COMP 4752
Computational Intelligence

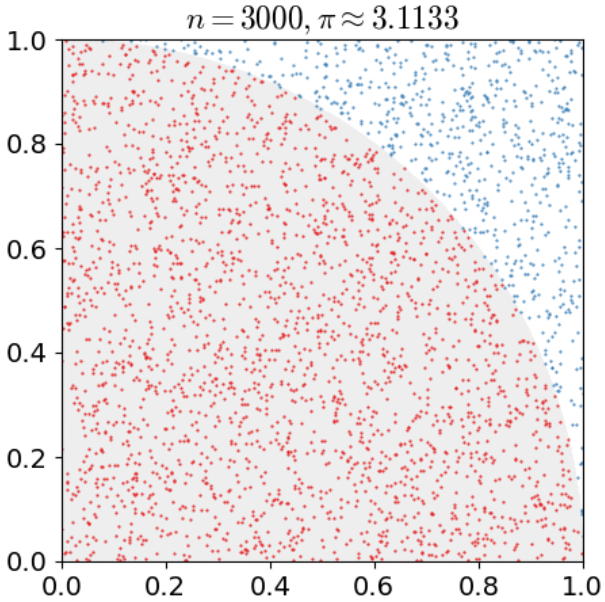
Lecture 23

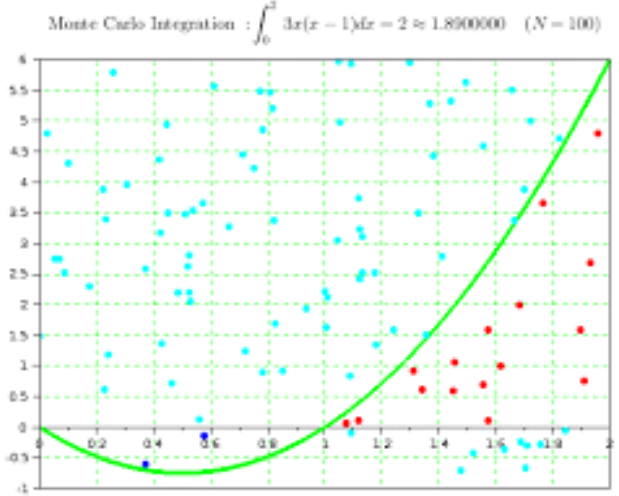
Monte-Carlo Tree Search (MCTS)

Monte-Carlo Methods

- Class of methods or algorithms that relies heavily on random sampling
- Very useful when traditional approaches cannot be applied to a problem
- Main idea: Over the course of many random samples, the true values will eventually emerge, decide on best action

