Grundlagen der Künstlichen Intelligenz (6/6)

Monte Carlo Tree Search

UNIVERSITÄT MANNHEIM
Fakultät für Wirtschaftsinformatik und Wirtschaftsmathematik

Dr. Christian Meilicke, Research Group Data and Web Science

Monte Carlo Tree Search

- Alternatives Verfahren zu Min-Max basierten Verfahren
  - Im folgenden MCTS

- Im allgemeinen anwendbar auf Spiele mit einem sehr hohen Branching Faktor
  - Sehr erfolgreich bei GO

- Kann auf unvollständig beobachtbare Spiele (Karten) oder nicht deterministische Spiele (Würfel) angewendet werden
  - Sehr erfolgreich bei Back-Gammon

- Grundidee: Simuliere eine große Anzahl an vollständig durchgespielten Partien und baue in abhängigkeit von den Ergebnissen den Suchbaum auf
  - Benötigt kein Expertenwissen über das Spiel (= keine Heuristik zur Bewertung von Zuständen)

Entwicklung

2006

erste Publikationen
die MCTS als Namen
verwenden

UCT für die
Selektionsphase

2012

Browne et al.: A survey of
monte carlo tree search methods

referenziert 240 Papiere

2017

2012 survey
> 800 zitiert

2019

2012 survey
> 1500 zitiert

Gliederung

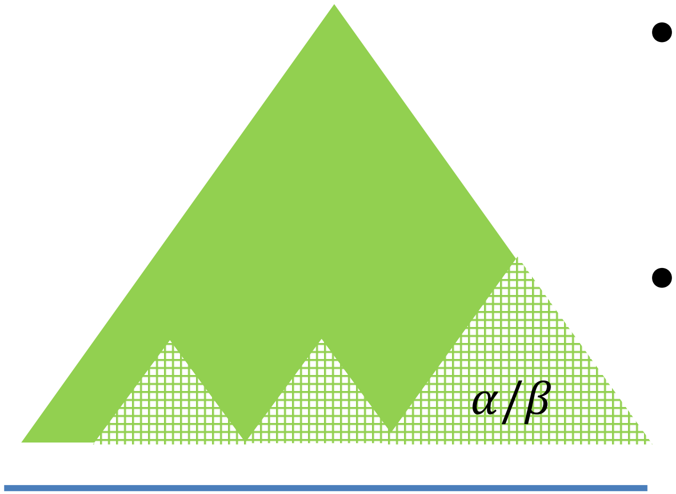
- Min-Max vs. MCTS

- Grundprinzip als Abfolge von vier Schritten
  - Selection mittels UCT
  - Expansion
  - Simulation (“playout”)
  - Backpropagation

- Anwendung auf Würfel- und Kartenspiele
  - Z.B. Determinisierung

Min-Max vs MCTS

- Suchbaum beim Min-Max Verfahren



tiefere Ebenen (mit Terminalknoten) werden nicht betrachtet

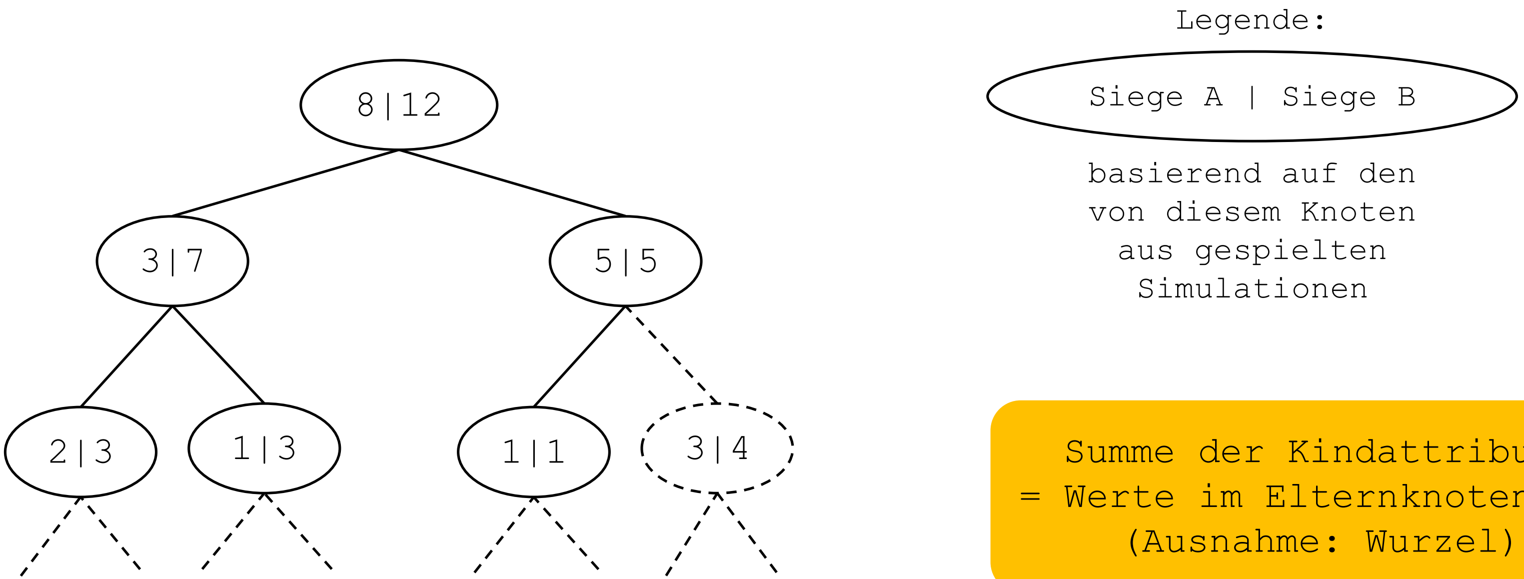
- Min-Max-Verfahren ≈ tiefenbeschränkte Suche
- Suchbaum vollständig ablaufen bis zu einer bestimmten Ebene
  - Wie bei einer Breitensuche
- Heuristik wird meist auf Nicht-Terminalzustände angewendet und hochpropagiert

Min-Max vs MCTS

- Suchbaum bei MCTS

- Suchbaum nicht gleichmäßig balanciert = Tiefe variiert
  - Ähnelt der A*-Suche
- Heuristikfunktion wird ersetzt duch aggregierte Simulationsergebnisse (=Statistik)
- Statistik entscheidet welcher Bereich genauer untersucht wird

