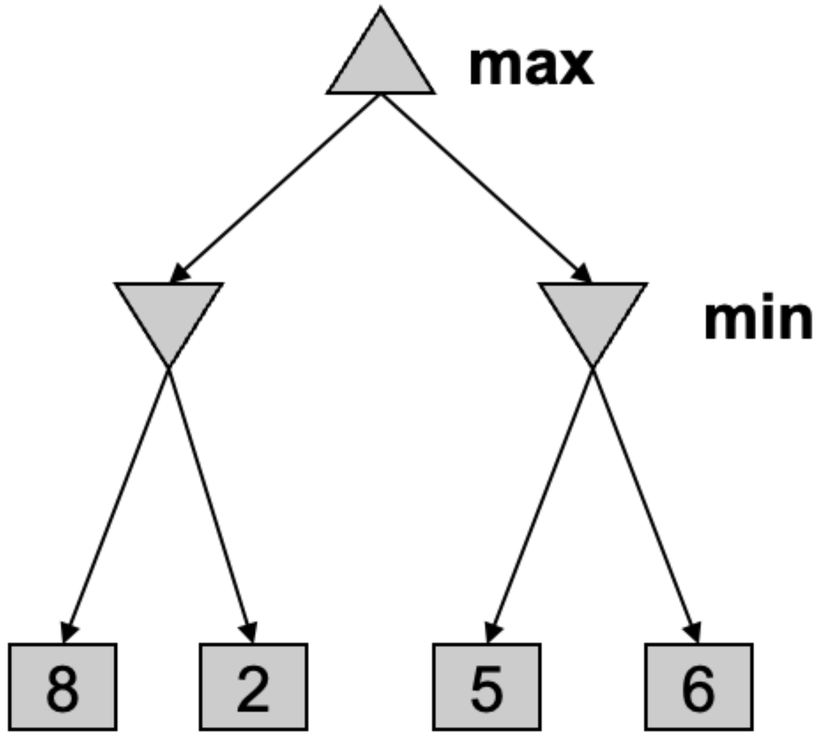
COMPSCI 761: ADVANCED TOPICS IN ARTIFICIAL INTELLIGENCE

ADVERSARIAL SEARCH I

Anna Trofimova, August 2022

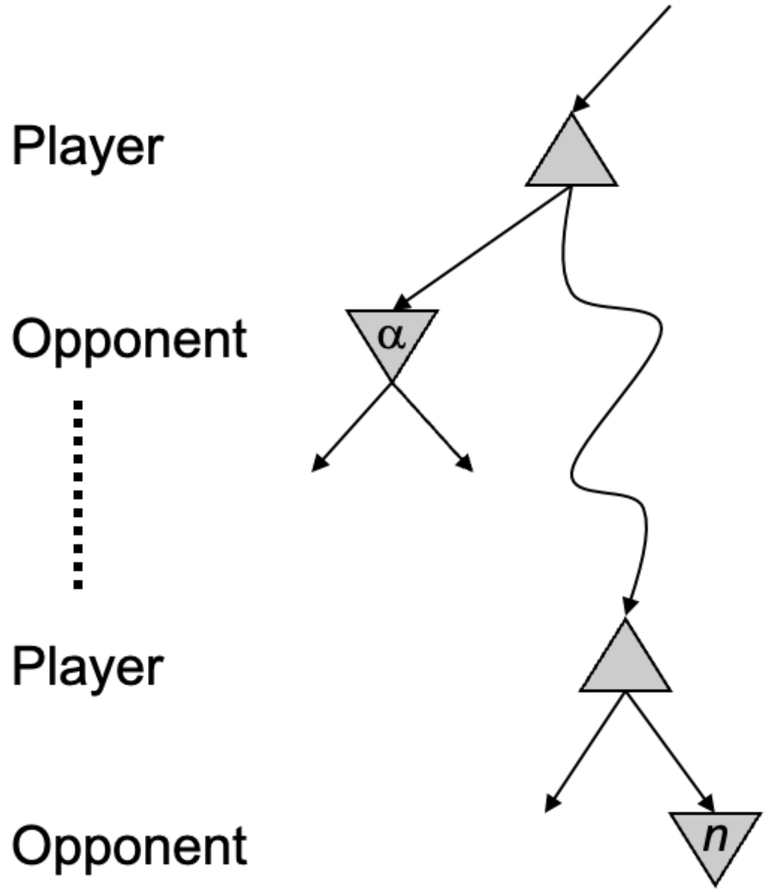
RECAP: DETERMINISTIC TWO-PLAYER

- E.g. tic-tac-toe, chess, checkers
- Minimax search
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves
  - Choose move to position with highest minimax value = best achievable utility against best play
- Zero-sum games
  - One player maximizes result
  - The other minimizes result



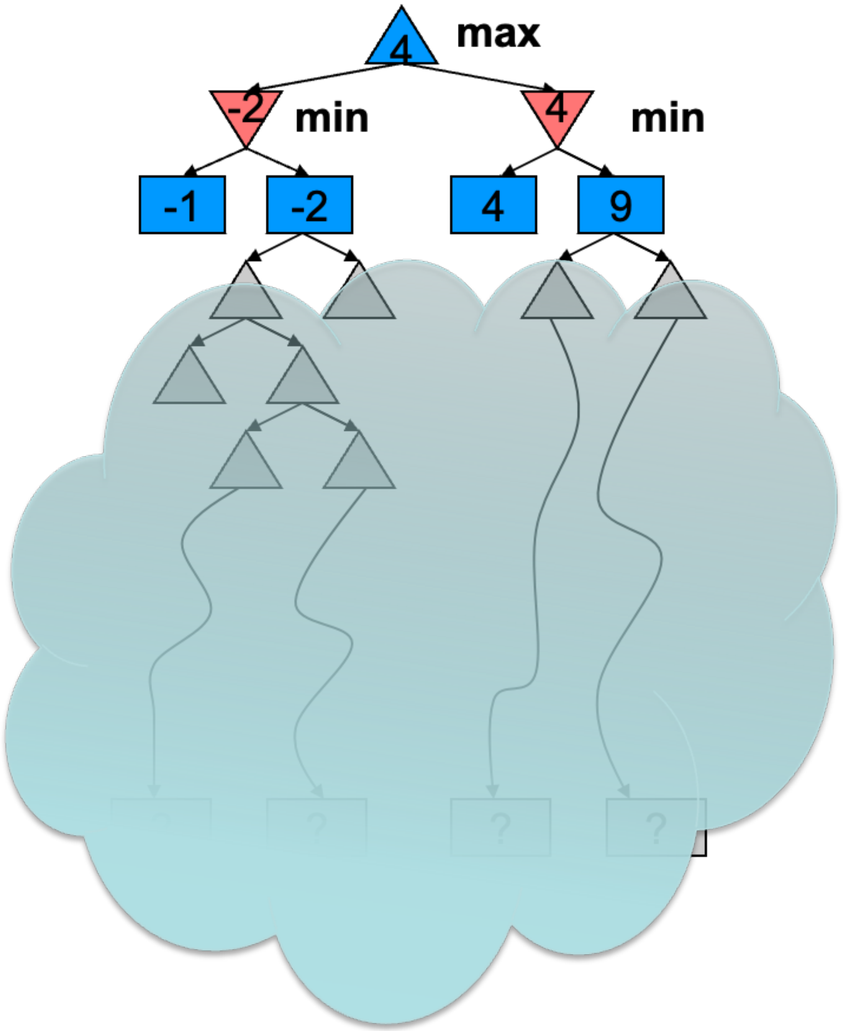
RECAP: *α-β* PRUNING

- 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



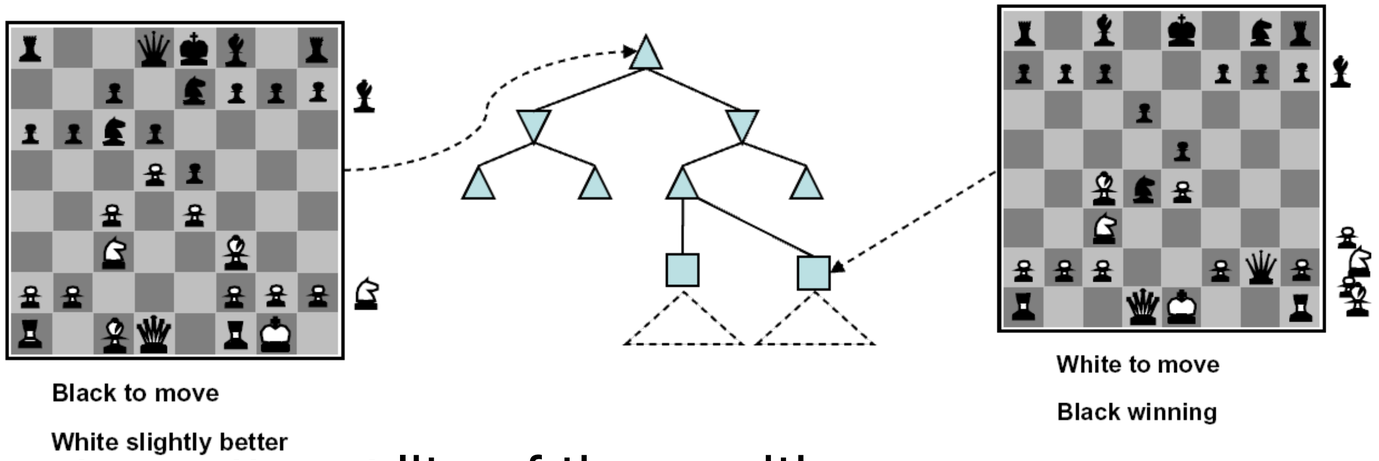
RECAP: RESOURCE LIMITS

- Cannot search to leaves
- Limited search
  - Instead, search a limited depth of the tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- 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



