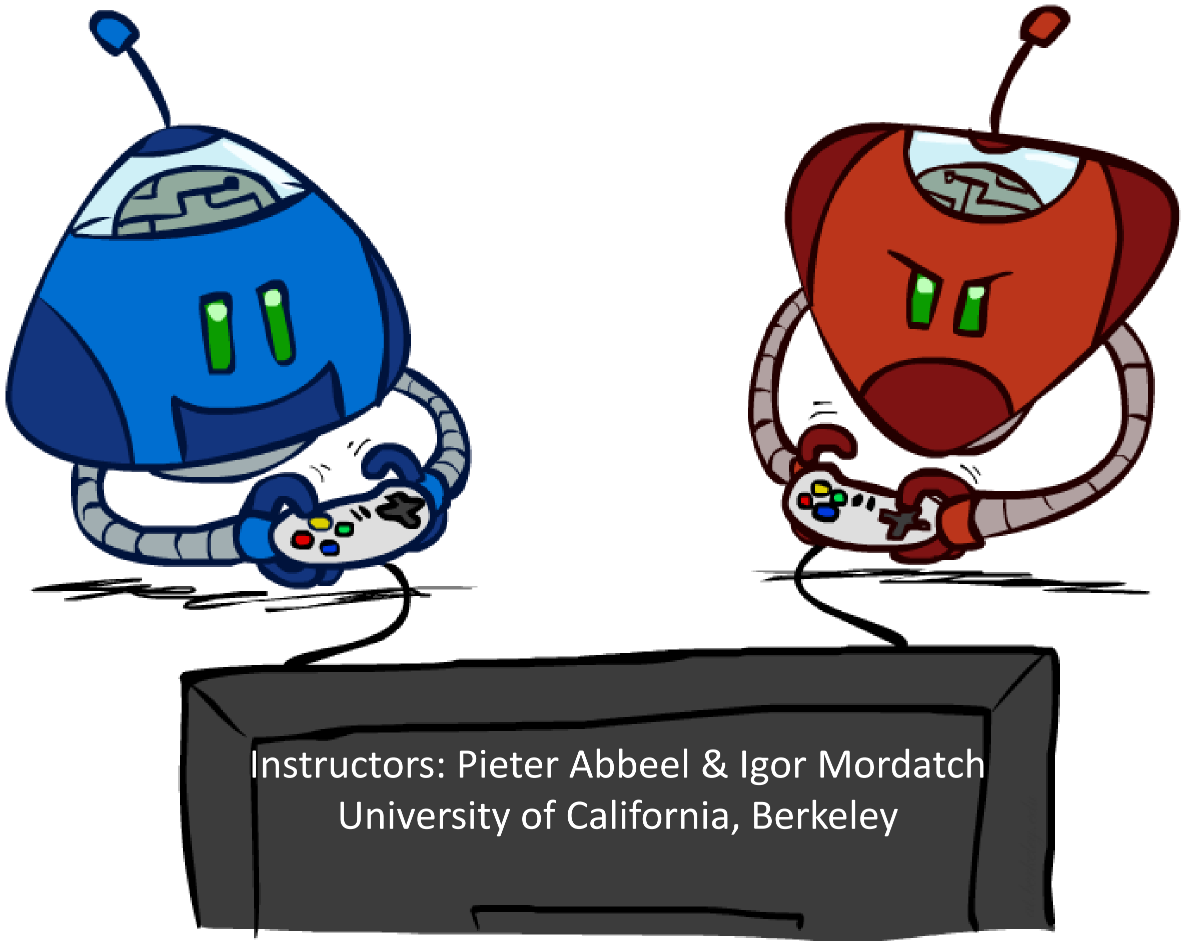
Announcements

- HW2 + Self-assessment HW1 due tonight
  - Electronic HW2
  - Written HW2
  - Self-assessment HW1
- HW3 Games will go out soon (due Tue 9/24 at 11:59pm)
- P2 Games will go out soon (due Fri 9/27 at 5pm)

CS 188: Artificial Intelligence

Adversarial Search

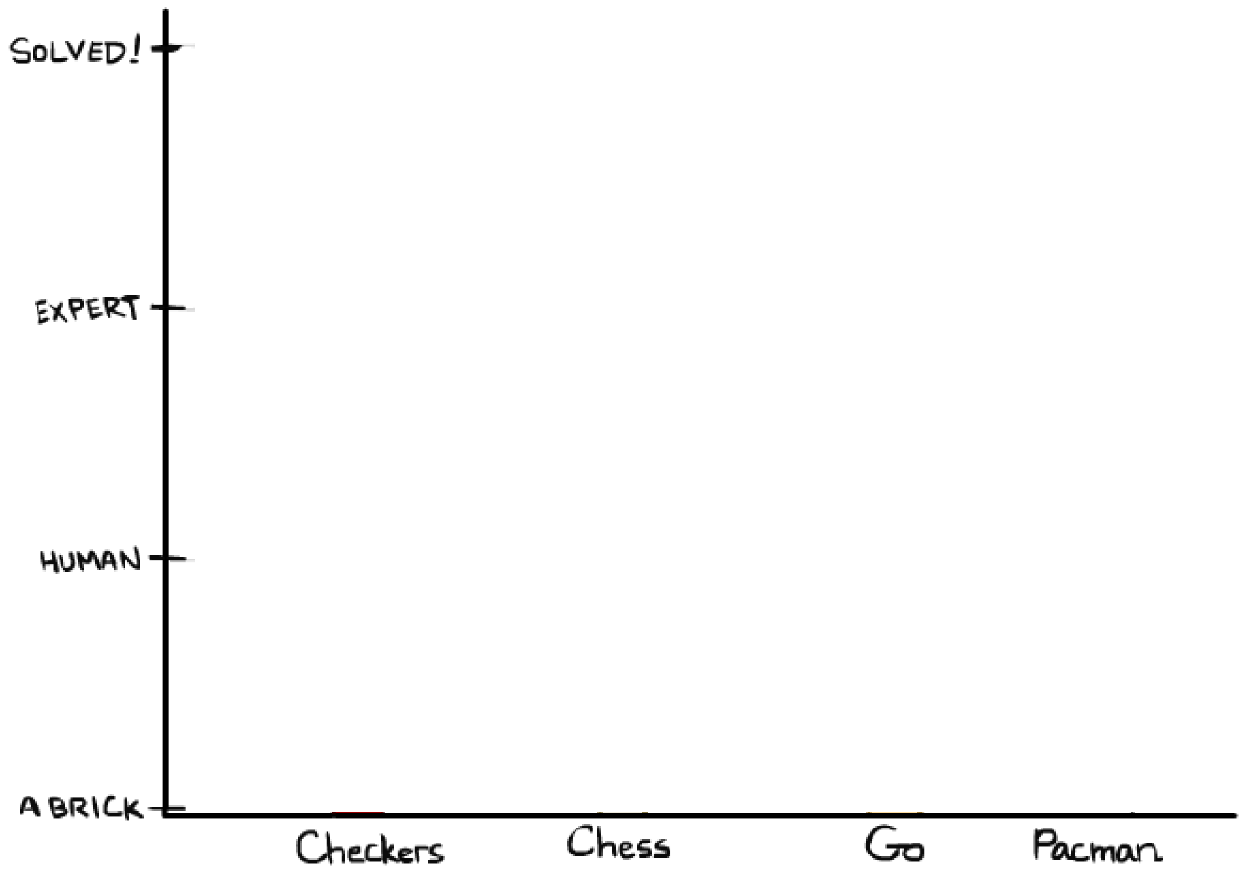


Instructors: Pieter Abbeel & Igor Mordatch
University of California, Berkeley

[Many of these slides were originally created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley]

Game Playing State-of-the-Art

- **Checkers:**
  - 1950: First computer player
  - 1959: Samuel's self-taught program
  - 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame.
  - 2007: Checkers solved!
- **Chess:**
  - 1945-1960: Zuse, Wiener, Shannon, Turing, Newell&Simon, McCarthy
  - 1960-1996: gradual improvements
  - 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match
  - 2024: Stockfish rating 3631 (vs 2847 for Magnus Carlsen)
- **Go:**
  - 1968: Zobrist's program plays legal Go, barely (b>300!)
  - 1968-2005: various ad hoc approaches tried, novice level
  - 2005-2014: Monte Carlo tree search -> strong amateur
  - 2017-2017: Alphago defeats human world champion
  - 2022: human exploits NN weakness to defeat top Go programs



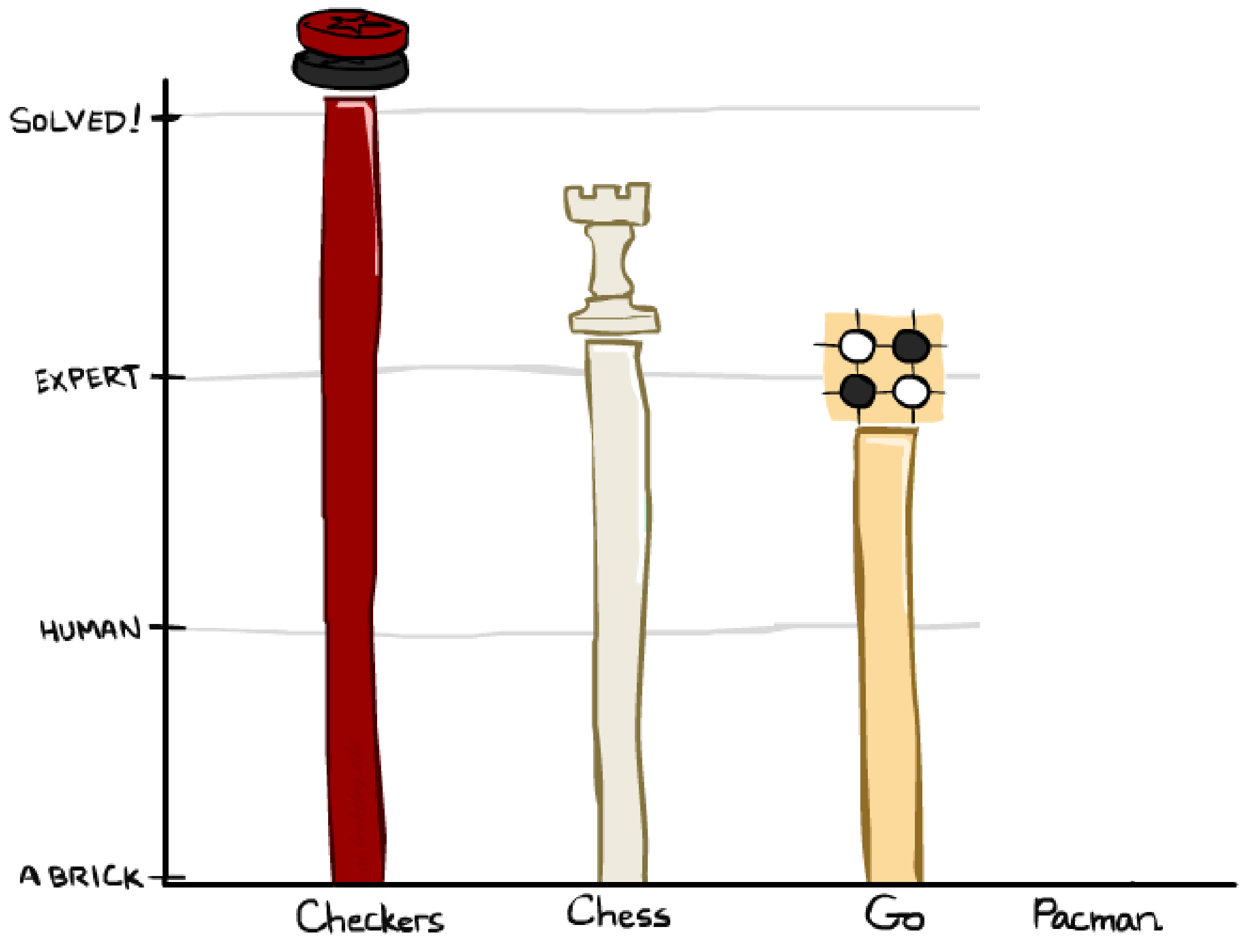
Game Playing State-of-the-Art

- **Checkers:**
  - 1950: First computer player
  - 1959: Samuel's self-taught program
  - 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame.
  - 2007: Checkers solved!

