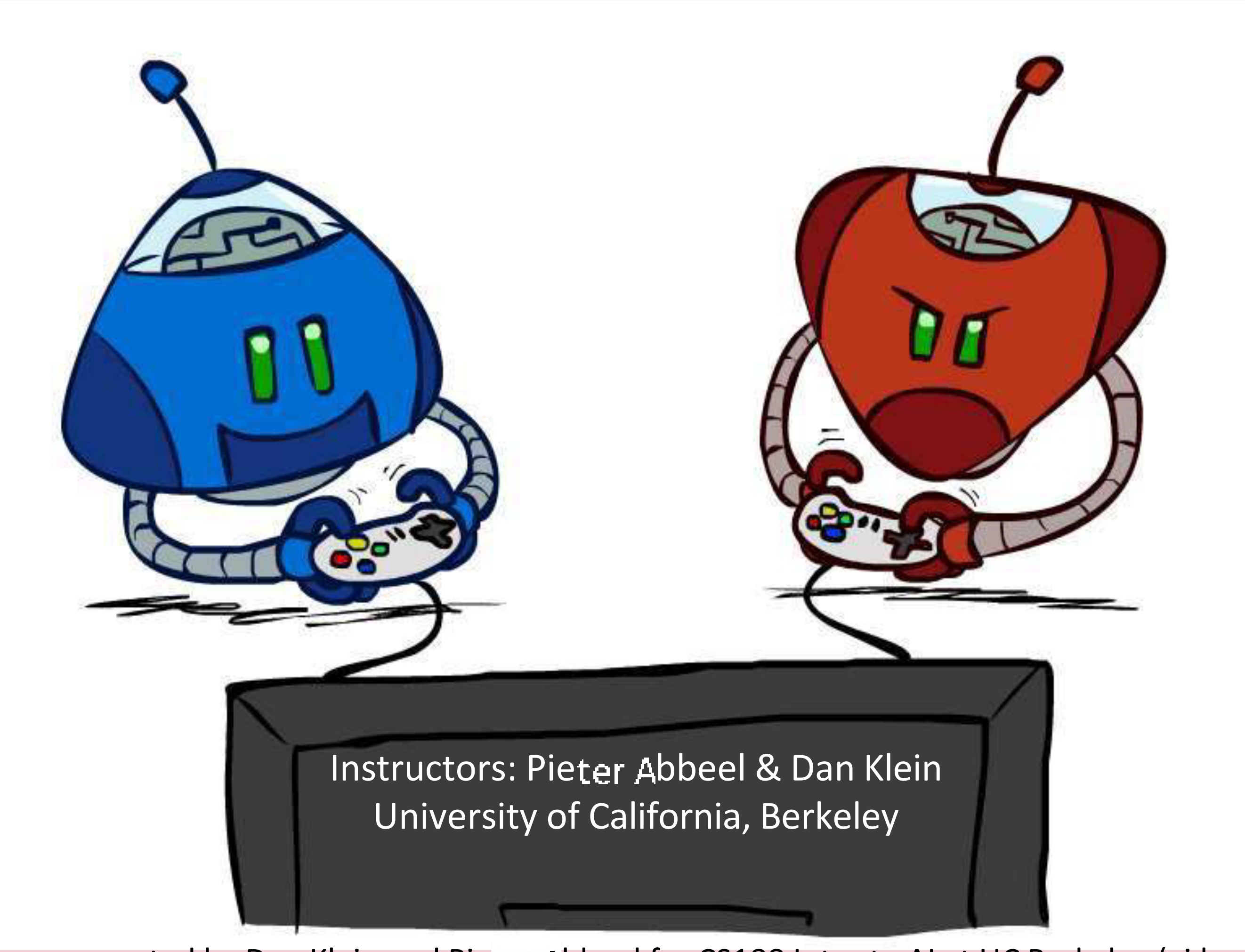
# CS 188: Artificial Intelligence

## Adversarial Search



Instructors: Pieter Abbeel & Dan Klein
University of California, Berkeley

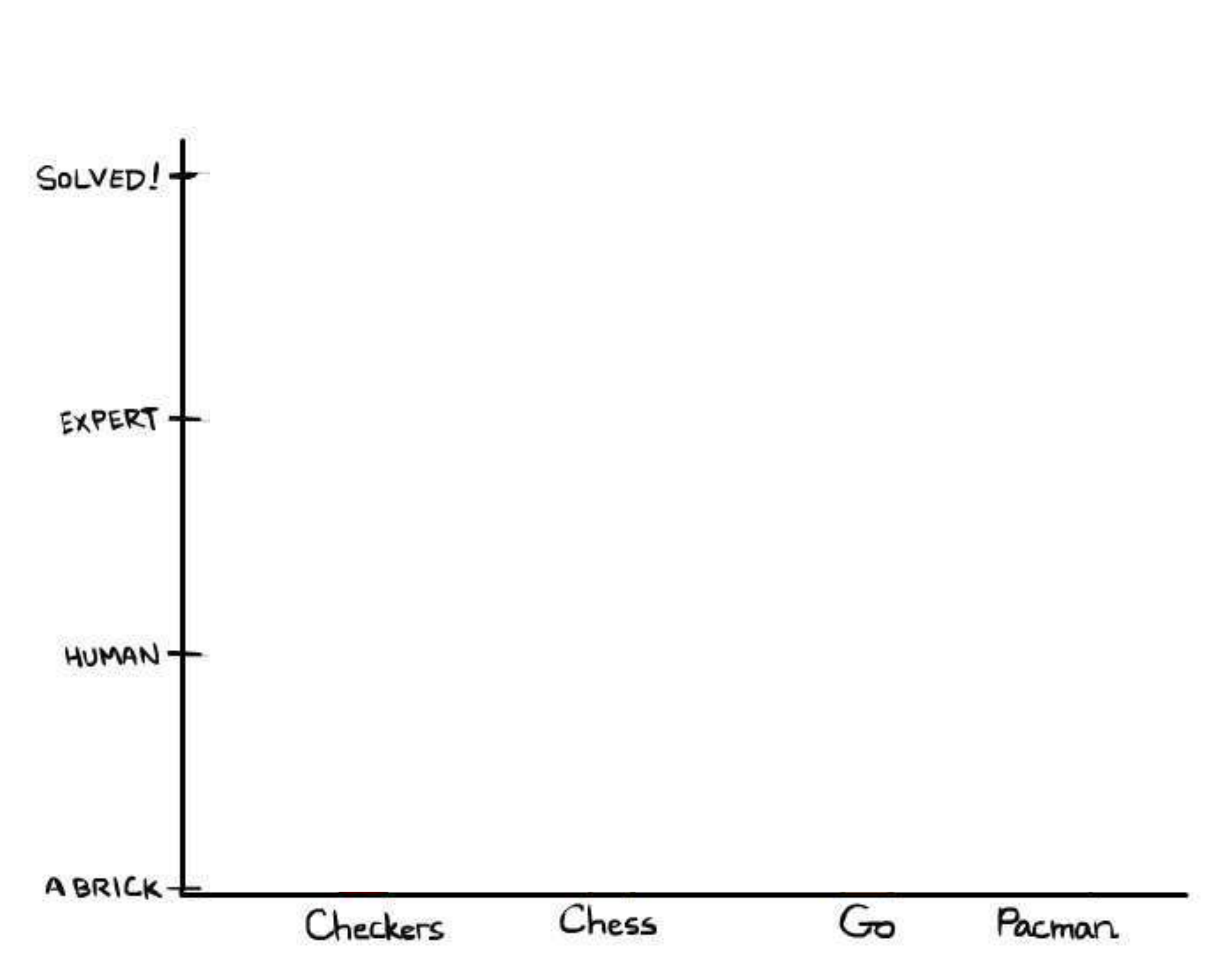
[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).]

