Thinking Machine

(http://www.turbulence.org/spotlight/thinking/)

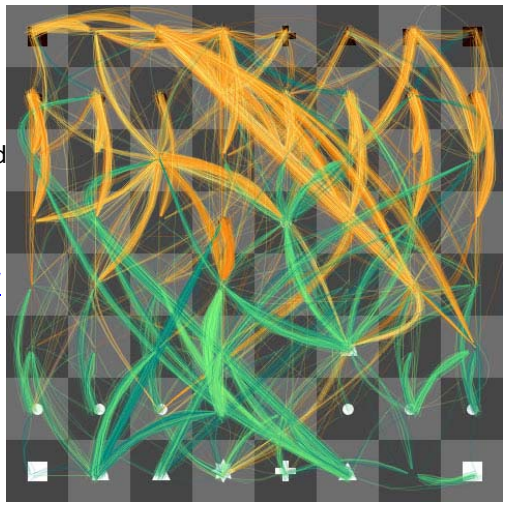
Idee

- Die Thinking Machine Visualisierung versucht, die “Denkprozesse” eines Schachcomputers sichtbar zu machen

- Sie wurde von Martin Wattenberg und Marek Walczak in processing erstellt und kann unter

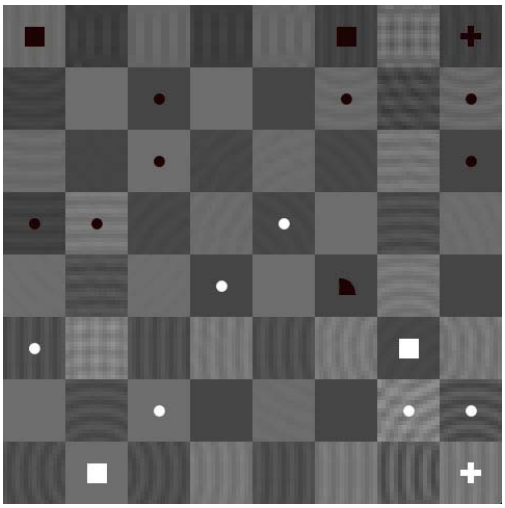
http://www.turbulence.org/spotlight/thinking/chess.html

ausprobiert werden.



Kodierte Information

- Ist der (menschliche) Spieler am Zug zeigt das Spielfeld den Einfluss anderer Spielsteine an: Jede Spielfigur sendet Wellen aus auf Feldern wo sie schlagen kann

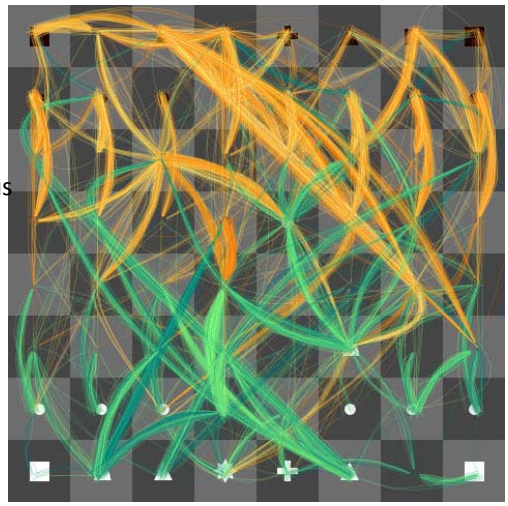


Kodierte Information II

- Ist der Computer am Zug, zeigt er mögliche Züge für sich selbst (orange) und den menschlichen Gegner (grün) an.

- Dabei werden mehrere Züge im voraus berechnet, so dass sich länger "Stränge" von Zügen ergeben.

