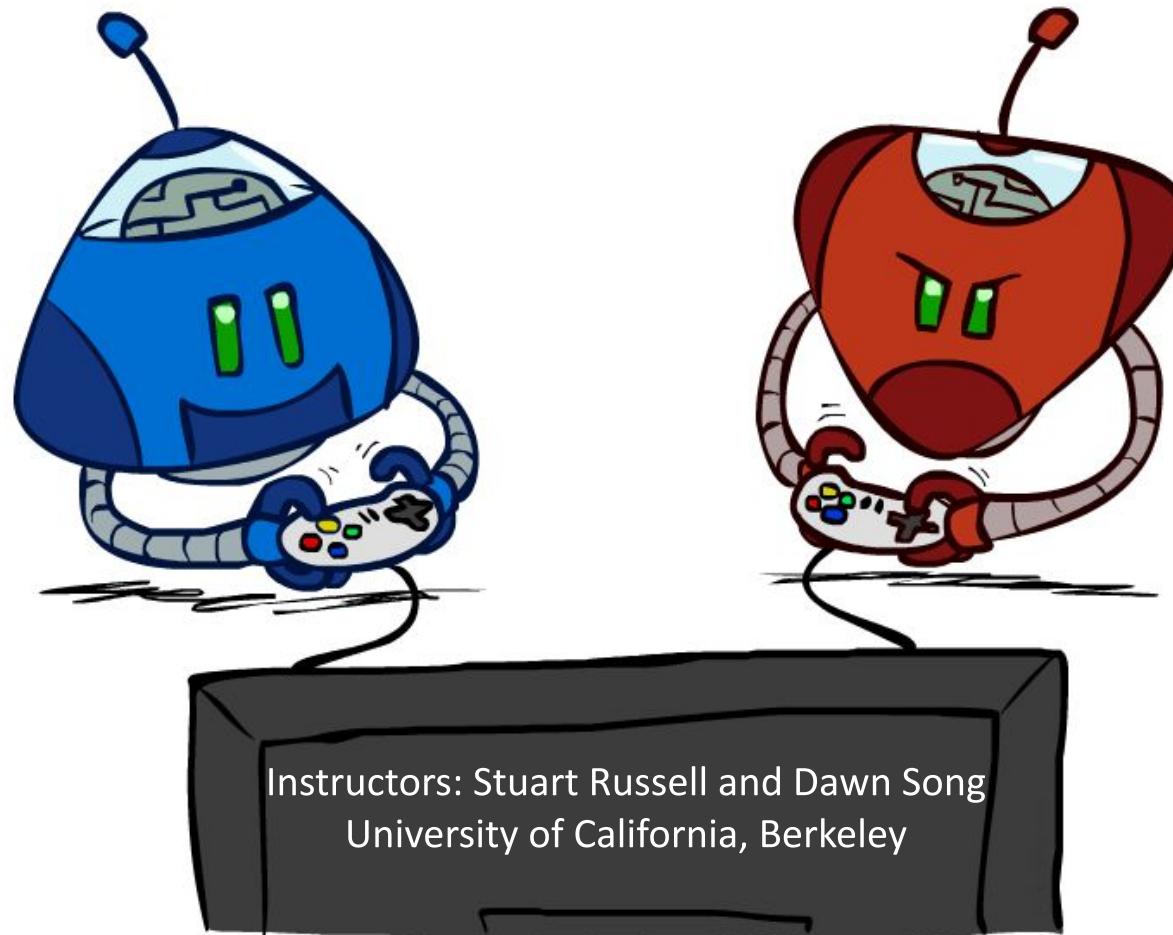


# CS 188: Artificial Intelligence

## Adversarial Search

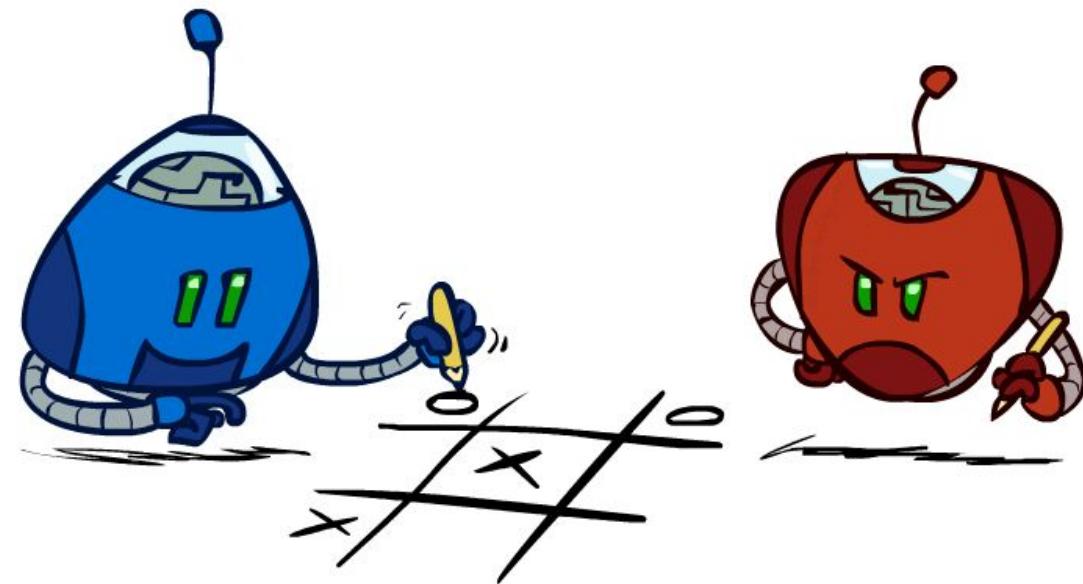


Instructors: Stuart Russell and Dawn Song  
University of California, Berkeley

# Outline

---

- History / Overview
- Minimax for Zero-Sum Games
- $\alpha$ - $\beta$  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

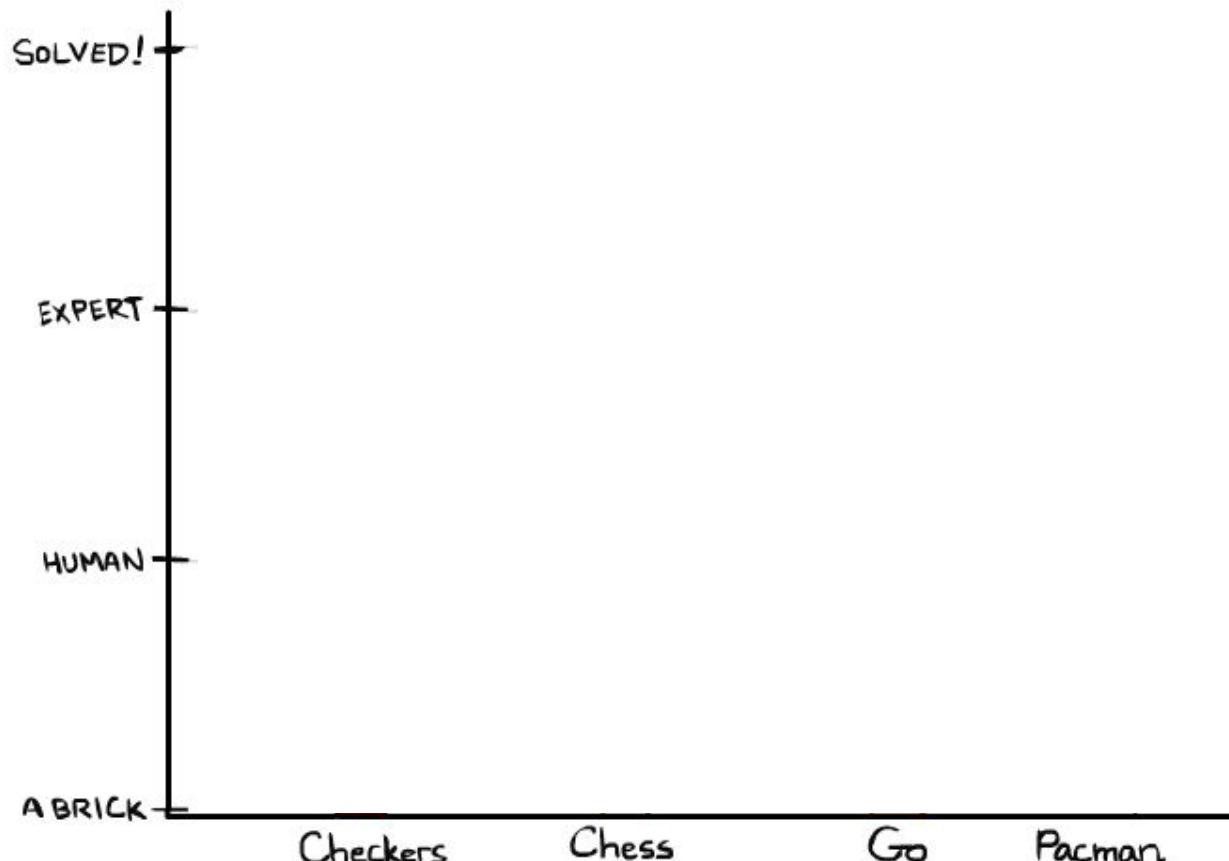
- **Chess:**

- 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCarthy.
- 1960s onward: gradual improvement under "standard model"
- 1997: Deep Blue defeats human champion Garry Kasparov
- 2021: Stockfish rating 3551 (vs 2870 for Magnus Carlsen).

