

Einführung in die Methoden der Künstlichen Intelligenz

Suche bei Spielen

PD Dr. David Sabel

SoSe 2014

Zwei-Spieler-Spiele

Ziel dieses Abschnitts

- Intelligenter Agent für Zweipersonenspiele
- Beispiele: Schach, Dame, Mühle, ...

Zwei-Spieler-Spiele

Ziel dieses Abschnitts

- Intelligenter Agent für Zweipersonenspiele
- Beispiele: Schach, Dame, Mühle, ...

Annahme zunächst:

- Spiele mit genau zwei Gegnern
- Kein Zufall im Spiel (keine Würfel, Kartenmischen) etc.

- Gute Strategien zum Schachspielen schon entwickelt bevor es Computer gab
- Untersuchungen zu Gewinnstrategien (Eröffnungen, ...)
- **Bewerten** von Spielsituationen

- Gute Strategien zum Schachspielen schon entwickelt bevor es Computer gab
- Untersuchungen zu Gewinnstrategien (Eröffnungen, ...)
- **Bewerten** von Spielsituationen

Im Wesentlichen 2 Ansätze für Programme zum Schachspielen

- 1 Verwende bekannte Schachtheorien (Verteidigung, Angriff, Bedrohung, Decken, Mittelfeld, ...). Versuche die „menschliche Methode“ zu programmieren
- 2 Absuchen aller Möglichkeiten, um den nächsten guten Zug zu finden.

Schach (2)

Moderne erfolgreiche Schachprogramme:

- verwenden im Wesentlichen das Absuchen
- verwenden statische Bewertung der Spielsituationen
- zusätzlich Bibliotheken für Eröffnung und Endspiel

Spielbäume

- **Zwei Spieler:** genannt **Maximierer** und **Minimierer**
- **Zustand** (Knoten): Darstellung der Spielsituation und welcher Spieler am Zug ist
- **Wurzel:** Aktuelle Spielsituation
- **Kinder** von Knoten K :
mögliche Spielsituationen nach einem Zug
Spieler wechselt
- Blätter mit **Gewinnwert** bewertet.
- Schach:
 - -1 = Minimierer hat gewonnen
 - 1 = Maximierer hat gewonnen
 - 0 = Remis

Beispiel: TicTacToe

Spielfeld:



- Maximierer: *X*
- Minimierer: *O*
- schreiben abwechselnd ihr Symbol in ein freies Feld
- Gewinner: Spieler hat sein Symbol in einer Reihe (horizontal, vertikal, diagonal)

Beispiel: TicTacToe

Spielfeld:



- Maximierer: *X*
- Minimierer: *O*
- schreiben abwechselnd ihr Symbol in ein freies Feld
- Gewinner: Spieler hat sein Symbol in einer Reihe (horizontal, vertikal, diagonal)

Beispiel: TicTacToe

Spielfeld:

<i>X</i>	<i>O</i>	

- Maximierer: *X*
- Minimierer: *O*
- schreiben abwechselnd ihr Symbol in ein freies Feld
- Gewinner: Spieler hat sein Symbol in einer Reihe (horizontal, vertikal, diagonal)

Beispiel: TicTacToe

Spielfeld:

<i>X</i>	<i>O</i>	
	<i>X</i>	

- Maximierer: *X*
- Minimierer: *O*
- schreiben abwechselnd ihr Symbol in ein freies Feld
- Gewinner: Spieler hat sein Symbol in einer Reihe (horizontal, vertikal, diagonal)

Beispiel: TicTacToe

Spielfeld:

<i>X</i>	<i>O</i>	
	<i>X</i>	
		<i>O</i>

- Maximierer: *X*
- Minimierer: *O*
- schreiben abwechselnd ihr Symbol in ein freies Feld
- Gewinner: Spieler hat sein Symbol in einer Reihe (horizontal, vertikal, diagonal)

Beispiel: TicTacToe

Spielfeld:

<i>X</i>	<i>O</i>	
<i>X</i>	<i>X</i>	
		<i>O</i>

- Maximierer: *X*
- Minimierer: *O*
- schreiben abwechselnd ihr Symbol in ein freies Feld
- Gewinner: Spieler hat sein Symbol in einer Reihe (horizontal, vertikal, diagonal)

Beispiel: TicTacToe

Spielfeld:

<i>X</i>	<i>O</i>	
<i>X</i>	<i>X</i>	
<i>O</i>		<i>O</i>

- Maximierer: *X*
- Minimierer: *O*
- schreiben abwechselnd ihr Symbol in ein freies Feld
- Gewinner: Spieler hat sein Symbol in einer Reihe (horizontal, vertikal, diagonal)

Beispiel: TicTacToe

Spielfeld:

<i>X</i>	<i>O</i>	
<i>X</i>	<i>X</i>	<i>X</i>
<i>O</i>		<i>O</i>

- Maximierer: *X*
- Minimierer: *O*
- schreiben abwechselnd ihr Symbol in ein freies Feld
- Gewinner: Spieler hat sein Symbol in einer Reihe (horizontal, vertikal, diagonal)

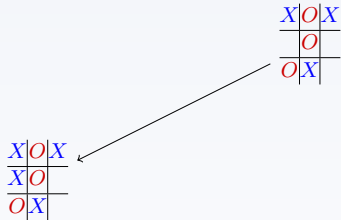
Beispiel: Tic Tac Toe, Spielbaum

Max

X	O	X
	O	
O	X	

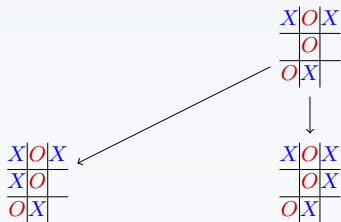
Beispiel: Tic Tac Toe, Spielbaum

Max



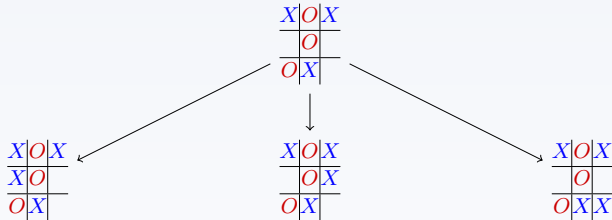
Beispiel: Tic Tac Toe, Spielbaum

Max

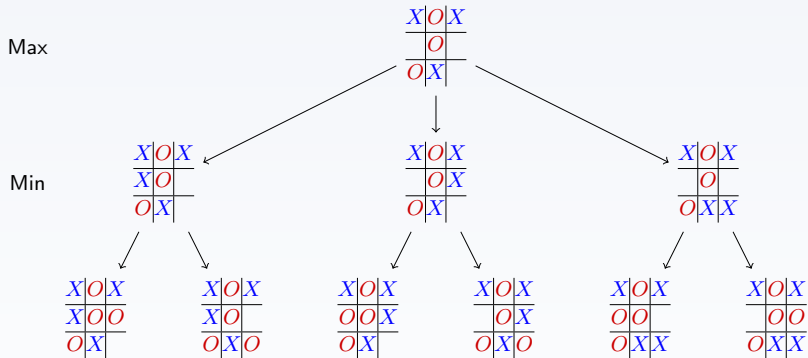


Beispiel: Tic Tac Toe, Spielbaum

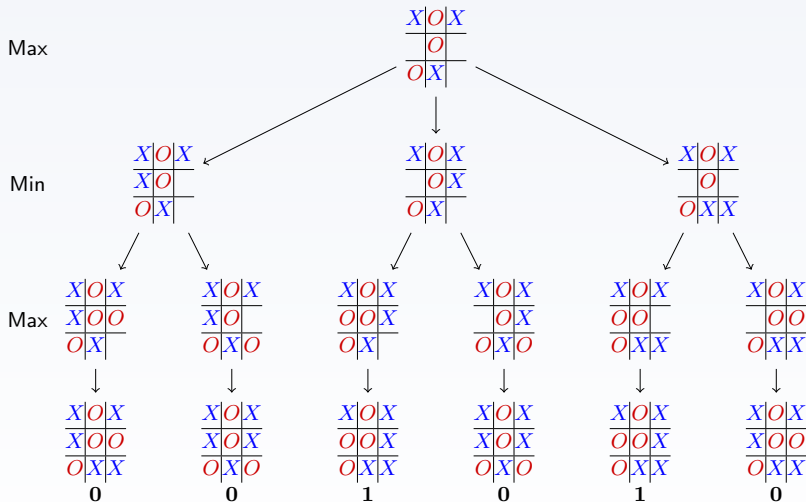
Max



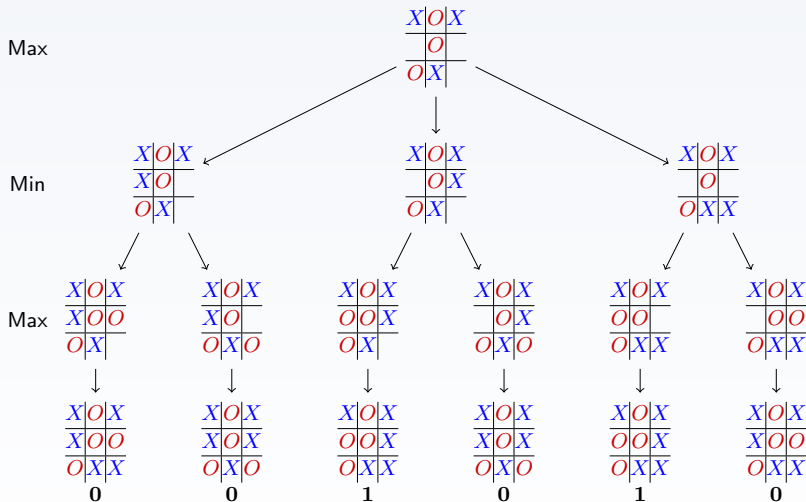
Beispiel: Tic Tac Toe, Spielbaum



Beispiel: Tic Tac Toe, Spielbaum



Beispiel: Tic Tac Toe, Spielbaum



Welchen Zug soll Max machen?