RECAP: EVALUATION FUNCTION

- Function which scores non-terminals

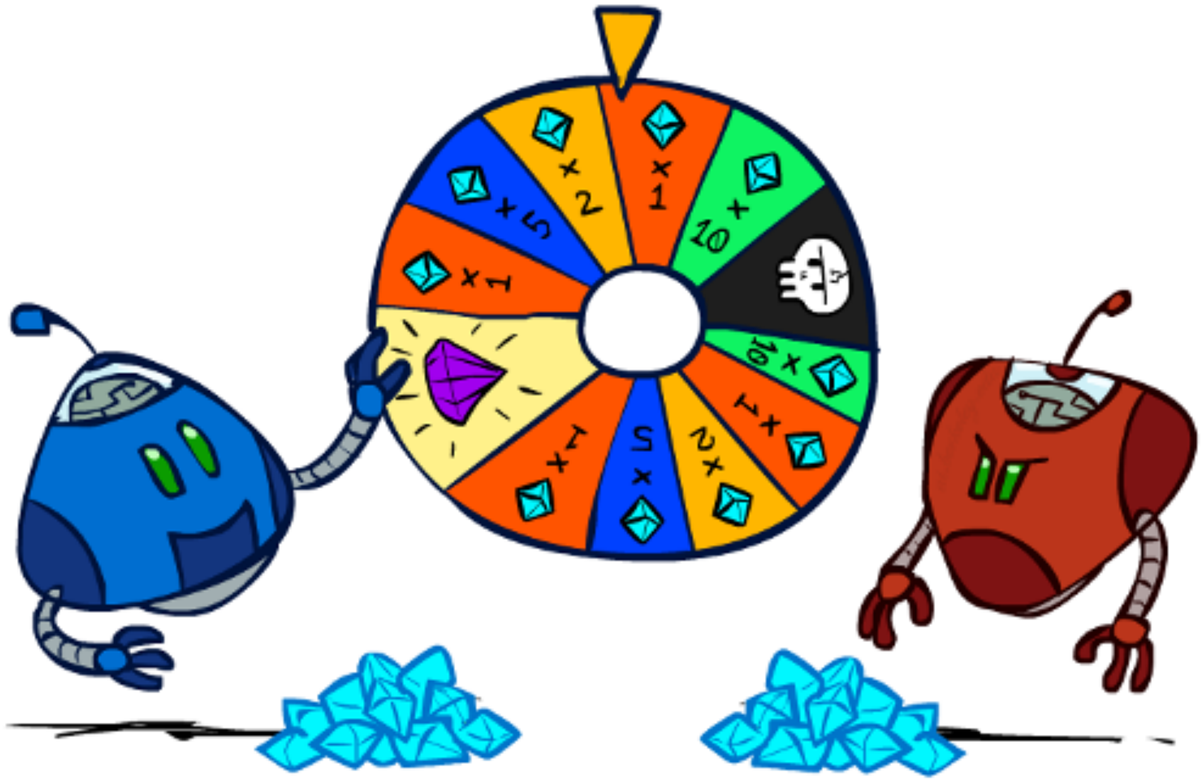


- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:

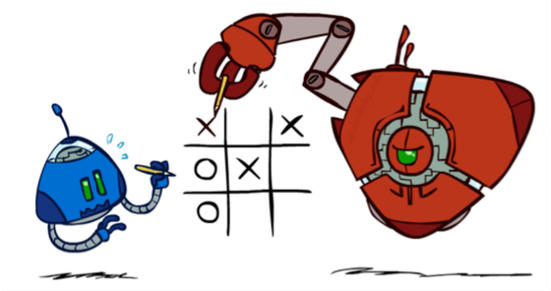
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

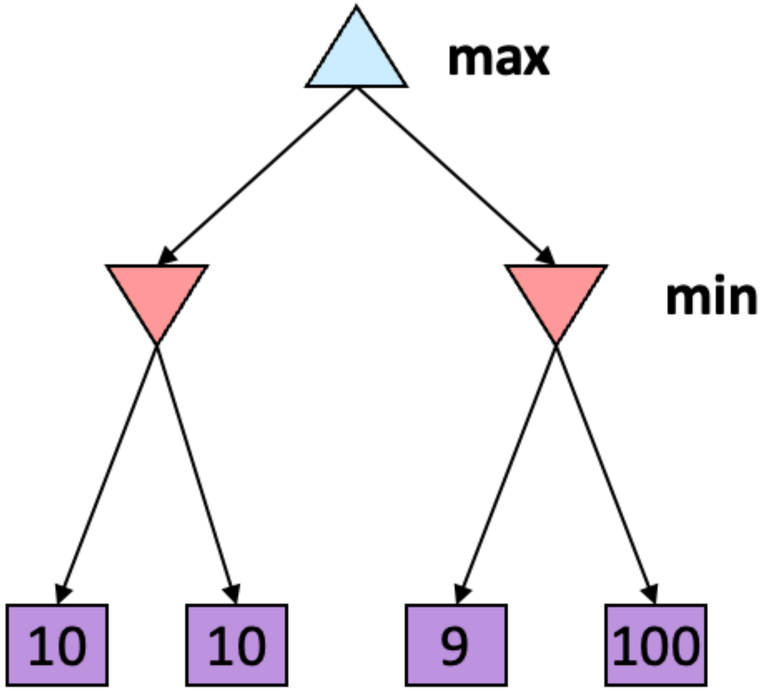
  - e.g. $f_1(s)$ = (num white queens – num black queens), etc.
  - Or a more complex nonlinear function (e.g., NN) trained by self-play RL

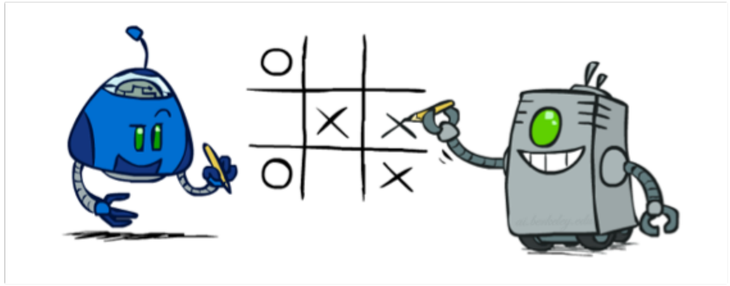
GAMES WITH UNCERTAIN OUTCOMES



WORST-CASE VS. AVERAGE CASE

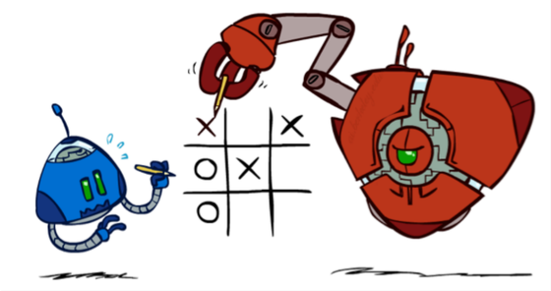


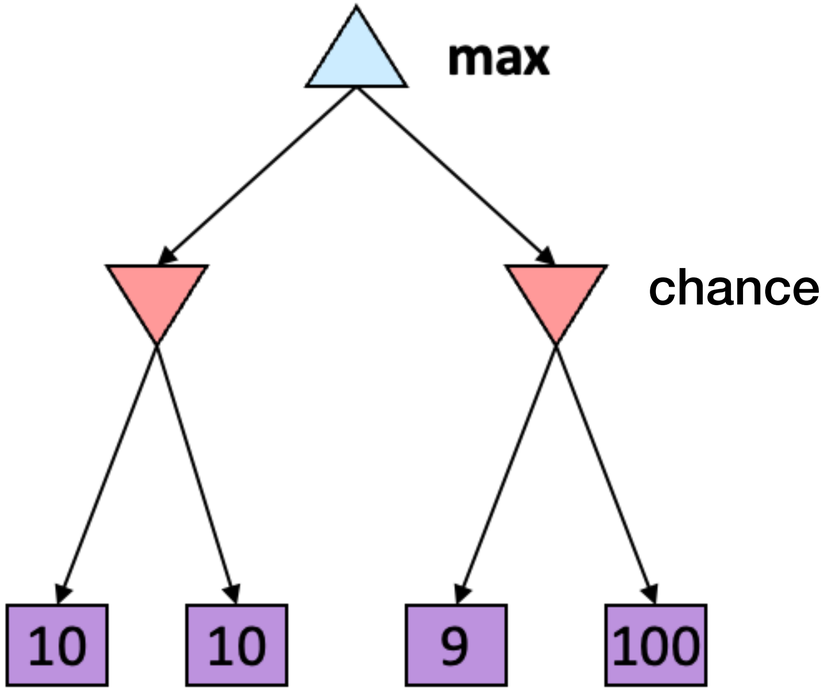


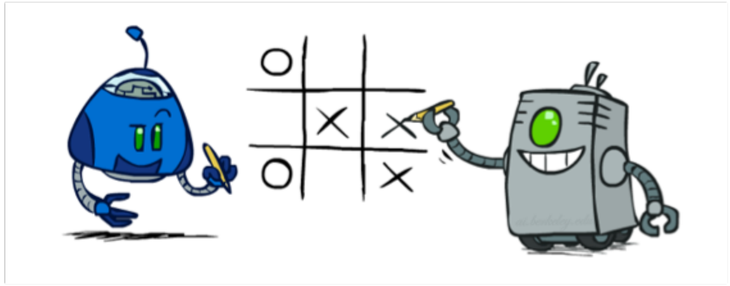


WORST-CASE VS. AVERAGE CASE

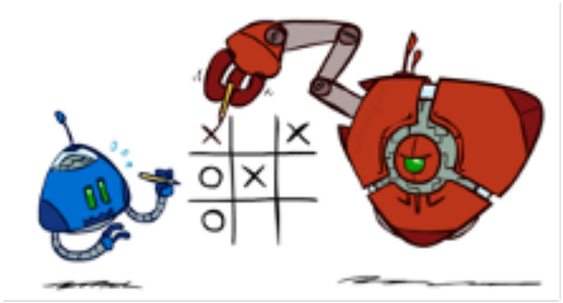
- Idea: uncertain outcomes controlled by chance, not an adversary!