- **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**



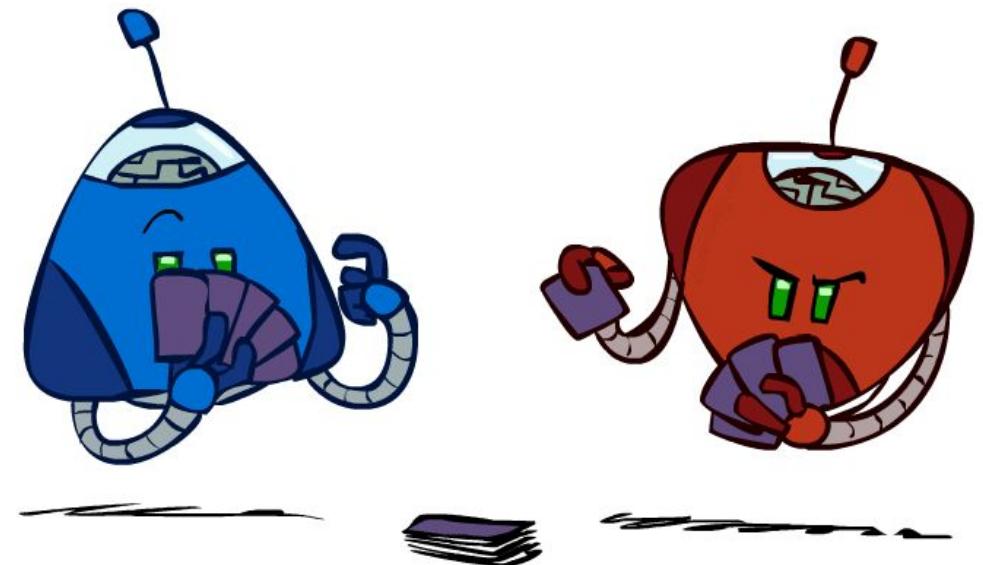
# Types of Games

- Game = task environment with > 1 agent

- Axes:

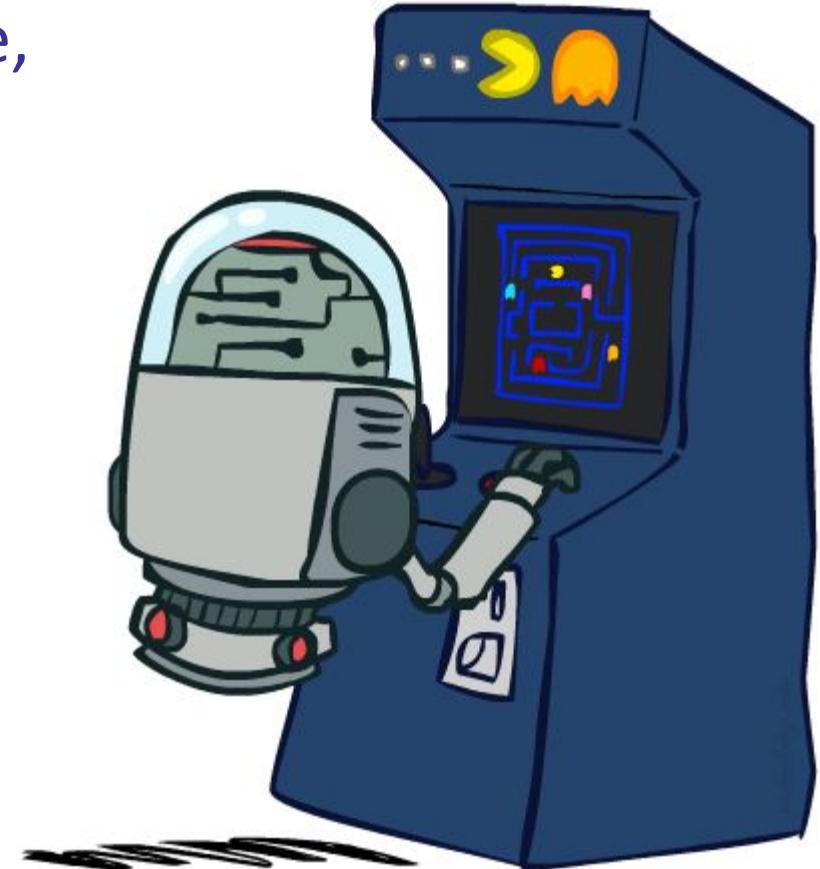
- Deterministic or stochastic?
- Perfect information (fully observable)?
- One, two, or more players?
- Turn-taking or simultaneous?
- Zero sum?

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

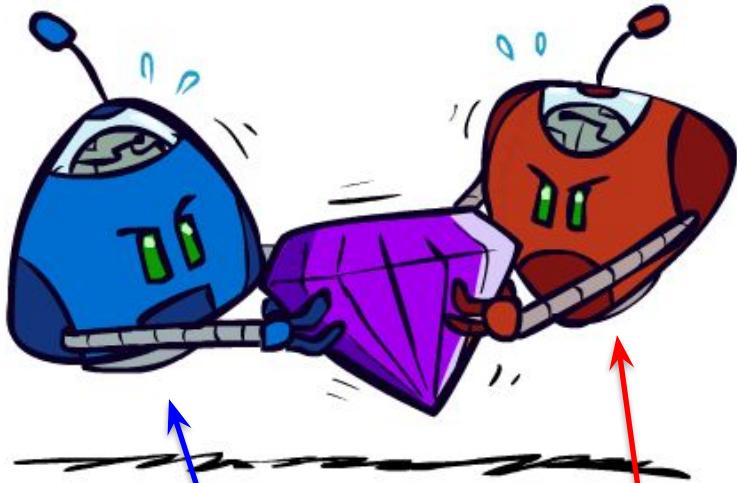


# “Standard” Games

- 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



# Zero-Sum Games

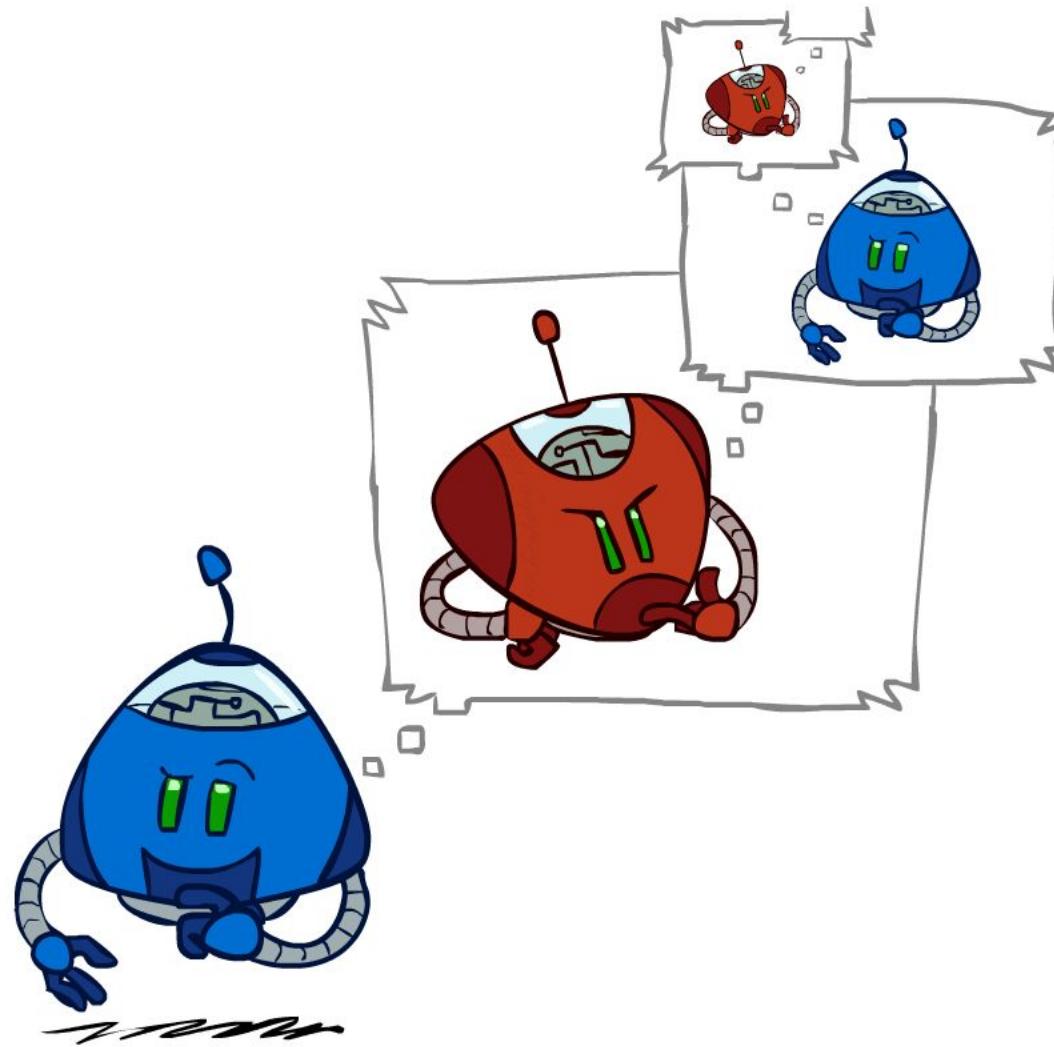


- Zero-Sum Games
  - Agents have **opposite** utilities
  - Pure competition:
    - One **maximizes**, the other **minimizes**

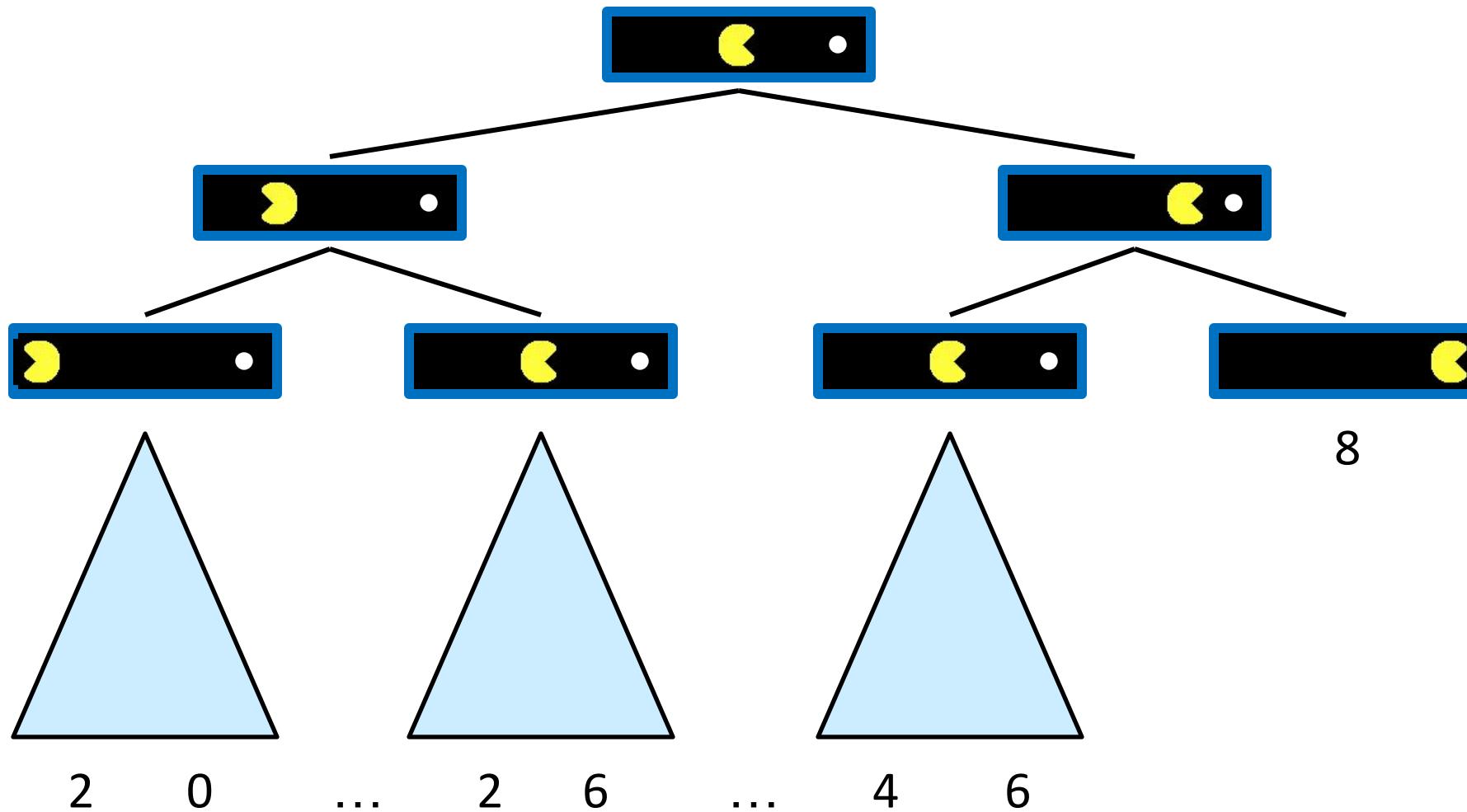
- General Games
  - Agents have **independent** utilities
  - Cooperation, indifference, competition, shifting alliances, and more are all possible

