Lecture 5
Adversarial Search

Thanapon Noraset
Faculty of ICT, Mahidol University

Adapted from AIMA by Stuart Russell and Peter Norvig, and UC Berkeley CS188 by Dan Klein and Pieter Abbeel (ai.berkeley.edu)

Agenda

- Competitive Environments
  - Types of Games
  - Zero-sum games
  - Behavior (Policy)
- Adversarial Search
  - Formulation
  - Optimal Decision
  - Minimax algorithms
- Improving Efficiency
  - Alpha-Beta Pruning
  - Cutoff and Evaluation function

Competitive Environments

Tic-Tac-Toe

What is the best next move?

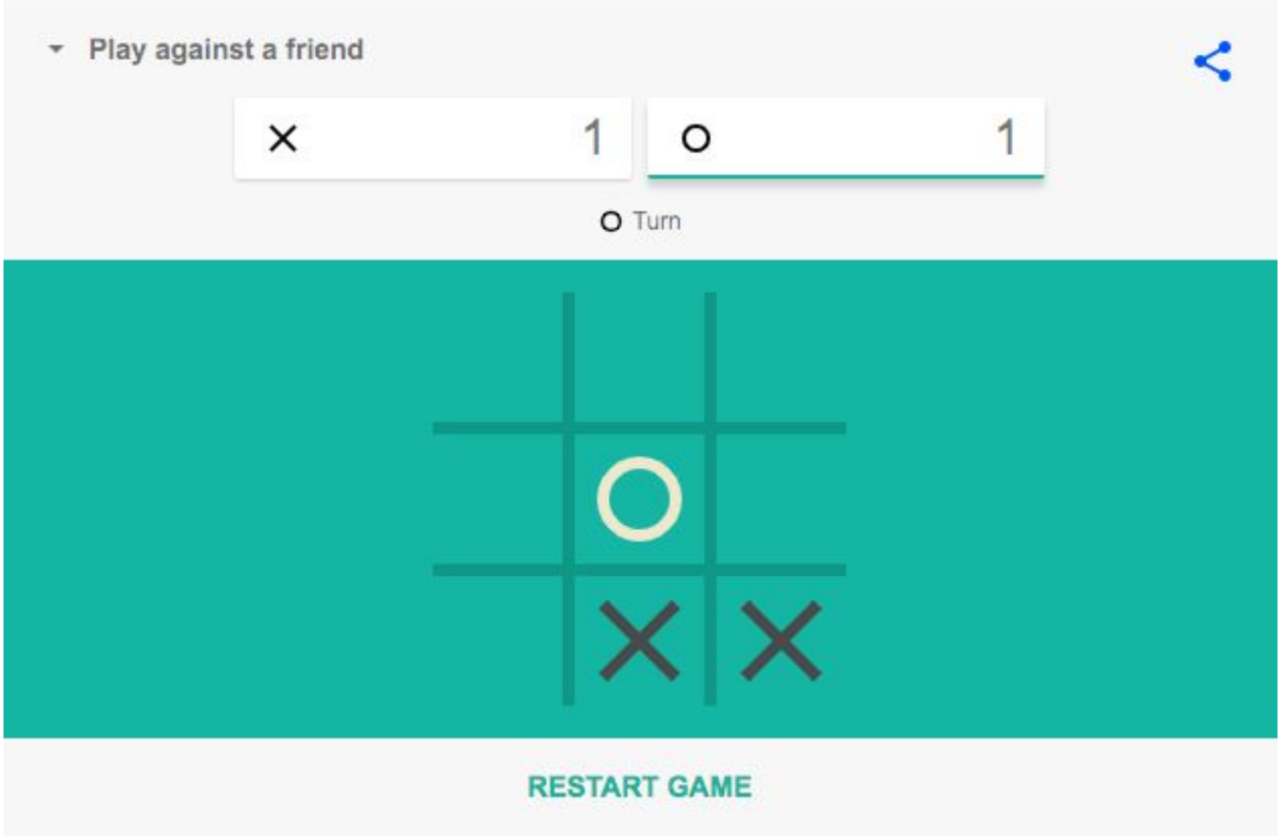


Image from Google Search Engine

Multiagent Environments

- An agent needs to consider the actions of other agents which can be unpredictable.

- This makes it hard to find an optimal sequence of actions.

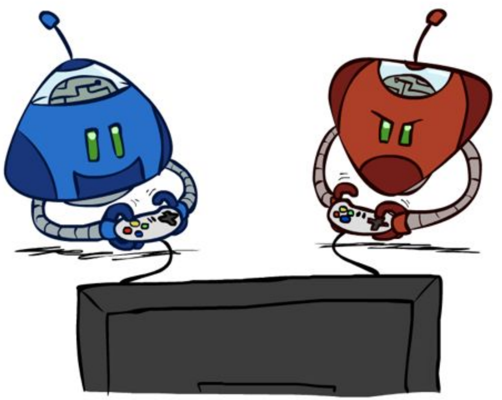
- In this class, we are going to find a simple optimal move.

Competitive Environments

- A type of multiagent environments where agents' goals are conflict.

- Adversarial games are a simple form of competitive environments
  - a few agents and a simple set of rules

- Game-playing problems are quite challenging.
  - Playground for AI researchers



Aspects of adversarial games

- Deterministic or Stochastic ?
  - Is the result of an action certainly known?
- Number of players ?
  - 2, 3, or more? Are they in teams?
- Perfect information (fully observable) ?
  - Can agent observe the full state?
- Zero-sum games ?
  - Total payoff to all player is the same

Zero-sum games of Perfect Information

General Games

- Independent “scores” on outcomes
- Cooperation, indifference, competitions

Zero-Sum Games

- Opposite “scores” on outcomes
- A single value where one maximizes and the other minimizes

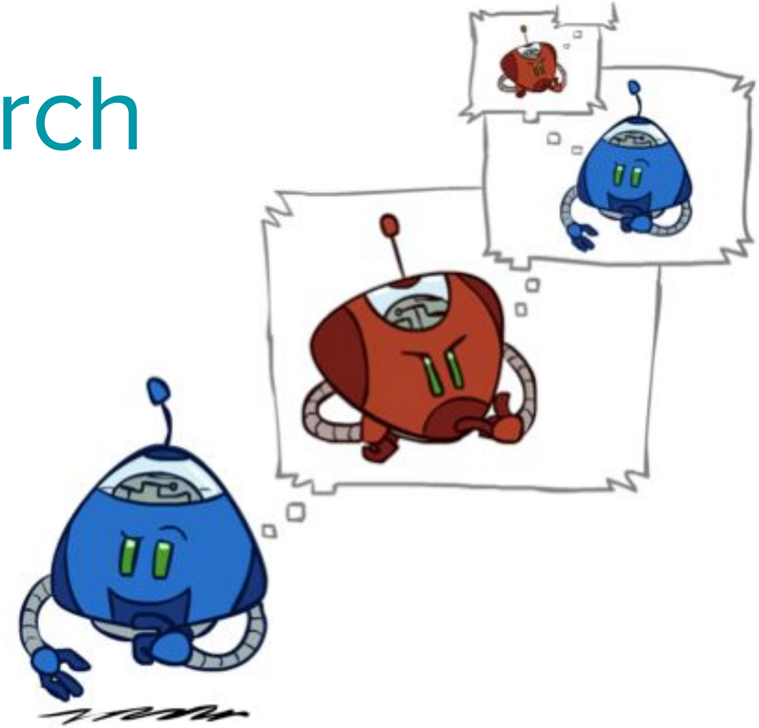
Agent Behavior: Policy

- A game-playing agent has to decide on what to do for the current state of them.

