Technische Hochschule Rosenheim

Entwicklung von Computerspielen: KI Rundenbasiert Teil 2

Fakultät Informatik
FWPM

Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Übersicht

- Transpositionstabellen
- Zobrist Schlüssel
- Monte Carlo Tree Search

Technische Hochschule **Rosenheim**

Transpositionstabellen
Übersicht

- Transposition = Permutation, bei der 2 Elemente vertauscht werden
- In Spielen: unterschiedliche Reihenfolge (Transpositionen) der gleichen Züge kann zu gleicher Stellung führen
- Idee:
  - Wurde eine Stellung bewertet → speichere diese in Hash-Tabelle
  - So werden doppelte Bewertungen vermieden
  - Bewertung/ Speicherung während der Spielzeit des Gegners möglich
- Umwandlung Suchbaum → Suchgraph
- Beispiel Schach
  - Suchbaum ca. $35^{100} \approx 2{,}5 \cdot 10^{154}$ Knoten
  - Suchgraph ca 10^{50} Knoten
  - → nur kleiner Teil kann wirklich gespeichert werden

Technische Hochschule Rosenheim

Transpositionstabellen
Hashing

- Prinzipiell: Verwendung eines beliebigen Hash-Verfahrens möglich
- In Spielen:
  - Viele Stellungen treten nie auf
    - Resultat ungültiger Züge oder „seltsamer“ Zugfolgen
- Anforderungen an Schlüssel:

  Verteile wahrscheinliche Stellungen so weit wie möglich im Schlüsselraum

  Kleine Änderungen auf dem Brett resultieren in großen Änderungen im Schlüssel

  Inkrementelle Aktualisierung des Schlüssels

Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Zobrist Schlüssel

- Idee: Berechne Schlüssel für die gesamte Stellung aus Einzelschlüsseln für jedes Feld
- Einzelschlüssel: Zufallszahl
  Achtung: sicherstellen, dass alle verschieden sind!
- Wie viele Einzelschlüssel werden benötigt?

	#Spielstein Arten	*	#Spielfelder	
Schach:	6 * 2	*	8*8	= 768
Dame:	2 * 2	*	8*8	= 256
Go:	1 * 2	*	19*19	= 722

- Zusätzliche Schlüssel zur Kodierung spezieller Züge möglich

Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Zobrist Schlüssel Schlüssellänge

- Abhängig vom Spiel
- Schach 64 Bit
- Dame 32 Bit
- Bei komplexeren Spielen evtl. 128 Bit nötig

- $2^{64} \approx 2 * 10^{19} << 10^{50}$ Knoten des Suchgraphs bei Schach
  Verschiedene Stellungen führen zum gleichen Schlüssel
  Das wird akzeptiert... in der Hoffnung, dass es
  nicht so schlimm ist

KI: Rundenbasiert Teil 2
Zobrist Schlüssel Schlüsselberechnung

- Berechnung des Schlüssels für eine Stellung:

  XOR – Verknüpfung der Einzelschlüssel

- Beispiel:

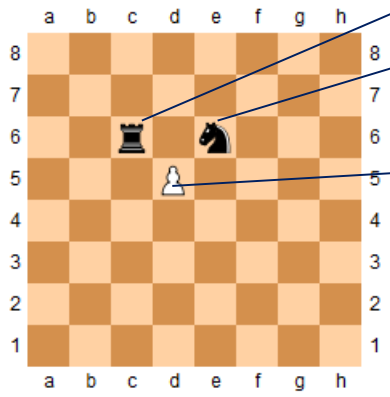


Bild: http://de.wikipedia.org/wiki/Schach

```
1100 1101 0000 1010
0011 0000 1111 0010   XOR
1111 1101 1111 1000
0000 0101 0101 1001   XOR
1111 1000 1010 0001
```

Technische Hochschule **Rosenheim**

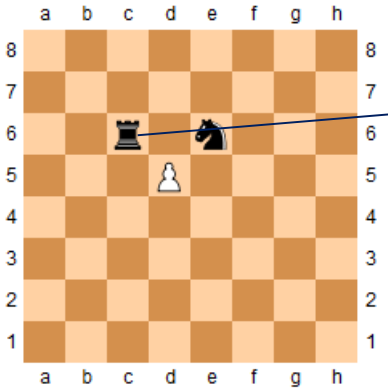
KI: Rundenbasiert Teil 2
Zobrist Schlüssel Aktualisierung

