CSC 665: Artificial Intelligence

Games: Adversarial Search

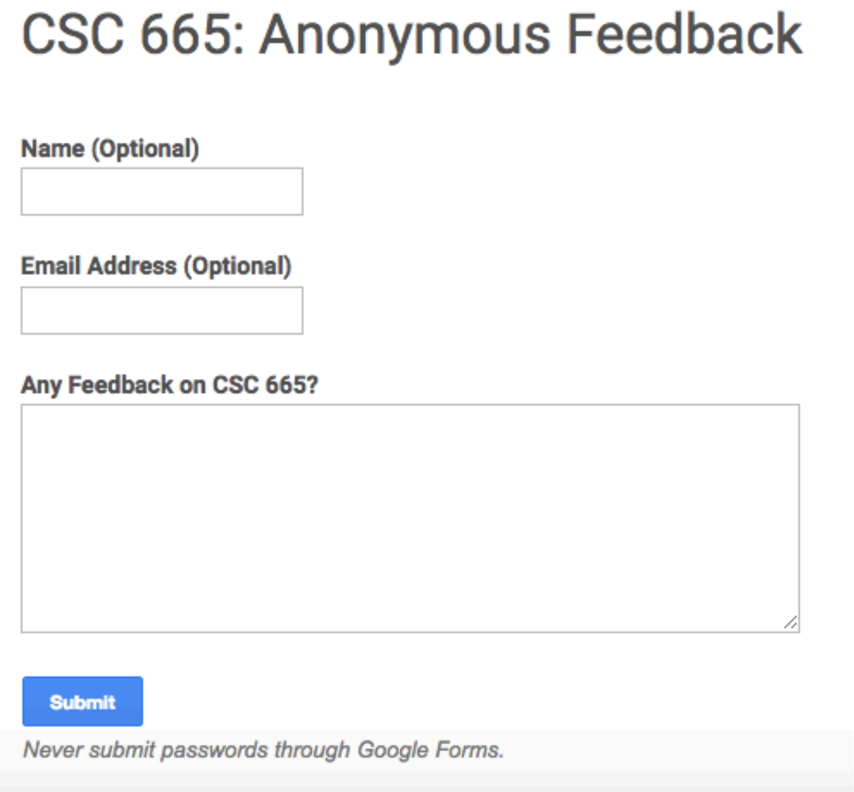
Instructor: Pooyan Fazli
San Francisco State University

How to Succeed in This Course

- Read all the slides
- Read all the required reading material
- Read the "Review Materials" on iLearn
- Concepts in this course take some time to sink in: be careful not to fall behind. Cramming will NOT do it
- Be active in lectures and on iLearn.
- Study in groups
- Ask us if you have the slightest doubt

Course Information: Feedback

- Please give feedback (positive or negative) as often as and as early as you can.



CSC 665: Anonymous Feedback

Name (Optional)

Email Address (Optional)

Any Feedback on CSC 665?

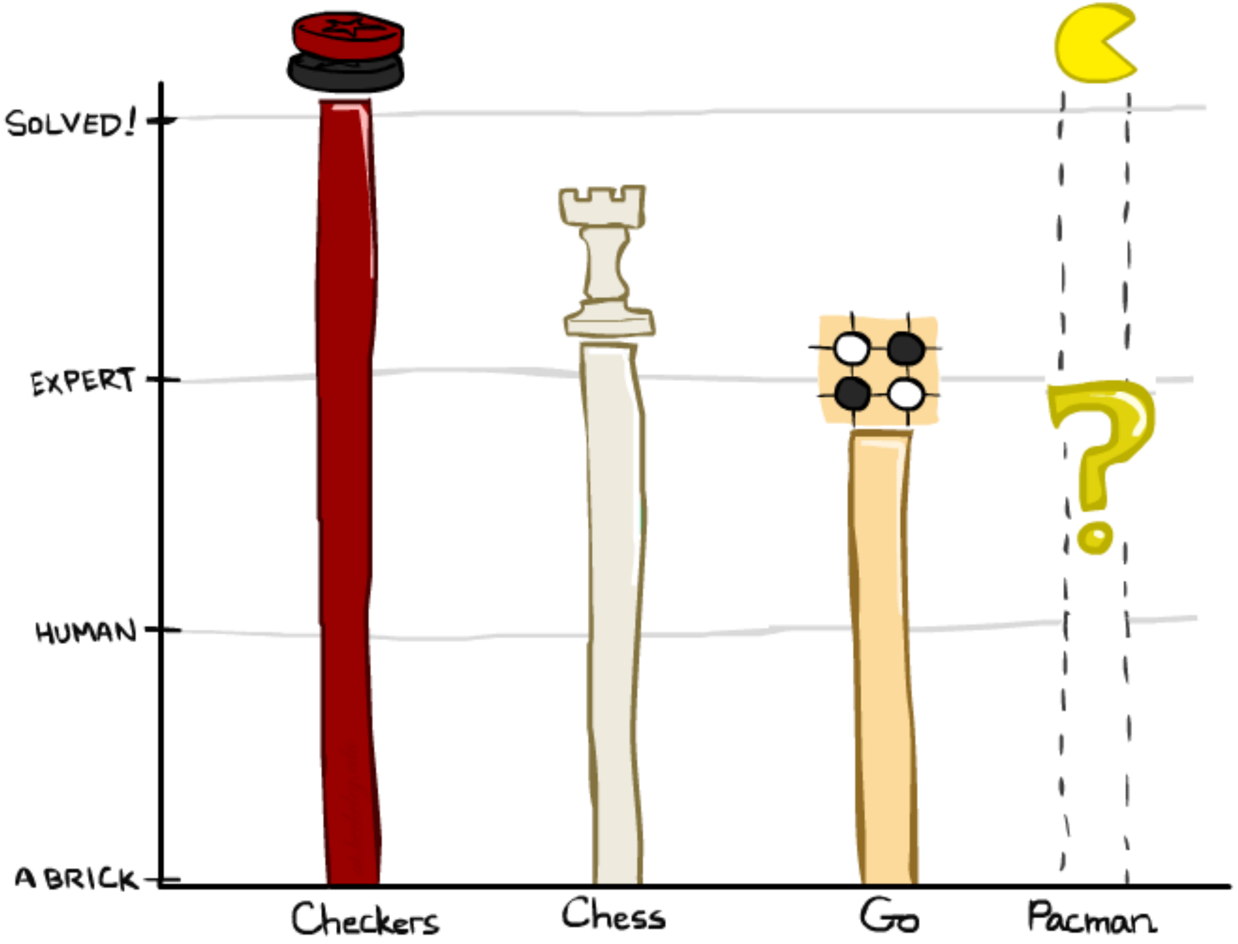
Submit

Never submit passwords through Google Forms.

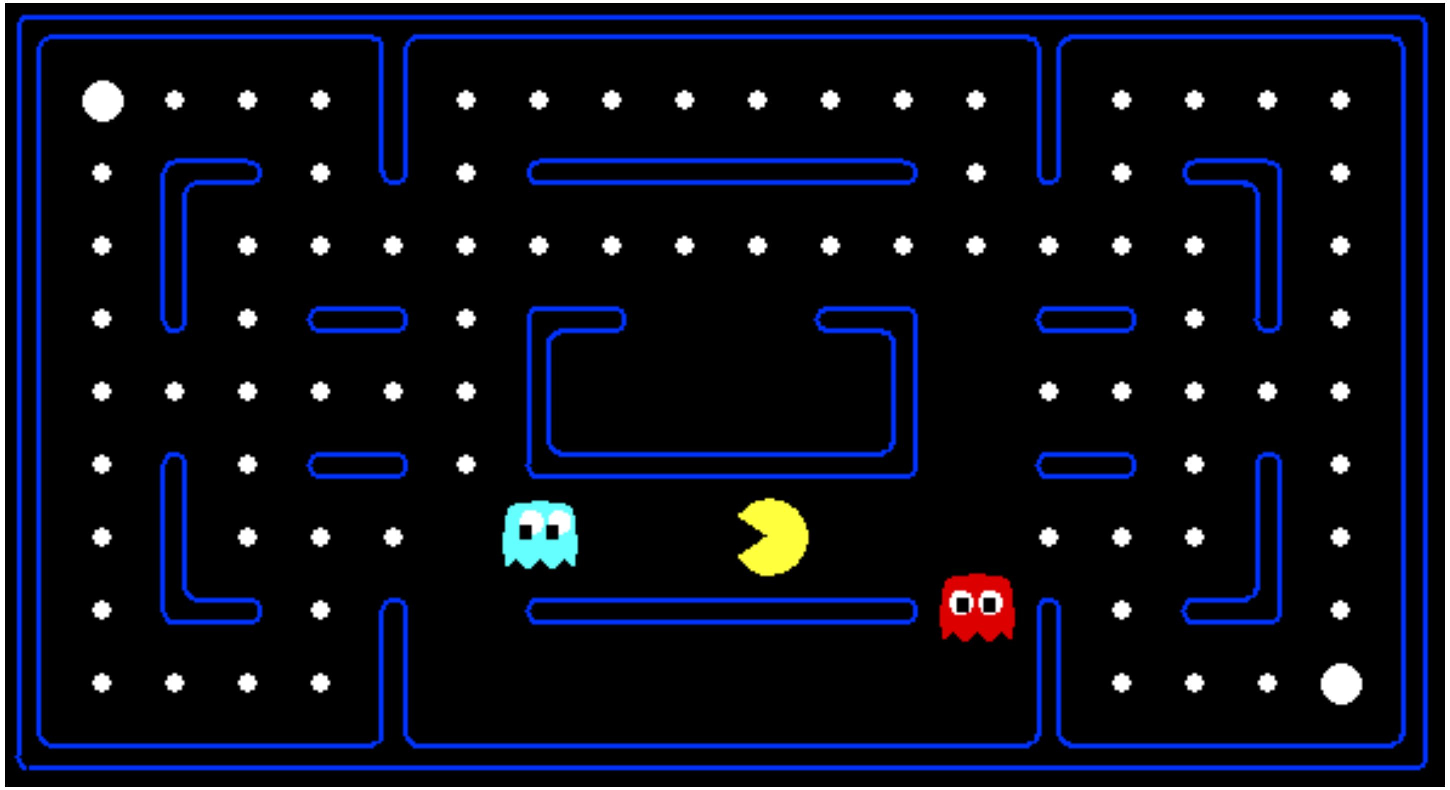
https://forms.gle/jeuK3BNmGW7vRaGW7

Game Playing State-of-the-Art

- **Checkers:**
  - 1950: First computer player.
  - 1959: Samuel's self-taught program.
  - 1994: First computer world champion: Chinook ended 40-year-reign of human champion Marion Tinsley.
  - 2007: Checkers solved! Endgame database of 39 trillion states
- **Chess:**
  - 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCarthy.
  - 1960s onward: gradual improvement
  - 1997: special-purpose chess machine Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second and extended some lines of search up to 40 ply. Current programs running on a PC rate > 3200 (vs 2870 for Magnus Carlsen).
- **Go:**
  - 1968: Zobrist's program plays legal Go, barely (b>200!)
  - 2005-2014: Monte Carlo tree search enables rapid advances: current programs beat strong amateurs, and professionals
  - 2016: Google AI Beat World Go Champion
- **Pacman**



Behavior from Computation



Video of Demo Mystery Pacman

Adversarial Games

Types of Games

- Many different kinds of games!