- A solution of a player is a policy function mapping: State ➔ Action

- Searching from the best move:
    - Think ahead guessing [all] other agents' actions to the ends of the games
    - Return an [optimal] move that leads to a win

Adversarial Search



Formulation (Deterministic Games)

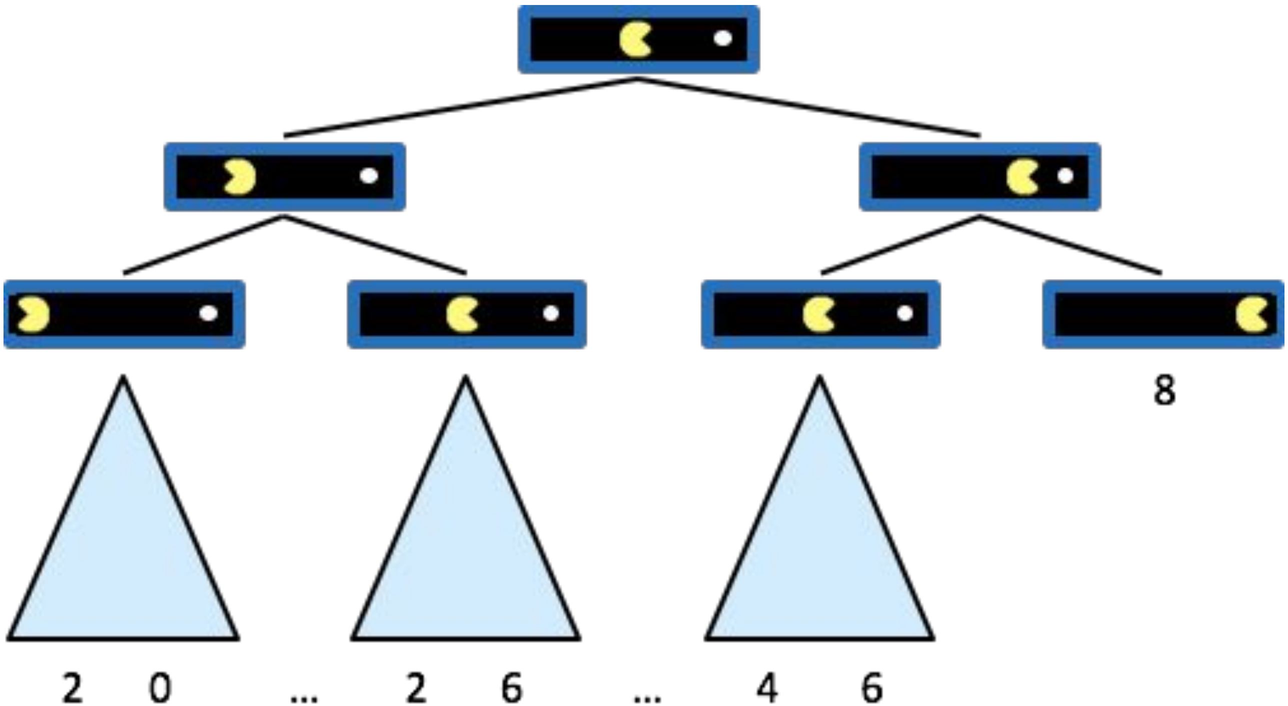
Many formulations. Here is one:

- State: the current setup
- Player(s): which player has the move
- Actions(s): A set of legal moves in a state
- Result(s,a): A transition model (successor)
- Terminal-Test(s)
- Utility(s, p): A utility function defines value for a terminal state s for a player p.

This will form a game tree (search tree)

Optimal Move: Single-Agent Trees

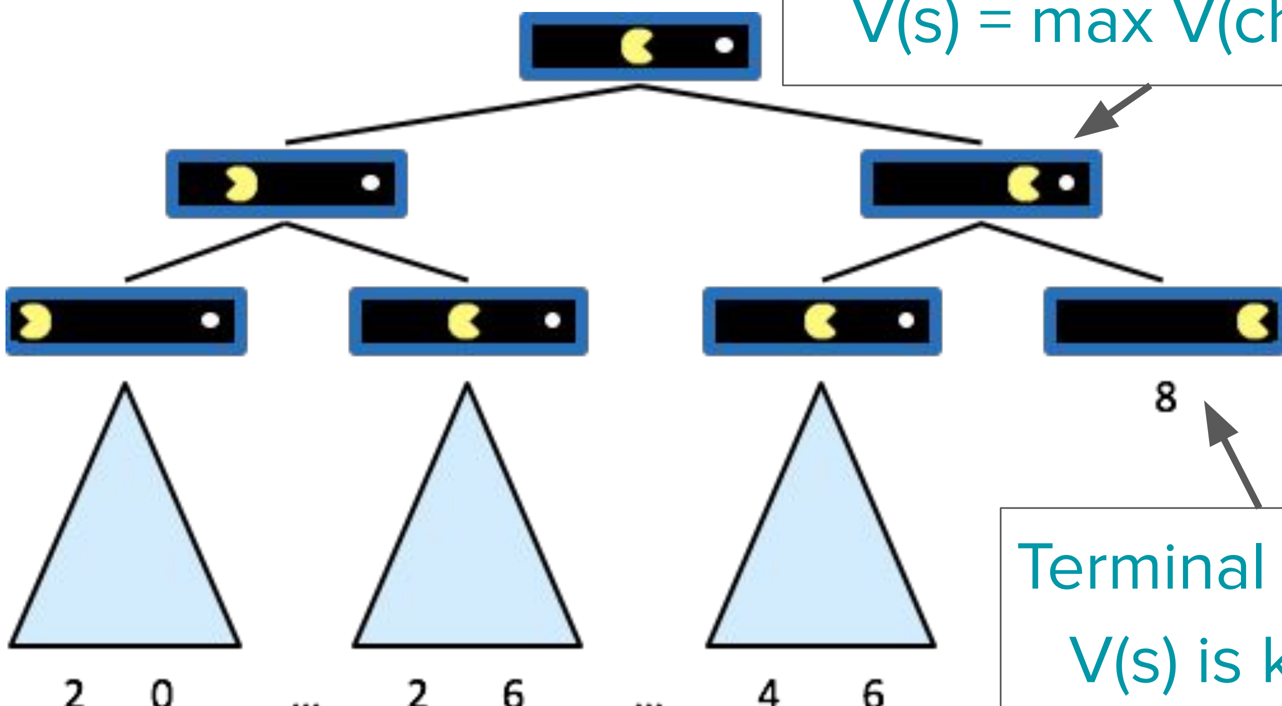
Which one maximize the value? Left or Right?



