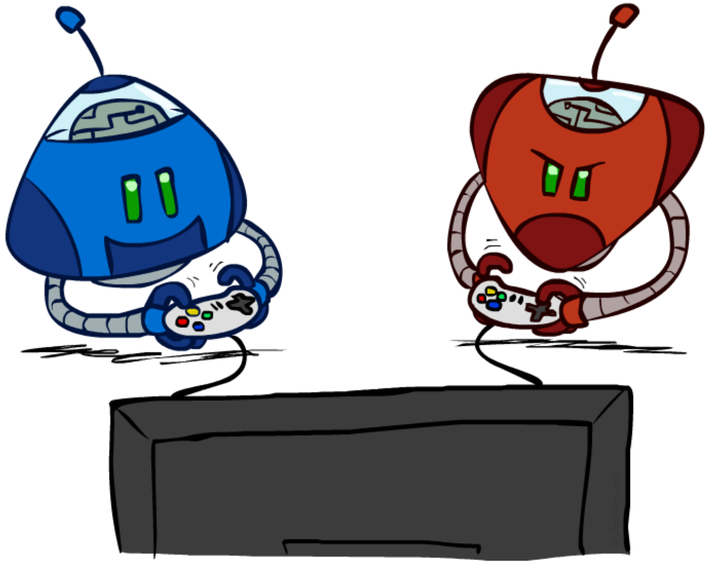
Artificial Intelligence - INFOF311

Games and adversarial search

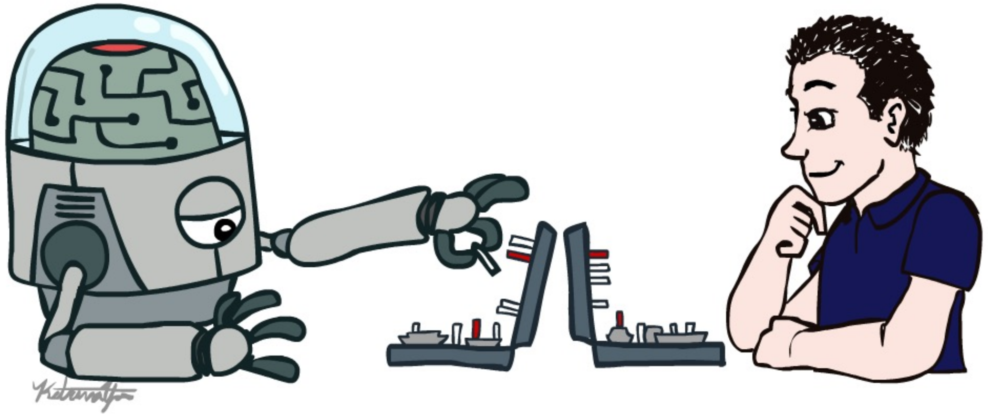


Instructor : Tom Lenaerts

Acknowledgement

We thank Stuart Russell for his generosity in allowing us to use the slide set of the UC Berkeley Course CS188, Introduction to Artificial Intelligence. These slides were created by Dan Klein, Pieter Abbeel and Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.

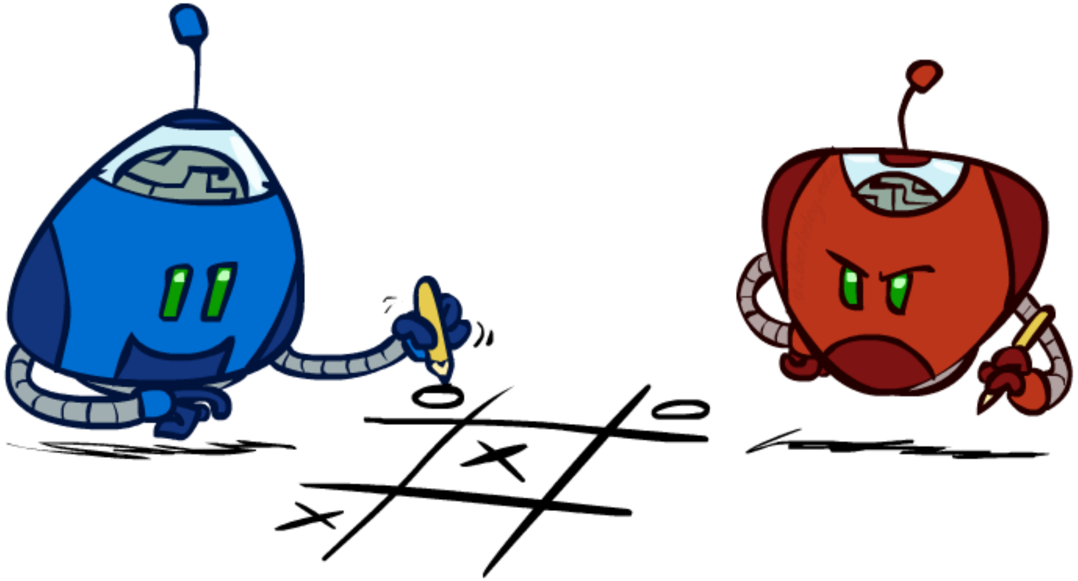
Center for
Human-Compatible
Artificial
Intelligence



The slides for INFOF311 are slightly modified versions of the slides of the spring and summer CS188 sessions in 2021 and 2022

Outline

- History / Overview
- Minimax for Zero-Sum Games
- α-β Pruning
- Finite lookahead and evaluation



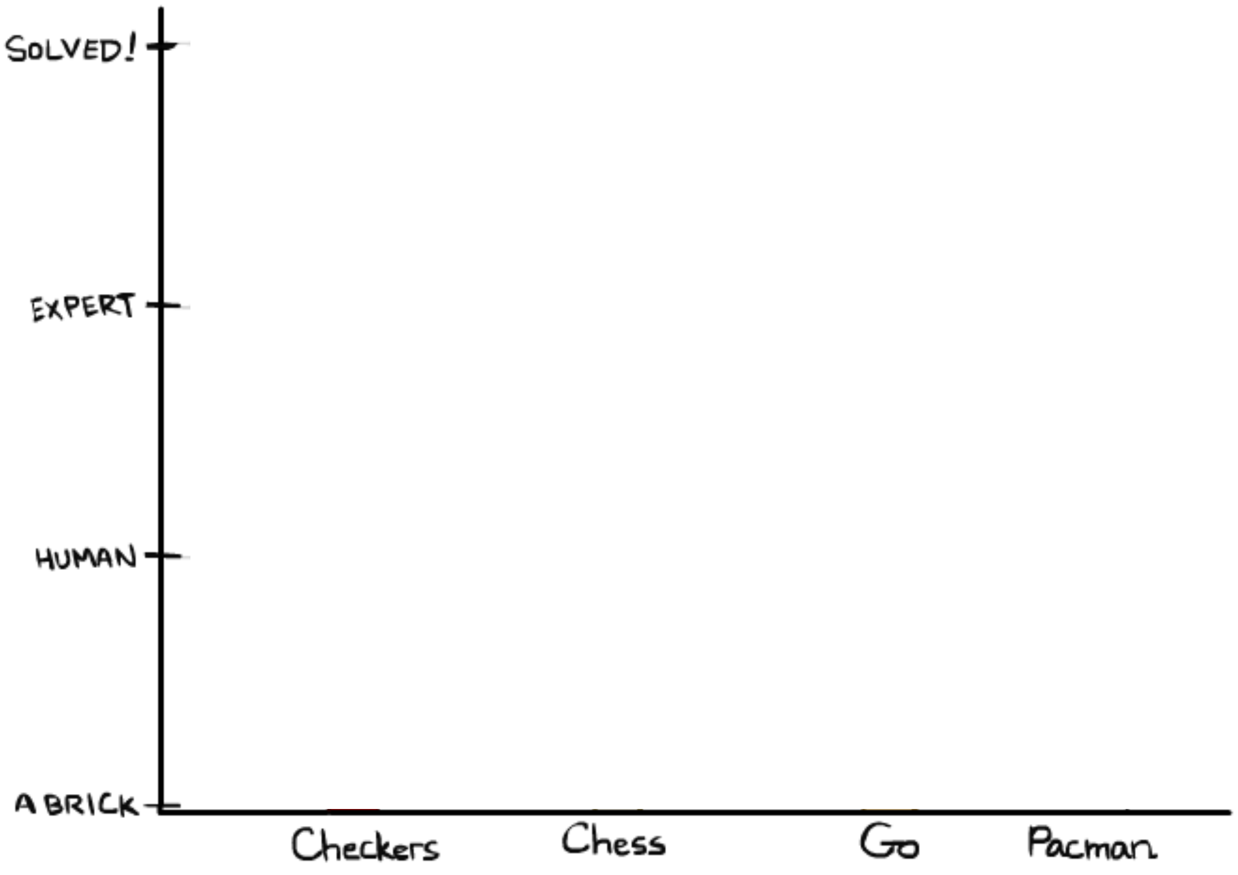
A brief history

- **Checkers:**
  - 1950: First computer player.
  - 1959: Samuel's self-taught program.
  - 1994: First computer world champion: Chinook defeats Tinsley
  - 2007: Checkers solved! Endgame database of 39 trillion states

- **Chess:**
  - 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCarthy.
  - 1960s onward: gradual improvement under "standard model"
  - 1997: Deep Blue defeats human champion Garry Kasparov
  - 2022: Stockfish (15+) rating 3541 (vs 2882 for Magnus Carlsen 2015).

