# Announcements

- HW1 was due **Wednesday, February 5**, 11:59 PM PT

- Project 1 is due **Friday, February 7**, 11:59 PM PT

- HW2 is due **Wednesday, February 12**, 11:59 PM PT

- Project 2 is due **Friday, February 21**, 11:59 PM PT

- Oliver's OH 2/6 moved to remote hours on 2/7 2:30-4:00 over class zoom meeting (Meeting ID: 995 0435 8998 Passcode: 852823)

- Please attend at least one discussion section per week

CS 188: Artificial Intelligence
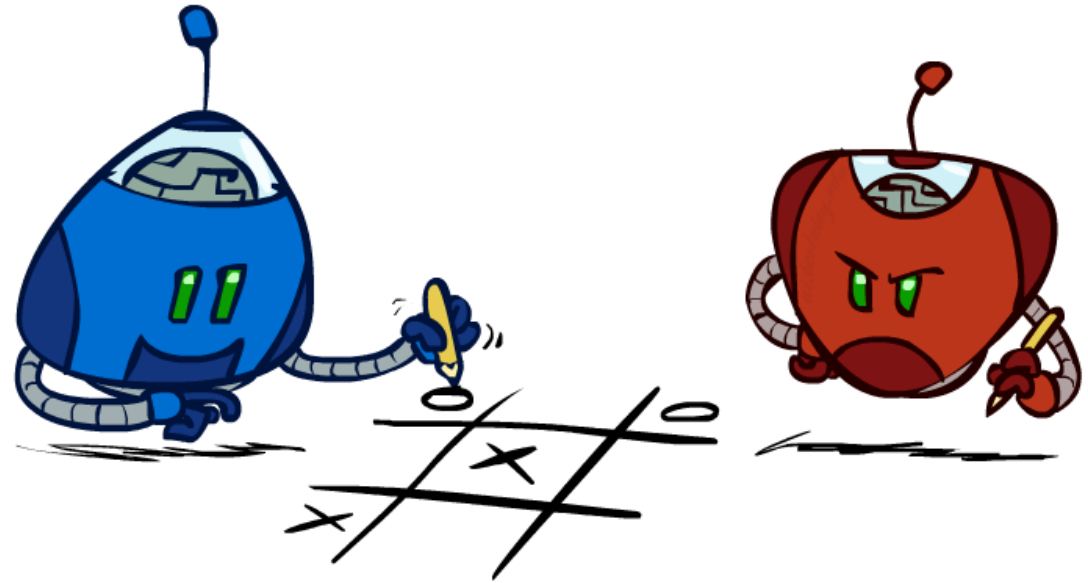## Game Trees: Adversarial Search
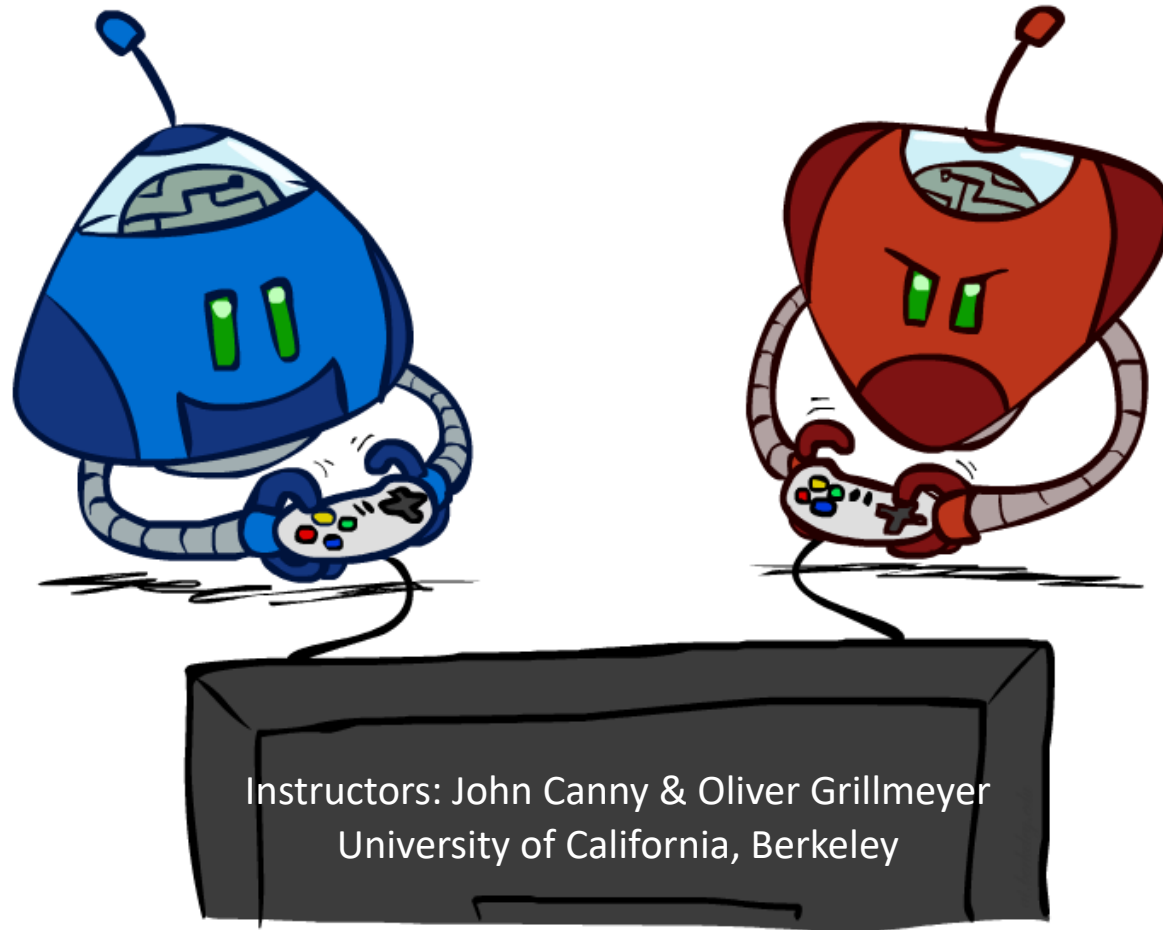
Instructors: John Canny & Oliver Grillmeyer
University of California, Berkeley

Outline

- History / Overview

- Minimax for Zero-Sum Games

- α-β Pruning

- Finite lookahead and evaluation

Game Playing State of the Art

- **Checkers:**
    - 1950: First computer player
    - 1959: Samuel's self-taught program
    - 1995: First computer world champion beat 40 year champion Marion Tinsley *
   - 2007: Checkers solved!

- **Chess:**
    - 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCarthy.
    - 1960-1996: gradual improvements
    - 1997: Deep Blue defeats human champion Garry Kasparov
   - 2024: Stockfish rating 3631 (vs 2847 for Magnus Carlsen)

- **Go:**
    - 1968: Zobrist's program plays legal Go, barely (b>300!)
    - 1968-2005: various ad hoc approaches tried, novice level
    - 2005-2014: Monte Carlo tree search -> strong amateur
    - 2016-2017: AlphaGo defeats human world champions
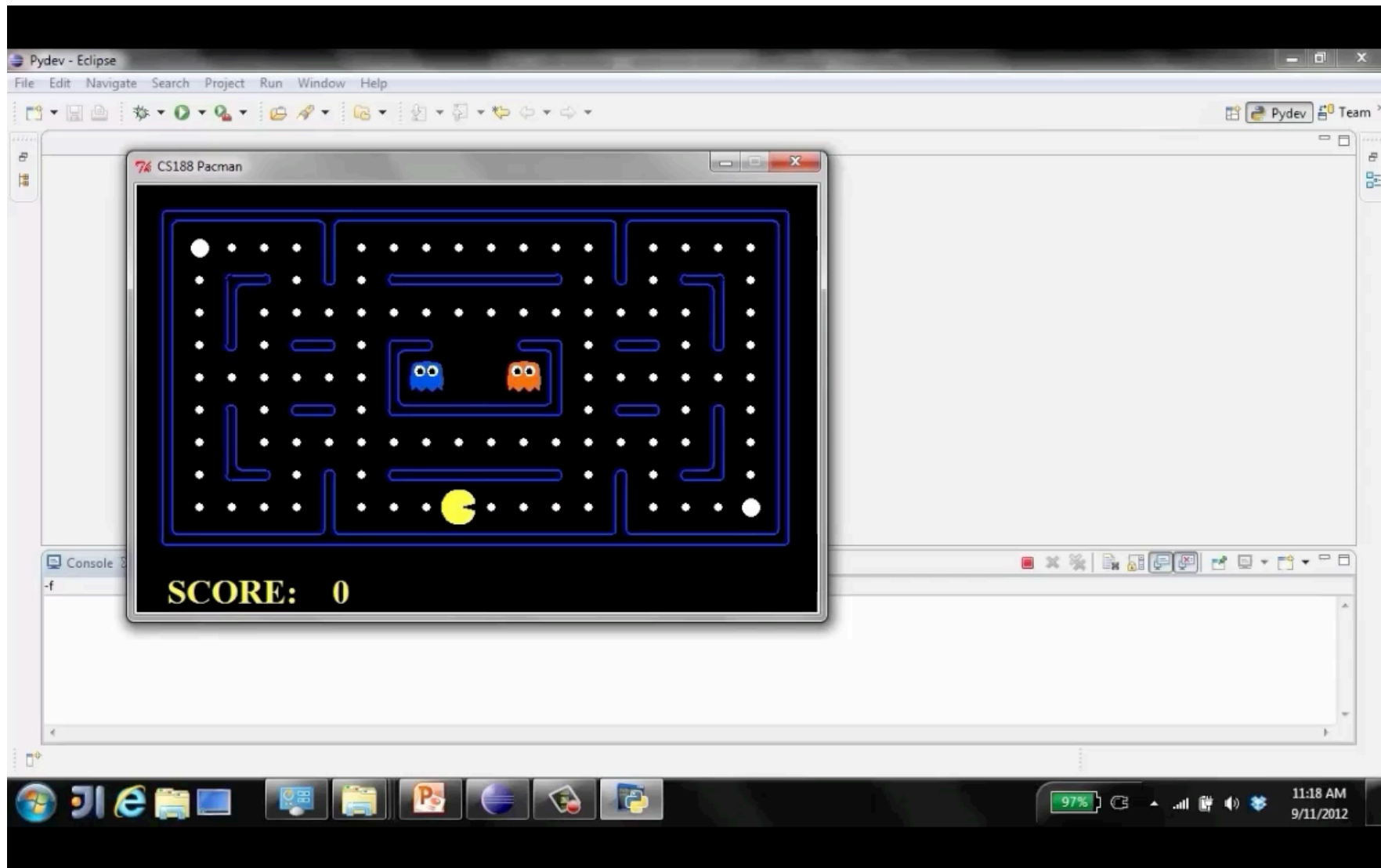   - 2022: Human exploits NN weakness to defeat top Go programs
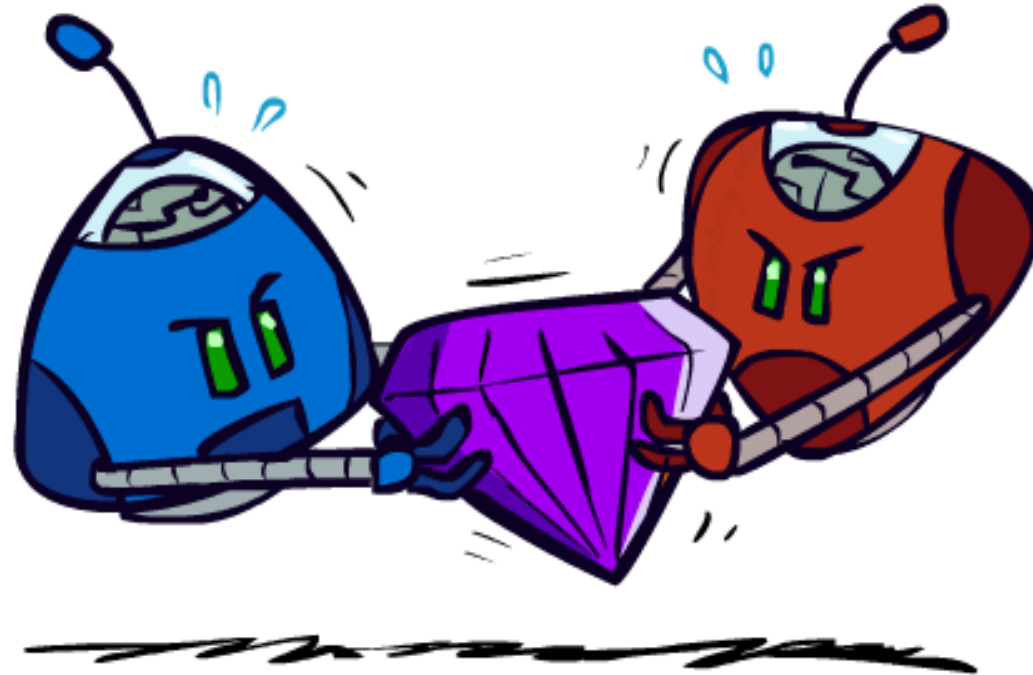
- **Pacman**

