CS5491: Artificial Intelligence

Adversarial Search

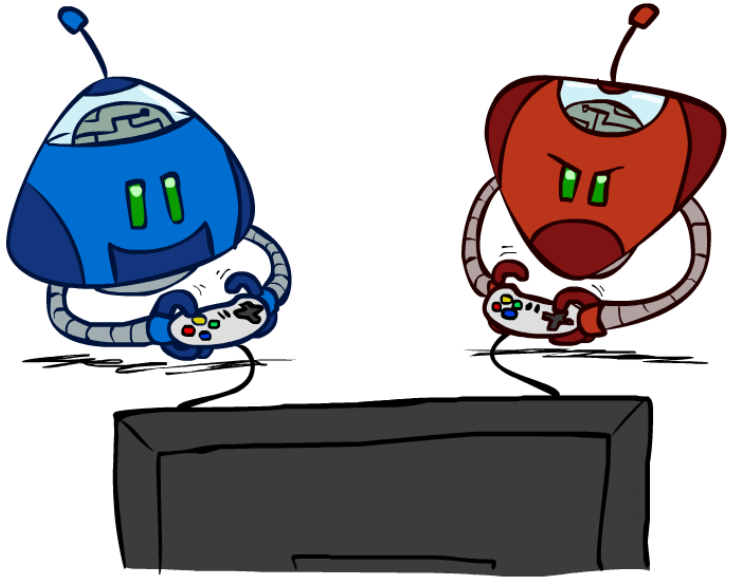
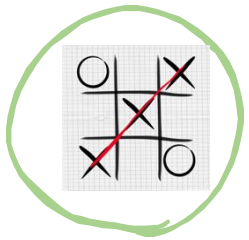


Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

Instructor: Kai Wang

Today



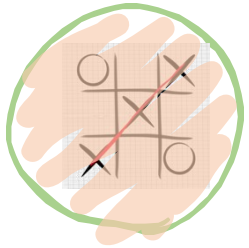
Games and
Game Trees

Minimax
Algorithm

Tree
Pruning

Monte Carlo
Tree Search

Today



Games and
Game Trees

Minimax
Algorithm

Tree
Pruning

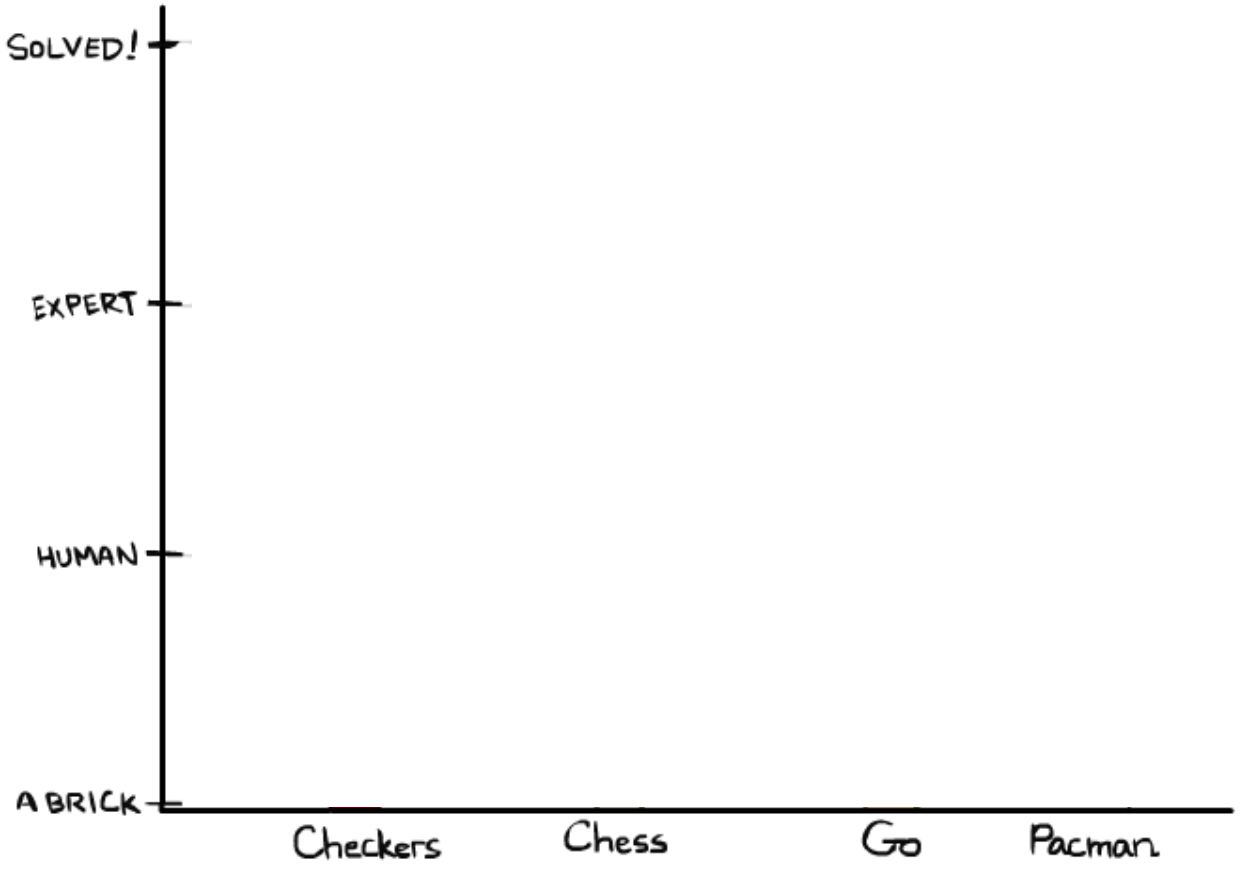
Monte Carlo
Tree Search

Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go:** Human champions are now starting to be challenged by machines. In go, b > 300! Classic programs use pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods.



Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: Firs computer champion: Chinook ended 40-year-reig of human champion Marion Tinsley using comple 8-piece endgame. 2007: Checkers solved!

- **Chess:** 1997: Deep Blue defeats human champior Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methoc for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go: 2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search, learned evaluatic function.**

- **Pacman**

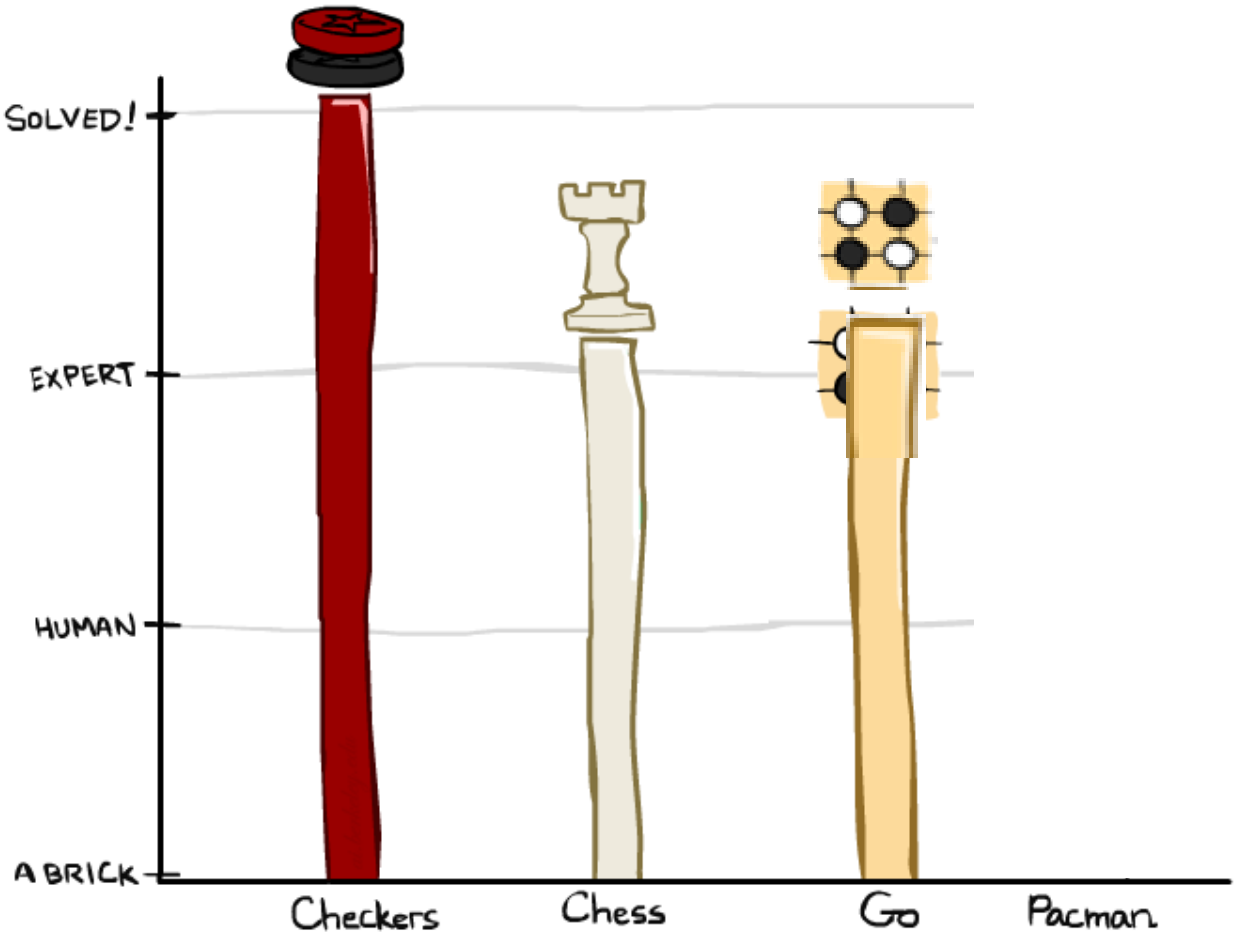
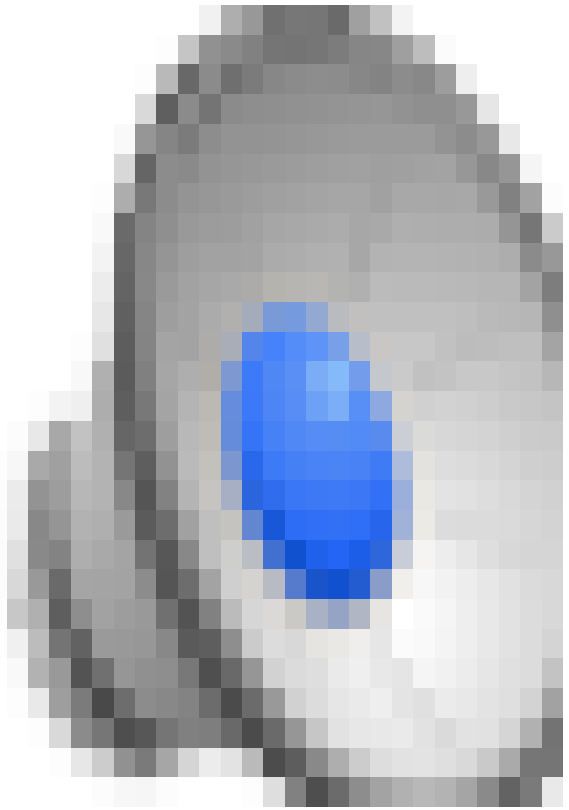