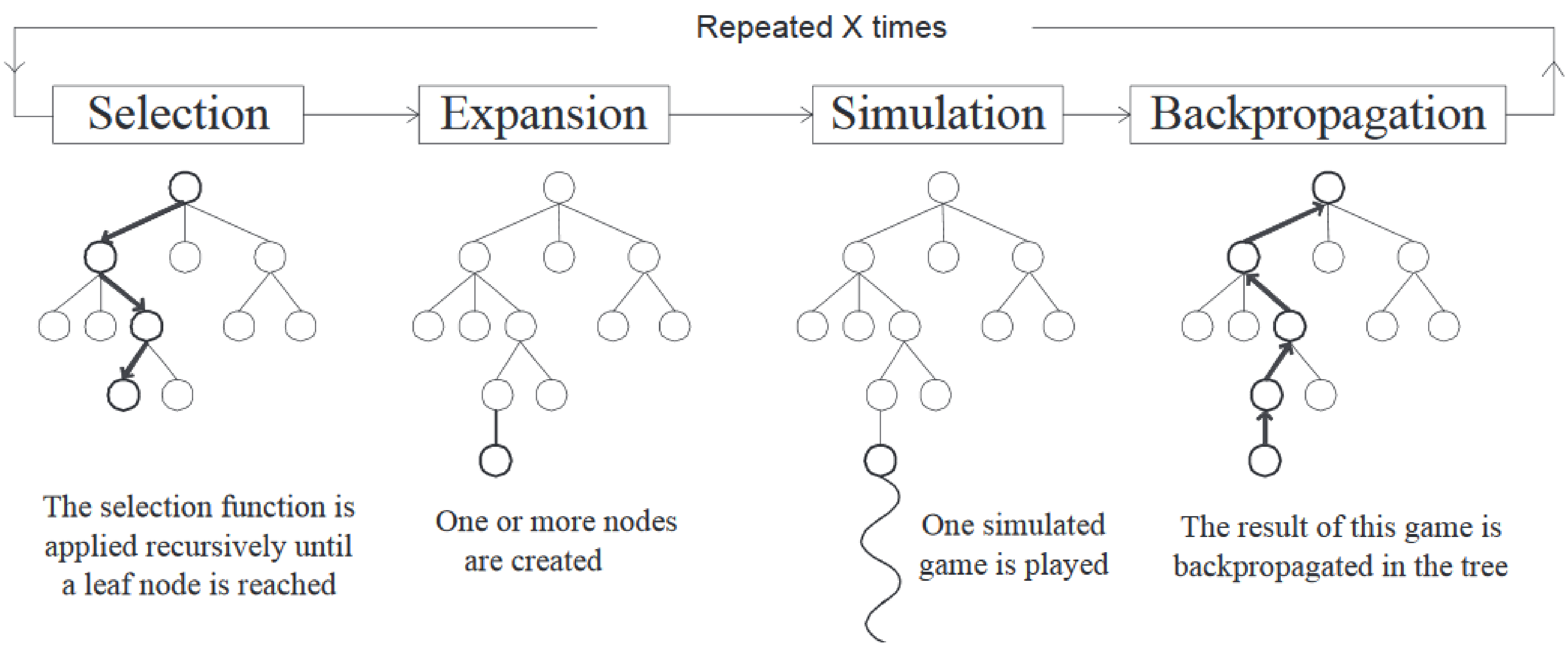
Datenstruktur / Statistik



Summe der Kindattribute
= Werte im Elternknoten + 1
(Ausnahme: Wurzel)

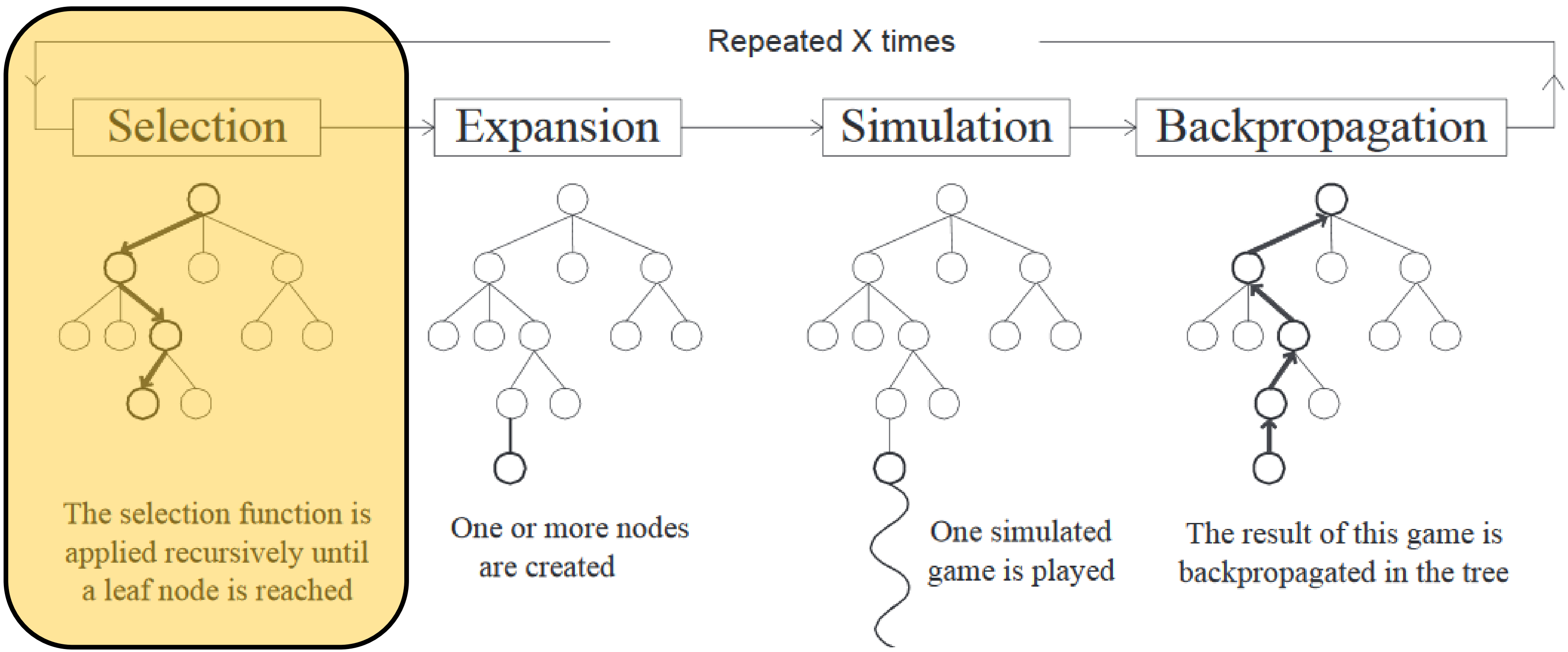
- Hinweis: Darstellung leicht modifiziert gegenüber der üblichen Beschreibung
- Siege (und Niederlagen) ergeben sich aus simulierten Spielen
- Je nach Implementierung kann es sein, dass ein Knoten, der kein Blattknoten ist weitere noch nicht expandierte Kinder hat
  - Der Knoten rechts unten ist so ein nicht expandiertes Kind

Phasen



Taken from CHASLOT et al.: PROGRESSIVE STRATEGIES FOR MONTE-CARLO TREE SEARCH, 2007.

Phasen



Taken from CHASLOT et al.: PROGRESSIVE STRATEGIES FOR MONTE-CARLO TREE SEARCH, 2007.

Selektion: UCB und UCT

- UCB = Upper Confidence Bound
- UCT = Upper Confidence Bound applied to Trees

- UCB Algorithmus bzw. Formel wurde ursprünglich angewendet auf das „Multiarmed Bandit Problem“
  - Aus mehreren einarmigen Banditen kann gewählt werden
  - Was ist die beste Taktik gegeben die bisherigen Erfahrungen

UCB

3 Spiele	10 Spiele	1 Spiel	5 Spiele
$\varnothing = +0.3$	$\varnothing = +0.4$	$\varnothing = -0.3$	$\varnothing = -0.1$

- Exploration-Exploitation Dilemma
  - Exploration = Erforsche durch Ausprobieren welches der beste Automat ist
  - Exploitation = Spiele am „besten“ Automaten um Gewinn zu machen

- Wann „explore“ und wann „exploit“?

UCB

3 Spiele	10 Spiele	1 Spiel	5 Spiele
∅ = +0.3	∅ = +0.4	∅ = -0.3	∅ = -0.1

- v_k = Durchschnittsgewinn am Automaten k
- C = Parameter (je höher C umso wichtiger wird Exploration)
- n = Anzahl aller Spiele
- n_k = Anzahl der Spiele an Automat k
- $UCB_k = v_k + C * \sqrt{\frac{\log n}{n_k}}$

log hier = logarithmus zur basis e
(natürlicher logarithmus)

UCB

C=1

3 Spiele ∅ = +0.3	10 Spiele ∅ = +0.4	1 Spiel ∅ = -0.3	5 Spiele ∅ = -0.1

- v_k = Durchschnittsgewinn am Automaten k
- C = Parameter (je höher C umso wichtiger wird Exploration)
- n = Anzahl aller Spiele
- n_k = Anzahl der Spiele an Automat k
- $UCB_k = v_k + C * \sqrt{\frac{\log n}{n_k}}$

UCB

C=1

3 Spiele	10 Spiele	1 Spiel	5 Spiele
∅ = +0.3	∅ = +0.4	∅ = -0.3	∅ = -0.1

- v_k = Durchschnittsgewinn am Automaten k
- C = Parameter (je höher C umso wichtiger wird Exploration)
- n = Anzahl aller Spiele
- n_k = Anzahl der Spiele an Automat k
- $UCB_k = v_k + C * \sqrt{\frac{\log n}{n_k}}$

