Search in Games

Artificial Intelligence (DD2380)

KTH – Royal Institute of Technology
Autumm 2021

Zero-sum games

- **Utility function** $\psi(s, p)$: numerical value for a game ending at state s by player p.
- **Zero-sum game**: game where the sum of the utility function for all players is zero. That is: $\sum_i \psi(s, p_i) = 0$.
  - Chess example: $\psi(\texttt{win}, p) = 1$, $\psi(\texttt{loss}, p) = -1$, and $\psi(\texttt{draw}, p) = 0$.
- We consider **2-player**, **adversarial** games (no collaboration).
- We consider **perfect information** games: **deterministic** and **fully observable** environments.

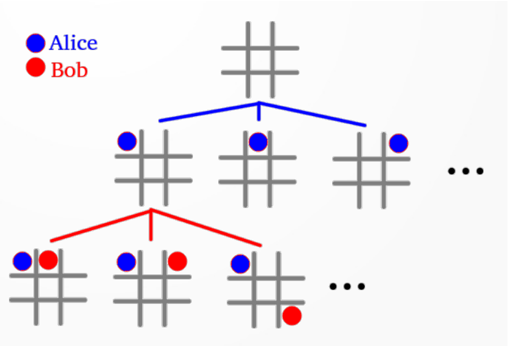
Zero-sum games

Tree representation:

- Nodes are states.
- Edges are moves (actions).

Optimizing next move:

- Creating the game tree in the background.
- Assume the opponent is as smart as you.
- Search for the move that reaches the best state.



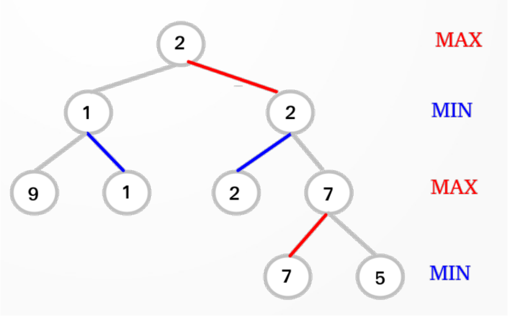
Mini-Max Algorithm

- Strategy to find the best move.
- Two players taking turns: `MAX` and `MIN`.
    - `MAX` wants to **maximize** their gain.
    - `MIN` wants to **minimize** the other's gain.
- We want to explore the game tree to find the best outcome: $\longrightarrow$ **Search!**
    - Depth First Search (DFS).

Two player types:

- MAX: selects highest value state.
- MIN: selects lowest value state.

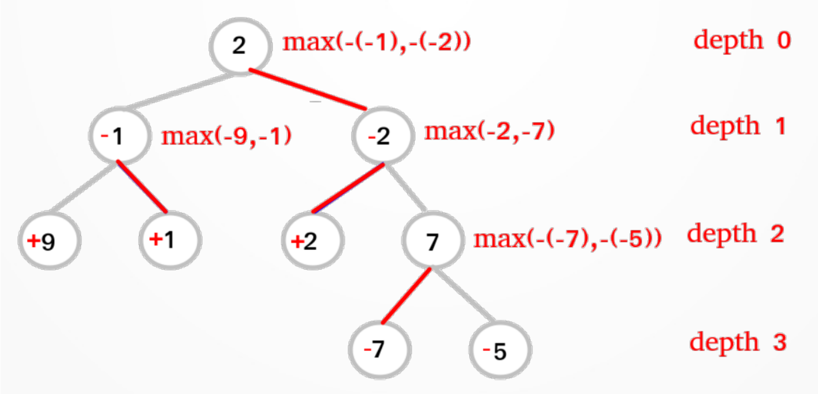
We assume both players are **equally smart**!



Nega-Max Algorithm

- Strategy to find the best move.
- We exploit the fact that **max**$(-a, -b) = -$**min**(a, b).
- We **negate utilities at odd depths** and always take **max**$(-\texttt{child}_1, -\texttt{child}_2)$.
- We want to explore the game tree to find the best outcome: $\rightarrow$ **Search!**
  - Depth First Search (DFS).