- **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
  - 2016-2017: AlphaGo defeats human world champions

- **Pacman**



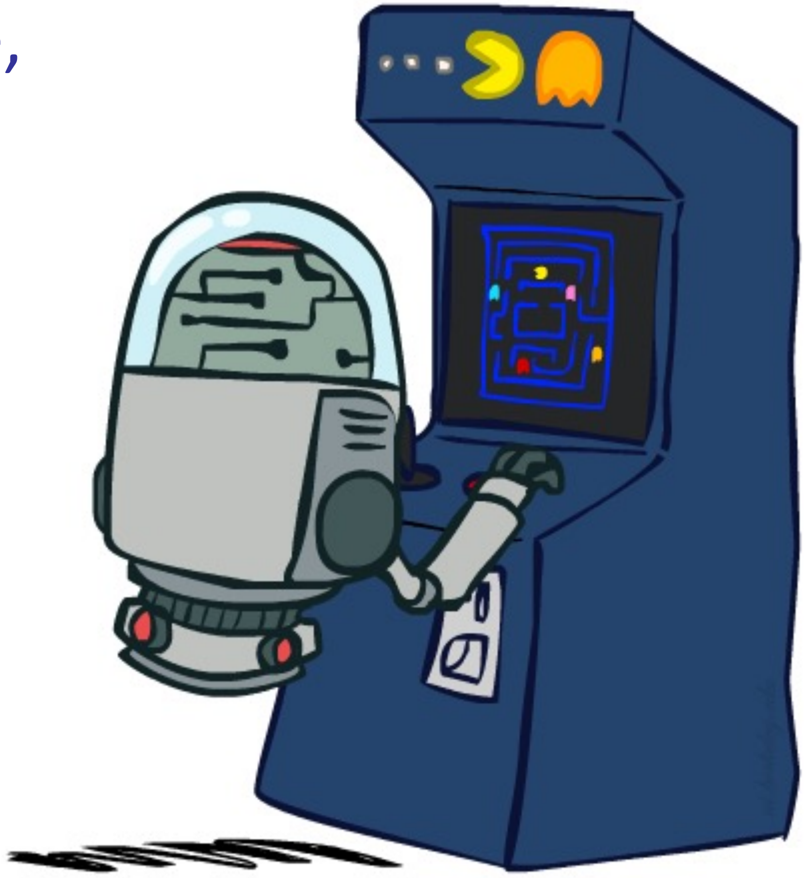
Types of Games

- Game = task environment with > 1 agent
- Axes:
  - Deterministic or stochastic?
  - Perfect information (fully observable)?
  - Two, three, or more players?
  - Teams or individuals?
  - Turn-taking or simultaneous?
  - Zero sum?

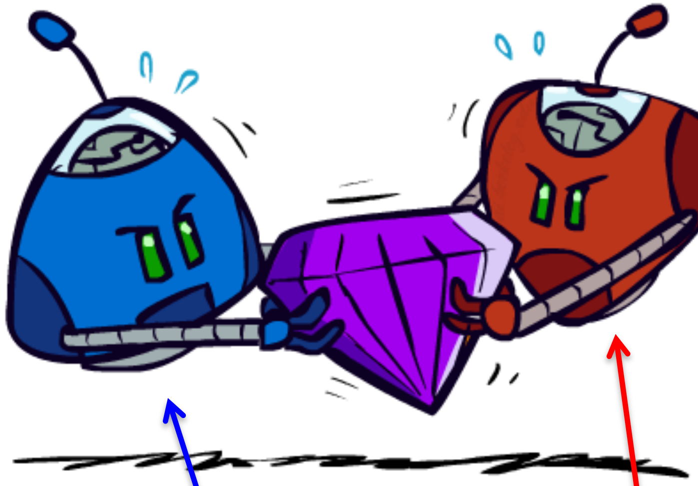
- Want algorithms for calculating a ***contingent plan*** (a.k.a. strategy or policy) which recommends a move for every possible eventuality

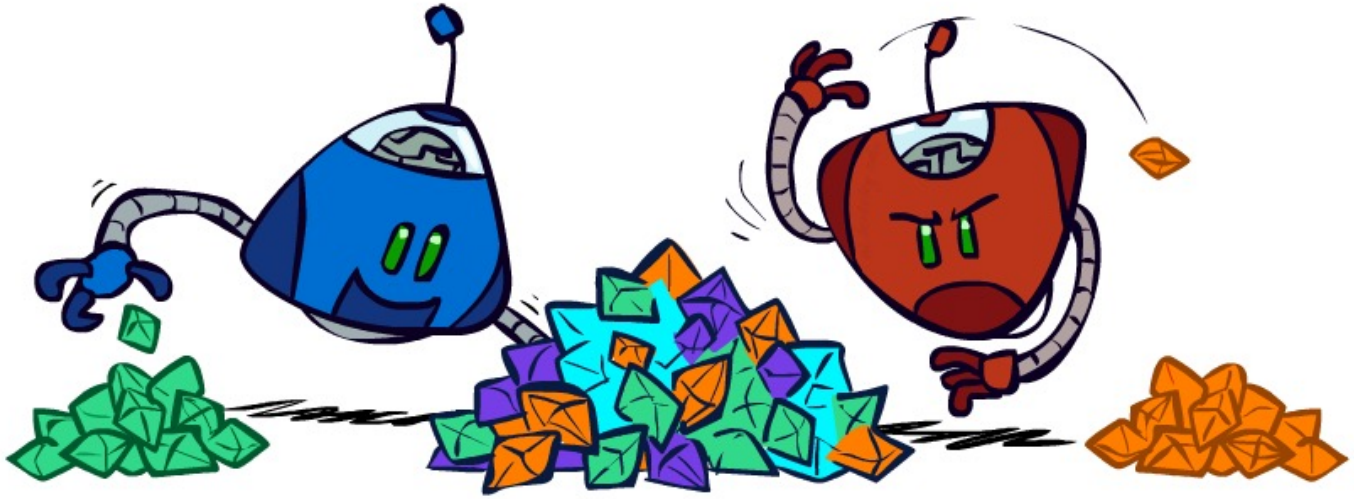
“Standard” Games

- Standard games are deterministic, observable, two-player, turn-taking, zero-sum
- Game formulation:
  - Initial state: s_0
  - Players: Player(s) indicates whose move it is
  - Actions: Actions(s) for player on move
  - Transition model: Result(s,a)
  - Terminal test: Terminal-Test(s)
  - Terminal values: Utility(s,p) for player p
    - Or just Utility(s) for player making the decision at root
- Solution for a player is a policy : S → A



