CPSC 481
Artificial Intelligence

Mr. Ayush Bhardwaj

abhardwaj@fullerton.edu

What we will cover this week

- Adversarial search
  - Game trees
  - Zero-sum games
  - Minimax algorithm
  - Evaluation functions
  - α- β pruning

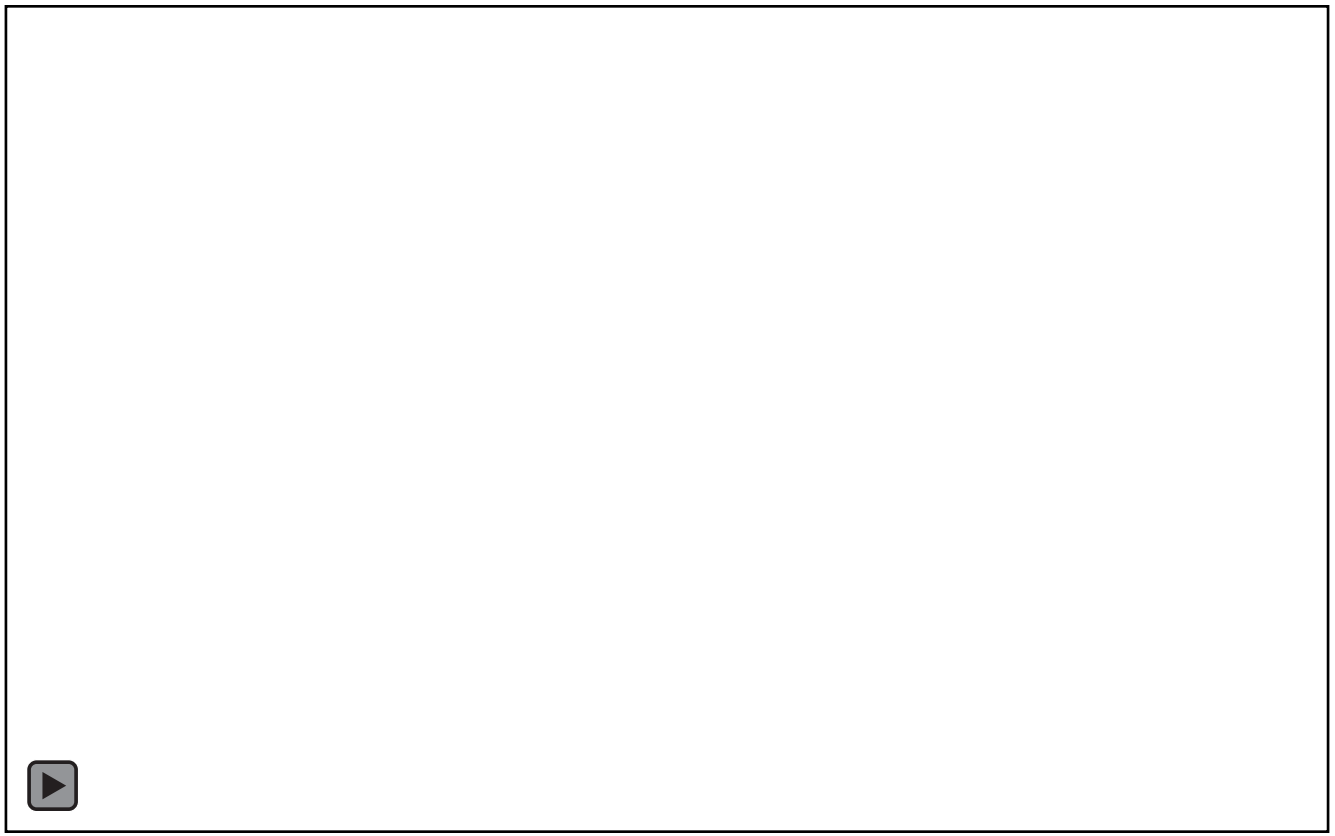
Game Playing

Why do AI researchers study game playing?

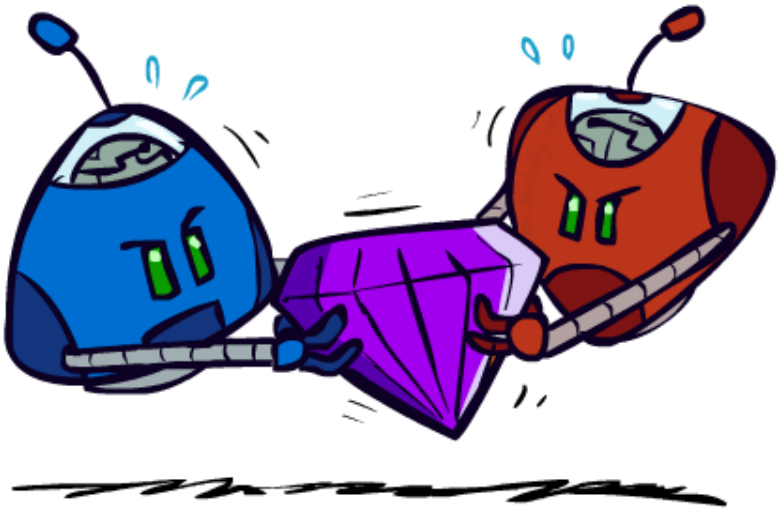
1. It's a good reasoning problem, formal and nontrivial.

2. Direct comparison with humans and other computer programs is easy.

Behavior from Computation

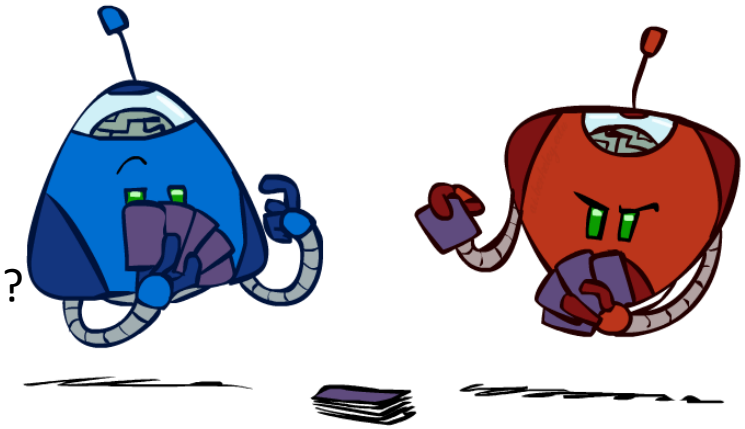


Adversarial Games



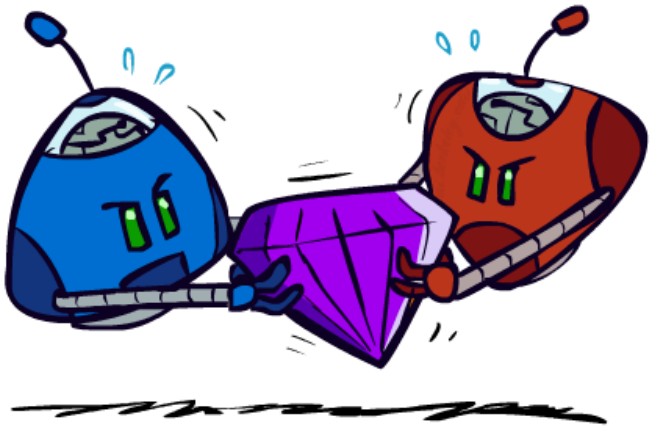
Types of Games

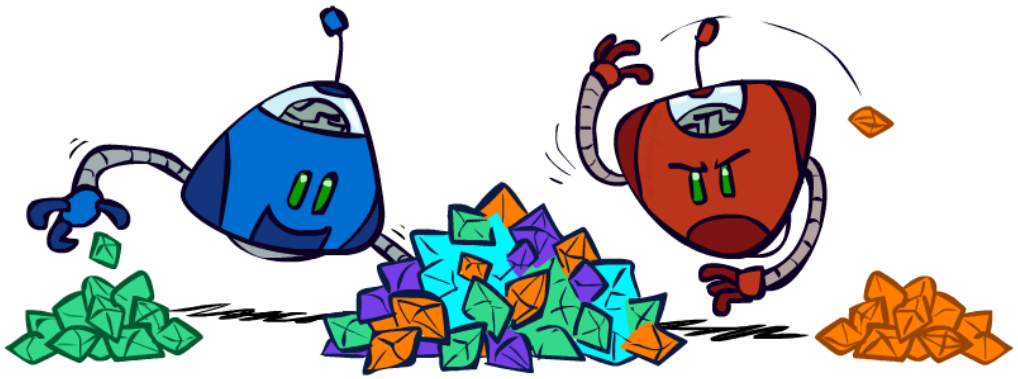
- Many different kinds of games
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?



