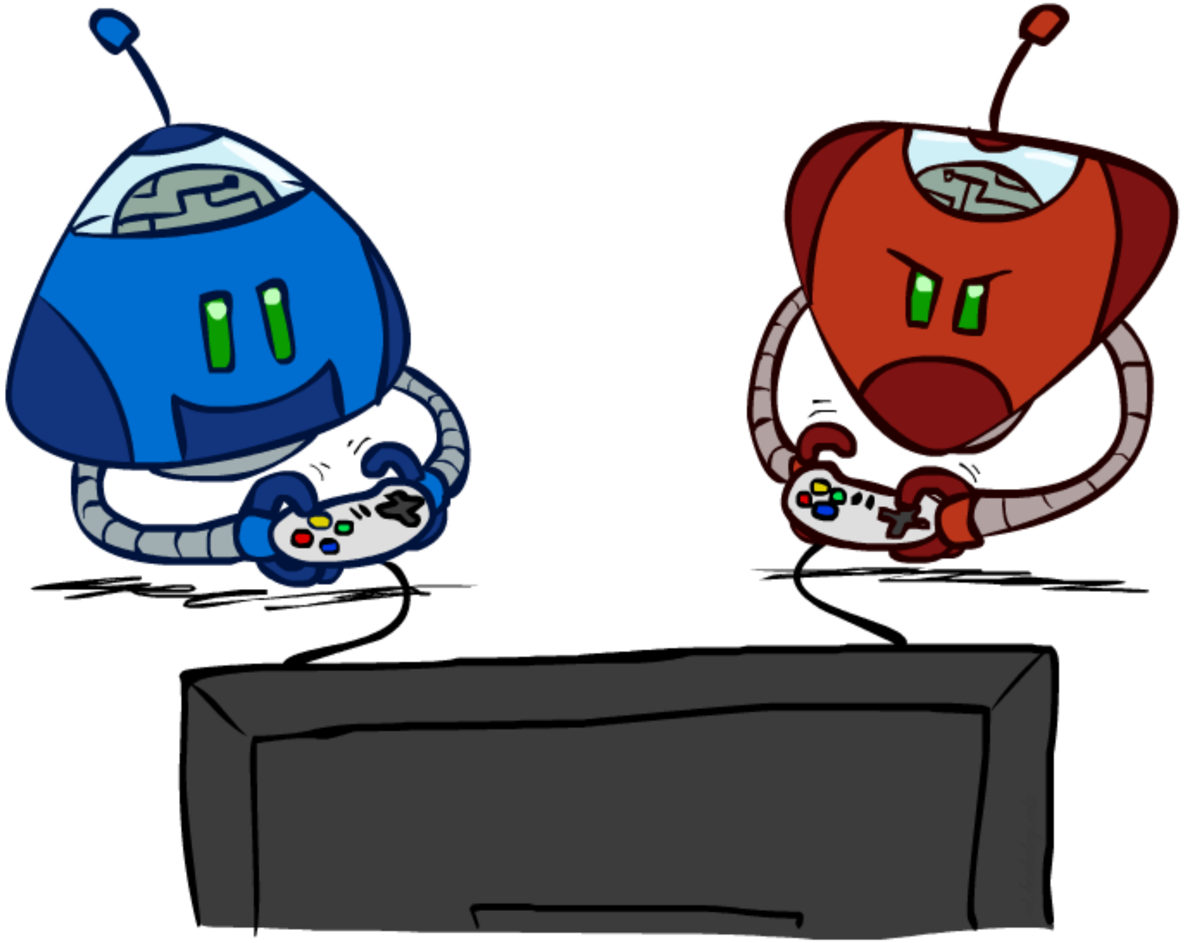
Announcements

- **Project 1** due last Friday (Sept 8)
- **Homework 1** due **today (Sept 12)** at 11:59pm PT
- **Project 2** released, due **Friday (Sept 22)** at 11:59pm PT

CS 188: Artificial Intelligence

Search with Other Agents I

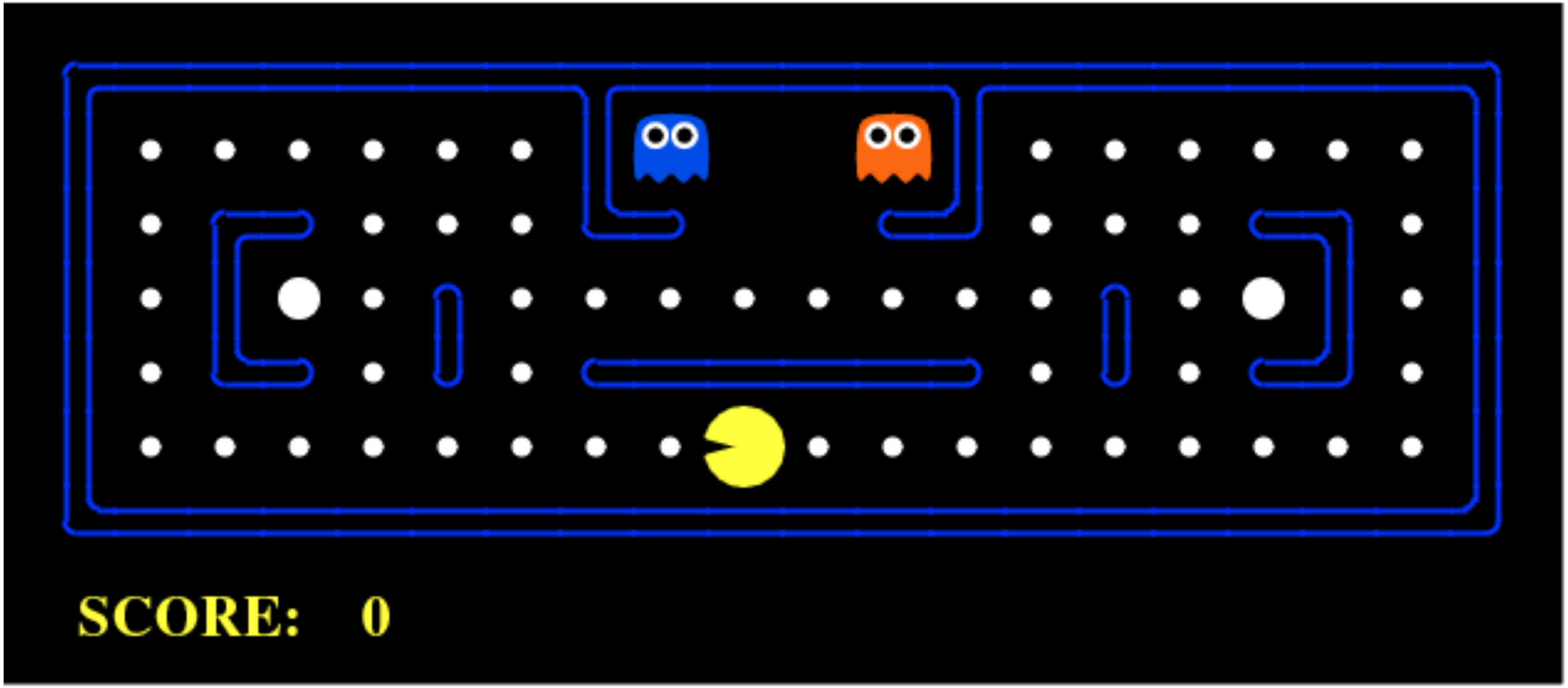


[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).]

Why Multiple Agents?

- AI agents that work with humans (or other AI agents)
  - Robot helpers, AI tutors, self-driving cars
- Multiple agents compete against each other to improve
  - Examples: AlphaGo
- Play multi-agent Pacman! (and other games)

Multi-Agent Pacman



We'll focus on games in this class, but multi-agent ideas come up in many areas of AI

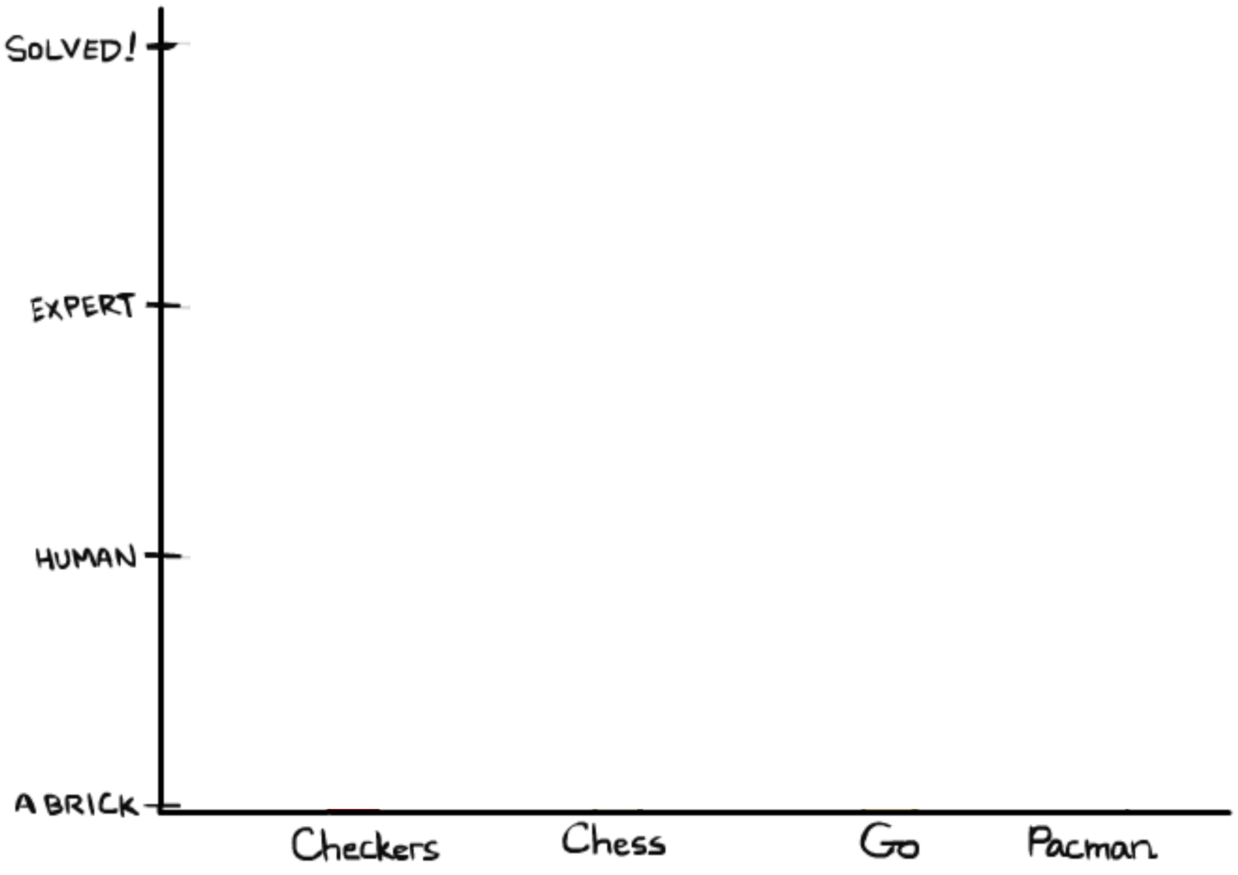
[Demo: mystery pacman (L6D1)]

Game Playing Progress

- **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:** AlphaGo defeats human in 2016. Uses Monte Carlo Tree Search and learned evaluation function.