Zero-Sum Games

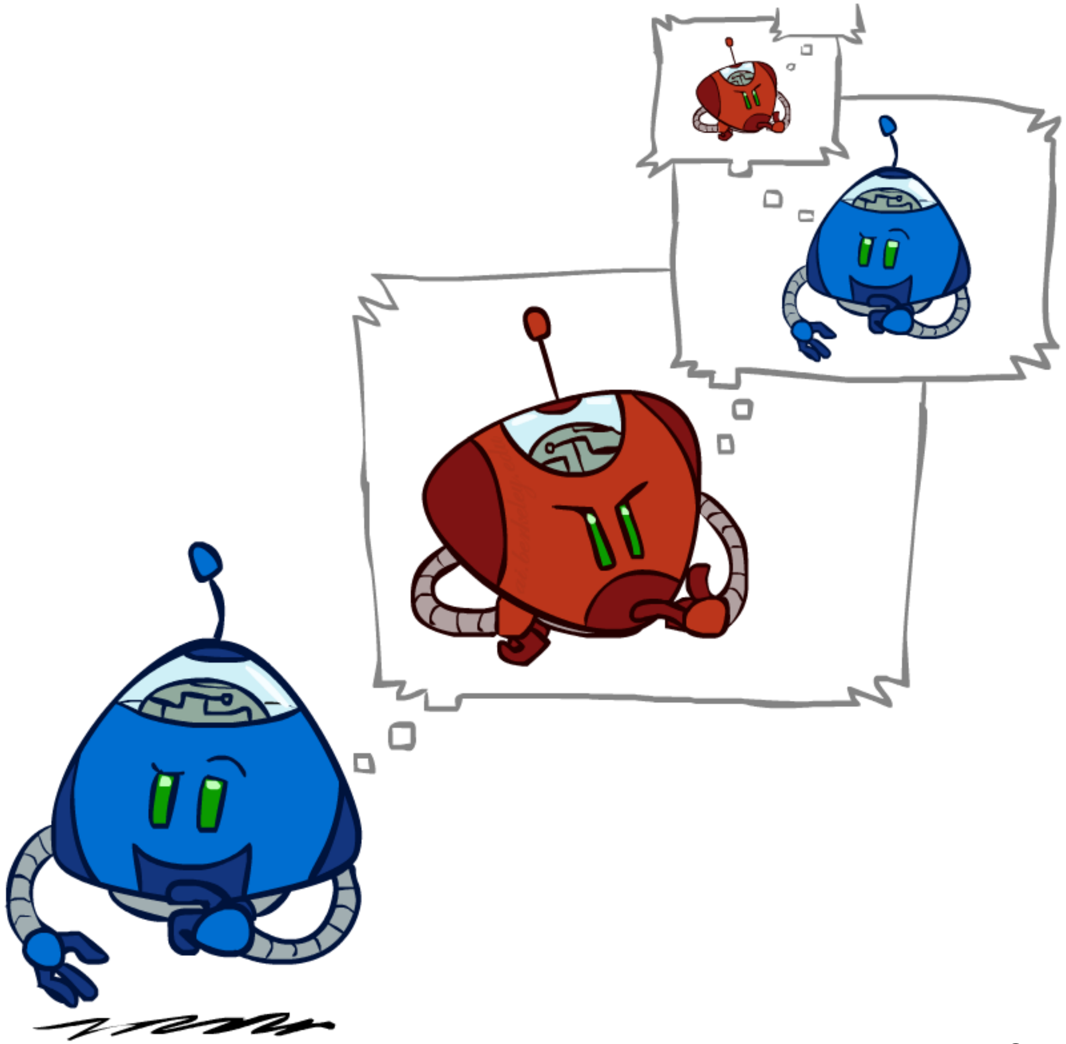




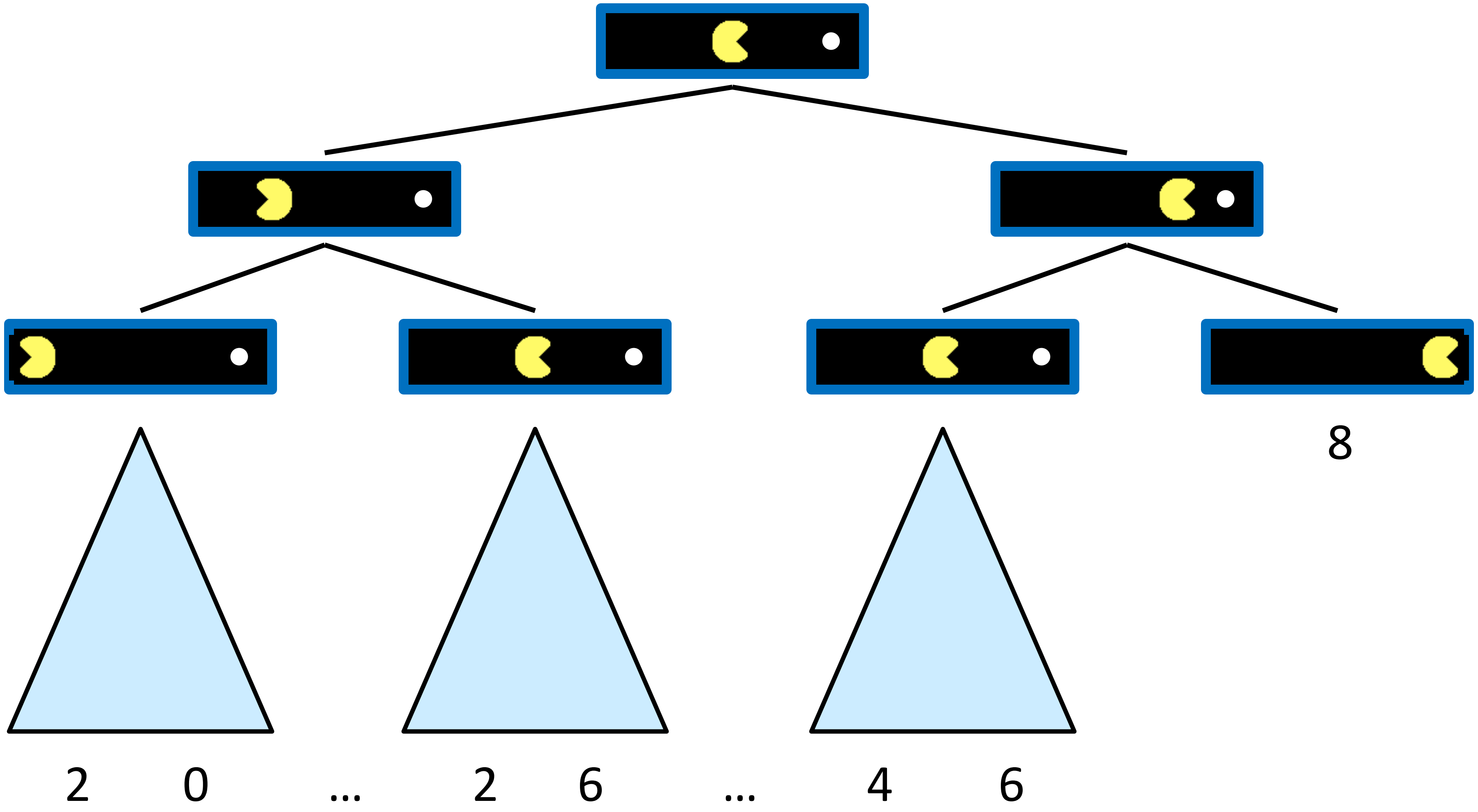
- Zero-Sum Games
  - Agents have ***opposite*** utilities
  - Pure competition:
    - One ***maximizes***, the other ***minimizes***

- General-Sum Games
  - Agents have ***independent*** utilities
  - Cooperation, indifference, competition, shifting alliances, and more are all possible
- Team Games
  - Common payoff for all team members

Adversarial Search



Single-Agent Trees

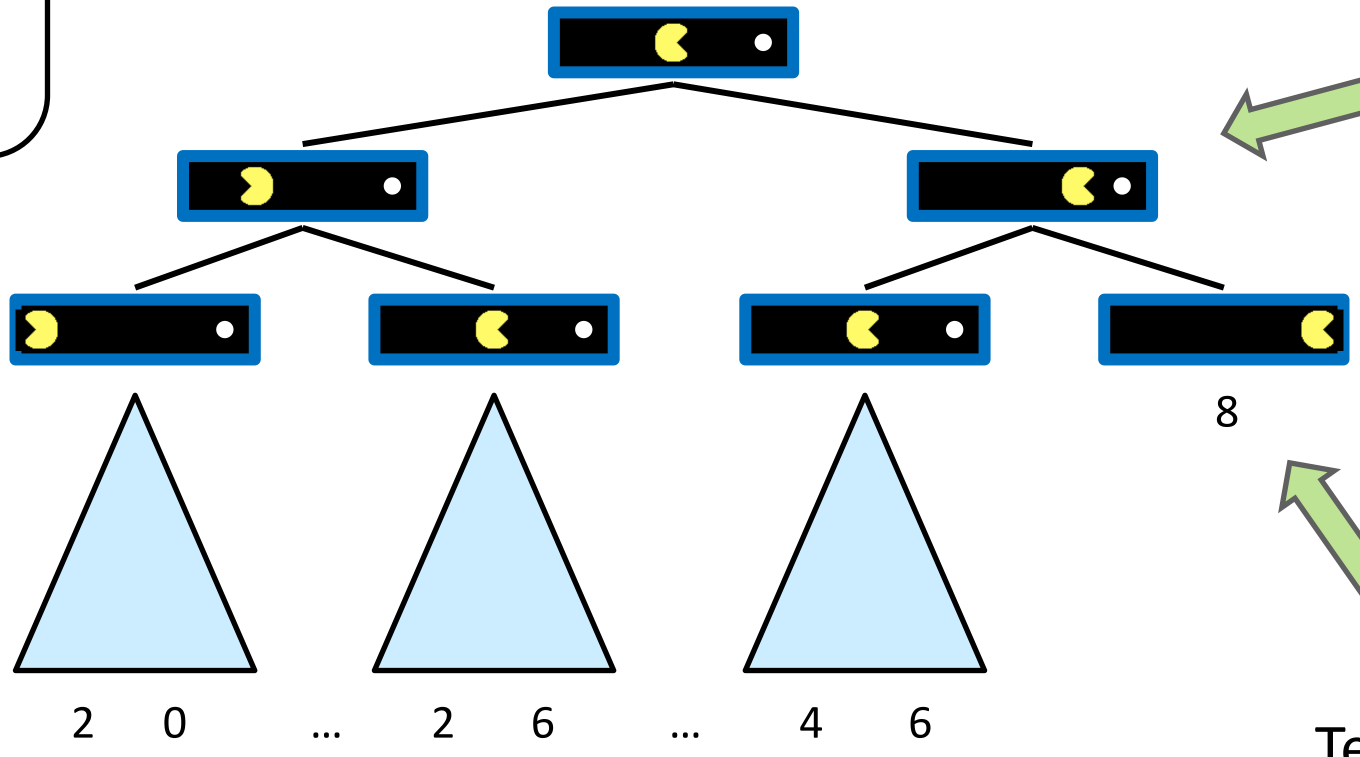


Value of a State

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

Non-Terminal States:

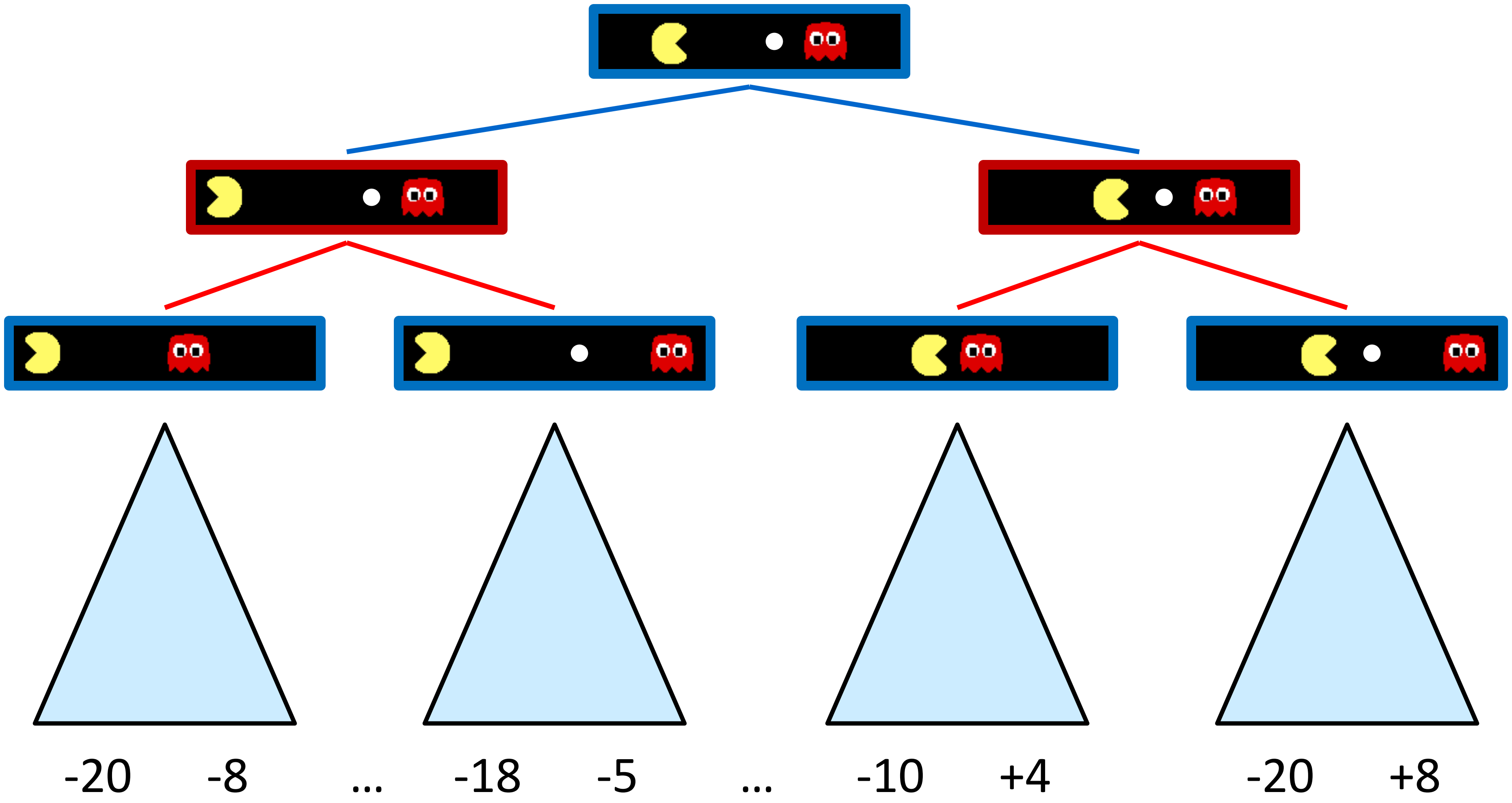
$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$



Terminal States:

$V(s)$ = known

Adversarial Game Trees



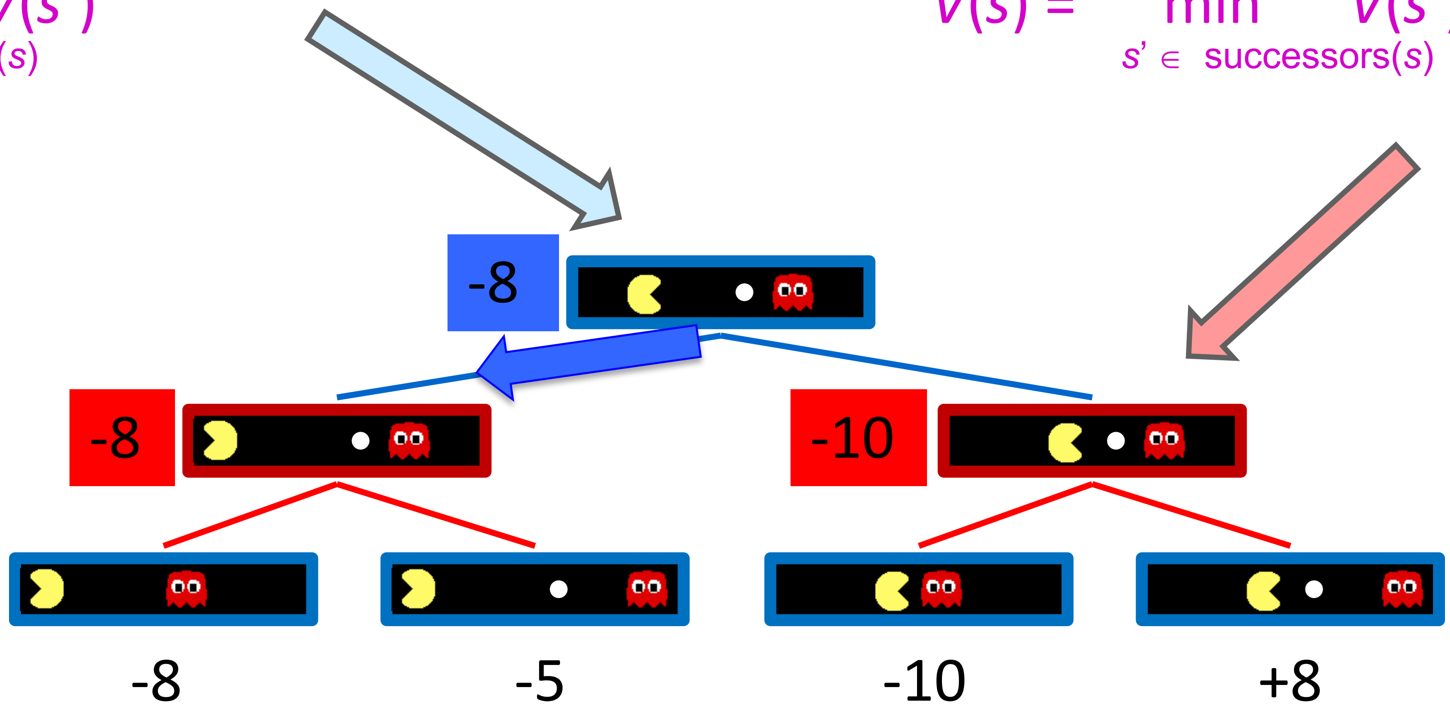
Minimax Values

MAX nodes: under Agent's control

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

MIN nodes: under Opponent's control

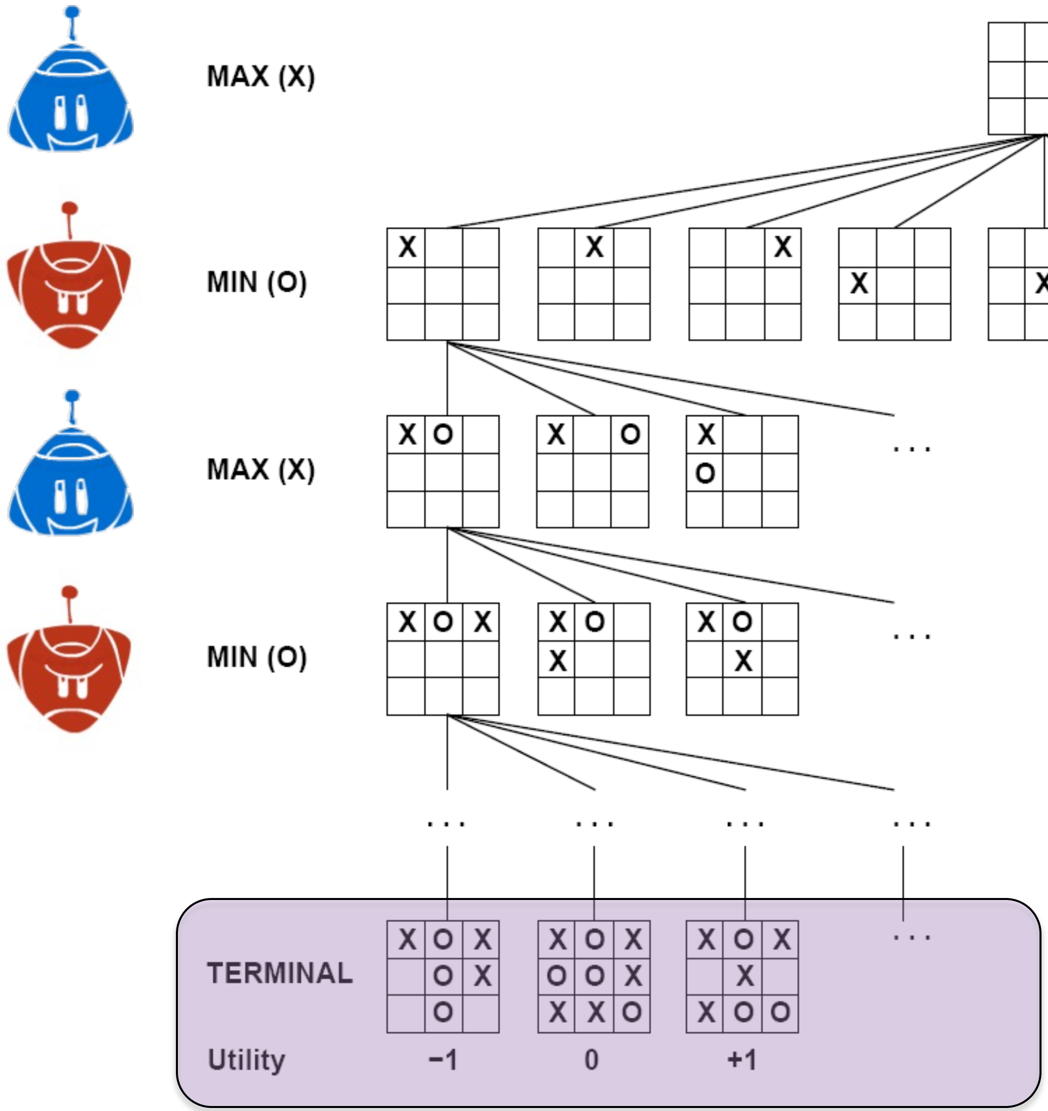
$$V(s) = \min_{s' \in \text{successors}(s)} V(s')$$

