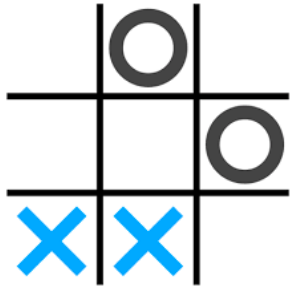
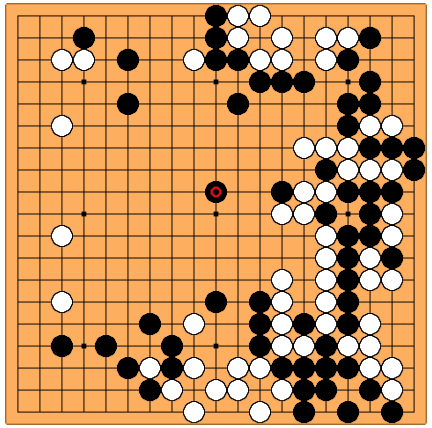
CS 6511: Artificial Intelligence

Adversarial Search





Amrinder Arora

The George Washington University

[An original version of these slides was created by Dan Klein and Pieter Abbeel for Intro to AI at UC Berkeley. http://ai.berkeley.edu]

Game Playing State-of-the-Art

- **Connect 4:** Published 1974, Solved 1988
- https://archive.org/details/isbn_9781402756214

- **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!**
- **https://www.science.org/doi/10.1126/science.1144079**

- **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. Current version of Stockfish (as of 2022) runs at nearly 3500 Elo rating

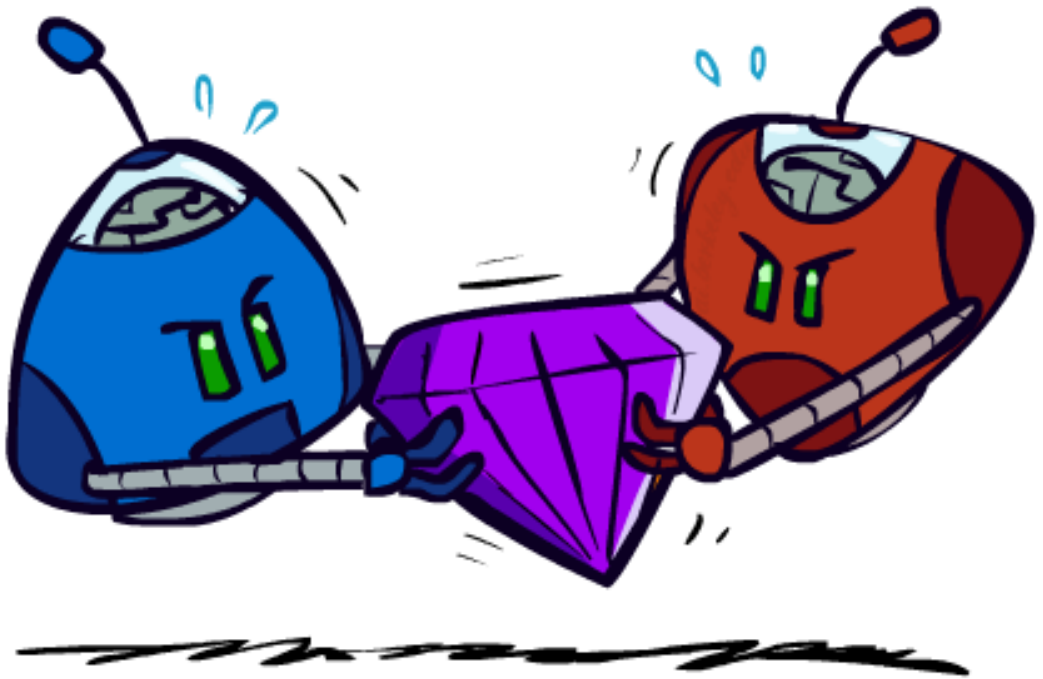
- **Go:** AlphaGo, 2016 "These specific ideas that are driving AlphaGo are going to drive our future. The technologies at the heart of AlphaGo, what are called Deep Neural Networks, which essentially mimic the web of neurons in the brain, it's a very old idea, but recently, due to increases in computing power, these neural networks have become extremely powerful, almost over night."

- What does it mean to "solve the game"?
  - The result is known when both play perfectly.

Learning Objectives

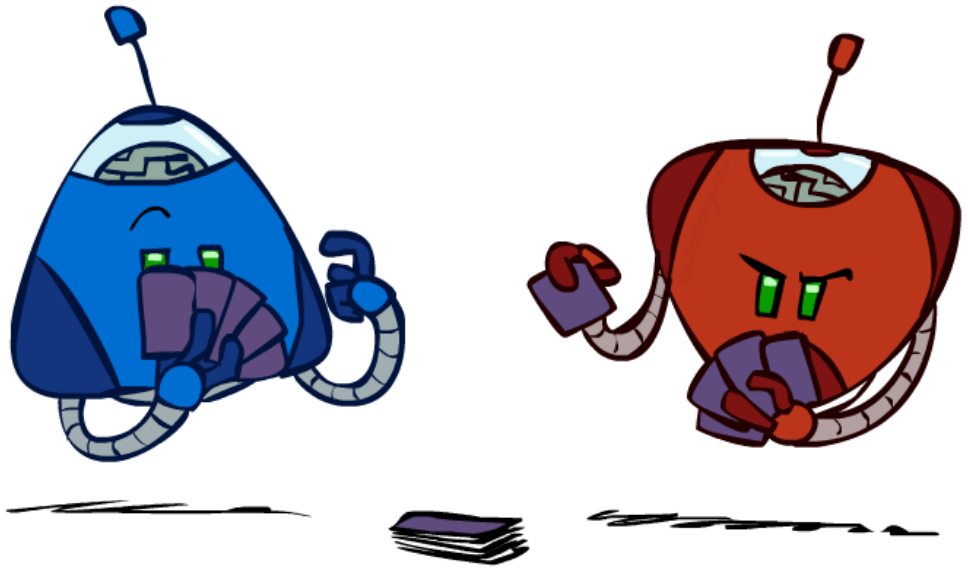
- Types of adversarial games
- Basic nature of adversarial search (min/max)
- New concept: value of a state
- Resource limits and value of terminal states
- Pruning – Alpha Beta
- Expectimax
- Imperfect Play & “Non Math” Factors