Example Monte-Carlo Method

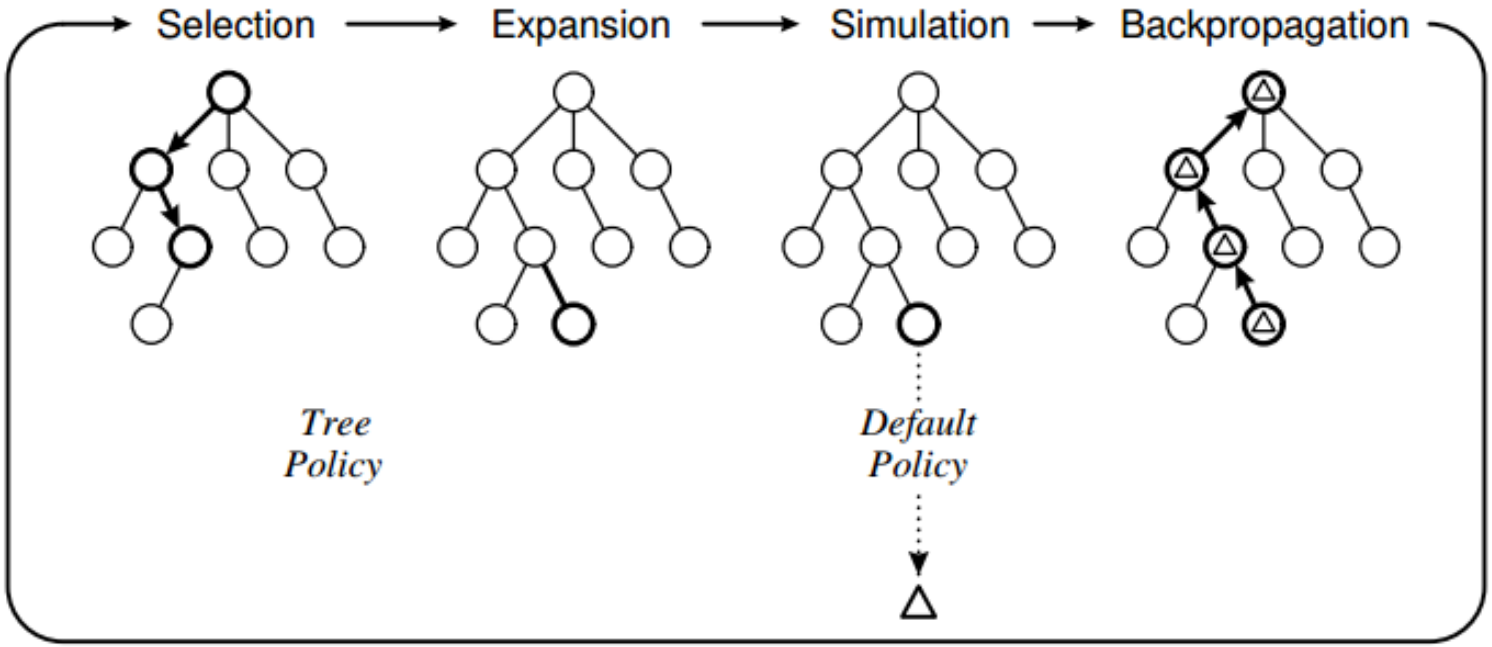




Monte-Carlo Tree Search

- Search starts at the root node
- Perform a number of traversals of the tree
- On each traversal we perform a number of steps which leads to tree expansion
- Simulation is performed to determine the quality of a node for a given player
- MCTS consists of 4 main steps

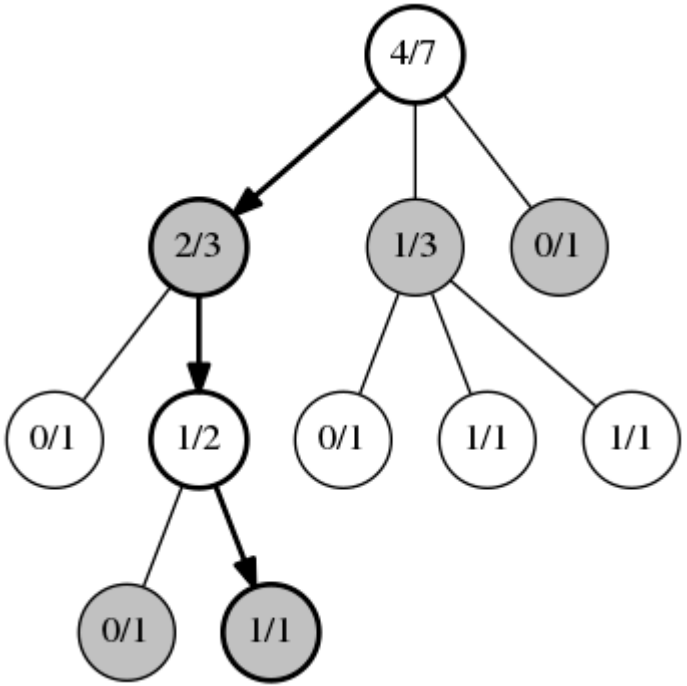
Monte-Carlo Tree Search



Node Representation

- Each node consists of 4 main components
- Visits: How many times has this node been visited in traversals
- Wins: How many times has passing through this node led to a player win
- Children: The children of this node that I have generated so far

