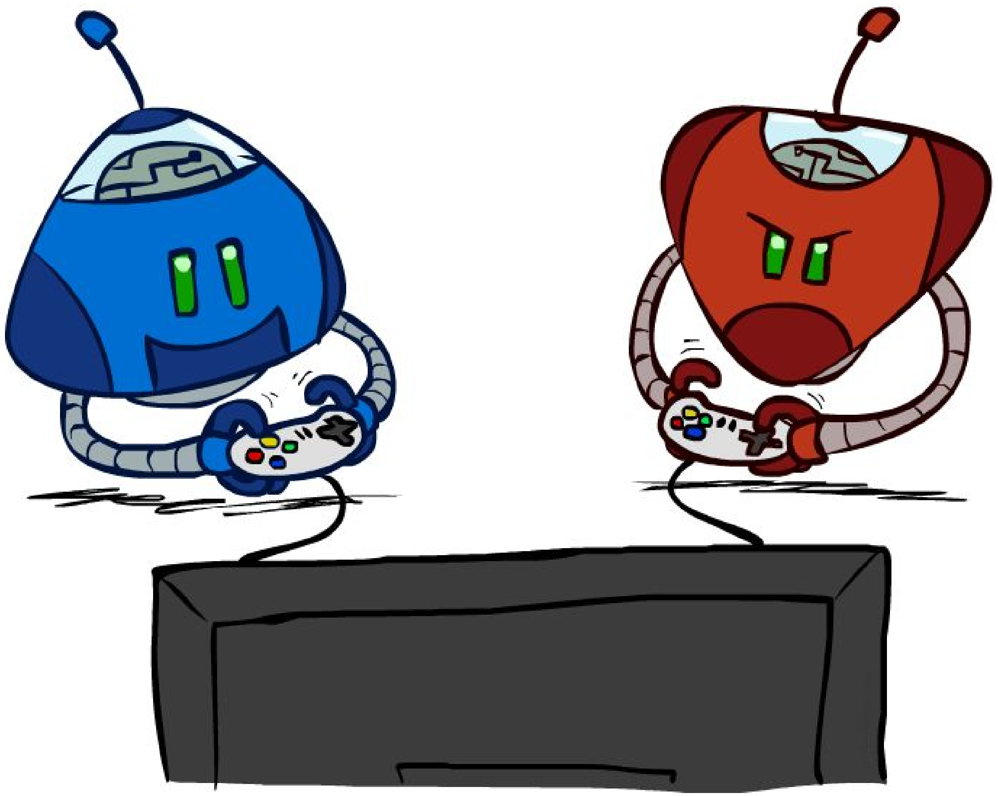
Adversarial Search



These slides are based on the slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley - http://ai.berkeley.edu. The artwork is by Ketrina Yim.

Academic Integrity

It is NOT OK:

✗ to share code, or copy code from fellow students or from the web.

➢ Infractions will be detected and will lead to an automatic 0 on the given assignment or test.

➢ Infractions will also be reported to the Office of Student Conduct and Ethical Development.

Academic Integrity

You copy and paste code from the web or from another student

⇒ Automatic 0 on the assignment and report to the Office of Student Conduct and Ethical Development.

You copy code from the web or from another student but you change some variable names.

⇒ Automatic 0 on the assignment and report to the Office of Student Conduct and Ethical Development.

Academic Integrity

You see someone else's code (from a friend or on the web)

To avoid an academic integrity violation, you must:

1. Cite the source that you saw - web address or friend's name
2. Rewrite the code with your own 'words' without looking at the source.
3. Comment extensively so that you explain what every line of code does.