Adversarial Games



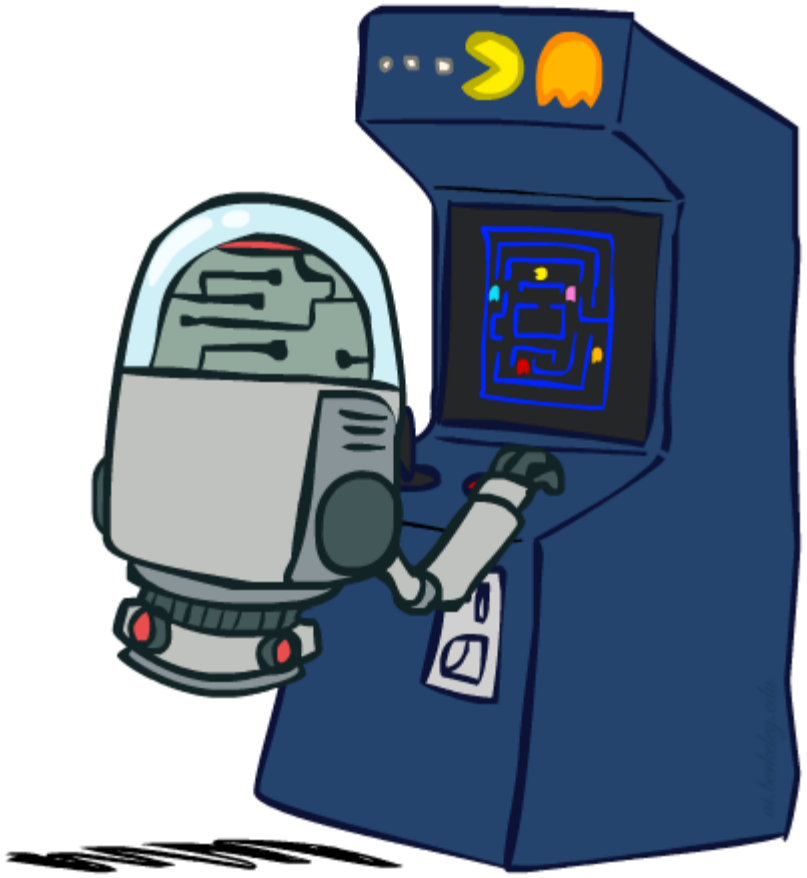
Many Different Types of Games

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

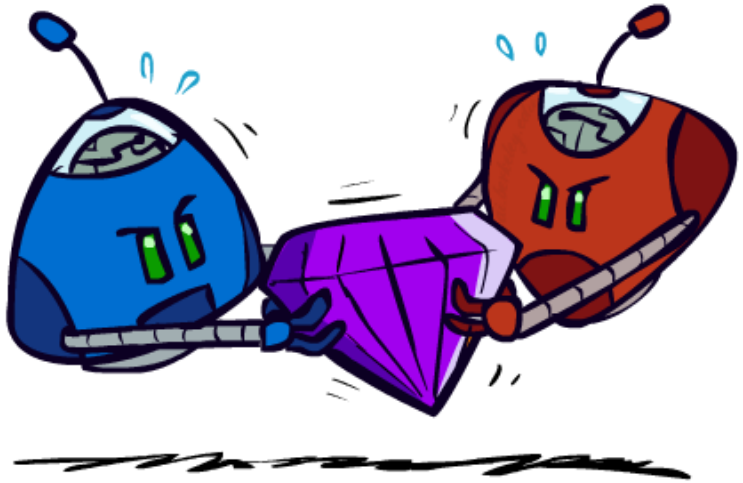


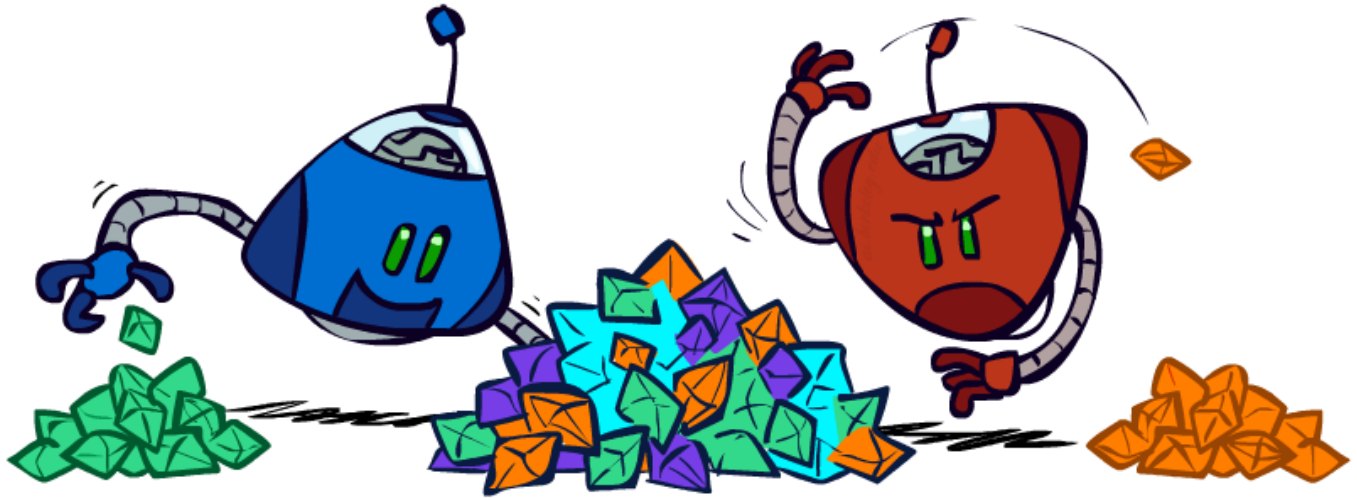
Deterministic Games

- 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





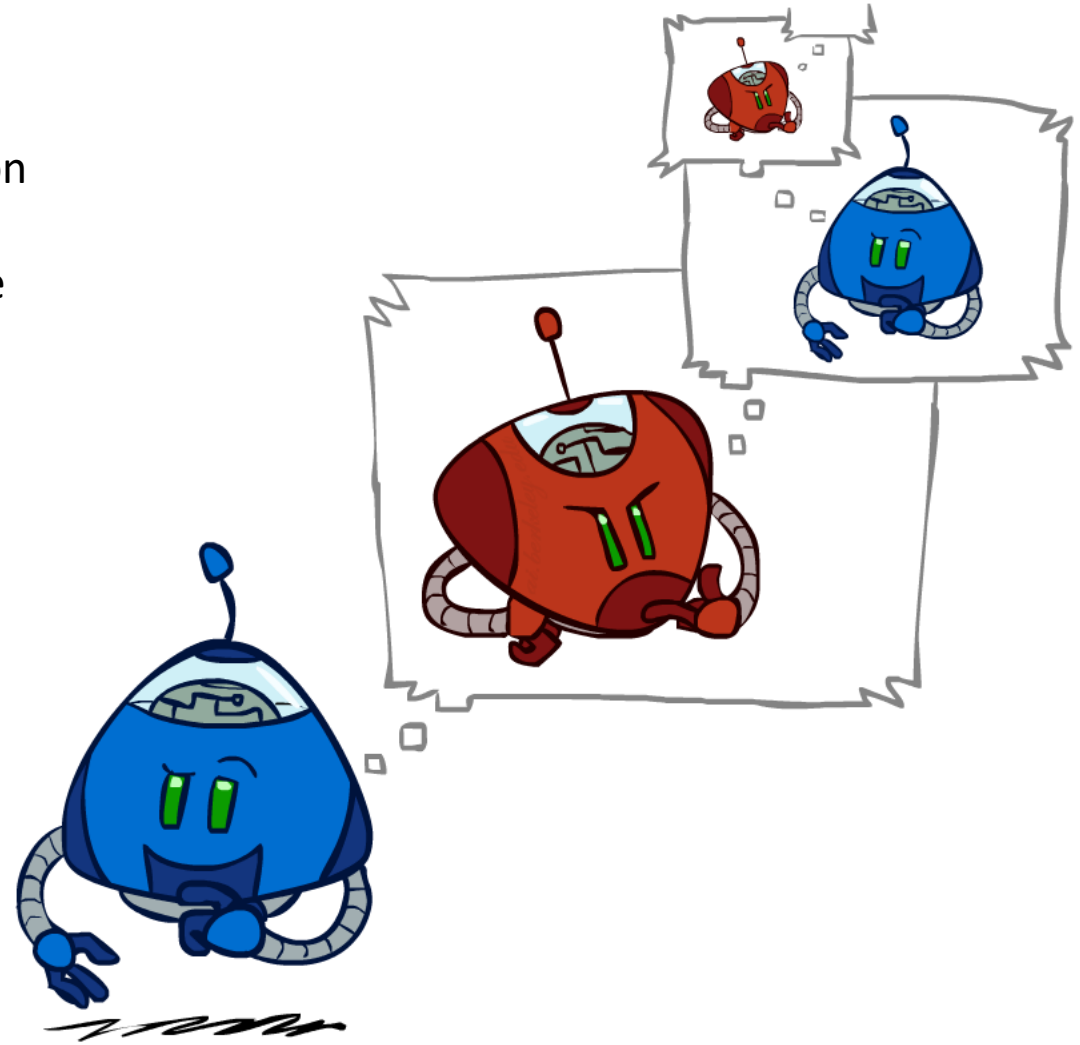
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

Learning Objectives

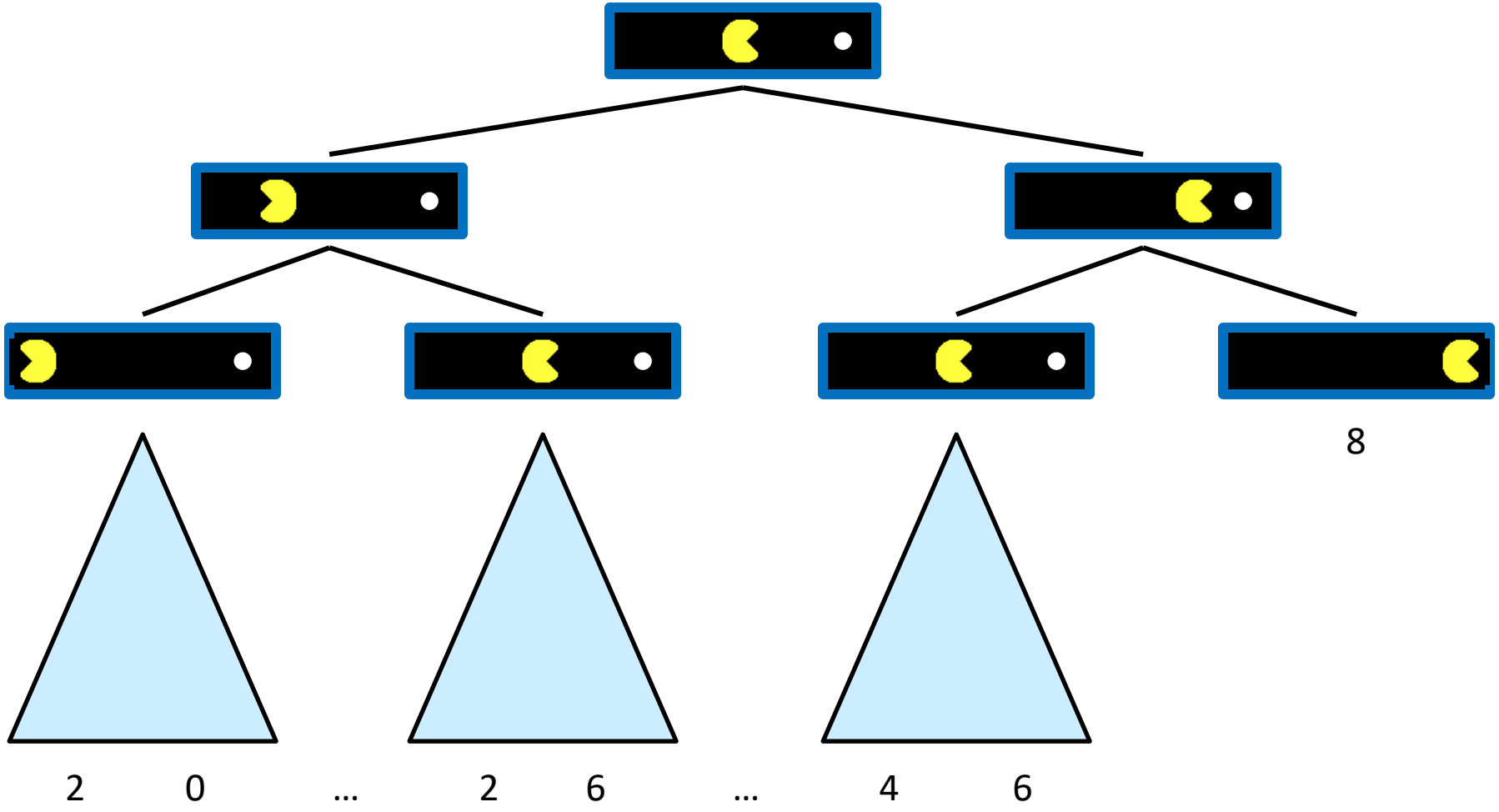
- ✓ Types of adversarial games
- Basic nature of adversarial search (min/max)
- New concept: value of a state
- Resource limits and value of non-terminal states
- Pruning – Alpha Beta
- Expectimax
- Imperfect Play → “Non Math” Factors

Basic Nature of Adversarial Search

Basic nature is that of recursion where we have to assume the opponent will also try to make the “best” move.



Single-Agent Trees



Learning Objectives

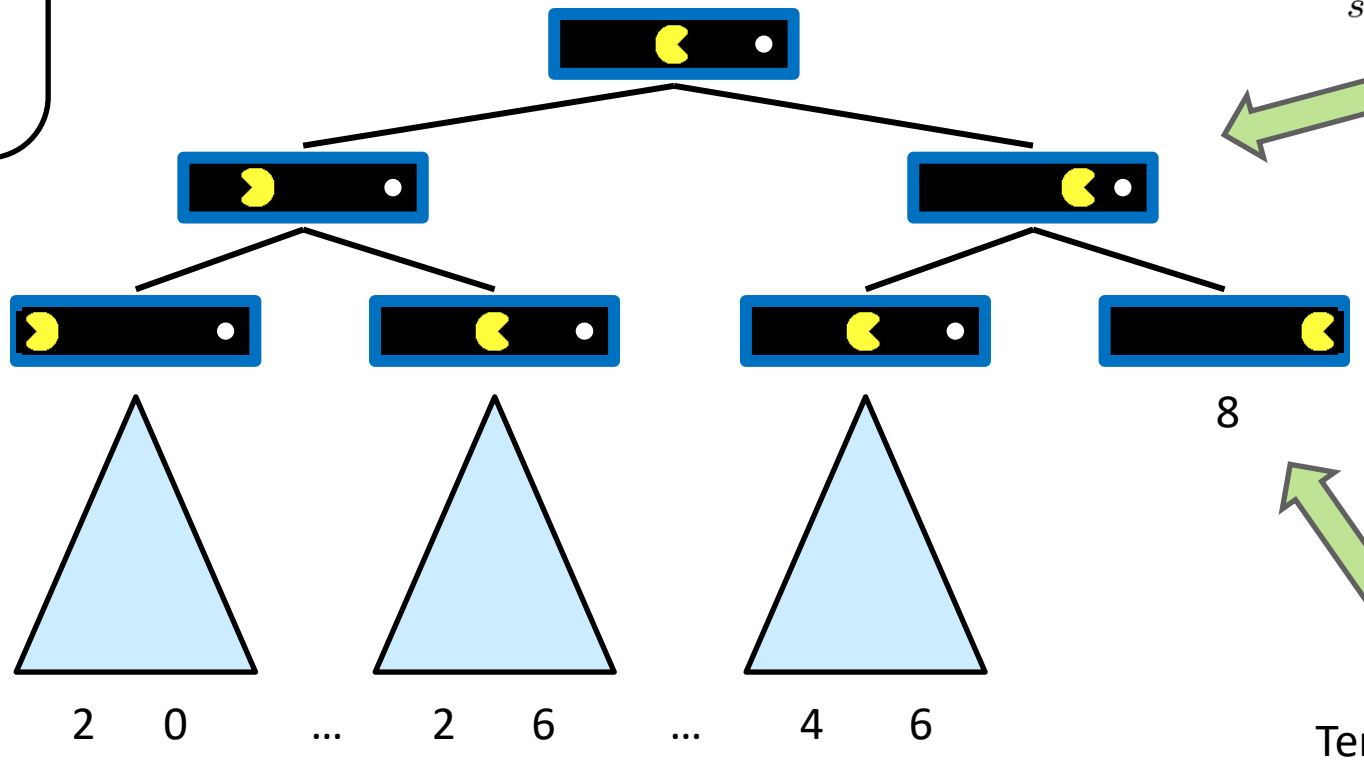
- ✓ Types of adversarial games
- ✓ Basic nature of adversarial search (min/max)
- New concept: value of a state
- Resource limits and value of non-terminal states
- Pruning – Alpha Beta
- Expectimax
- Imperfect Play → “Non Math” Factors