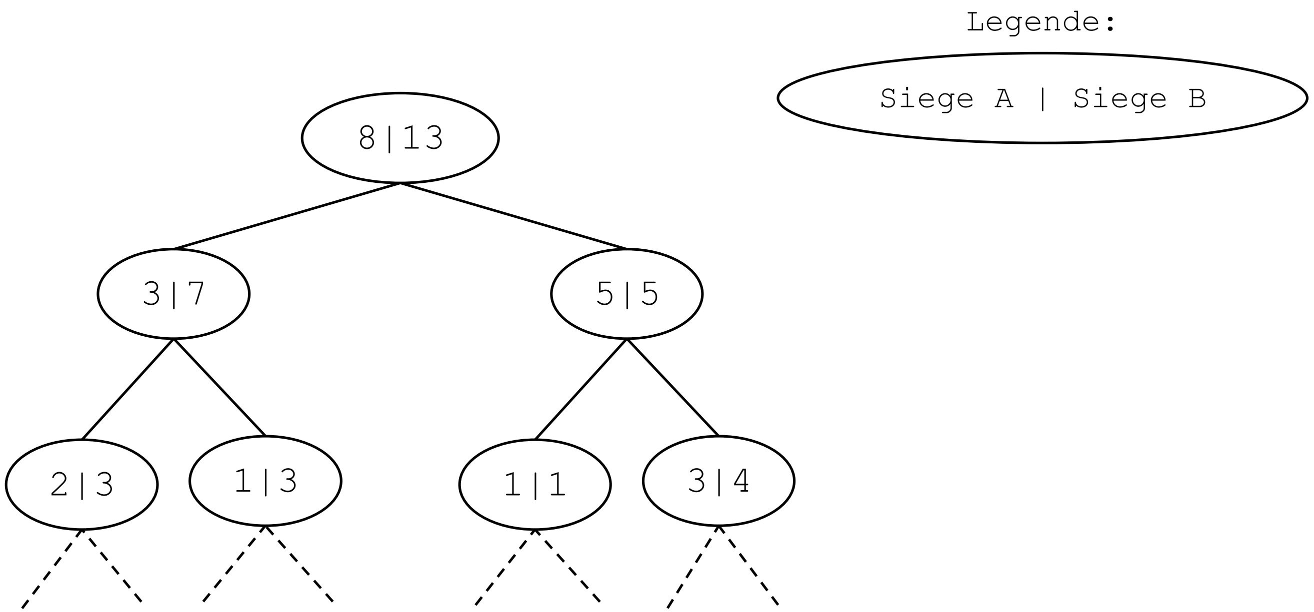
Wähle Automat mit dem höchsten UCB Wert und spiele dort weiter!

UCT

- Anwendung auf Bäume
- Entscheidungen an den inneren Knoten

- v_k = Anteil der gewonnenen Simulationen am Knoten k
  - **Aus Sicht des Spieler der den Zug macht, der zu Knoten k führt**
- C = Parameter (je höher C umso wichtiger wird Exploration)
- n_p = Anzahl der von k's Elternknoten gespielten Simulationen
- n_k = Anzahl der von Knoten k gespielten Simulationen
- $UCB_k = v_k + C * \sqrt{\frac{\log n_p}{n_k}}$

UCT



v_k = Anteil der gewonnenen Simulationen am Knoten k

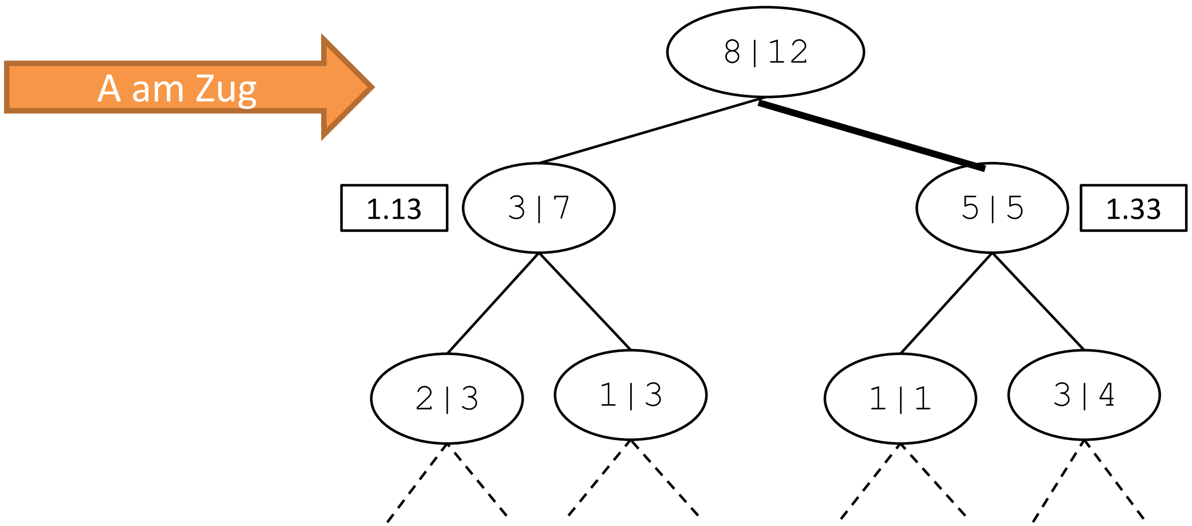
C = Parameter (je höher C umso wichtiger wird Exploration)

n_p = Anzahl der von k‘s Elternknoten aus gespielten Simulationen

n_k = Anzahl der von Knoten k aus gespielten Simulationen

$$\boldsymbol{UCB_k = v_k + C * \sqrt{\frac{\log n_p}{n_k}}}$$

UCT



v_k = Anteil der gewonnenen Simulationen am Knoten k

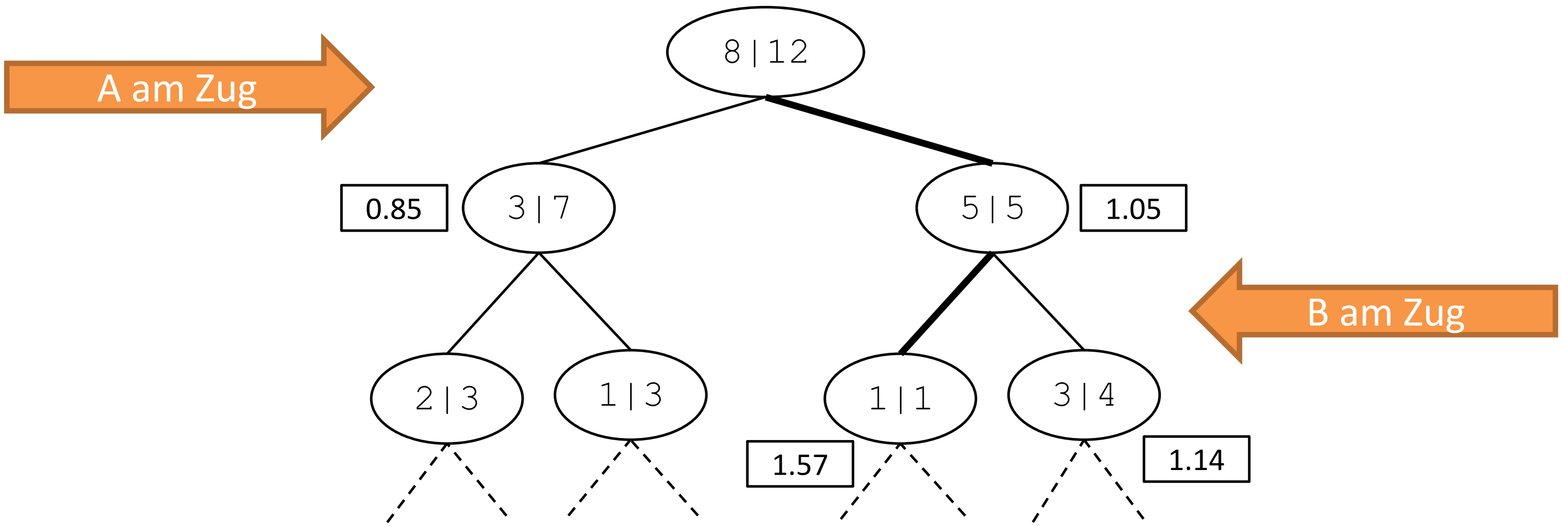
C = Parameter (je höher C umso wichtiger wird Exploration)

n_p = Anzahl der von k‘s Elternknoten aus gespielten Simulationen

n_k = Anzahl der von Knoten k aus gespielten Simulationen

$$\boldsymbol{UCB_k = v_k + C * \sqrt{\frac{\log n_p}{n_k}}}$$

UCT



v_k = Anteil der gewonnenen Simulationen am Knoten k

C = Parameter (je höher C umso wichtiger wird Exploration)

