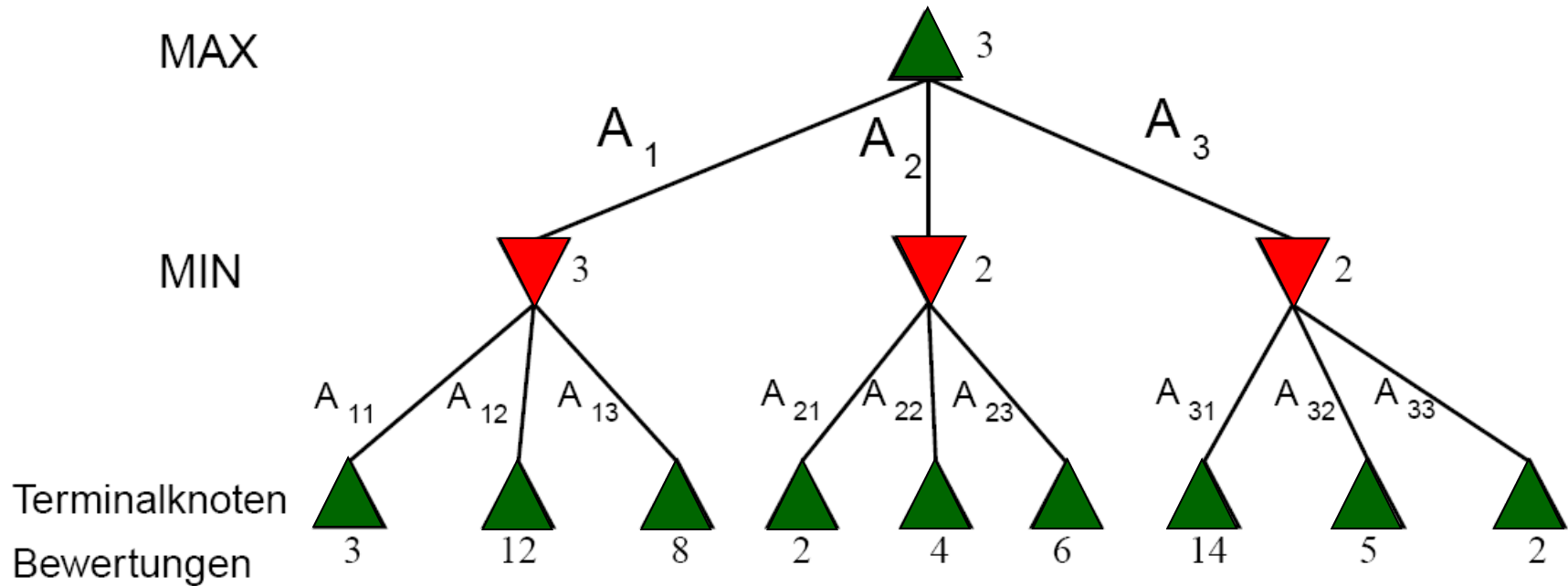
Jede Stufe des Suchbaums, auch **Spielbaum** genannt, wird mit dem Namen des Spielers bezeichnet, der am Zug ist (**MAX**- und **MIN**-Stufen). Wenn es wie hier möglich ist, den gesamten Spielbaum zu erzeugen, liefert der **Minimax-Algorithmus** eine **optimale Strategie für MAX**.



Minimax

1. Erzeuge *vollständigen* Spielbaum mit **Tiefensuche**.
2. Wende die **Nutzenfunktion auf jeden Terminalzustand** an.
3. Beginnend bei Terminalzuständen, **berechne Werte der Vorgängerknoten**:
 - **Knoten ist MIN-Knoten**:
→ Wert übernimmt das Minimum der Nachfolgerknoten.
 - **Knoten ist MAX-Knoten**:
→ Wert übernimmt das Maximum der Nachfolgerknoten.
 - Dann wählt MAX im Anfangszustand (Wurzel des Spielbaums) den Zug, der zu dem Nachfolgerknoten mit größtem berechneten Nutzen führt (Minimax-Entscheidung).
- **Beachte**: Minimax geht von **perfektem Spiel von MIN** aus. Jede Abweichung (d.h. Fehler von MIN) kann das Ergebnis für MAX nur verbessern.

Beispiel für Minimax



Ein Zwei-Schichten-Spielbaum: Die Δ -Knoten sind die MAX-Knoten und die ∇ -Knoten sind die MIN-Knoten. Die Terminalknoten zeigen die Nutzenwerte für MAX. Alle anderen Knoten zeigen ihre MiniMax-Werte. MAX ist am Zug.

Minimax Algorithmus

Berechne rekursiv den besten Zug von der Anfangssituation ausgehend:

```
function MINIMAX-DECISION(state) returns an action
```

```
   $v \leftarrow \text{MAX-VALUE}(\text{state})$ 
```

```
  return the action in SUCCESSORS(state) with value  $v$ 
```

```
function MAX-VALUE(state) returns a utility value
```

```
  if TERMINAL-TEST(state) then return UTILITY(state)
```

```
   $v \leftarrow -\infty$ 
```

```
  for  $a, s$  in SUCCESSORS(state) do
```

```
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$ 
```

```
  return  $v$ 
```

```
function MIN-VALUE(state) returns a utility value
```

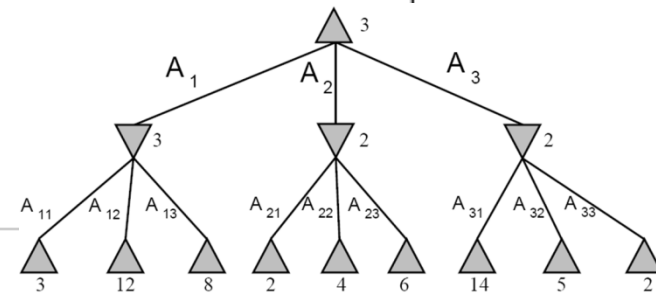
```
  if TERMINAL-TEST(state) then return UTILITY(state)
```

```
   $v \leftarrow \infty$ 
```

```
  for  $a, s$  in SUCCESSORS(state) do
```

```
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$ 
```

```
  return  $v$ 
```



Beachte: Minimax funktioniert so nur, wenn der Spielbaum nicht zu tief ist.
Andernfalls sind *Approximationen* der Minimax-Werte zu bestimmen.

Evaluierungsfunktion

Bei zu **großem Suchraum** kann der Spielbaum nur bis zu einer **maximalen Suchtiefe** erzeugt werden.

Die Kunst besteht dann darin, die **Güte der den Blättern entsprechenden Spielpositionen (i.A. keine Endpositionen!)** korrekt zu bewerten.

Beispiele einfacher Bewertungskriterien im Schach:

- **Material**bewertung: Bauer = 1, Springer = 3, Läufer = 5, Dame = 9, etc.
- **Positions**bewertung: Sicherheit des Königs bis Positionen der Bauern
- **Faustregeln** wie „3-Punkte Vorsprung = (fast) sicherer Sieg“

Die Wahl der Evaluierungsfunktion ist spielentscheidend!

Der Wert einer Spielposition sollte die Gewinnchancen widerspiegeln, d.h. die Gewinnchancen bei einem Vorteil von +1 sollten geringer sein als die bei einem Vorteil von +3.

Evaluierungsfunktion - allgemein

Bevorzugte Evaluierungsfunktionen sind *gewichtete lineare Funktionen*:

$$w_1 \cdot f_1 + w_2 \cdot f_2 + \dots + w_n \cdot f_n.$$

Hierbei ist f_i das *i-te Kriterium* und w_i das *i-te Gewicht*.

→ Bsp: $w_1 = 3$, f_1 = Zahl der eigenen Springer auf dem Brett.

Annahme: Die Kriterien sind unabhängig voneinander.

- Die *Gewichte* können u.U. gelernt werden.
- Die *Kriterien* müssen allerdings vorgegeben werden. Die automatische Auswahl von geeigneten Kriterien ist generell nicht möglich.