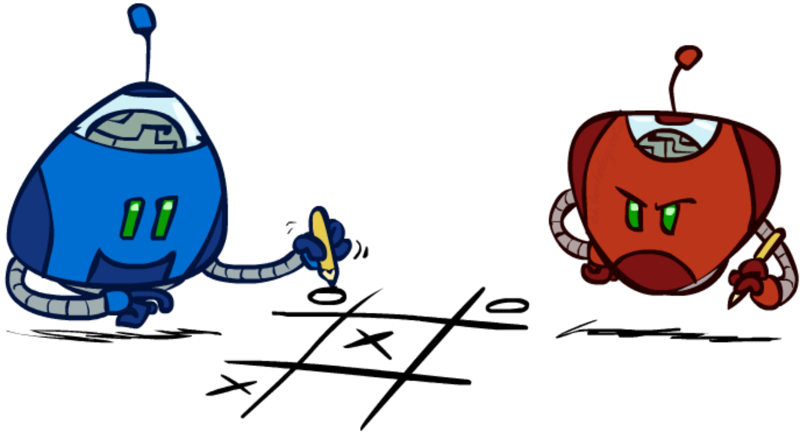


Terminal States:

$V(s)$ = known

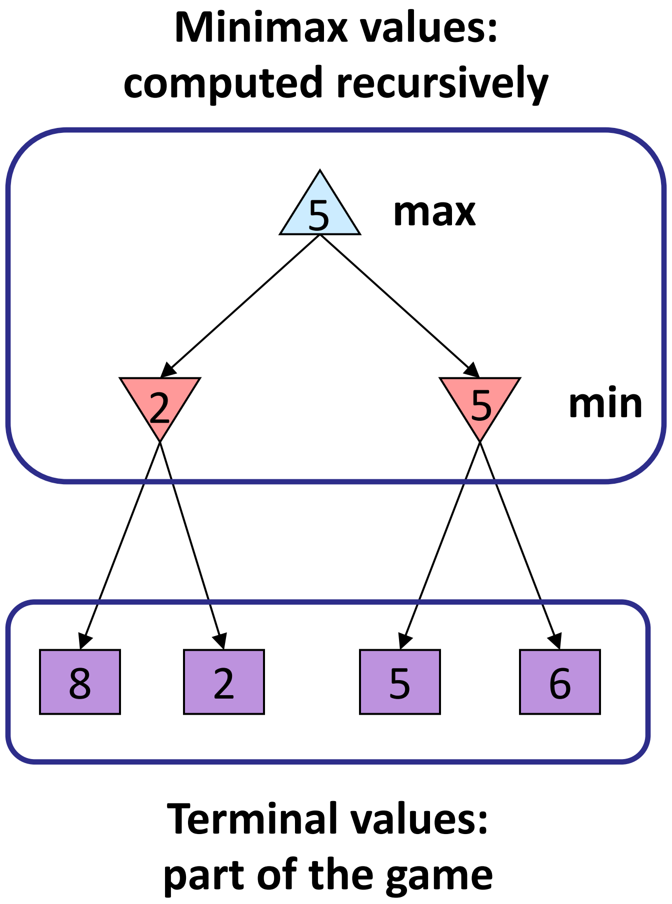
Tic-Tac-Toe Game Tree





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
```

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

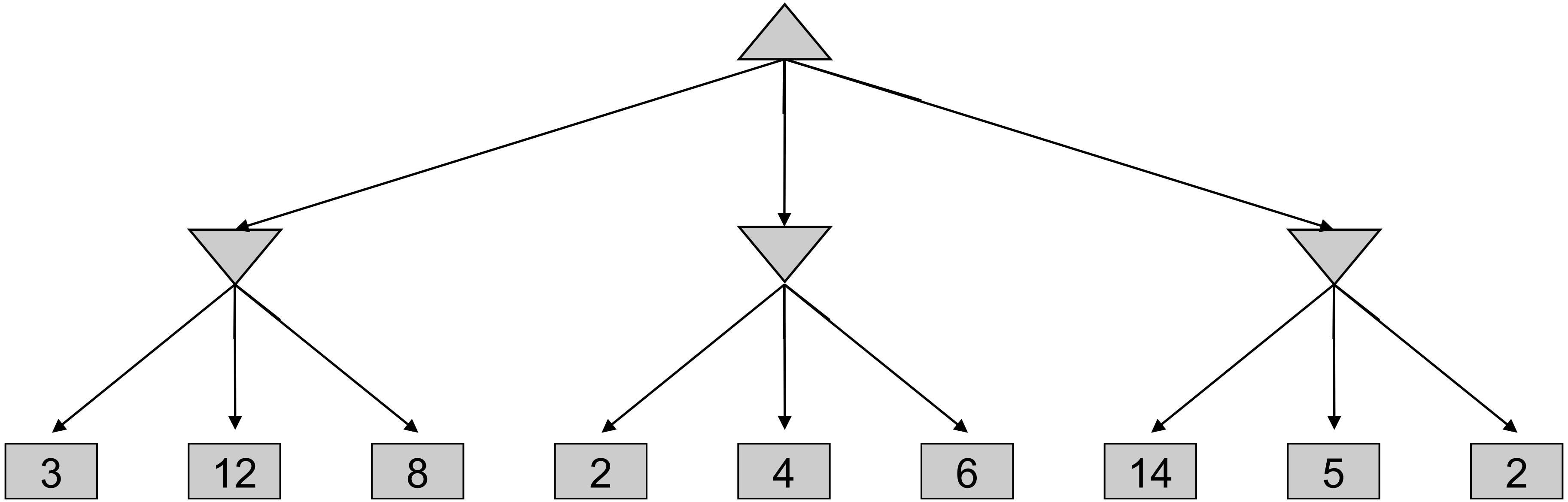
Minimax Implementation (Dispatch)