Optimal Move: Value of a State

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

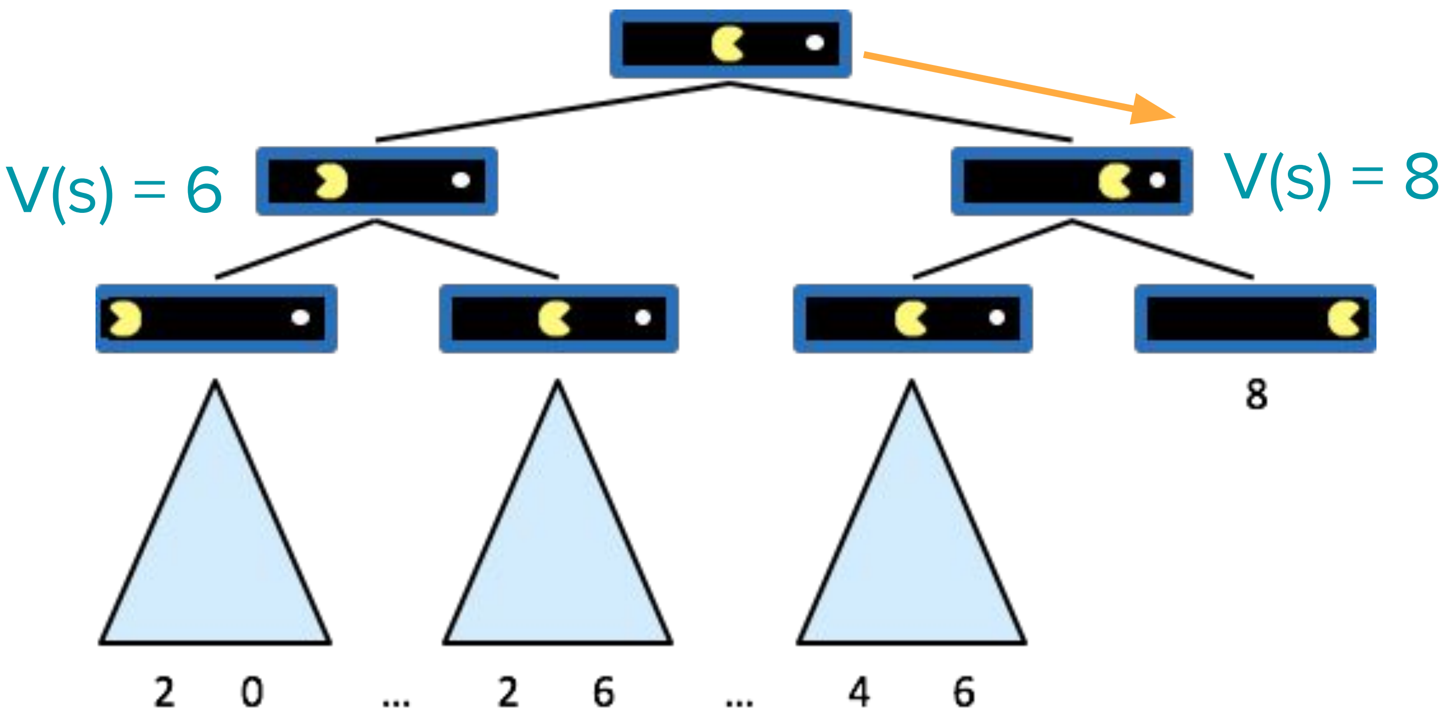
Non-Terminal states:
$V(s) = \max V(\text{children})$



Terminal states:
$V(s)$ is known

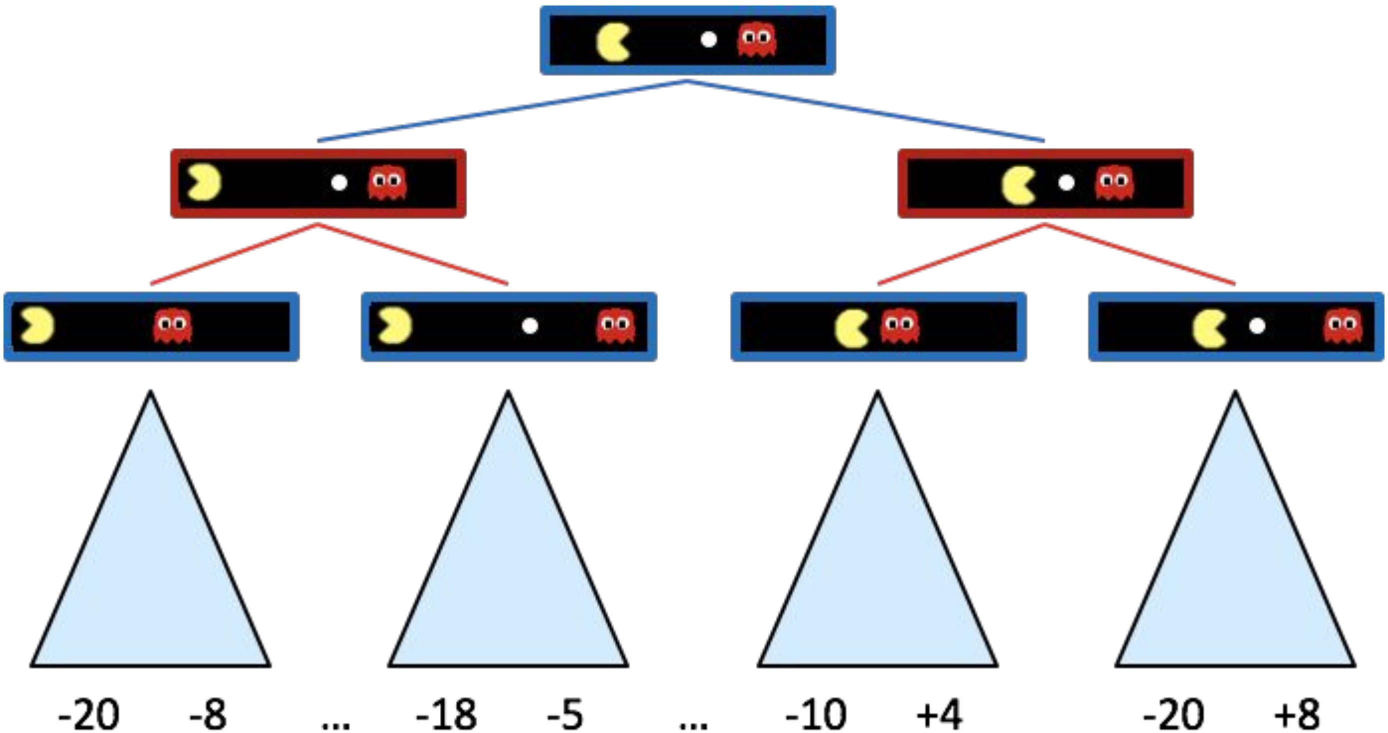
Optimal Move

An optimal move is an action that leads to a maximum-value state.



Optimal Move: Adversarial Game Trees

What is the value of non-terminal nodes?