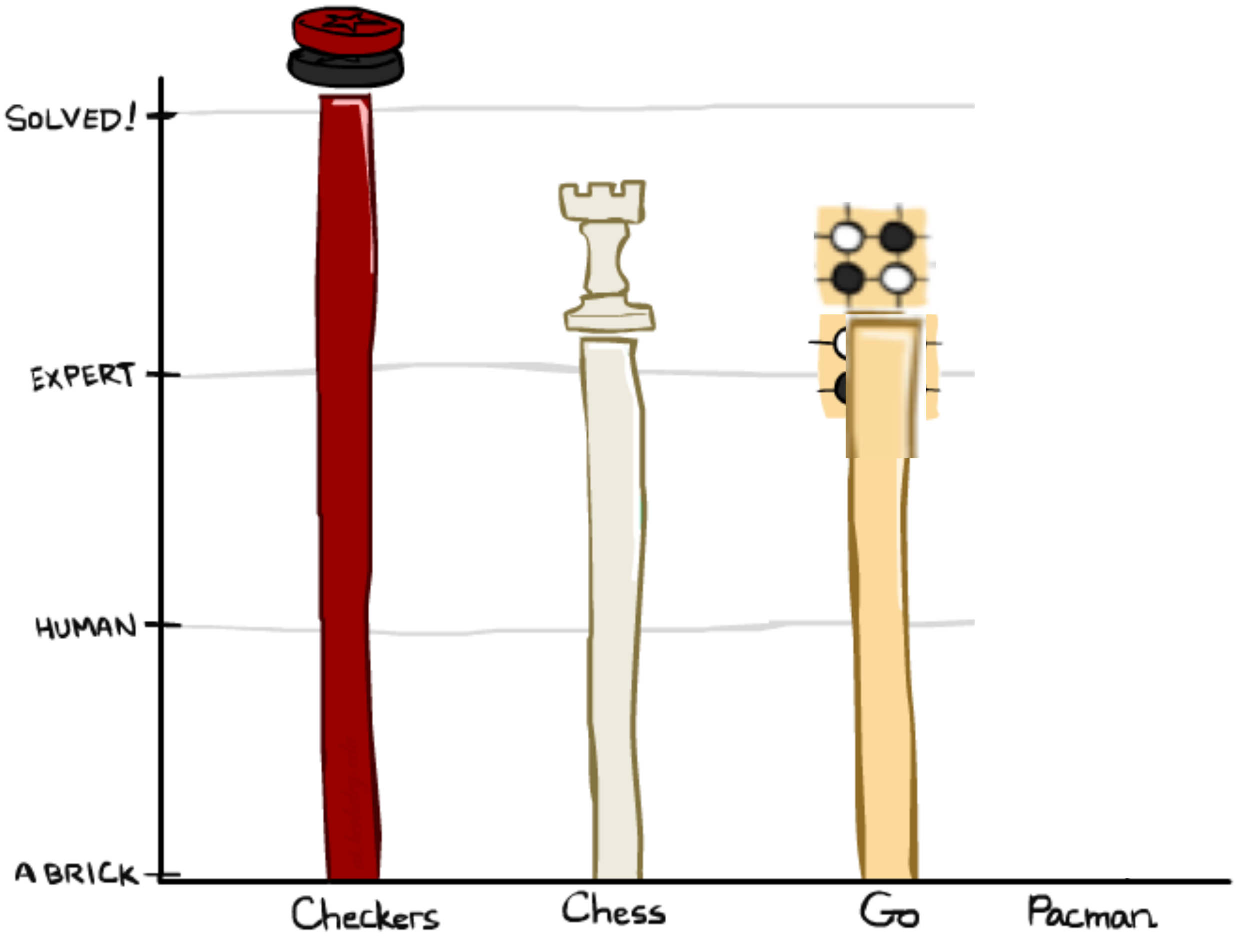
Game Playing Progress

- **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.

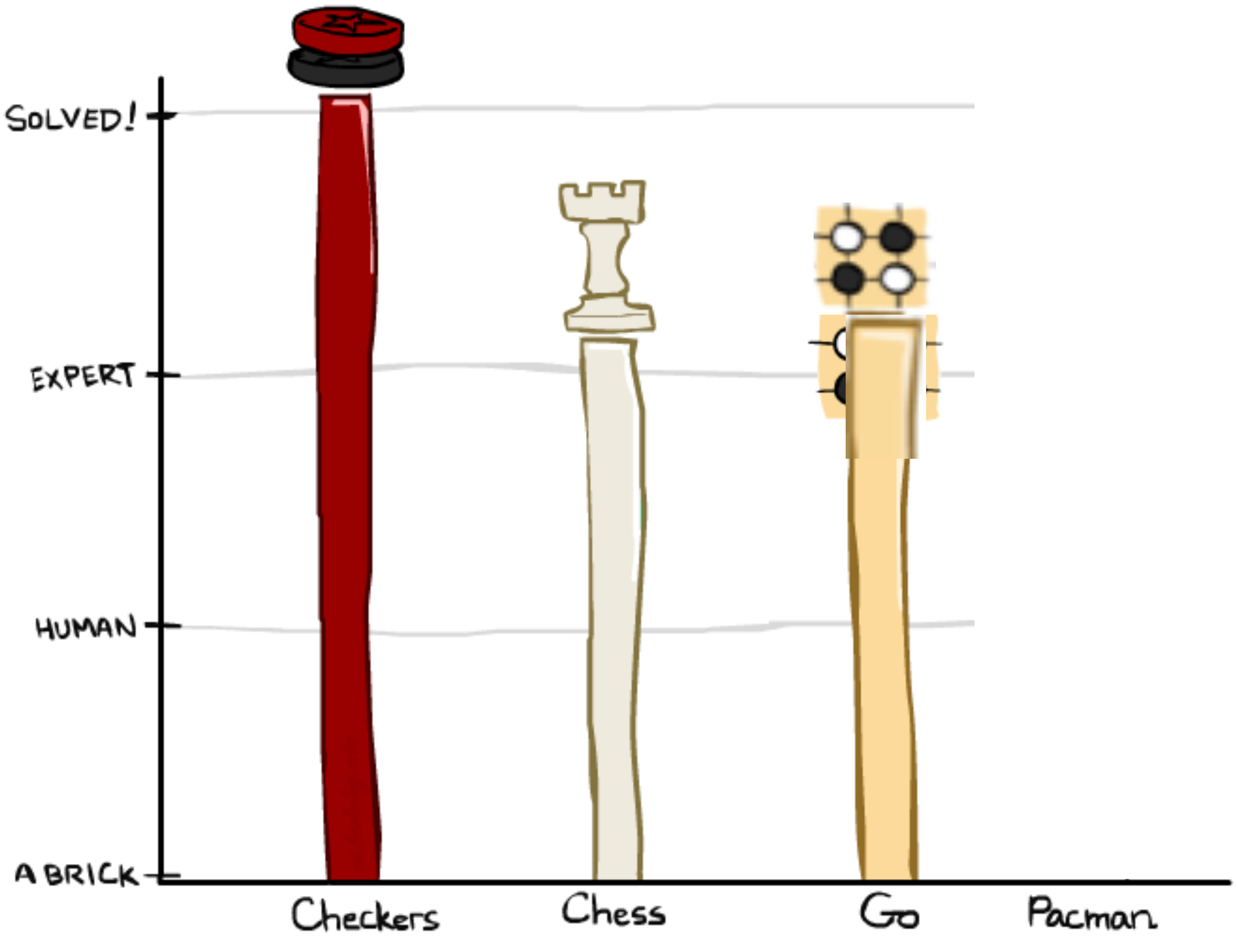
- **Go:** AlphaGo defeats human in 2016. Uses Monte Carlo Tree Search and learned evaluation function.

- **Pacman**



Game Playing Progress

- **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:** AlphaGo defeats human in 2016. Uses Monte Carlo Tree Search and learned evaluation function.
- **Pacman**
- **Why play games?** Helps track progress in AI



Types of Games

- Many different kinds of games!

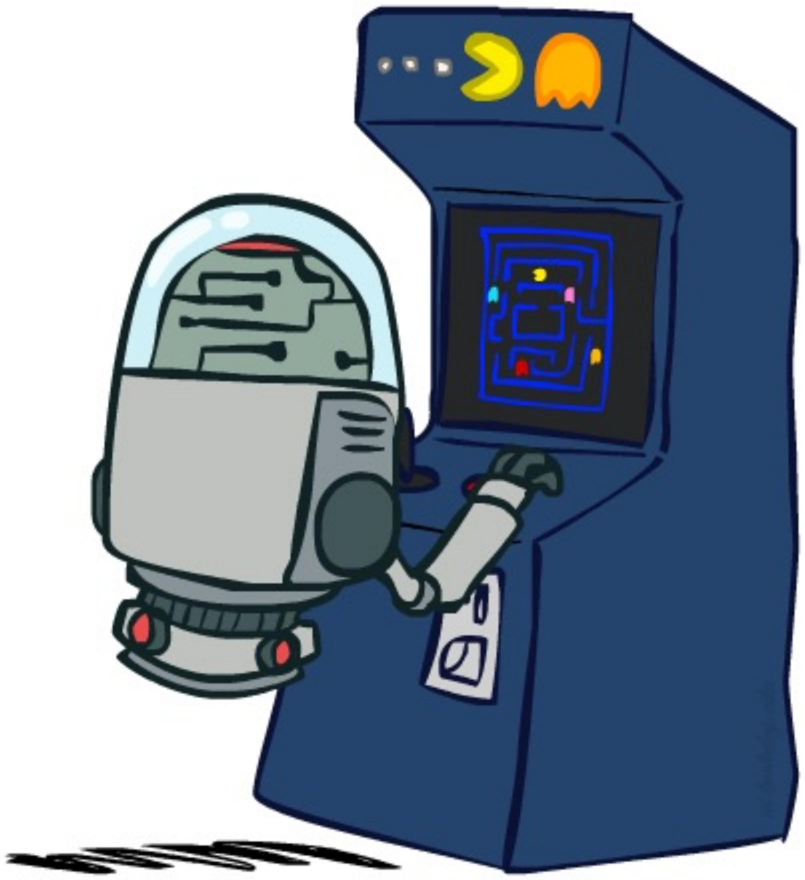
- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?

- Want algorithms for calculating a strategy (policy) which recommends a move from each state

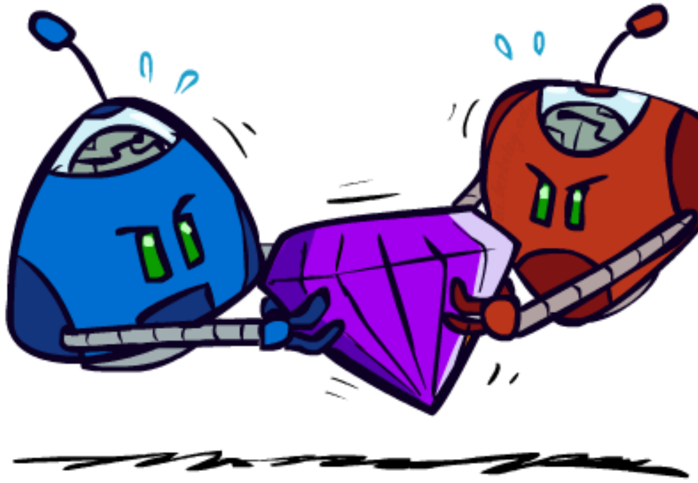
Deterministic Games

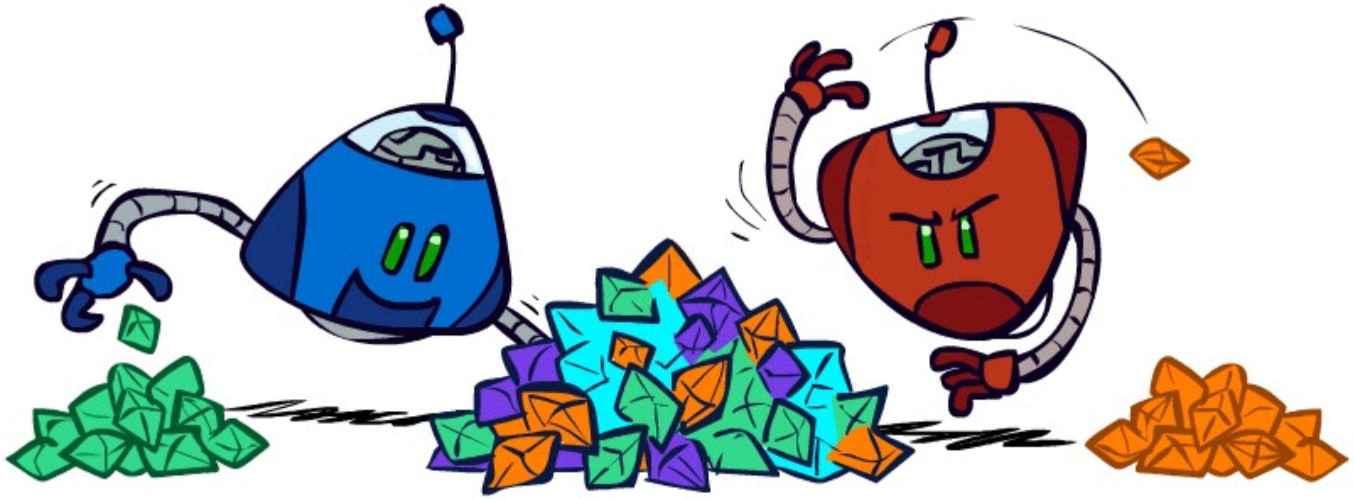
- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: SxA $\rightarrow$ S
  - Terminal Test: S $\rightarrow$ {t,f}
  - Terminal Utilities: SxP $\rightarrow$ R