```
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
```

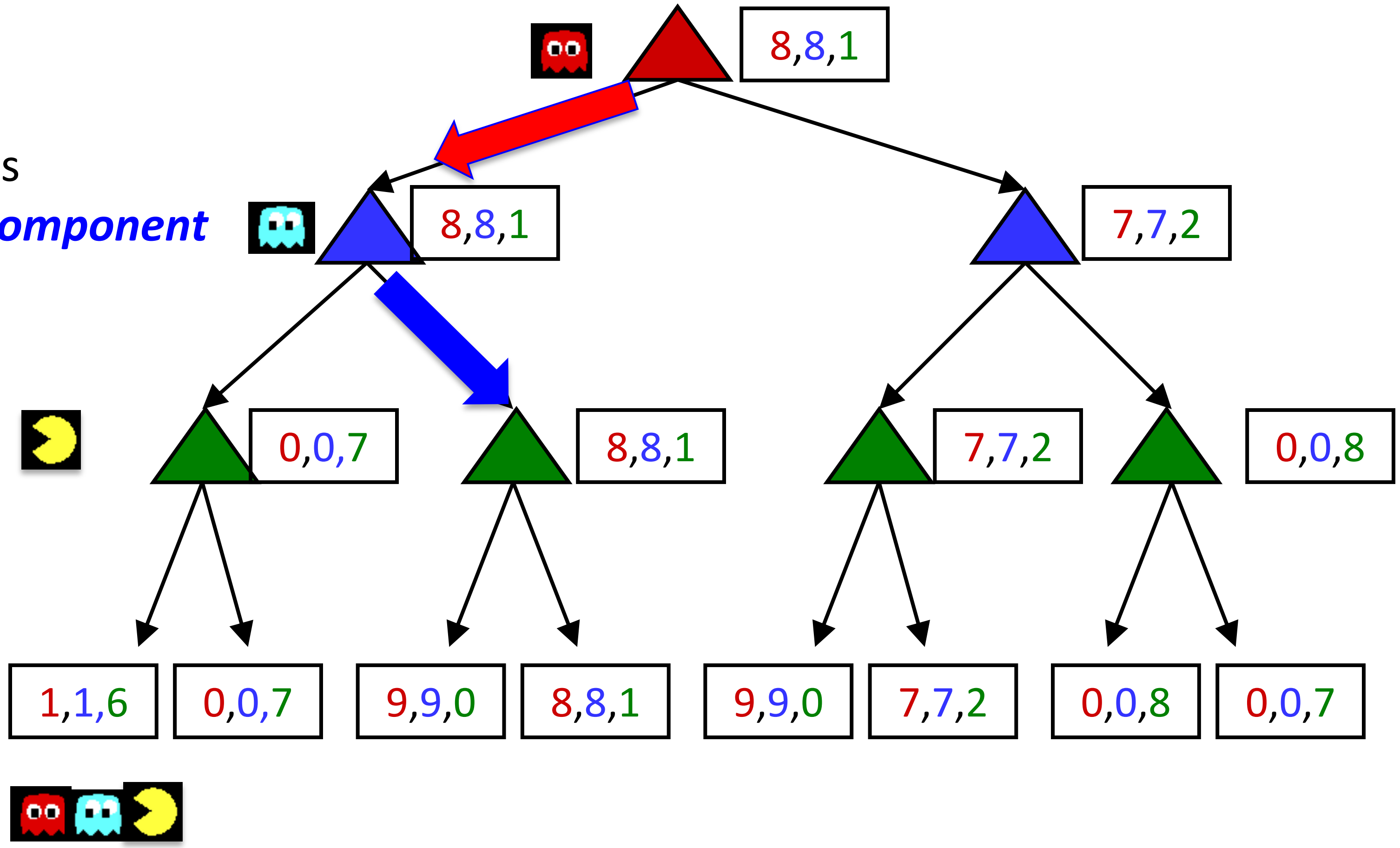
Minimax Example

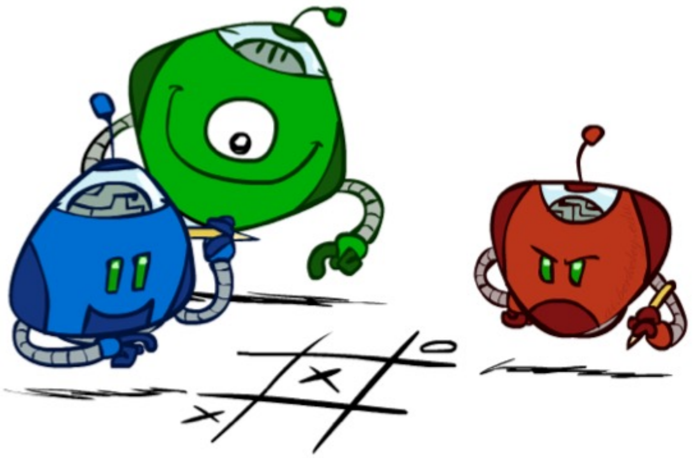


Generalized minimax

- What if the game is not zero-sum, or has multiple players?

- Generalization of minimax:
  - Terminals have ***utility tuples***
  - Node values are also utility tuples
  - ***Each player maximizes its own component***
  - Can give rise to cooperation and competition dynamically…



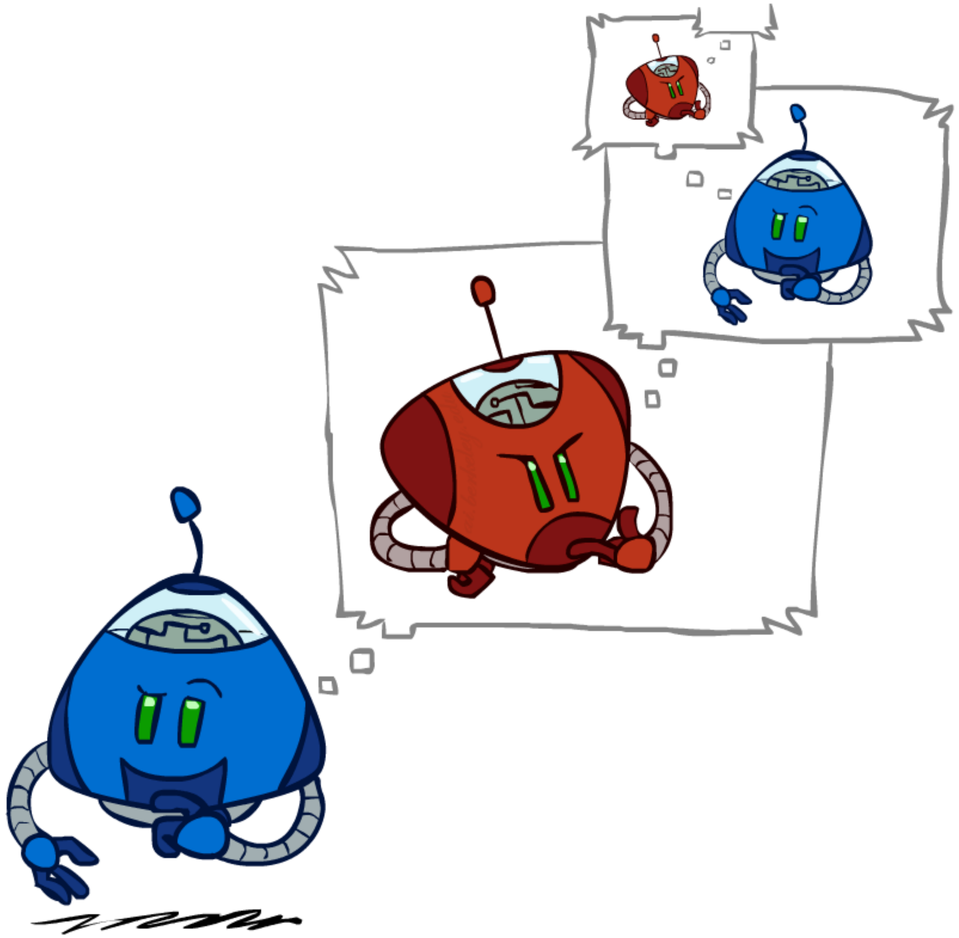


Emergent coordination in ghosts

Minimax Efficiency

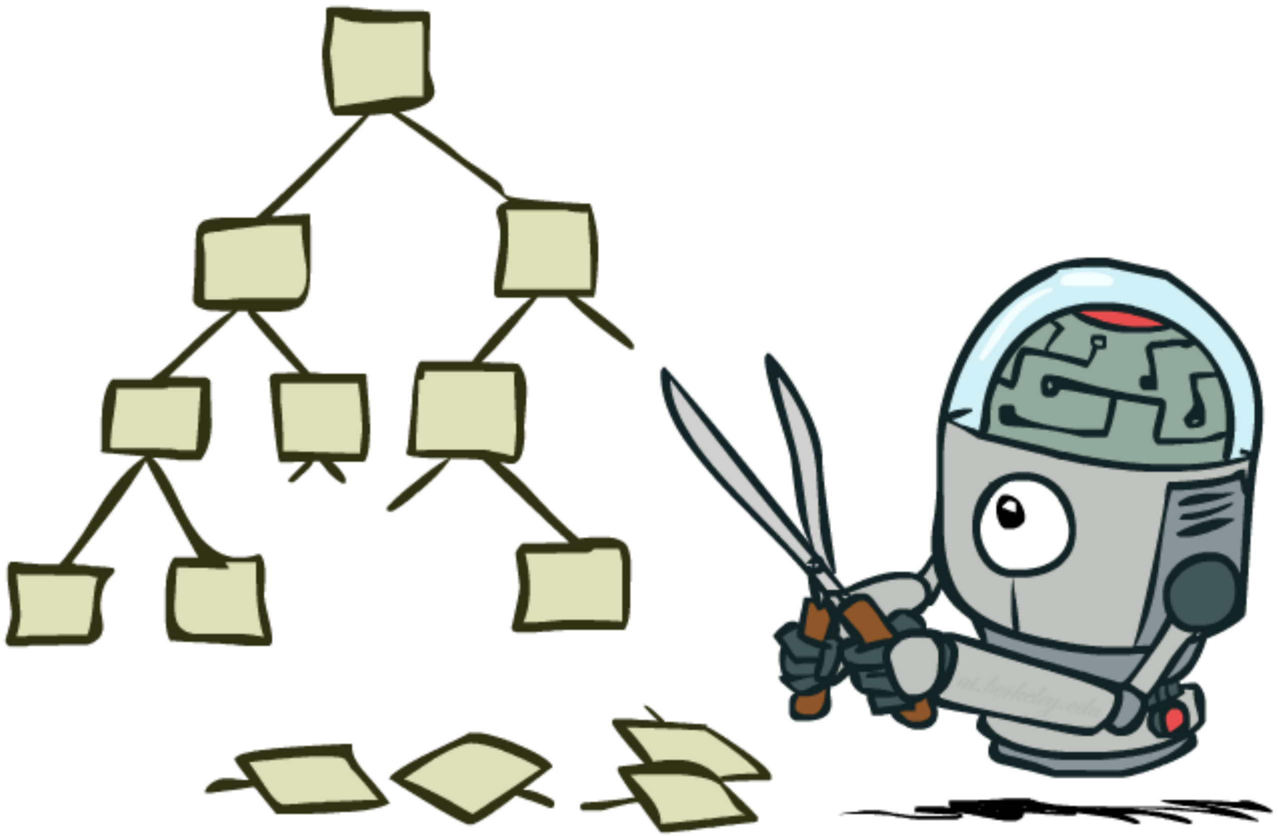
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - Humans can't do this either, so how do we play chess?

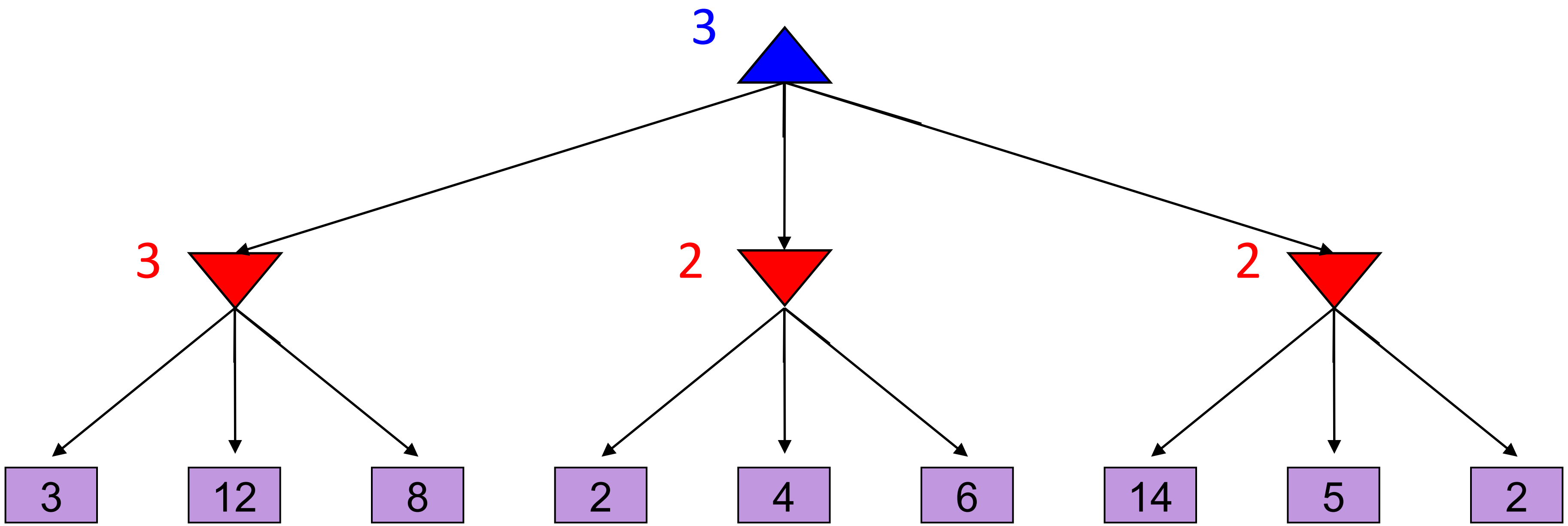


Resource Limits

Game Tree Pruning

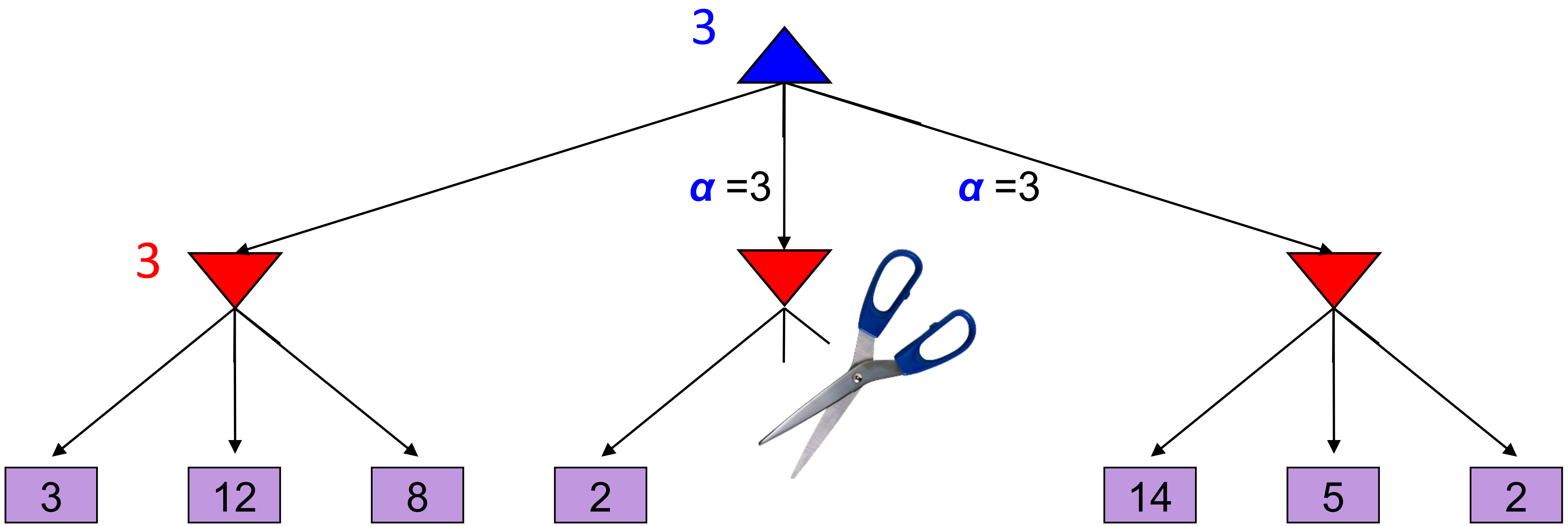


Minimax Example



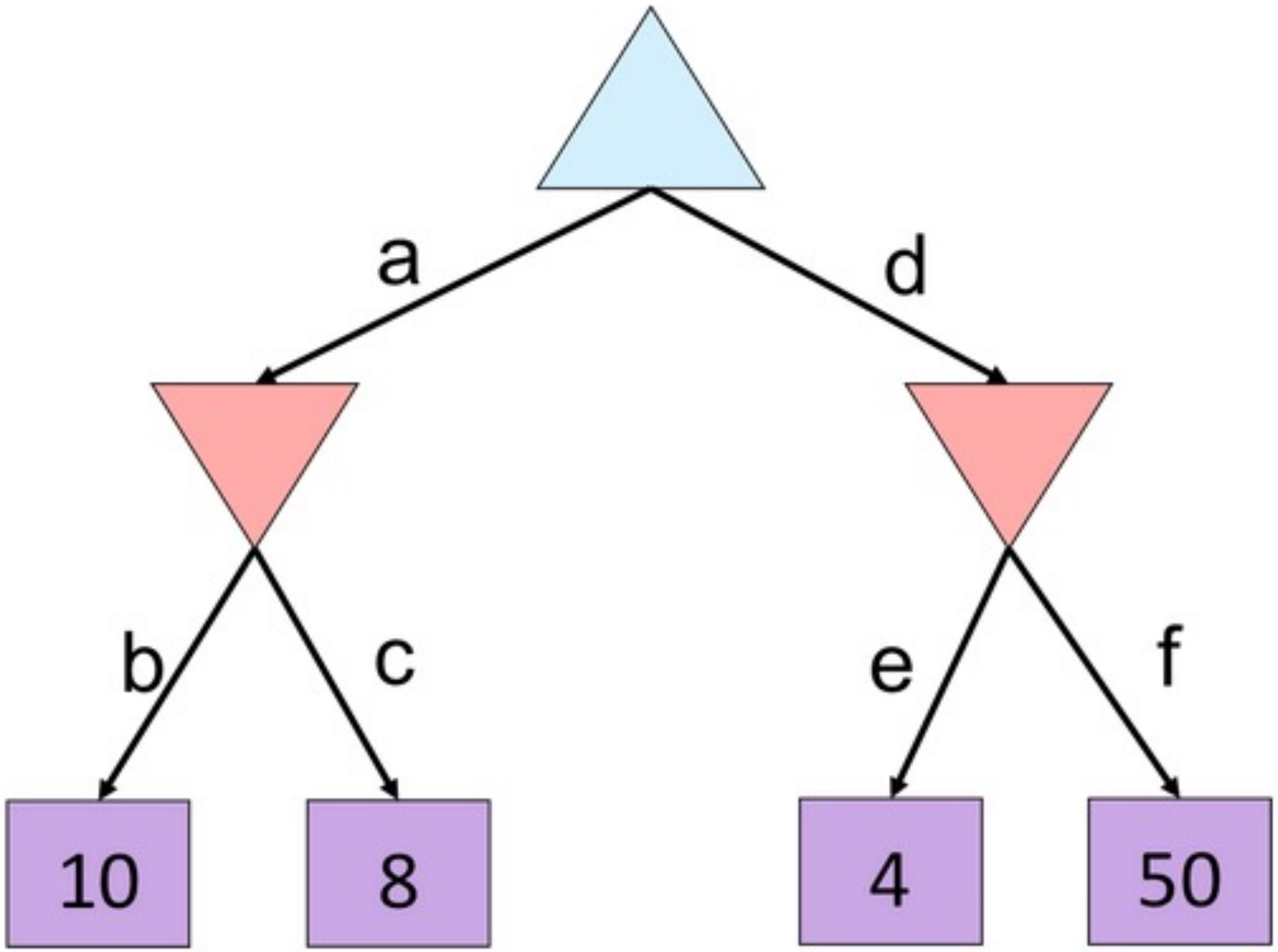
Alpha-Beta Example