# 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.



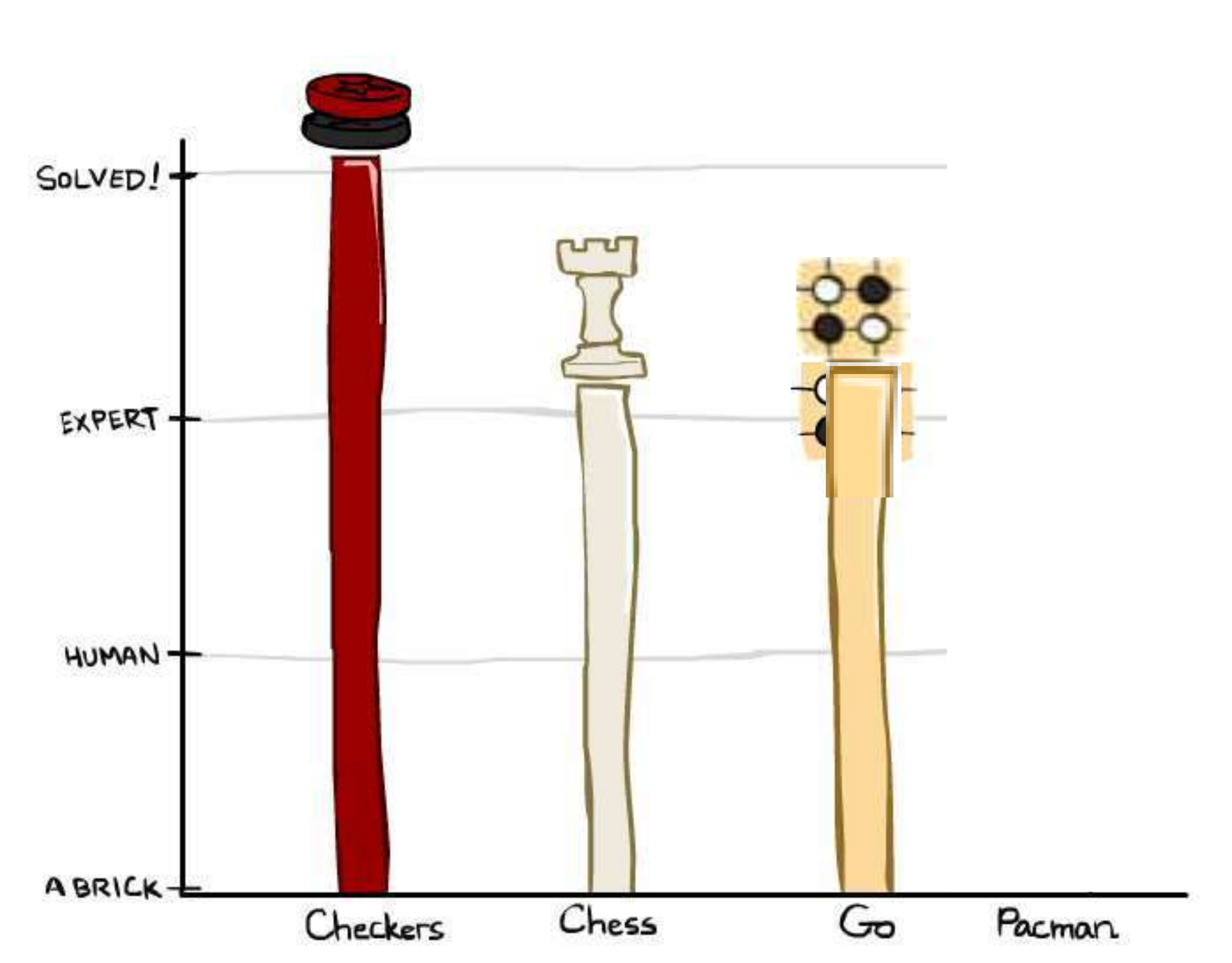
# Game Playing State-of-the-Art

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

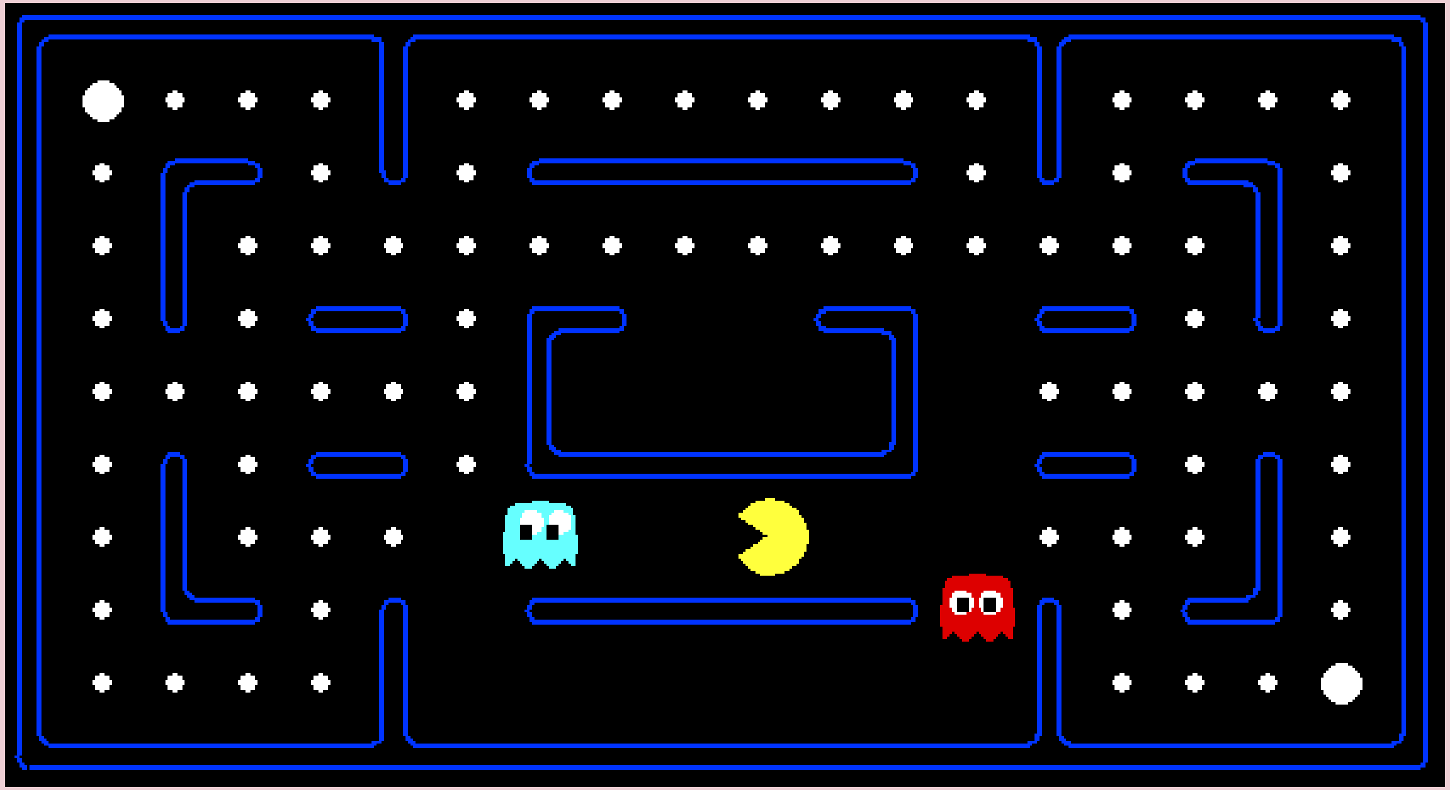
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