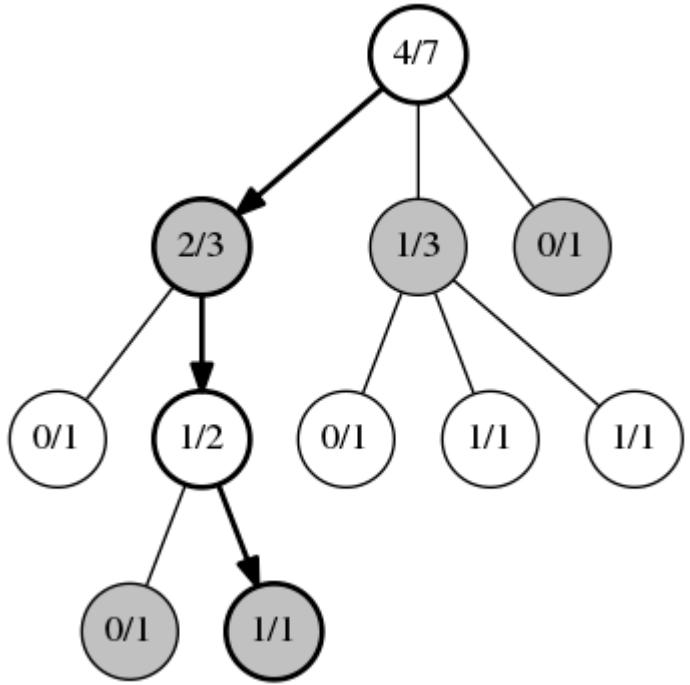
Node Representation



Images: https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/

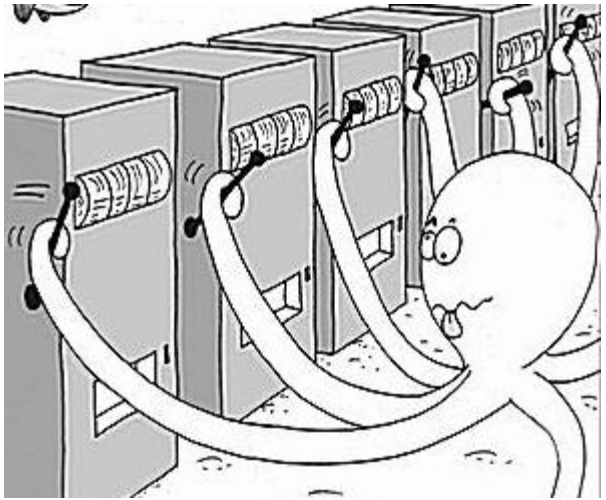
Step 1: Selection

- Multi-armed bandit
- Node selection policy
  - Most wins
  - Most visits
  - Highest win %
  - Formula n = (w, v)
- Follow until a leaf node
- Usually: all children visited once before any twice



N-Armed Bandit Problem

- Repeatedly make a choice among n different actions
- After each action you receive a reward from a stationary probability distribution depending on the action
- Objective is to maximize your expected total reward over a number of action selections