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

Zero-Sum Games





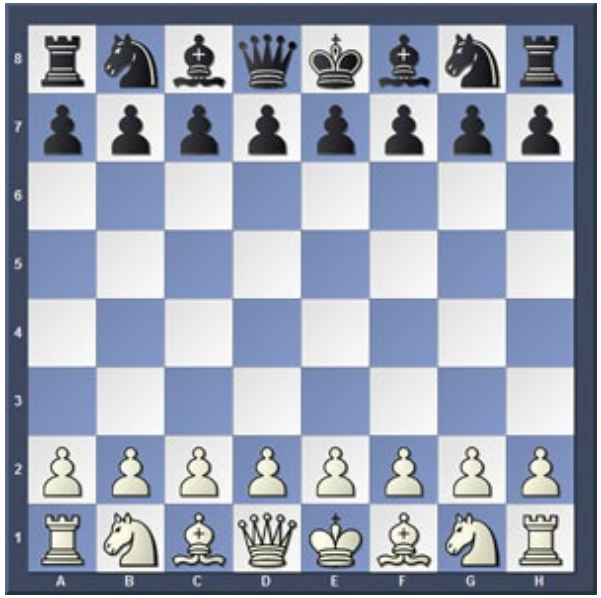
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - One player tries to maximize value, the other player minimizes it
  - a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities
  - Cooperation, indifference, competition, and more are all possible

In this class, we will focus on:

- Deterministic, 2-player, zero-sum games with perfect information
- Examples: Tic-tac-toe, Chess, checkers, Othello

Example: chess

- Players: Black, White
- State *s*: position of all pieces, whose turn it is
- Actions(*s*): legal chess moves that Player(s) can make
- IsTerminal(*s*): whether *s* is checkmate or draw
- Utility(*s*): +1 if white wins, 0 if draw, -1 if black wins

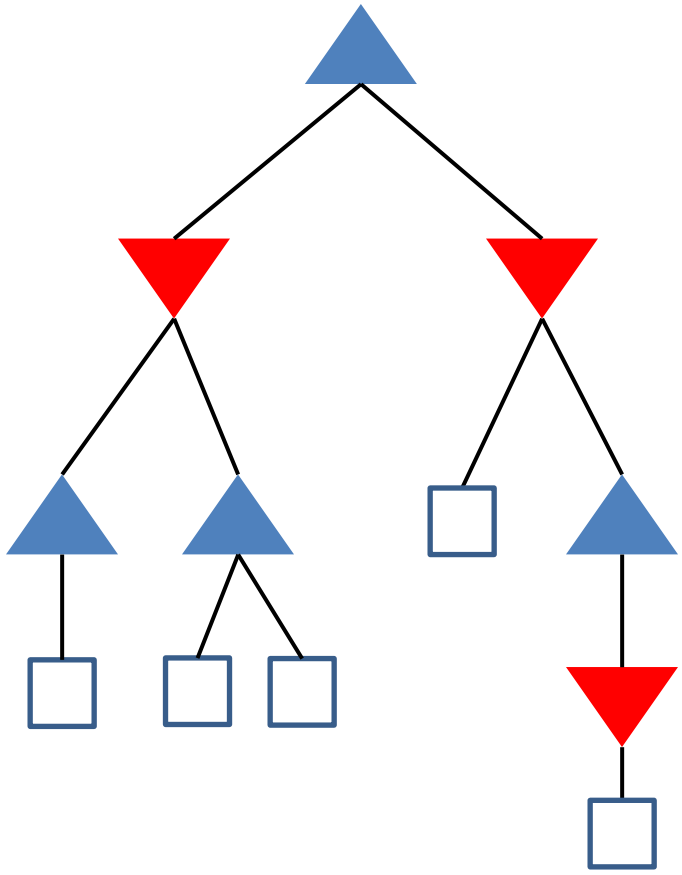


Halving game

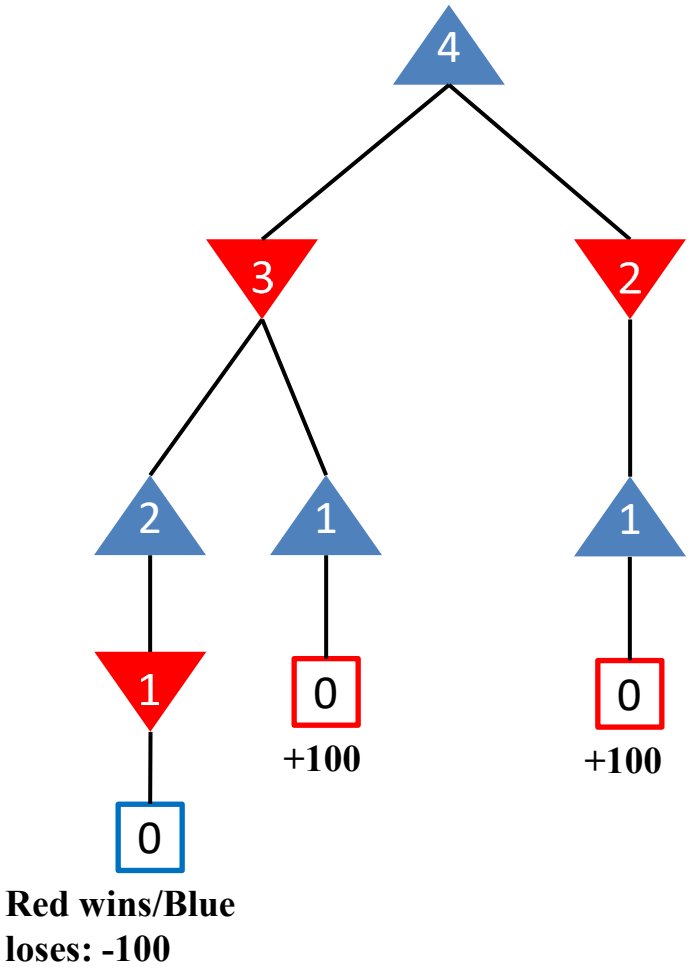
- Start with some number N
- Players take turns. Two actions:
  - Decrement number, or
  - Replace with floor(N/2) (e.g., floor(5/2) = 2)
- The player that makes it 0 wins
- A win is worth +100 points, loss is -100
- Start with N=4
- What is the optimal action at the beginning?

