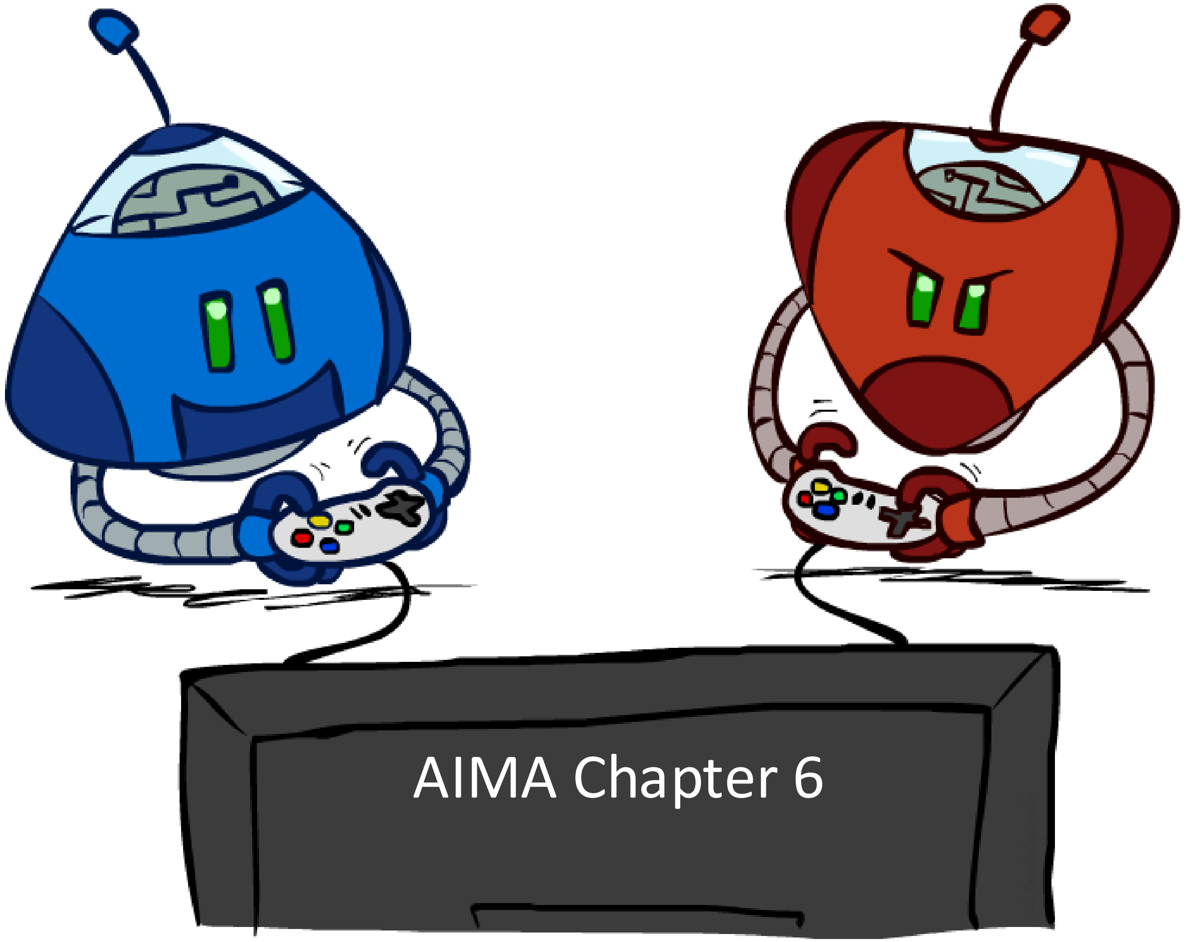
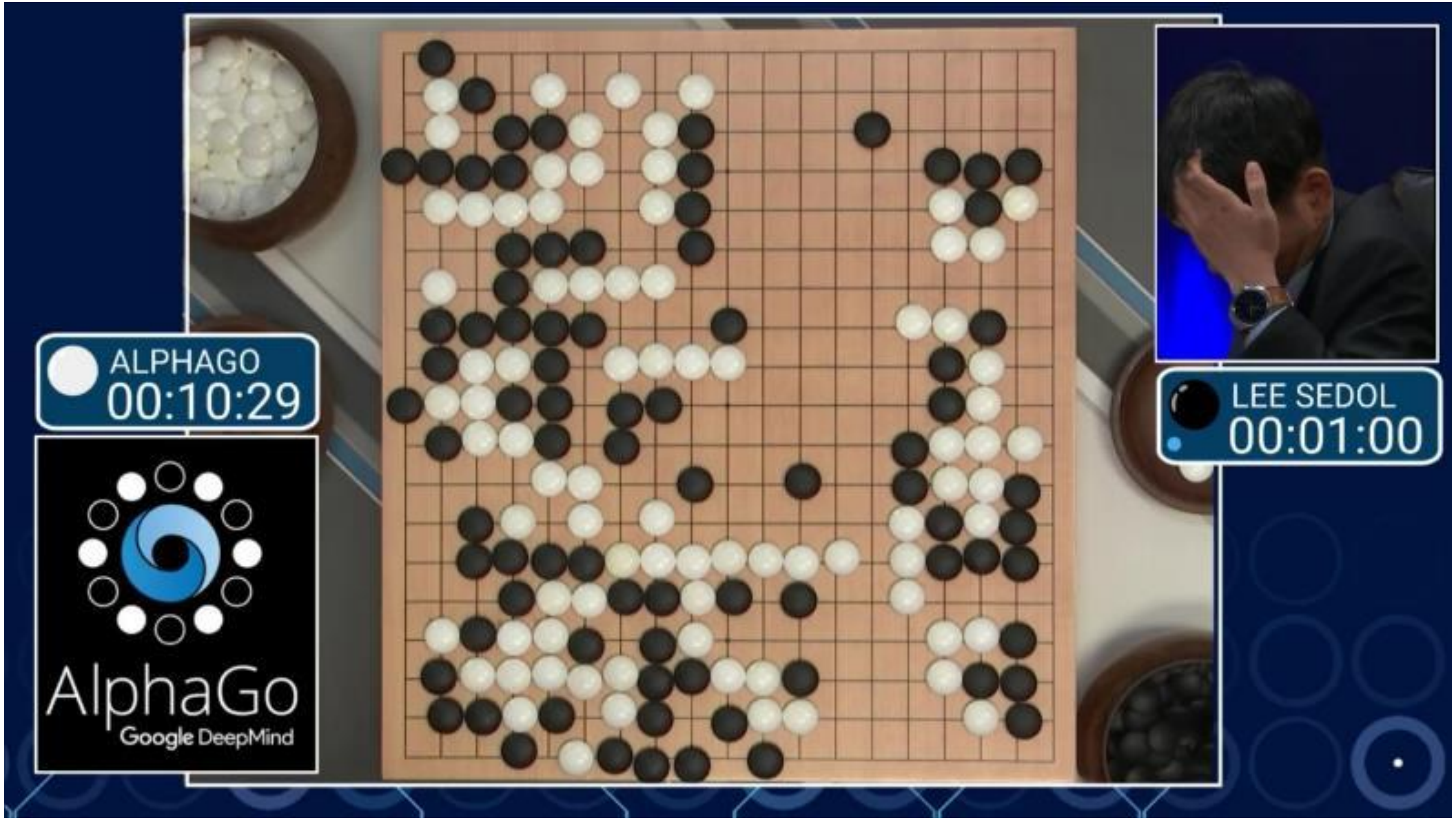
Adversarial Search



[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley (ai.berkeley.edu).]

AlphaGo (2016)



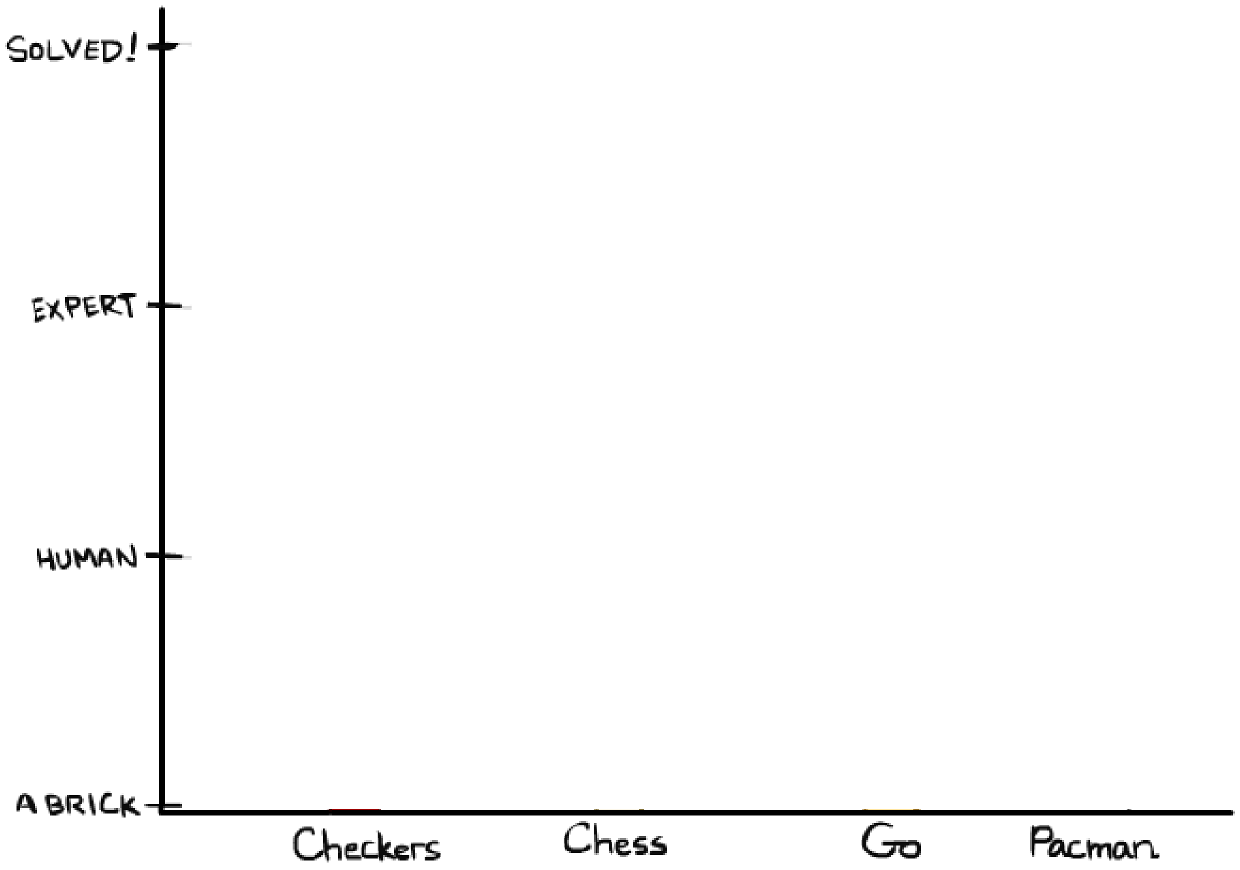
Game Playing State-of-the-Art

- **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: AlphaGo defeats human champion! Uses Monte Carlo Tree Search, learned evaluation function.**

- **Pacman**



Mahjong


微软AI最新突破：超级麻将AI Suphx
探索隐藏的不确定性
Super Mahjong AI Suphx: exploring hidden information
Hidden information
Next challenge, Mahjong!
Observable status
天在上海我要向大家宣布。微软亚洲研究院创造出了历史上最强的麻将AI。
Today in Shanghai, I would like to announce to you that Microsoft Research Asia has created the strongest mahjong AR in history
2019世界人工智能大会
WORLD ARTIFICIAL INTELLIGENCE CONFERENCE
WAIC
2019世界人工智能大会
WORLD ARTIFICIAL INTELLIGENCE CONFERENCE
2019世界人工智能大会


隐藏的不确定信息
麻将
10^{48}
桥牌
10^{15}
德州扑克
围棋
10^{3}
0
10^{67}
10^{121}
10^{162}
10^{172}
可观测的状态信息

Online evaluation: highest ranking in the expert room

- Suphx is the first and only AI to achieve 10 DAN.
- Suphx is the highest ranking in the Expert room, the only room that AI is allowed to play in.

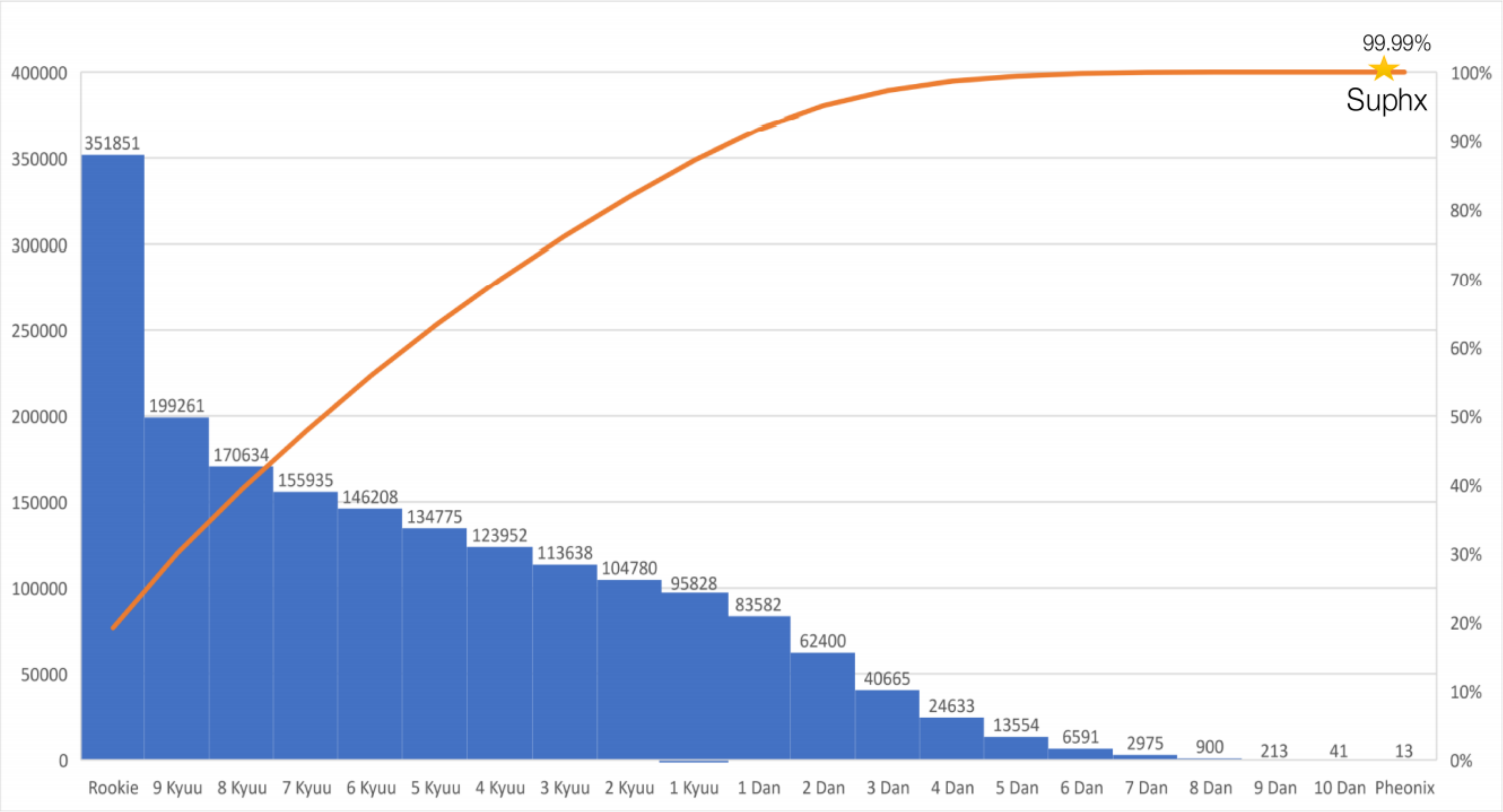
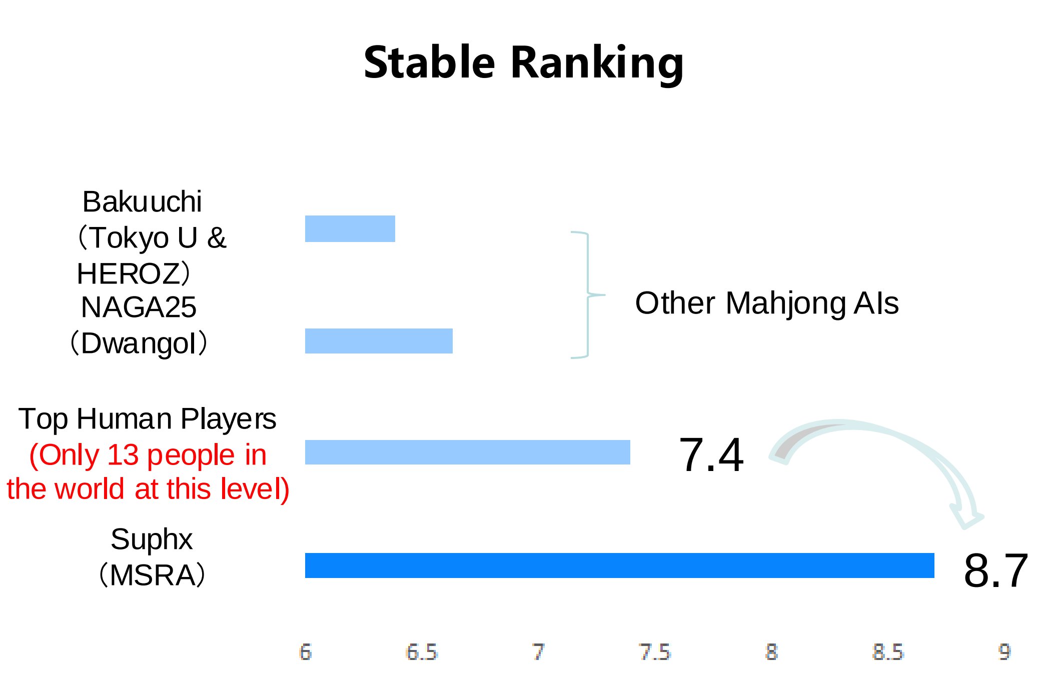


Figure 11: Distributions of record ranks of human players in Tenhou. Each bar indicates the number of human players above a certain level in Tenhou.