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



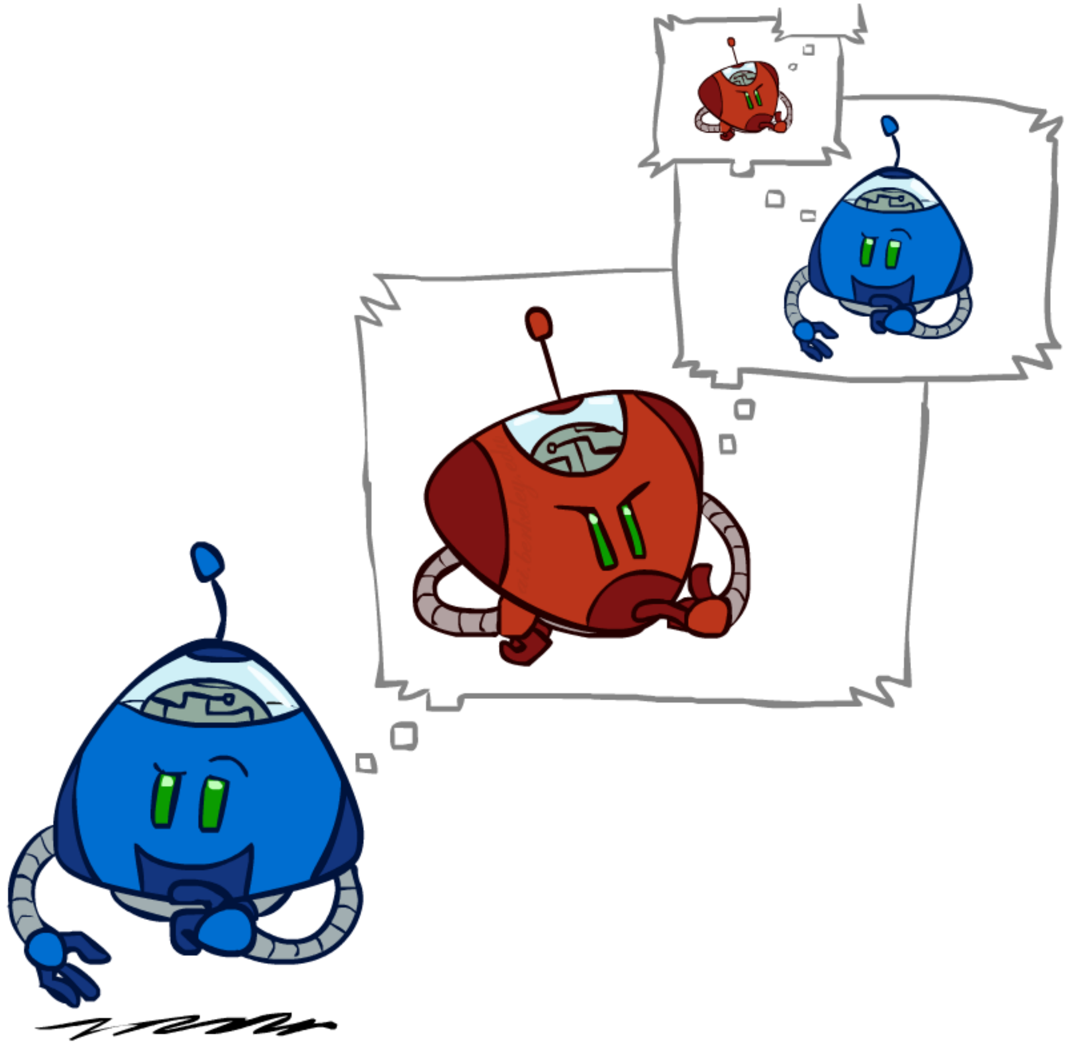
Zero-Sum Games



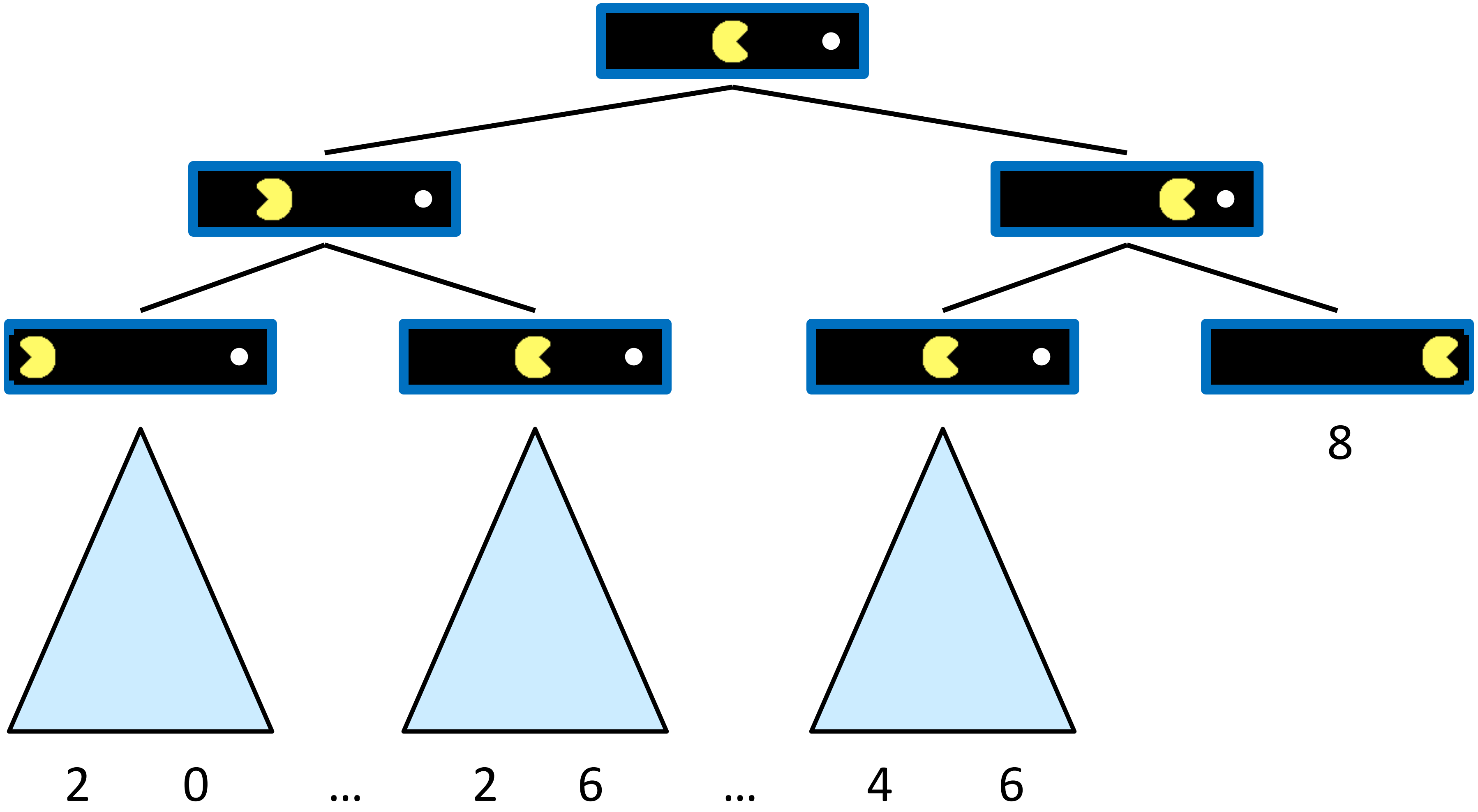


- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition,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

Solving Zero-Sum Games



Single-Agent Search Trees

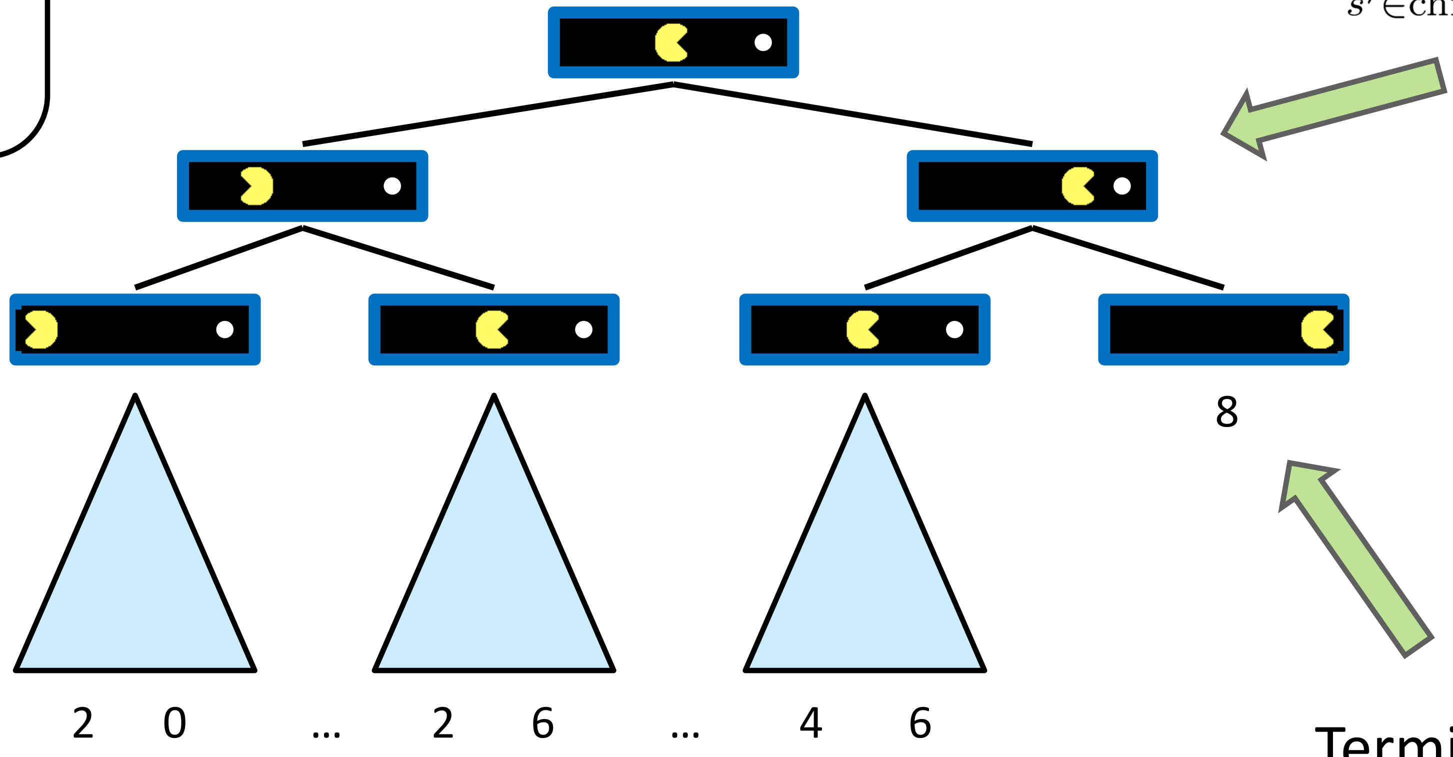


Value of a State

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

Non-Terminal States:

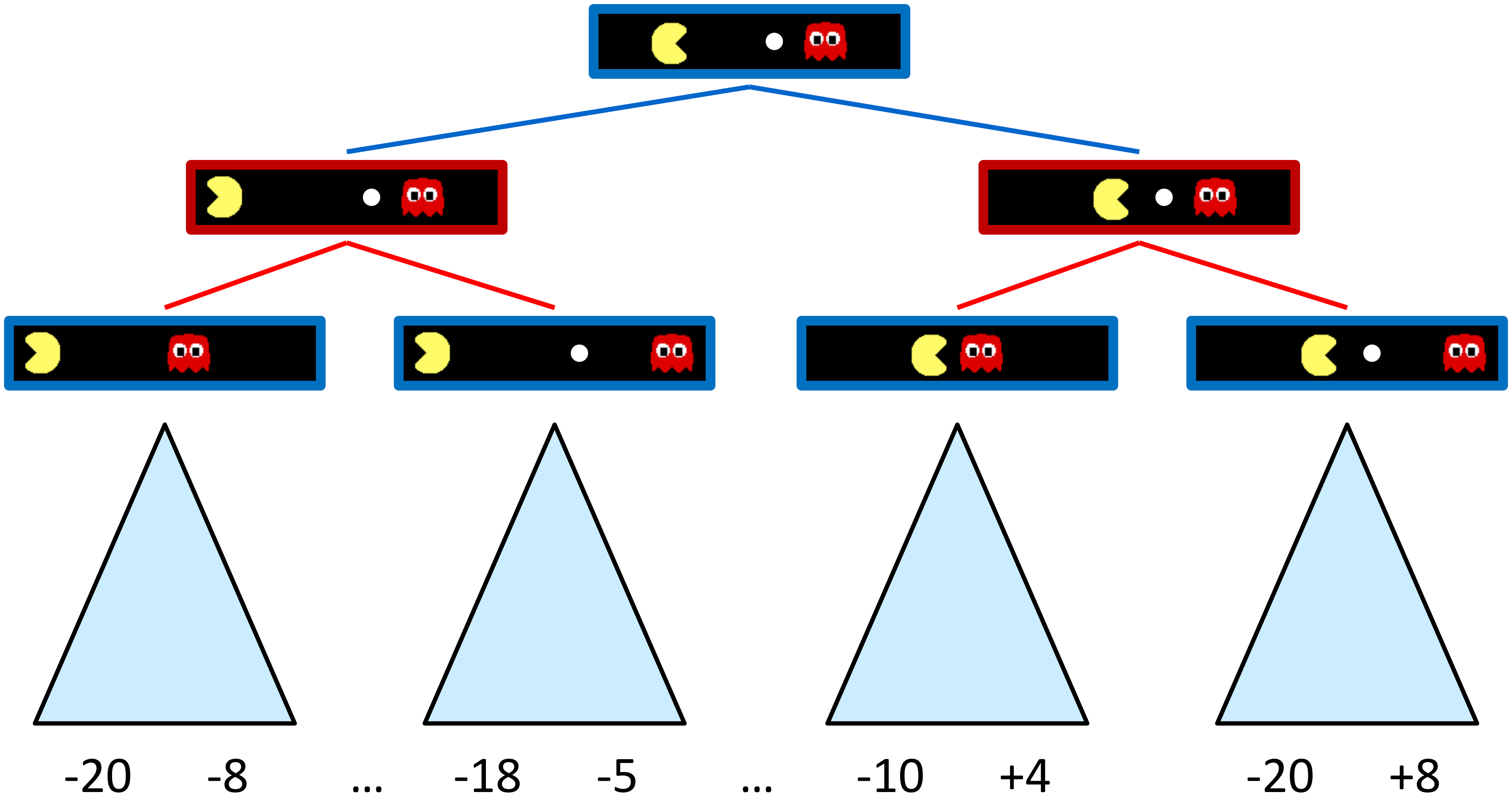
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



Terminal States:

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

Adversarial Game Trees



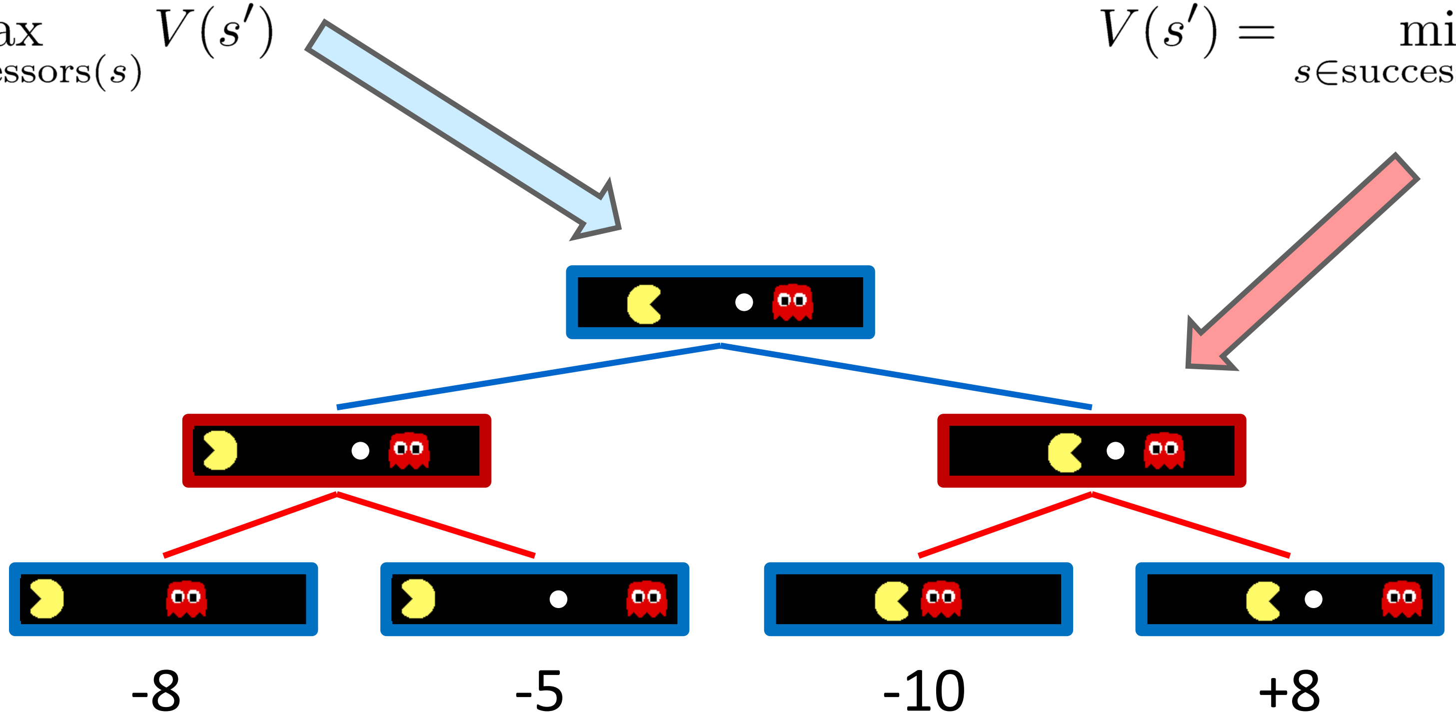
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

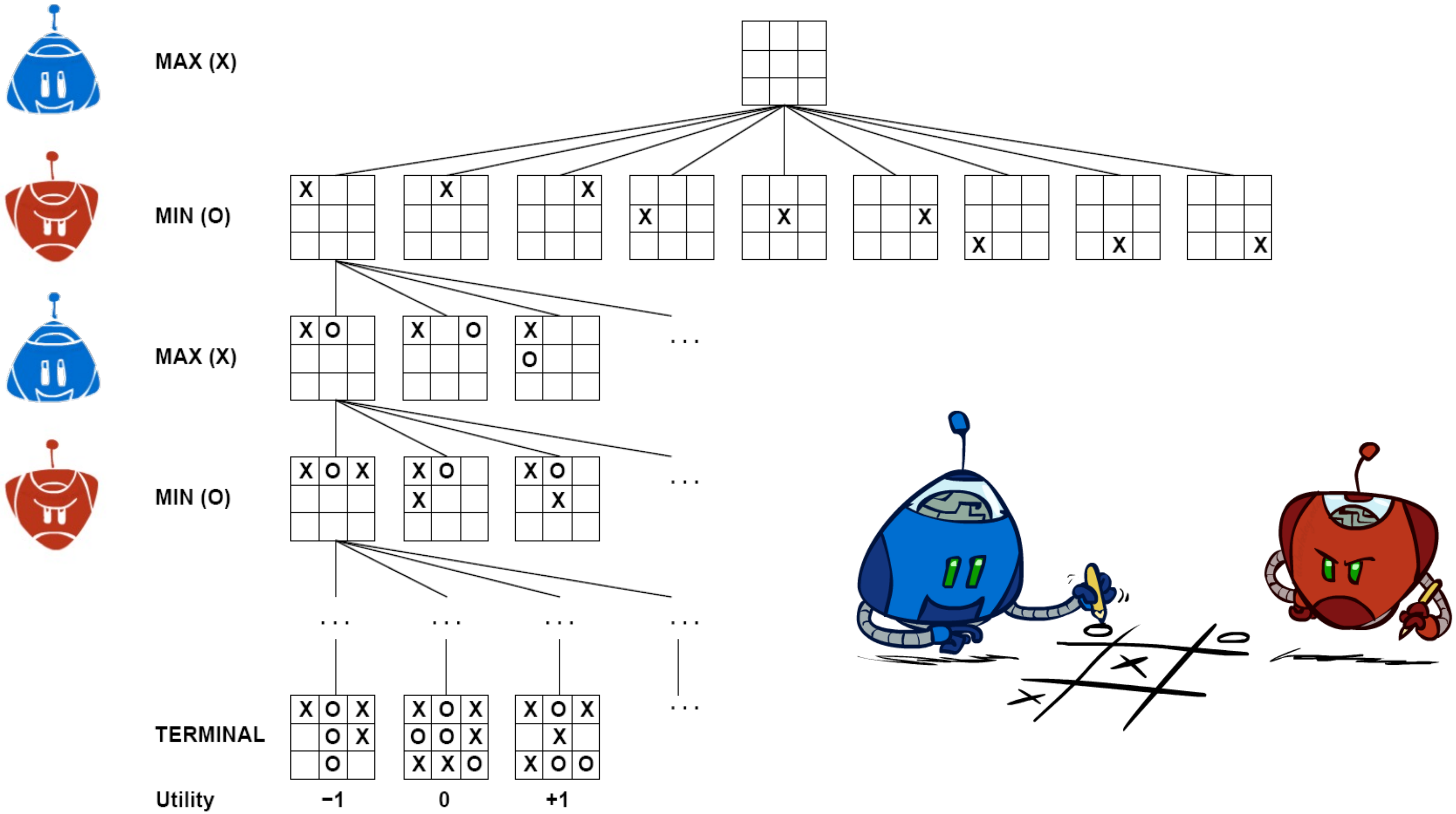
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

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

Tic-Tac-Toe Game Tree

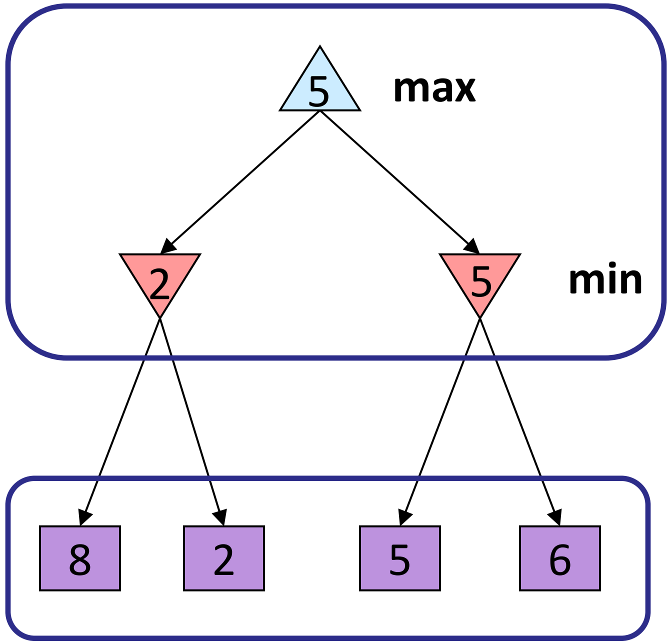


Adversarial Search (Minimax)