Image credit: Dan Klein and Pieter Abbeel http://ai.berkeley.edu

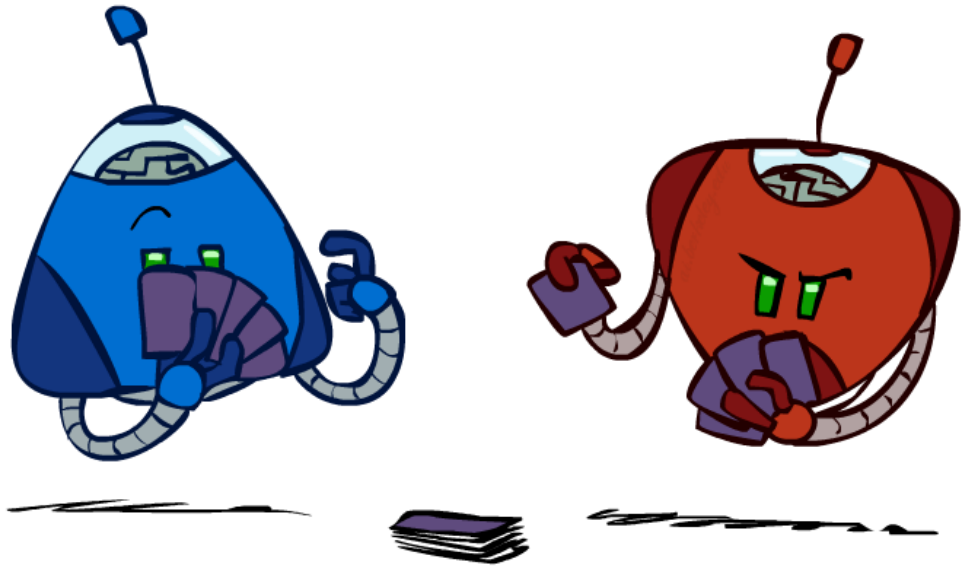
Demo Mystery Pacman



Demo: AlphaStar by DeepMind

Games

- There are many different kinds of games



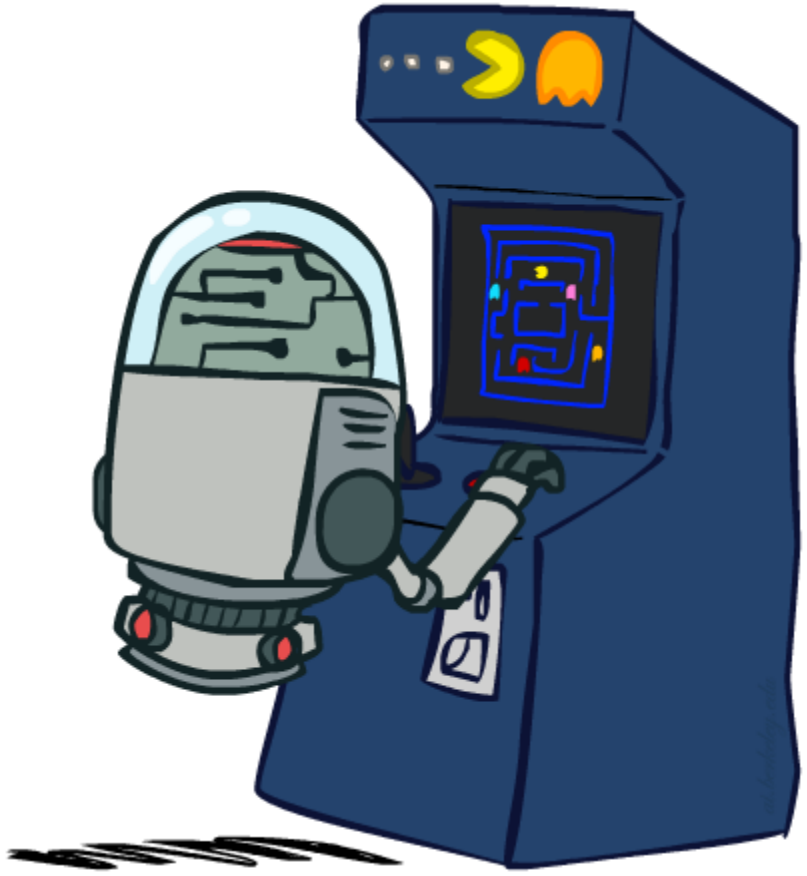
- Axes (dimensions):
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?

- Goal: get algorithms for calculating a strategy (policy) which recommends a move from each state (to win the game, if possible)

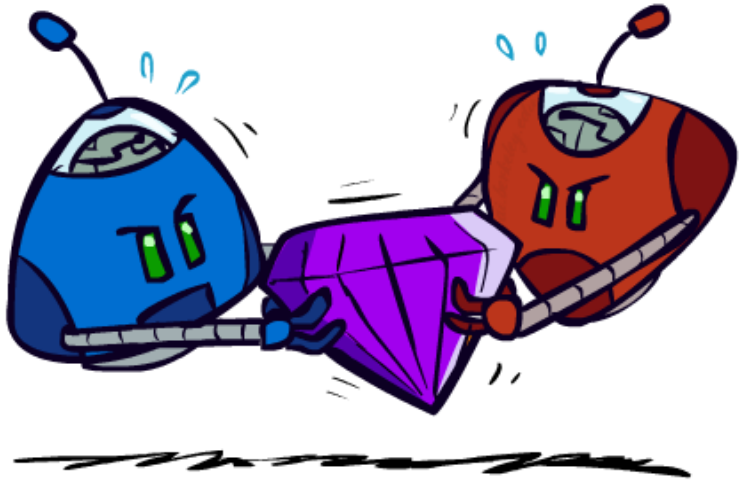
Deterministic Games

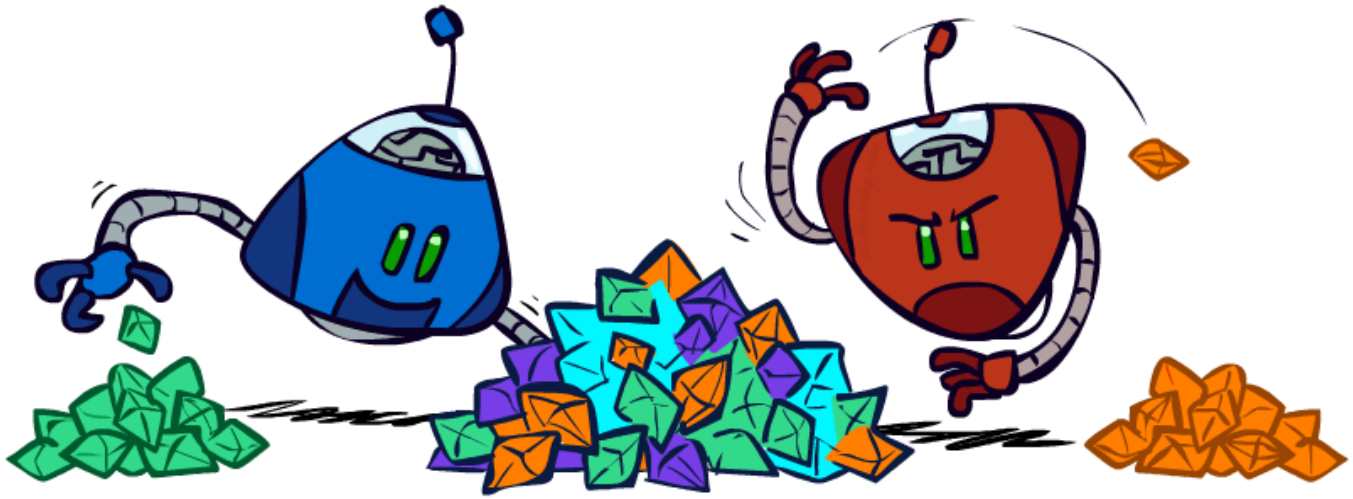
- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: SxA $\rightarrow$ S
  - Terminal Test: S $\rightarrow$ {t,f}
  - Terminal Utilities: SxP $\rightarrow$ R

- Solution for a player is a policy: S $\rightarrow$ A



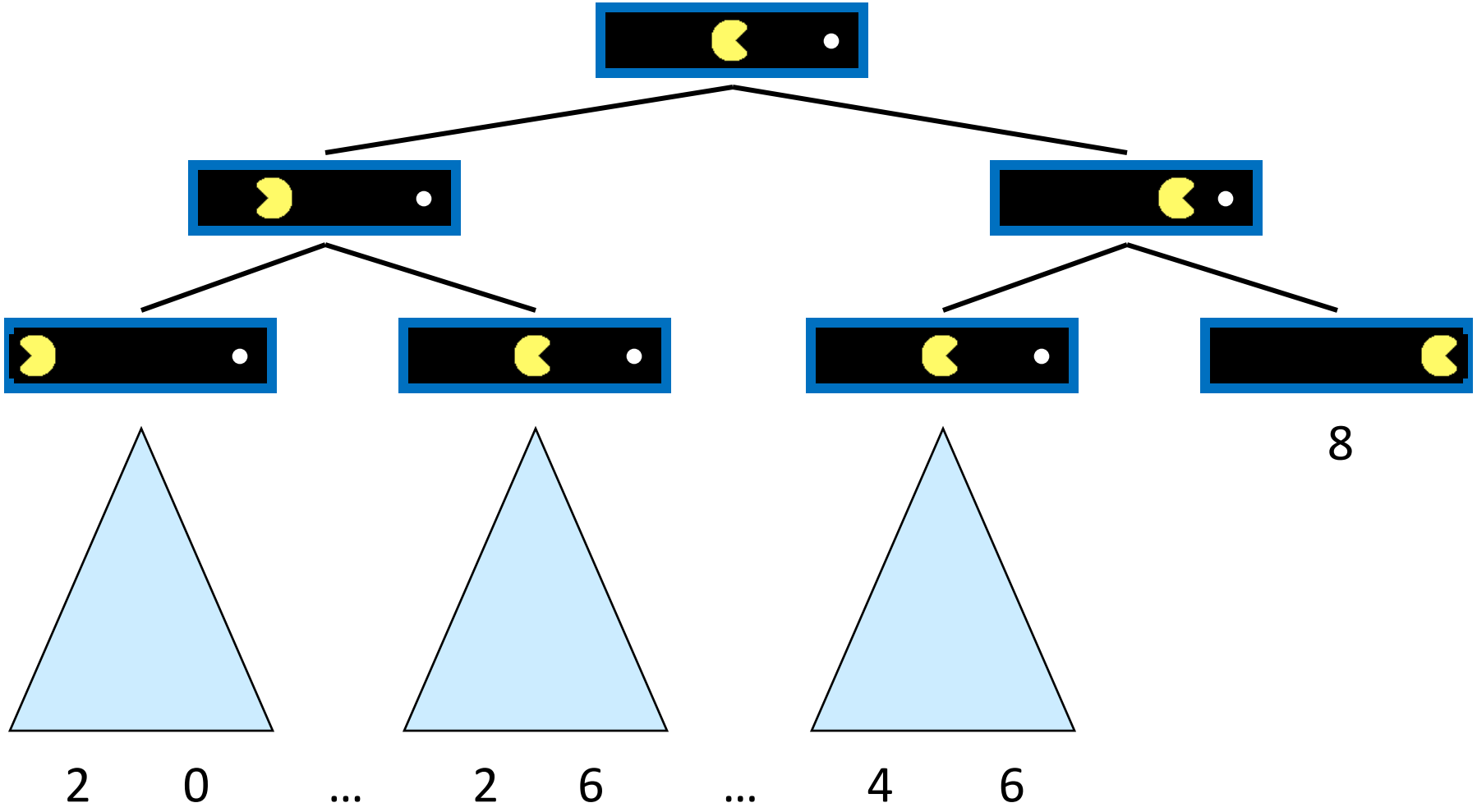
Zero-Sum Games





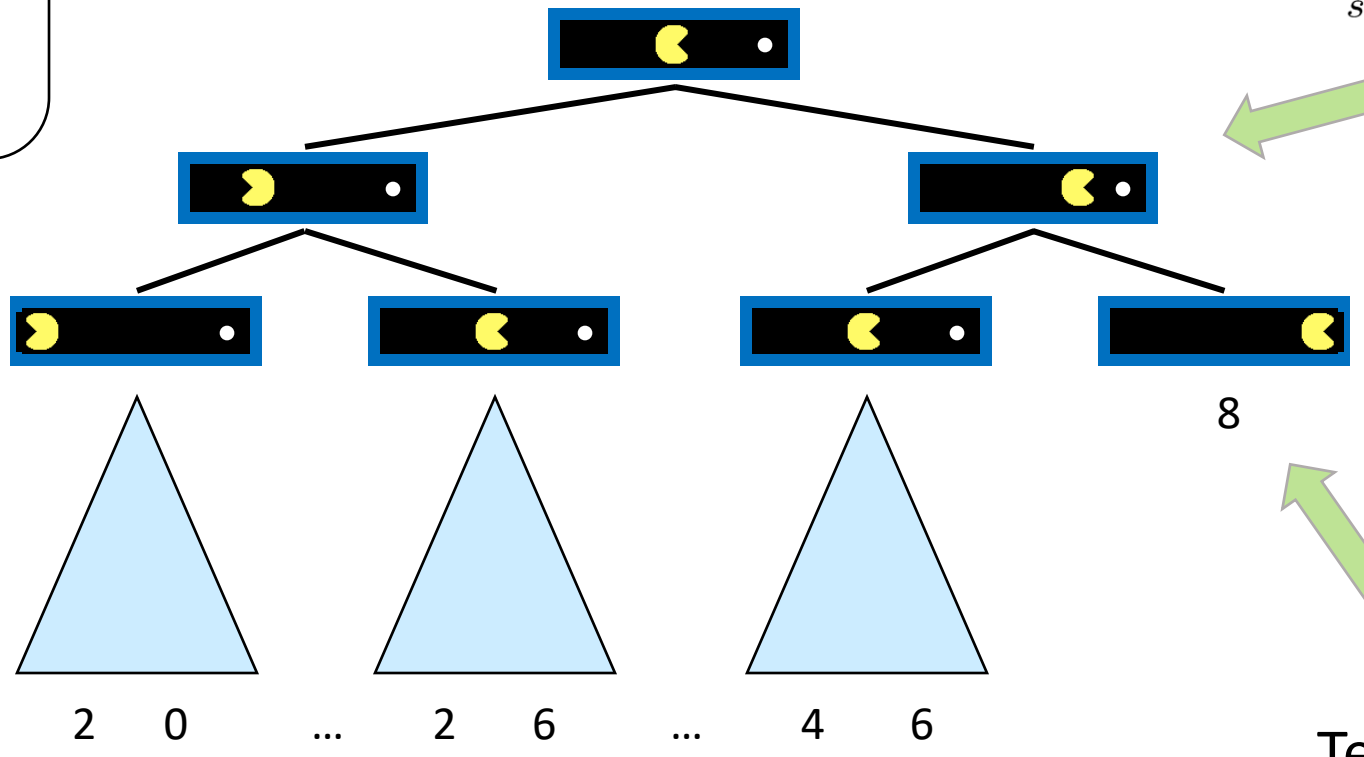
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

