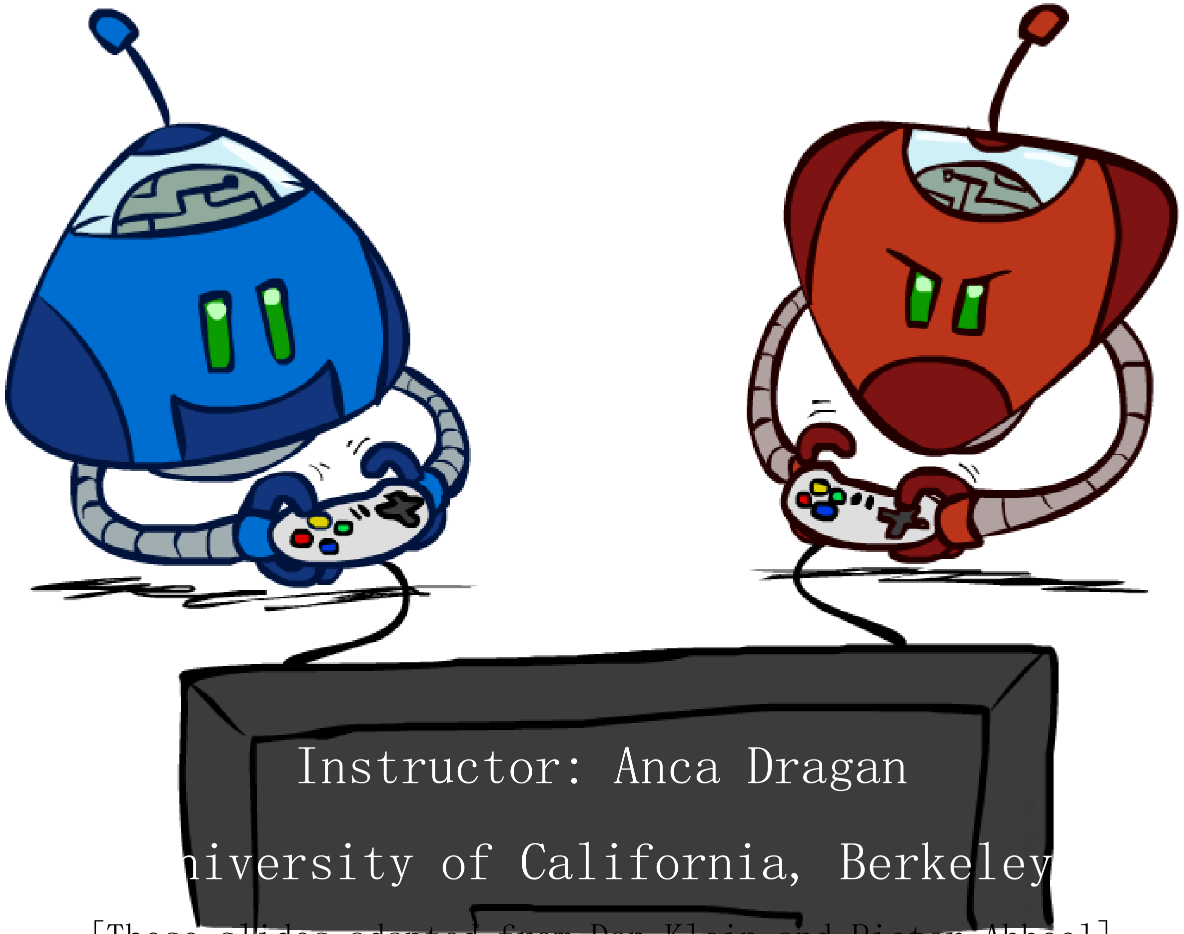
CS 188: Artificial Intelligence

Search with Other Agents

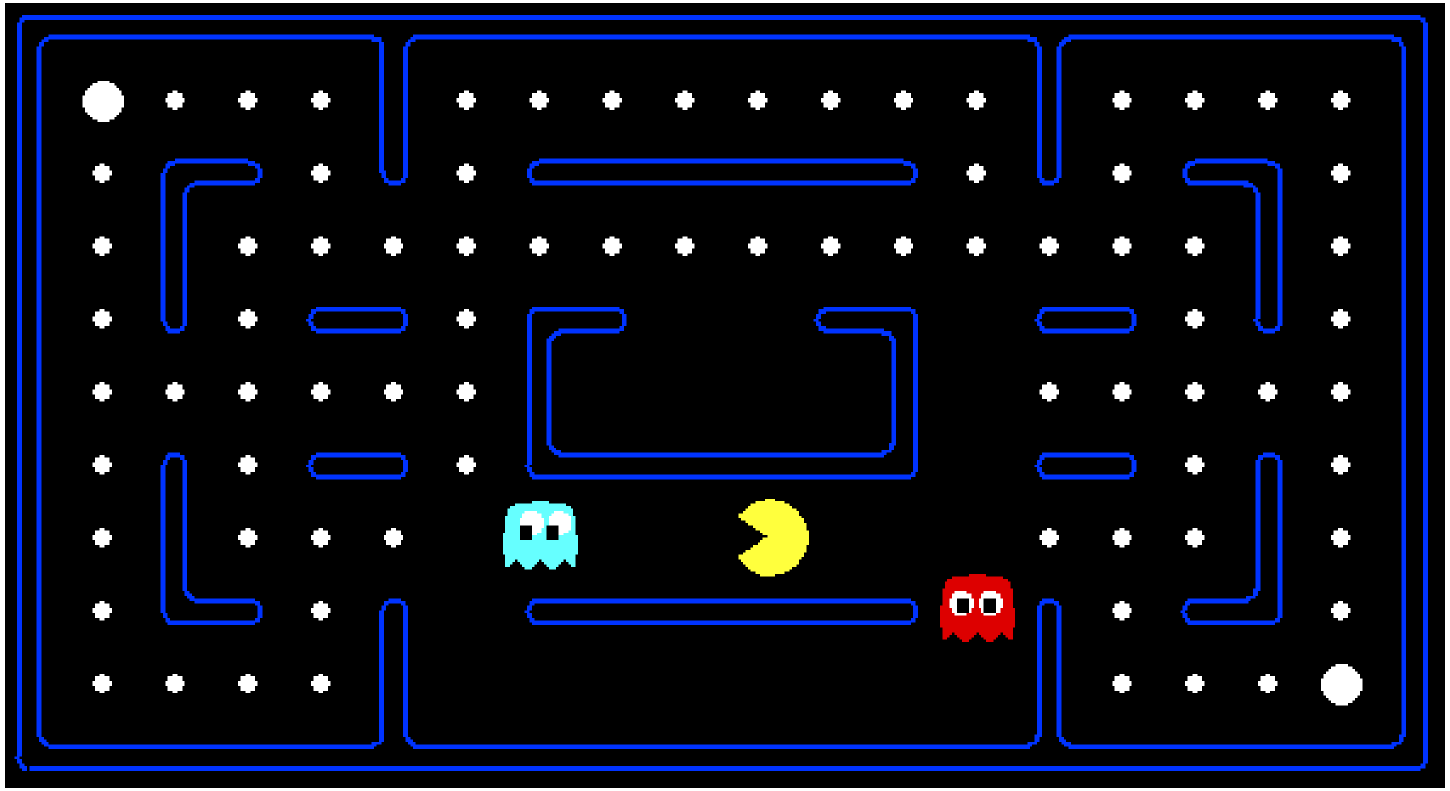


Instructor: Anca Dragan

niversity of California, Berkeley

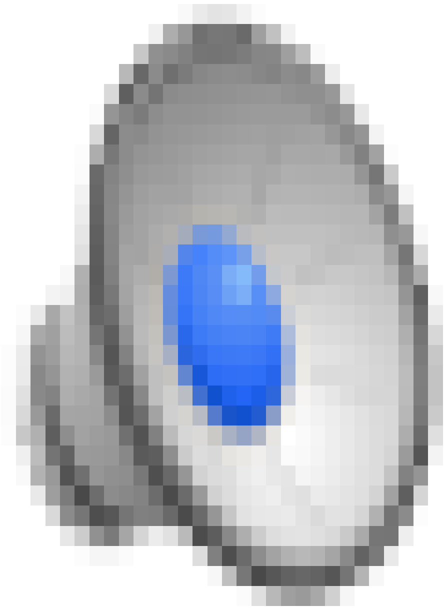
[These slides adapted from Dan Klein and Pieter Abbeel]

Behavior from Computation



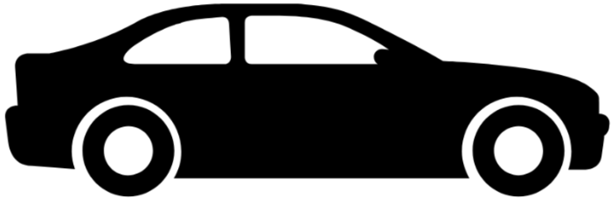
[Demo: mystery pacman (L6D1)]

