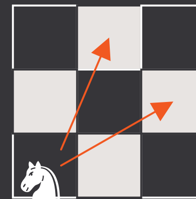
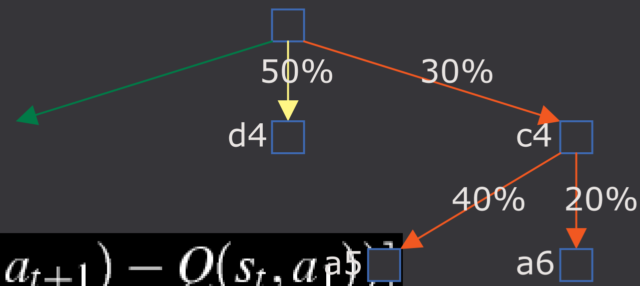
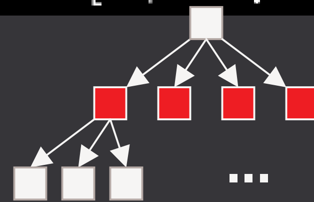
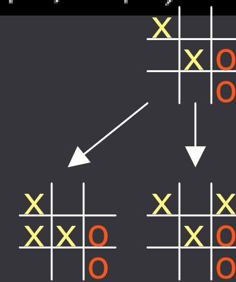
Reinforcement Learning in der Schachprogrammierung





$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + c * [r_{t+1} + \gamma * max_a(Q(s_{t+1},a_{t+1}) - Q(s_t,a_t))]$$





Marco Block

Übersicht

1. Einführung Schachprogrammierung (sehr kurz)
   - Struktur von Schachprogrammen verstehen
   - Problematik der Spielqualität (Koeffizientenwahl)

2. Reinforcement Learning (RL)
   - Konzept und Einordnung in KI

3. KnightCap
   - erstes RL-Schachprogramm (erstaunliche Erfolge)
   - Studienarbeit

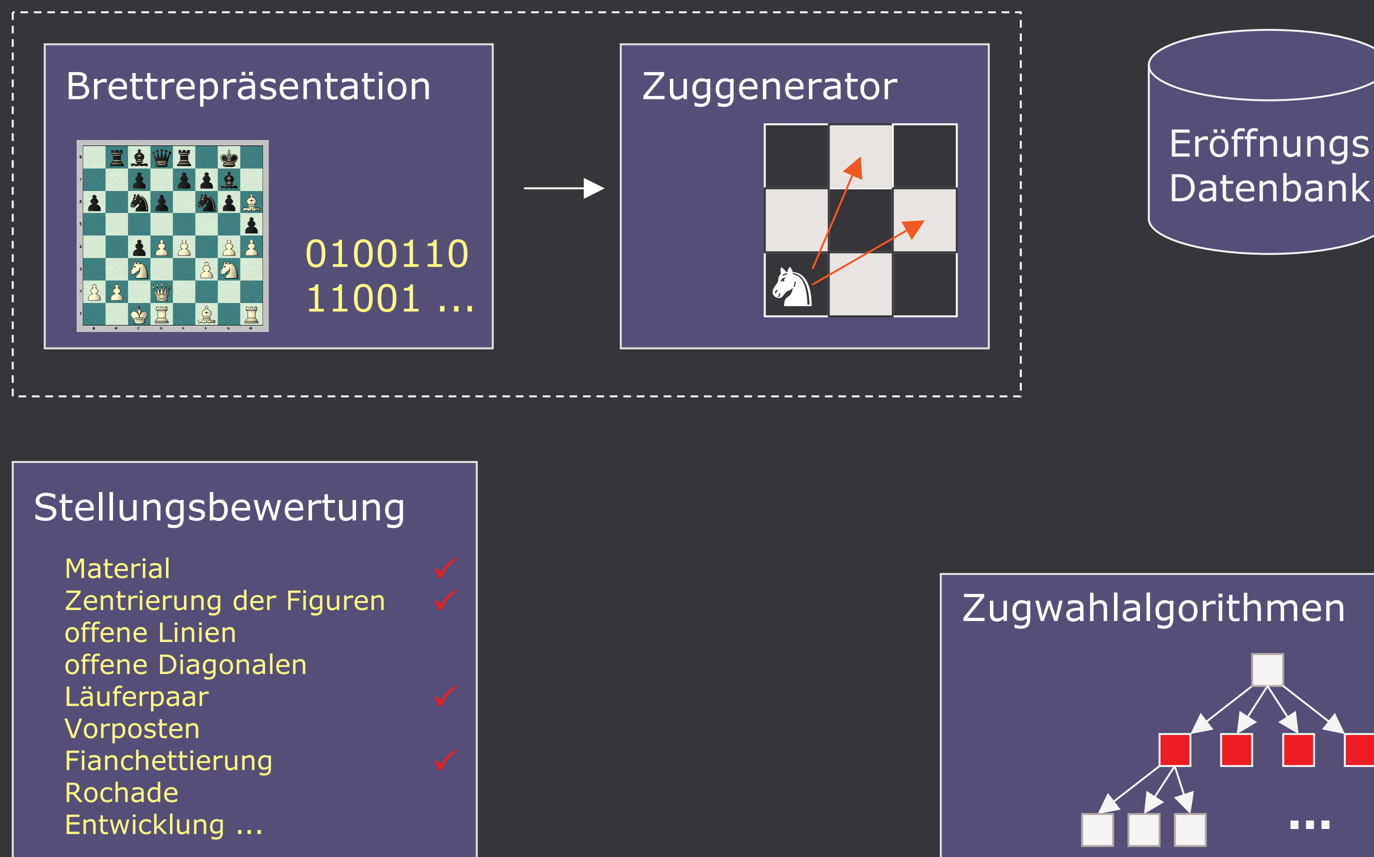
4. FUSC#
   - Open Source Projekt
   - FUSC# && RL == Kasparov ? ☺

Schachprogrammierung

1. Einführung Schachprogrammierung (sehr kurz)
   - Struktur von Schachprogrammen verstehen
     + Brettdarstellung
     + Evaluation
     + Zugwahl

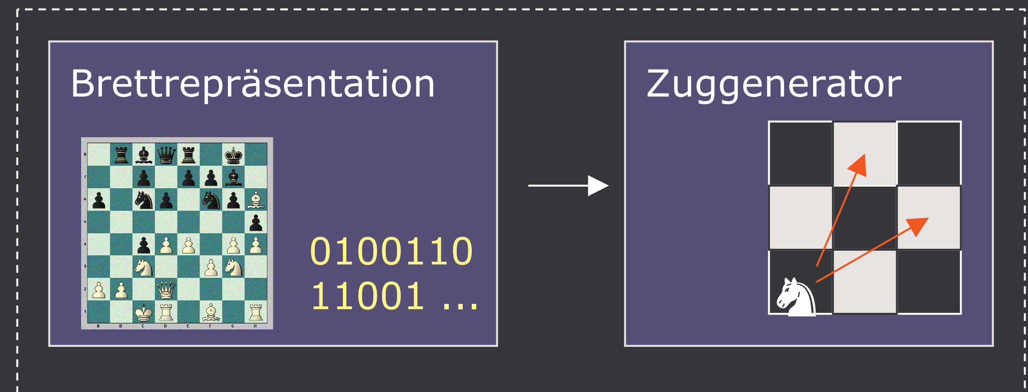
   - Problematik der Spielqualität (Koeffizientenwahl)

Komponenten



Brettrepräsentation

Brettrepräsentation



8x8-Brettdarstellung

-4	-2	-3	-5	-6	-3	-2	-4
-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
4	2	3	5	6	3	2	4

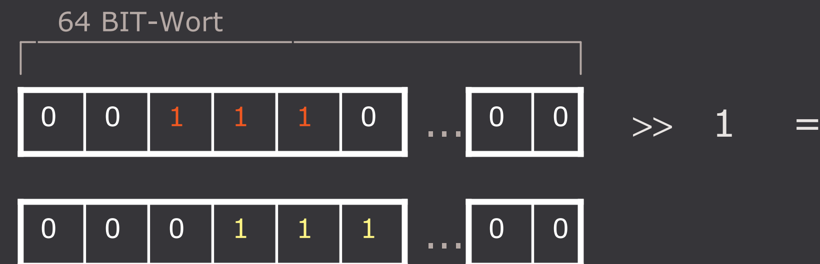
Brettmatrix durchlaufen

- Figur identifizieren
- mögliche Zugfelder betrachten und Regelsatz befolgen
- wenn Zug gültig in Zugliste speichern

BitBoards

BitBoarddarstellung

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0



+ Zuggenerierung sehr schnell
+ Evaluationsfaktoren leicht in BitMuster giessbar (Königssicherheit)
- keine „offiziellen" Standards

Evaluation

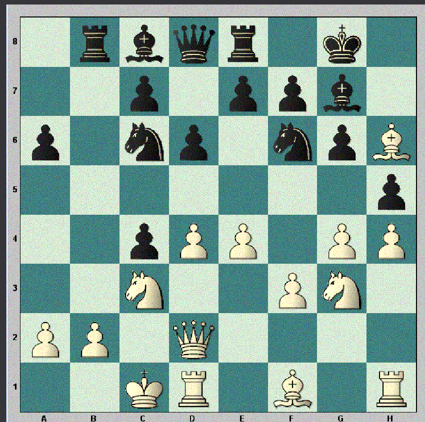
Stellungsbewertung

- Material ✓
- Zentrierung der Figuren ✓
- offene Linien
- offene Diagonalen
- Läuferpaar ✓
- Vorposten
- Fianchettierung ✓
- Rochade
- Entwicklung ...