Single-Agent Trees



Value of a State

Value of a state:
The best achievable outcome (utility) from that state



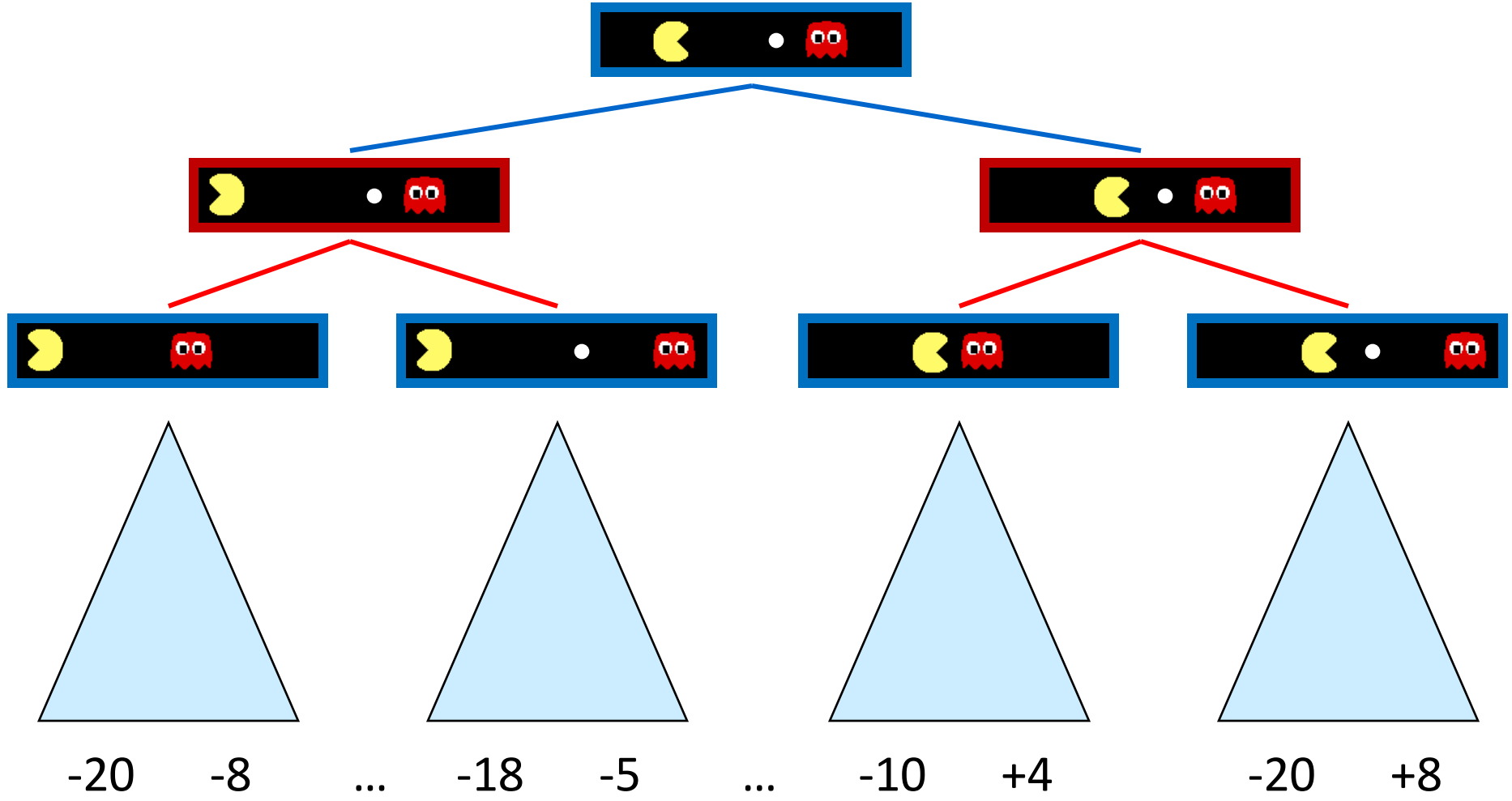
Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

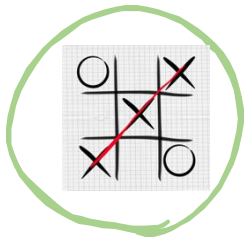
Terminal States:

$$V(s) = \text{known}$$

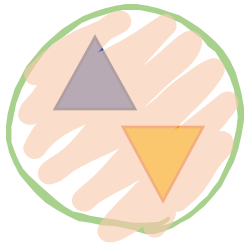
Adversarial Game Trees



Today



Games and Game Trees



Minimax Algorithm

Tree Pruning

Monte Carlo Tree Search

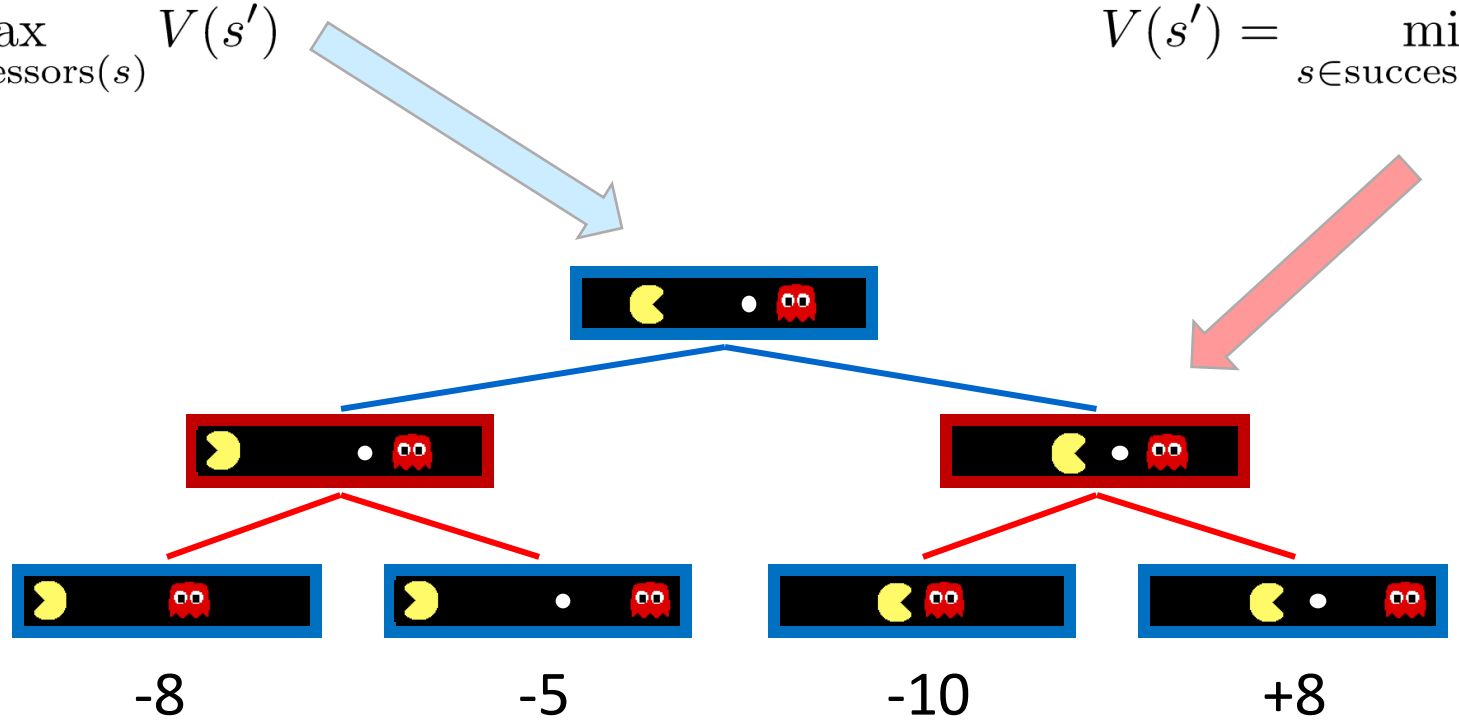
Minimax Values

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

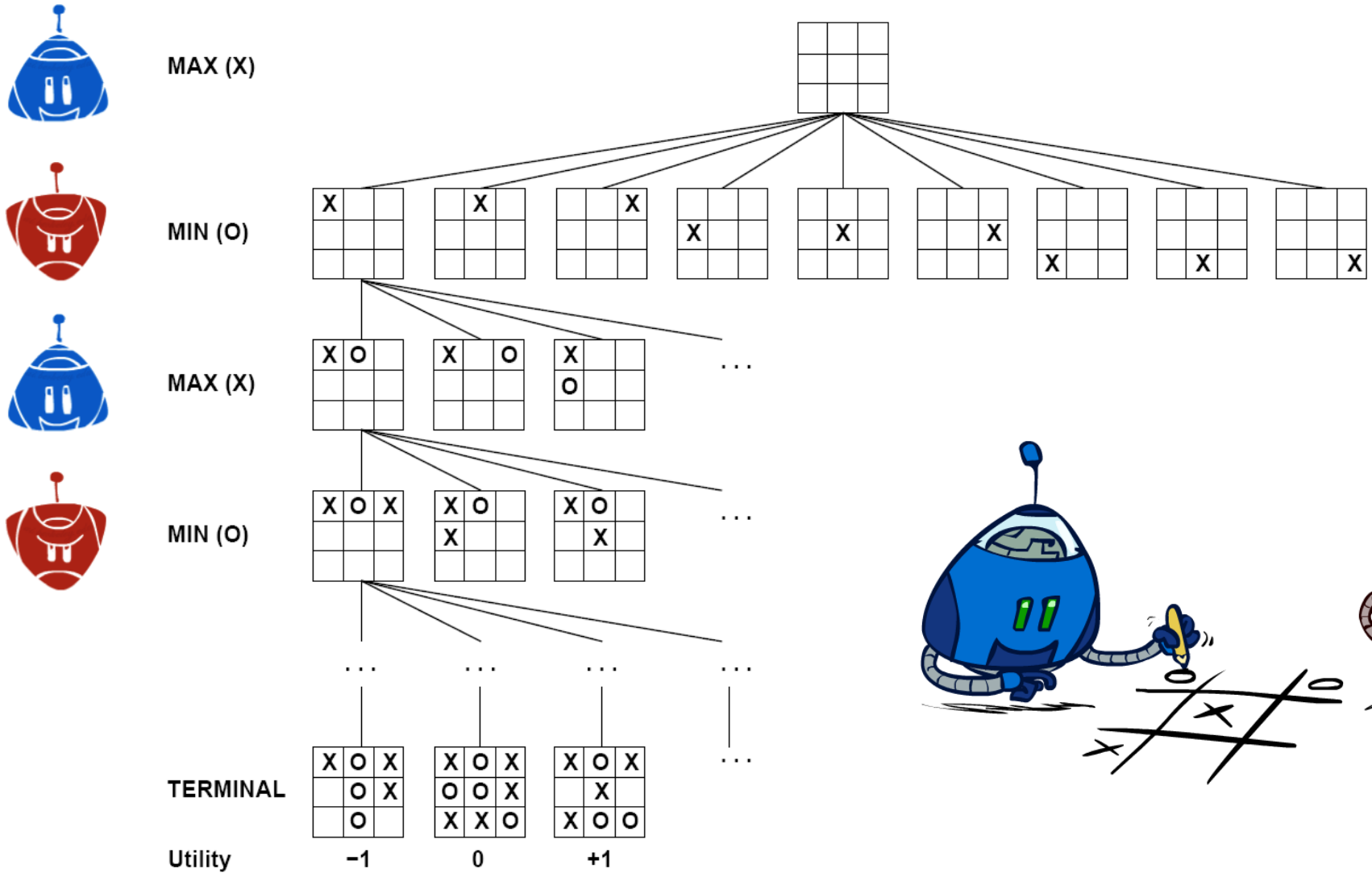
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