- Je heller die Kurven, desto besser hält der Computer sie für Weiss (und damit schlecht für sich selbst)



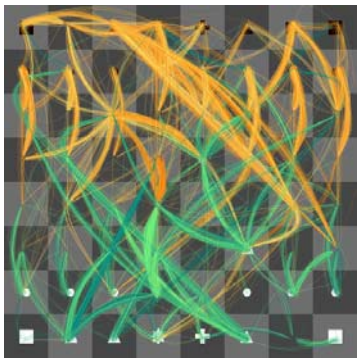
Verwendete KI

- Thinking Machine's Fokus lag nach Auskunft der Autoren auf der Visualisierung – weniger auf der künstlichen Intelligenz. Daher benutzen sie relativ simple Algorithmen

- Alpha-Beta Pruning

- Quiescence Search

- Keine Bibliothek mit Startzügen



Alpha-Beta Pruning

- Alpha-Beta Pruning ist ein Suchverfahren das mit Suchbäumen für Spielsituationen arbeitet.

- Es basiert auf dem – etwas einfacheren – Minimax-Verfahren.

Das Minimax-Verfahren

Bei Problemen, in denen *nichtkooperative Aktoren* beteiligt sind (klassisches Beispiel: Gesellschaftsspiele wie Schach, Go, Tic-Tac-Toe), ist der Suchgraph meist extrem groß.

z.B.: Schach, Verzweigungsfaktor: ca. 35,
Züge pro Spieler: ca. 50;
Suchgraph hätte also ca. 35^{100} Knoten.

daher: Zu Beginn eines Spiels ist sogar eine heuristische Suche nach einem vollständigen Lösungsweg aussichtslos.

Ziel der Suche daher:
Bewertung des nächsten Handlungsschrittes

Allgemeines Vorgehen:
Vorausschau und *Bewertung* einiger Züge, für die eine Expansion des Suchgraphen vorgenommen wird.

Das Minimax-Verfahren

- Zwei Spieler Max und Mini, *statische Bewertungsfunktion* für Knoten *K* des Suchgraphen sei B(*K*), Knoten entspricht z.B. Brettposition bei Spielen.
  - Falls *K* Vorteil für Max: B(*K*) > 0 (Max sucht maximalen Wert)
  - Falls *K* Vorteil für Min: B(*K*) < 0 (Mini sucht minimalen Wert)
  - Bei 'ausgeglichener' Position gilt: B(*K*) etwa = 0
- Max soll als nächster agieren (wählt höchstbewerteten Knoten).
- Expandiere Suchgraph bis zur Tiefe $D = 2N - 1$, $N >= 2$.
- Bewerte alle Blätter mit der statischen Bewertungsfunktion B.
- Verteile die Bewertungen nach folgendem Schema weiter zu den Vorgängern:
  - Knoten der Tiefe $D - 1$ erhalten jeweils das Minimum der Bewertungen ihrer Nachfolger ('Min-Zug').
  - Knoten der Tiefe $D - 2$ erhalten jeweils das Maximum der Bewertungen ihrer Nachfolger ('Max-Zug').
  - Knoten der Tiefe 0 erhalten das Maximum.

Das Minimax-Verfahren

- Der Wechsel zwischen der Wahl von Knoten mit minimaler und maximaler Bewertung gab dem Minimax-Verfahren seinen Namen.

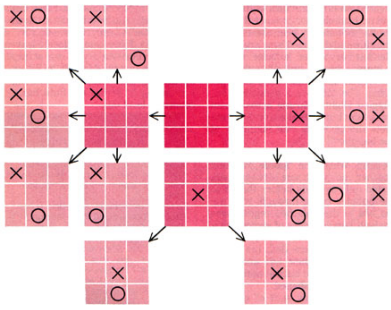
- Annahmen bei Minimax:
  - Die durch die Vorausschau erhaltene Bewertung der Nachfolger des Startknotens ist zuverlässiger als die durch direkte Bewertung der Nachfolger mit B erhaltenen Hinweise.
  - Der jeweilige Gegenspieler entscheidet sich immer rational für den bestbewerteten Zug nach dem Minimax-Verfahren.