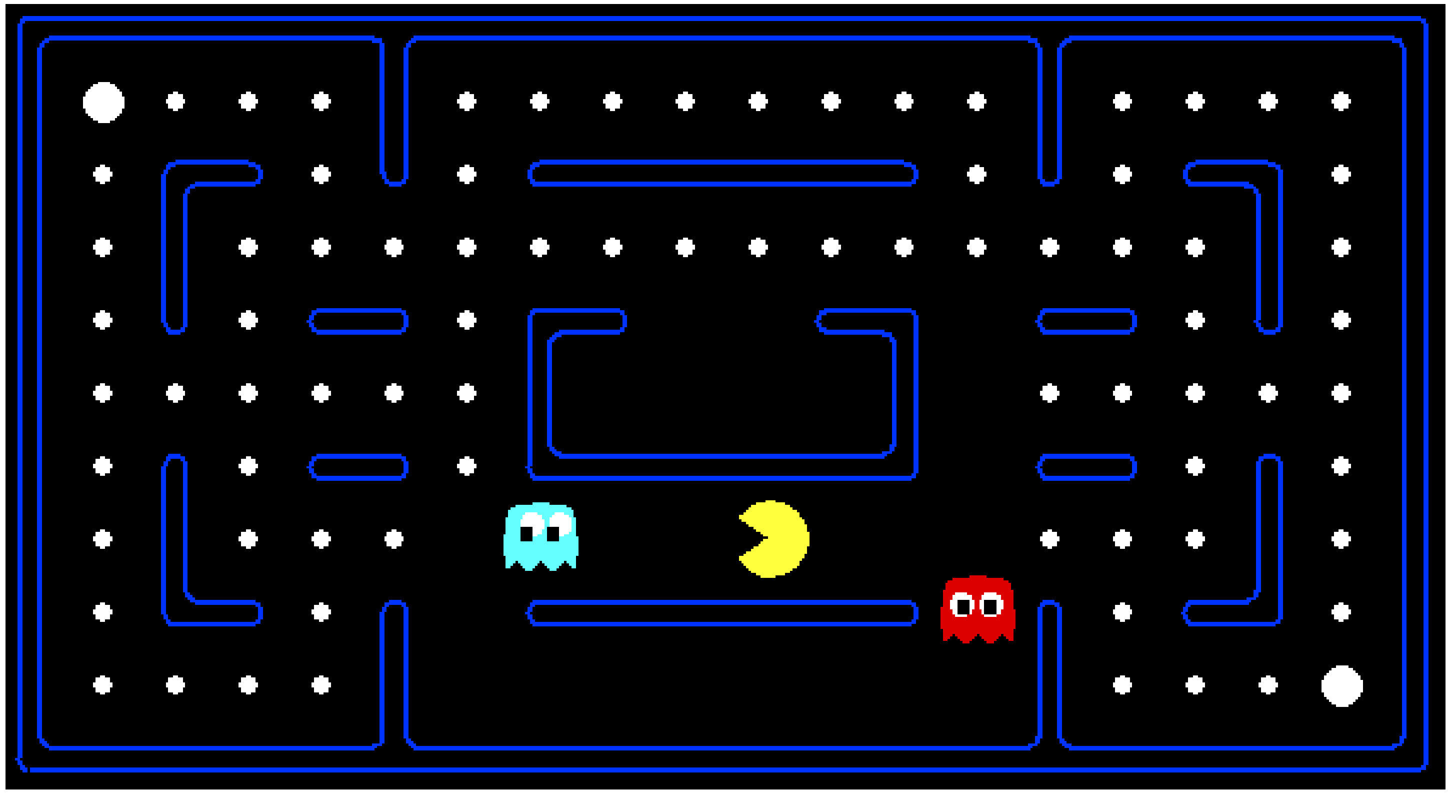
- **Chess:**
  - 1945-1960: Zuse, Wiener, Shannon, Turing, Newell&Simon, McCarthy
  - 1960-1996: gradual improvements
  - 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match
  - 2024: Stockfish rating 3631 (vs 2847 for Magnus Carlsen)

- **Go:**
  - 1968: Zobrist's program plays legal Go, barely (b>300!)
  - 1968-2005: various ad hoc approaches tried, novice level
  - 2005-2014: Monte Carlo tree search -> strong amateur
  - 2017-2017: Alphago defeats human world champion
  - 2022: human exploits NN weakness to defeat top Go programs

- **Pacman:**



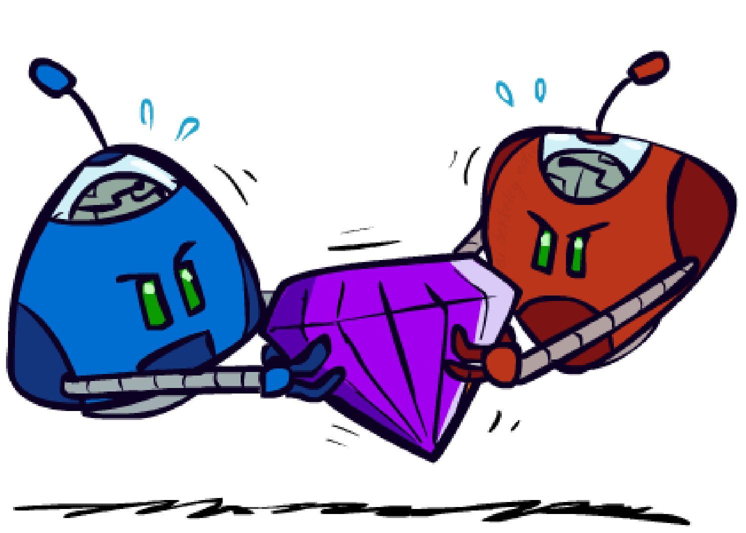
Behavior from Computation

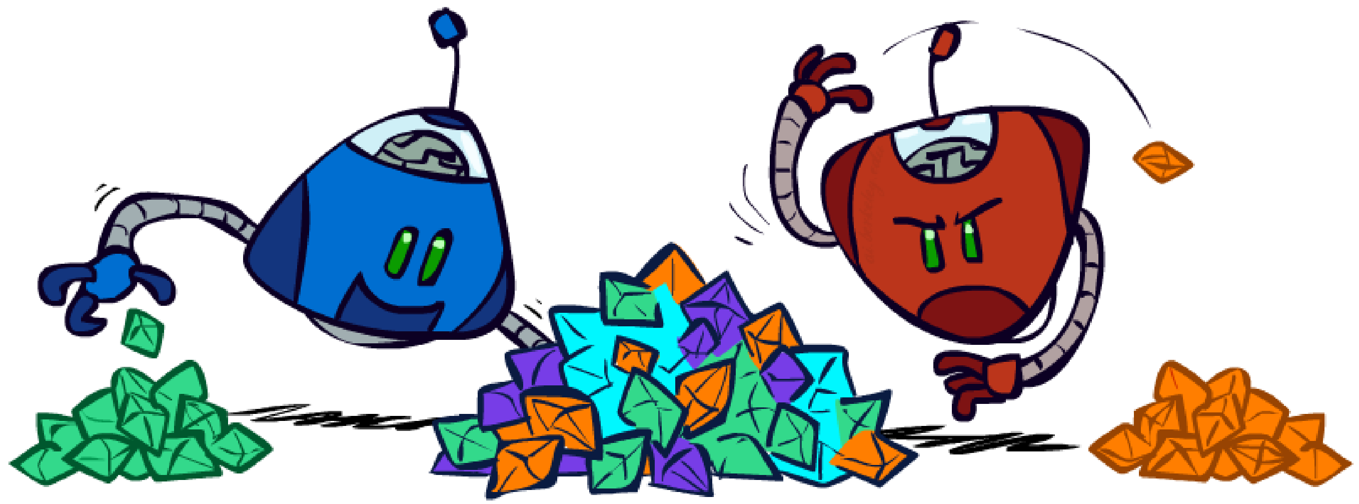


[Demo: mystery pacman (L6D1)]

Video of Demo Mystery Pacman

Types of 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

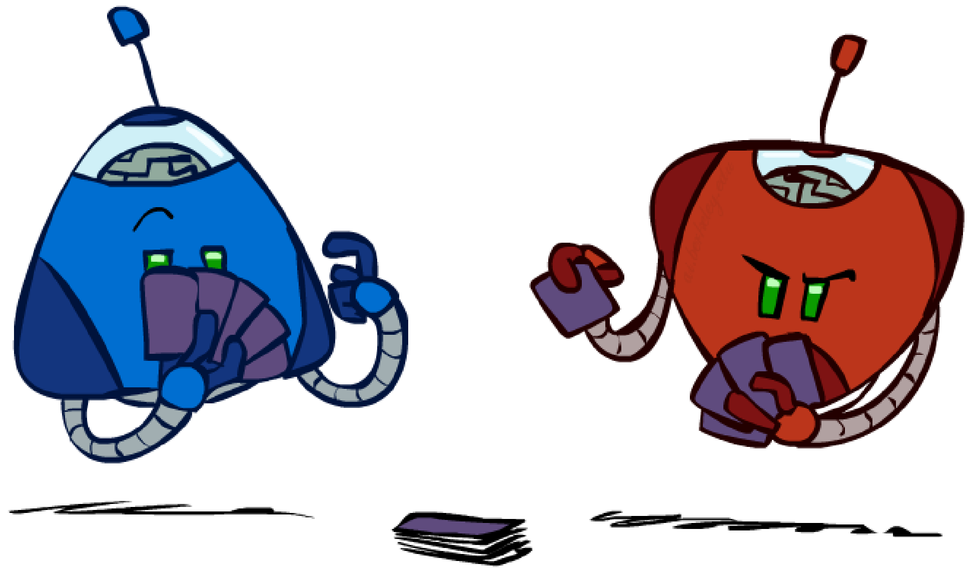
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
    - We don’t make AI to act in isolation, it should a) work around people and b) help people
    - That means that every AI agent needs to solve a game

Types of Games

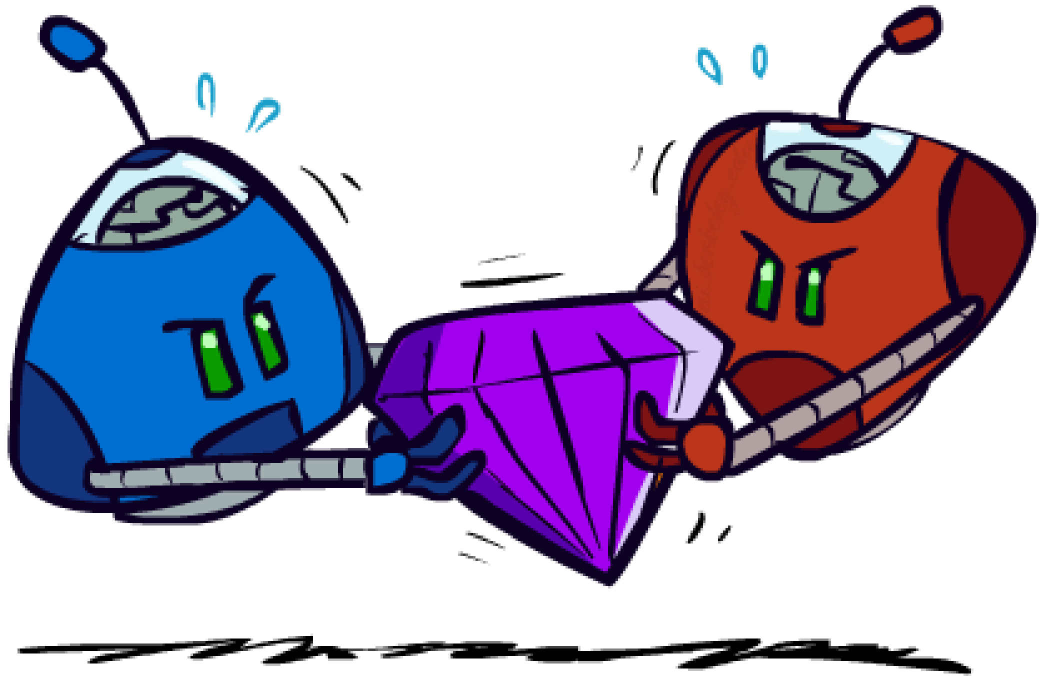
- Many different kinds of games!

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

- Want algorithms for calculating a strategy (policy) which recommends a move from each state --- i.e. not just a sequence of actions