Wann die Suche beenden? (1)

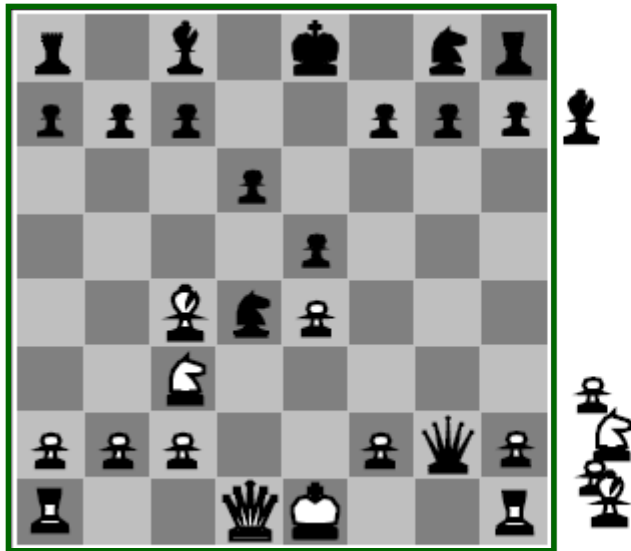
- Einfach: **Tiefenbeschränkte Suche** entsprechend vorgegebenem **Zeitlimit**.
- Robuster: **Iterative Tiefensuche** mit Abbruch beim Erreichen des **Zeitlimits**
→ liefert beste Lösung der tiefsten *vollständigen* Suche.

Bedeutet hier bei großen Spielbäumen i.A.

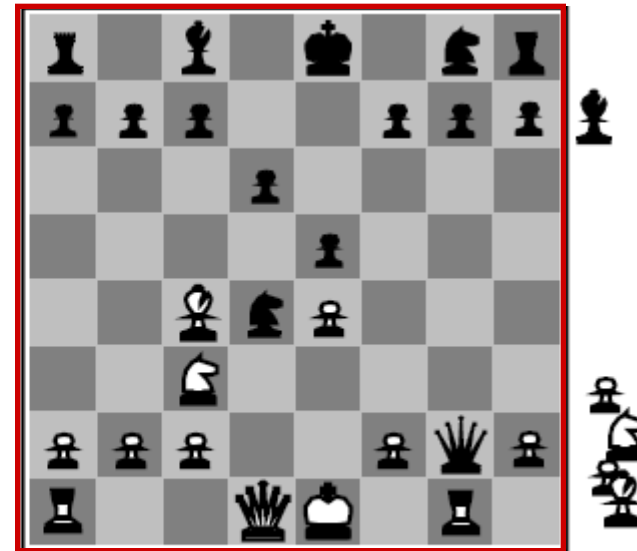
- nicht die optimale Problemlösung bzw. Zielstrategie zum Sieg,
- sondern die *optimale Zwischenlösung* bzw. Strategie zur *besten eigenen Stellung, die bei max. Suchtiefe m erreichbar ist.*

Wann die Suche beenden? (2)

- Evaluierung sollte für „**ruhende**“ Positionen, die nicht zu großen Schwankungen bei der Evaluierung in den folgenden Zügen führen, enden.
 - Annahme: alleinige Bewertung durch Materialvorteil!
- Beide Stellungen (a) und (b) zeigen denselben Materialvorteil für Schwarz (1 Springer, 2 Bauern). Im Ggs. zur „**ruhenden**“ **Position in (a)** gibt es eine „**nicht ruhende**“ **Position in (b)**, da der nächste Zug von Weiß durch den Turm zum unweigerlichen Damenverlust führt.
- u.U. noch etwas weiter suchen und einen Schlagabtausch zu Ende führen
- Extrasuche bis zu „stabiler“ Bewertungsposition.



(a) White to move



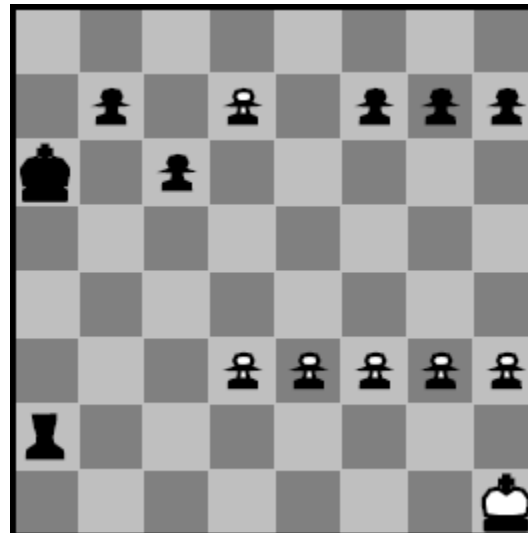
(b) White to move

Wann die Suche beenden? (3)

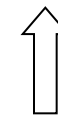
- **Horizonteffekt:** ein letztlich unvermeidbarer schädigender Gegenzug wird durch eigene verzögernde Züge aus dem durch eine Tiefenschranke begrenzten „sichtbaren“ Suchraum hinaus geschoben.



Vermeintliche Lösung: setze mehrmals Schach über Turm und verschiebe den Damenzug über den „Suchhorizont“ hinaus, wo er nicht erkannt wird.



Langfristig wird weißer Bauer zu Dame



Schwarz mit Materialvorteil am Zug

Lösung entweder durch generell größere Suchtiefe (→ Hardware-Erweiterung) oder *singuläre Erweiterung* der Tiefensuche für „deutlich bessere Züge“ (hier die Züge des schwarzen Turms und des weißen Königs).

Komplexität von Minimax \Rightarrow Alpha-Beta-Kürzung

Minimax verwendet Tiefensuche:

\Rightarrow Speicherkomplexität $O(b \cdot m)$ für max. Spielbaumtiefe m und b erlaubte Züge

\Rightarrow Speicherkomplexität $O(m)$ für Backtracking-Variante

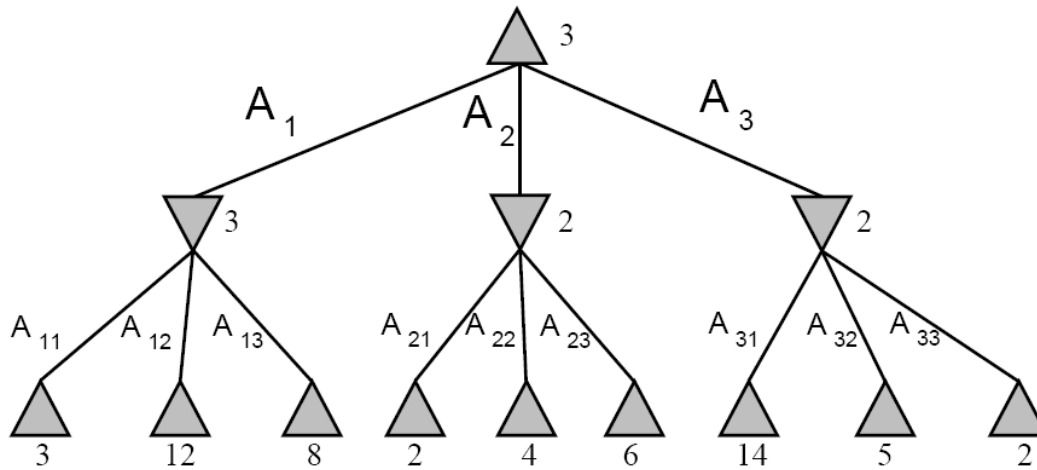
\Rightarrow Zeitkomplexität $O(b^m)$

Für reale Spiele ist die Zeitkomplexität i.A. völlig impraktikabel!