Game tree

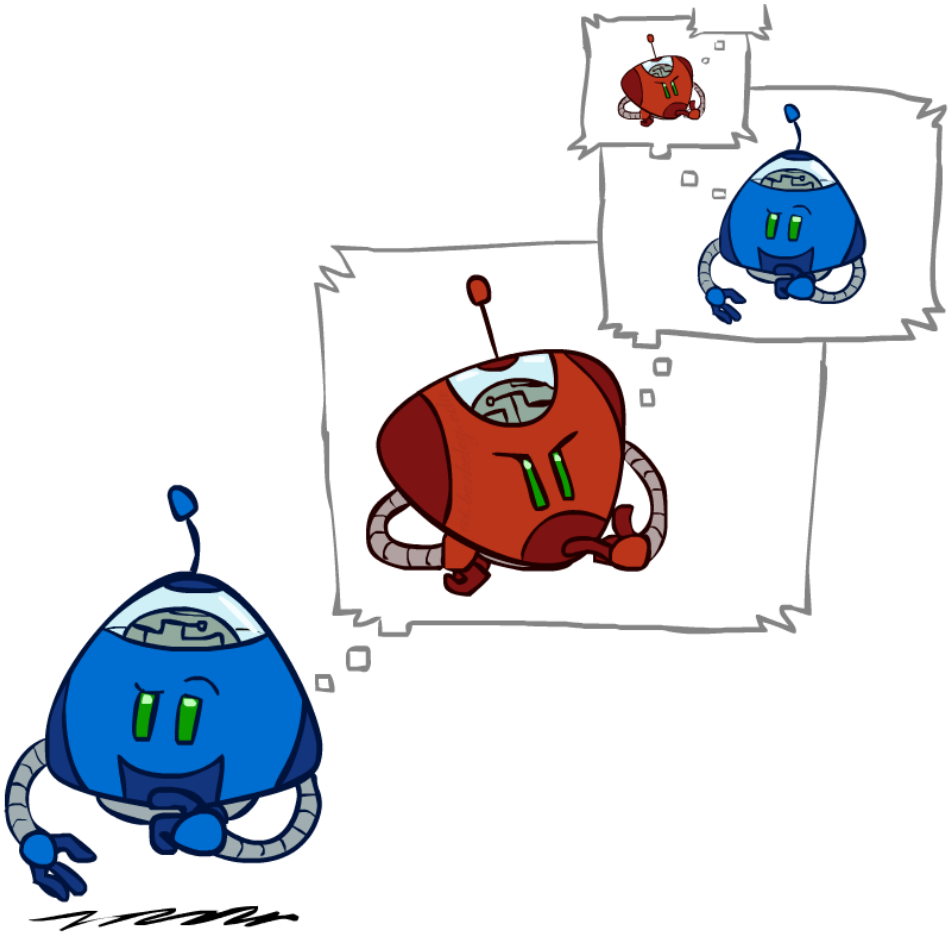
- Each node is a decision point for a player
- Decisions alternate at every level
  - Root: me (maximizing player)
  - 2nd level: opponent (minimizing player)
  - 3rd level: opponent (maximizing player again)
  - …
- Leaves are terminal states: outcome is known
  - a win/loss/tie/points
  - Can assign a utility value from the point of view of the maximizing player
- Each root-to-leaf path is a possible outcome of the game



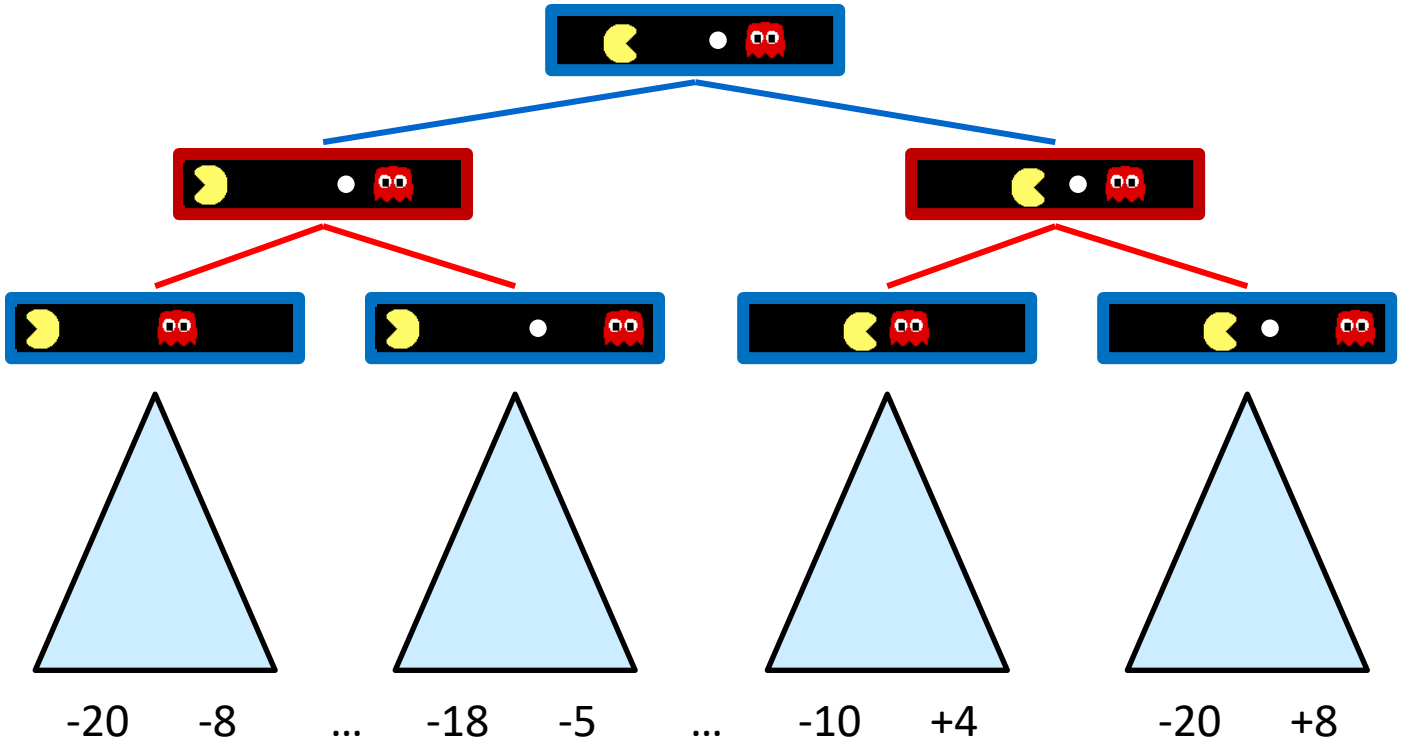
Game tree for the halving game with N=4



Adversarial Search



Adversarial Game Trees



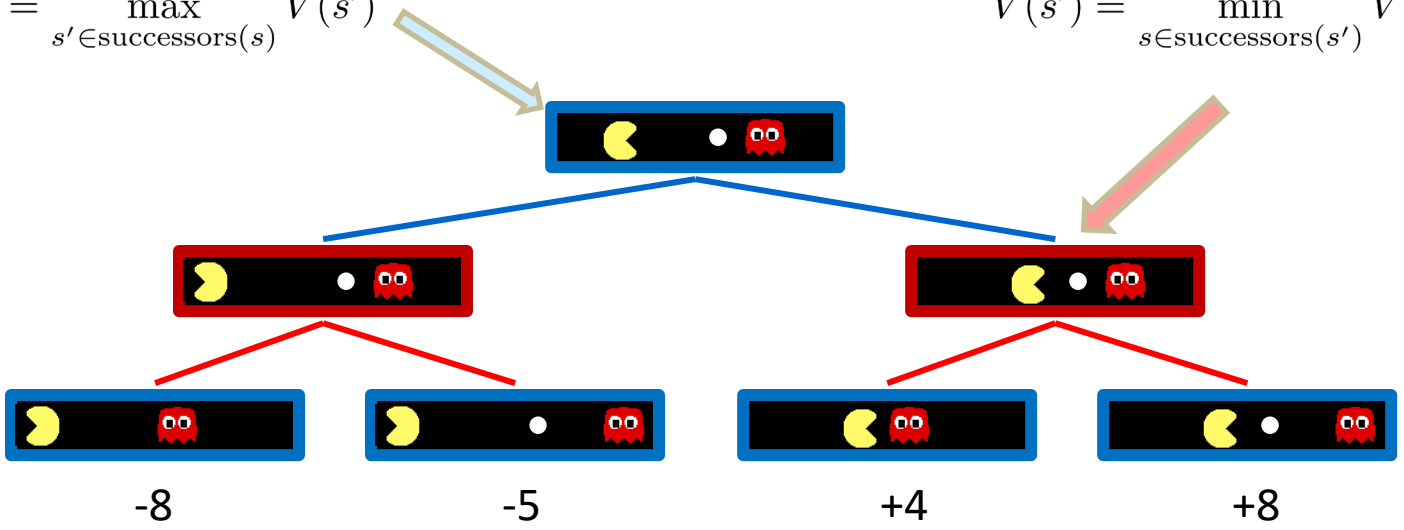
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