Evaluation

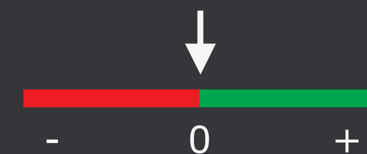
Stellungsbewertung

10 * Material(S) + 2 * Mobilität(S) + ...



$$\sum w_i * f_i(S)$$

Faktoren
- Material
- Mobilität
- gute/schlechte Figuren
- starke/schwache Felder
- Raum
- Königsstellung
- offene Linien/Diagonalen
- Läuferpaar
- Vorposten
- Rochade
- Bauernstruktur
- Läufer gegen Springer
- ...



Material

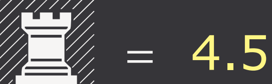
Material

- Figurenbewertungen (weisse Figurensumme - schwarze)
- verwendeten Daten von Kasparov vorgeschlagen

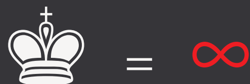
= 1.0

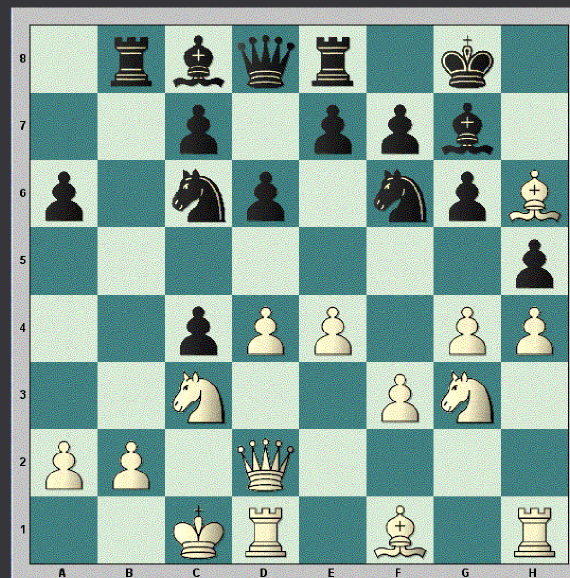
= 3.0

= 3.0

 = 4.5

= 9.0

 = ∞



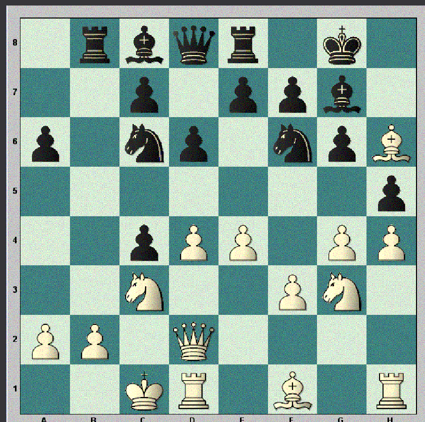
- 0 +

	Weiss	Schwarz
	7x1.0	7x1.0
	2x3.0	2x3.0
	2x3.0	2x3.0
	2x4.5	2x4.5
	1x9.0	1x9.0
	37.0	37.0
		0.0

Evaluation

Stellungsbewertung

10 * **0.0** + 2 * Mobilität(S) + ...

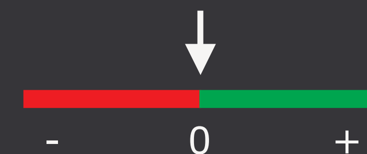


Faktoren
- Material
- Mobilität
- gute/schlechte Figuren
- starke/schwache Felder
- Raum
- Königsstellung
- offene Linien/Diagonalen
- Läuferpaar
- Vorposten
- Rochade
- Bauernstruktur
- Läufer gegen Springer
- ...

$$\sum w_i * f_i(S)$$

relativer Massstab (→ w_i)

konkrete Werte (→ $f_i(S)$)



Evaluation

Problem:

Welche Konfiguration ist die richtige?
Gibt es überhaupt eine richtig Konfiguration?
=> Falls JA, dann müsste schachliches Wissen als eine *lineare Funktion* darstellbar sein!
(sehr unwahrscheinlich ;))

10 * Material(S) + 2 * Mobilität(S) + ...

11 * Material(S) + 2.5 * Mobilität(S) + ...

???

Lösung (bisher):

Für professionelle Schachprogramme
=> Grossmeister engagiert, kritisiert Bewertung

Lösung (KnightCap):

Temporale Differenz auf Blattknoten

TD & Brettspiele V

TD(λ)-Algorithmus dafür da, den Parametersatz $w \in \Re^k$ zu lernen.

Also den Vektor so zu approximieren, dass er gegen die optimale Strategie konvergiert.

Evaluation

Stellungsbewertungsfaktoren

BishopPair
Knight_Outpost
Supported_Knight_Outpost
Connected_Rooks
Opposite_Bishops
Opening_King_Advance
King_Proximity
Blocked_Knight
Draw_Value
No_Material
Bishop_XRay
Rook_Pos
Pos_Base
Pos_Queenside
Bishop_Mobility
Queen_Mobility
Knight_SMobility
Rook_SMobility
King_SMobility
Threat
Overloaded_Penalty
Q_King_Attack_Opponent
NoQ_King_attack_Opponent
NoQueen_File_Safty
Attack_Value
Unsupported_Pawn
Passed_Pawn_Control
Doubled_Pawn
Odd_Bishop_Pawn_Pos

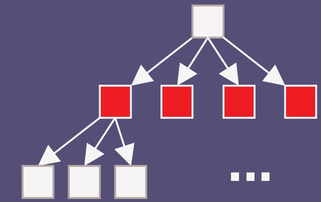
King_Passed_Pawn_Supported
Passed_Pawn_Rook_Supported
Blocked_EPawn
Pawn_Advance
King_Passed_Pawn_Defence
Pawn_Defence
Mega_Weak_Pawn
Castle_Bonus
Bishop_Outpost
Supported_Bishop_Outpost
Seventh_Rank_Rooks
Early_Queen_Movement
Mid_King_Advance
Trapped_Step
Useless_Piece
Near_Draw_Value
Mating_Positions
Ending_King_Pos
Knight_Pos
Pos_Kingside
Knight_Mobility
Rook_Mobility
King_Mobility
Bishop_SMobility
Queen_SMobility
Piece_Values
Opponents_Threat
Q_King_Attack_Computer
NoQ_King_Attack_Computer

Queen_File_Safty
Piece_Trade_Bonus
Pawn_Trade_Bonus
Adjacent_Pawn
Unstoppable_Pawn
Weak_Pawn
Blocked_Pawn
Passed_Pawn_Rook_Attack
Blocked_DPawn
Pawn_Advance
Pawn_Advance2
Pawn_Pos
Isolated_Pawn
Weak_Pawn_Attack_Value

... und viele viele mehr !

⇒ Evaluation sehr aufwendig
und demnach „teuer“

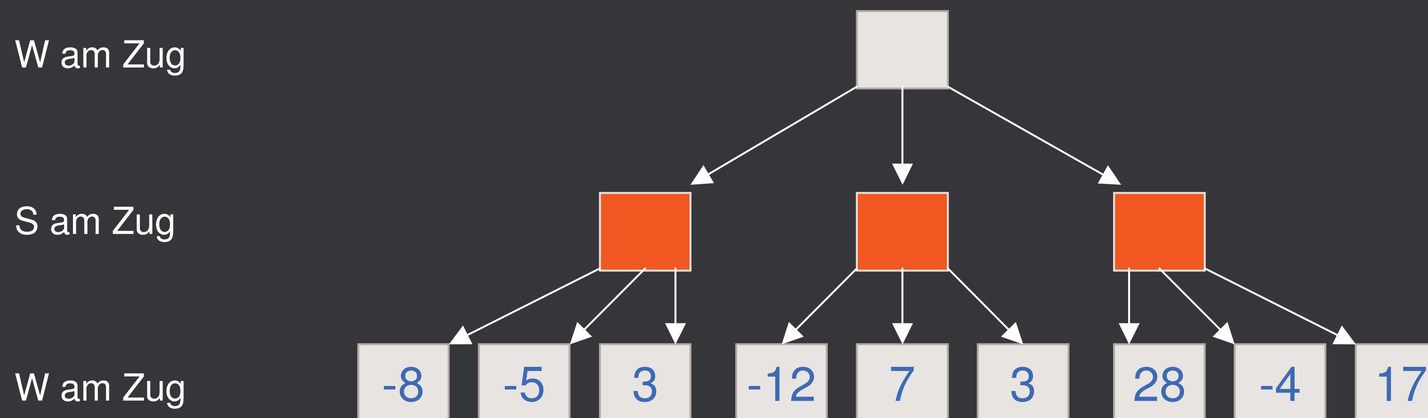
Algorithmen



Brute-Force-Methode

Shannon-A-Strategie (Brute-Force)

- Tiefensuche bis zu einer vorgegebenen festen Suchtiefe
- Minimax-Idee: Weiß maximiert, Schwarz minimiert Nachfolger