α = best option so far from any MAX node on this path

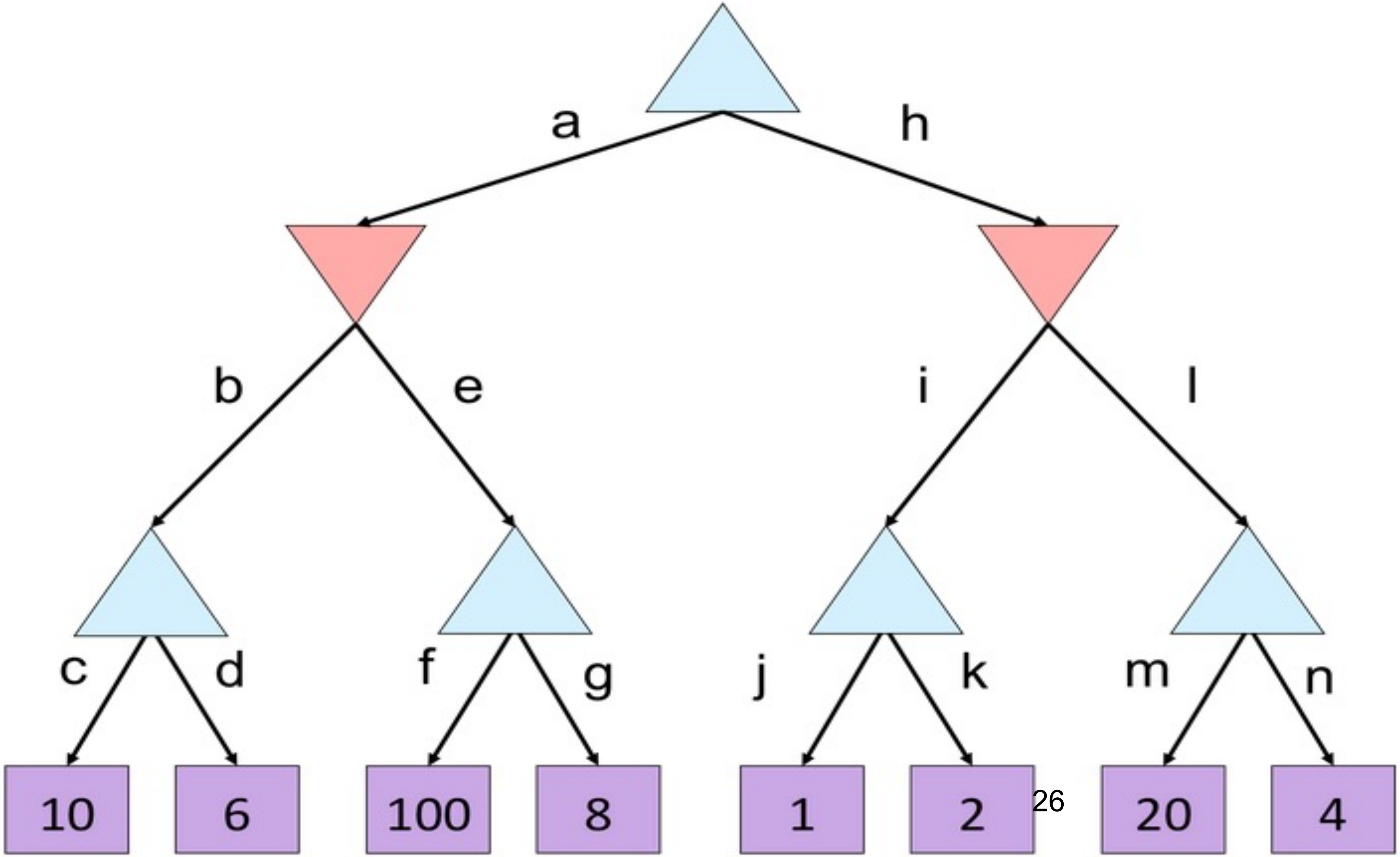


- ***The order of generation matters***: more pruning is possible if good moves come first

Alpha-Beta Quiz

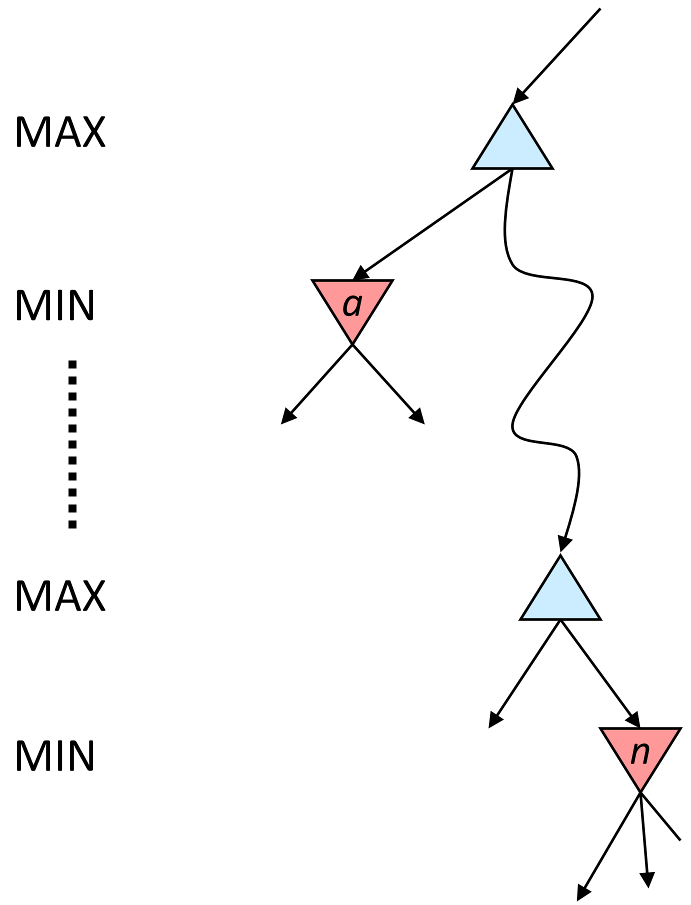


Alpha-Beta Quiz 2



Alpha-Beta Pruning

- General case (pruning children of MIN node)
  - 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 ***α*** be the best value that MAX can get so far at any choice point along the current path from the root
  - If *n* becomes worse than ***α***, MAX will avoid it, so we can prune *n*'s other children (it's already bad enough that it won't be played)
- Pruning children of MAX node is symmetric
  - Let **β** be the best value that MIN can get so far at any choice point along the current path from the root



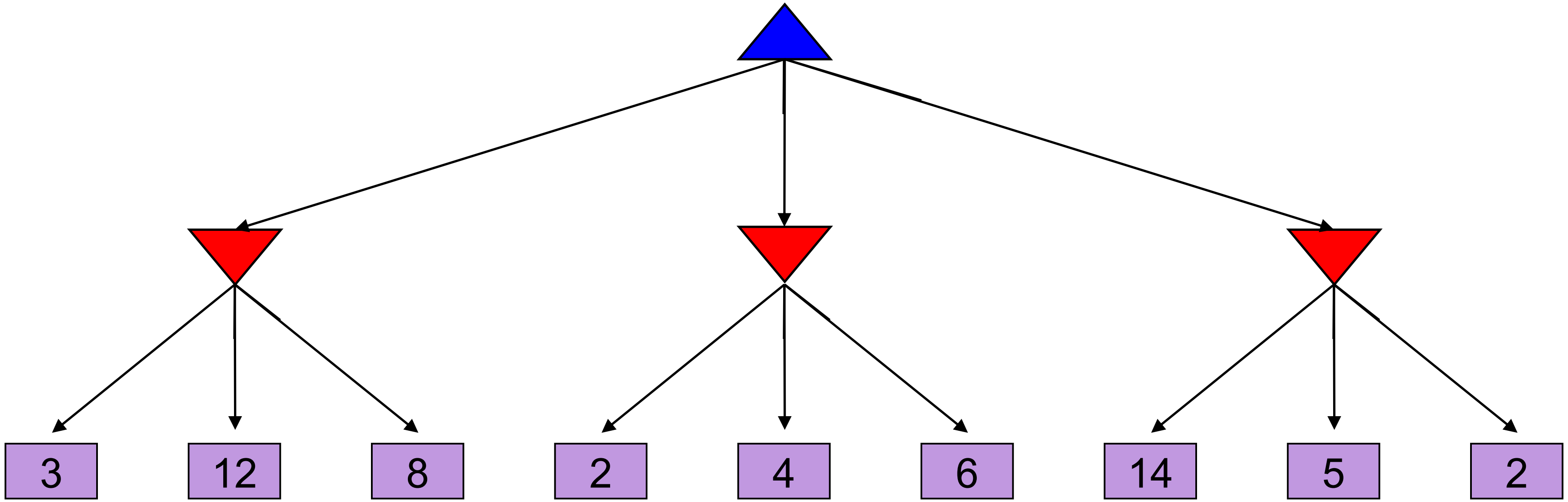
Alpha-Beta Implementation

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