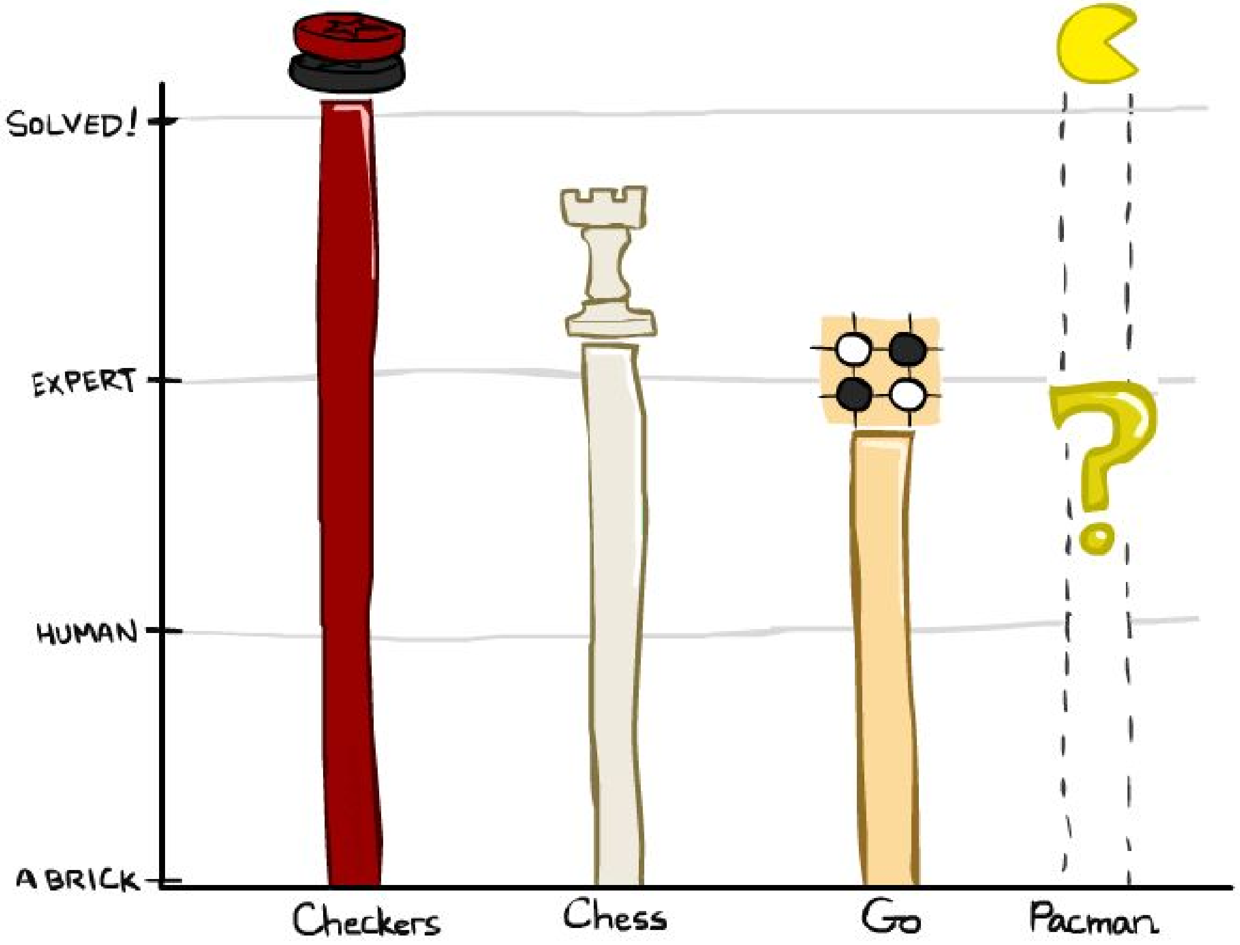
Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook used 8 piece endgame database 2007: Checkers solved!

- **Chess:** 1997: Deep Blue defeated human champion Gary Kasparov. Current programs are even better, if less historic.

- **Go:** b > 300. Oct 2015: Google DeepMind program AlphaGo defeated Fan Hui, the European Go champion. It uses a combination of machine learning and randomized tree search techniques.

- **Pacman**



Adversarial Games

Types of Games

- Many different kinds of games!
- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Fully observable?

Games

How are games different from the search problems we've seen so far?

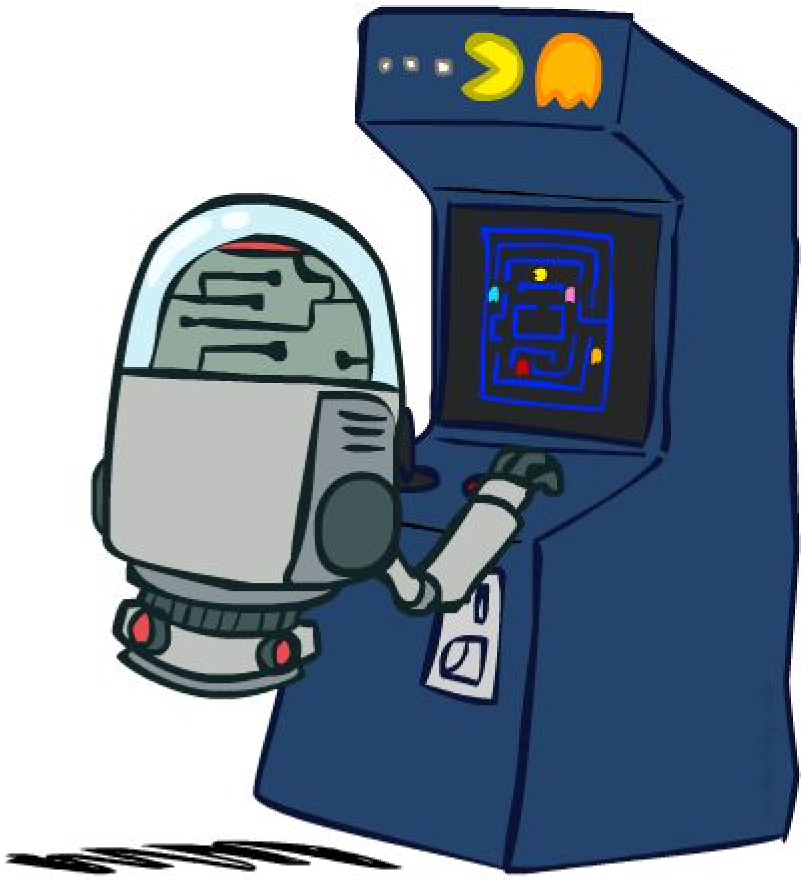
- so far we have been looking for a sequence of actions, that are guaranteed to succeed
- that does not work here because we don't control our opponent
- we need to be able to decide on a move from each state

We need algorithms for calculating a strategy (policy) which recommends a move from each state