Evaluation on Tenhou

- Played 5,000+ games in Expert room of Tenhou since March 2019 and achieved 10 DAN in June 2019.



I've watched Suphx's playing for quite a while. I feel like "it is even stronger than me."

ASAPIN
The first player to achieve the highest DAN on 4-player Mahjong

Futokunaio
The only player who achieved the highest DAN on both 3-player and 4-player Mahjong

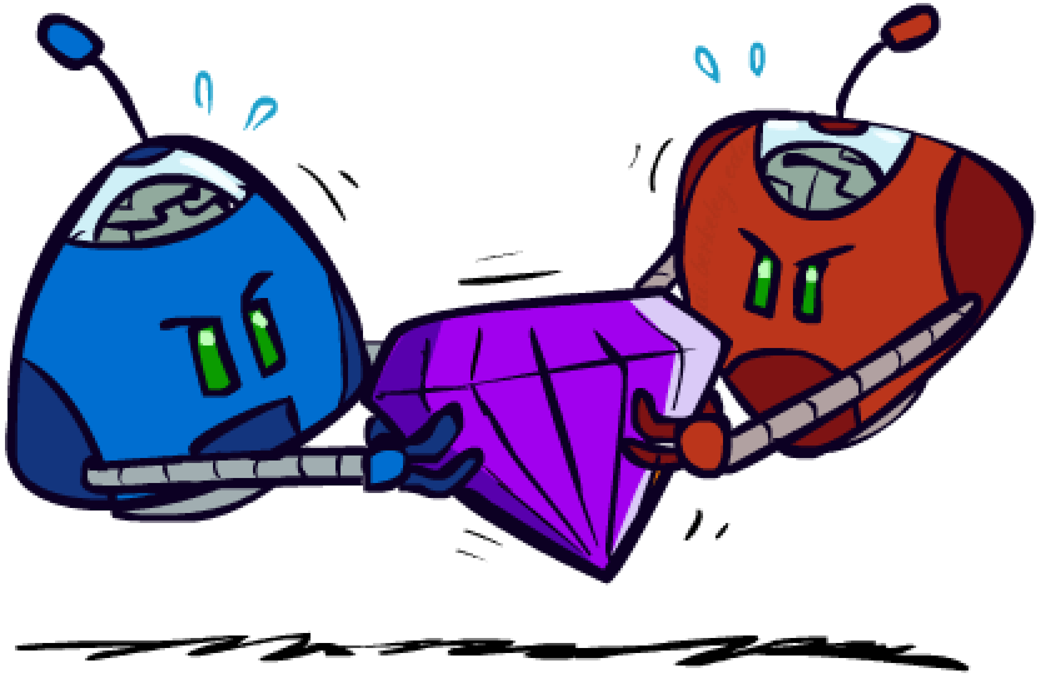
I've watched 300+ games of Suphx and it was really doing great. I am studying its style and I stopped watching human players' games.

* Suphx was trained with 30-million rounds of self-play on 100 GPUs on Azure (which takes about 2 weeks)

Game review by top human players

Some reviews are by highest ranking professional players in Japan.
“best textbook”

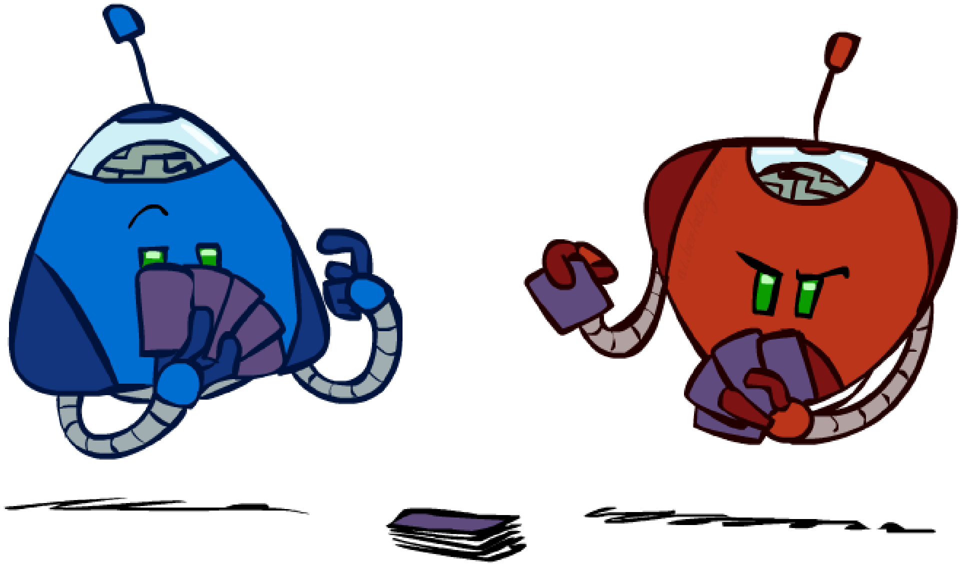
Adversarial Games



Types of Games

- Many different kinds of games!

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

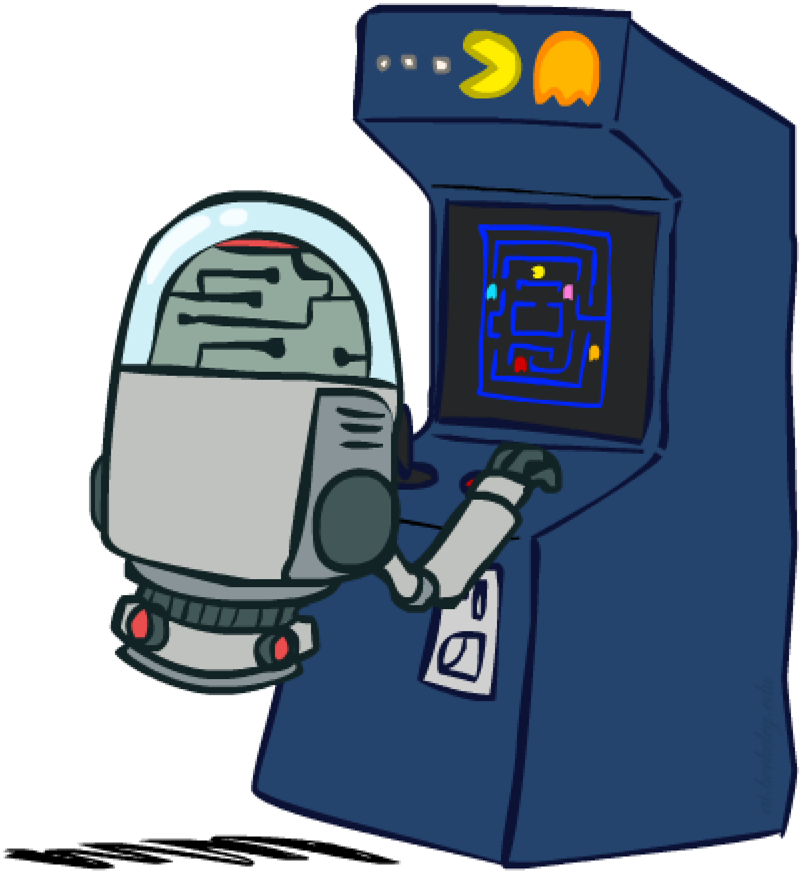


- Want algorithms for calculating a strategy (policy) which recommends a move from each state

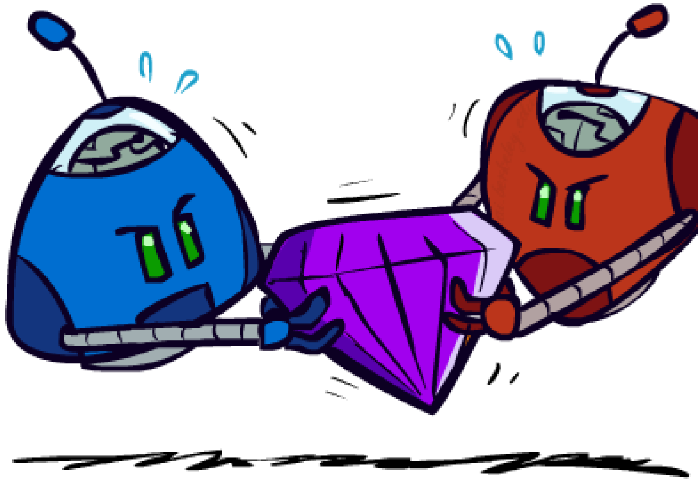
Deterministic Games

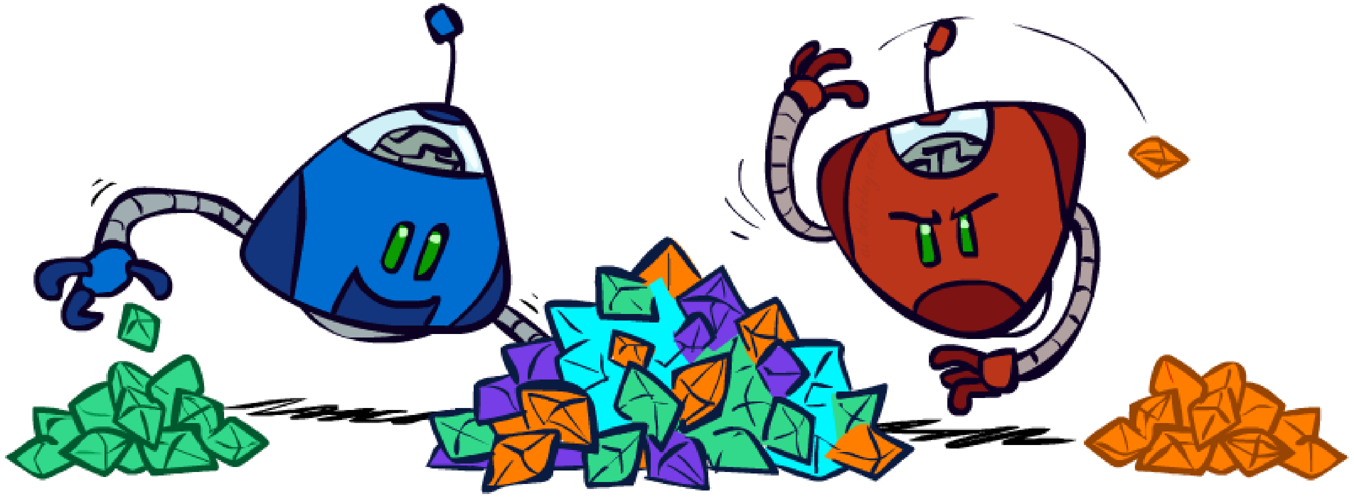
- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: $S \times A \rightarrow S$
  - Terminal Test: $S \rightarrow \{t,f\}$
  - Terminal Utilities: $S \times P \rightarrow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