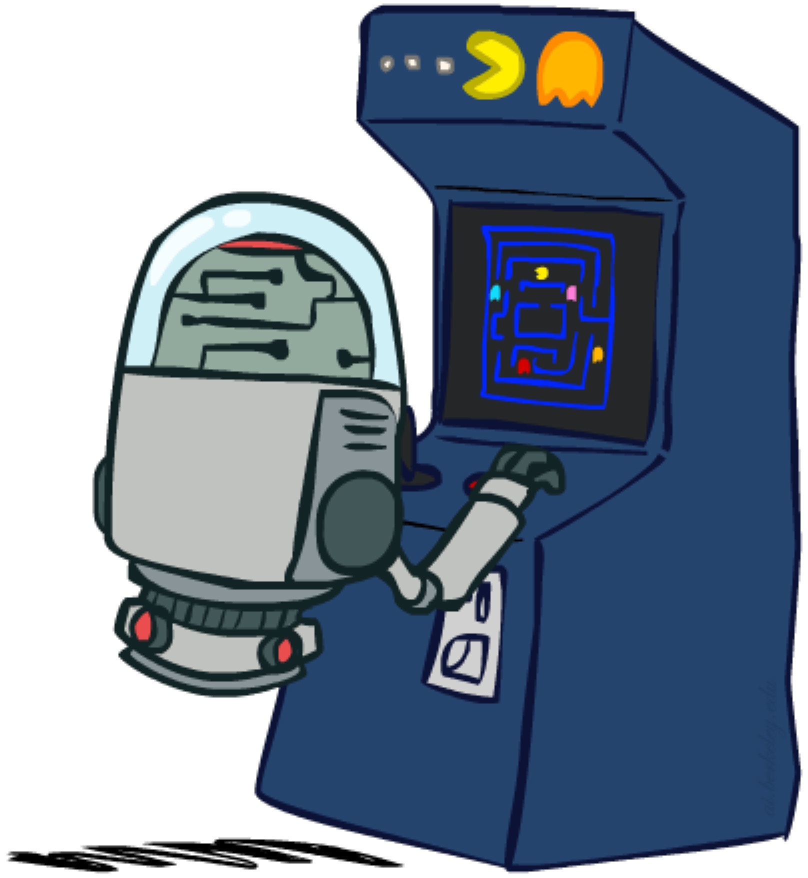


Adversarial Games:
Deterministic, 2-player, zero-sum, perfect information



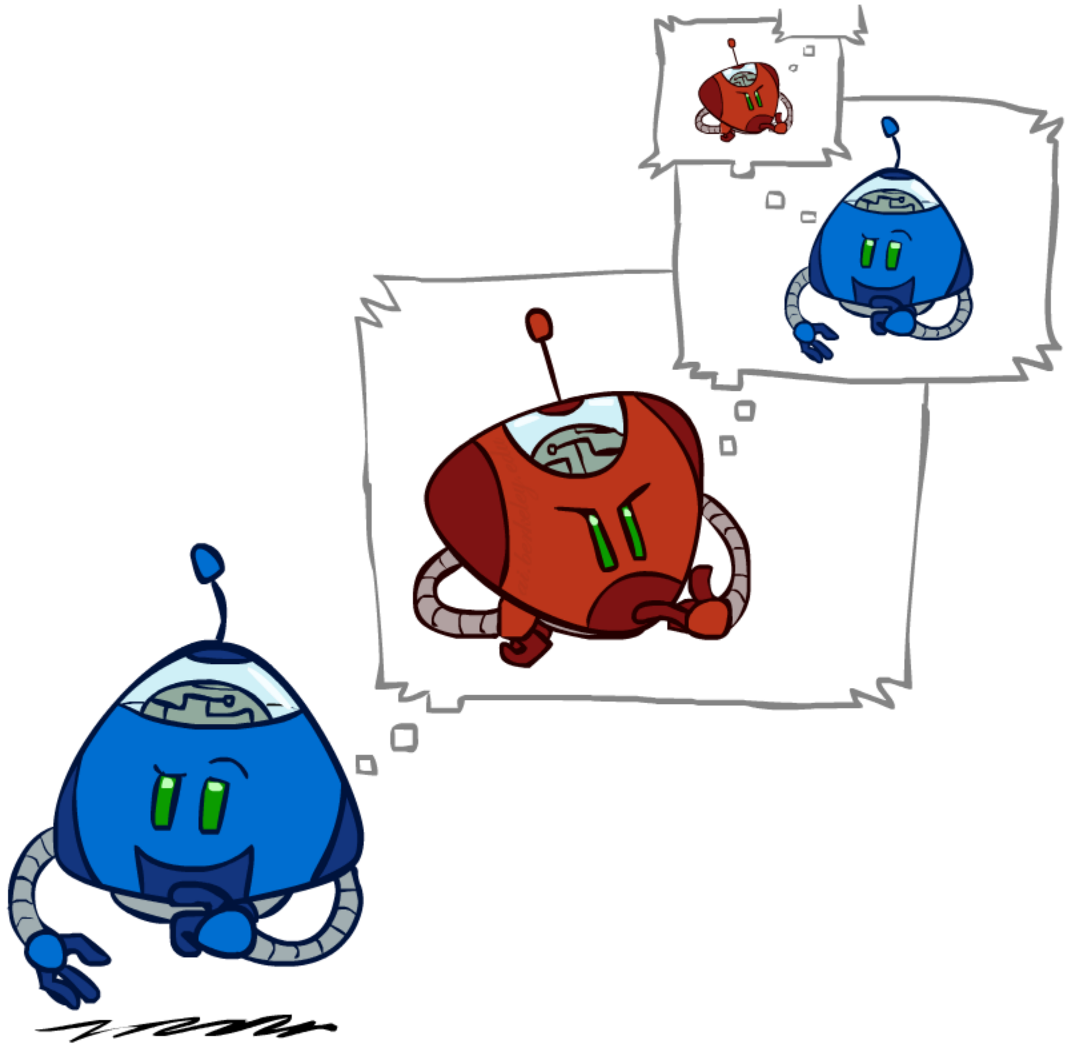
Formalization

- Our formalization of adversarial games:
  - States: S (start at s_0)
  - Players: P={MAX, MIN}
  - Actions: A (may depend on player / state)
  - Transition Function: SxA $\rightarrow$ S
  - Terminal Test: S $\rightarrow$ {true, false}
  - Terminal Utilities: S $\rightarrow$ R (R = "Reward" = ~score)
    MAX maximizes R
    MIN minimizes R

