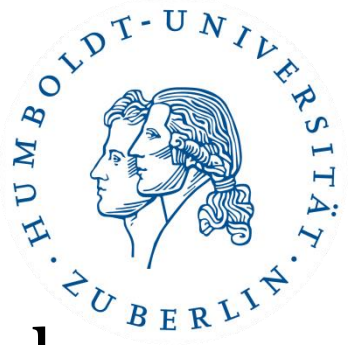


Semesterprojekt
Implementierung eines Brettspiels
(inklusive computergesteuerter Spieler)

Wintersemester 16/17

(Kleine) Einführung zu
Künstlicher Intelligenz in Brettspielen

Patrick Schäfer
patrick.schaefer@hu-berlin.de

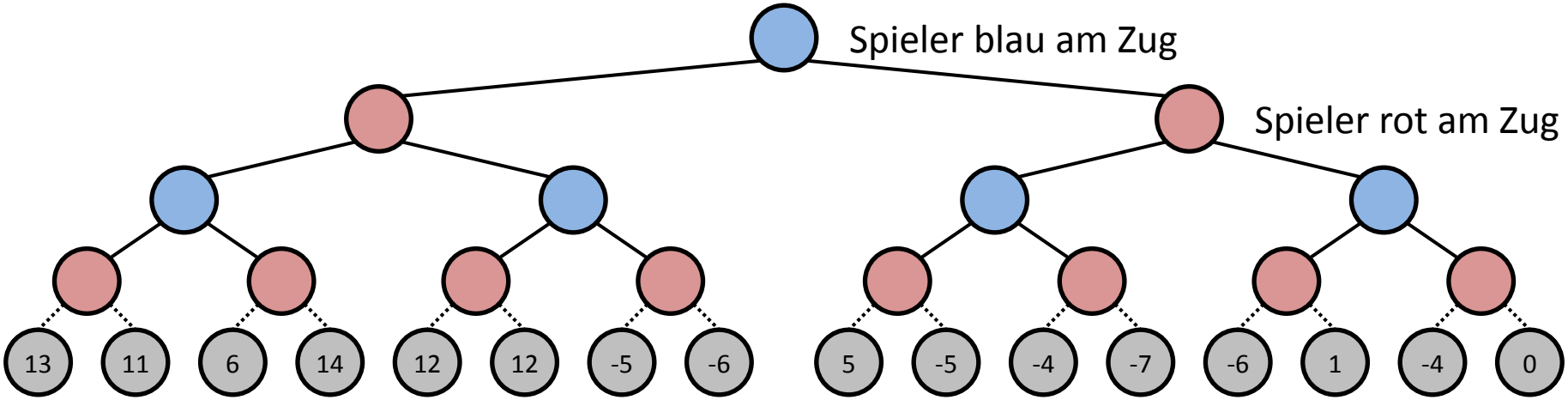
Marc Bux
buxmarcn@informatik.hu-berlin.de

Herangehensweise

- Einstieg: Blog-Post zu Brettspiel-KI in Unity
  https://madewith.unity.com/stories/boardgame-ai
- hilfreiche Werkzeuge:
  - Transparenz: Denkweise der KI widergeben, um Probleme zu erkennen
  - Batch-Modus: Neue Builds wiederholt (1000 Mal) gegen alte Builds antreten lassen, um Verbesserungen zu messen
- regelbasierter Ansatz:
  - Spielsituation analysieren, Regeln durchgehen
  - wende Regel mit höchster Priorität an, deren Vorbedingung erfüllt ist
  - wenn keine Regel greift, auf Standard-KI ausweichen
- Workflow: Spiel gegen KI – neue Ideen implementieren – Test im Batchmodus – Änderungen ggf. übernehmen