$$V(s) = \text{known}$$

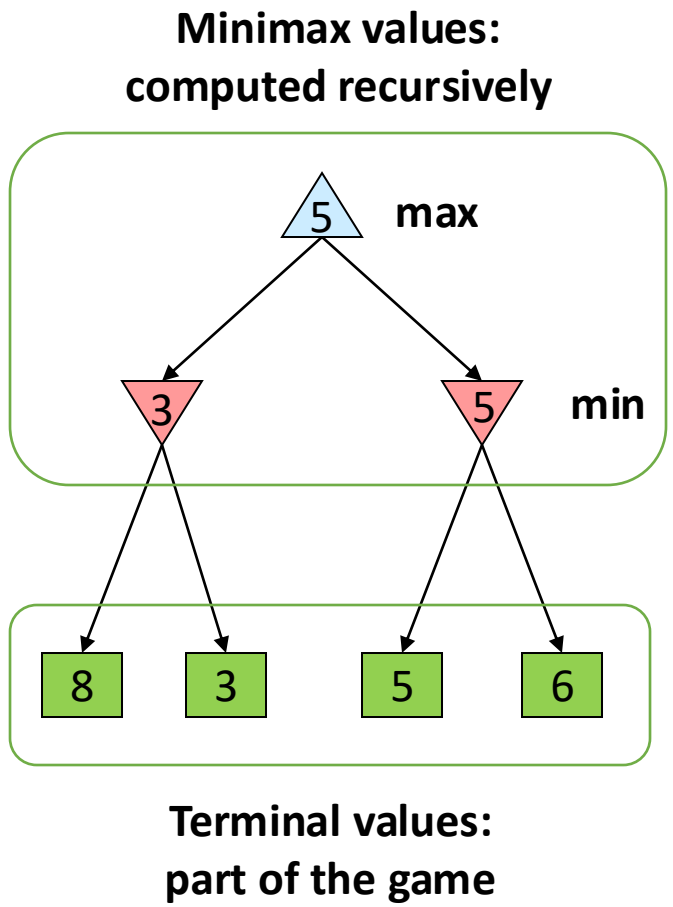
Tic-Tac-Toe Game Tree



Adversarial Search (Minimax)

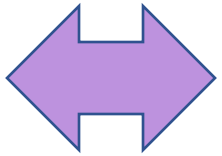
- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result

- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary



Minimax Implementation

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```



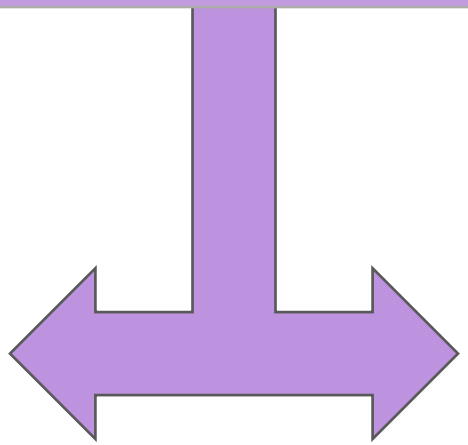
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Minimax Implementation (Dispatch)

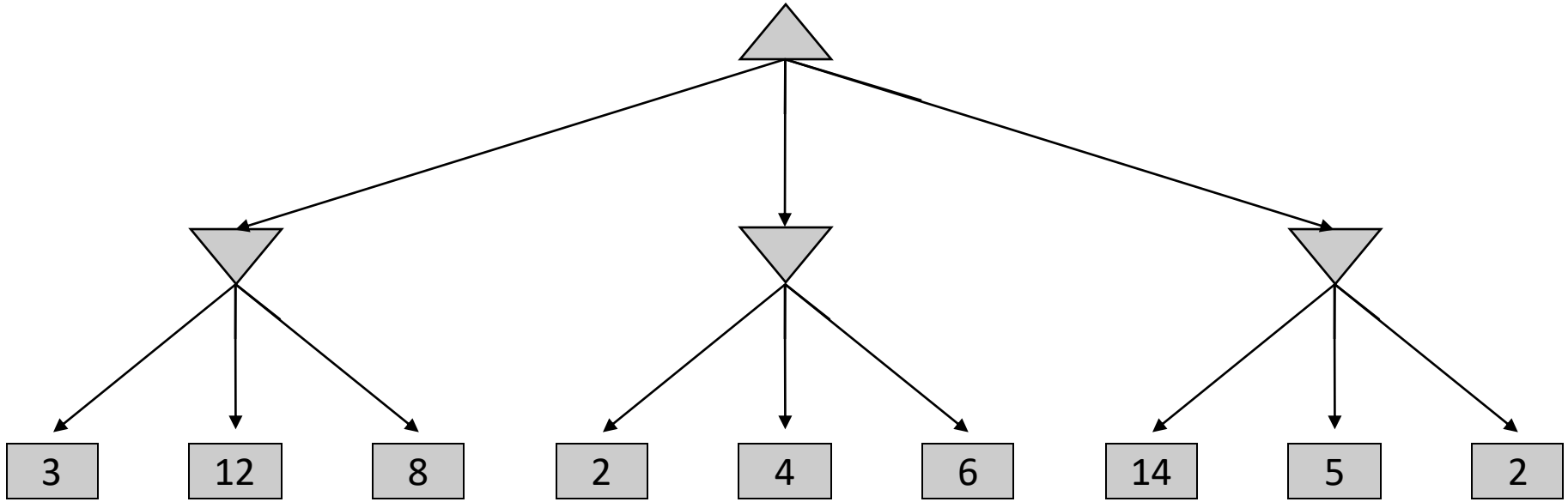
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```



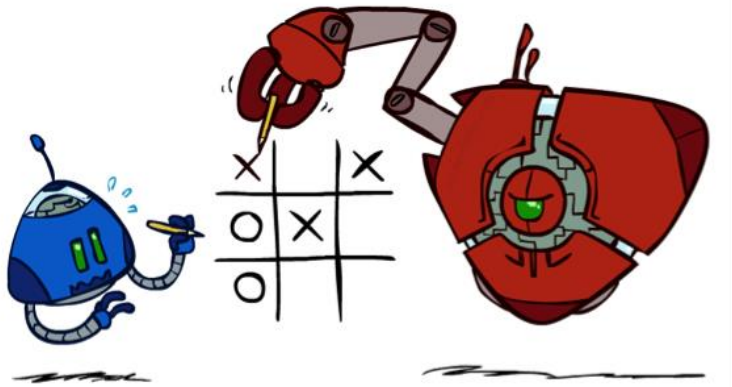
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

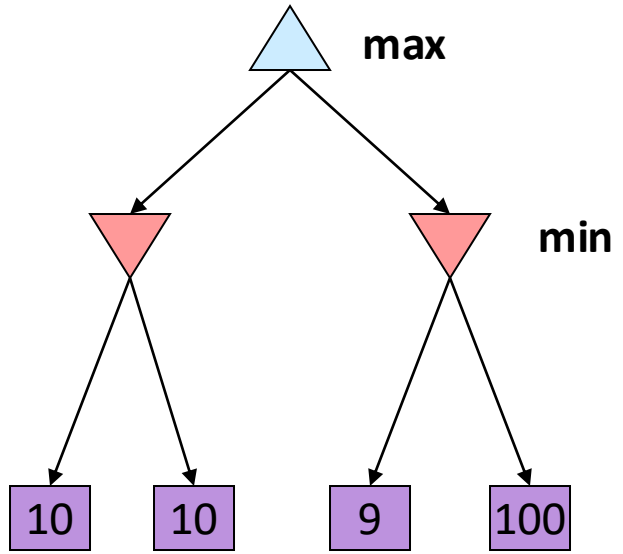
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

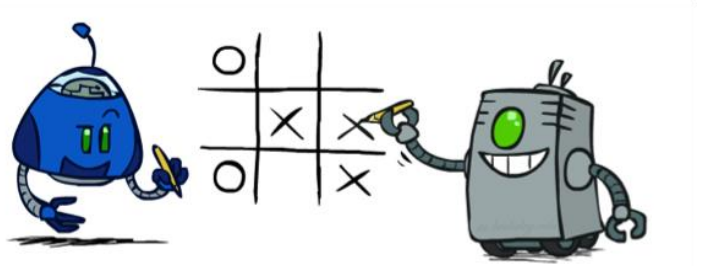
Minimax Example



Minimax Properties

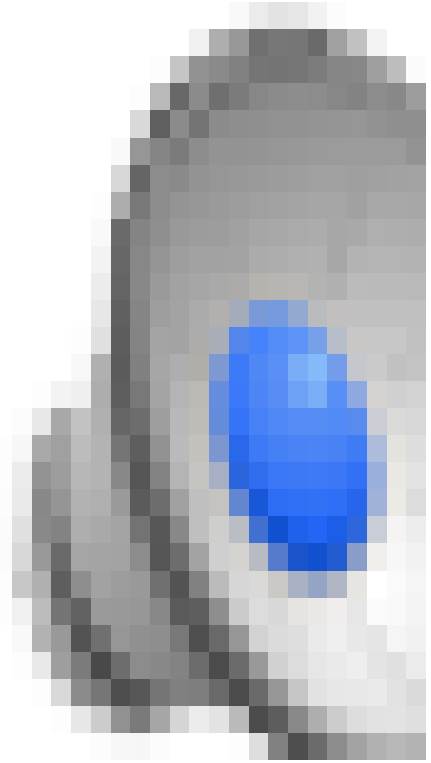




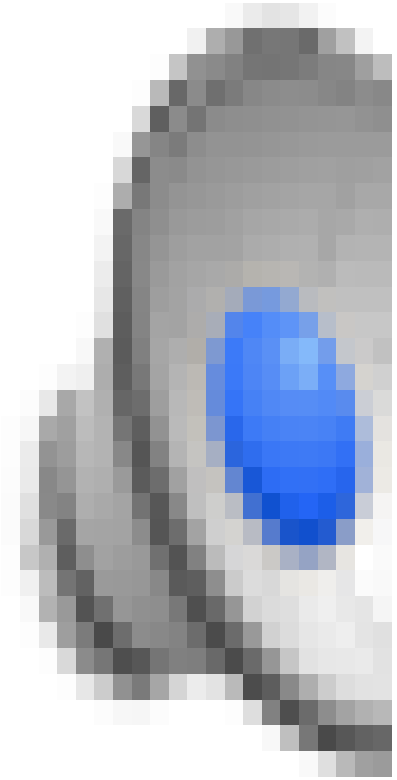


Optimal against a perfect player. Otherwise?

Minimax Properties



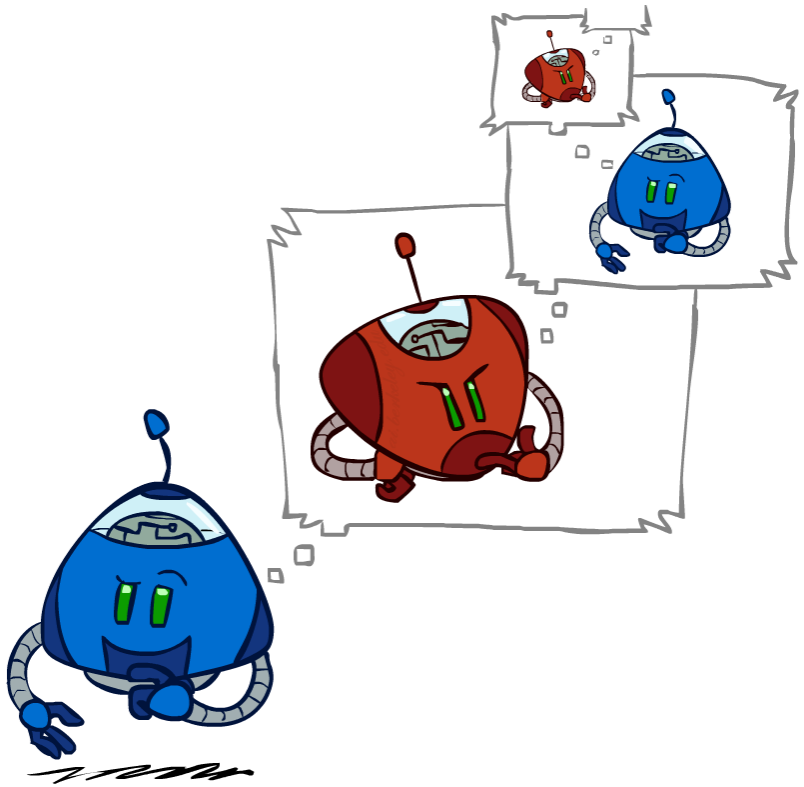
Minimax Properties



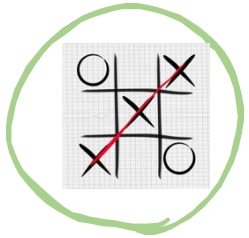
Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: O(bm)

- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Today



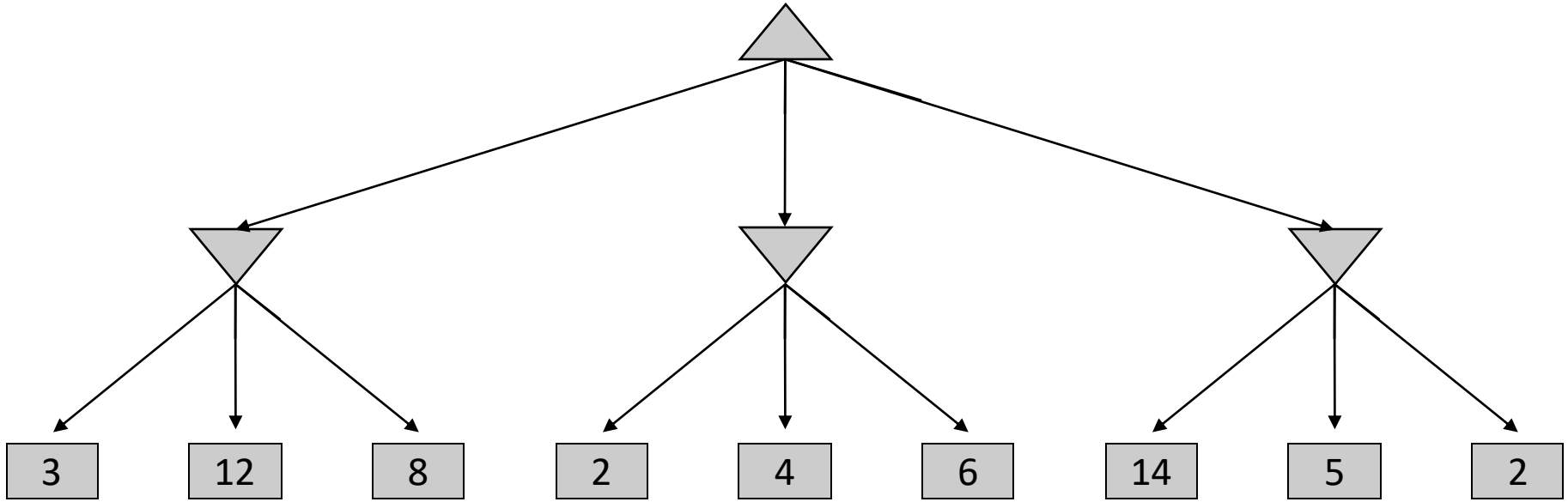
Games and
Game Trees

Minimax
Algorithm

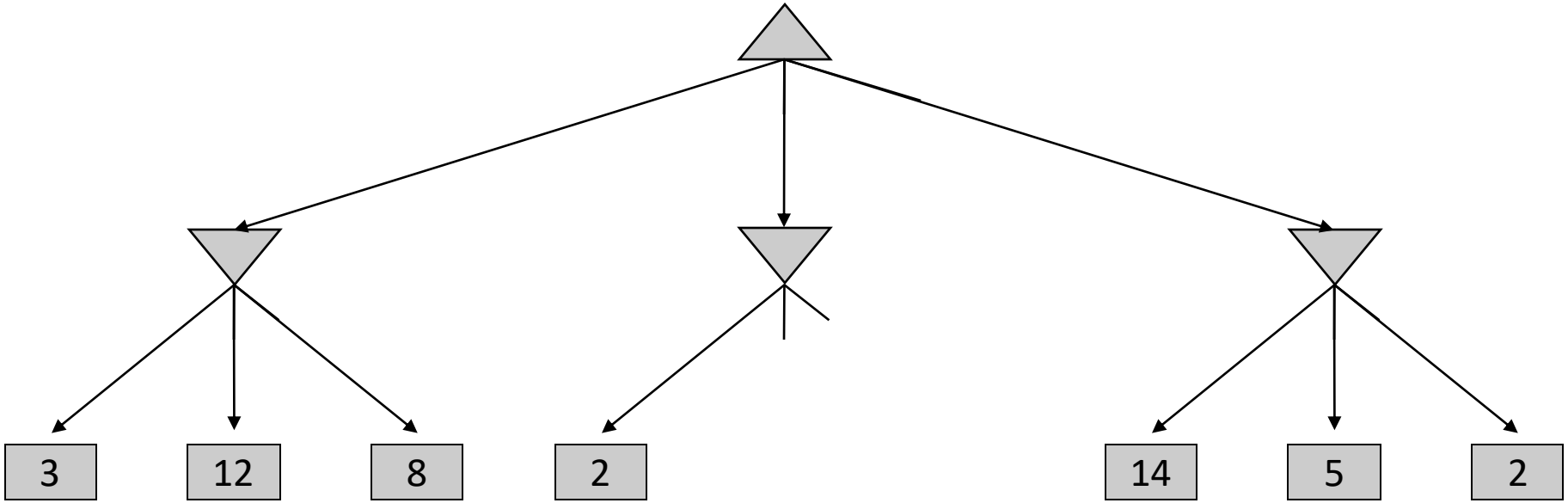
Tree
Pruning

Monte Carlo
Tree Search

Minimax Example

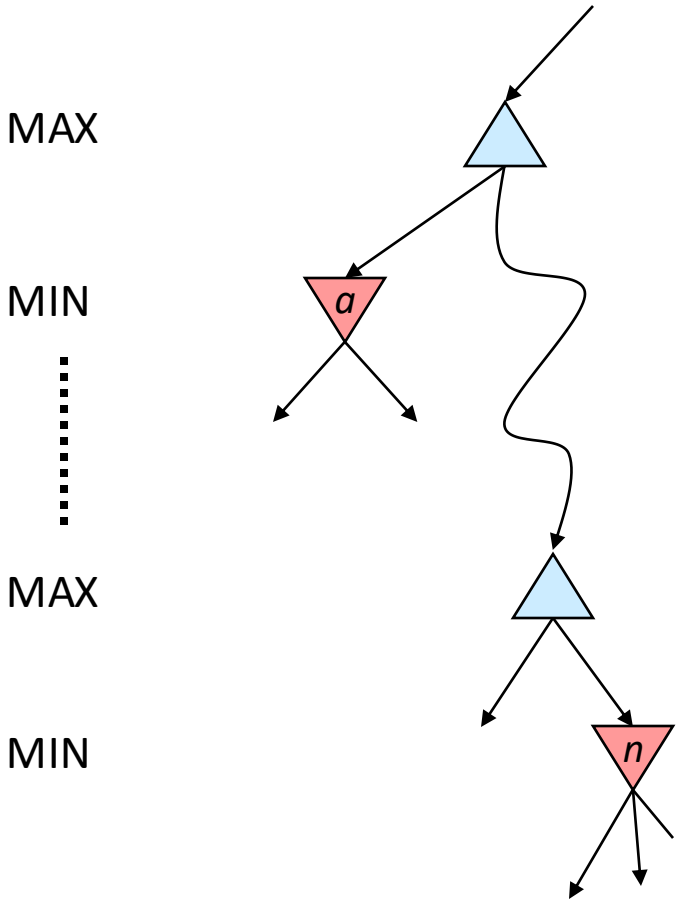


Minimax Pruning

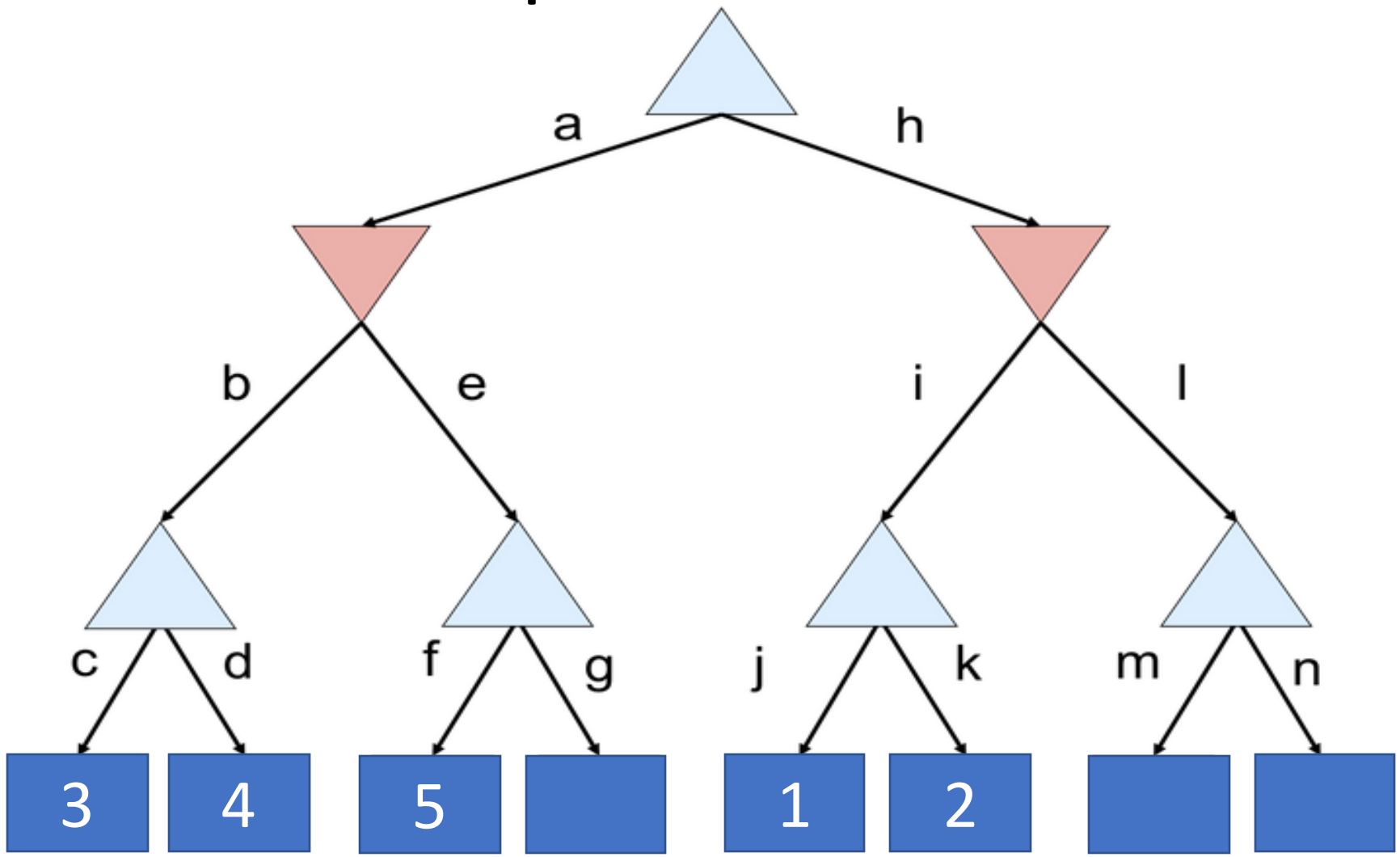


Alpha-Beta Pruning

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the childrens' min is dropping
  - Who cares about *n*'s value? MAX
  - Let *a* be the best value that MAX can get at any choice point along the current path from the root
  - If *n* becomes worse than *a*, MAX will avoid it, so we can stop considering *n*'s other children (it's already bad enough that it won't be played)
- MAX version is symmetric



Alpha-Beta Example



Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state, α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

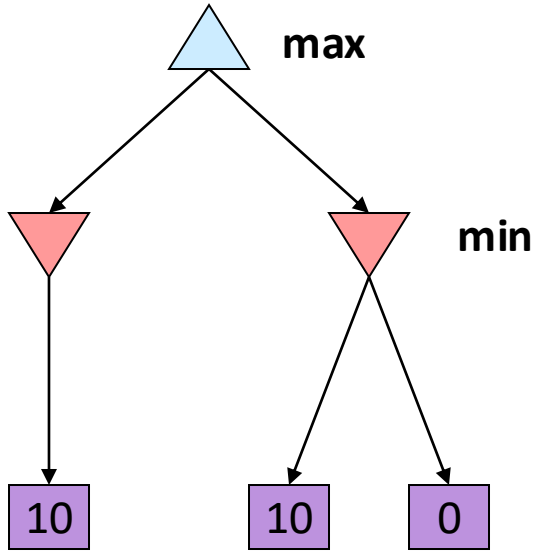
Alpha-Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!