n_p = Anzahl der von k‘s Elternknoten aus gespielten Simulationen

n_k = Anzahl der von Knoten k aus gespielten Simulationen

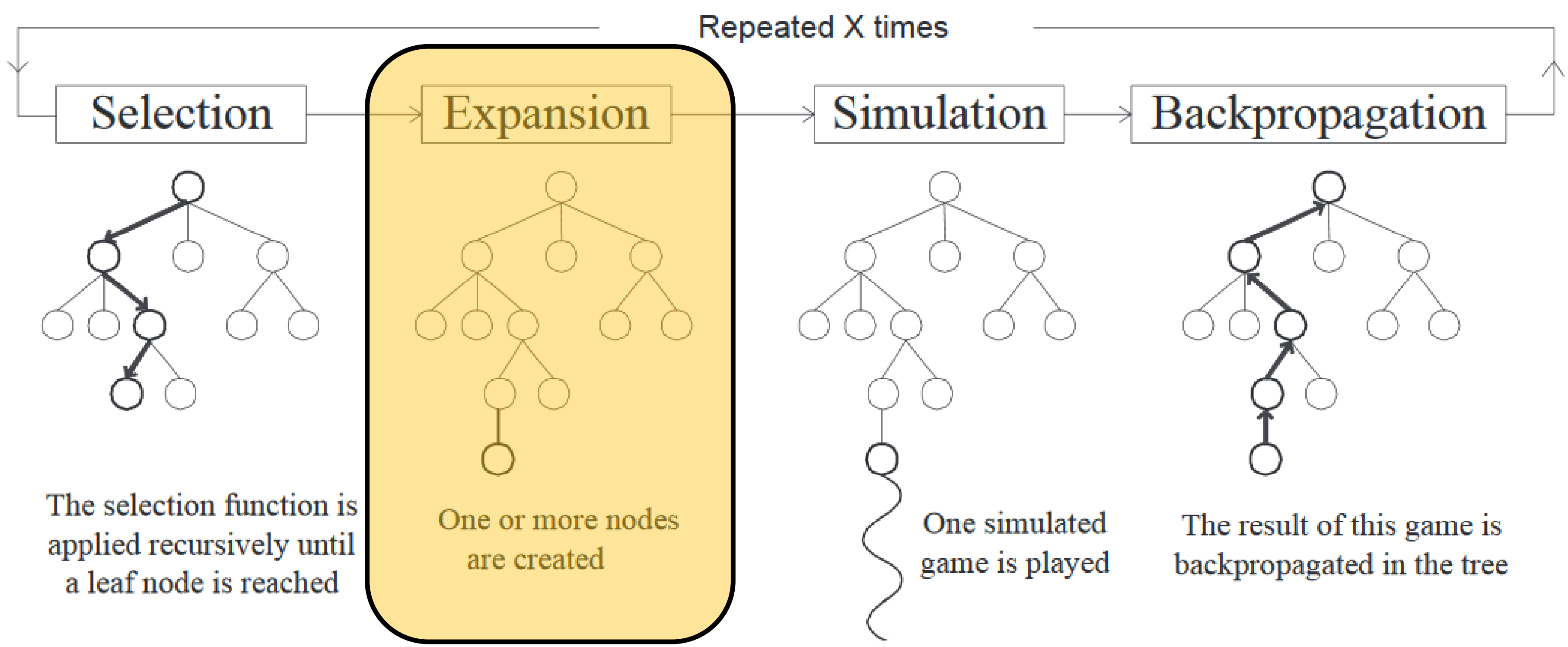
$$\boldsymbol{UCB_k = v_k + C * \sqrt{\frac{\log n_p}{n_k}}}$$

Wahl von C

- C ist ein Parameter, den man in der Regel experimentell ermittelt
  - Lasse KIs mit C=1, C=1.5, C=2, C=2.5, ... sehr oft gegeneinander antreten
  - Eventuell auch gegen andere KIs (z.B. Min-Max basiert)
  - Wähle den C-Wert, der am besten abschneidet

- Gute Ergebnisse können für $\sqrt{2}$ erwartet werden
  - Kocsis & Szepesvari: Bandit based monte-carlo planning. ECML, 2006.

Phasen



Taken from CHASLOT et al.: PROGRESSIVE STRATEGIES FOR MONTE-CARLO TREE SEARCH, 2007.

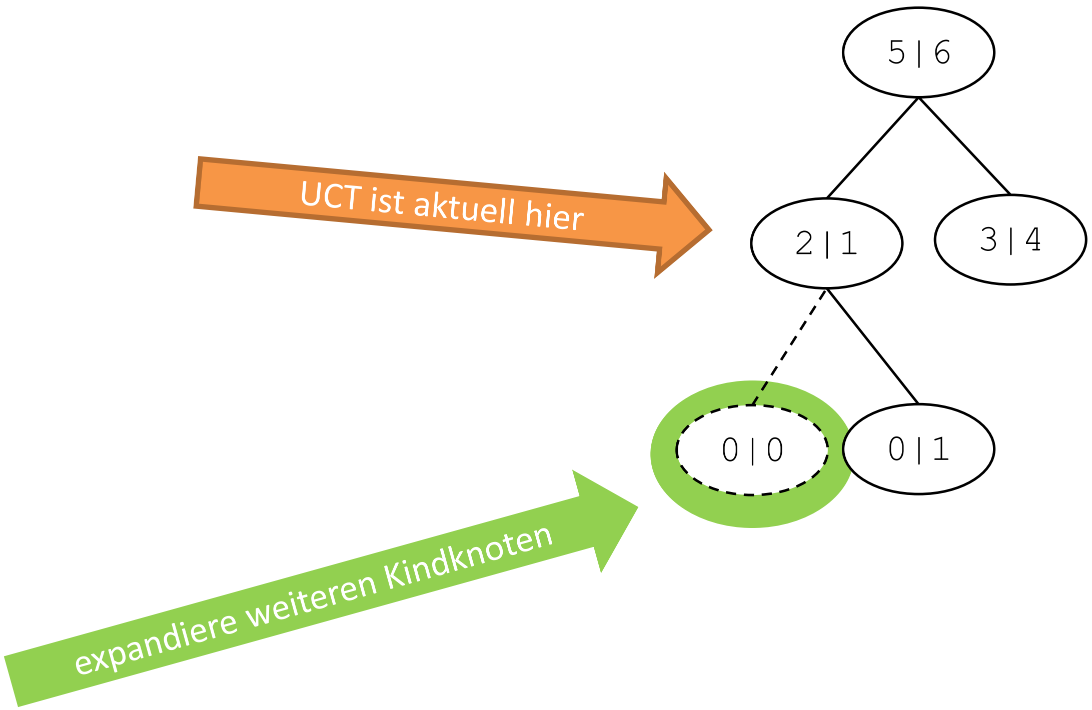
Expansionsregeln

- Innere Knoten, bei denen alle Kinder bereits expandiert sind, können nicht weiter expandiert werden
  - Auswahl (Selection) eines Kindes per UCT

- Blattknoten ohne Kinder
  - Variante 1:
    - Erzeuge alle Kinder, initialisiere diese mit 0|0 und führe für jeden Kindknoten eine Simulation durch
  - Variante 2:
    - Erzeuge einen zufällig ausgewählten Kindknoten, initialisiere diesen mit 0|0 und führe eine Simulation durch

Sonderregel

- Innere Knoten, bei denen mindestens ein Kind aber noch nicht alle Kinder expandiert wurden
  - Kann es nur in Variante 2 geben!
  - Erzeuge einen weiteren zufällig ausgewählten Kindknoten, initialisiere diesen mit 0|0 und führe eine Simulation durch