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

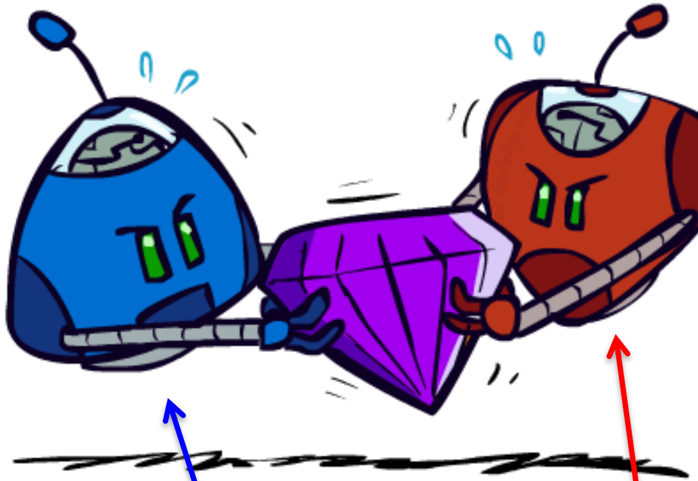
- Want algorithms for calculating a strategy (policy) which recommends a move from each state

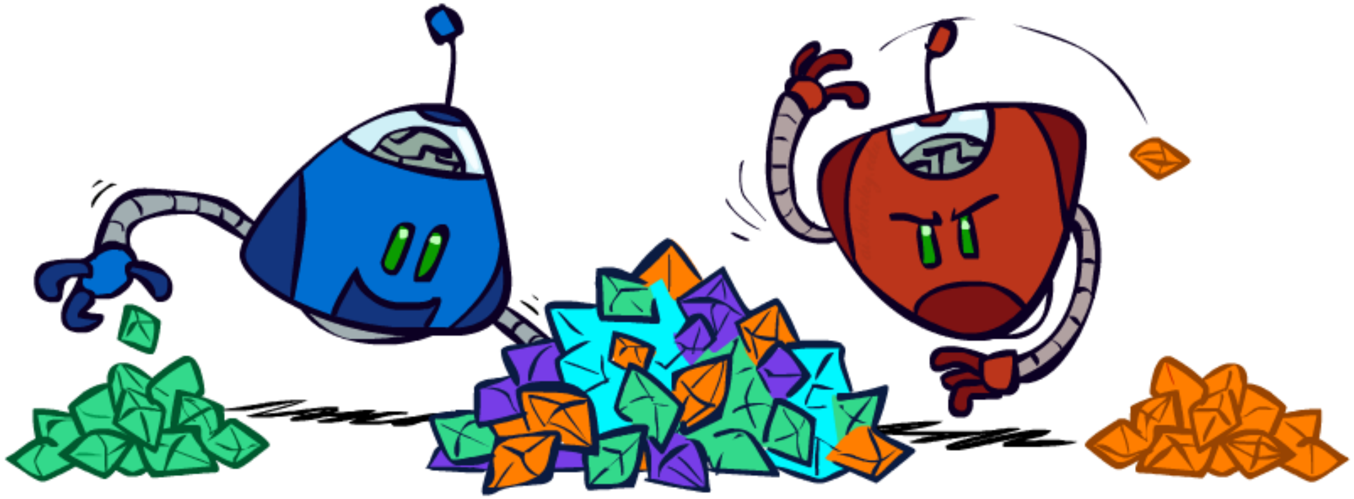
Deterministic Games

- 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: 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



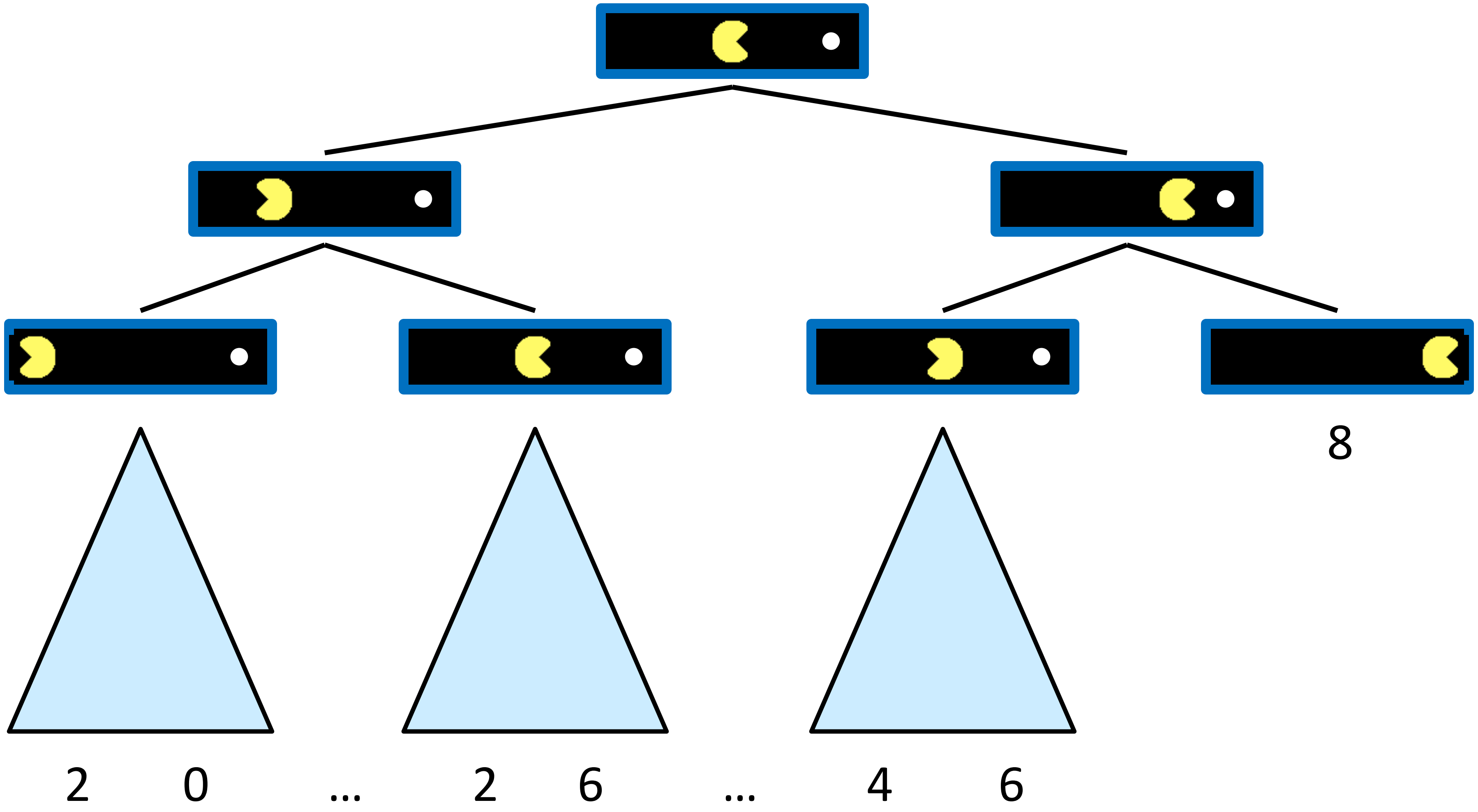


- Zero-Sum Games
  - Agents have ***opposite*** utilities
  - Pure competition:
    - One ***maximizes***, the other ***minimizes***