# Adversarial Search

---

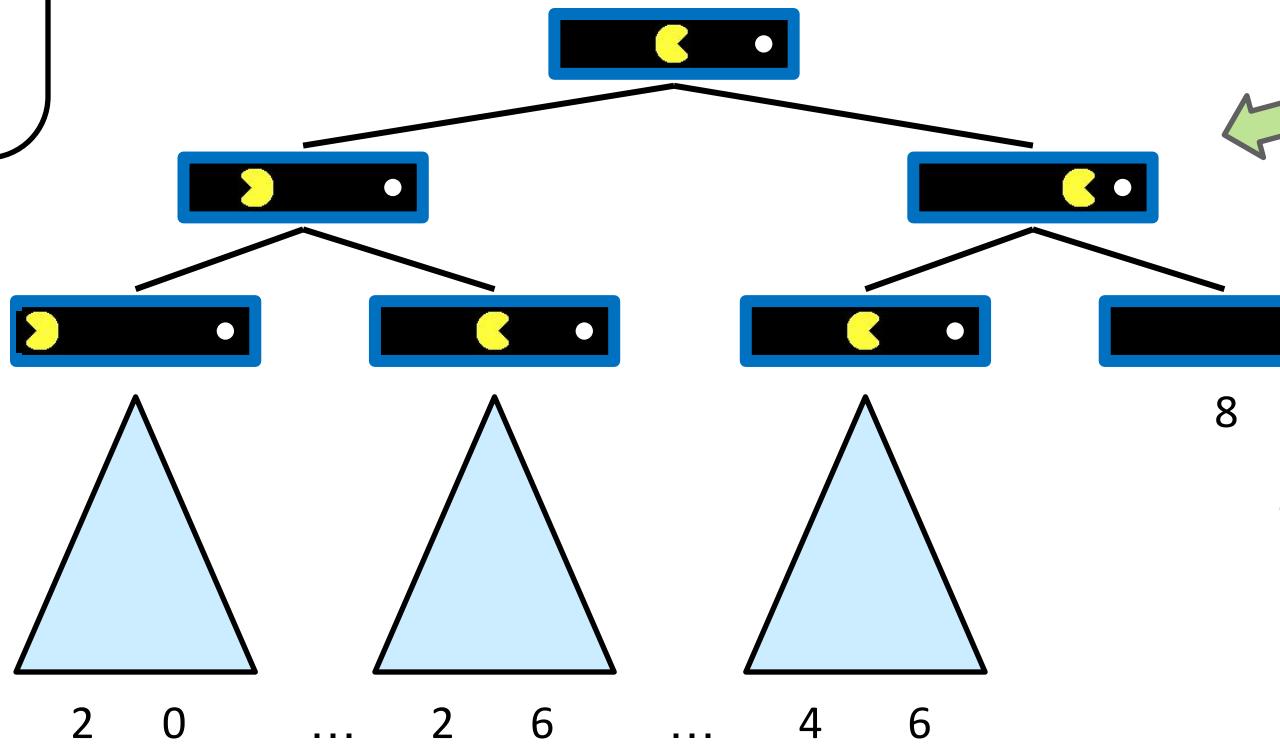


# Single-Agent Trees



# Value of a State

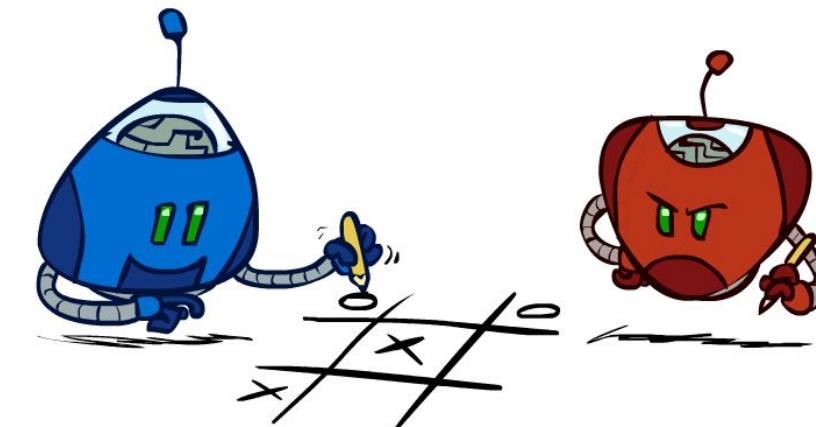
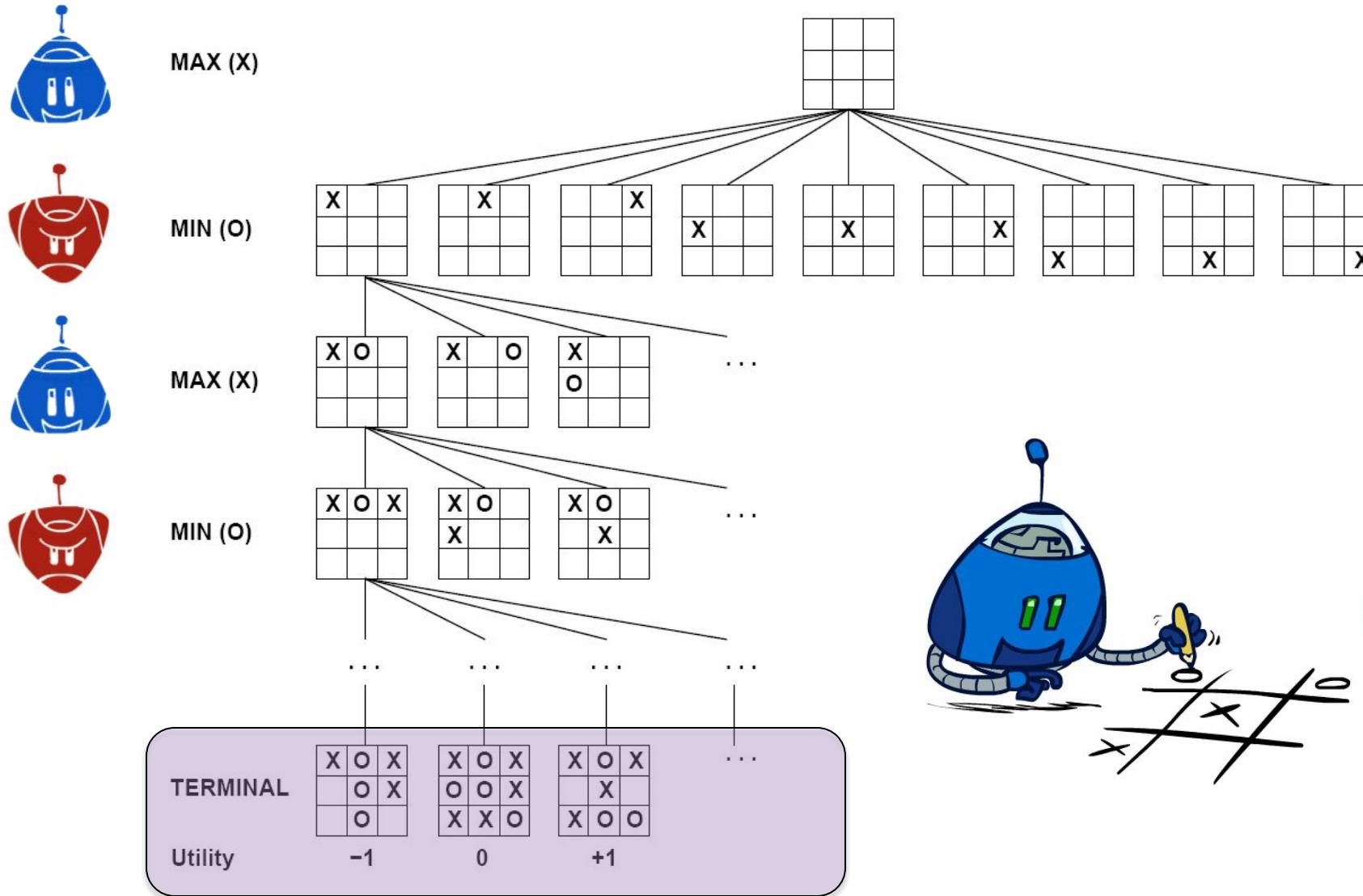
Value of a state:  
The best achievable  
outcome (utility)  
from that state