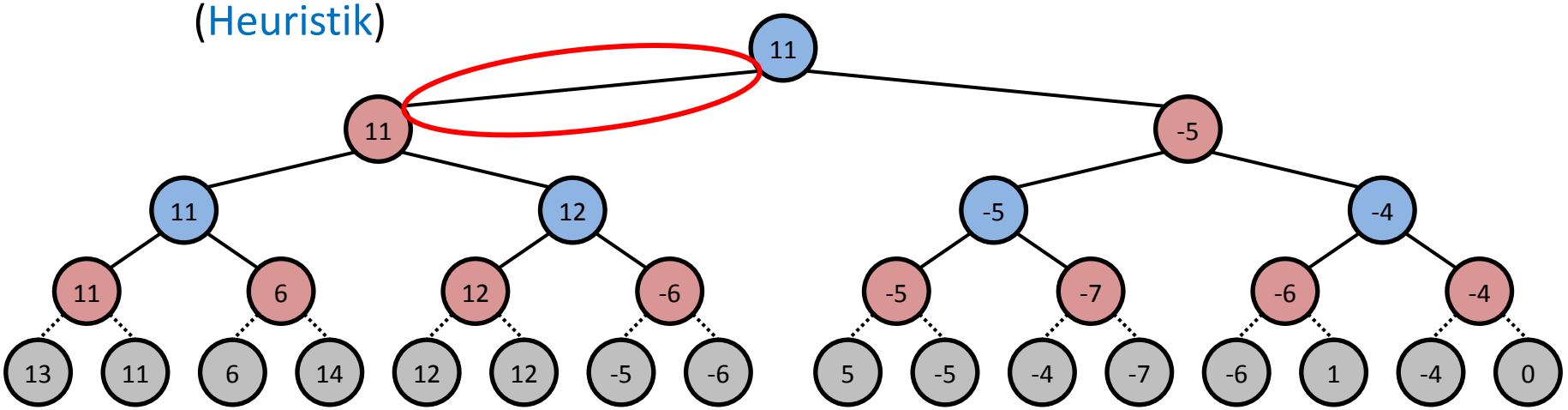
Entscheidungsbäume

- Spiele können als Bäume modelliert werden:
  - jeder Knoten repräsentiert einen Zustand des Spiels
  - jede Kante repräsentiert einen möglichen Spielzug
  - jedes Blatt repräsentiert ein mögliches Spielende und lässt sich mittels Gütefunktion (Utility) bewerten
- Strategien zur Lösungsfindung (Auswahl):
  - Minimax (für 1on1-Nullsummenspiele)
  - Monte-Carlo-Sampling