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

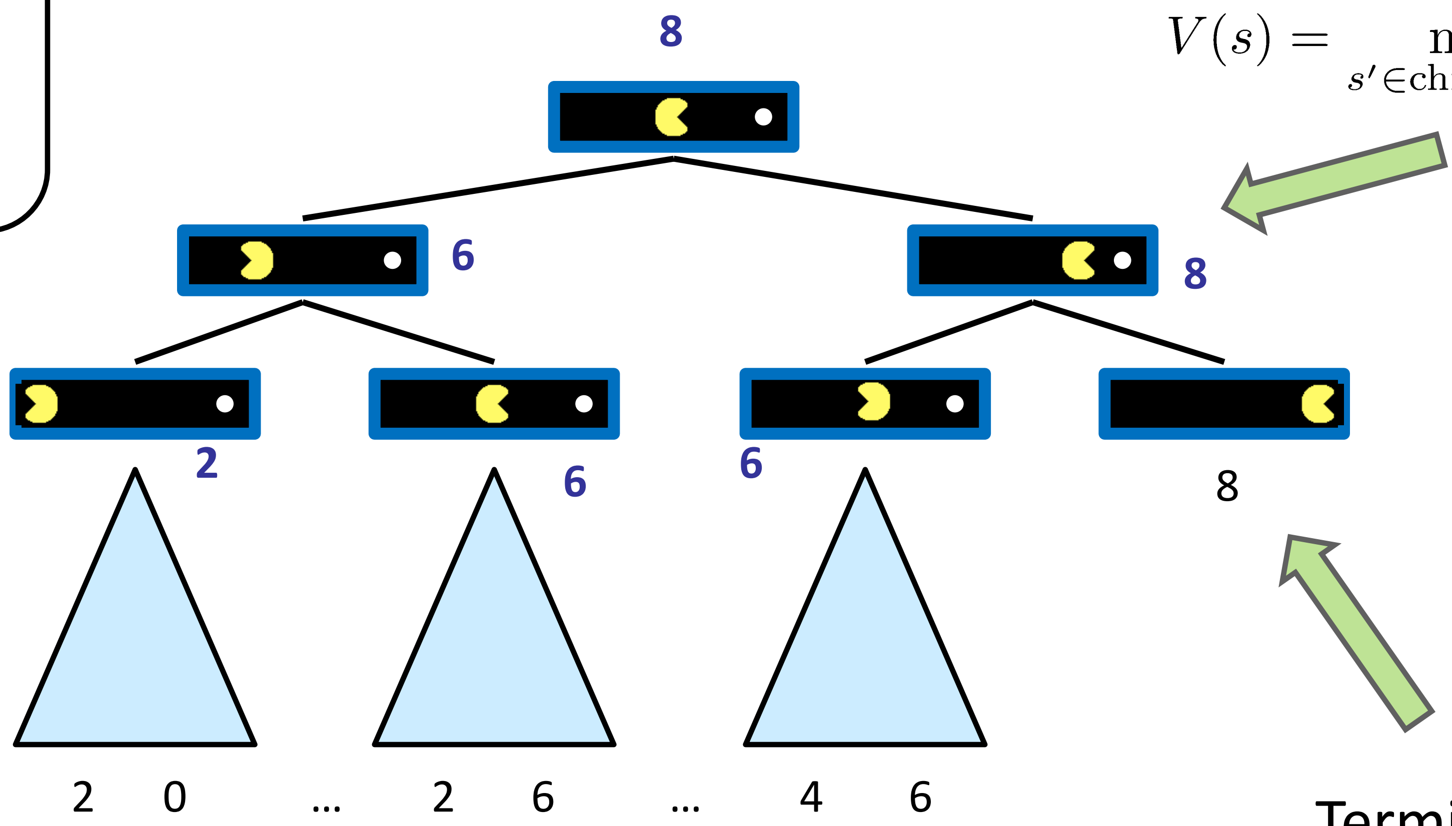
Adversarial Search

Single-Agent Trees



Value of a State

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



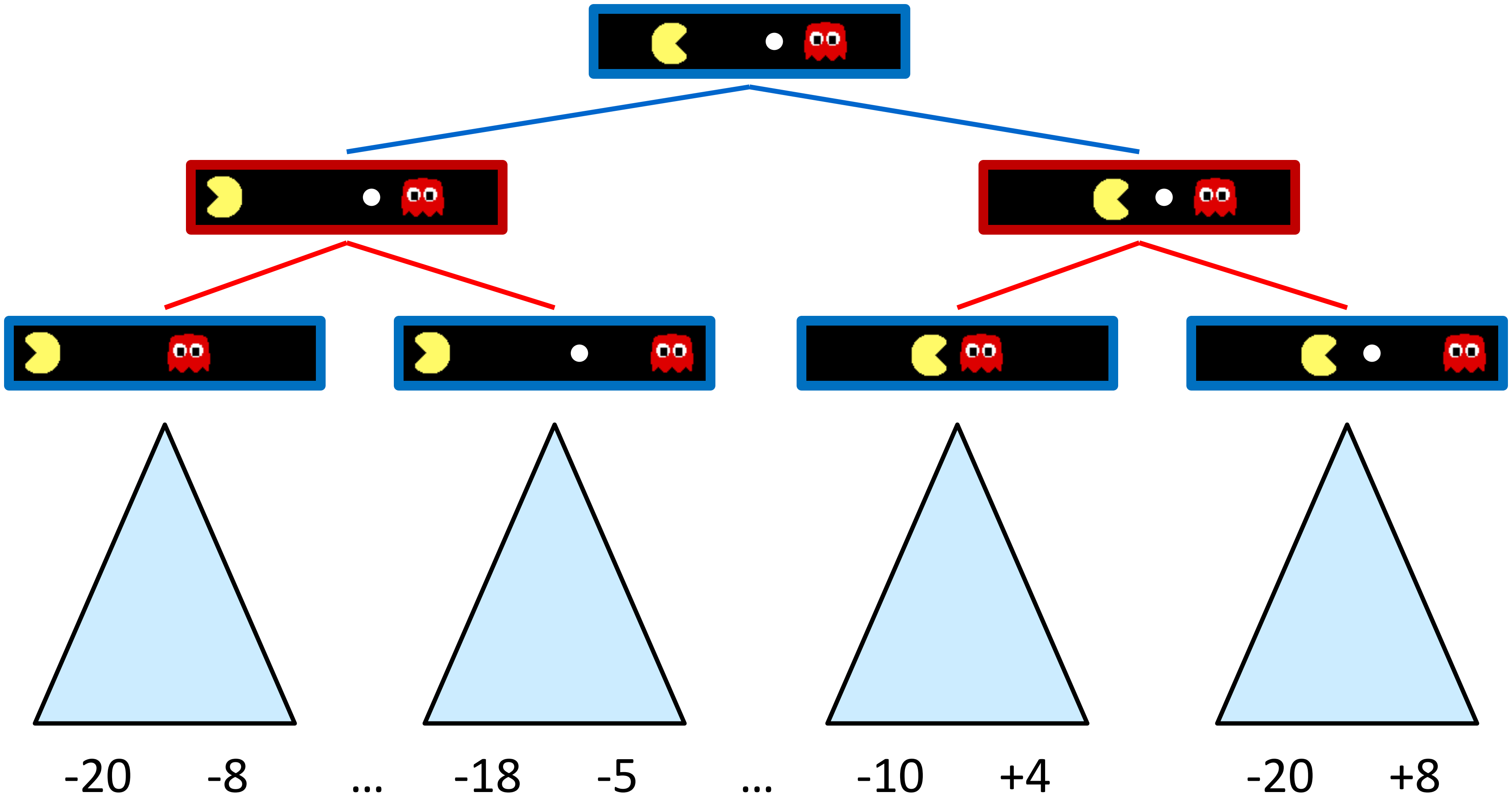
Non-Terminal States:

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

Terminal States:

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

Adversarial Game Trees



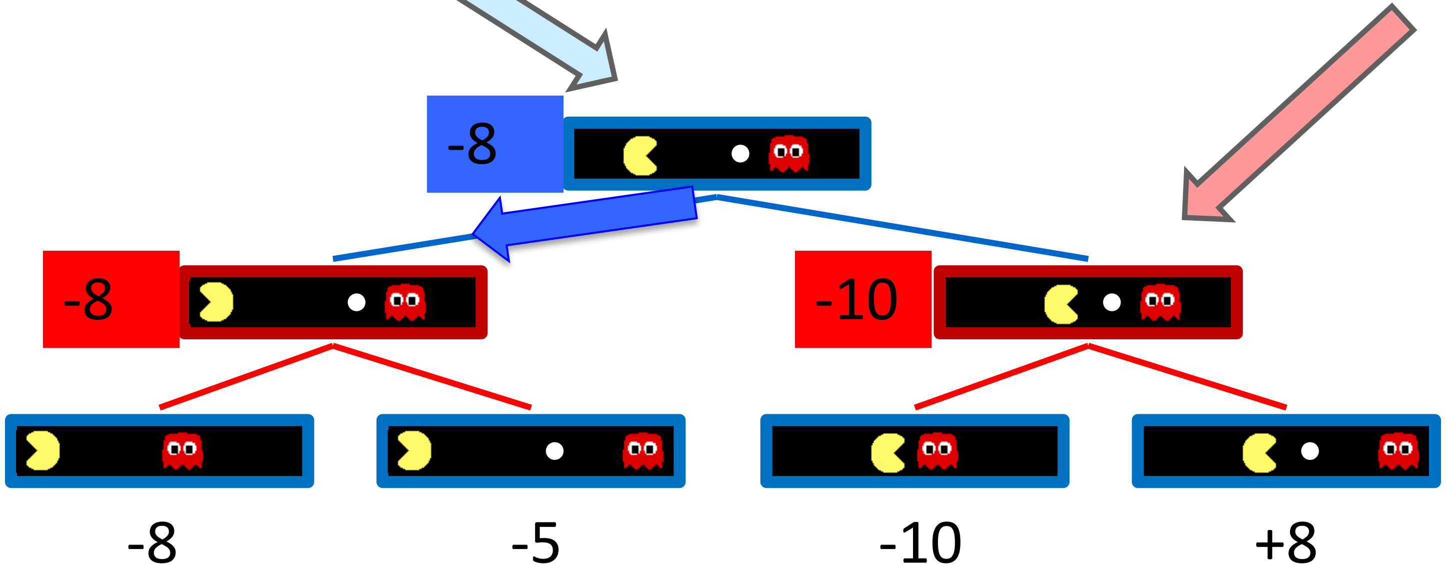
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

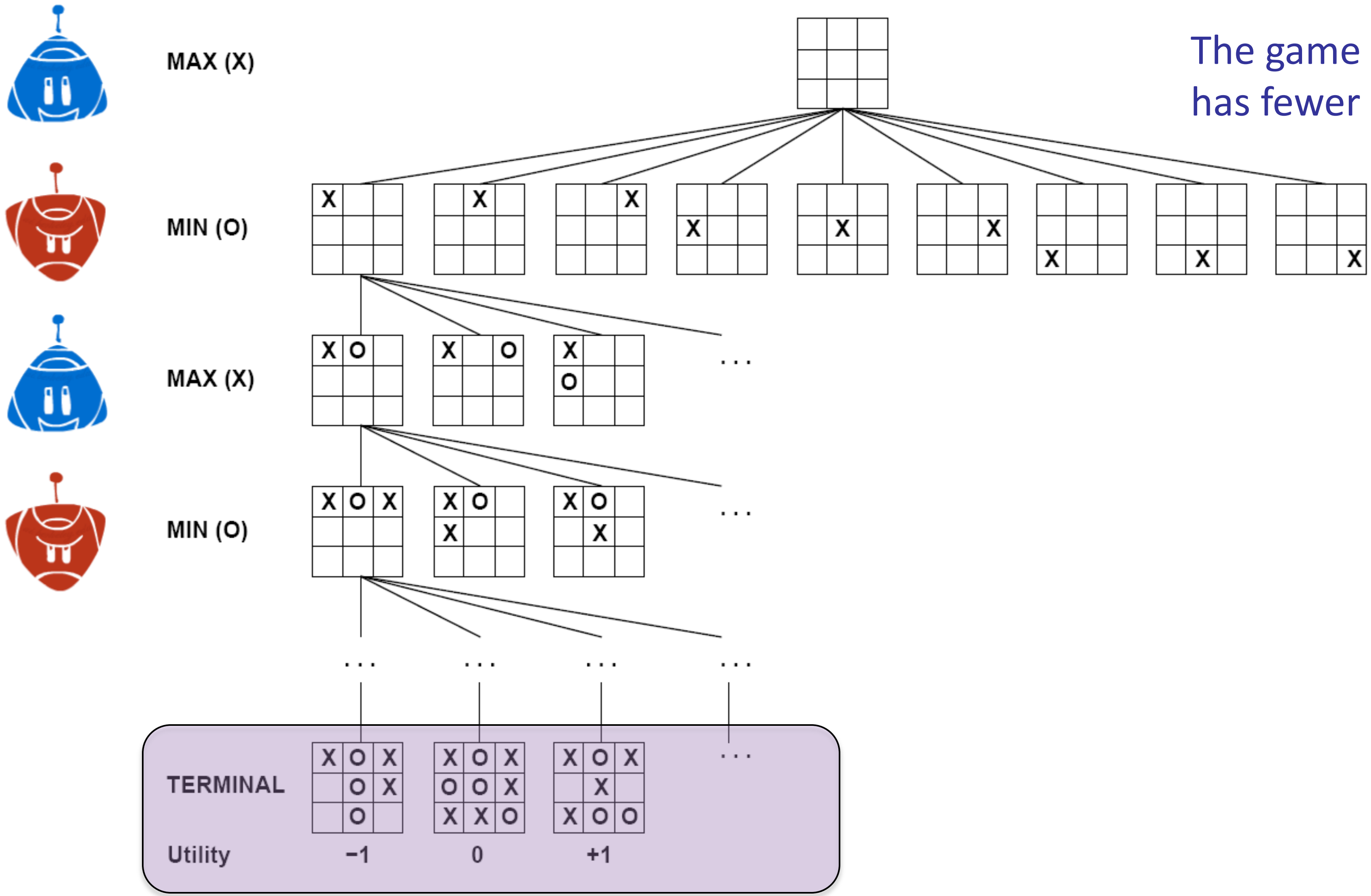
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

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

Tic-Tac-Toe Game Tree



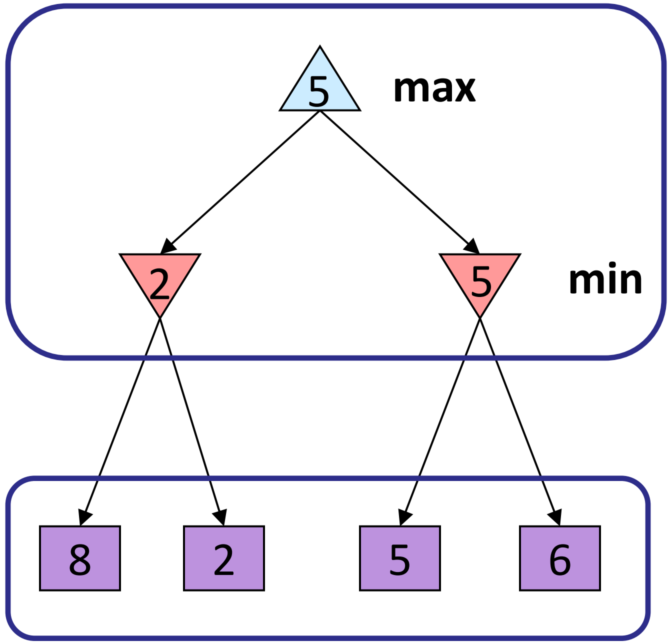
The game tree for tic-tac-toe has fewer than 362,880 leaf nodes.