Video of Demo Mystery Pacman



Agents Getting Along with Other Agents

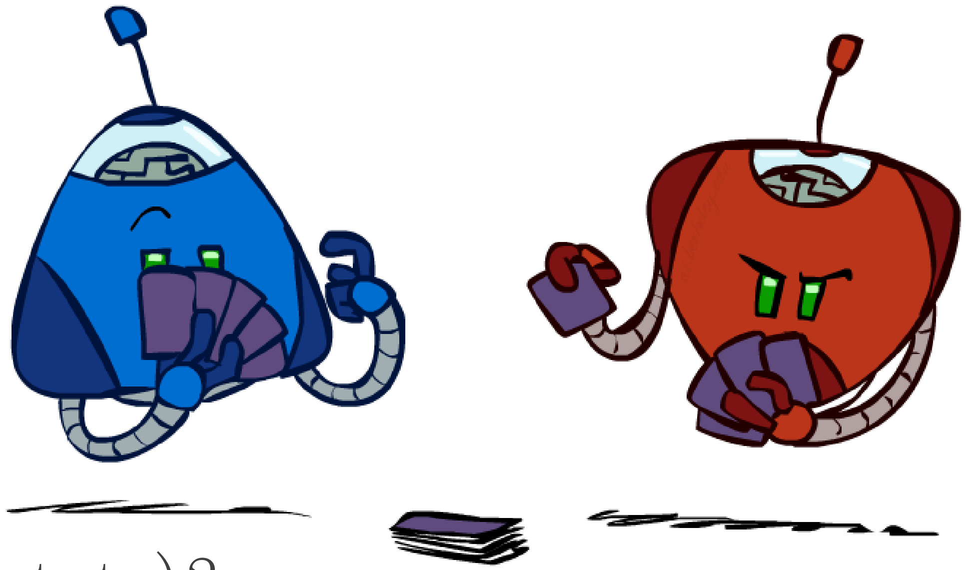
Agents Getting Along with Humans



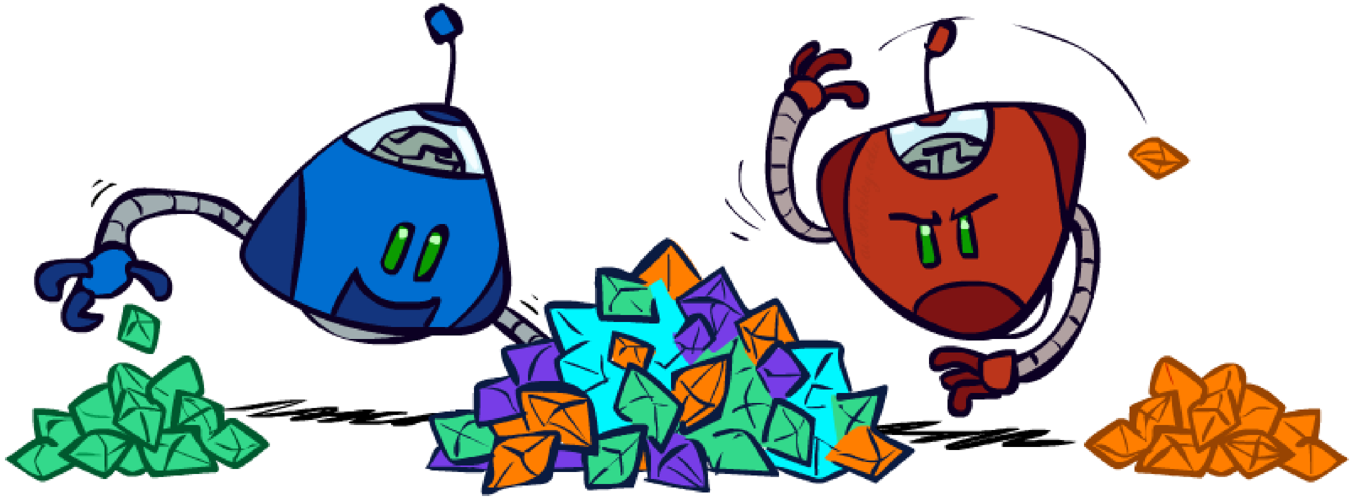
Types of Games

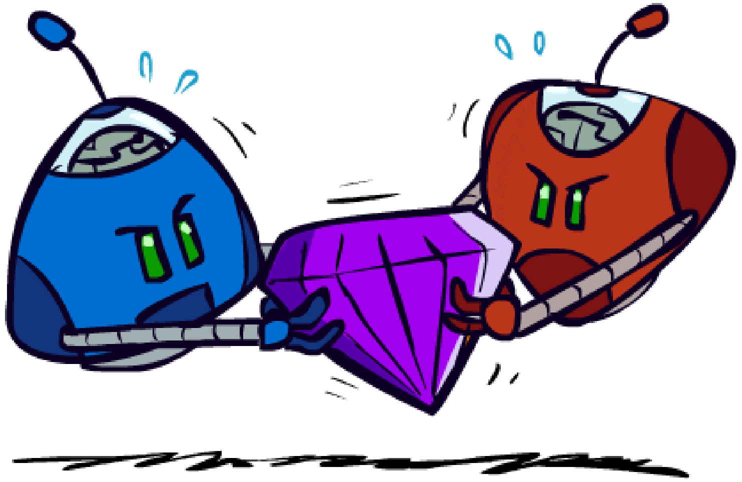
- Many different kinds of games!

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



Types of Games





- 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

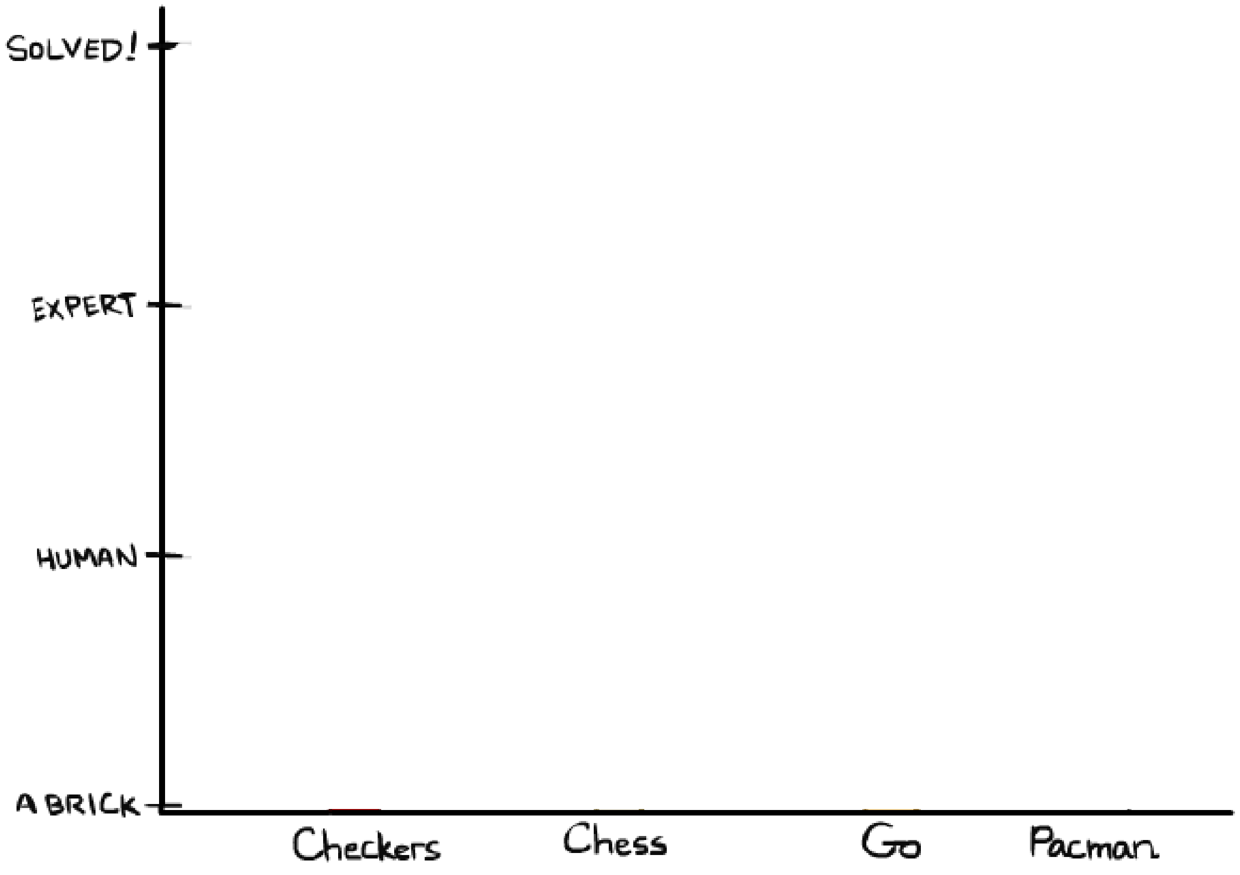
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition

Zero-Sum Game Games ☺

- **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, though the best humans still beat the best machines. In go, b > 300! Classic programs use pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods.



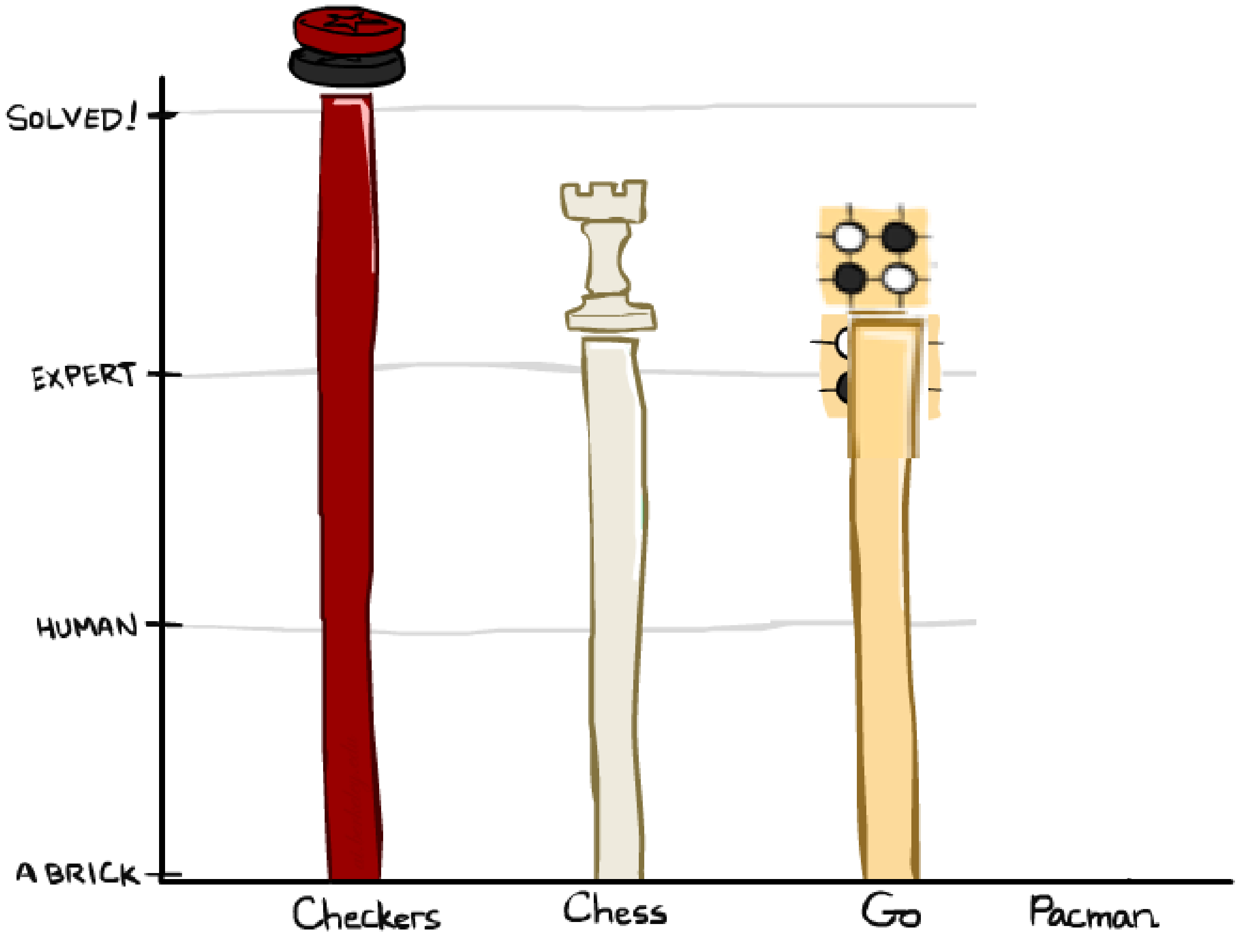
Zero-Sum Game Games ☺

- **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.

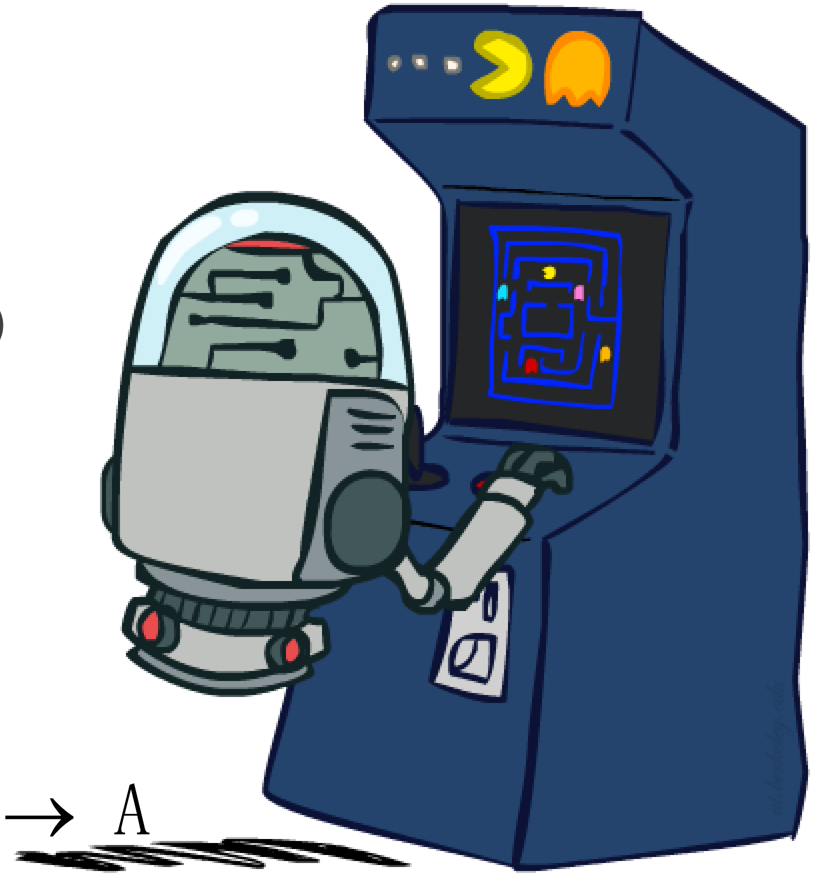
- **Go :2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search, learned evaluation function.**