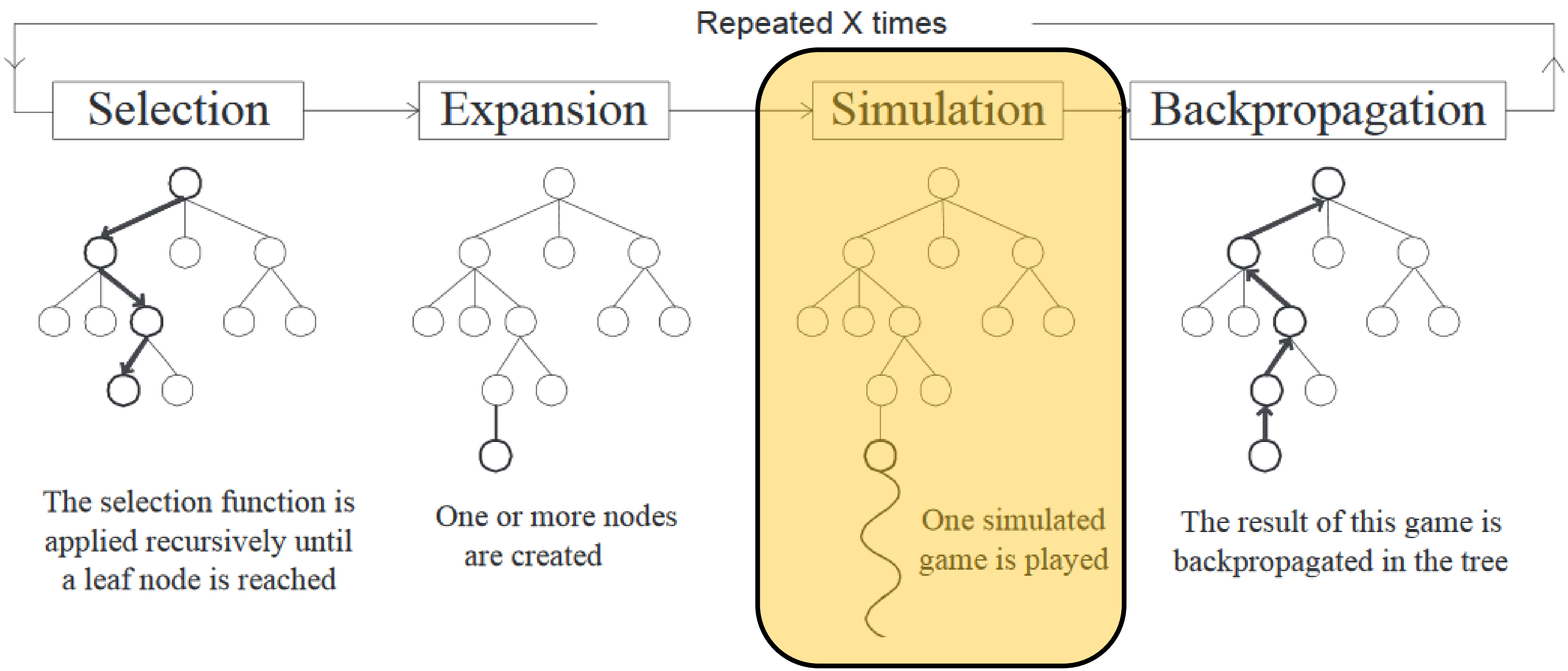
Vergleich

- Variante 1:
  - Es werden immer alle Kindknoten erzeugt
  - Kann bei hohem Branchingfaktor teuer sein

- Variante 2:
  - Nachdem mehrere Kindknoten (aber nicht alle) erzeugt wurden, kann der Wert das Elternknoten so schlecht werden, dass dieser nicht mehr besucht wird
  - Fokus geht auf anderen Bereich des Baums
  - Verbleibende Kindknoten werden nicht mehr expandiert

Variante 2 bei hohem Branchingfaktor sinnvoll

Phasen



Taken from CHASLOT et al.: PROGRESSIVE STRATEGIES FOR MONTE-CARLO TREE SEARCH, 2007.

Simulation

- Light Playout
  - In der Simulation werden vollkommen zufällige Züge gemacht

- Heavy Playout
  - Nicht (ganz) zufällig, stattdessen
    - Einfache Heuristik
    - Zufall gemischt mit Heuristiken
    - Min-Max KI mit geringer Suchtiefe
    - ...

- Vorsicht beim Heavy Playout mit Heuristik
  - Werden dabei systematisch gute Zugmöglichkeiten übersehen, so führt dies zu geringer Spielstärke

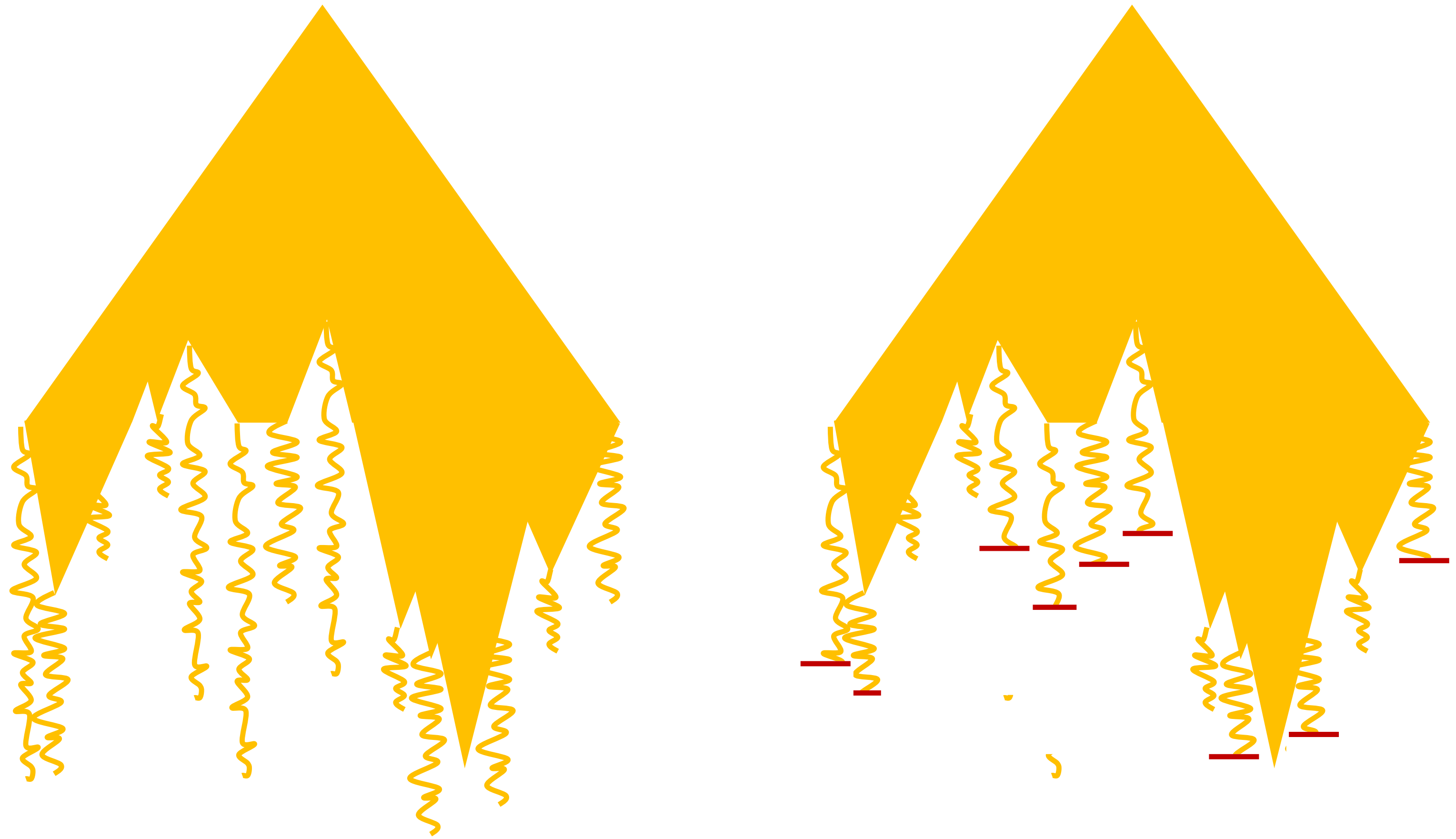
Light vs. Heavy Playout

- Heavy Playout ist besser als Light Playout bei gleicher Anzahl an Iterationen
  - Klar, zufälliges Spiel wird in der Realität nicht auftauchen
  - Statistiken, die hierauf basieren sind nicht aussagekräftig

- Aber: Heavy Playout ist in der Regel (deutlich) teurer, d.h. eine Simulation dauert (viel) länger

- Abwägen: Sind 10 vernünftige Simulationen besser als 1000 zufällige?
  - Auch hier gilt: Kann letztlich nur experimentell bestimmt werden