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 values: computed recursively



Terminal values: part of the game

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
```

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

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

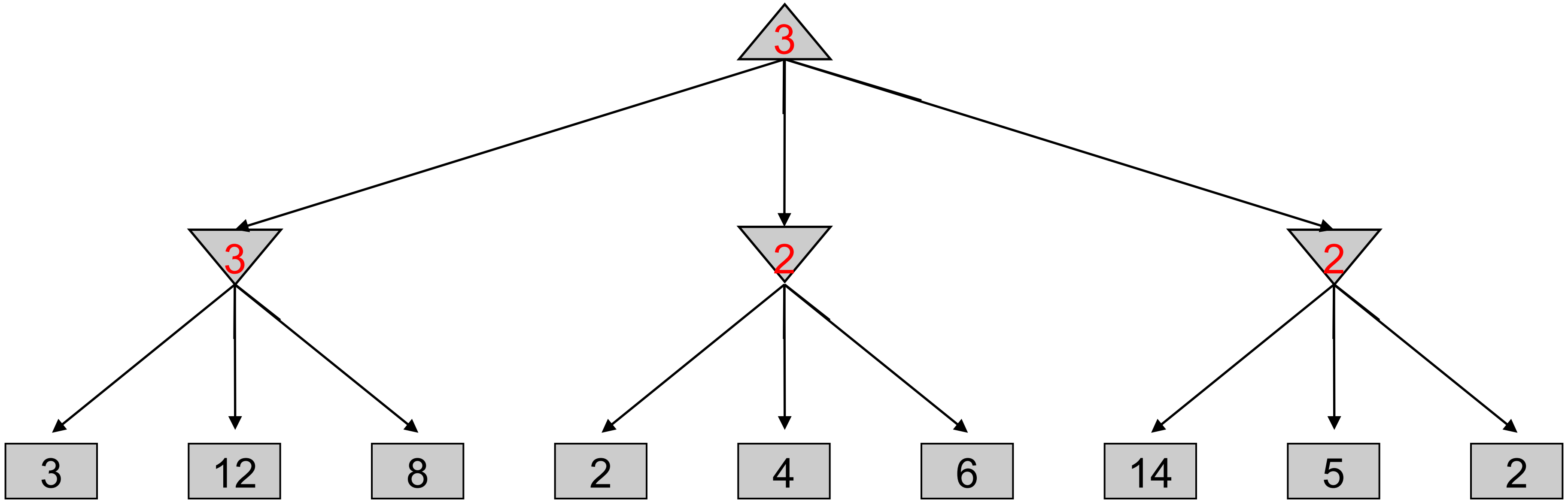
Minimax Implementation (Dispatch)