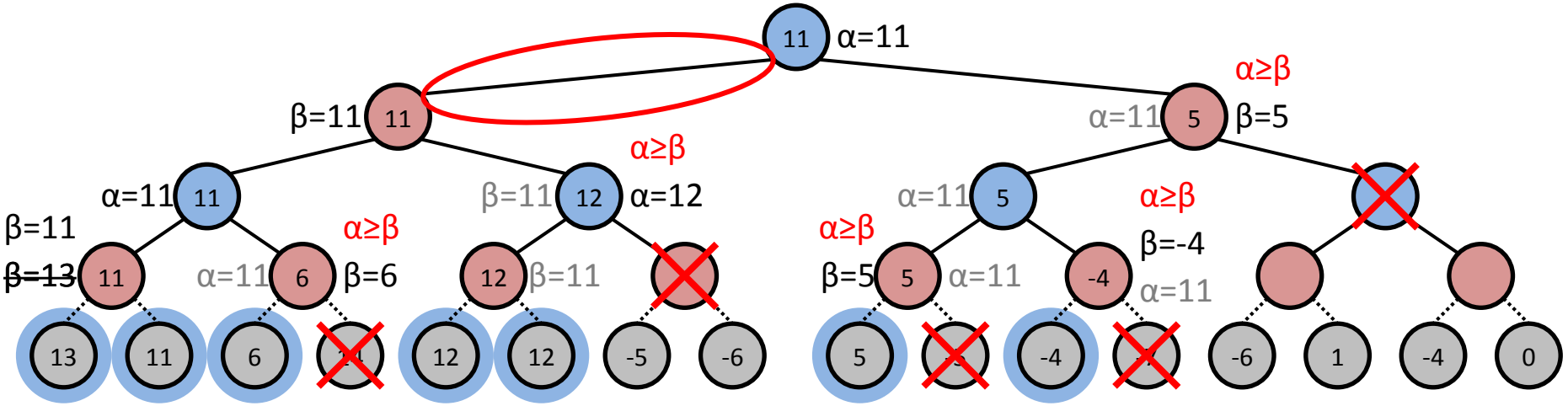
Minimax

- Idee: minimiere Verlust bei maximal guten Zügen des Gegners
- Vorgehensweise:
  - propagiere Bewertungen der Blätter von unten nach oben
  - alterniere zwischen Minimum und Maximum
  - wähle Kind der Wurzel mit höchstem Wert
- wenn k Spielzüge zur Auswahl stehen, so ergibt sich beim Vorausschauen von n Spielzügen ein Baum mit Größe $O(k^n)$
  - Alpha-Beta-Pruning (Optimierung), Beschränkung des Suchhorizonts (Heuristik)



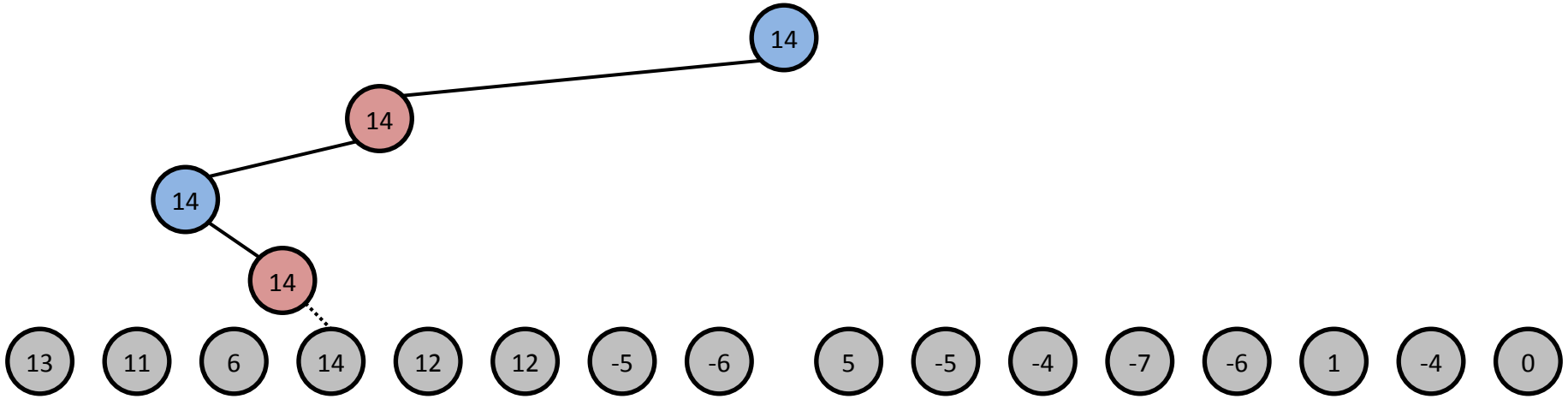
Alpha-Beta-Pruning

- Idee: Knoten, von denen wir bereits wissen, dass sie suboptimal sind, brauchen nicht weiter betrachtet zu werden
- Vorgehensweise:
  - Tiefensuche zu den Blättern, beginnend bei der Wurzel
  - Beim Backtracking werden Knoten alternierend mit mit α- und β-Werten versehen
  - α-Wert: maximaler von Spieler Blau garantierbarer Utility-Wert
  - β-Wert minimaler von Spieler Rot garantierbarer Utility-Wert
  - Beim Absteigen werden α- und β-Werte nach unten weitergereicht
  - Verwerfe Äste ab Knoten, für die gilt α≥β