Ausspielen vs. Abbrechen



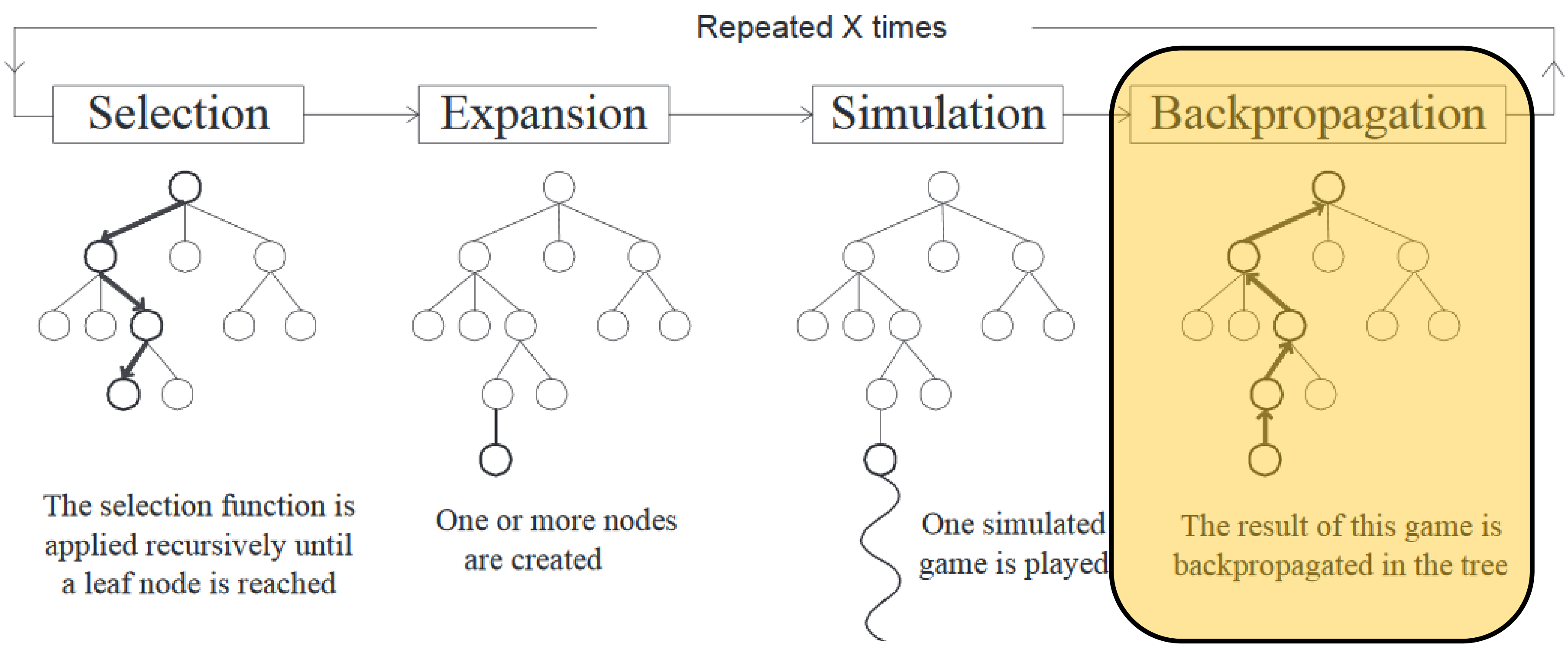
Ausspielen vs. Abbrechen

- Bei Spielen, die erst nach vielen Zügen beendet sind, kann es Sinn machen, die Simulation abzubrechen

- Bei Abbruch ⇒ Anwendung einer Heuristik
  - Entspricht (ungefähr) der Heuristik beim Min-Max Verfahren
  - Entweder, um dann Sieg oder Niederlage zu schätzen
  - Oder um Heuristikwert „nach oben zur propagieren“

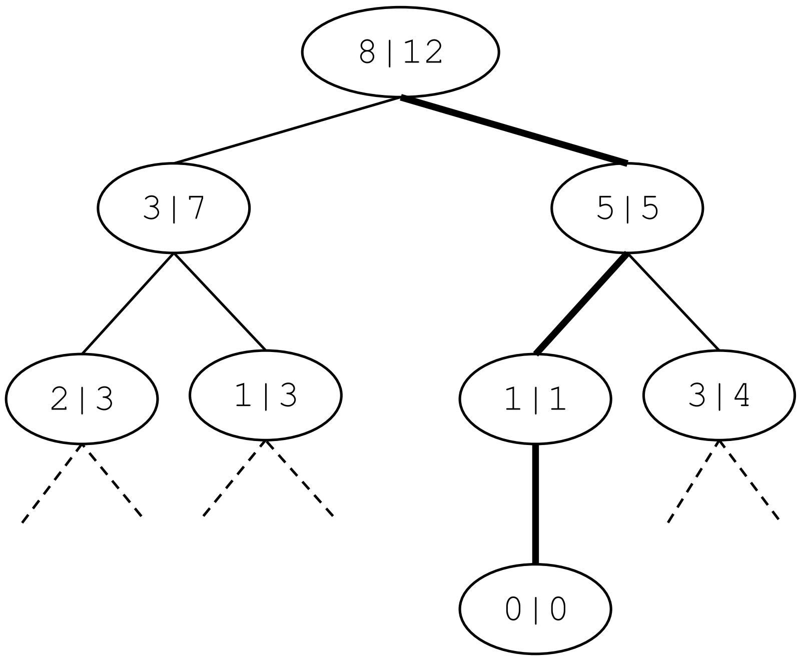
- Wann genau kommt die Heuristik zum Einsatz?
  - Wenn bestimmte Suchtiefe erreicht ist oder,
  - Wird alle m Züge berechnet
    - Klarer Ausschlag wird als Sieg oder Niederlage gewertet
    - Kein klarer Ausschlag => Simulation fortführen

Phasen

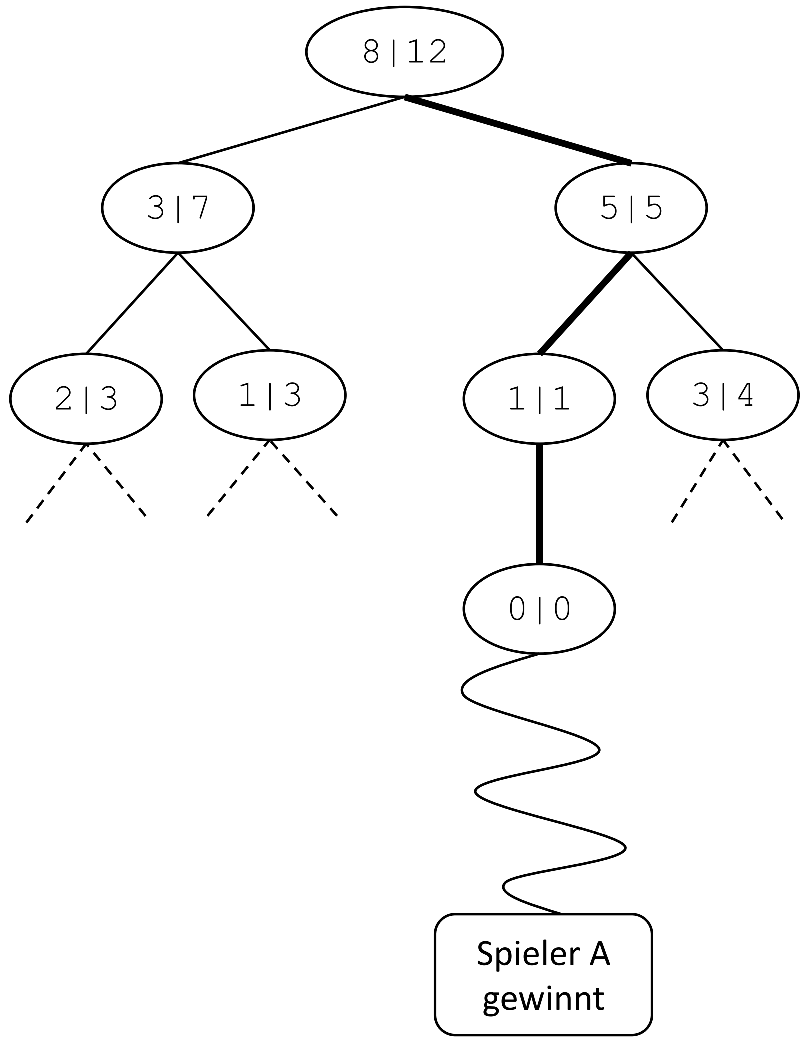


Taken from CHASLOT et al.: PROGRESSIVE STRATEGIES FOR MONTE-CARLO TREE SEARCH, 2007.