\Rightarrow Reduktion der Laufzeit durch *Alpha-Beta-Kürzung* (engl. *Pruning*) durch Abschneiden von Knoten und Teilbäumen, die die Entscheidung überhaupt nicht beeinflussen können

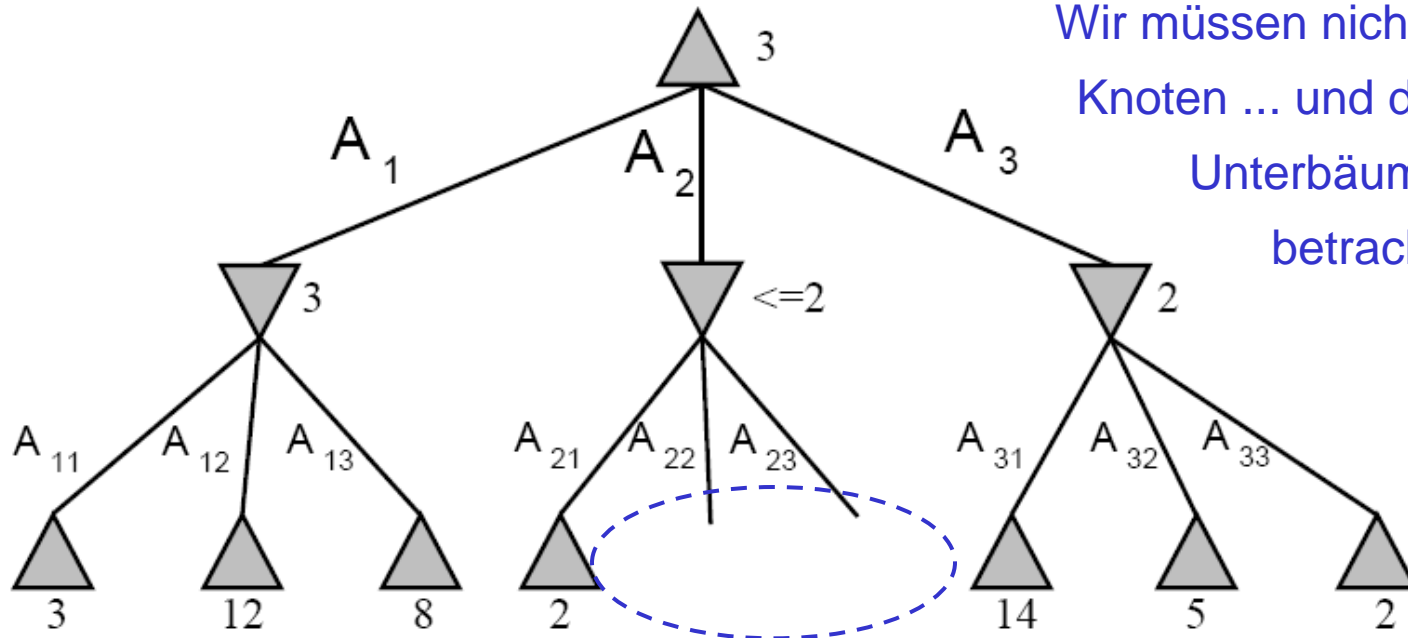
\Rightarrow Qualitativ *dasselbe* Ergebnis wie mit Minimax bei besserer Laufzeit!

Alpha-Beta-Kürzung: Beispiel



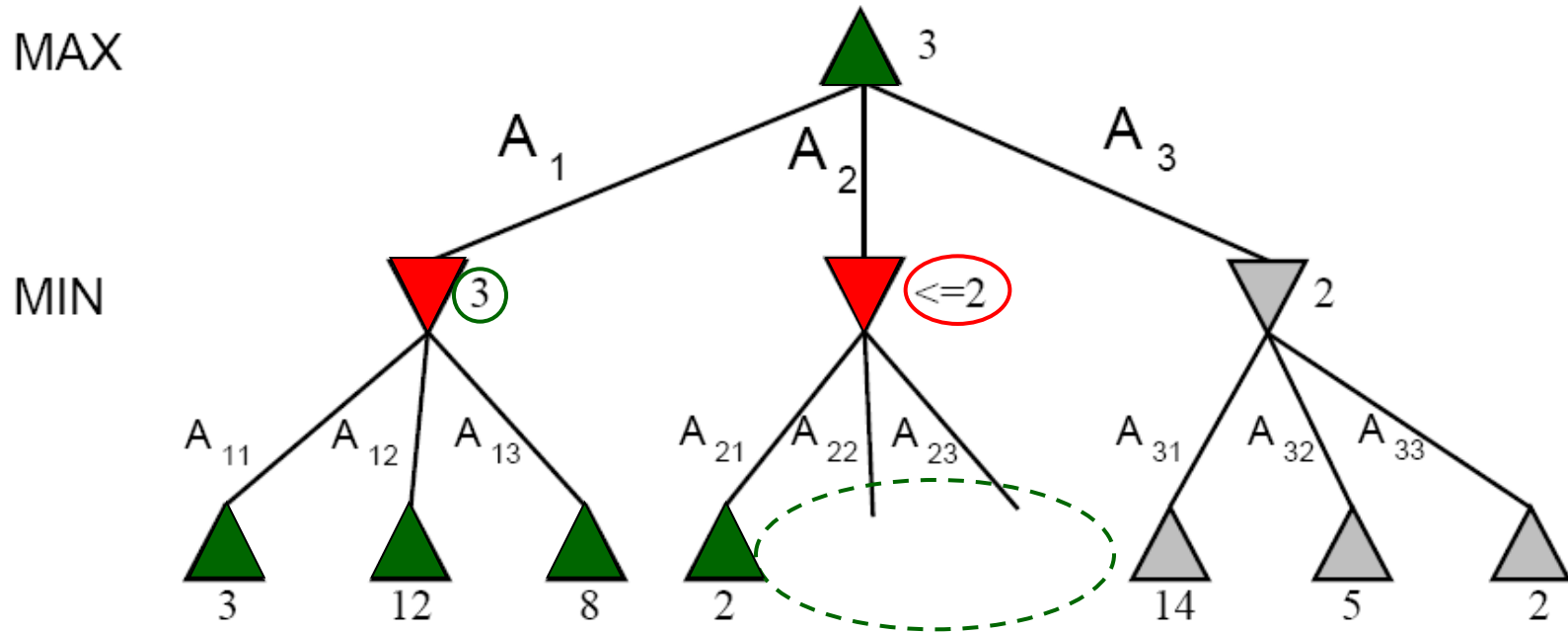
MAX

MIN



Wir müssen nicht alle
Knoten ... und deren
Unteräume (!)
betrachten!

Alpha-Beta-Kürzung: Beispiel



- Falls die Züge A_{22} , A_{23} höhere Bewertungen als A_{21} erhalten, wird MIN diese verwerfen, da A_{21} besser ist für MIN.
- Falls Züge A_{22} , A_{23} niedrigere Bewertungen als A_{21} erhalten, wird MAX diese genauso wie A_{21} verwerfen, da A_1 besser für MAX ist als das resultierende A_2 .
- Allgemein: sobald für den aktuellen Knoten n durch Betrachtung einiger Nachfolger fest steht, dass eine bessere Wahl in einem Vorgängerknoten möglich ist, kann n mit gesamtem Unterbaum verworfen werden.

Alpha-Beta-Werte

zur Erinnerung: Minimax-Algorithmus setzt Tiefensuche ein!

→ immer Betrachtung eines aktiven Pfades.