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

- **Pacman**



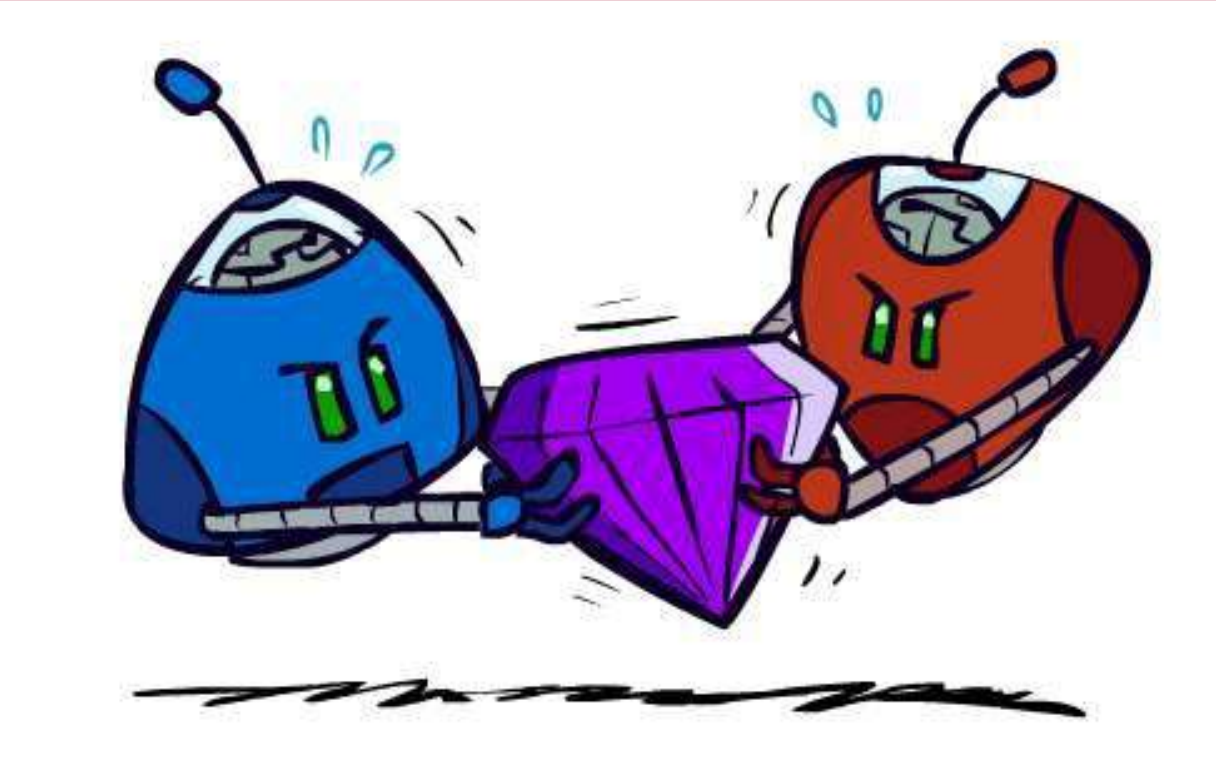
# Behavior from Computation



[Demo: mystery pacman (L6D1)]

# Video of Demo Mystery Pacman

# Adversarial Games



# Types of Games

- Many different kinds of games!

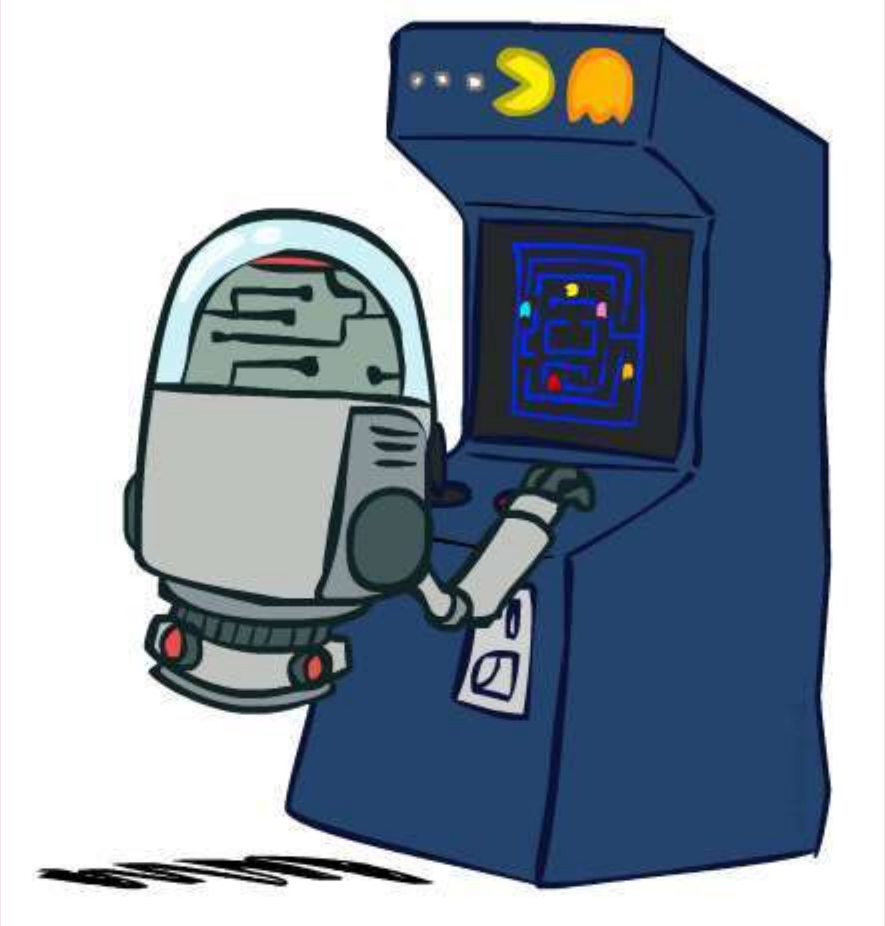
- Axes:
- Deterministic or stochastic?
- One, two, or more players?
- Zero sum?
- Perfect information (can you see the state)?