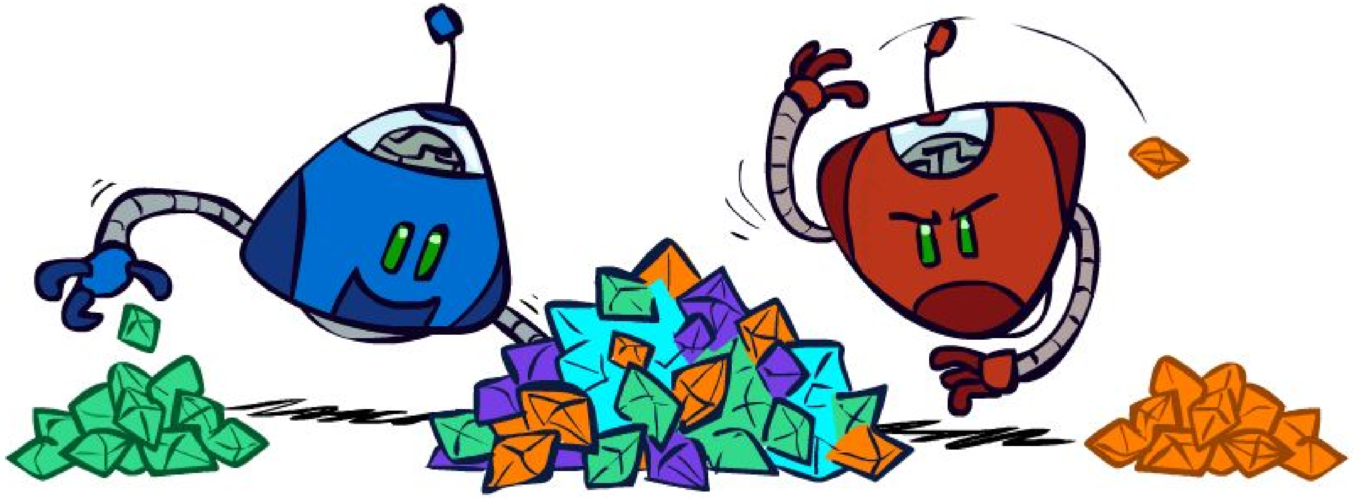
Games

- 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 (game over?): 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

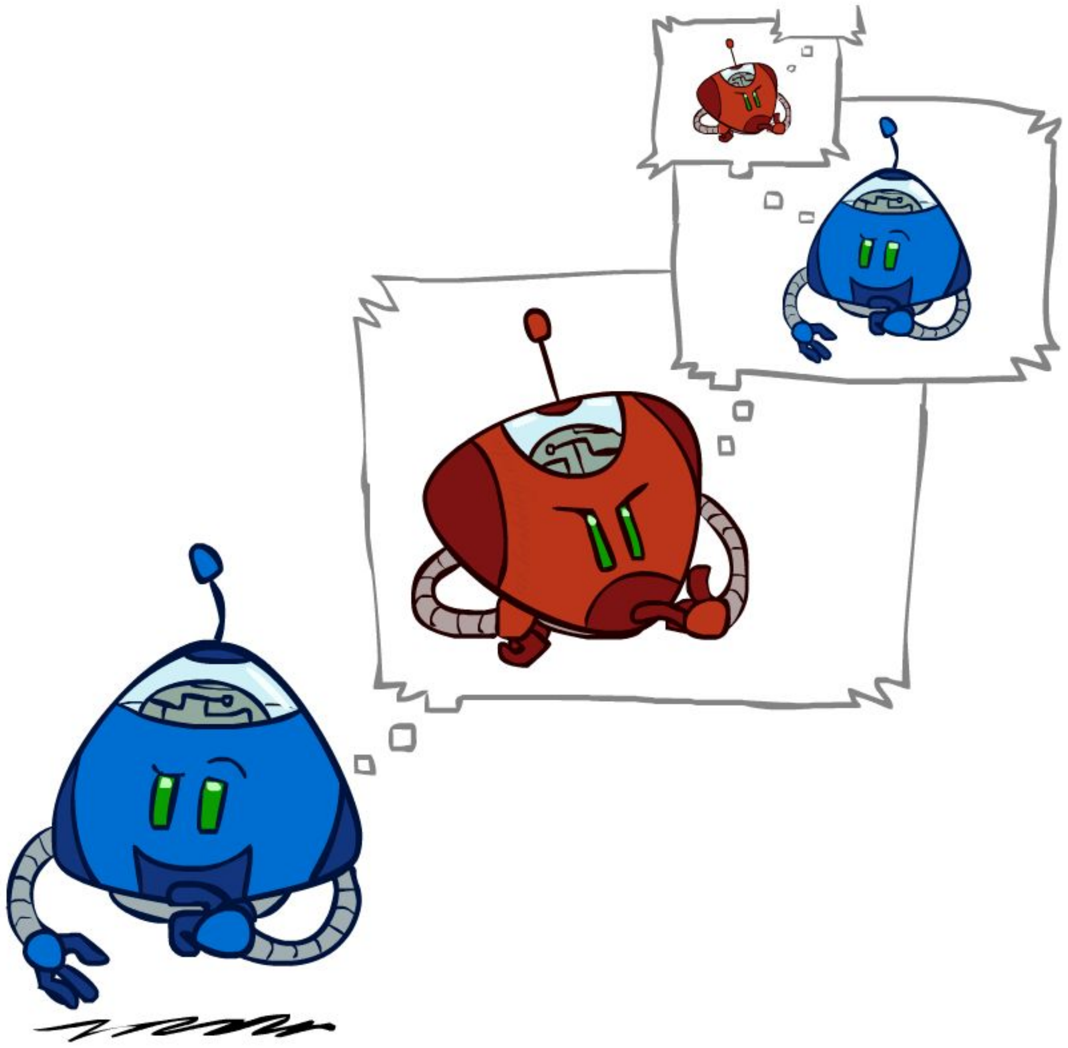
- Agents have opposite utilities (values on outcomes)
- Lets us think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition

General Games

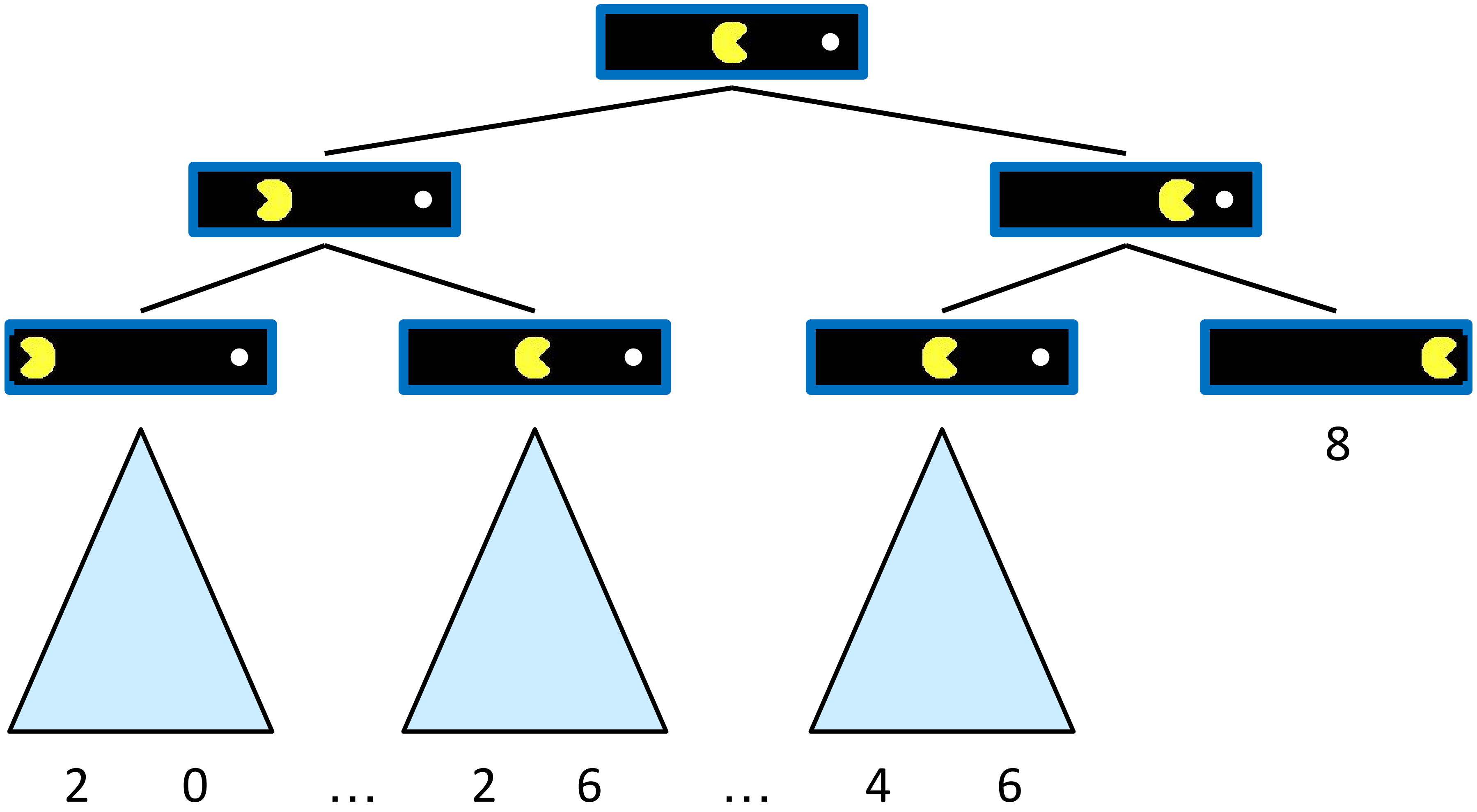
- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible

Adversarial Search

- We still need to think about the consequences of our actions.
- We also need to think about our opponent and how they will respond to our actions.



