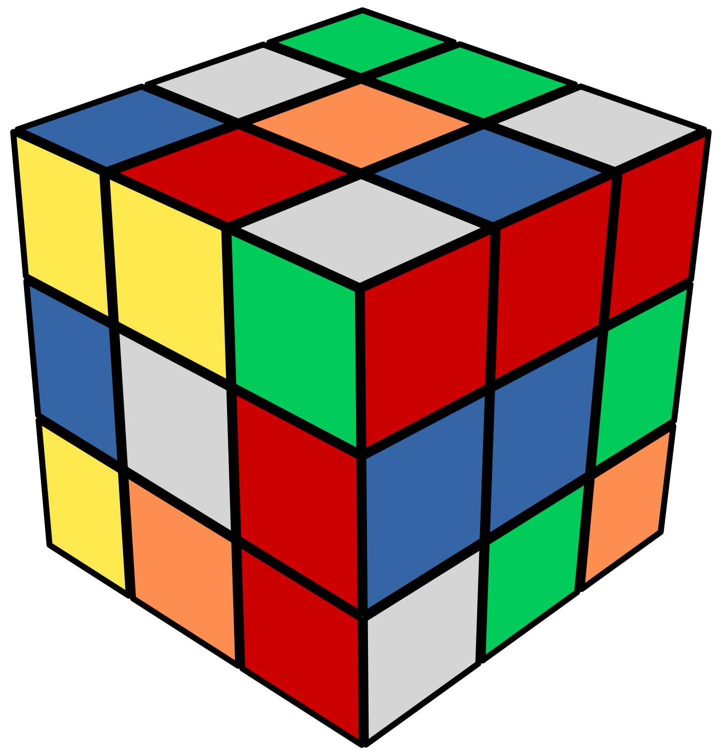
COMPSCI 761: ADVANCED TOPICS IN ARTIFICIAL INTELLIGENCE

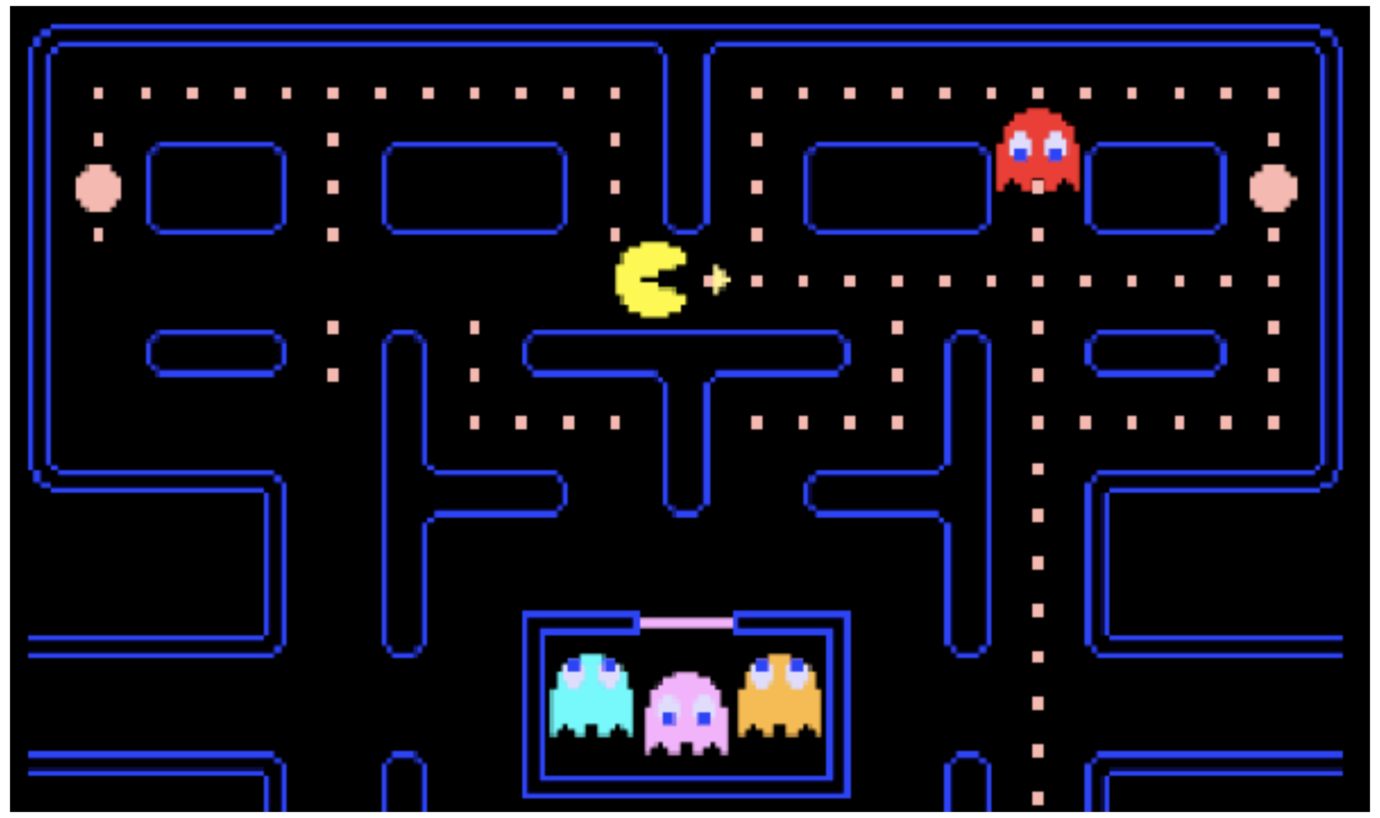
ADVERSARIAL SEARCH I

Anna Trofimova, August 2022

RECAP: SEARCH PROBLEM VS CSP VS GAME



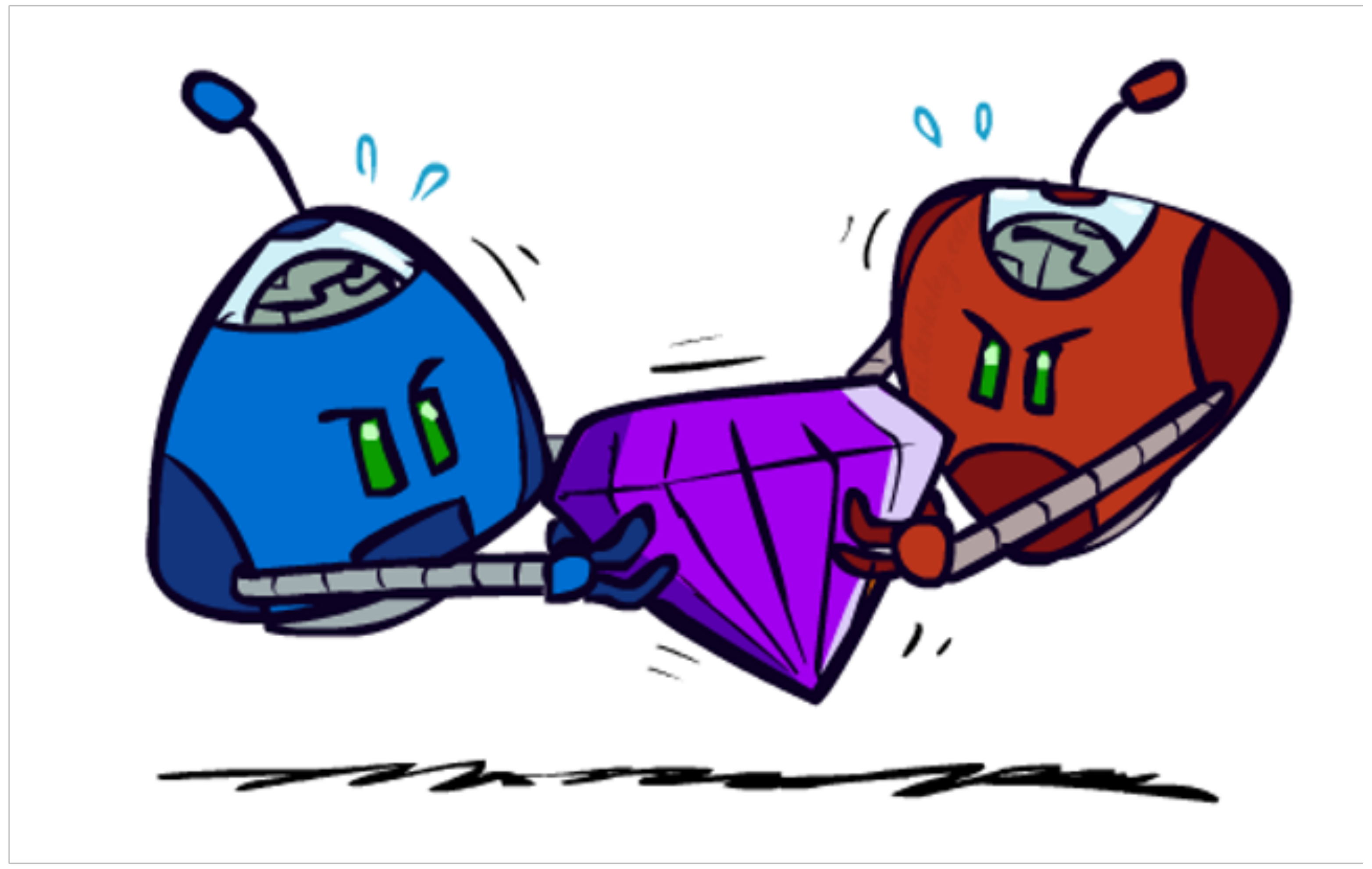
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