- 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 values: computed recursively



Terminal values: part of the game

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
```

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

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

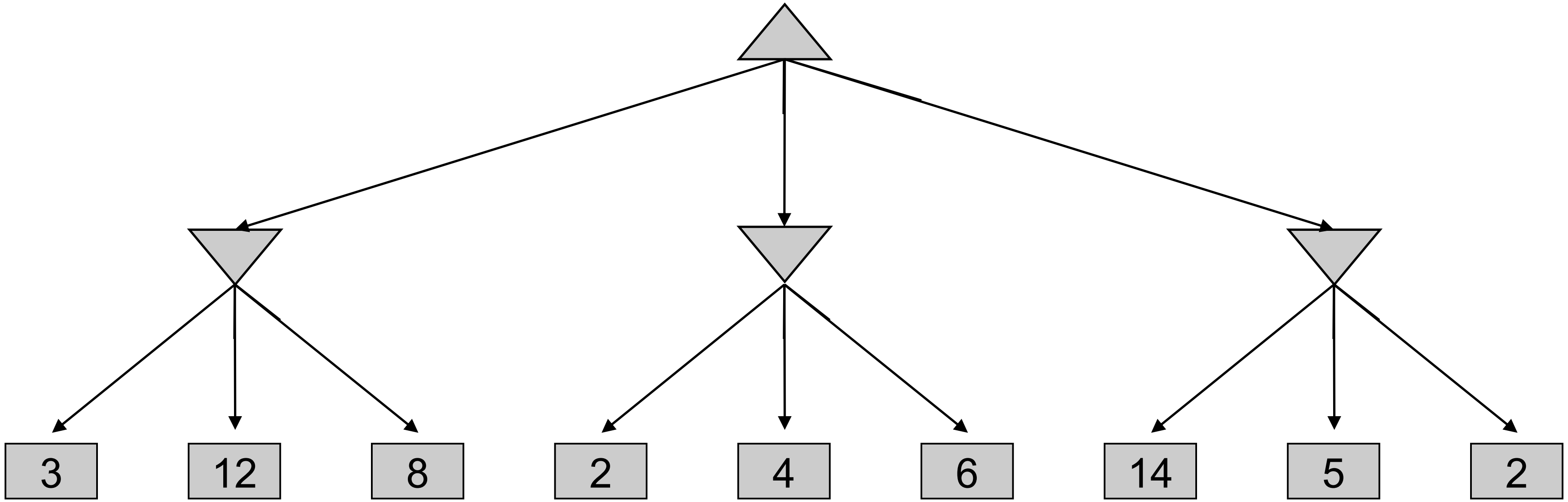
Minimax Implementation

```
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)
```

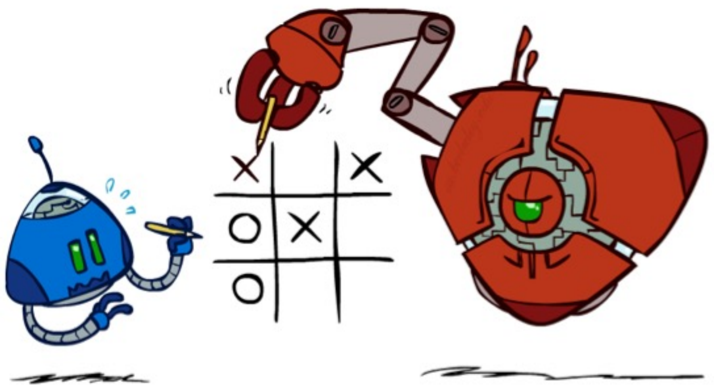
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

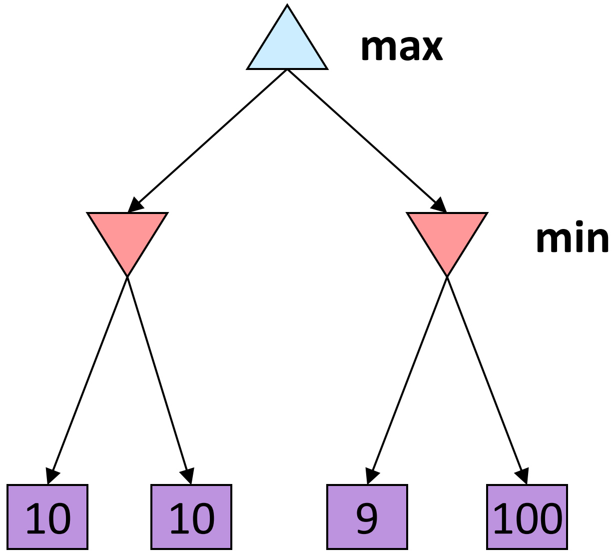
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

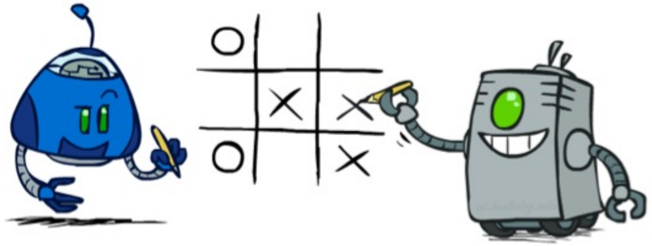
Minimax Example



Minimax Properties







Optimal against a perfect player. Otherwise?

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

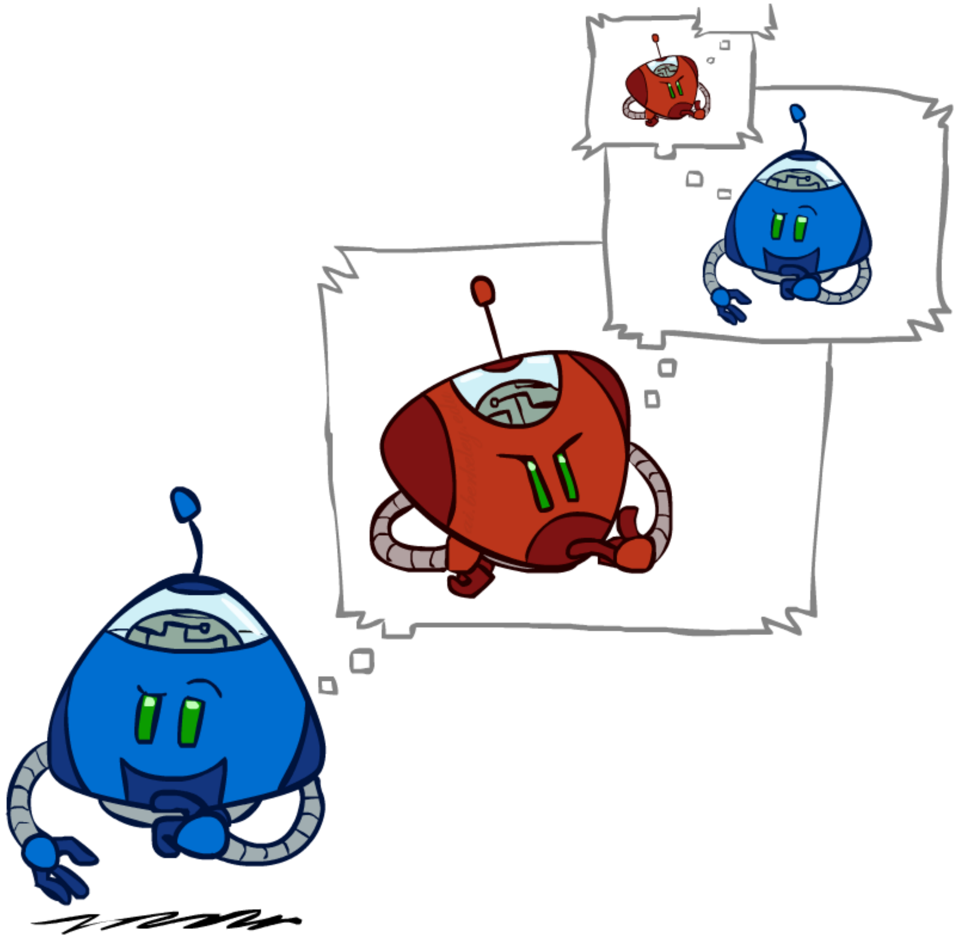
Video of Demo Min vs. Exp (Min)

Video of Demo Min vs. Exp (Exp)

Minimax Efficiency

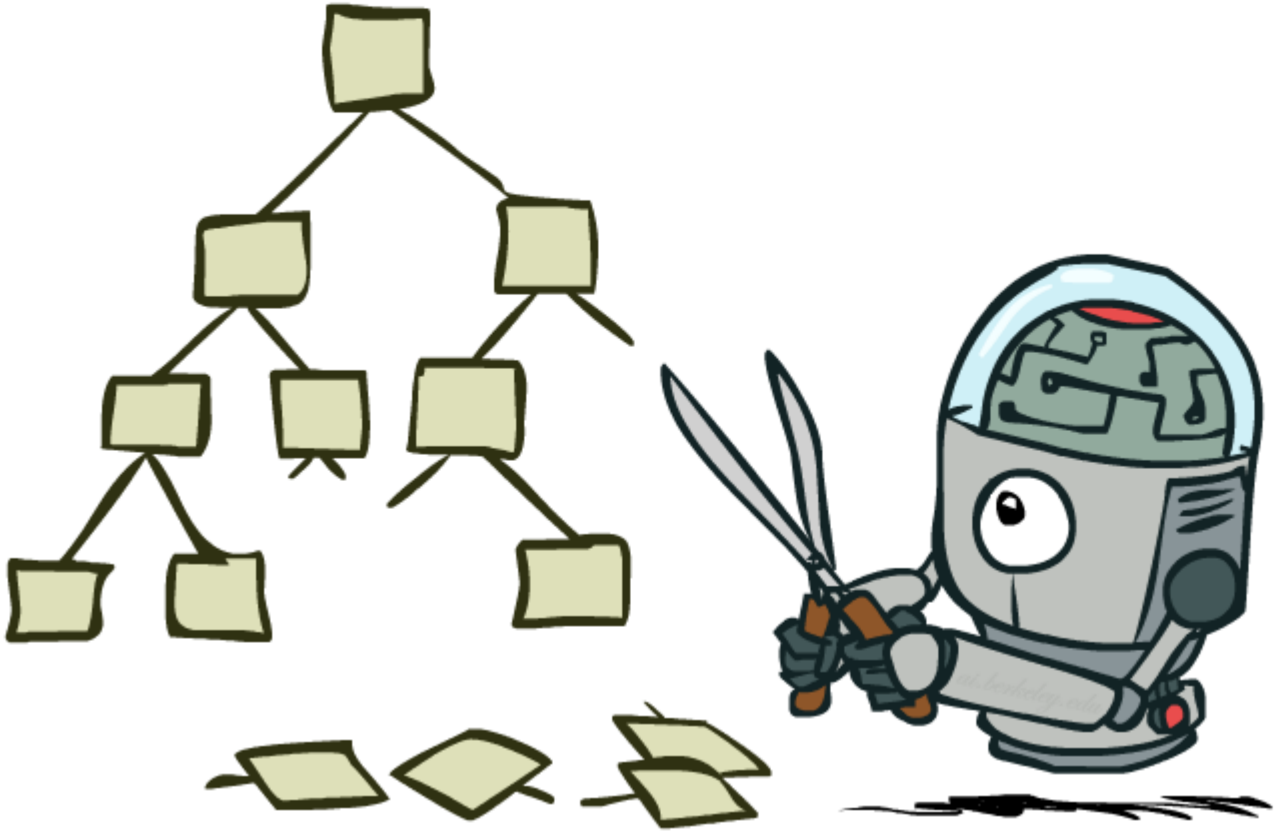
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

- Example: For chess, $b \approx 35$, $m \approx 100$
  For Go, $b \approx 250\text{-}300$, $m \approx 150$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?

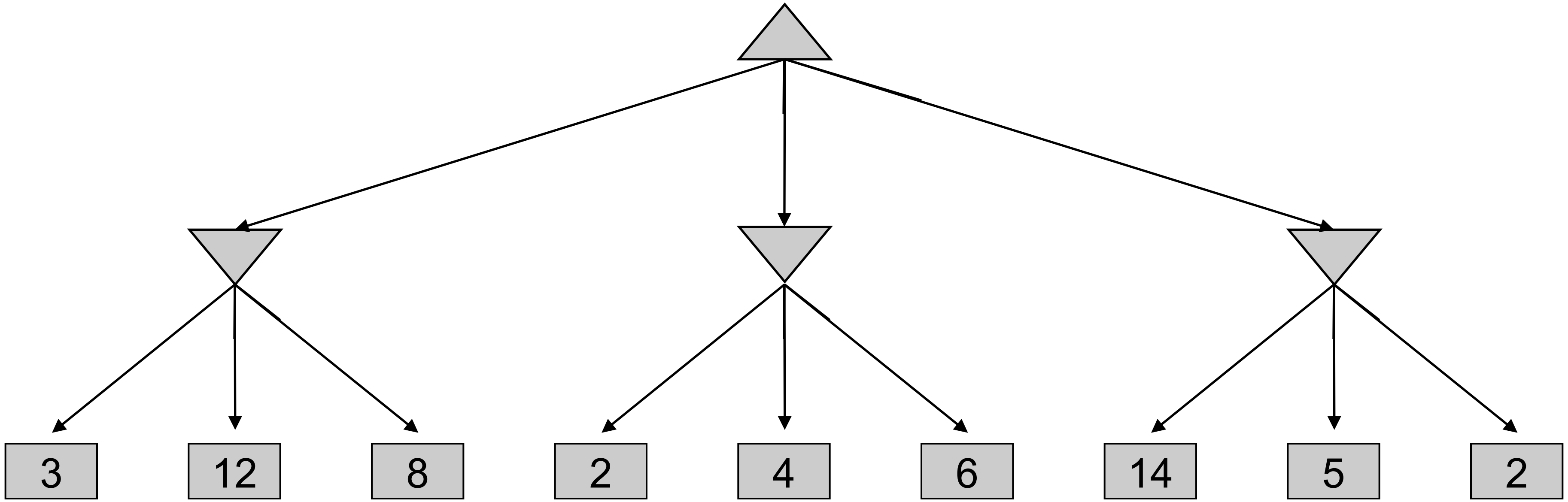


Overcoming Resource Limits

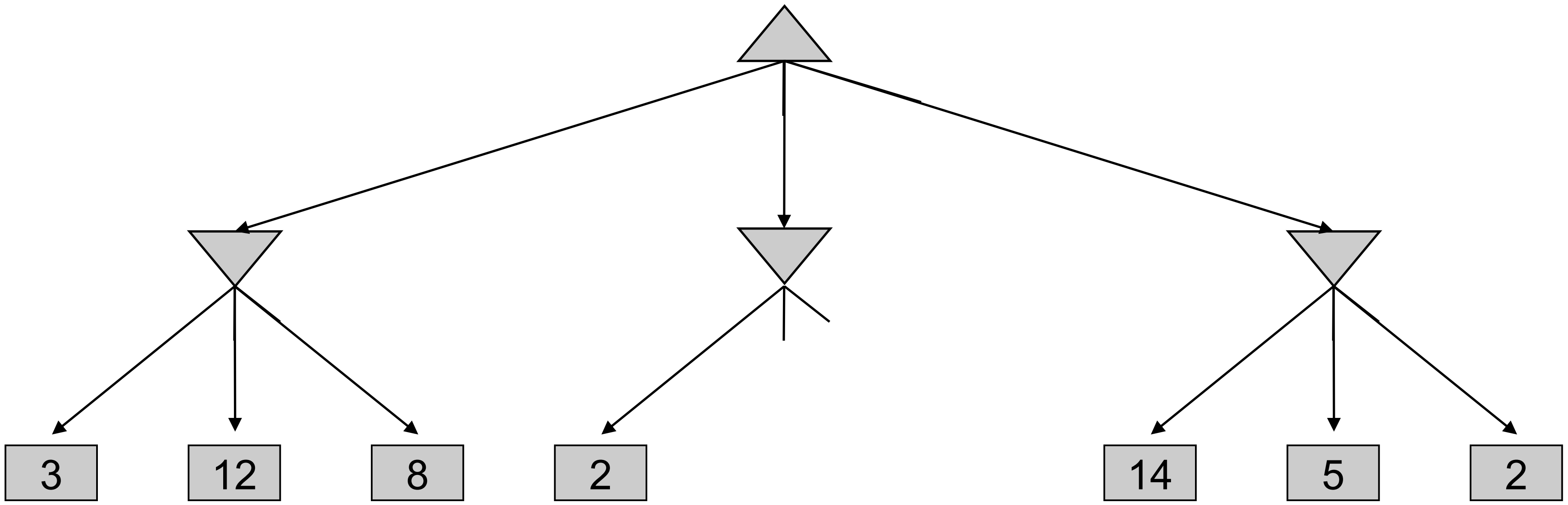
Game Tree Pruning



Minimax Example

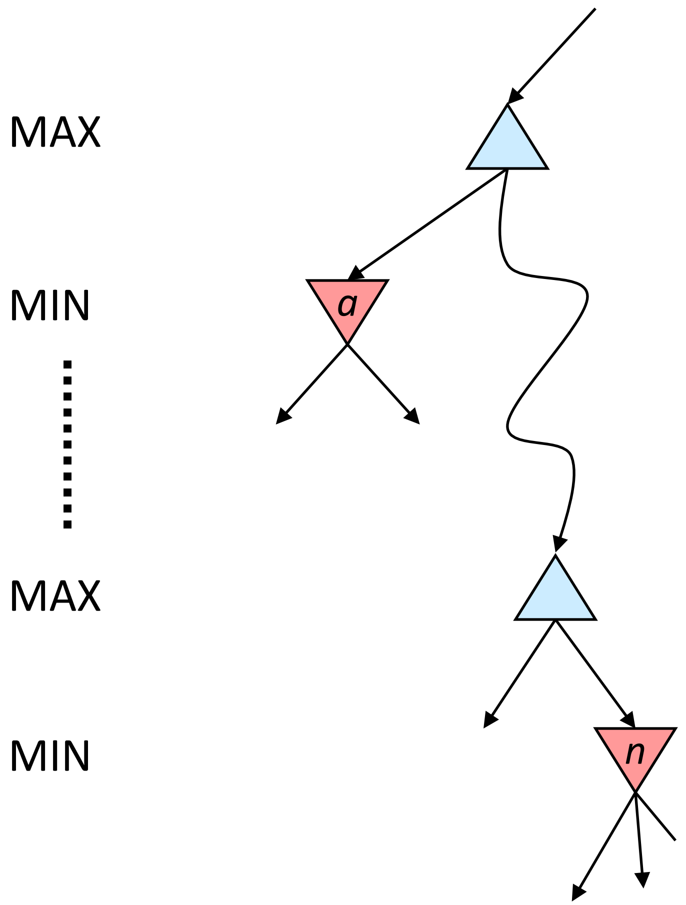


Minimax Pruning



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

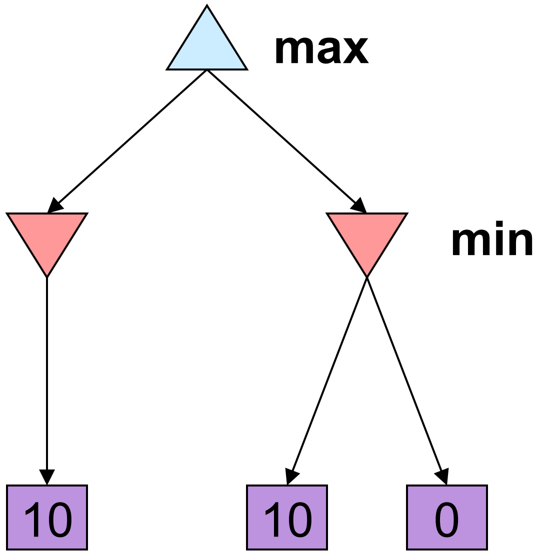
α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, 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
```