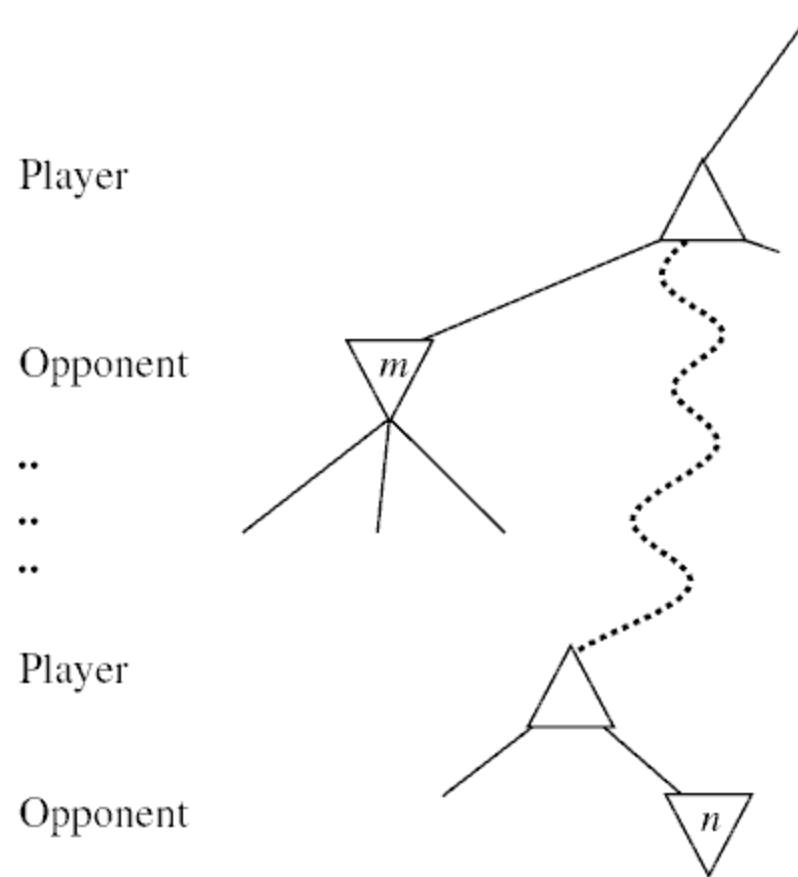
Für diesen werden zwei Werte gehalten:

- α = aktuell bester Wert für MAX auf *aktivem* Pfad.
- β = aktuell bester Wert für MIN auf *aktivem* Pfad.

Initialisierung: $\alpha = -\infty$, $\beta = +\infty$

Strategie: Fortlaufende Aktualisierung von α und β und sowie Verwerfen von Unterbäumen, deren Ergebnisse schlechter sein müssen als die aktuellen Werte von α und β .

Alpha-Beta-Kürzung: Allgemein



Allgemein: Falls Spieler in Elternknoten von n oder in noch höherem Vorgängerknoten von n bessere Alternative m findet, wird der Spieler nie zum Knoten n gehen!

Alpha-Beta-Algorithmus (1)

function ALPHA-BETA-SEARCH(*state*) *returns an action*

inputs: *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)$

Initialisierung mit Zustand
und initialen α - und β -Werten

return the *action* in SUCCESSORS(*state*) with value *v*

function MAX-VALUE(*state*, α , β) *returns a utility value*

inputs: *state*, current state in game

α , the value of the best alternative for MAX along the path to *state*

β , the value of the best alternative for MIN along the path to *state*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

for *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$

if $v \geq \beta$ **then return** *v*

Pruning

$\alpha \leftarrow \text{MAX}(\alpha, v)$

Nur MAX-Knoten verändern
(= erhöhen) Alpha-Werte

return *v*

Alpha-Beta-Algorithmus (2)

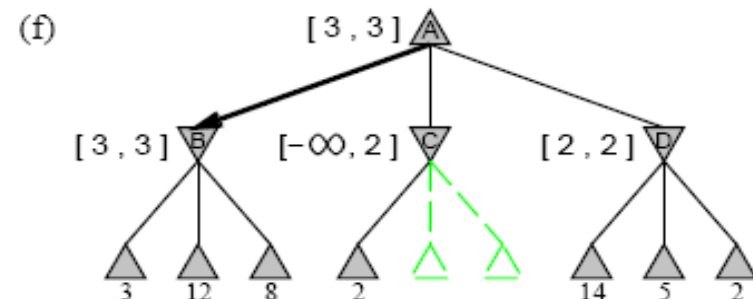
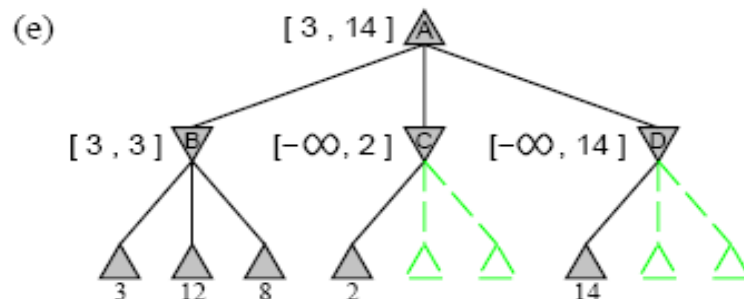
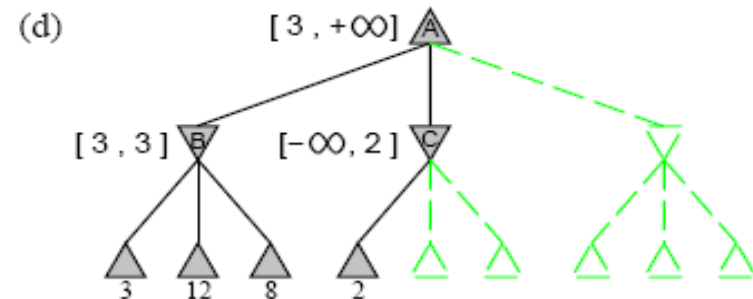
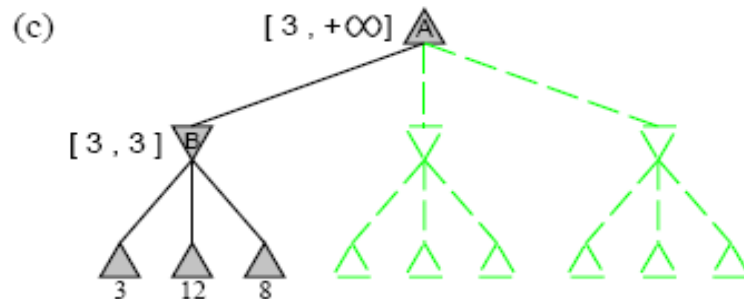
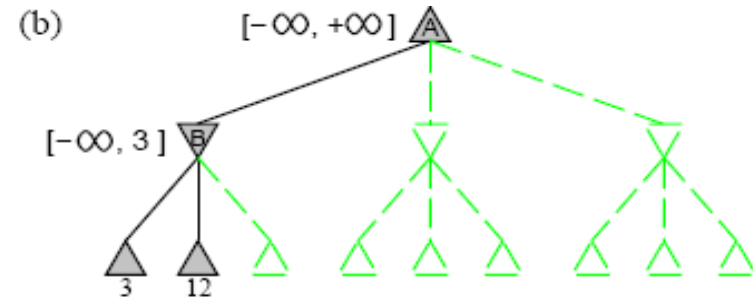
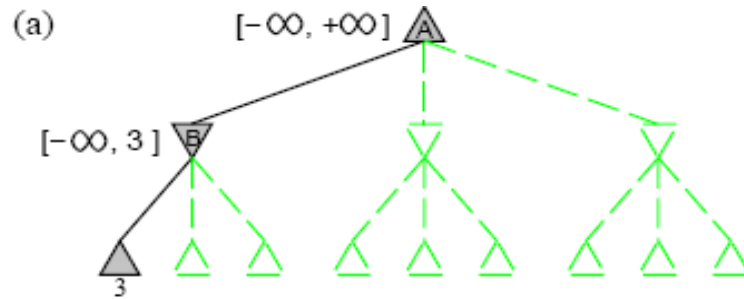
```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
           $\alpha$ , the value of the best alternative for MAX along the path to state
           $\beta$ , the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
    if  $v \leq \alpha$  then return  $v$  ← Pruning
     $\beta \leftarrow \text{MIN}(\beta, v)$  ← Nur MIN-Knoten verändern
                                (= verringern) Beta-Werte
  return  $v$ 
```

Alpha-Beta-Algorithmus: Beispiel

Alpha-Beta-Algm. im Zwei-Schichten-Spielbaum*:

Bereiche möglicher Werte
für alle inneren Knoten



Alpha-Beta für Tic-Tac-Toe (1)

Beispiel für Tic-Tac-Toe auf Zwei-Schichten-Spielbaum (1 Zug MAX, 1 Zug MIN):

Bewertungs- bzw. Evaluierungsfunktion $e(p)$ eines Spielzustandes p sei

- wenn Gewinnposition für MAX: $+\infty$
- wenn Gewinnposition für MIN: $-\infty$
- sonst: Zahl der offenen 3-Ketten für MAX
 – Zahl der offenen 3-Ketten für MIN.