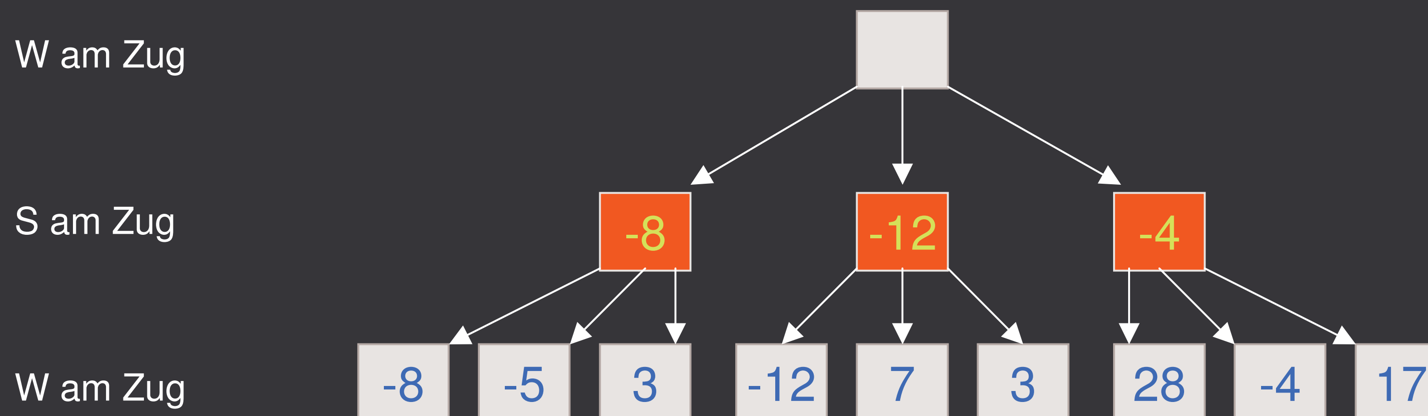


Bewertungsroutine an den Blättern

Brute-Force-Methode

Shannon-A-Strategie (Brute-Force)

- Tiefensuche bis zu einer vorgegebenen festen Suchtiefe
- Minimax-Idee: Weiß maximiert, Schwarz minimiert Nachfolger



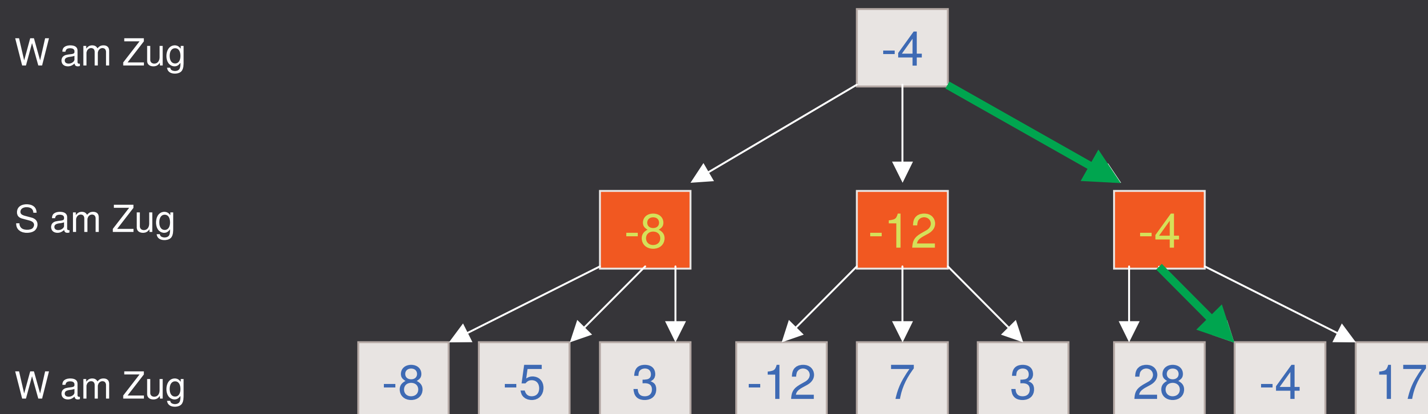
Bewertungsroutine an den Blättern

Brute-Force-Methode

Shannon-A-Strategie (Brute-Force)

- Tiefensuche bis zu einer vorgegebenen festen Suchtiefe

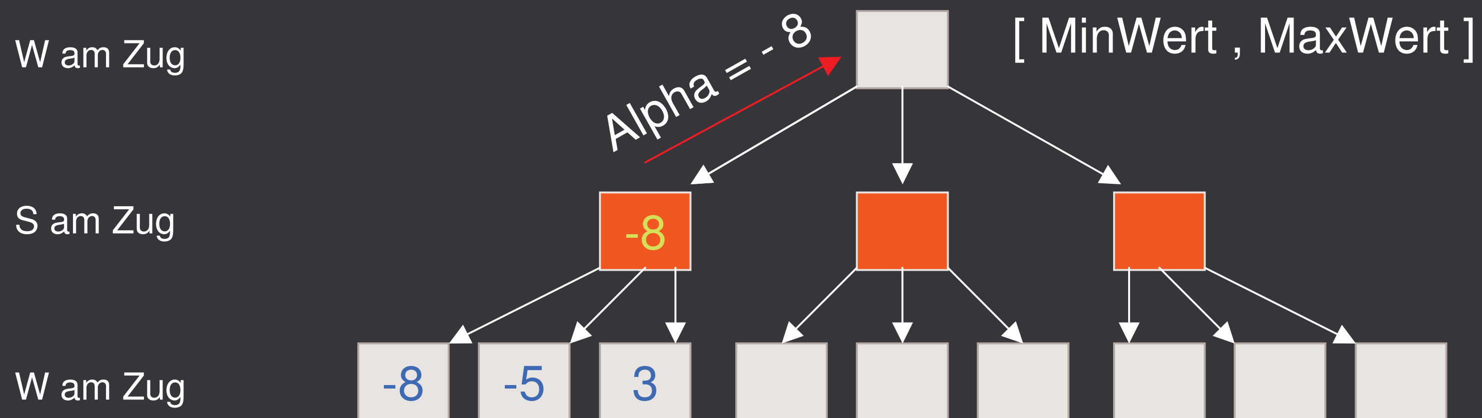
- Minimax-Idee: Weiß maximiert, Schwarz minimiert Nachfolger



Alpha-Beta-Algorithmus

Alpha-Beta-Algorithmus

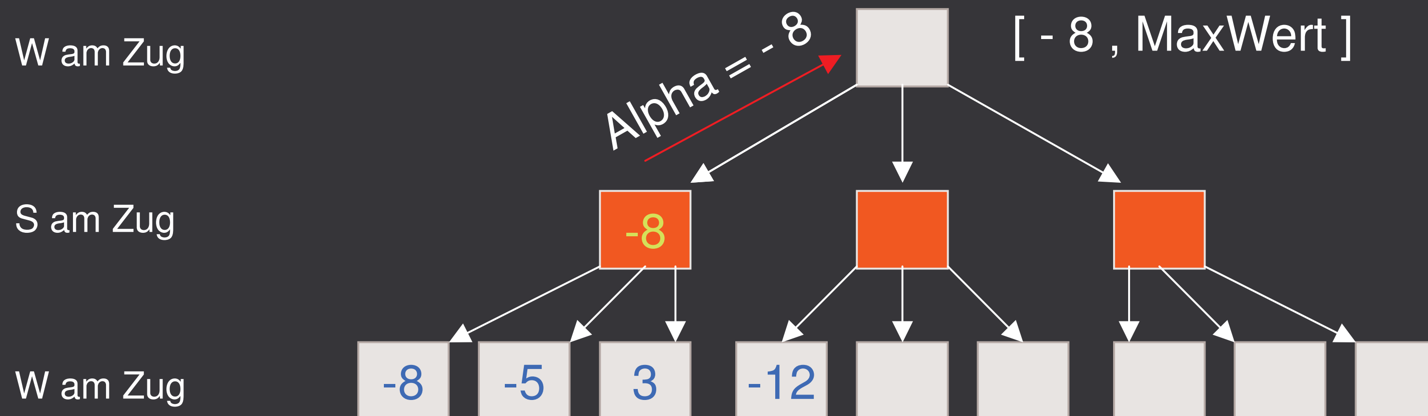
- Abschneiden von unwichtigen Ästen des Spielbaumes
- Alpha-/Beta-Werte bilden Erwartungsfenster



Alpha-Beta-Algorithmus

Alpha-Beta-Algorithmus

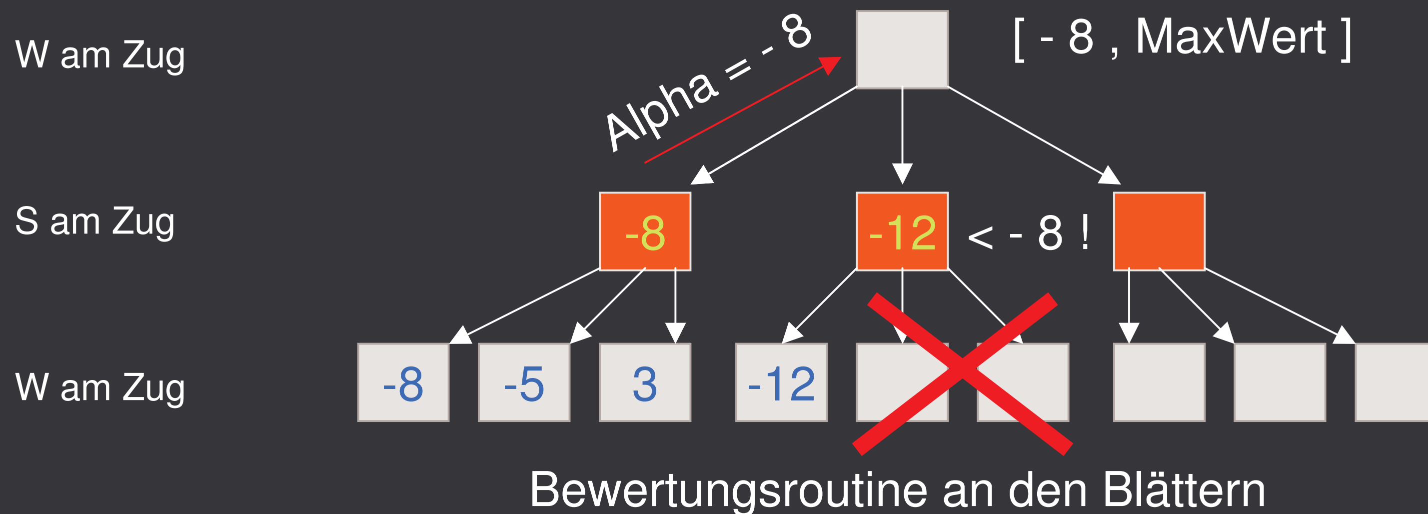
- Abschneiden von unwichtigen Ästen des Spielbaumes
- Alpha-/Beta-Werte bilden Erwartungsfenster