MinMax-Suche (1)

- **Eingabe:** Spielsituation, Nachfolgerfunktion, Bewertung
- **Ausgabe:** Nächster optimaler Zug

MinMax-Suche (1)

- **Eingabe:** Spielsituation, Nachfolgerfunktion, Bewertung
- **Ausgabe:** Nächster optimaler Zug

Informelle Beschreibung der MinMax-Suche:

- Bewerte alle Blätter
- Berechne den Wert der anderen Knoten:
 - Stufenweise von unten nach oben
 - über die Werte der Kinder wird maximiert bzw. minimiert
 - je nachdem welcher Spieler am Zug ist
- Dabei: Züge merken, damit der optimale Zug bekannt ist

MinMax-Suche (2)

Algorithmus MinMax-Suche

Datenstrukturen: Nachfolgerfunktion, Wert(Zustand) für Endsituationen, Datenstruktur für Spielzustand, Spieler

Funktion Minmax(Zustand, Spieler):

$NF :=$ alle Nachfolgezustände zu (Zustand, Spieler);

if $NF = \emptyset$ then return Wert(Zustand) else

 if Spieler == Maximierer then

 return $\max\{\text{Minmax}(Z, \overline{\text{Spieler}}) \mid Z \in NF\}$

 else // Spieler ist Minimierer

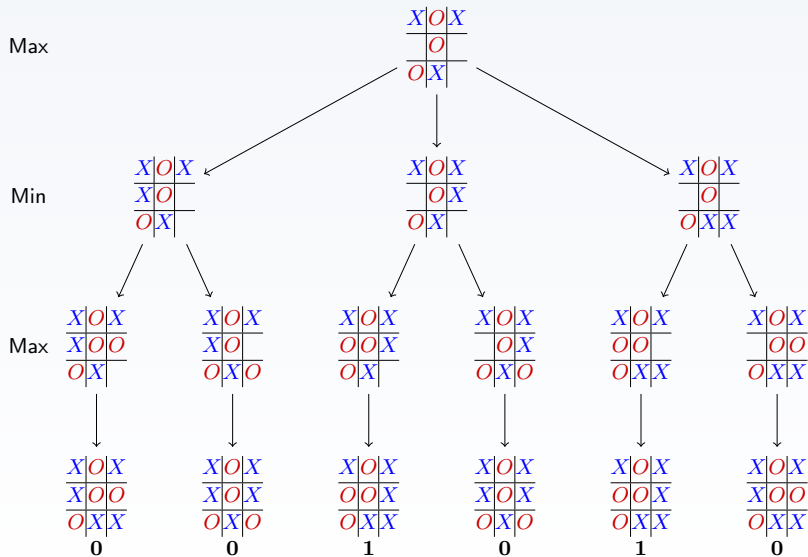
 return $\min\{\text{Minmax}(Z, \overline{\text{Spieler}}) \mid Z \in NF\}$

 endif

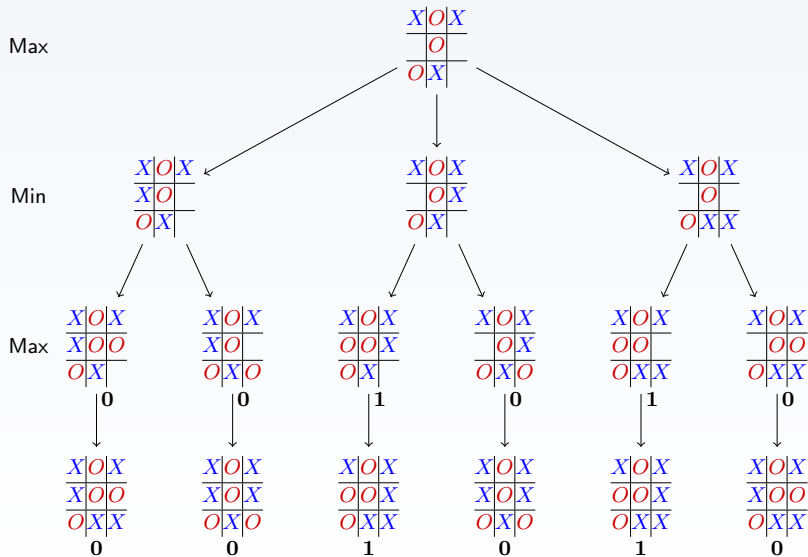
endif

- wobei $\overline{\text{Spieler}}$ = der jeweils anderen Spieler
- Implementierung: sollte zus. den Weg speichern, damit der nächste optimale Zug als Ausgabe zur Verfügung steht.

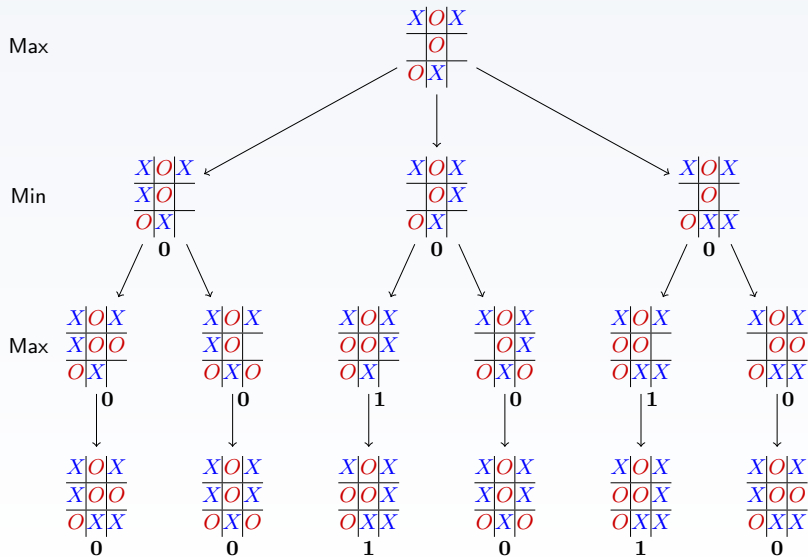
Beispiel, TicTacToe



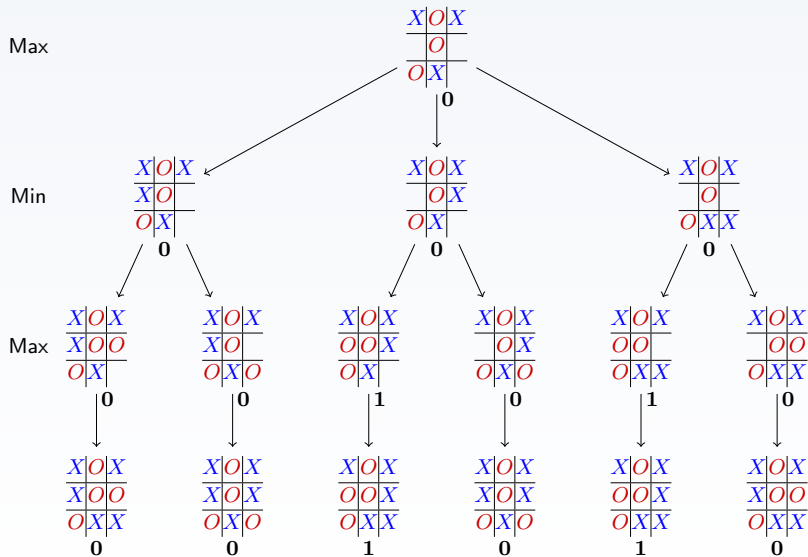
Beispiel, TicTacToe



Beispiel, TicTacToe



Beispiel, TicTacToe



MinMax in Haskell

```
minmax endZustand wert nachfolger spieler zustand =
  go (zustand, []) spieler
  where
    go (zustand,p) spieler
      | endZustand zustand = (wert zustand,reverse $ zustand:p)
      | otherwise =
          let l = nachfolger spieler zustand
              in case spieler of
                Maximierer -> maximumBy
                    (\(a,_) (b,_) -> compare a b)
                    [go (z,zustand:p) Minimierer | z <- l]
                Minimierer -> minimumBy
                    (\(a,_) (b,_) -> compare a b)
                    [go (z,zustand:p) Maximierer | z <- l]
```

MinMax-Eigenschaften

- MinMax kann als **Tiefensuche** durchgeführt werden
- Laufzeit $O(c^d)$ bei mittlerer Verzweigungsrate c und Blättern in Tiefe d

MinMax-Eigenschaften

- MinMax kann als **Tiefensuche** durchgeführt werden
- Laufzeit $O(c^d)$ bei mittlerer Verzweigungsrate c und Blättern in Tiefe d
- **Praktisches Problem**: Bei mittelschweren Spielen: **unmöglich** in annehmbarer Zeit berechenbar
- TicTacToe: Vom leeren Spielfeld aus: $9! = 362.880$ Zugfolgen, geht gerade noch

MinMax-Eigenschaften

- MinMax kann als **Tiefensuche** durchgeführt werden
- Laufzeit $O(c^d)$ bei mittlerer Verzweigungsrate c und Blättern in Tiefe d
- **Praktisches Problem**: Bei mittelschweren Spielen: **unmöglich** in annehmbarer Zeit berechenbar
- TicTacToe: Vom leeren Spielfeld aus: $9! = 362.880$ Zugfolgen, geht gerade noch

Daher: **Abwandlung** des Verfahrens!