Value of a State

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

Non-Terminal States:

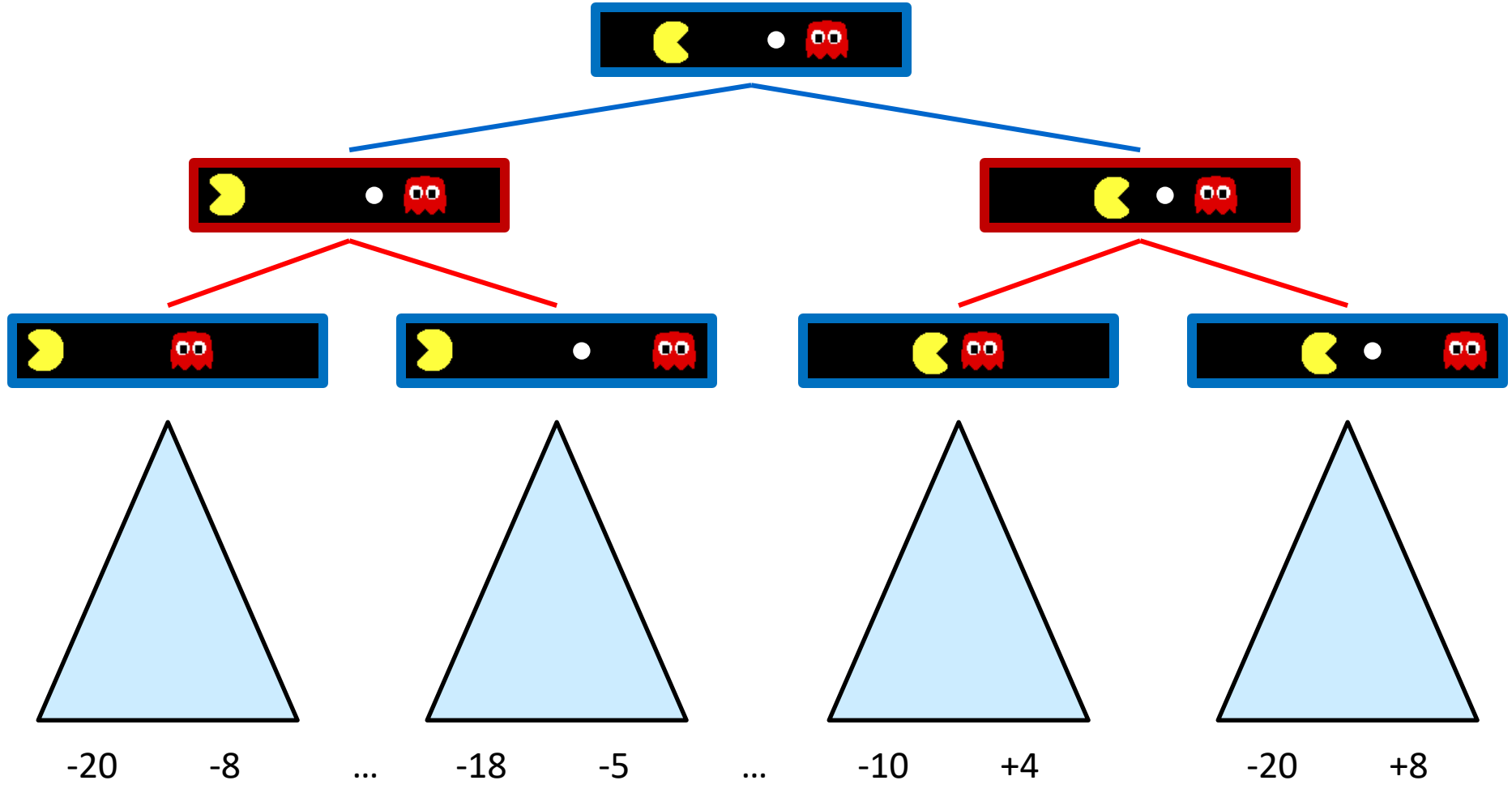
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



Terminal States:

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

Adversarial Game Trees



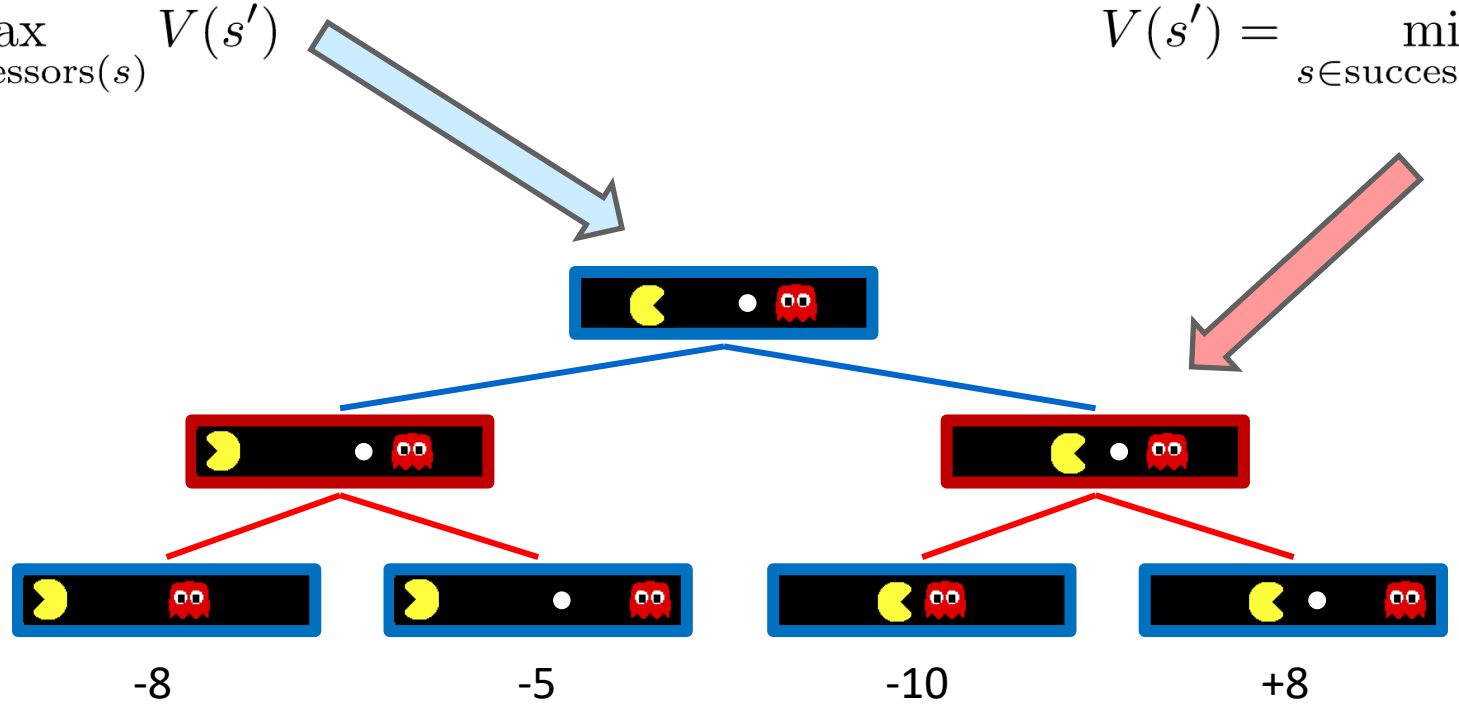
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

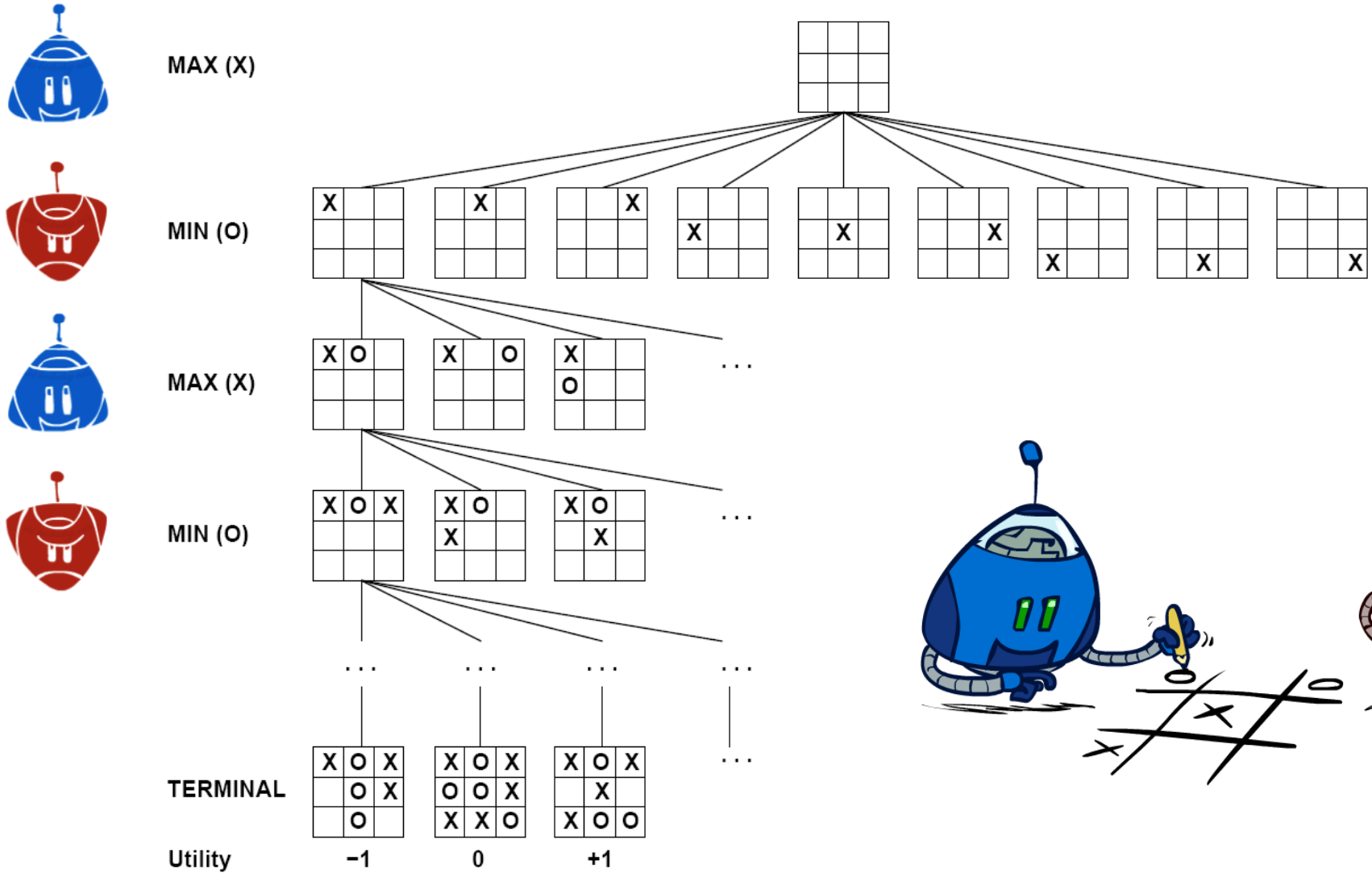
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