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

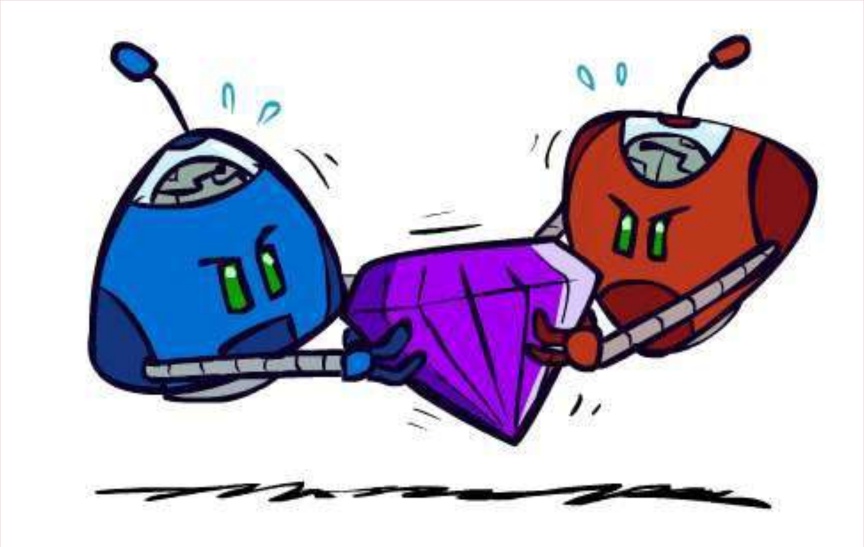
# Deterministic Games

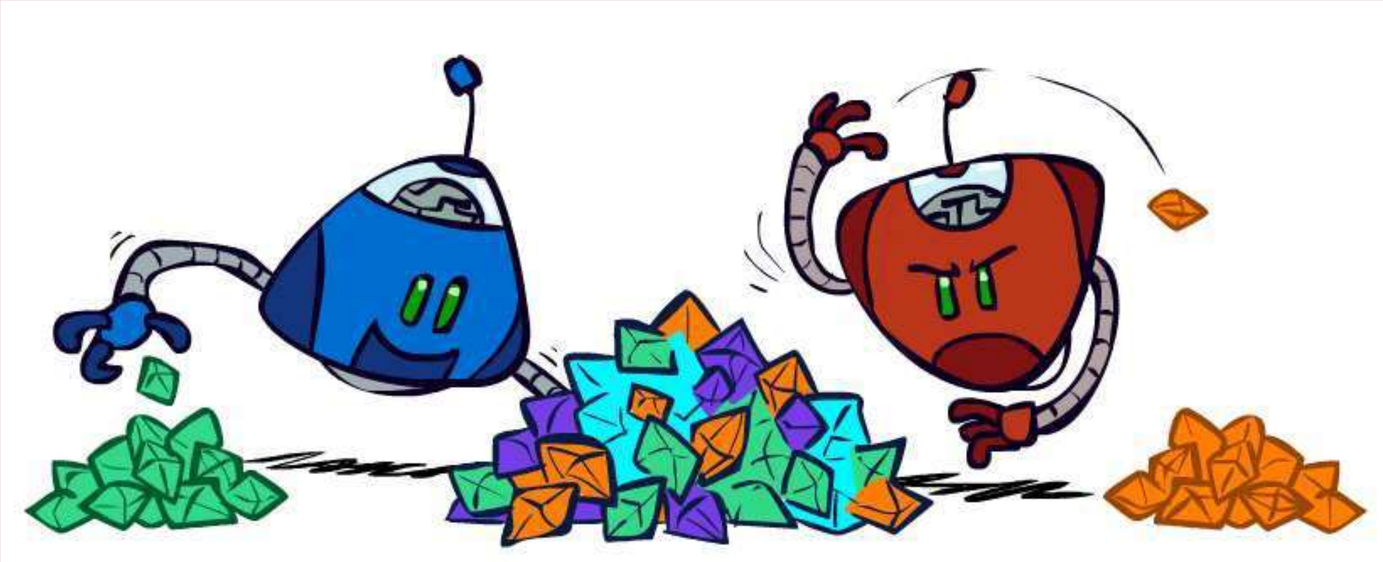
- Many possible formalizations, one is:
  - States: S (start at $s_0$)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: SxA $\rightarrow$ S
  - Terminal Test: S $\rightarrow$ {t,f}
  - Terminal Utilities: SxP $\rightarrow$ R