```
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, min-value(successor))
    return v
```

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

Minimax Example



Minimax Efficiency

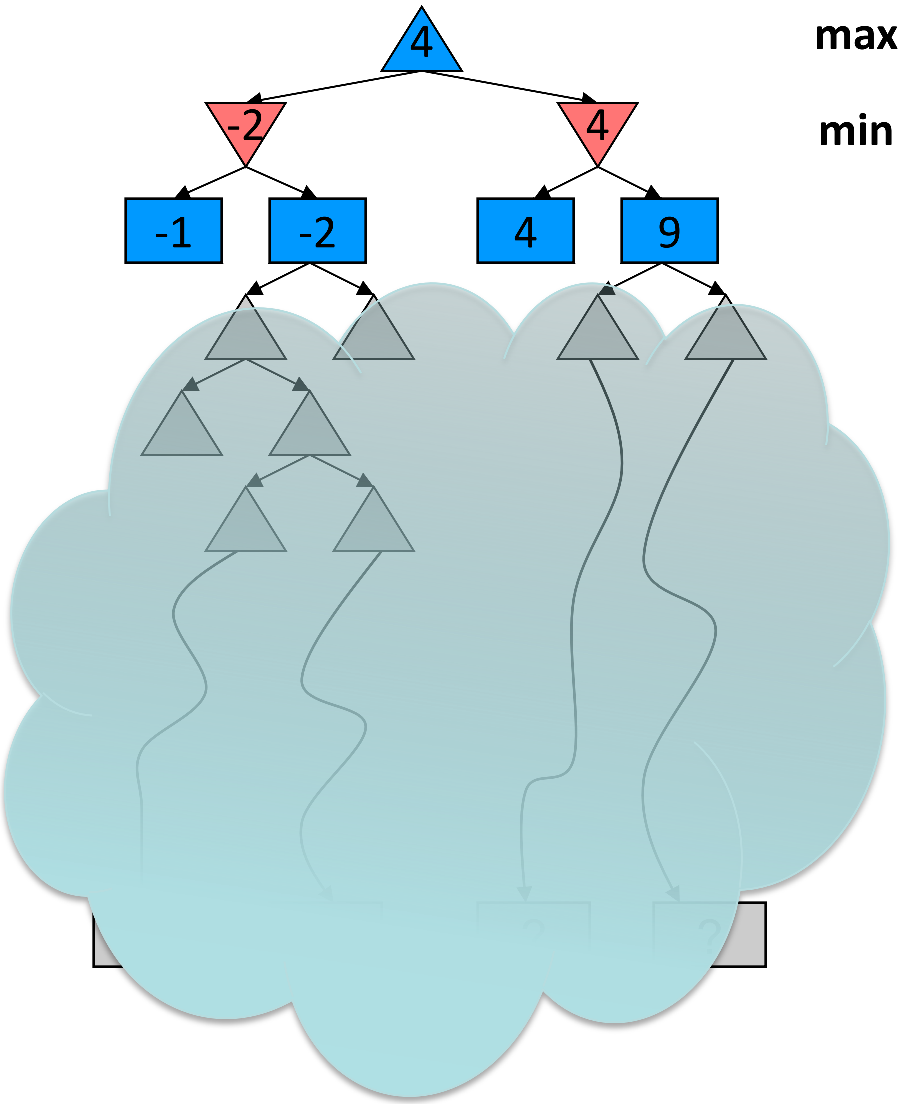
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

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

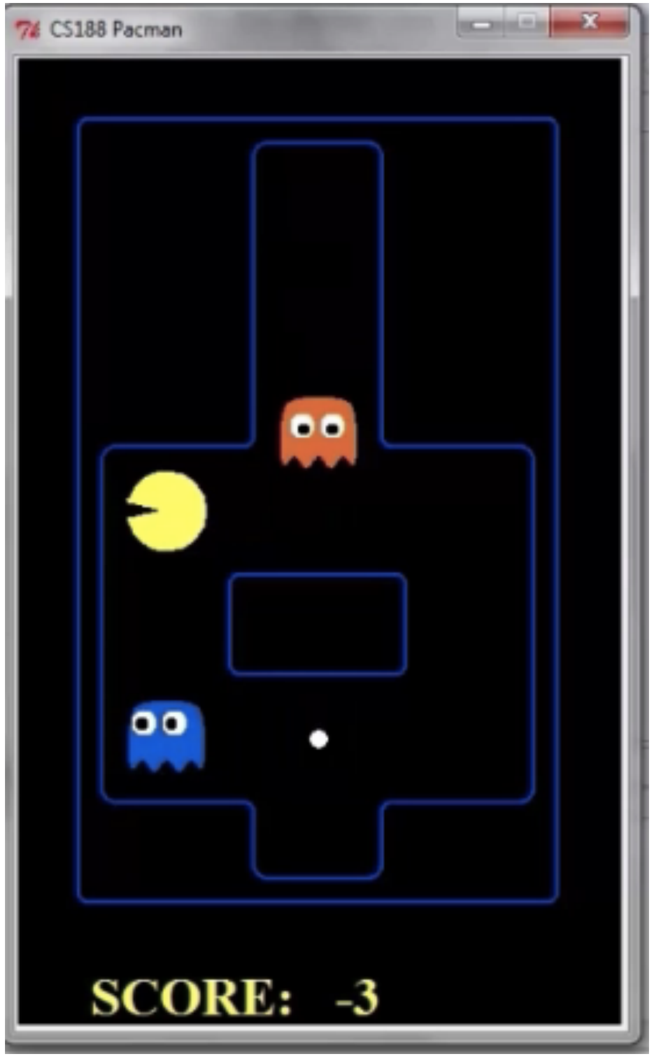
Resource Limits

Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution: Depth Limited Search
  - Search only to a preset **depth limit** or **horizon**
  - Use an **evaluation function** for non-terminal positions
- Guarantee of optimal play is gone
- Deeper search makes a BIG difference
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - For chess, b=~35 so reaches about depth 4 – not so good



Video of Demo Limited Depth (2)

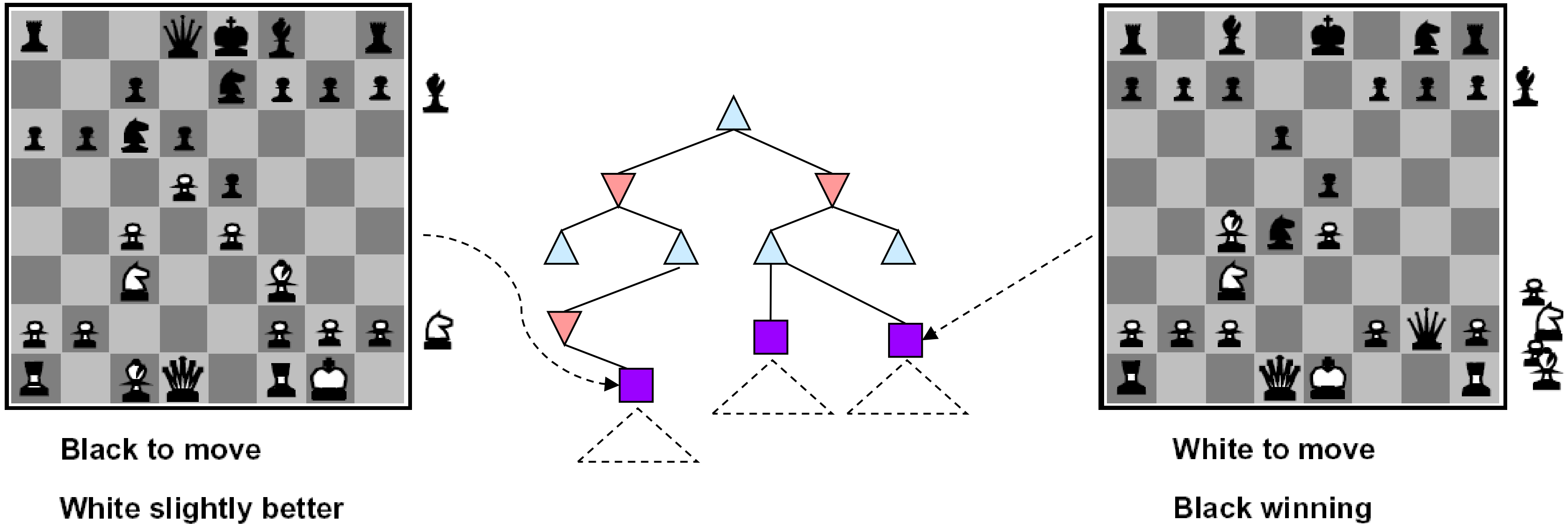


Video of Demo Limited Depth (10)

Evaluation Functions

Evaluation Functions

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