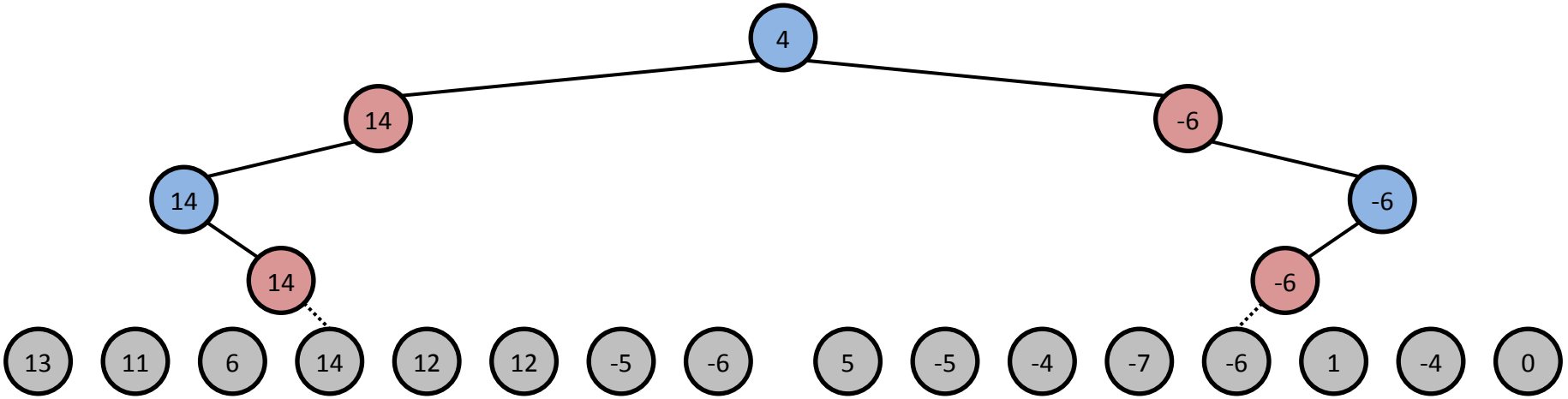
Monte-Carlo-Sampling

- Idee: Erweitere Entscheidungsbaum durch stichprobenartige Simulationen von Spieldurchläufen mit zufälligen Zügen
- Exploration vs. Exploitation / UCT-Formel: An Knoten n wähle Nachfolger n_i, für den $\frac{w_i}{s_i} + c\sqrt{\ln t / s}$ maximal ist
  - w_i: Anzahl bisher ermittelter Siege bzw. zu erwartender Punktestand ab Knoten n_i
  - s_i: Anzahl bisher durchgeführter Simulationen ab Knoten n_i
  - c: Erkundungs-Parameter (z.B. $\sqrt{2}$)
  - s: Anzahl bisher durchgeführter Simulationen ab Knoten s