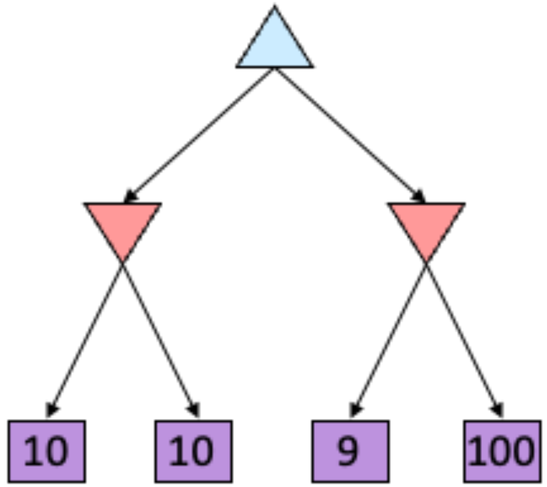




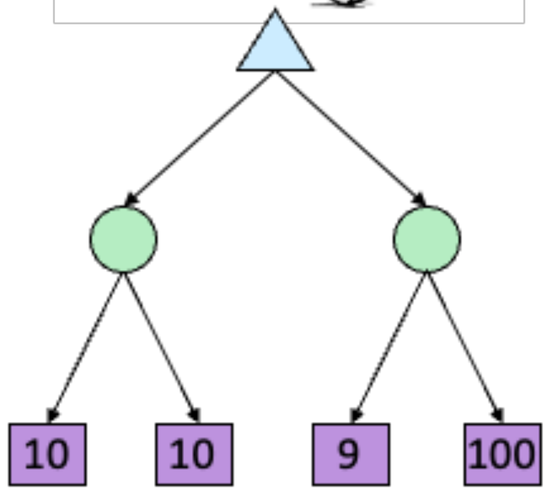


CHANCE OUTCOMES IN TREES

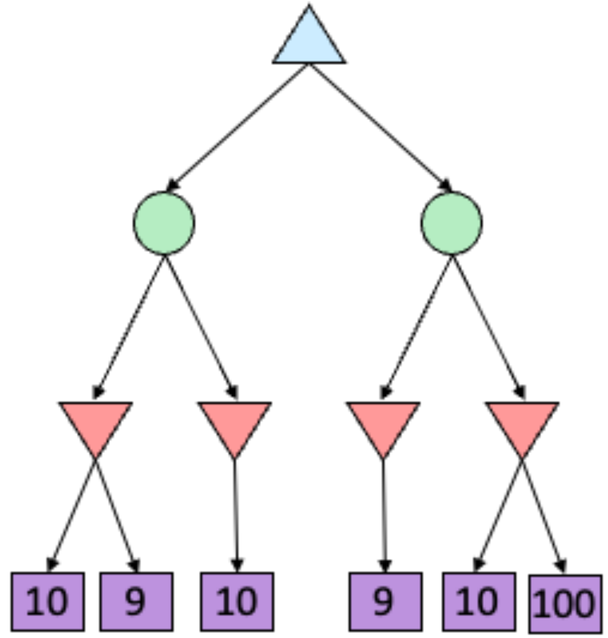




Tictactoe, chess
Minimax



Tetris, investing
Expectimax



Backgammon, Monopoly
Expectiminimax

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

MINIMAX SEARCH

function minimax-decision(s) returns an action

return the action a in Actions(s) with the highest minimax_value(Result(s,a))

function minimax_value(s) returns a value

if Terminal-Test(s) then return Utility(s)

if Player(s) = MAX then return $\max_{a \text{ in Actions(s)}}$ minimax_value(Result(s,a))

if Player(s) = MIN then return $\min_{a \text{ in Actions(s)}}$ minimax_value(Result(s,a))

EXPECTIMAX SEARCH

function decision(s) returns an action

return the action a in Actions(s) with the highest value(Result(s,a))

function value(s) returns a value

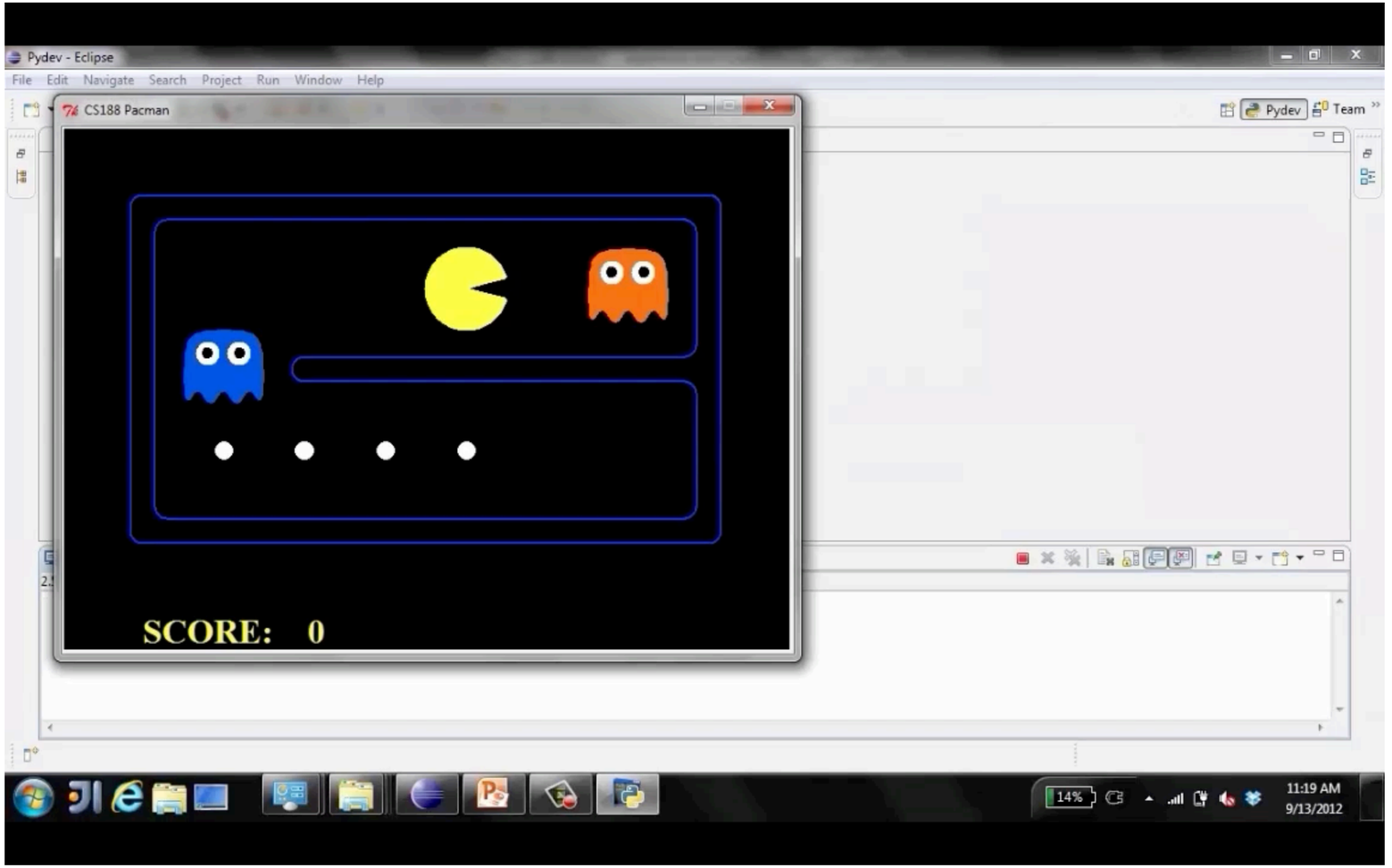
if Terminal-Test(s) then return Utility(s)

if Player(s) = MAX then return $\max_{a \text{ in Actions}(s)}$ value(Result(s,a))