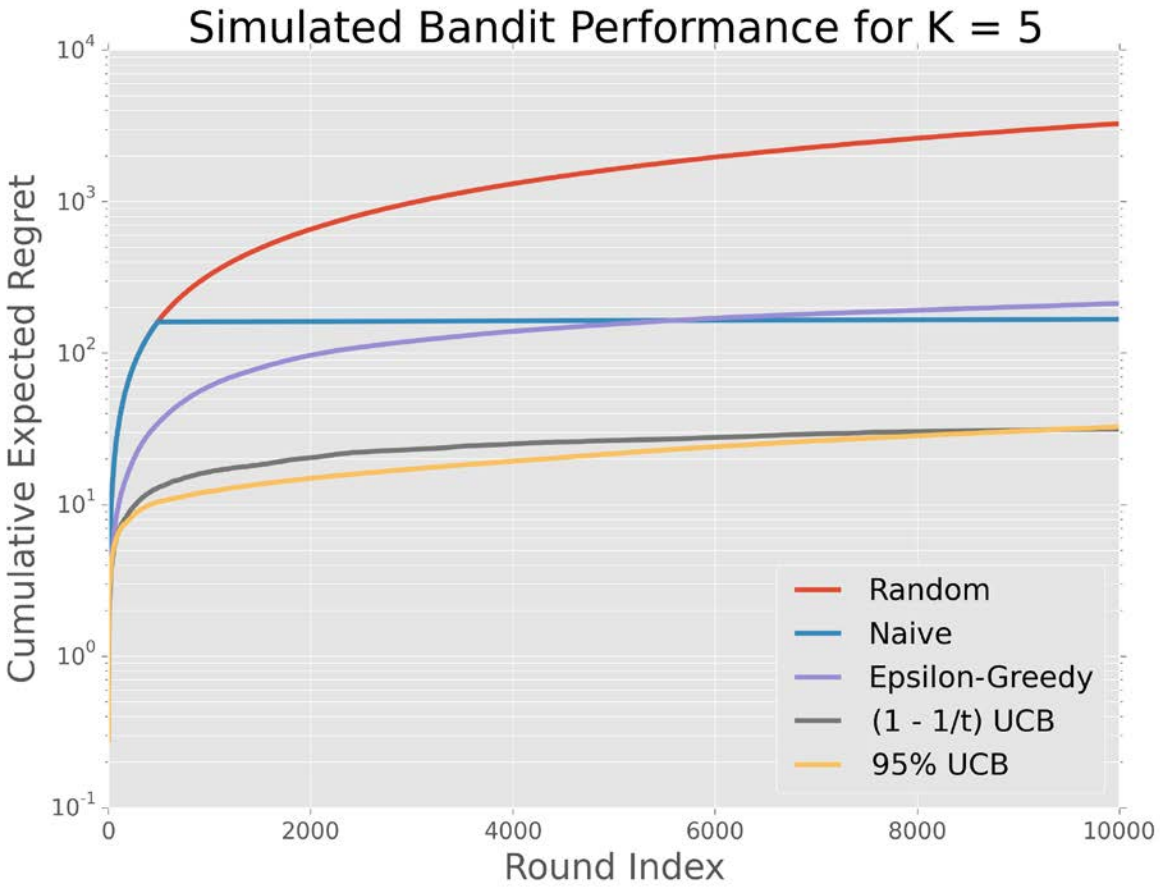


Exploitation vs. Exploration

- If we maintain an estimate of action values, at any time there is one greatest
  - The ‘Greedy Action’
- Exploitation: Choosing the Greedy Action
  - Maximizes single action returns
- Exploration: Choosing a non-greedy action to improve your action estimates
  - Required for future reward maximization
- How to balance exploitation vs. exploration?

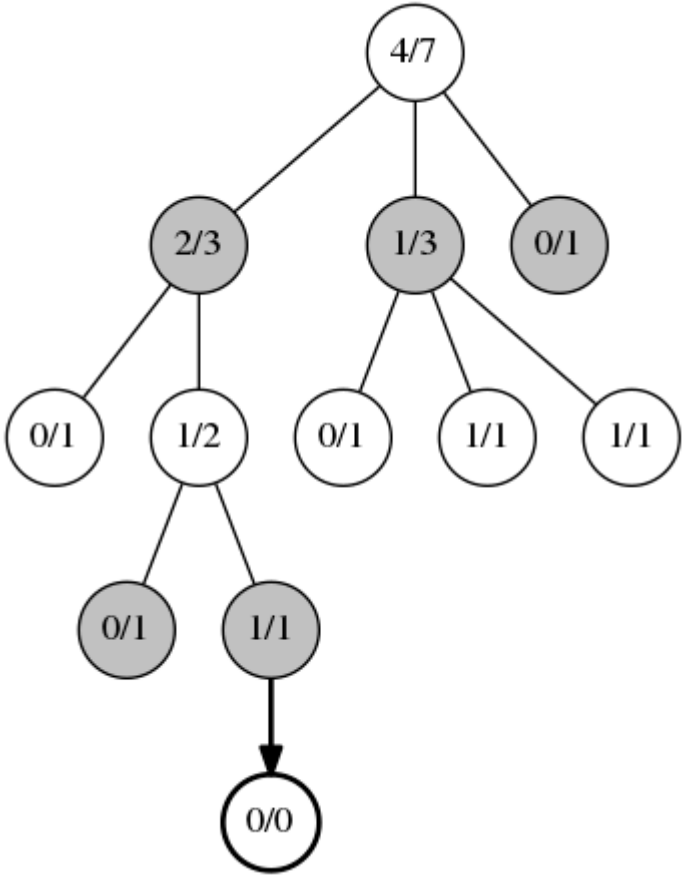
Selection Formulas

- Greedy selection doesn't explore
- Epsilon greedy works fairly well
- Best option: Upper Confidence Bound UCB
- MCTS + UCB = UCT
- UCT Formula: $\frac{S_c}{n_c} + K\sqrt{\frac{2 \ln n_p}{n_c}}$


Simulated Bandit Performance for K = 5
Cumulative Expected Regret
10^4
10^3
10^2
10^1
10^0
10^{-1}
0
2000
4000
6000
8000
10000
Round Index
Random
Naive
Epsilon-Greedy
(1 - 1/t) UCB
95% UCB