Non-Terminal States:  
 $V(s) = \max_{s' \in \text{successors}(s)} V(s')$

Terminal States:  
 $V(s) = \text{known}$

# Tic-Tac-Toe Game Tree



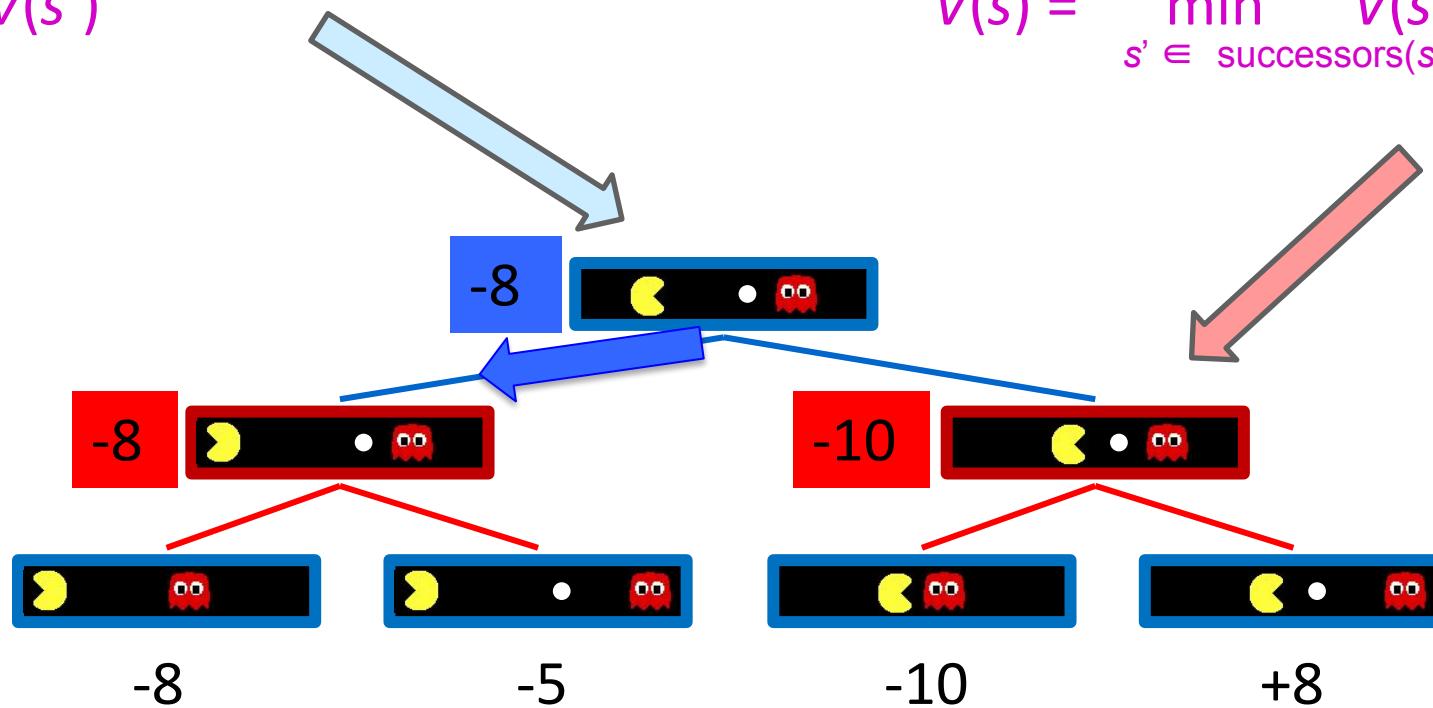
# Minimax Values

MAX nodes: under Agent's control

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

MIN nodes: under Opponent's control

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



Terminal States:

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

# Minimax algorithm

---

- Choose action leading to state with best *minimax value*
- Assumes all future moves will be optimal
- => rational against a rational player

# Implementation

```
function minimax-decision(s) returns an action  
    return the action a in Actions(s) with the highest  
        minimax_value(Result(s,a))
```



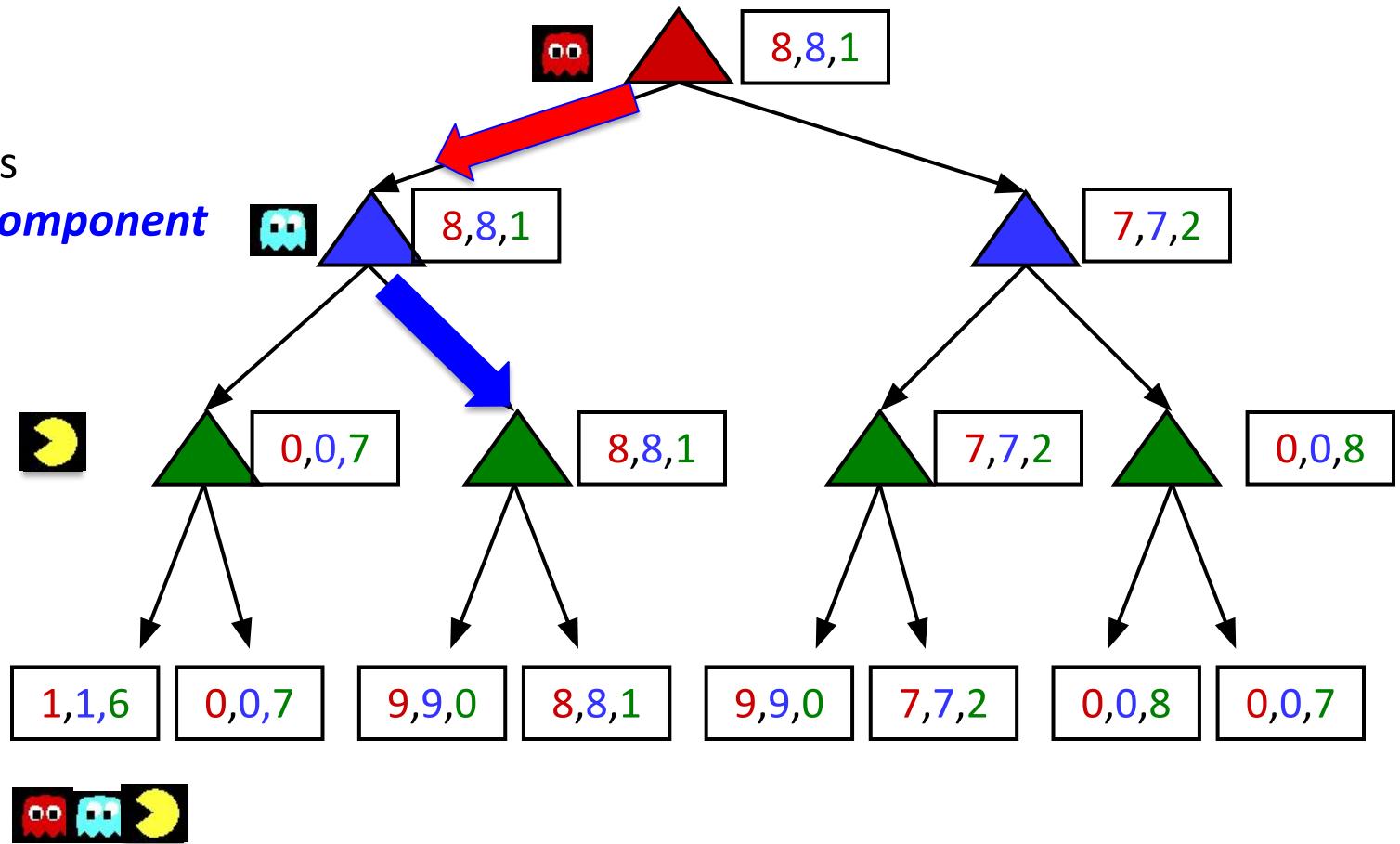
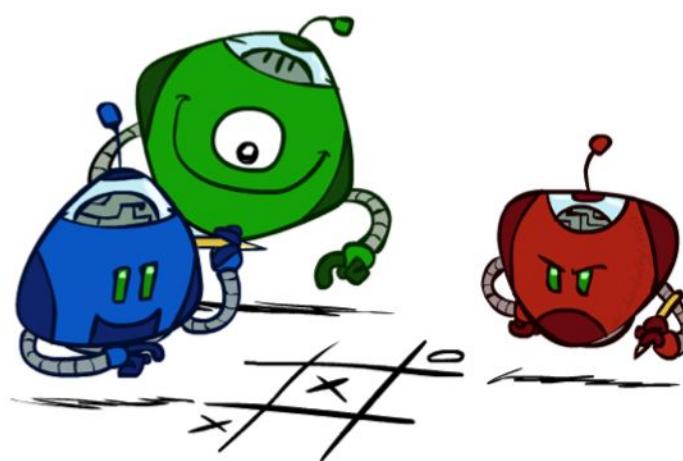
```
function minimax_value(s) returns a value  
    if Terminal-Test(s) then return Utility(s)  
    if Player(s) = MAX then return maxa in Actions(s) minimax_value(Result(s,a))  
    if Player(s) = MIN then return mina in Actions(s) minimax_value(Result(s,a))
```

# Generalized minimax

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

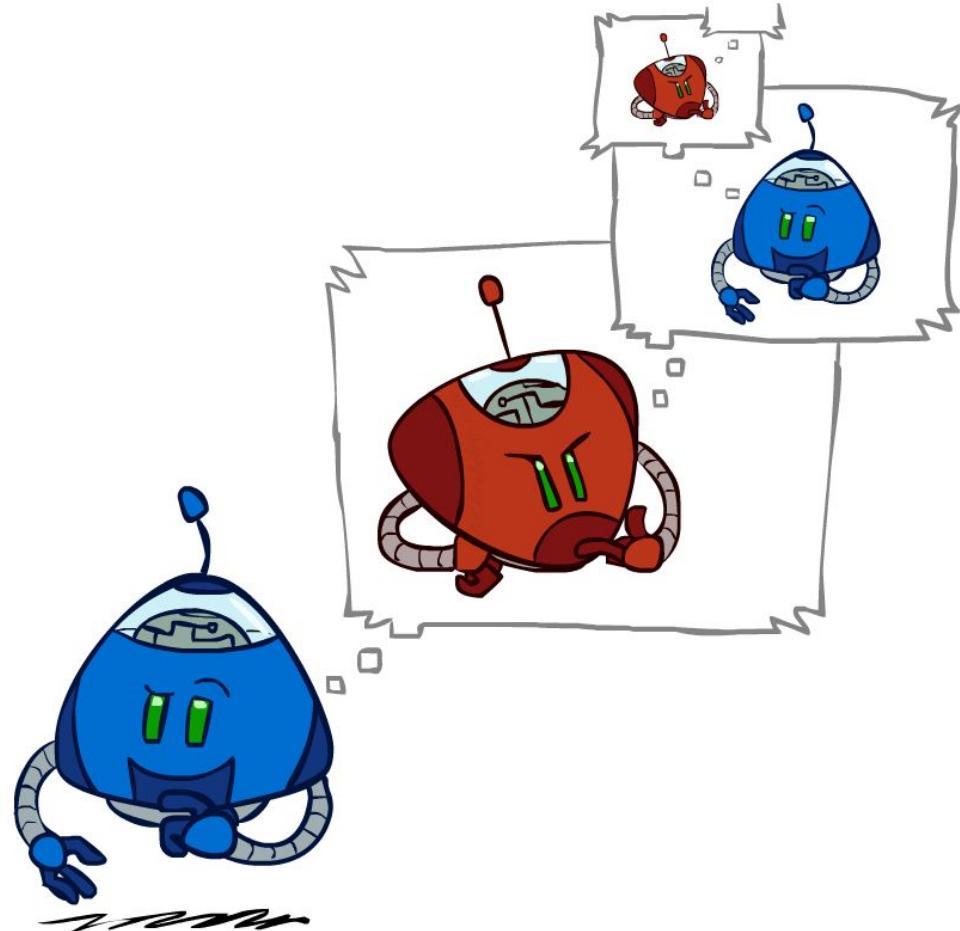
- Generalization of minimax:

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



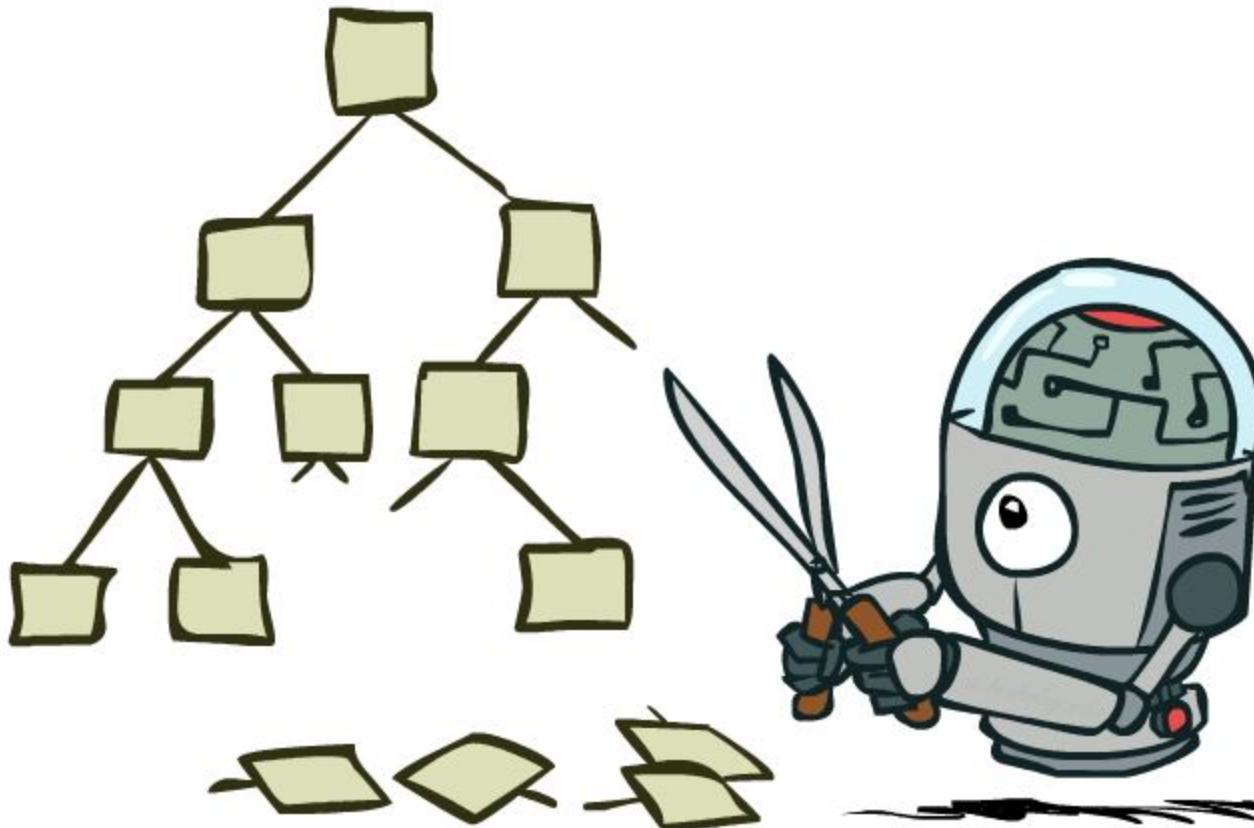
# Minimax Efficiency

- 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?

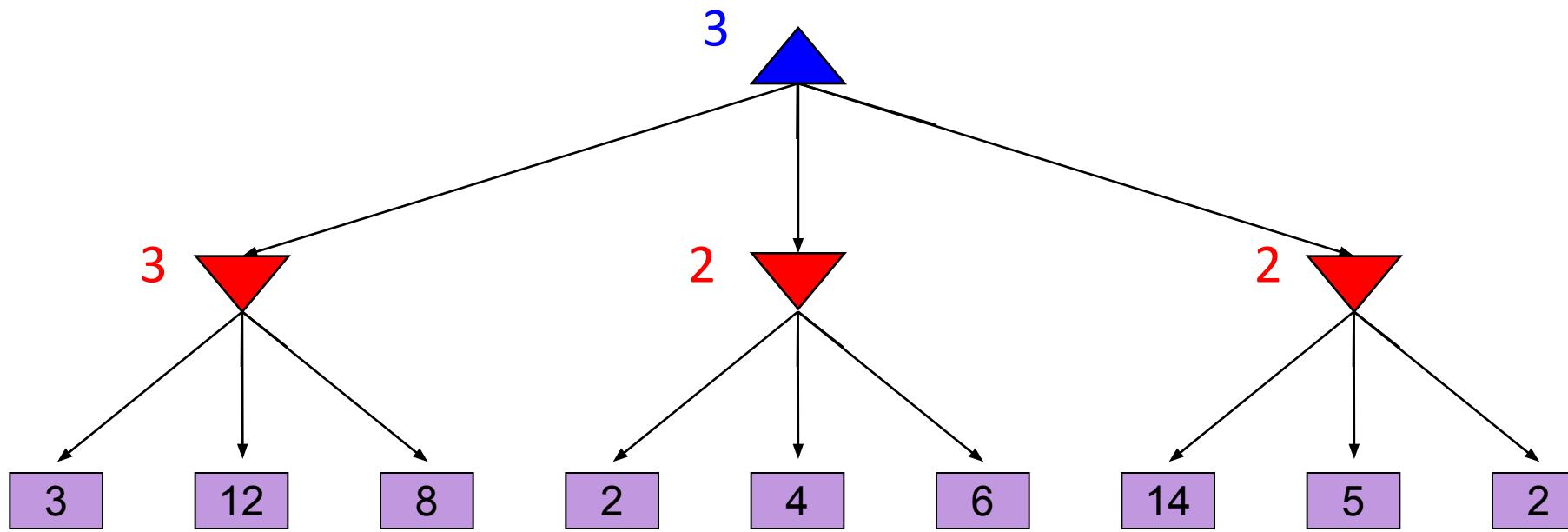


# Game Tree Pruning

---

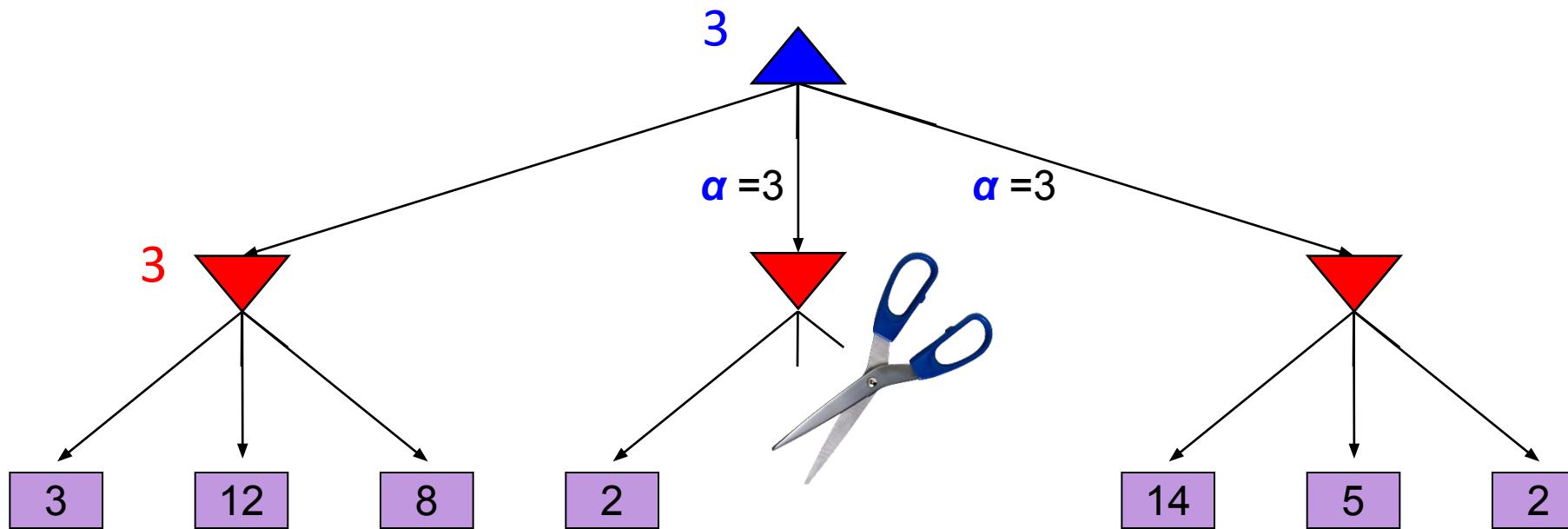


# Minimax Example



# Alpha-Beta Example

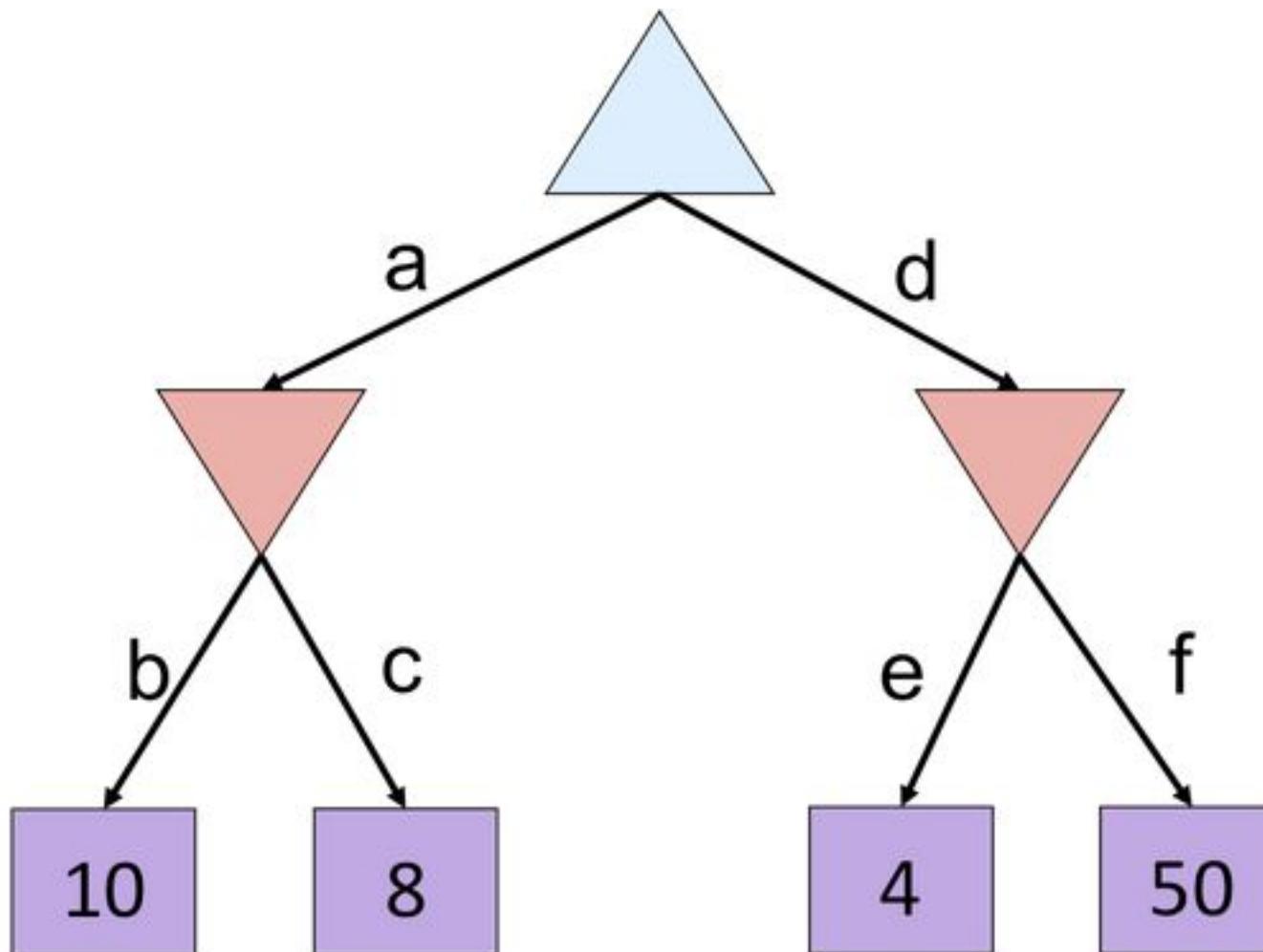
$\alpha$  = best option so far from any MAX node on this path