Alpha-Beta-Algorithmus

Alpha-Beta-Algorithmus

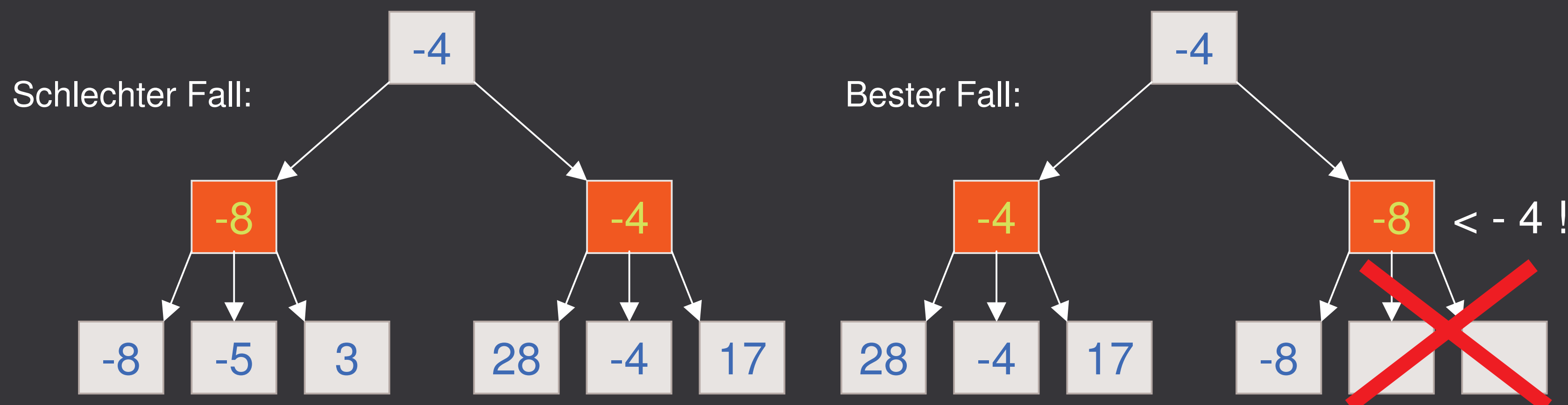
- Abschneiden von unwichtigen Ästen des Spielbaumes
- Alpha-/Beta-Werte bilden Erwartungsfenster



Zugsortierung

Zugsortierung

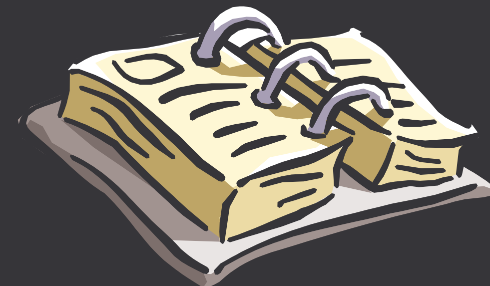
- iteratives Suchverfahren
- Transpositions-, Killerzug- und Historytabellen



diverse Programmiertricks

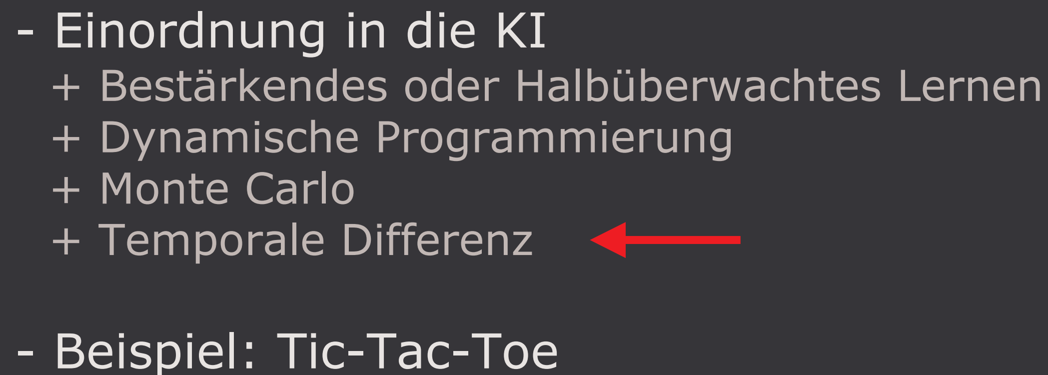
- selbstlernendes Eröffnungsbuch
- Hashtables
- heuristische Zugsortierungen
- KillerMoves
- parallele Suche
- Ruhesuche
- iterative Suche
- NullMoves
- weitere Heuristiken

...



Reinforcement Learning

2.Reinforcement Learning (RL)



Lernstrategie I

1. überwachtes Lernen
   - *„Ziel wird vom Lehrer vorgegeben"*
   - Neuronale Netze (BackPropagation)

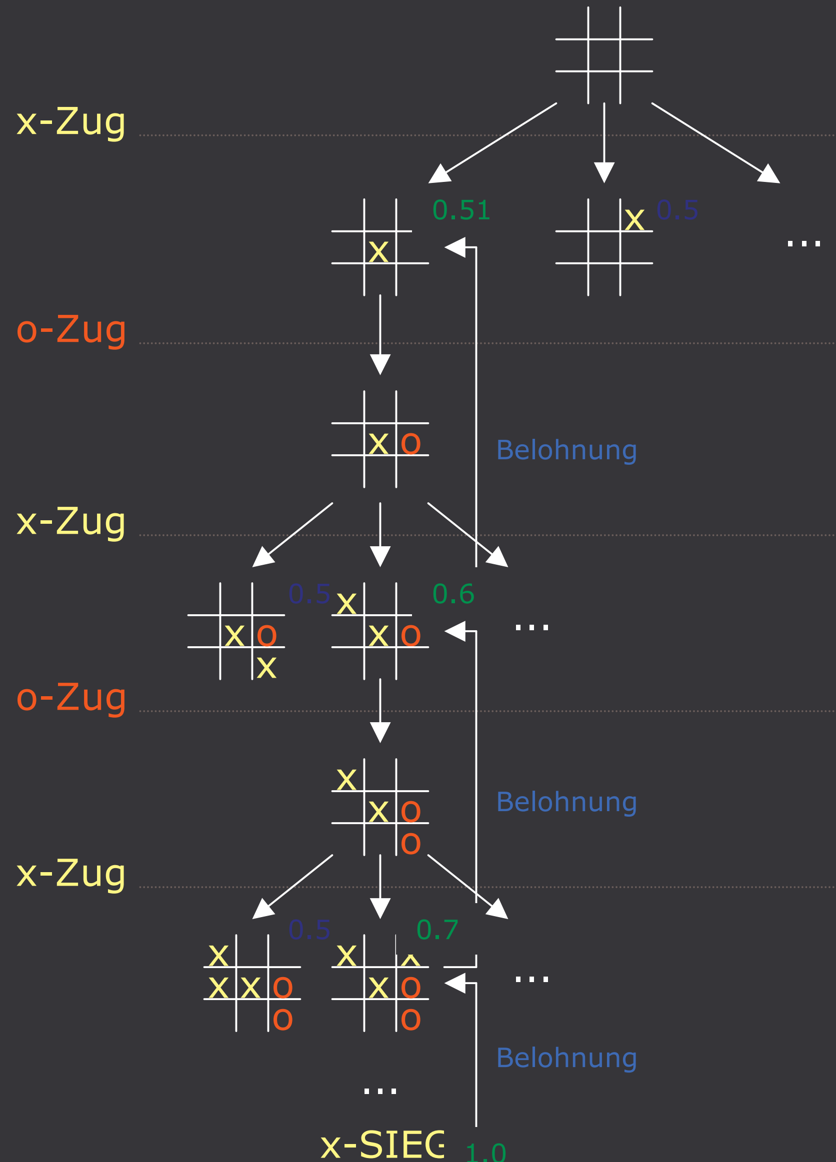
2. unüberwachtes Lernen
   - *„System klassifiziert nach eigenem Ermessen"*
   - EM, LBG

3. Reinforcement Learning
   - *„System (Agent) findet durch Ausprobieren und Rückmeldung (Reinforcement-Signal) aus der Umwelt eine optimale Strategie, um Ziele innerhalb seiner Umgebung zu erreichen"*
   - „halbüberwachtes Lernen" [Zhang]
   - Dynamische Programmierung, Monte Carlo, Temporale Differenz

Lernstrategie II

Agenten, die immer oder niemals weiterforschen, werden stets scheitern.