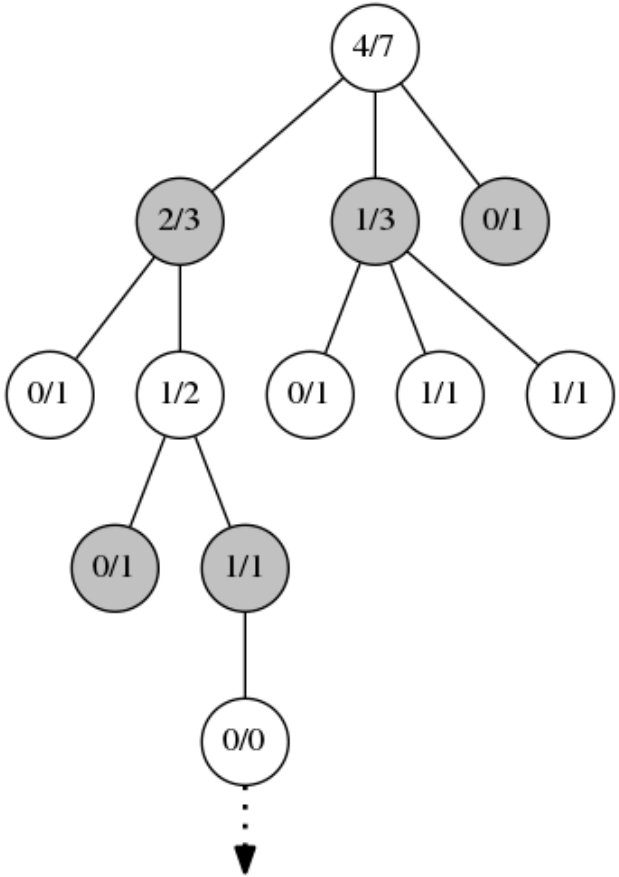
Step 2: Expansion

- We have reached a leaf node in the tree
- Can no longer select a node
- Generate a child of the leaf node to be evaluated
- This evaluation will be done via simulation



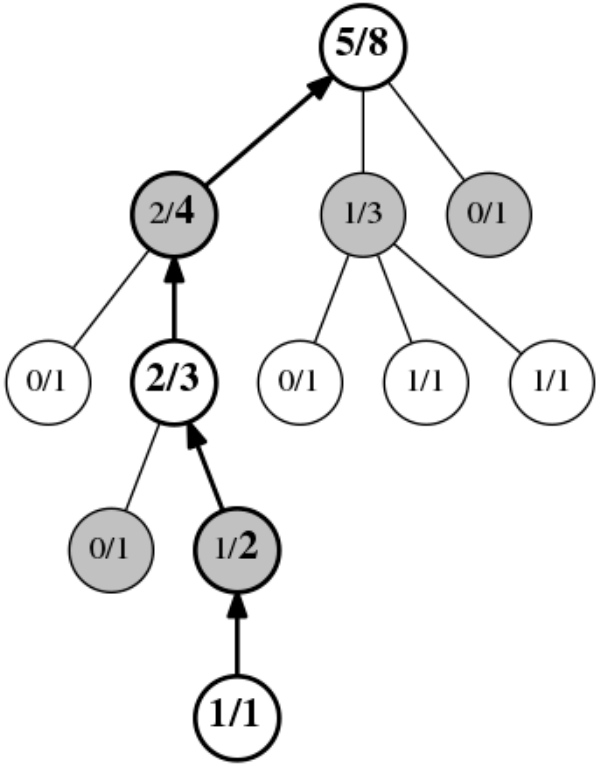
Step 3: Simulation

- Simulation occurs by playing the game from a given state to the end
- Some policy is needed to guide play, usually random
- The winner of the game using this policy is recorded
- We are now ready to update the parent node values