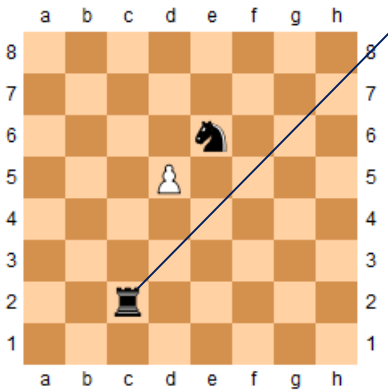
- XOR ist eine **Involution** (selbstinverse Abbildung):
  a XOR **b** XOR **b** = **a**
- Schlüsselaktualisierung bei Zug eines Steins:
  1. XOR des Stellungsschlüssels mit Einzelschlüssel der aktuellen Position des Steins
  2. XOR des Stellungsschlüssels mit Einzelschlüssel der neuen Position

Anders gesagt:
Der Stein verlässt seine alte Position, daher muss diese Position herausgerechnet werden. Danach muss die neue Position wiederum eingerechnet werden.

KI: Rundenbasiert Teil 2

Zobrist Schlüssel Aktualisierung; Beispiel






```
1111 1000 1010 0001
1100 1101 0000 1010
0011 0101 1010 1011
```

Stellungsschlüssel
Einzelschlüssel Turm c6
Ergebnis XOR

```
0101 0100 1011 0111
0110 0001 0001 1100
```

Einzelschlüssel Turm c2
Ergebnis XOR =
neuer Stellungsschlüssel

Stein Aufheben

Stein Ablegen

Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Zobrist Schlüssel Anmerkungen

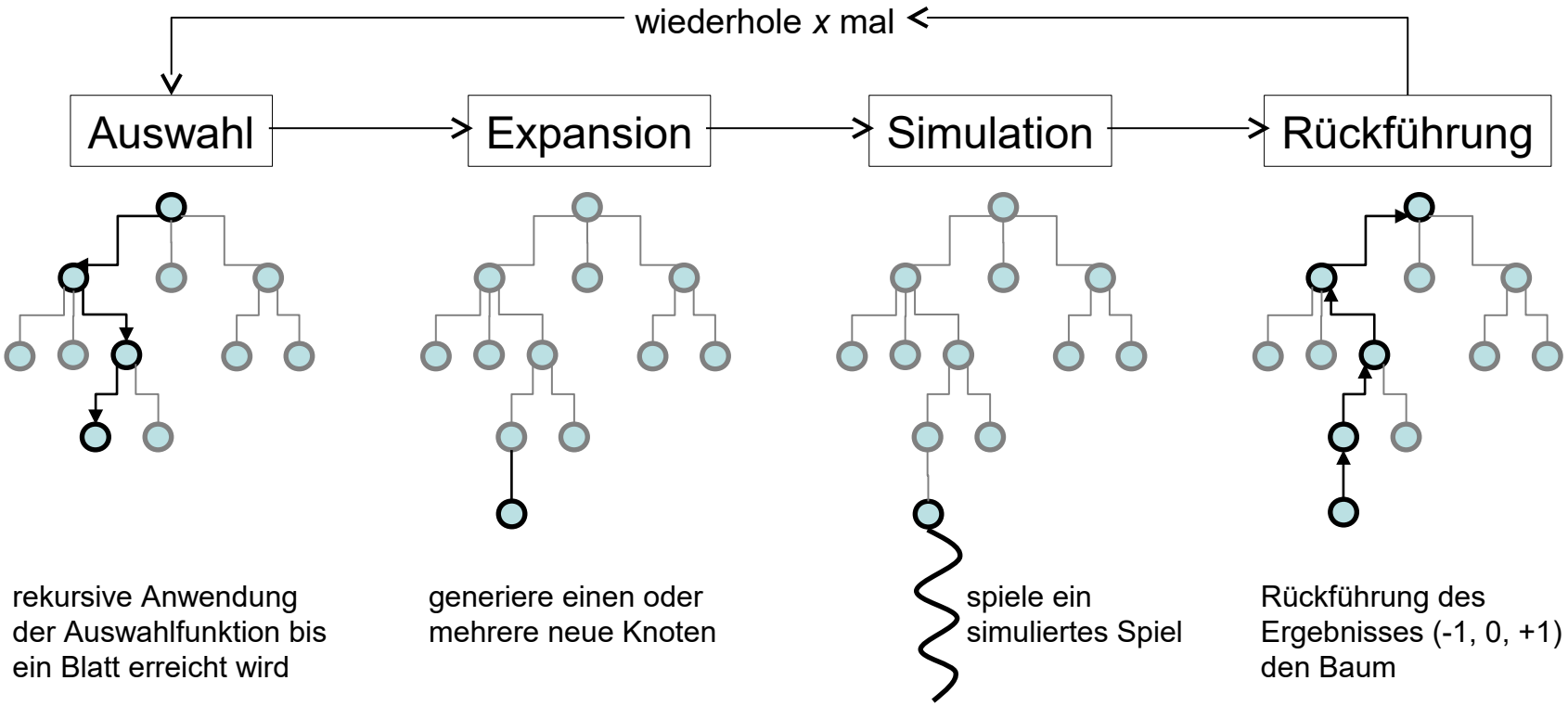
- Inkrementelle Aktualisierung
  - Extrem schnell
  - Stellungsschlüssel muss nur einmal am Anfang für die Startstellung berechnet werden
  - Danach nur noch Aktualisierungen
- Gespeichert werden sollte in der Hash-Tabelle auf jeden Fall:
  - Zobrist-Schlüssel (Erkennung von Kollisionen)
  - Statische Bewertung
  - Zug
- #Einträge in Hash-Tabelle << #Schlüssel
  - Berechnung Index i in Array: i = *Schlüssel* **mod** *MaxElemente*
  - Was tun bei Kollision?
    - Einfachste Methode → Eintrag einfach überschreiben
      - => normalerweise völlig ausreichend
    - Viele komplexere Strategien möglich, Nutzen ist aber umstritten

Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Monte Carlo Baumsuche

- Oberbegriff für Algorithmen zum Aufbau des Spielbaums, die auf Monte-Carlo-Methoden basieren
- Grundidee
  - Spiele das Spiel ausgehend von der aktuellen Stellung komplett durch
  - Verwende eine zufällige Zugauswahl
  - Wiederhole dies sehr oft
  - Speichere, wie oft ein Zug zum Sieg geführt hat
  - Wähle den Zug, der am aussichtsreichsten erscheint
- Führt man dies häufig genug durch, kann man davon ausgehen, dass eine Stellung besser ist, wenn sie öfter zum Sieg geführt hat

KI: Rundenbasiert Teil 2
MCTS (nach [Cha08])



jeder Knoten enthält:

- wie oft er besucht wurde (n_i)
- die Anzahl der gewonnenen Spiele w_i

Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Monte Carlo Baumsuche: Auswahl (Selection)

- Wie wählt man einen vielversprechenden Knoten aus?
- Wichtig: Abwägung zwischen
  - Selektiere einen Knoten, der bereits gut bewertet ist (Exploitation)
  - Selektiere einen Knoten, der noch nicht ausreichend untersucht wurde (Exploration)
- Typisch: **UCT-Strategie** nach [Koc06]
  - UCT = **U**pper **C**onfidence bound applied to **T**rees
  - Wähle den Knoten *i*, für den

$$B_i = \frac{w_i}{n_i} + C \cdot \sqrt{\frac{\ln n_p}{n_i}}$$

am größten ist

w_i: Anzahl der gewonnen Spiele des Knotens i
n_i: wie oft wurde der Knoten i besucht
n_p: wie oft wurde der Vaterknoten p von i besucht
C: Konstante, die theoretisch $\sqrt{2}$ sein muss ([Cha08] nehmen aber z.B. 0,7)