- Suche nur bis zur einer bestimmten Tiefe k
- **Bewerte** die Knoten in Tiefe k mit einer **Heuristik**

Beispiele für die Bewertungsfunktion:

- **Schach**: Materialvorteil, evtl. Stellungsvorteil, Gewinnsituation, ...
- **Mühle**: Material, #Mühlen, Bewegungsfreiheit, ...

Beispiel: Heuristische Bewertung für TicTacToe

- (#einfach X-besetzte Zeilen/Spalten/Diag) * 1
- + (# doppelt X-besetzte Zeilen/Spalten/Diag) * 5
- + (20, falls Gewinnsituation)
- (#einfach O-besetzte Zeilen/Spalten/Diag) * 1
- (# doppelt O-besetzte Zeilen/Spalten/Diag) * 5
- (20, falls Verlustsituation)

Bewertungsfunktionen

- Güte bestimmt Güte des Verfahrens
- muss algorithmisch (möglichst schnell) berechenbar sein
- sollte Endzustände entsprechend minimal und maximal bewerten

Es gilt (in Spielen ohne Zufall):

Berechnung des besten Zuges ist **unabhängig** von den exakten Werten der Bewertungsfunktion;
Die **Ordnung** zwischen den Spielsituationen bestimmt.

Beispiel: TicTacToe

	Sieg	Niederlage	Unentschieden
Bewertung 1	1	-1	0
Bewertung 2	100	10	50
Bewertung 3	1	0	0

- Bewertung 1 und Bewertung 2 führen zum gleichen optimalen Zug, da
Sieg $>$ Unentschieden $>$ Niederlage
- Bewertung 3 hat eine andere Ordnung:
Sieg $>$ Unentschieden = Niederlage
und kann daher zu anderem optimalen Zug führen

Wenn man ein sehr gute Bewertungsfunktion hat:

Warum MinMax ausführen und nicht gleich
nur die Bewertungen der Kinder anschauen?

Einwand

Wenn man ein sehr gute Bewertungsfunktion hat:

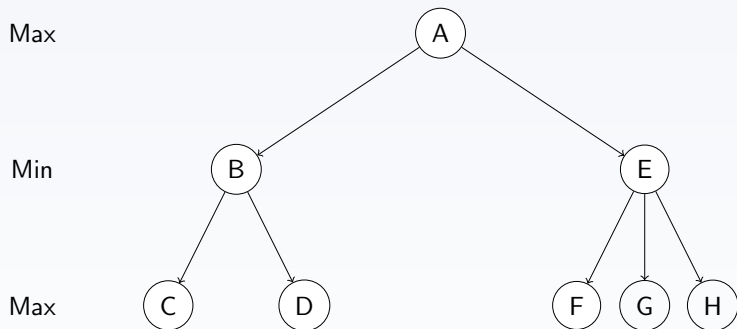
Warum MinMax ausführen und nicht gleich
nur die Bewertungen der Kinder anschauen?

Üblicherweise:

Je tiefer man geht, umso **besser** wird die Bewertungsfunktion!

- Mittlere Verzweigungsrate c
- Tiefenschranke m
- Laufzeit $O(c^m)$, wenn die Heuristik in konstanter Zeit berechenbar ist

Optimierung der MinMax-Suche: Beispiele



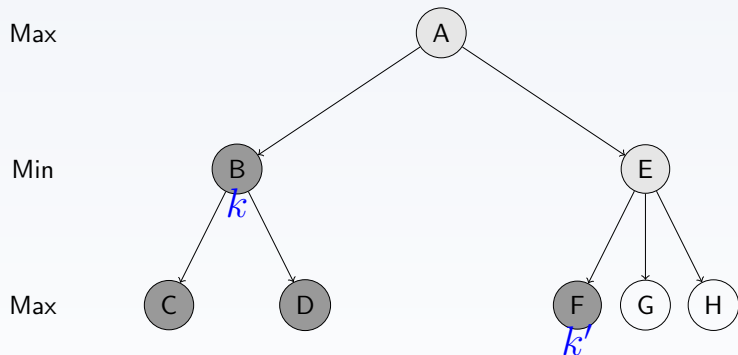
$$\text{MinMax}(A, \text{Max})$$

$$= \max\{\text{MinMax}(B, \text{Min}), \text{MinMax}(E, \text{Min})\}$$

$$= \max\{\min\{\text{MinMax}(C, \text{Max}), \text{MinMax}(D, \text{Max})\},$$

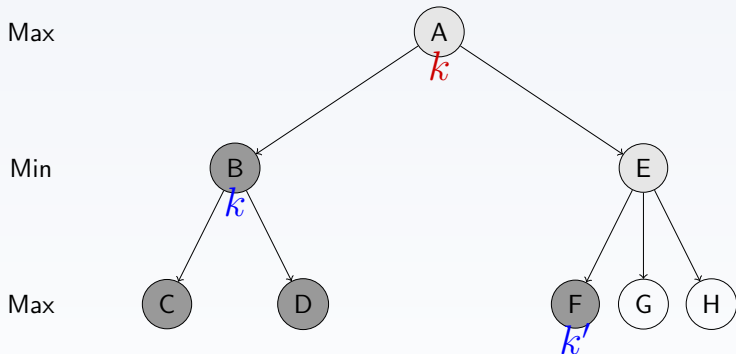
$$\min\{\text{MinMax}(F, \text{Max}), \text{MinMax}(G, \text{Max}), \text{MinMax}(H, \text{Max})\}\}$$

Optimierung der MinMax-Suche: Beispiele



$$\begin{aligned}
 & \text{MinMax}(A, \text{Max}) \\
 = & \max\{\text{MinMax}(B, \text{Min}), \text{MinMax}(E, \text{Min})\} \\
 = & \max\{k, \min\{k', \text{MinMax}(G, \text{Max}), \text{MinMax}(H, \text{Max})\}\}
 \end{aligned}$$

Optimierung der MinMax-Suche: Beispiele



$$\text{MinMax}(A, \text{Max})$$

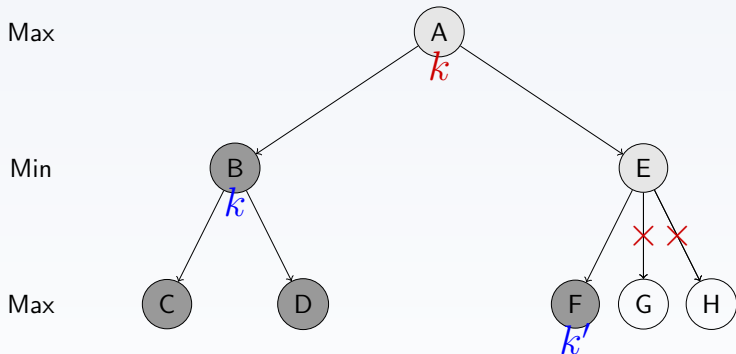
$$= \max\{\text{MinMax}(B, \text{Min}), \text{MinMax}(E, \text{Min})\}$$

$$= \max\{k, \min\{k', \text{MinMax}(G, \text{Max}), \text{MinMax}(H, \text{Max})\}\}$$

Wenn $k \geq k'$:

$$= k$$

Optimierung der MinMax-Suche: Beispiele



$$\text{MinMax}(A, \text{Max})$$

$$= \max\{\text{MinMax}(B, \text{Min}), \text{MinMax}(E, \text{Min})\}$$

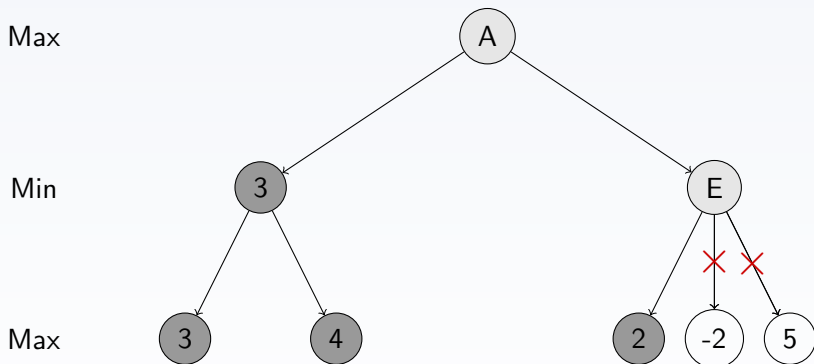
$$= \max\{k, \min\{k', \text{MinMax}(G, \text{Max}), \text{MinMax}(H, \text{Max})\}\}$$