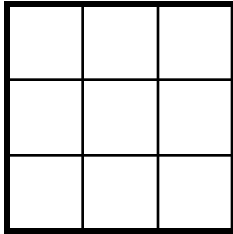
Beispiel:

	X	
	O	

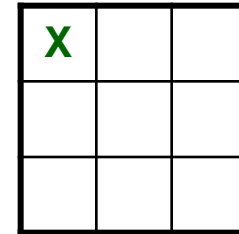
$$e(p) = 6 - 4 = 2$$

Alpha-Beta für Tic-Tac-Toe (2)

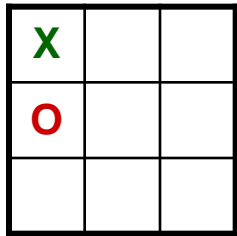
1) Start



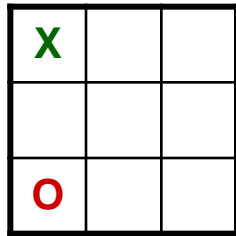
2) erster mögl. Zug MAX



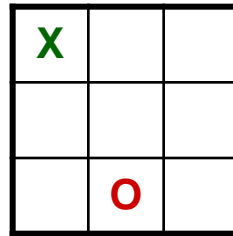
3) Mögliche Reaktionen MIN (unter Ausnutzung der Brettsymmetrie):



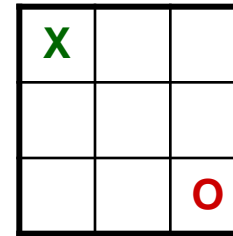
$$6 - 5 = 1$$



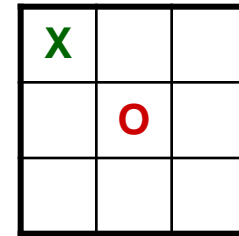
$$5 - 5 = 0$$



$$6 - 5 = 1$$



$$5 - 5 = 0$$



$$4 - 5 = -1$$

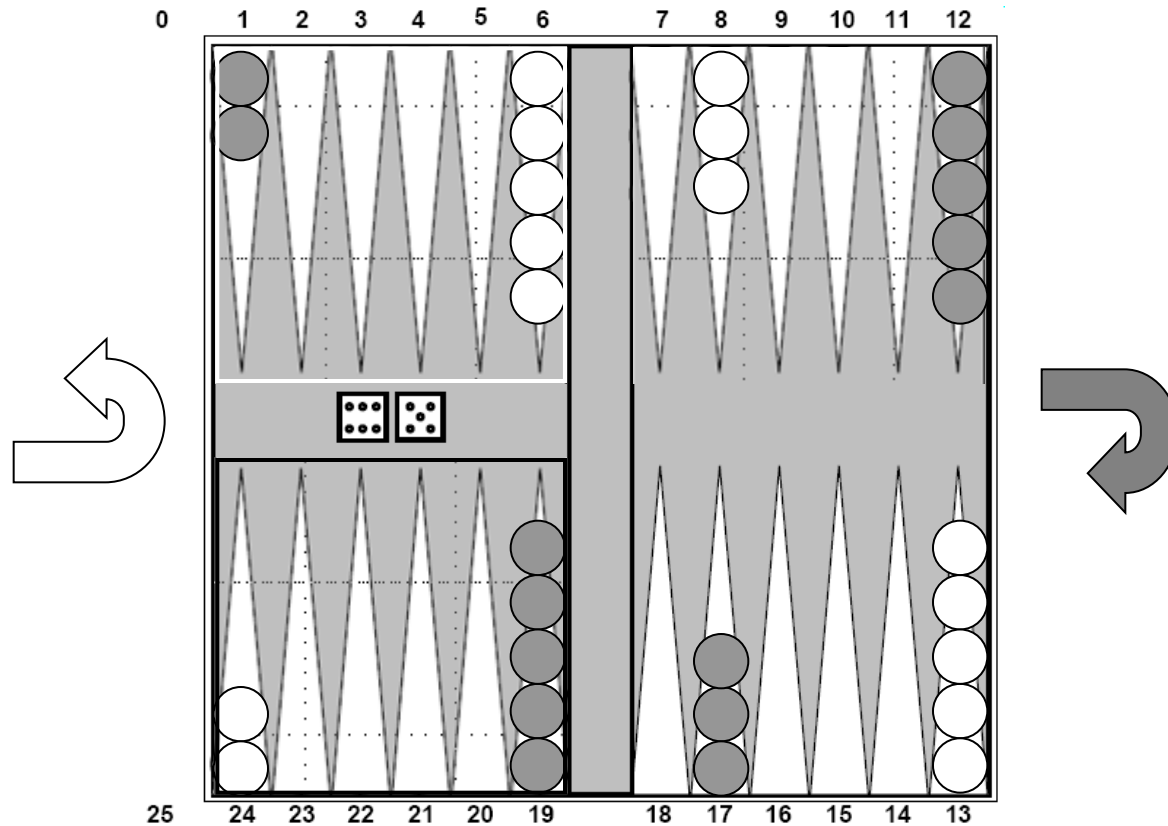
MIN wird -1 wählen:

- MIN wird alternativ nur Zug mit Wert $w \leq -1$ wählen:
- obere Schranke für MIN im entspr. Zustand: $\beta = -1$
- für MAX ist -1 *aktuell* maximale Bewertung eines Nachfolgers von Start,
- MAX wird in Start als Zug nur einen mit Wert größer oder gleich -1 wählen,
- untere Schranke für Bewertung des Zuges von Start für MAX: $\alpha = -1$

Evaluierungsfunktion

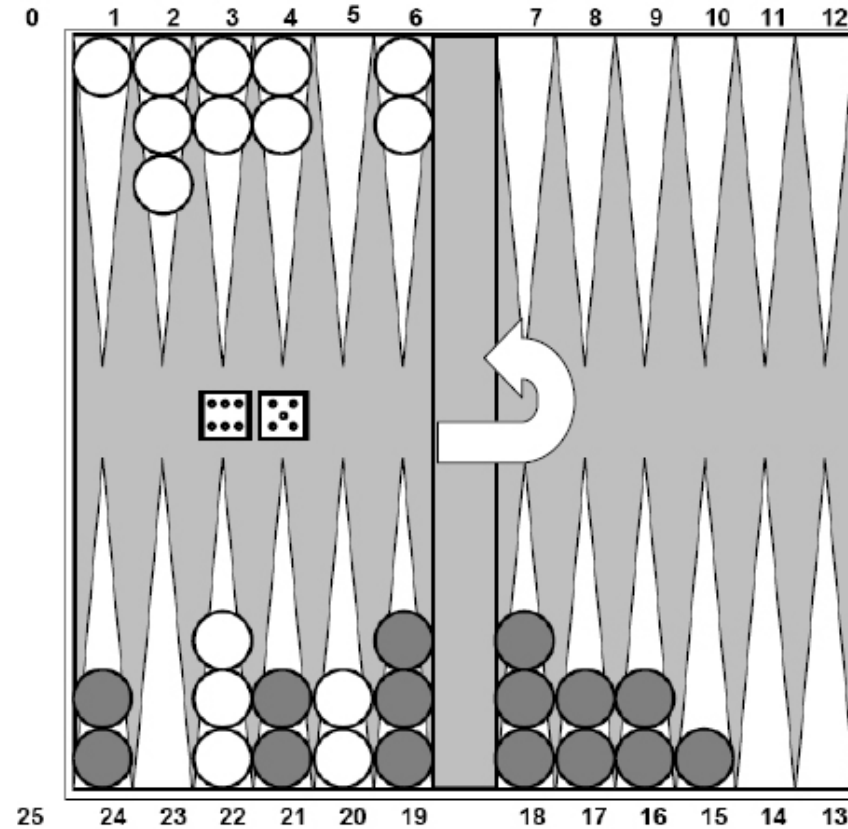
- Alpha-Beta-Suche **schneidet am meisten ab**, wenn jeweils der **beste Zug als erstes** betrachtet wird.
- **Bester Fall** (immer bester Zug zuerst) reduziert den Suchaufwand auf $O(b^{m/2})$ und nicht $O(b^m)$ wie bei Minimax.
- D.h. wir können doppelt so tief in der gleichen Zeit suchen.
- Knuth & Moore (1975): **durchschnittlicher Fall** (zufällig verteilte Züge) reduziert den Suchaufwand auf $O(b^{3m/4})$.
- **Praktischer Fall**: Schon mit relativ einfachen *Anordnungsheuristiken* (zuerst Schlagen, dann Drohen, dann Vorwärtsgehen, dann Rückzug) kommt man in die Nähe des besten Falls.