- **Pacman**



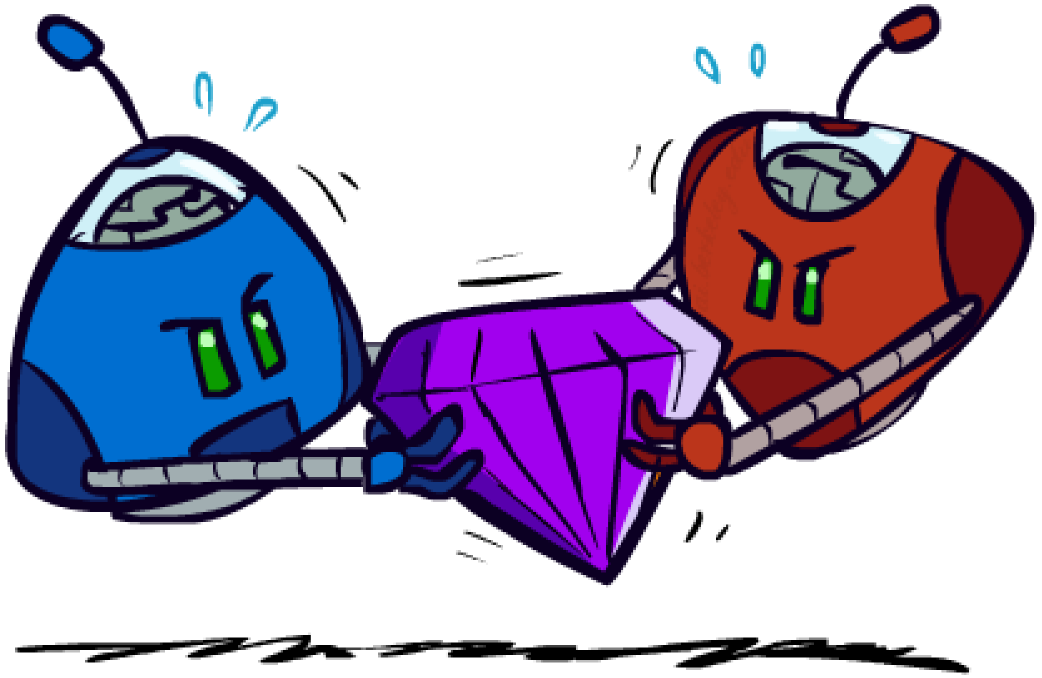
Deterministic Games with Terminal Utilities

- 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

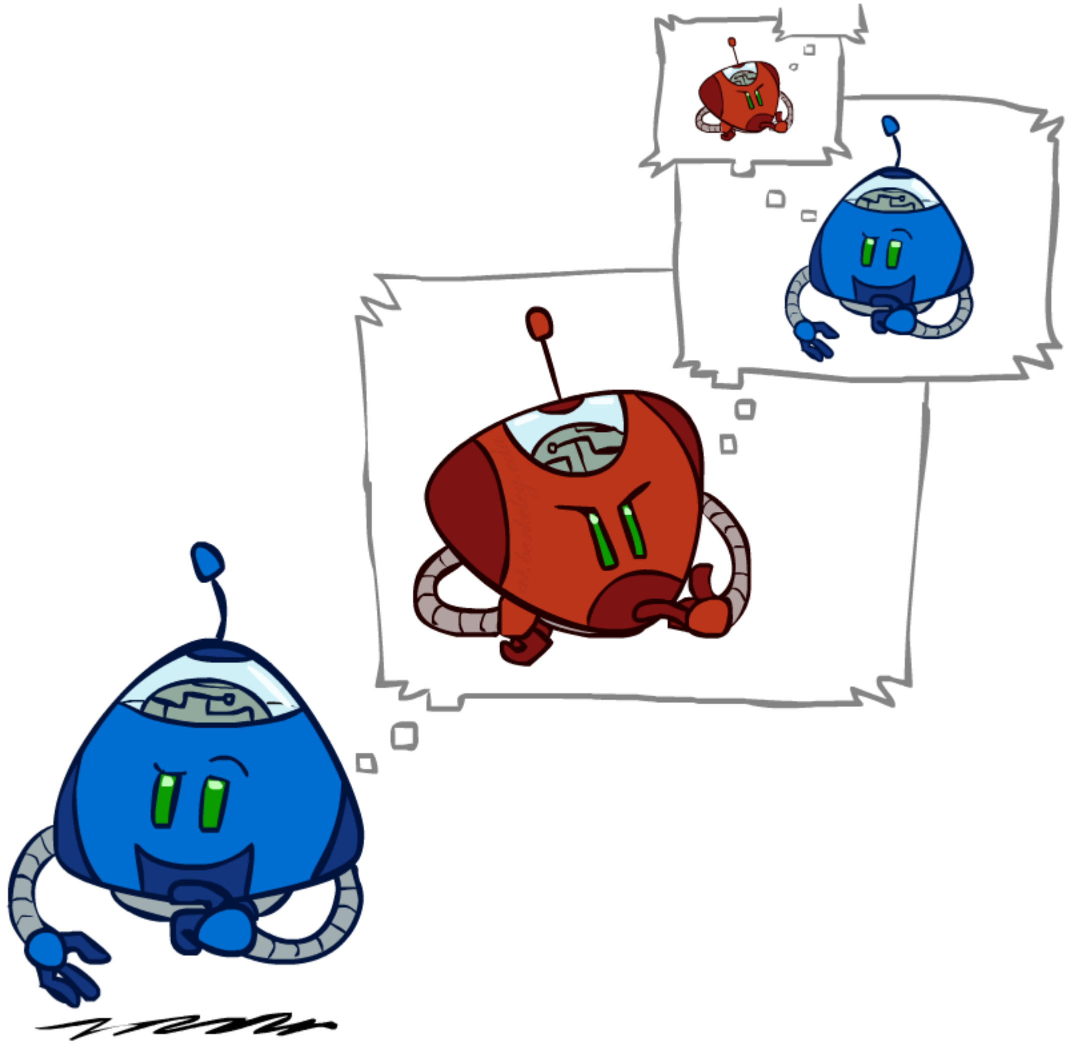


- Solution for a player is a policy: S $\rightarrow$ A

Adversarial Games



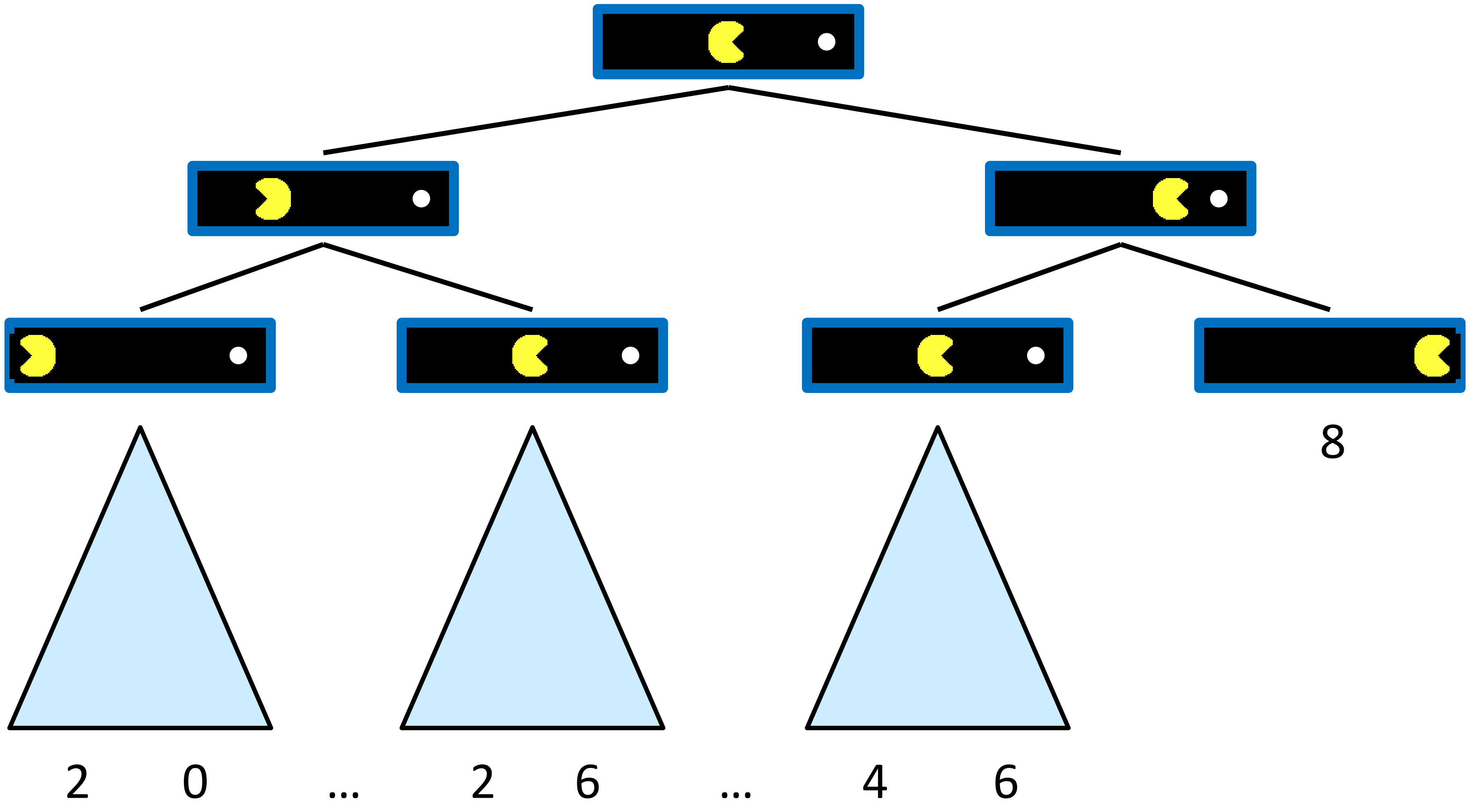
Adversarial Search



188 News: Cost -> Utility!

- no longer minimizing cost!
- agent now wants to maximize its score/utility!

Single-Agent Trees

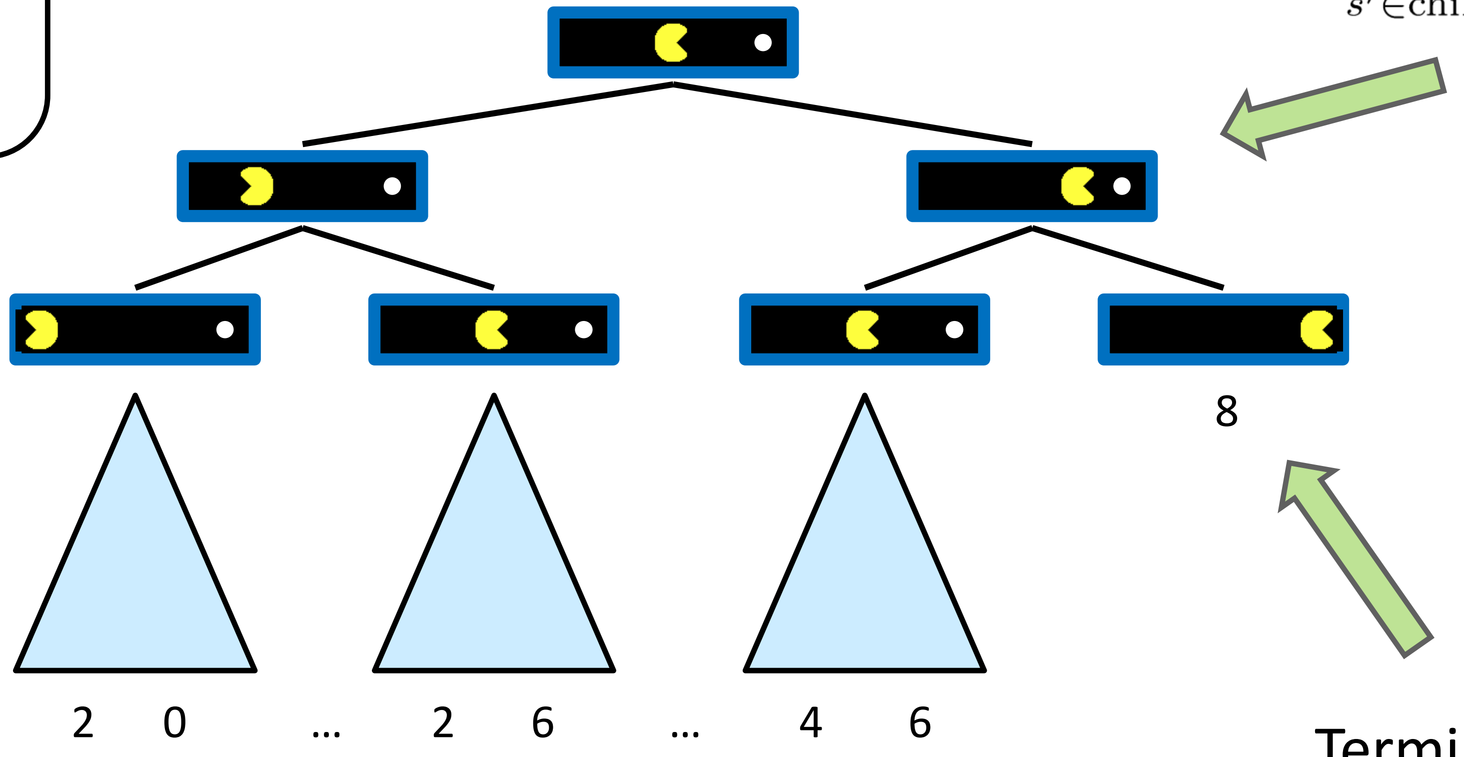


Value of a State

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

Non-Terminal States:

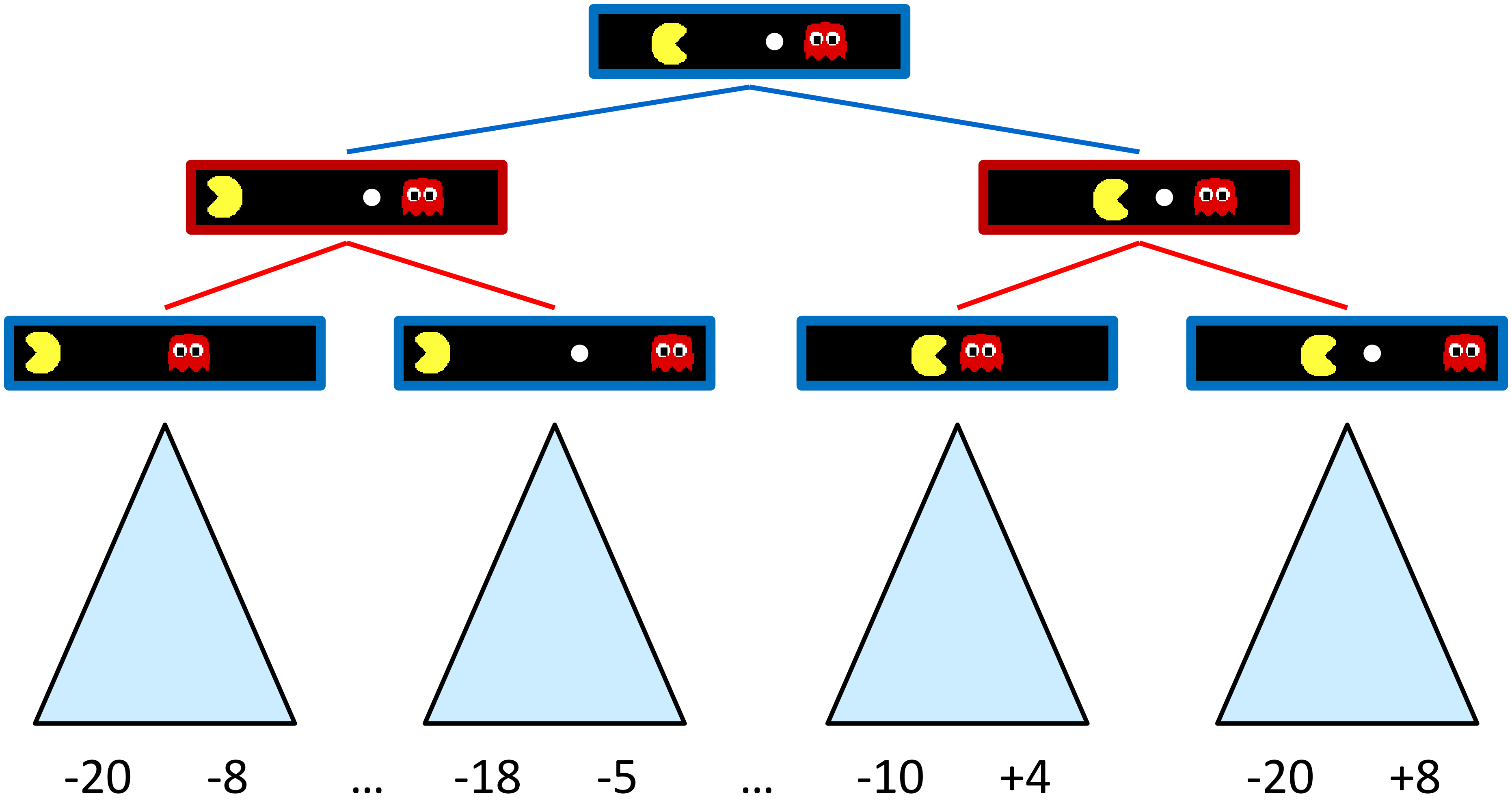
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



Terminal States:

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

Adversarial Game Trees



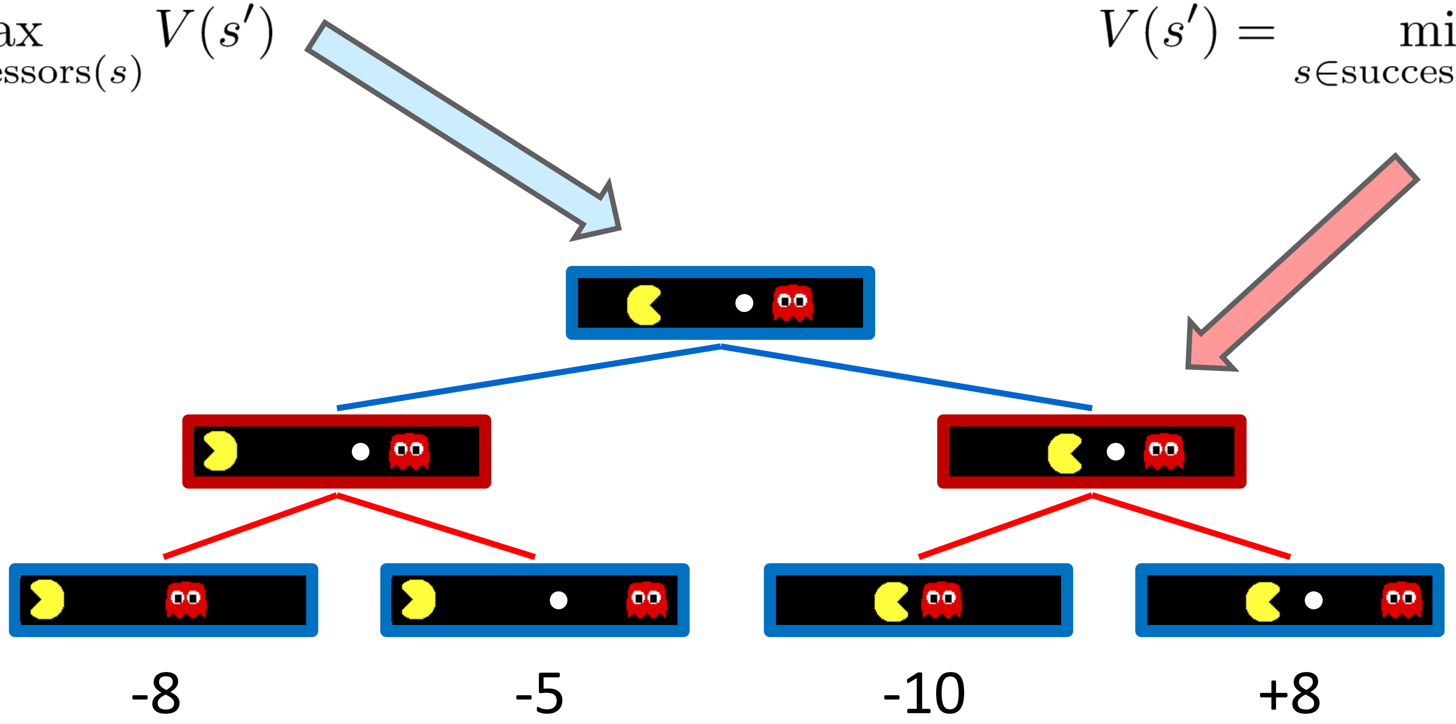
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

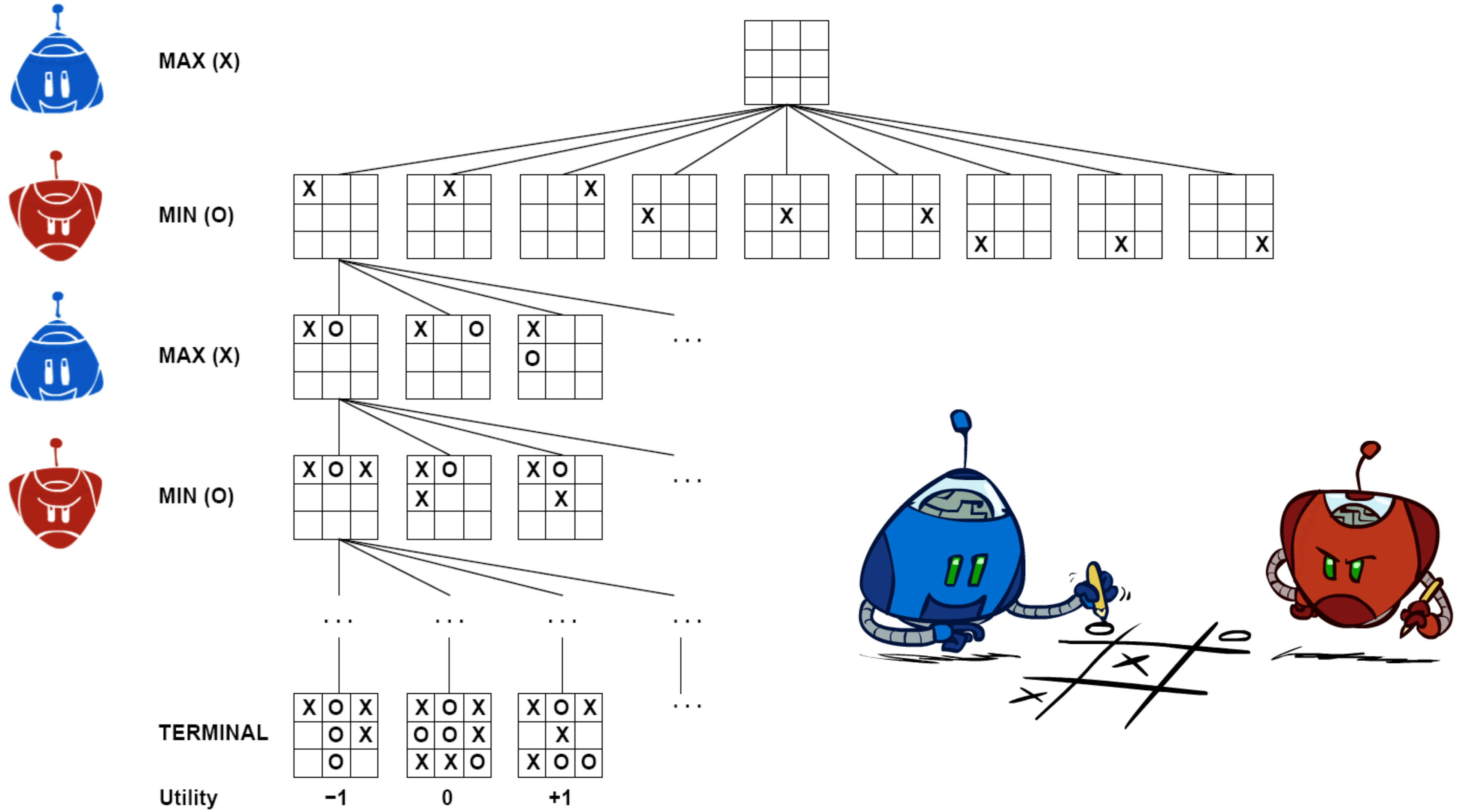
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