Wenn $k \geq k'$:

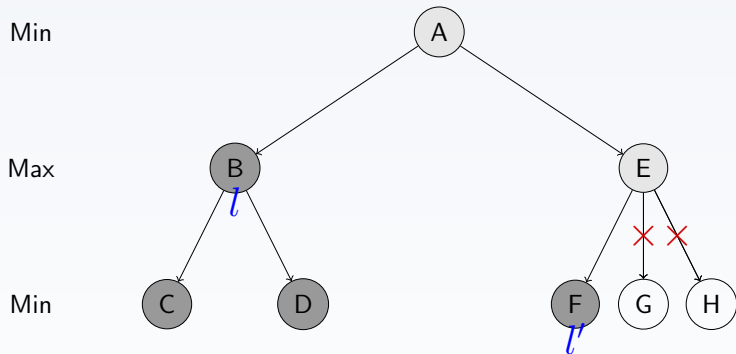
$$= k$$

\implies Knoten (bzw. Teilbäume) G und H **braucht man nicht betrachten!**

Beispiel mit konkreten Werten



Symmetrischer Fall



$$\text{MinMax}(A, \text{Min})$$

$$= \min\{\text{MinMax}(B, \text{Max}), \text{MinMax}(E, \text{Max})\}$$

$$= \min\{l, \max\{l', \text{MinMax}(G, \text{Min}), \text{MinMax}(H, \text{Min})\}\}$$

Wenn $l \leq l'$:

$$= l$$

- Kürzt genau diese Fälle weg
- Durch Verwendung eines Suchfensters $[\alpha, \beta]$
- Tiefensuche mit Tiefenschranke notwendig
Anpassung der Tiefenschranke im folgenden Algorithmus
implizit

Algorithmus $\alpha - \beta$ -Suche

Aufruf: AlphaBeta(Zustand, Spieler, $-\infty, +\infty$)

Funktion: AlphaBeta(Zustand, Spieler, α, β)

- ① Wenn Tiefenschranke erreicht, bewerte die Situation, return Wert
- ② Sei NF die Folge der Nachfolger von (Zustand, Spieler)
- ③ Wenn Spieler = Minimierer:


```

 $\beta_l := \infty;$ 
for-each  $L \in NF$ 
   $\beta_l := \min\{\beta_l, \text{AlphaBeta}(L, \text{Maximierer}, \alpha, \beta)\}$ 
  if  $\beta_l \leq \alpha$  then return  $\beta_l$  endif // verlasse Schleife
 $\beta := \min\{\beta, \beta_l\}$ 
end-for
return  $\beta_l$ 

```
- ④ Wenn Spieler = Maximierer


```

 $\alpha_l := -\infty;$ 
for-each  $L \in NF$ 
   $\alpha_l := \max\{\alpha_l, \text{AlphaBeta}(L, \text{Minimierer}, \alpha, \beta)\}$ 
  if  $\alpha_l \geq \beta$  then return  $\alpha_l$  endif // verlasse Schleife
 $\alpha := \max\{\alpha, \alpha_l\}$ 
end-for
return  $\alpha_l$ 

```

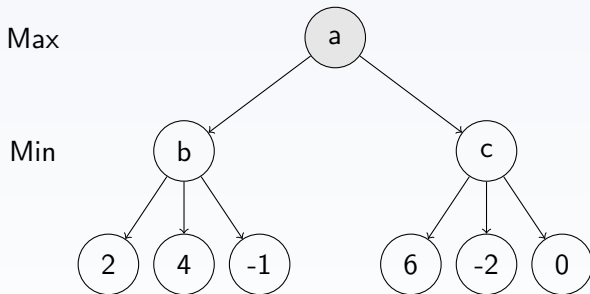


```

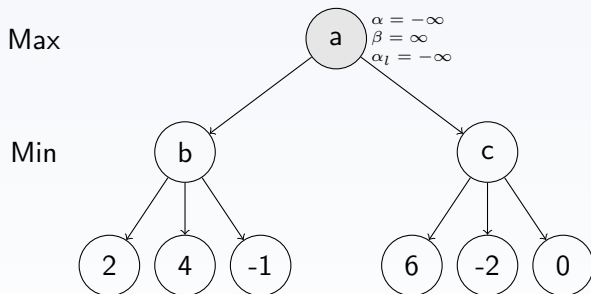
alphaBeta wert nachfolger maxtiefe neginfy infy spieler zustand =
  ab 0 neginfy infy spieler zustand
where
  ab tiefe alpha beta spieler zustand
  | null (nachfolger spieler zustand) || maxtiefe <= tiefe = wert zustand
  | otherwise =
    let l = nachfolger spieler zustand
    in
      case spieler of
        Maximierer -> maximize tiefe alpha beta l
        Minimierer -> minimize tiefe alpha beta l
maximize tiefe alpha beta xs = it_maximize alpha neginfy xs
where
  it_maximize alpha alpha_l []      = alpha_l
  it_maximize alpha alpha_l (x:xs) =
    let alpha_l' = max alpha_l (ab (tiefe+1) alpha beta Minimierer x)
    in if alpha_l' >= beta then alpha_l' else
       it_maximize (max alpha alpha_l') alpha_l' xs
minimize tiefe alpha beta xs = it_minimize beta infy xs
where
  it_minimize beta beta_l []      = beta_l
  it_minimize beta beta_l (x:xs) =
    let beta_l' = min beta_l (ab (tiefe+1) alpha beta Maximierer x)
    in if beta_l' <= alpha then beta_l' else
       it_minimize (min beta beta_l') beta_l' xs