Beispiel: Tic Tac Toe



1. Tabelle V(s) erzeugen mit
   (Zustand s → Bewertung)

   Wahrscheinlichkeit zu gewinnen

Gewinnstrategie
1 Sieg, 0 Niederlage, 0.5 Remis

2. Es werden viele Partien gespielt
   - meistens besten Zug wählen
   - gelegentlich aber experimentieren! (zufällige Auswahl)
   - während des Spiels wird Wertefunktion verändert (Belohnung)
   - vorheriger Zustand verschiebt sich etwas in Richtung des nachfolgenden

$$V(s_t) \rightarrow V(s_t) + alpha * [V(s_{t+1} - V(s_t))]$$

3.KnightCap

- erstes RL-Schachprogramm (Studienarbeitauszug)
  + Erfolg
  + Struktur

Historie

KnightCap

unkommentiertes, kryptisches C

„Open Source Projekt" von
Jonathan Baxter, Andrew Tridgell und Lex Weaver

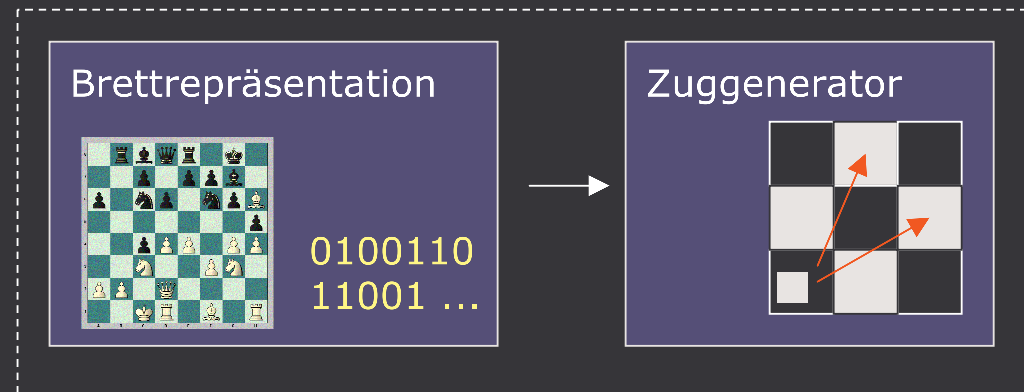
Reinforcement Learning eingesetzt
⇒ Evaluations-Koeffizienten „getunt"

Motivation: Tesauros TD-Gammon schlägt Weltmeister

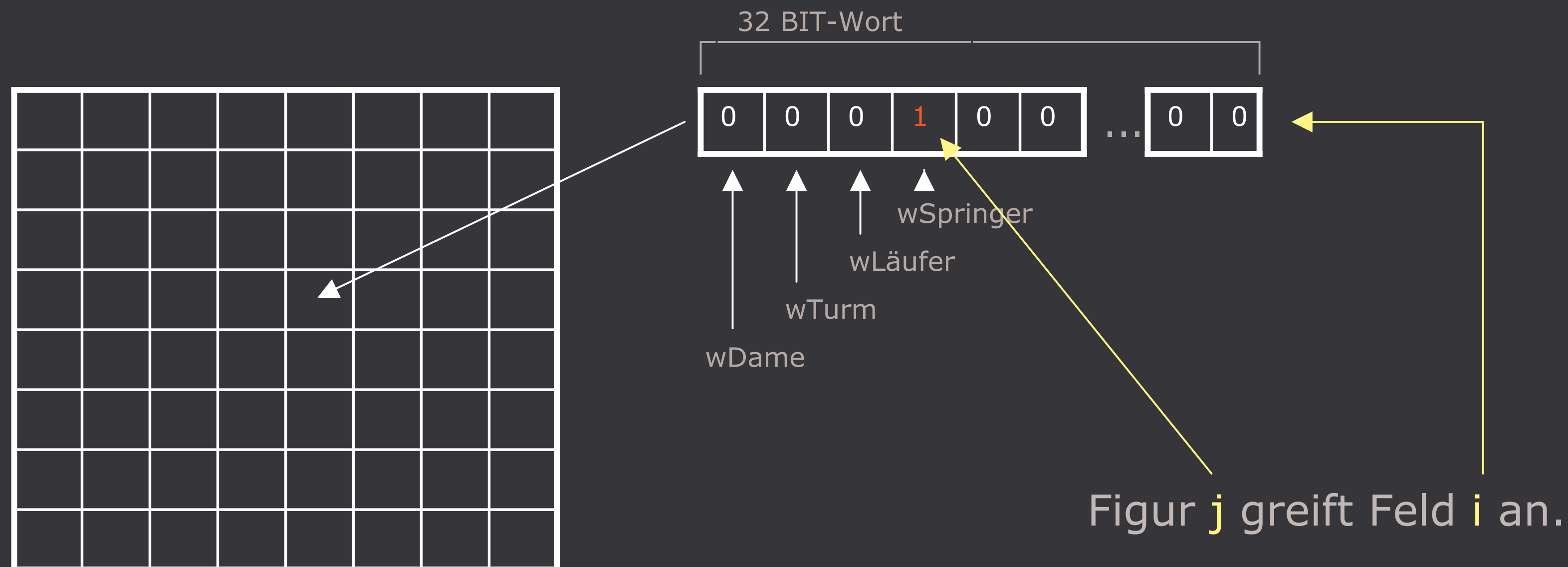
Startrating von 1650 in 3 Tagen (308 Spiele) auf 2150

+ Eröffnungsbuch
+ weitere Programmierraffinessen
⇒ KnightCap hat ein Niveau von über 2500 (max 2575)

Struktur

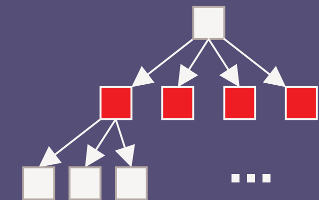


Brettdarstellung:
ToPiecesBoard



Figur j greift Feld i an.

Struktur



Zugwahlalgorithmus:

- MTD(f) = weiterentwickelter AlphaBeta-Algorithmus mit minimalem Fenster

- einige Heuristiken (Killerheuristik, ...)

Struktur

Stellungsbewertung

- Material ✓
- Zentrierung der Figuren ✓
- offene Linien
- offene Diagonalen
- Läuferpaar ✓
- Vorposten
- Fianchettierung ✓
- Rochade
- Entwicklung ...

- KnightCap
- FUSC# V1.09

Stand: August 2003

Stellungsbewertungsfaktoren:

BishopPair
Knight_Outpost
Supported_Knight_Outpost
Connected_Rooks
Opposite_Bishops
Opening_King_Advance
King_Proximity
Blocked_Knight
Draw_Value
No_Material
Bishop_XRay
Rook_Pos
Pos_Base
Pos_Queenside
Bishop_Mobility
Queen_Mobility
Knight_SMobility
Rook_SMobility
King_SMobility
Threat
Overloaded_Penalty
Q_King_Attack_Opponent
NoQ_King_attack_Opponent
NoQueen_File_Safty
Attack_Value
Unsupported_Pawn
Passed_Pawn_Control
Doubled_Pawn
Odd_Bishop_Pawn_Pos

King_Passed_Pawn_Supported
Passed_Pawn_Rook_Supported
Blocked_EPawn
Pawn_Advance
King_Passed_Pawn_Defence
Pawn_Defence
Mega_Weak_Pawn
Castle_Bonus
Bishop_Outpost
Supported_Bishop_Outpost
Seventh_Rank_Rooks
Early_Queen_Movement
Mid_King_Advance
Trapped_Step
Useless_Piece
Near_Draw_Value
Mating_Positions
Ending_King_Pos
Knight_Pos
Pos_Kingside
Knight_Mobility
Rook_Mobility
King_Mobility
Bishop_SMobility
Queen_SMobility
Piece_Values
Opponents_Threat
Q_King_Attack_Computer
NoQ_King_Attack_Computer

Queen_File_Safty
Piece_Trade_Bonus
Pawn_Trade_Bonus
Adjacent_Pawn
Unstoppable_Pawn
Weak_Pawn
Blocked_Pawn
Passed_Pawn_Rook_Attack
Blocked_DPawn
Pawn_Advance
Pawn_Advance2
Pawn_Pos
Isolated_Pawn
Weak_Pawn_Attack_Value

Struktur

Besonderheiten:

- Nullmoves
- Hashtables
- Asymetrien
- kann auf Parallelrechner betrieben werden
- 4 Stellungstypen: *Eröffnung*, *Mittelspiel*, *Endspiel*, *Mattstellungen*
- evolutionäres Eröffnungsbuch

FUSC#-Projekt

4.FUSC#

- RL goes to Fusc#
  + Einbettung von RL (KnightCap)
  + KnightCap-Experimente
  + Fusc#-Experimente

- Diplomarbeit
  + Erweiterung: Stellungsklassifikator

Wiederholung

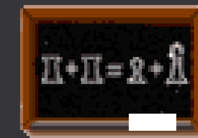
- Problematik: Evaluierung einer Stellung

- Reinforcement Learning
  + *„... durch Ausprobieren und Rückmeldung der Umwelt"*

- Backgammon sehr gut für TD(λ) geeignet
  + kleine Stellungsänderung $\Rightarrow$ kleine Bewertungsveränderung
  + komplexe Evaluationsfunktion möglich