SOLVED! ┼

EXPERT ┼

HUMAN ┼

A BRICK ┼

Checkers   Chess   Go   Pacman
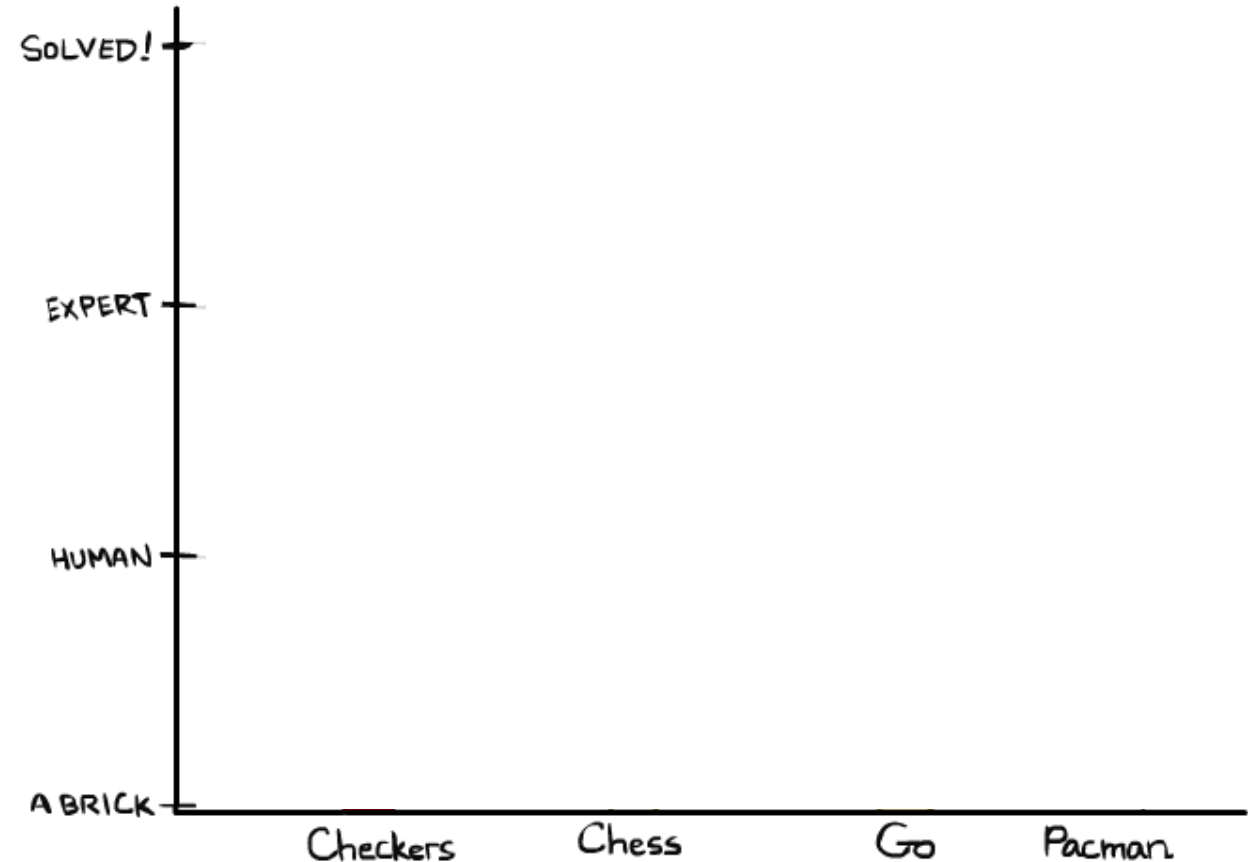
Behavior from Computation
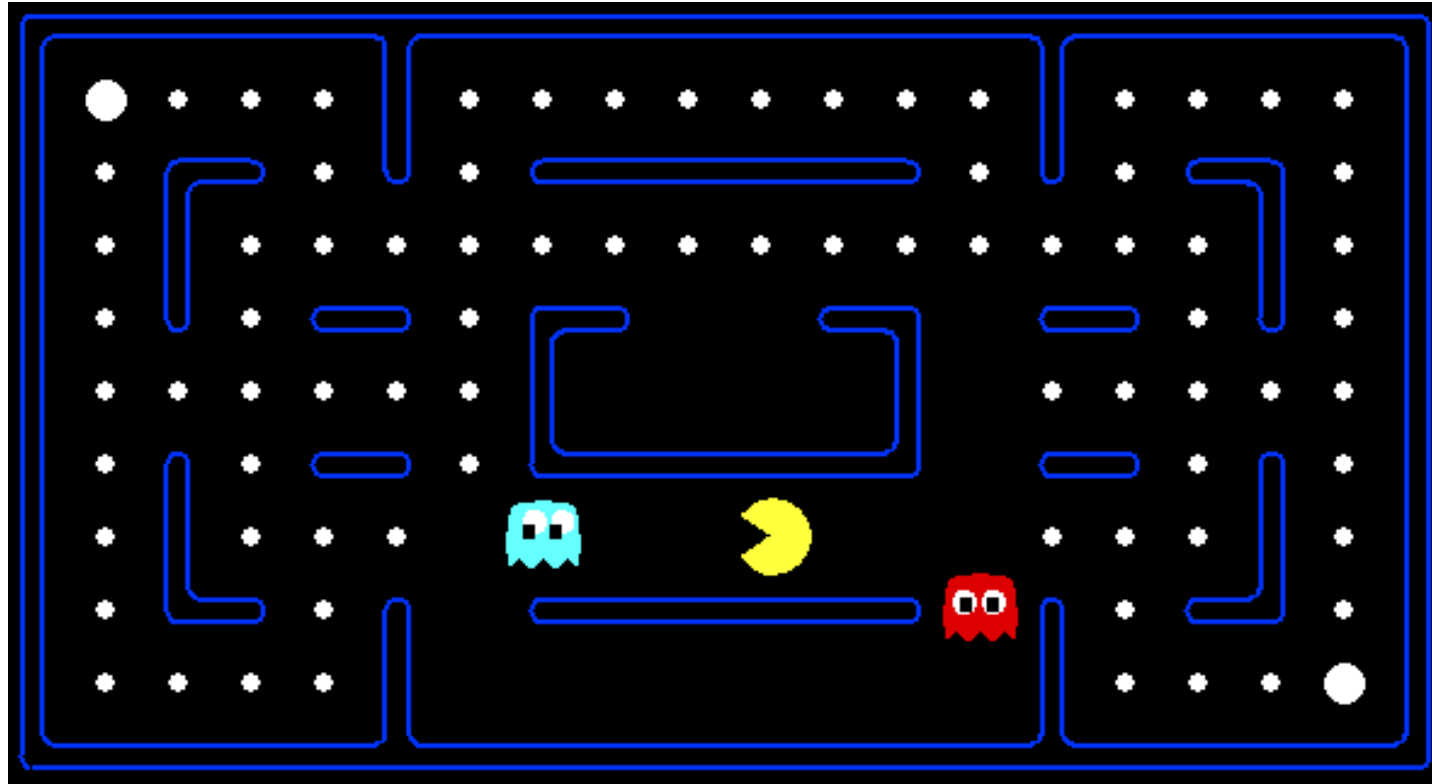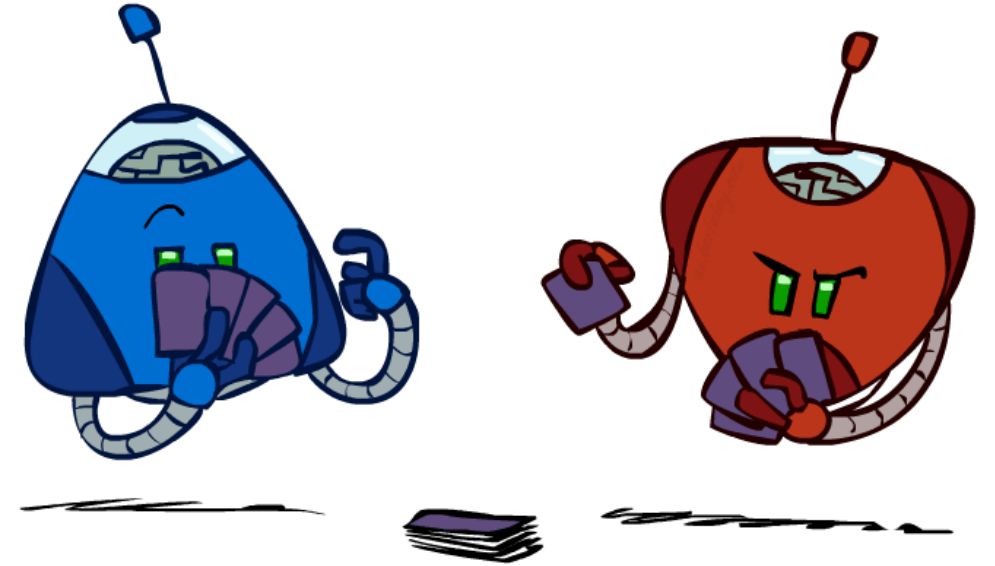
Video of Demo Mystery Pacman

Adversarial Games

Types of Games

- Game = task environment with > 1 agent
- Axes:
  - Deterministic or stochastic?
  - Perfect information (fully observable)?
  - Two, three, or more players?
  - Teams or individuals?
  - Turn-taking or simultaneous?
  - Zero sum?

- Want algorithms for calculating a **strategy** (policy) which recommends a move from every possible 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 x A → S
  - Terminal test: S → {true, false}
  - Terminal utilities: S x P → R
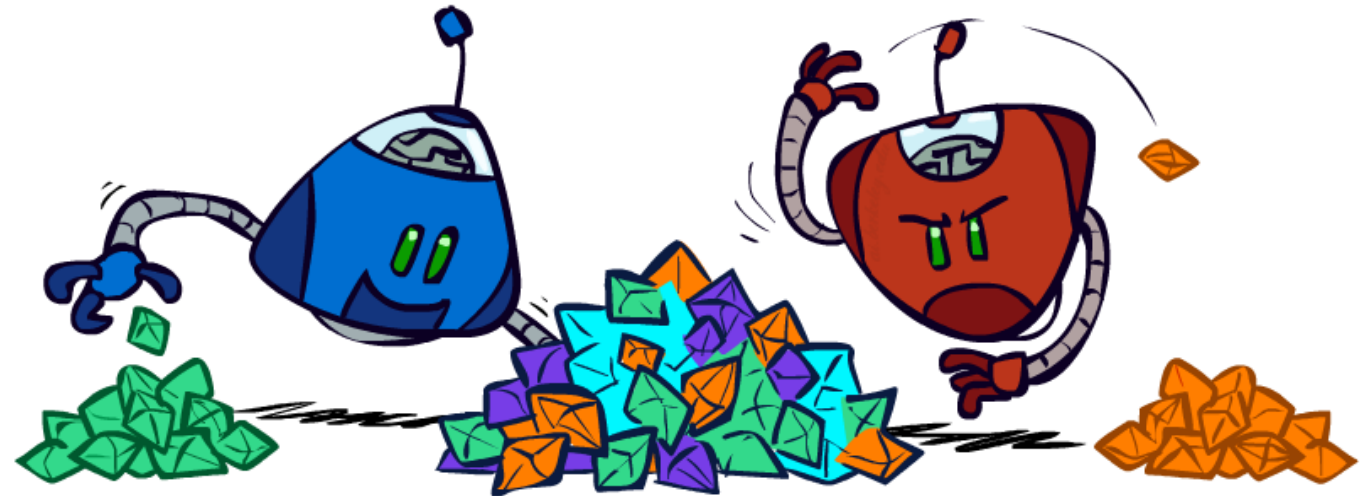
- Solution for a player is a policy: S → A
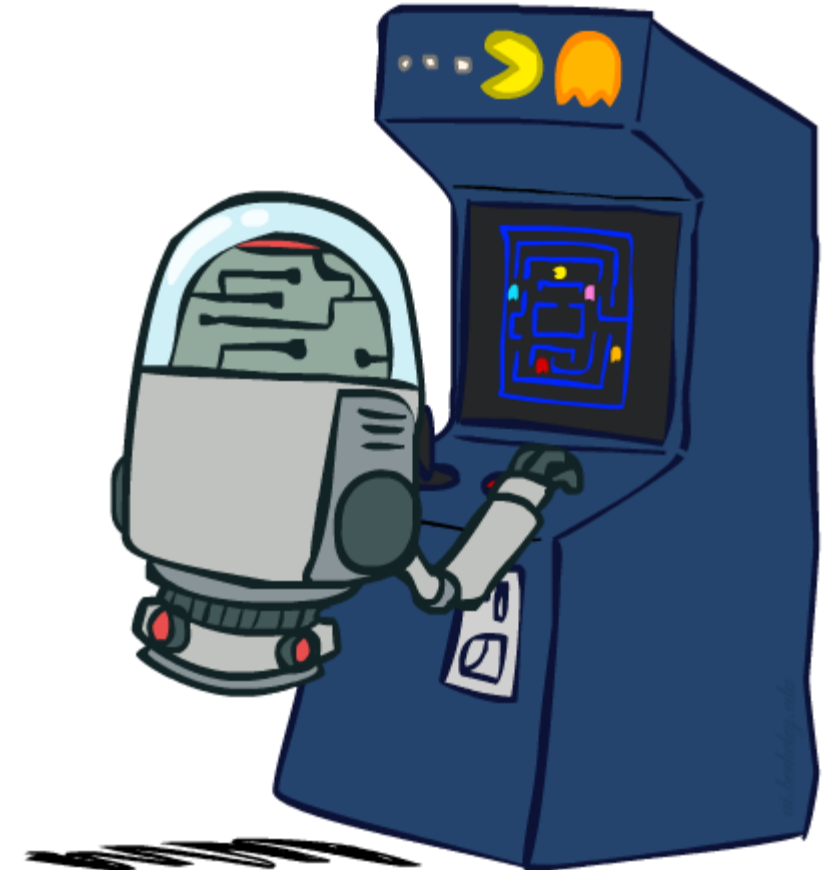
Zero-Sum Games
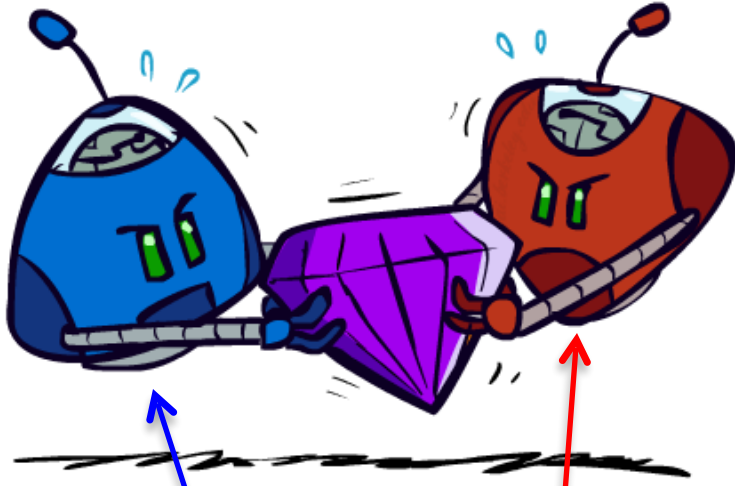
- Zero-Sum Games
  - Agents have *opposite* utilities (values on outcomes)
  - Pure competition:
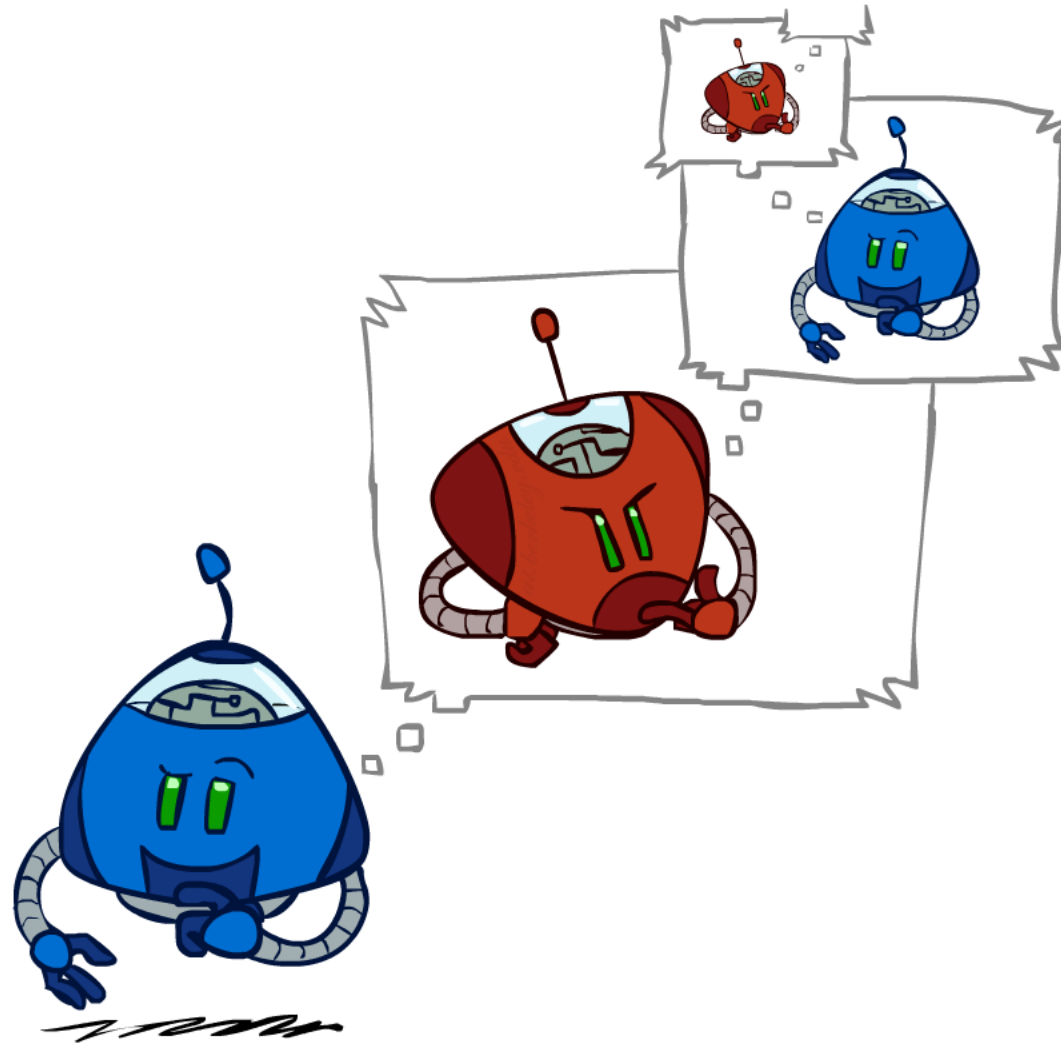   - One *maximizes*, the other *minimizes*