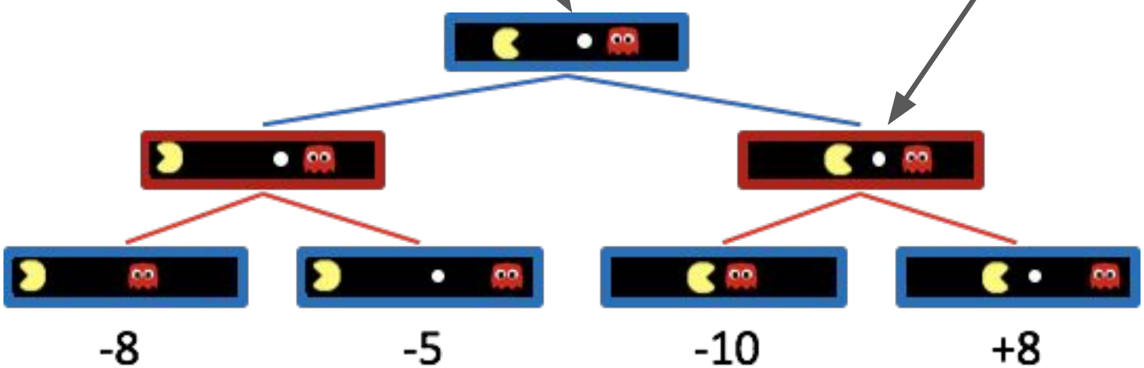
Minimax Values

States under agent's control:

V(s) = max V(children)

States under opponent's control:

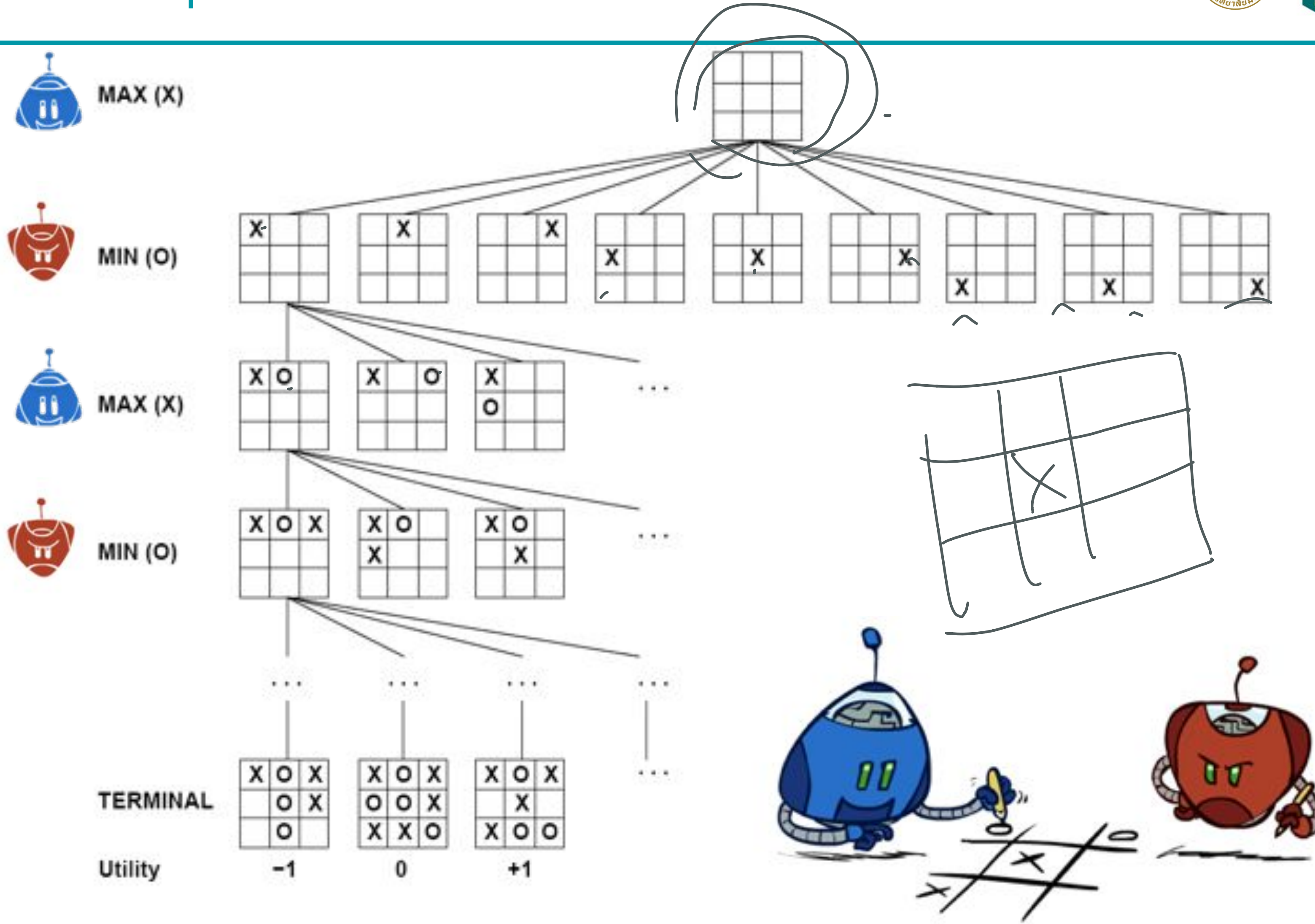
V(s) = min V(children)



Terminal states:

V(s) is known

Example: Tic-Tac-Toe



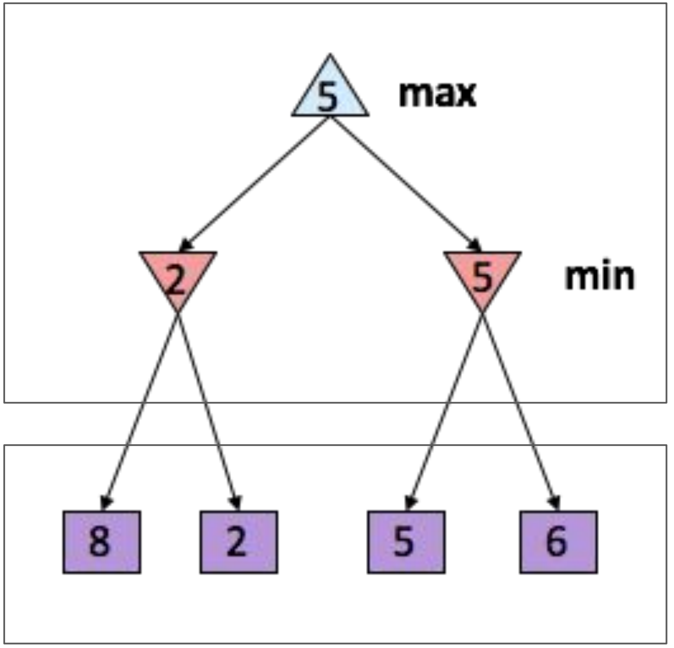
Adversarial Search: Minimax

Minimax Search

- A search tree
- Players alternate turns
- Compute each node's minimax value:
    - Assume the worst case (optimal adversary)
    - Calculate the best achievable utility

Minimax values:

Computed recursively



Terminal values:

come from the game

The minimax algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

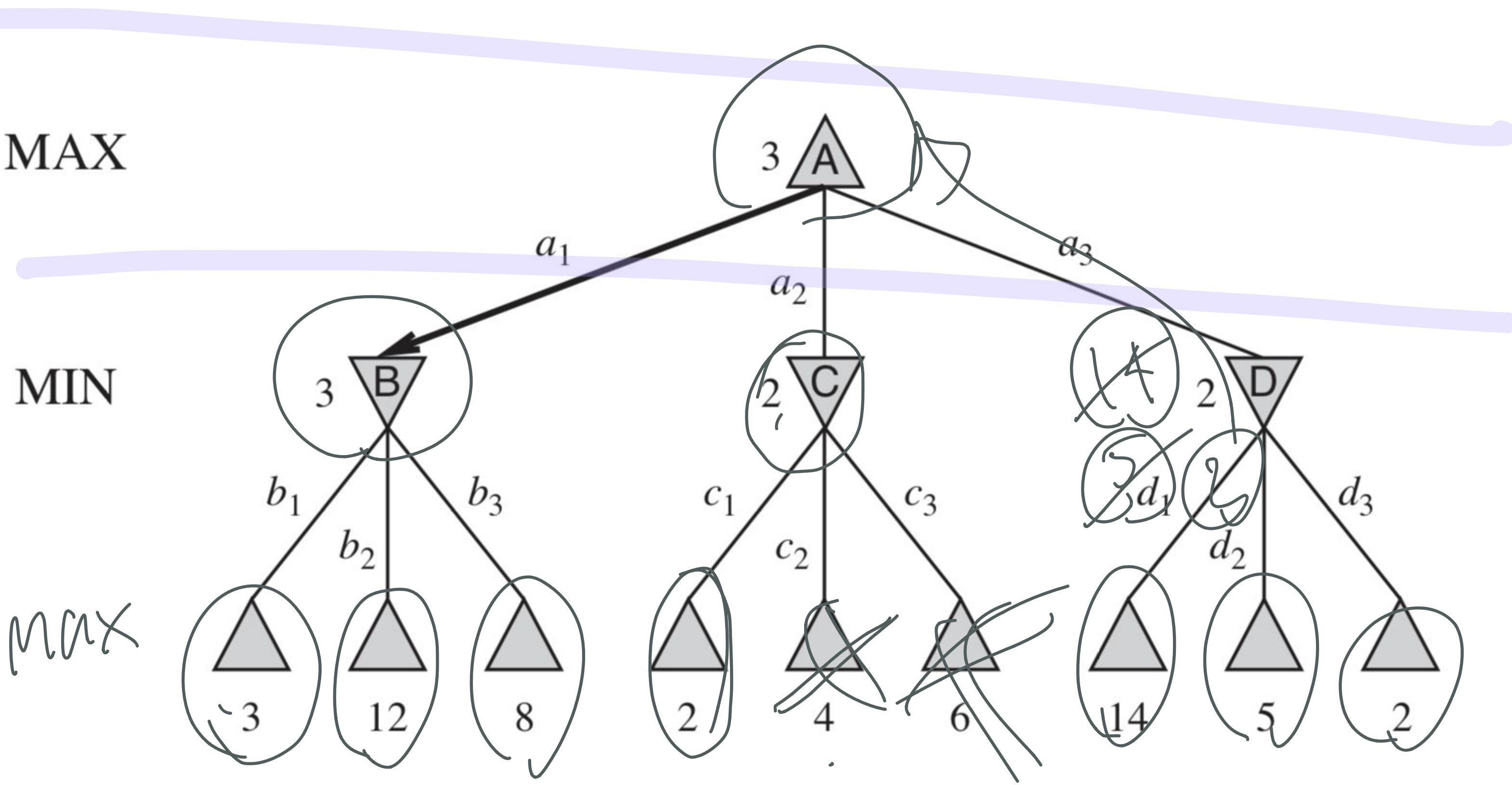
---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(*s*, *a*)))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(*s*, *a*)))
  return v

Example

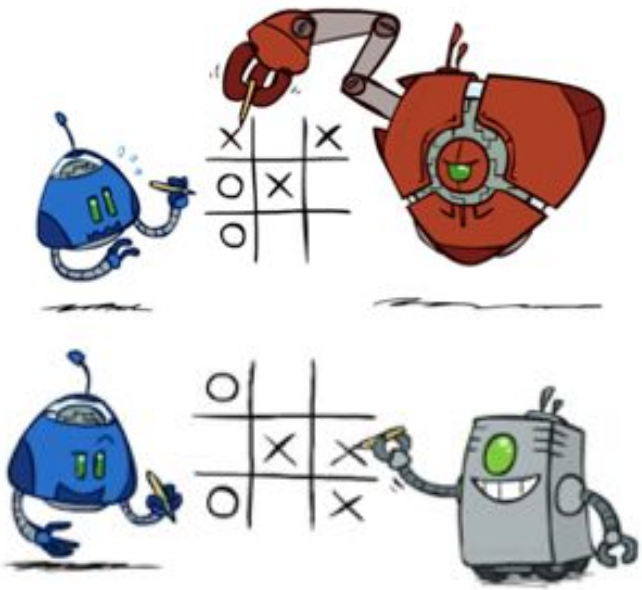


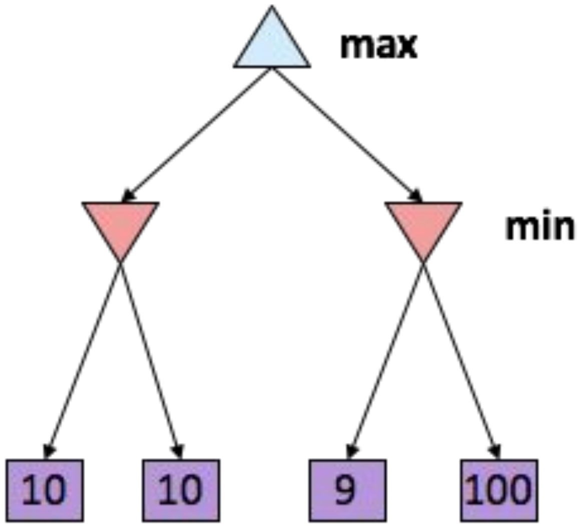
MAX
MIN
max
3
A
a_1
a_2
a_3
3
B
2
C
14
2
D
5
2
b_1
b_2
b_3
c_1
c_2
c_3
d_1
d_2
d_3
3
12
8
2
4
6
14
5
2

Minimax Properties

- Optimal?
  Yes, against an optimal player

- What if the opponent is not an optimal player?





Minimax Properties

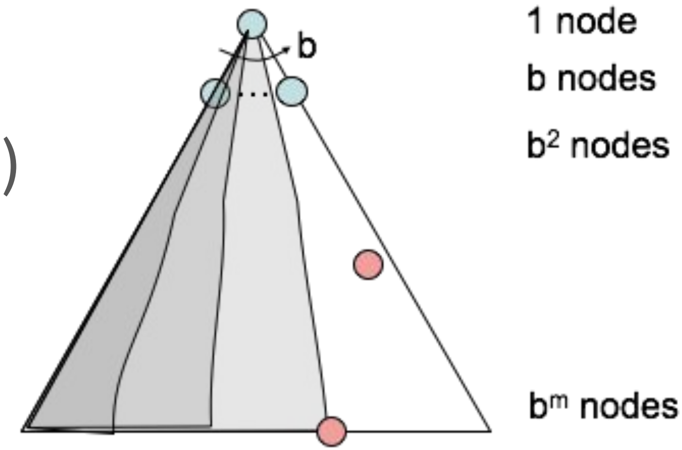
Minimax does not include reasoning about the opponent behaviors other than optimal ones.