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



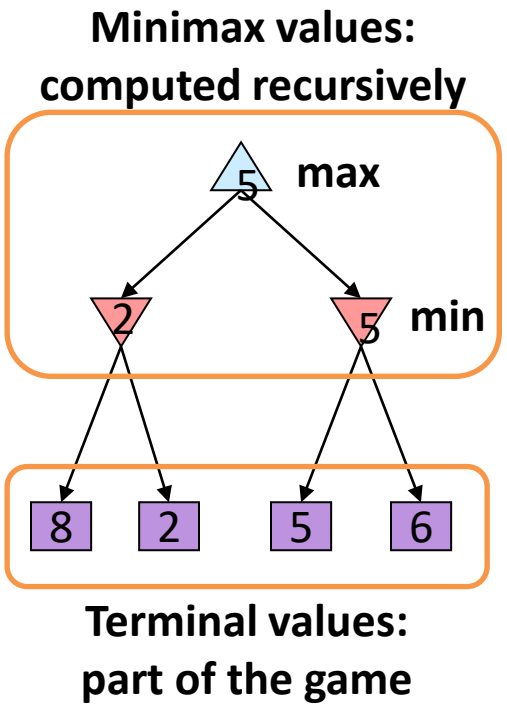
Terminal States:

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

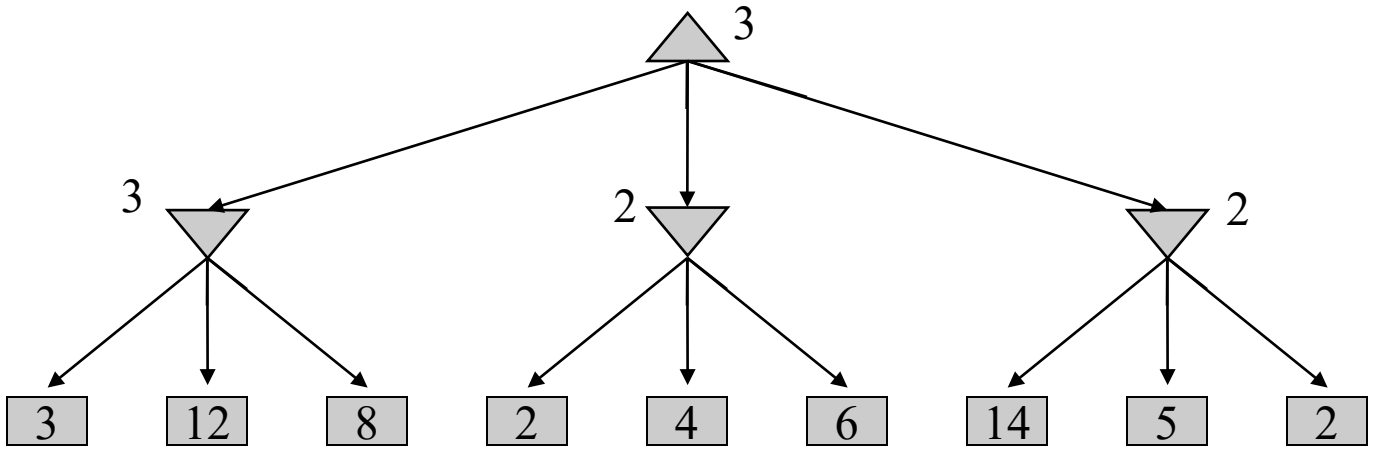
Adversarial Search (Minimax)

- 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



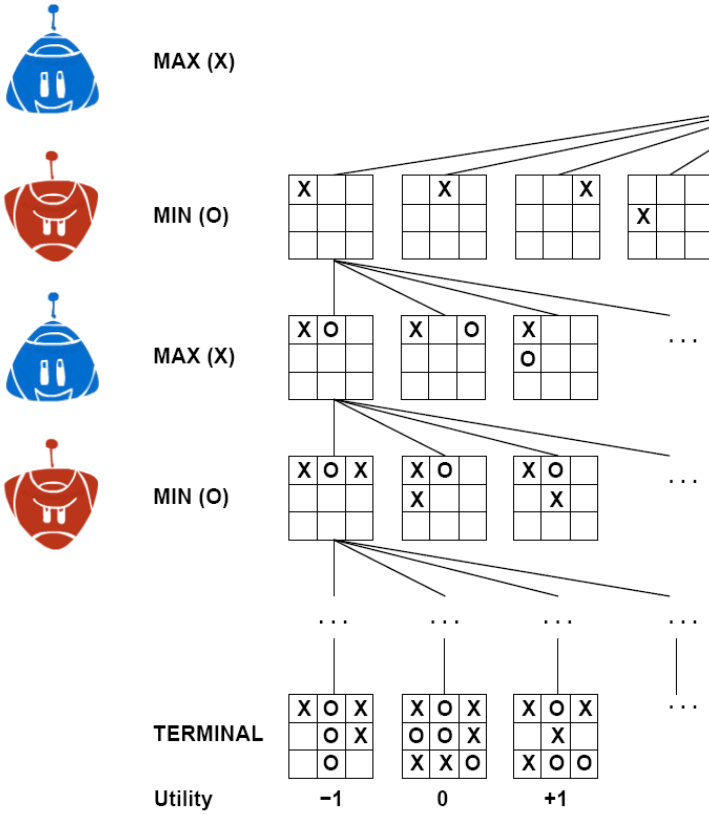
Minimax Example

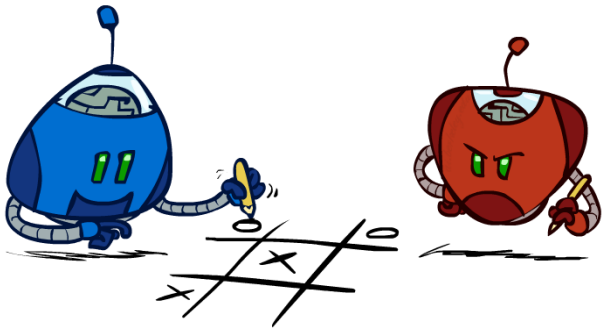


Class work

- Halving game
  - Start with a number N.
  - Players take turns. Actions
    - Decrement number, or
    - Replace with floor(N/2) (e.g., floor(5/2) = 2)
  - The player that makes it 0 wins
  - A win is worth +100 points, loss is -100
- Draw the game tree starting with N=5
- Calculate the minimax values at all nodes
  - In this game, don't be confused by the state of a node and its utility/value. The state just happens to also be a number in this game
- What is the optimal action?