- General-Sum Games
  - Agents have *independent* utilities (values on outcomes)
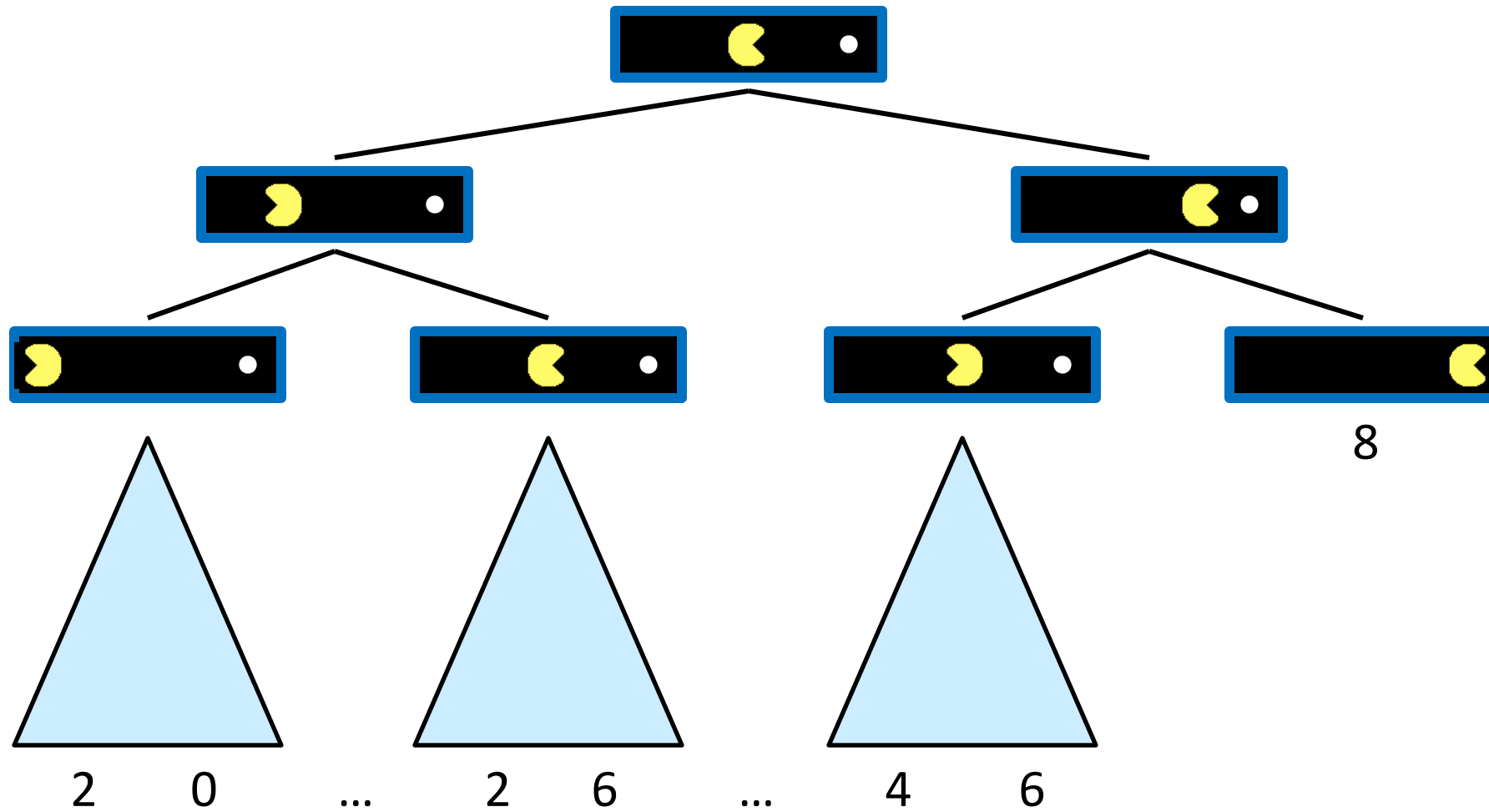 - Cooperation, indifference, competition, shifting alliances, and more are all possible

- Team Games
  - Common payoff for all team members
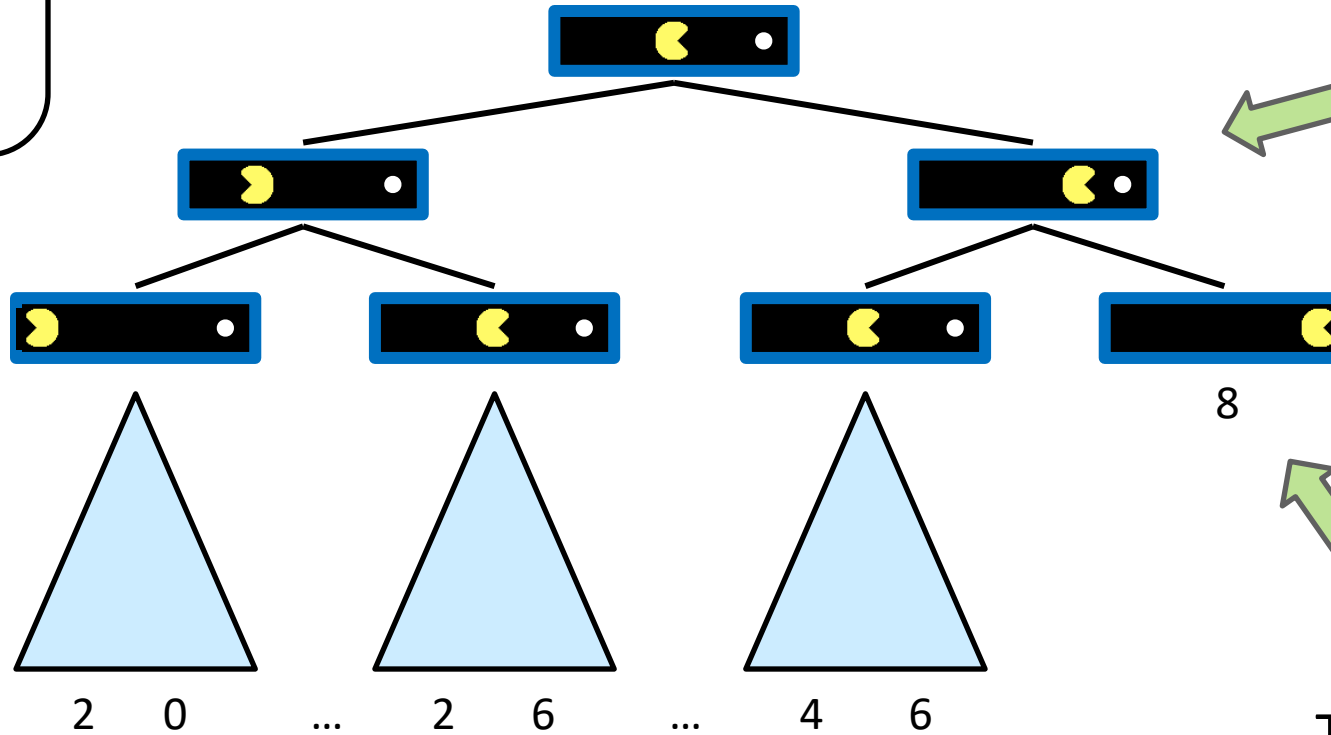
Adversarial Search

Single-Agent Trees

8

2   0   ...   2   6   ...   4   6

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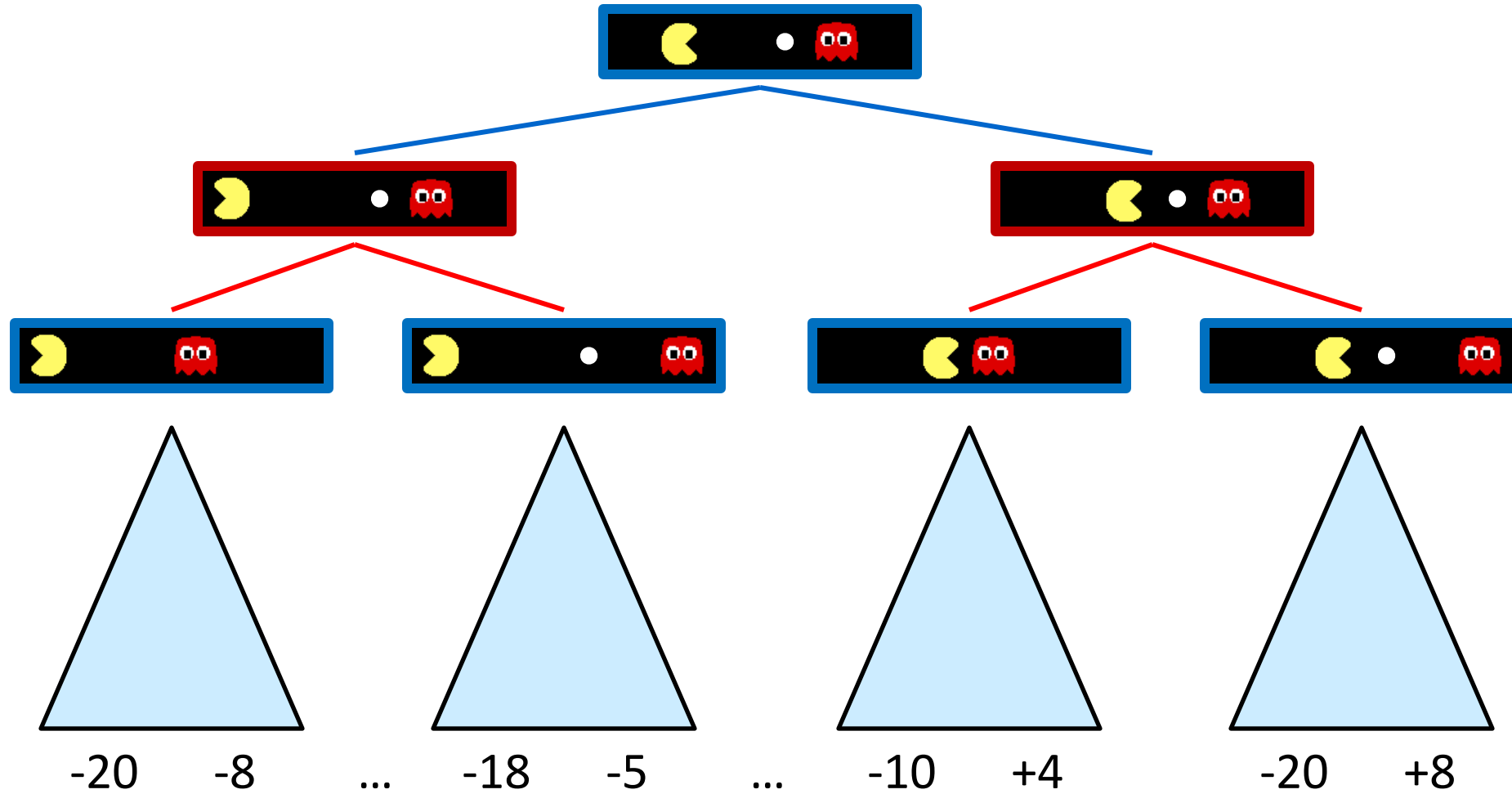
Value of a state:
The best achievable
outcome (utility)
from that state

Non-Terminal States:

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

8

Terminal States:

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

2   0   ...   2   6   ...   4   6

Adversarial Game Trees

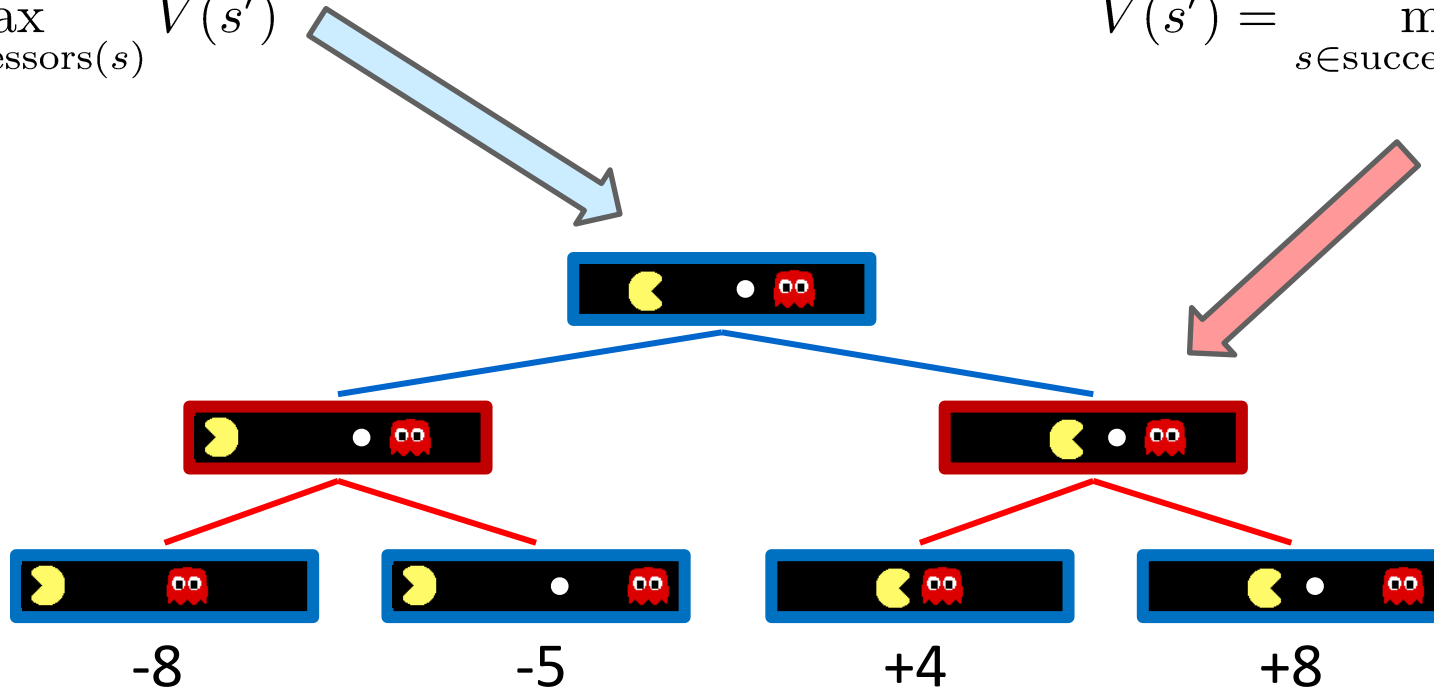-20   -8   ...   -18   -5   ...   -10   +4   -20   +8

Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

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

-8   -5   +4   +8

Terminal States:

$$V(s) = \text{known}$$
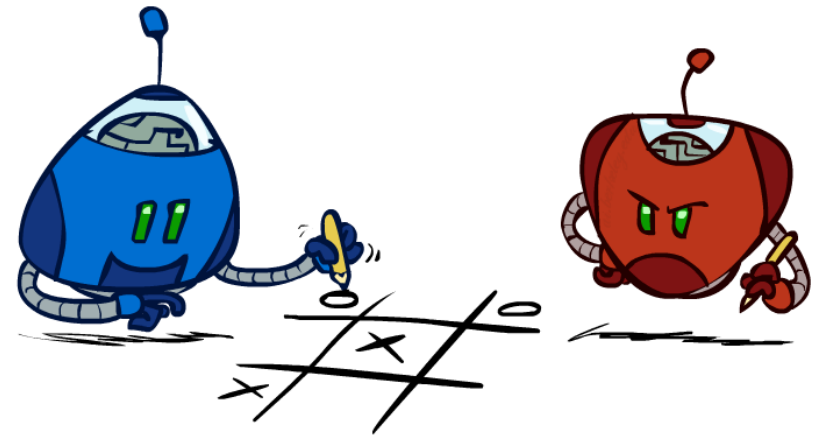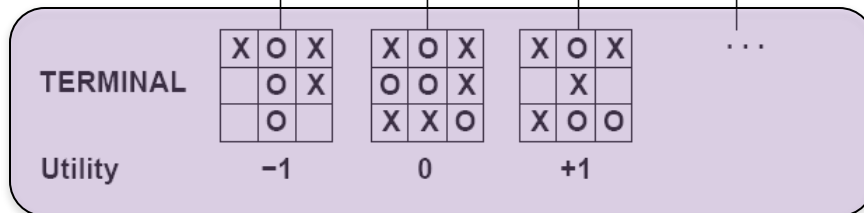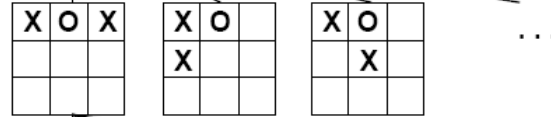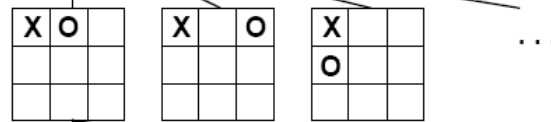
Tic-Tac-Toe Game Tree
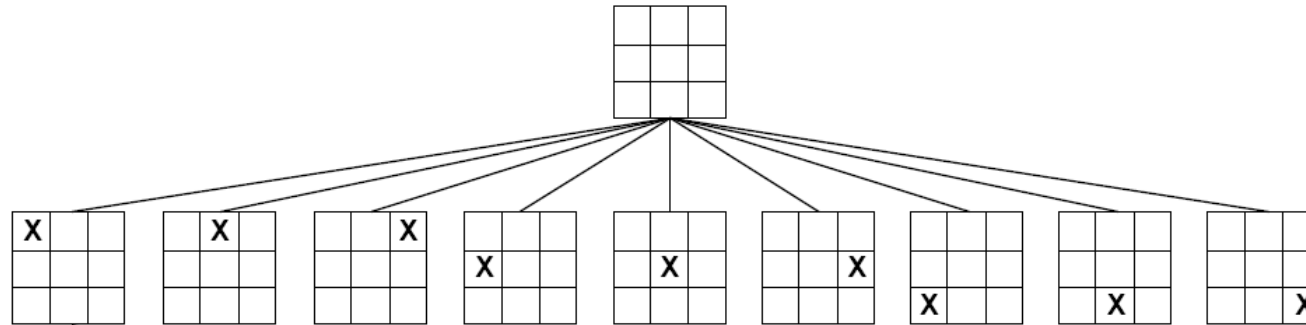
Adversarial Search (Minimax)

- 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 values:
computed recursively

Terminal values:
part of the game
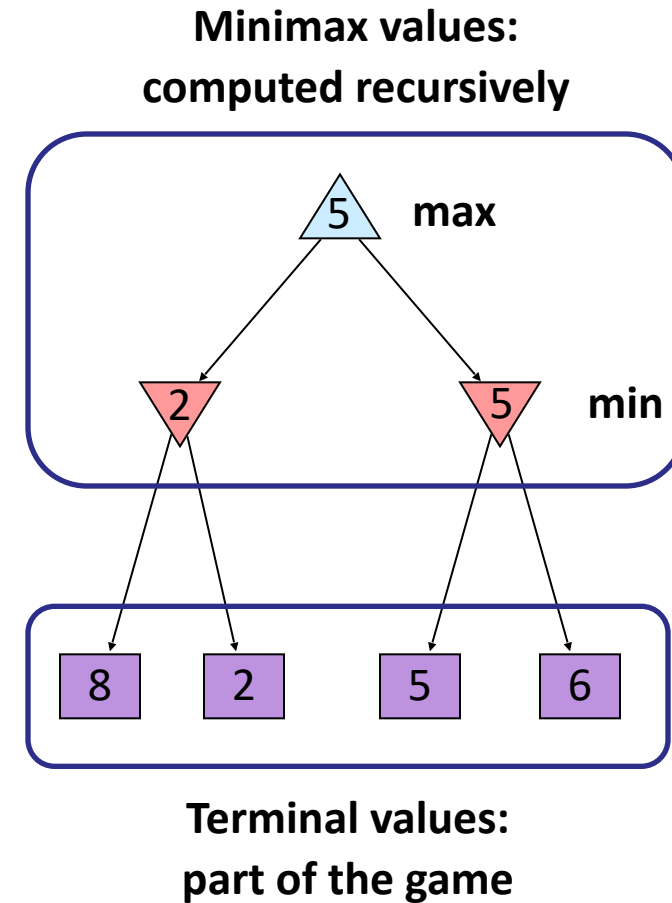
Minimax Implementation

def max-value(state):

    initialize v = $-\infty$

    for each successor of state:

        v = max(v, min-value(successor))
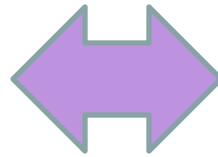
    return v

def min-value(state):