```

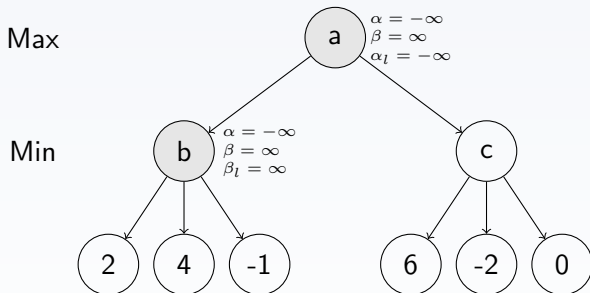
Alpha-Beta-Suche: Beispiel



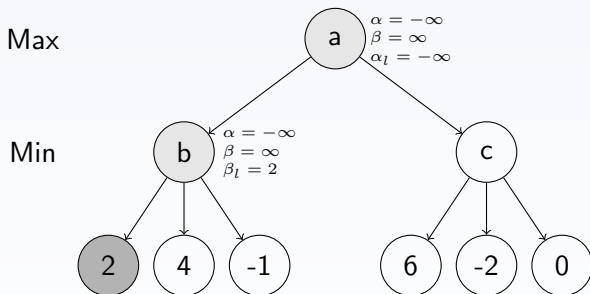
Alpha-Beta-Suche: Beispiel



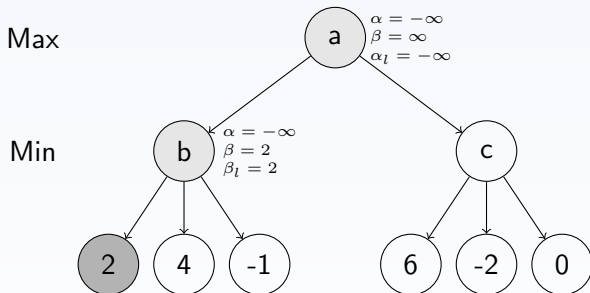
Alpha-Beta-Suche: Beispiel



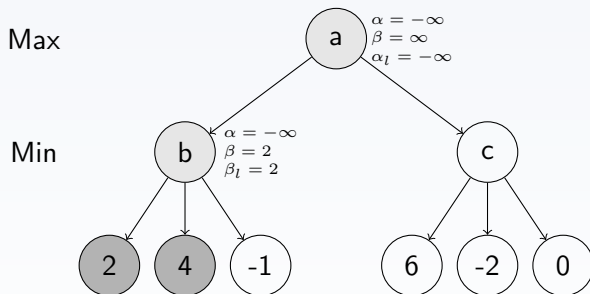
Alpha-Beta-Suche: Beispiel



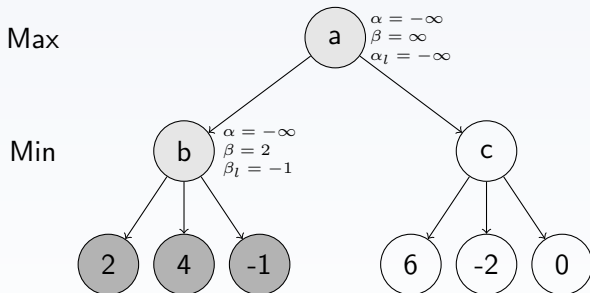
Alpha-Beta-Suche: Beispiel



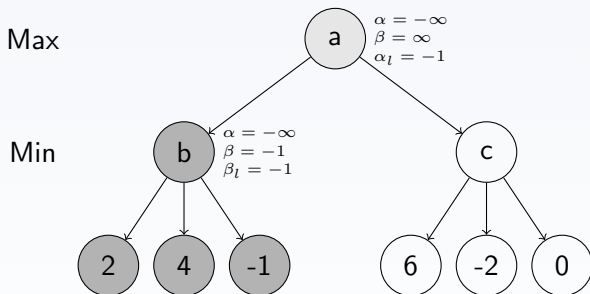
Alpha-Beta-Suche: Beispiel



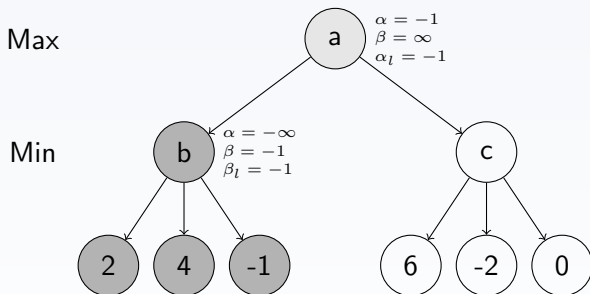
Alpha-Beta-Suche: Beispiel



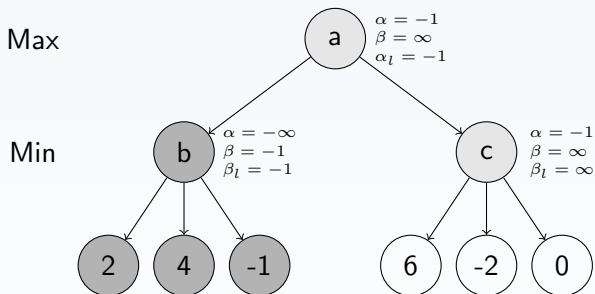
Alpha-Beta-Suche: Beispiel



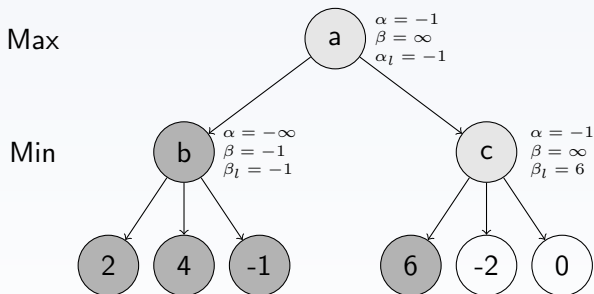
Alpha-Beta-Suche: Beispiel



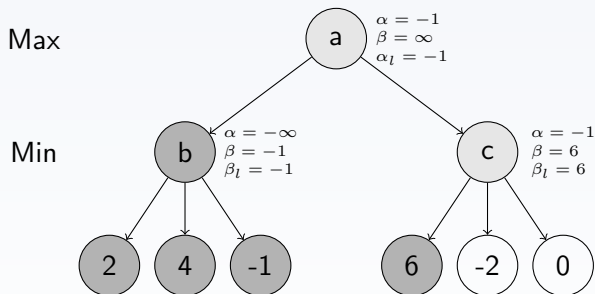
Alpha-Beta-Suche: Beispiel



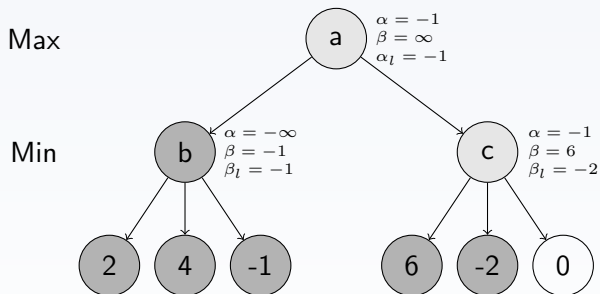
Alpha-Beta-Suche: Beispiel



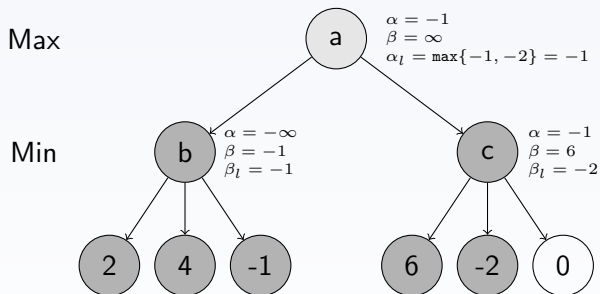
Alpha-Beta-Suche: Beispiel



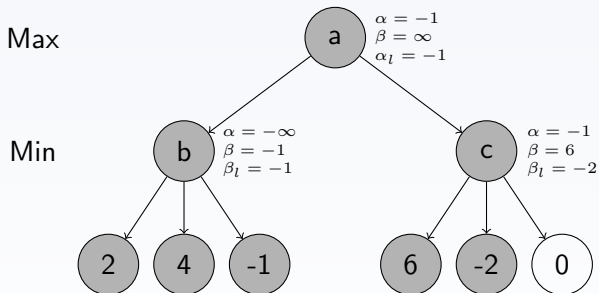
Alpha-Beta-Suche: Beispiel



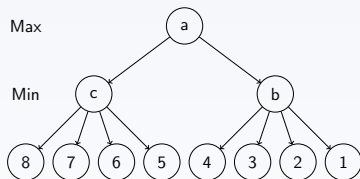
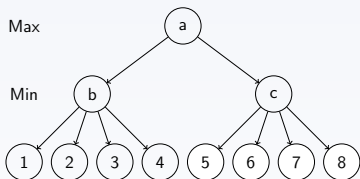
Alpha-Beta-Suche: Beispiel



Alpha-Beta-Suche: Beispiel

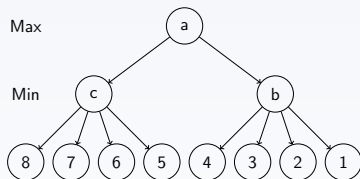
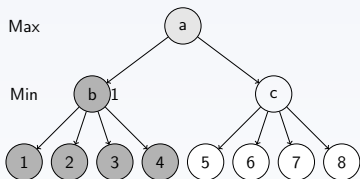


Reihenfolge der Nachfolger



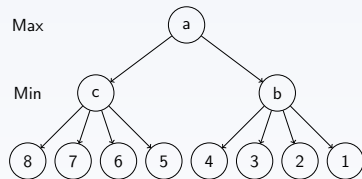
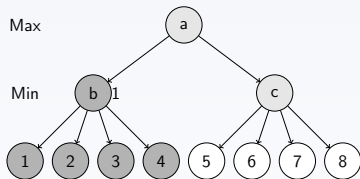
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



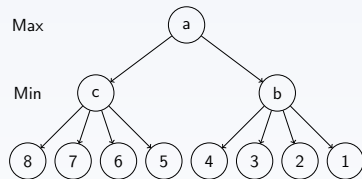
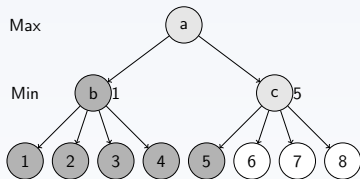
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



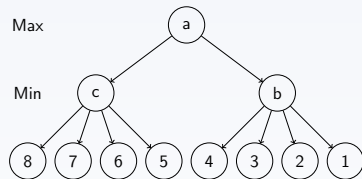
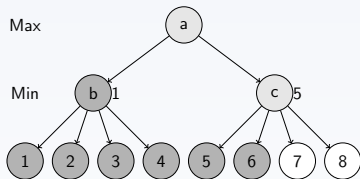
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



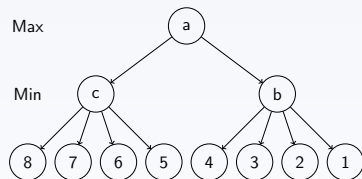
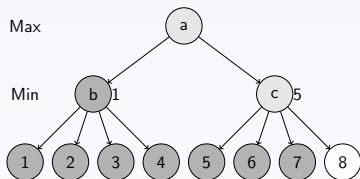
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



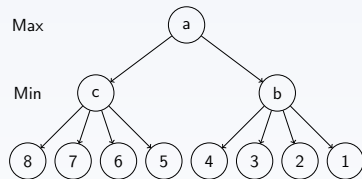
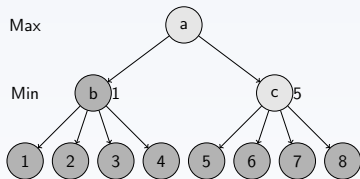
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