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



Zero-Sum Games

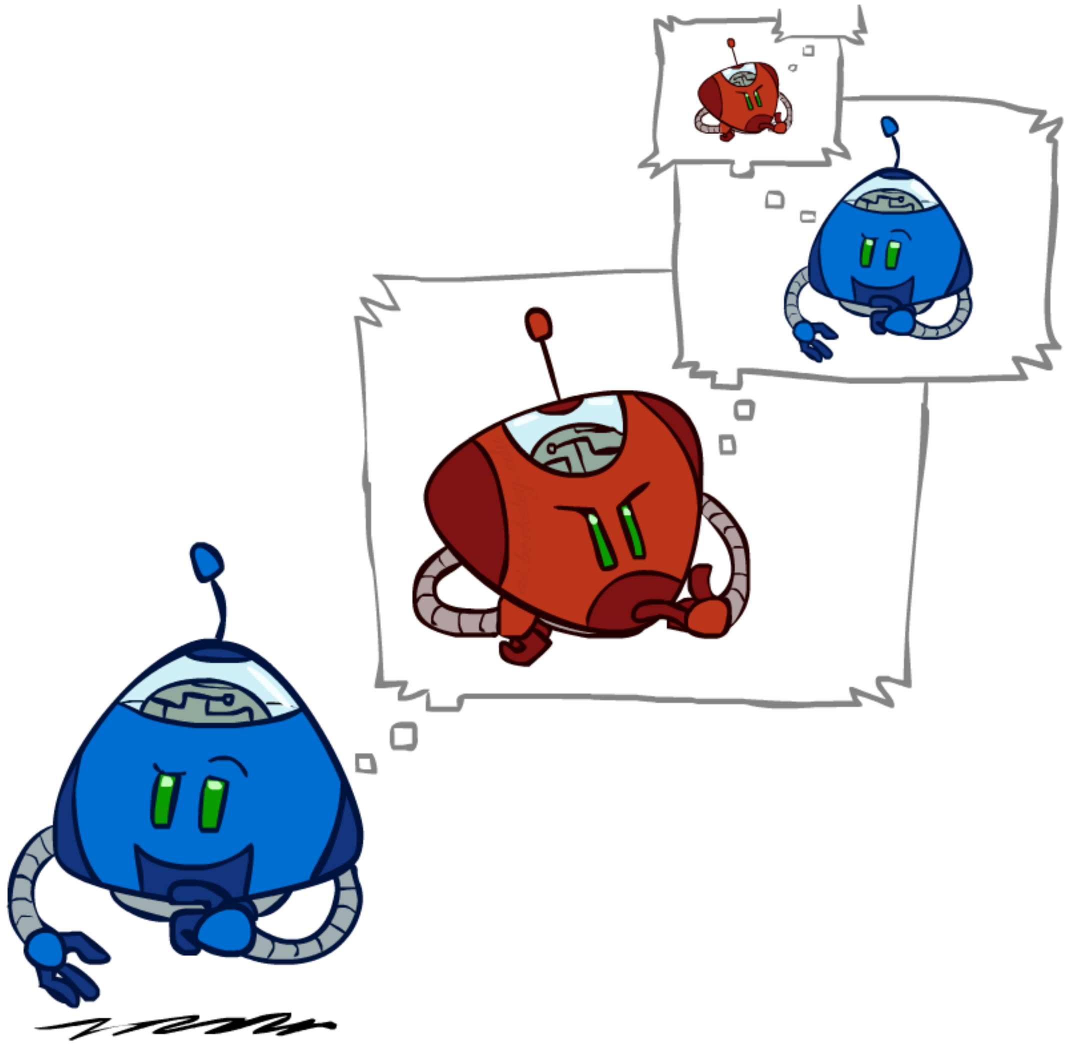




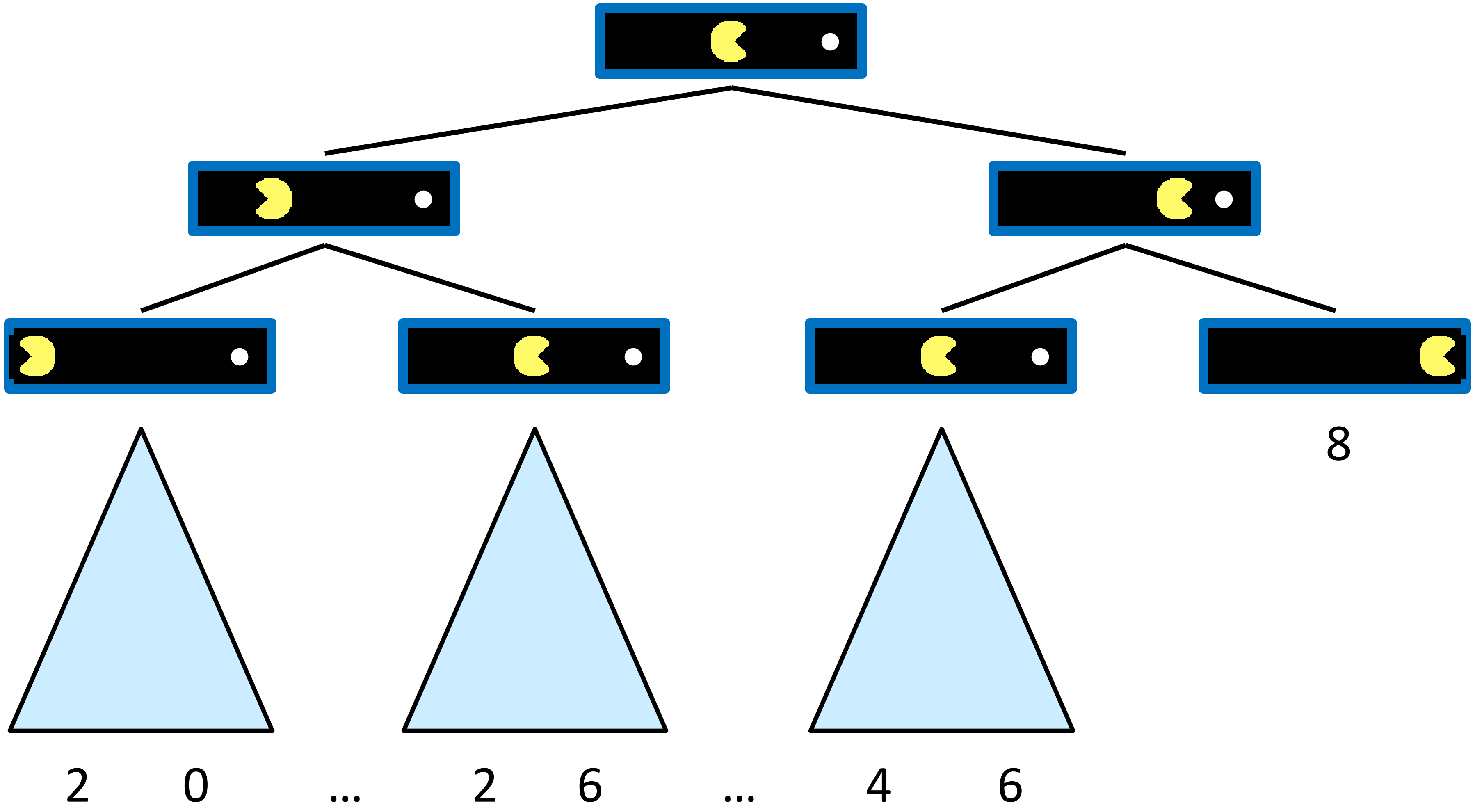
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition

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

Adversarial Search



Single-Agent Trees



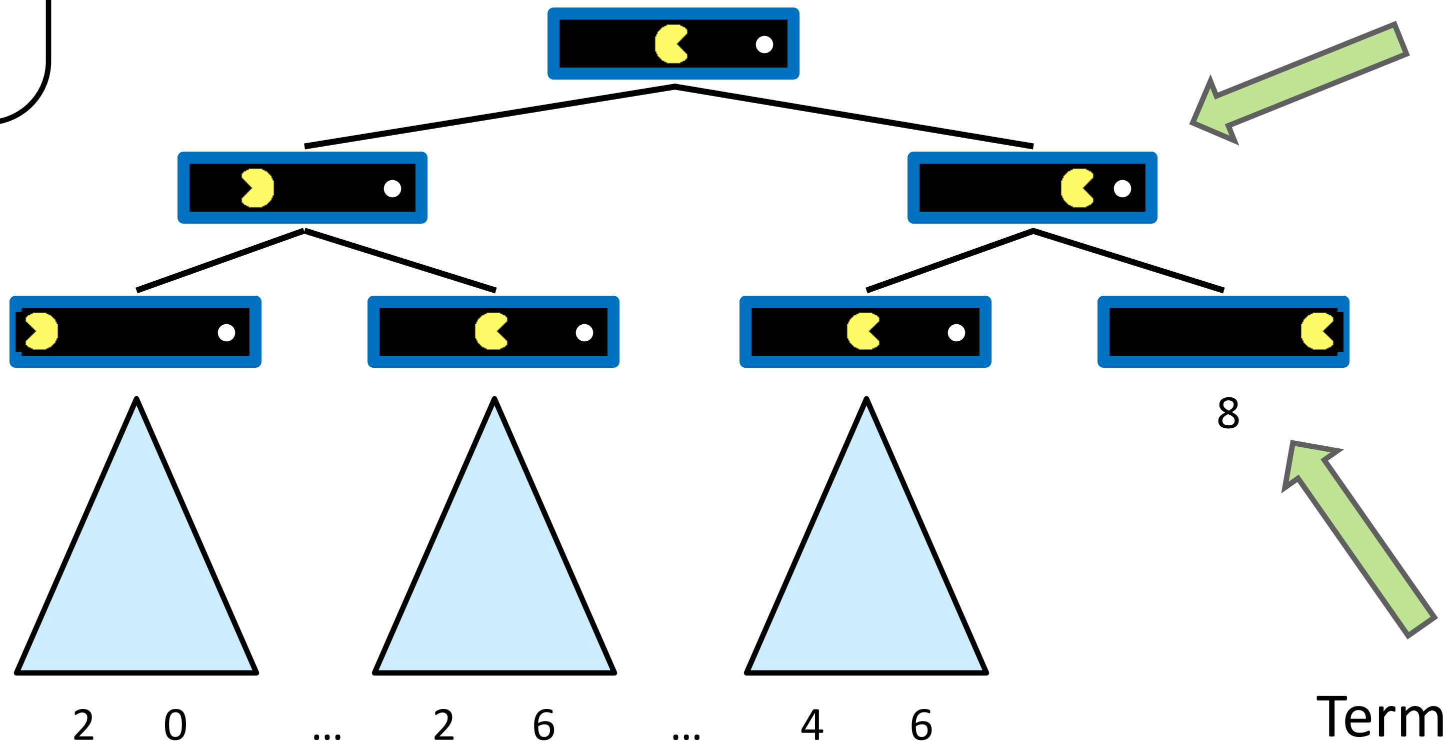
Value of a State

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

Policy: the agent should choose an action leading to the state with the largest value

Non-Terminal States:

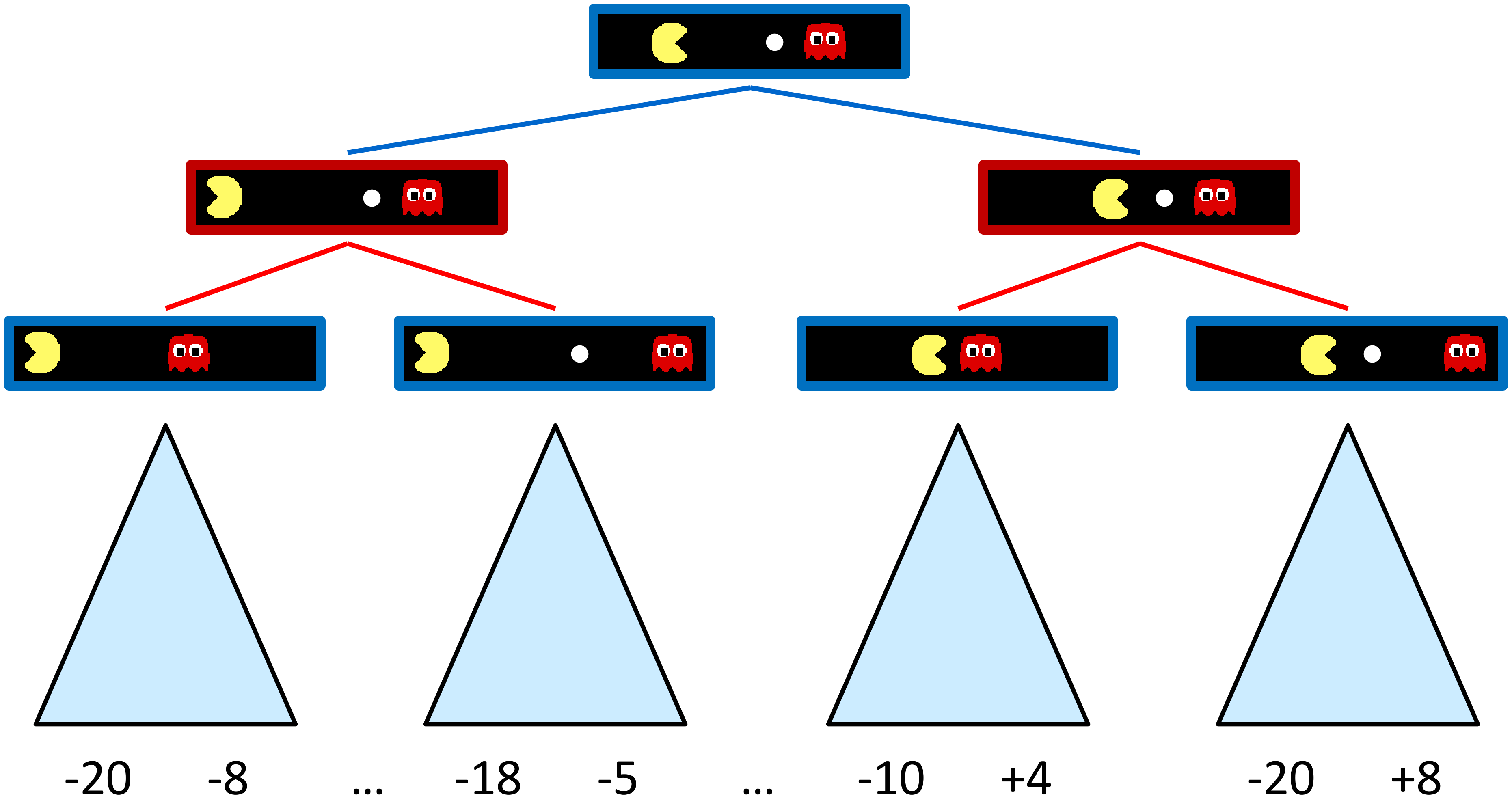
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



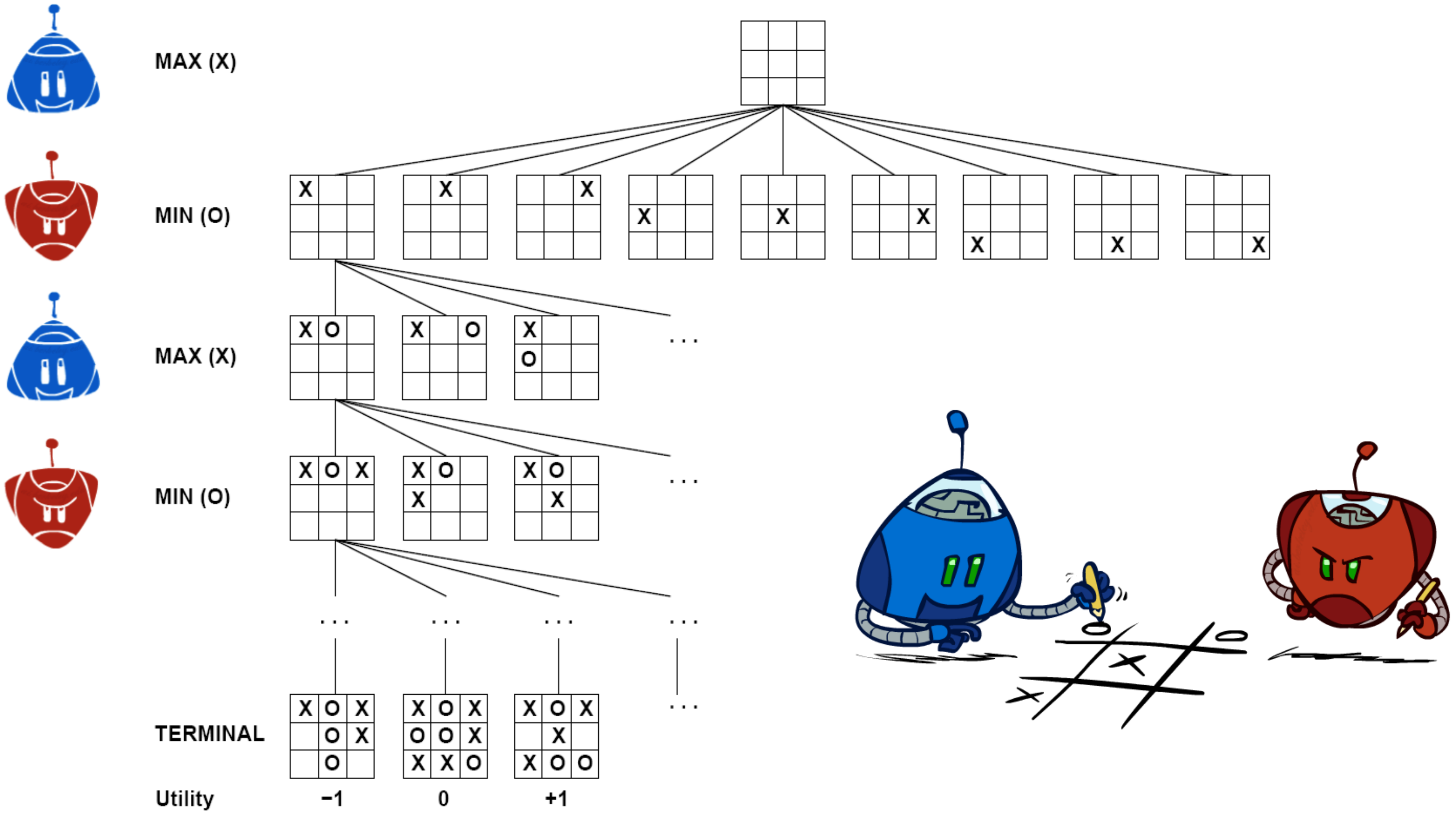
Terminal States:

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

Adversarial Game Trees



Tic-Tac-Toe Game Tree



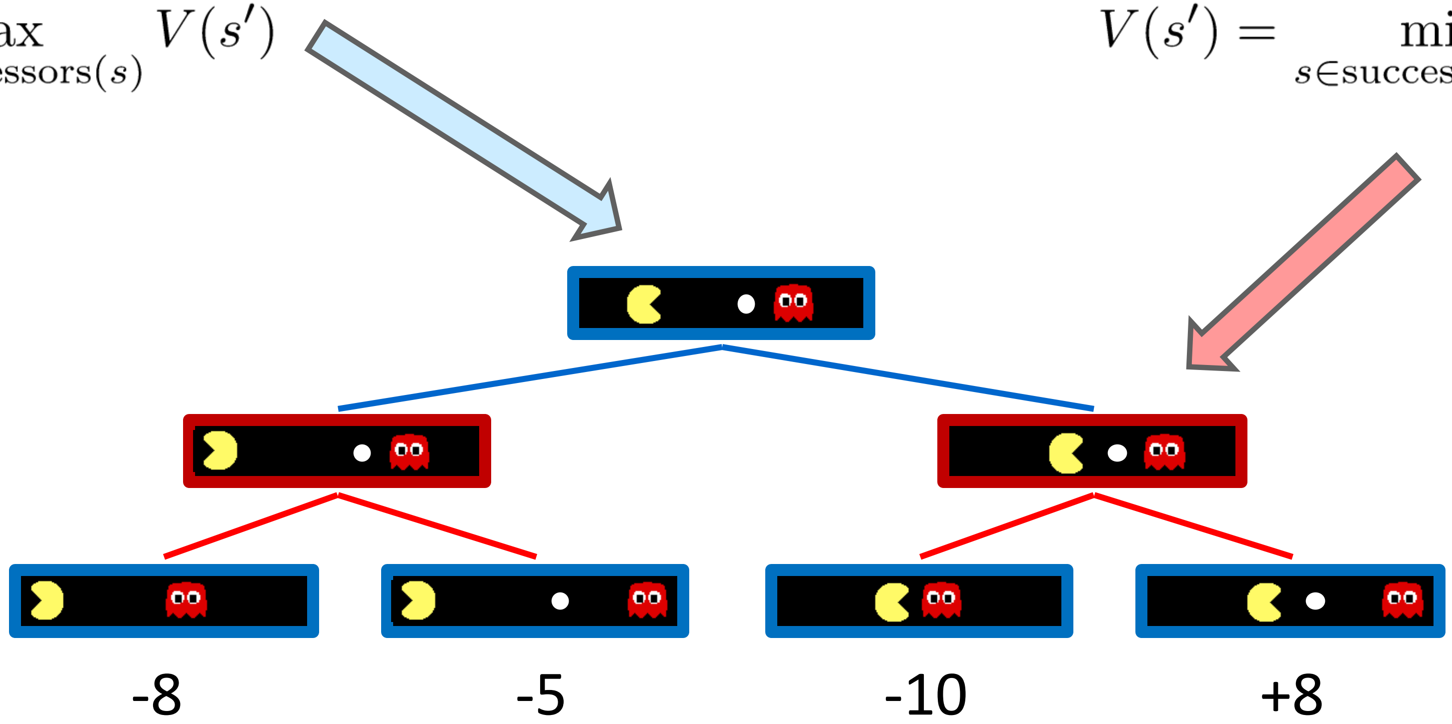
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