Tic-Tac-Toe Game Tree





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)
```

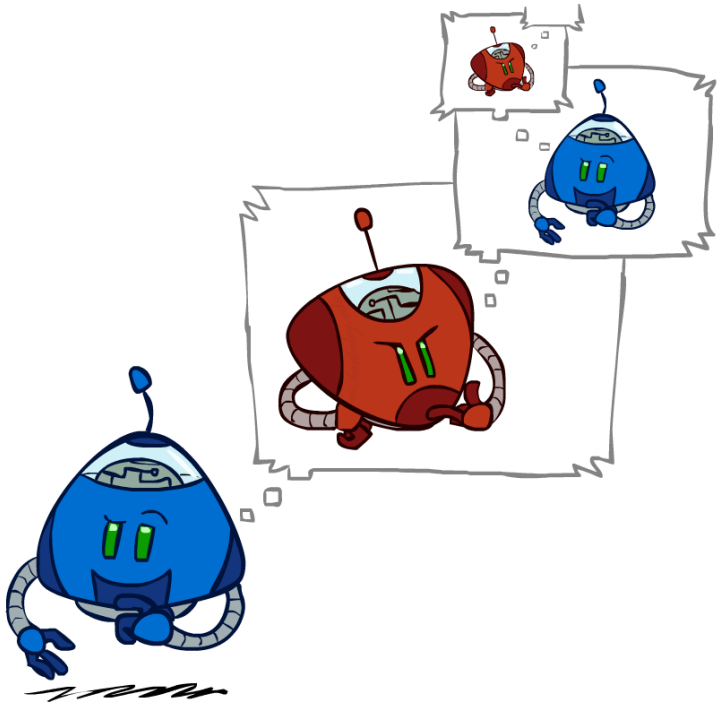
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

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

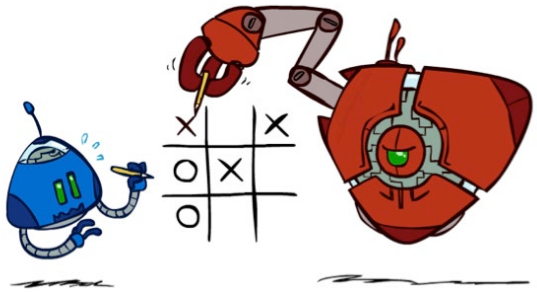
Minimax Efficiency

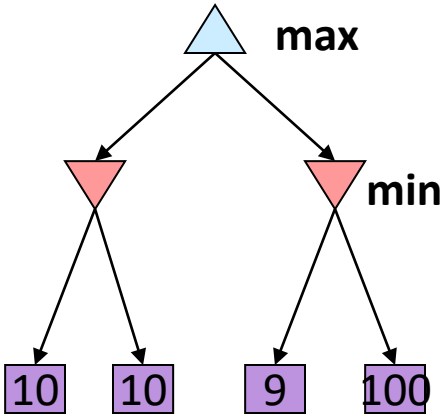
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: O(bm)

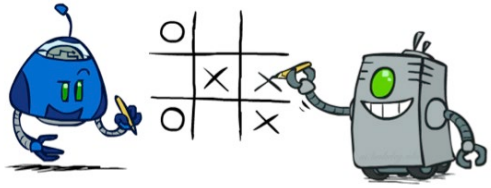
- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Minimax Properties







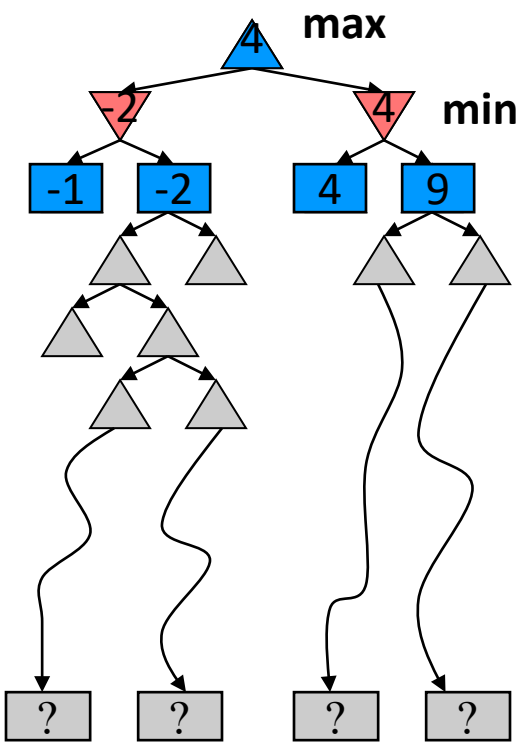
Optimal against a perfect player.

If player is known to be imperfect, other strategies could be better

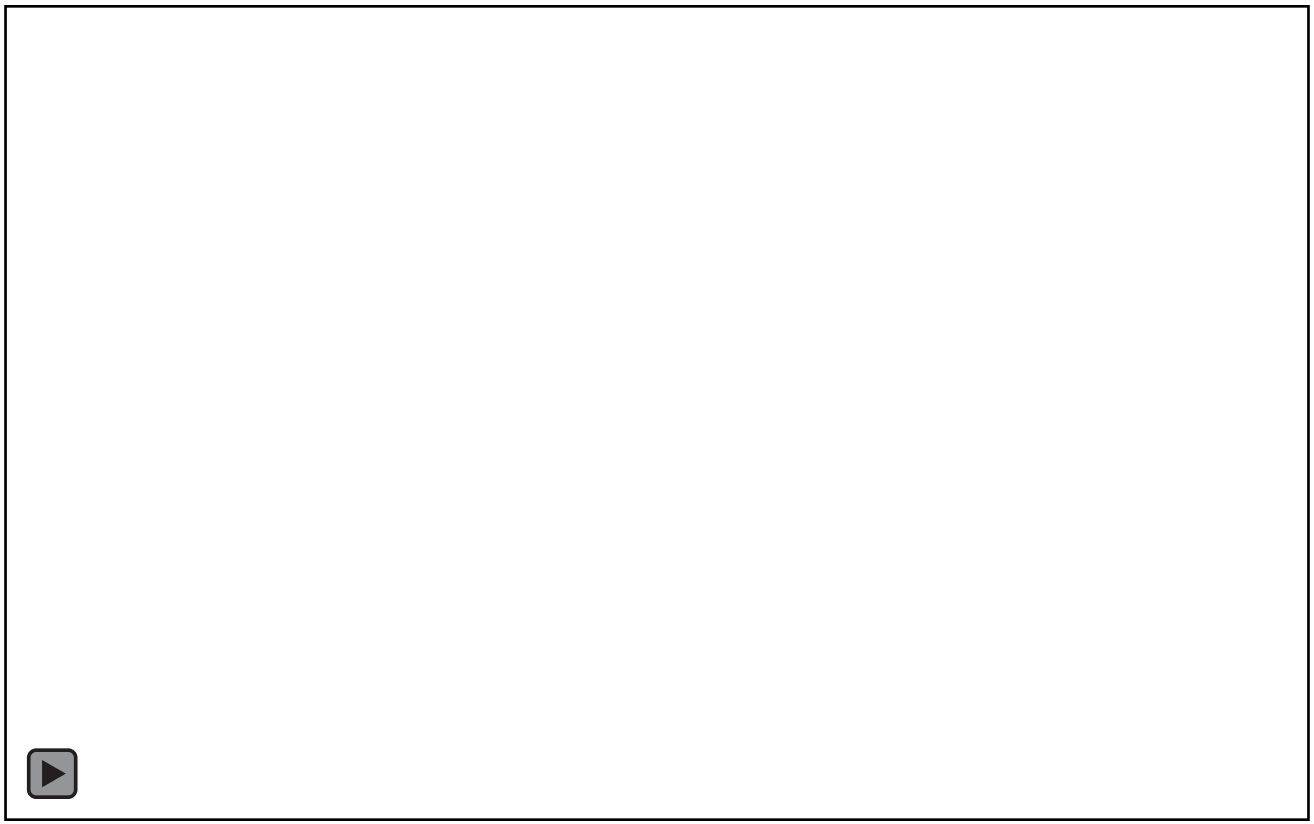
Resource Limits

Speeding up minimax