    initialize v = $+\infty$

    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
```
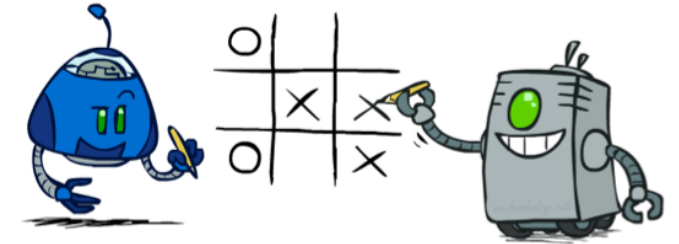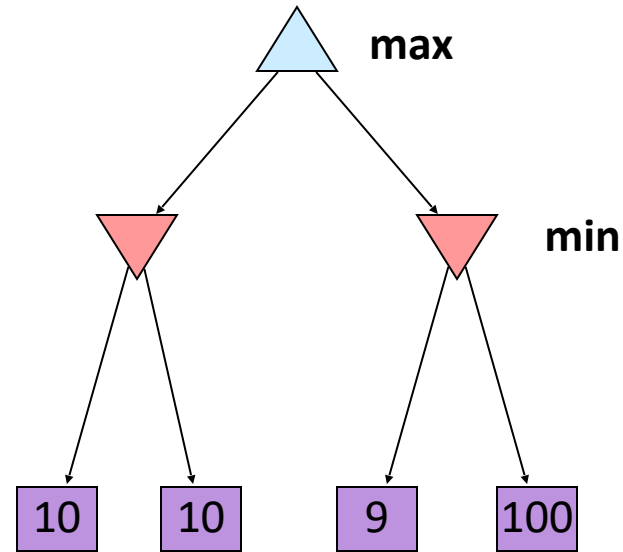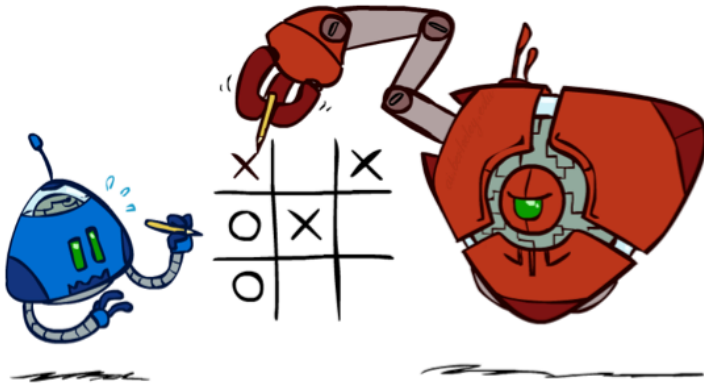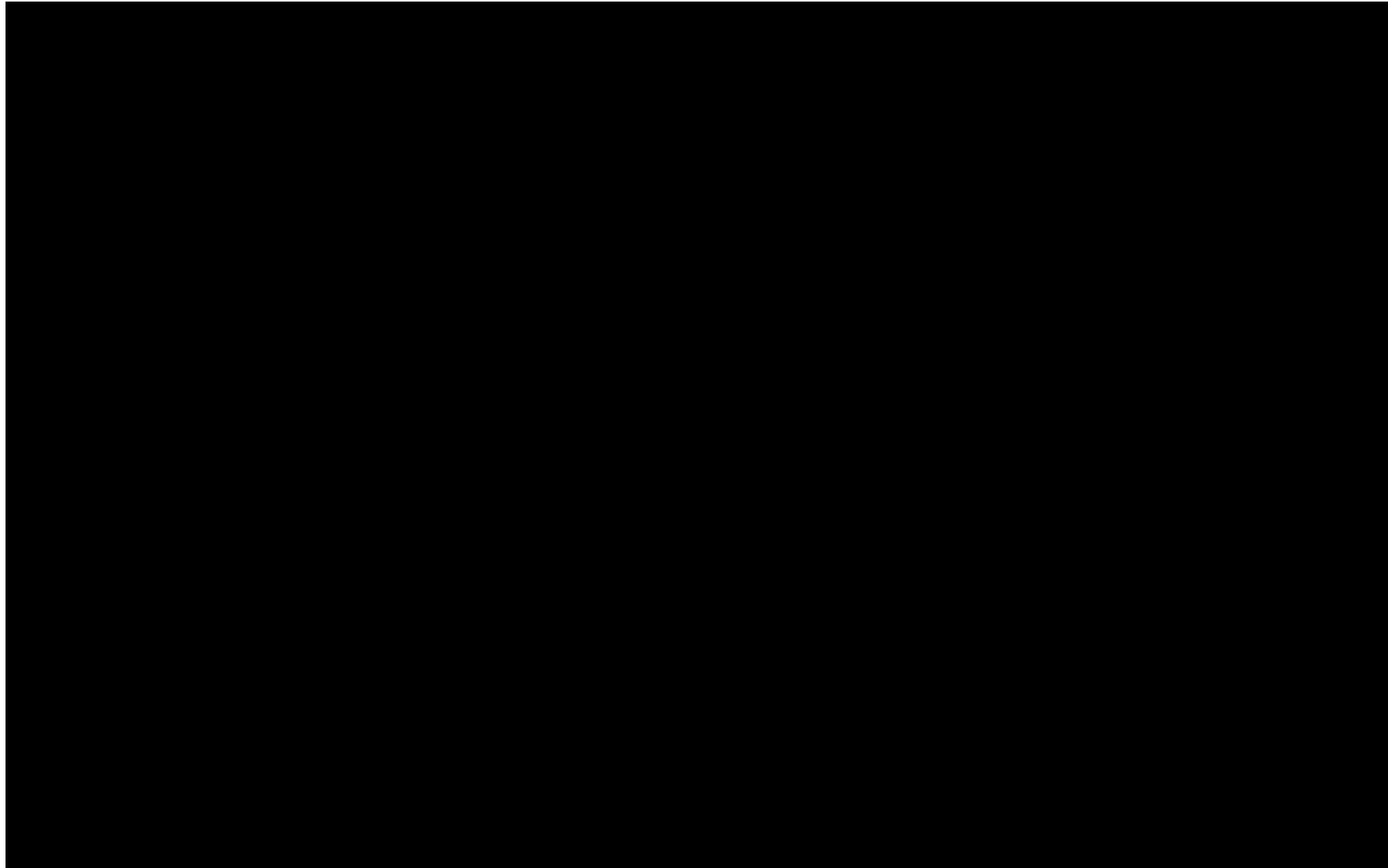
Minimax Example
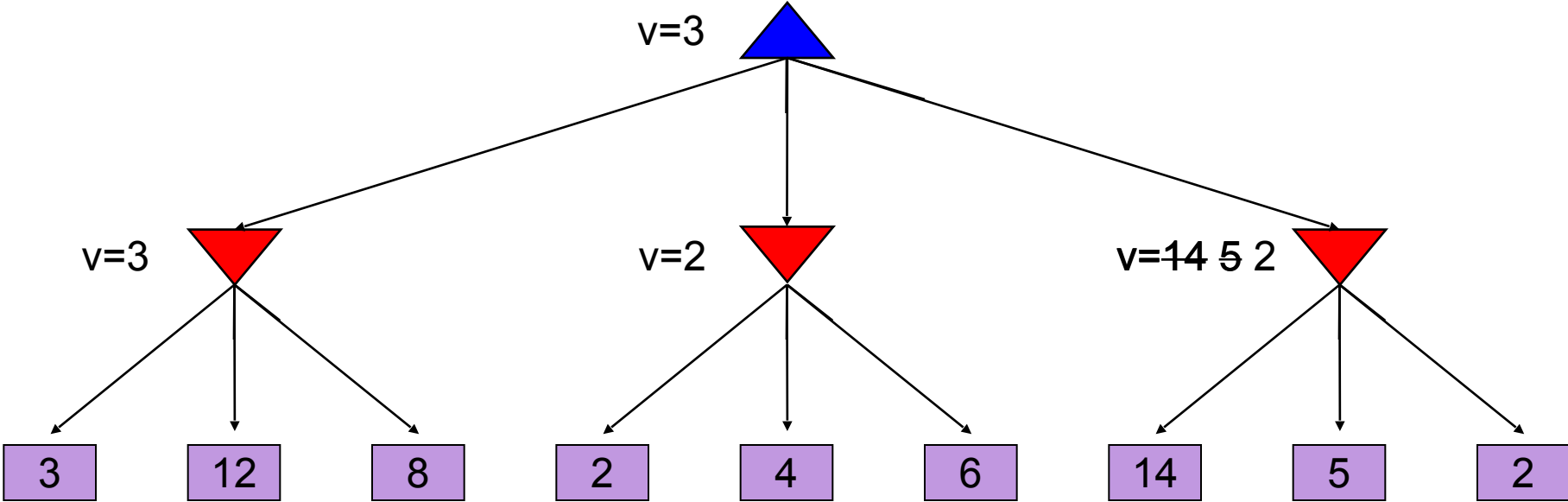
Minimax Properties

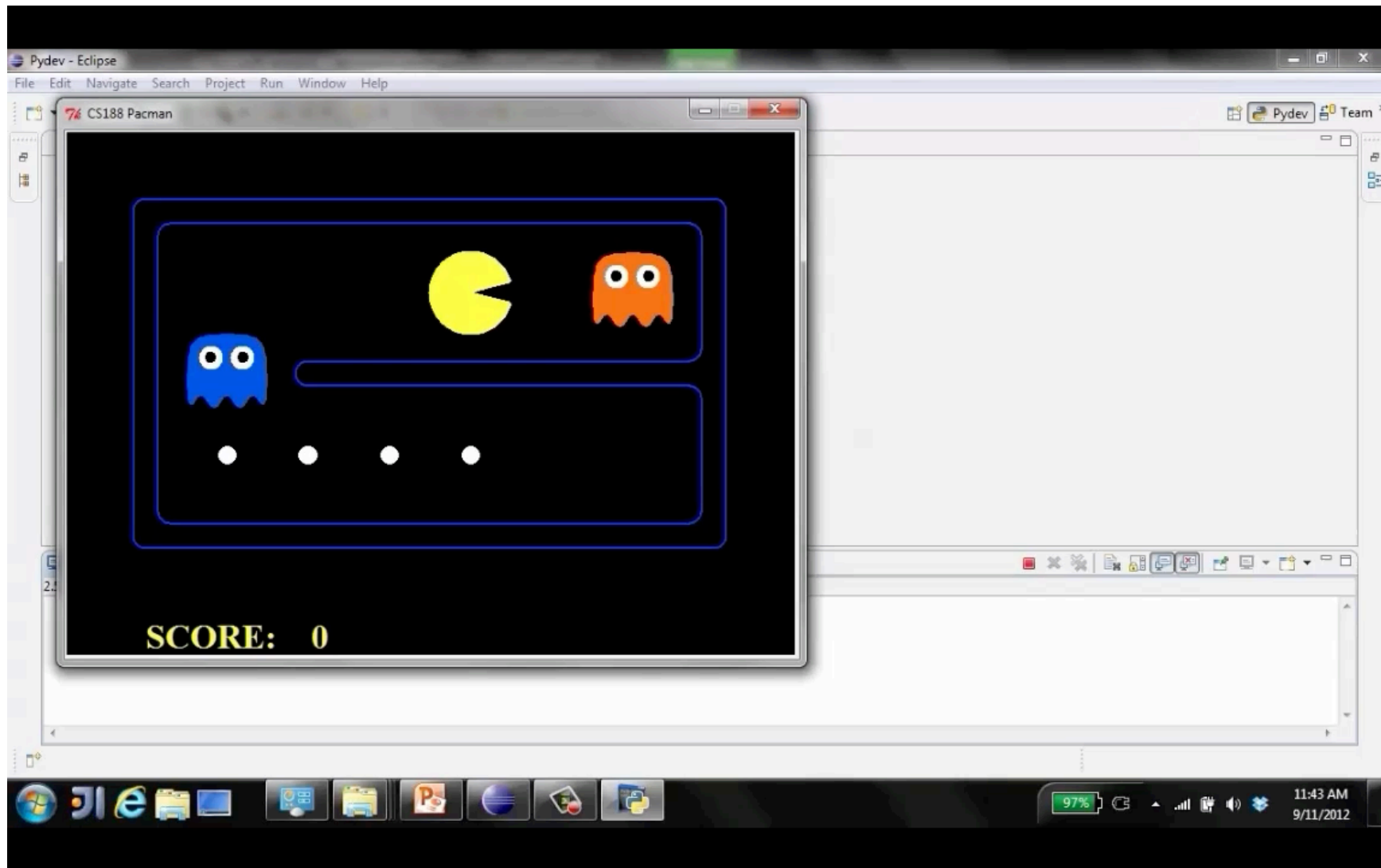Optimal against a perfect player. Otherwise?

Video of Demo Min vs. Exp (Min)

Video of Demo Min vs. Exp (Exp)

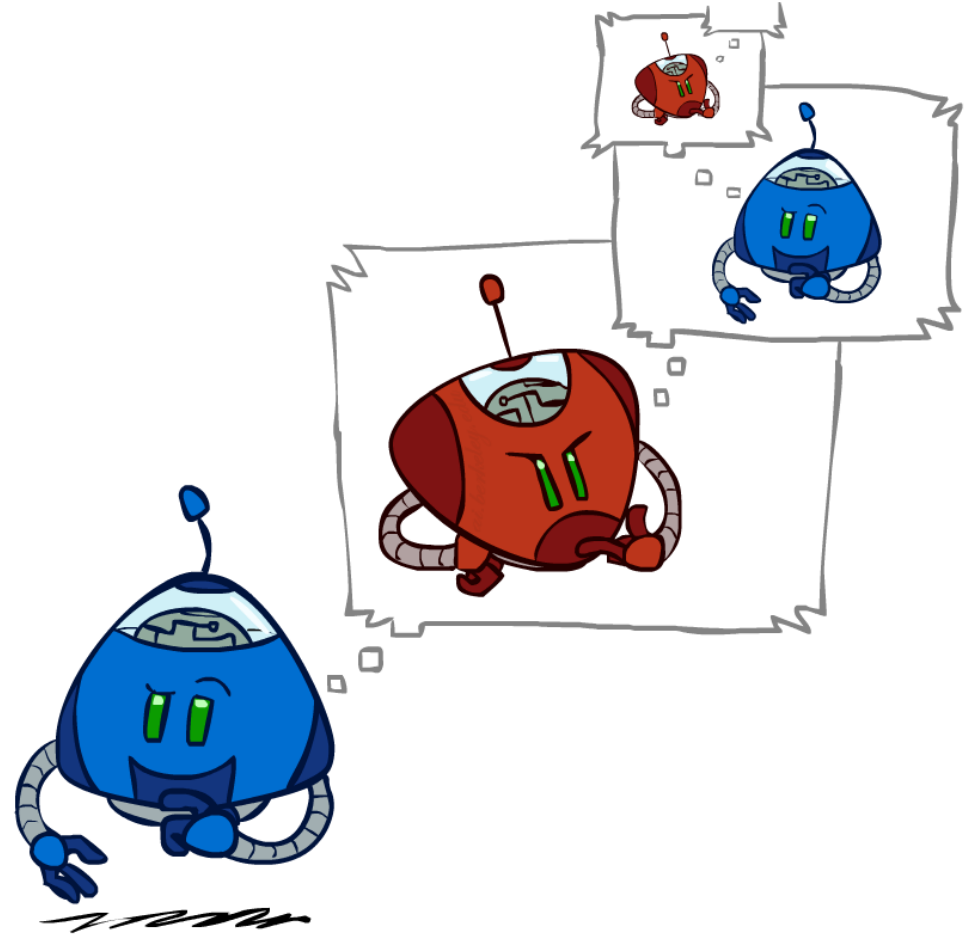# Minimax Efficiency
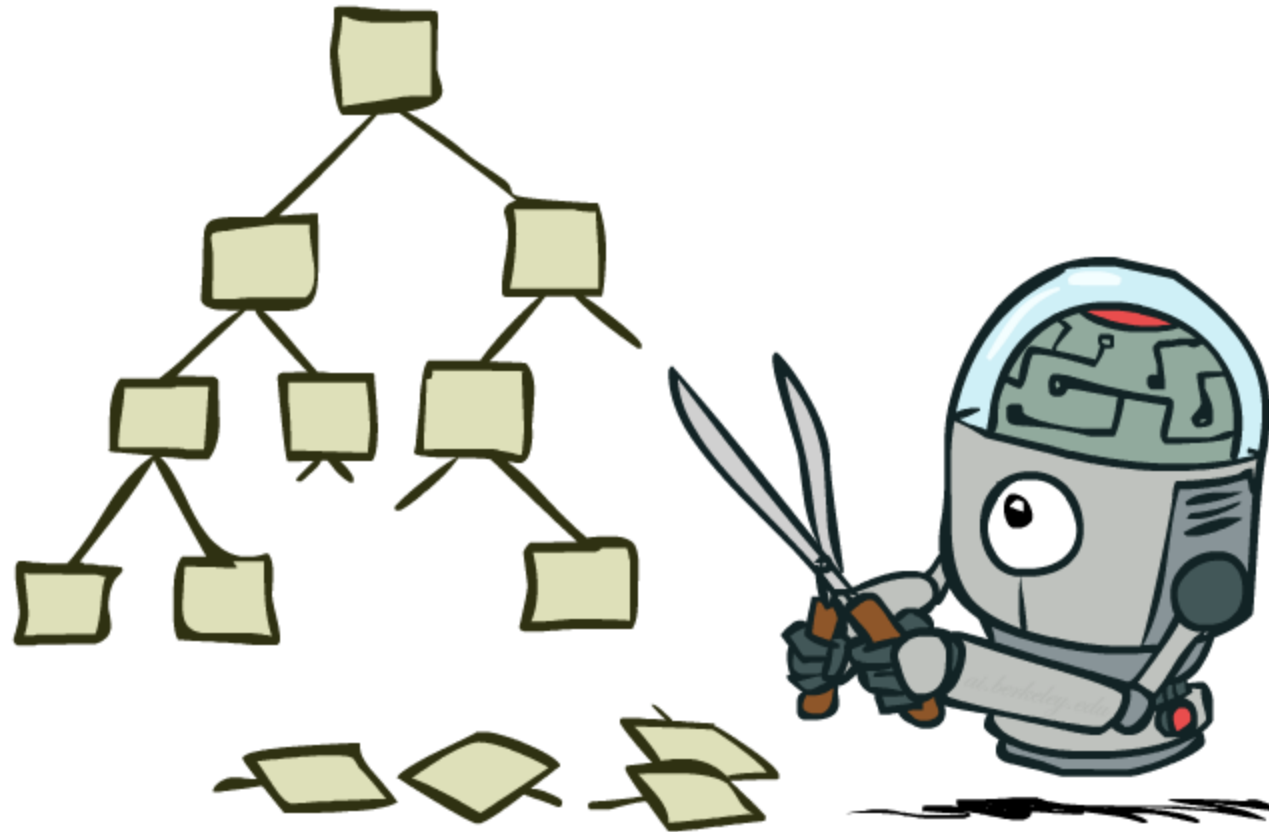
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

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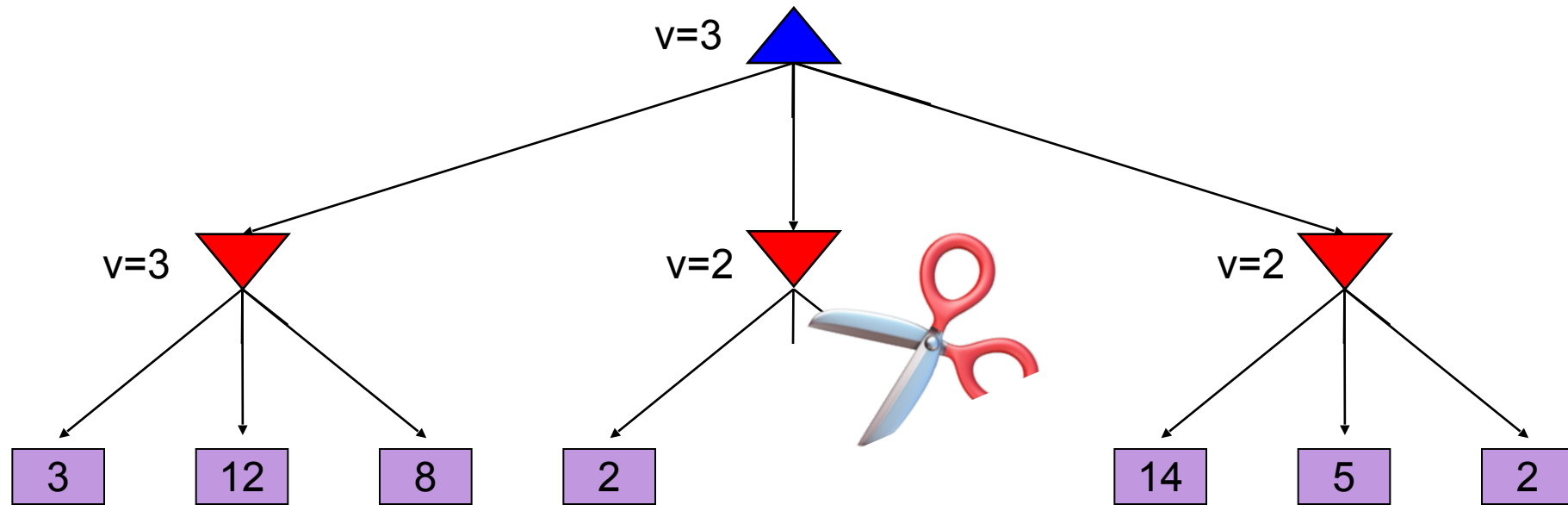 - But, do we need to explore the whole tree?

Resource Limits

Game Tree Pruning

Minimax Pruning

The order of generation matters:
more pruning is possible if good moves come first

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 childrens' 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

MAX