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



Policy: the agent should choose an action leading to the state with the largest value

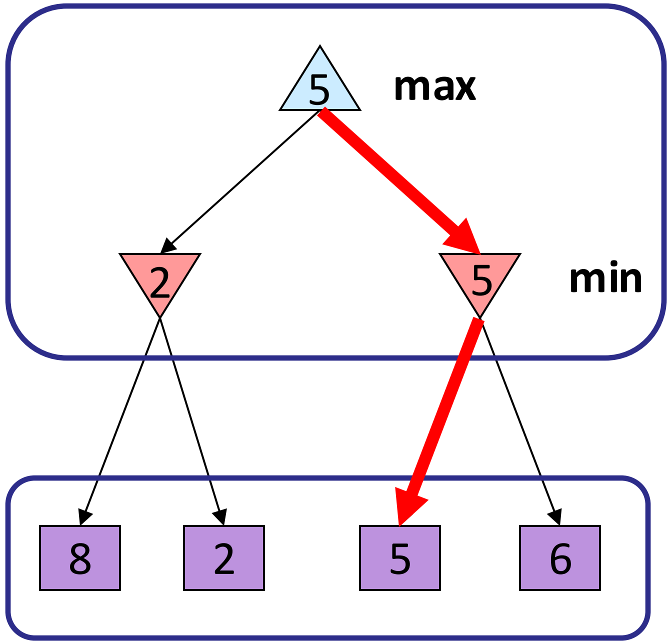
Terminal States:

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

Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - Players alternate turns
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary

Minimax values: computed recursively



Terminal values: part of the game

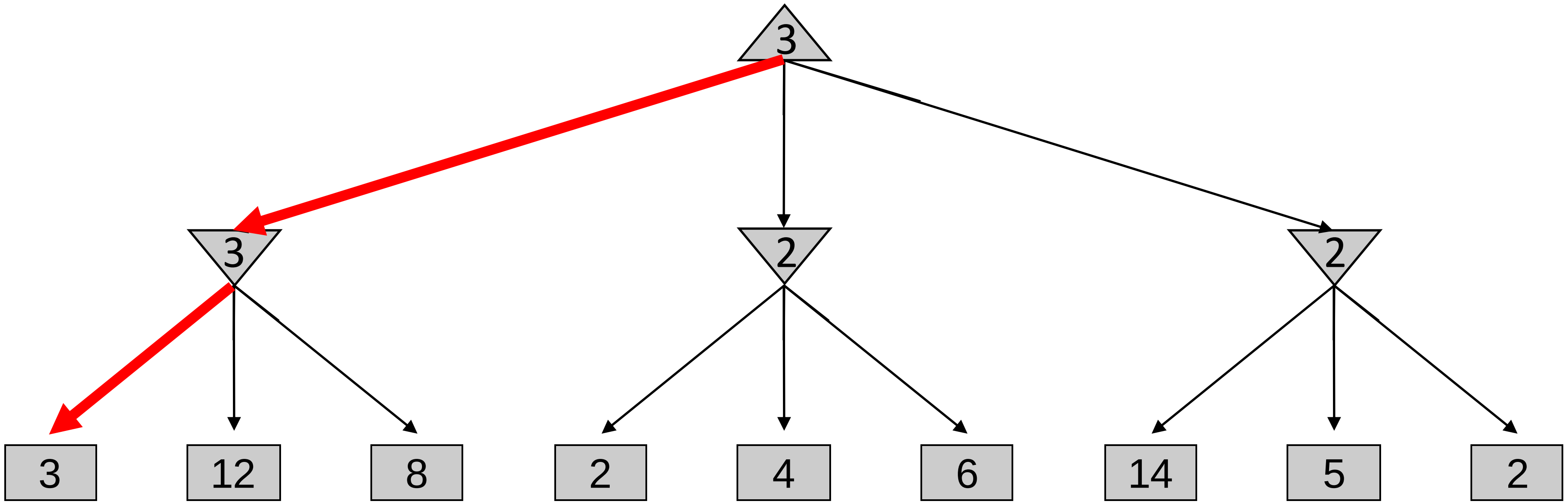
Minimax Implementation

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the 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
```

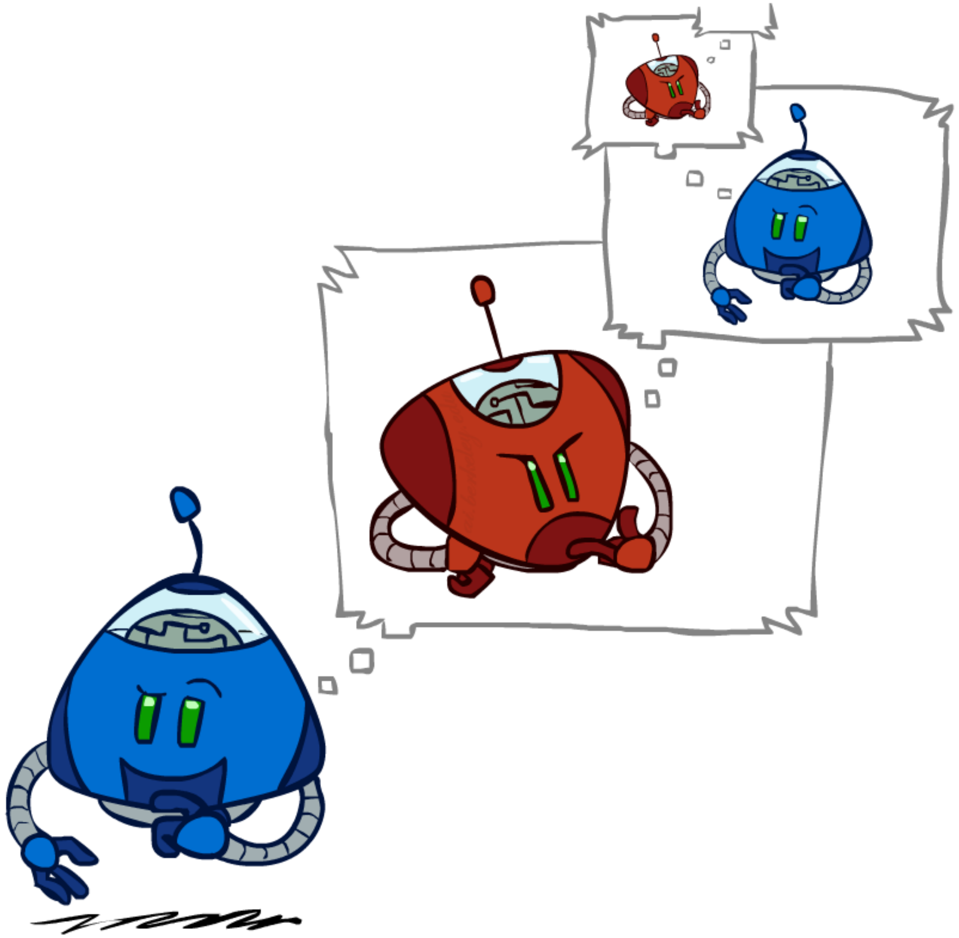
Minimax Example



Minimax Efficiency

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

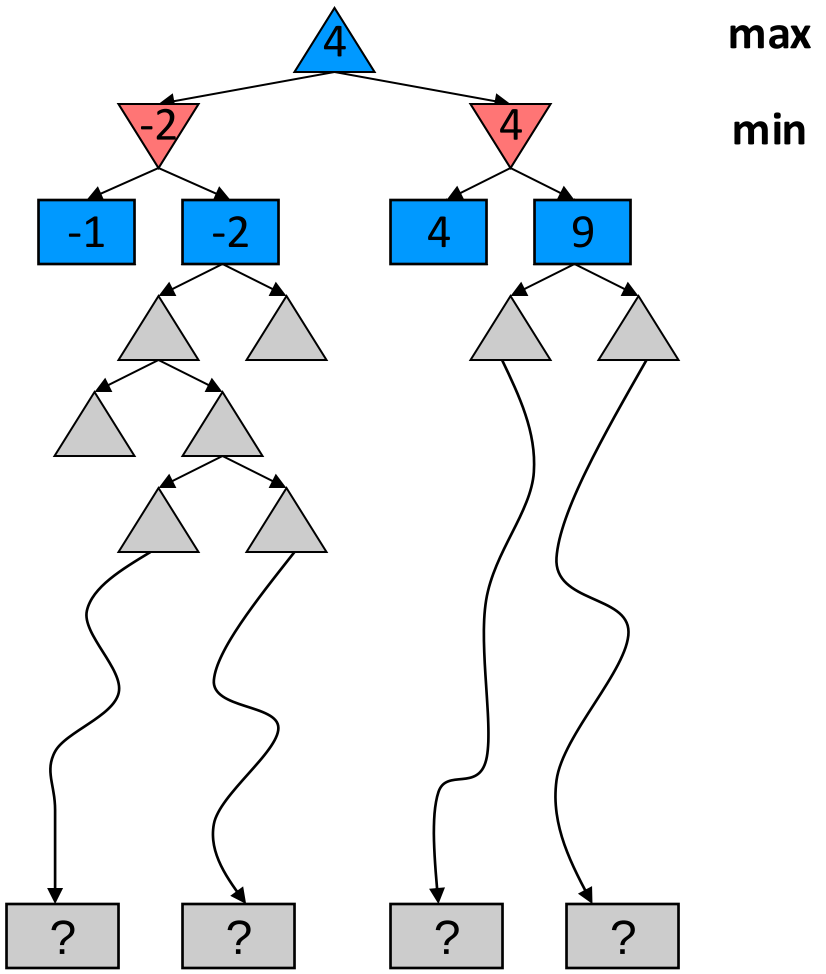
- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Resource Limits

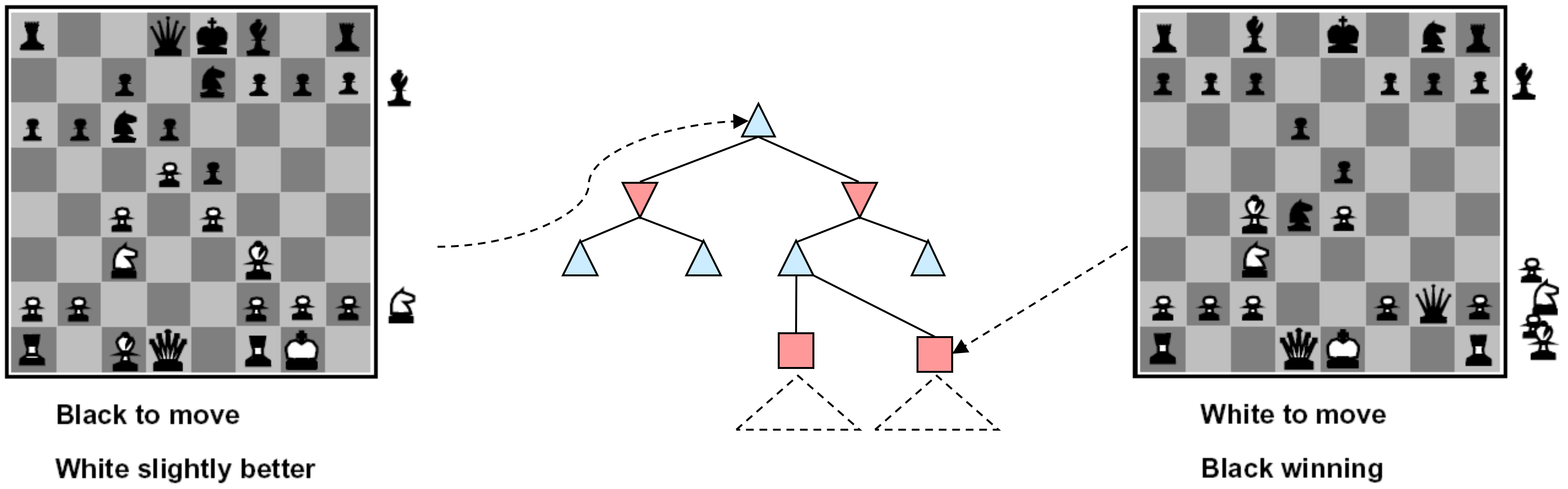
Resource Limits

- 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 depth makes a BIG difference



Evaluation Functions

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



- Ideal function: returns the actual minimax value of the position
- A simple solution in practice: weighted linear sum of features:

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

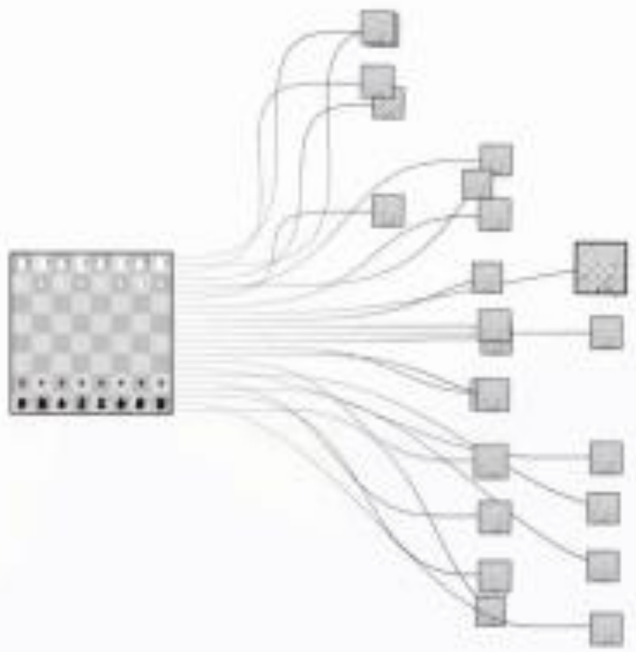
- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Evaluation Functions

- More advanced solutions
  - Monte Carlo Tree Search
    - Randomly choose moves until the end of game
    - Repeat for many many times
    - Evaluate the state based on these simulations, e.g., the winning rate
  - Convolutional Neural Network (value network in AlphaGo)
    - Trained from records of game plays to predict a score of the state

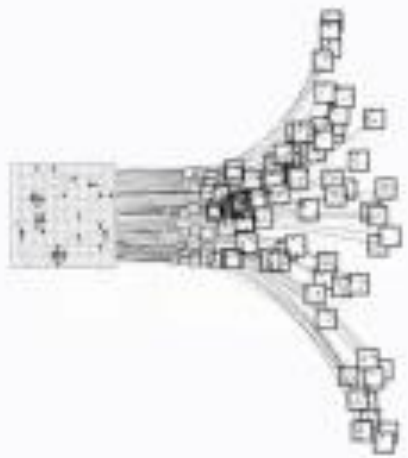
Branching Factor

- Chess



Branching Factor

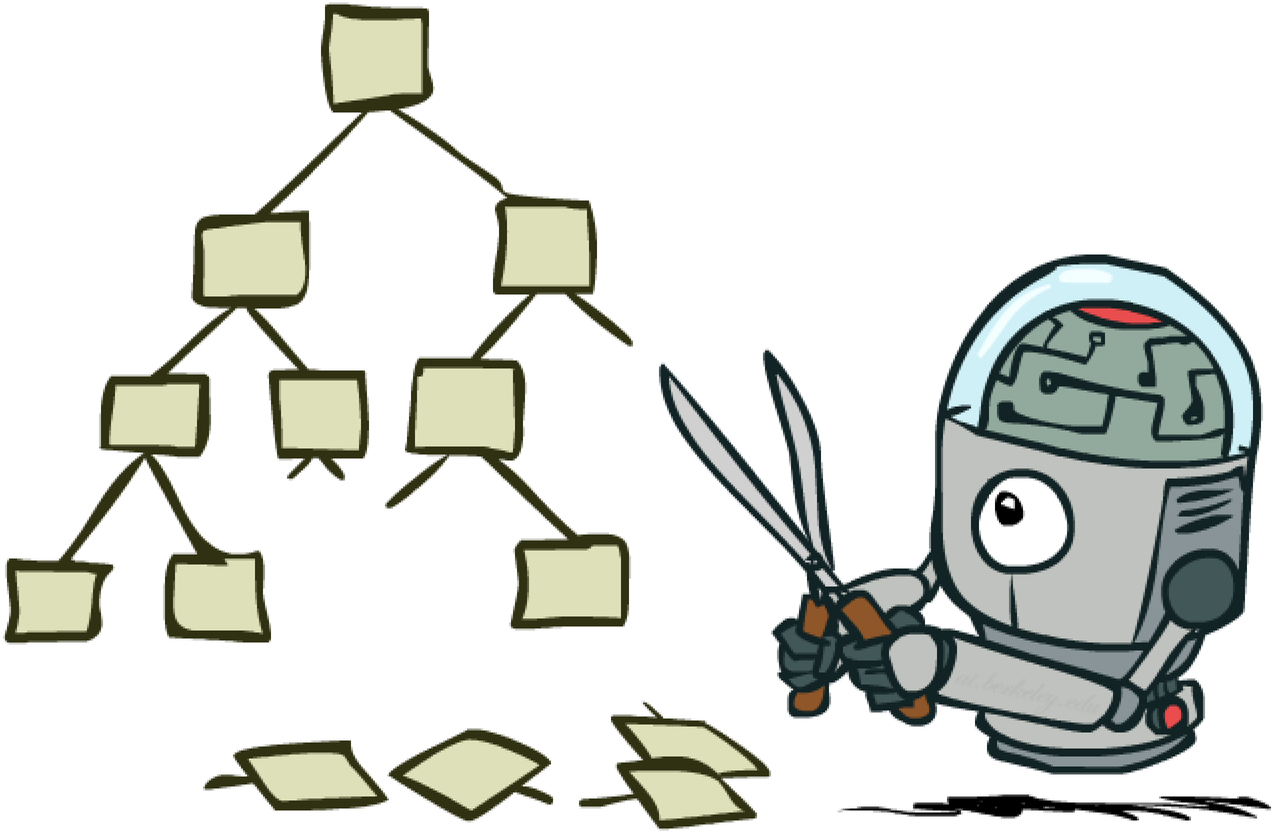
- Go



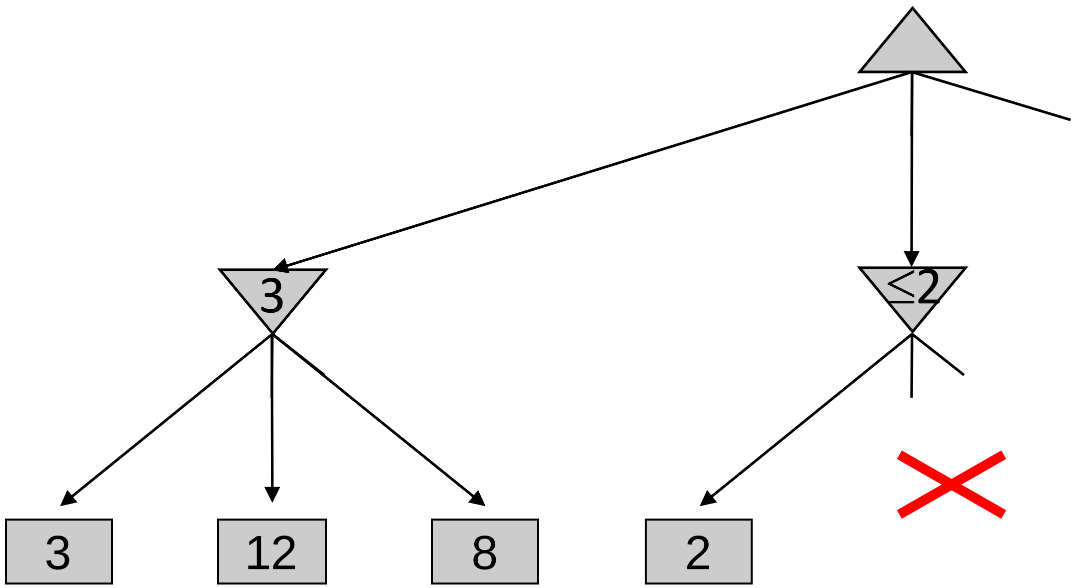
Branching Factor

- Go has a branching factor of up to 361
- Idea: limit the branching factor by considering only good moves
  - AlphaGo uses a Convolutional Neural Network (policy network)
    - Trained from records of game plays
    - Trained using reinforcement learning
      - AlphaGo Zero uses RL only

Game Tree Pruning

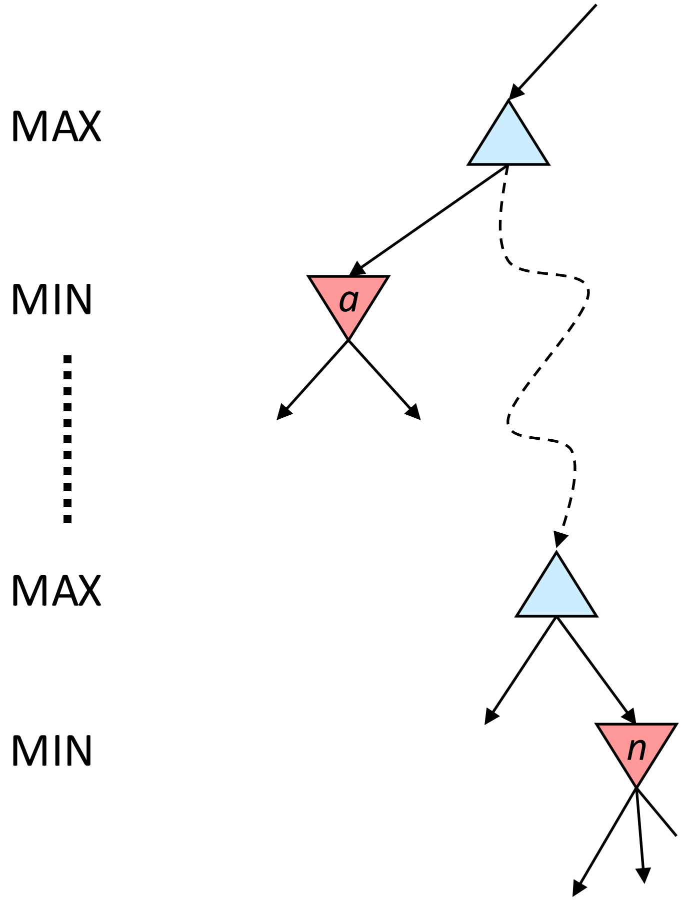


Minimax Pruning



Alpha-Beta Pruning

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children, so *n*'s estimate is decreasing
  - Let *a* be the best value that MAX can get at any choice point along the current path from the root
  - <u>If *n* becomes worse than *a*, then we can stop considering *n*'s other children</u>
  - Reason: if *n* is eventually chosen, then the nodes along the path shall all have the value of *n*, but *n* is worse than *a* and hence the path shall not be chosen at the MAX



- MAX version is symmetric

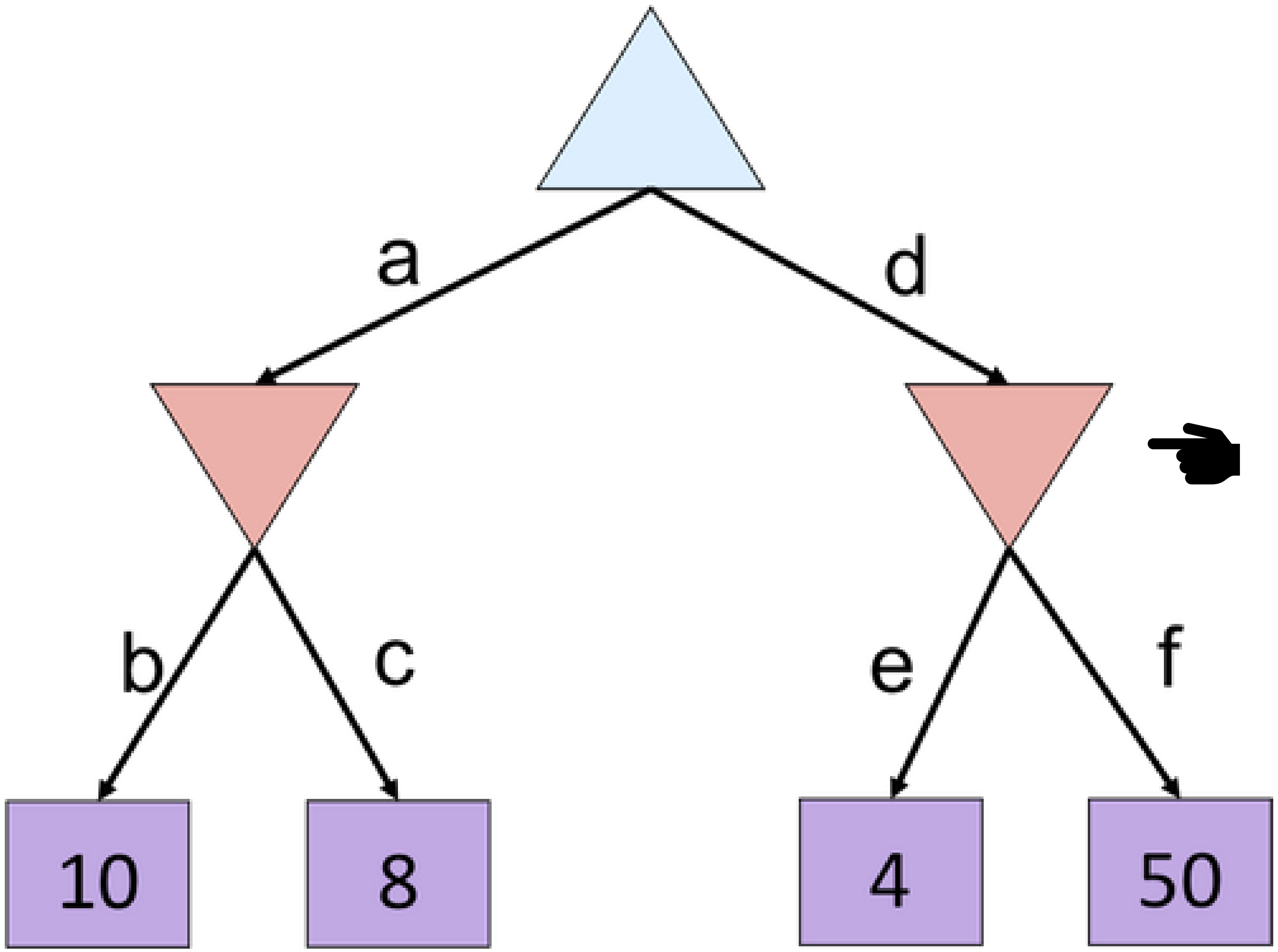
Alpha-Beta Implementation

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

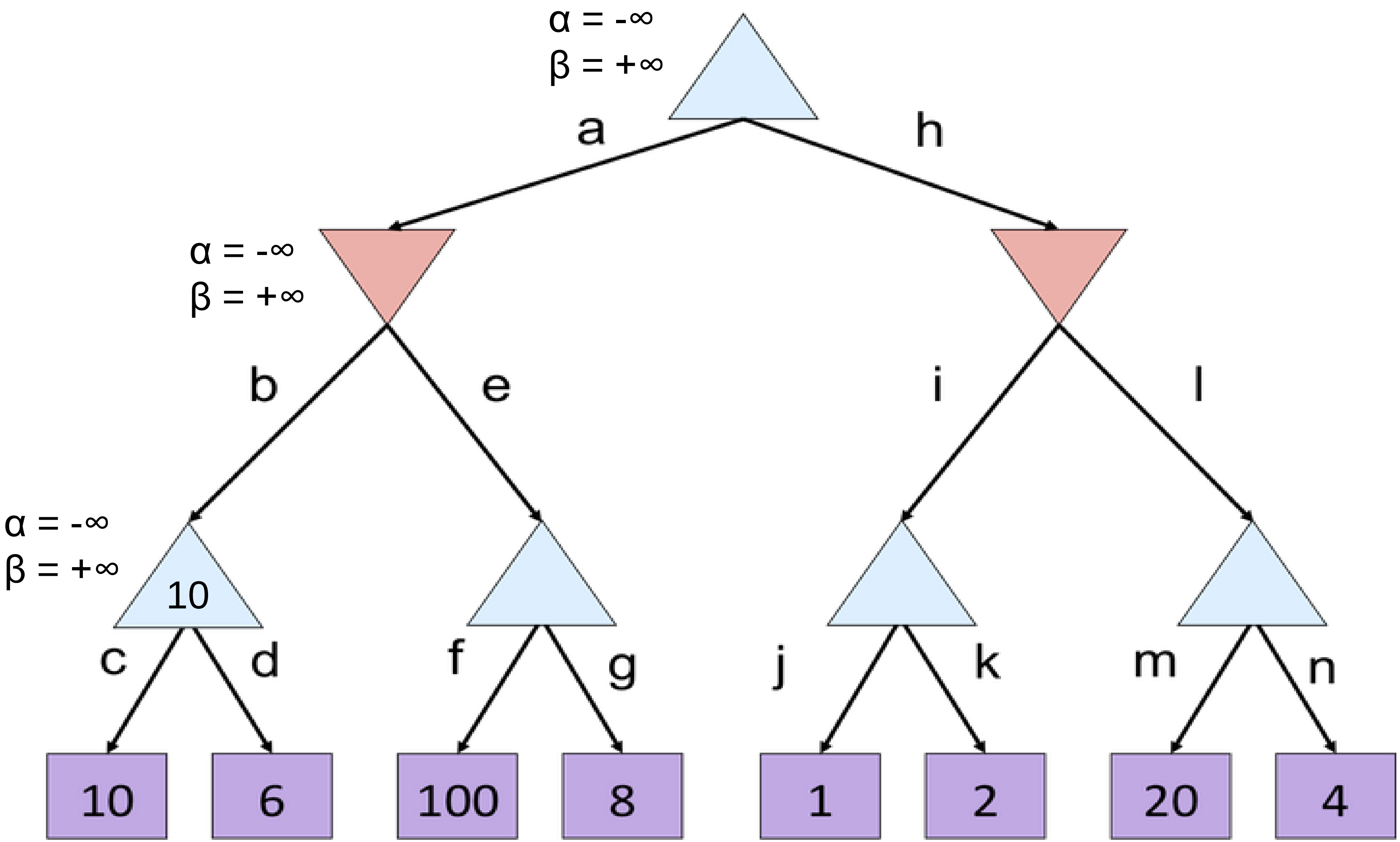
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
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
```