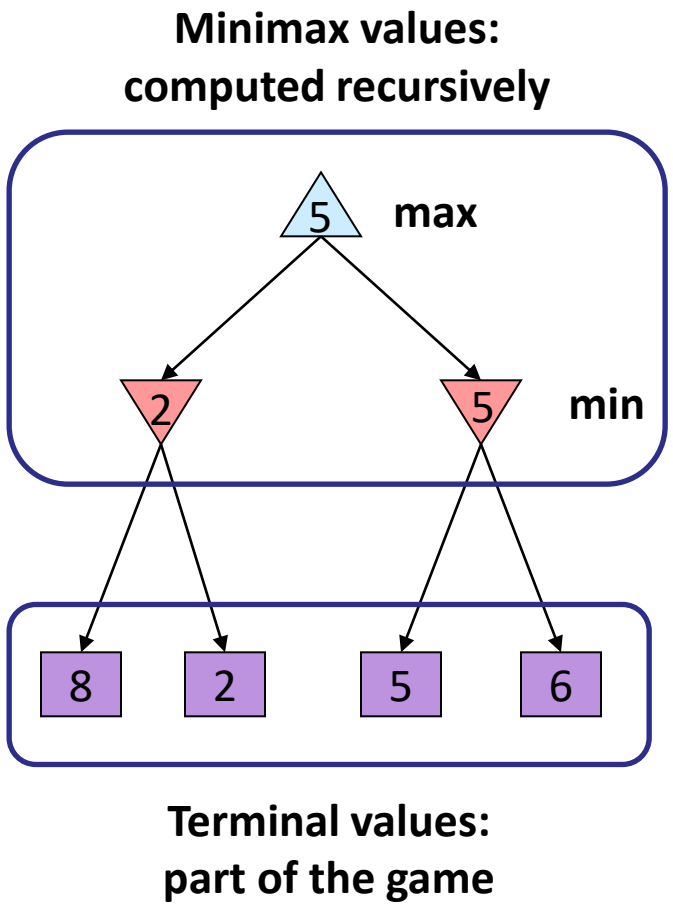
$$V(s) = \text{known}$$

Tic-Tac-Toe Game Tree



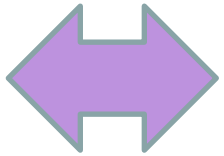
Adversarial Search (Minimax)

- 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
```



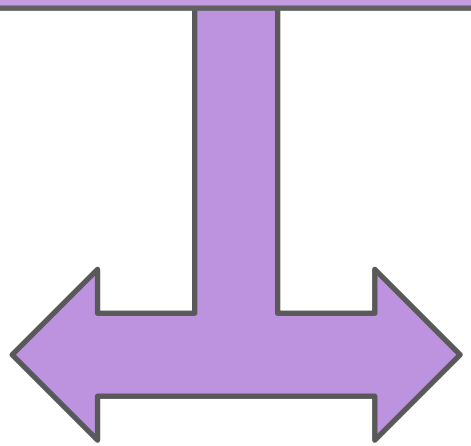
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

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

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

Minimax Implementation (Dispatch)

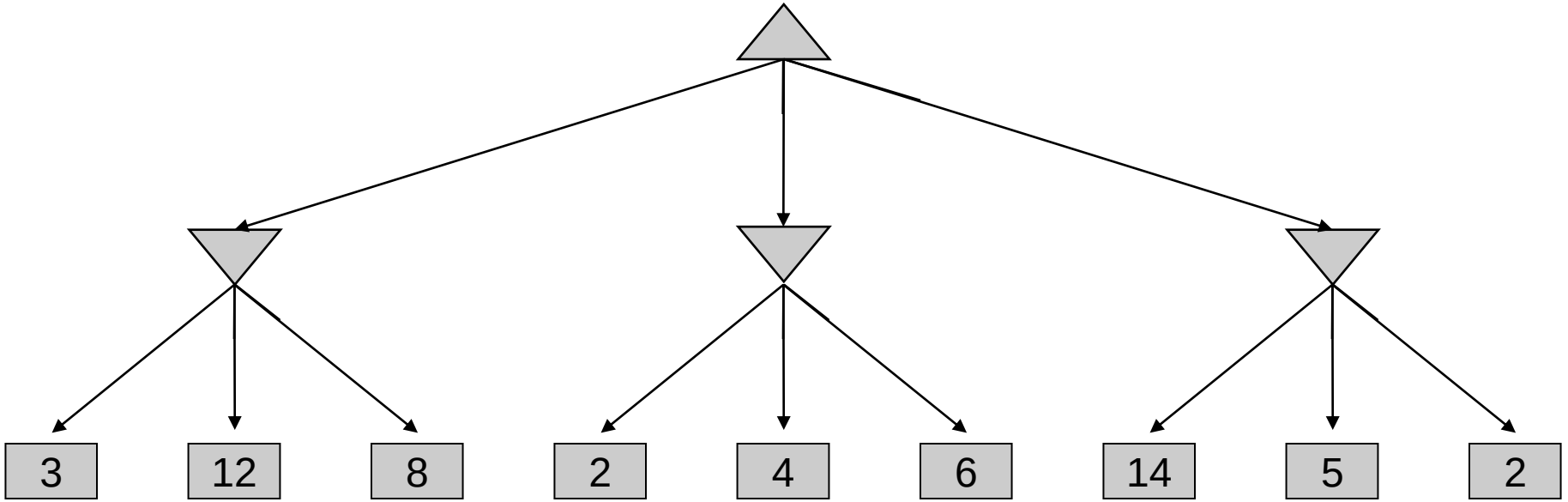
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```