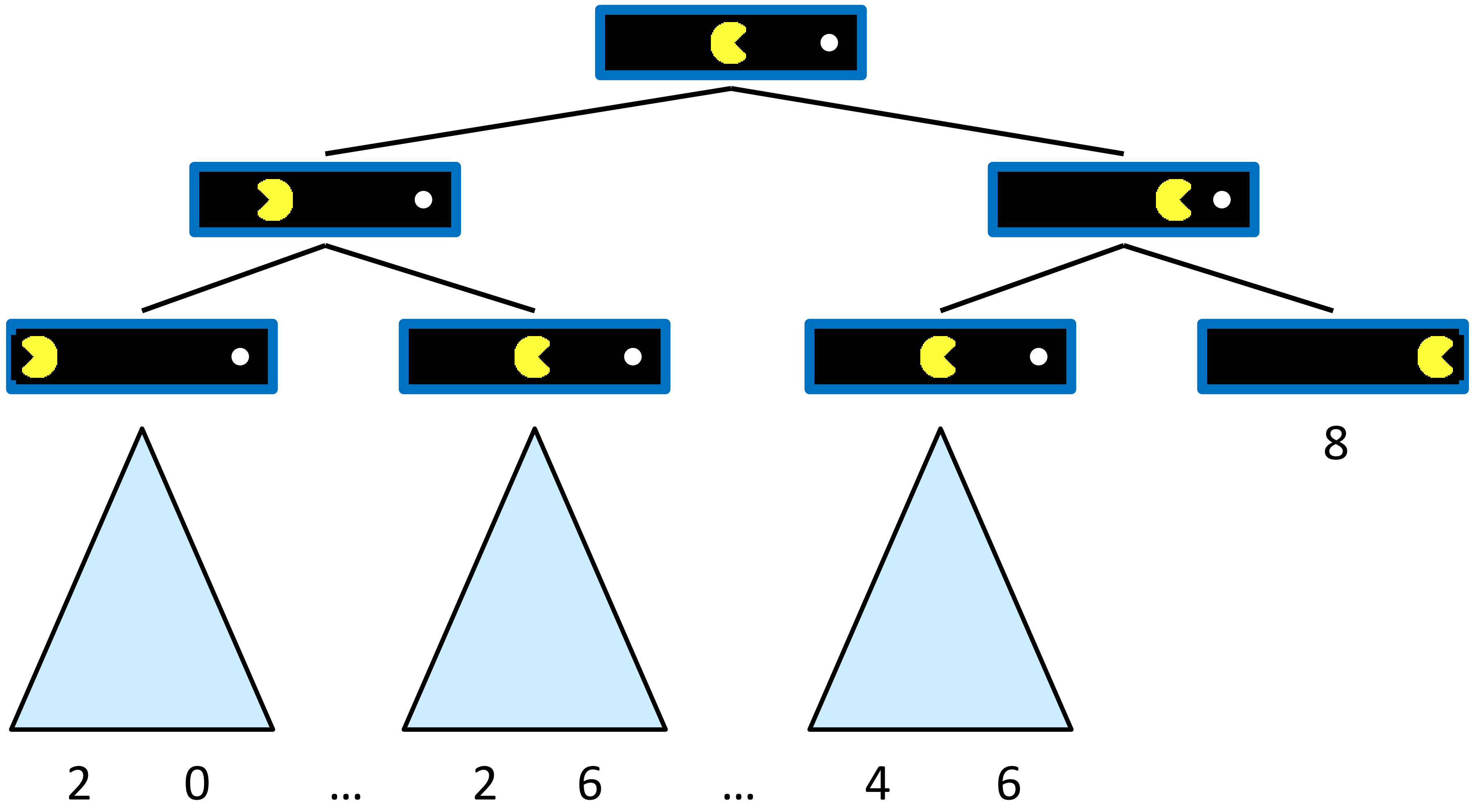


- Solution for a player is a policy: S $\rightarrow$ A

Adversarial Search

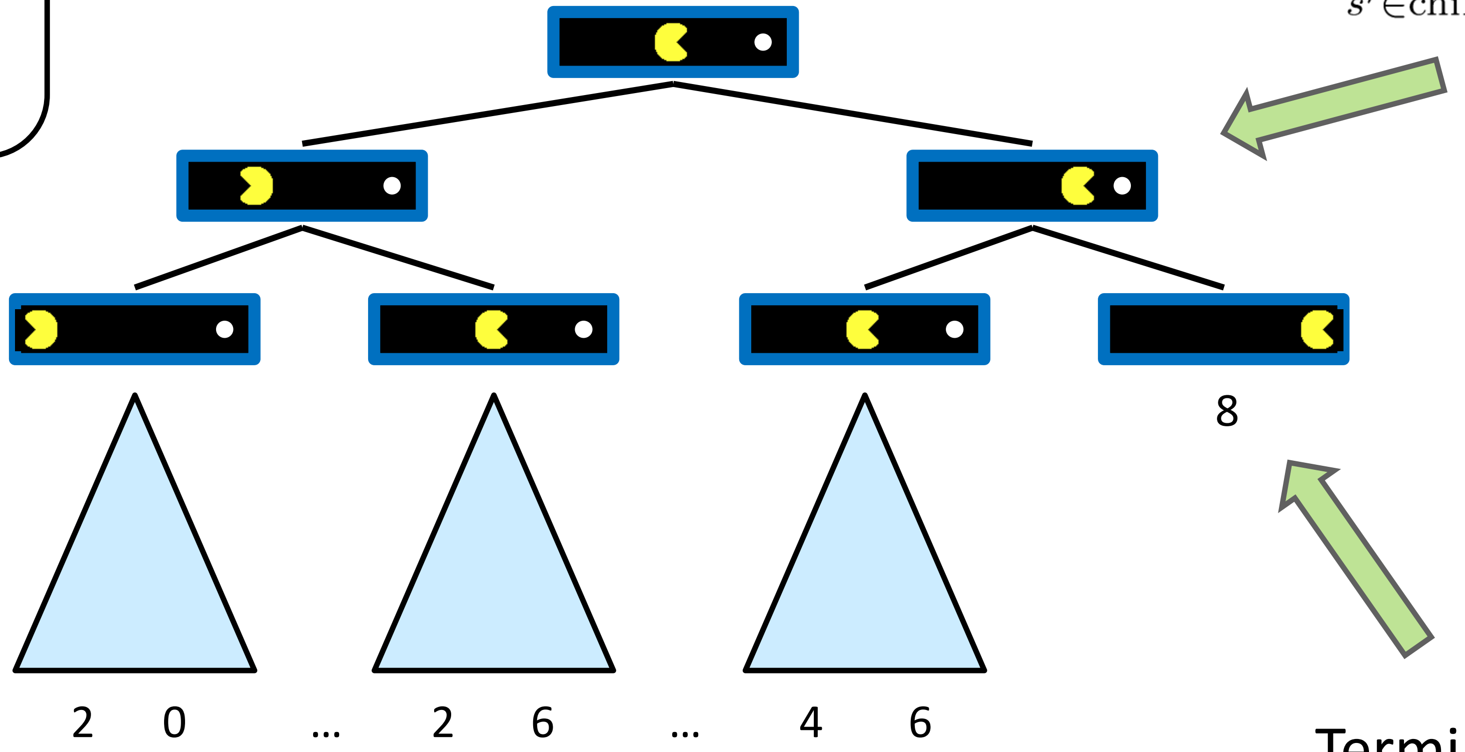


Single-Agent Trees



Value of a State

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



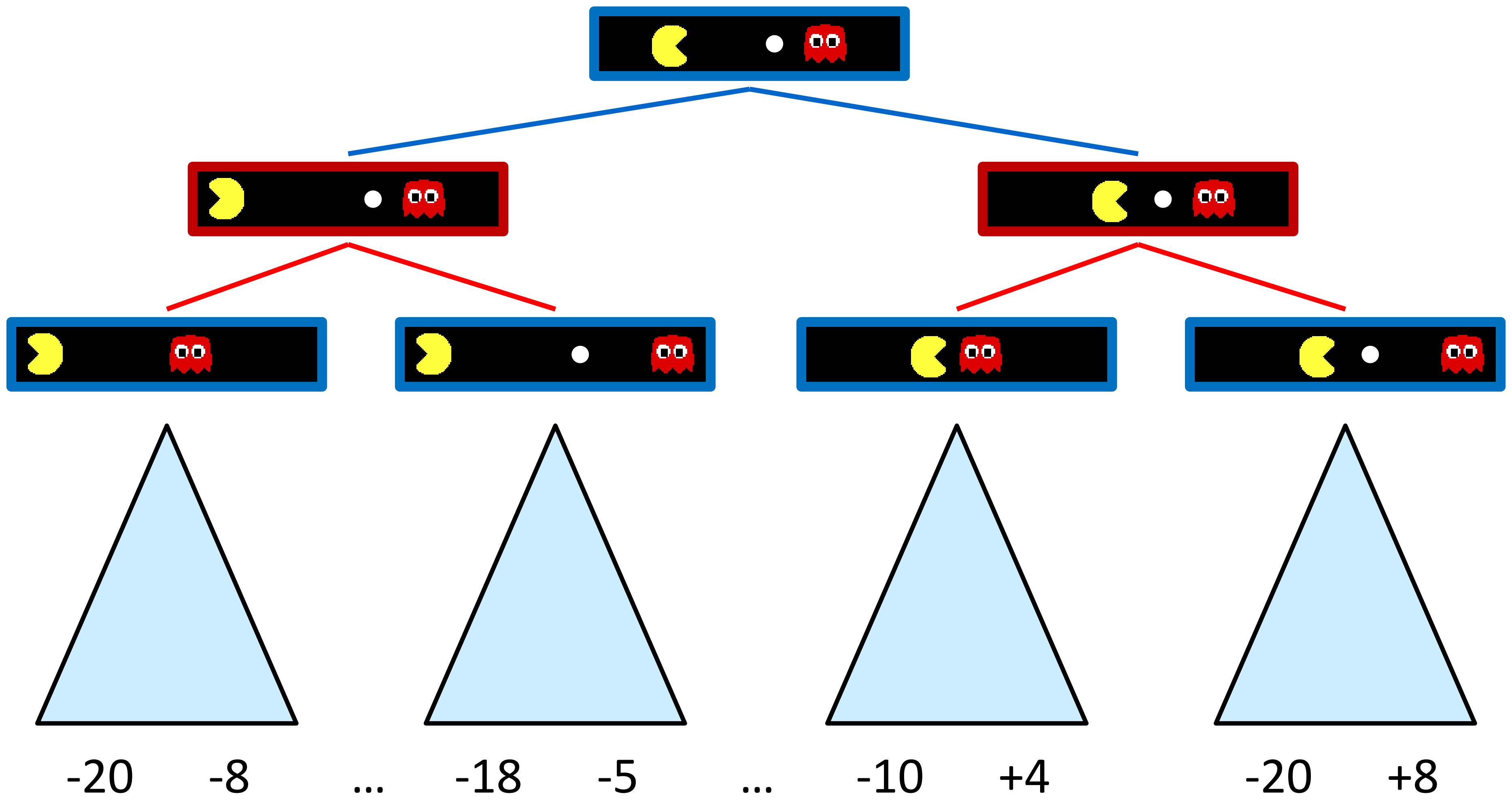
Non-Terminal States:

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

Terminal States:

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

Adversarial Game Trees



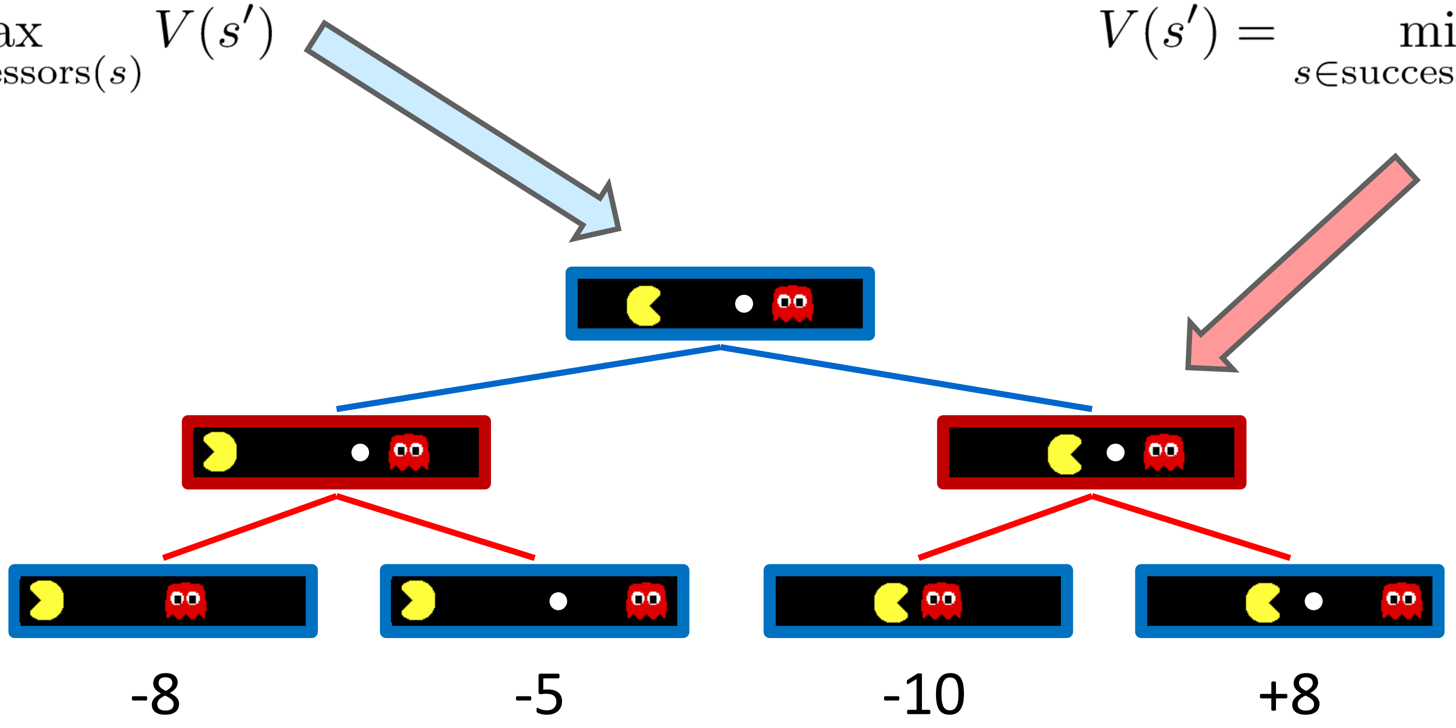
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

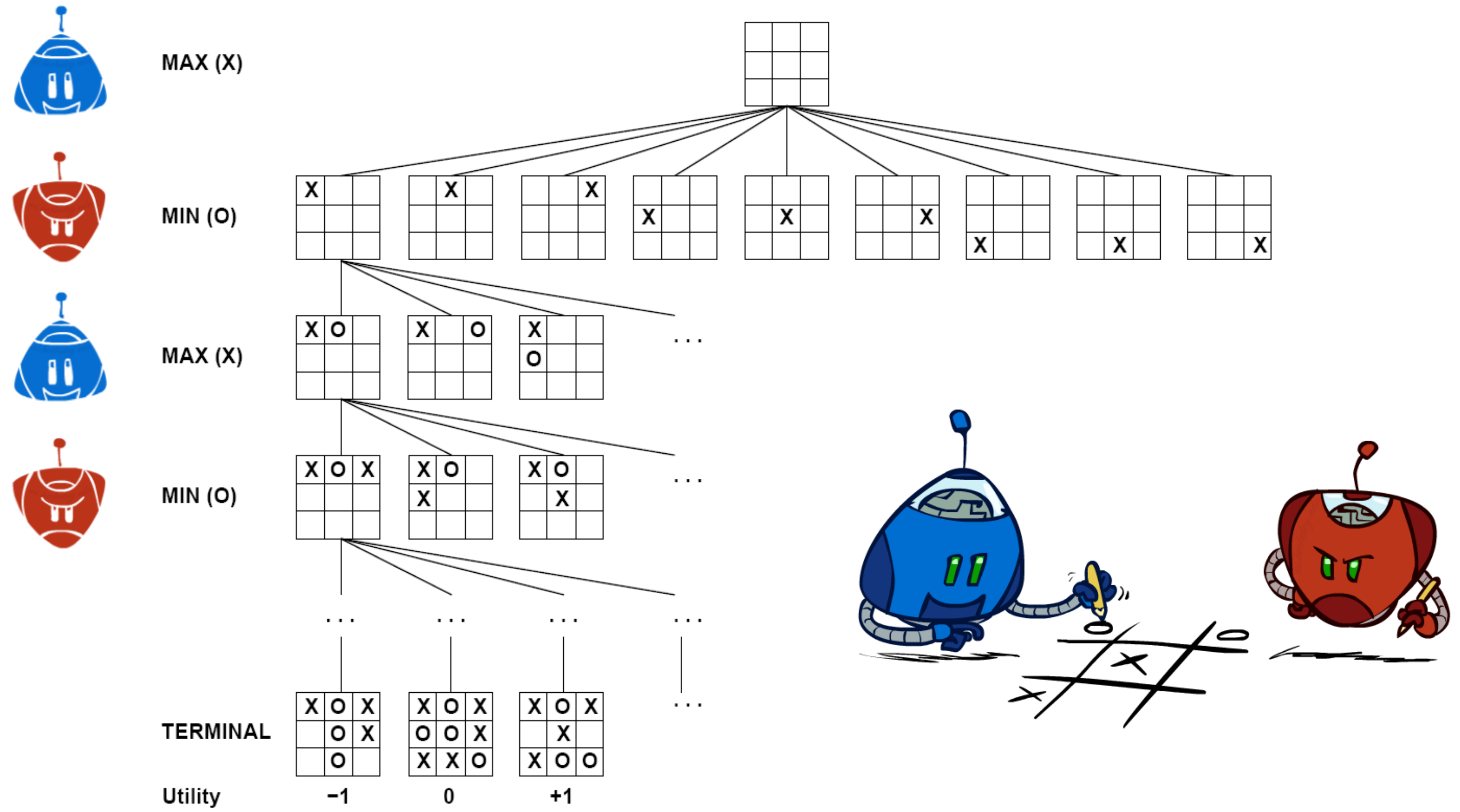
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

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

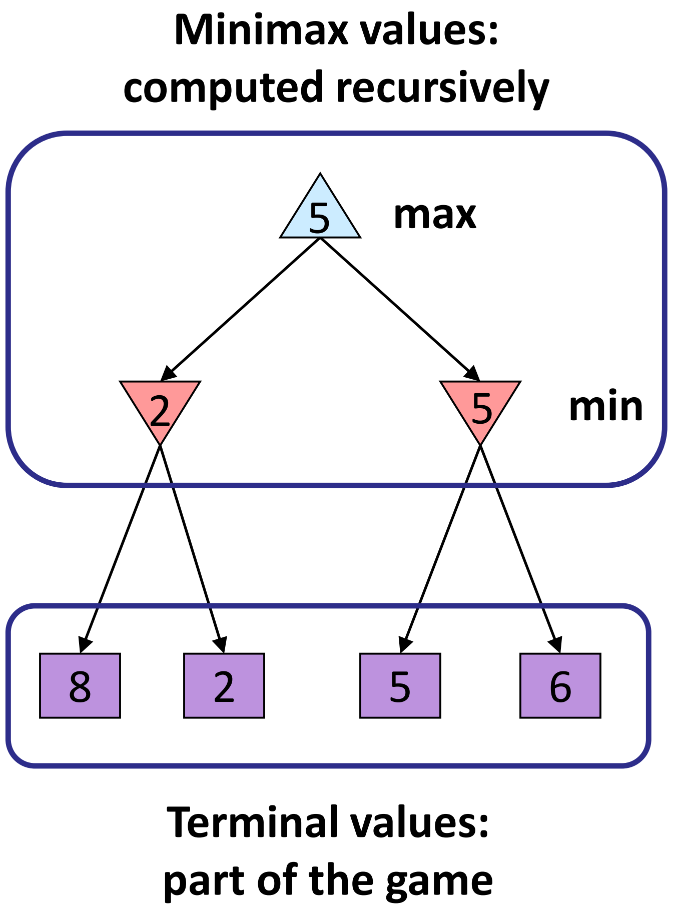
Tic-Tac-Toe Game Tree



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
```

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

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

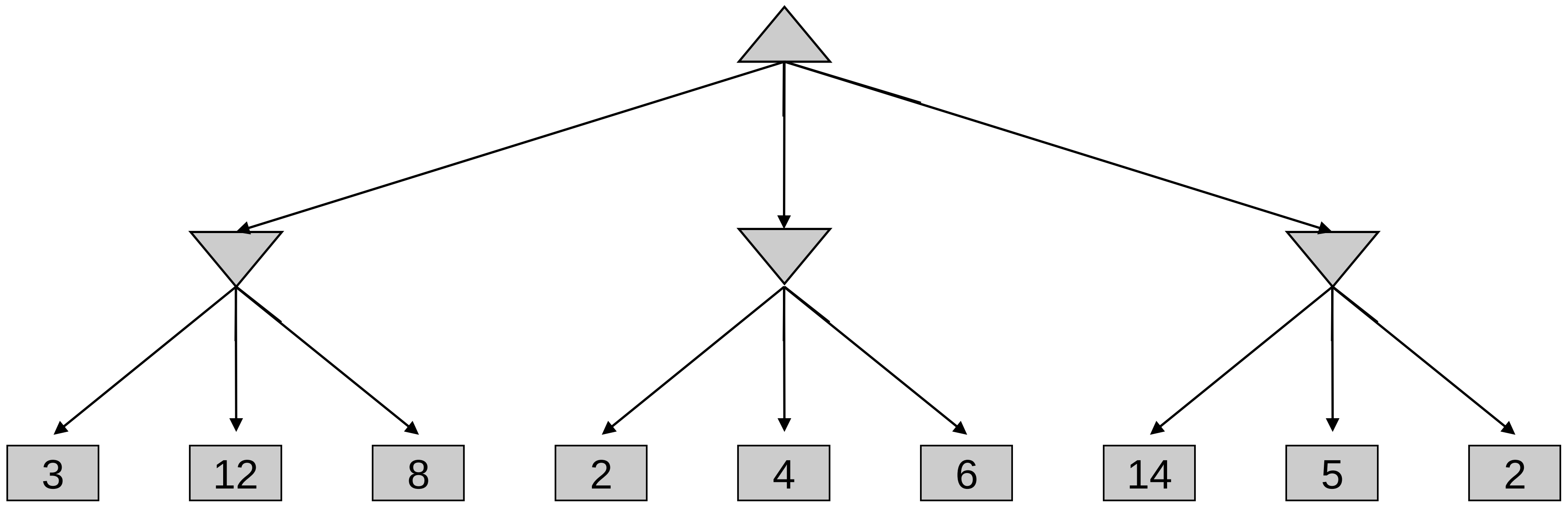
Minimax Implementation (Dispatch)