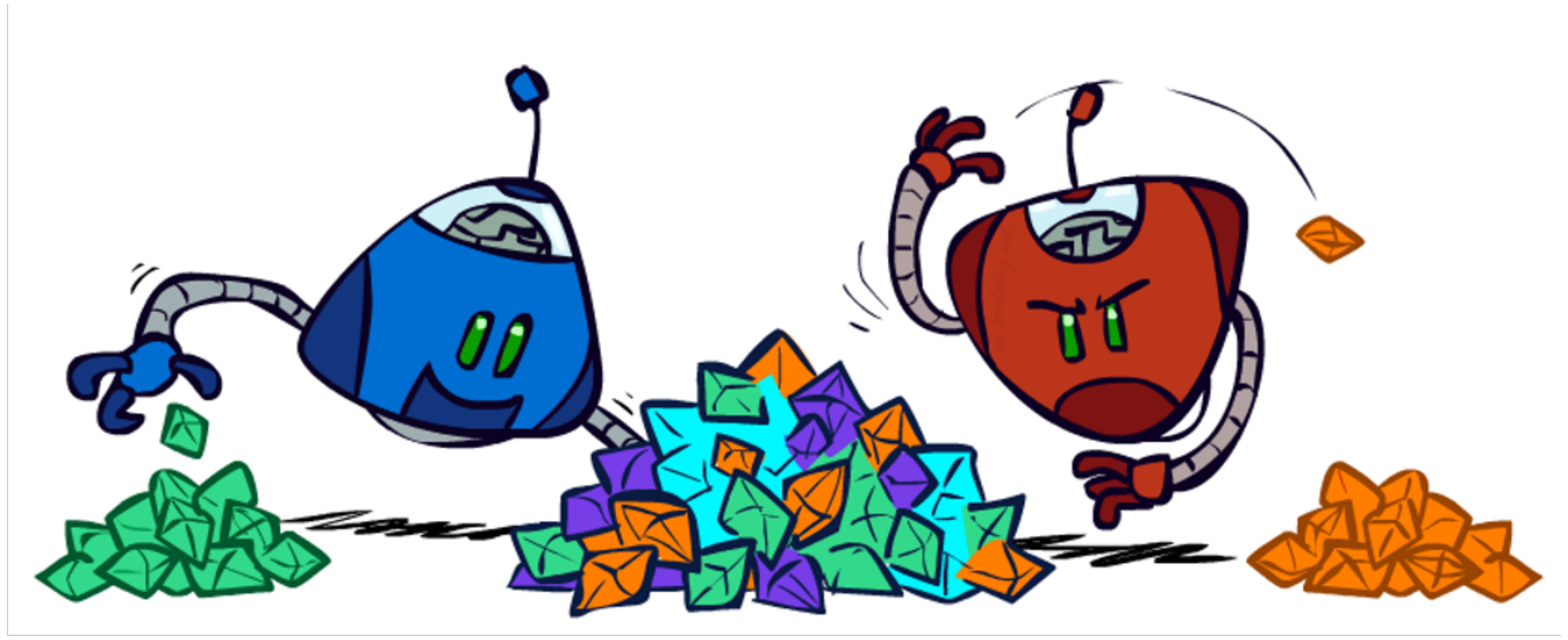


- "Unpredictable" opponent → specifying a move for every possible opponent reply

- Time limits → unlikely to find optimal solution, must approximate

RECAP: ZERO-SUM GAMES



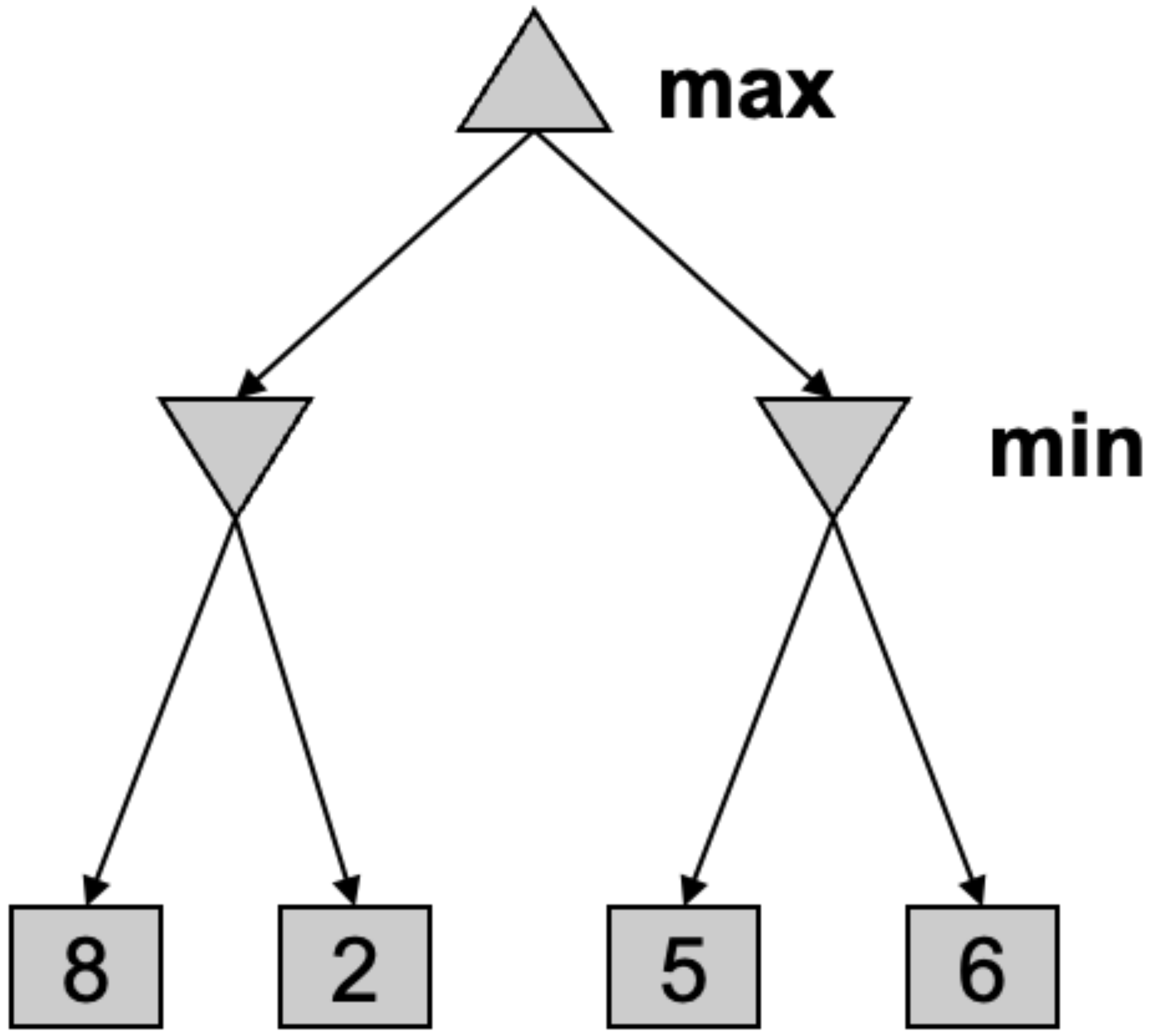


- Zero-Sum Games
  - Agents have ***opposite*** utilities
  - Pure competition:
  - One ***maximizes***, the other ***minimizes***

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

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



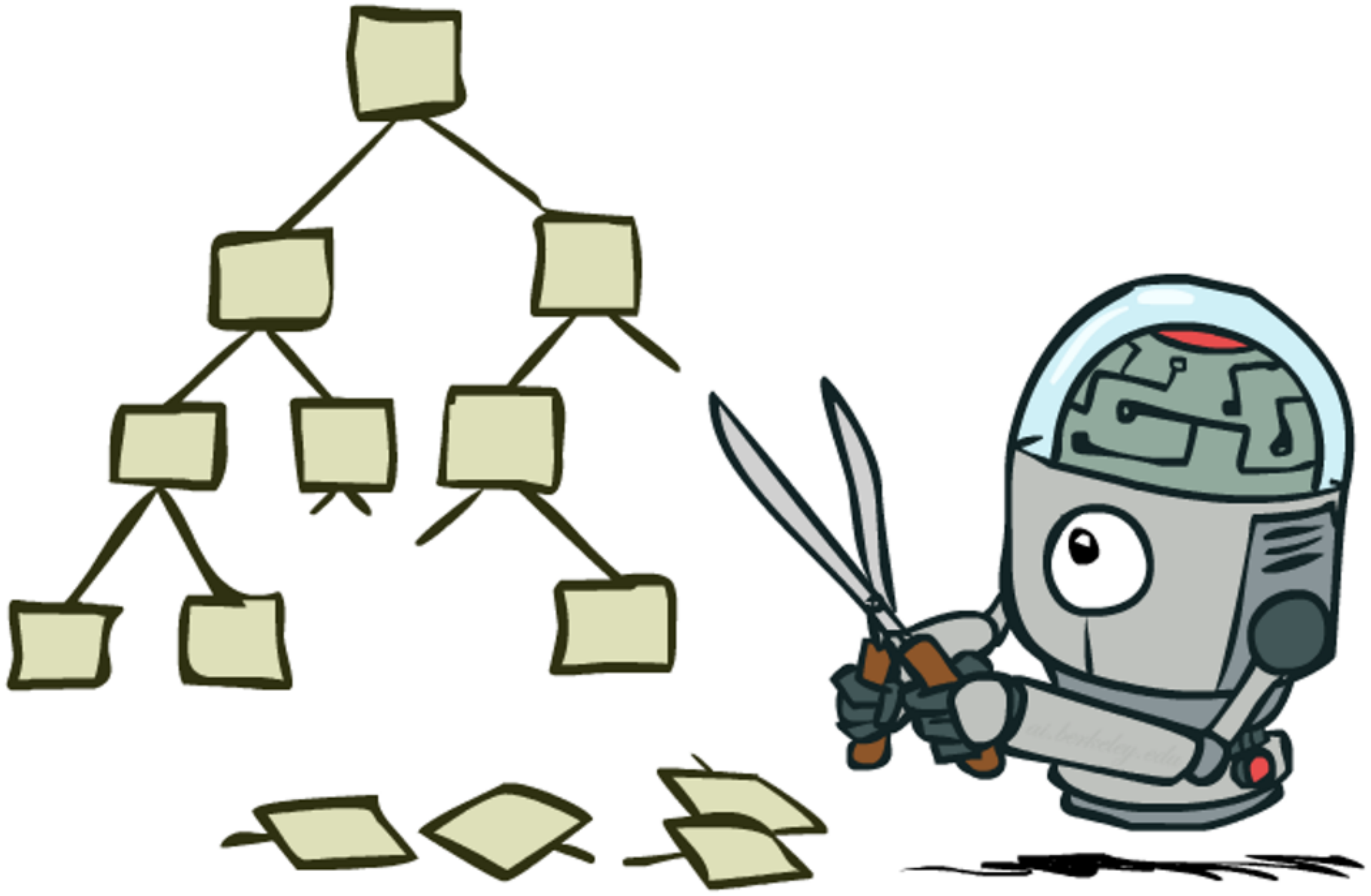
RECAP: MINIMAX PROPERTIES

- Optimal against a perfect player. Otherwise?
- Time complexity?
    - $O(b^m)$
    - m = maximum depth of search tree, b = branching factor
- Space complexity?
    - O(bm)
- For chess, b ~ 35, m ~ 100
    - Exact solution is completely infeasible
    - But, do we need to explore the whole tree?

TODAY

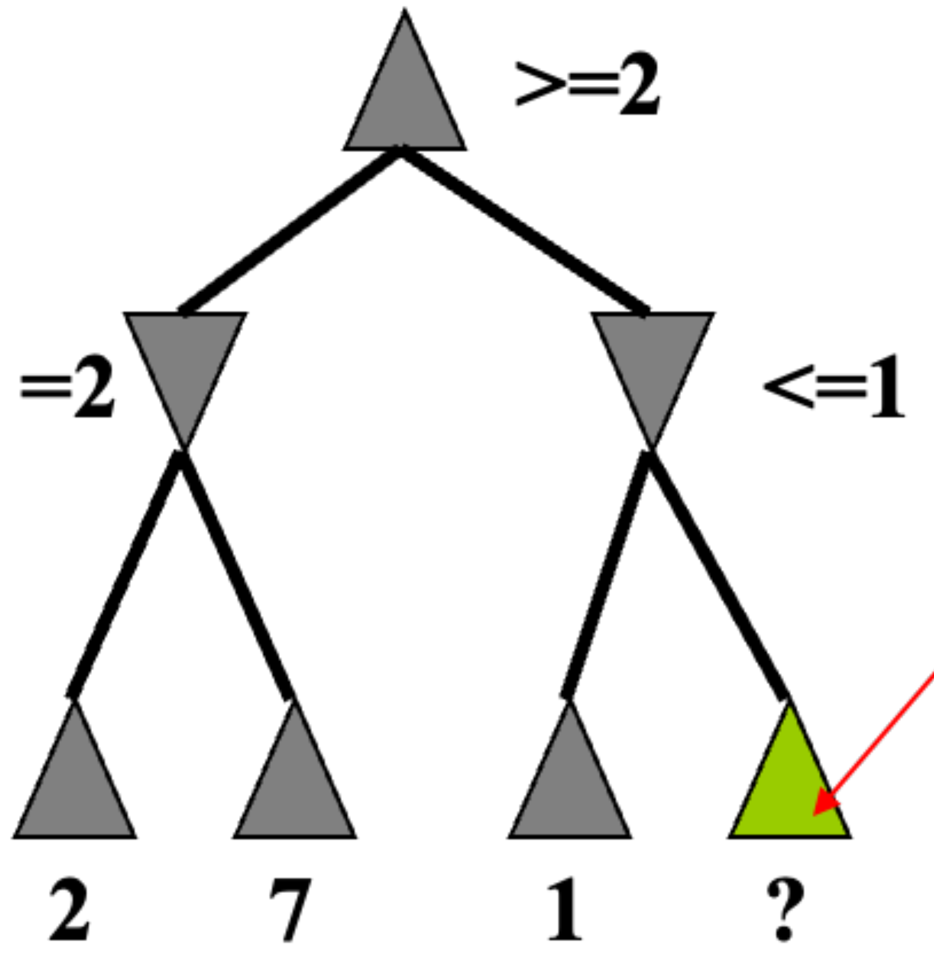
- α-β pruning
- Expectimax

GAME TREE PRUNING



α-β PRUNING

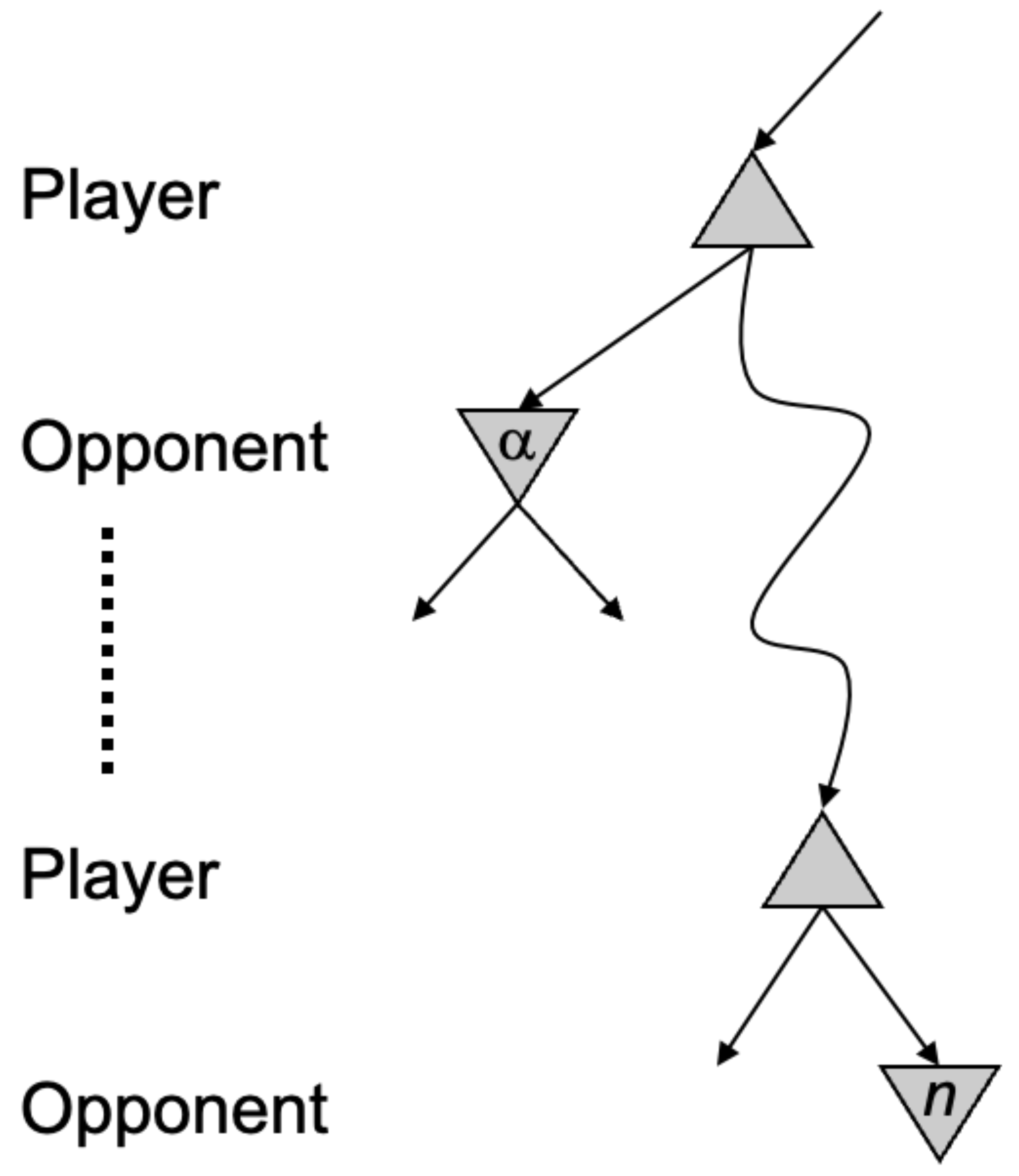
- A way to improve the performance of the Minimax Procedure
- Basic idea: "If you have an idea which is surely bad, don't take the time to see how truly awful it is" ~ Pat Winston



- We don't need to compute the value at this node.
- No matter what it is it can't effect the value of the root node.