```
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
```

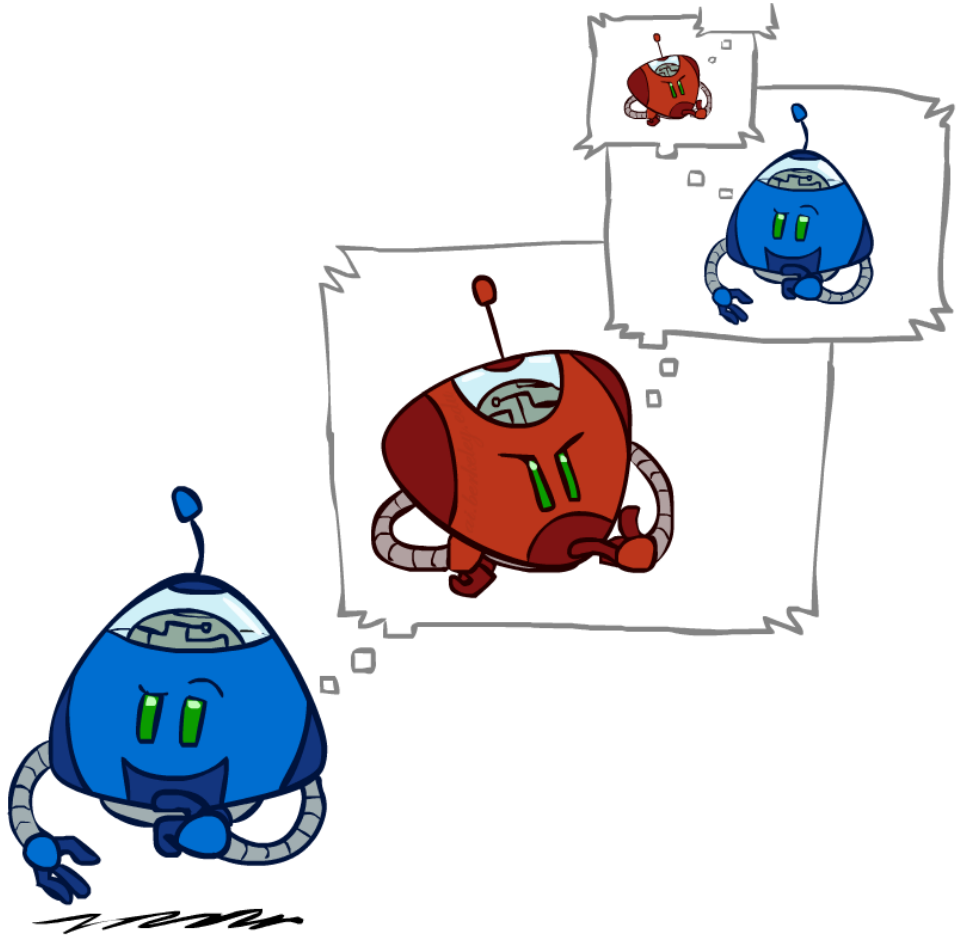
Minimax Example



Minimax 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?



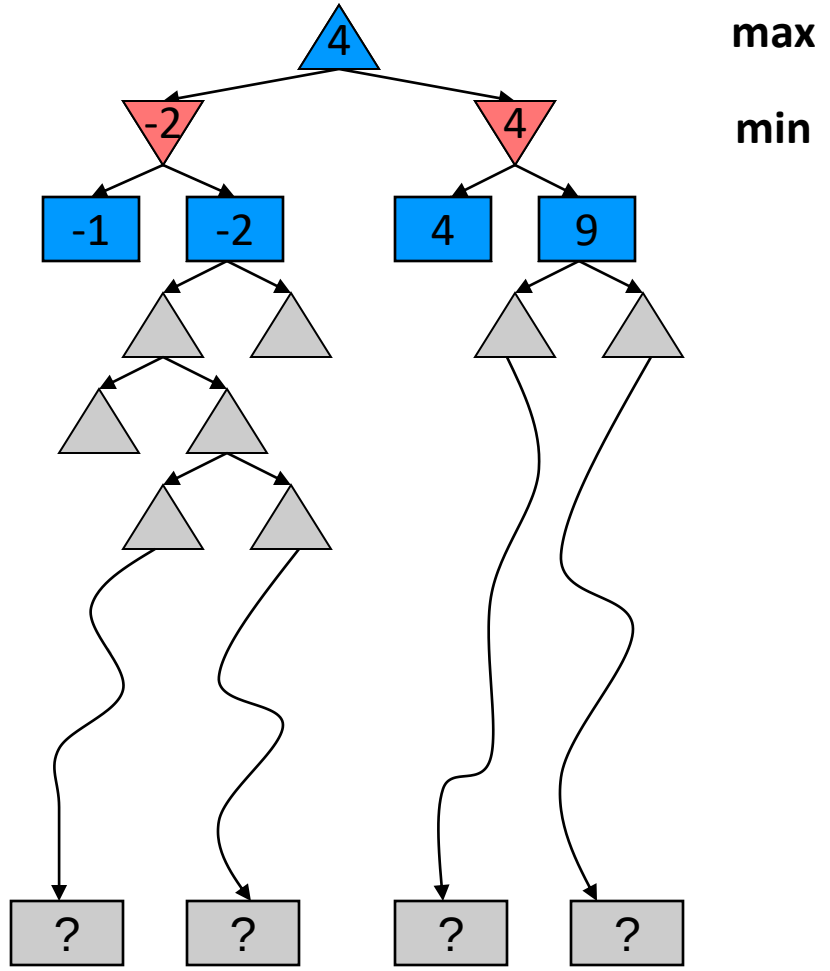
Learning Objectives

- ✓ Types of adversarial games
- ✓ Basic nature of adversarial search (min/max)
- ✓ New concept: value of a state
- Resource limits and value of non-terminal states
- Pruning – Alpha Beta
- Expectimax
- Imperfect Play → “Non Math” Factors

Resource Limits: How far (deep) can we see?

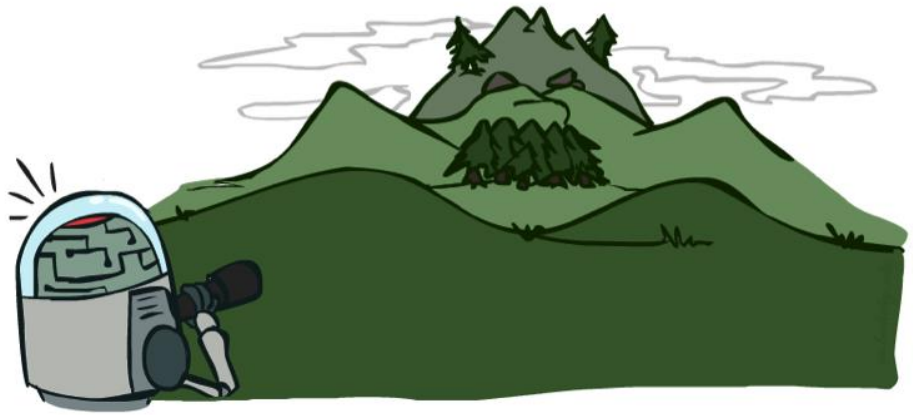
Resource Limits

- 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 plies makes a BIG difference
- Use iterative deepening for an anytime algorithm



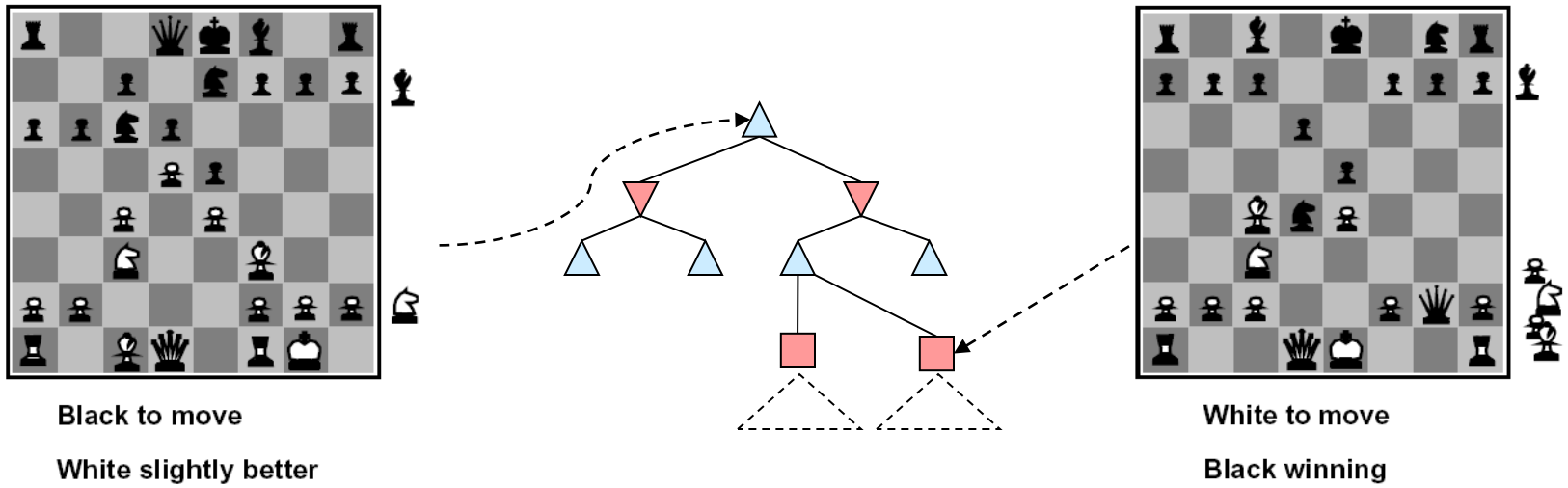
Depth vs. Evaluation Function

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation



Evaluation Functions

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

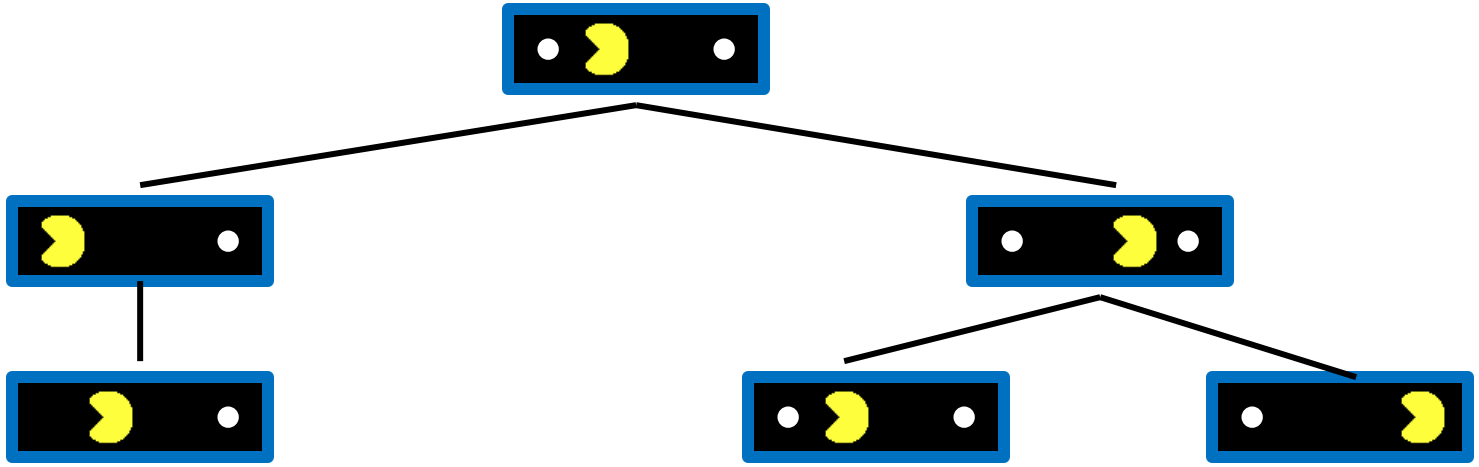


- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

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

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

Implementation Concern: Inability to make decision

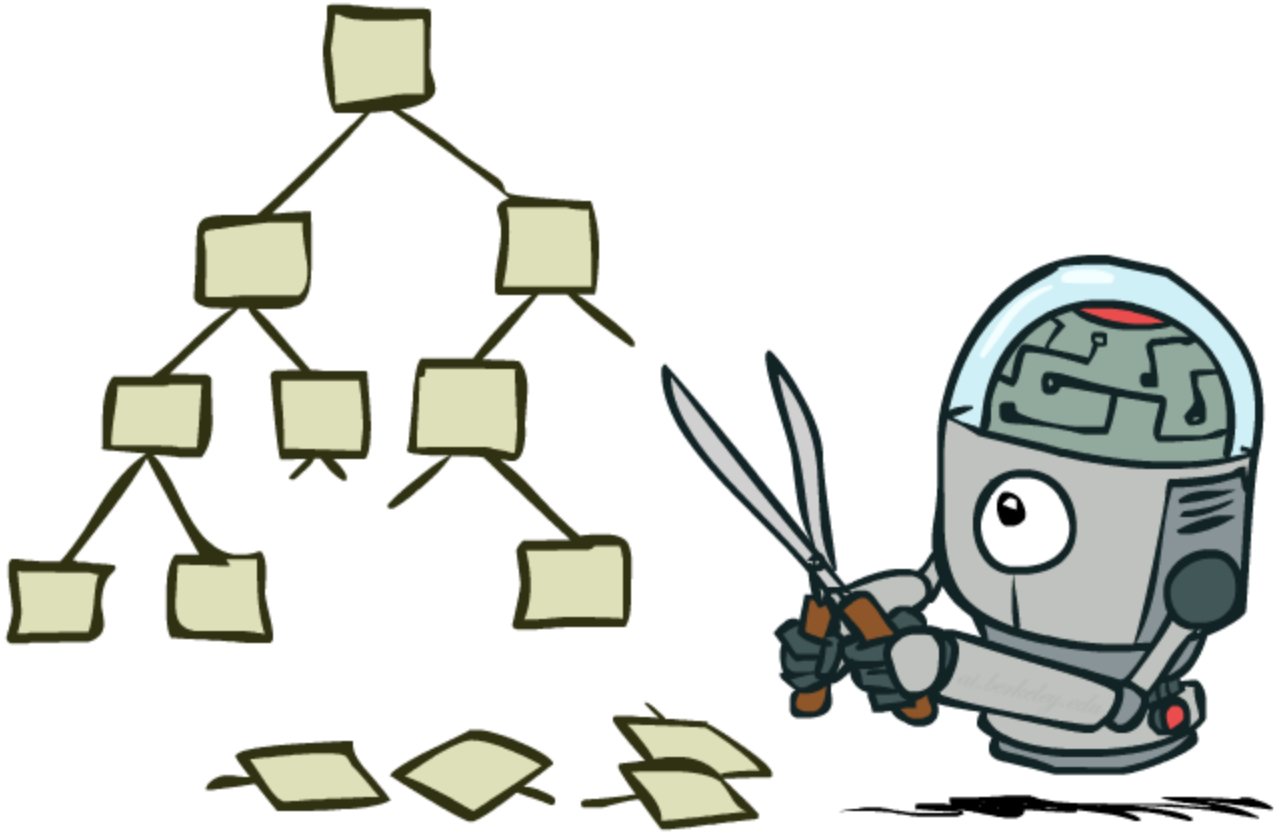