```
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)
```

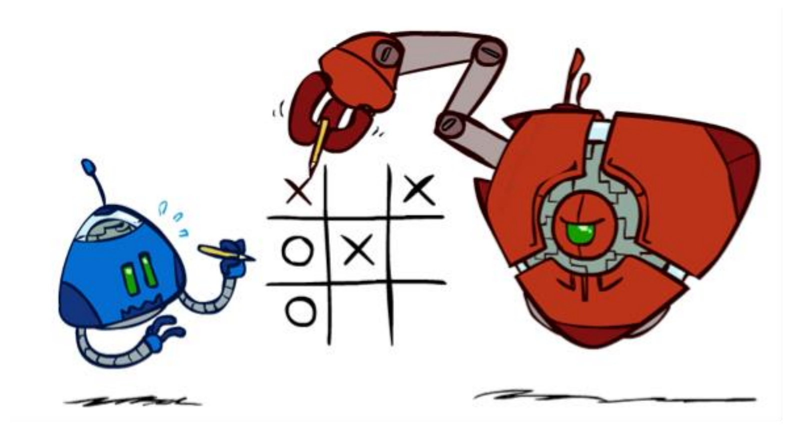
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

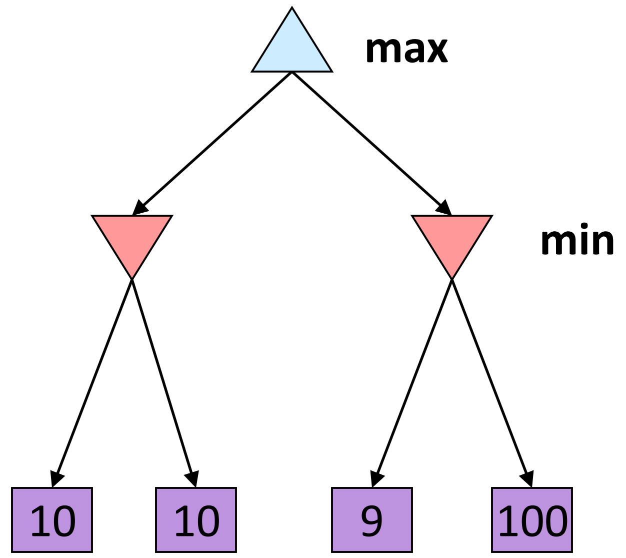
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

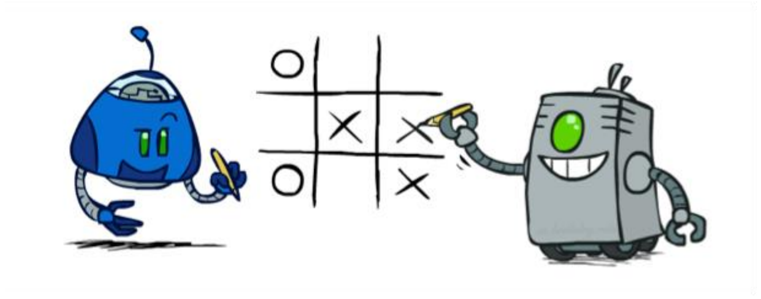
Minimax Example



Minimax Properties







Optimal against a perfect player. Otherwise?

[Demo: min vs exp (L6D2, L6D3)]

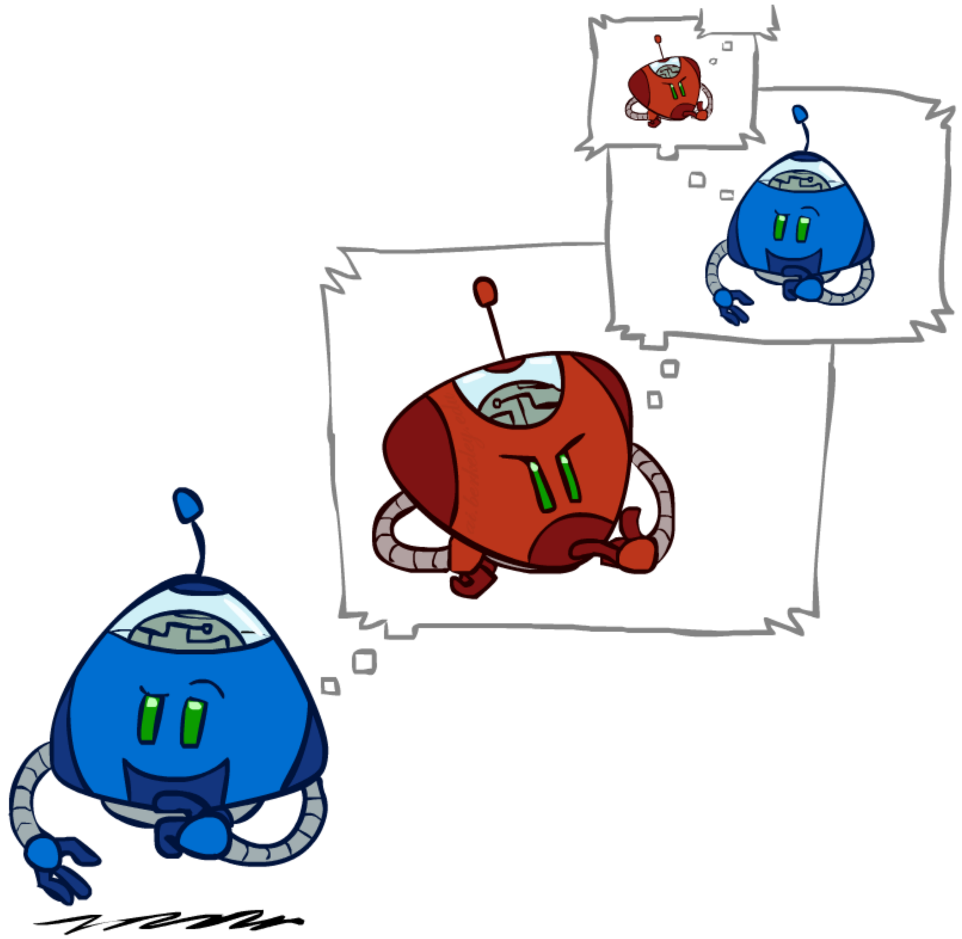
Video of Demo Min vs. Exp (Min)

Video of Demo Min vs. Exp (Exp)

Minimax Efficiency

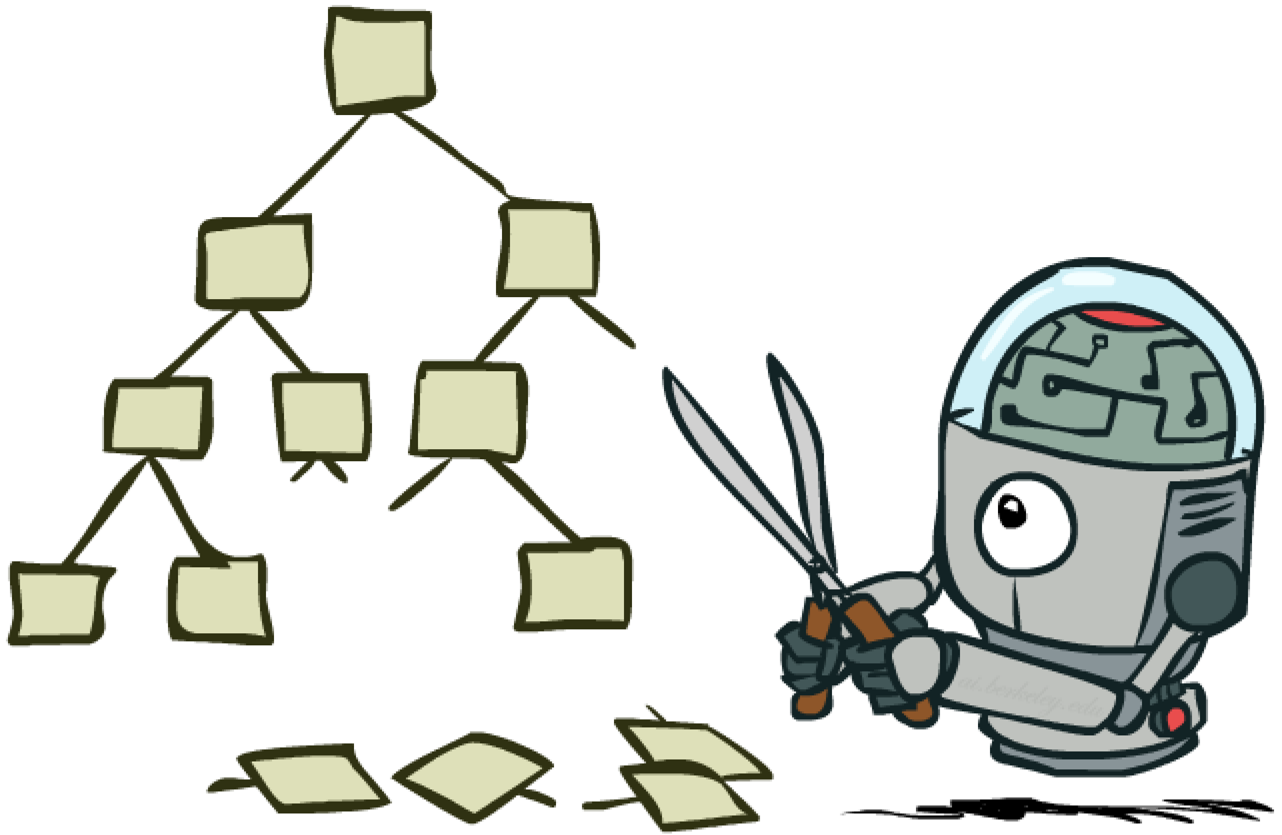
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: O(b^m)
  - Space: O(bm)

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

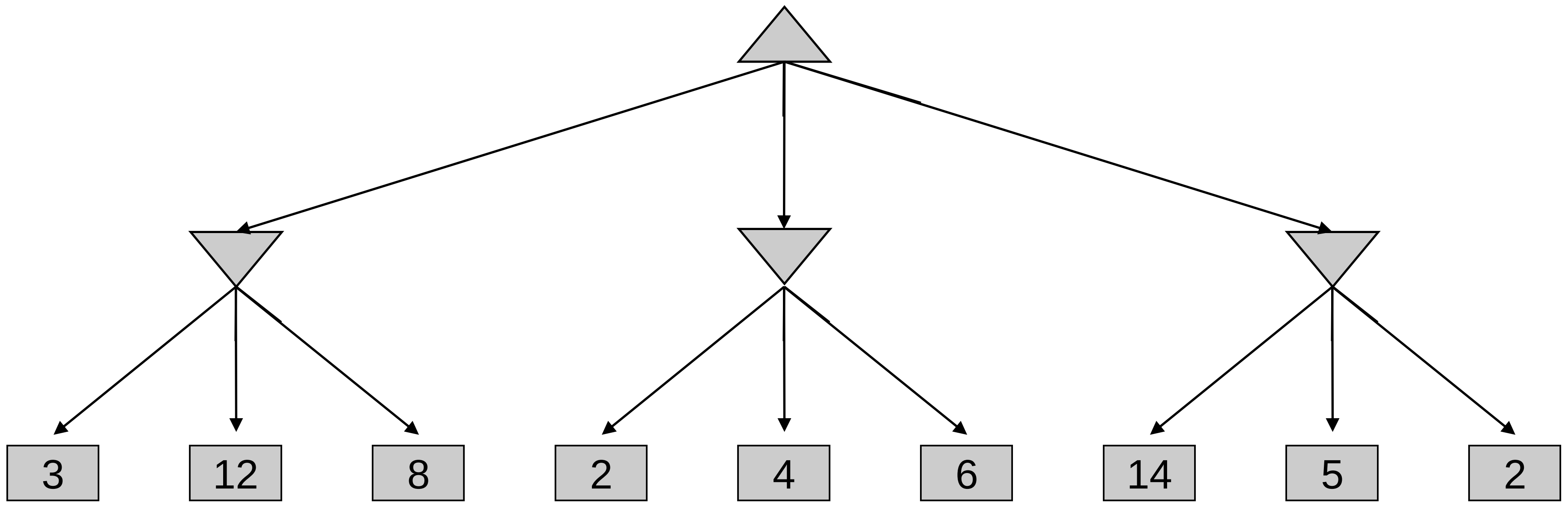


Resource Limits

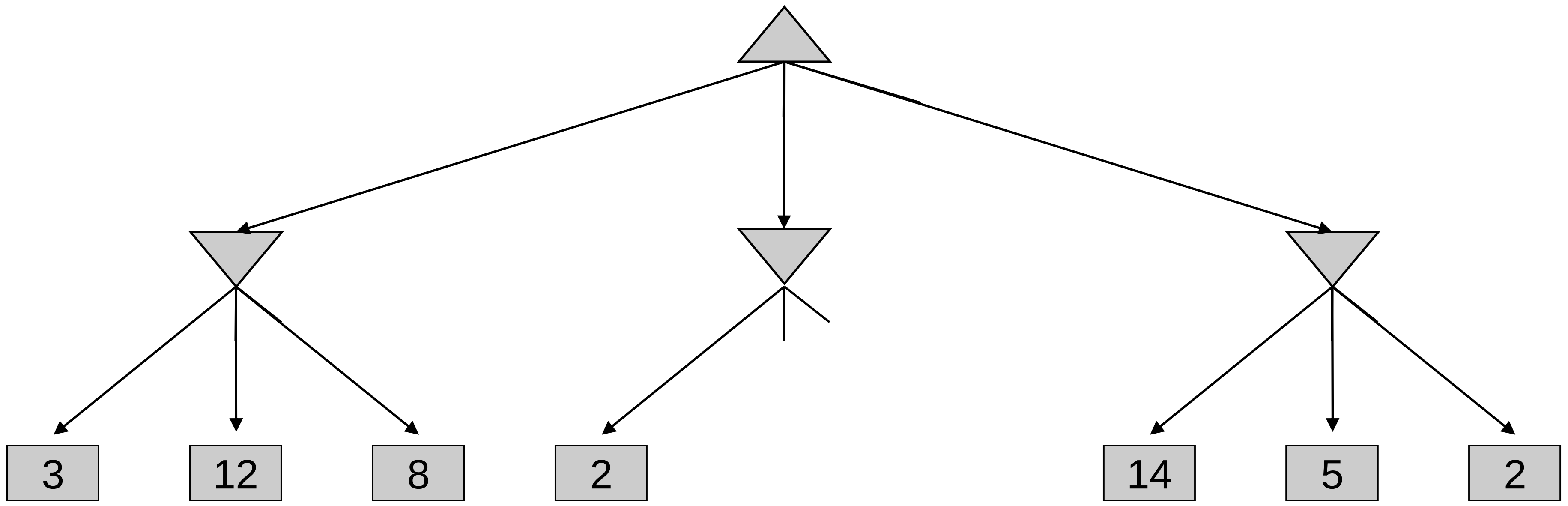
Game Tree Pruning



Minimax Example



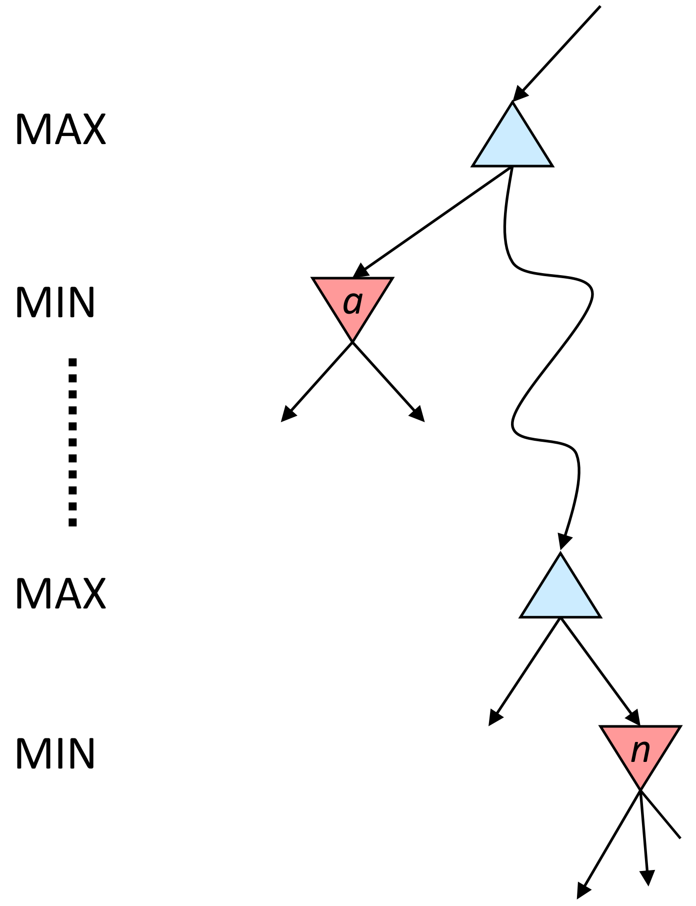
Minimax Pruning



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 Implementation

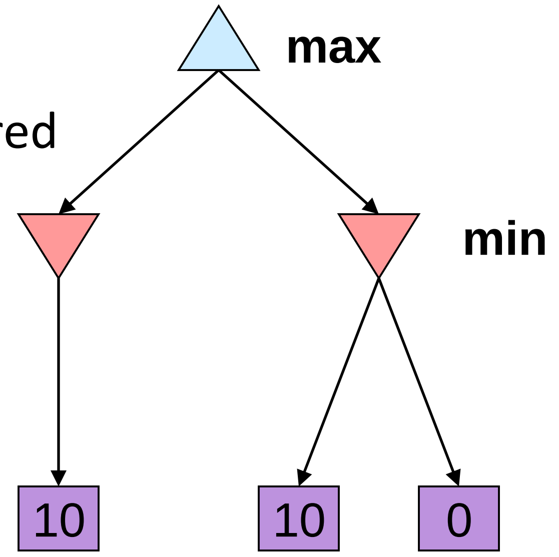
α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, $\alpha$, $\beta$):
    initialize v = $-\infty$
    for each successor of state:
        v = max(v, value(successor, $\alpha$, $\beta$))
        if $v \geq \beta$ return v
        $\alpha$ = max($\alpha$, v)
    return v
```