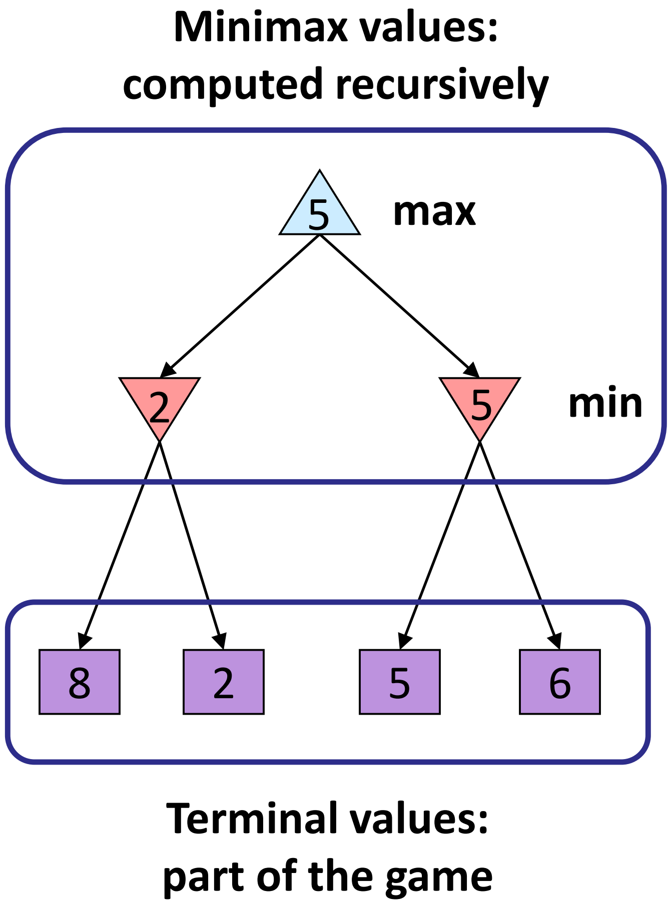
$$V(s) = \text{known}$$

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
```

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

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

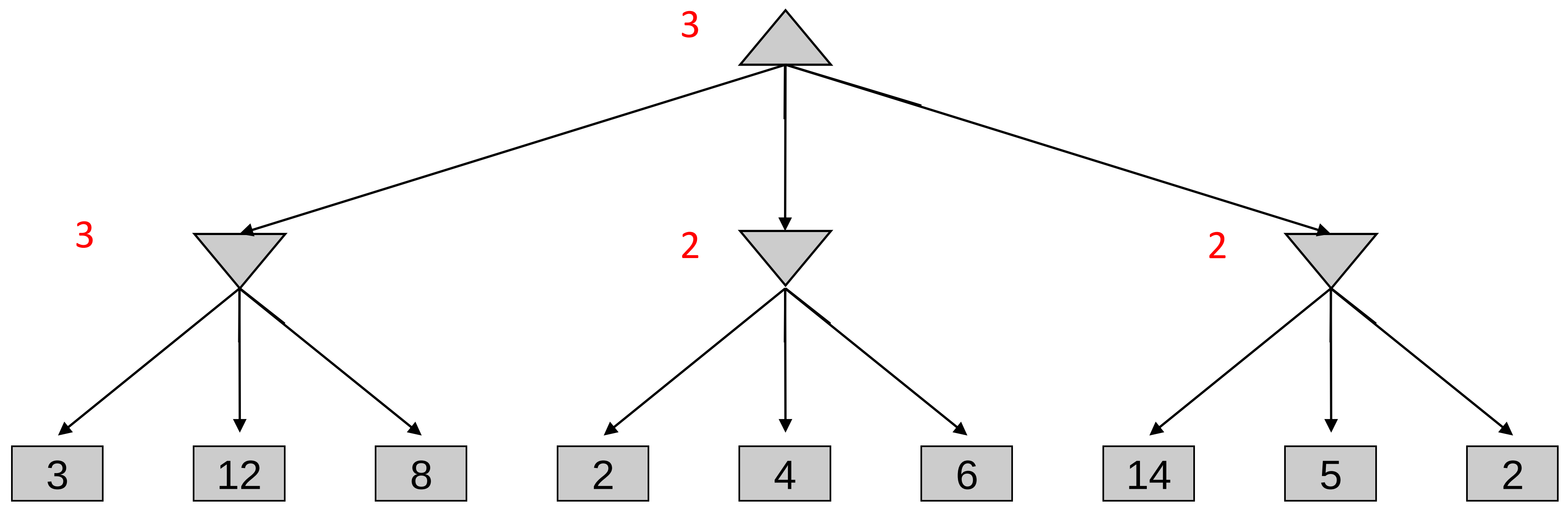
Minimax Implementation (Dispatch)