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



# Zero-Sum Games

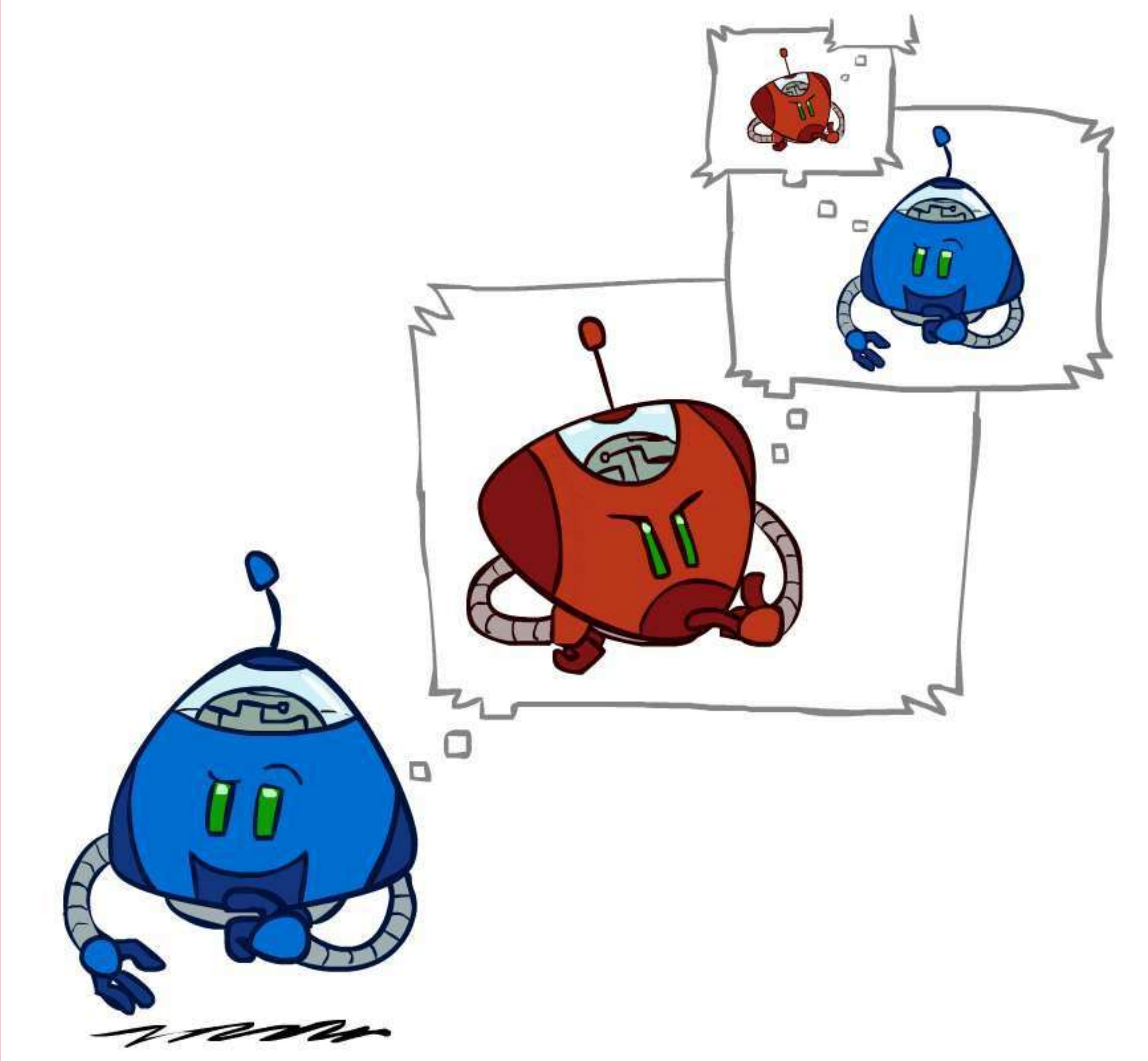




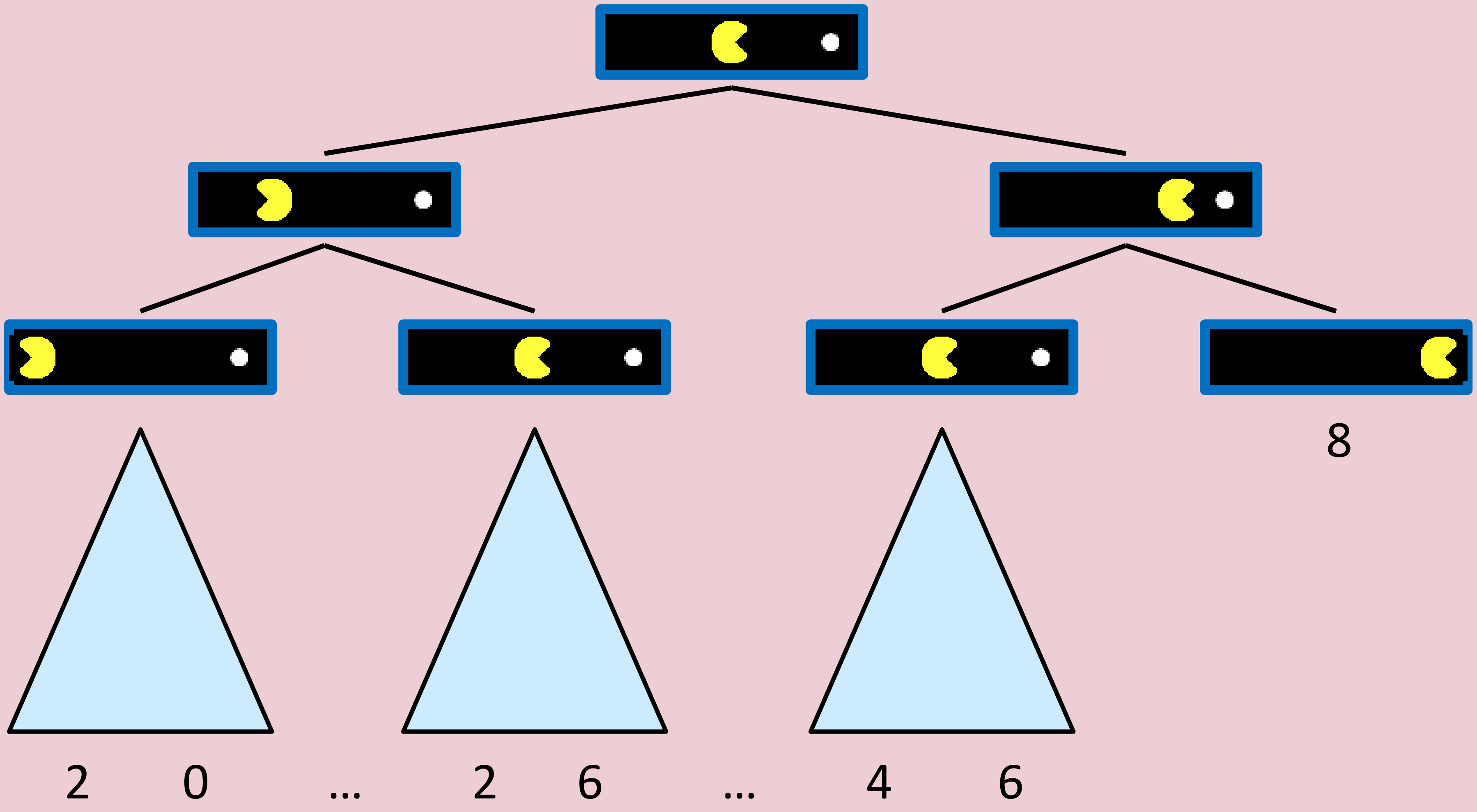
- Zero-Sum Games
- Agents have opposite utilities (values on es)
- Lets us think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition

- General Games
- Agents have independent utilities (values on es)
- Cooperation, indifference, competition, and more are all possible
- More later on non-zero-sum games

# Adversarial Search



# Single-Agent Trees

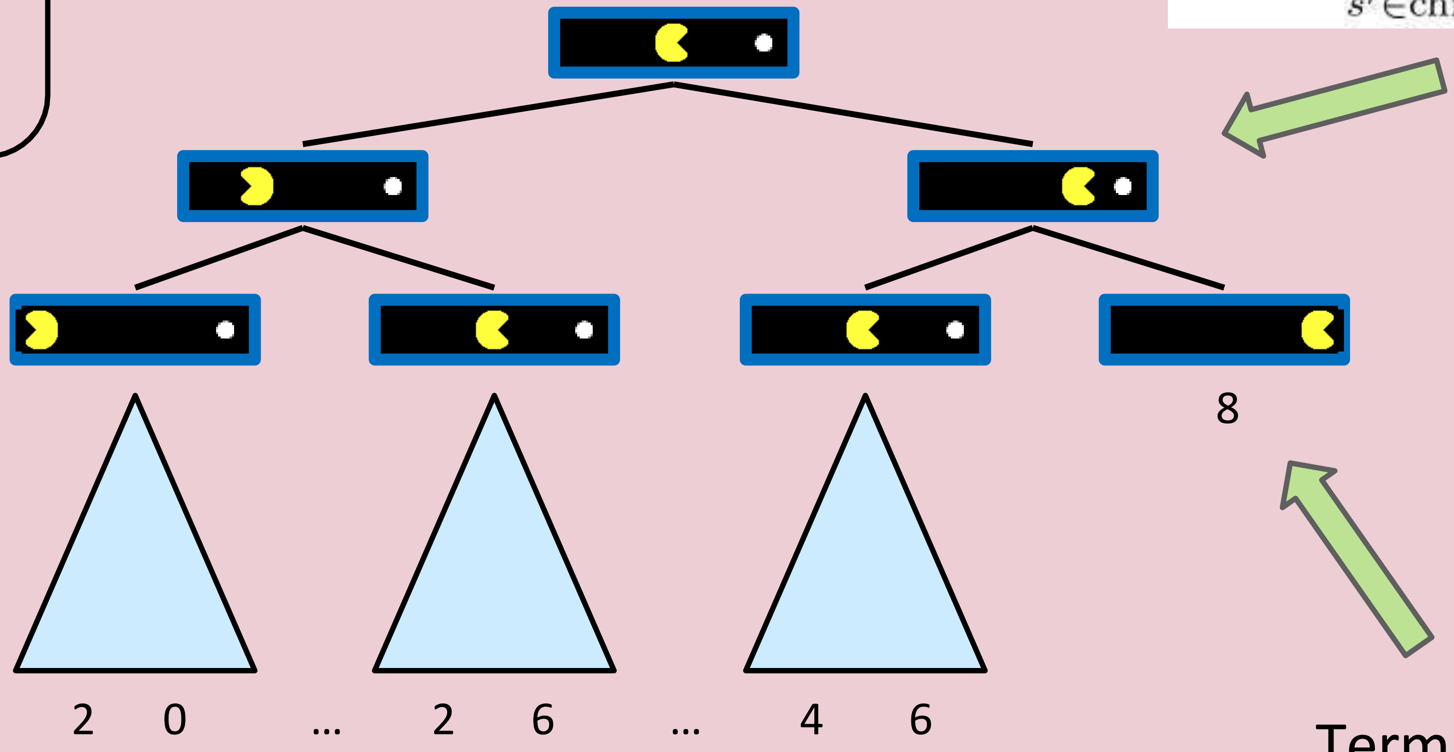


# Value of a State

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

Non-Terminal States:

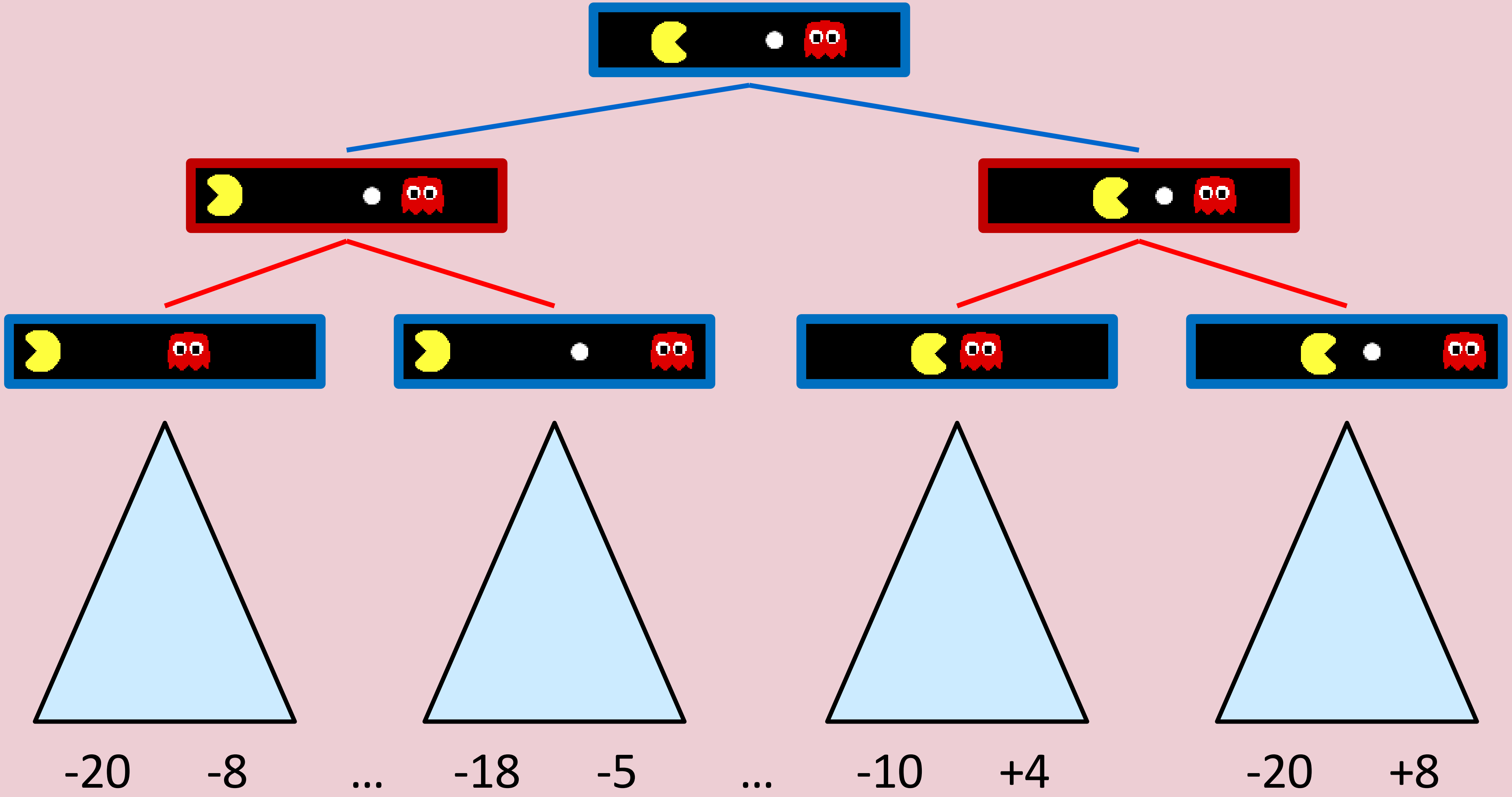
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$




Terminal States:

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

# Adversarial Game Trees



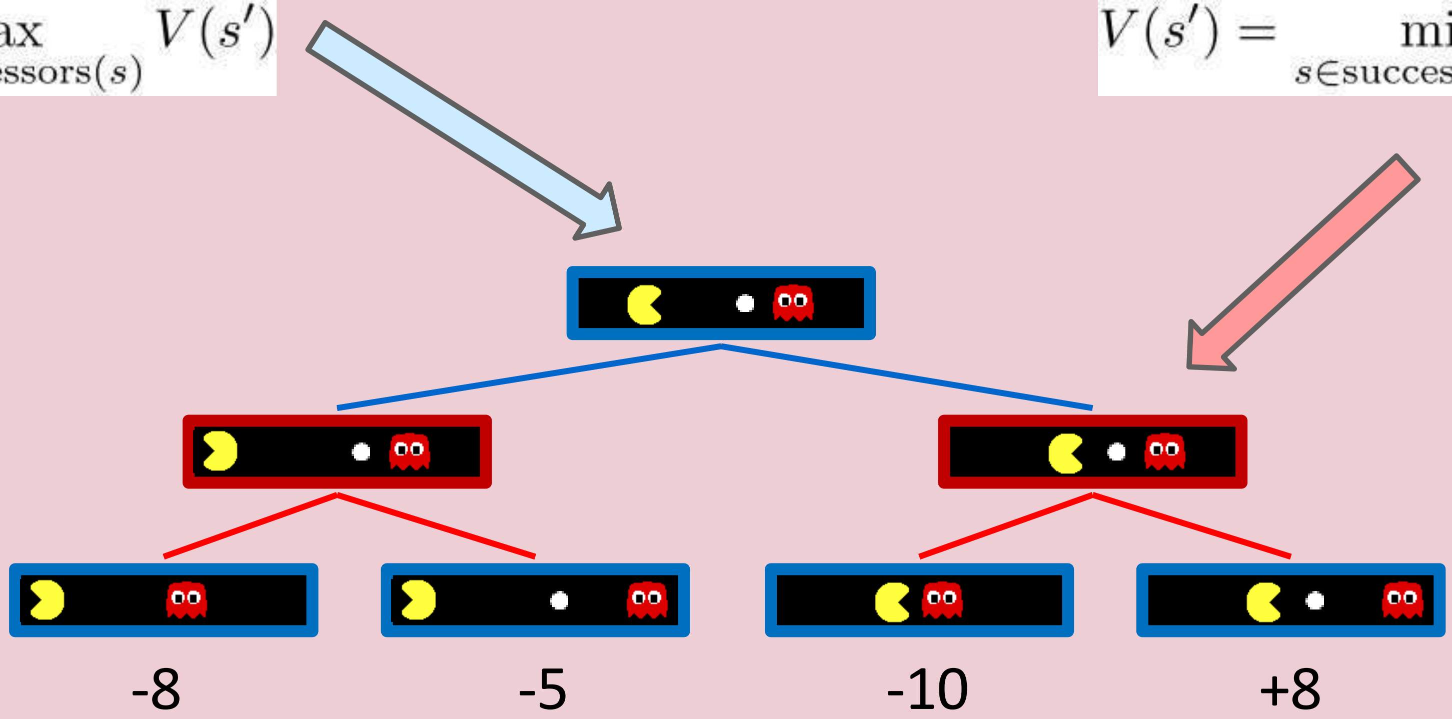
# Minimax Values

States Under Agent’s Control:

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

States Under Opponent’s Control:

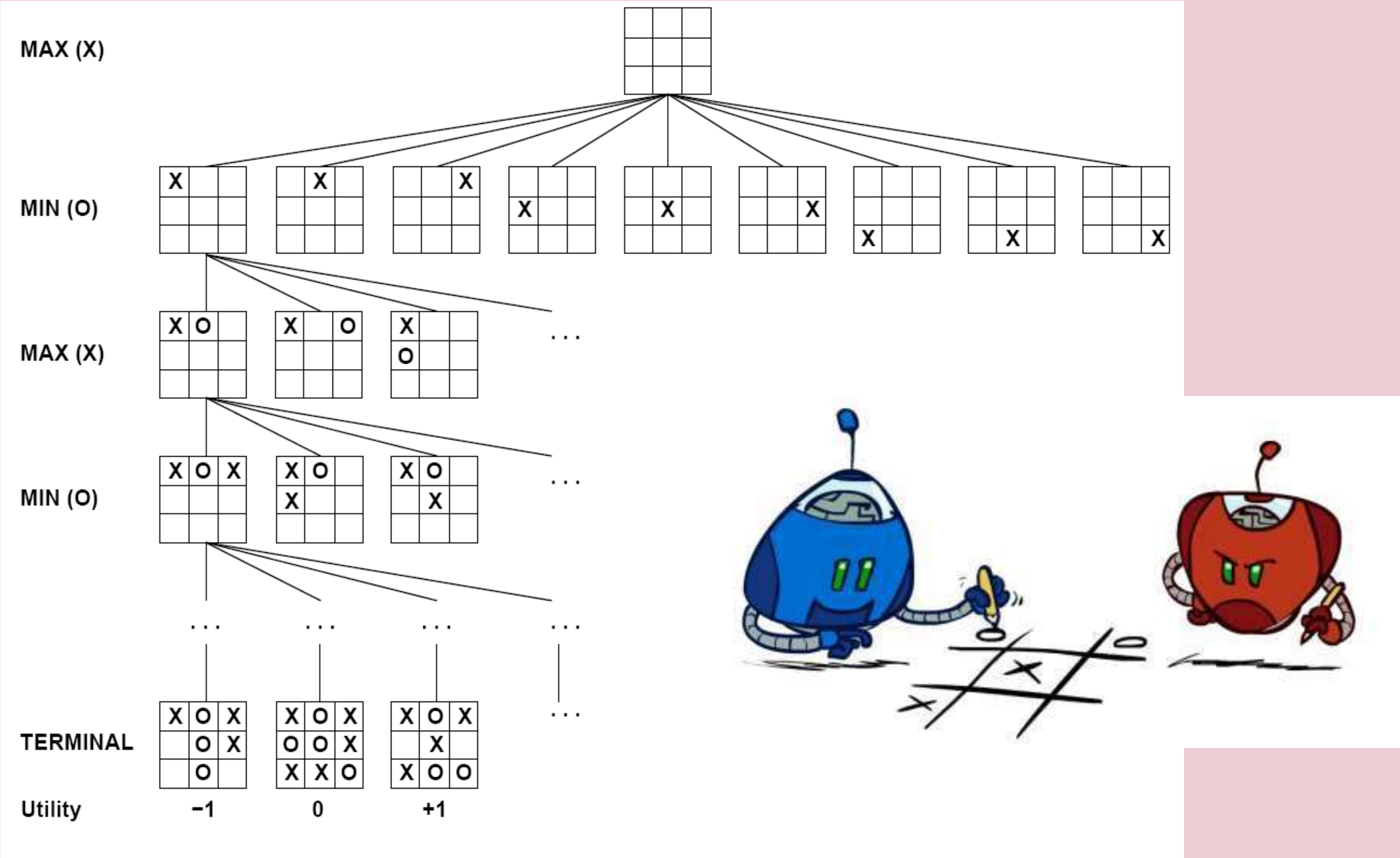
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$




Terminal States:

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

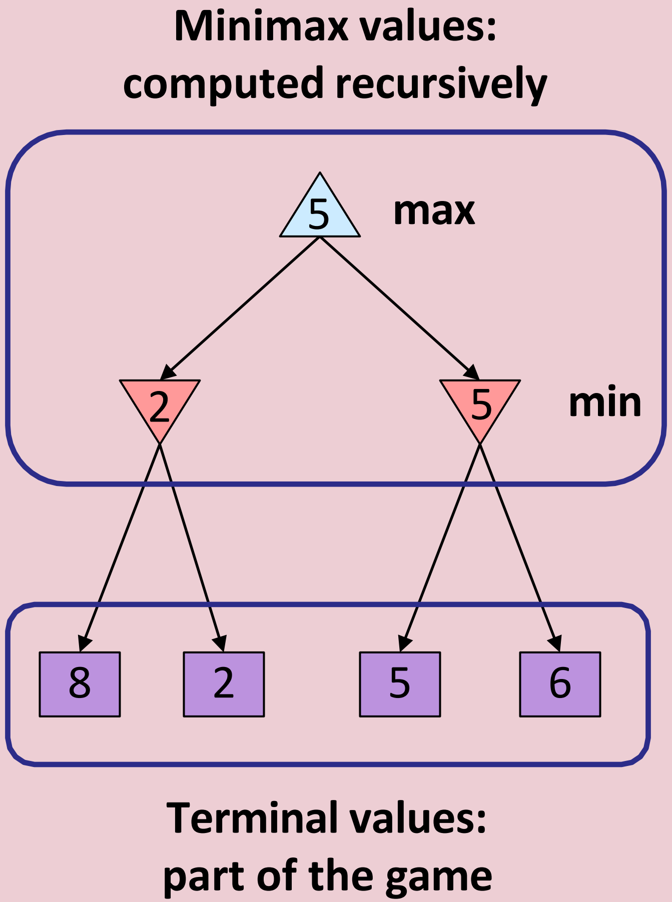
# Tic-Tac-Toe Game Tree



# 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
```

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

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

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

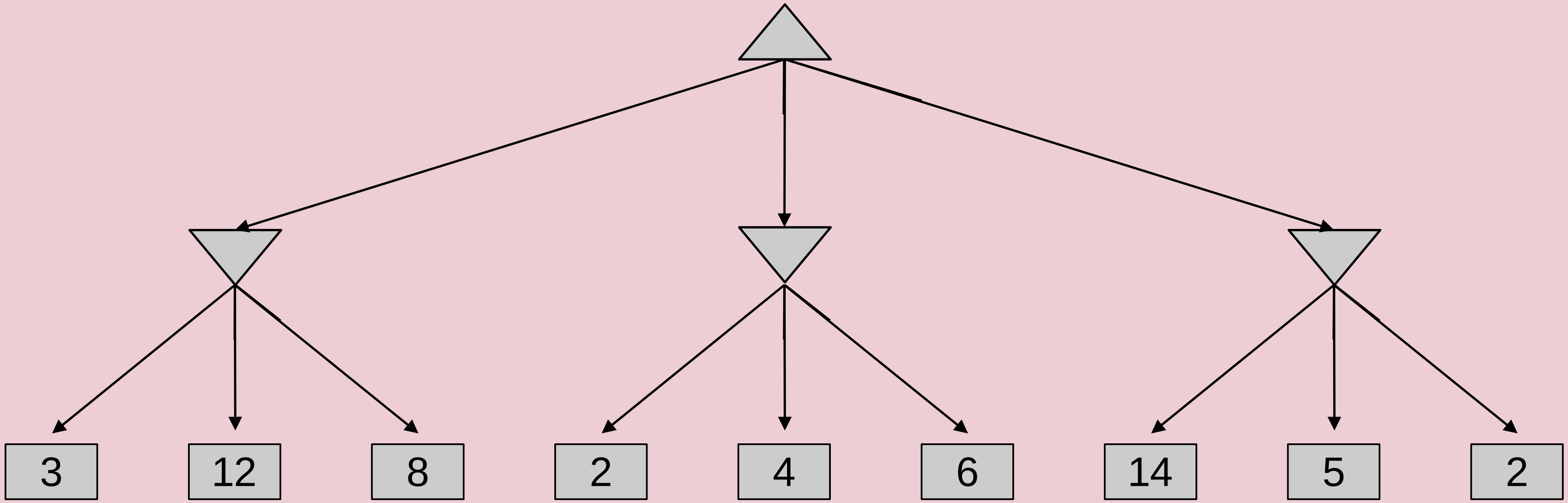
# Minimax Implementation (Dispatch)