Single-Agent Trees

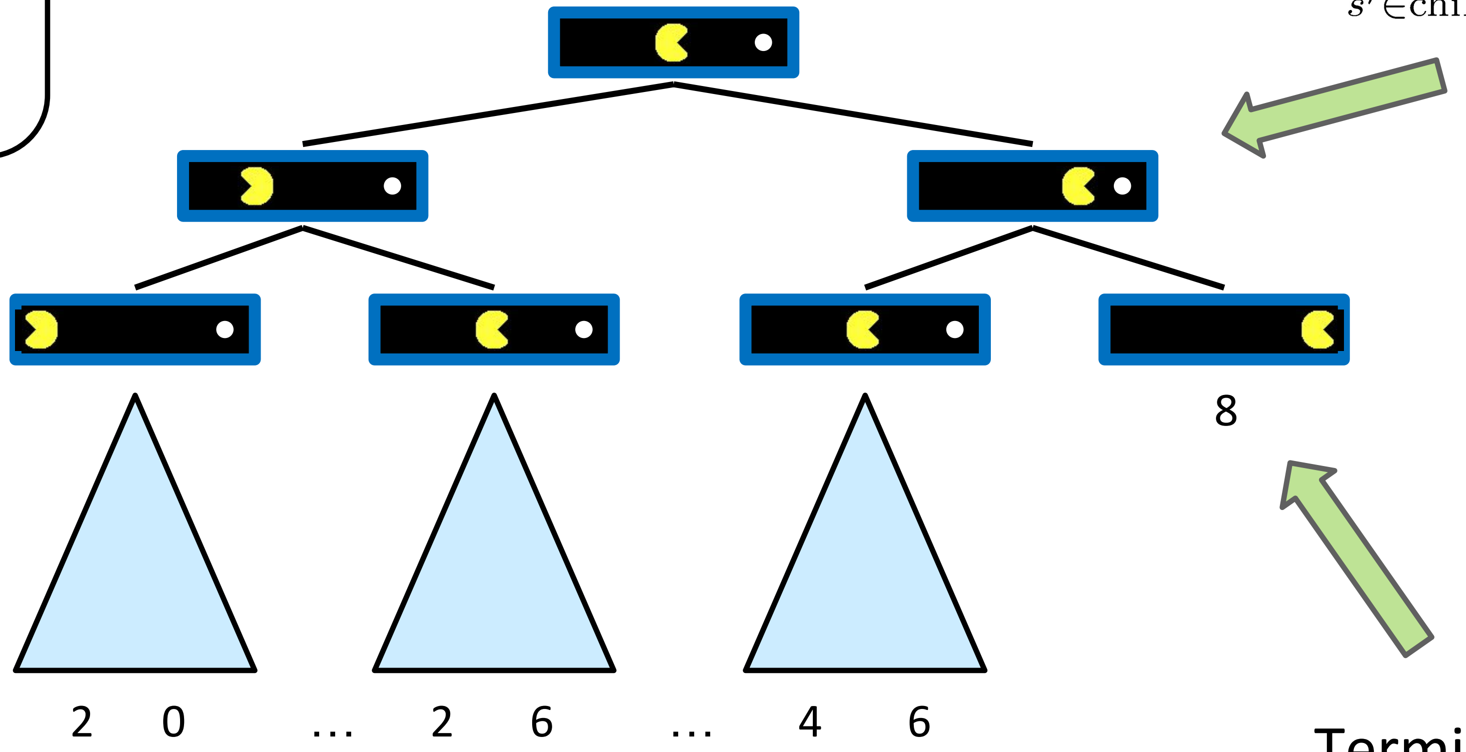


Value of a State

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

Non-Terminal States:

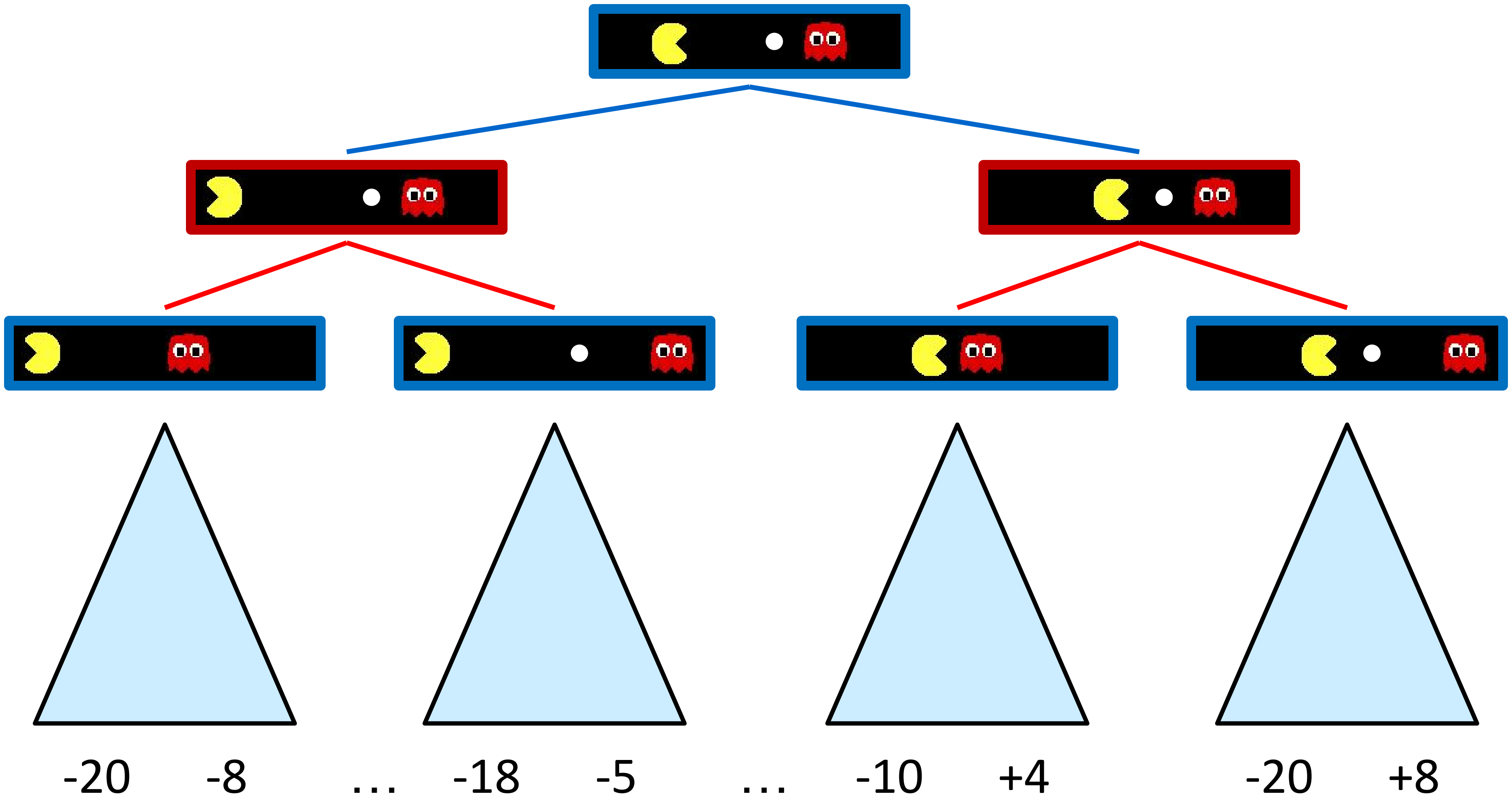
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