- Expectimax: values should be a weighted average of different outcomes
- More formally, Markov Decision Processes
- Not covered in this class

Minimax Properties (Continued)

Minimax explores a tree in DFS order

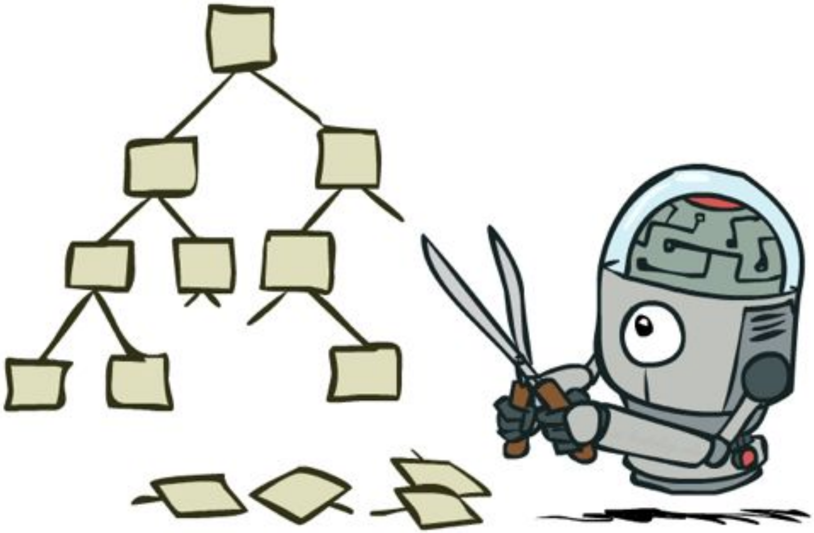
- Time Complexity: $O(b^m)$
- Space Complexity: $O(bm)$



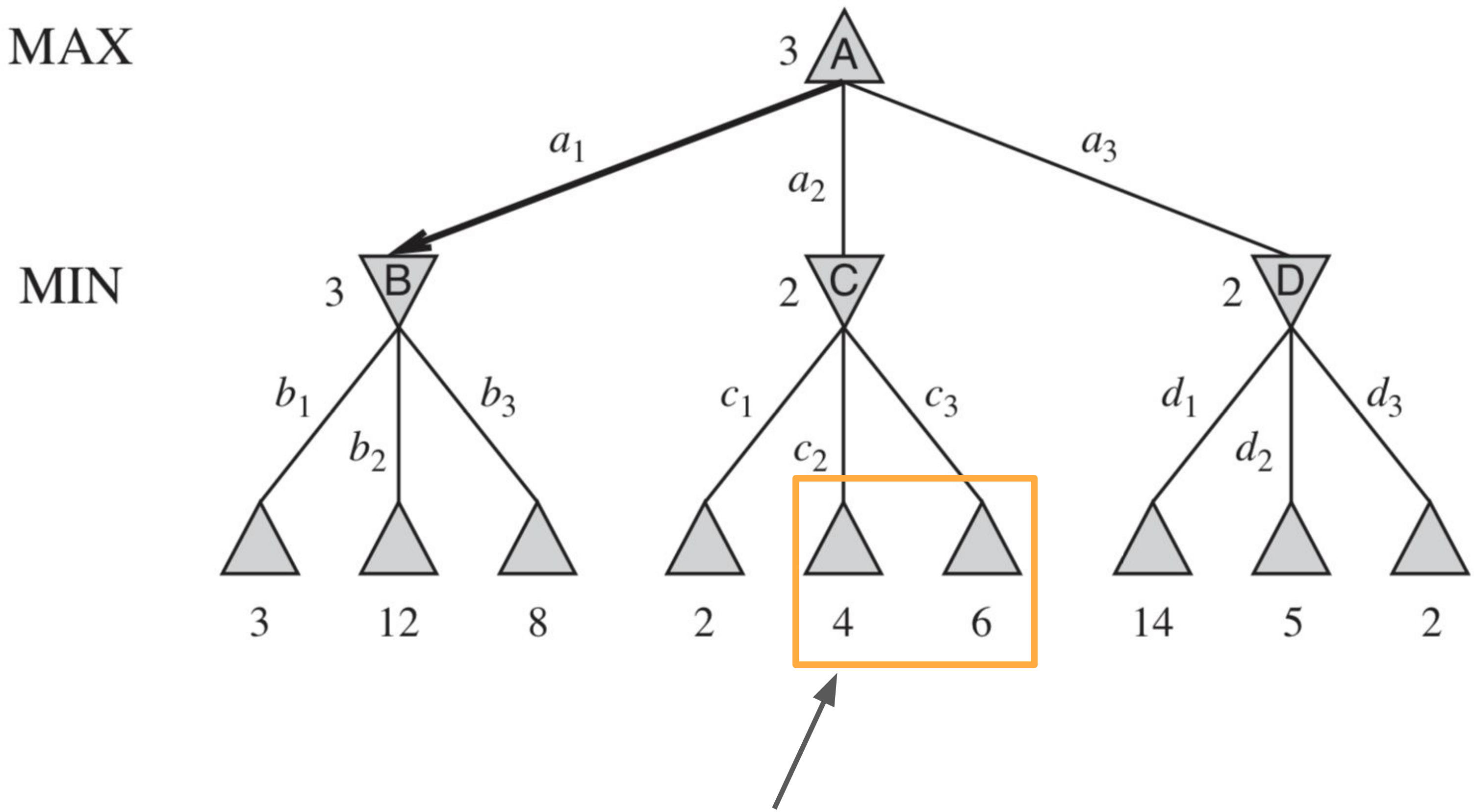
For example:

- In chess, $b \approx 35$, $m \approx 100$
- In Go, $b \approx 250$, $m \approx 210$

Improving Efficiency:
(1) Game Tree Pruning

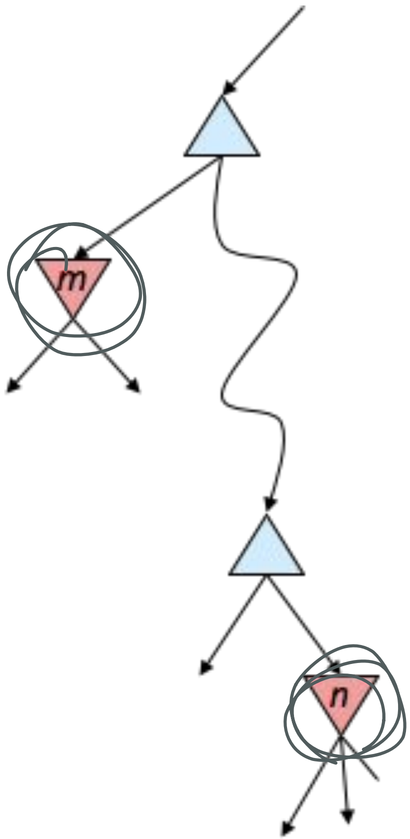


Minimax Example



Do we need to explore more after c_1?

Alpha-Beta Pruning (MIN)



General Idea (MIN):

Whenever *n falls below m*, the MAX agent will avoid it.

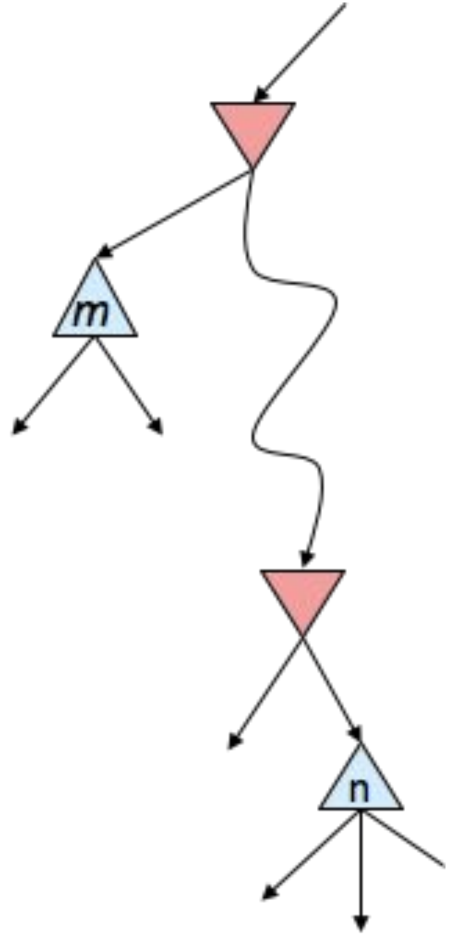
MIN can stop exploring that branch

Alpha-Beta Pruning (MAX)

General Idea (MAX):

Whenever *n* exceeds *m*, the MIN agent will avoid it.