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



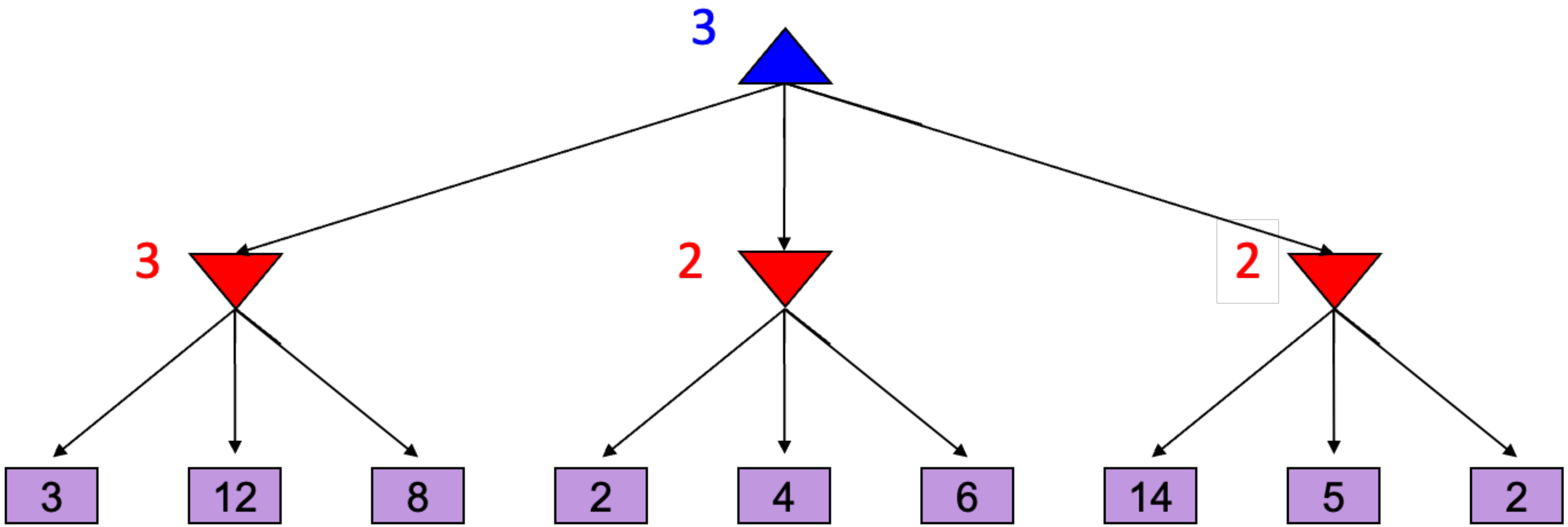
α-β PRUNING ALGORITHM

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
```

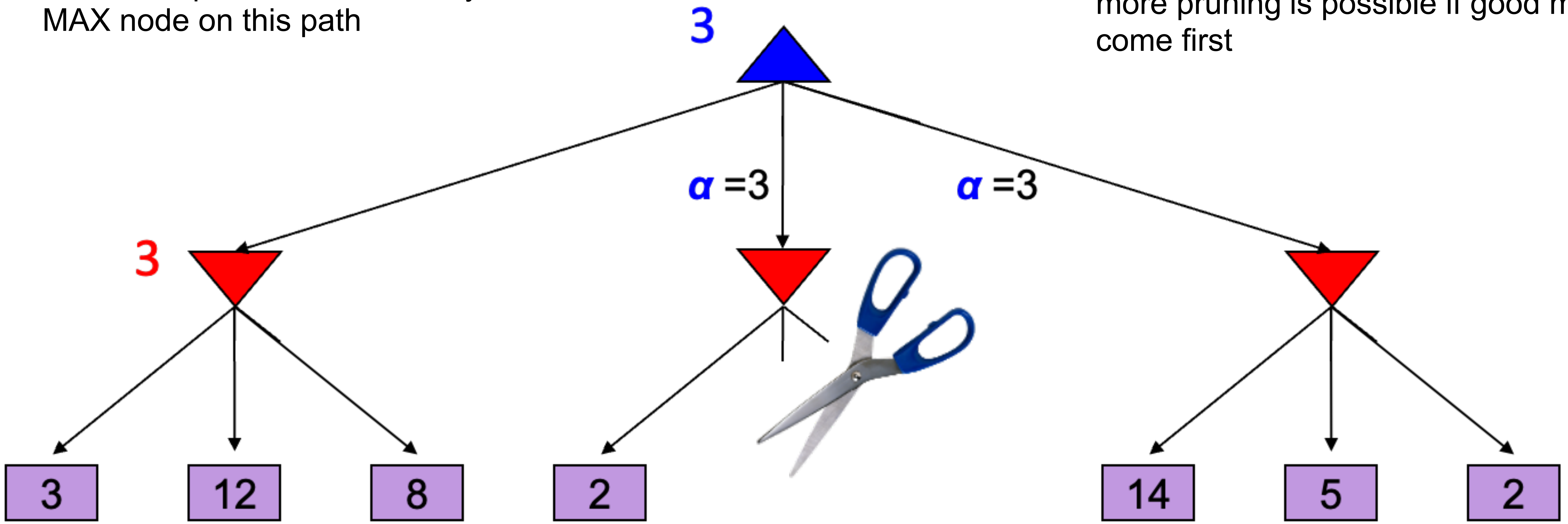
MINIMAX EXAMPLE



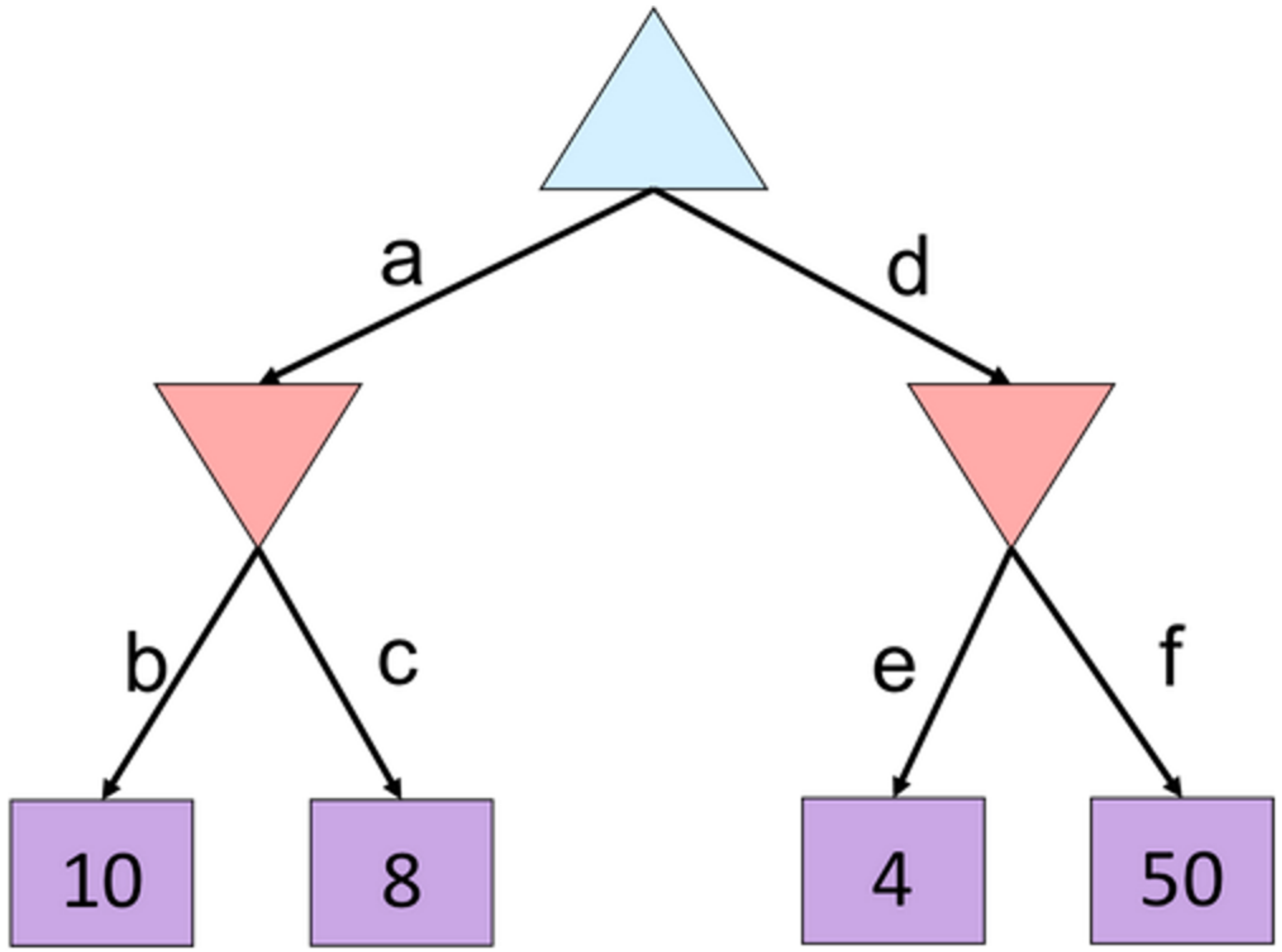
α-β PRUNING EXAMPLE

α = best option so far from any MAX node on this path

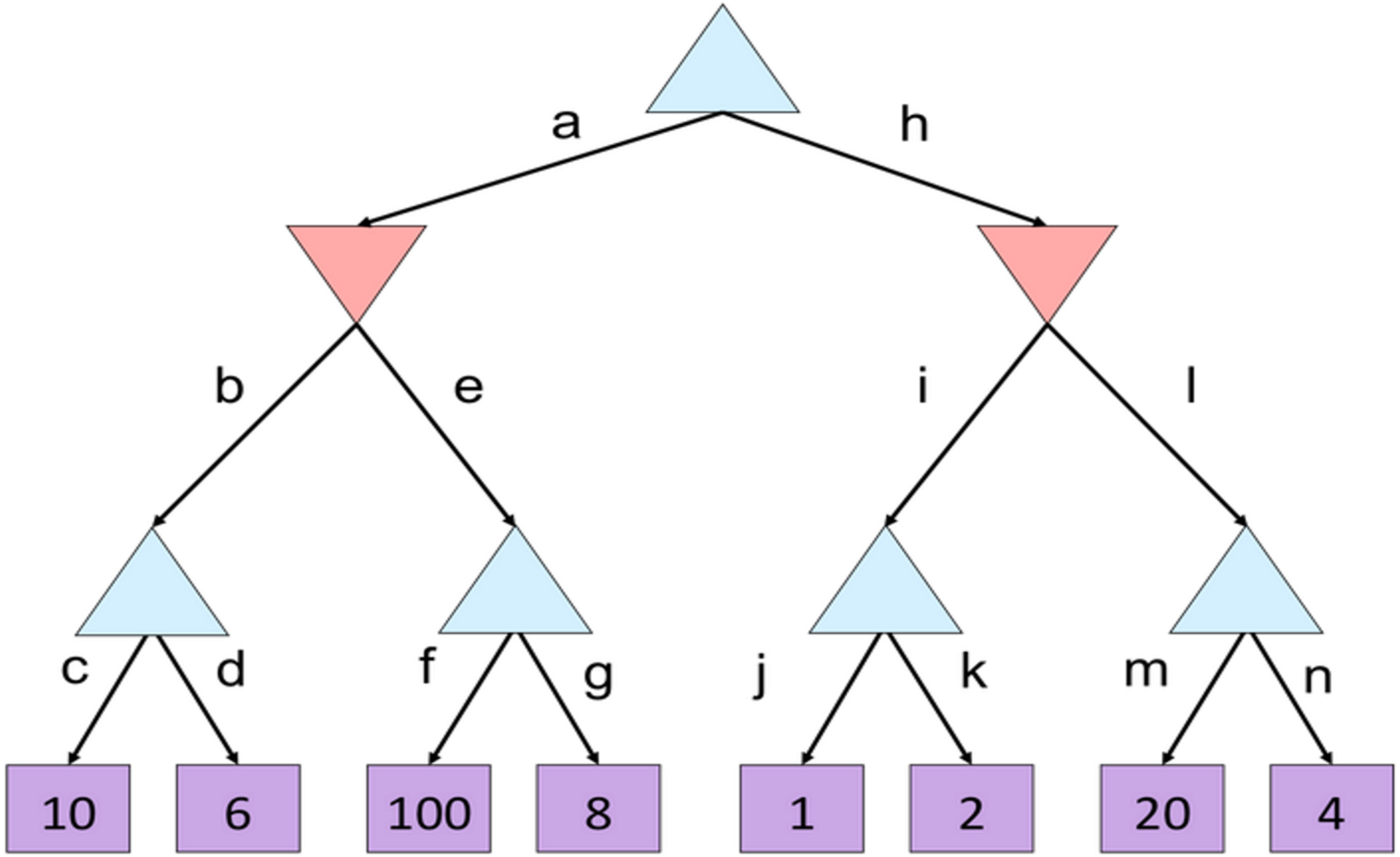
The order of generation matters:
more pruning is possible if good moves come first



α-β PRUNING QUIZ



α-β PRUNING QUIZ 2



α-β PRUNING PROPERTIES

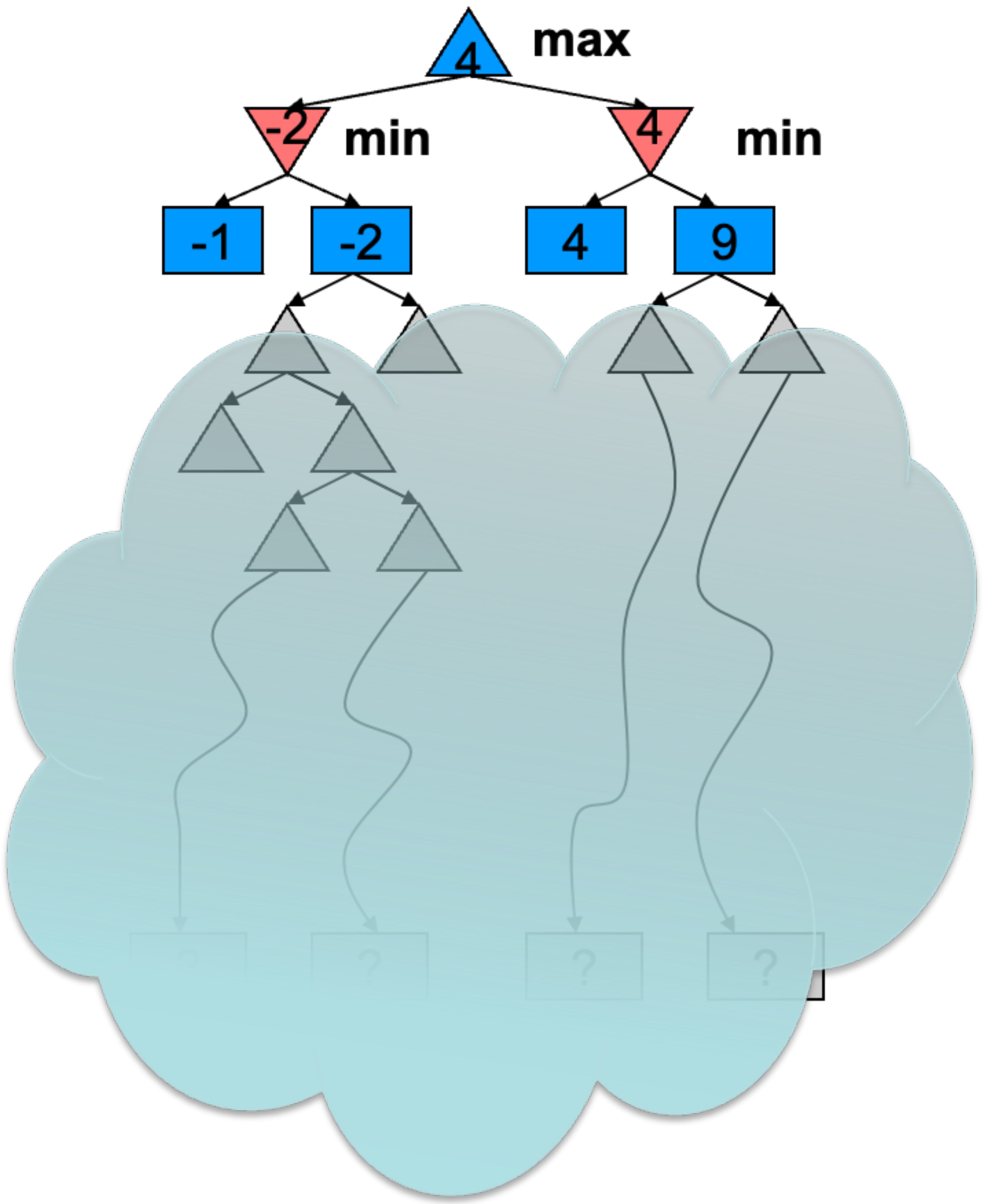
- Pruning has no effect on final result
- Good move ordering improves effectiveness of pruning
- With “perfect ordering”:
    - Time complexity drops to $O(b^{m/2})$
    - Doubles solvable depth