```
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)
```

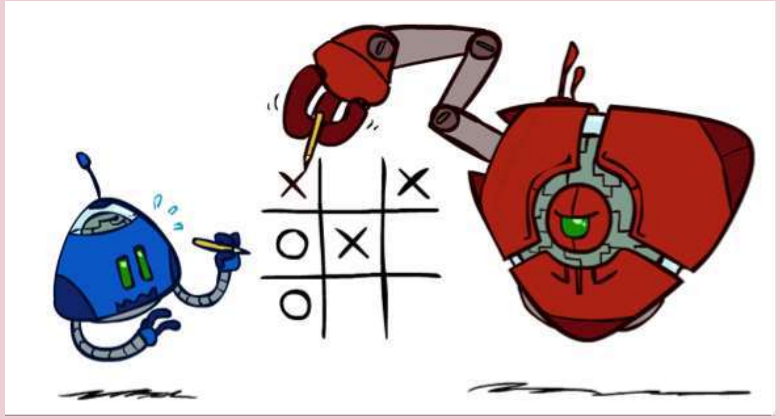
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

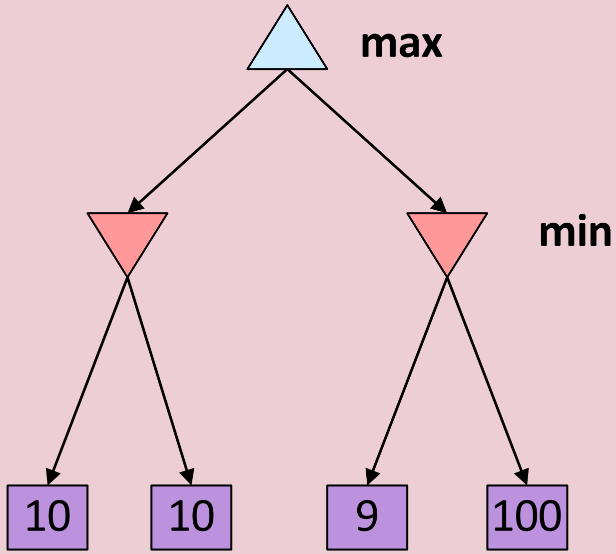
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

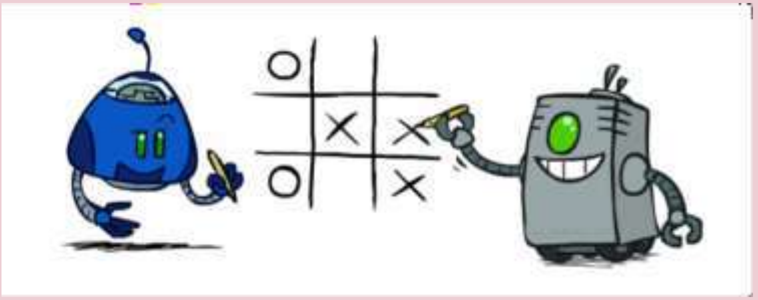
# Minimax Example



# Minimax Properties





Optimal against a perfect player. Otherwise?

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

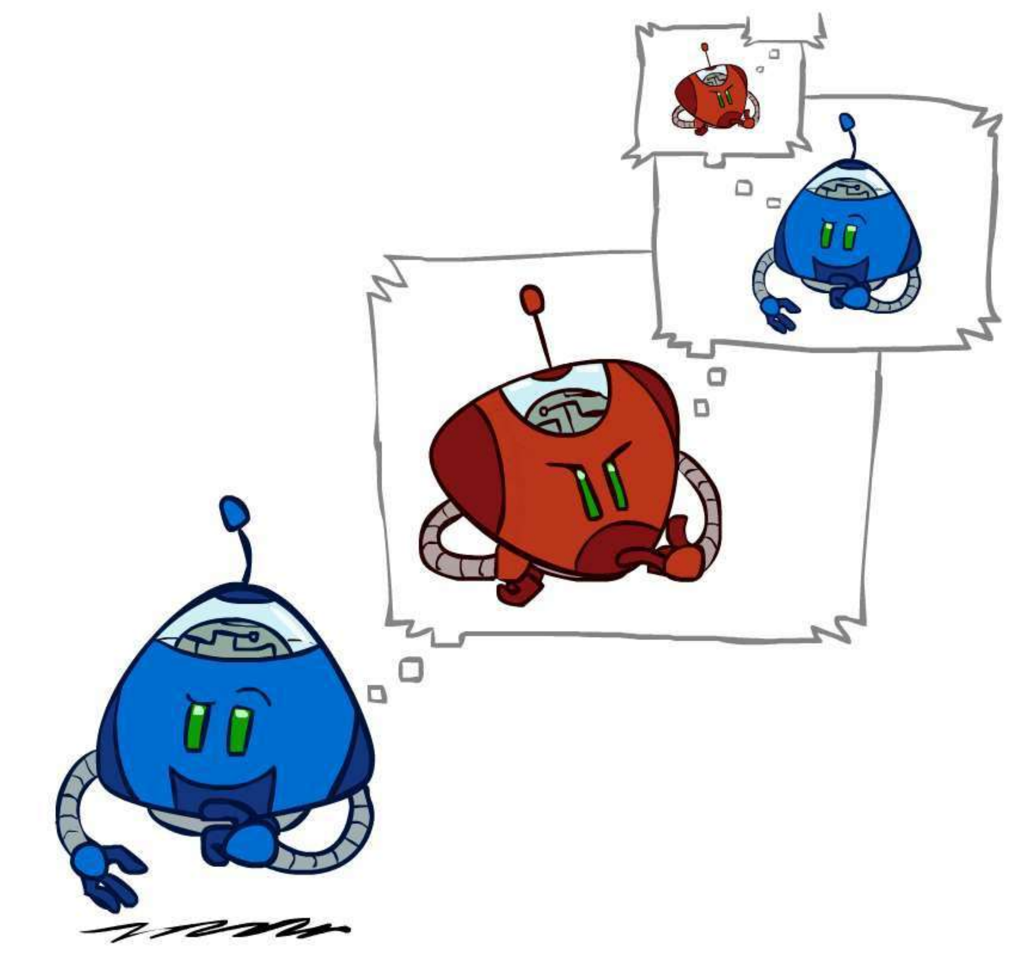
# Video of Demo Min vs. Exp (Min)

# 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