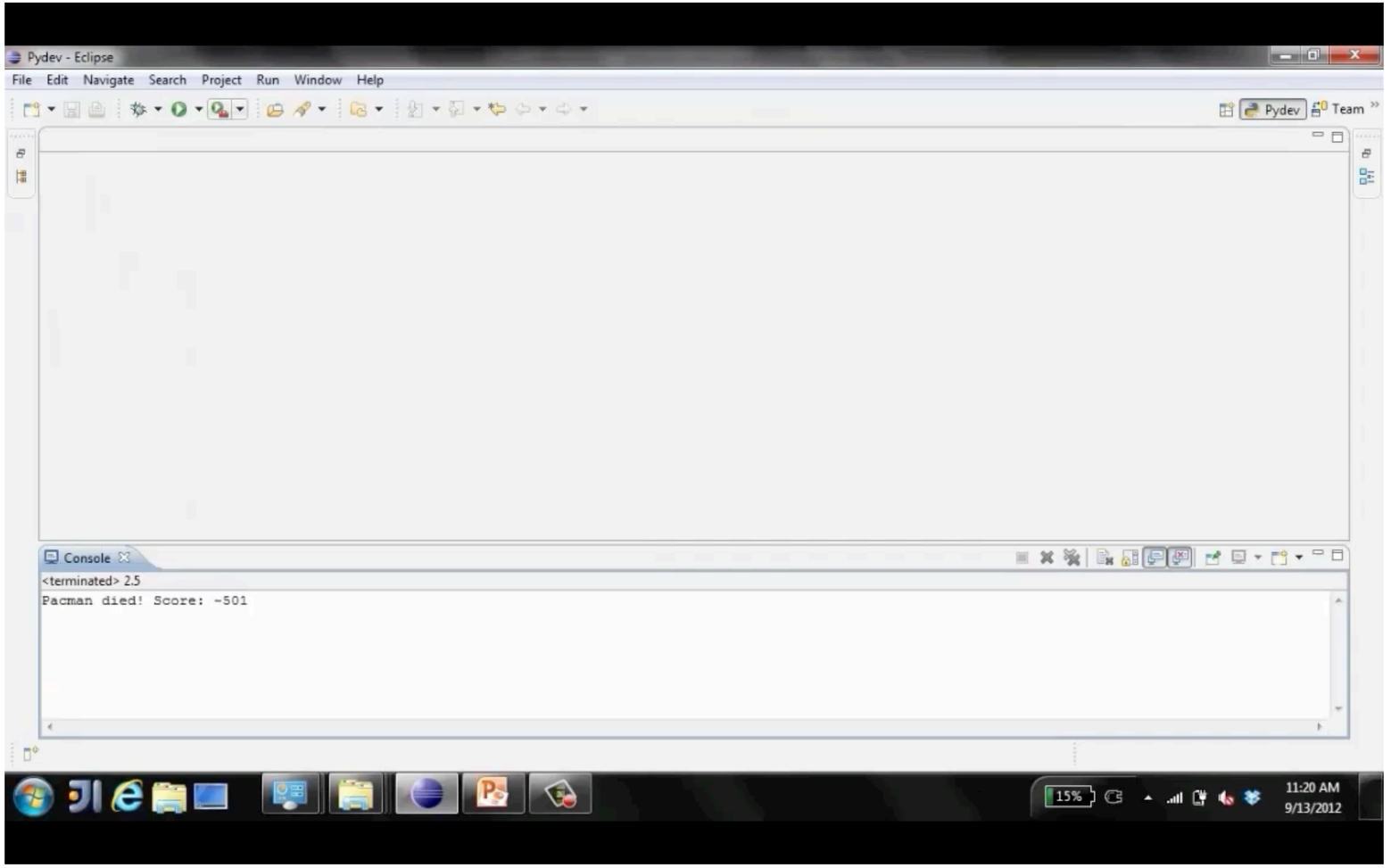
if Player(s) = MIN then return $\min_{a \text{ in Actions}(s)}$ value(Result(s,a))

if Player(s) = CHANCE then return $\text{sum}_{r \text{ in chanceEvent}(s)}$ Pr(r) * value(Result(s,r))