Spiele mit Zufallsereignissen: Backgammon



Hier relevant:

- **Ziel:** Eigene Steine erst ins eigene Feld und dann aus dem Spiel bringen.
 - Weiß zieht im Gegen-UZS ins Feld 1-6 und dann auf die 0.
 - Schwarz zieht im Uhrzeigersinn ins Feld 19-24 und dann auf die 25.
- Gültige Züge: dorthin, wo nicht mind. 2 gegnerische Steine sind.

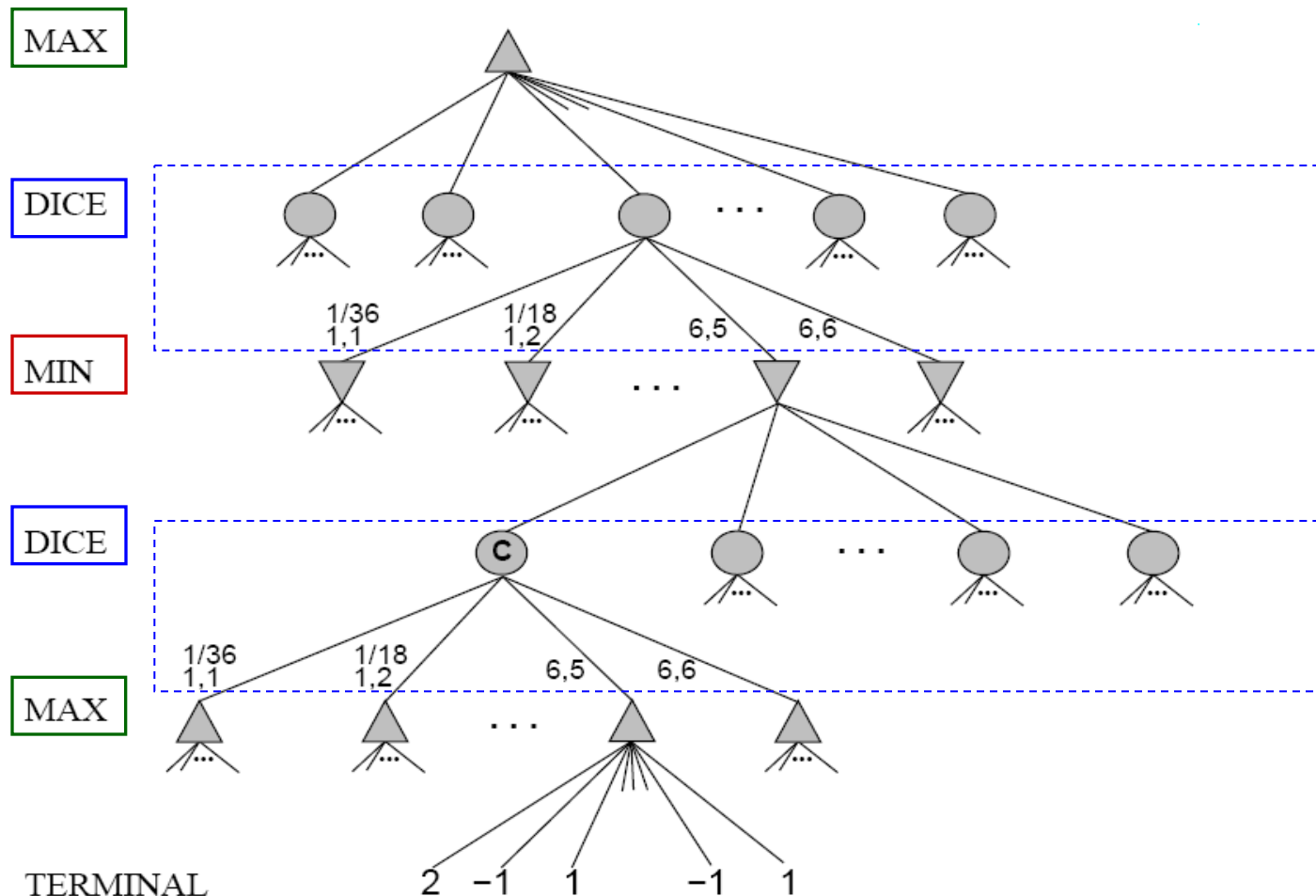


Weiß hat gerade 5 und 6 gewürfelt und hat 4 legale Züge:

$(20 \xrightarrow{6} 14, 20 \xrightarrow{5} 15), (20 \xrightarrow{6} 14, 6 \xrightarrow{5} 1), (20 \xrightarrow{6} 14, 14 \xrightarrow{5} 9), (20 \xrightarrow{5} 15, 15 \xrightarrow{6} 9).$

Spielbaum für Backgammon

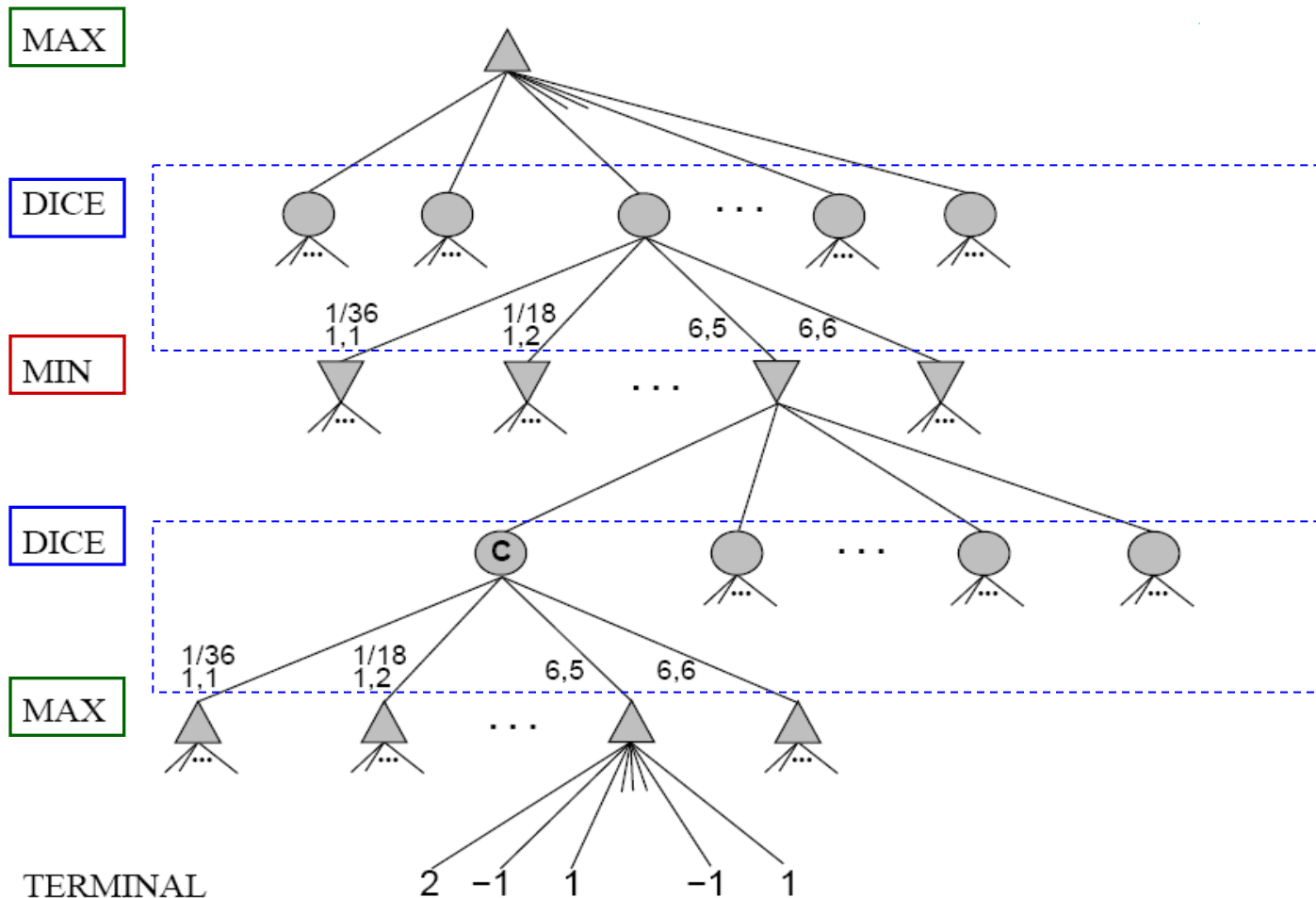
Zusätzlich zu MIN- und MAX-Knoten brauchen wir *Würfelknoten* (*chance nodes* oder *dice nodes*). Deren ausgehende Kanten beschreiben die möglichen Ergebniswerte und deren Wahrscheinlichkeiten.