    - Full search of, e.g. chess, is still hopeless!
- A simple example of metareasoning, here reasoning about which computations are relevant
- For chess: only 35^{50} instead of 35^{100}!! Yaaay!!!!! Still not feasible…

"The whole question of making an automaton play any game depended upon the possibility of the machine being able to ***represent all the myriads of combinations*** relating to it."

\- Charles Babbage

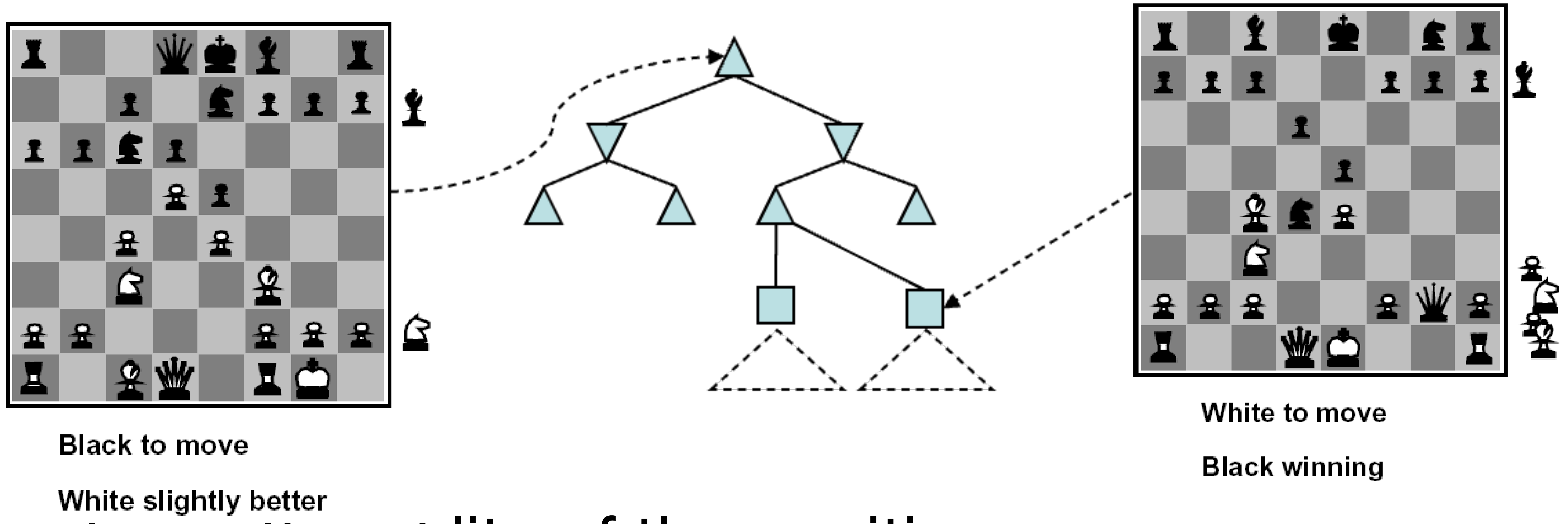
RESOURCE LIMITS

- 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



EVALUATION FUNCTION

- Function which scores non-terminals

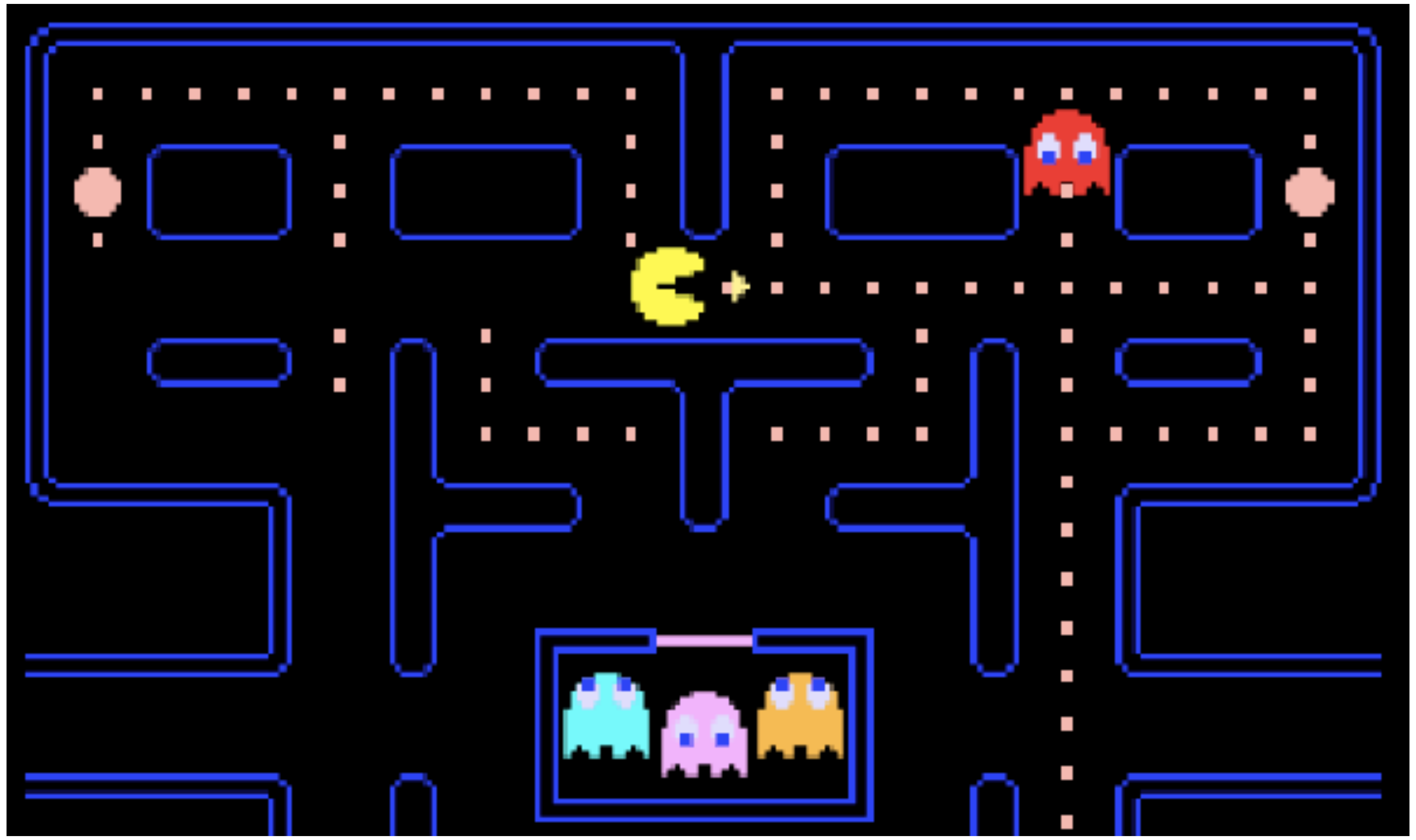


- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:

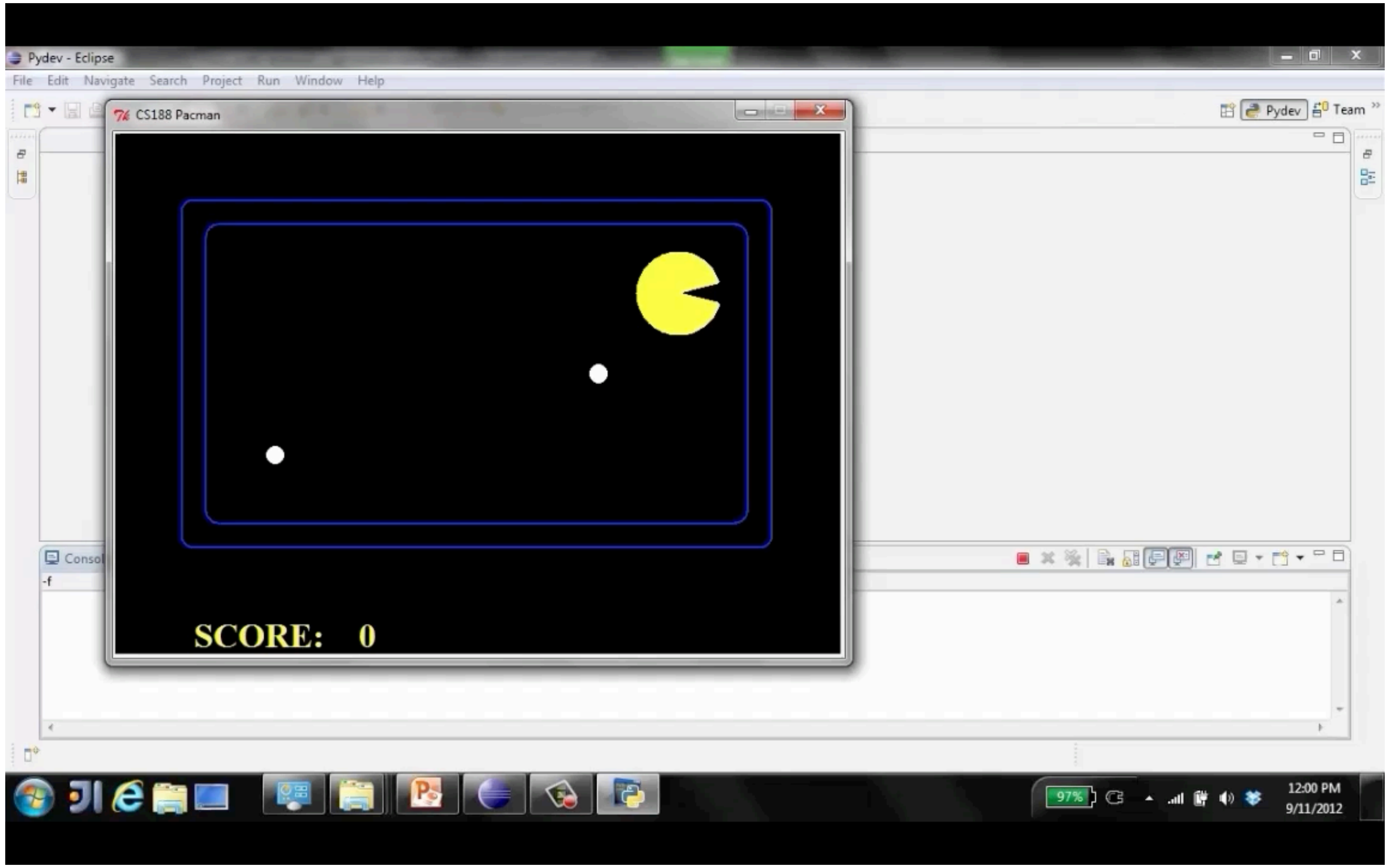