DEMO MINIMAX VS EXPECTIMAX

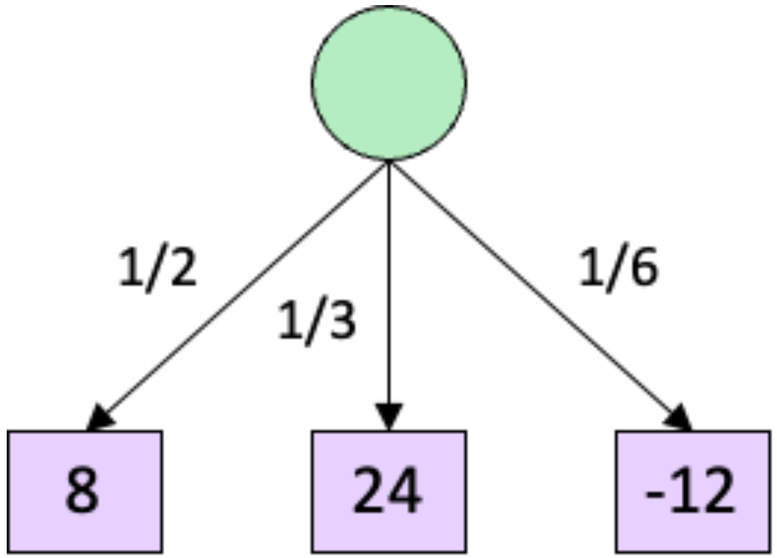


DEMO MINIMAX VS EXPECTIMAX



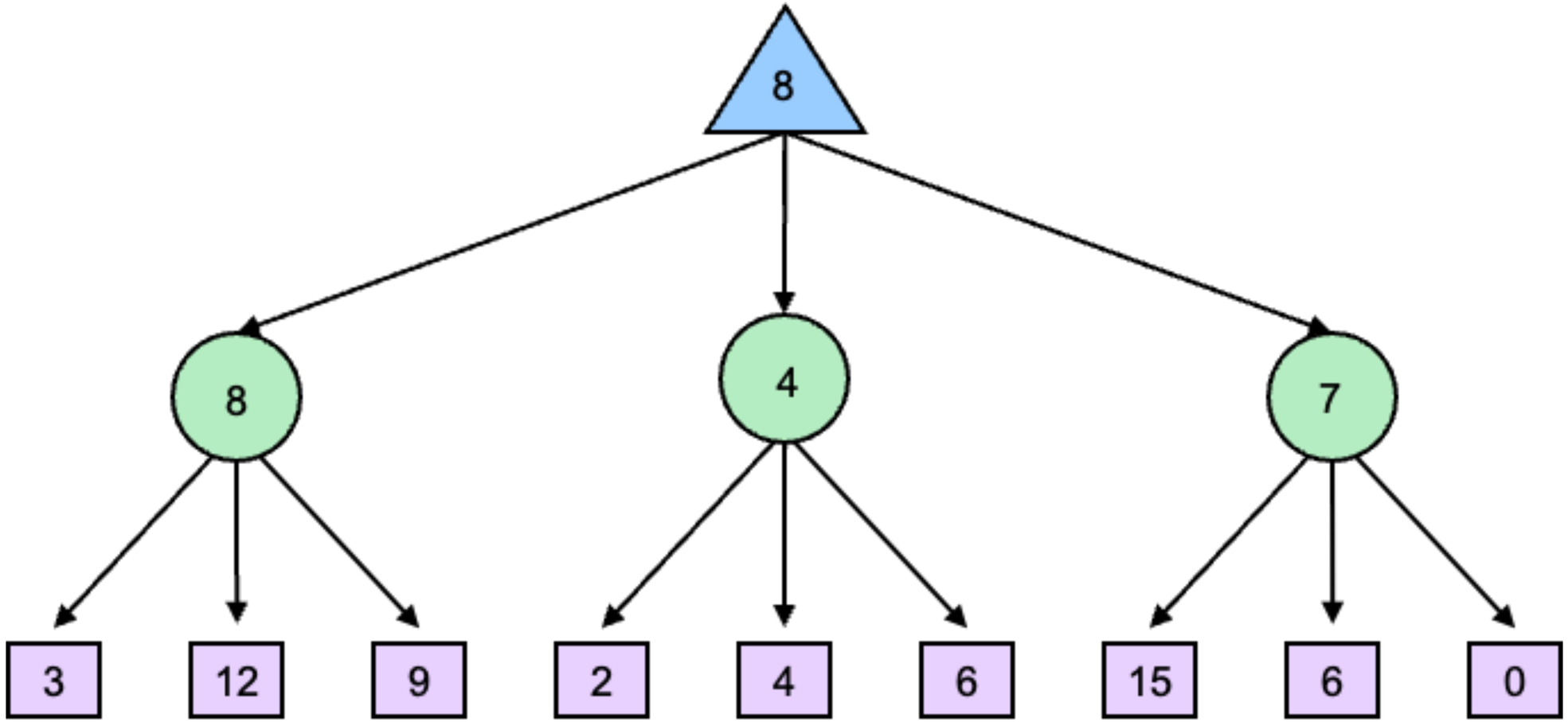
EXPECTIMAX PSEUDOCODE

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

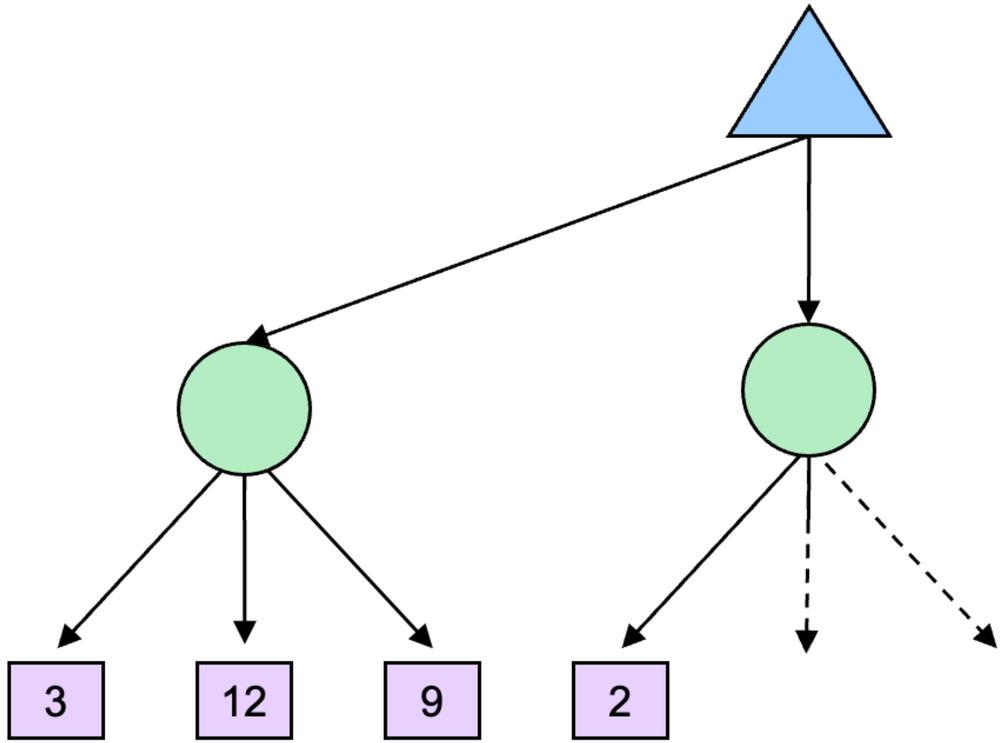


$$v = (1/2)\ (8) + (1/3)\ (24) + (1/6)\ (-12) = 10$$

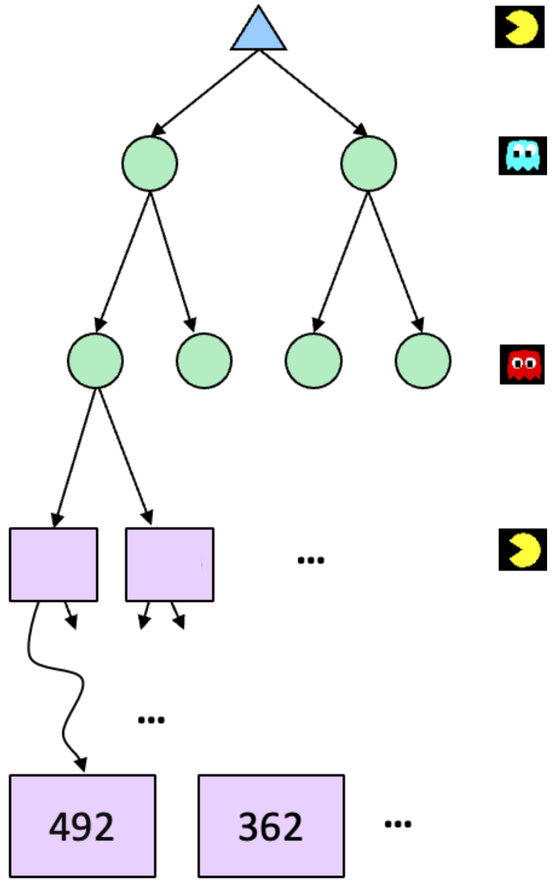
EXPECTIMAX EXAMPLE



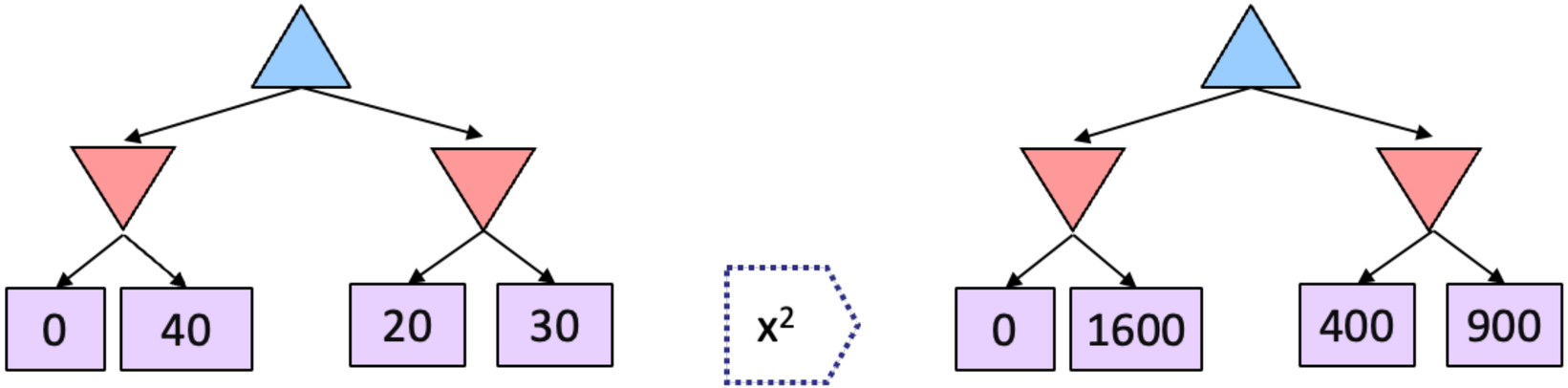
EXPECTIMAX PRUNING?



DEPTH-LIMITED EXPECTIMAX

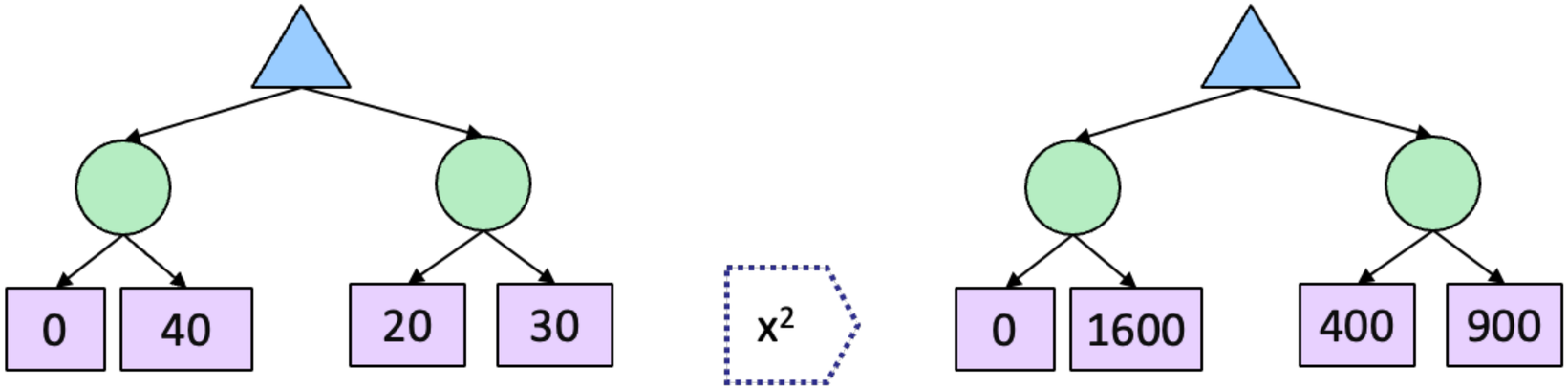


WHAT VALUES TO USE?



- For worst-case minimax reasoning, evaluation function scale doesn't matter
  - We just want better states to have higher evaluations (get the ordering right)
  - Minimax decisions are *invariant with respect to monotonic transformations on values*
  - x>y ⇒ f(x) > f(y)

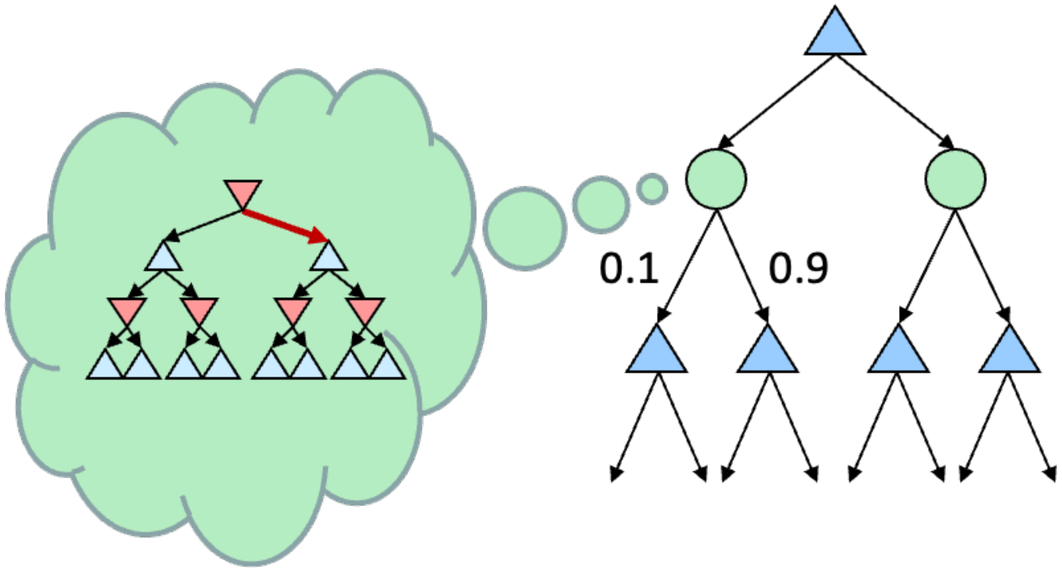
WHAT VALUES TO USE?