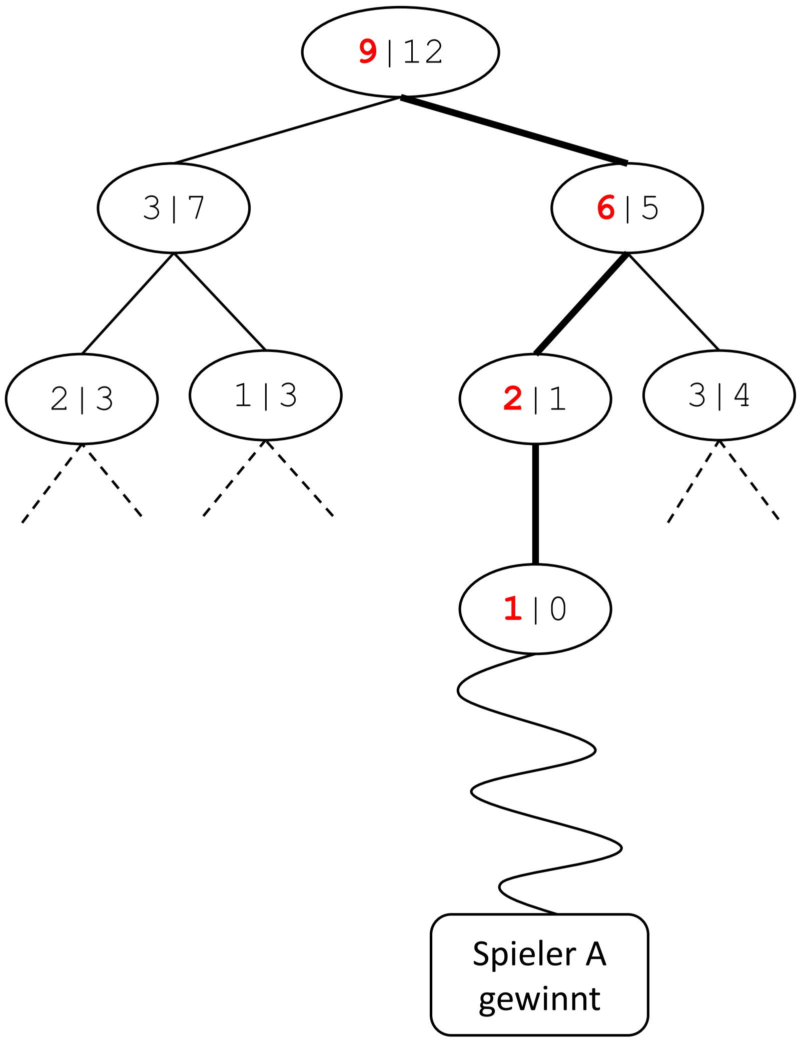
MCTS: Backpropagation



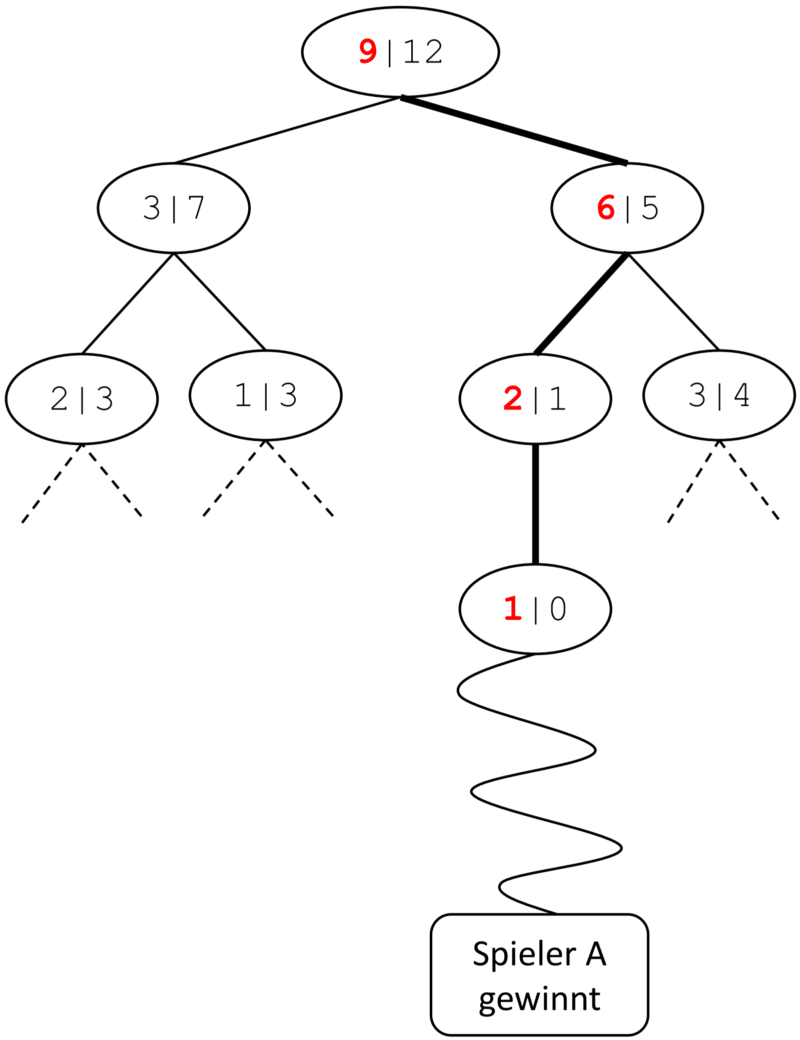
MCTS: Backpropagation



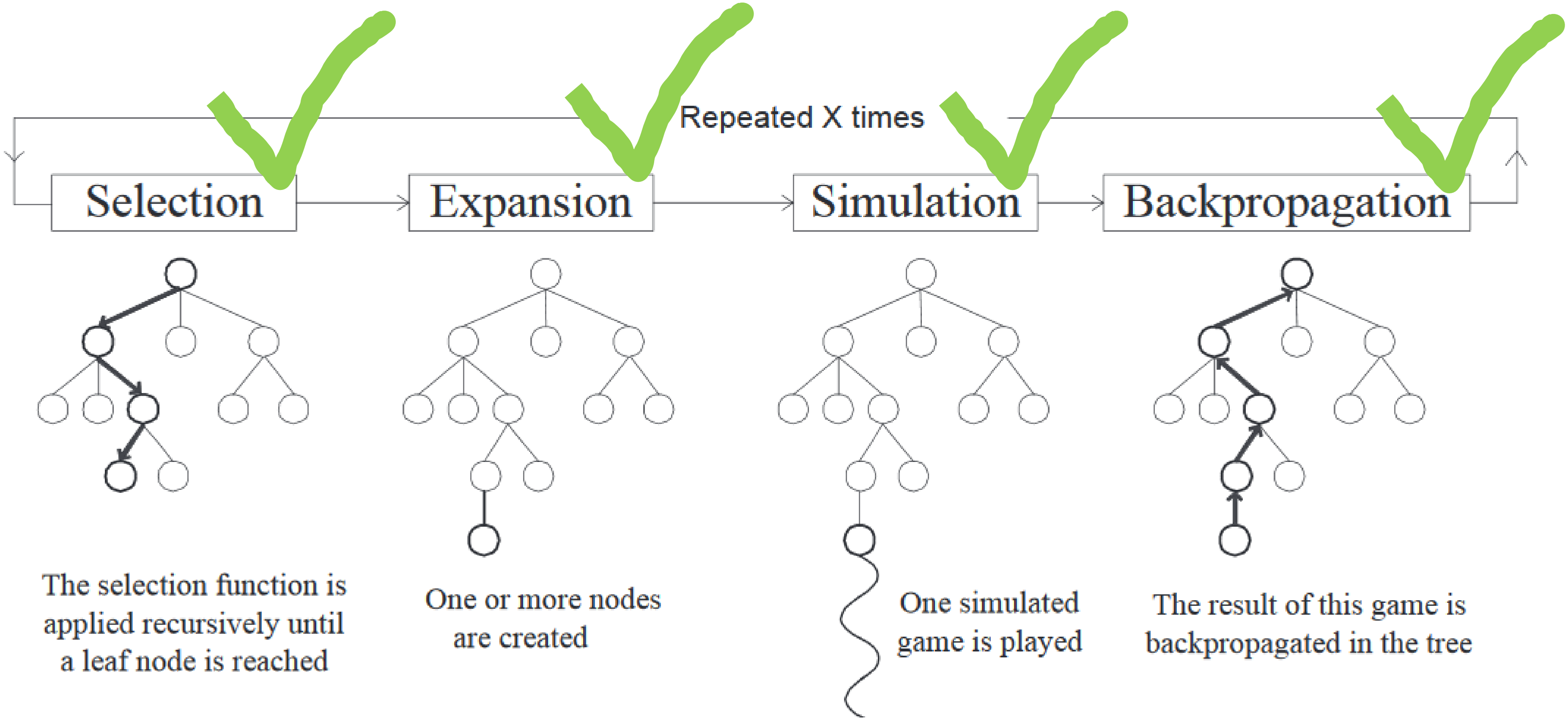
MCTS: Backpropagation



MCTS: Backpropagation



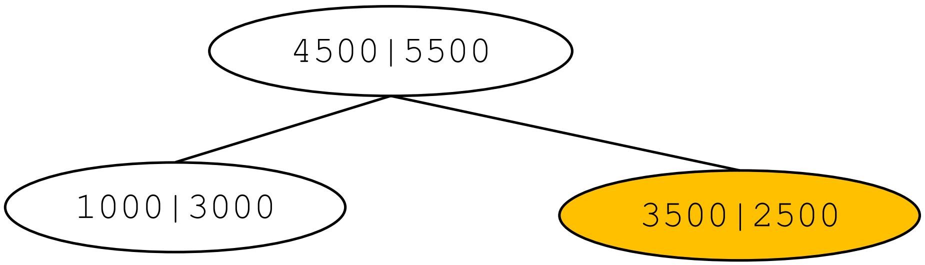
Phasen



Taken from CHASLOT et al.: PROGRESSIVE STRATEGIES FOR MONTE-CARLO TREE SEARCH, 2007.