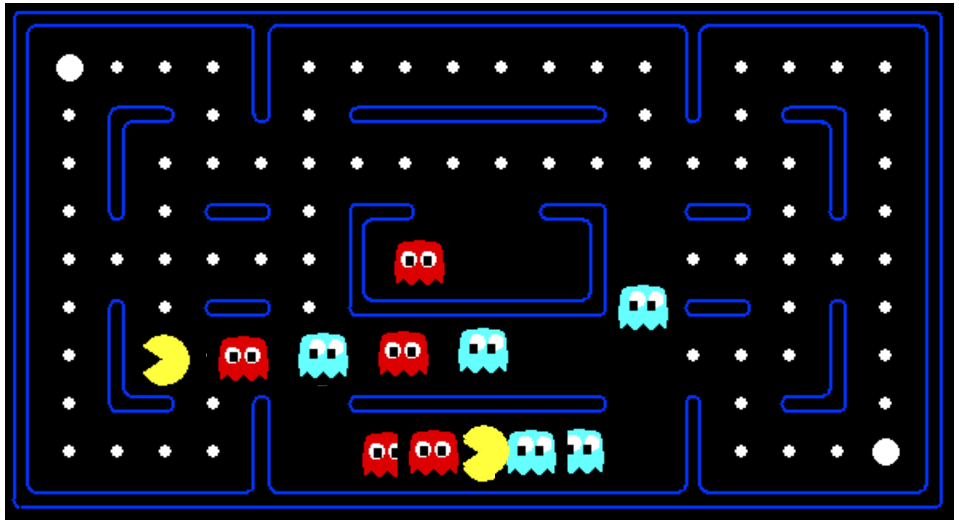


- Ideal evaluation function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

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

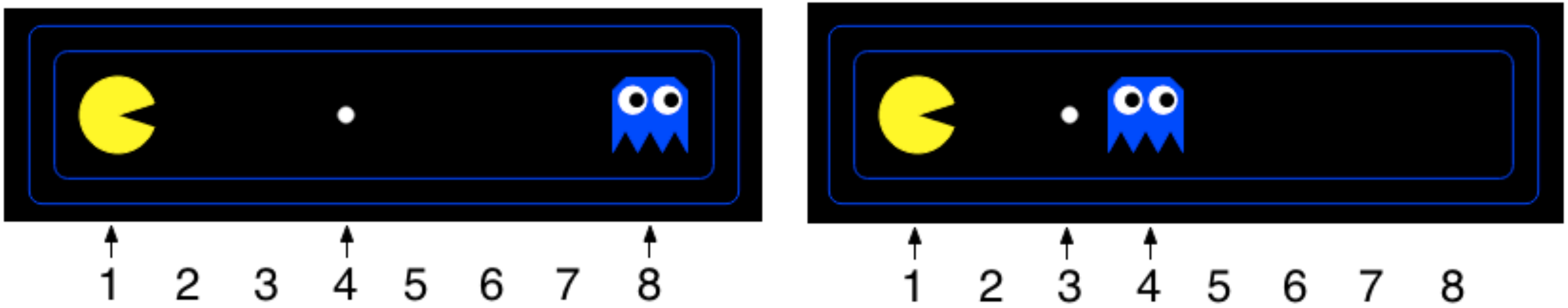
- e.g. $f_1(s)$ = (num white queens – num black queens),
  $f_2(s)$ = (num white pawns – num black pawns),

Evaluation for Pacman



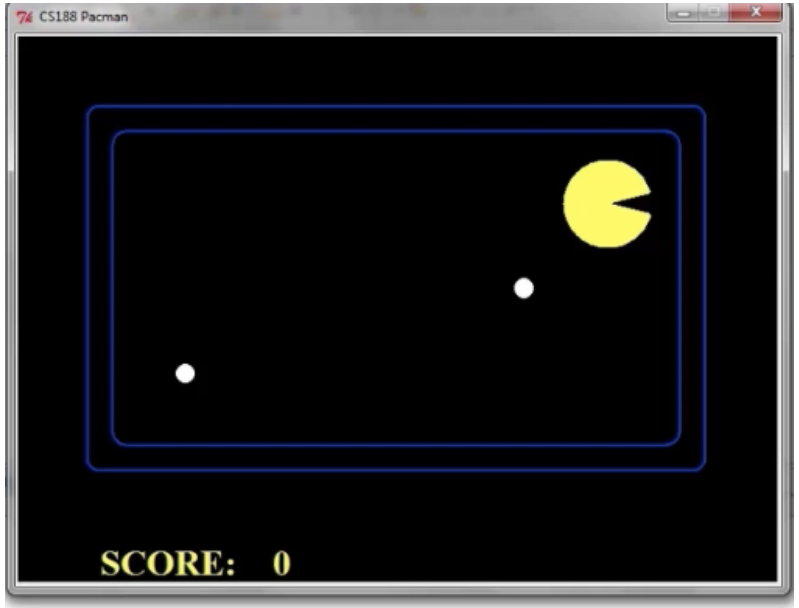
Evaluation Function Quiz

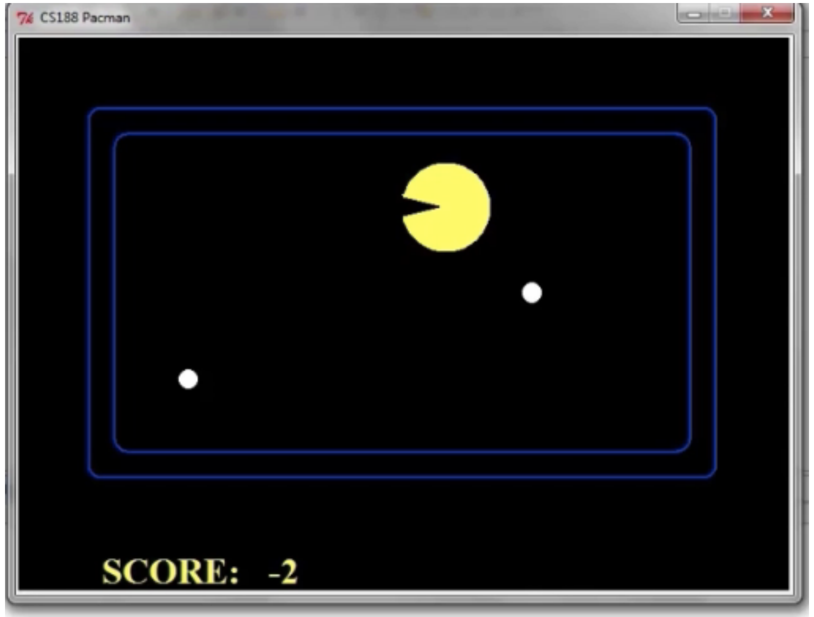
For the two situations shown below, which evaluation functions will give the situation on the left a higher score than the situation on the right?

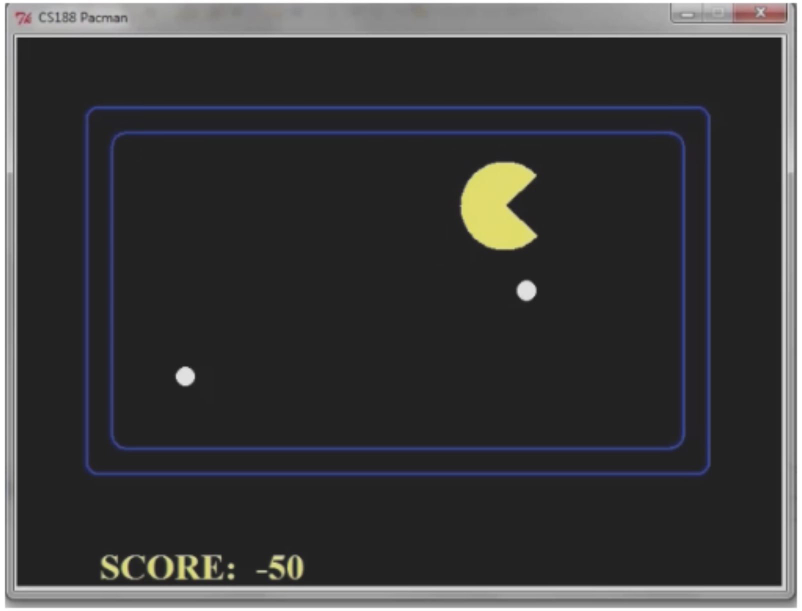


1. 1 / (Pac-Man's distance to the nearest food pellet)
2. Pac-Man's distance to the nearest ghost
3. Pac-Man's distance to the nearest ghost + 1 / (Pac-Man's distance to the nearest food pellet)
4. Pac-Man's distance to the nearest ghost + 1000 / (Pac-Man's distance to the nearest food pellet)

Video of Demo Thrashing (d=3)



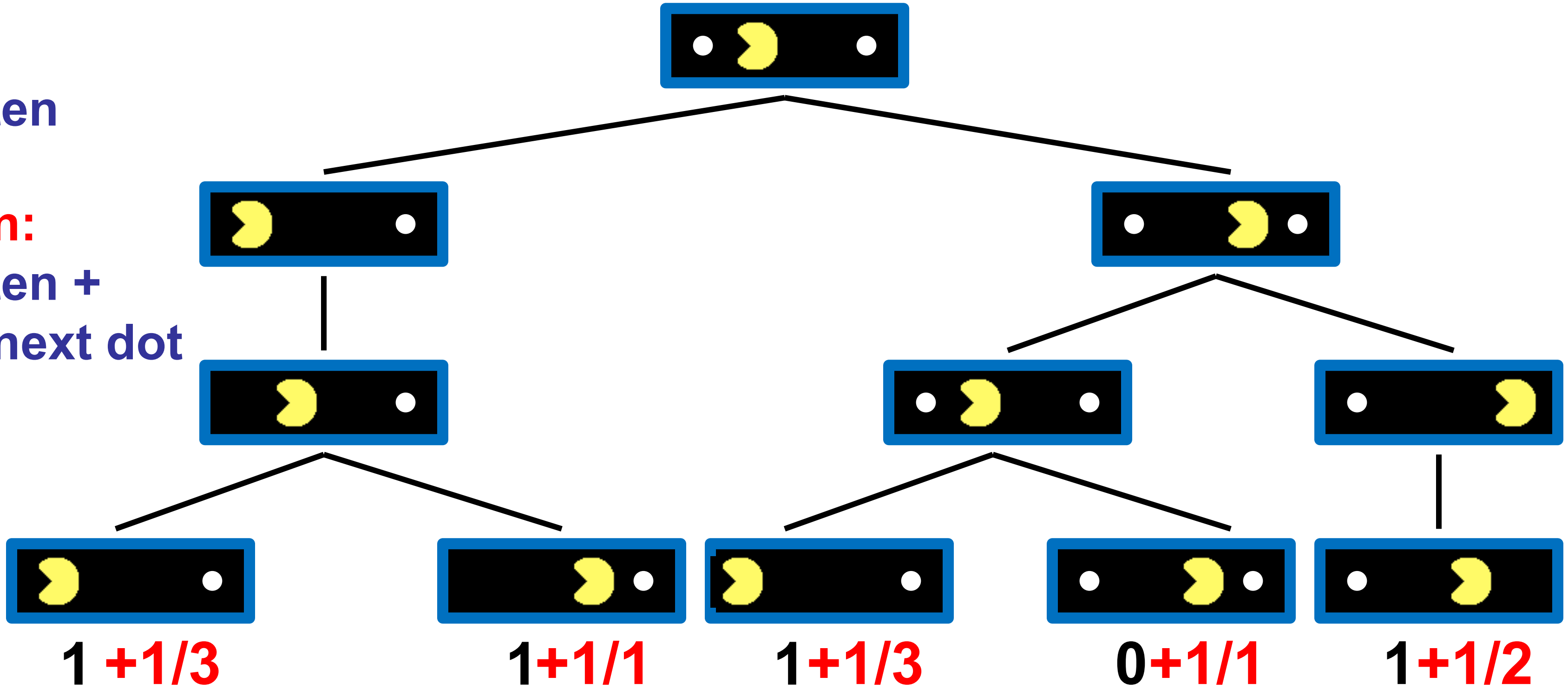