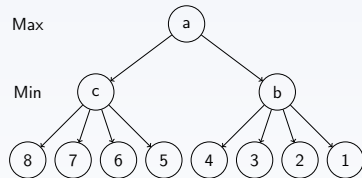
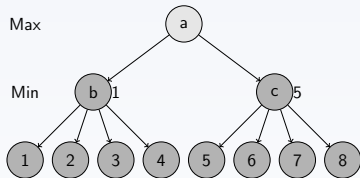
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



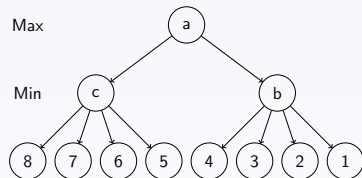
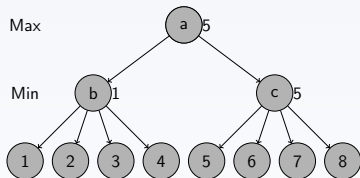
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



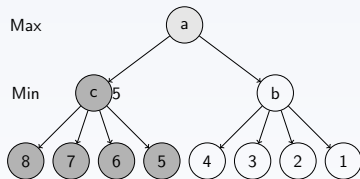
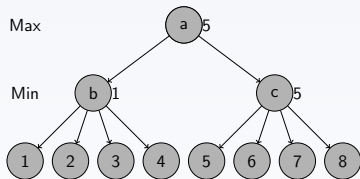
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



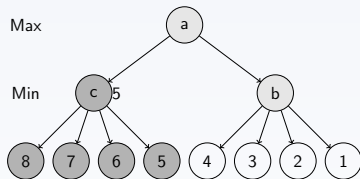
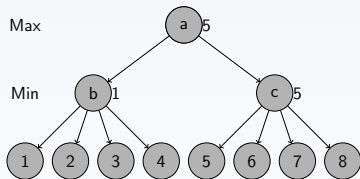
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



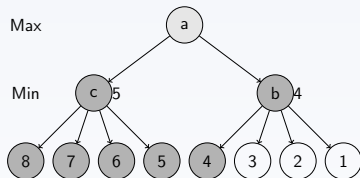
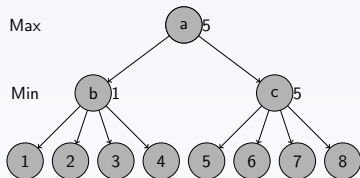
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



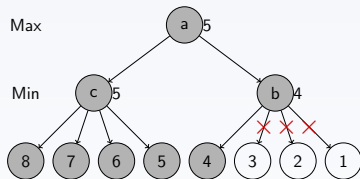
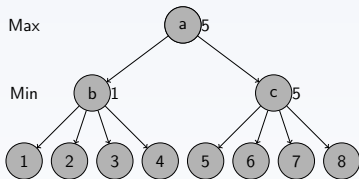
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



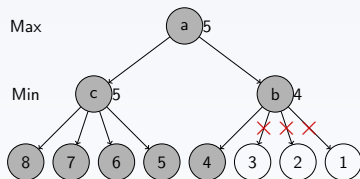
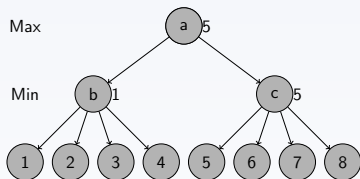
- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge

Reihenfolge der Nachfolger



- Beide Bäume sind gleich
- Außer: Nachfolger in umgekehrter Reihenfolge
- Reihenfolge bestimmt Güte von Alpha-Beta
- Abhilfe: Nachfolger vorsortieren
- Heuristik zum Sortieren verwenden

Eigenschaften Alpha-Beta

Zur Erinnerung:

- MinMax: Laufzeit $O(c^d)$ bei Tiefenschranke d und Verzweigungsrate c

Alpha-Beta-Suche:

- Worst-Case: Wie MinMax
- Best-Case $O(c^{\frac{d}{2}})$
- Bei guter Vorsortierung $O(c^{\frac{d}{2}})$ (Daumenregel)
- Bei zufälliger Vorsortierung $O(c^{\frac{3d}{4}})$

Bei guter Vorsortierung:

Bei gleichen Ressourcen kann man in mit **Alpha-Beta-Suche** **doppelt so tief** vorausschauen als bei MinMax-Suche!

Optimierungen

- Speicherung von bewerteten Stellungen
- Nutzung von Symmetrien

Probleme:

- Manchmal: Tiefere Suche schlechter als nicht so tiefe Suche
- **Horizonteffekt**: Durch Tiefenschranke wird genau der **gute** Zug übersehen

- Falls bei Suche in Tiefe d ein Zug am Knoten K besonders gut (Maximierer) ist:
- besonders gut: alle Brüder von K haben schlechtere Bewertung
- Dann Suche K mit größerer Tiefe ab

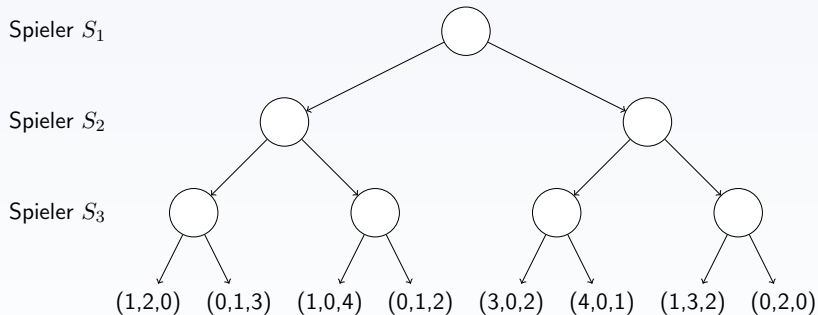
Tiefere Suche ist auch ratsam bei **Zugzwang**, denn

- Erhöht den Suchhorizont
- **ohne** Vergrößerung des Suchraums

Mehr als 2 Spieler

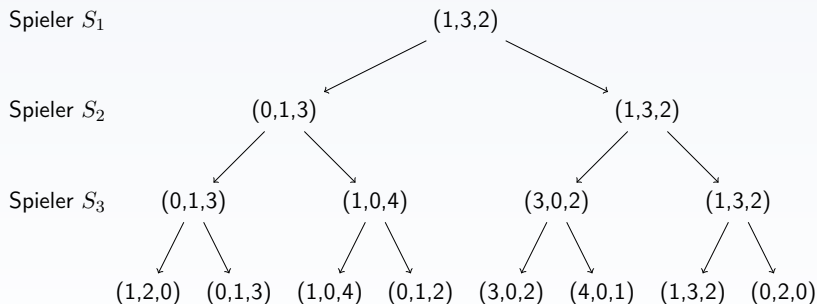
- Wir erläutern nur die Probleme (keine Lösung)
- Erkenntnis: Probleme sind wesentlich schwieriger
- Vereinfachung: Genau 3 Spieler
- Blattbewertung: Statt Wert, nun Drei-Tupel (A, B, C) :
Ergebnis des jeweiligen Spielers

3-Spieler-Baum



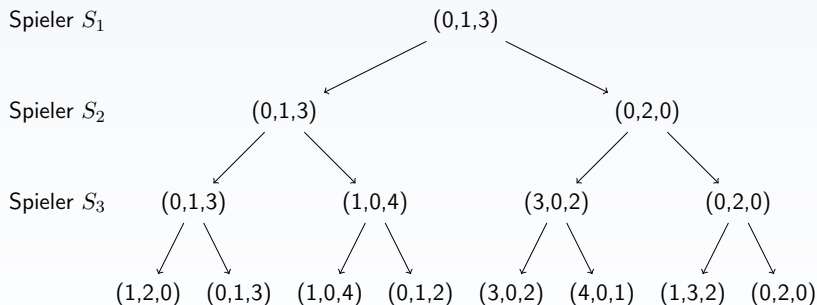
3-Spieler-Baum: Bewertung (1)

Strategie: Jeder **maximiert** sein eigenes Ergebnis



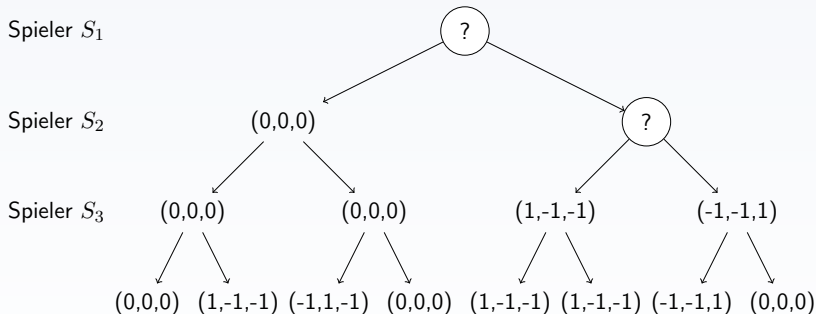
3-Spieler-Baum: Bewertung (1)

Strategie: Spieler 1 und Spieler 3 **verbünden sich**
Ziel: **Minimiere** Ergebnis von Spieler 2