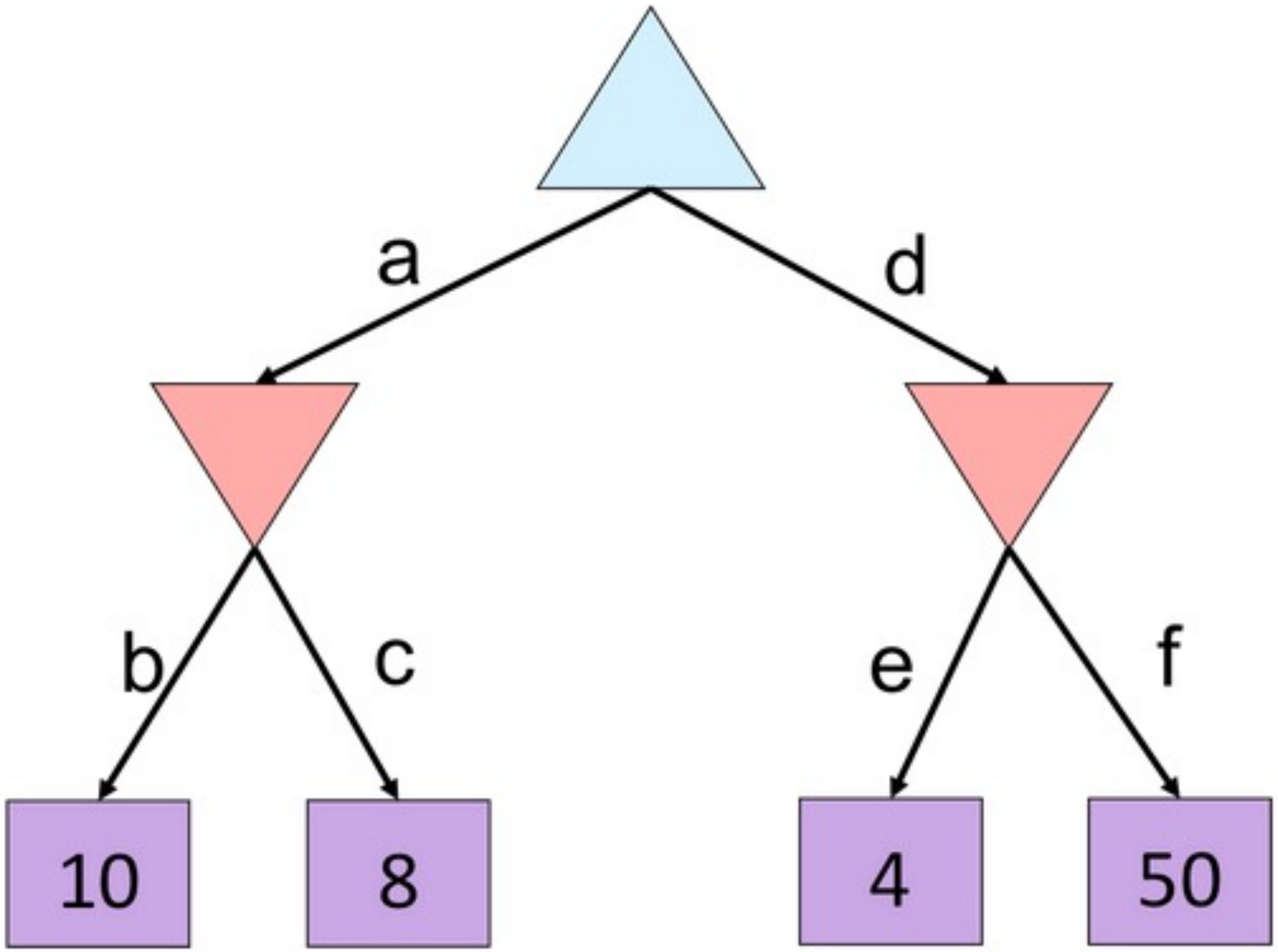
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-Beta Pruning Properties