```
def max-value(state, α, β):
    if terminal(state) return Utility(state)
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor, α,β))
        if v ≥ β
            return v
        α = max(α, v)
    return v
```

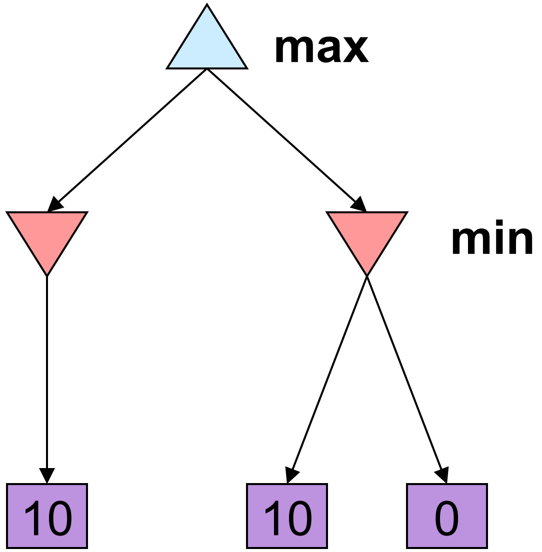
```
def min-value(state , α, β):
    if terminal(state) return Utility(state)
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor, α, β))
        if v ≤ α
            return v
        β = min(β, v)
    return v
```

Alpha-beta code on example



Alpha-Beta Pruning Properties

- Theorem: This pruning has ***no effect*** on minimax value computed for the root!
- Good child ordering improves effectiveness of pruning
  - Iterative deepening helps with this
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
- This is a simple example of metareasoning (reasoning about reasoning)
- For chess: only 35^{50} instead of 35^{100}!! Yaaay!!!!!



Summary

- Games are decision problems with $\geq$ 2 agents
  - Huge variety of issues and phenomena depending on details of interactions and payoffs
- For zero-sum games, optimal decisions defined by minimax
  - Simple extension to n-player “rotating” max with vectors of utilities
  - 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