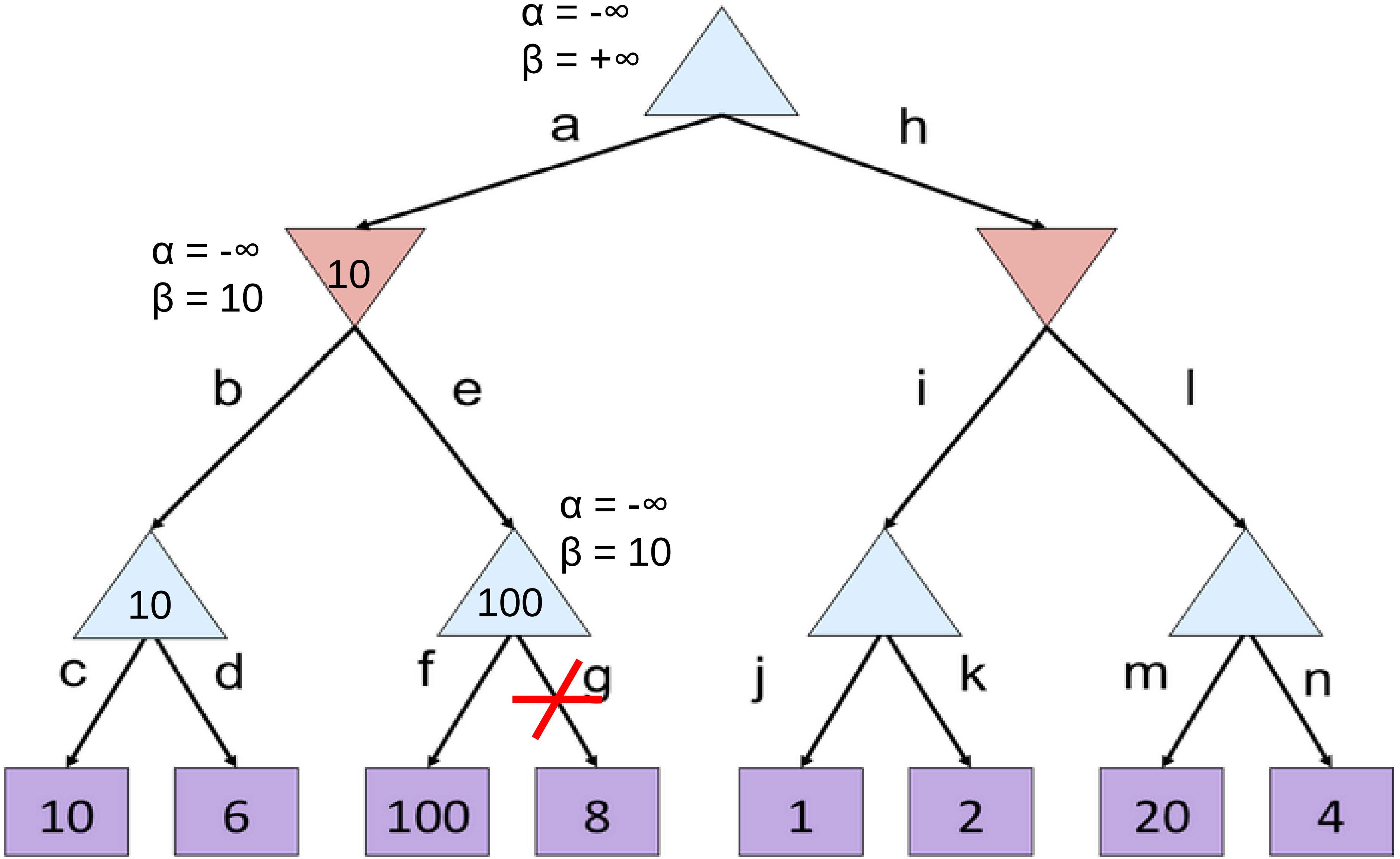
Alpha-Beta Example



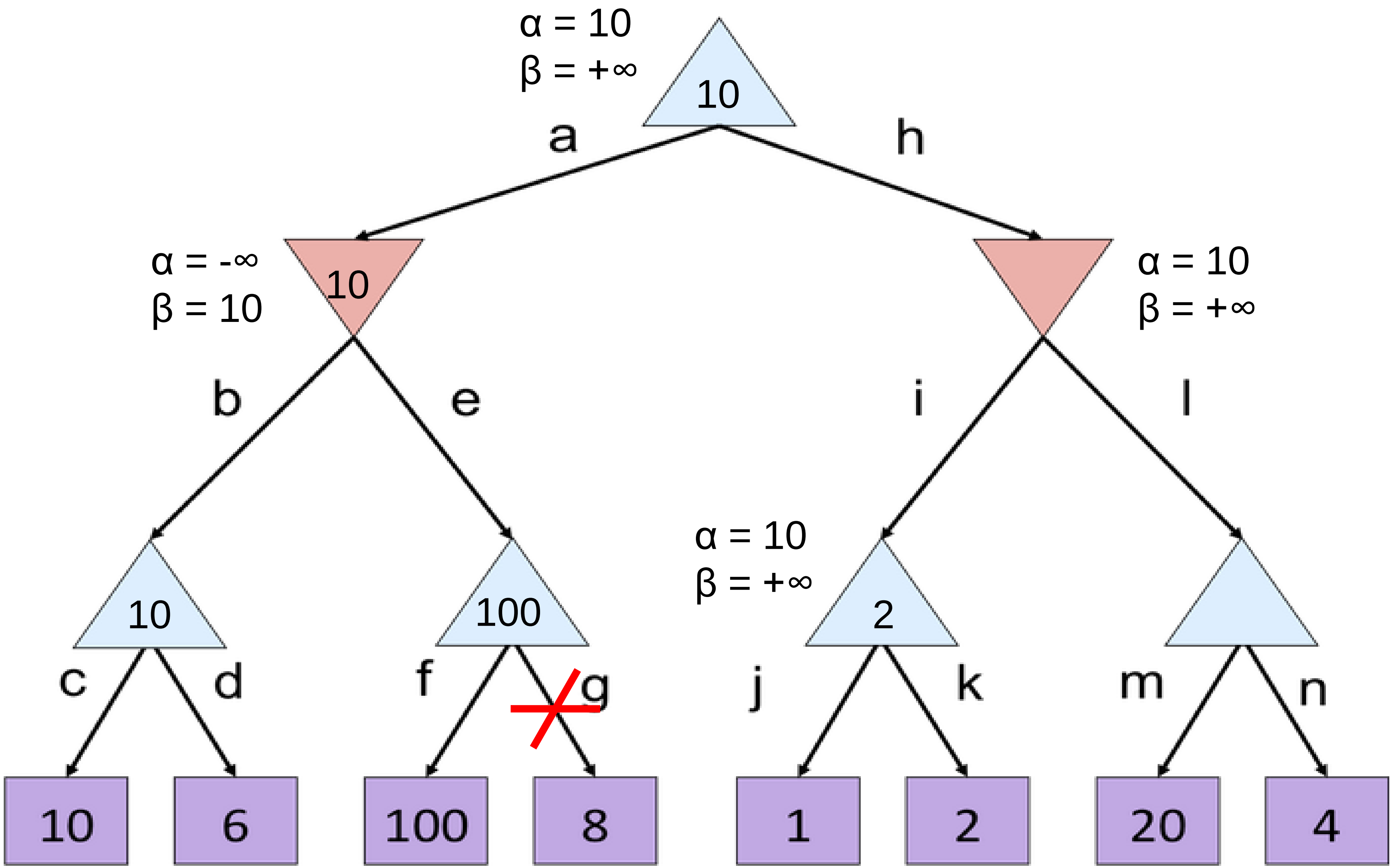
Alpha-Beta Example 2



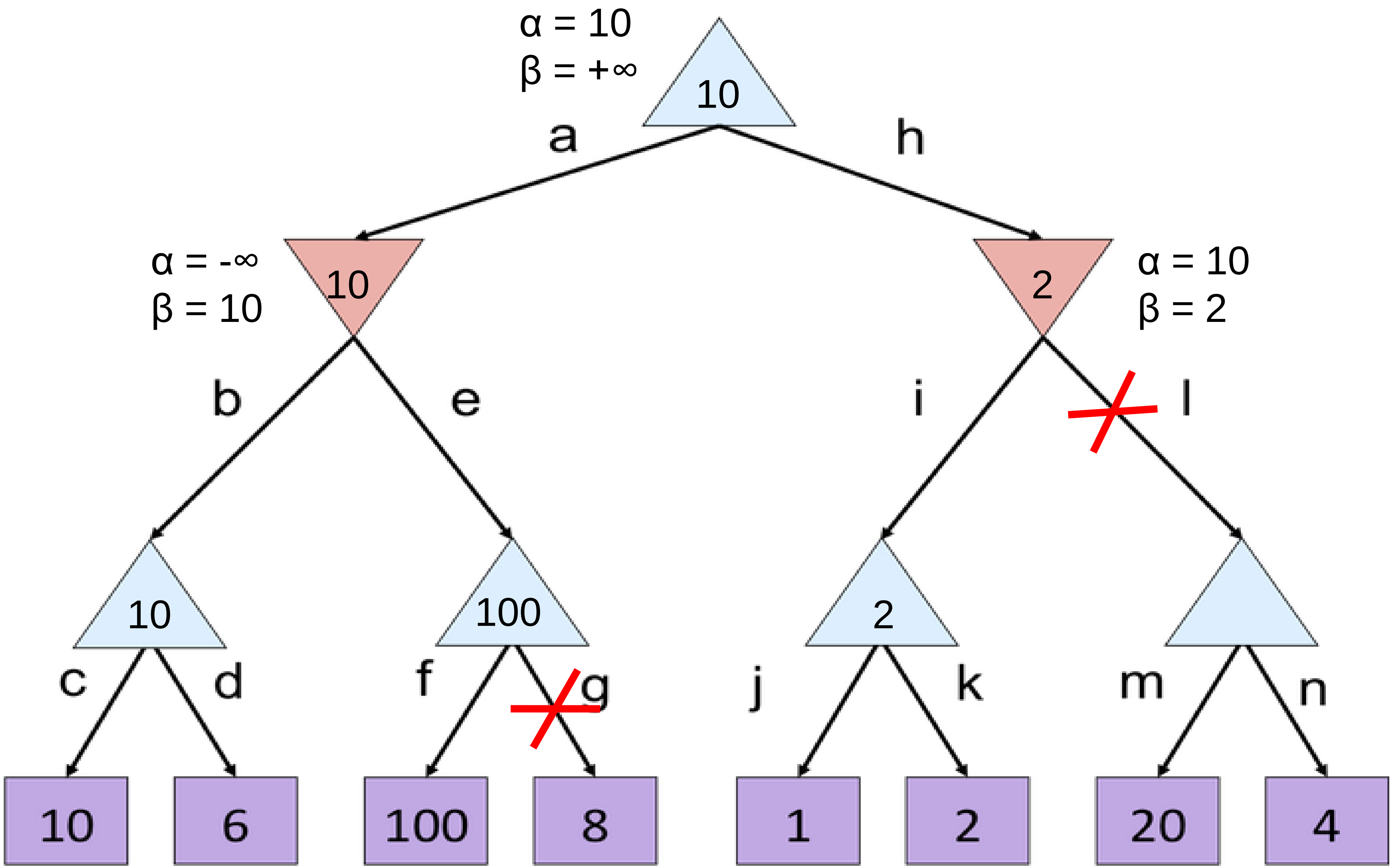
Alpha-Beta Example 2



Alpha-Beta Example 2



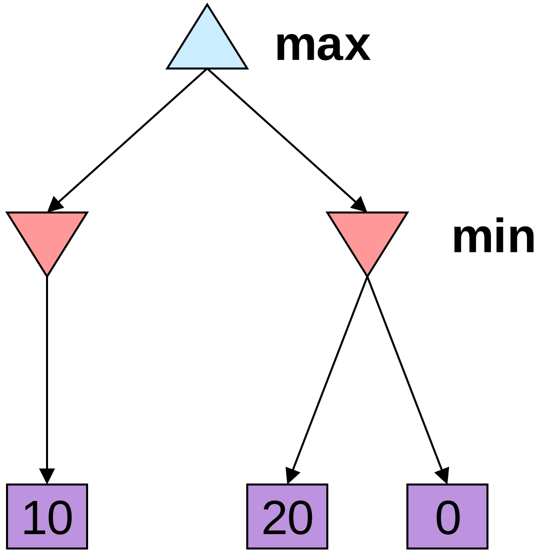
Alpha-Beta Example 2



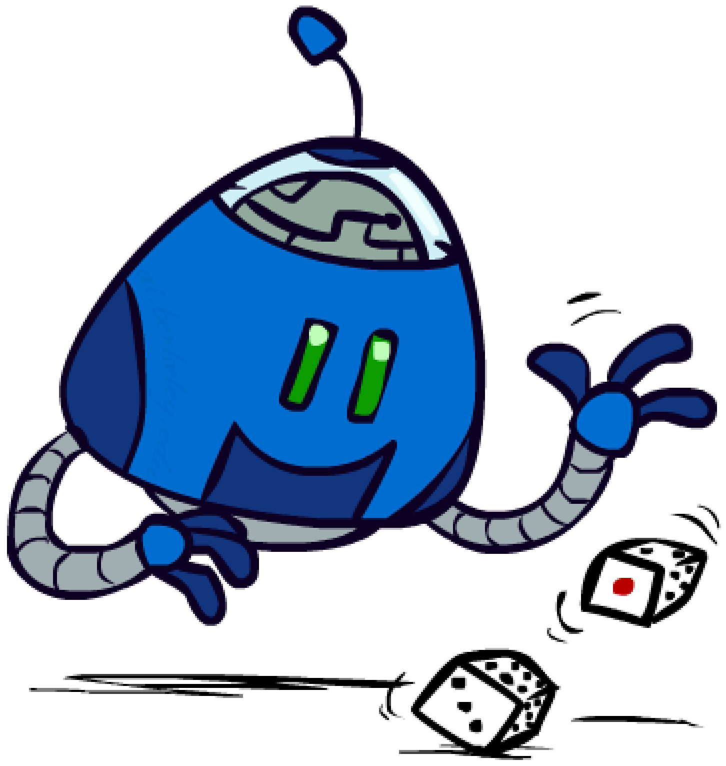
Alpha-Beta Pruning Properties

- Good child ordering improves effectiveness of pruning

- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!

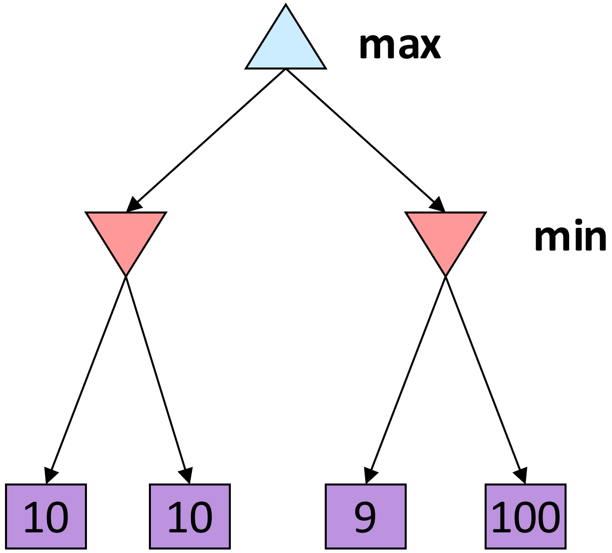


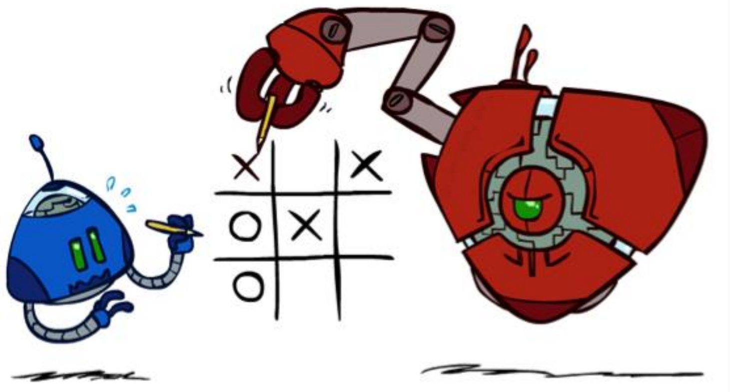
Uncertain Outcomes



Worst-Case vs. Average Case

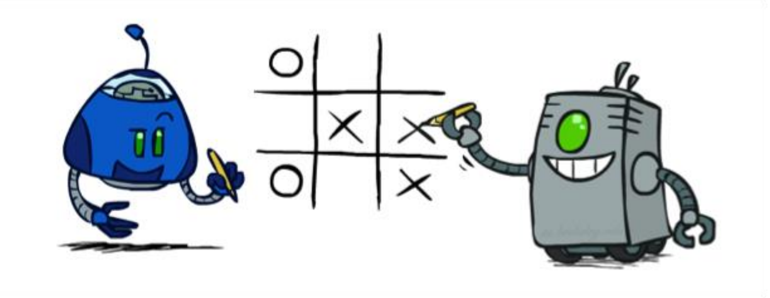
- The hidden assumption behind minimax
  - Your opponent is rational and smart

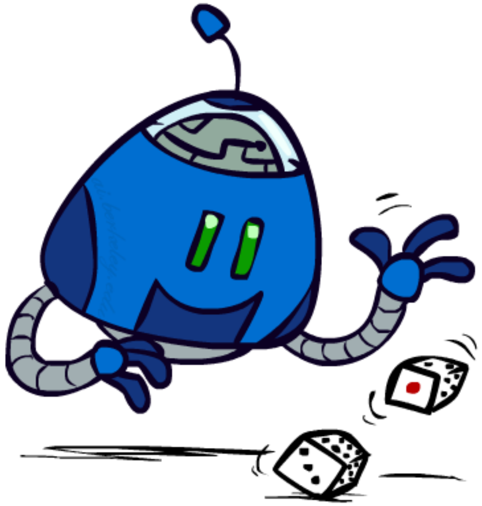




Worst-Case vs. Average Case

- What if…
  - Unpredictable opponents
    - E.g., the ghosts respond randomly
  - Explicit randomness
    - E.g., rolling dice
  - Actions can fail
    - E.g., when moving a robot, wheels might slip