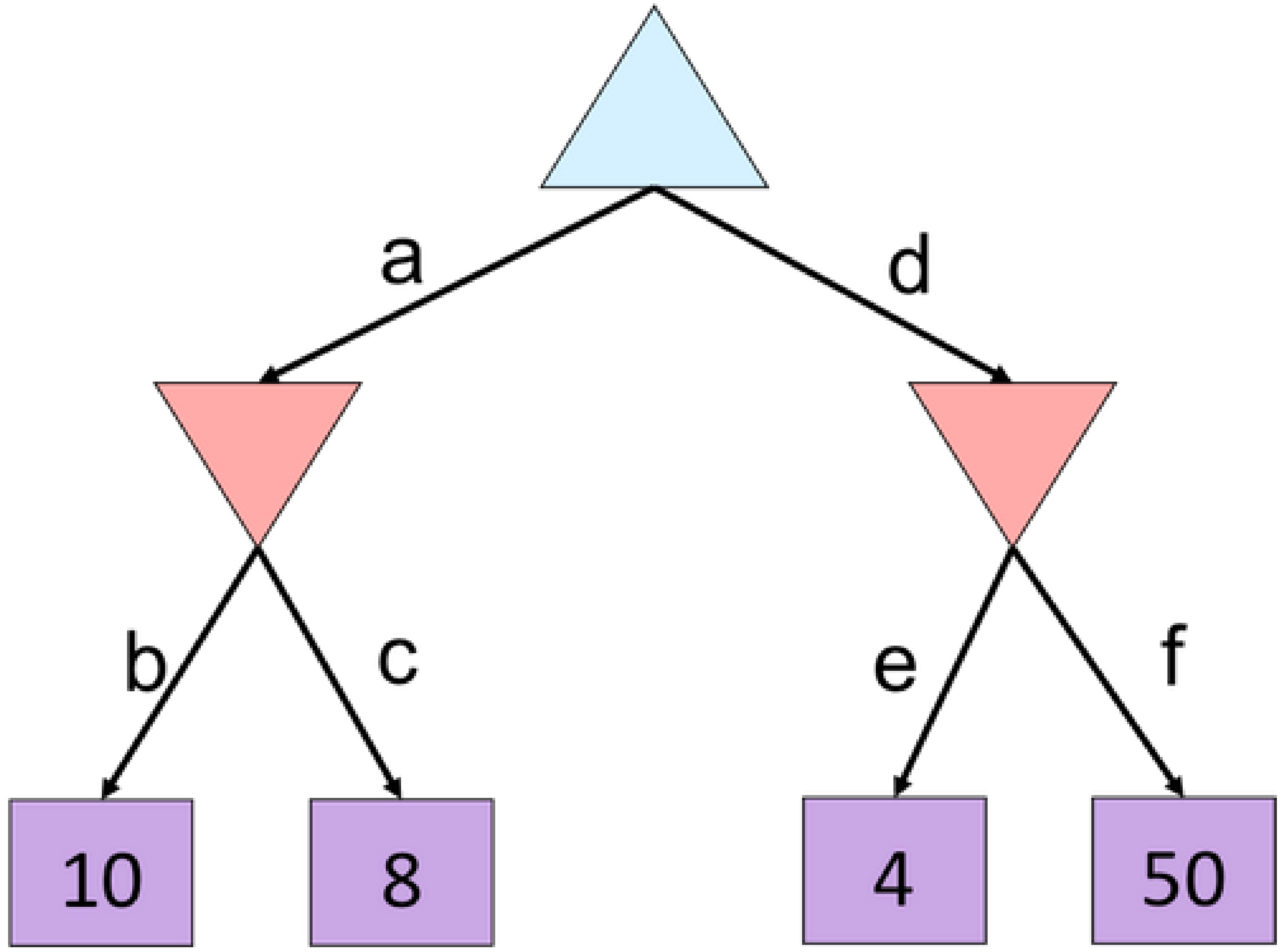
```
def min-value(state , $\alpha$, $\beta$):
    initialize v = $+\infty$
    for each successor of state:
        v = min(v, value(successor, $\alpha$, $\beta$))
        if $v \leq \alpha$ return v
        $\beta$ = min($\beta$, v)
    return v
```

Alpha-Beta Pruning Properties

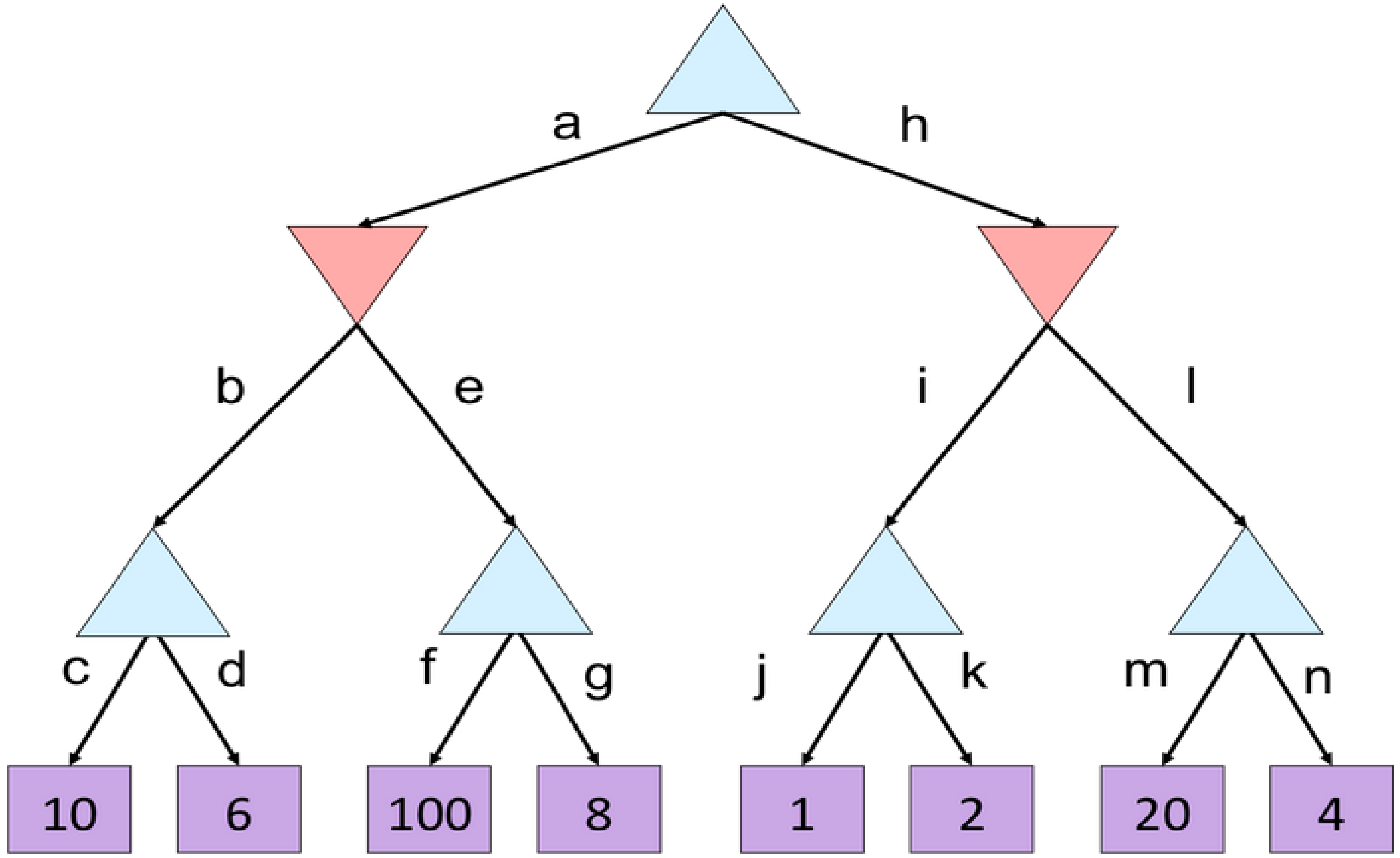
- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - Important: tie-break for action selection to favor the earlier node explored
- Good child ordering improves effectiveness 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…
- This is a simple example of metareasoning (computing about what to compute)