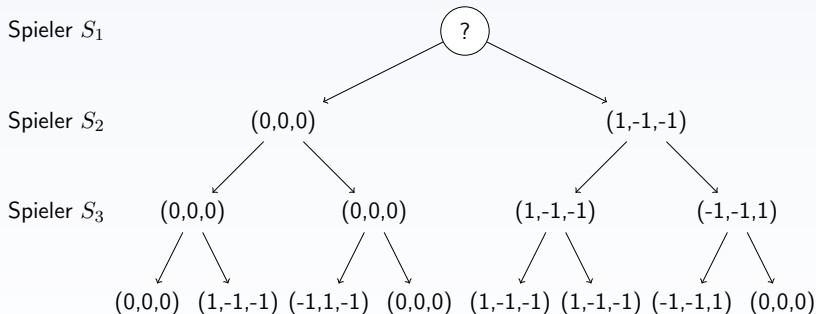
Nicht-eindeutige Strategie

Strategie: Jeder **maximiert** sein eigenes Ergebnis



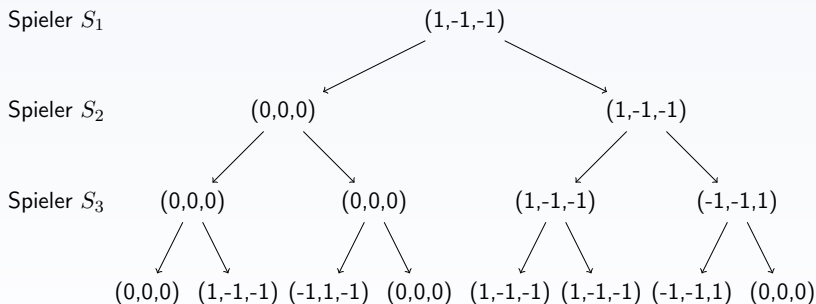
Nicht-eindeutige Strategie

Strategie: Jeder **maximiert** sein eigenes Ergebnis



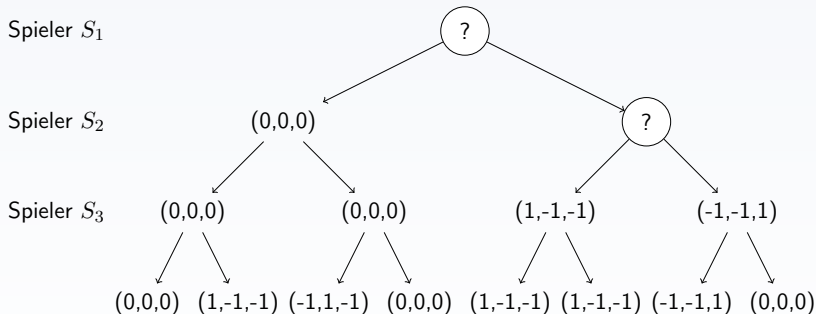
Nicht-eindeutige Strategie

Strategie: Jeder **maximiert** sein eigenes Ergebnis



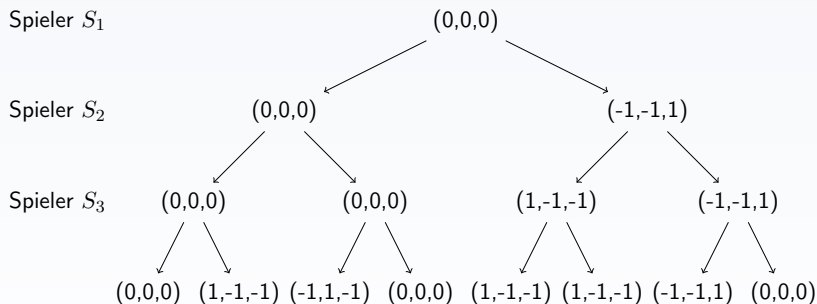
Nicht-eindeutige Strategie

Strategie: Jeder **maximiert** sein eigenes Ergebnis



Nicht-eindeutige Strategie

Strategie: Jeder **maximiert** sein eigenes Ergebnis



Spiele mit Zufall

- Wir betrachten nur 2-Spieler-Spiele
- Mit Zufall: Z.B. Würfeln, Kartenmischen, etc.

Naiver Ansatz:

Verwende MinMax (bzw. $\alpha - \beta$) über alle Möglichkeiten

Besser:

Benutze die Wahrscheinlichkeit und den Erwartungswert

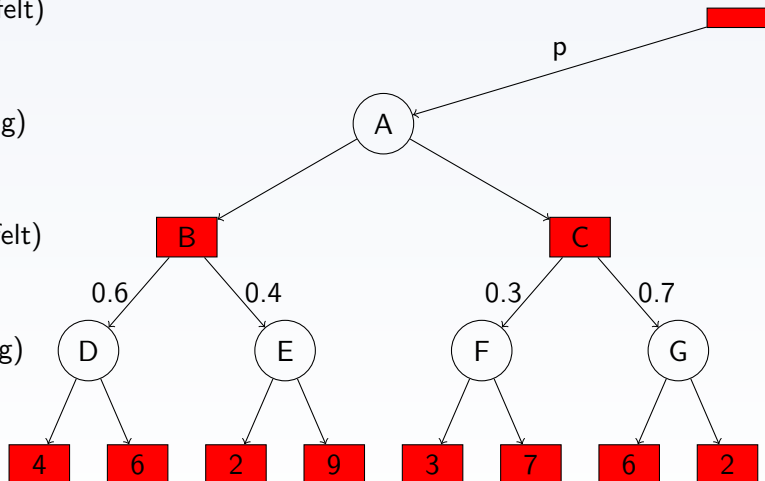
Spielbaum mit Zufallsknoten

Max (würfelt)

Max (Zug)

Min (würfelt)

Min (Zug)



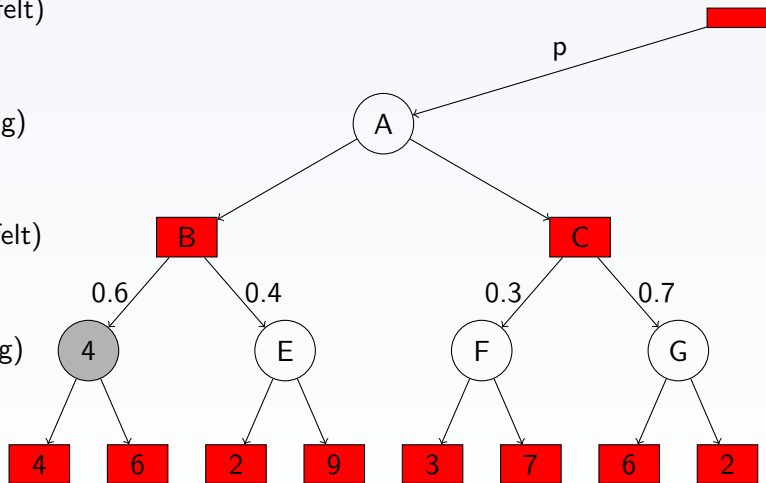
Spielbaum mit Zufallsknoten

Max (würfelt)

Max (Zug)

Min (würfelt)

Min (Zug)



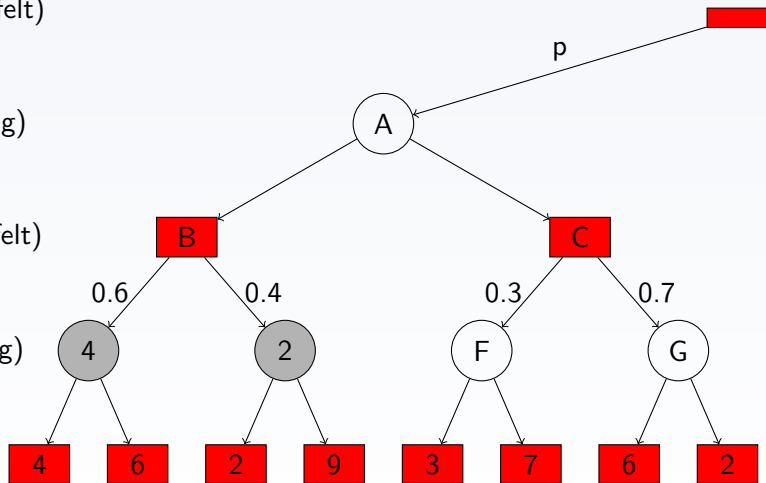
Spielbaum mit Zufallsknoten

Max (würfelt)

Max (Zug)

Min (würfelt)

Min (Zug)



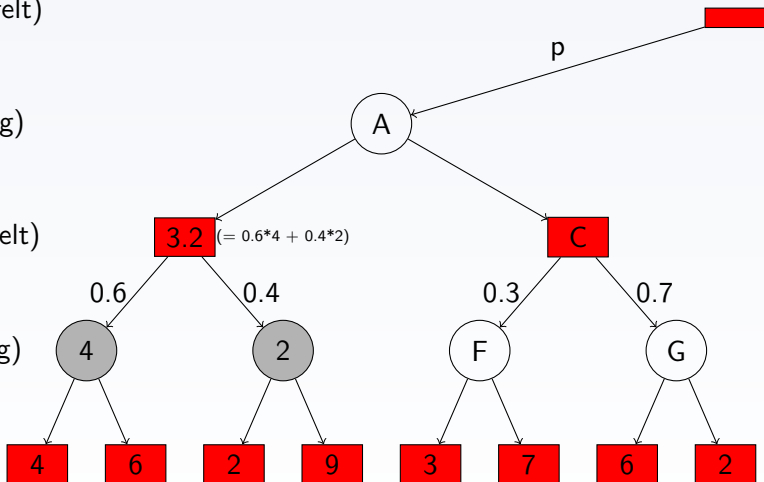
Spielbaum mit Zufallsknoten

Max (würfelt)