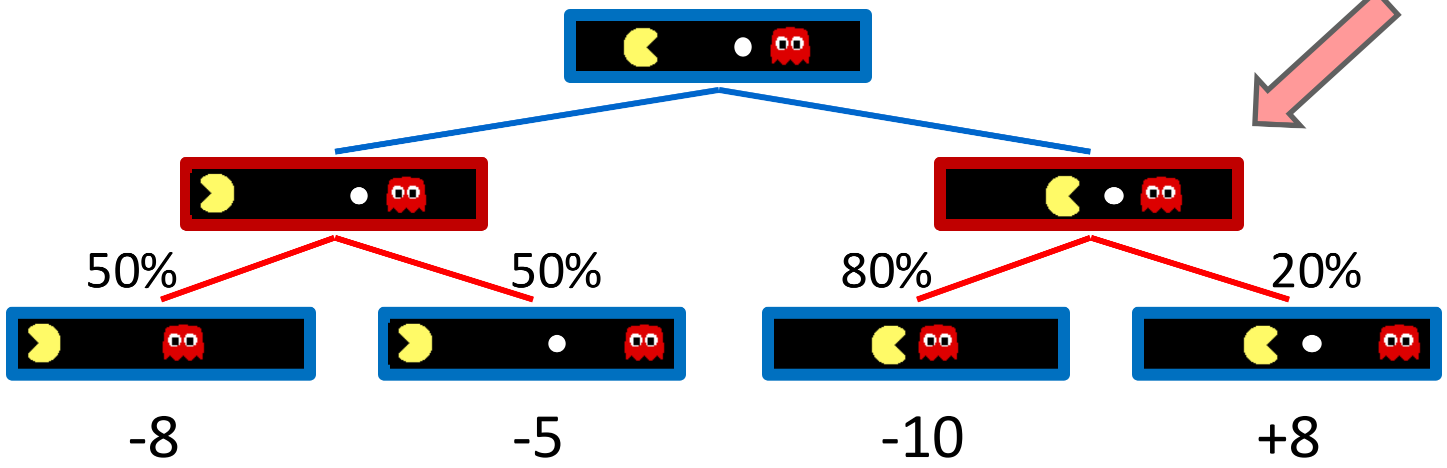
State Values

States Under Agent's Control:

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

States Under Opponent's Control:

$$V(s') = \sum_{s \in \text{successors}(s')} P(s) \times V(s)$$



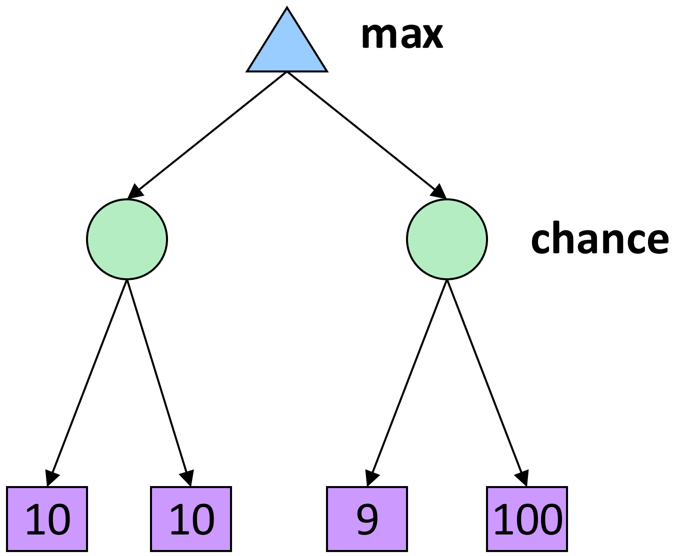
Terminal States:

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

Value of a state: The best achievable expected utility from that state

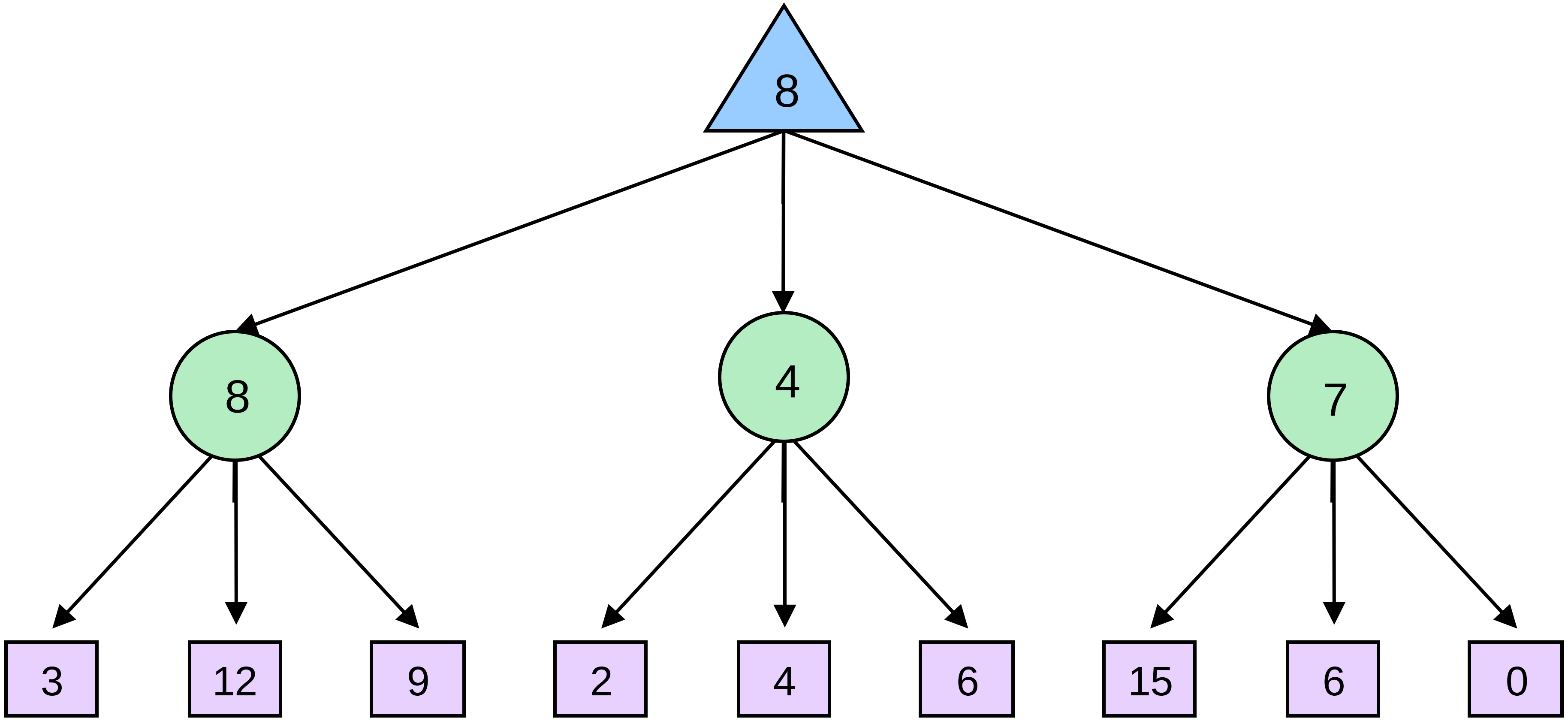
Expectimax Search

- Expectimax search: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities, i.e. taking weighted average (expectation) of children

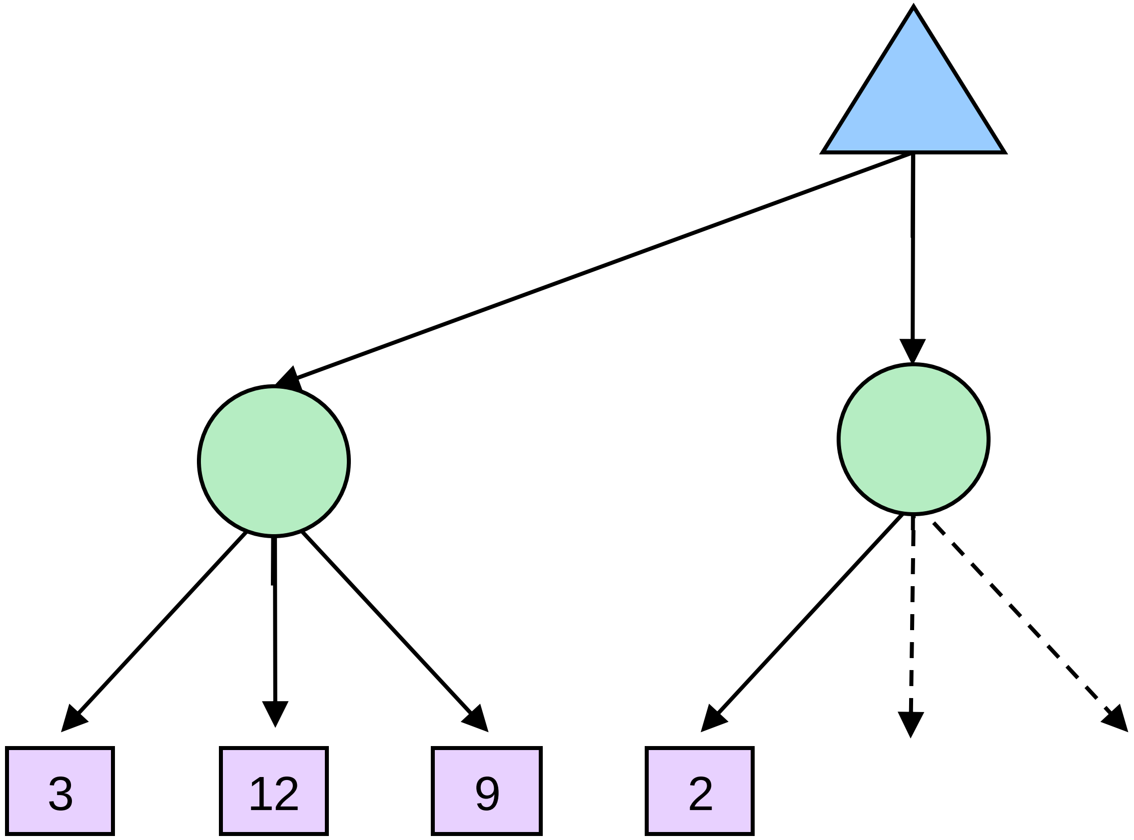


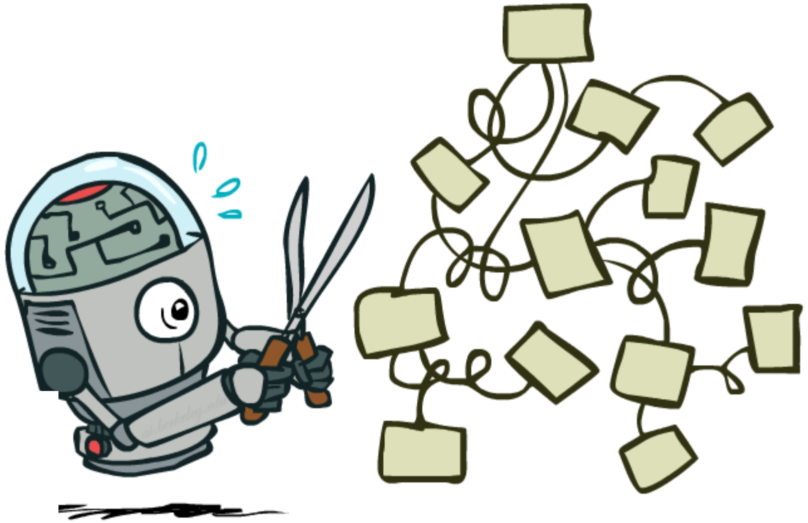
- Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes

Expectimax Example



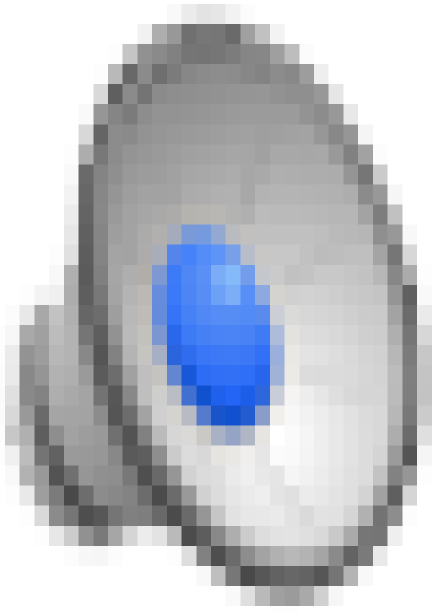
Expectimax Pruning?





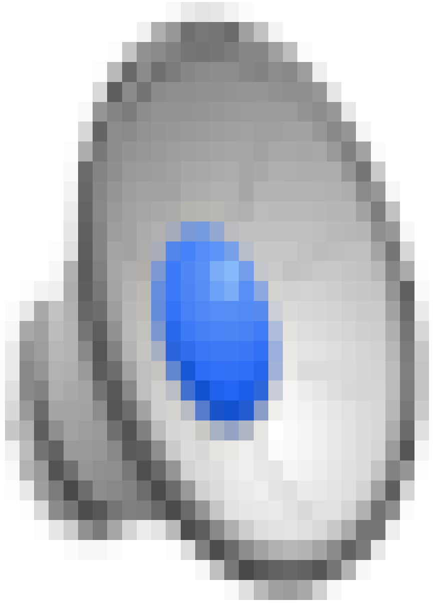
Video of Demo Minimax vs Expectimax (Min)

The game:
+10 for eating each dot
+500 for eating all the dots
-500 for being eaten
-1 for each move



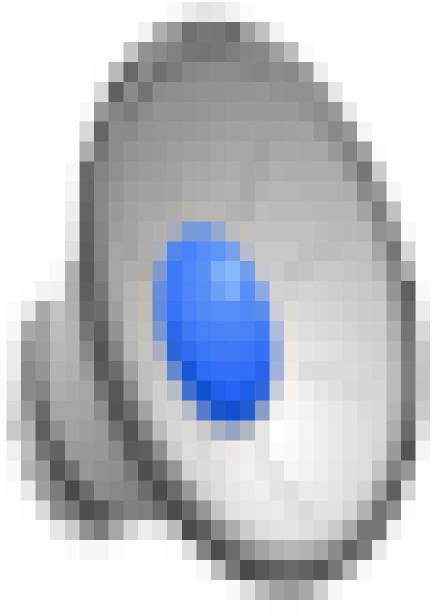
Video of Demo Minimax vs Expectimax (Exp 1)

The game:
+10 for eating each dot
+500 for eating all the dots
-500 for being eaten
-1 for each move



Video of Demo Minimax vs Expectimax (Exp 2)

The game:
+10 for eating each dot
+500 for eating all the dots
-500 for being eaten
-1 for each move