Nega-Max Algorithm



- We exploit the fact that **max**$(-a, -b) = -$**min**(a, b).
- We **negate utilities at odd depths** and always take **max**$(-\texttt{child}_1, -\texttt{child}_2)$.

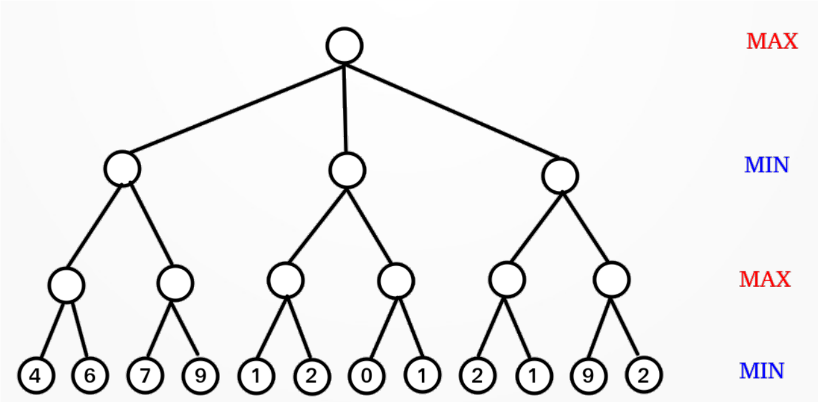
Alpha-Beta Prunning

- Some values are useless: Their value will not make any effect.
  - Example: **min**(**max**(7, **min**($F(\cdot)$)), 2) $= 2$ regardles of what $F(\cdot)$ returns.
- α: Highest value found along the path to the root for `MAX` $\rightarrow$ **Lower bound**
- β: Lowest value found along the path to the root for `MIN` $\rightarrow$ **Upper bound**
- Prune when $\beta \leq \alpha$. Why?
  - For `MAX`: **max**(α, **min**(β, $\texttt{sth}$)) $= \alpha$.
  - For `MIN`: **min**(β, **max**(α, $\texttt{sth}$)) $= \beta$.

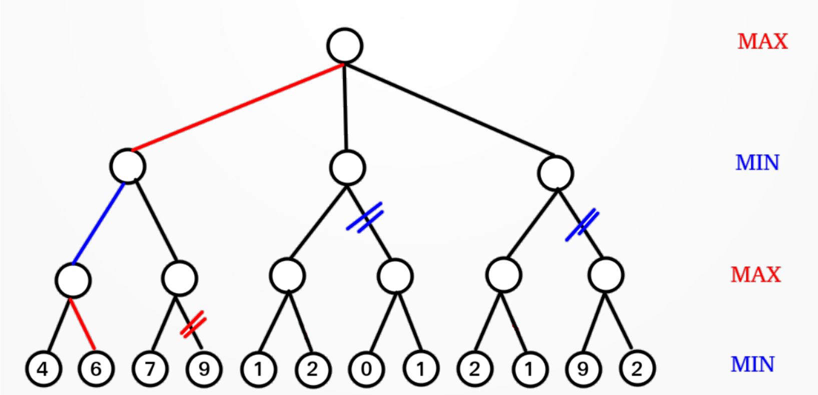
Alpha-Beta Prunning

- α and β values are propagated down the tree.
- α and β values are **never** propagated up the tree.
- α values are evaluated in `MAX` nodes.
- β values are evaluated in `MIN` nodes.

Alpha-Beta Prunning



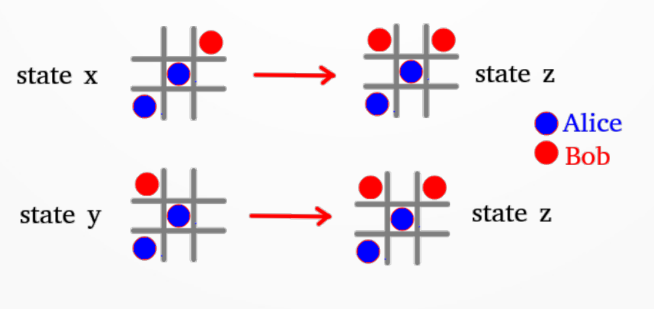
Alpha-Beta Prunning



Repeated States

- Search algorithms can visit the same state via different sequence of moves $\rightarrow$ **Computationally expensive**.
- We can implement a data structure to avoid calculating the same states $\rightarrow$ **Hash function** and **transposition table**.
- Which states are equivalent?
  - Same placement of pieces.
  - **Symmetric** placements.
  - Is it important **who** made the move?