Max (Zug)

Min (würfelt)

Min (Zug)



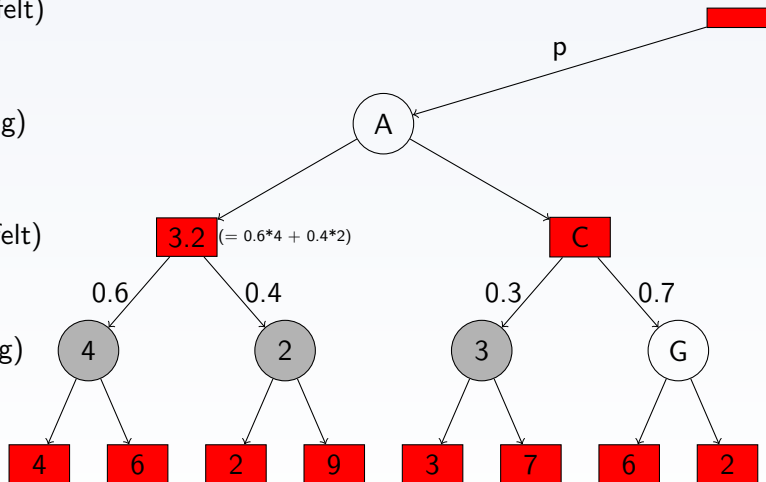
Spielbaum mit Zufallsknoten

Max (würfelt)

Max (Zug)

Min (würfelt)

Min (Zug)



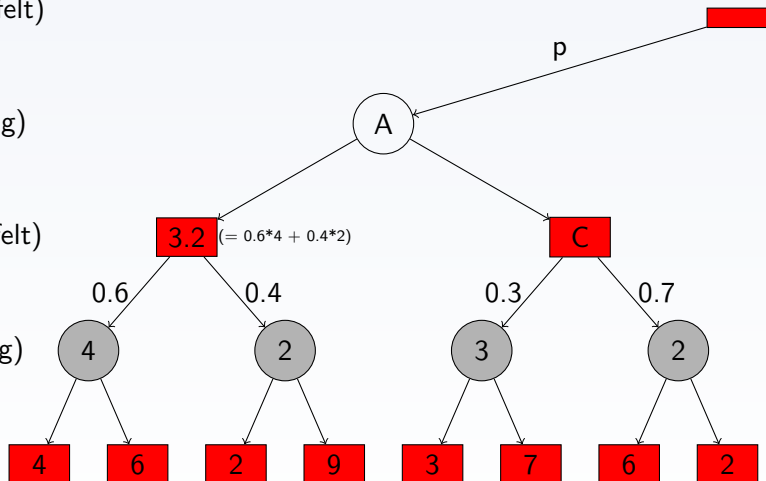
Spielbaum mit Zufallsknoten

Max (würfelt)

Max (Zug)

Min (würfelt)

Min (Zug)



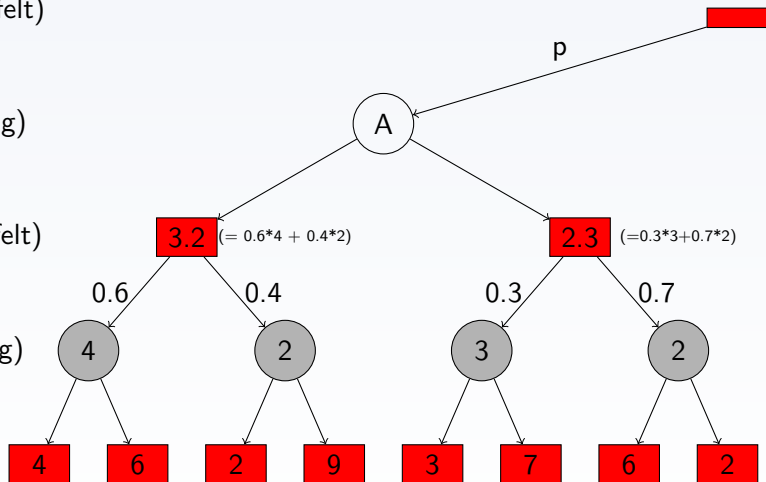
Spielbaum mit Zufallsknoten

Max (würfelt)

Max (Zug)

Min (würfelt)

Min (Zug)



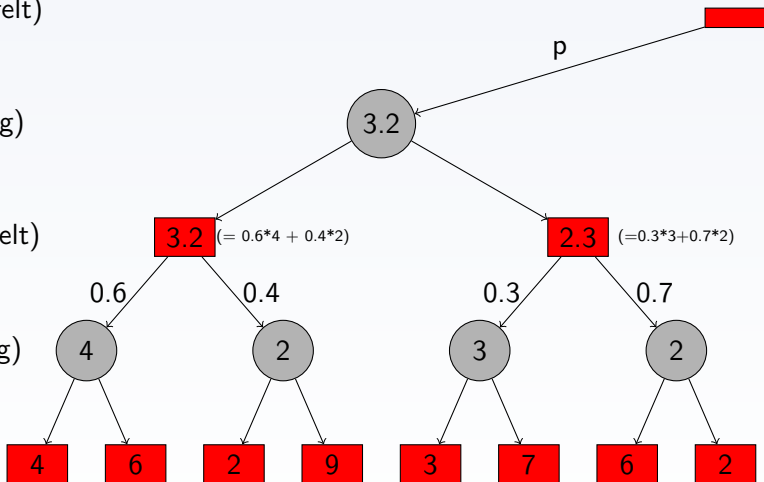
Spielbaum mit Zufallsknoten

Max (würfelt)

Max (Zug)

Min (würfelt)

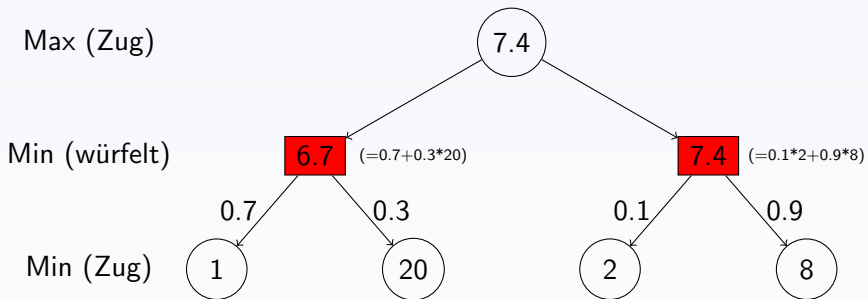
Min (Zug)



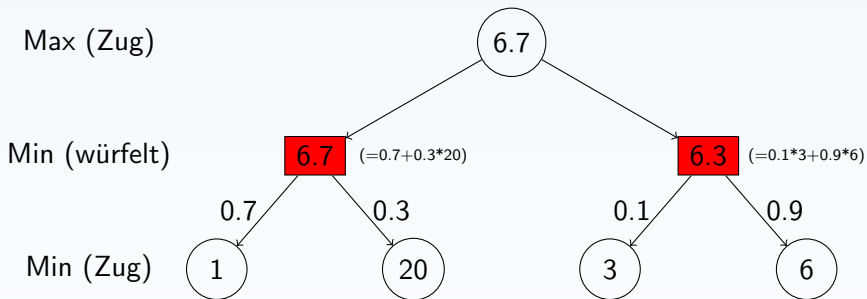
Satz

Es gilt: Wenn Wahrscheinlichkeit eine Rolle spielt, dann ist die Berechnung des Zuges mit dem besten Erwartungswert abhängig von den exakten Werten der Bewertungsfunktion; nicht nur von der relativen Ordnung auf den Situationen.

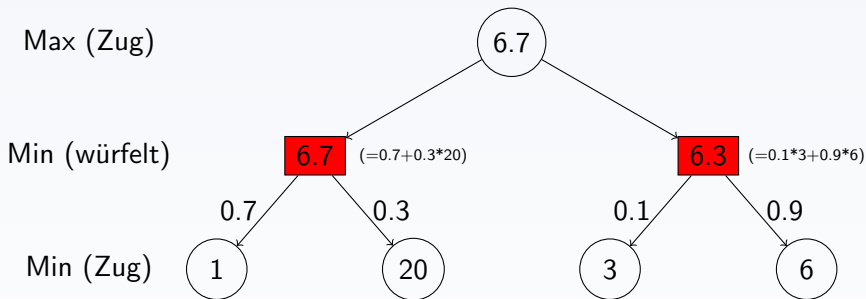
Beweis durch Gegenbeispiel



Beweis durch Gegenbeispiel



Beweis durch Gegenbeispiel



⇒ Die Bewertungsfunktion muss vorsichtig gewählt werden!