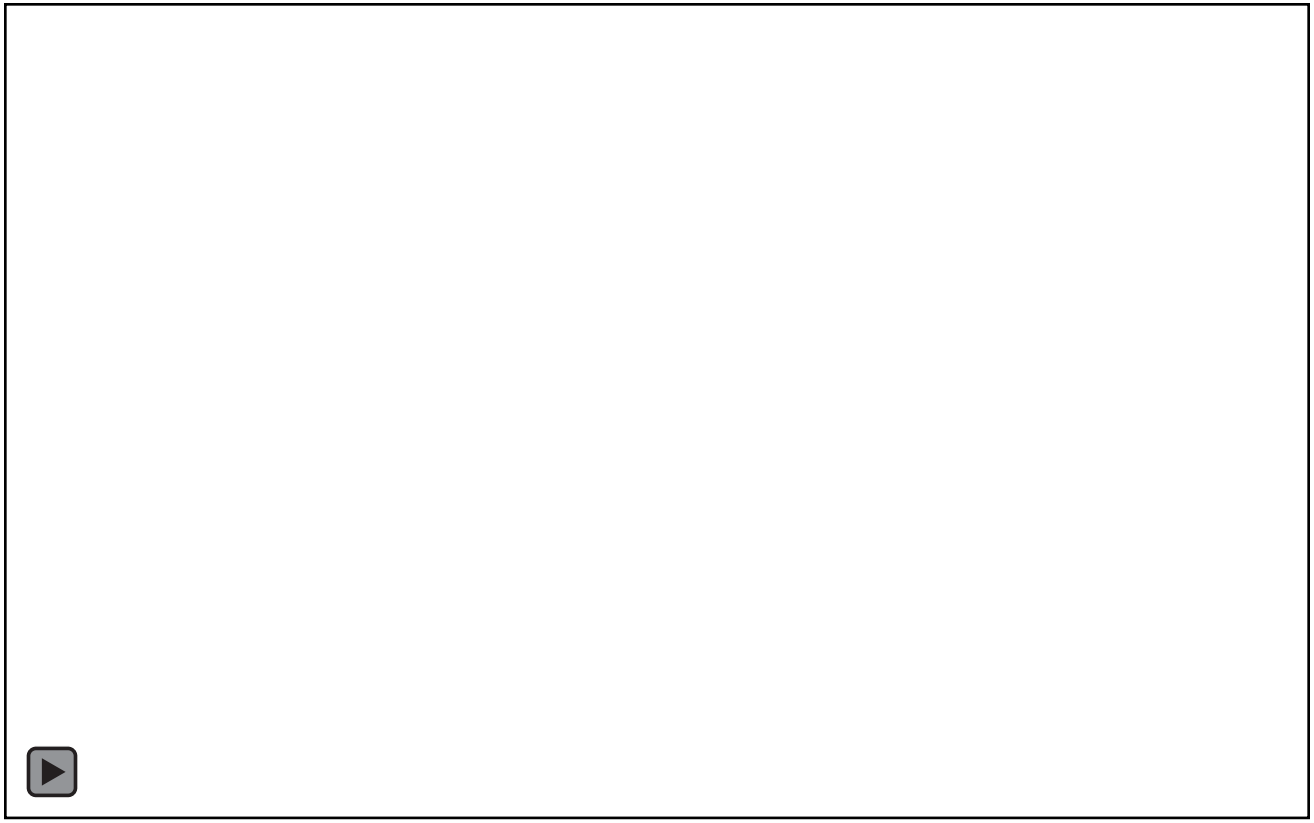
- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an **evaluation function** for non-terminal positions
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm



Video: Limited Depth (2)



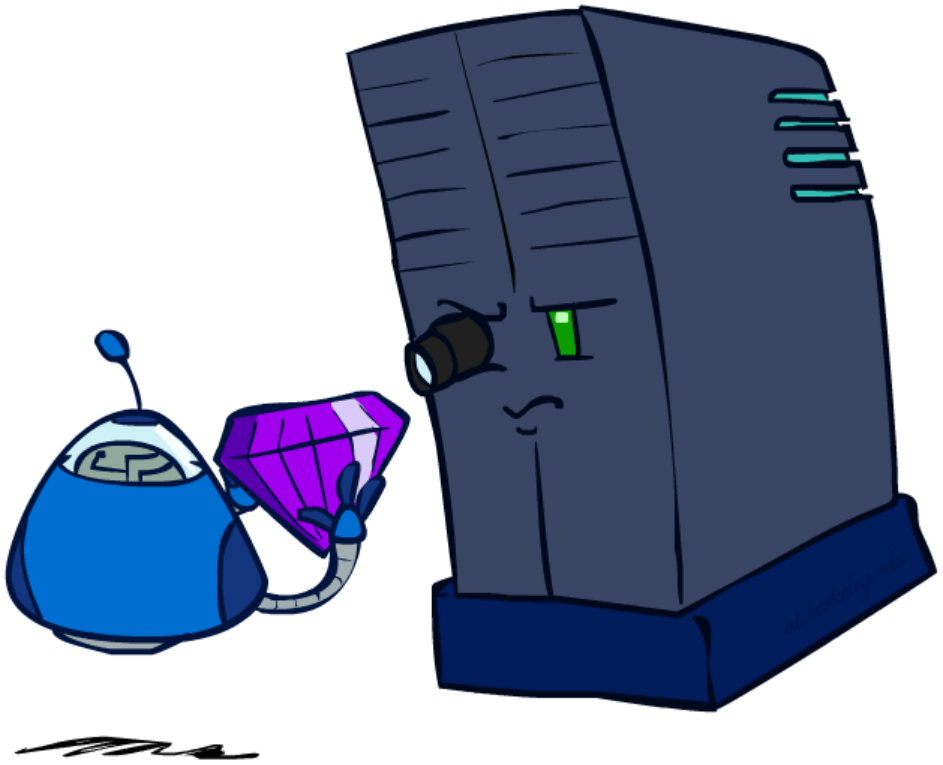
Video: Limited Depth (10)



Depth Matters

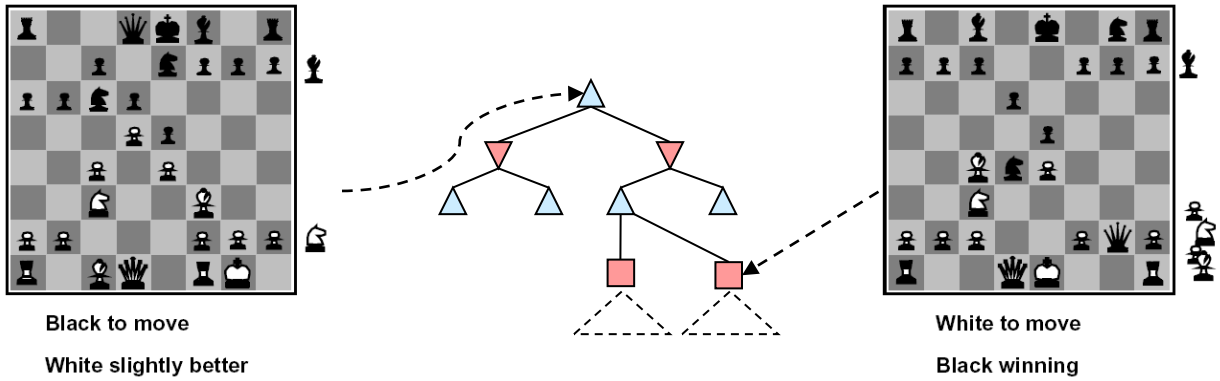
- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation

Evaluation Functions



Evaluation Functions

- Definition: An evaluation function is a (possibly very weak) estimate of the minimax value of an internal node
- Evaluation functions score non-terminals in depth-limited search

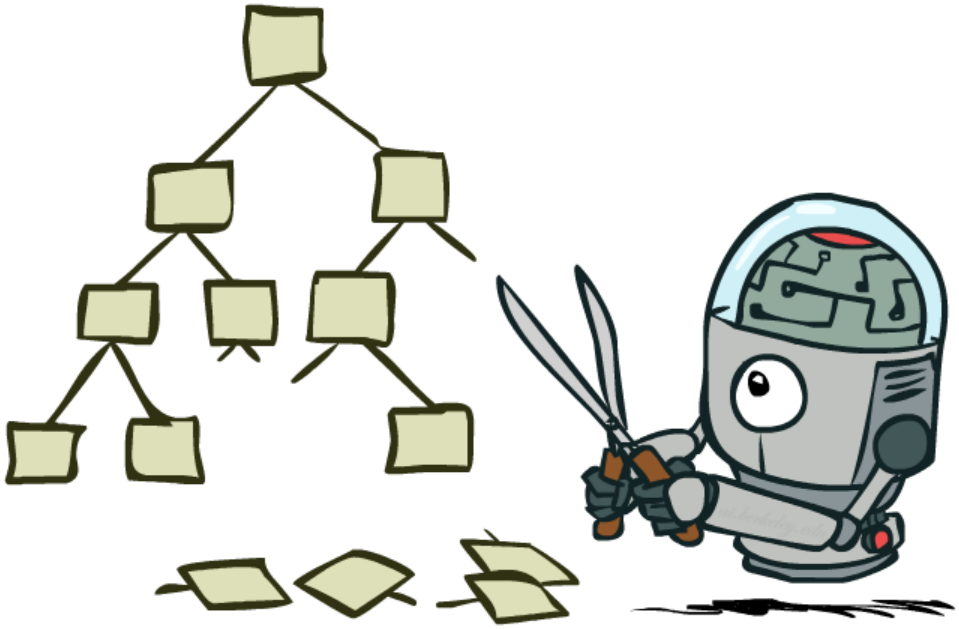


- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