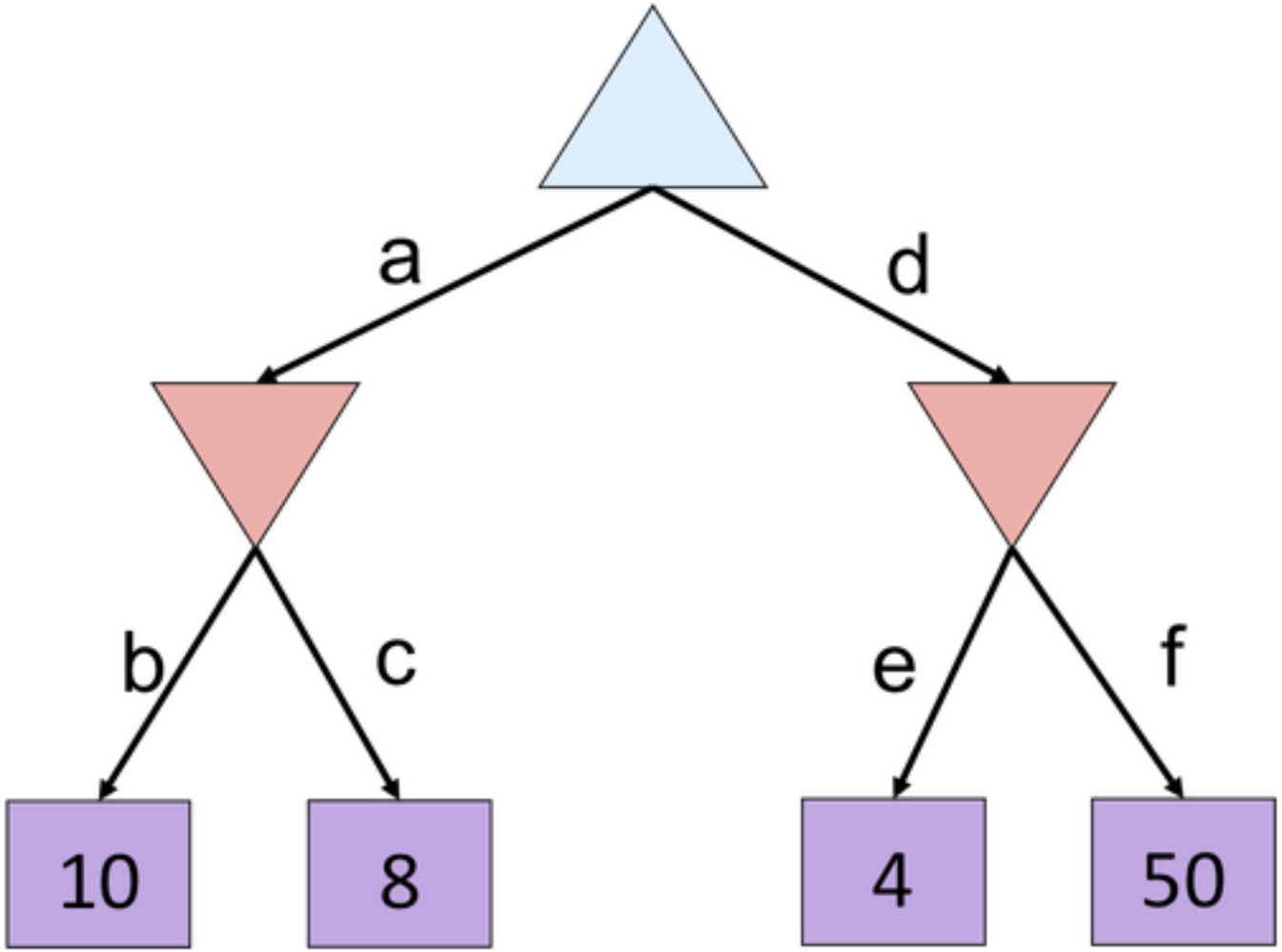
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection

- Good child ordering improves effectiveness of pruning

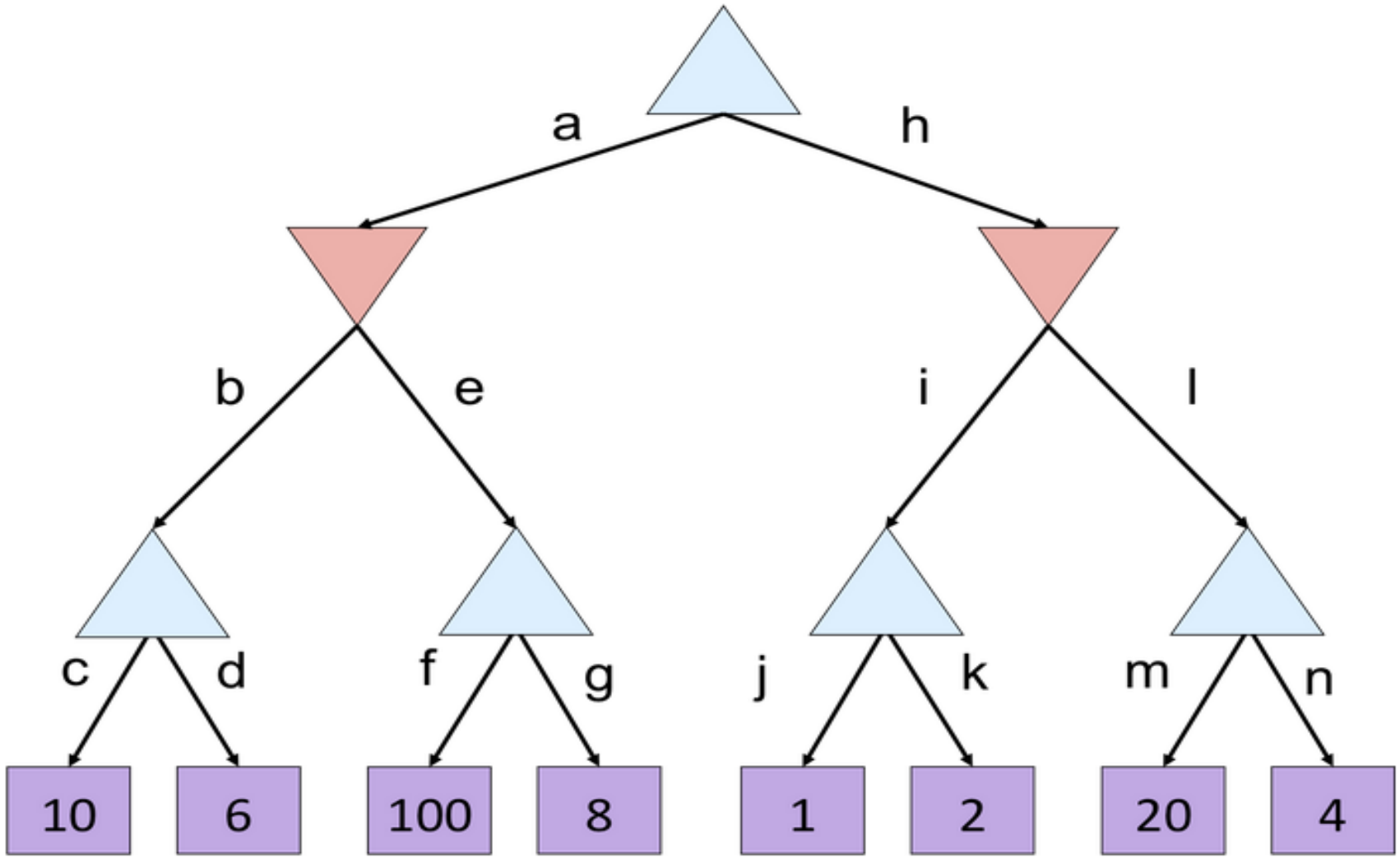
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…



Alpha-Beta Quiz

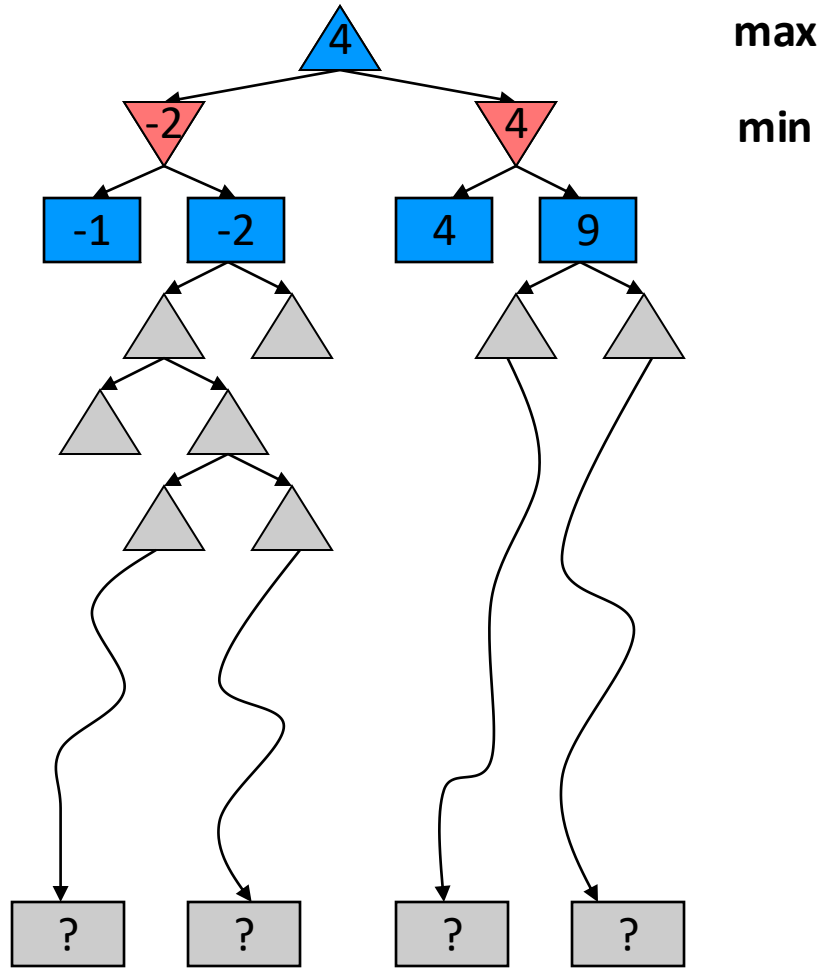


Alpha-Beta Quiz 2



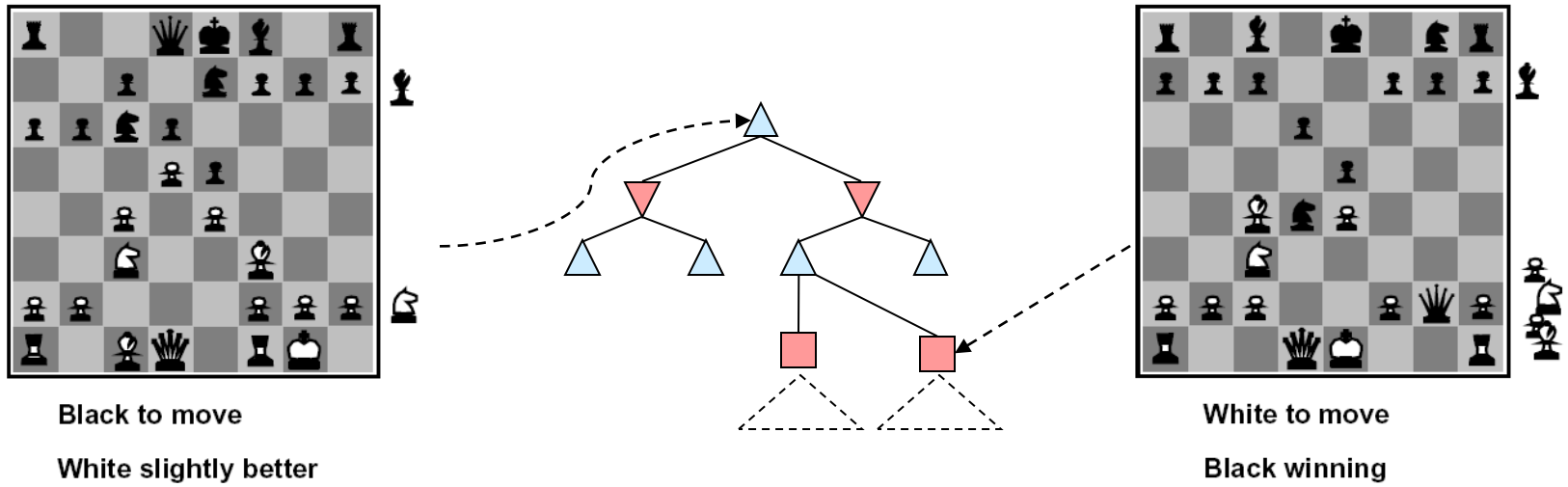
Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm



Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search

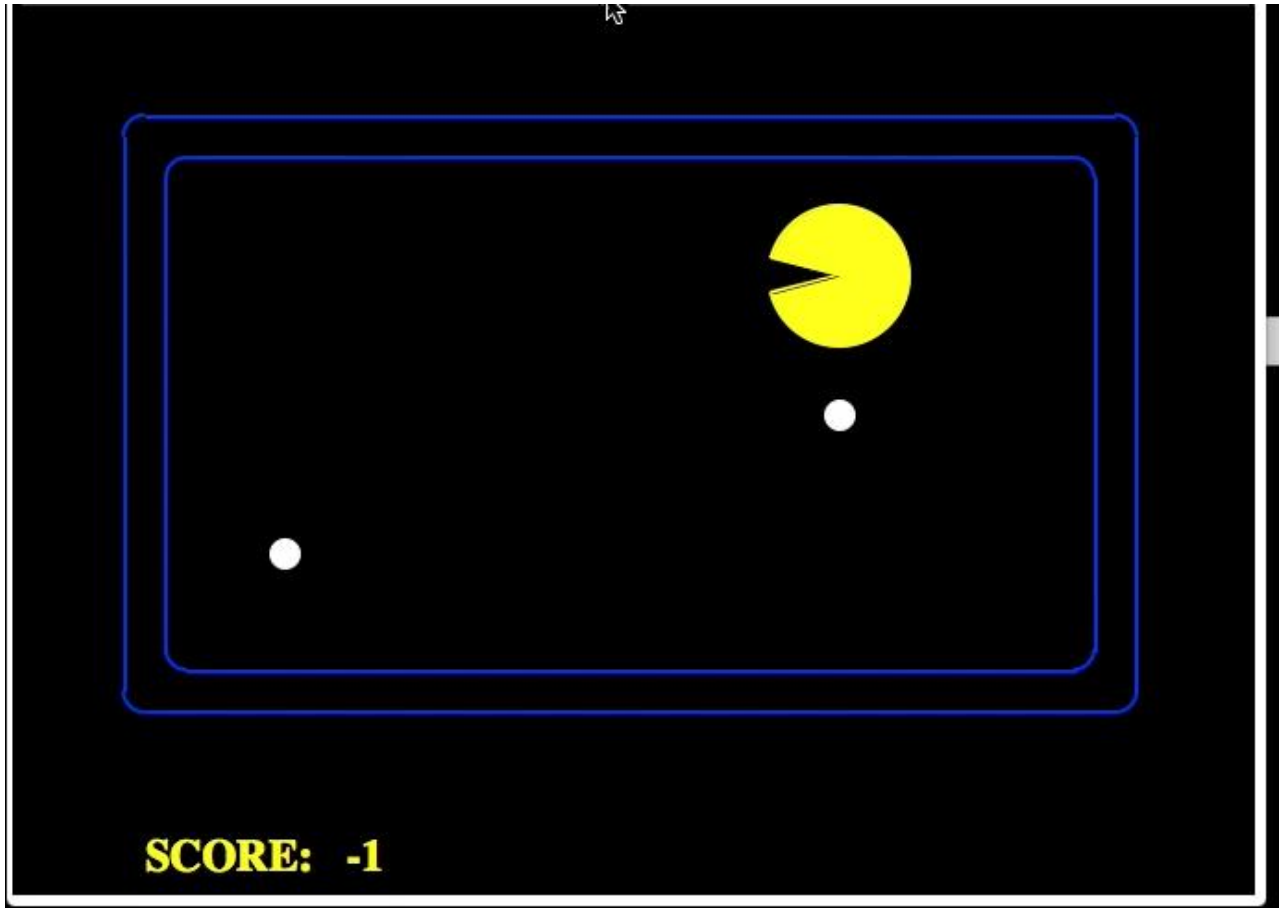


- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

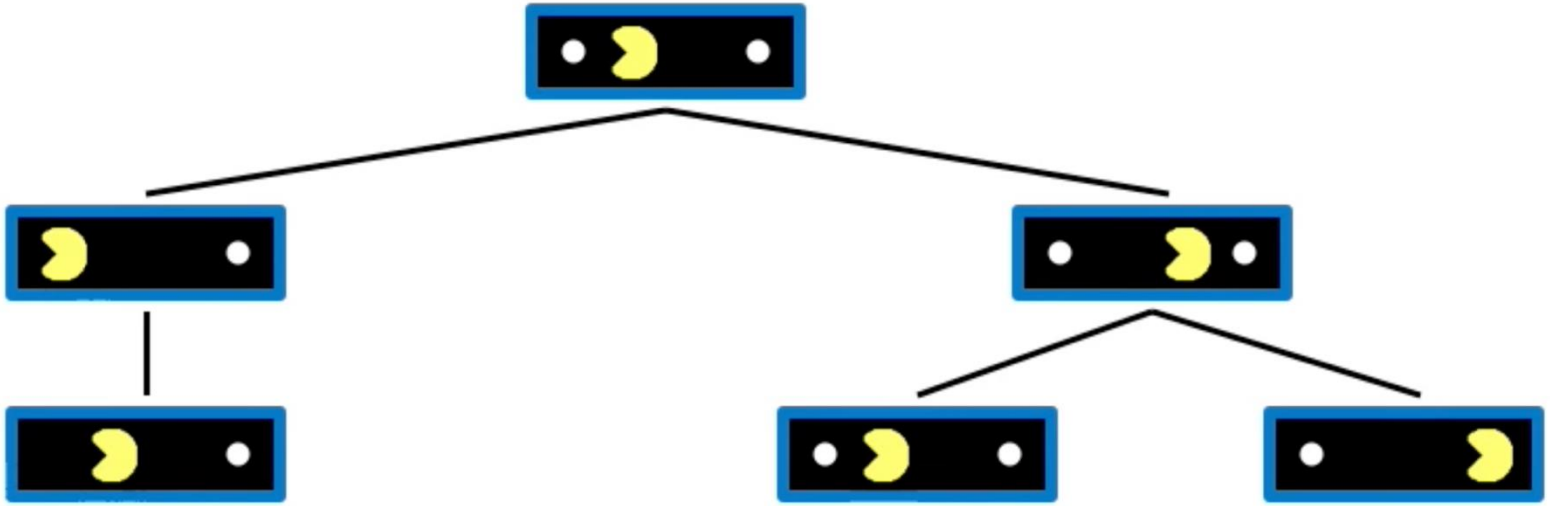
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Evaluation for Pacman

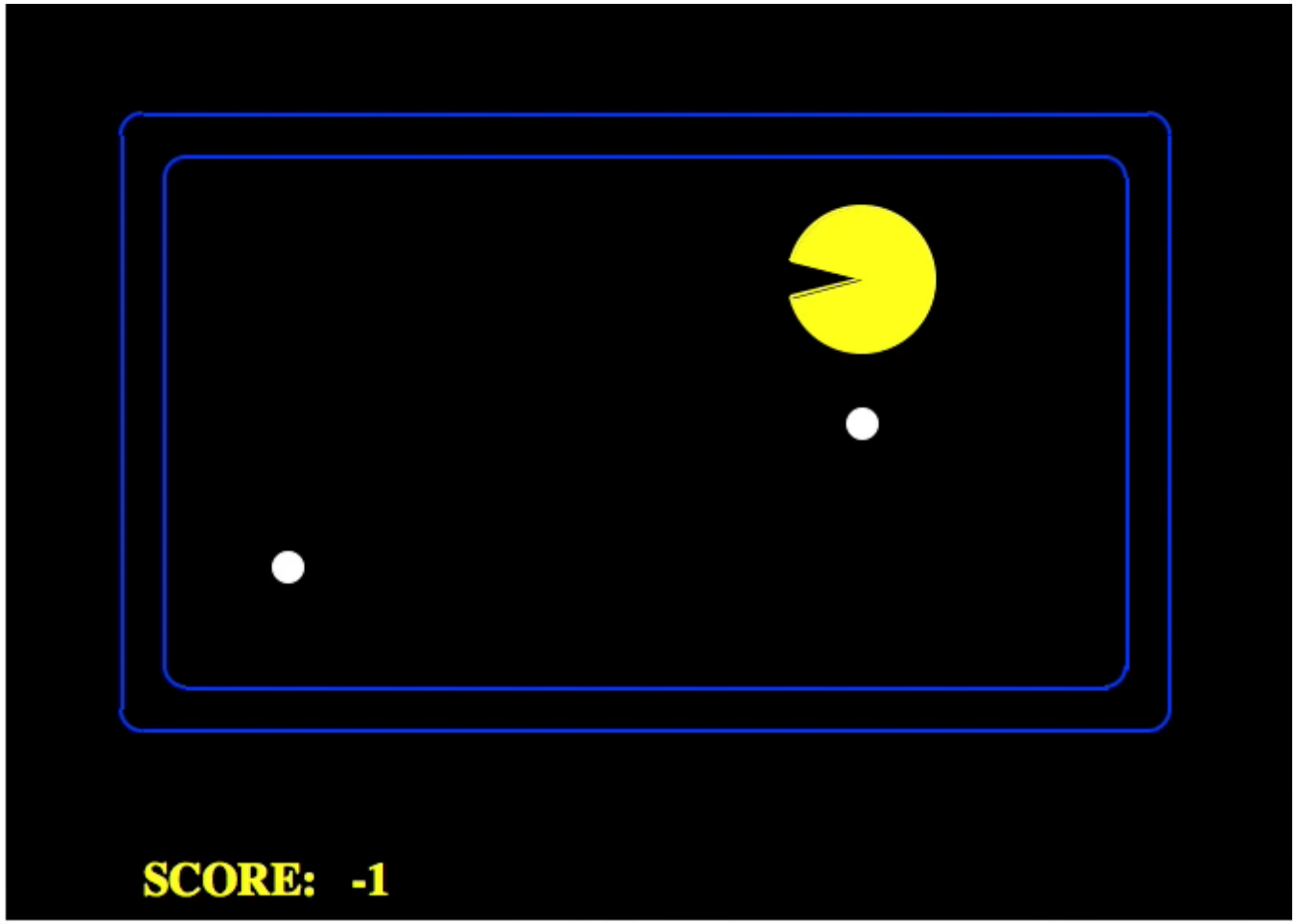


Why Pacman Starves



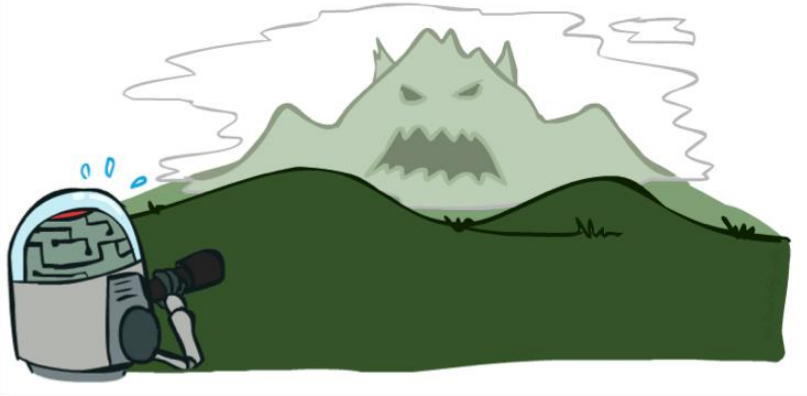
- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then go back west in the next round of replanning!