Terminal States:

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

Adversarial Game Trees



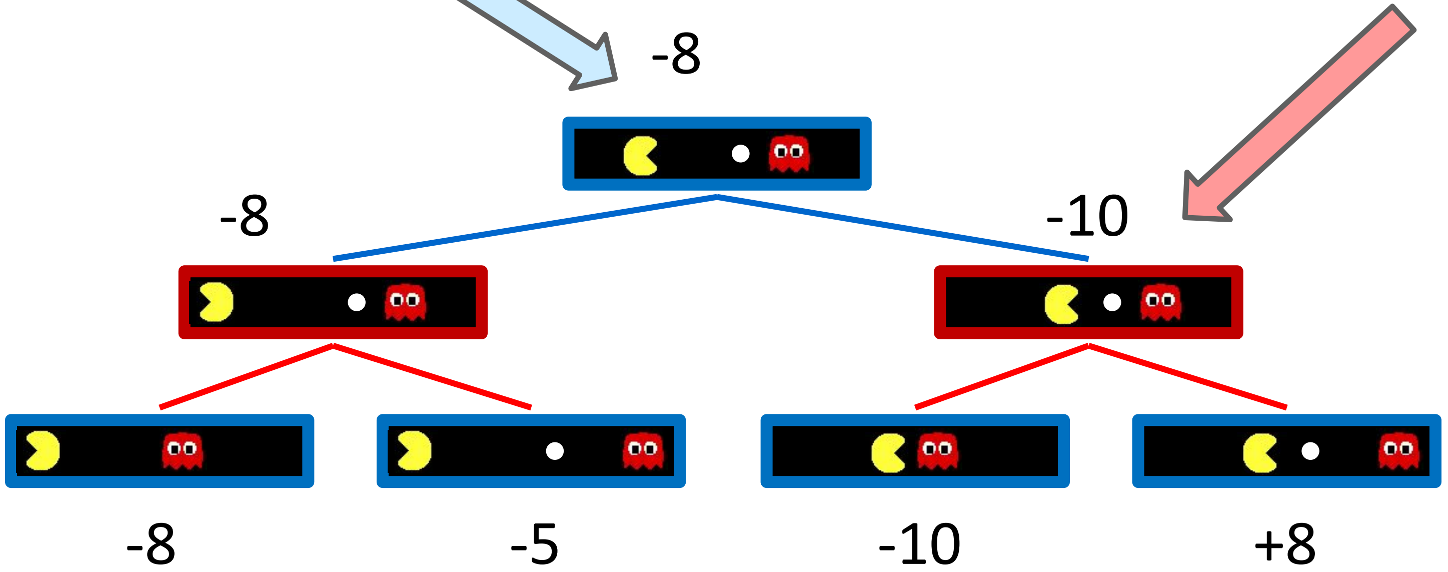
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