MAX can stop exploring that branch



Alpha-Beta Search Algorithm

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

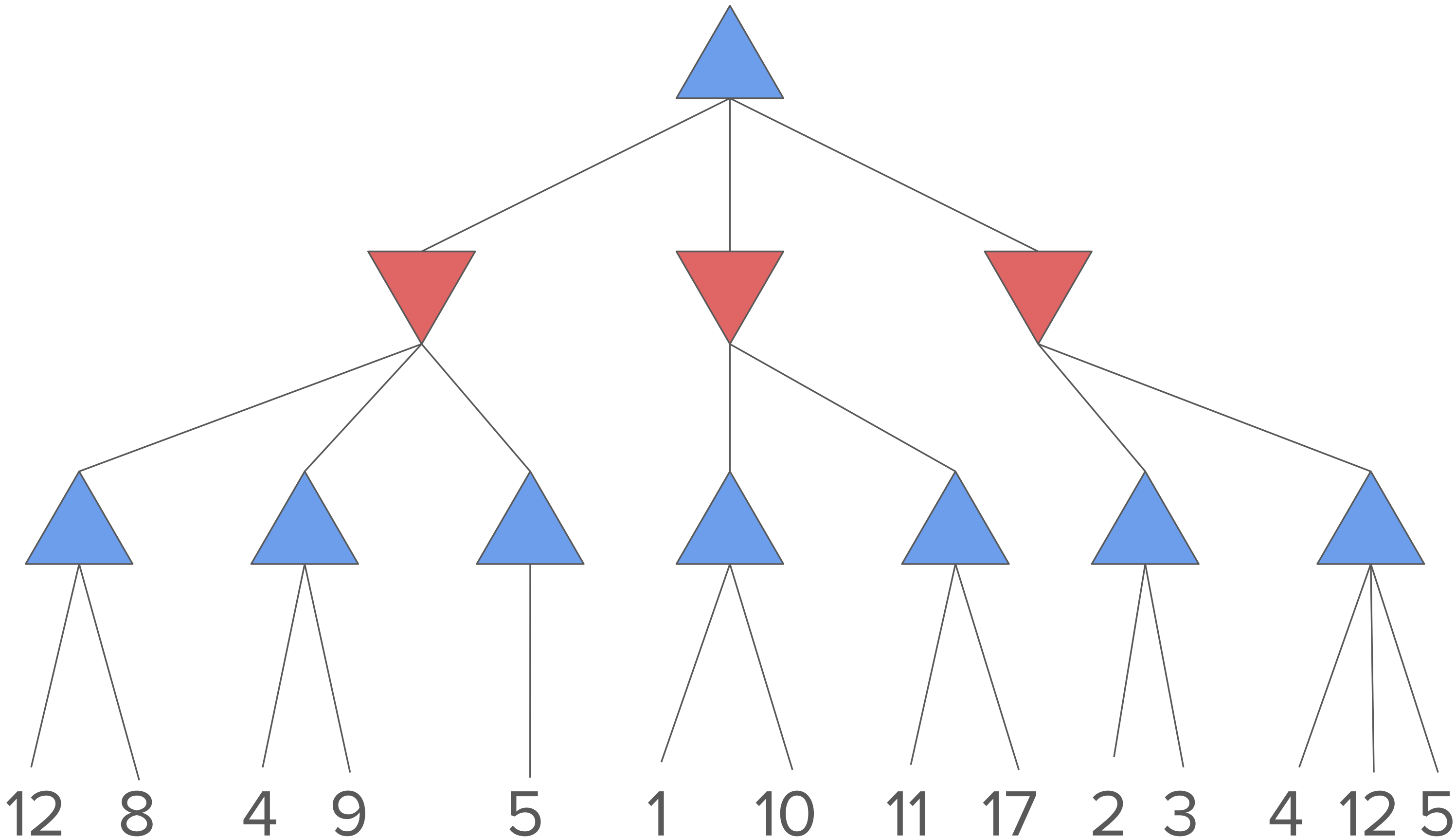
---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, $\boxed{\text{MIN-VALUE(RESULT}(s,a), \alpha, \beta)}$)
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, $\boxed{\text{MAX-VALUE(RESULT}(s,a), \alpha, \beta)}$)
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Example

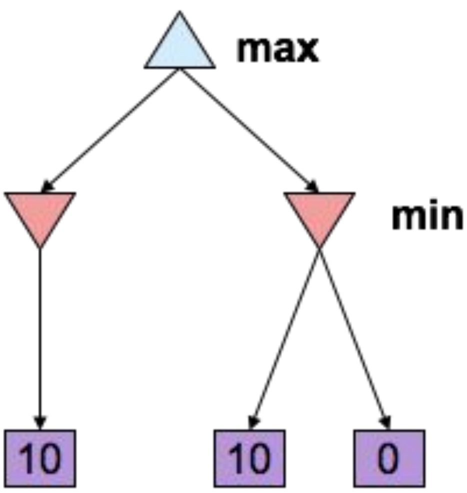


Alpha-Beta Search Properties

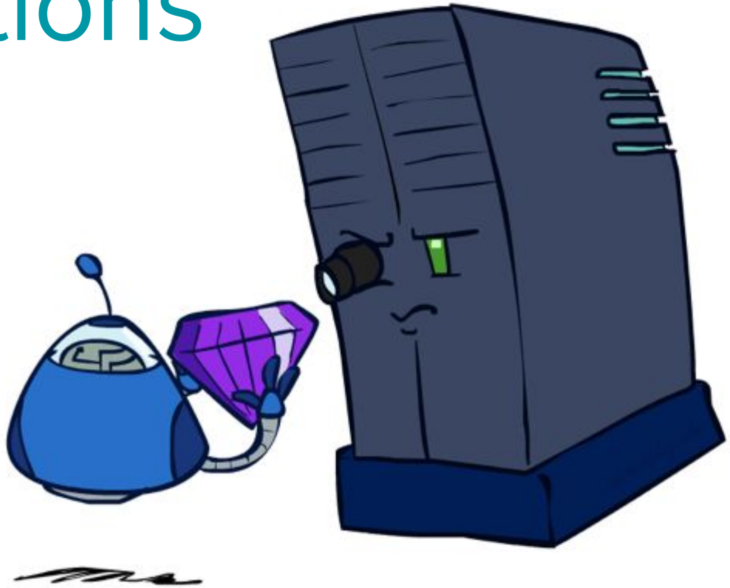
- This pruning has no effect on minimax value for the root

- But the values of intermediate nodes might be wrong ➜ Cannot use in action selection

- Good children ordering improves Effectiveness of pruning
  - Impossible to get a perfect ordering