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

  - 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

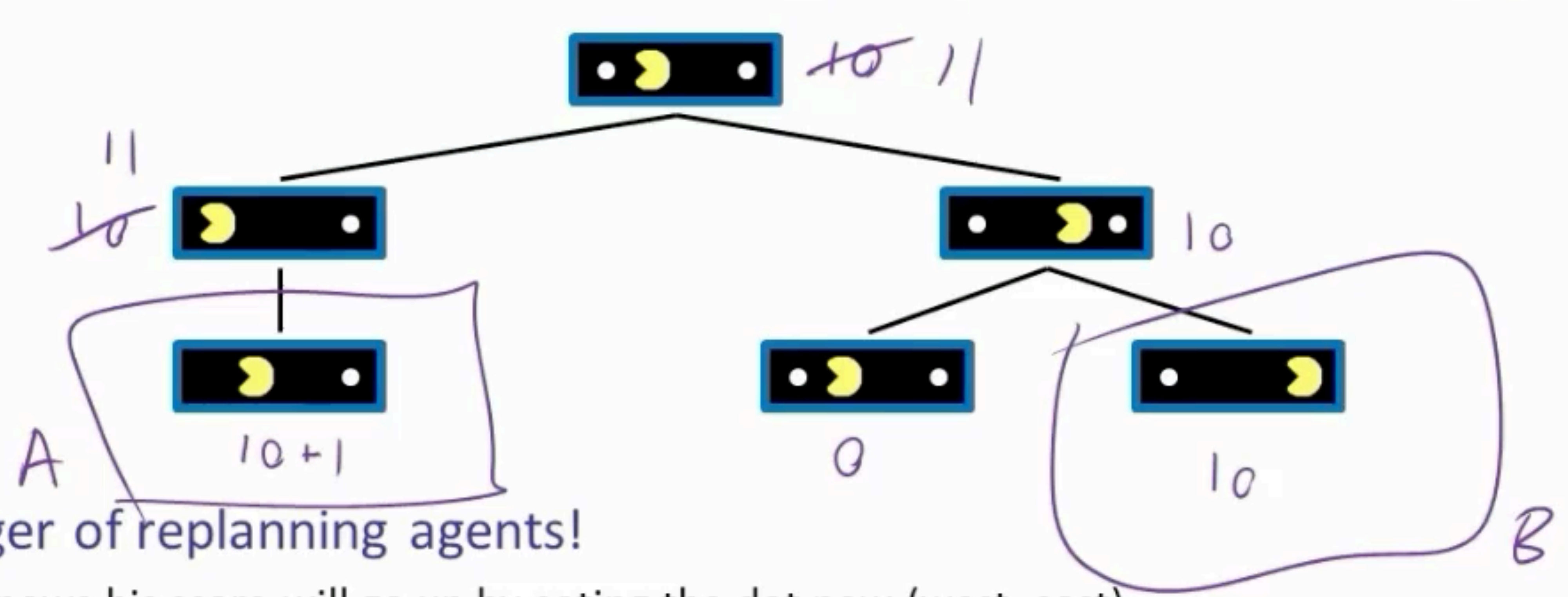
EVALUATION FUNCTION FOR PACMAN?



VIDEO OF DEMO THRASHING (D=2)



VIDEO OF DEMO THRASHING (D=2) FIXED



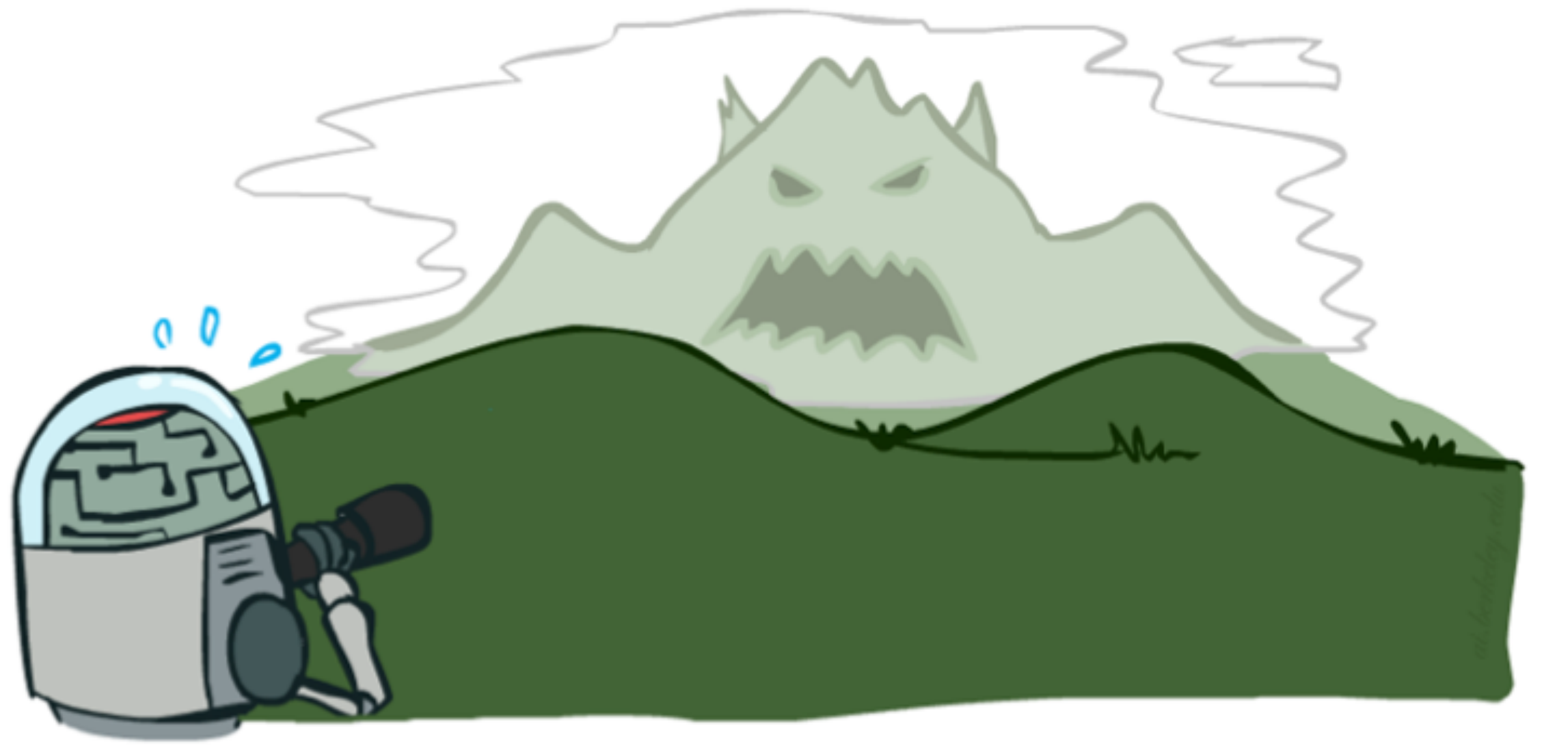
- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

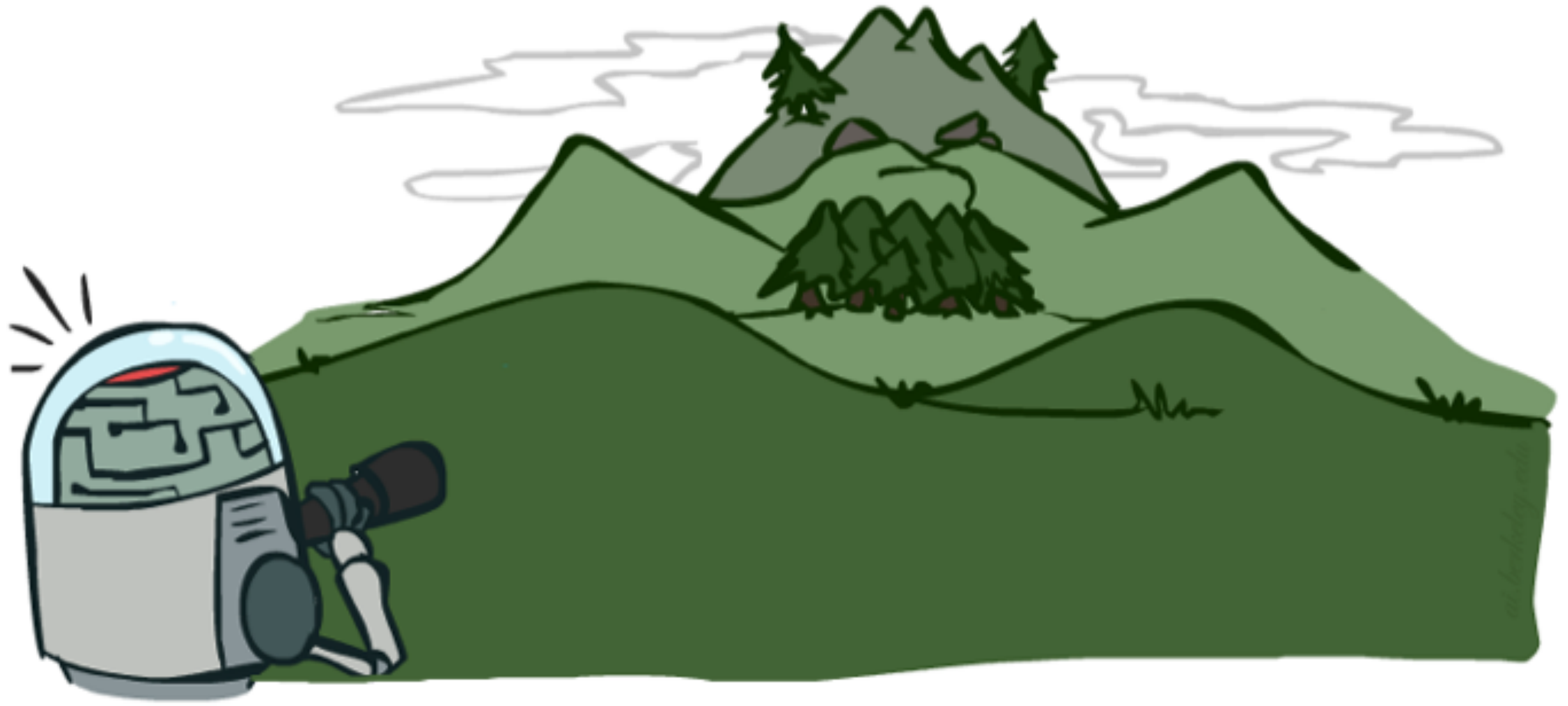
DEPTH MATTERS

- Evaluation functions are always imperfect

- Deeper search => better play (usually)

- Or, deeper search gives same quality of play with a less accurate evaluation function





ITERATIVE DEEPENING

- Iterative deepening uses DFS as a subroutine:

    1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)

    2. If “1” failed, do a DFS which only searches paths of length 2 or less.