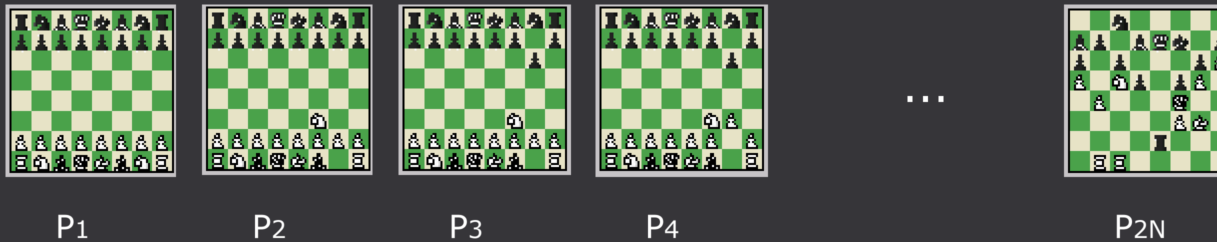
- Schach eigentlich nicht für TD(λ) geeignet
  + kleine Stellungsänderung $\Rightarrow$ grosse Bewertungsveränderung
  + schnelle, lineare Bewertungsfunktion nötig (MinMax)
  + TD-Leaf(λ) scheint eine Lösung zu sein
    (Baxter, Tridgell, Weaver -> KnightCap)

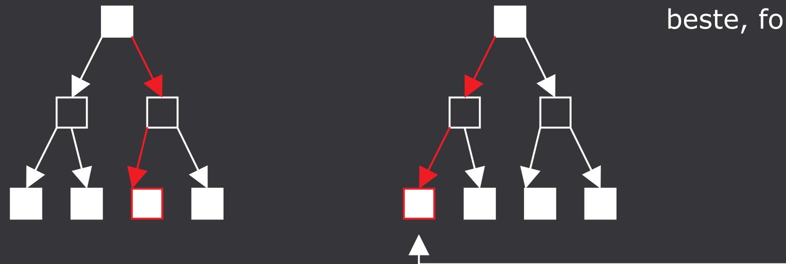
- Ziel: Evaluationsvektor w lernen



Daten vorbereiten

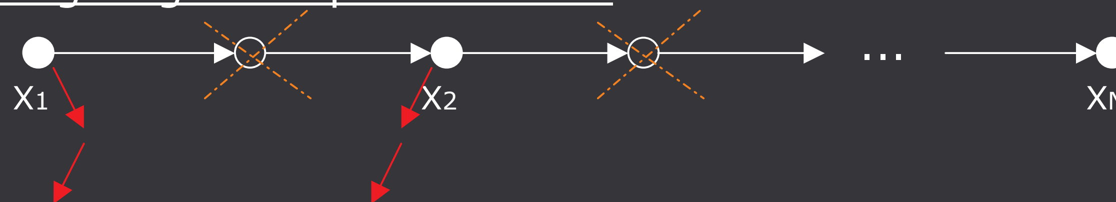
Partie: Fusch–Internetgegner 0-1 (Evaluationsvektor w wird benutzt)





Alpha-Beta als Zugwahlalgorithmus, Hauptvariante wird gespeichert

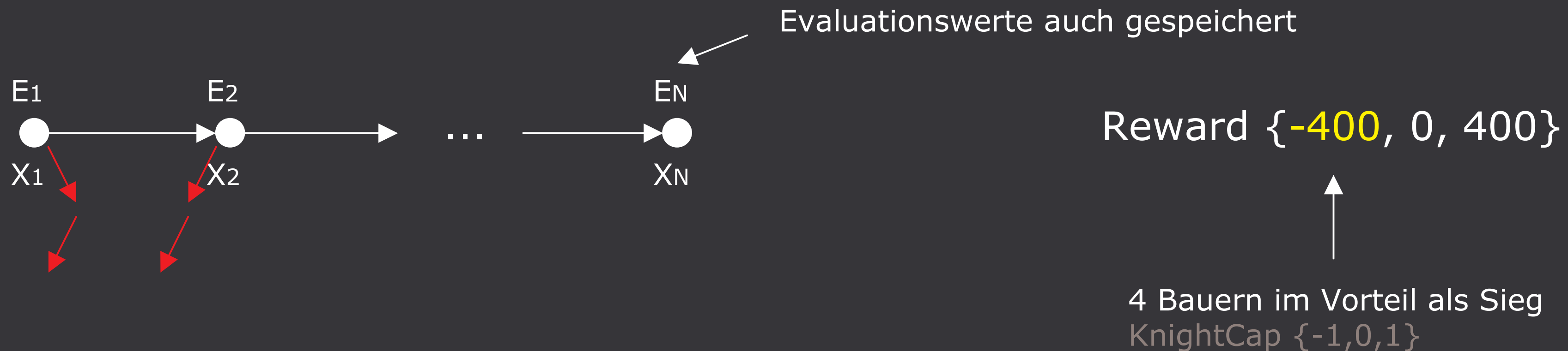
Stellungsfolge+HauptVarianten :



für Evaluation nur eigene Stellungen

Arbeitsweise von TD-Leaf(λ)

Stellungsfolge+HauptVarianten(HV) :



Lernalgorithmus:

```
E[N] := -400
for i:=1 to N-1 do
     td[i] := E[i+1] - E[i]
     go in HV[i]
          compute pd[i]
     go out HV[i]
end i
vectorUpdate
```

E[x] = Stellungsbewertung der Stellung x
mit Evaluationsvektor w bestehend aus n Stellungsfaktoren
lineare Funktion: w1f1(x) + w2f2(x) + w3f3(x) + ... + wnfn(x)

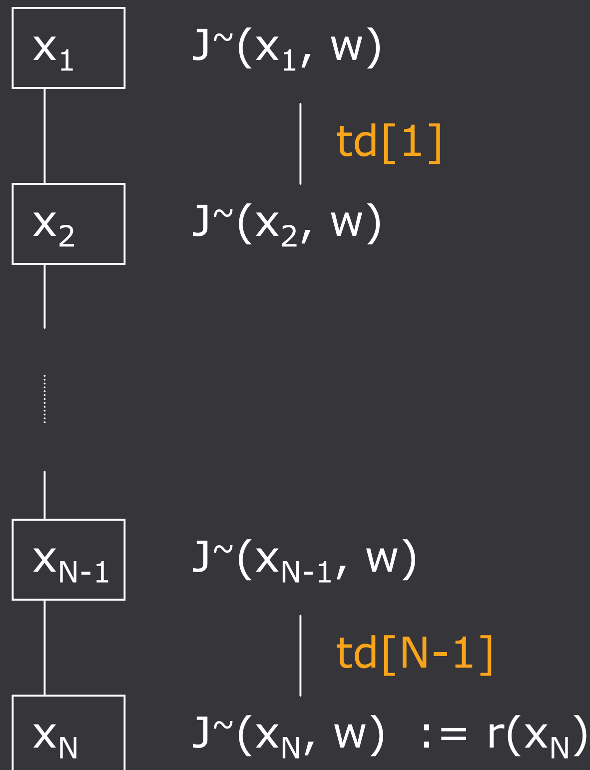
pd[i] =

$$\begin{bmatrix} w1f1(xi) + 0*f2(xi) + 0*f3(xi) + \dots + 0*fn(xi) \\ 0*f1(xi) + w2f2(xi) + 0*f3(xi) + \dots + 0*fn(xi) \\ 0*f1(xi) + 0*f2(xi) + w3f3(xi) + \dots + 0*fn(xi) \\ \dots \\ 0*f1(xi) + 0*f2(xi) + 0*f3(xi) + \dots + wnfn(xi) \end{bmatrix}$$

Evaluationsvektor w auf diesen Berechnungen basierend erneuern
und anschliessend normalisieren (auf Bauernwert 100)

Temporale Differenz

$x_1, \ldots, x_{N-1}, x_N$ eigene Stellungen unserer Partie.



$$E_{x_{t+1}|x_t}\left[J^*(x_{t+1}) - J^*(x_t)\right] = 0$$

„ich stehe besser, ergo erwarte ich einen Sieg...“

Modifizierung :

positive td[i]
- auf 0 setzen, es sei denn KnightCap sagt den richtigen Zug des Gegners voraus

negative td[i]
- bleiben unverändert (eigene Fehler)

VectorUpdate bei TD(λ)

Aktualisierung des Evaluationsvectors w :