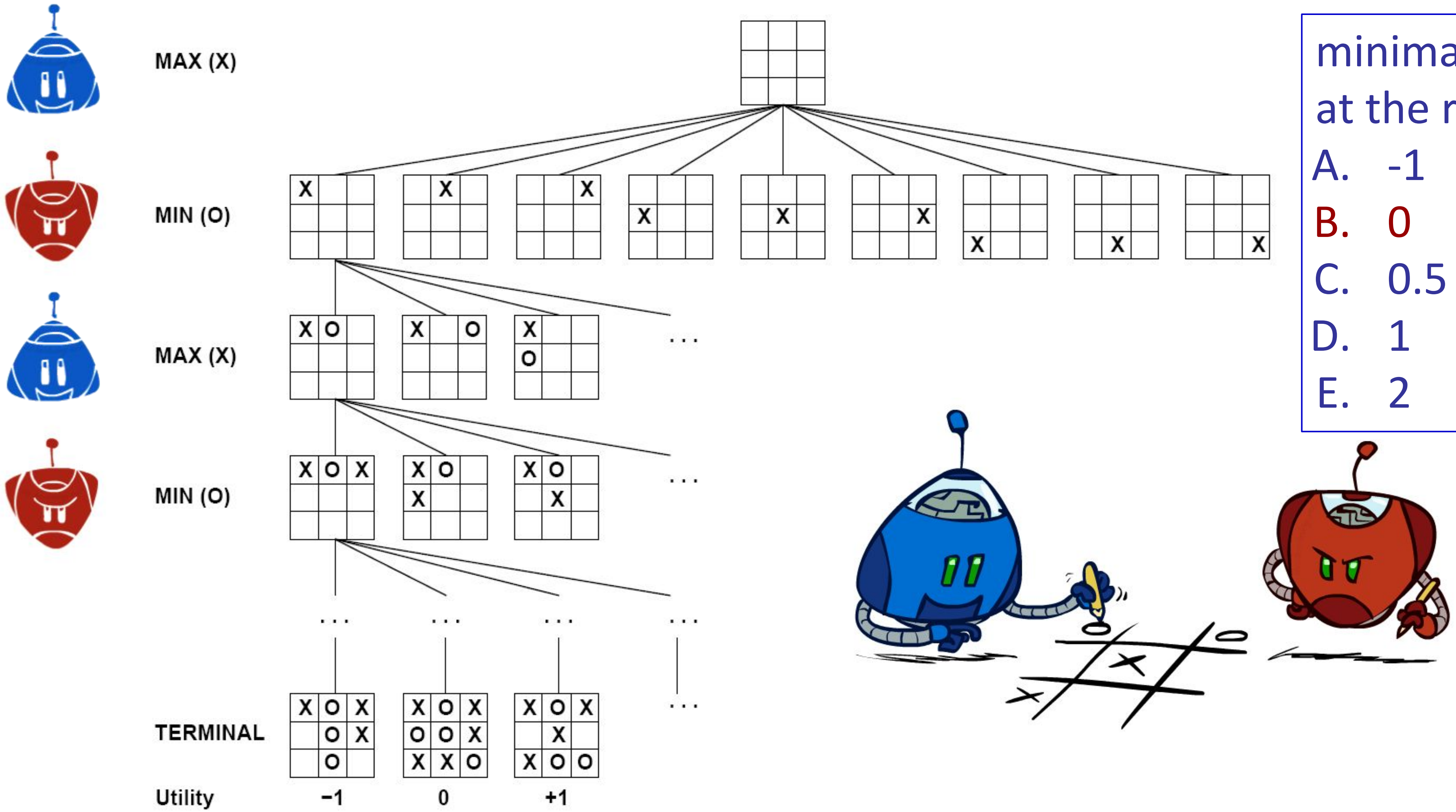
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

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

Tic-Tac-Toe Game Tree



minimax value at the root?

A. -1
B. 0
C. 0.5
D. 1
E. 2

Adversarial Search (Minimax)

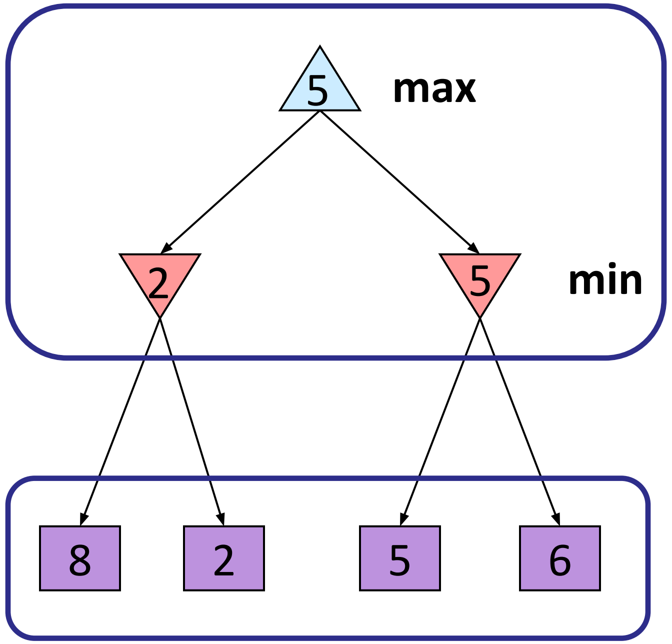
Deterministic, zero-sum games:

- 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 adversary

Minimax values: computed recursively



Terminal values: part of the game

Minimax Implementation

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

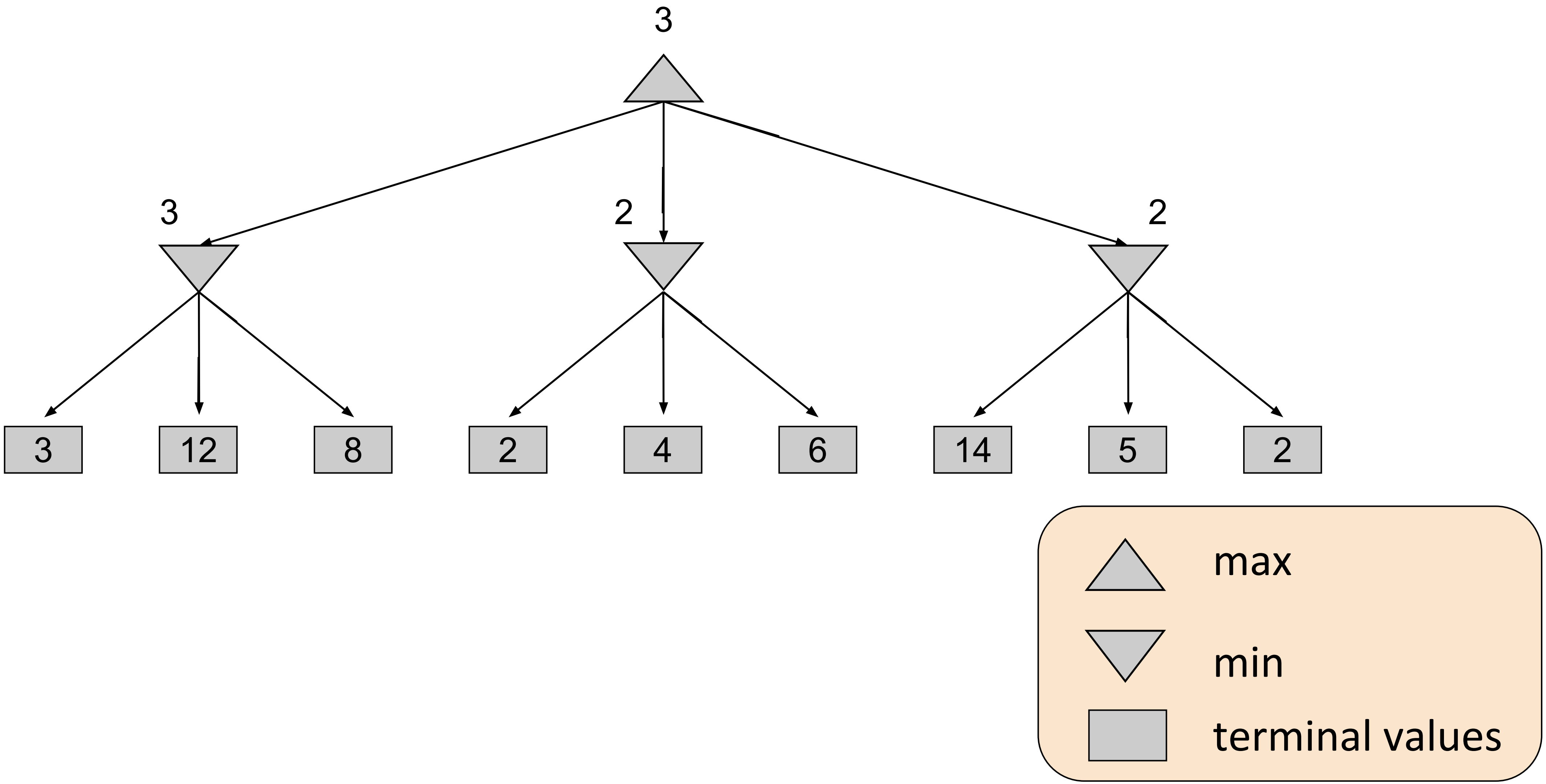
$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

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

```
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
```

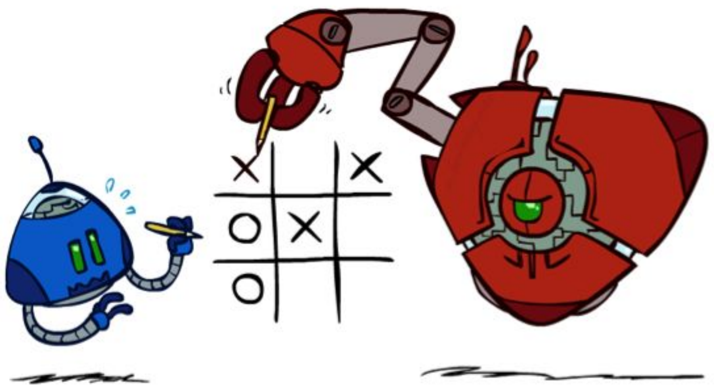
Minimax Example

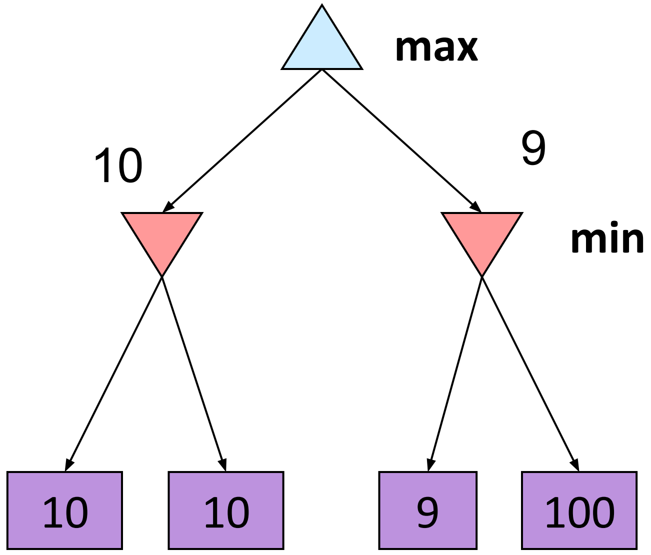


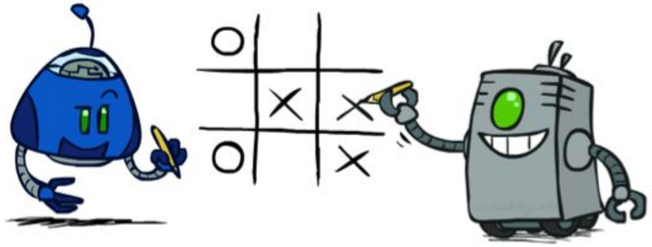
Minimax Properties

- Complete?
  - Yes, if tree is finite
- Optimal?

Minimax Properties







Optimal against a perfect player. Otherwise?