    3. If “2” failed, do a DFS which only searches paths of length 3 or less.

- ….and so on.

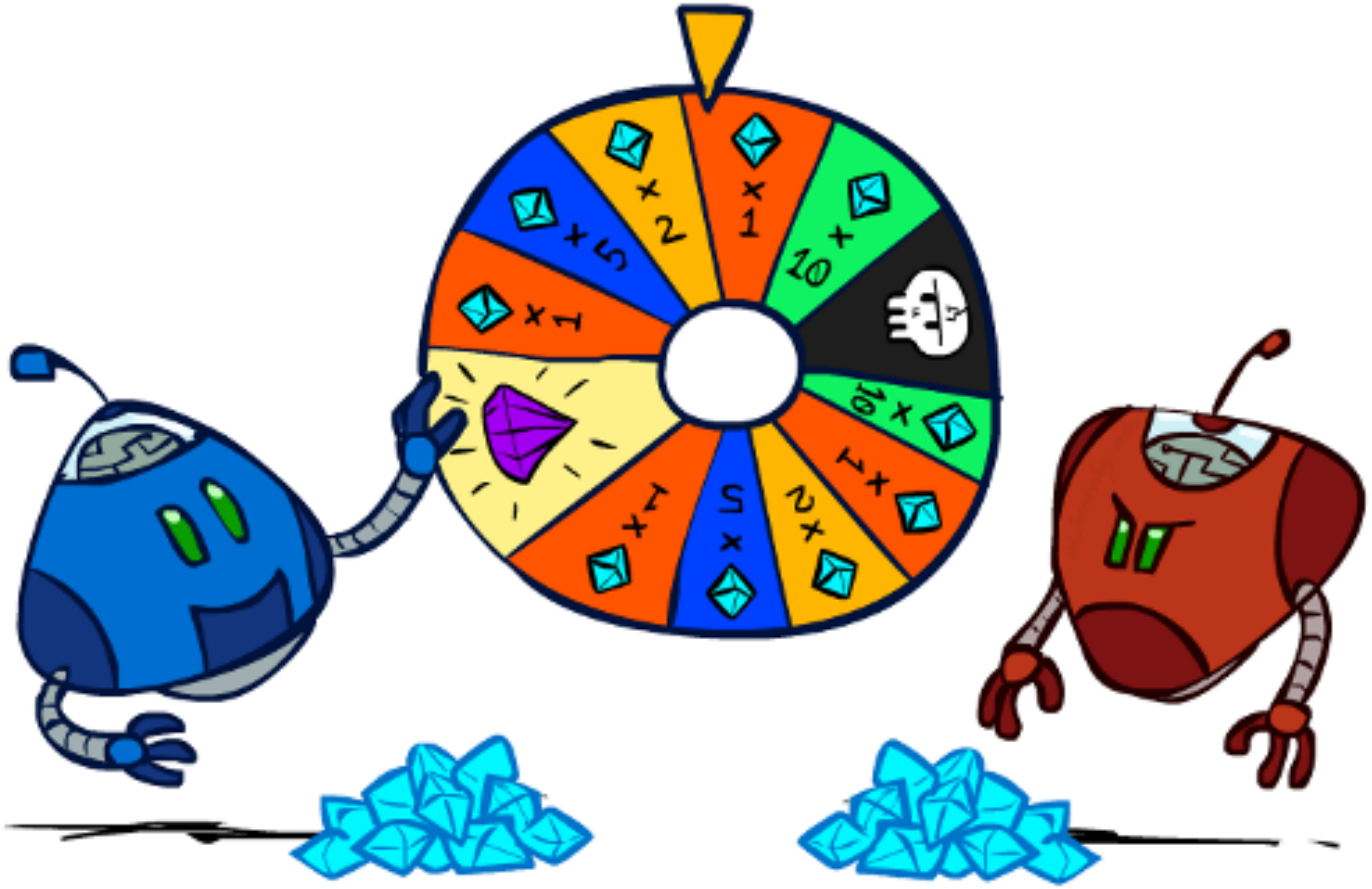
- This works for single-agent search as well!

- Why do we want to do this for multiplayer games?

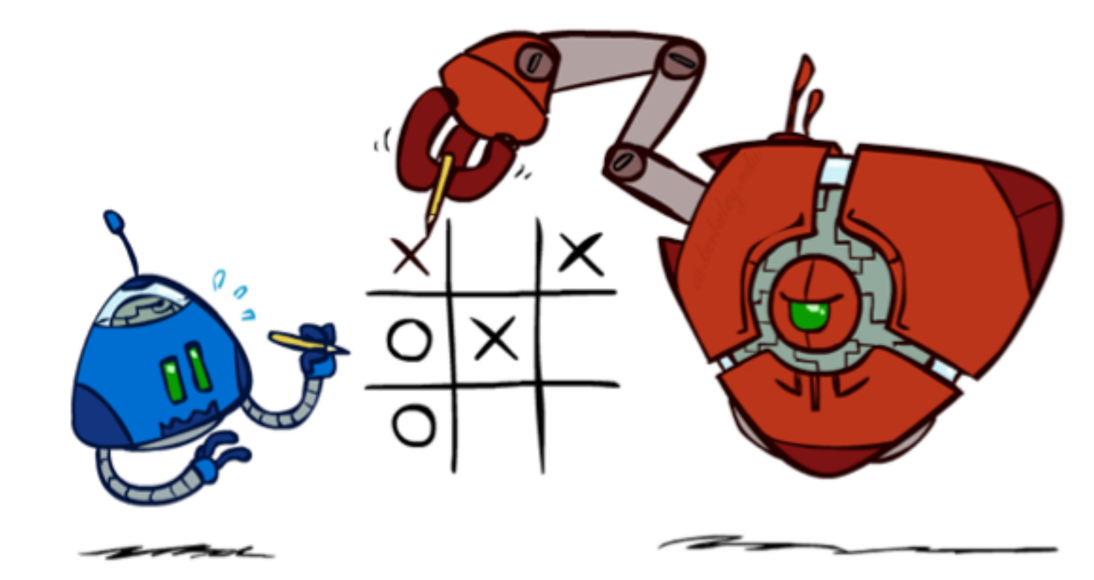
THE STORY SO FAR

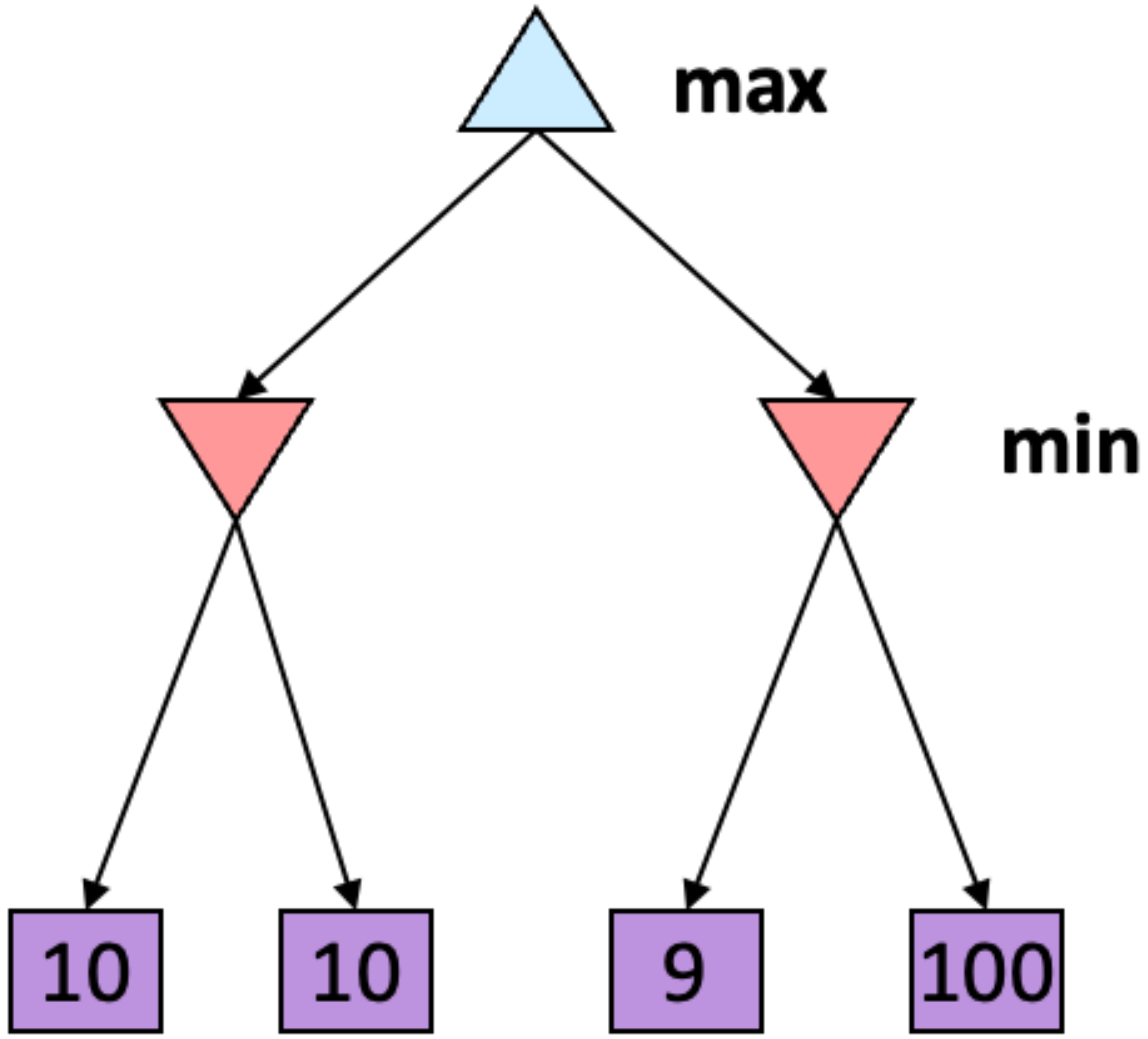
- Focus on two-player, zero-sum, deterministic, observable, turn-taking games
- Minimax defines rational behavior
- Recursive DFS implementation: space complexity $O(bm)$, time complexity $O(b^m)$
- Alpha-beta pruning with good node ordering reduces time complexity to $O(b^{m/2})$
- Still nowhere close to solving chess, let alone Go or StarCraft

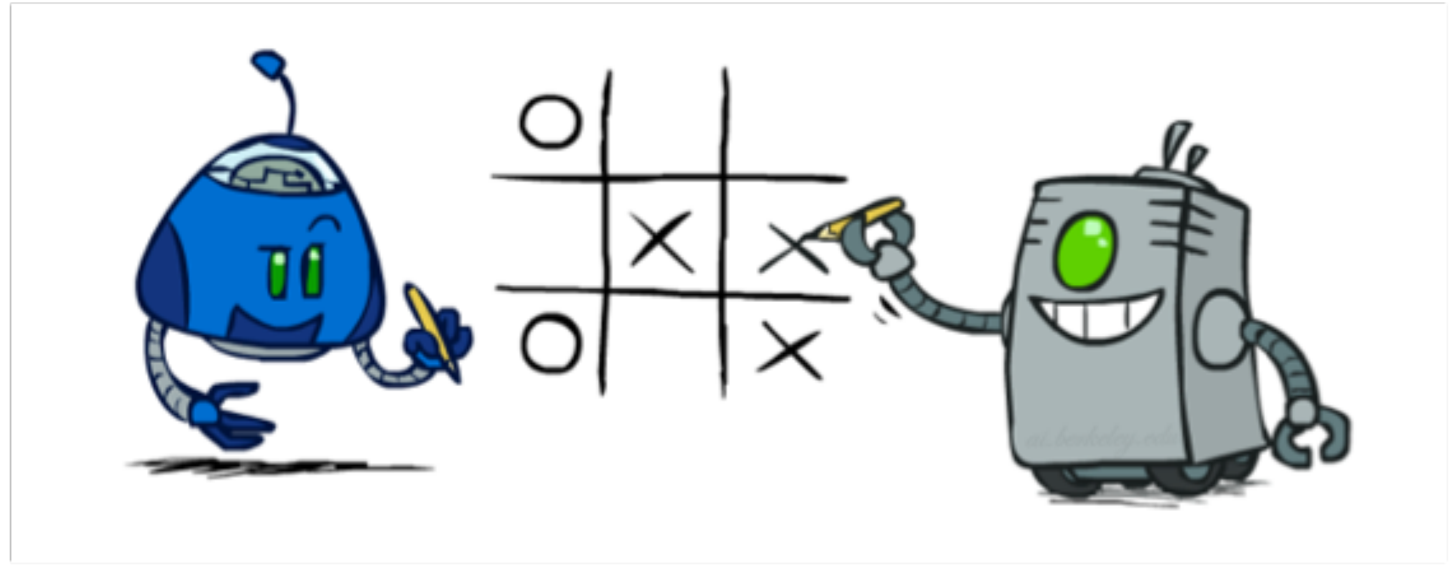
GAMES WITH UNCERTAIN OUTCOMES