- For worst-case minimax reasoning, evaluation function scale doesn't matter
  - We just want better states to have higher evaluations (get the ordering right)
  - Minimax decisions are *invariant with respect to monotonic transformations on values*
  - $x > y \Rightarrow f(x) > f(y)$
- Expectiminimax decisions are *invariant with respect to positive affine transformations*
- *$f(x) = Ax + B$ where $A > 0$*
- Expectiminimax evaluation functions have to be aligned with actual win probabilities!

QUIZ

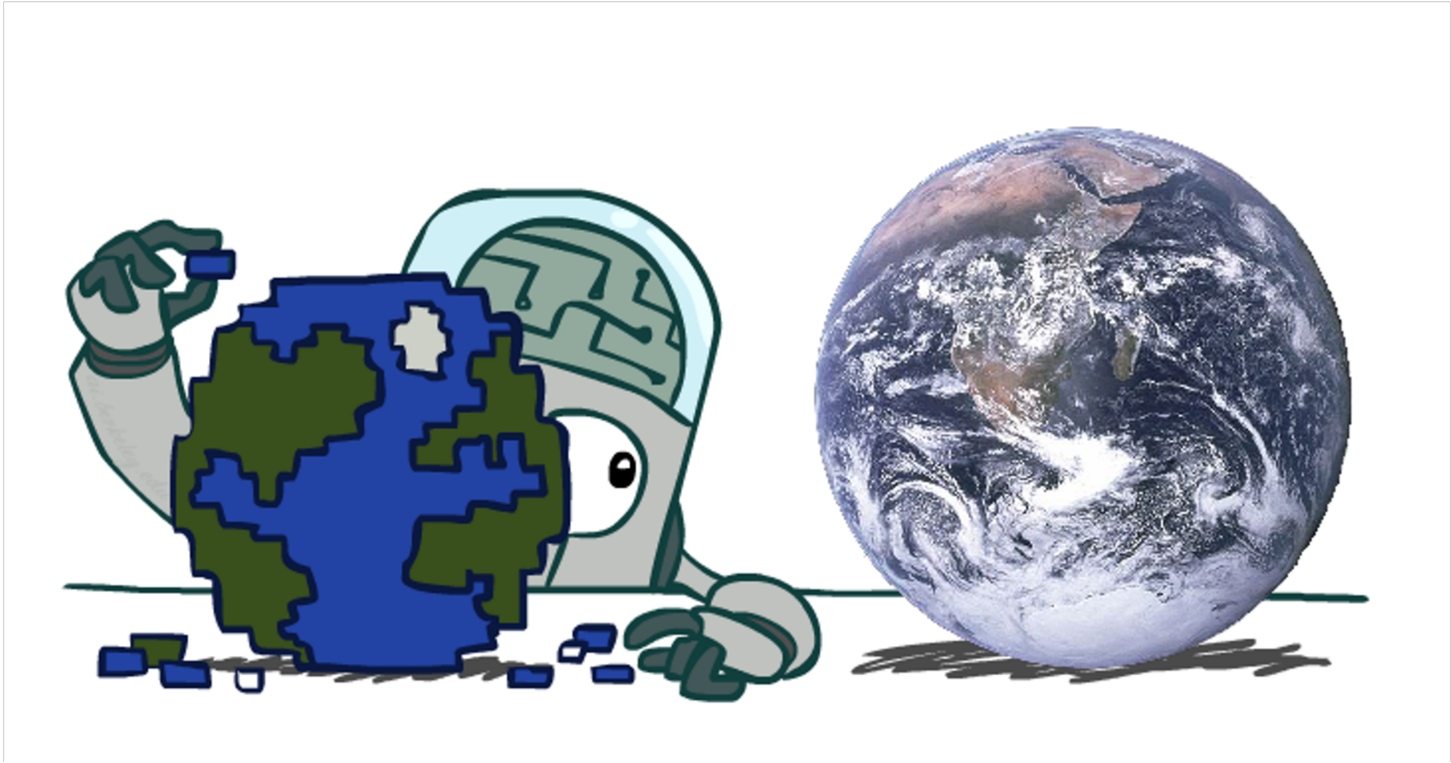
- Let's say you know that your opponent is running a depth-2 minimax, using the result 80% of the time, and moving randomly otherwise
- What search tree should you use?



Answer: Expectimax!

- To figure out EACH chance node's probabilities, you have to run a simulation of your opponent
- This kind of thing gets very slow very quickly
- Even worse if you have to simulate your opponent simulating you…
- … except for minimax, which has the nice property that it all collapses into one game tree

MODELLING ASSUMPTIONS



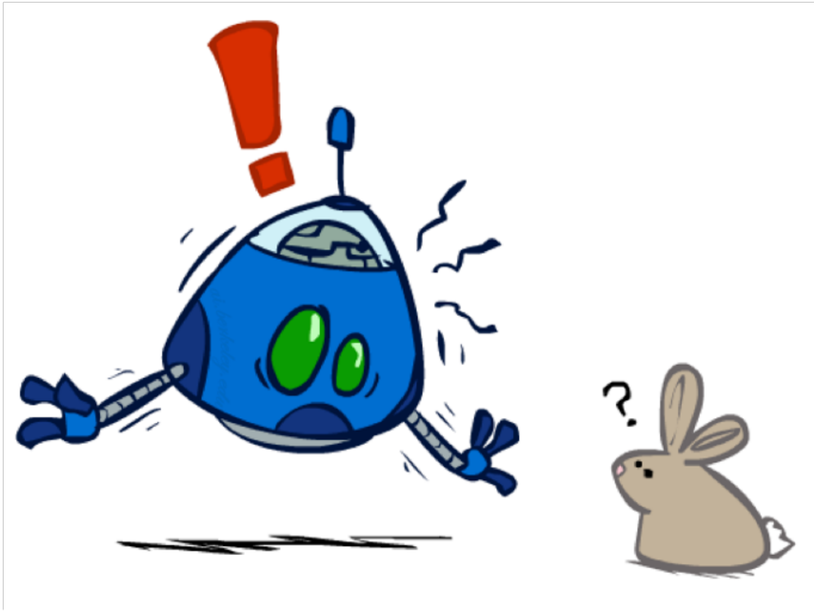
THE DANGER OF OPTIMISM AND PESSIMISM

Dangerous Optimism

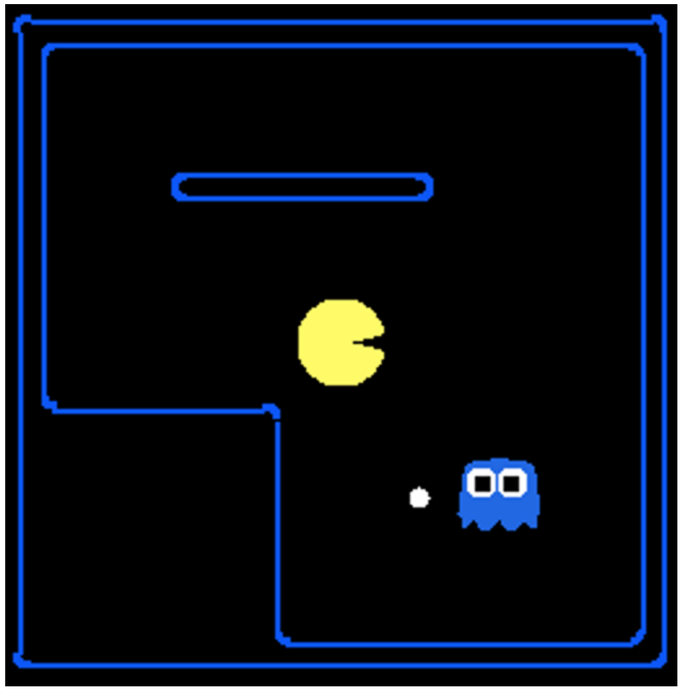
Assuming chance when the world is adversarial