Berechnung des erwarteten Nutzens

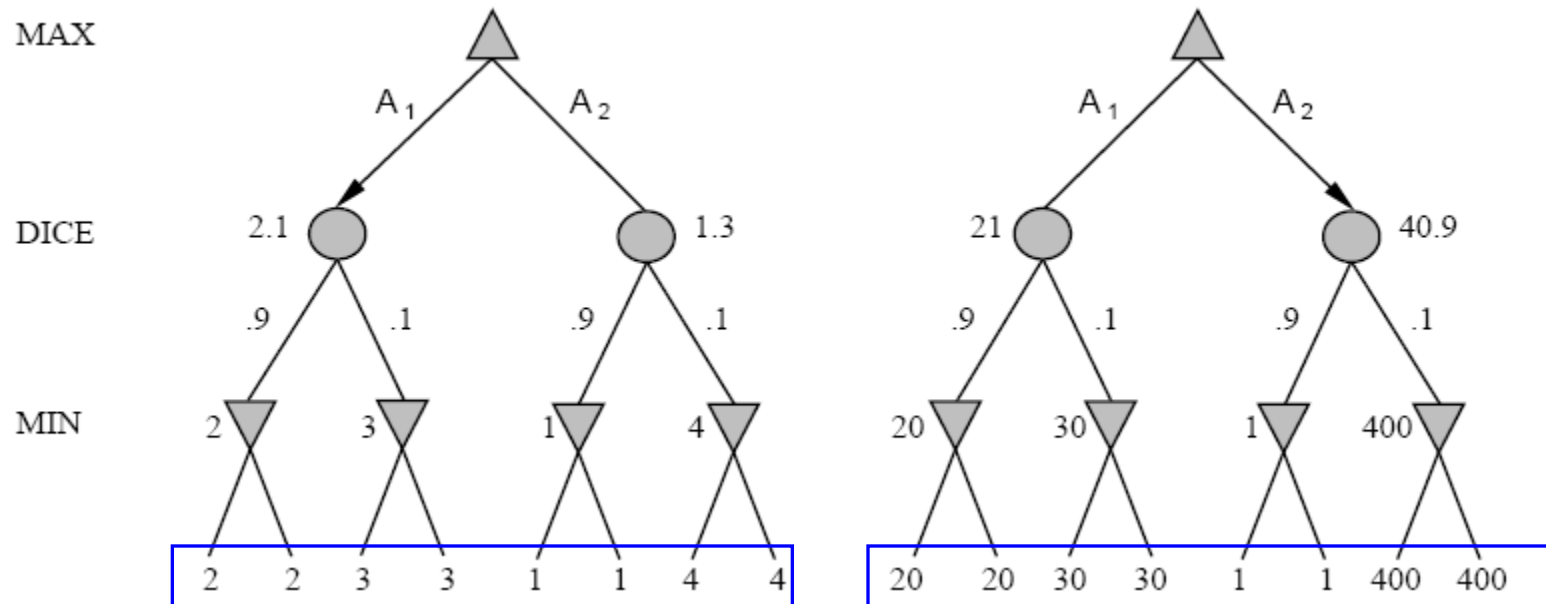
EXPECTIMAX(n) =

- $UTILITY(n)$ if n is a terminal state
- $\max_{s \in Successors(n)} EXPECTIMAX(s)$ if n is a MAX node
- $\min_{s \in Successors(n)} EXPECTIMAX(s)$ if n is a MIN node
- $\sum_{s \in Successors(n)} P(s) \cdot EXPECTIMAX(s)$ if n is a chance node



Probleme

- Varianten von Evaluierungsfunktionen erhalten i. A. nicht die Ordnung zwischen Zügen:
Auswahl links: A_1 , Auswahl rechts: A_2 .



Allg.: Bewertungsfunktion muss **positiv lineare Transformation** der Gewinnchancen sein.

- Die Suchkosten steigen:** Statt des $O(b^m)$ von Minimax haben wir $O(b^m \cdot n^m)$, wenn n die Anzahl möglicher Würfelergebnisse ist. Bei Backgammon ist $n = 21$ und b durchschnittlich 20, manchmal aber sogar um 4000. Unter diesen Umständen kann m maximal 2 sein (also 2 Halbzüge).

Stand der Kunst (1)

Dame (*Checkers, Draughts*, d.h. nach internat. Regeln): Das Programm CHINOOK ist amtierender Weltmeister im Mensch-Maschine-Vergleich (anerkannt von ACF (American Checkers Federation) und EDA (English Draughts Ass.)) und höchst bewerteter Spieler.

Am 19. Juli 2007 veröffentlichte die Zeitschrift Science den Artikel von Schaeffers Team "Checkers Is Solved", der bewies, dass das beste Resultat eines Gegners von Chinook nur unentschieden sein kann.

[http://de.wikipedia.org/wiki/Chinook_\(Software\)](http://de.wikipedia.org/wiki/Chinook_(Software)) (27.4.15)


Stand der Kunst (2)

Backgammon: Das Programm BKG schlug den amtierenden Weltmeister 1980. Ein neueres Programm ist unter den ersten 3 Spielern.

Reversi (*Othello*): Sehr gut auch auf normalen Computern. Programme werden bei Turnieren nicht zugelassen.

Stand der Kunst (3)

Go: Die besten Programme spielen etwas besser als Anfänger (Verzwei-



The screenshot shows the AlphaGo website with a dark blue background featuring a Go board pattern. At the top left is the Google DeepMind logo. To the right are navigation links: Home, AlphaGo, DQN, Health, Press, Join us, and Publications. In the center is the AlphaGo logo, which consists of a stylized Go stone icon followed by the text "AlphaGo". Below the logo is a horizontal line, and then the text "THE FIRST COMPUTER PROGRAM TO EVER BEAT A PROFESSIONAL PLAYER AT THE GAME OF GO." in all caps. At the bottom of the screenshot is a white banner with the text: "Google DeepMind's AlphaGo schlägt den als stärkster Go-Spieler der Welt geltenden Lee Sedol im März 2016 by 4 – 1".