WORST-CASE VS. AVERAGE CASE

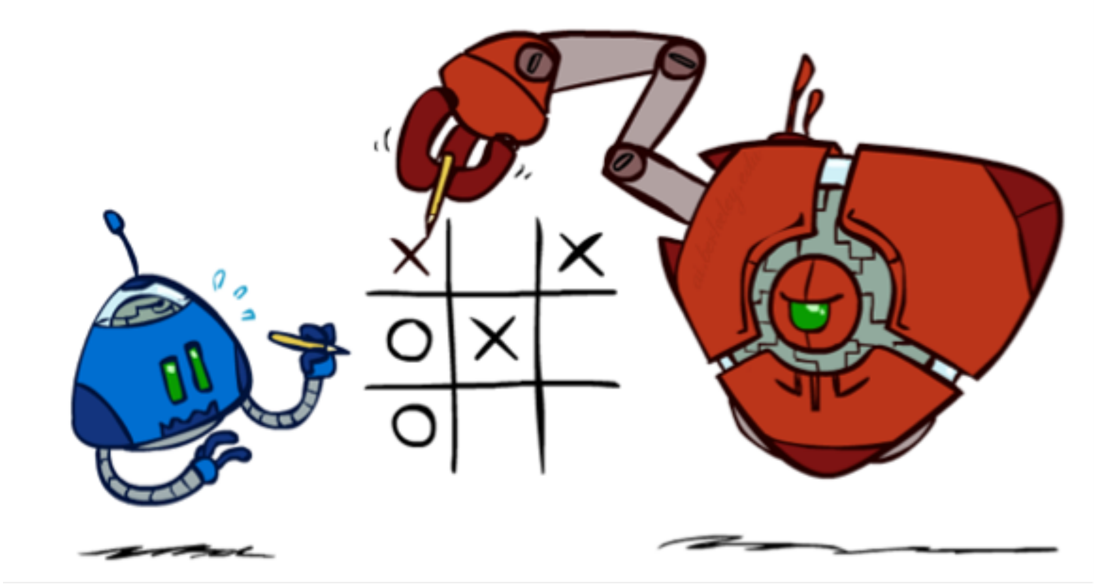


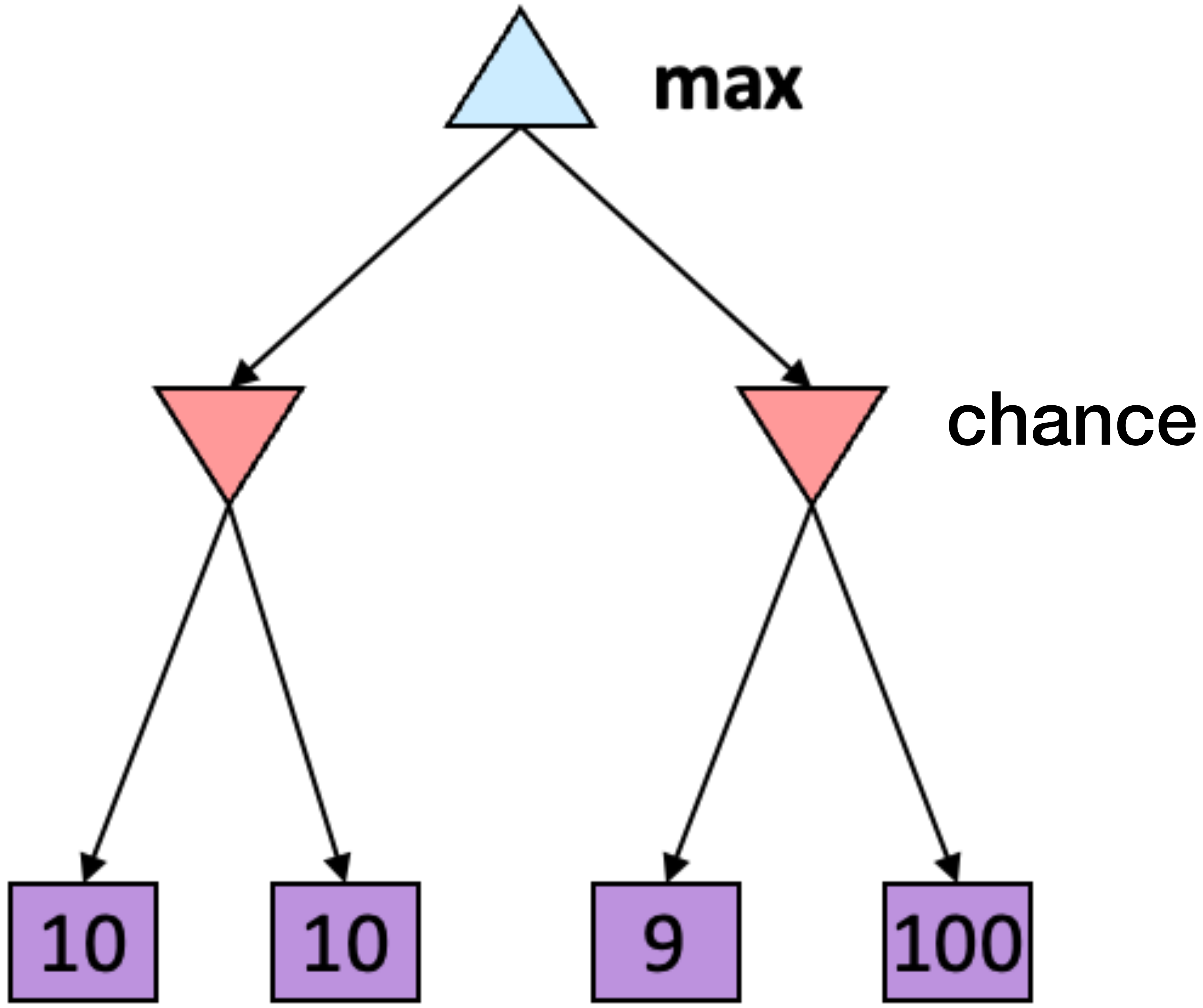


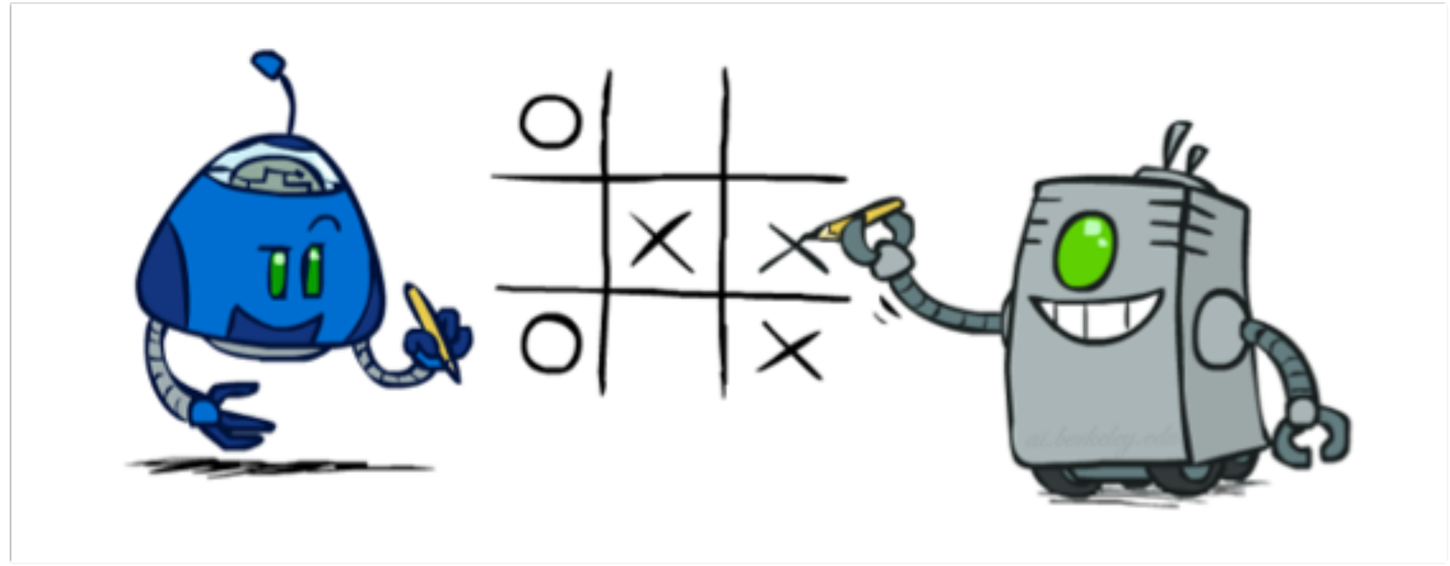


WORST-CASE VS. AVERAGE CASE

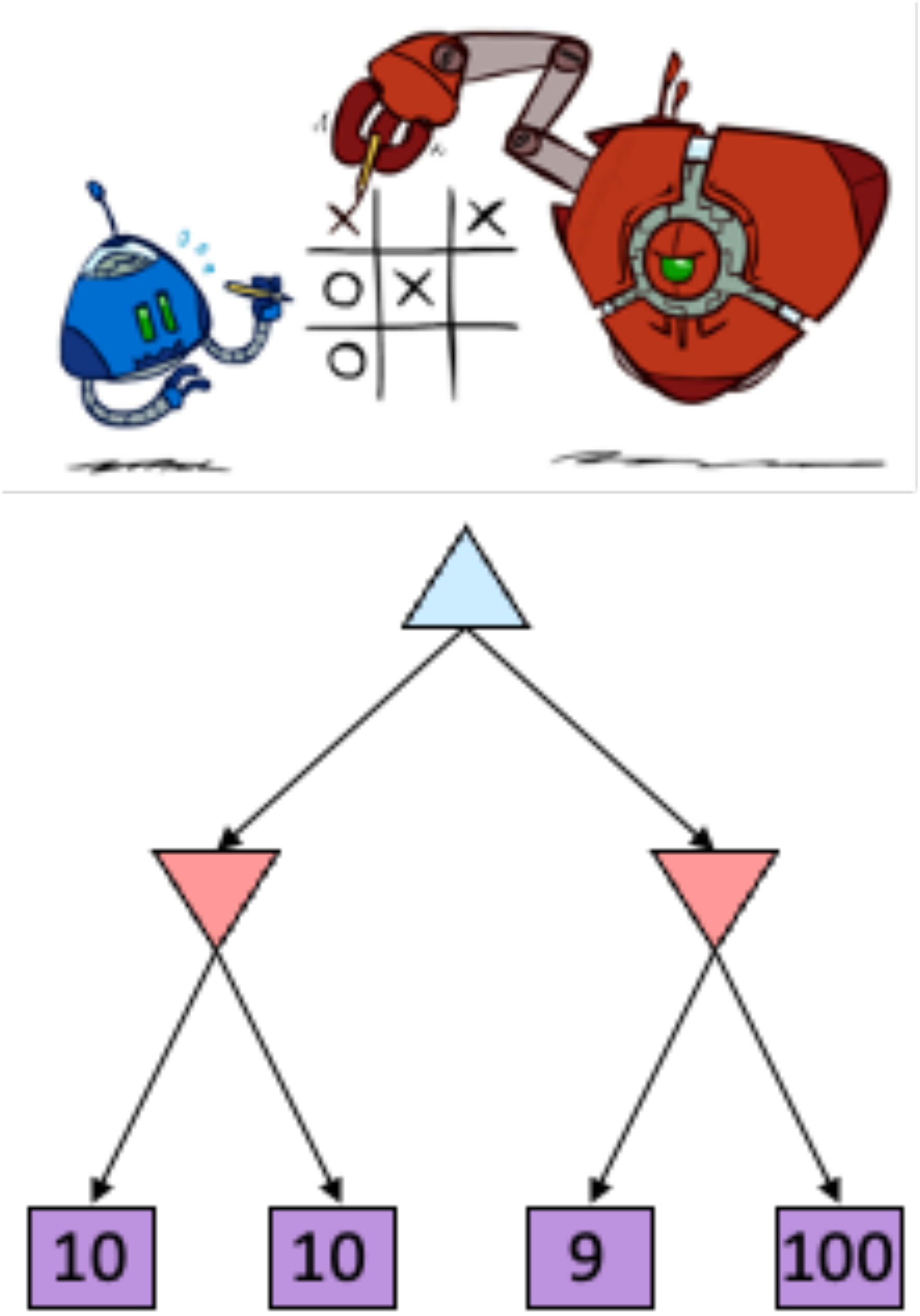
- Idea: uncertain outcomes controlled by chance, not an adversary!



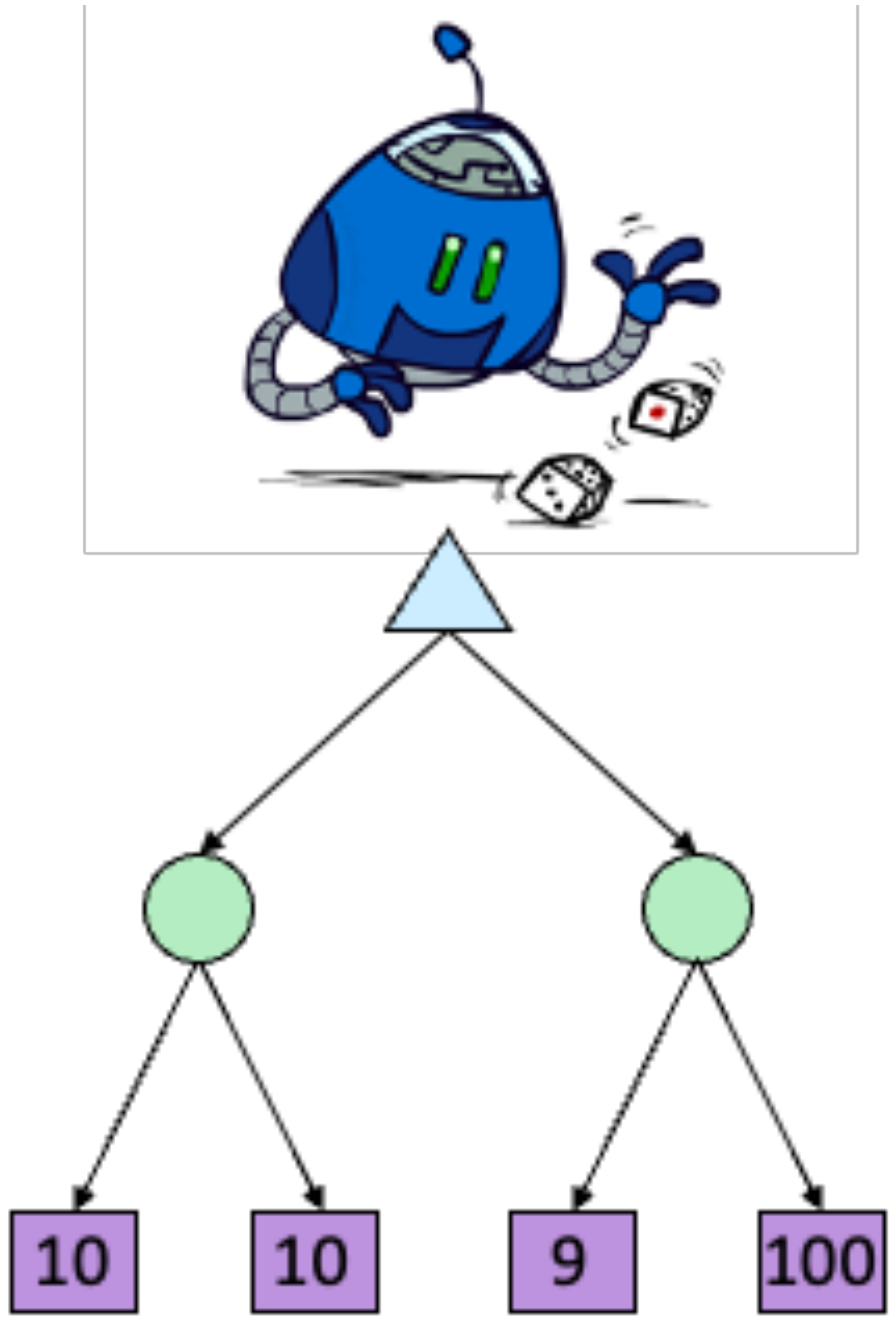




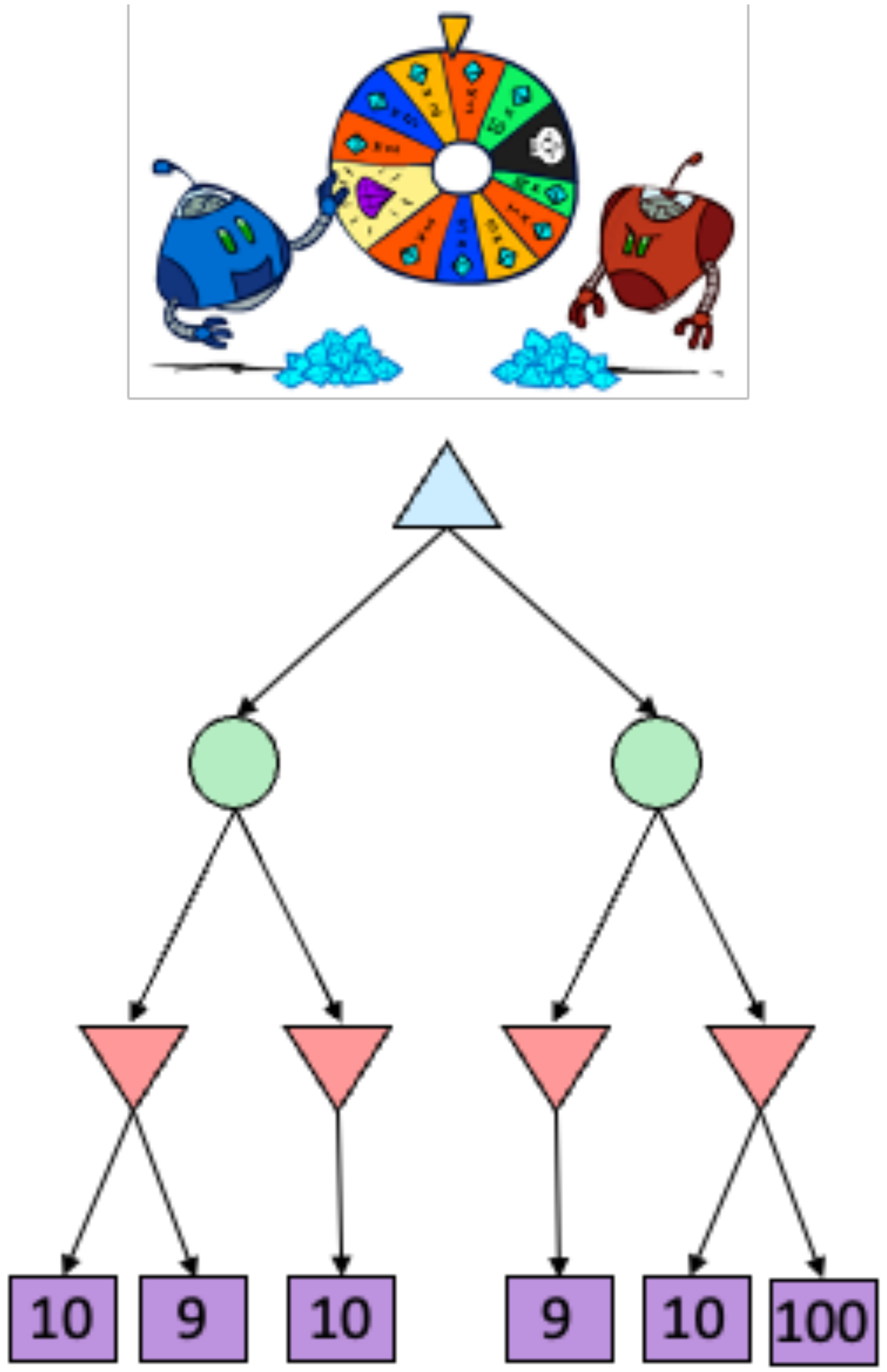
CHANCE OUTCOMES IN TREES



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))

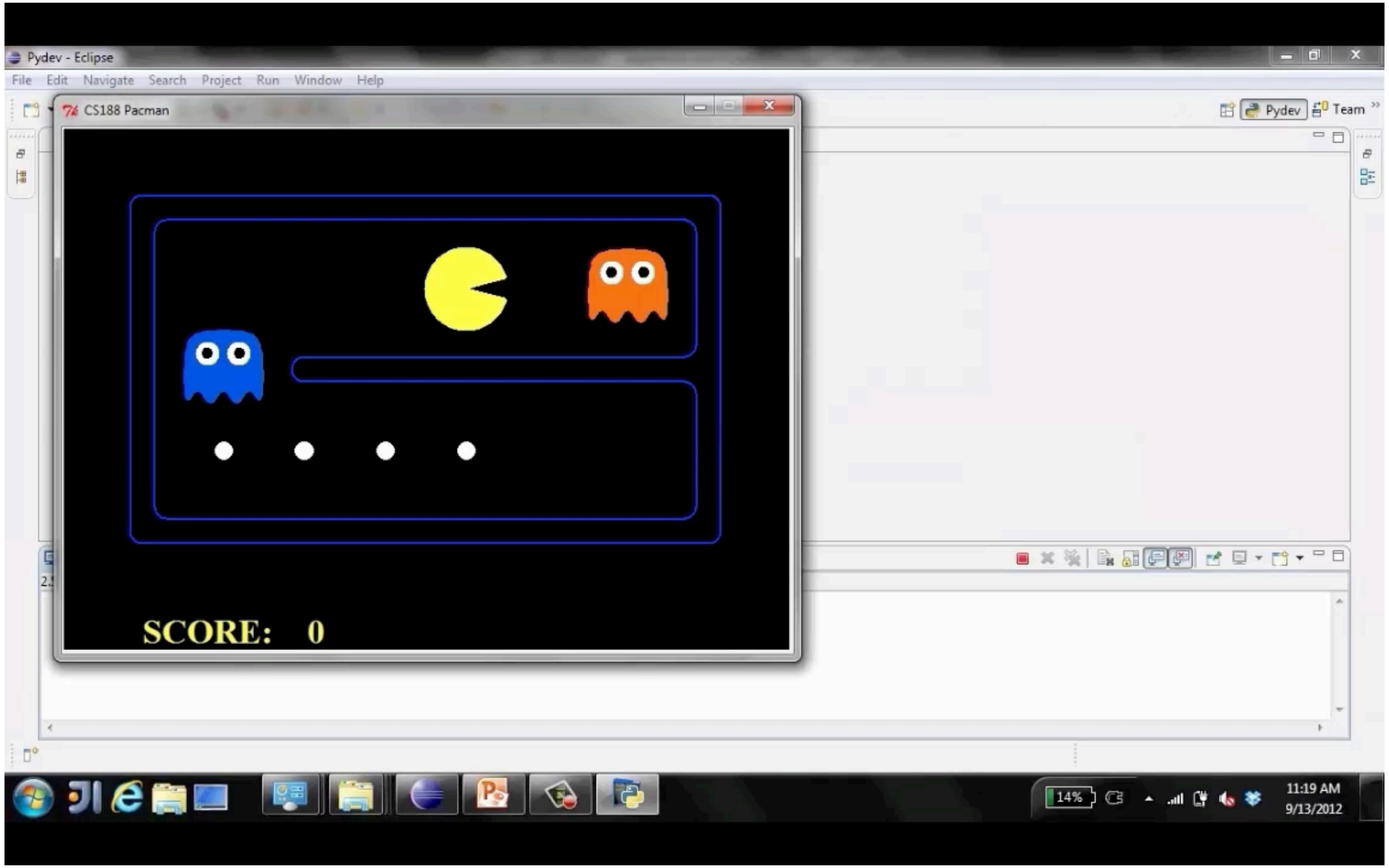
if Player(s) = MIN then return $\min_{a \text{ in Actions}(s)}$ minimax_value(Result(s,a))

EXPECTIMAX SEARCH

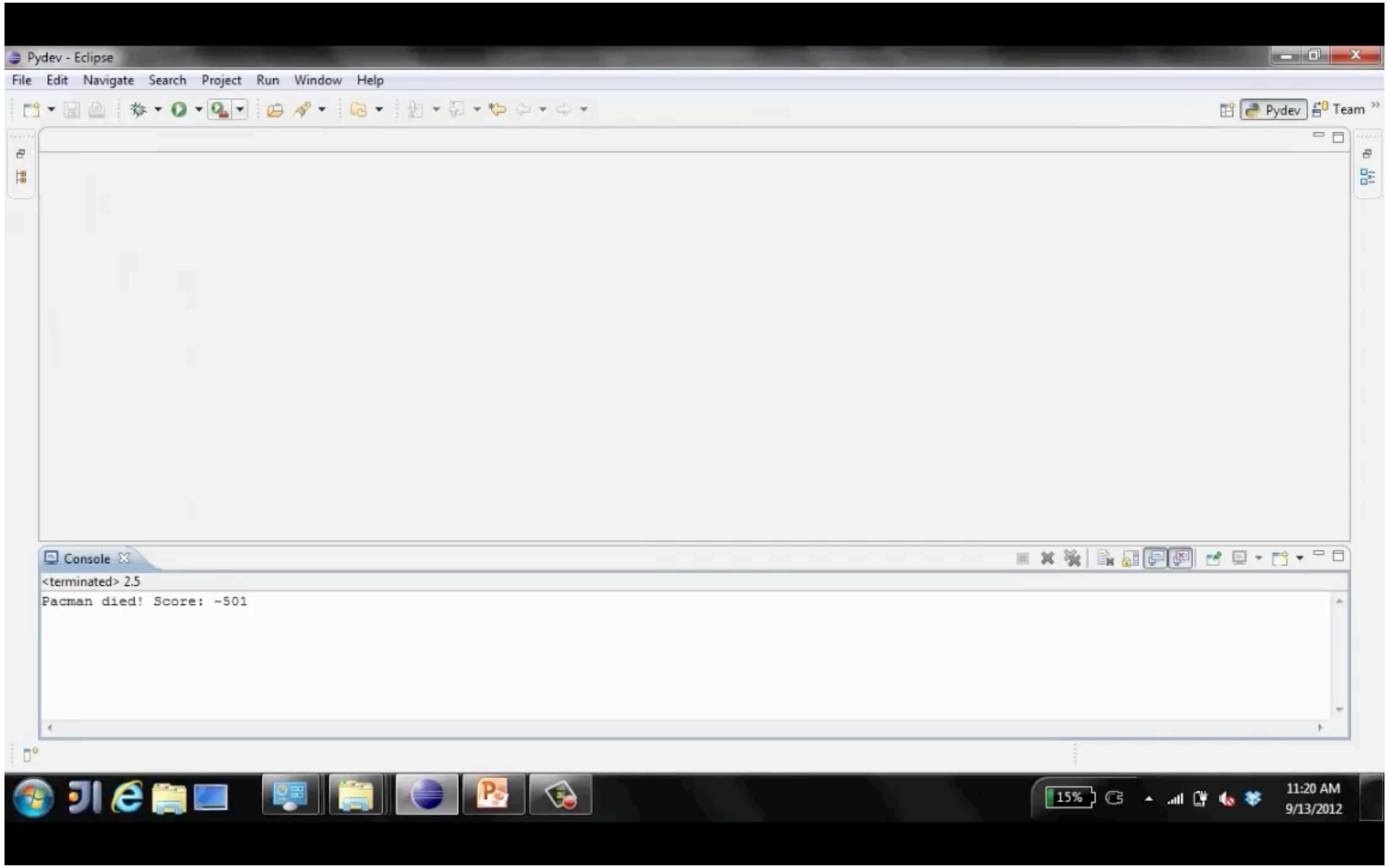
```
function decision(s) returns an action
    return the action a in Actions(s) with the highest
     value(Result(s,a))
```