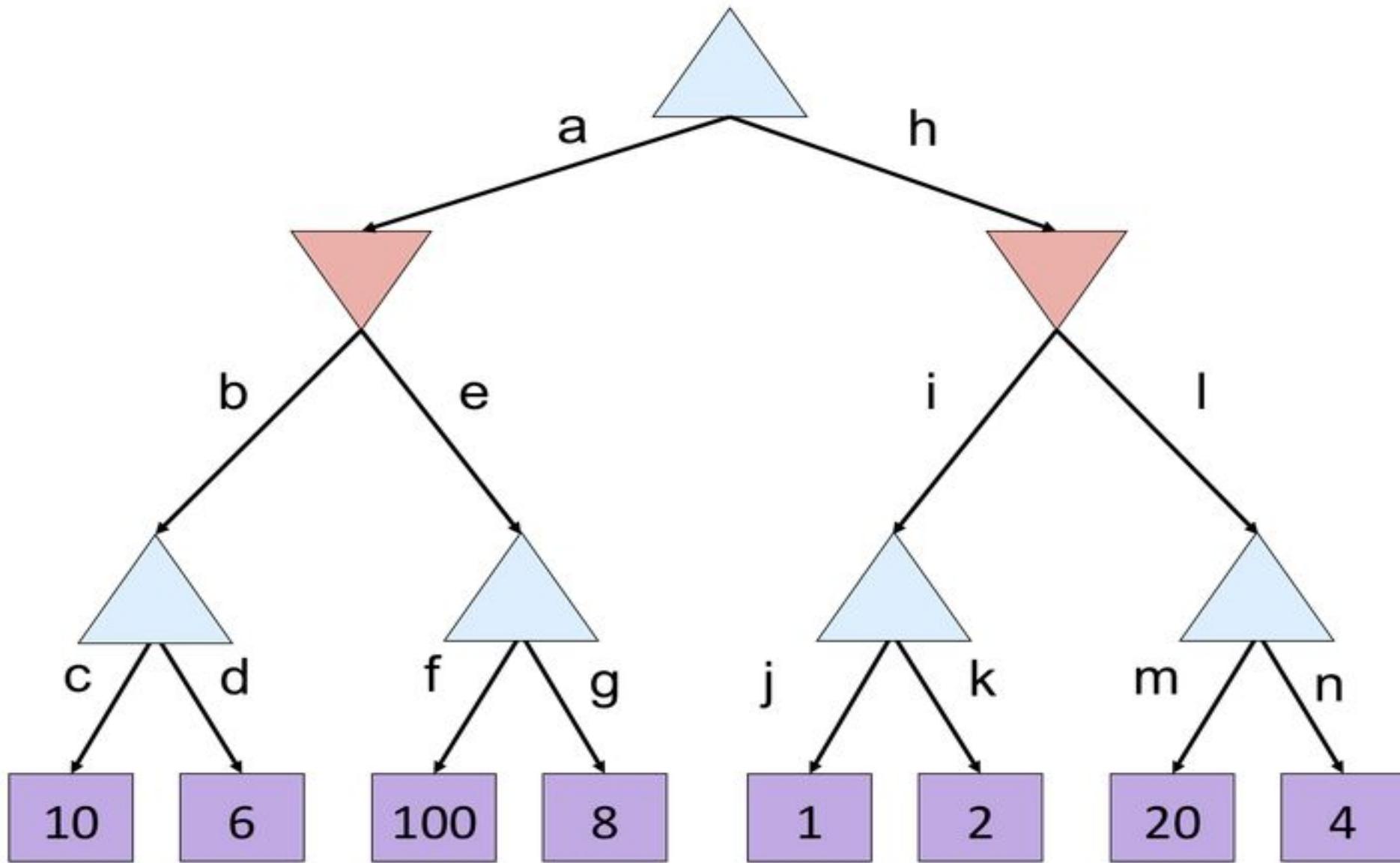
***The order of generation matters:*** more pruning is possible if good moves come first

# Alpha-Beta Quiz

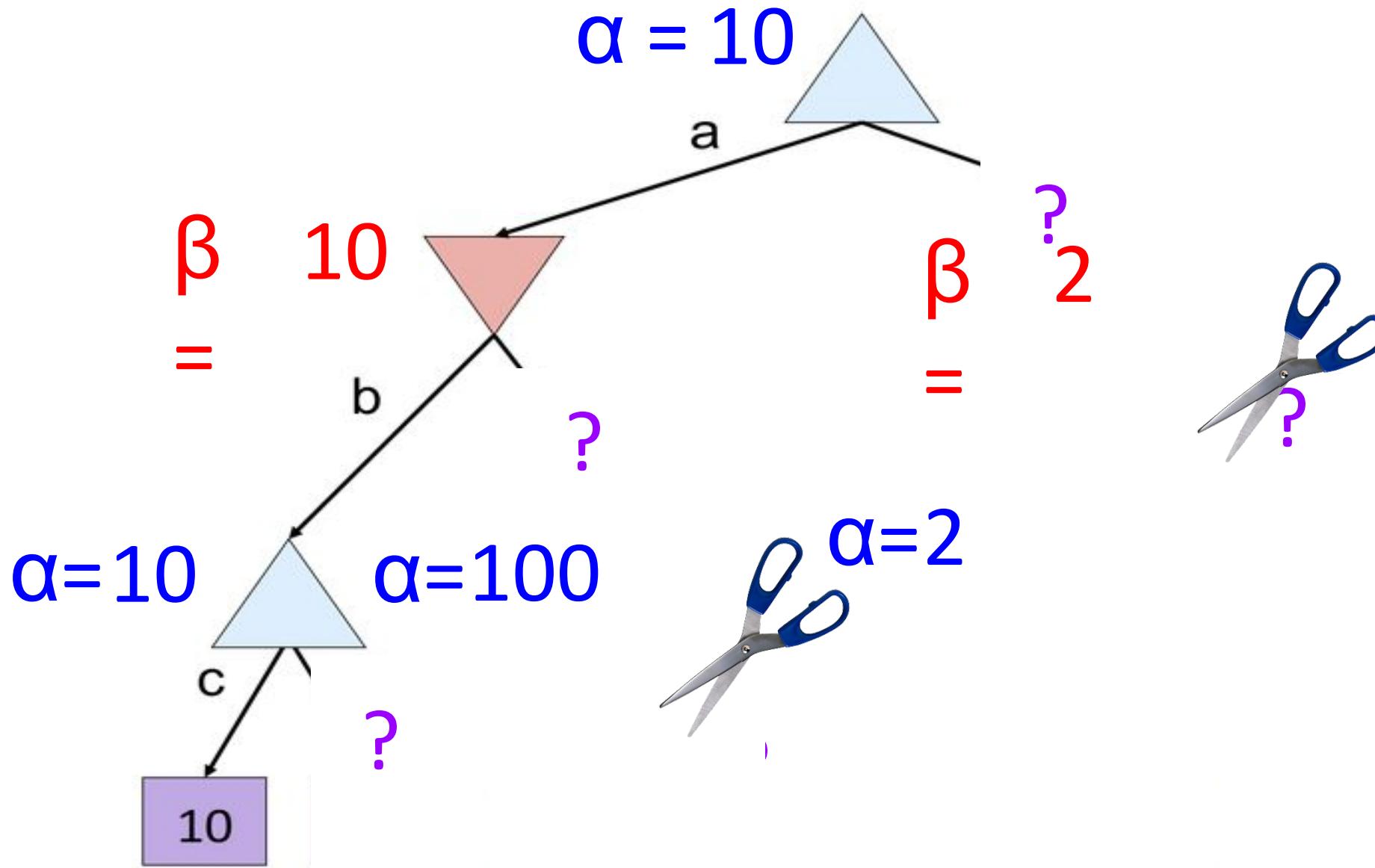
---



# Alpha-Beta Quiz 2

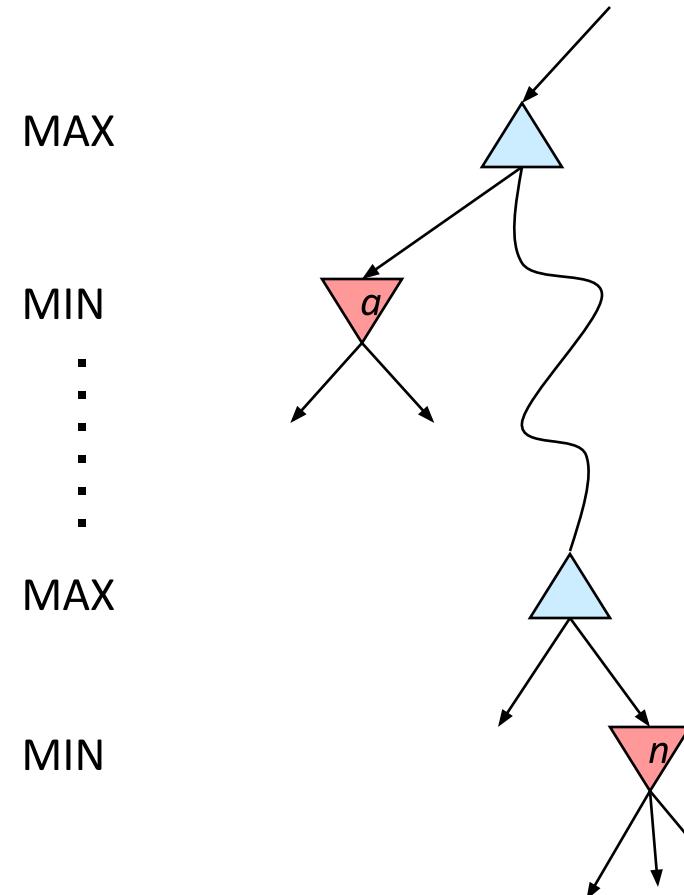


# Alpha-Beta Quiz 2



# 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  $\alpha$  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  $\alpha$ , **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  $\beta$  be the best value that **MIN** can get so far at any choice point along the current path from the root



# Alpha-Beta Implementation

$\alpha$ : MAX's best option on path to root  
 $\beta$ : 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

- Theorem: This pruning has ***no effect*** on minimax value computed for the root!