MIN

MAX

MIN
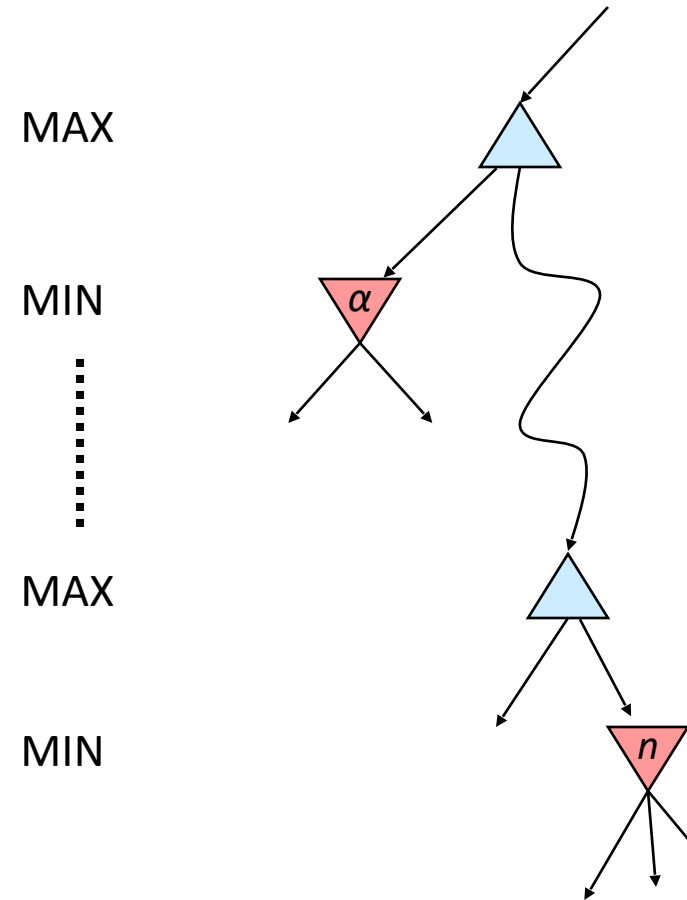
Alpha-Beta Implementation

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)

**max**

min

10   10   0
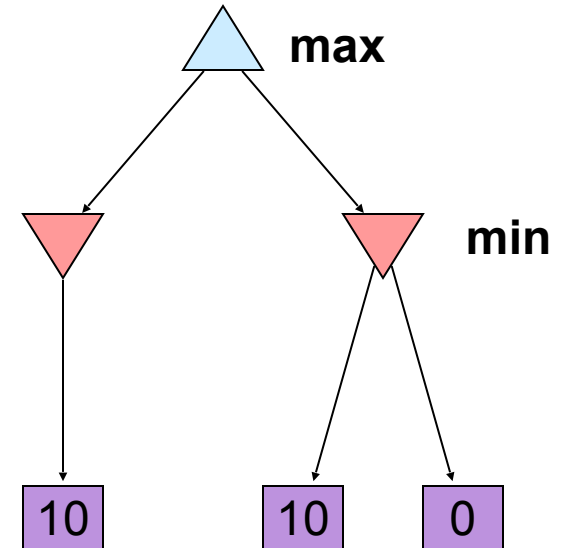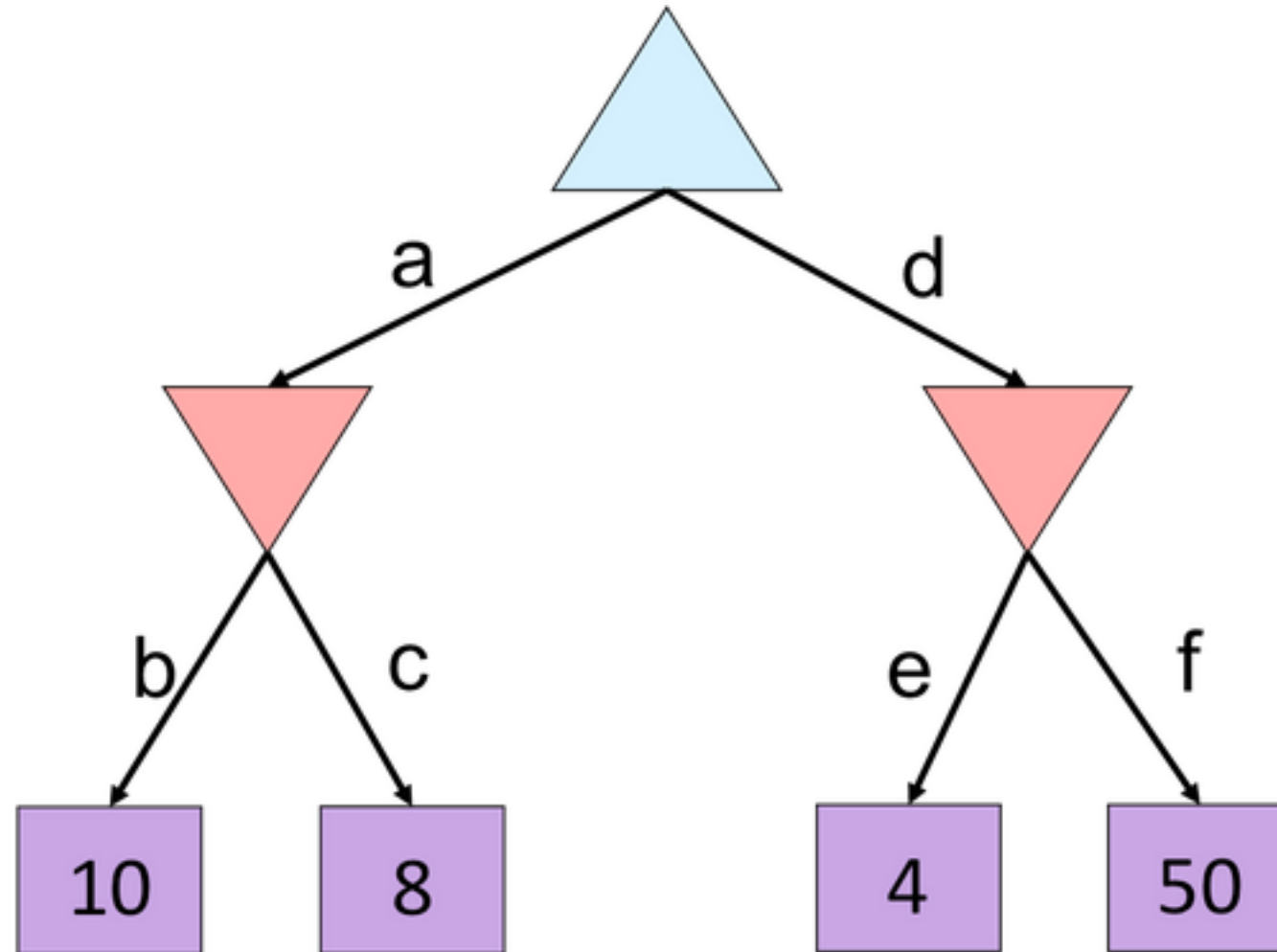
Alpha-Beta Quiz
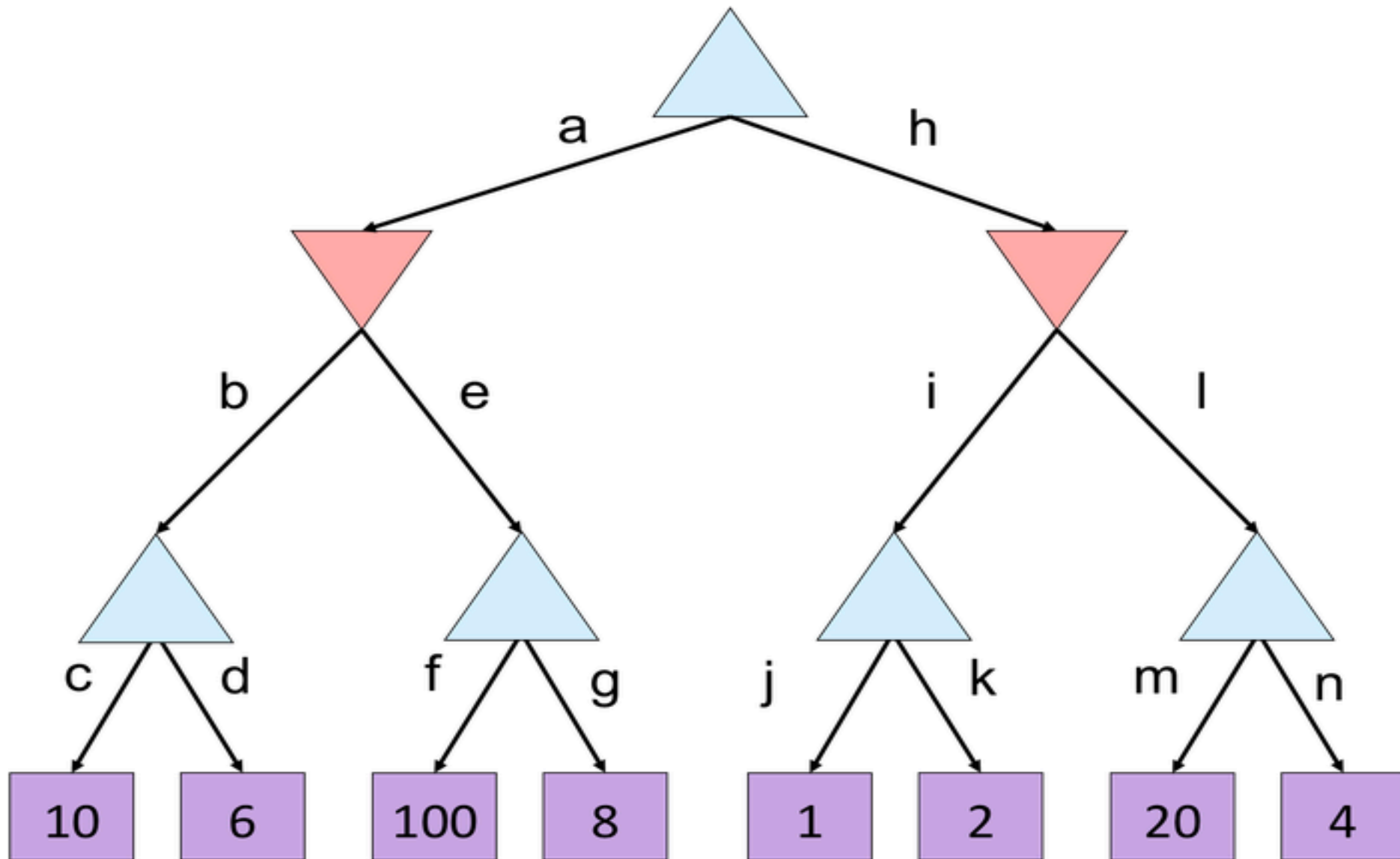
Resource Limits

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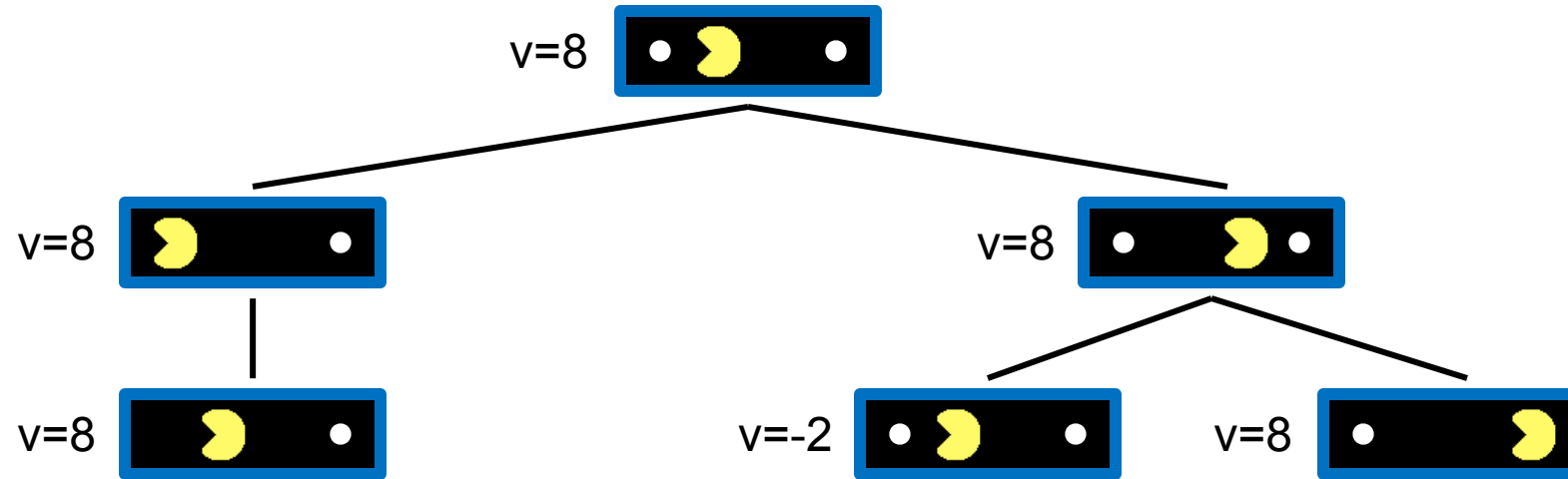
- Use iterative deepening for an anytime algorithm

max

min
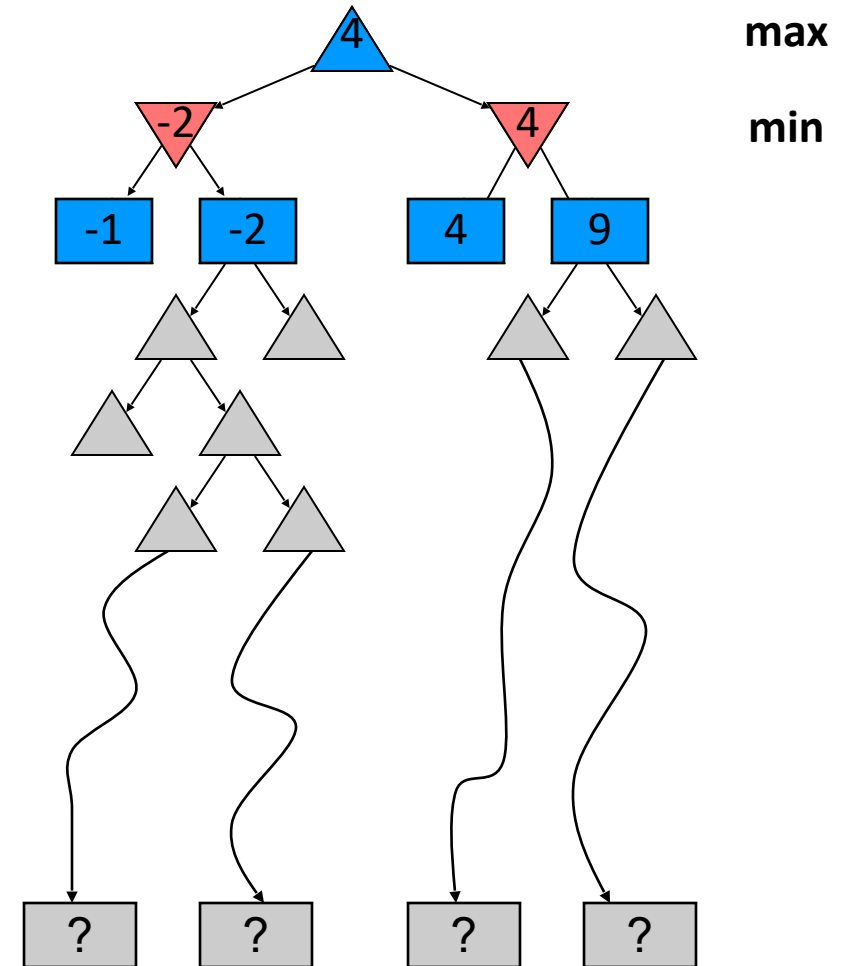
Video of Demo Thrashing (d=2)
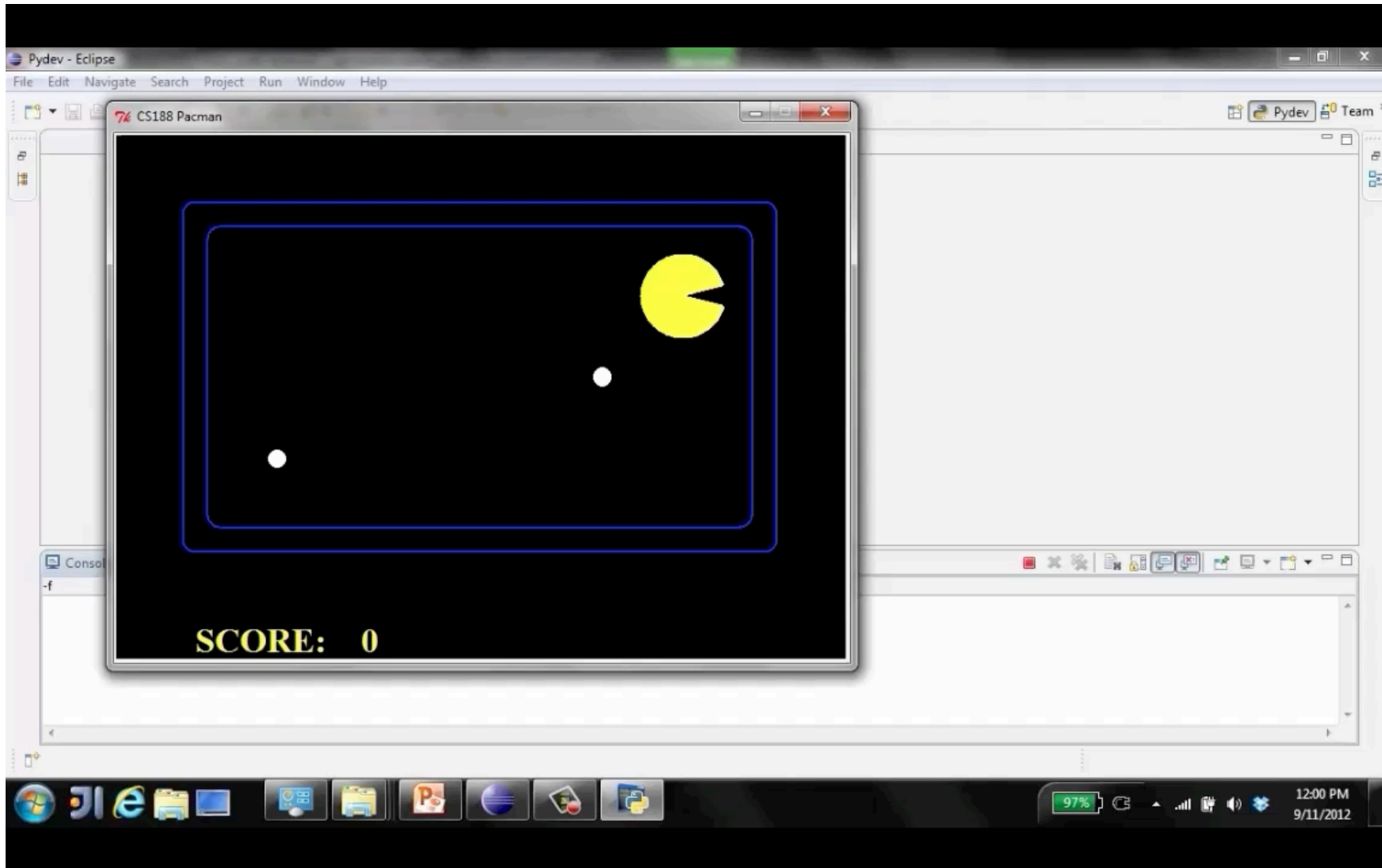
Why Pacman Starves

- ## A danger of replanning agents!
 - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are 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!

Evaluation Functions
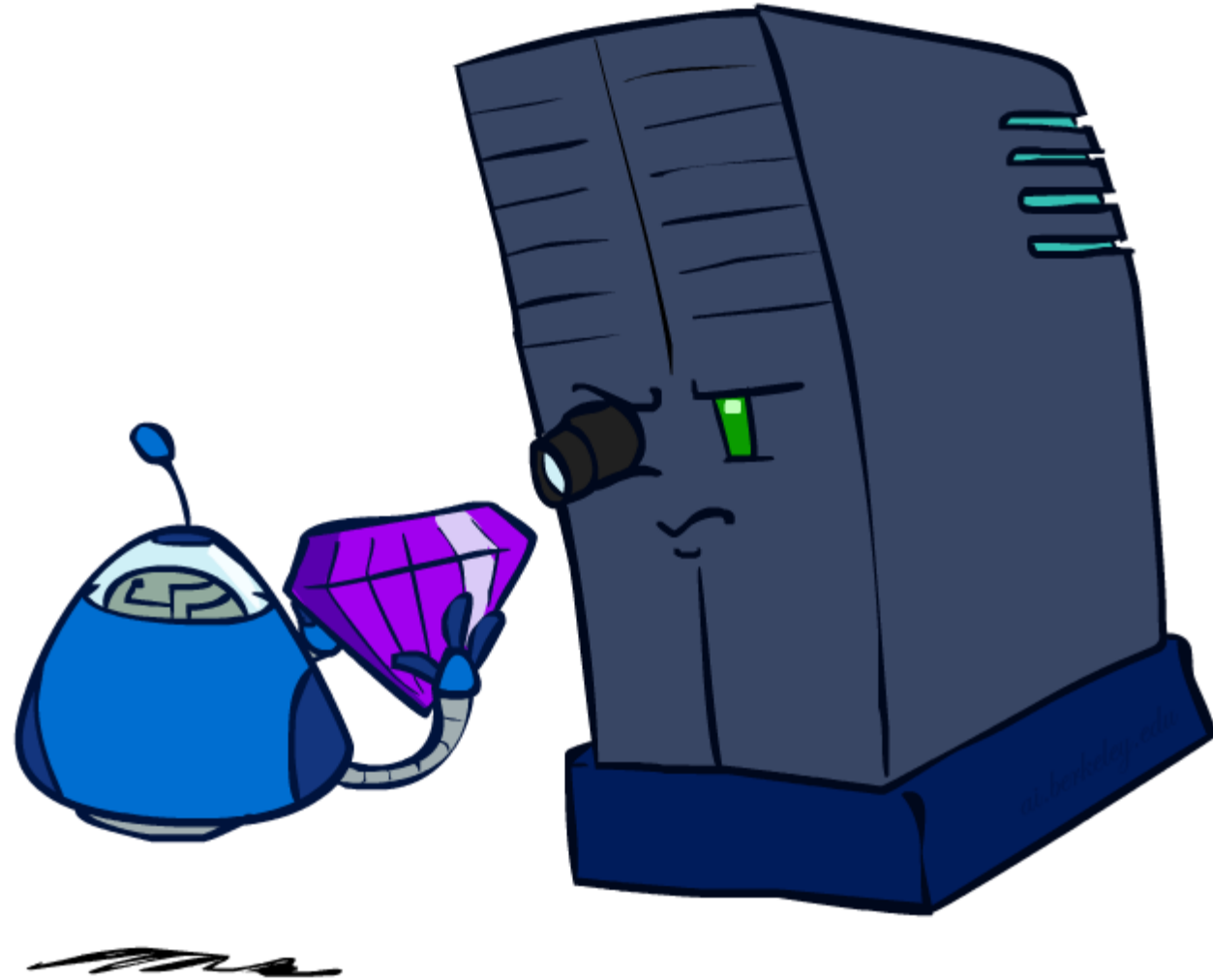
Evaluation Functions

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

Black to move

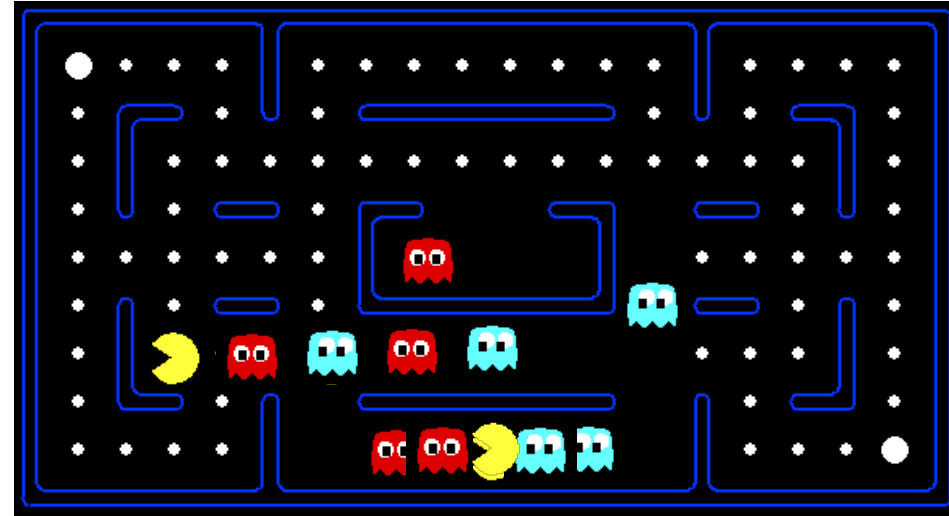White slightly better

White to move

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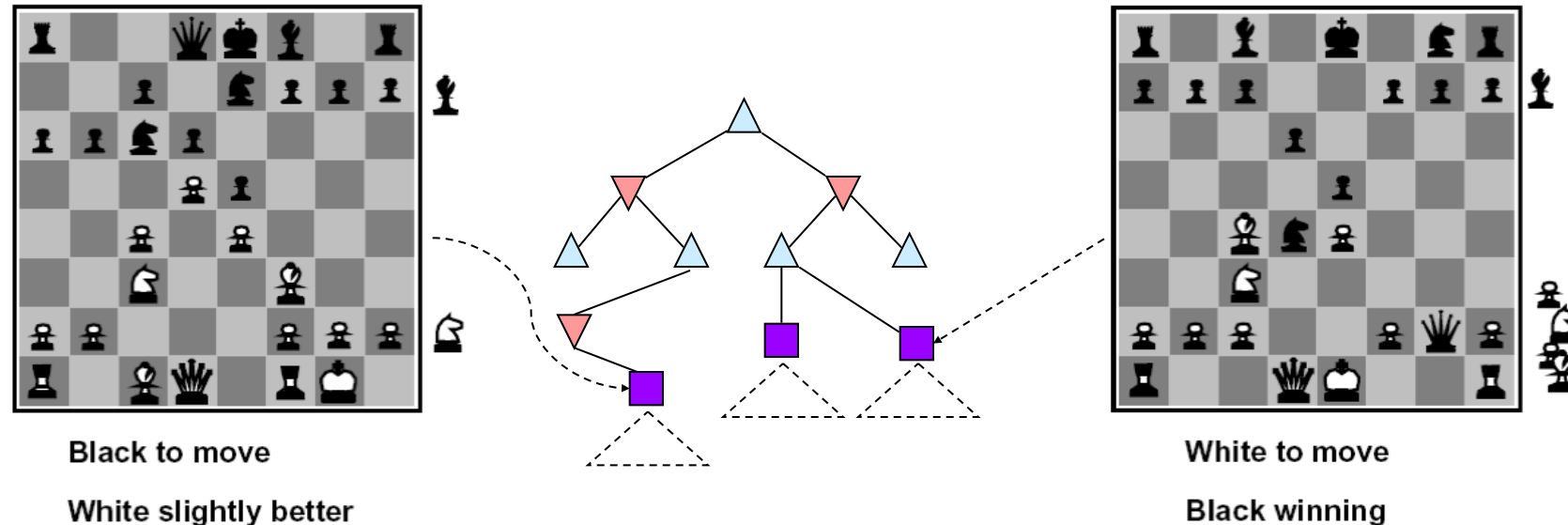
- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

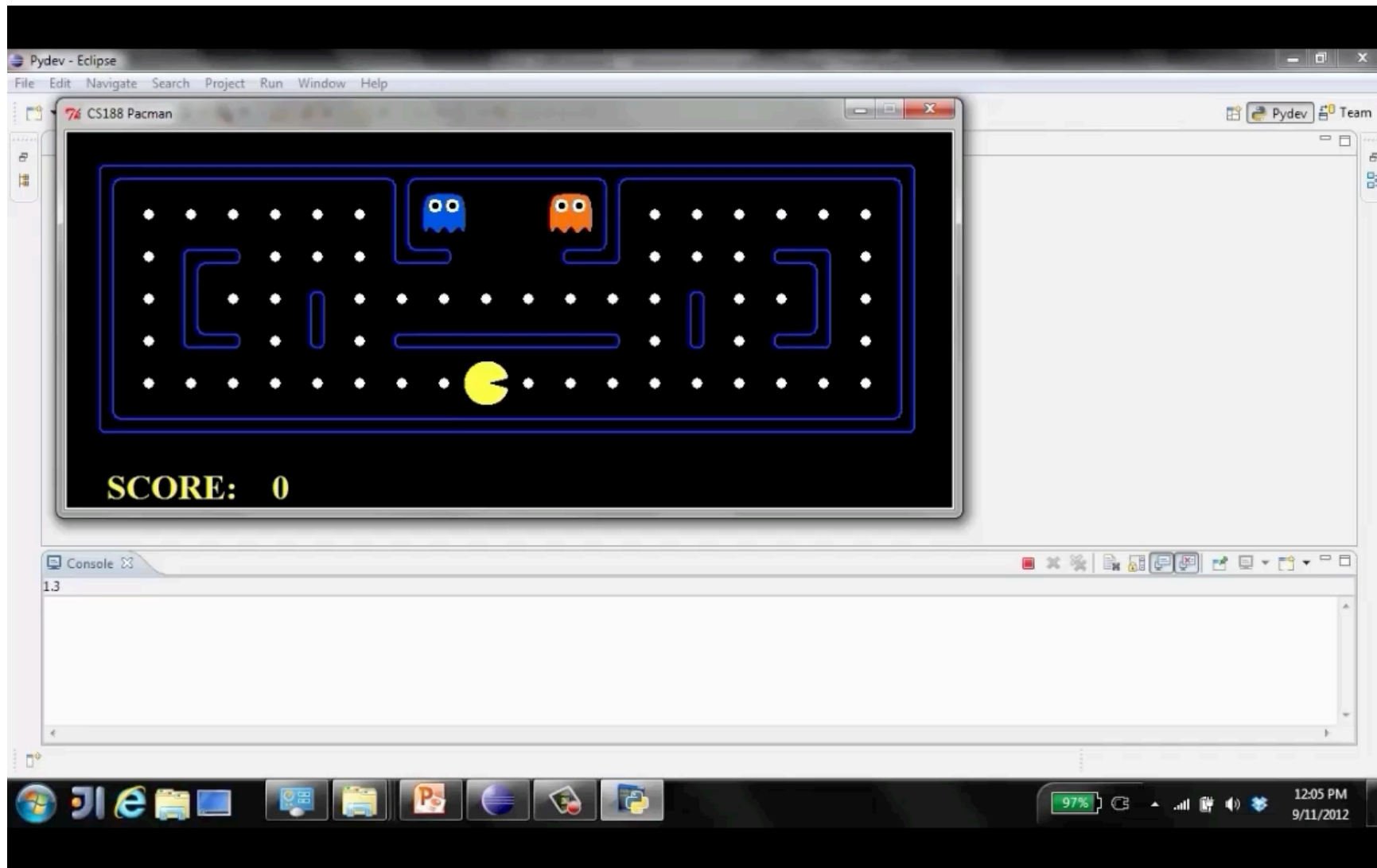$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

  - E.g. $f_1(s)$ = (num white queens – num black queens), etc.
- Or a more complex nonlinear function (e.g., NN) trained by self-play RL

Evaluation for Pacman

Video of Demo Smart Ghosts (Coordination)

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
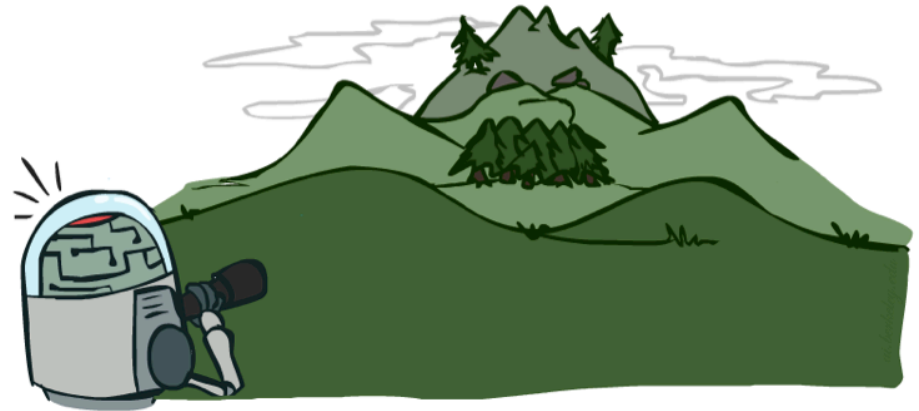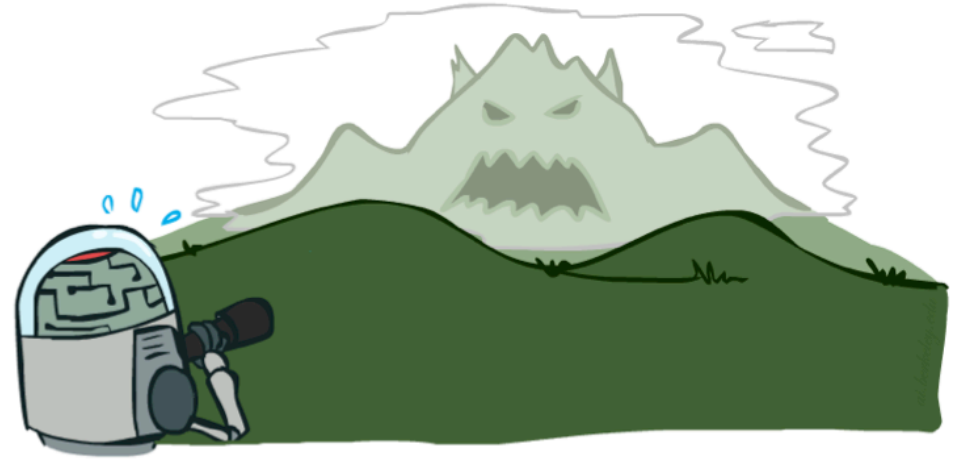
Video of Demo Limited Depth (2)

Video of Demo Limited Depth (10)

Synergies between Evaluation Function and Alpha-Beta?
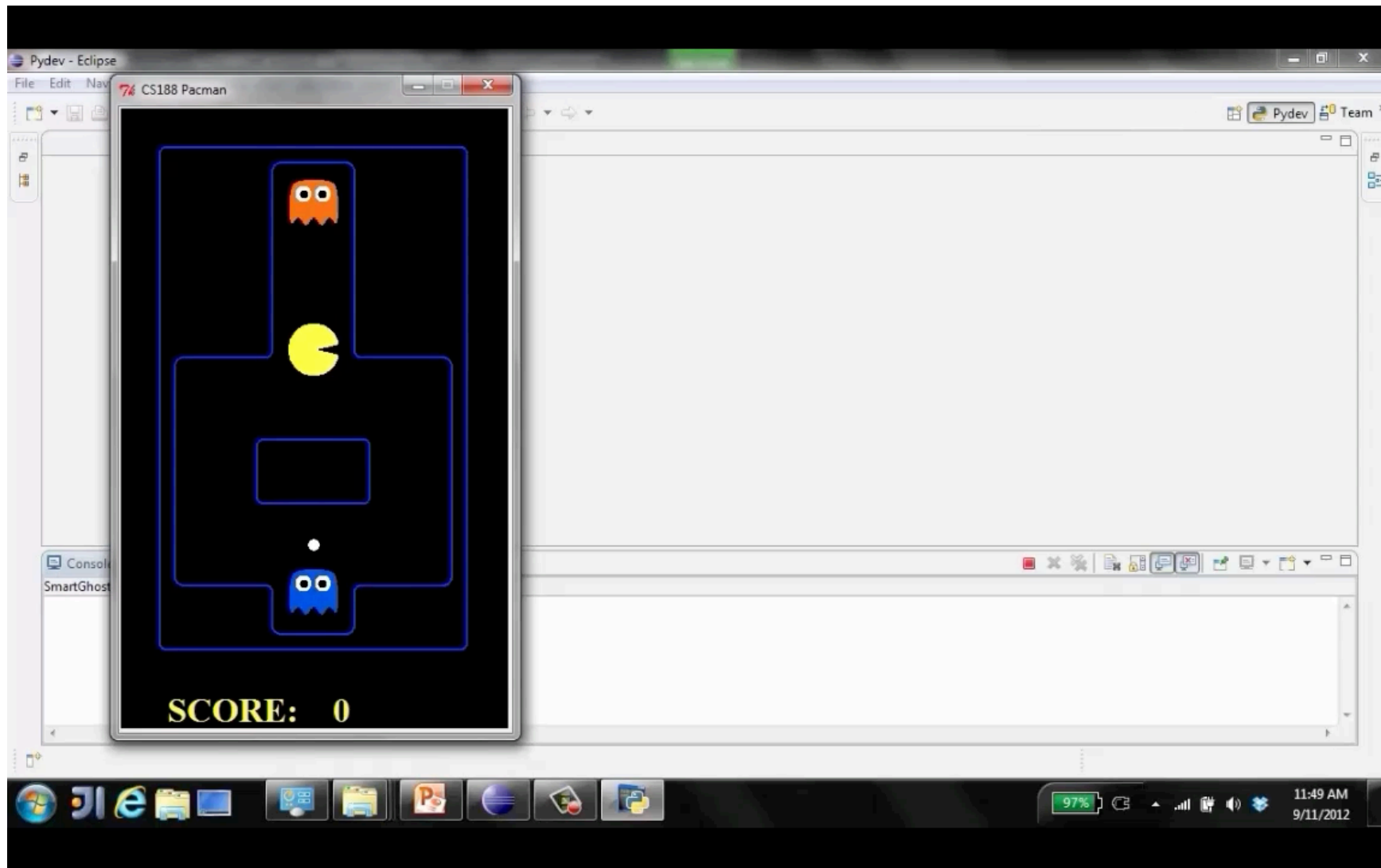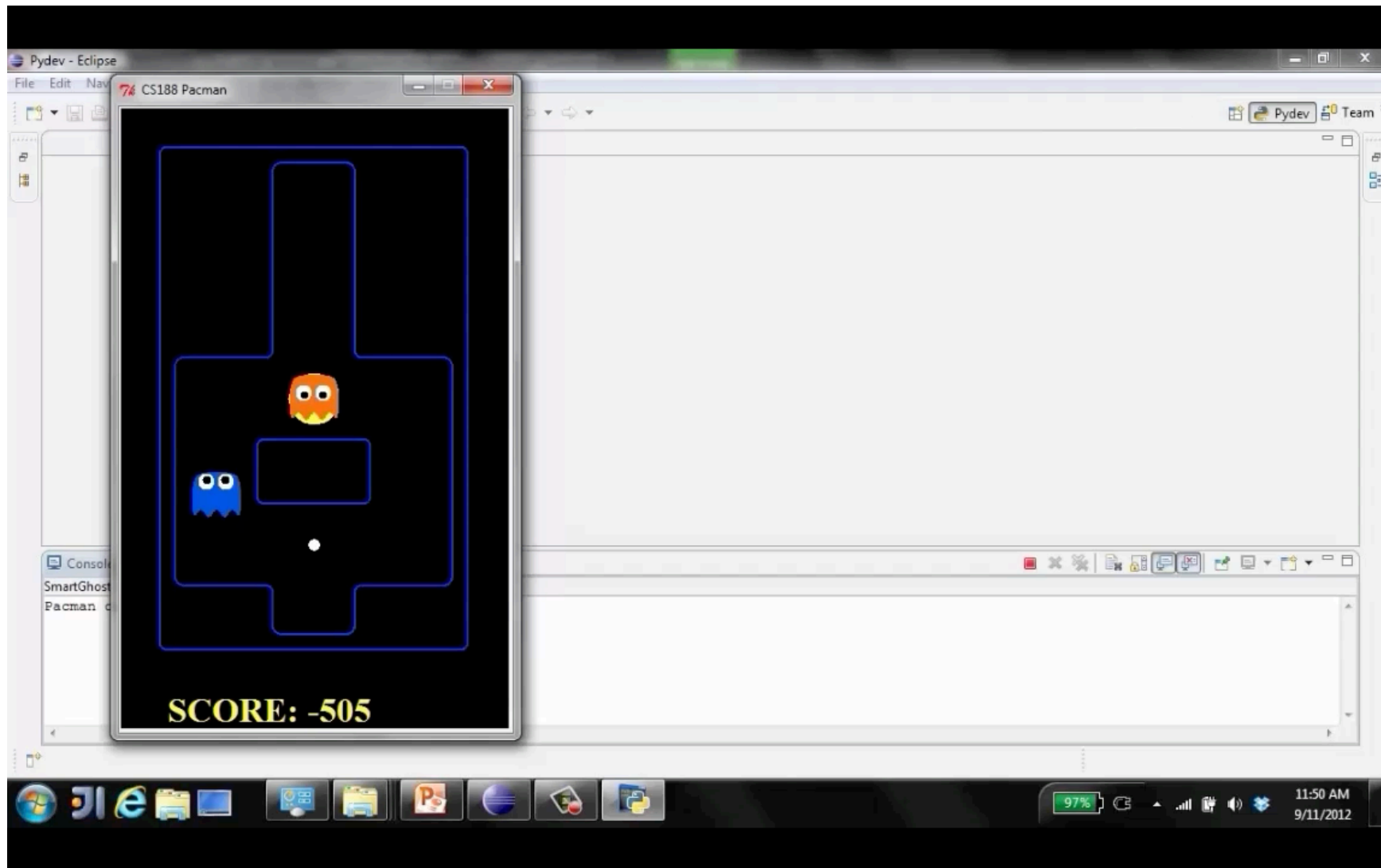
- **Alpha-Beta: amount of pruning depends on expansion ordering**

  - Evaluation function can provide guidance to expand most promising nodes first (which later makes it more likely there is already a good alternative on the path to the root)

    - (somewhat similar to role of A* heuristic, CSPs filtering)

- **Alpha-Beta: (similar for roles of min-max swapped)**

  - Value at a min-node will only keep going down

  - Once value of min-node lower than better option for max along path to root, can prune

  - Hence: IF evaluation function provides upper-bound on value at min-node, and upper-bound already lower than better option for max along path to root
    THEN can prune

Summary

- Games are decision problems with multiple agents
  - Huge variety of issues and phenomena depending on details of interactions and payoffs
- For zero-sum games, optimal decisions defined by minimax
  - Implementable as a depth-first traversal of the game tree
  - Time complexity $O(b^m)$, space complexity $O(bm)$
- Alpha-beta pruning
  - Preserves optimal choice at the root
  - Alpha/beta values keep track of best obtainable values from any max/min nodes on path from root to current node
  - Time complexity drops to $O(b^{m/2})$ with ideal node ordering
- Exact solution is impossible even for "small" games like chess

Next Time: Uncertainty!