Evaluation for Pacman

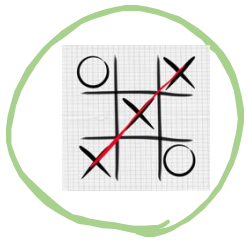


Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation



Today



Games and
Game Trees

Minimax
Algorithm

Tree
Pruning

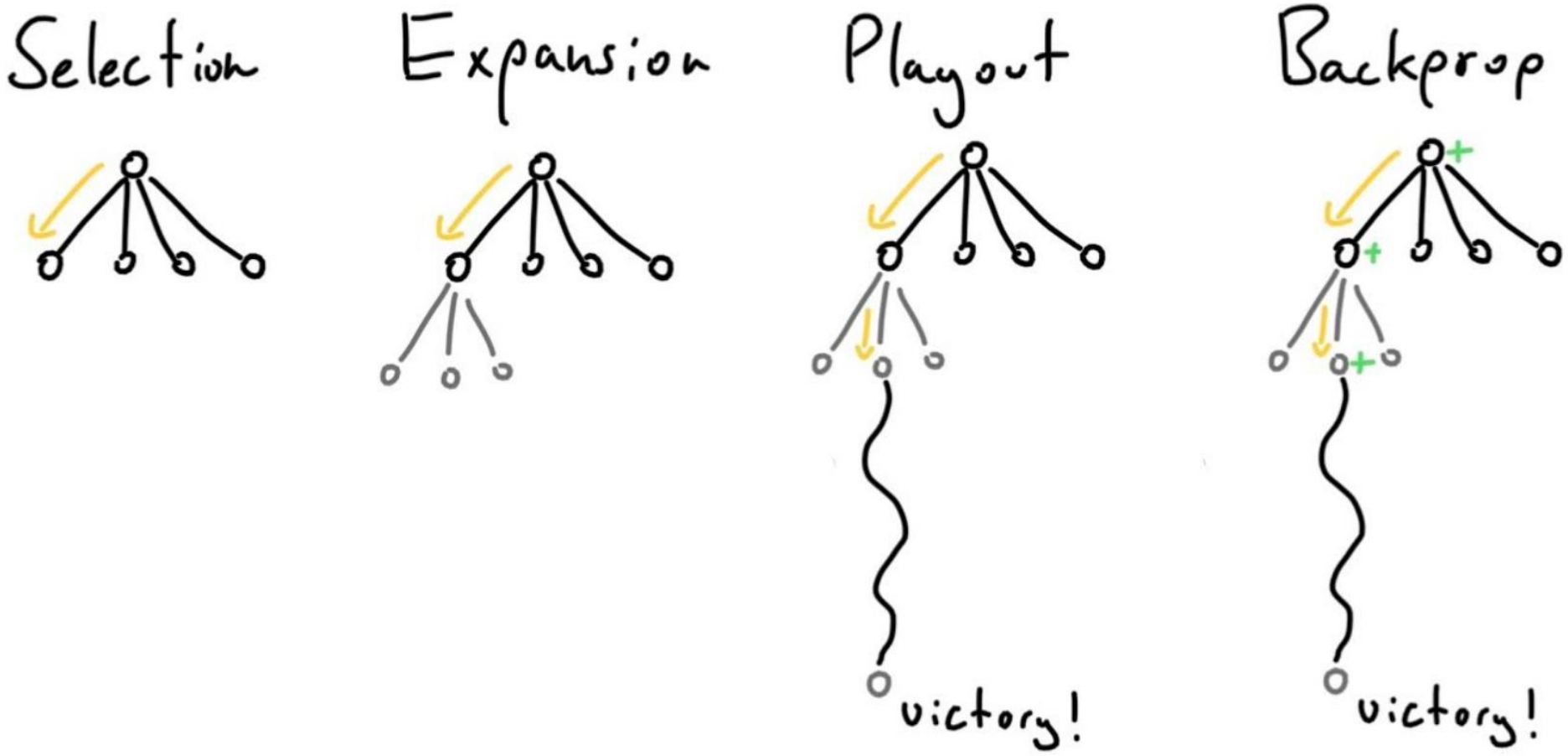
Monte Carlo
Tree Search

Monte Carlo Methods

- A Monte Carlo method is an application of sampling

- E.g. to calculate area of a polygon
- E.g. in a 3D renderer, sample random rays of light
- E.g. in game playing, sample actions to estimate utility

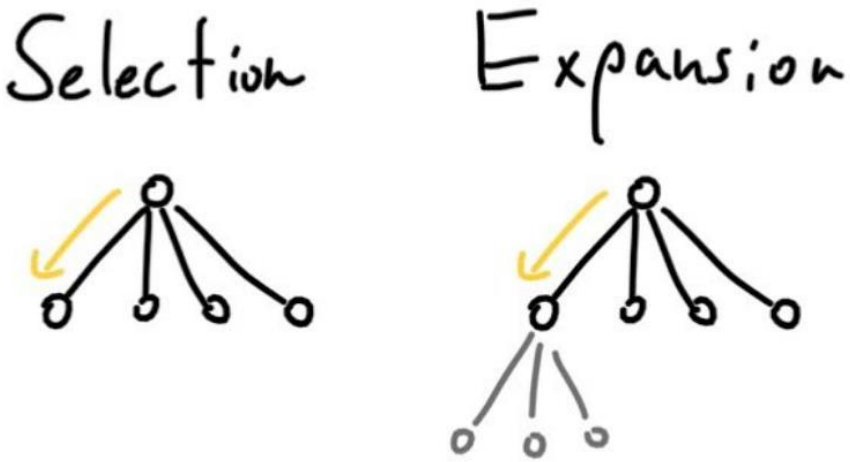
Monte Carlo Tree Search (MCTS)



MCTS Basic Algorithm

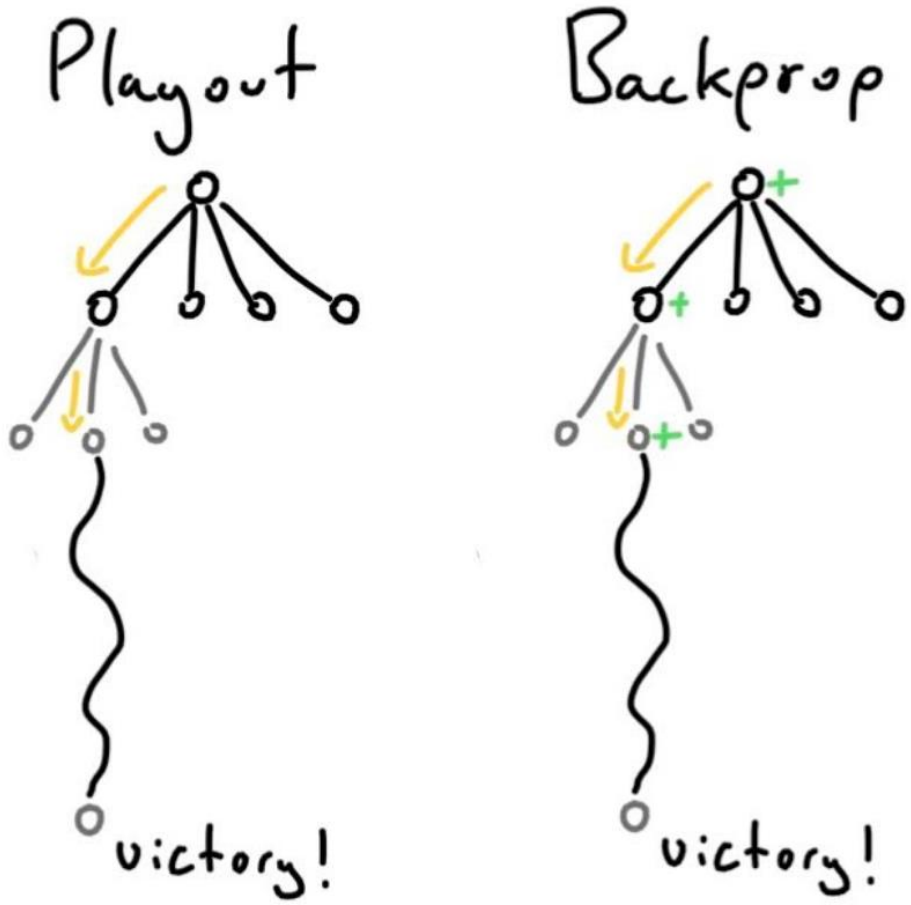
- Selection
  - Starting at root node *R*, recursively select ***optimal*** child nodes (explained soon) until a leaf node *L* is reached.

- Expansion
  - If *L* is a not a terminal node (i.e. it does not end the game) then create one or more child nodes and select one *C*.

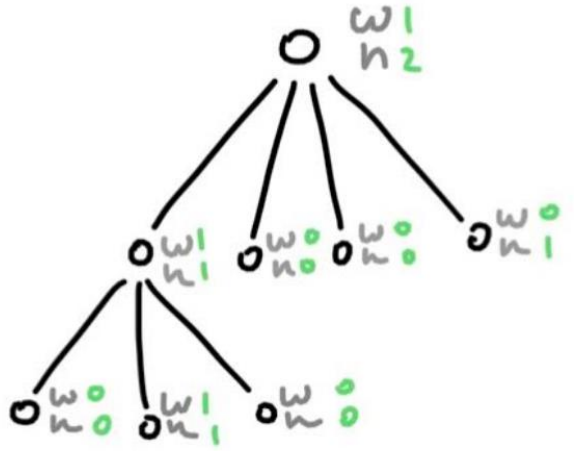


MCTS Basic Algorithm

- Simulation
  - Run a simulated roll out randomly from C until a result is achieved.
- Backpropagation
  - Update the current move sequence with the simulation result.
  - Each node must contain two important pieces of information: an estimated value based on simulation results and the number of times it has been visited.



Node Selection



- Bandits and UCB
  - Node selection during tree descent is achieved by choosing the node that maximizes some quantity, analogous to the multi-armed bandit problem in which a player must choose the slot machine (bandit) that maximizes the estimated reward each turn.

- An Upper Confidence Bounds (UCB) formula of the following form is typically used:

$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$

v_i: the estimated value of the node,
n_i: the number of the times the node has been visited
N: the total number of times that its parent has been visited
C: a tunable bias parameter.

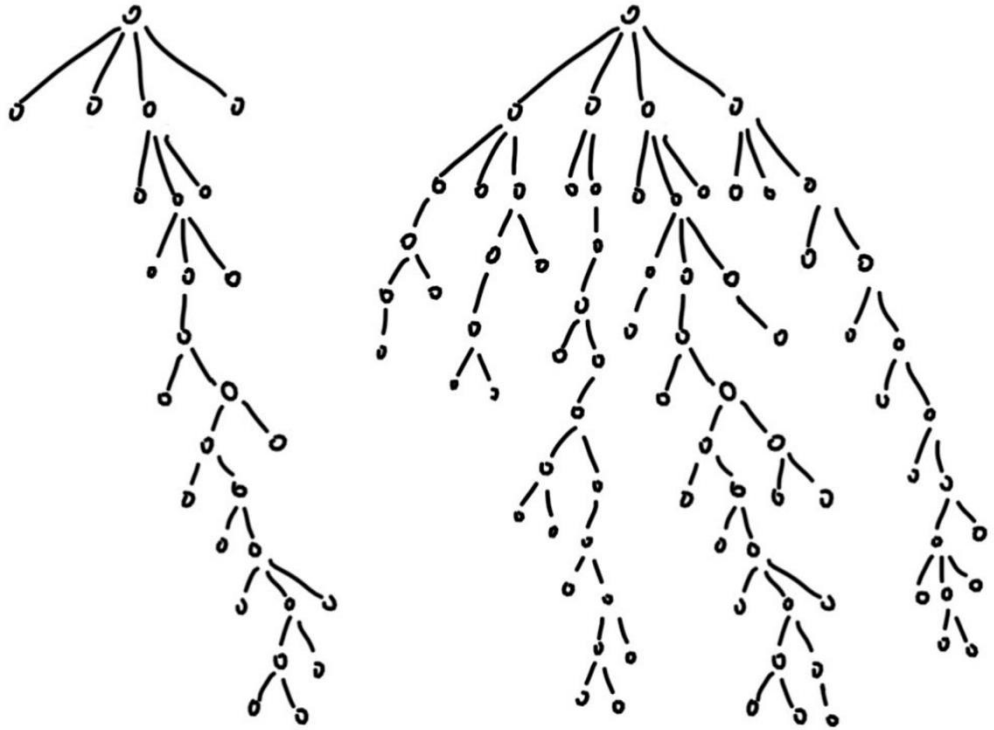
Exploitation vs Exploration

Choose the action that maximizes:

$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$

Exploitation

Exploration



Early on in the simulation, *exploration* dominates,
but later, *exploitation* is more important

Upper Confidence Bounds for Trees (UCT)

$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$

- Predicted mean: number of wins through node I (w_i) divided by number of simulations through node I (n_i)

- **UCT = MCTS + UCB**

Applications of MCTS

- Alpha* family (AlphaGo) algorithms combine RL and MCTS
- RL to learn a value function
- MCTS to pick moves
  - Only rolls out one step each "Simulation" step

Benefits

- Heuristic
  - MCTS does not require any strategic or tactical knowledge about the given domain to make reasonable decisions.
  - The algorithm can function effectively with no knowledge of a game apart from its legal moves and end conditions
- Asymmetric
  - MCTS performs asymmetric tree growth that adapts to the topology of the search space. The algorithm visits more interesting nodes more often, and focusses its search time in more relevant parts of the tree
- Anytime
- Elegant

Drawbacks

- Playing Strength
  - The MCTS algorithm can fail to find reasonable moves for even games of medium complexity within a reasonable amount of time.
  - Mostly due to the sheer size of the combinatorial move space and the fact that key nodes may not be visited enough times to give reliable estimates.

- Speed
  - MCTS search can take many iterations to converge to a good solution

Goals

- Describe and formulate a game problem.
- Describe zero-sum games and game trees
- Implement minimax tree search and alpha-beta pruning.
- Understand how Monte Carlo Tree Search works.

Important This Week

- Do some exercises on the textbook, and ask questions (optionally).
- Register for the class Piazza – not all of you have registered.
- Practice the tutorial this week for map coloring.
- Start forming final project groups if you want to work in a group.