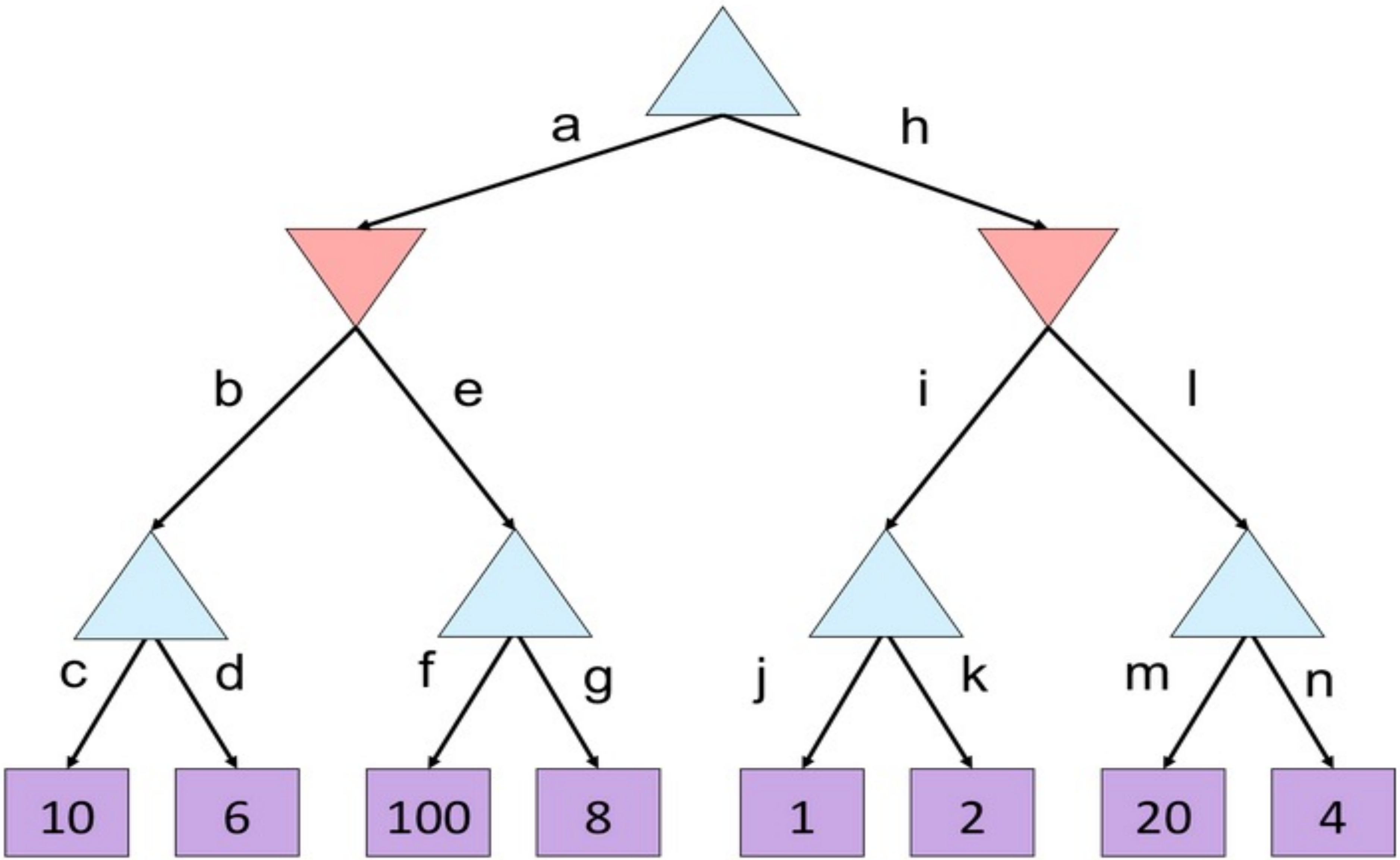
- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- This is a simple example of metareasoning (computing about what to compute)



Alpha-Beta Quiz



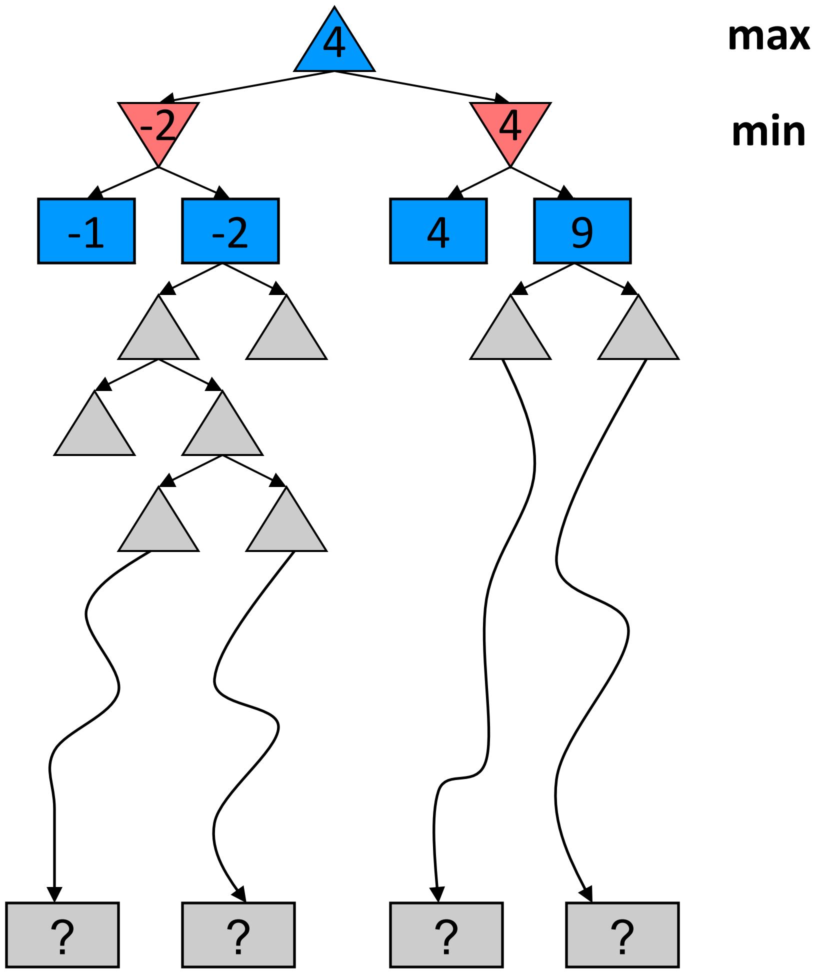
Alpha-Beta Quiz 2



Overcoming Resource Limits

Limiting Depth

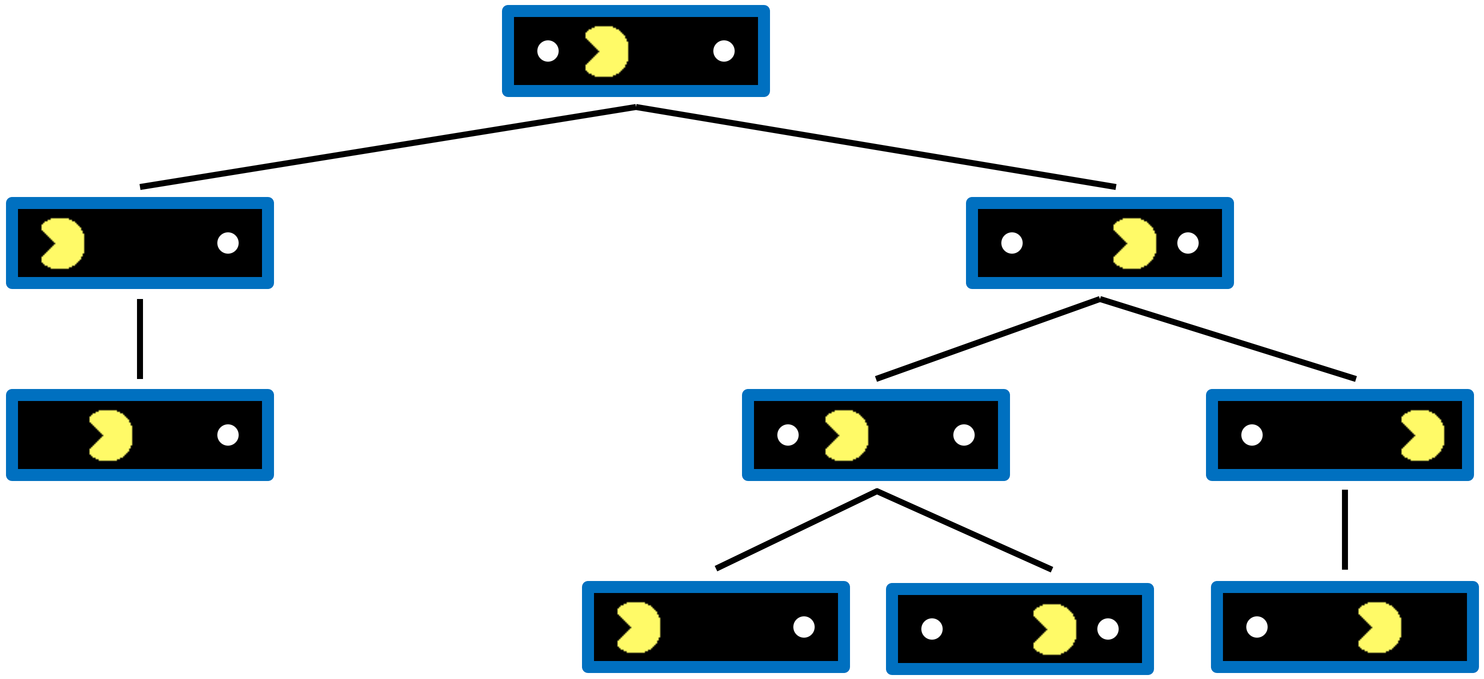
- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - 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



Video of Demo Thrashing (d=2)

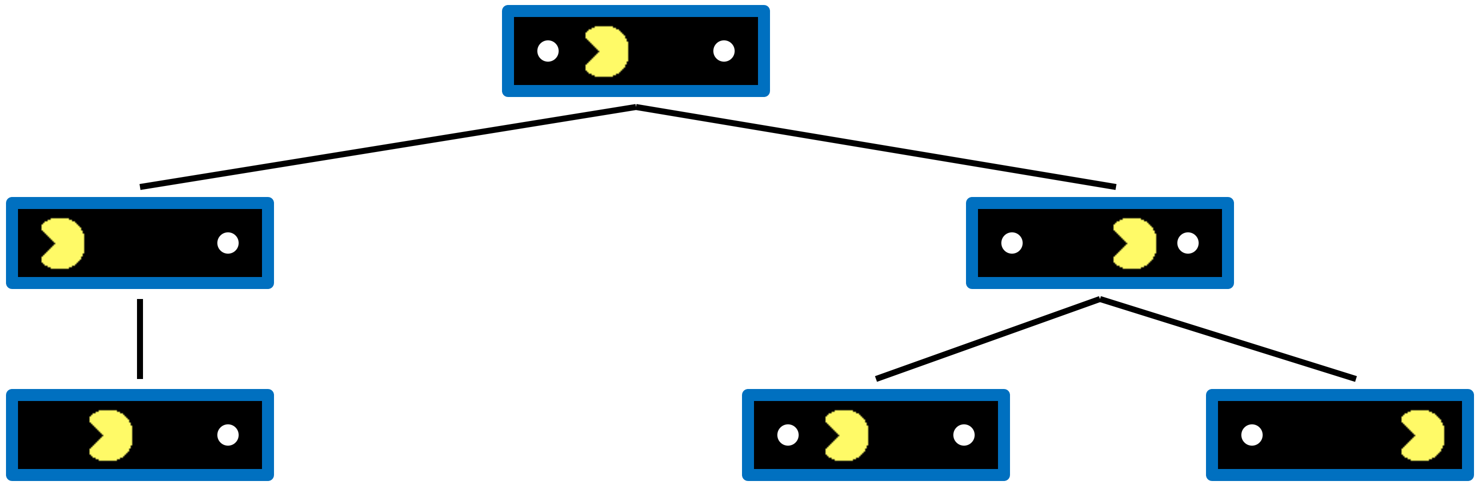
[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function) (L6D6)]