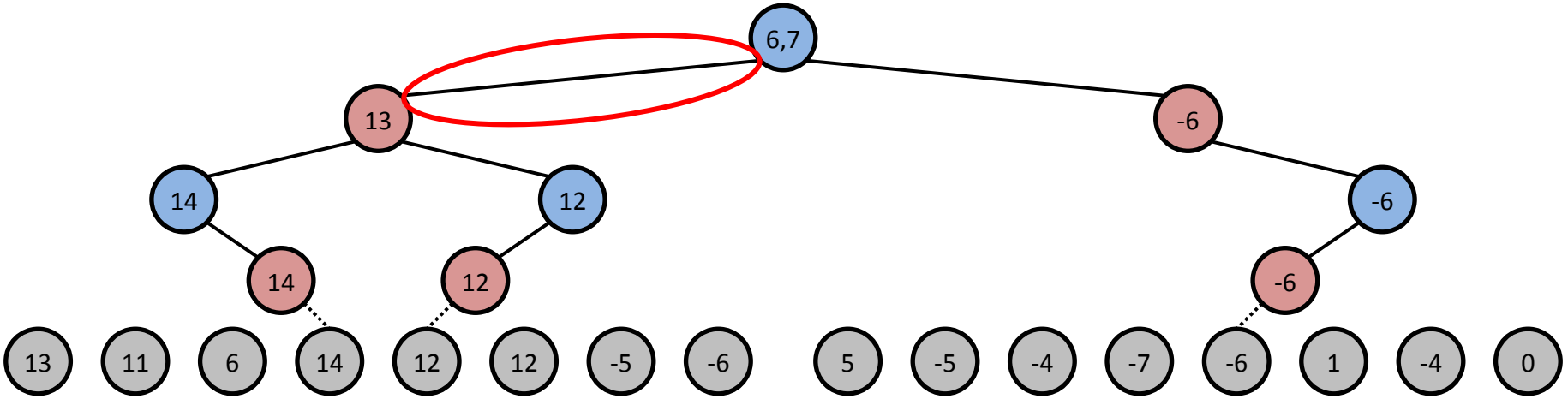
Monte-Carlo-Sampling

- Idee: Erweitere Entscheidungsbaum durch stichprobenartige Simulationen von Spieldurchläufen mit zufälligen Zügen
- Exploration vs. Exploitation / UCT-Formel: An Knoten n wähle Nachfolger n_i, für den $\frac{w_i}{s_i} + c\sqrt{\ln t / s}$ maximal ist
  - w_i: Anzahl bisher ermittelter Siege bzw. zu erwartender Punktestand ab Knoten n_i
  - s_i: Anzahl bisher durchgeführter Simulationen ab Knoten n_i
  - c: Erkundungs-Parameter (z.B. $\sqrt{2}$)
  - s: Anzahl bisher durchgeführter Simulationen ab Knoten s



Monte-Carlo-Sampling

- Idee: Erweitere Entscheidungsbaum durch stichprobenartige Simulationen von Spieldurchläufen mit zufälligen Zügen
- Exploration vs. Exploitation / UCT-Formel: An Knoten n wähle Nachfolger n_i, für den $\frac{w_i}{s_i} + c\sqrt{\ln t / s}$ maximal ist
  - w_i: Anzahl bisher ermittelter Siege bzw. zu erwartender Punktestand ab Knoten n_i
  - s_i: Anzahl bisher durchgeführter Simulationen ab Knoten n_i
  - c: Erkundungs-Parameter (z.B. $\sqrt{2}$)
  - s: Anzahl bisher durchgeführter Simulationen ab Knoten s



Weiterführende Links

- Vorlesung „General Game Playing“ von Sebastian Wandelt im WS 13/14:
  https://hu.berlin/vl_game

- Ticket to Ride in Java mit implementierter (aber unkommentierter) Minimax- und Monte-Carlo-Baumsuche
  https://github.com/cooijmanstim/hobo

- Reddit-Thread mit allerlei nützenswerten Informationen und Erfahrungswerten:
  https://www.reddit.com/r/gameai/comments/44l6w7/building_an_ai_for_a_multiplayer_turnbased_board/

KI-Challenge: Ablauf

- eine User Story, einwöchige Sprints, kurze Tech. Refinements
- erste lauffähige KI zum 16.1.; wöchentliche Verbesserungen
- wöchentliches 1on1-Turnier (blau-gelb, blau-rot, gelb-rot)
- zunehmende Punkte für erfolgreiche Teilnahme und Platzierung
  - 16.1.: 1-1-2, 23.1.: 1-2-3, 30.1.: 2-3-5, 6.2.: 3-5-8, 13.2.: 5-8-13
  - 0 Punkte, wenn KI wiederholt abstürzt oder Spiel nicht terminiert
- optional: zusätzliches Turnier für Einzelabgaben
  - K.O.-System; Beginn im Halbfinale
  - vorzugsweise (aber nicht zwingend) 1on1
- verteiltes Setting mit lokalem Netzwerk:
  - wir stellen Rechner für Spielserver und visualisieren Spiel auf Beamer
  - Turnierteilnehmer verbinden sich mit eigenen Rechnern