Das Minimax-Verfahren für Tic-Tac-Toe

- Statische Bewertungsfunktion:

B(*K*) = <Anzahl der noch offenen Reihen/Spalten/Diagonalen für Max bei K>
- <Anzahl der noch offenen Reihen/Spalten/Diagonalen für Mini bei K>

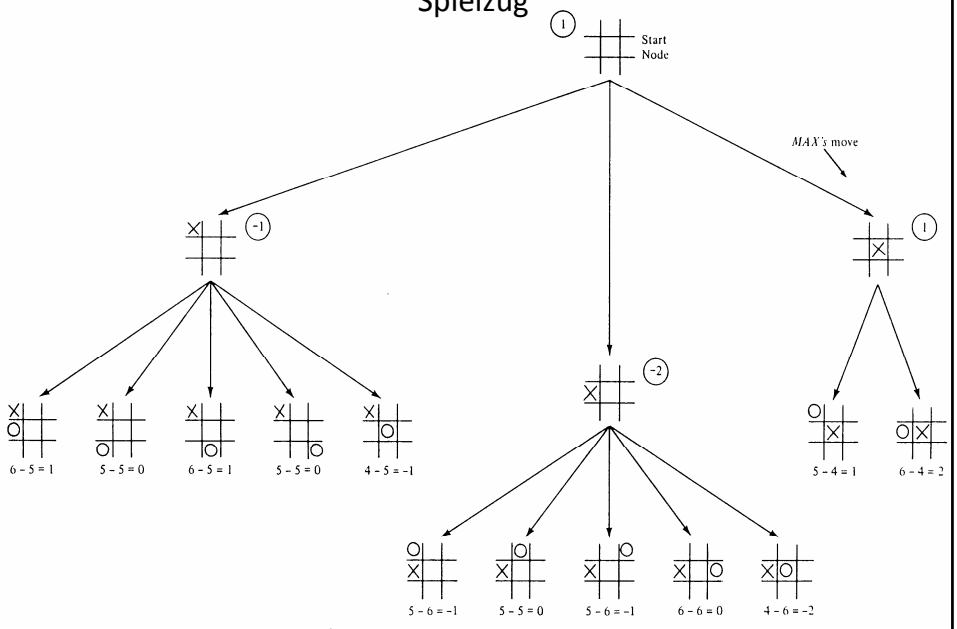
- Durch Symmetrien (Drehung und Spiegelung am Gitter), bleibt der Verzweigungsfaktor zu Beginn klein.
- Später bleibt er durch die verringerte Zahl der noch offenen Gitterplätze klein.
- Im Beispiel: Breitensuche bis alle Knoten der Tiefe 2 expandiert sind.