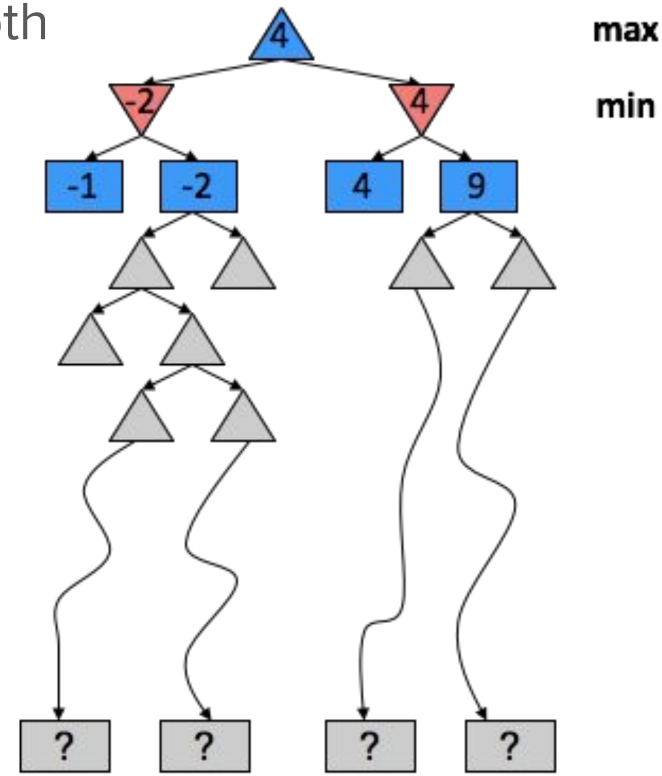


Improving Efficiency: (2) Cutting off and Evaluation Functions



Depth-limited search

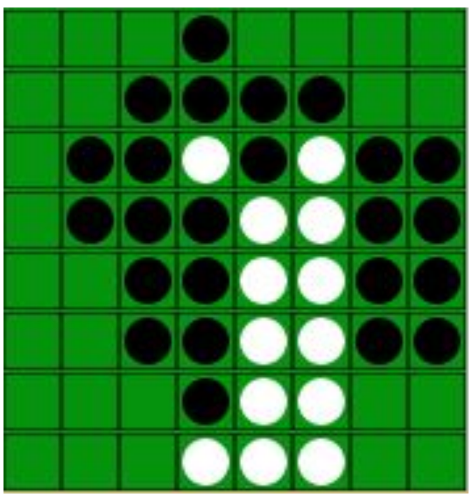
- Search only a limited depth in the tree
  - Cut off the search a some depth
  - Similar to IDS
  - What is the values of non-terminal nodes?
- Use evaluation function!
  - Approximate the values without looking ahead
  - Not optimal anymore



Evaluation Functions

- Score non-terminal nodes
- Ideal function: actual minimax value
- In practice a weighted linear sum of features:

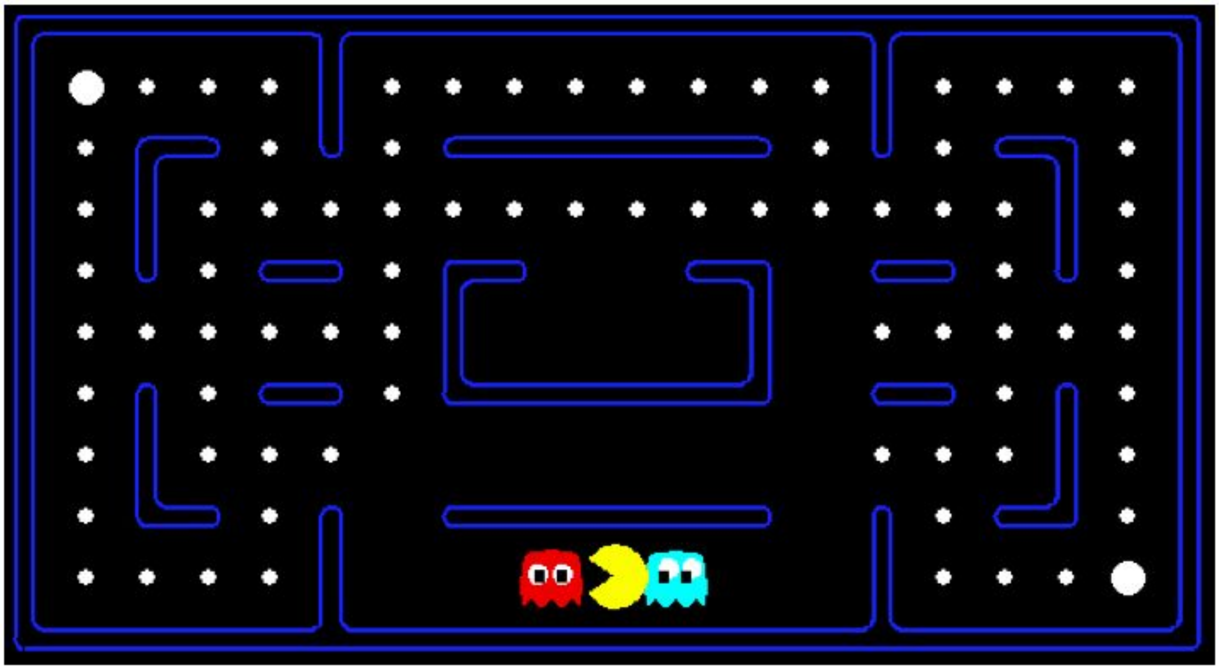
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$



Example: Othello

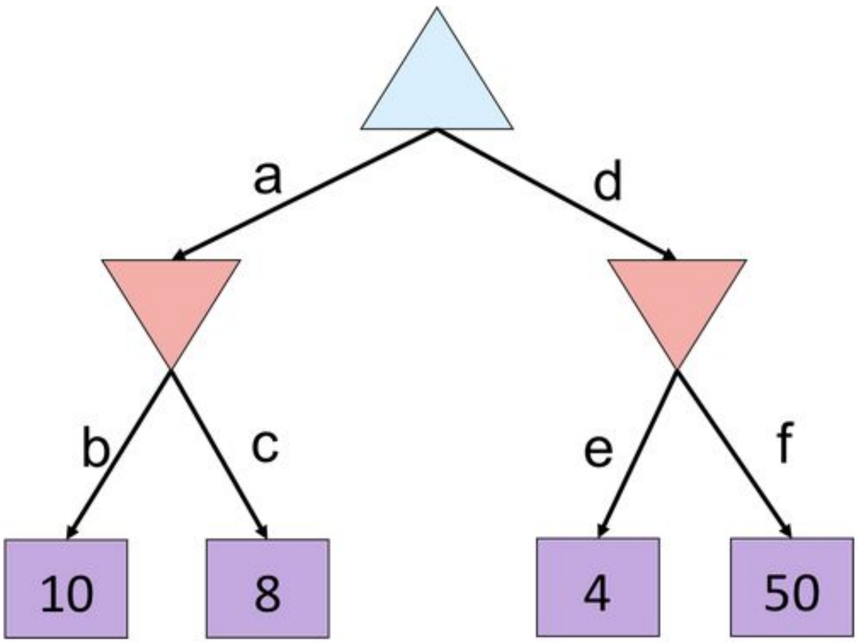
- $f_1(s)$ = #white legal moves
- $f_2(s)$ = #white - #black
- ...

Evaluation for Pacman



Quiz

- What is the minimax value of the root node?
- What are the actions (alphabet) that the agent explore?



Quiz

What about this one?