- A danger of replanning agents!
  - Knows his score will go up by eating the dot now (west, east)
  - Knows his score will go up just as much by eating the dot later (east, west)
  - No point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

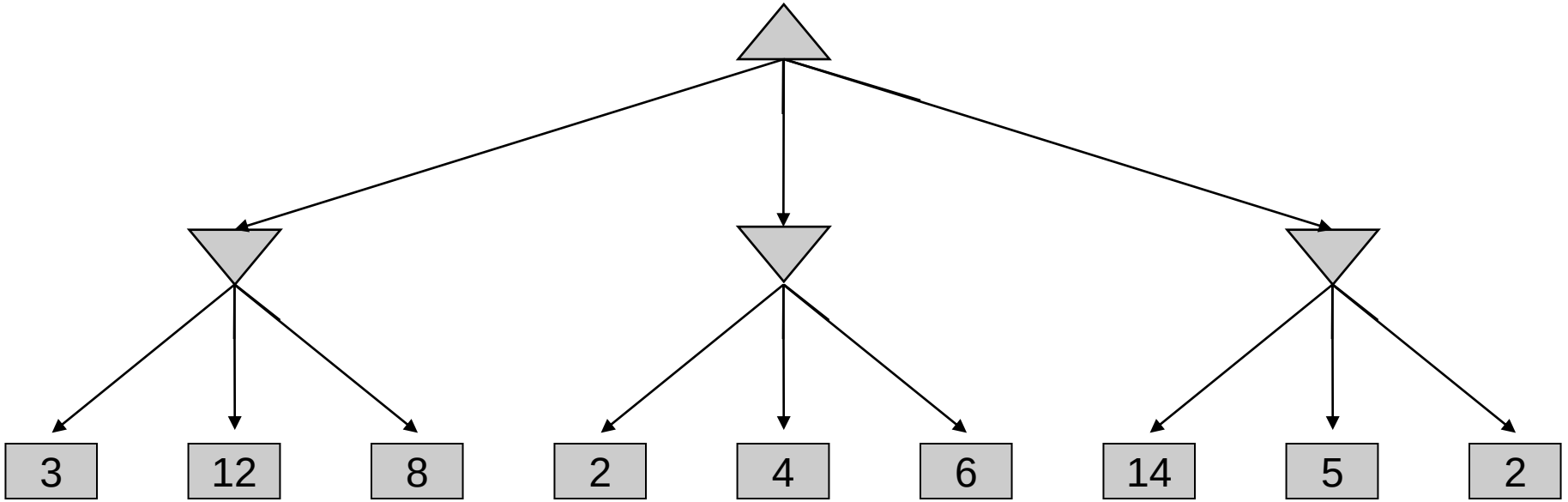
Learning Objectives

- ✓ Types of adversarial games
- ✓ Basic nature of adversarial search (min/max)
- ✓ New concept: value of a state
- ✓ Resource limits and value of non-terminal states
- Pruning – Alpha Beta
- Expectimax
- Imperfect Play → "Non Math" Factors

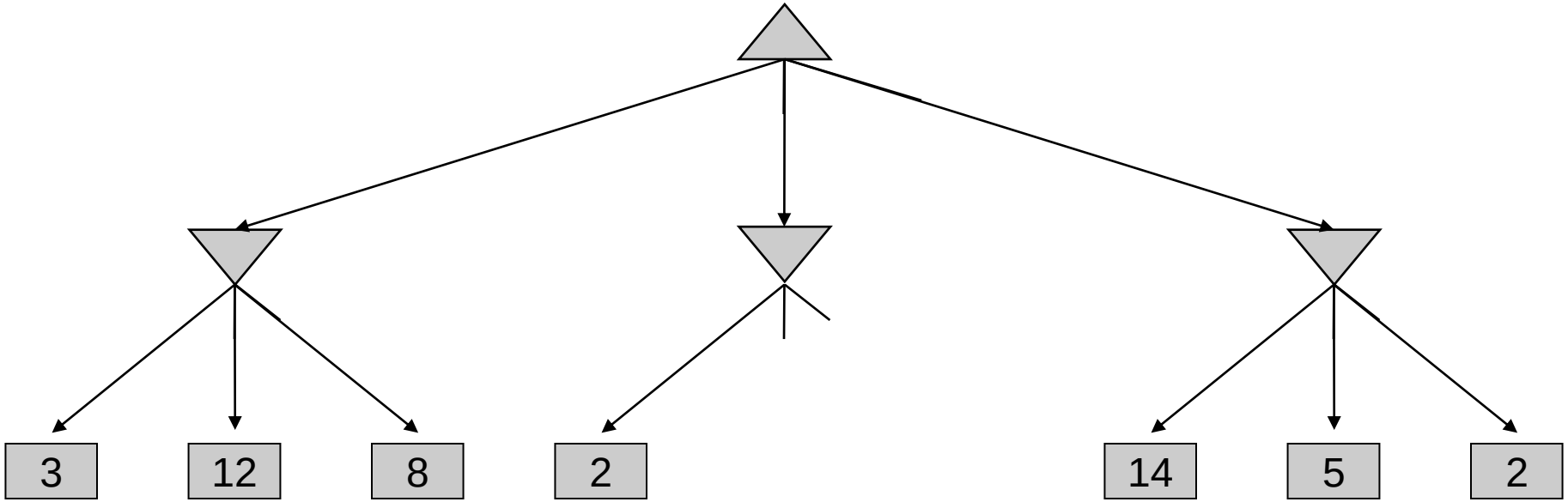
Game Tree Pruning



Minimax Example

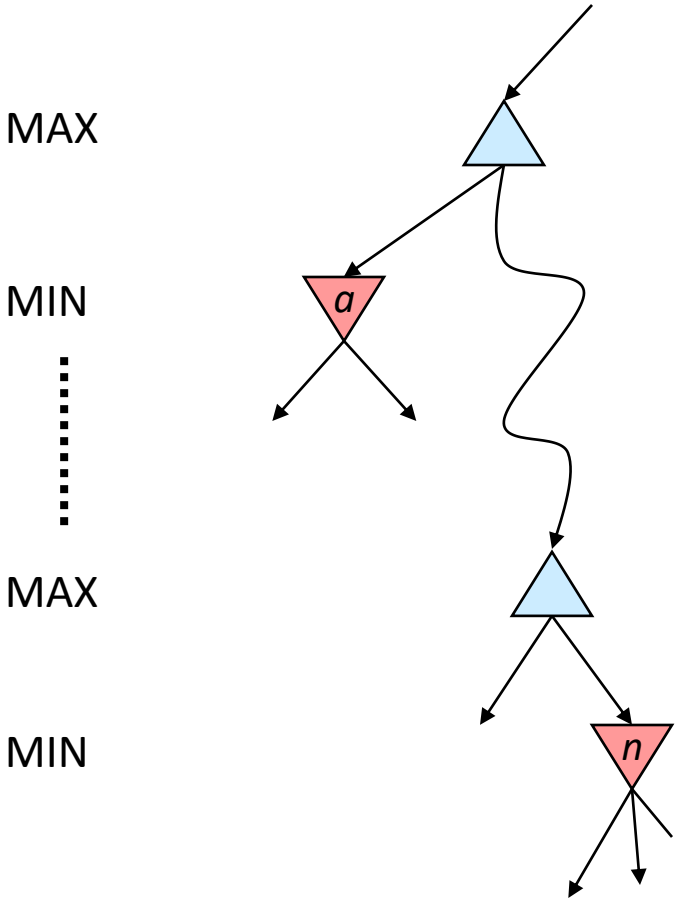


Minimax Pruning



Alpha-Beta Pruning

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the childrens' min is dropping
  - Who cares about *n*'s value? MAX
  - Let *a* be the best value that MAX can get at any choice point along the current path from the root
  - If *n* becomes worse than *a*, MAX will avoid it, so we can stop considering *n*'s other children (it's already bad enough that it won't be played)



- MAX version is symmetric

Alpha-Beta Implementation

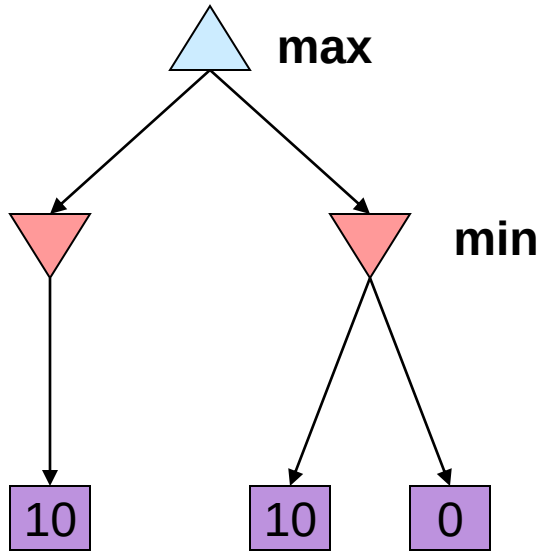
α: MAX's best option on path to root
β: MIN's best option on path to root