```
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)
```

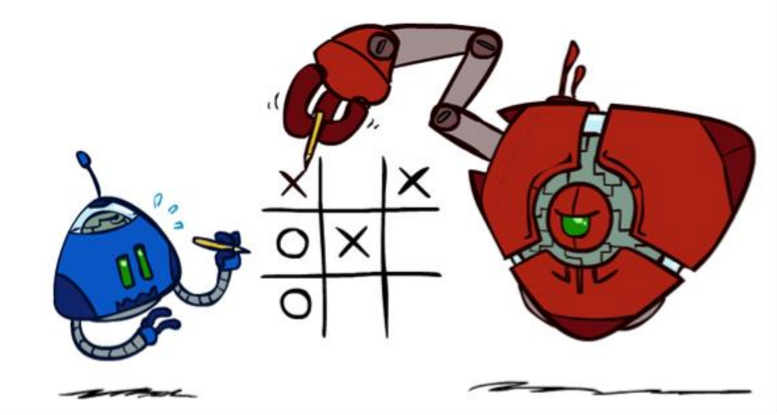
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

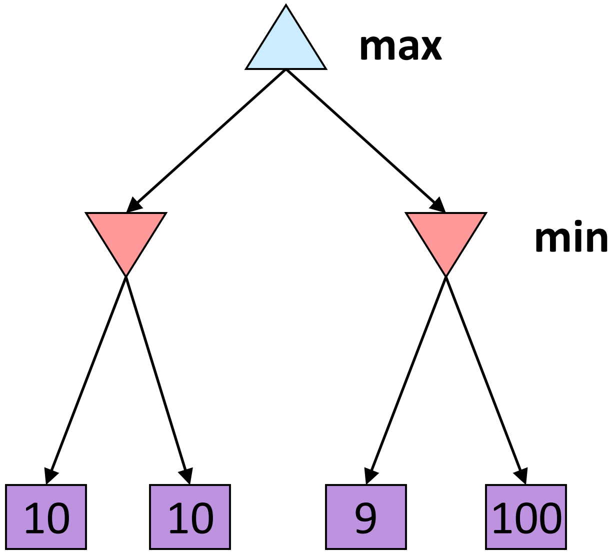
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

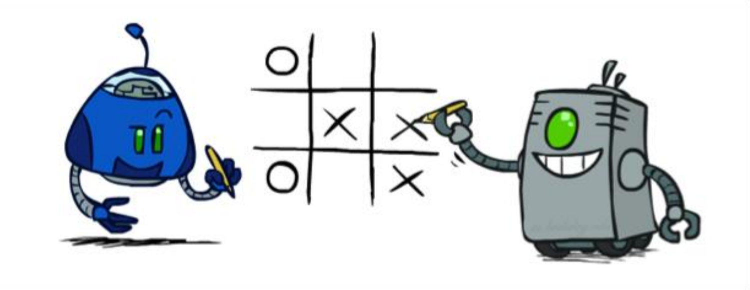
Minimax Example



Minimax Properties





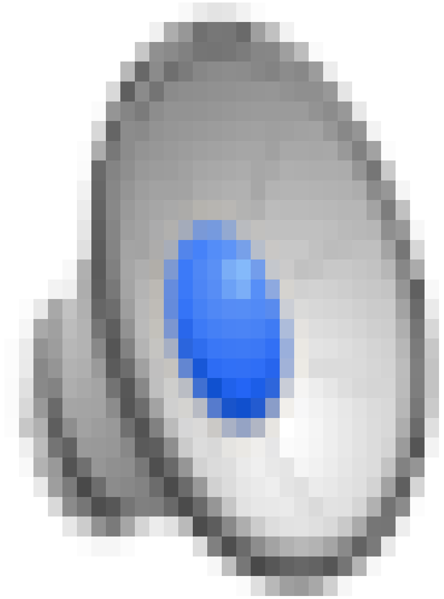


Optimal against a perfect player.
Otherwise?

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

Video of Demo Min vs. Exp (Min)

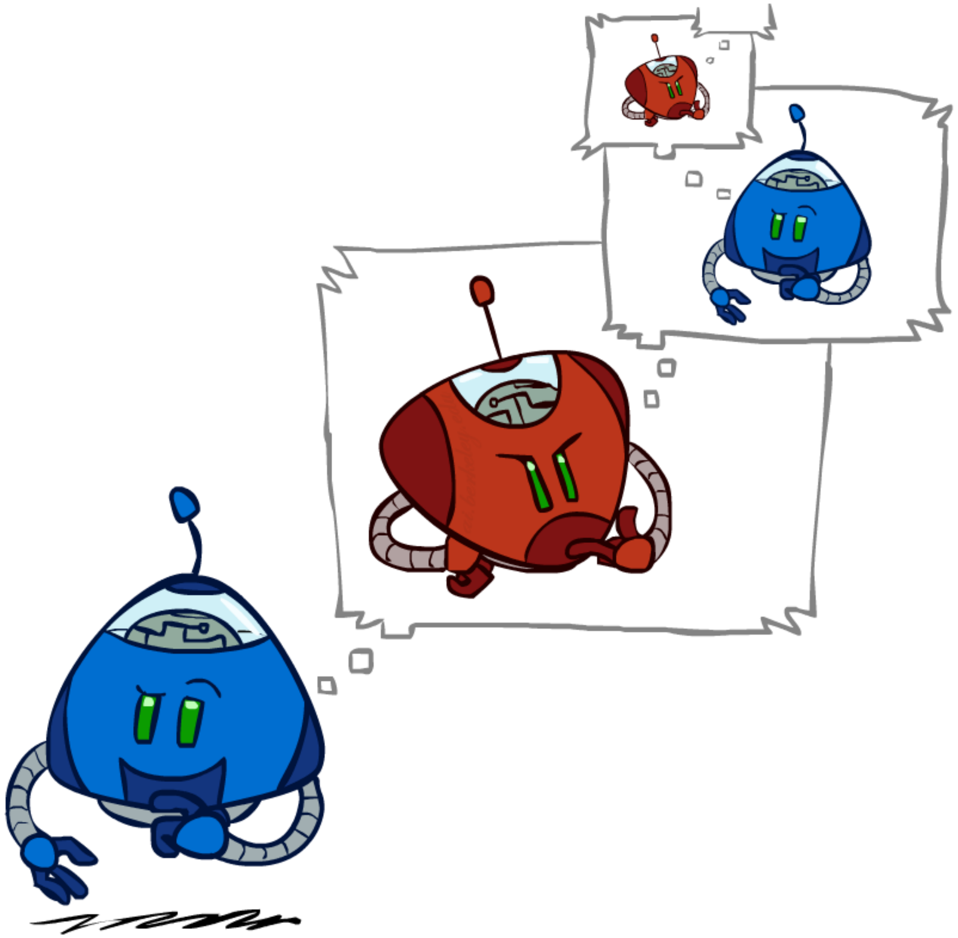
Video of Demo Min vs. Exp (Exp)



Minimax Efficiency

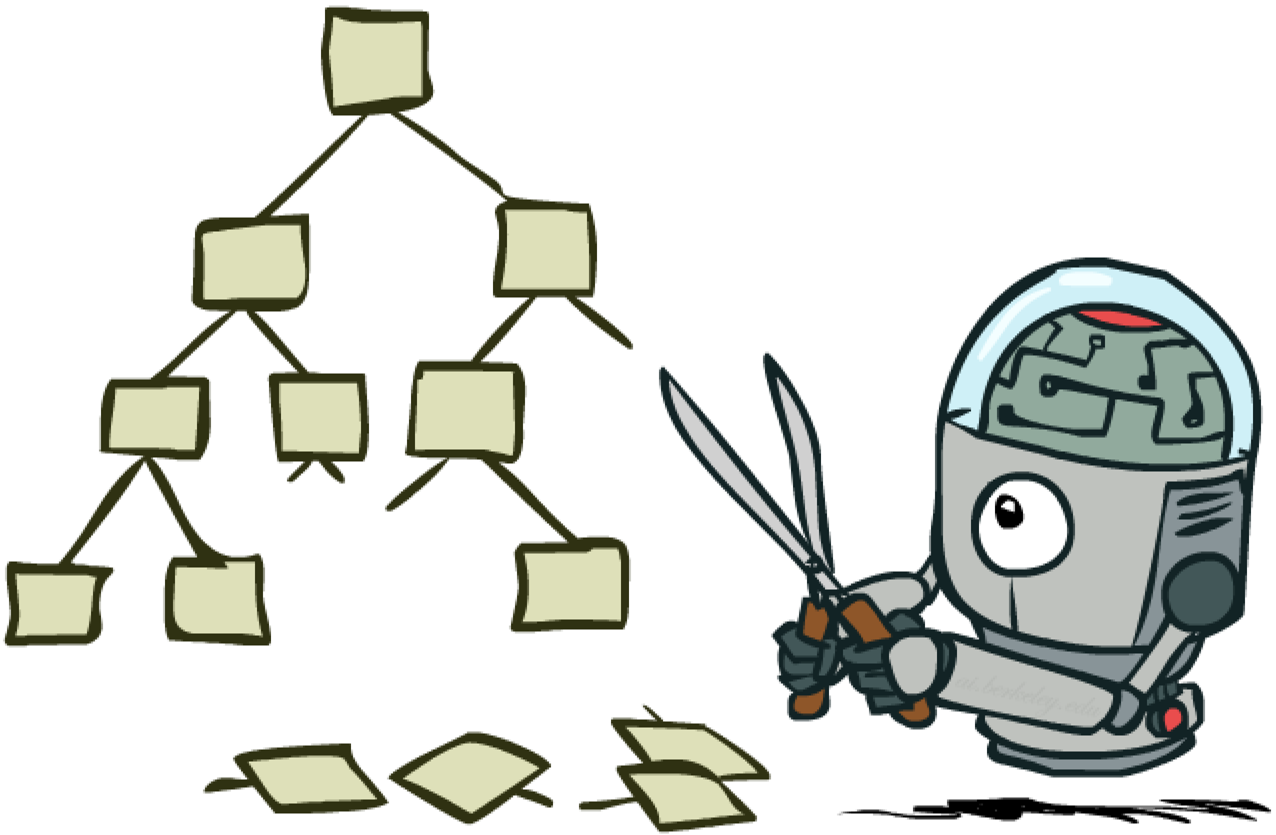
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

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

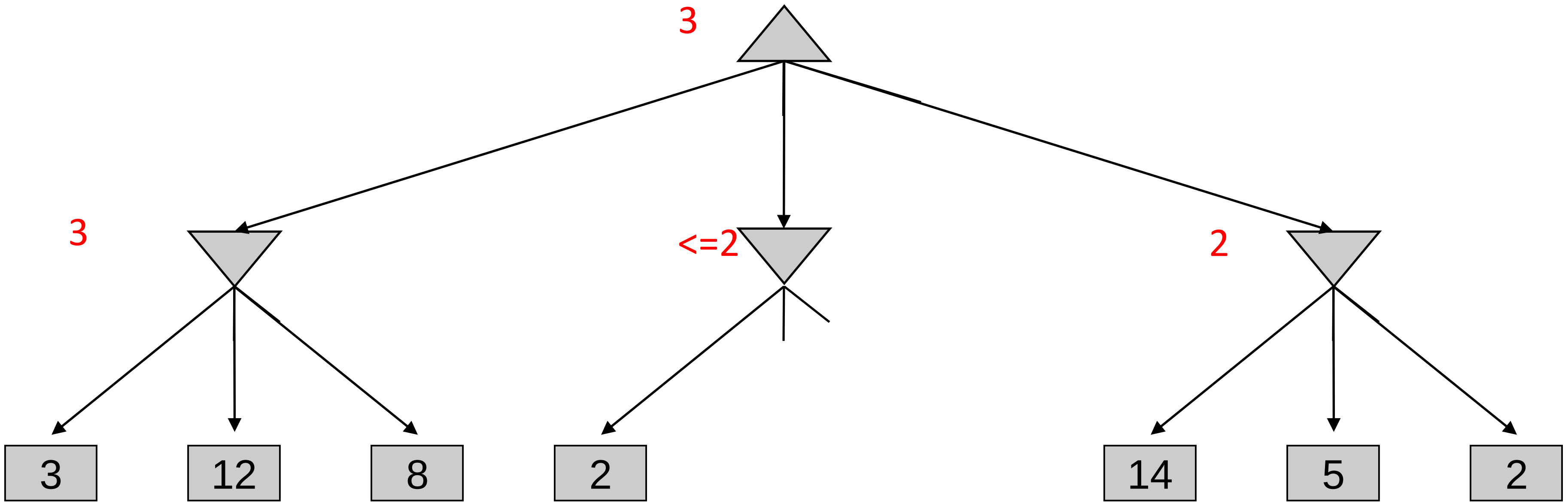


Resource Limits

Game Tree Pruning

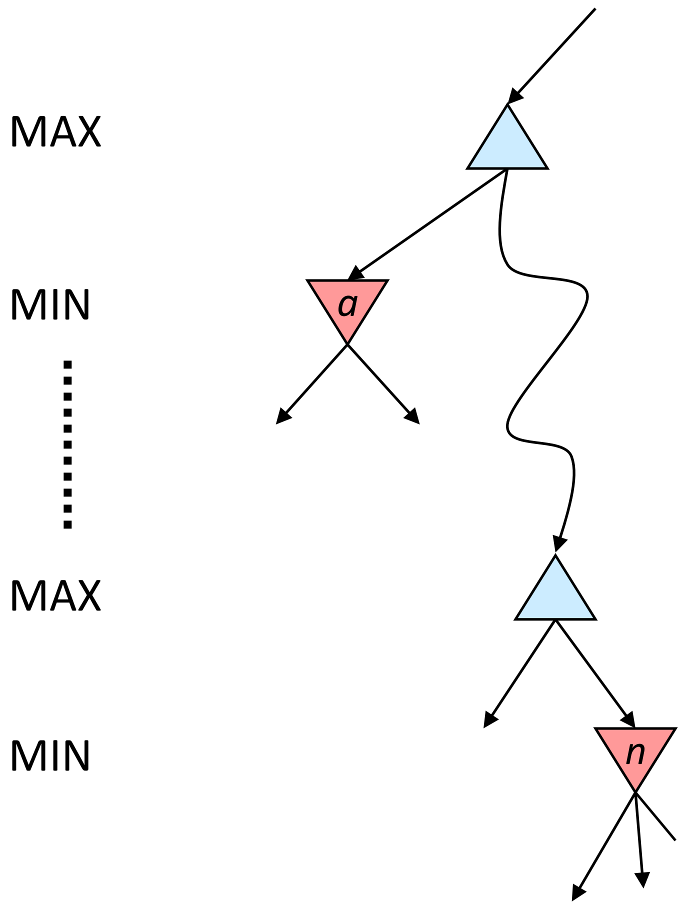


Minimax Example



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)



Alpha-Beta Implementation

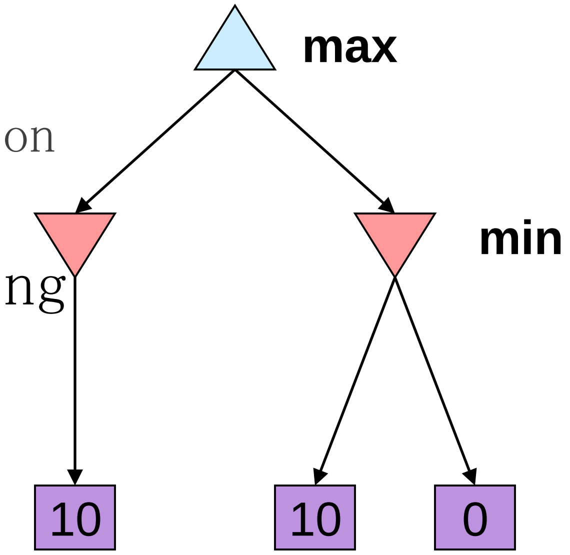
α : MAX's best option on path to root
β : MIN's best option on path to root