- Example: **tic-tac-toe**



Scoring Function / Heuristic

- **Heuristic** or **scoring function** $h(s,p)$: **experience-based** approximation of the utility function: $h(s,p) \approx \psi(s,p)$.
- To calculate the utility function we need to explore the tree until the leaves $\rightarrow$ Potentially **very computationally expensive**.
- There are many number of heuristics and strategies $\rightarrow$ Usually none is optimal.
- Example: **checkers**
  - Count the number of your pieces and substract the enemy pieces.

The problem with DFS

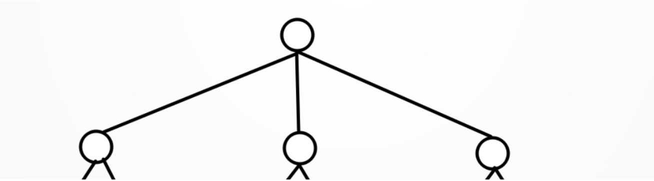
- In some games, the depth of the state is very large (almost infinite).
- Possible solution: **serach to a fixed depth**.
- But, what is the appropriate depth? And what if the branching factor is very big?
  - Branching factor of chess: ≈ 35.
  - Branching factor of Go: ≈ 250.

Iterative Deepening Search (IDS)

Iterate the search depth: start at depth 0, then 1, then 2, and so on until the resources run out.

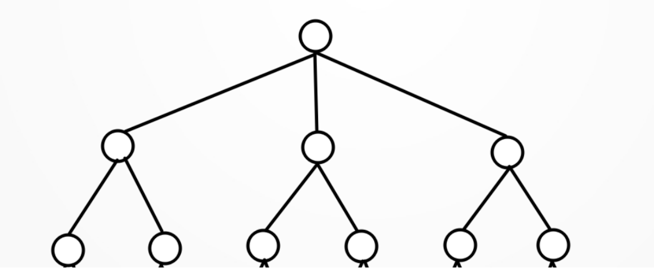
Iterative Deepening Search (IDS)

Iterate the search depth: start at depth 0, then 1, then 2, and so on until the resources run out.



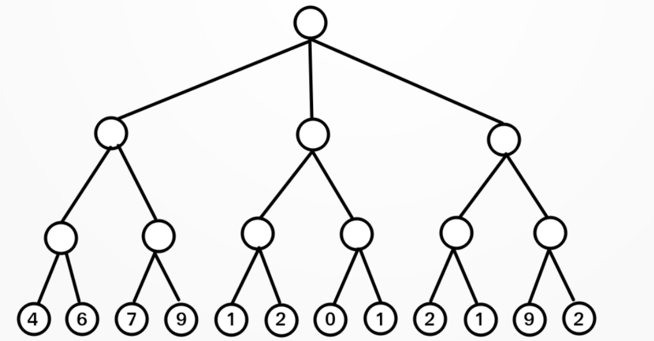
Iterative Deepening Search (IDS)

Iterate the search depth: start at depth 0, then 1, then 2, and so on until the resources run out.



Iterative Deepening Search (IDS)

Iterate the search depth: start at depth 0, then 1, then 2, and so on until the resources run out.



IDS and move ordering

Order the nodes at depth $d + 1$ based on the estimation from depth d:

- If better nodes are searched first, **AlphaBeta** will prune large portions of the tree $\rightarrow$ **faster exploration**

Tips and Questions

1. Make sure your **MiniMax**/**NegaMax** works.
2. Make sure your **Alpha-Beta** prunning works.
3. Check for **repeated states**.
   - Apply symmetry breaking.
4. Implement **iterative deepening** for exploration.
5. Almost EOG? Search until the end.
6. Implement move ordering for **Alpha-Beta** pruning.
7. Implement an ending states **look-up table**.

Tips and Questions

- Kattis has a time limit, return the best in hand when time is up.
- Questions?
- **Have fun! :)**.