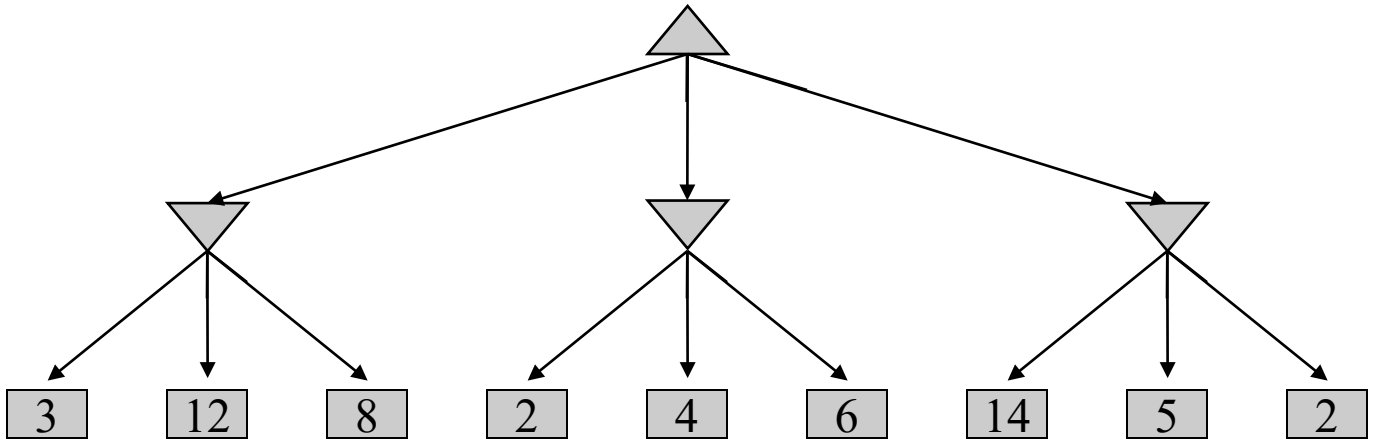
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g. $f_1(s)$ = 9*(QW – QB) + 5*(RW – RB) + 3*(KW – KB) + 3*(BW – BB) + 1*(PW – PB)

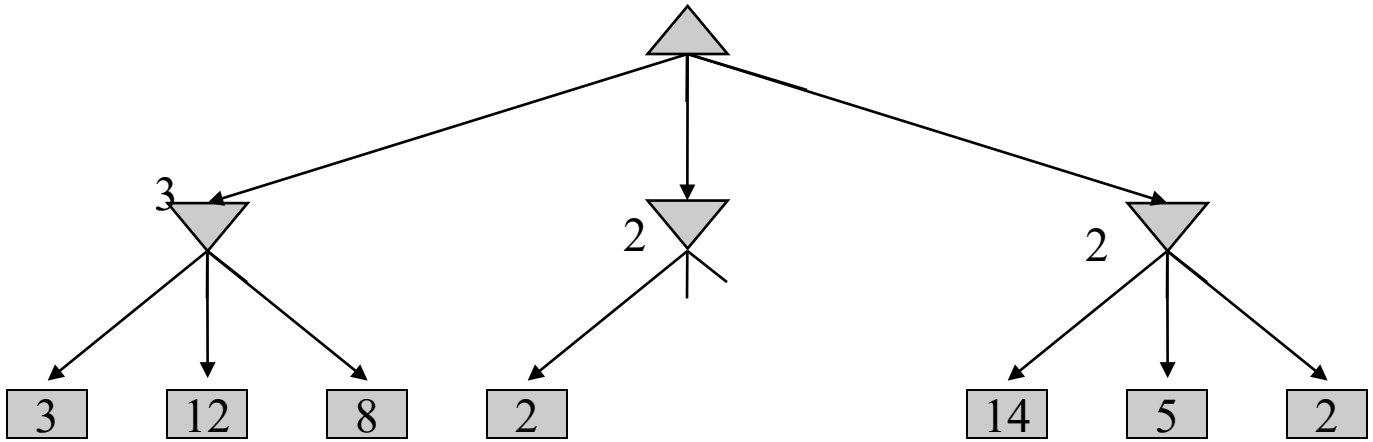
Game Tree Pruning



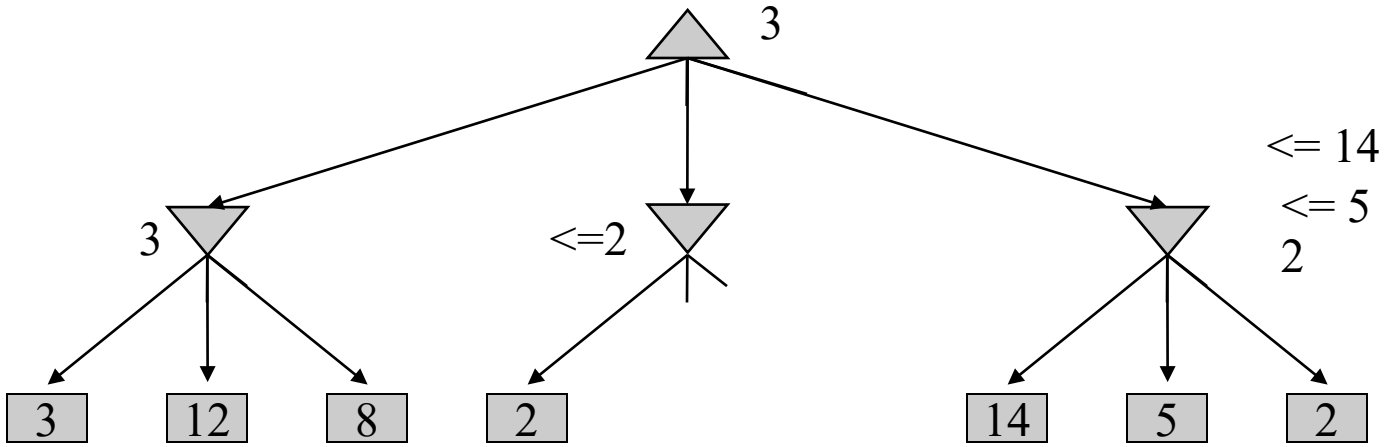
Minimax Example



Minimax Pruning

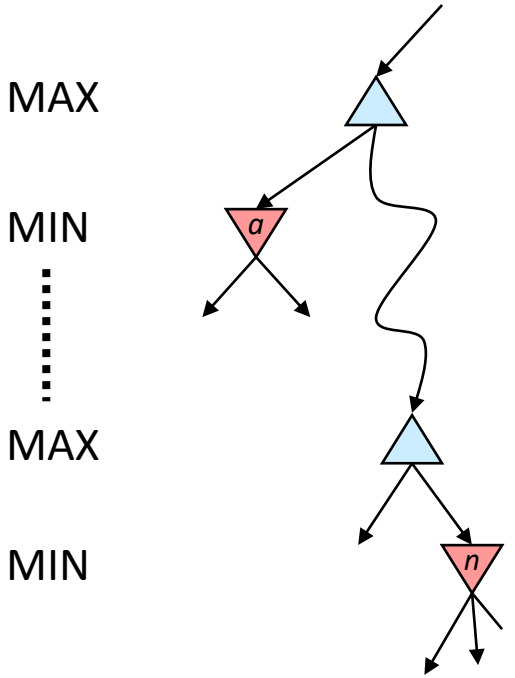


Minimax Pruning



Alpha-Beta Pruning

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *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

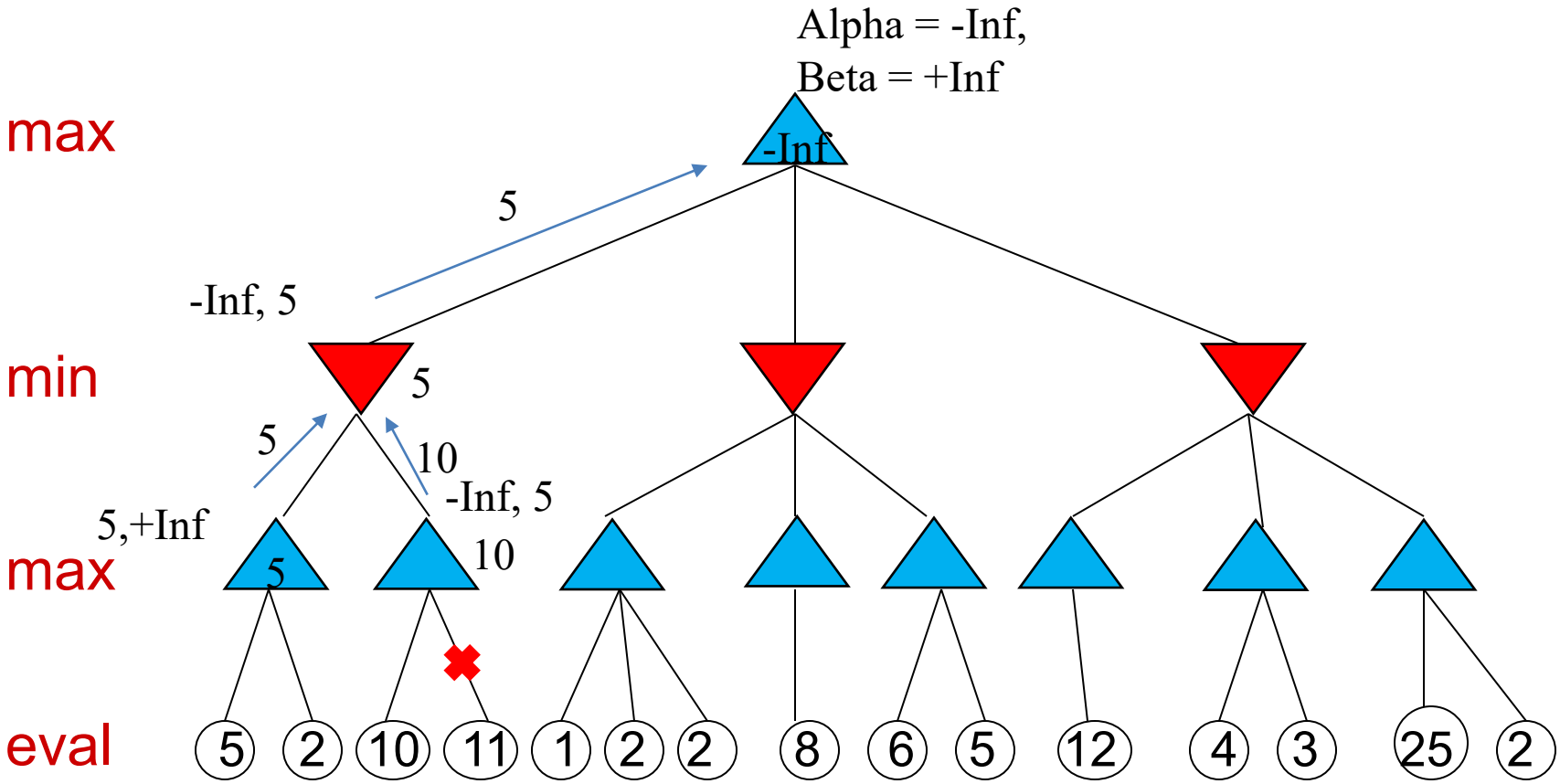
```
def value(state, α= -∞, β= +∞):
        if the state is a terminal state: return the state's utility
        if the next agent is MAX: return max-value(state)
        if the next agent is MIN: return min-value(state)
```