```
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
```

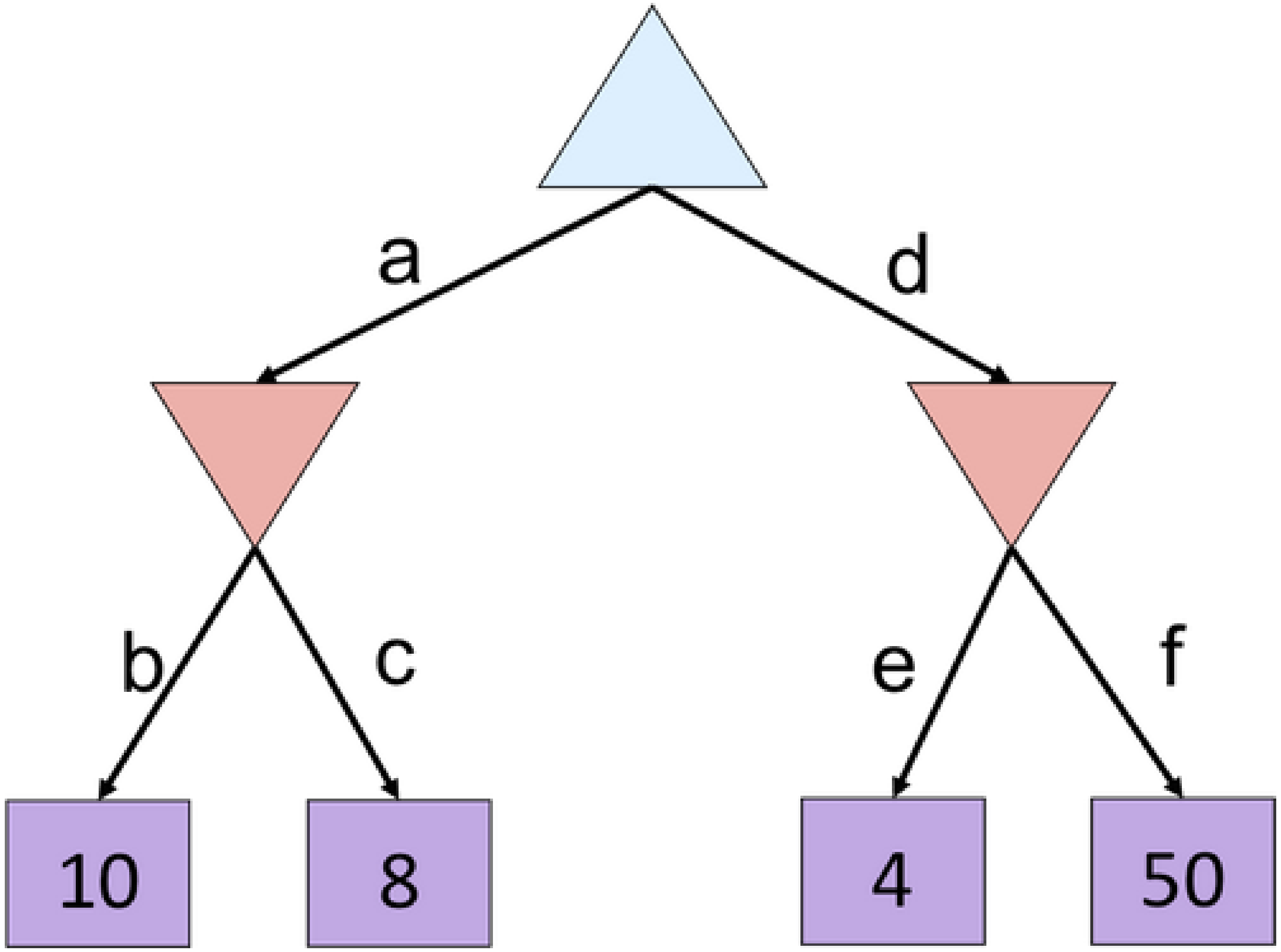
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-Beta Pruning Properties

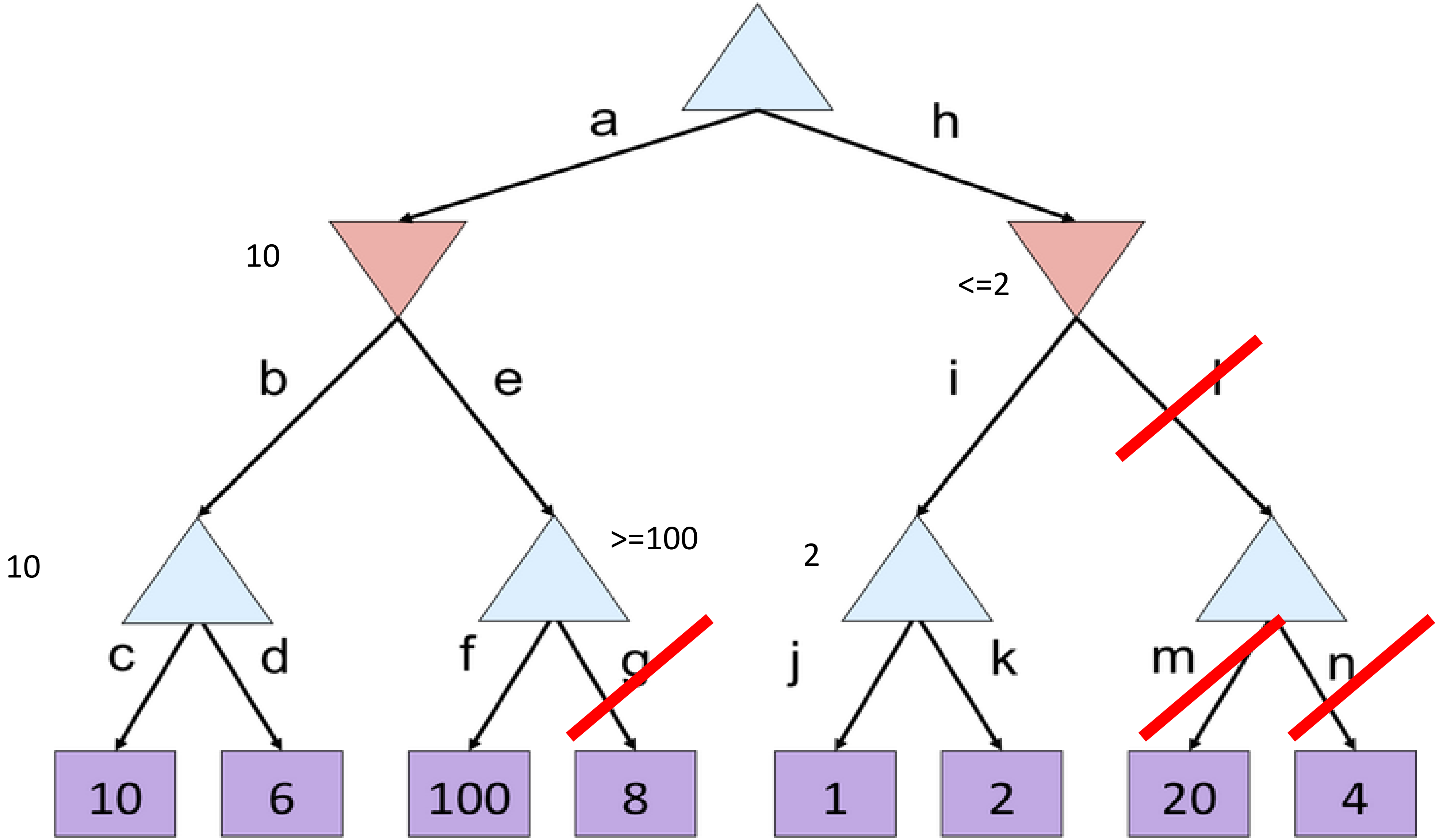
- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won' t let you do action selection
- Good child ordering improves effectiveness of pruning
- With "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



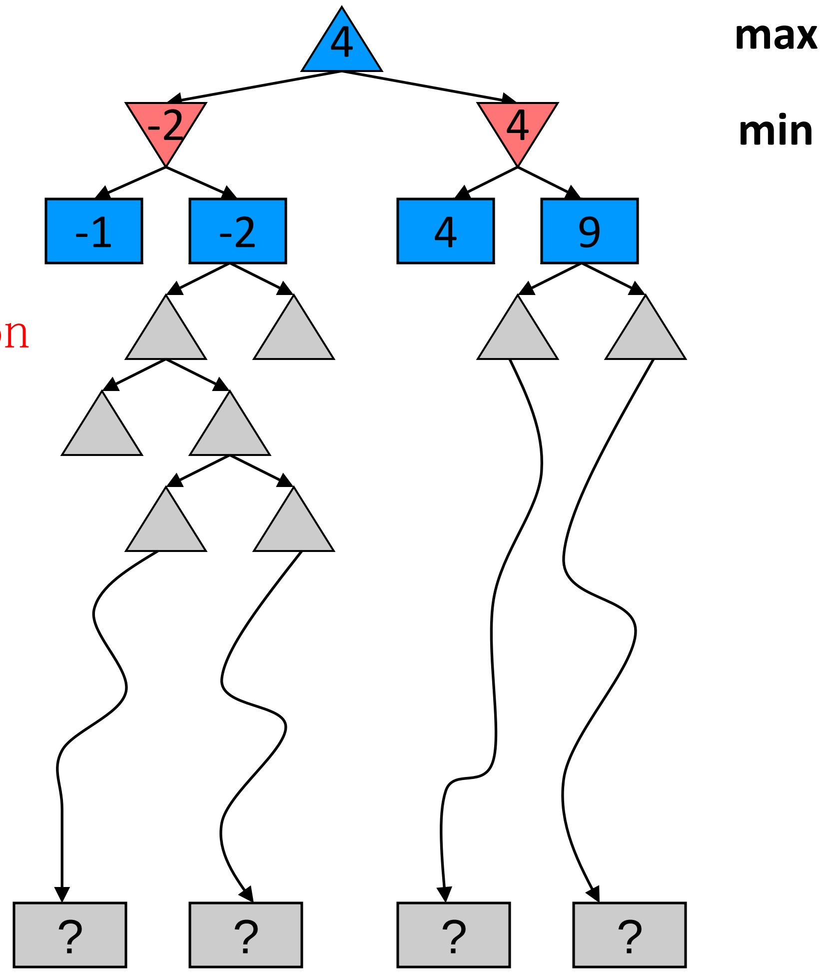
Alpha-Beta Quiz 2



Resource Limits

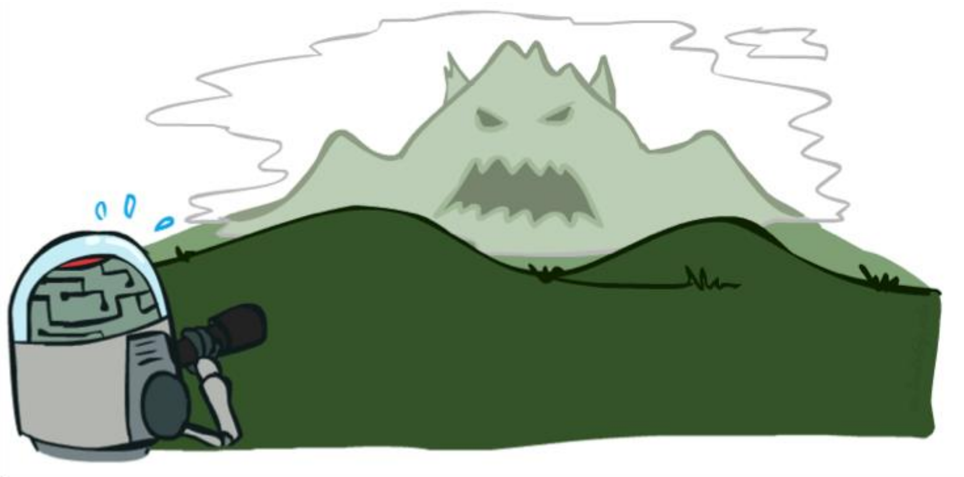
Resource Limits

- 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



Depth Matters

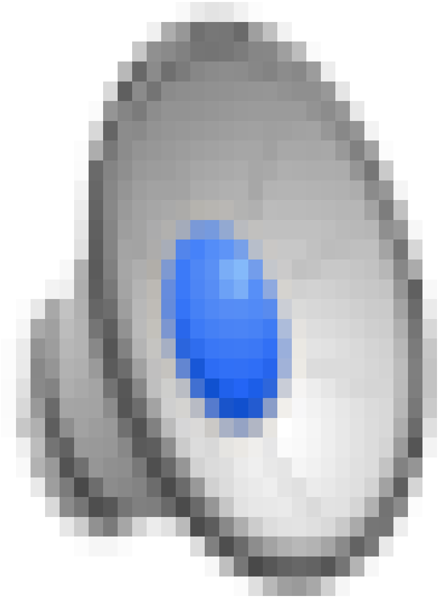
- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation



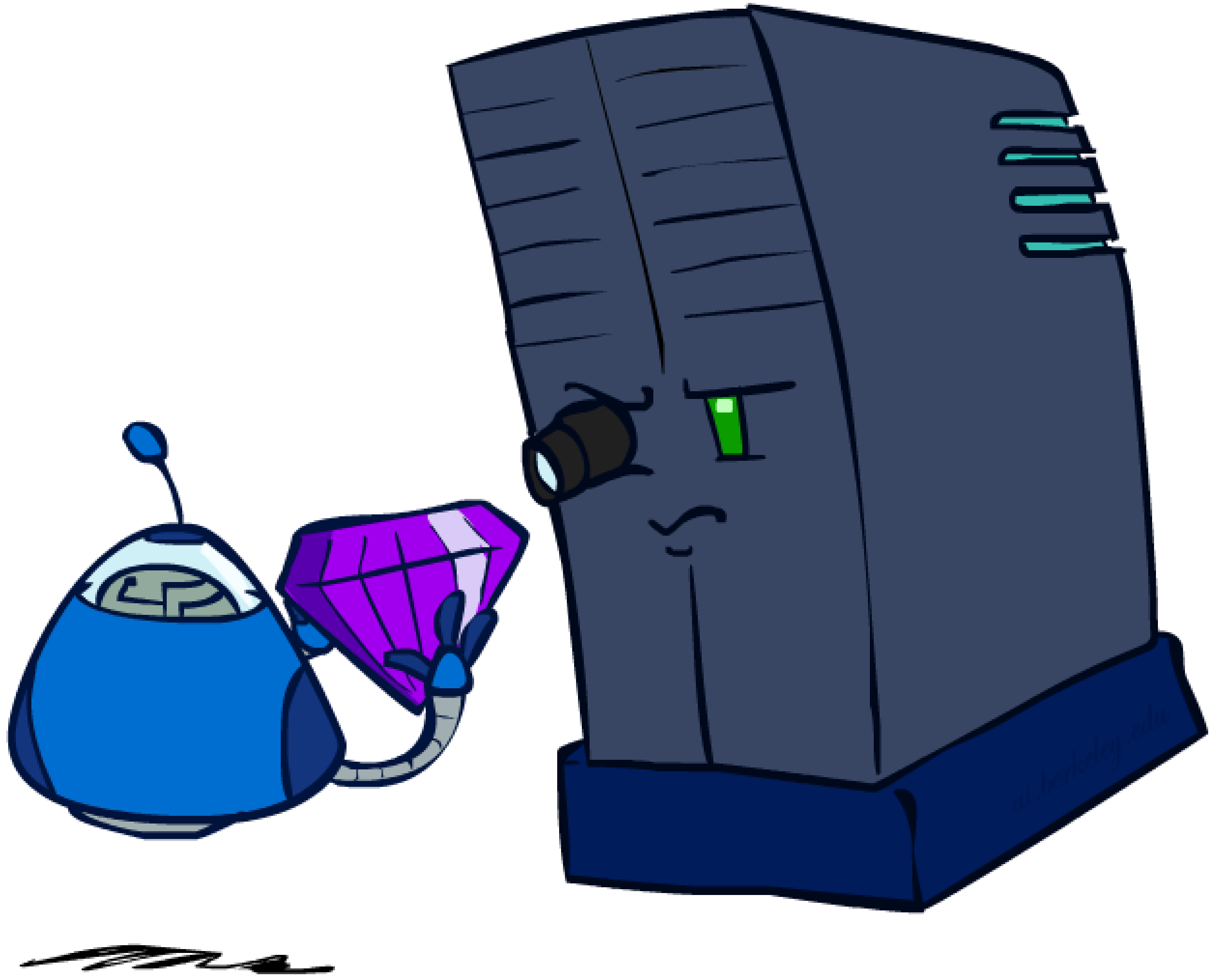
[Demo: depth limited (L6D4, L6D5)]

Video of Demo Limited Depth (2)

Video of Demo Limited Depth (10)

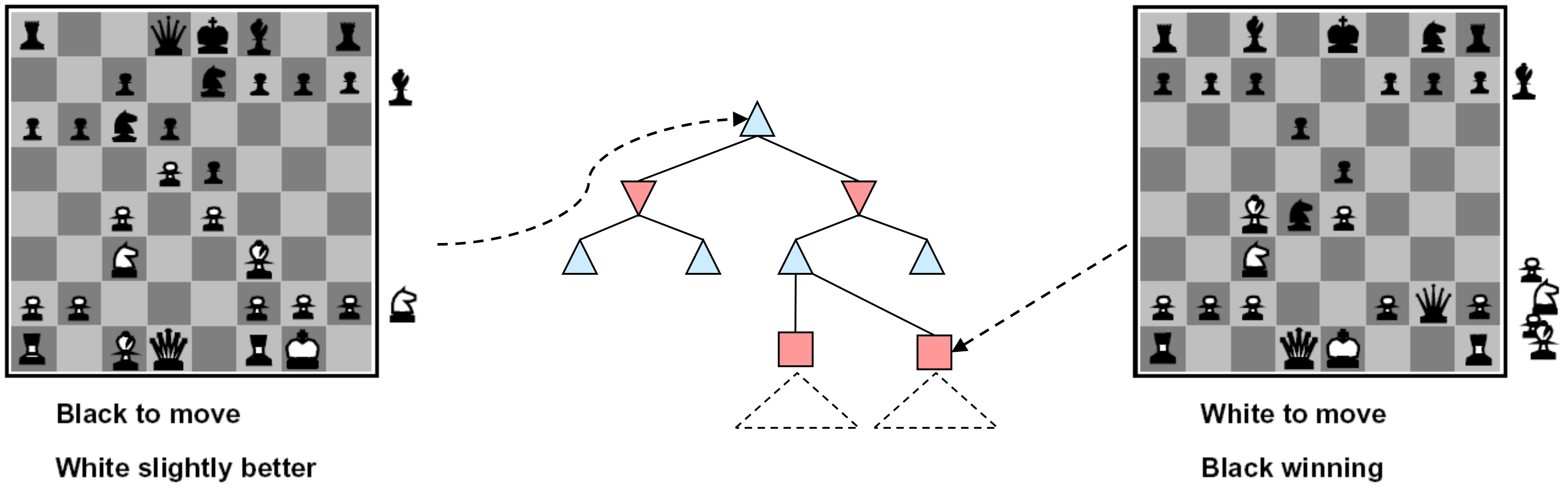


Evaluation Functions



Evaluation Functions

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

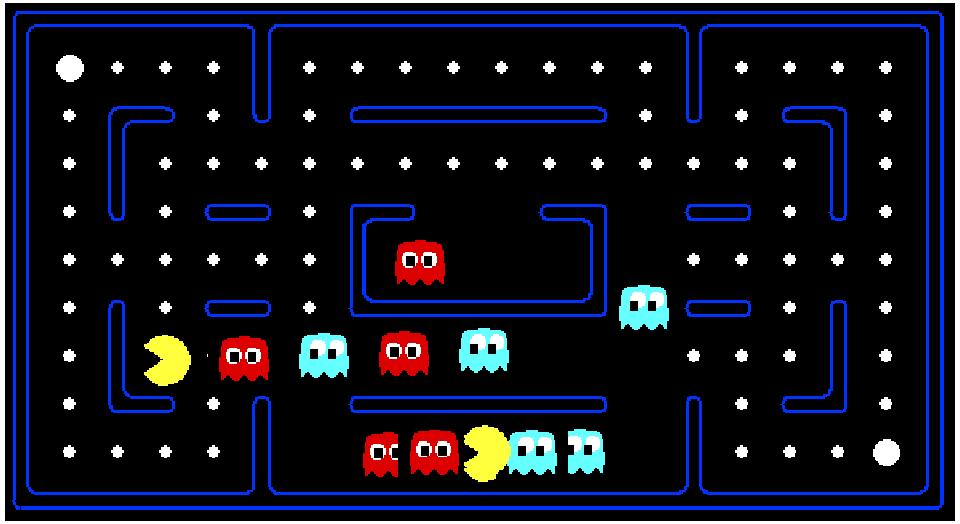


- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