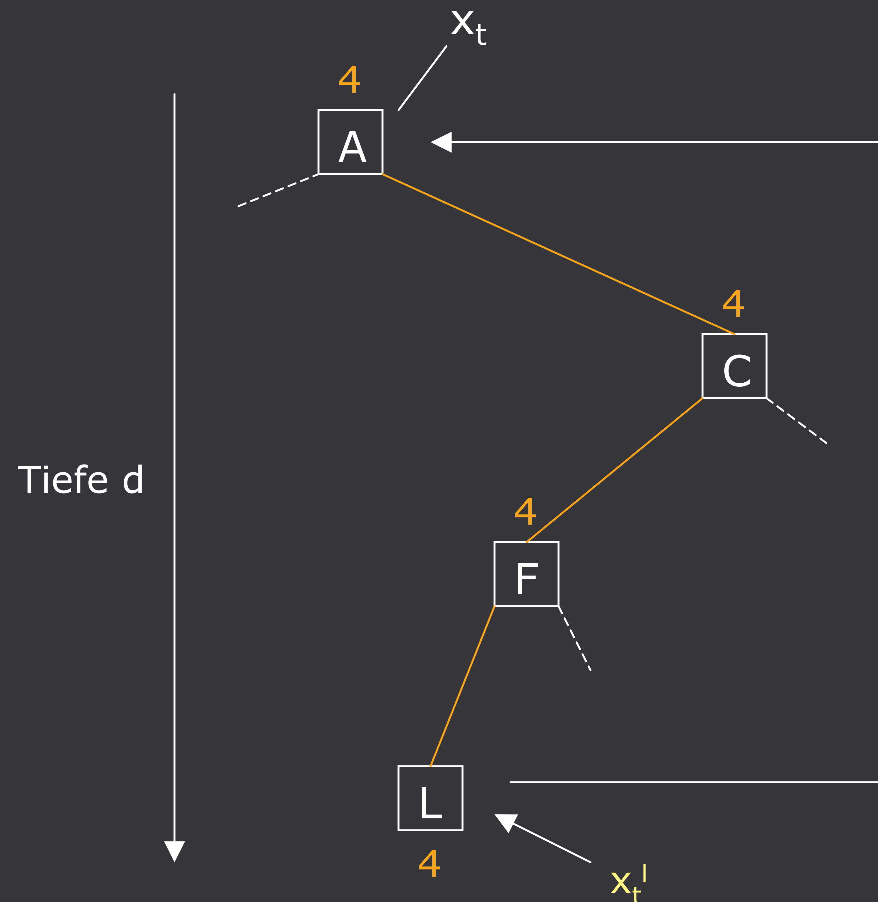
$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

Lernrate

Gradient

gewichtete Summe der Differenzen
im Rest der Partie

LeafNodes



MinMax + TD(λ) = TD-Leaf(λ)

VectorUpdate bei TD-Leaf(λ)

Aktualisierung des Evaluationsvectors w :

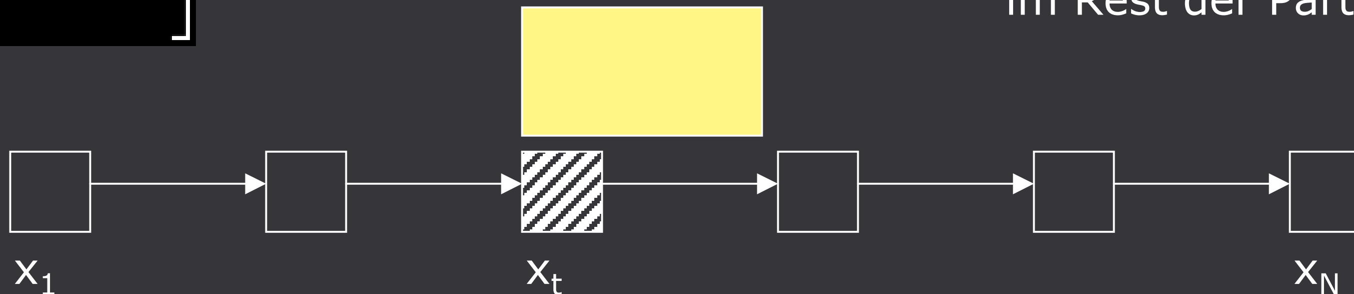
$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}_d(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

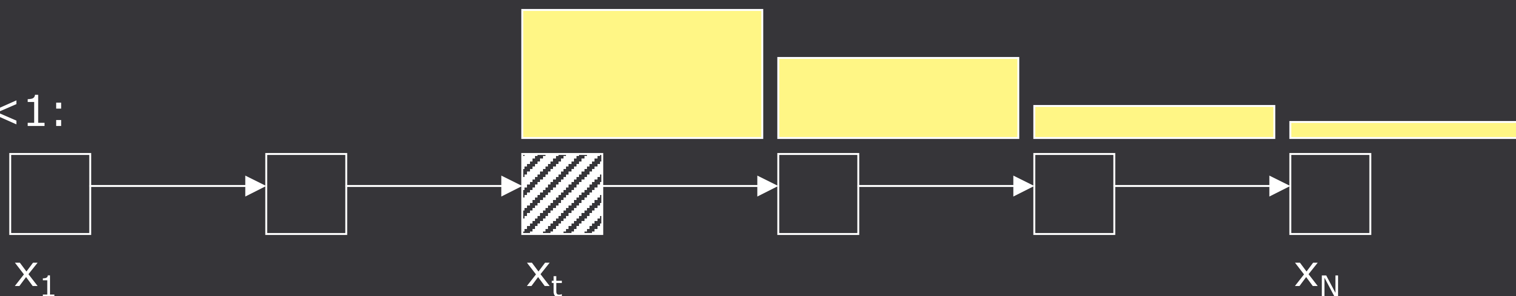
Welches λ wählen?

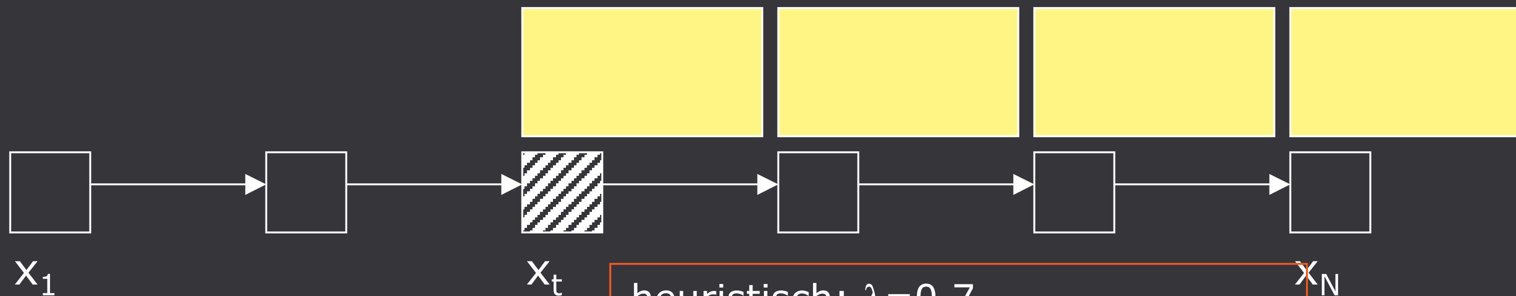
$$\left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t\right]$$

td[t]

Gewichtung der Differenzen im Rest der Partie







heuristisch: λ=0.7
(bis Fehler einen Effekt hervorruft)

VectorUpdate

Aktualisierung des Evaluationsvectors w :

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}_d(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

kurz: Δt

um wieviel

was muss ich ändern

Δt > 0:

⇒ Stellung x_t wurde vermutlich unterbewertet
⇒ w` = w + pos. Vielfaches des Gradienten
⇒ J~(x_t, w`) > J~(x_t, w)

Δt < 0:

⇒ Stellung x_t wurde vermutlich überbewertet
⇒ w` = w + neg. Vielfaches des Gradienten
⇒ J~(x_t, w`) < J~(x_t, w)

Fazit: TD(λ) -> TD-Leaf(λ)

TD(λ):

$$d_t := J'(x_t, \omega) - J'(x_t, \omega)$$

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

TDLeaf(λ):

$$d_t := J_d'(x_{t+1}, \omega) - J_d'(x_t, \omega)$$

$$w := w + \alpha \sum_{t=1}^{N-1} \nabla \tilde{J}_d(x_t, w) \left[\sum_{j=t}^{N-1} \lambda^{j-t} d_t \right]$$

KnightCap Experimente I

Experiment 1

- alle Koeffizienten auf 0, Material auf Standard
  1=Bauer, 4=Springer und Läufer, 6=Turm und 12=Dame

- nach 25 Spielen besitzt KnightCap ein rating von 1650 (+-50)

- TD-Leaf nun „on" mit $\lambda=0.7$ „heuristisch gewählt" und $\alpha=1.0$ „grosse Sprünge"

- 3 Veränderungen wurden vorgenommen:
  + $J'(x_i^l,w)$ wurde konvertiert zu einem Wert zwischen –1 und 1 mit
    $v_i^l=\tanh[\beta\, J'(x_i^l,w)]$
  + $d_t=v_{t+1}^l - v_t^l$ wurde modifiziert,
    negative Werte von d_t wurden nicht verändert,
    alle positiven temporalen Differenzen wurden auf 0 gesetzt
  + der Bauernwert wird mit 1 als Basiswert fixiert

- nach 300 Spielen (in 3 Tagen) hatte KnightCap ein rating von 2150

KnightCap Experimente

Experiment 2

- Experiment wurde wiederholt aber in Abhängigkeit der Wurzelknoten
- nach 300 Spielen hatte KnightCap ein rating von 1850
  (signifikant kleinerer peak und langsame Konvergenz)

Experiment 3

- alle Koeffizienten auf Bauernwert=1
- startrating=1250 nach 1000 Spielen rating=1550

Experiment 4

- KnightCap vs. KnightCap
- 11% Punkte für KnightCap(selfplay) nach 600 Spielen gegen
  KnightCap(humanplay) nach 300 Spielen

FUSC# Experimente

Experimente

- Fusc# (damalige Version) konnte im Experiment keine Partie gewinnen

- Temporale Differenz schlug fehl
  Grund: keine Siege => Fusch zweifelt an allem; keine positiven Updates
  => neue Engine musste her
  + Darkfusch im Januar 2004 geboren
  + heute 350.000 Knoten pro Sekunde! (Ruffian 600.000 Knoten pro Sekunde)

- 1.Versuch mit RL:
  SAL von Michael Gherrity spielte gegen GNUChess
  => RL für Schach nicht geeignet

- Deep Blue verwendete RL (Match gegen Kasparov)!