```
def max-value(state, 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
```

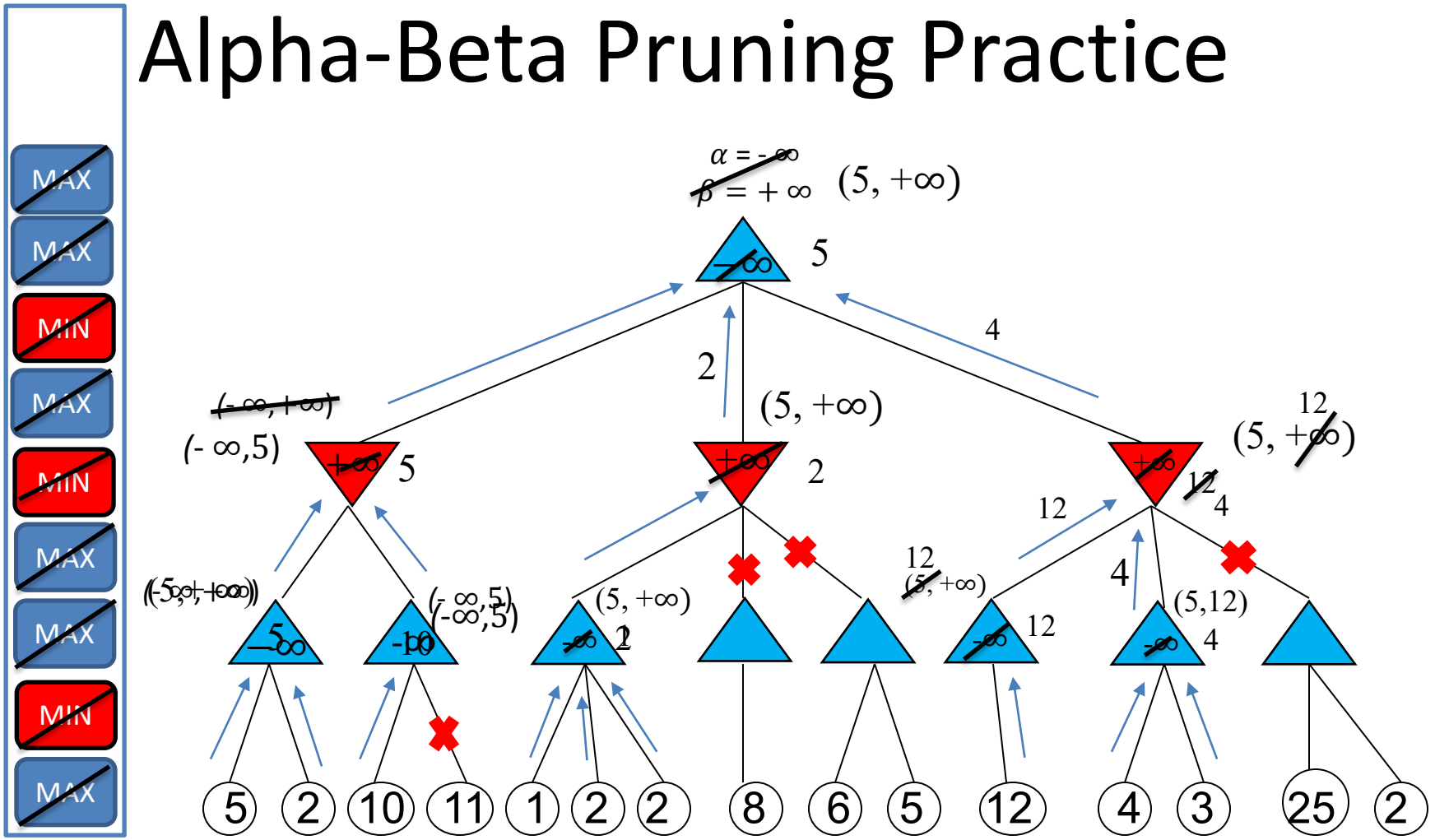
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

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

Alpha-Beta Pruning Practice

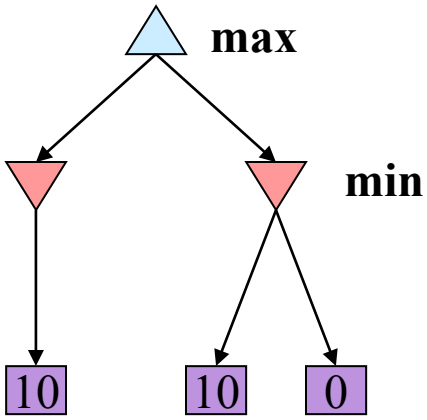


Alpha-Beta Pruning Practice



Alpha-Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!
  - No loss in accuracy
- 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 node ordering improves effectiveness of pruning
  - Ideally, expand "most promising" children (moves) first
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…



Other adversarial search algorithms

- Monte Carlo tree search (MTCS)
  - Developed in 2006
  - MCTS was combined with neural networks in 2016 for computer Go
  - Instead of trying to search every path, randomly pick one and go down that path
    - And repeat many times
- Expectimax
  - A variant of minmax for games with randomness
  - E.g., a roll of a dice
  - MAX player now maximizes the *expected value* of the opponent's moves

References

- George Luger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th edition, Addison Wesley, 2009. **Chapter 4.4**.
- Russel and Norvig, Artificial Intelligence: A Modern Approach, 3rd edition, Prentice Hall, 2010. **Chapter 5.1-5.3.**

Acknowledgement

- http://ai.berkeley.edu/lecture_slides.html
- **https://www.youtube.com/watch?v=cwbjLIahbv8**