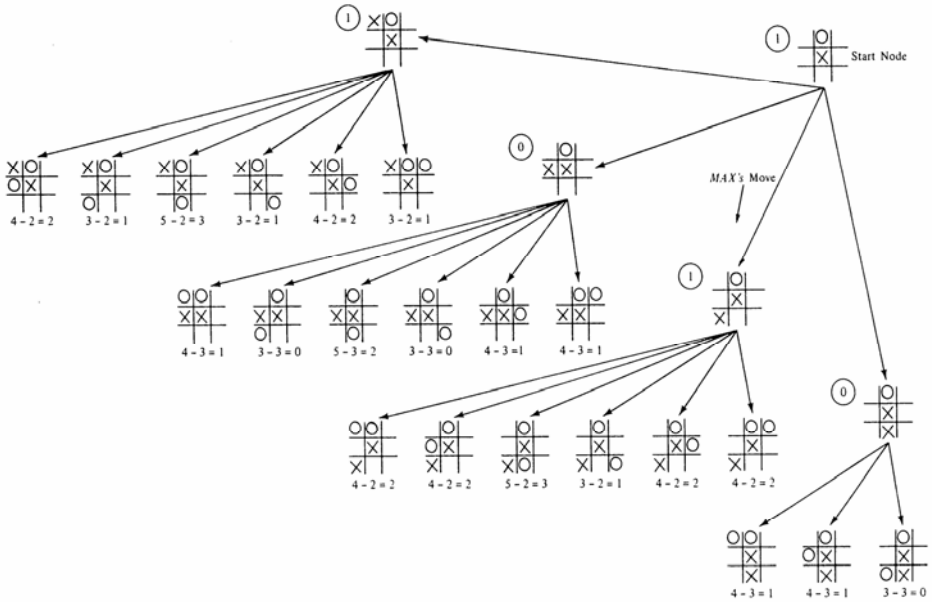
Das Minimax-Verfahren für Tic-Tac-Toe

- Nachteile:
- Eine *Zahl* als Positionsbewertung ist *wenig aussagekräftig*; sie sagt nichts darüber aus, wie sie entstanden ist.
- Hoher Aufwand von Minimax:
  - *Alle möglichen Nachfolger* werden erzeugt.
  - Für *alle möglichen Pfade* wird erst statische *Bewertungsfunktion* berechnet.

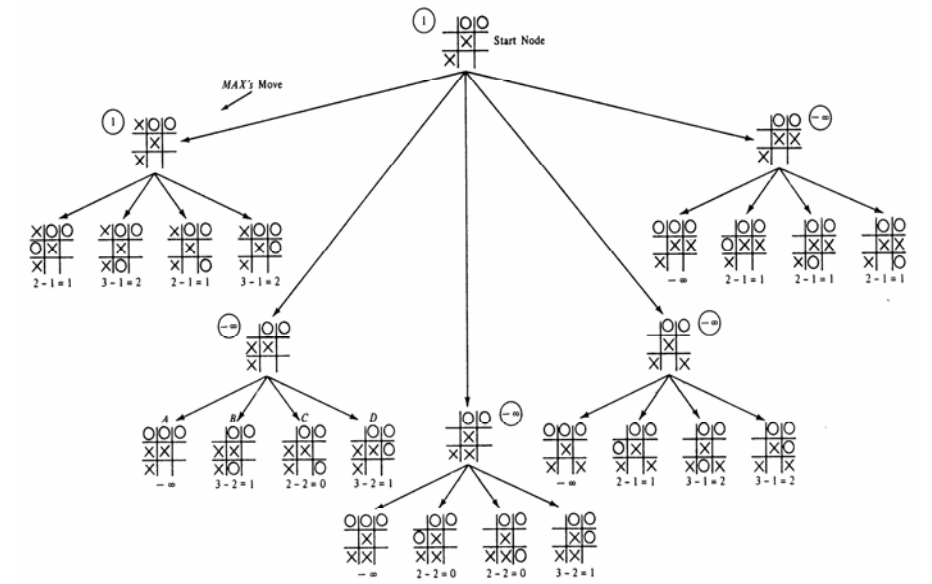
Das Minimax-Verfahren am Beispiel von Tic-Tac-Toe: erster Spielzug



Das Minimax-Verfahren am Beispiel von Tic-Tac-Toe: zweiter Spielzug



Das Minimax-Verfahren am Beispiel von Tic-Tac-Toe: dritter Spielzug



Quiescence Search

- “Horizon” Problem: Wenn der Computer z.B. fünf Züge im Voraus berechnet könnte er sich in eine ungünstige Situation für den sechsten Zug bugsieren.

- Quiescence Search löst dieses Problem dadurch, dass sie berechnet wie “ruhig” (uninteressant) ein Zug ist und für spannende Zugkombinationen tiefer in den Baum einsteigt.

- Das Maß für “ruhige” Züge ist von Spiel zu Spiel verschieden. Bei Schach kann es z.B. die Zugweite von Spielfiguren sein oder die Anzahl von geschlagenen Figuren.

Quelle: http://en.wikipedia.org/wiki/Quiescence_search