```
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
```

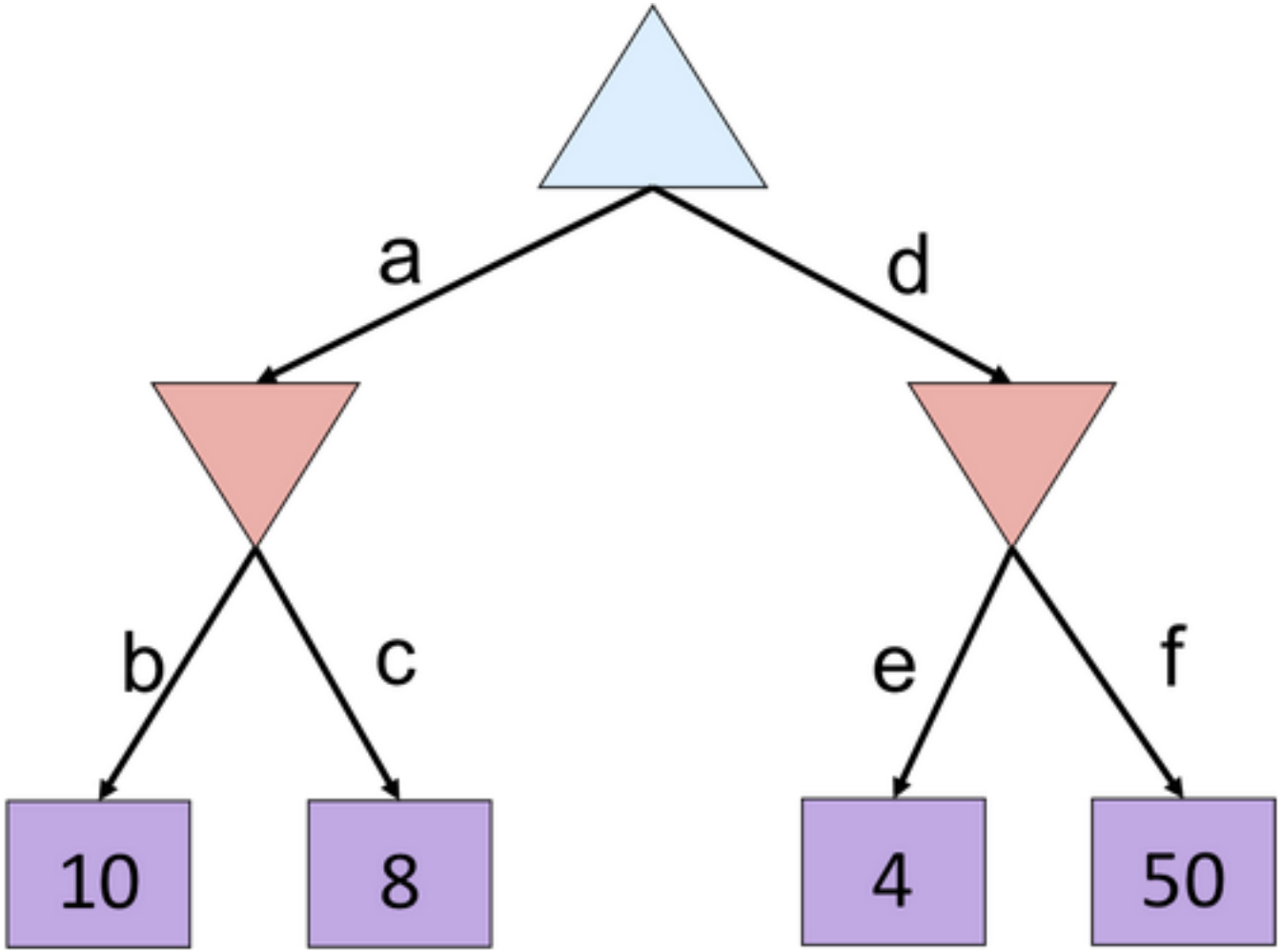
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-Beta Pruning Properties

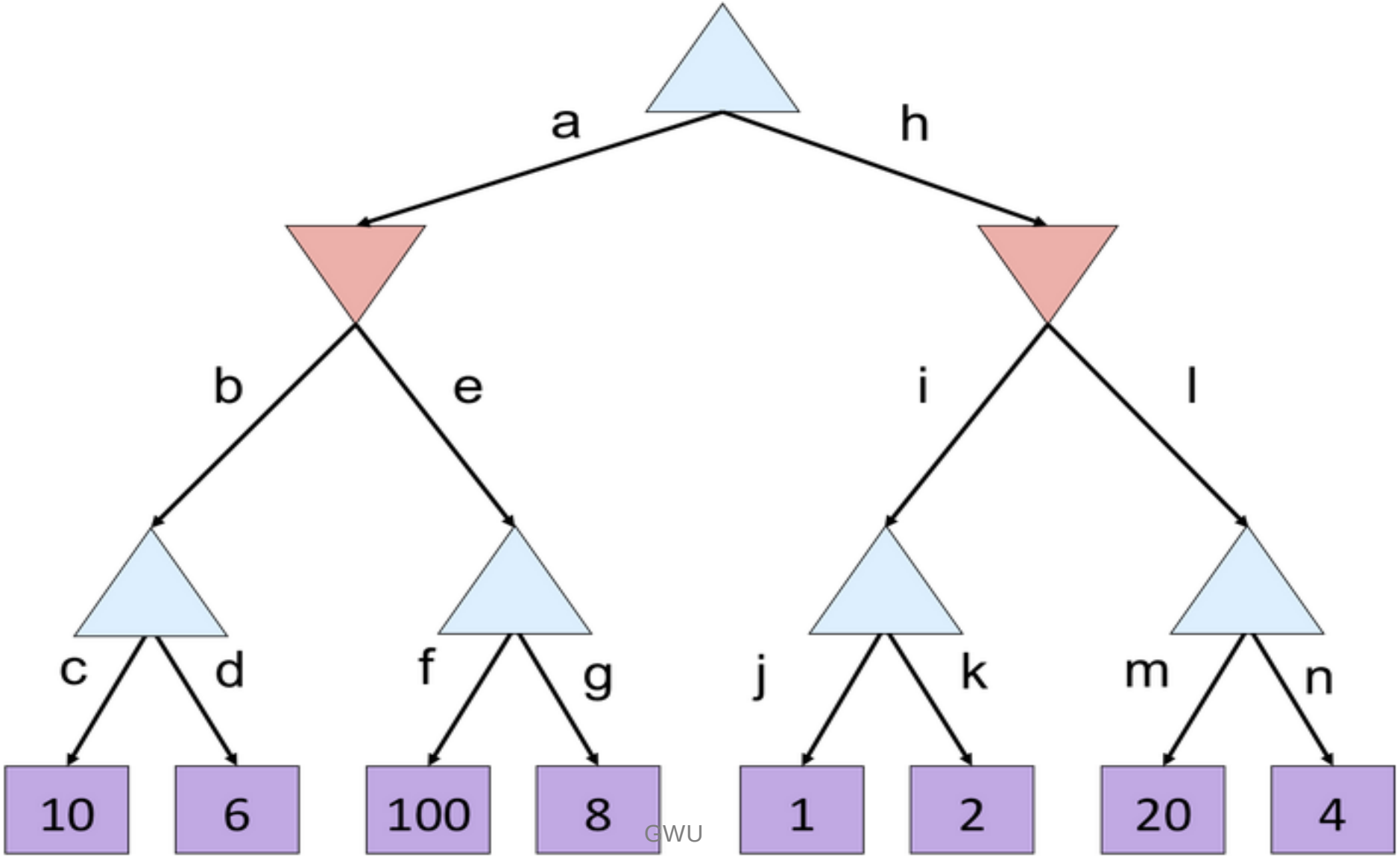
- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- This is a simple example of metareasoning (computing about what to compute)



Alpha-Beta Quiz



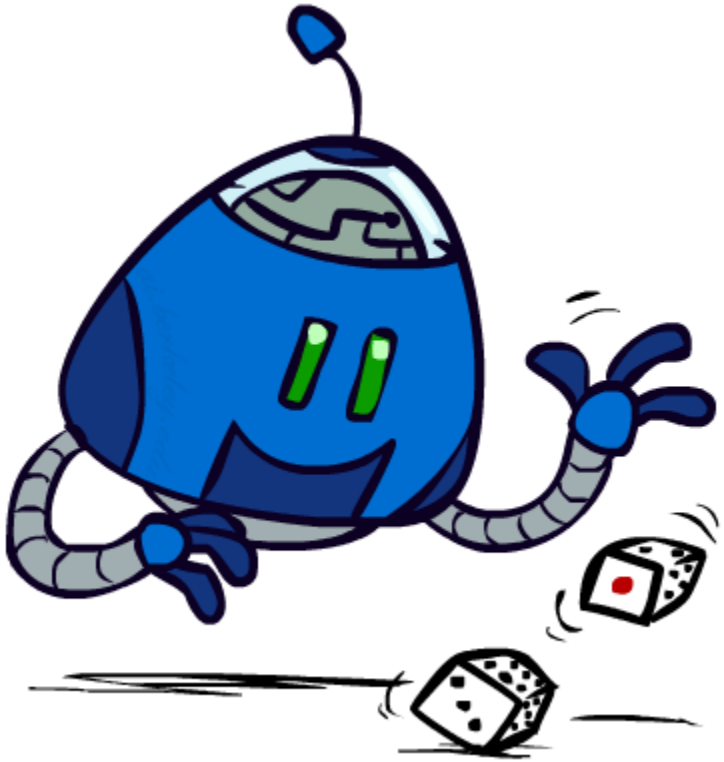
Alpha-Beta Quiz 2



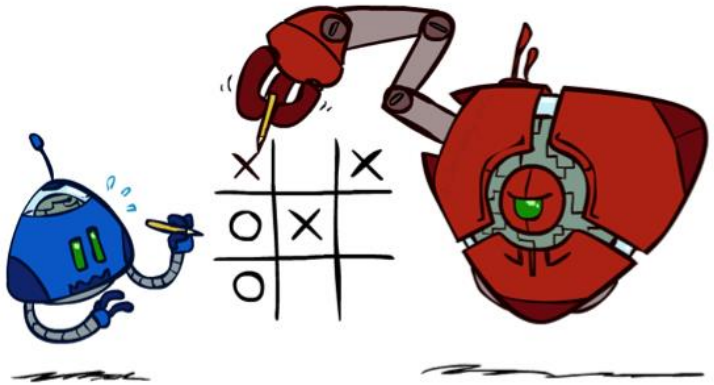
Learning Objectives

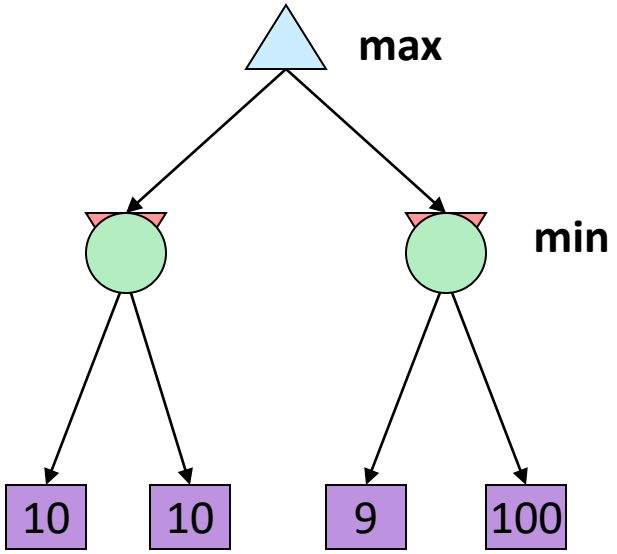
- ✓ Types of adversarial games
- ✓ Basic nature of adversarial search
- ✓ New concept: value of a state
- ✓ Resource limits and value of terminal states
- ✓ Pruning – Alpha Beta
- Expectimax
- Imperfect Play → "Non Math" Factors

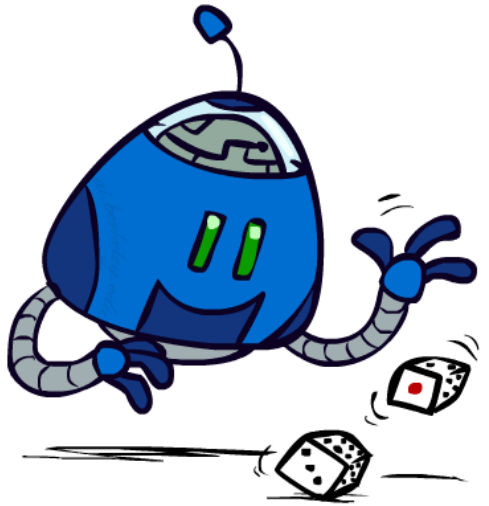
Uncertain Outcomes



Worst-Case vs. Average Case



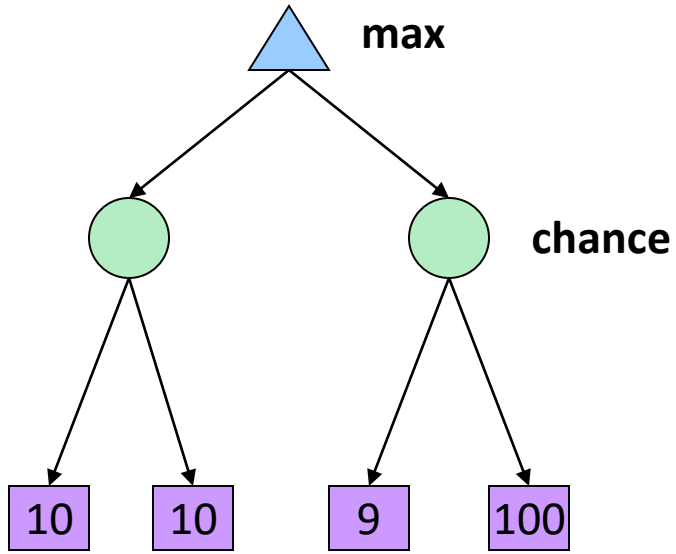




Idea: Uncertain outcomes controlled by chance, not an adversary!

Expectimax Search

- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip
- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes
- 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. take weighted average (expectation) of children
- Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes



Expectimax Pseudocode

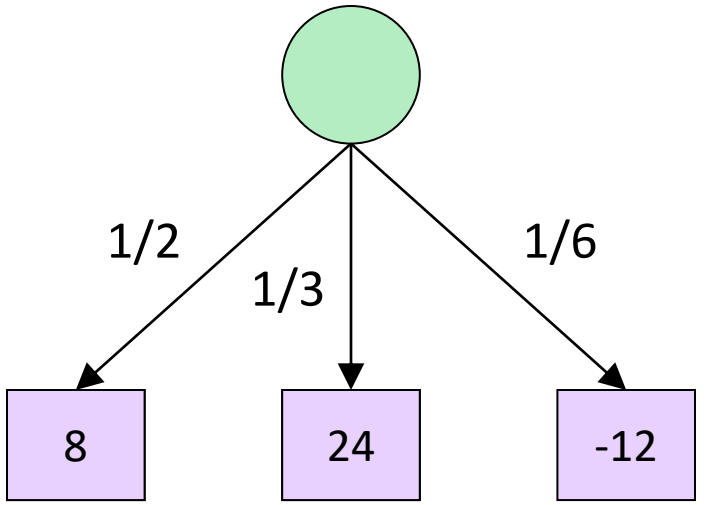
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

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

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

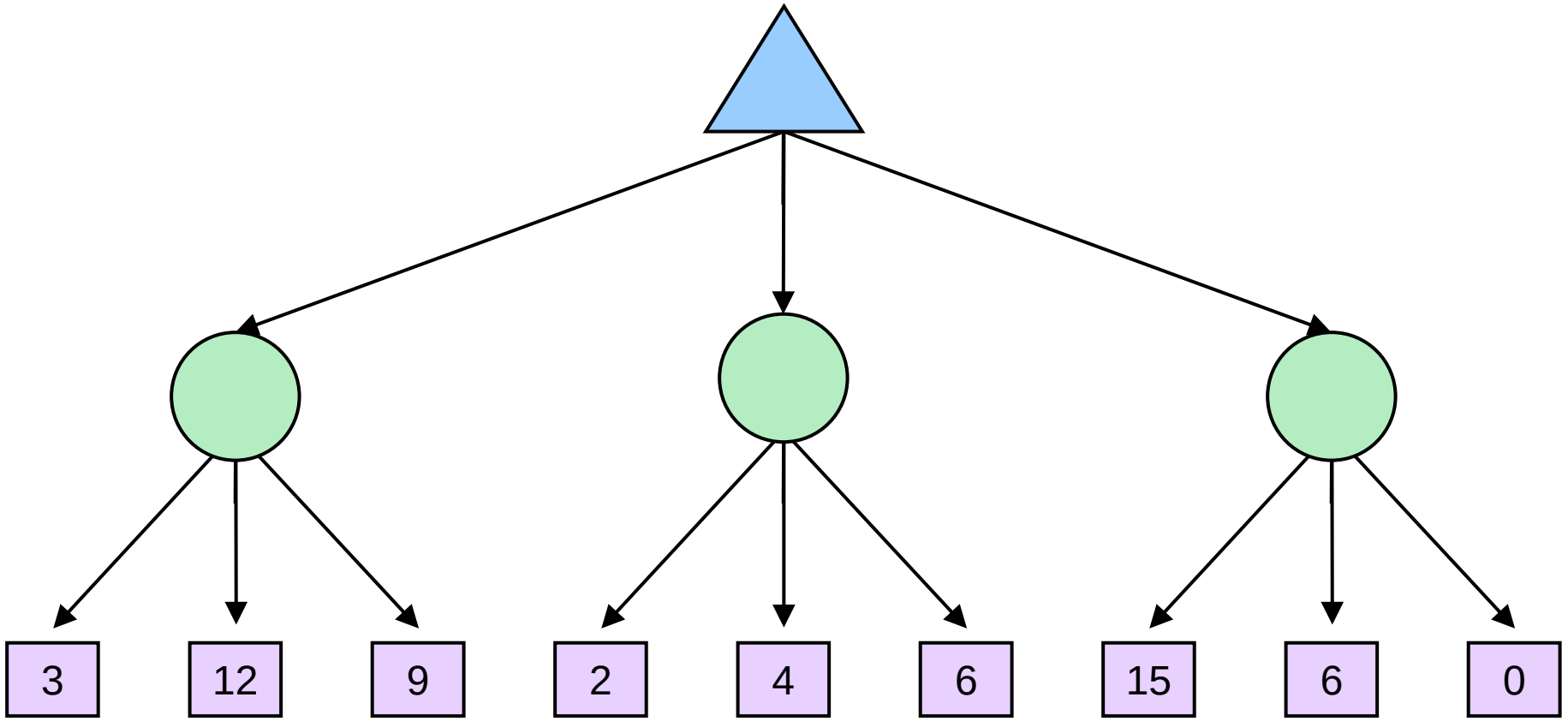
Expectimax Pseudocode