Dangerous Pessimism

Assuming the worst case when the it's not likely

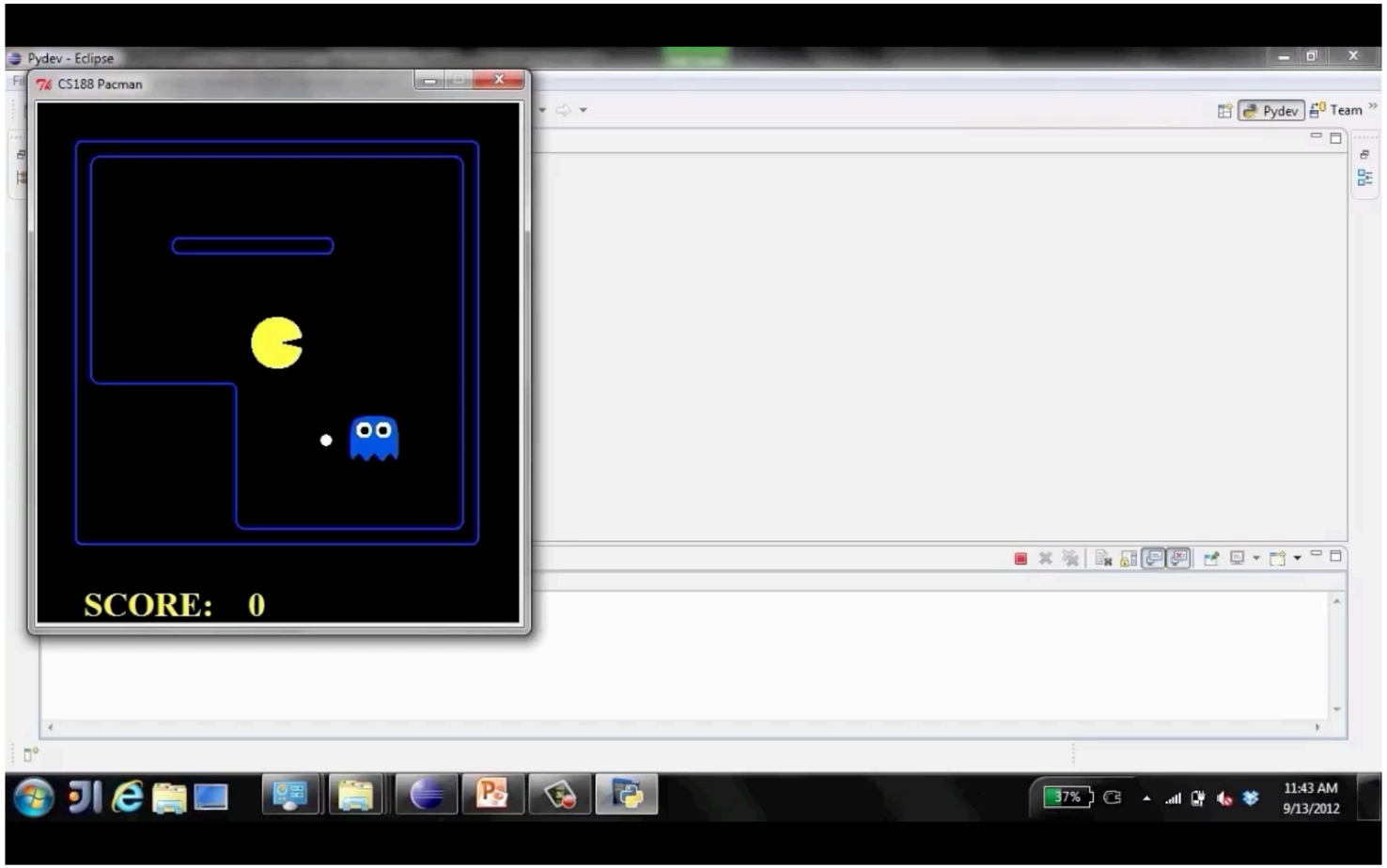


ASSUMPTIONS VS. REALITY

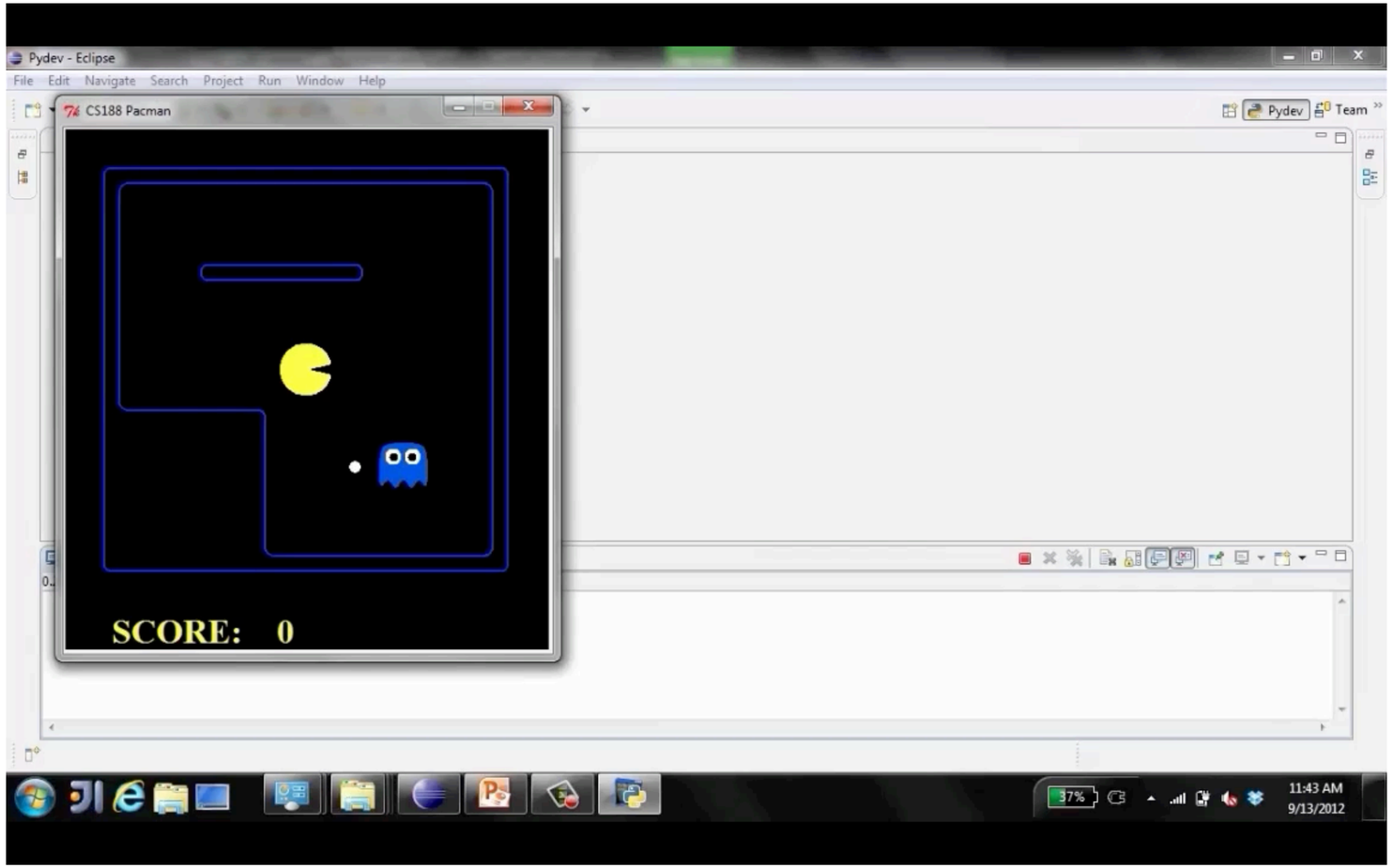


	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg score: 483	Won 5/5 Avg score: 493
Expectimax Pacman	Won 1/5 Avg score: -303	Won 5/5 Avg score: 503