- Good child ordering improves effectiveness of pruning

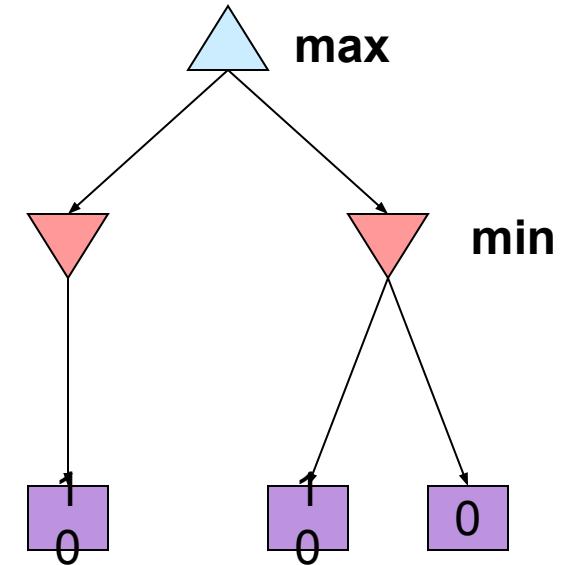
- Iterative deepening helps with this

- 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!!!!



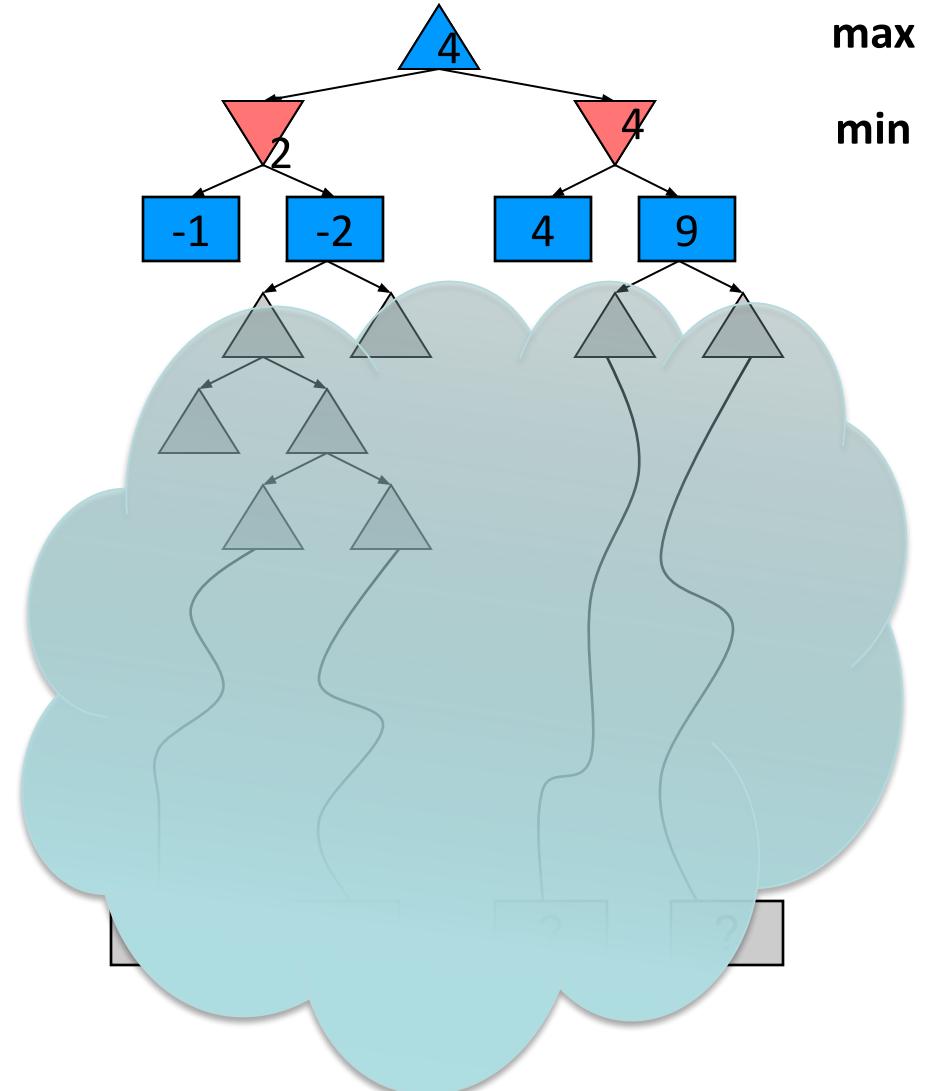
# Resource Limits

---



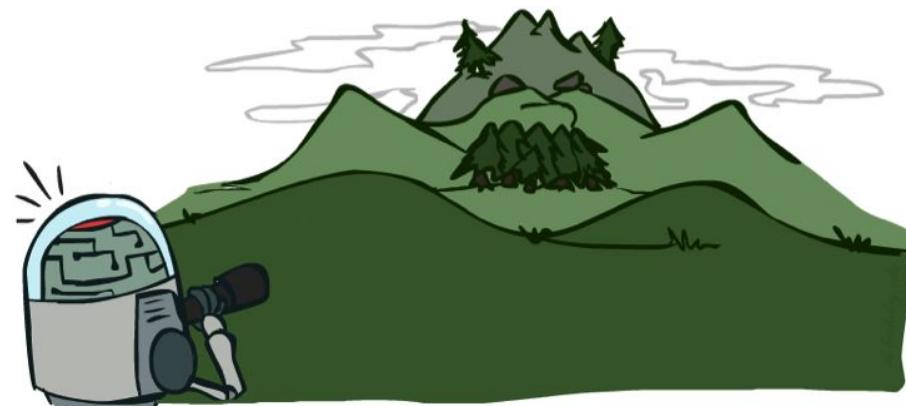
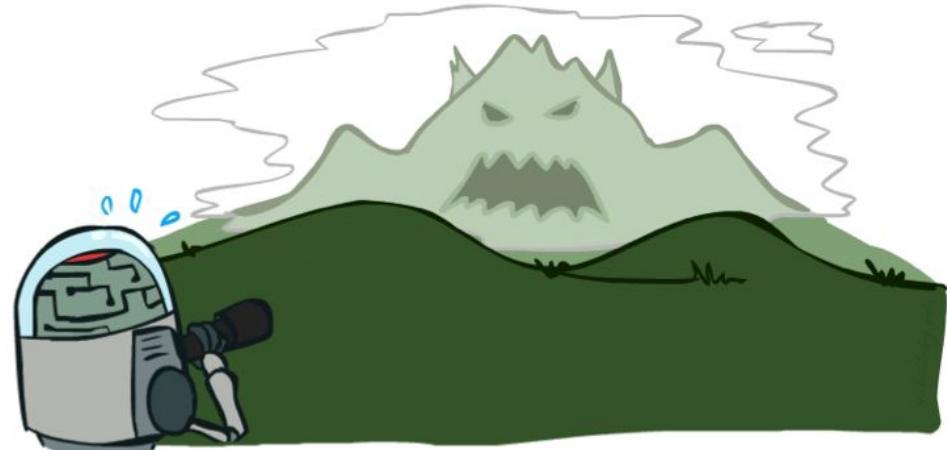
# Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution 1: Bounded lookahead
  - Search only to a preset **depth limit** or **horizon**
  - Use an **evaluation function** for non-terminal positions
- Guarantee of optimal play is gone
- More plies make a BIG difference
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - Chess with alpha-beta,  $35^{(8/2)} \approx 1M$ ; depth 8 is good



# Depth Matters

- Evaluation functions are always imperfect
- Deeper search => better play (usually)
- Or, deeper search gives same quality of play with a less accurate evaluation function
- An important example of the tradeoff between complexity of features and complexity of computation



[Demo: depth limited (L6D4, L6D5)]