Le4!!

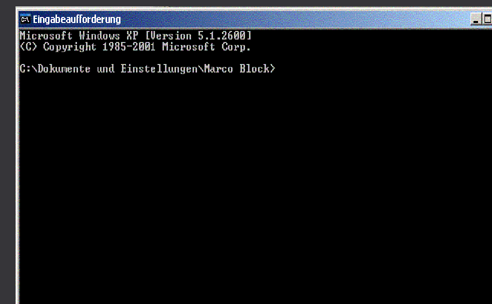
Kommunikation

UCI-Protokoll (Universal Chess Interface)

Fritz, Arena

UCI-
Kommunikation

Konsole



www.schach.de

Pseudonym :)

Ergebnisse der Wertungspartien: deepfusch

Computer | Zentaur

Zahl der Partien	2350
Gespeicherte Ergebnisse	398
Weiß	198
Siege	24
Remisen	1
Niederlagen	373
Ergebnis	24.5/398 = 6.2%
Gegner Elo	2362
N Gegner	151
Wertung	**1744**
Datum	1.12.2003

Gegner	Ergebnis	Elo	Seite	#
omega500	0	2352	White	2105.
cardiologist	1	2242	White	2104.
subaja	0	2315	Black	2103.
subaja	0	2312	White	2102.
rhasalgheti	0	2500	White	2101.
no man chess	1	2350	Black	2100.
yxvs	0	2318	Black	2099.
borstel13	0	2271	White	2098.
yxvs	1	2352	White	2097.
cardiologist	1	2392	Black	2096.
shackelton	0	2477	Black	2095.
hugo	0	2319	White	2094.
hugo	0	2317	Black	2093.
jml26	0	2375	White	2092.

Bild | Performance | Verteilung | Chart

OK | Abbrechen | Hilfe

deepfusch online

Datei Bearbeiten Ansicht

Info | Spieler | Partien | Wel

Name	Compu
roem_engine	2418
Myrddin	2415
yovova	2411
guesttt	2377
venom	2358
Kid Durcell	2352
gdr	2321
Murx	2319
ViriatoPT	2309
Felix 2	2266
Albert Einstein	2202
ElAd	2189
peere	
Nicos_Lycos	
Nexusgab	
Guest88163	
erman9us	
devils point	
deepfusch	

Herausfordern | Verfolge

Gegner | Bedingung

Annehmen | Löschen

Fritz 8

Start | Schach - Micros... | TODOMittwoch.... | UltraEdit-32 - [Z:... | AG Schachprogr... | BlitzC - Microsoft... | Microsoft Excel -... | deepfusch o... | 11:43 Mittwoch

Diplomarbeit

Fragen, Experimente:

- In welcher Korrelation liegen Spielstärke und erlernbare Faktorenanzahl?

- Was geschieht, wenn man die Sache auf die Spitze treibt und z.B. 1000 verschiedene Stellungstypen einführt? Jeder hat ca. 1500 Stellungsfaktoren
  => 1.500.000 zu erlernende Bewertungskriterien!!!

- Kann Fusch in annehmbarer Zeit die Koeffizienten mit „guten" Werten füllen?

- Algorithmuserweiterung behebt Problem.

- Ist λ=0.7 wirklich der richtige Parameter?

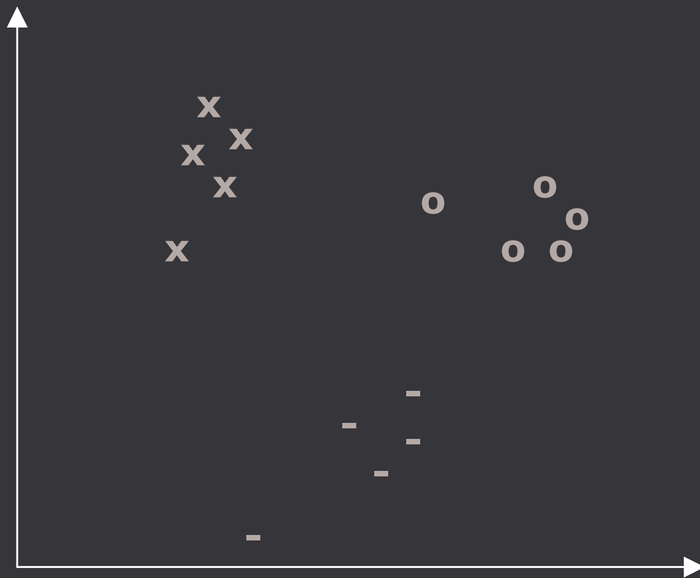
- Wie muss α sinnvoll konvergieren?

- Startkoeffizienten sinnvoll?

Zusätzlich:

- Guide, um dieses Lernverfahren in jeden Schachmotor zu bringen

Stellungsklassifikator



∀ Stellungen aus GM-DB
- Stellungsmuster identifizieren
- Vektor bauen
- Vektoren aufspannen

EM-Algorithmus => Pool

eine Evaluation für jede Stellungsklasse

Vorteil: Stellungen können viel besser evaluiert werden (Eröffnung, Mittel- und Endspiel zu grobe Einteilung)

Nachteil: Für jedes Stellungsmuster neue Klassifizierung

RL „feiner“ nutzen

KnightCap

α
0.50
1.20
1.00
0.97
0.20
0.50
1.05
0.44
1.21
1.50

4x1468 = 5872
(74 angegeben …)

FUSC#

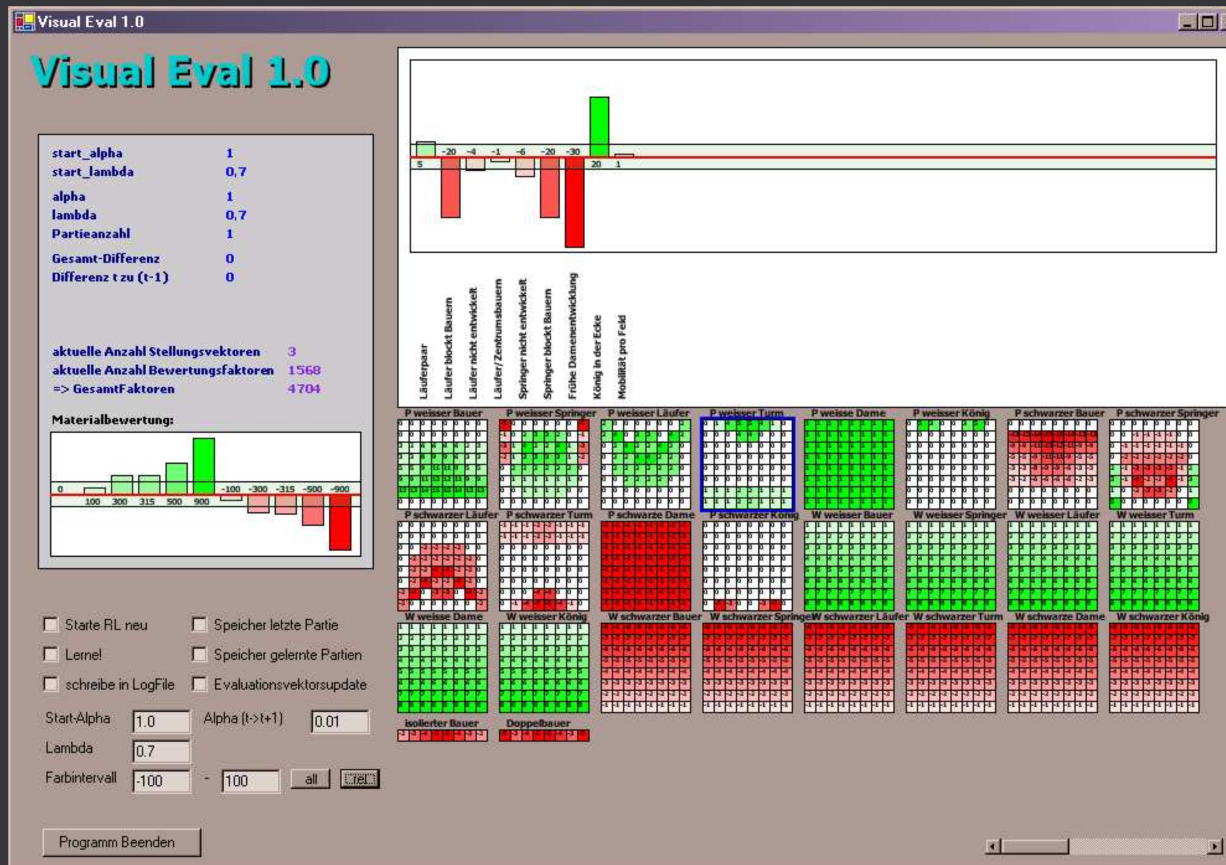
α_1	α_2	…	α_n
0.50	0.40		0.60
1.20	1.25		1.14
1.00	1.10		1.00
0.97	0.90		0.94
0.20	0.20		0.22
0.50	0.55	…	0.48
1.05	1.15		0.50
0.44	0.40		0.49
1.21	1.11		1.31
1.50	1.50		1.52

1000x1500 = 1.500.000

Visual Eval 1.0

Reinforcement Learning

Anpassung des Evaluationsvektors



das wars

vielen Dank fürs zuhören ...

Fusc# http://page.mi.fu-berlin.de/~fusch/

Schachserver http://www.schach.de/