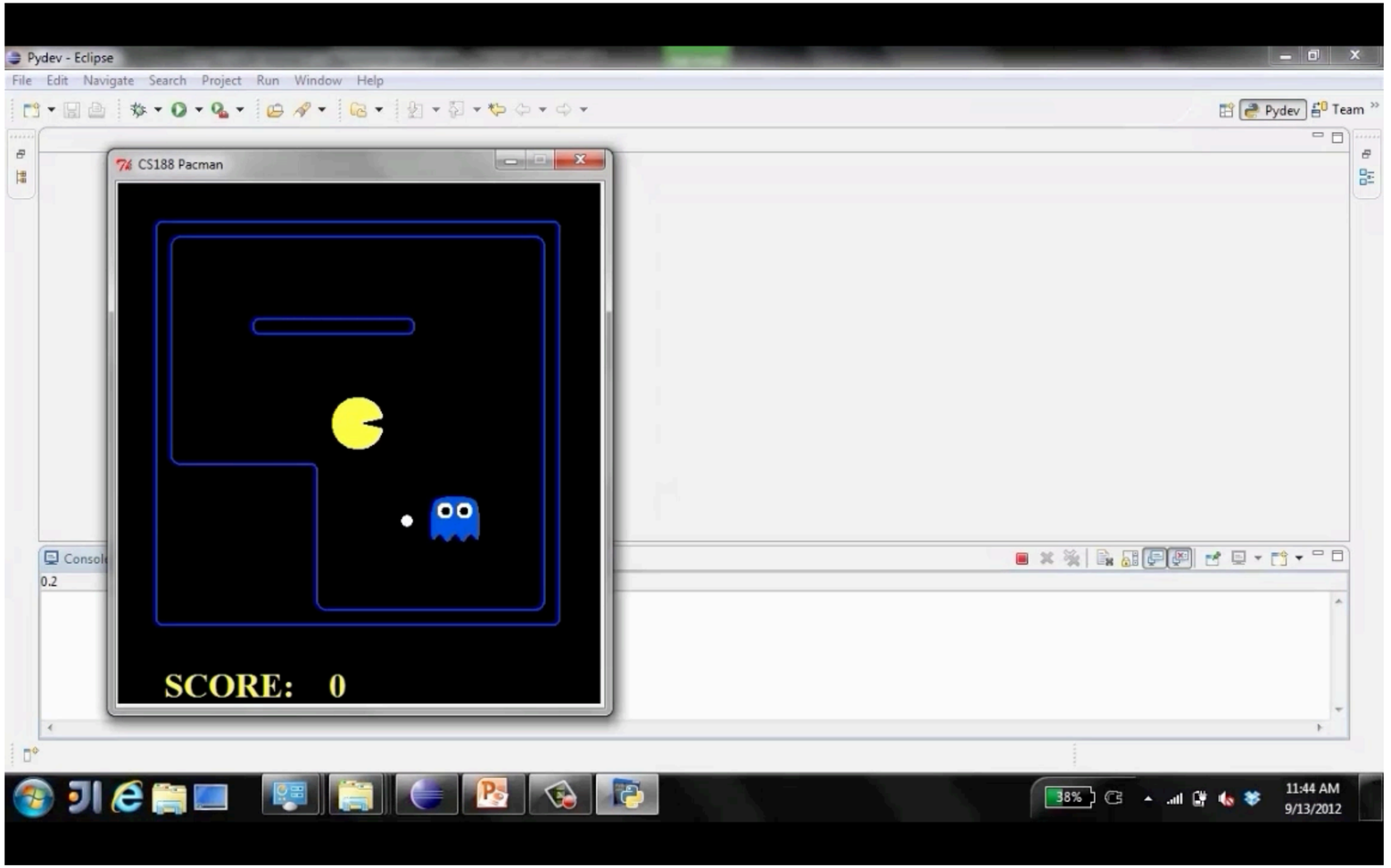
EXPECTIMAX PACMAN VS RANDOM GHOST



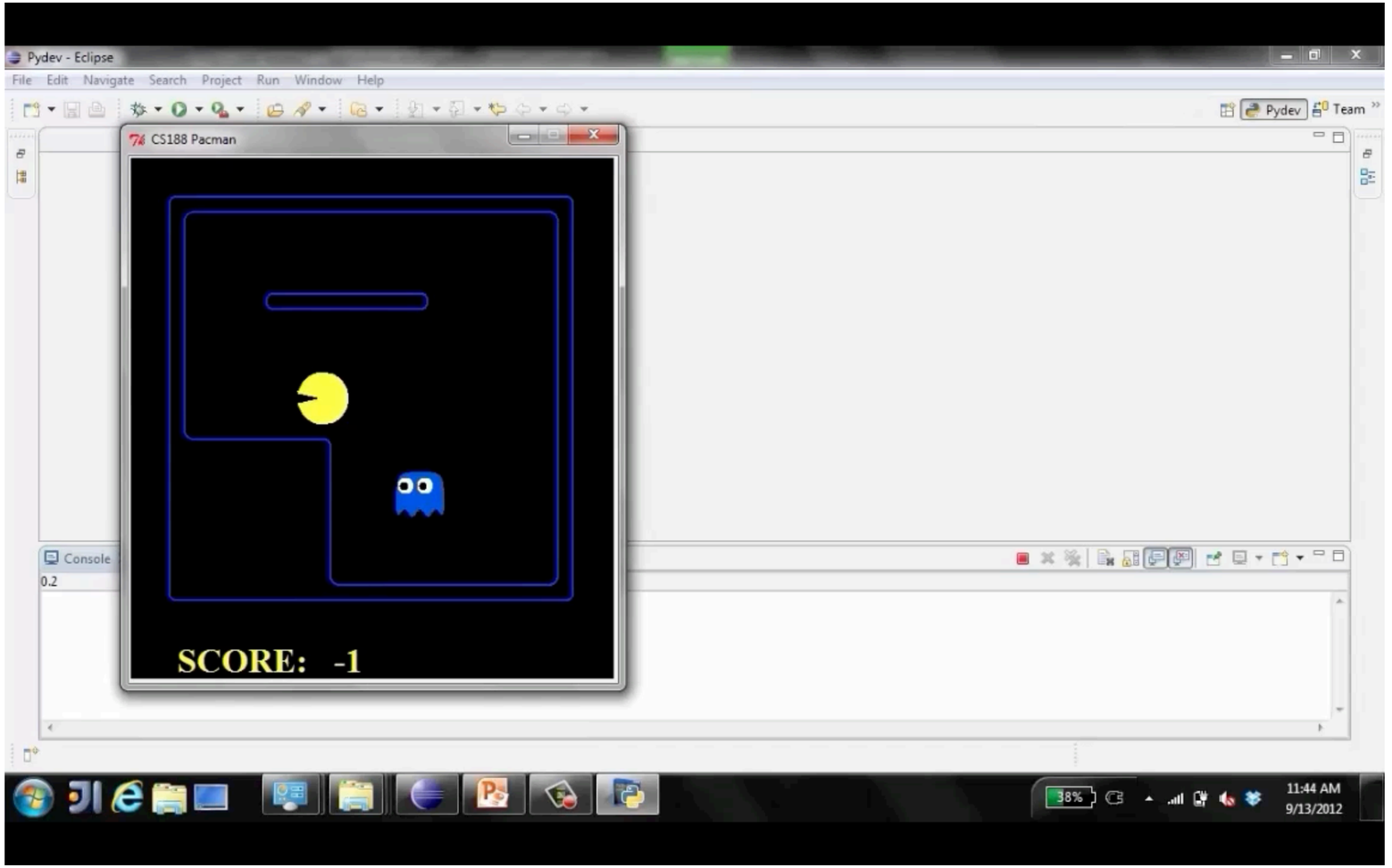
MINIMAX PACMAN VS ADVERSARIAL GHOST



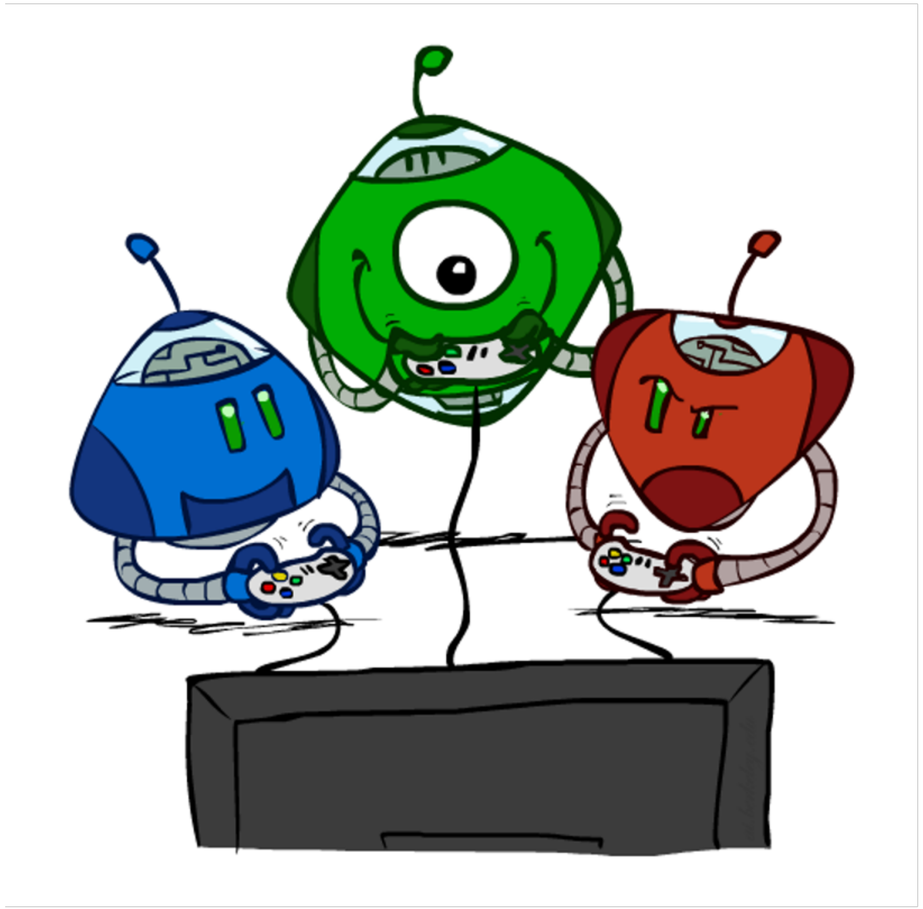
EXPECTIMAX PACMAN VS ADVERSARIAL GHOST



MINIMAX PACMAN VS RANDOM GHOST

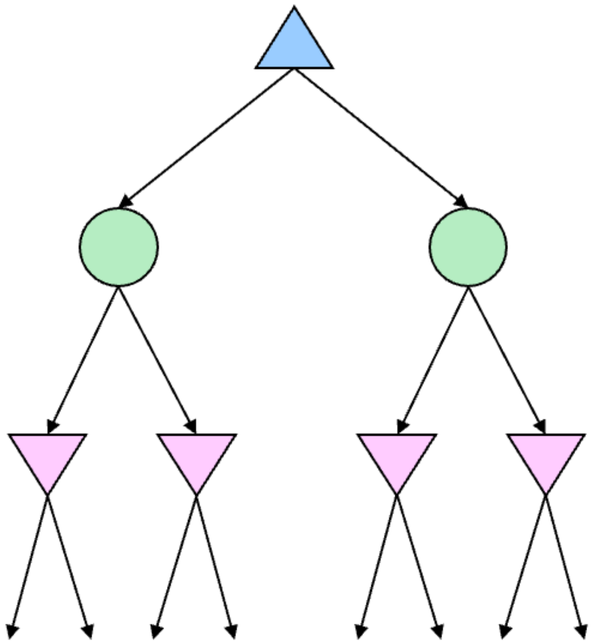


OTHER GAME TYPES



MIXED LAYER TYPES

- E.g. Backgammon
- Expectiminimax
  - Environment is an extra “random agent” player that moves after each min/max agent
  - Each node computes the appropriate combination of its children



EXAMPLE: BACKGAMMON

- Dice rolls increase *b*: 21 possible rolls with 2 dice
  - Backgammon » 20 legal moves
  - 4 plies = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$
- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier…
- Historic AI: TDGammon uses depth-2 search + very good evaluation function + reinforcement learning:
  - world-champion level play