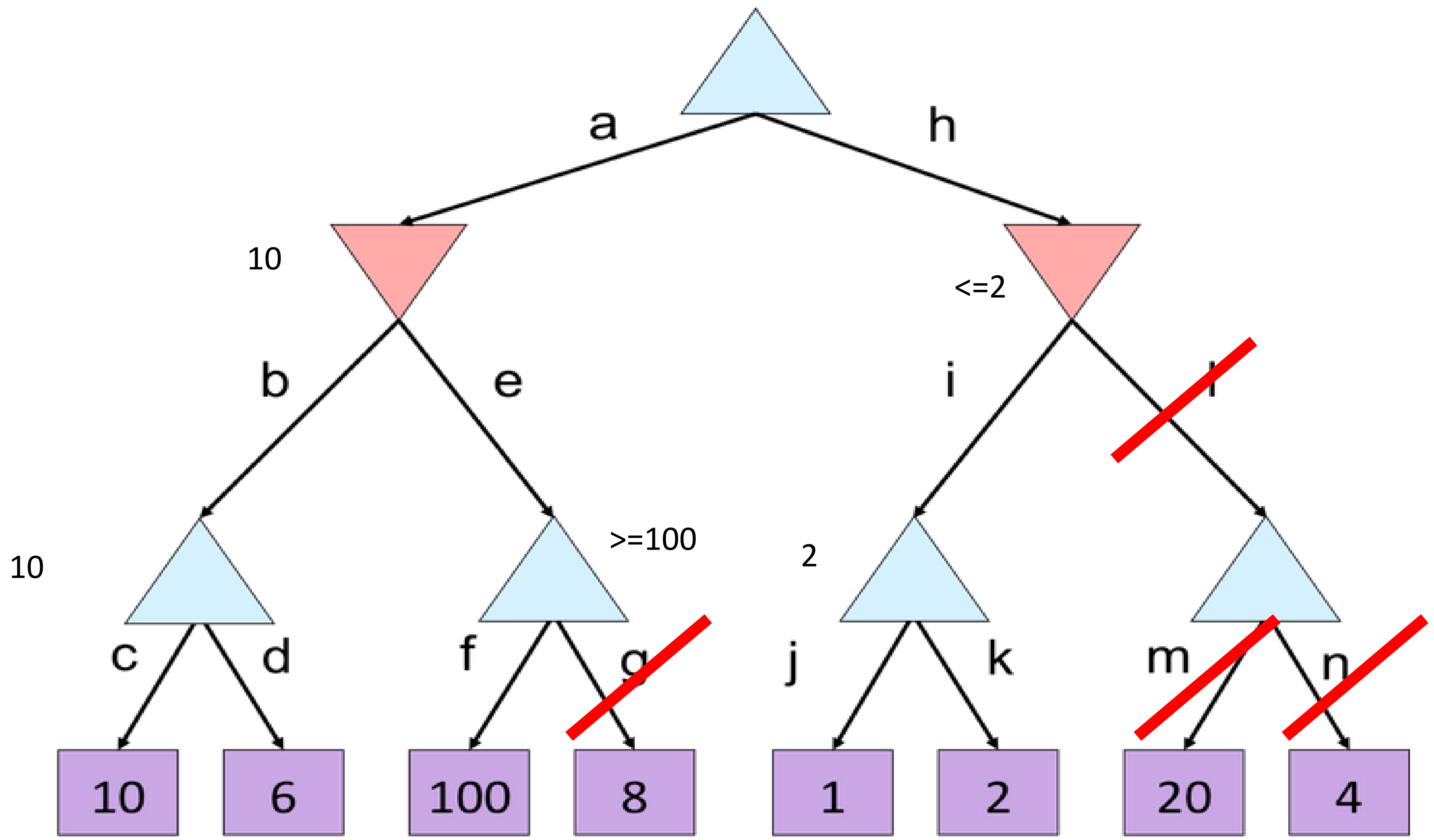
Alpha-Beta Quiz



Alpha-Beta Quiz 2



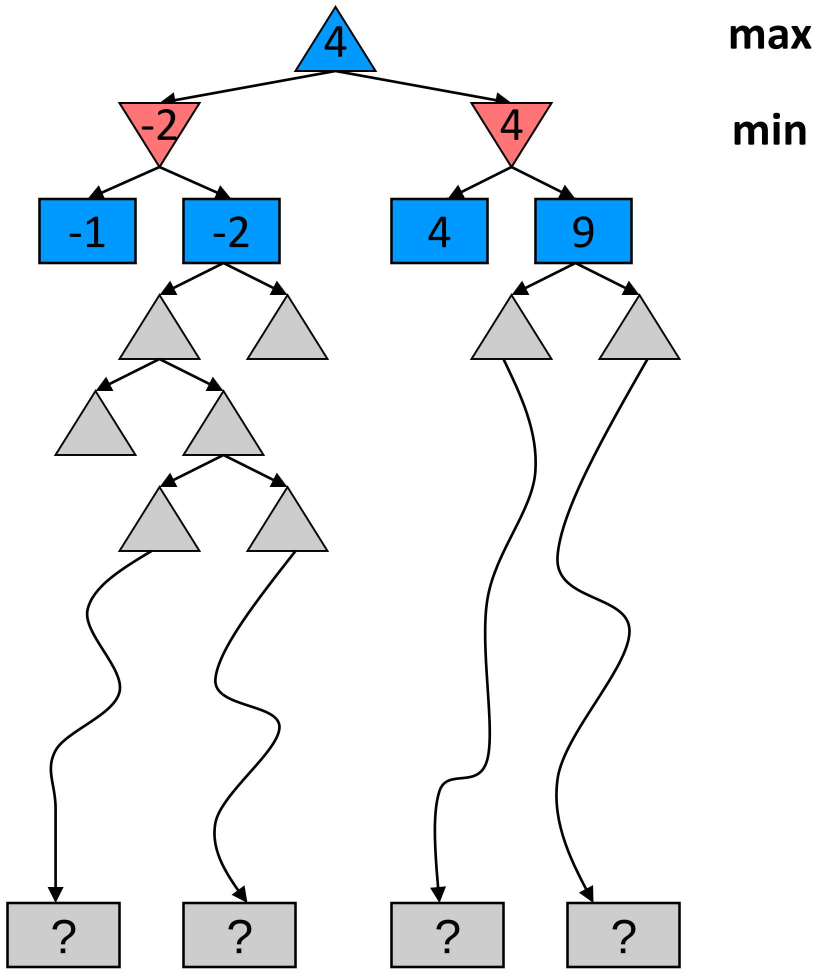
Alpha-Beta Quiz 2



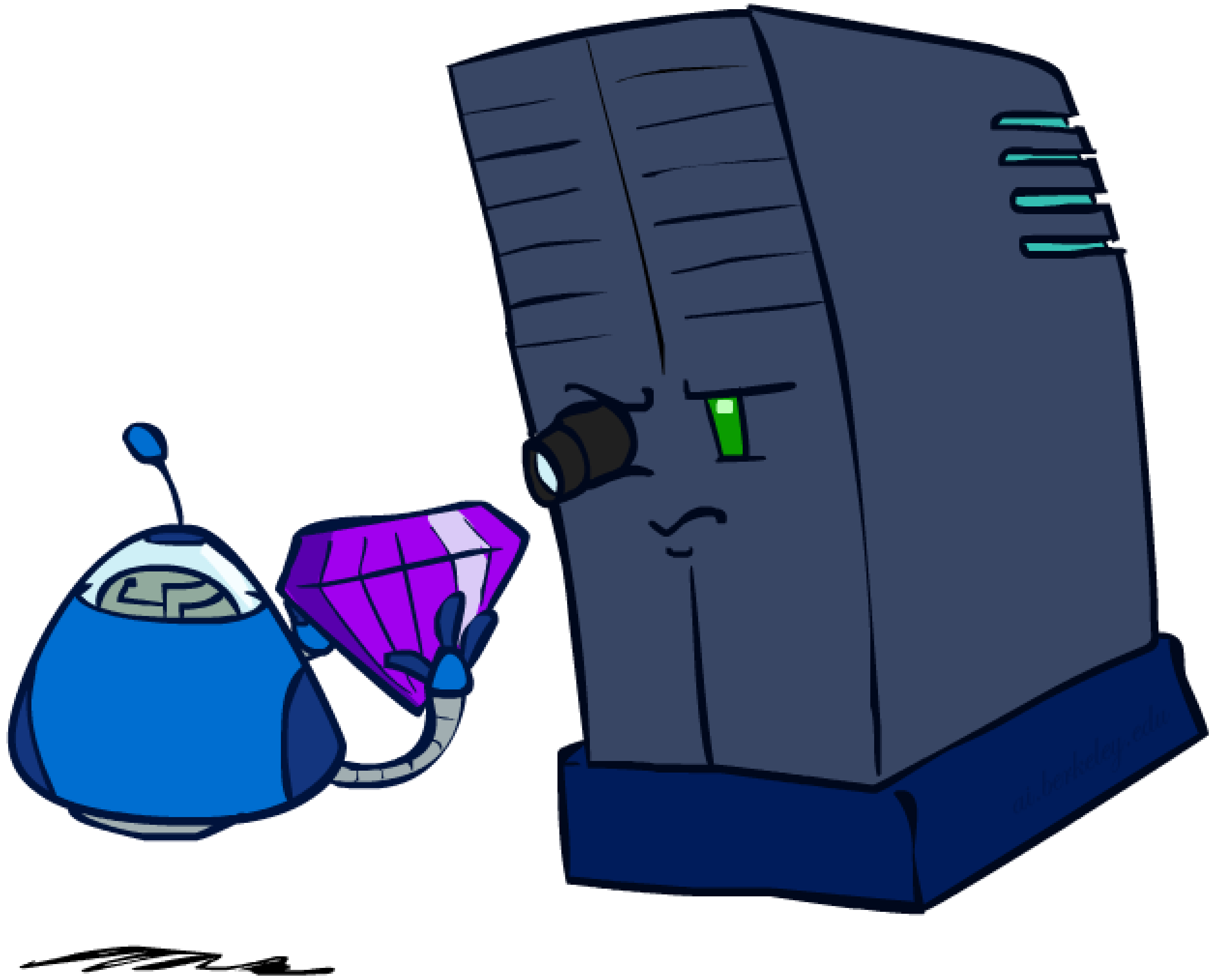
Resource Limits

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

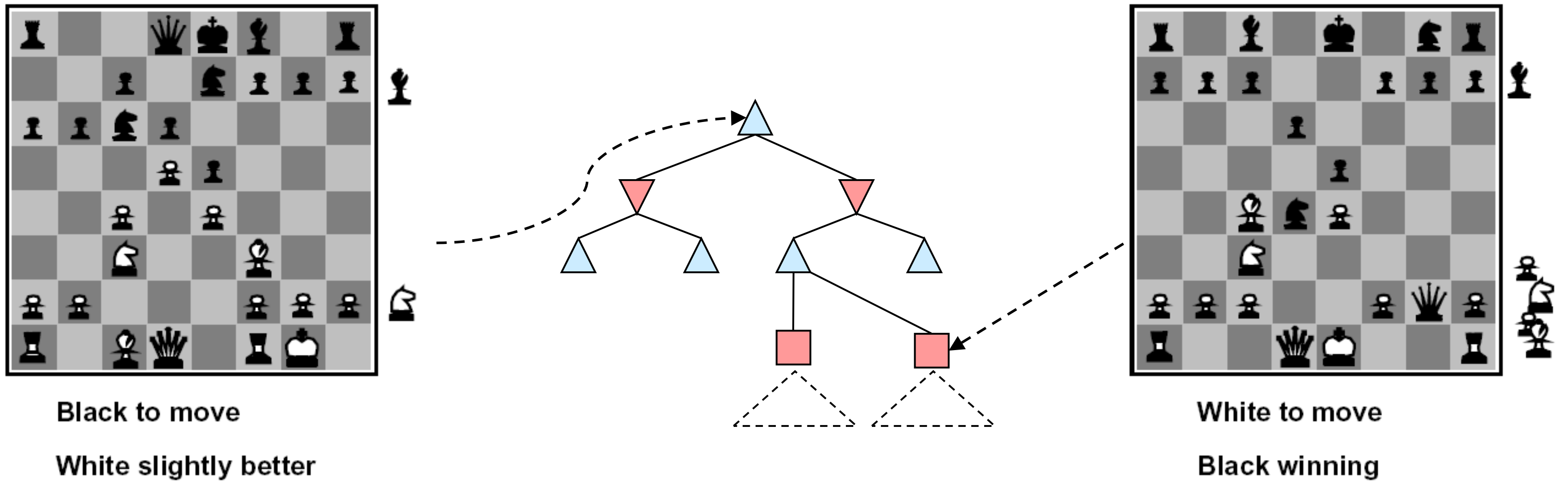


Evaluation Functions



Evaluation Functions

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

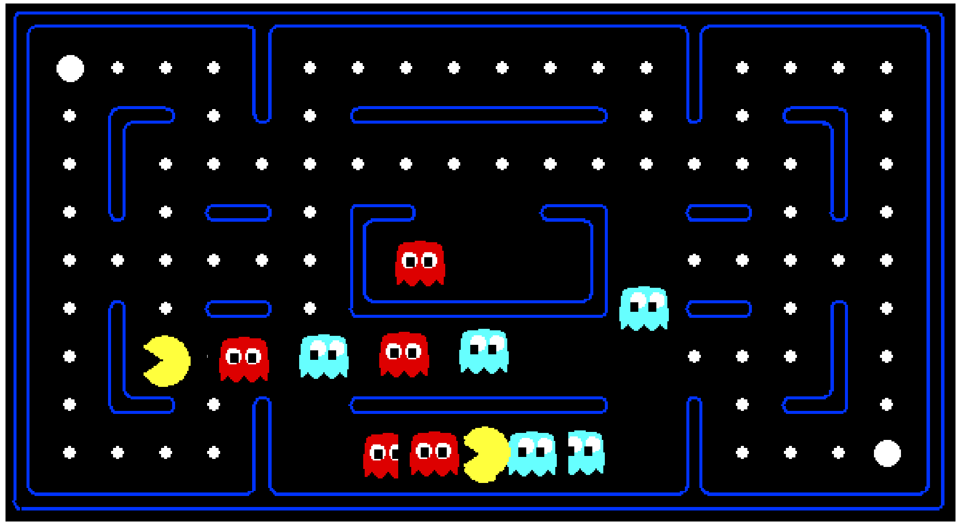


- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

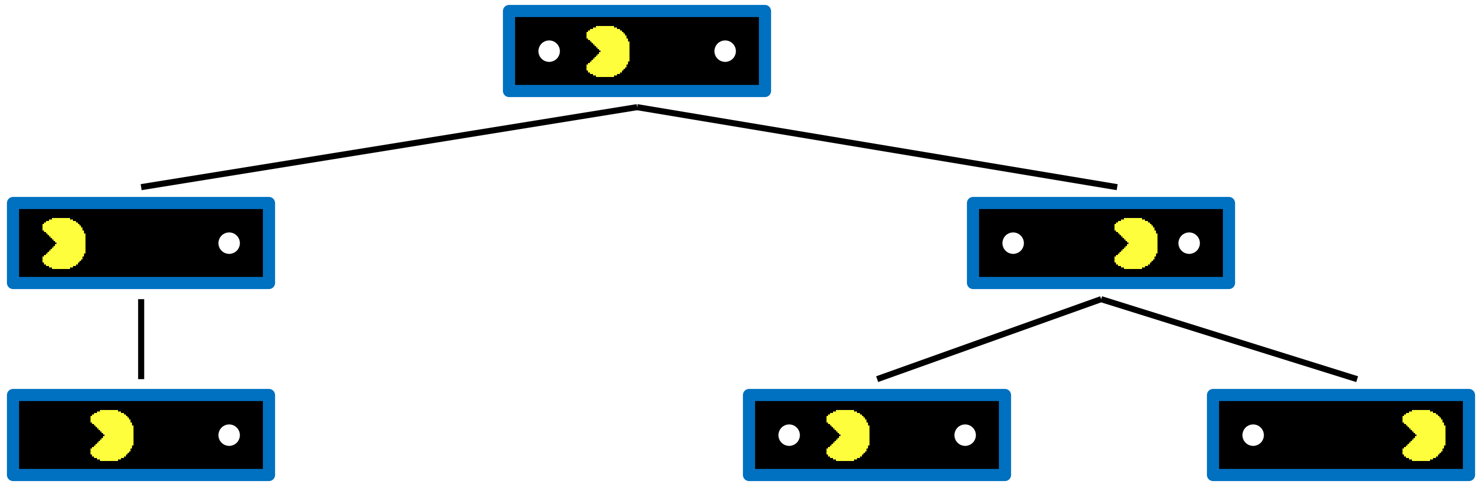
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

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

Evaluation Function Design: Check Against Examples



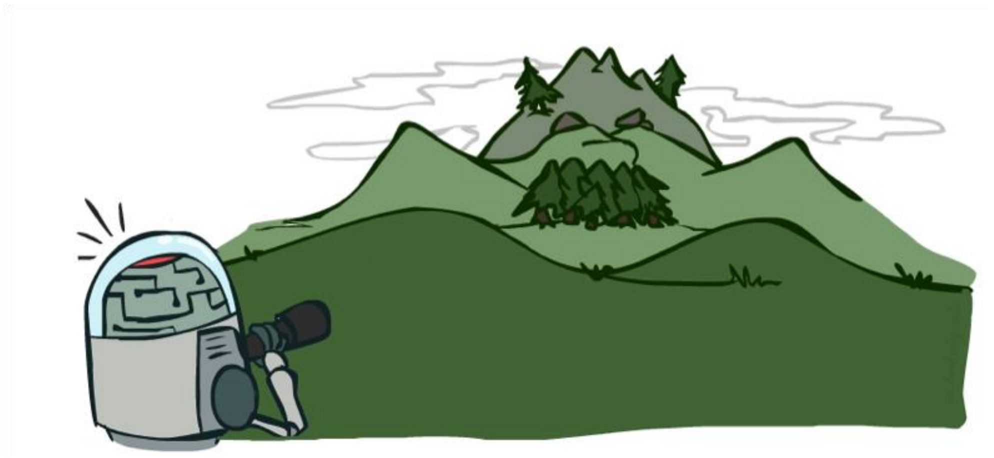
Pitfall: Thrashing with Bad Evaluation Function



- A danger of depth-limited search with not-so-great evaluation functions
  - Pacman knows his score will go up by eating the dot now (west, east)
  - Pacman 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 back west in the next round of replanning!

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



[Demo: depth limited (L6D4, L6D5)]

Video of Demo Limited Depth (2)

Video of Demo Limited Depth (10)

Iterative Deepening

Iterative deepening using Minimax (or AlphaBeta) as subroutine:

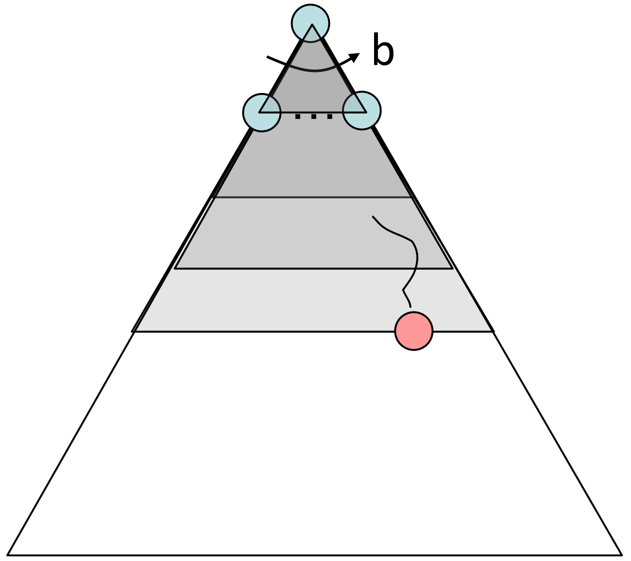
Until run out of time:

1. Do a Minimax up to depth 1, using evaluation function at depth 1
2. Do a Minimax up to depth 2, using evaluation function at depth 2
3. Do a Minimax up to depth 3, using evaluation function at depth 3
4. Do a Minimax up to depth 4, using evaluation function at depth 4

…

When out of time:

Return the result from the deepest search that was fully completed



Synergies between Evaluation Function and Alpha-Beta?

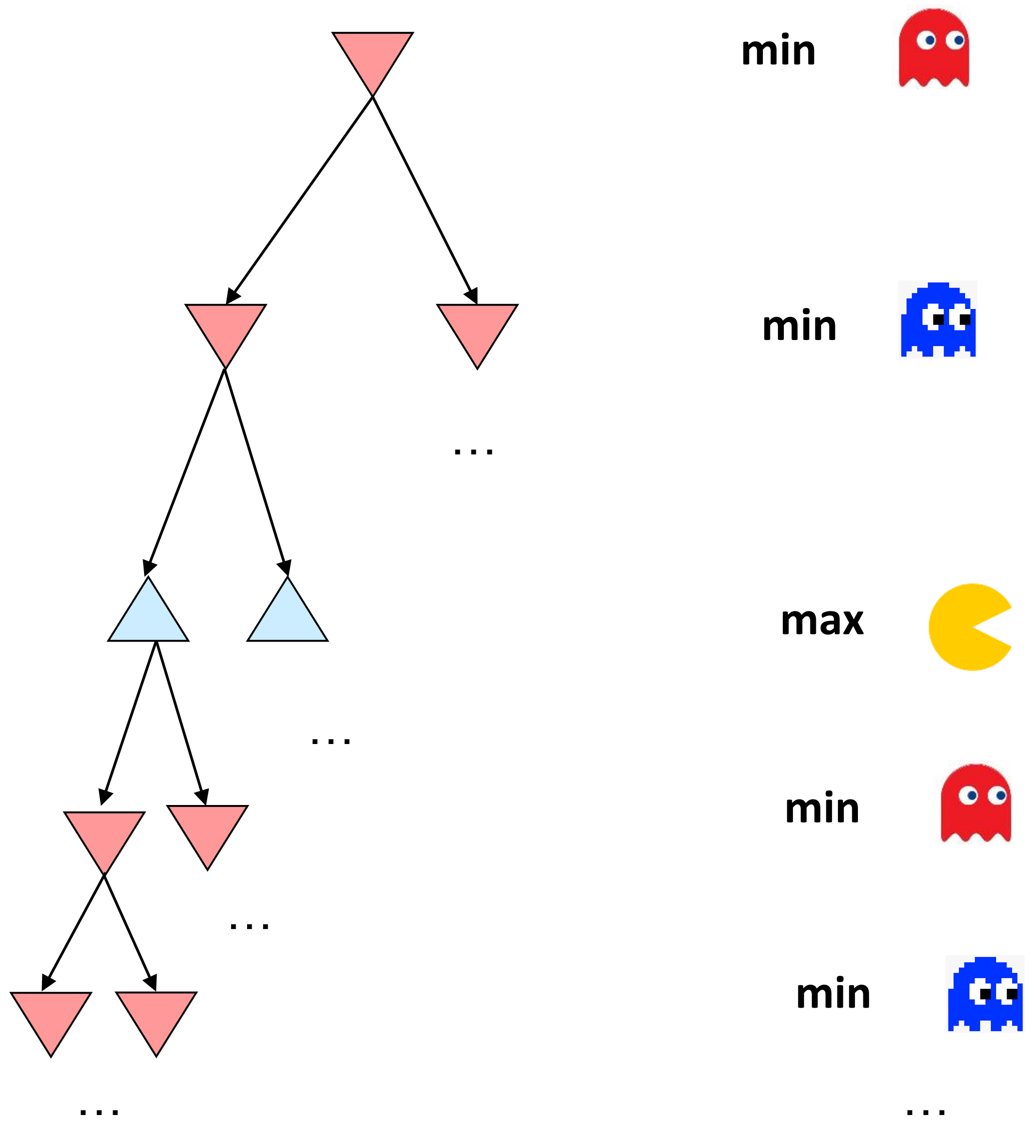
- Alpha-Beta: amount of pruning depends on expansion ordering
  - Evaluation function can provide guidance to expand most promising nodes first (which later makes it more likely there is already a good alternative on the path to the root)