Technische Hochschule Rosenheim

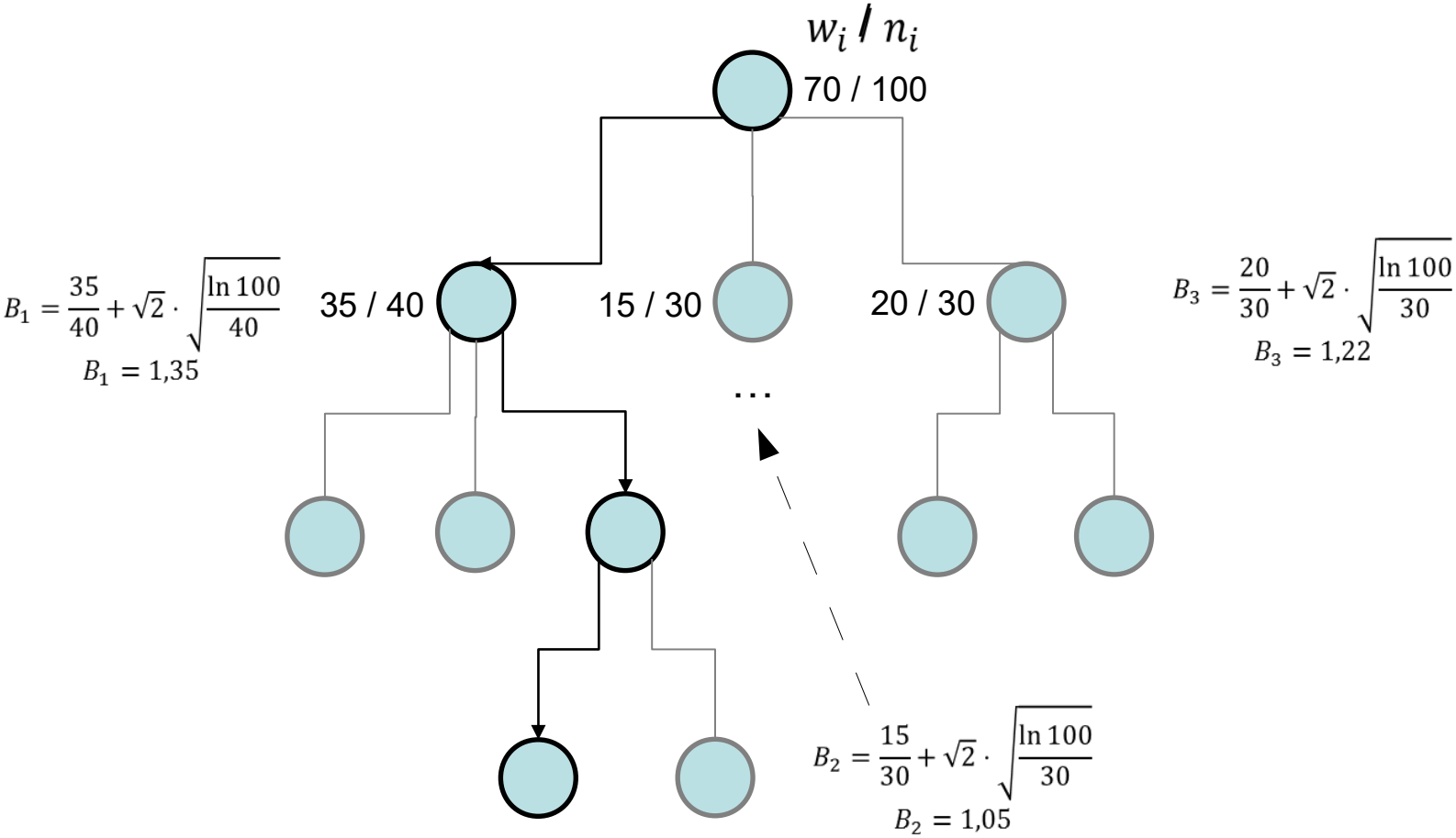
KI: Rundenbasiert Teil 2
Monte Carlo Baumsuche: Auswahl (Selection)

- Anmerkung:

  In der Praxis:
  Verwendung von UCT nur, wenn n_i einen Schwellwert übersteigt
  (in [Cha08] z.B. 30)
  Sonst: Auswahlstrategie wie bei *Simulation*

KI: Rundenbasiert Teil 2

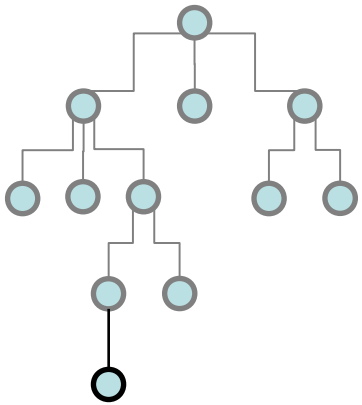
Monte Carlo Baumsuche: Auswahl Beispiel



Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Monte Carlo Baumsuche: Expansion

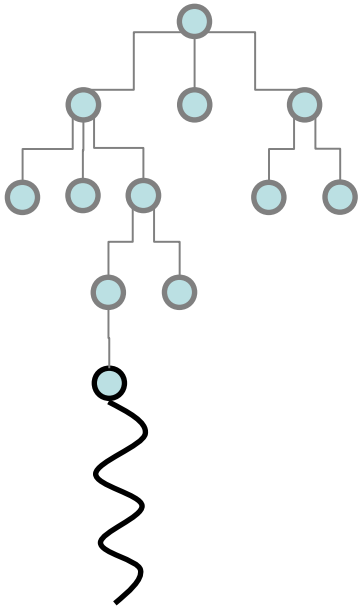
- Einfachste Variante: expandiere einen einzigen Knoten für jedes simulierte Spiel
- Alternative: expandiere alle Knoten, deren Besuchszahl n_i größer als ein Schwellenwert ist



Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Monte Carlo Baumsuche: Simulation (Playout)