Why Pacman Starves

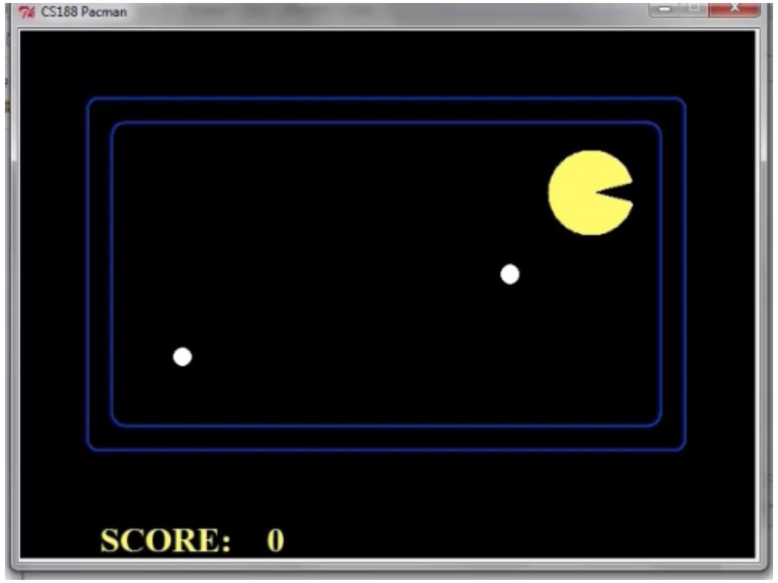
Eval Function:
Number of Dots Eaten

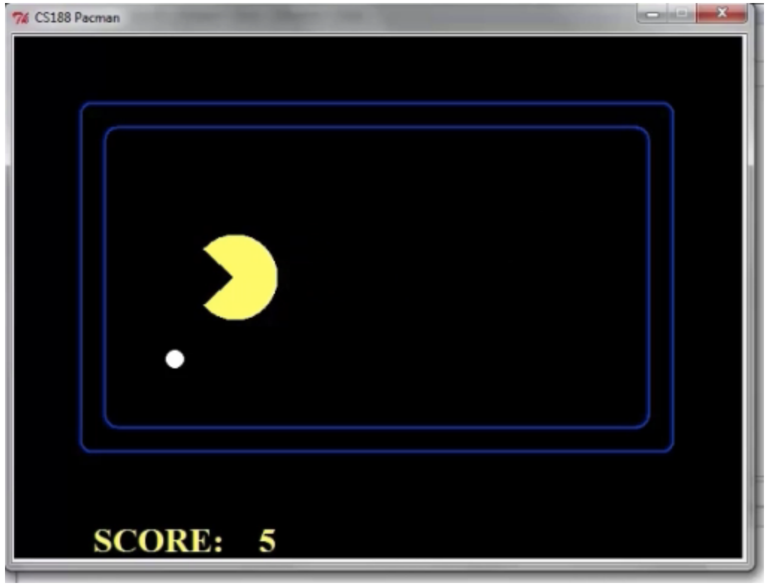
Better Eval Function:
Number of Dots Eaten +
1/Closeness to the next dot

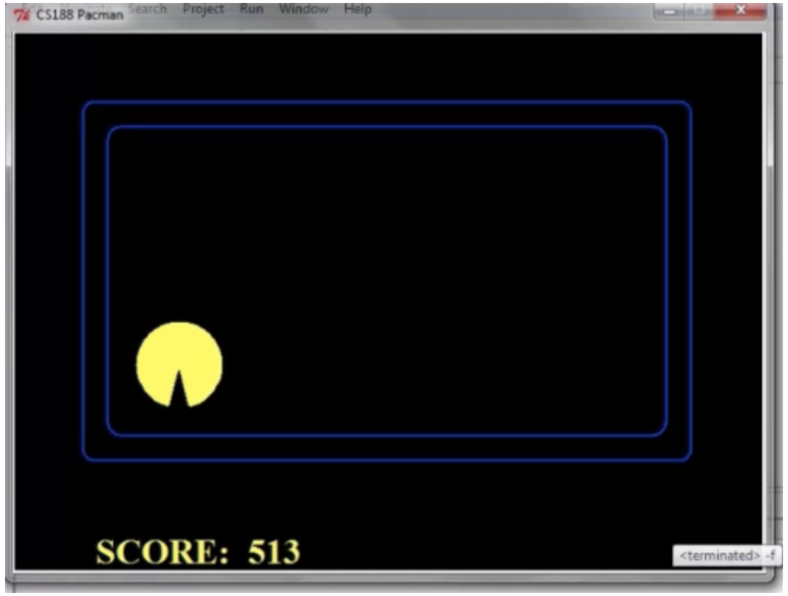


- A danger of replanning agents!
  - He knows his score will go up by eating the dot now
  - He knows his score will go up just as much by eating the dot later
  - Therefore, waiting seems just as good as eating.

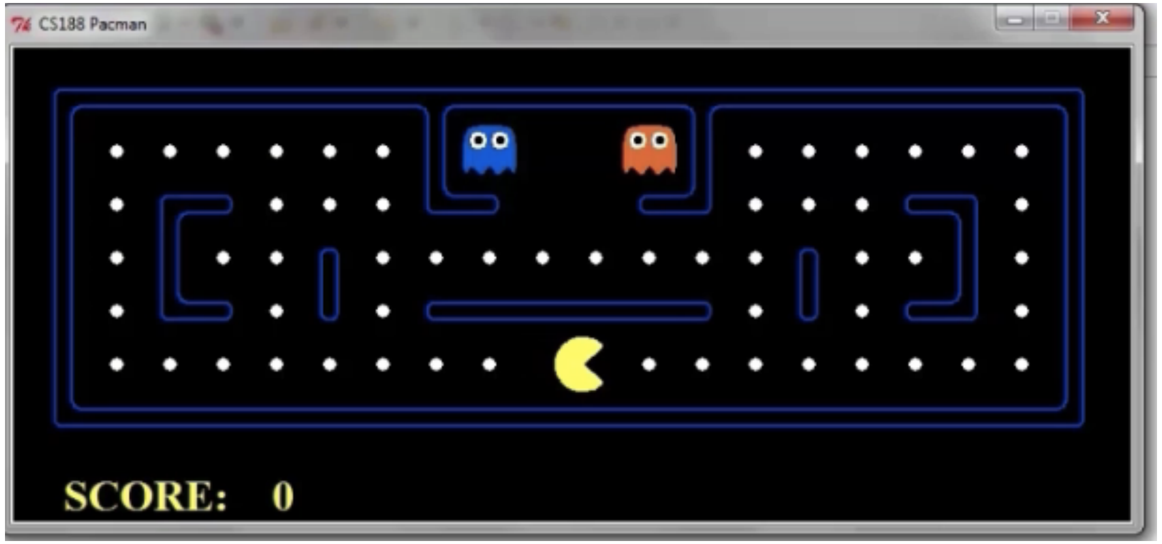
Video of Demo Thrashing -- Fixed (d=3)

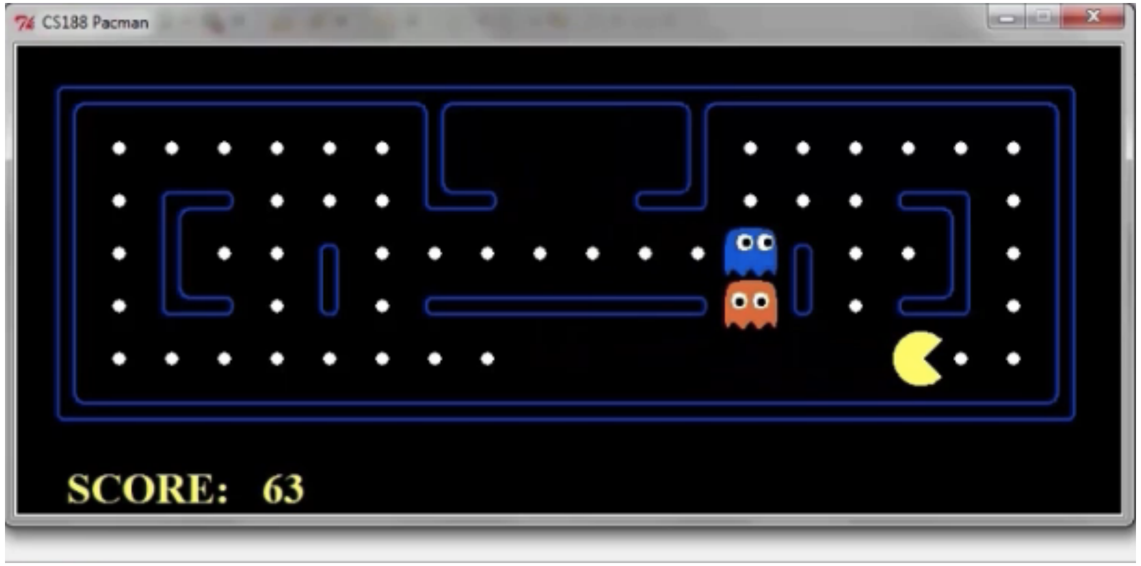


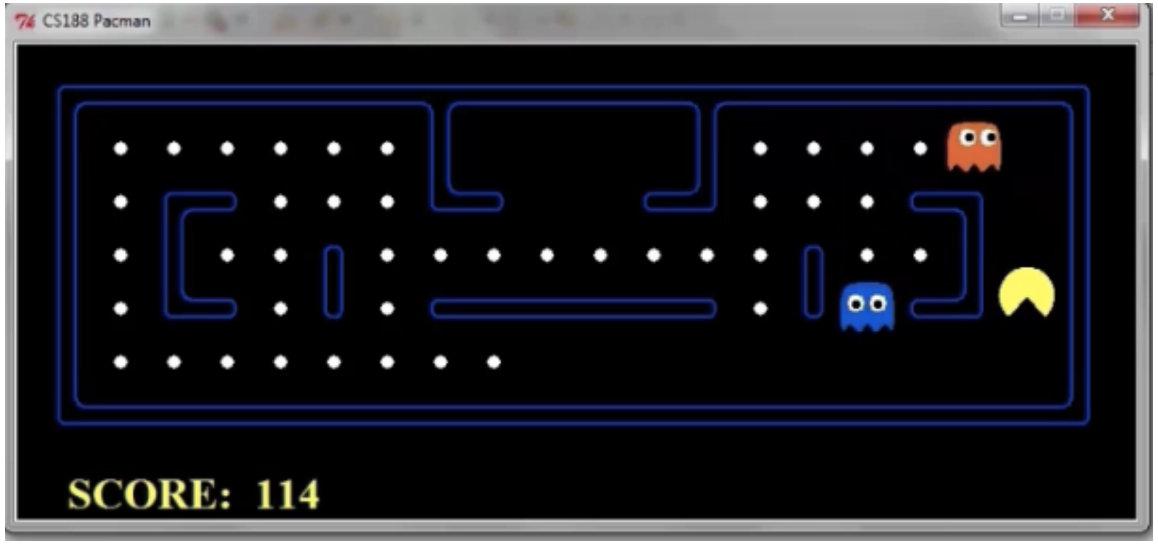


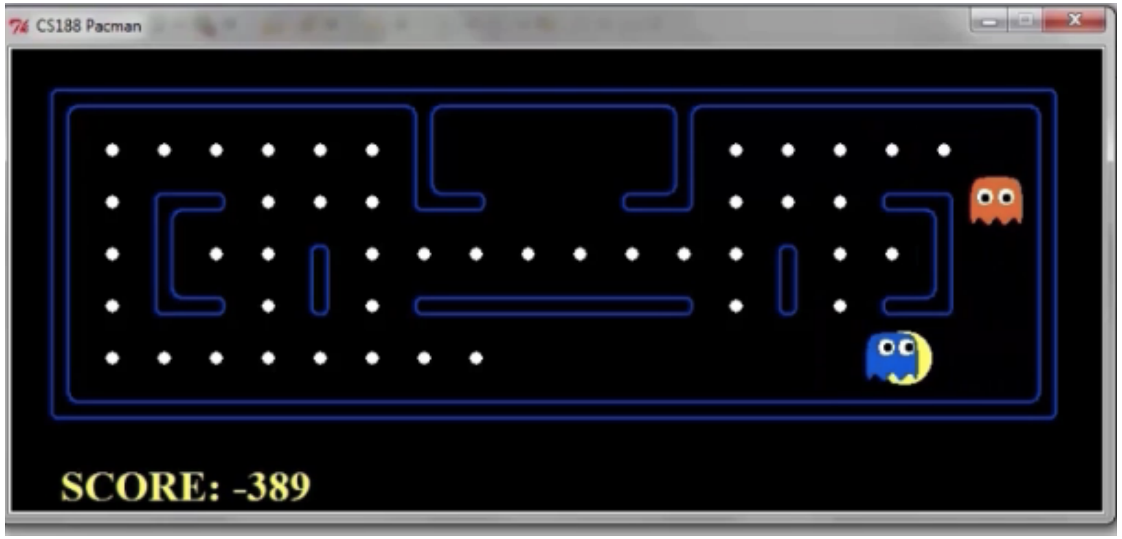


Video of Demo Smart Ghosts (Coordination)



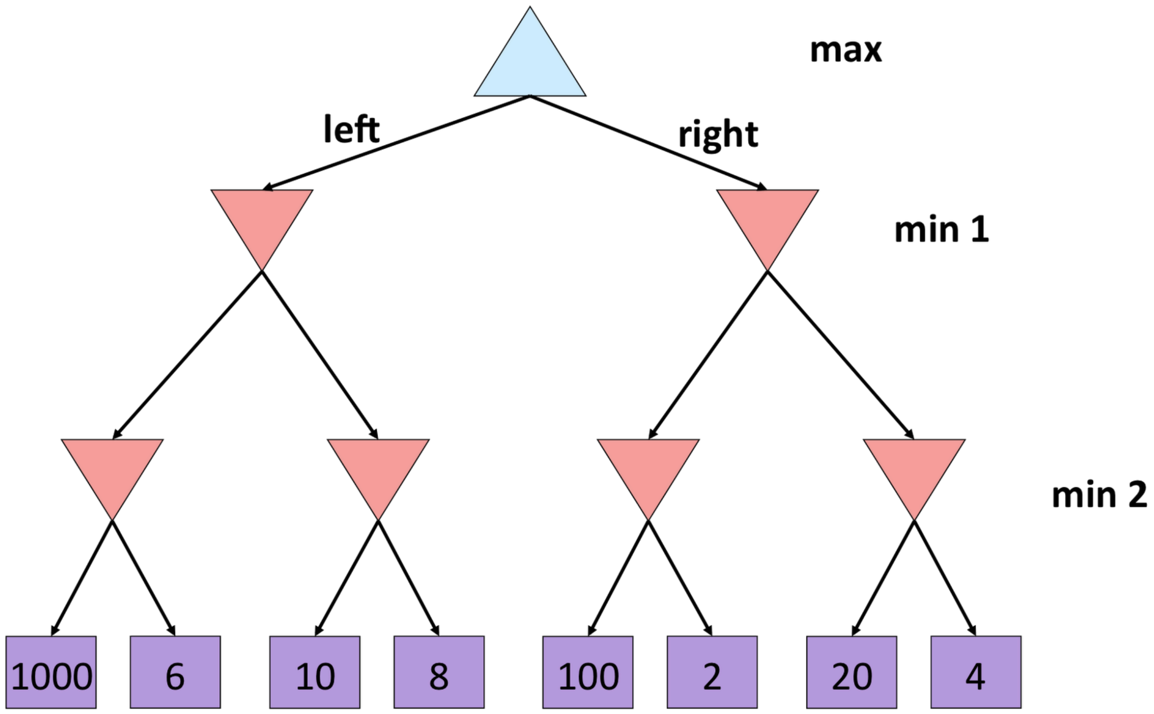






Collaboration Quiz

Below is an example of a game tree with two minimizer players (min 1 and min 2), and one maximizer player.

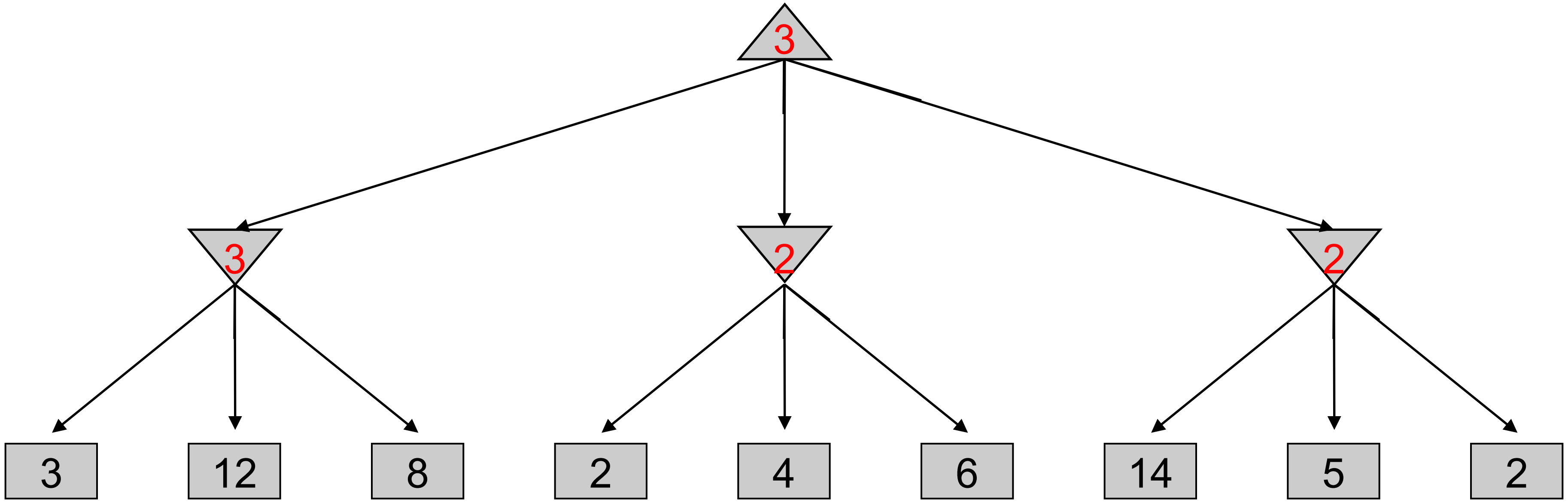


What is the minimax value of this game tree? 6