Auswahl der Handlung

- Bei Sieg/Niederlage Spielen wird die Handlung gewählt, die zu dem Knoten mit den meisten Besuchen führt
  - Ist in der Regel auch der Knoten mit der höchsten Gewinnrate



- Eventuell andere Strategie bei Spielen mit verschiedenen Ausgängen
  - Ziel kann es sein ein möglichst geringes Risioko einzugehen
  - Andere „Statistiken“ müssen gespeichert werden

Zwischenfazit

- Verfahren ähnelt bis zu einem gewissen Grad dem Min-Max Verfahren
  - Propagieren der Simulationsergebnisse
    ~ Propagieren der Heuristikwerte bei Min-Max
  - UCT Entscheidungen (wenn Exploration ignoriert wird) ~ Abwechselnde Zugwahl durch Min/Max Spieler
- Unterschiede
  - Vielversprechende Bereiche im Suchbaum werden intensiver durchsucht
  - Bei Light-Playout wird keinerlei Wissen über das Spiel benötigt
  - Erweiterbar auf partiell beobachtbare Spiele, z.B. Kartenspiele

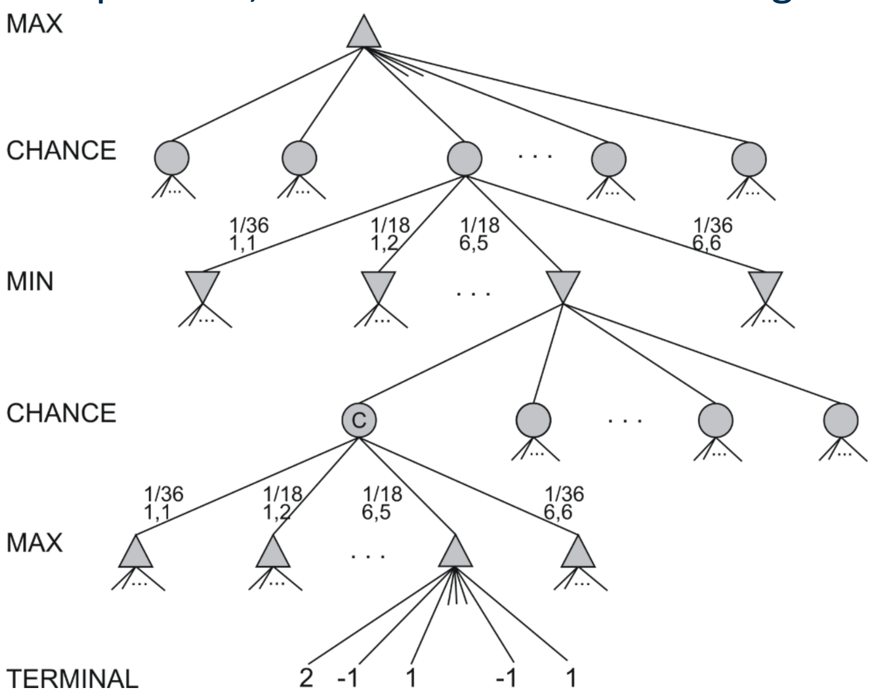
Nicht-deterministische und nur partiell beobachtbare Spiele

- Problem bei Kartenspielen: Karten des Gegners unbekannt (ebenso die Karten auf dem Stapel)
  - Zugmöglichkeiten des Gegners sind somit unbekannt
  - Bei der Expansion und Simulation ein Problem

- Problem bei Würfelspielen: Wir wissen nicht wie die Würfel des Gegners fallen werden
  - Zugmöglichkeiten des Gegners sind somit unbekannt
  - Bei der Expansion und Simulation ein Problem

Erinnerung

- Bisher hatten wir nur ExpectiMax kennengelernt
  - Min-Max mit Erweiterung um Chance Nodes
  - Suchraum explodiert, kann oft nicht effizient angewendet werden



Einfacher Ansatz: Determinisierung

- Würfelspiel: Erzeuge n zufällige Abfolgen von Würfelergebnissen
- Kartenspiel: Erzeuge n zufällige Verteilungen der verbleibenden Karten auf Gegner und Stapel

- Wende MCTS m-mal auf jedes der n vollständig beobachtbaren und deterministischen Probleme an

- Aggregiere die erste Ebene der Knoten der m MCTS Bäume und entscheide auf dieser Basis
  - Die erste Ebene hat bei allen Bäumen dieselben Knoten
  - Erst danach unterscheiden sich die Bäume

- Auch bekannt im Kontext der Bezeichung „Perfect Information Monte“ Carlo (PIMC)

Unvollständige Information vs. Nicht-Deterministisch

- Würfelspiel ist nicht deterministisch
  - Weder A noch B wissen, wie die Würfel im weiteren Spielverlauf fallen
  - **Der Zugang zu diesem Wissen ist prinzipiell nicht gegeben**

- Beim Kartenspiel sind nicht alle relevanten Informationen bekannt
  - A kennt eigene Karten und hat Vermutungen über die Karten von B
  - B kennt eigene Karten und hat Vermutungen über die Karten von A

- Grundlegender Unterschied zwischen Würfel- und Kartenspiel
  - Beim Kartenspiel sind möglichst umfangreiche und korrekte Vermutungen wichtig
  - Spezielle Erweiterungen/Anpassungen notwendig
    1. Um Vermutungen zu **generieren**
    2. Um Vermutungen **auszunutzen**

Modell des Gegners

1

- Vergangene Züge des Gegners erlauben (probabilistische) Schlussfolgerungen über die Hand des Gegners
- Beispiel:
  - Gegner hätte wichtigen Stich verhindern können, wenn er Karte X oder Y gehabt hätte
  - Er hat X oder Y nicht gelegt, und damit den Stich verloren
  - Also: Er hat (wahrscheinlich) weder X noch Y

2

- Kann bei der Determinisierung ausgenutzt / berücksichtigt werden
  - Nur solche Determinisierungen erzeugen, die zum Wissen über den Gegner passen
  - Beispiel: Verwerfe Determinisierung bei der Gegner X oder Y hat

Offene Probleme

- Es gibt Aktionen, die ausgeführt werden, um Wissen zu erhalten
  - Beispiel, Spiel, bei dem man Stiche sammelt: Eine niedrige Karte wird angespielt, um zu sehen ob der Gegner noch Trumpf hat
  - Solche Aktionen können unmöglich in einer Determinisierung „modelliert werden“
- Strategy Fusion: Unvereinbare Strategien werden vermischt
  - In einer Determinisierung kann eine zentrale Strategie gespielt werden, die unvereinbar ist mit der Strategie, die in einer anderen Determinisierung gespielt wird (und dort zu den besten Ergebnissen führt)
  - Dennoch werden die resultierenden Statistiken aggregiert
- Unnötige (doppelte) Berechnungen
  - Zwei Determinisierungen können sich in relativ unwesentlichen Teilen unterscheiden
  - Teile der MCTS Bäume werden werden zweimal zweimal identisch aufgebaut
- **Bestimmte Aspekte der „Unvollständigen Information“ können nicht durch Determinisierung abgebildet werden**

Zusammenfassung

- Min-Max vs. MCTS (siehe auch folgende Folie)
- Grundprinzip als Abfolge von vier Schritten / Datenstruktur
  - Selection mittels UCT
  - Expansion
  - Simulation (“playout”)
  - Backpropagation
- Anwendung auf Würfel- und Kartenspiele
  - Z.B. Determinisierung

- MCTS ist eine Familie neuerer Algorithmen die ab ca. 2006 intensiver untersucht wurden
  - Zum Teil (vielleicht nur dem Dozenten) nicht ganz klar unter welchen Bedingungen MCTS in welcher Variante gut funktioniert

Fragen

Monte Carlo

Tree