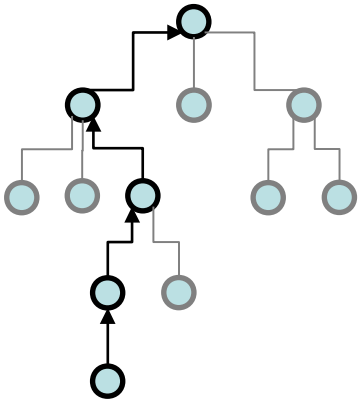
- Auswahl von Zügen, bis das Spiel zu Ende ist
  - Zufällig oder
  - Nach einer Simulationsstrategie
  - (d.h., mit bestimmten Regeln/ Einschränkungen der Züge)
- Es ergibt sich typischerweise ein schwaches Spiel bei
  - Zu stochastischer Strategie (sehr zufällige Züge)
  - Zu deterministischer Strategie
  - (bei gleicher Stellung immer der gleiche Zug)



Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Monte Carlo Baumsuche: Rückführung (Backpropagation)

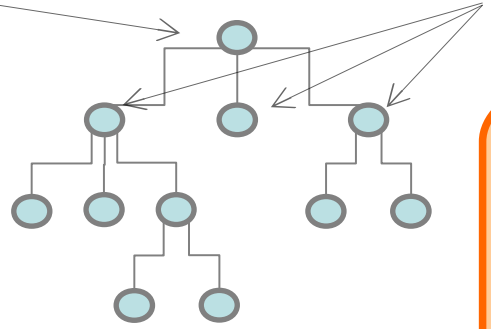
- Am Ende der Simulation ist klar, wer das simulierte Spiel gewonnen hat
- Aktualisierung aller Knoten von unten nach oben
  - -1: verloren
  - 0: unentschieden
  - +1: gewonnen



Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Monte Carlo Baumsuche: Simulation (Playout)

aktuelle Stellung



welcher Zug soll ausgeführt werden?

Untersuchungen in [Cha08] zeigten keinen signifikanten Unterschied, wenn genügend Simulationen durchgeführt wurden. Bei kleiner Zahl Simulationen war das Max Kind wesentlich schlechter als die anderen Kriterien (was man auch erwarten würde). Verwendet wurde in [Cha08] das robuste Kind.

- **Max Kind:** Der Kindknoten mit der höchsten Bewertung
- **Robustes Kind:** Das Kind mit der höchsten Besuchszahl n_i
- **Robustes Max Kind:** Das Kind, das sowohl die höchste Bewertung als auch die höchste Besuchszahl hat. Gibt es kein solches, wird so lange weiter simuliert, bis eines entsteht
- **Sicheres Kind:** Das Kind, das eine Konfidenzschranke maximiert, d.h. $\frac{w_i}{n_i} + \frac{A}{\sqrt{n_i}}$ Wobei A ein frei wählbarer Parameter (z.B. A = 4) ist

Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Monte Carlo Baumsuche: Vor/ Nachteile

- Keine statische Bewertungsfunktionen notwendig
- Der Algorithmus kann jederzeit unterbrochen werden
- Gut bei Spielen mit hohem Verzweigungsgrad
  (Spielbaum wächst asymmetrisch)
- MCTS konvergiert nachweislich gegen Minimax, wenn auch in der reinen Zufallsversion sehr langsam

KI: Rundenbasiert Teil 2
Google DeepMind AlphaGo

Technische Hochschule Rosenheim

- APV-MCTS (Asynchronous Policy and Value MCTS)
- Kombiniert neuronale Netze mit MCTS
  Auswertung der netze erfolgt Asynchron zu MCTS (Rechenzeit der Netze)
- Policy Network
  Macht Vorschläge für gute Züge
  Eingang: eine Stellung
  Ausgang: Warscheinlichkeitsverteilung über alle möglichen Züge
  Je höher die Wkt., desto besser der Zug
- Value Network
  Statische Bewertung einer Stellung
  → Wahrscheinlichkeit, damit das Spiel zu gewinnen
- MCTS Knoten enthalten zusätzlich
  Stellungsbewertung aus Value Network
  Wert aus Policy Network als Anfangswahrscheinlichkeit

Details siehe:
D. Silver et al.: Mastering the game of Go with deep neural networks and tree search. Nature 529 (7587): 484–489, 2016.
PDF: http://willamette.edu/~levenick/cs448/goNature.pdf

Technische Hochschule Rosenheim

KI: Rundenbasiert Teil 2
Quellen

- [Koc06]
  L. Kocsis, C. Szepesvári: Bandit based Monte-Carlo Planning. Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18–22, 2006, Proceedings. Lecture Notes in Computer Science 4212, 2006.
- [Cha08]
  G.M.J.B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, B. Bouzy: Progressive Strategies for Monte-Carlo Tree Search. New Mathematics and Natural Computation 4 (3): 343–359, 2008.