Why Pacman Starves (d=2)



Why Pacman Starves (d=2)

Why Pacman Starves (d=2)

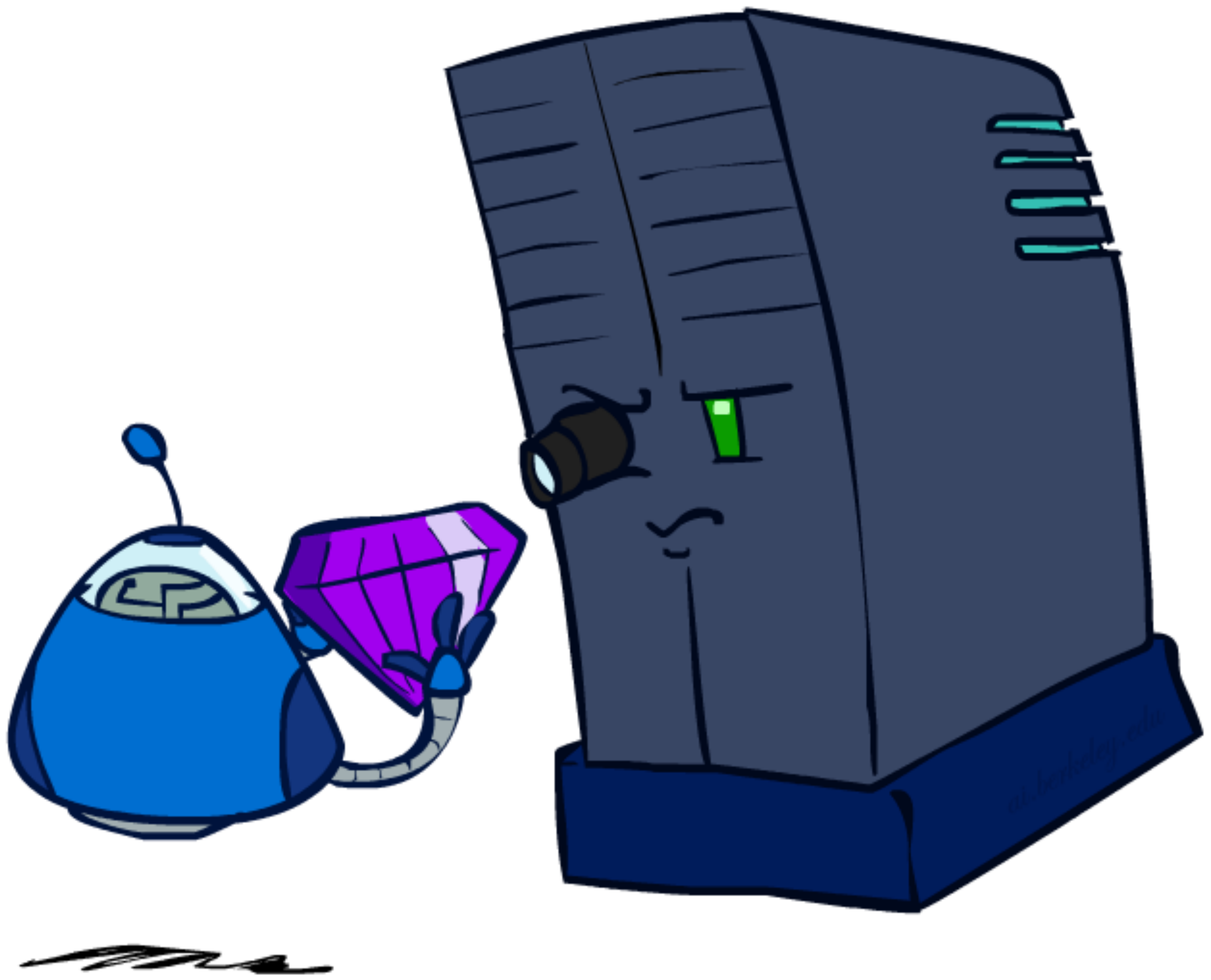


- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting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

Video of Demo Thrashing -- Fixed (d=2)

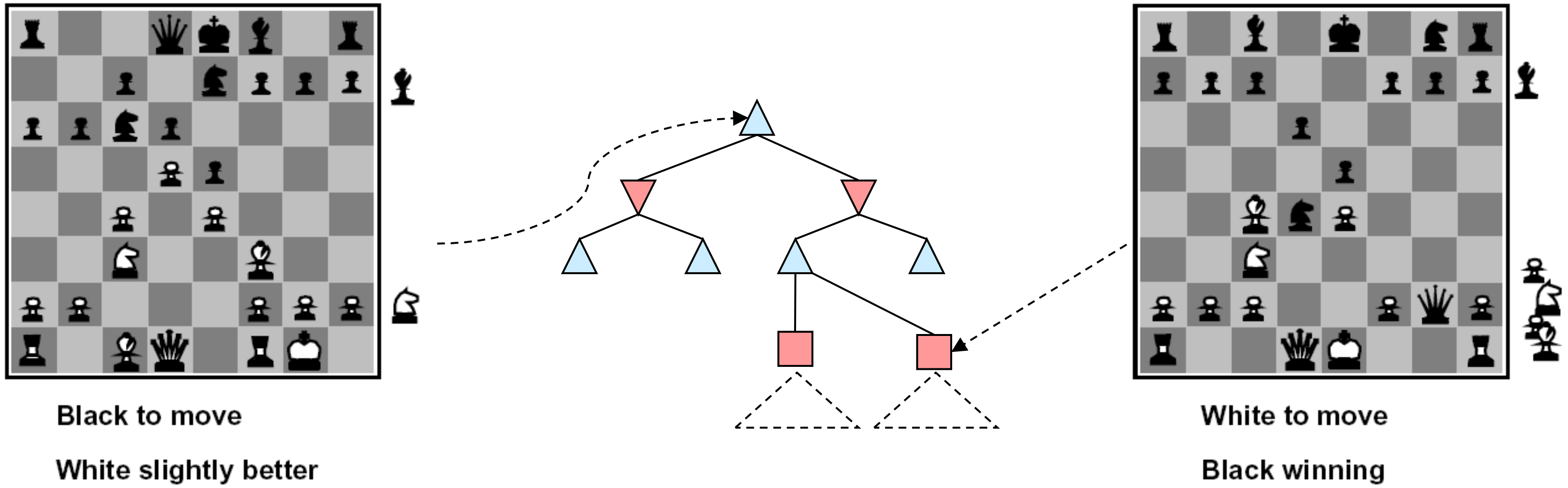
[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function) (L6D7)]

Evaluation Functions



Evaluation Functions

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

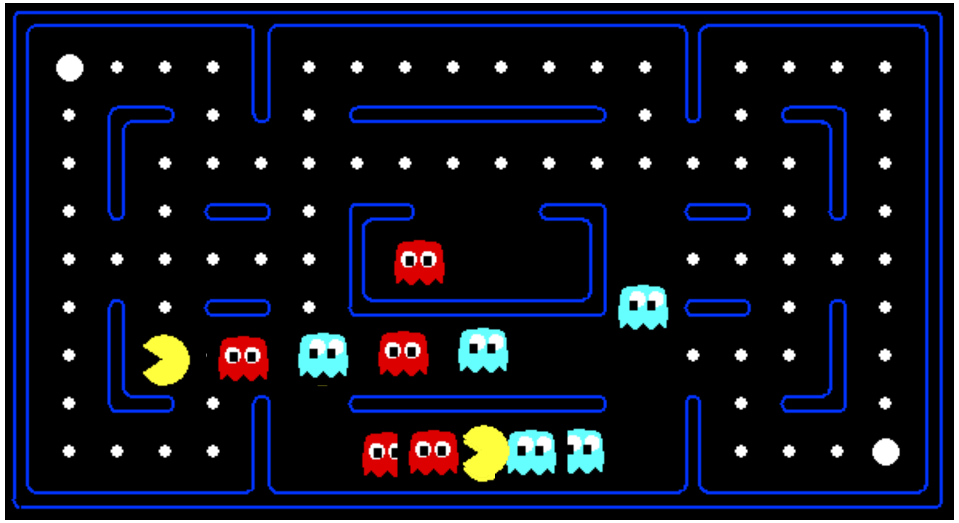


- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

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

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

Evaluation for Pacman



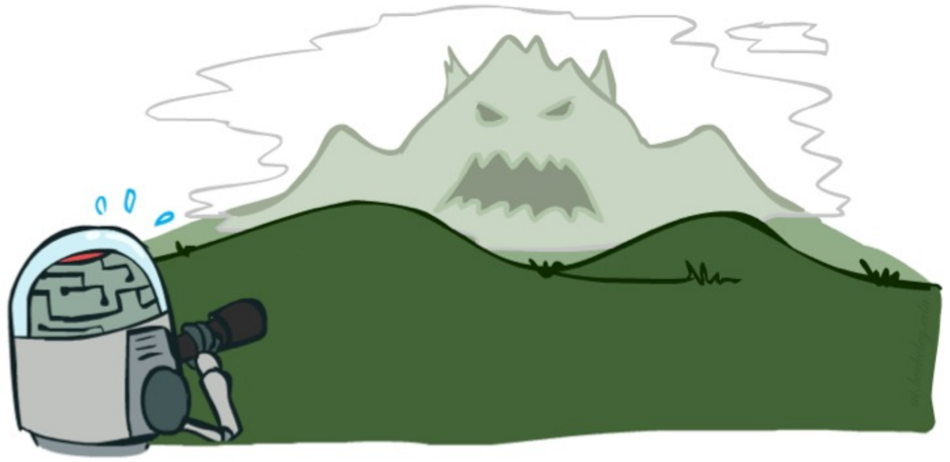
[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function), smart ghosts coordinate (L6D6,7,8,10)]

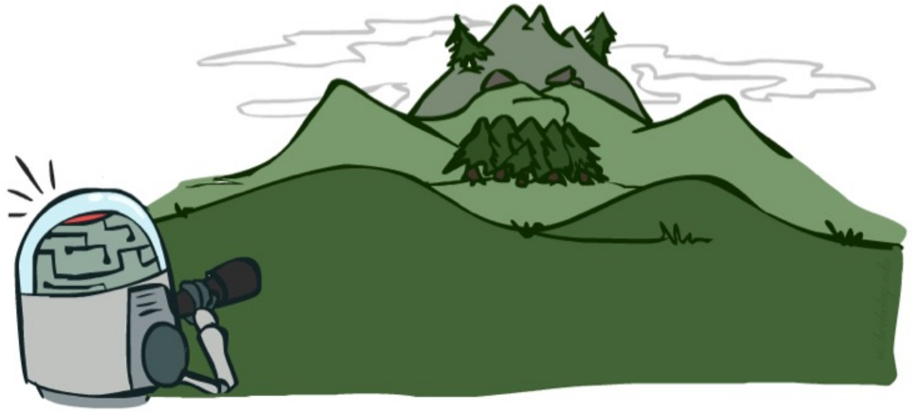
Video of Demo Smart Ghosts (Coordination)

Video of Demo Smart Ghosts (Coordination) – Zoomed In

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)

What we did today

- Introduced multi-agent games
  - Come up in many places in AI
  - Different types of games (focused on *zero-sum* today)
- Reviewed single-agent search trees (from previous lectures)
  - *Value* of a state is an important concept that will come up in the future
- Modified search trees to include opponent actions
  - Assumed opponent acts in a way that is worst for you
  - Called this *Minimax Search*
- Looked at efficiency (not good) and suggested two ways to improve it
  - Alpha-Beta pruning (exact, some gains but not huge)
  - Limiting tree depth (big gains, but not exact and needs heuristics)

Next Time: Uncertainty!

- We’ll extend search to deal with uncertain outcomes