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

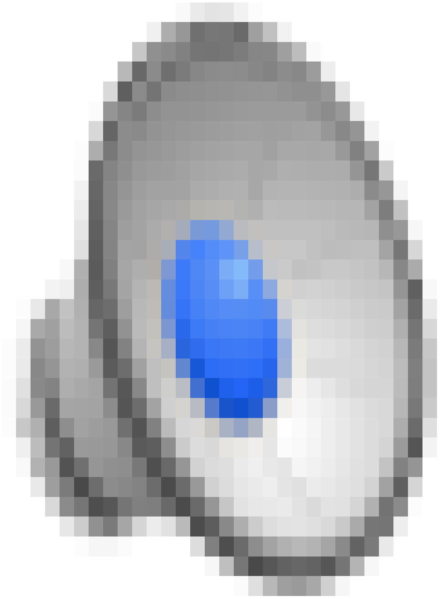
- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Evaluation for Pacman

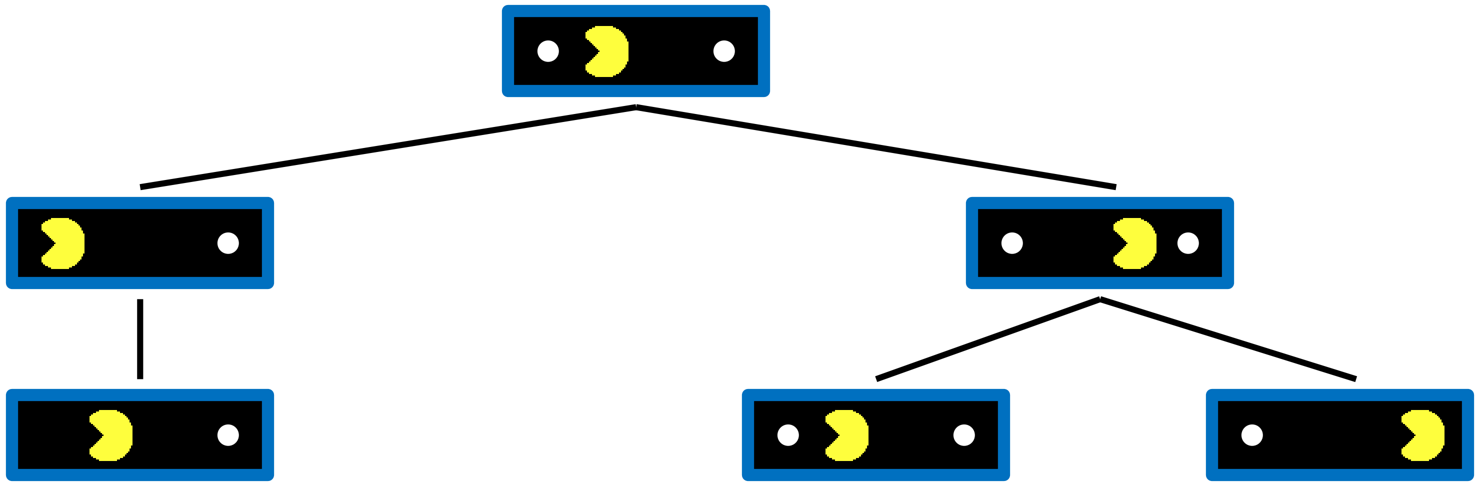


[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function), smart ghosts coordinate (L6D6,7,8,10)]

Video of Demo Thrashing (d=2)

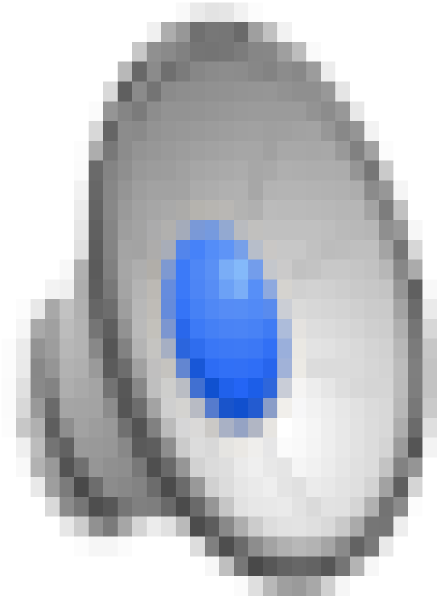


Why Pacman Starves

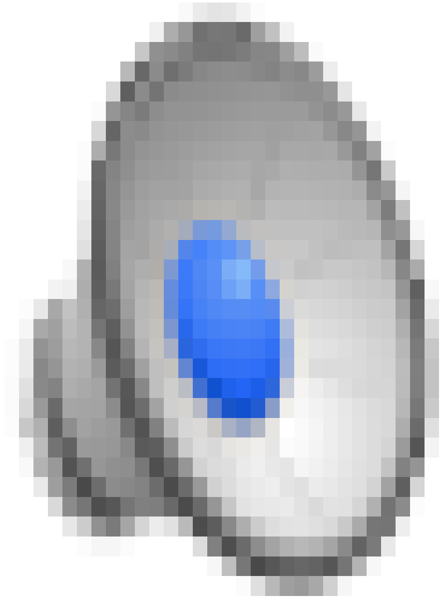


- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting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

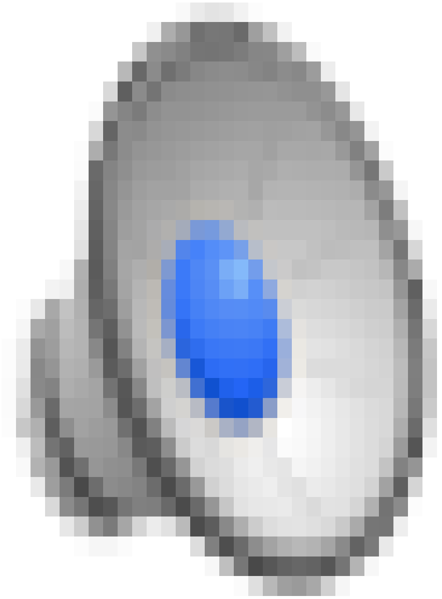
Video of Demo Thrashing -- Fixed (d=2)



Video of Demo Smart Ghosts (Coordination)



Video of Demo Smart Ghosts (Coordination) - Zoomed In



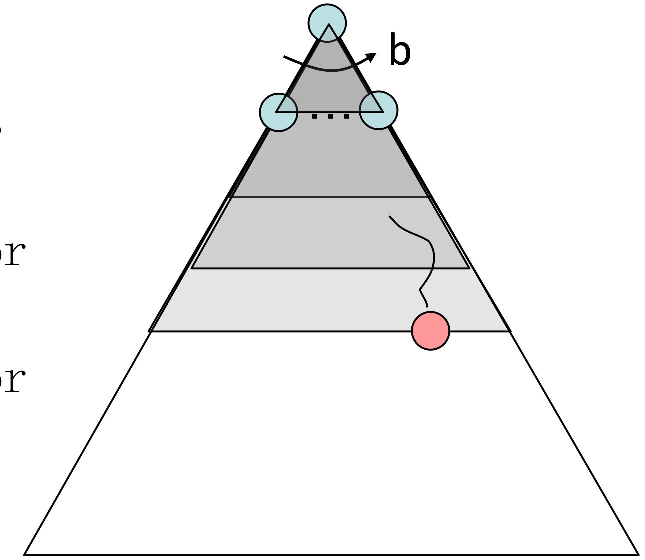
Next Time: Uncertainty!

Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If "1" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.

….and so on.



Why do we want to do this for multiplayer games?

Note: wrongness of eval functions matters less and less the deeper the search goes!