Modeling Assumptions

The Dangers of Optimism and Pessimism

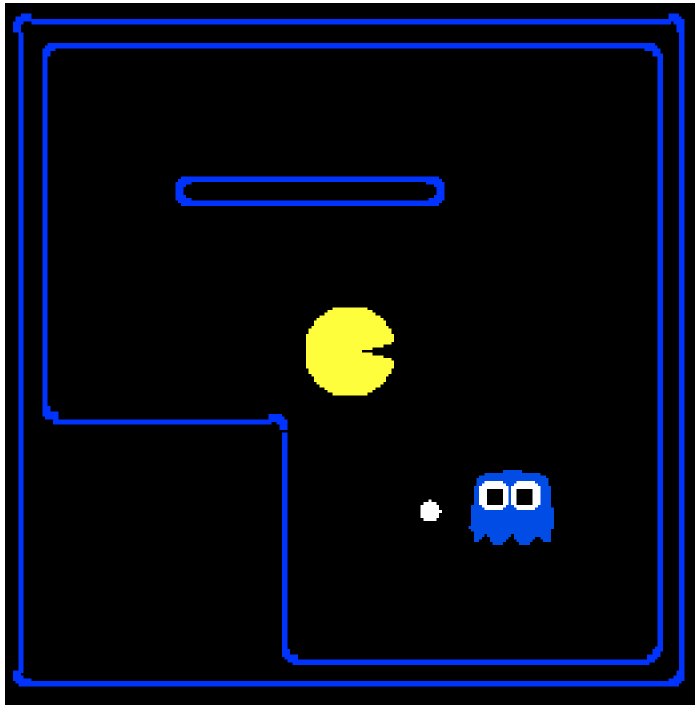
Dangerous Optimism

Assuming chance when the world is adversarial

Dangerous Pessimism

Assuming the worst case when it's not likely

Assumptions vs. Reality

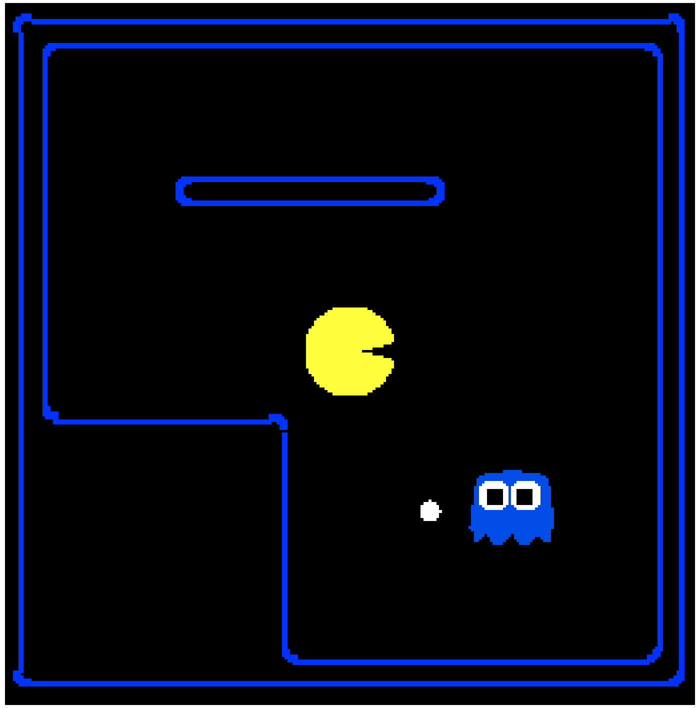


	Adversarial Ghost	Random Ghost
Minimax Pacman		
Expectimax Pacman		

Results from playing 5 games

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

Assumptions vs. Reality

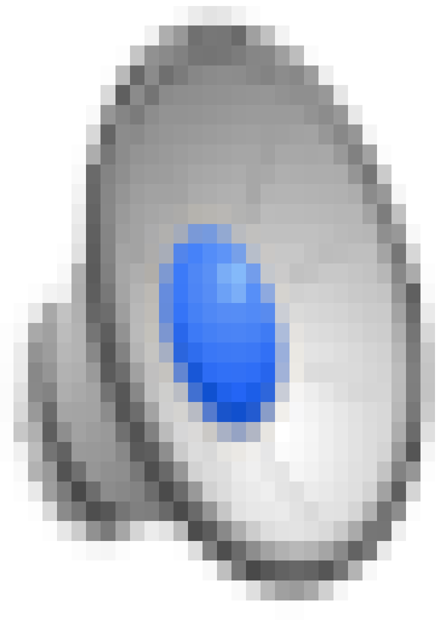


	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 453
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

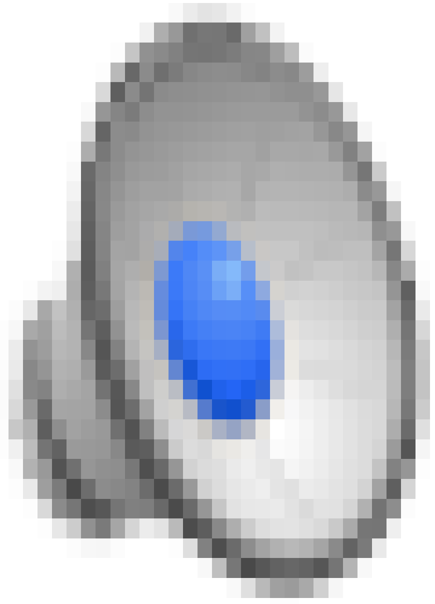
Results from playing 5 games

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

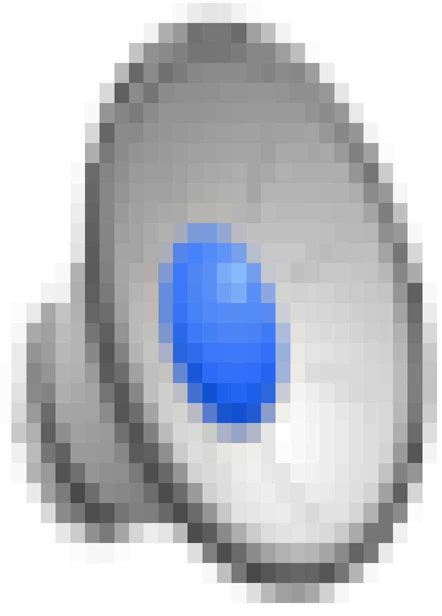
Video of Demo World Assumptions
Adversarial Ghost – Minimax Pacman



Video of Demo World Assumptions
Adversarial Ghost – Expectimax Pacman

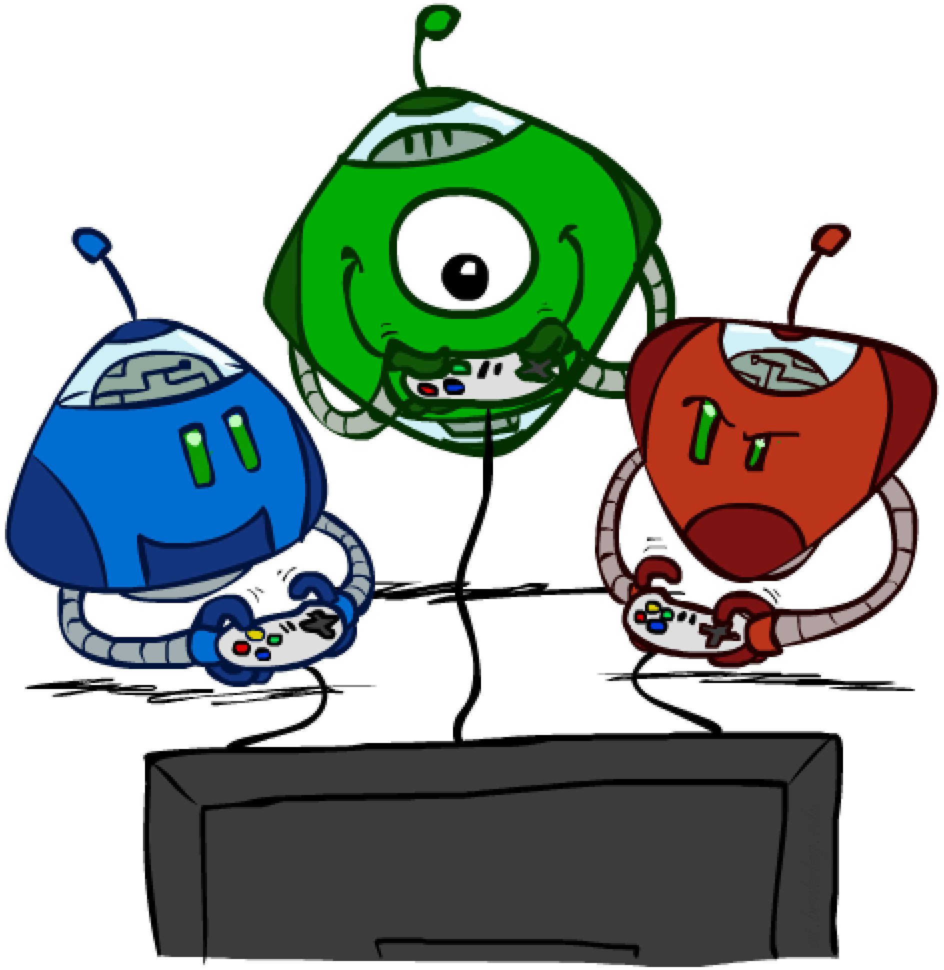


Video of Demo World Assumptions
Random Ghost – Minimax Pacman



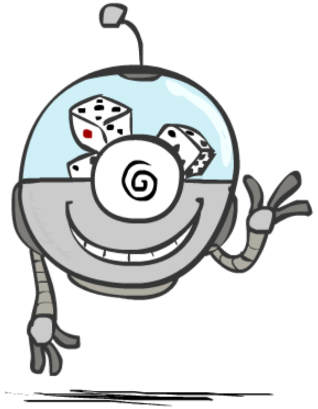
Video of Demo World Assumptions
Random Ghost – Expectimax Pacman

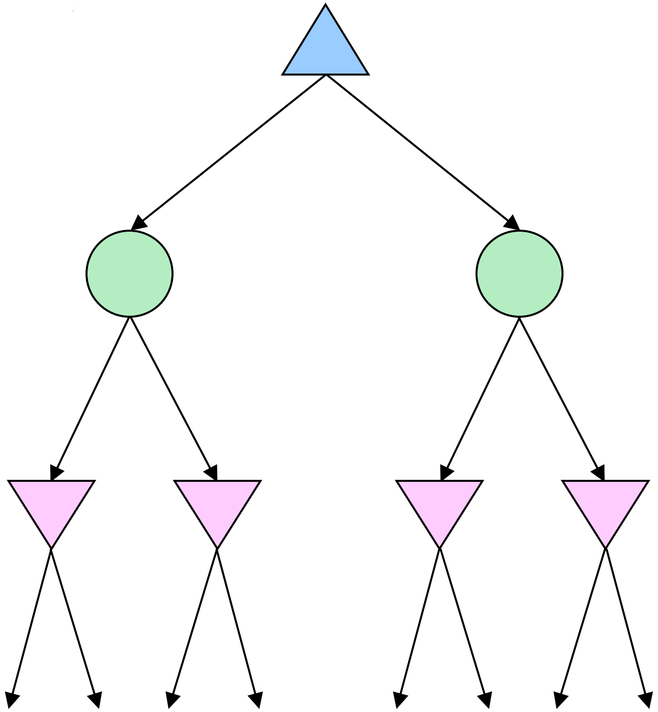
Other Game Types



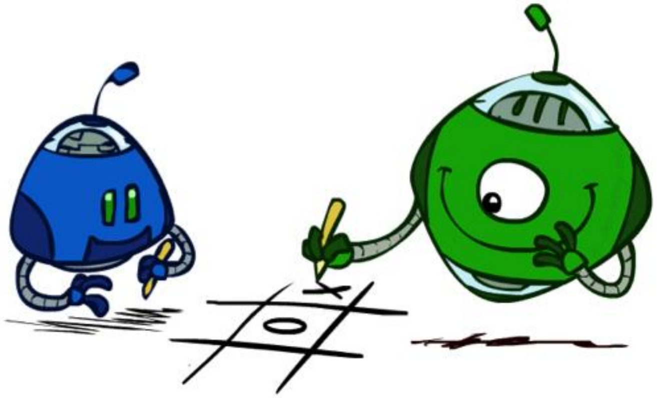
Mixed Layer Types

- Backgammon
- Expectiminimax
  - Environment is an extra “random agent” player that moves after each min/max agent
  - Each node computes the appropriate combination of its children

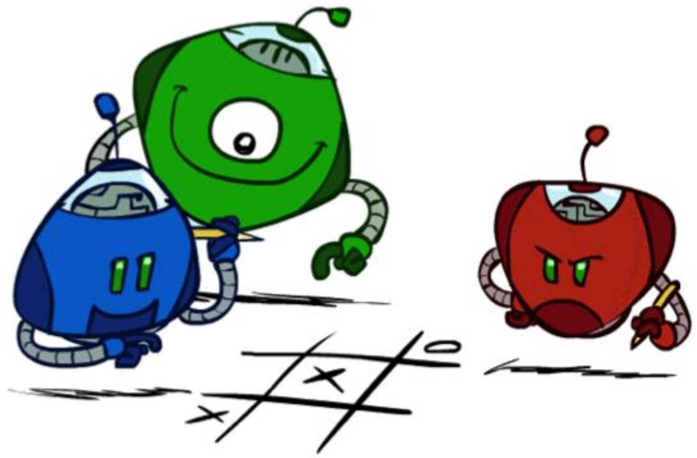


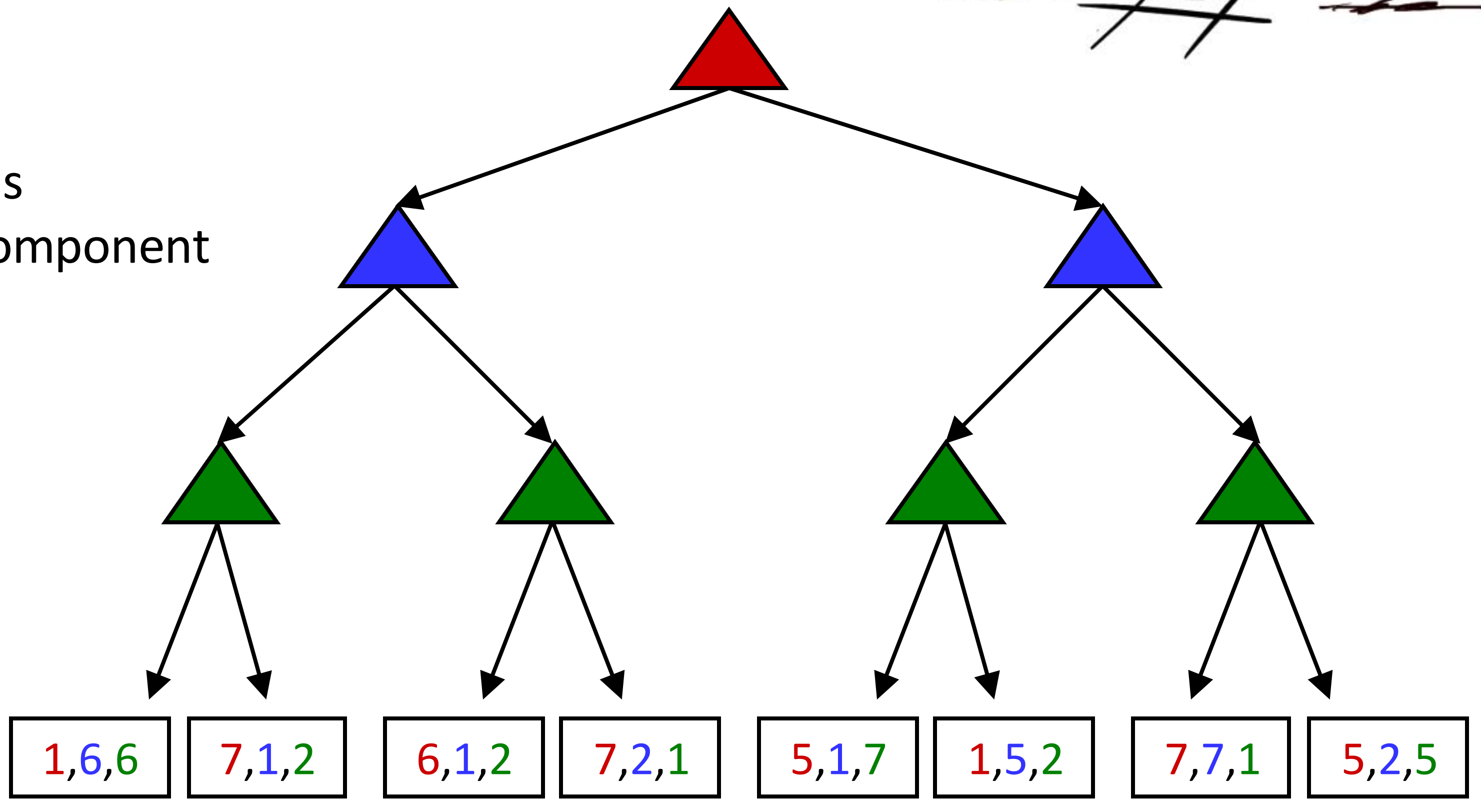


Multi-Agent Utilities

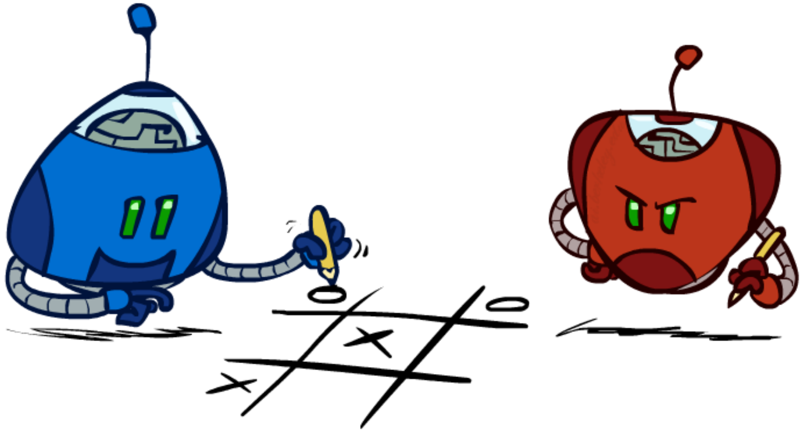


- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…





Summary



- Adversarial Games
- Adversarial Search
  - Minimax
- Resource Limits
  - Depth-limited search, limiting branching factor
- Game Tree Pruning (alpha-beta pruning)
- Uncertain Outcomes
  - Expectimax
- Other Game Types