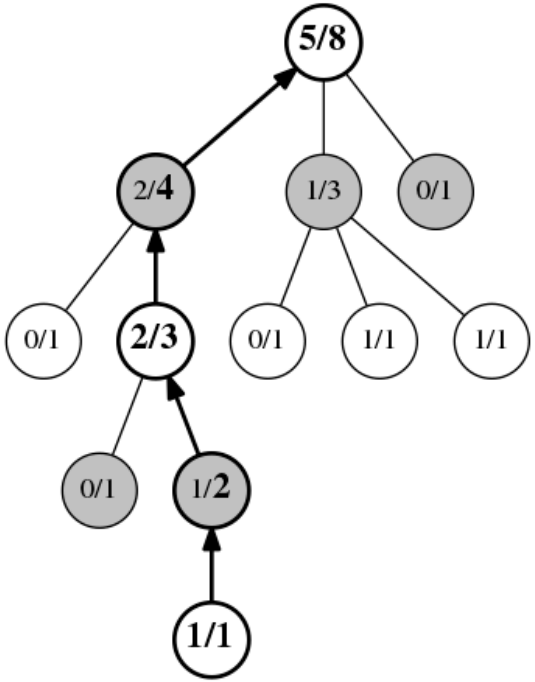
Step 4: Back Propagation

- Follow parent nodes back to the root node
- Add one visit to each node
- If the simulation won, add a win to each node, if not then don’t add anything
- Update is complete!

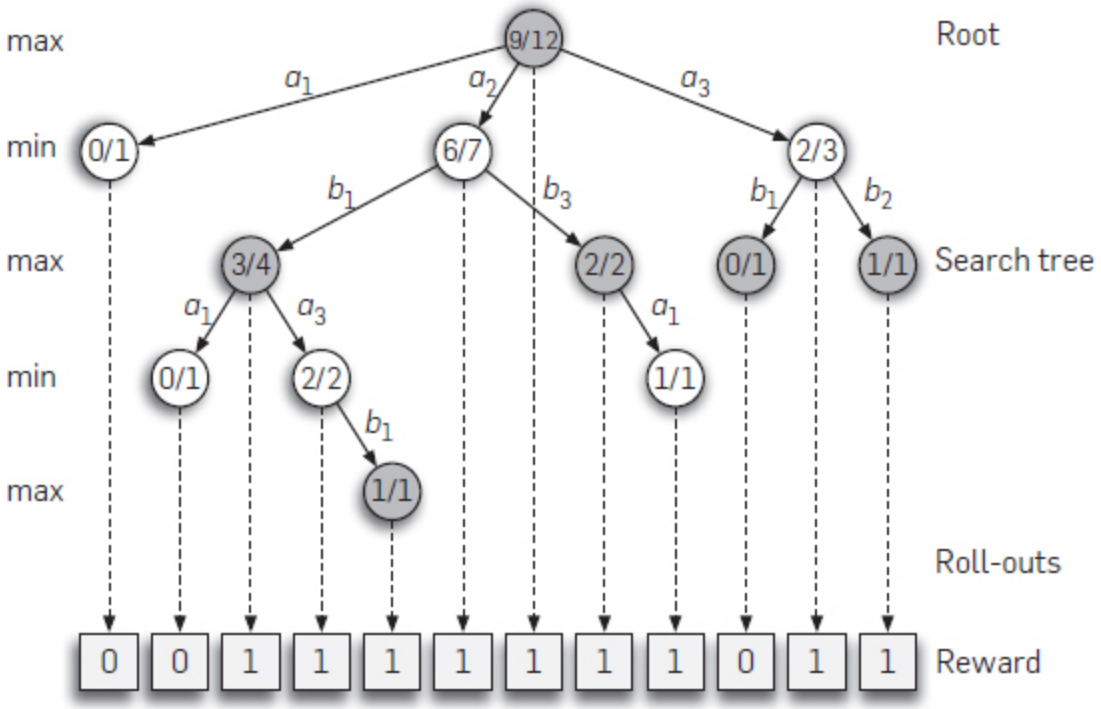


Final Action Selection

- After some traversals, we now select the root action
- No longer want to do any exploration
- Selection action with:
  - Highest win %
  - Most visits



Example Tree



MCTS Advantages

- Copes with large branching factors
- Any-time algorithm, stop at # traversals
- Doesn’t rely on a heuristic function, simulation determines state value
- Fairly simple implementation
- Can also use heuristics if available

MCTS Disadvantages

- Tree grows large quickly, memory usage is very intense, unlike DFS Alpha Beta
- Doesn’t find ‘narrow path to victory’ as well as algorithms like minimax
- Harder to make use of heuristics
- Doesn’t work if random playouts are bad in a given domain (ex: RTS games)