```
function value(s) returns a value
    if Terminal-Test(s) then return Utility(s)
    if Player(s) = MAX       then return max_{a in Actions(s)} value(Result(s,a))
    if Player(s) = MIN       then return min_{a in Actions(s)} value(Result(s,a))
    if Player(s) = CHANCE then return sum_{r in chanceEvent(s)} Pr(r) * value(Result(s,r))
```

DEMO MINIMAX VS EXPECTIMAX

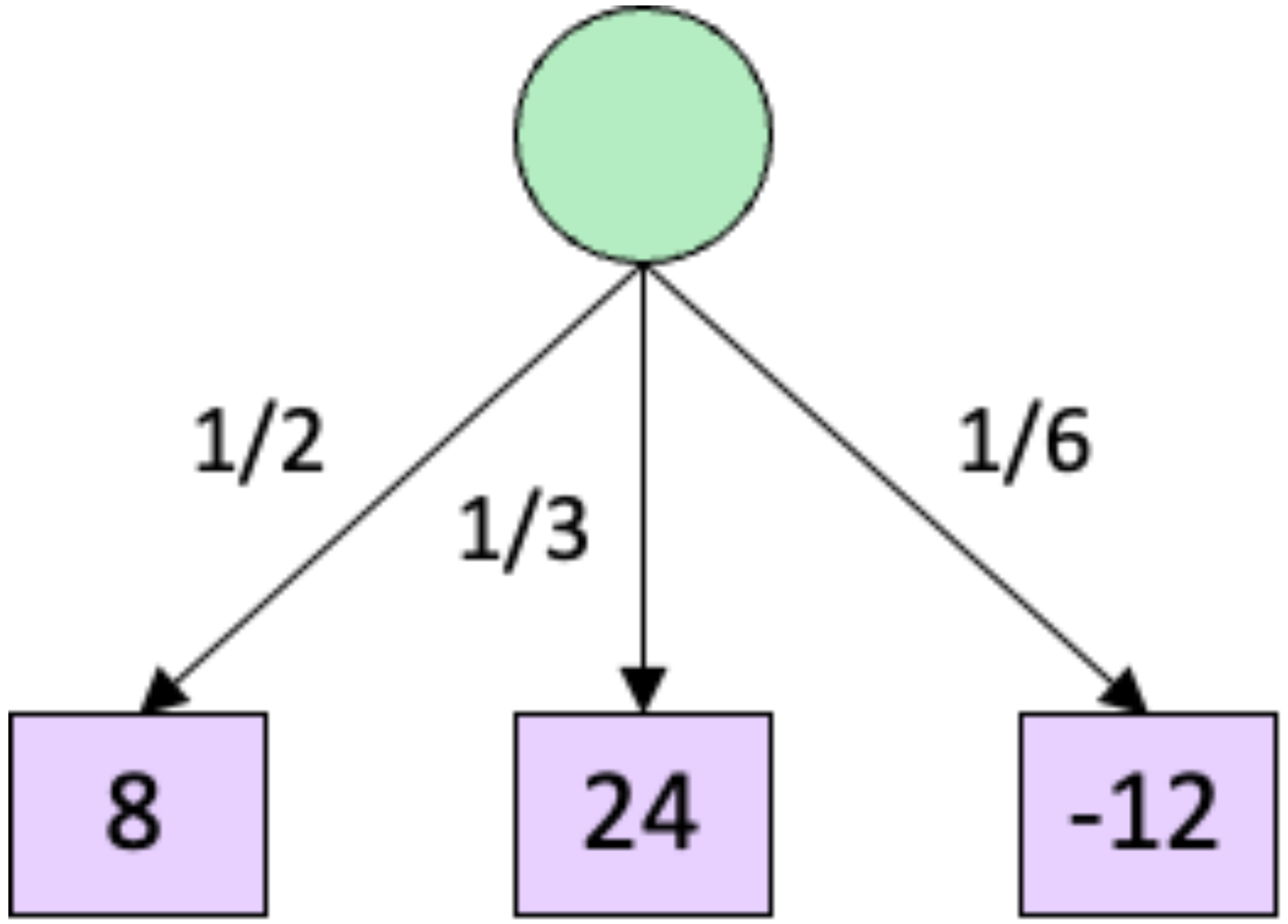


DEMO MINIMAX VS EXPECTIMAX



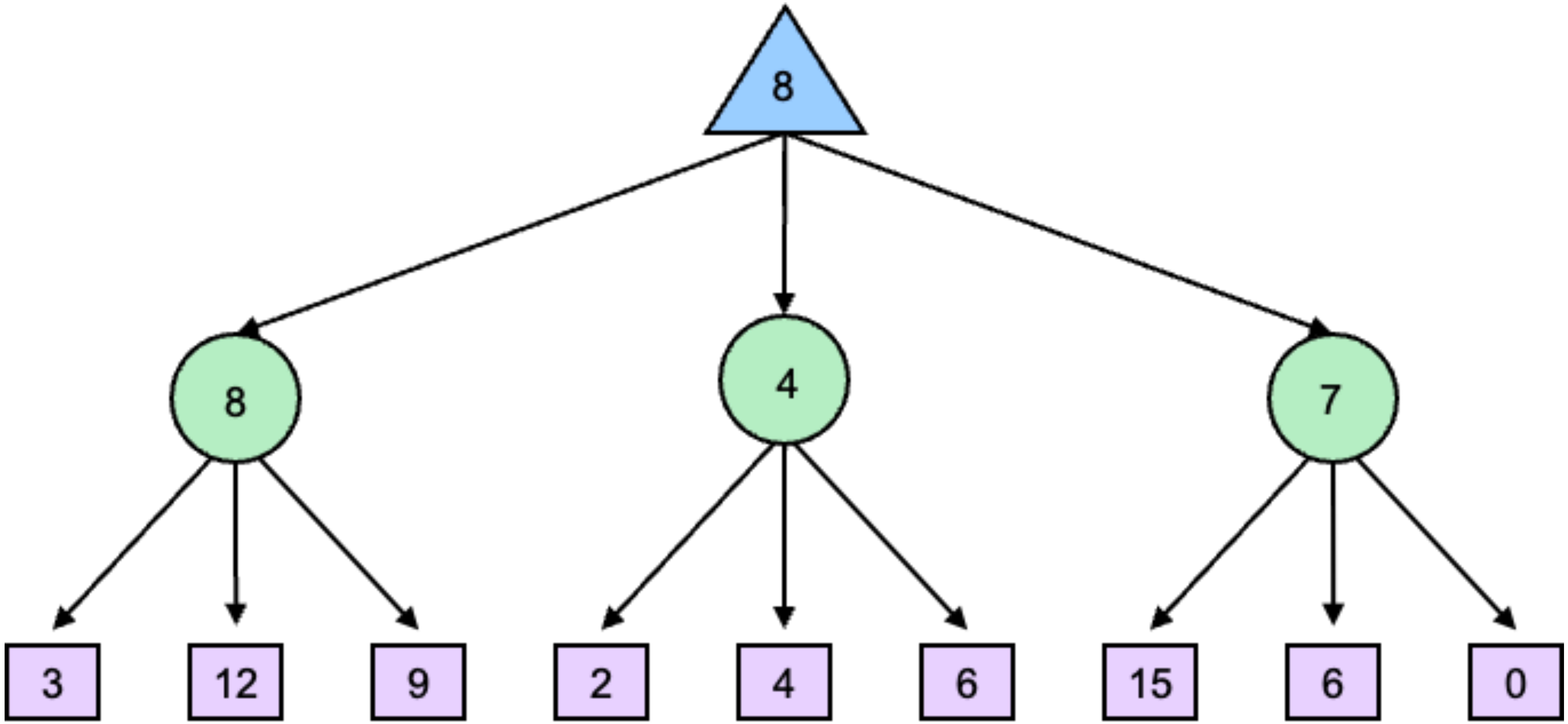
EXPECTIMAX PSEUDOCODE

```
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



SUMMARY

- Games require decisions when optimality is impossible
  - Bounded-depth search and approximate evaluation functions
- Games force efficient use of computation
  - Alpha-beta pruning
- Game playing has produced important research ideas
  - Reinforcement learning
  - Iterative deepening
  - Rational metareasoning