Google DeepMind's AlphaGo schlägt den als stärkster Go-Spieler der Welt geltenden Lee Sedol im März 2016 by 4 – 1

Zitat aus [http://de.wikipedia.org/wiki/Go_\(Spiel\)](http://de.wikipedia.org/wiki/Go_(Spiel)) (19.04.2012)

Schach (1)

Schach als „Drosophila“ der KI-Forschung:

- begrenzte Anzahl von Regeln bringt unbegrenzte Zahl von Spielverläufen hervor. Für eine Partie von 40 Zügen gibt es 1.5×10^{128} mögliche Spielverläufe;
- suggeriert Sieg durch Logik, Intuition, Kreativität, Vorwissen;
- erfordert nur spezielle Schachintelligenz, keine „Alltagsintelligenz“.

Spielstärke wird in ELO-Punkten gemessen:

Spielstärken (Stand Juli 1999):

<i>G. Kasparow</i>	2851 ELO
<i>V. Anand</i>	2758 ELO
<i>A. Karpow</i>	2710 ELO
<i>Deep Thought 2</i>	2680 ELO

* Arpad E. Elo, Administrator (1935 - 1937) of the Amer. Chess Federation.

Siehe auch: <http://de.wikipedia.org/wiki/Elo-Zahl> (27.04.2015)

Schach (2)

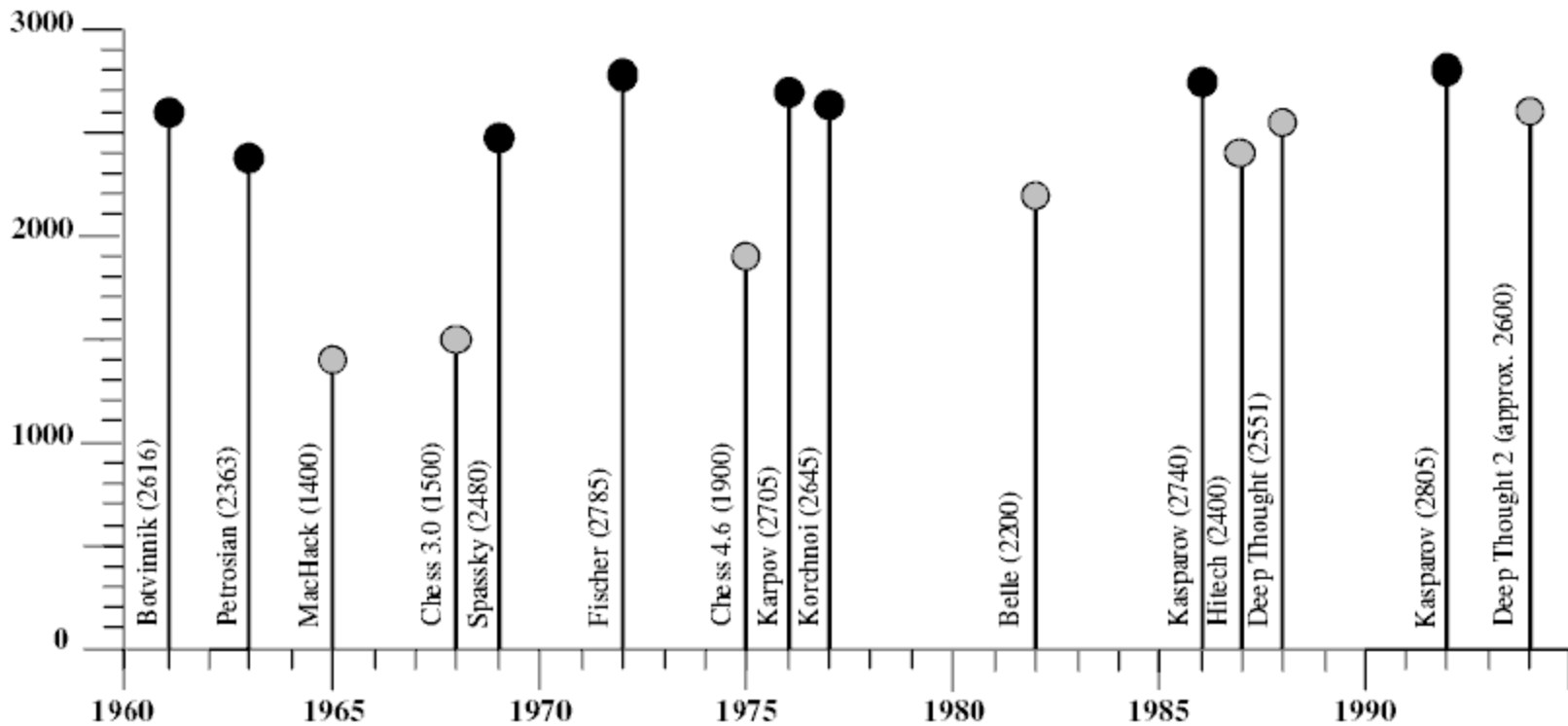
1997 wird der amtierende Weltmeister G. Kasparow **erstmal**s in einem Spiel aus 6 Partien **von einem Computer geschlagen**!

→ Deep Thought 2 (IBM Thomas J. Watson Research Center)

- spezielle Hardware (32 Rechner mit 8 Chips, 2 Mio Berechnungen pro Sekunde)
- heuristische Suche
- fallbasiertes Schließen + Lerntechniken
 - 1996 Wissen aus 600.000 Schachpartien
 - 1997 Wissen aus 2 Mio. Schachpartien
 - Training durch Großmeister

Duell „Maschinenmensch Kasparow - Menschmaschine Deep Thought 2“

Schach (3)



Schach (4)

Elo-Zahlen der stärksten Schachprogramme^[6]

Rang	Name	Punkte
1	Stockfish 7.0 x64 12CPU	3458
2	Komodo 9.3 x64 12CPU	3416
3	Houdini 4.0 x64 12CPU	3247
4	Gull 3.0 x64 12CPU	3242
5	Fritz 15 x64 12CPU	3180
6	Rybka 4.1 x64 12CPU	3133
7	Critter 1.6 x64 4CPU	3122
8	Equinox 3.00 x64 4CPU	3112
9	Ginkgo 1.5 x64 4CPU	3103
10	Sting SF 5 x64 4CPU	3046

CEGT-Rangliste

Quelle:

<http://de.wikipedia.org/wiki/Schachprogramm>

(Abruf: 21.04.2016)

Zum Vergleich: Ein Schachweltmeister von heute zeigt ca. ELO 2850 (Magnus Carlsen 2851 (<https://de.wikipedia.org/wiki/Elo-Zahl>, 21.04.2016)). Diese ELO-Zahlen für Schachprogramme sind aber nicht ohne weiteres mit denen menschlicher Schachspieler zu vergleichen, da sie überwiegend durch Partien zwischen Computern ermittelt wurden und nicht durch Teilnahme an offiziellen Turnieren.

Schach (5)

Eingesetzte Methoden und Techniken in Schachcomputern:

- Alpha-Beta-Suche
- . . . mit dynamischer Tiefenfestlegung bei unsicheren Positionen,
- gute (aber normalerweise einfache) Evaluierungsfunktionen,
- große Eröffnungsdatenbanken,
- sehr große Endspieldatenbanken (für Dame: alle 8-Steine Situationen),
- und sehr schnelle und parallele Rechner!

Zusammenfassung

- Ein **Spiel** ist i. A. ein Suchproblem in einer **Multiagenten-Umgebung**.
- Ein Spiel kann durch Angabe der **Zustandsmenge**, des **Anfangszustands**, der **Operatoren** (legale Züge), eines **Terminaltests** und einer **Nutzenfunktion** (Ausgang des Spiels) beschrieben werden.
- In 2-Personen-Brettspielen kann der **Minimax-Algorithmus** den besten Zug bestimmen, indem er den ganzen Spielbaum aufbaut.
- Der **Alpha-Beta-Algorithmus** liefert das gleiche Ergebnis, ist jedoch effizienter, da er überflüssige Zweige abschneidet.
- Normalerweise kann nicht der ganze Spielbaum aufgebaut werden, so dass man Zwischenzustände mit einer **Evaluierungsfunktion** bewerten muss.
- **Spiele mit Zufallselement** kann man mit einer **Erweiterung** des **Minimax** – und auch des **Alpha-Beta-Algorithmus** behandeln.