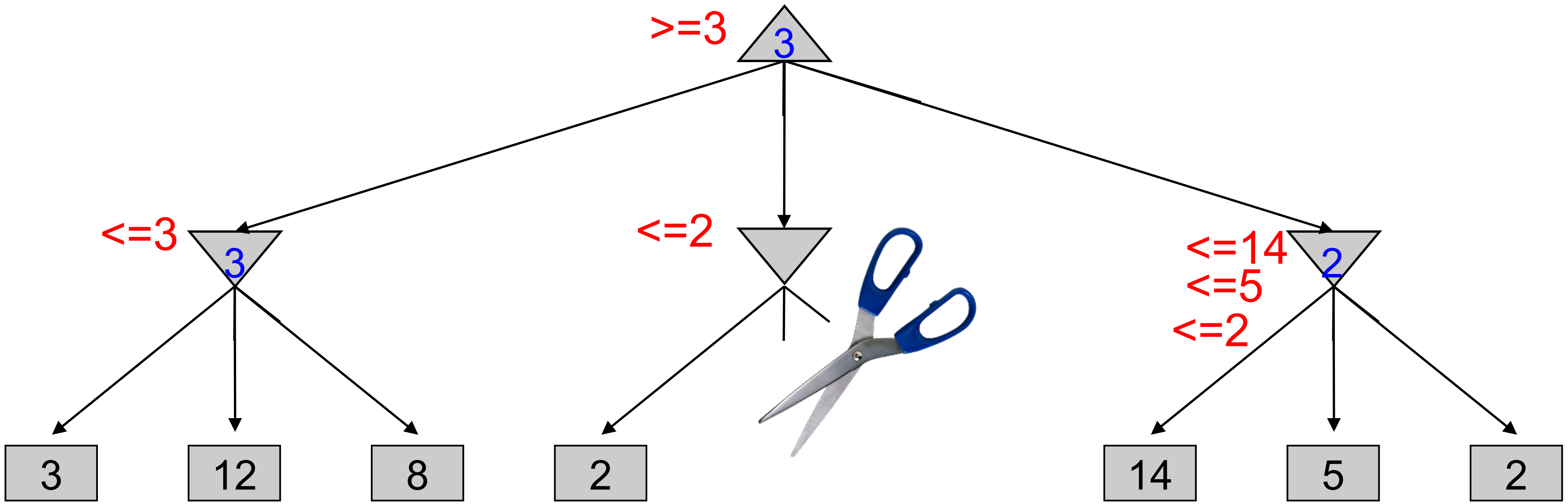
Which action will the maximizer take when playing according to the minimax strategy? Left

Game Tree Pruning

Minimax Example

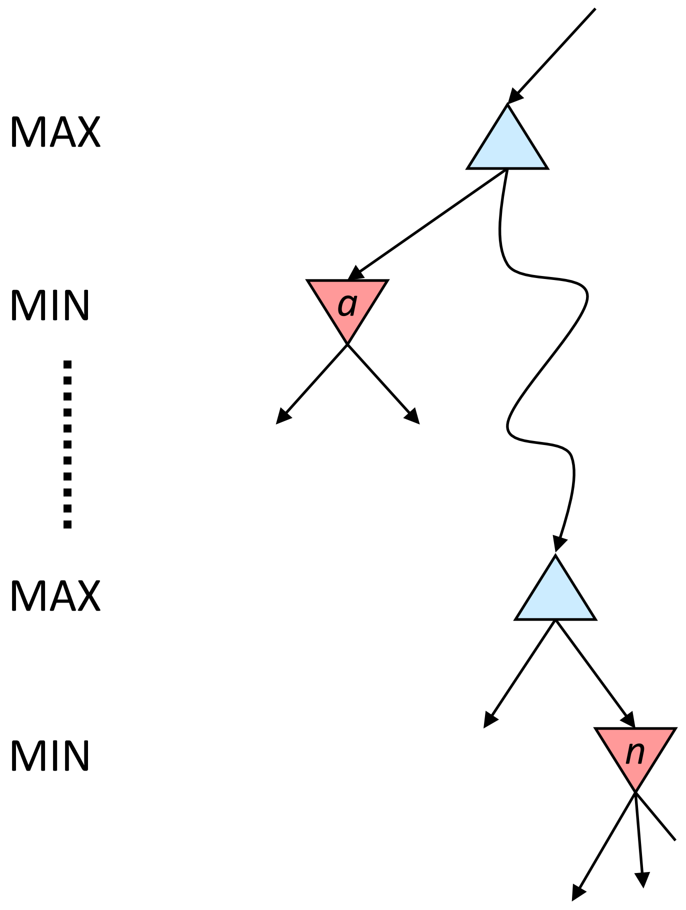


Minimax Pruning



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 children's 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):
    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

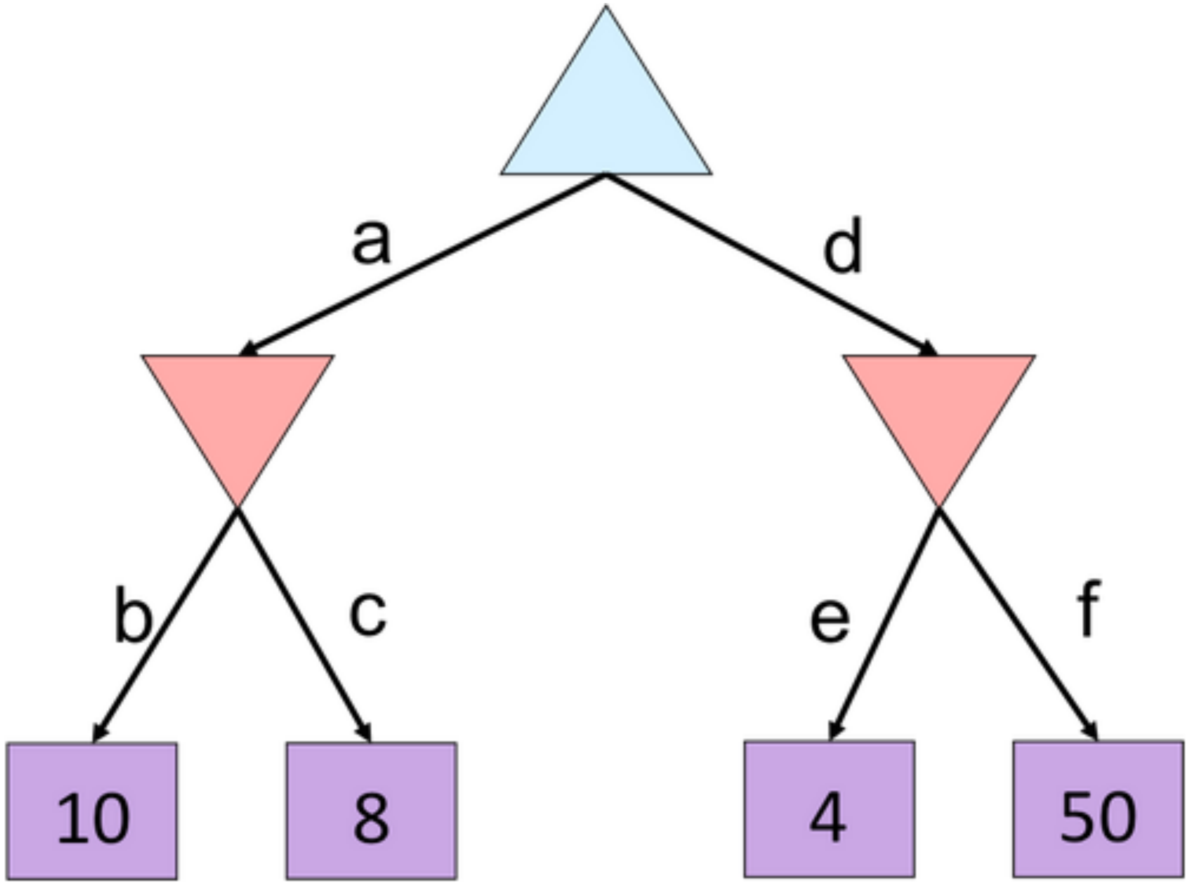
- Theorem: This pruning has no effect on minimax value computed for the root!

- Good child ordering improves effectiveness of pruning

- With “perfect ordering”:
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - 1M nodes/move => depth=8, respectable

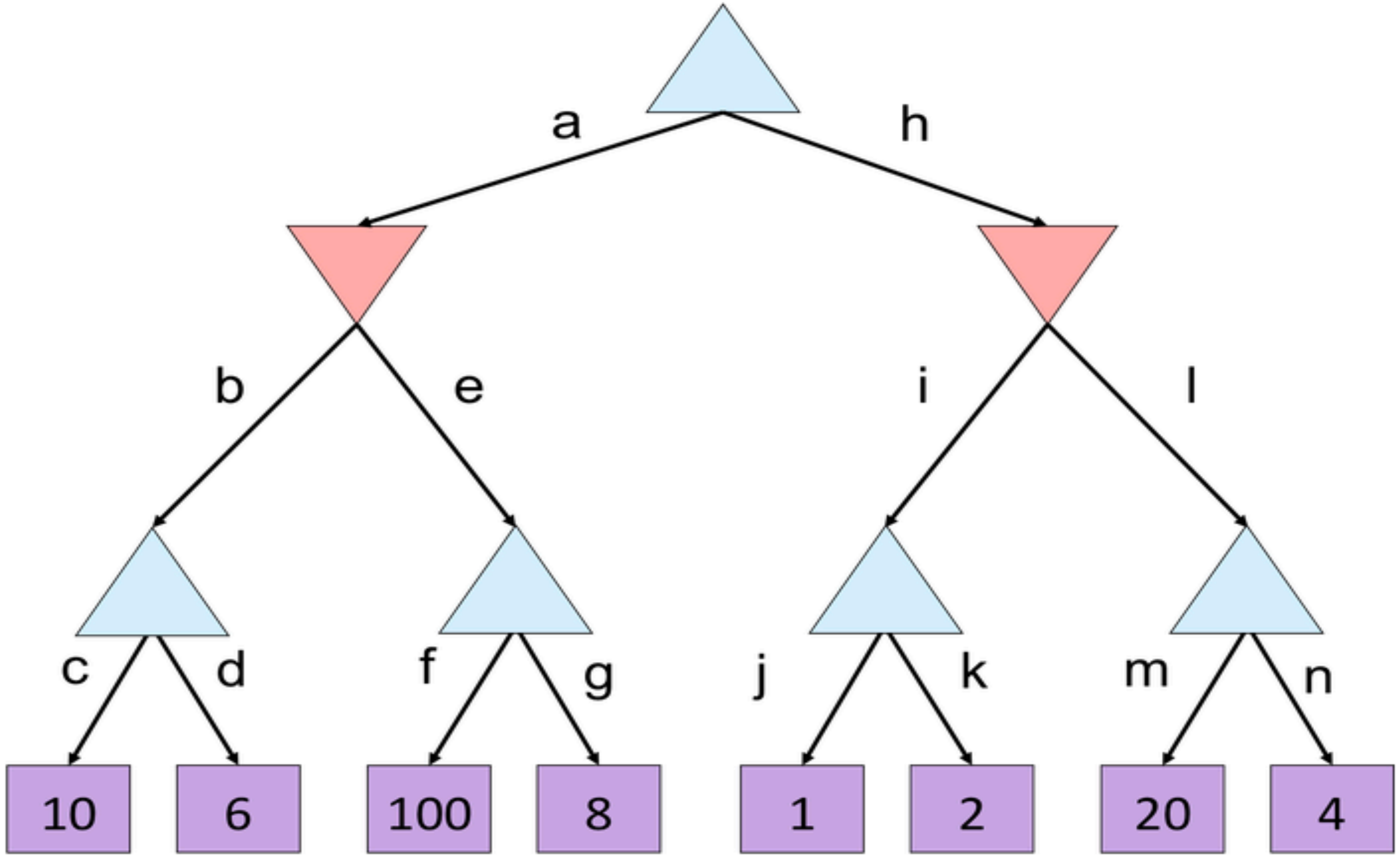
Alpha-Beta Quiz

For the game tree shown below, which branches will be pruned by alpha-beta pruning? f



Alpha-Beta Quiz 2

For the game tree shown below, which branches will be pruned by alpha-beta pruning? g and l



Reading

- Chapter 5.1-5.3, 5.5 in the AIMA textbook