```
def exp-value(state):
        initialize v = 0
        for each successor of state:
                p = probability(successor)
                v += p * value(successor)
        return v
```

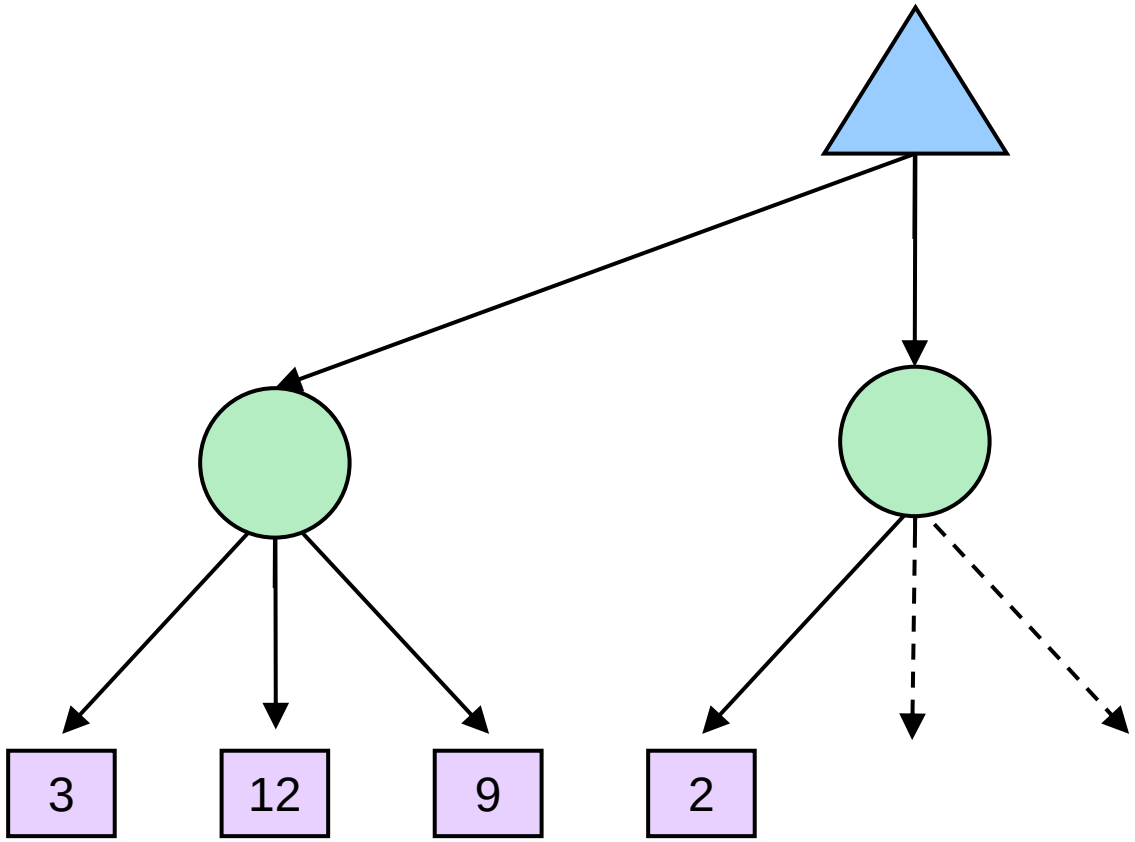


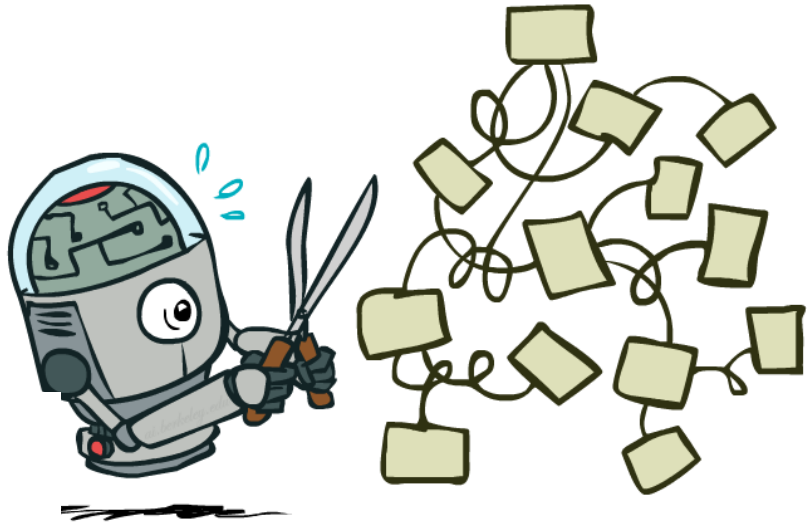
v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

Expectimax Example



Expectimax Pruning?

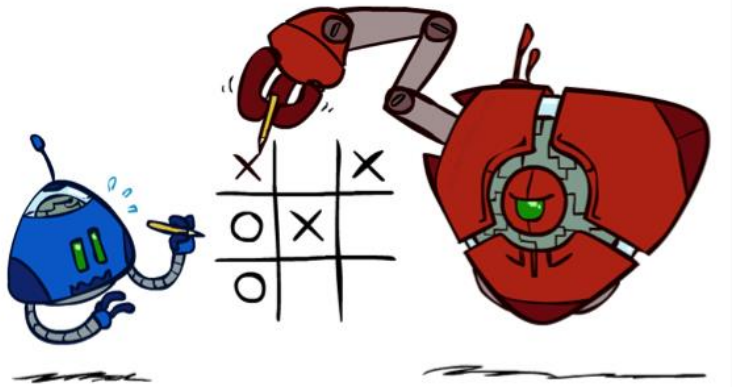


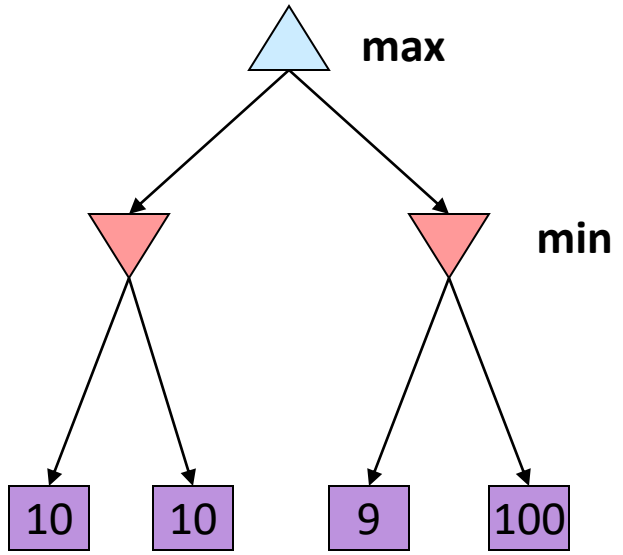


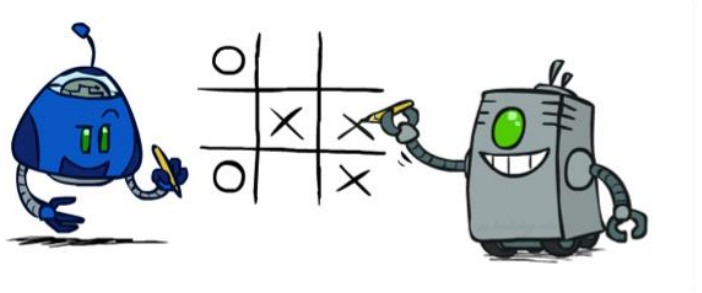
Learning Objectives

- ✓ Types of adversarial games
- ✓ Basic nature of adversarial search
- ✓ New concept: value of a state
- ✓ Resource limits and value of terminal states
- ✓ Pruning – Alpha Beta
- ✓ Expectimax
- Imperfect Play & “Non Math” Factors

Perfect Play vs. Imperfect Play







Optimal against a perfect player. Otherwise?