    - (somewhat similar to role of A* heuristic, CSPs filtering)

- Alpha-Beta: (similar for roles of min-max swapped)
  - Value at a min-node will only keep going down
  - Once value of min-node lower than better option for max along path to root, can prune
  - Hence: IF evaluation function provides upper-bound on value at min-node, and upper-bound already lower than better option for max along path to root
    THEN can prune

MiniMiniMax and Emerging Coordination

- Minimax can be extended to more than 2 players
  - e.g. 2 ghosts and 1 pacman

- Result: even though the 2 ghosts independently run their own MiniMiniMax search, they will naturally coordinate because:
  - They optimize the same objective
  - They know they optimize the same objective (i.e. they know the other ghost is also a minimizer)



Video of Demo Smart Ghosts (Coordination)

Video of Demo Smart Ghosts (Coordination) – Zoomed In

Summary

- Games are decision problems with 2 or more agents
  - Huge variety of issues and phenomena depending on details of interactions and payoffs
- For zero-sum games, optimal decisions defined by minimax
  - Implementable as a depth-first traversal of the game tree
  - Time complexity $O(b^m)$, space complexity $O(bm)$
- Alpha-beta pruning
  - Preserves optimal choice at the root
  - alpha/beta values keep track of best obtainable values from any max/min nodes on path from root to current node
  - Time complexity drops to $O(b^{m/2})$ with ideal node ordering
- Exact solution is impossible even for "small" games like chess
  - Evaluation function
  - Iterative deepening (i.e. go as deep as time allows)
- Emergence of coordination:
  - For 3 or more agents (all MIN or MAX agents), coordination will naturally emerge from each independently optimizing their actions through search, as long as they know for each other agent whether they are MIN or MAX

Next Time: Uncertainty!