# Pacman with Depth-2 Lookahead

---



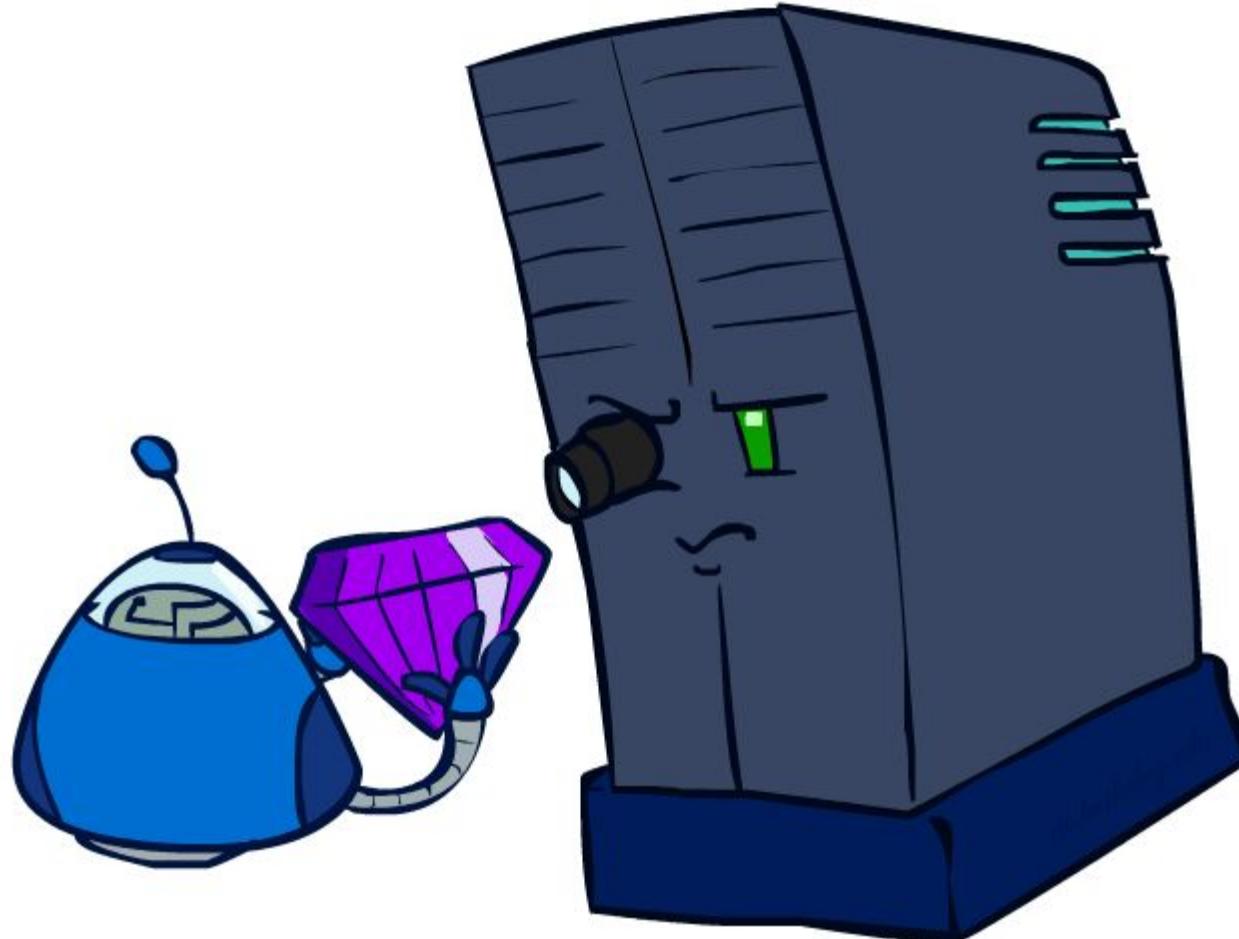
# Pacman with Depth-10 Lookahead

---



# Evaluation Functions

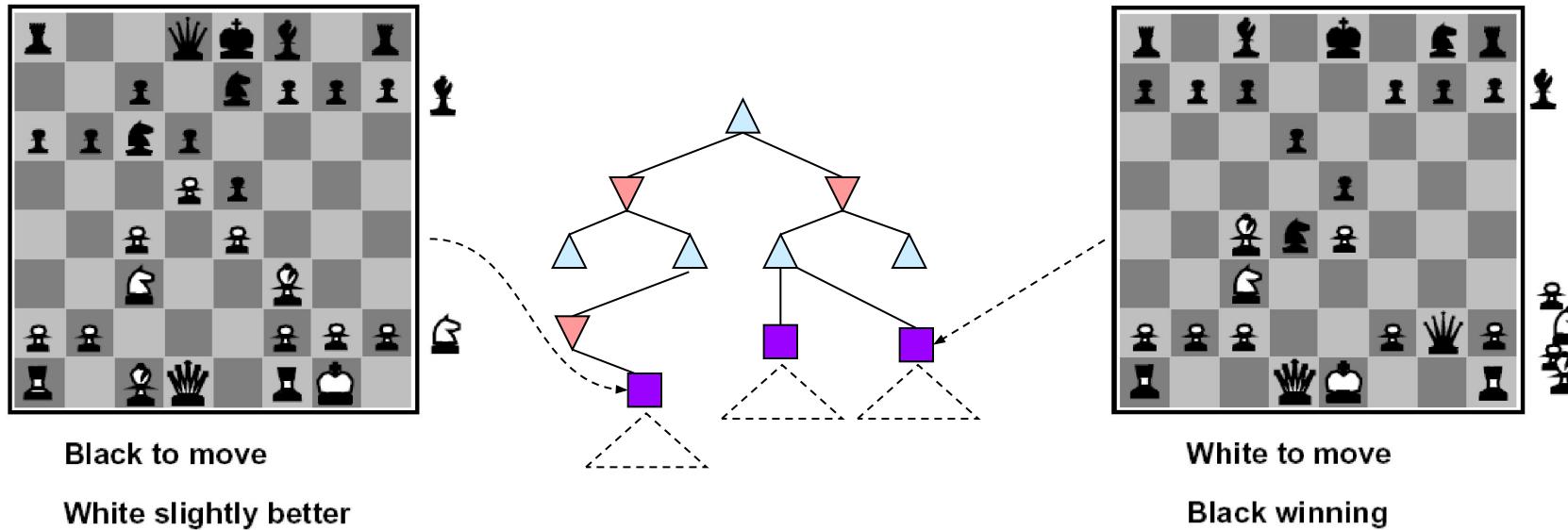
---



ms

# Evaluation Functions

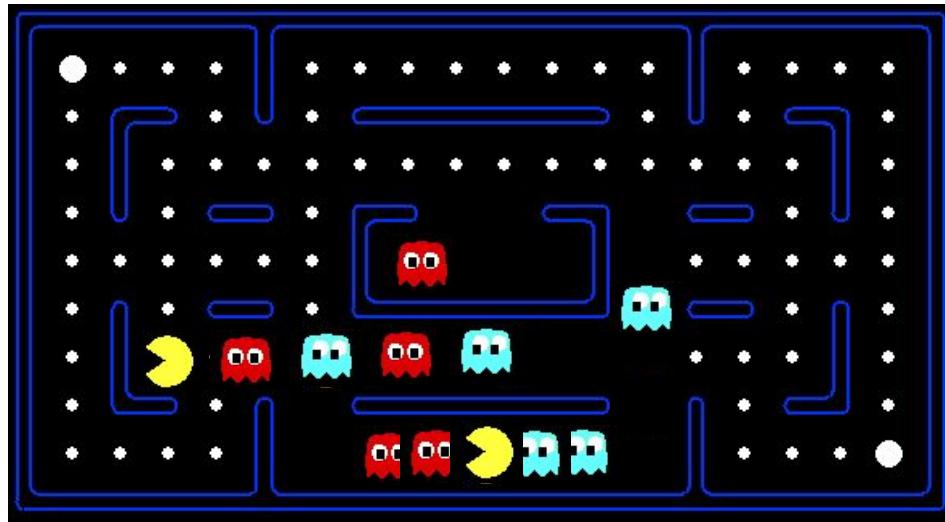
- Evaluation functions score non-terminals in depth-limited search



- Typically weighted linear sum of features:
  - $\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
  - E.g.,  $w_1 = 9$ ,  $f_1(s) = (\text{num white queens} - \text{num black queens})$ , etc.
- Or a more complex nonlinear function (e.g., NN) trained by self-play RL
- Terminate search only in **quiescent** positions, i.e., no major changes expected in feature values

# Evaluation for Pacman

---

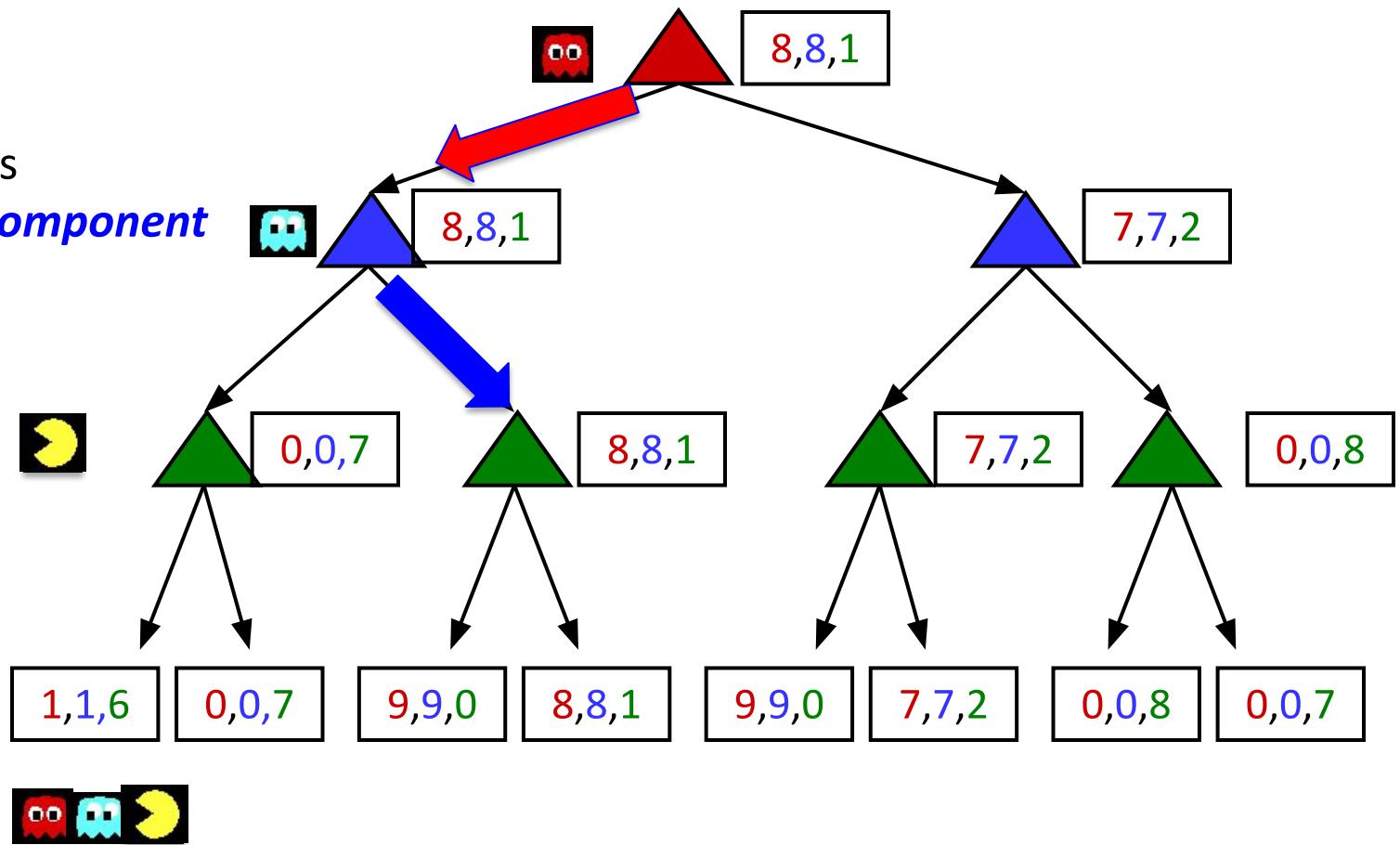
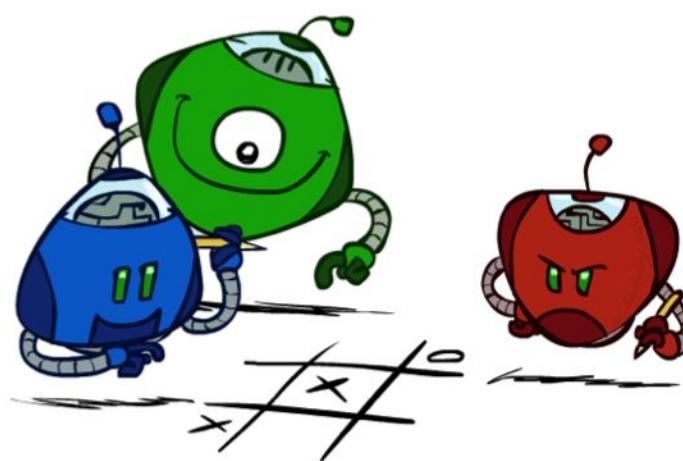


# Generalized minimax

- 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

---



# Summary

---

- Games require decisions when optimality is impossible
  - Bounded-depth search and approximate evaluation functions
- Games force efficient use of computation
  - Alpha-beta pruning
- Game playing has produced important research ideas
  - Reinforcement learning (checkers)
  - Iterative deepening (chess)
  - Rational metareasoning (Othello)
  - Monte Carlo tree search (Go)
  - Solution methods for partial-information games in economics (poker)
- Video games present much greater challenges – lots to do!
  - $b = 10^{500}$ ,  $|S| = 10^{4000}$ ,  $m = 10,000$