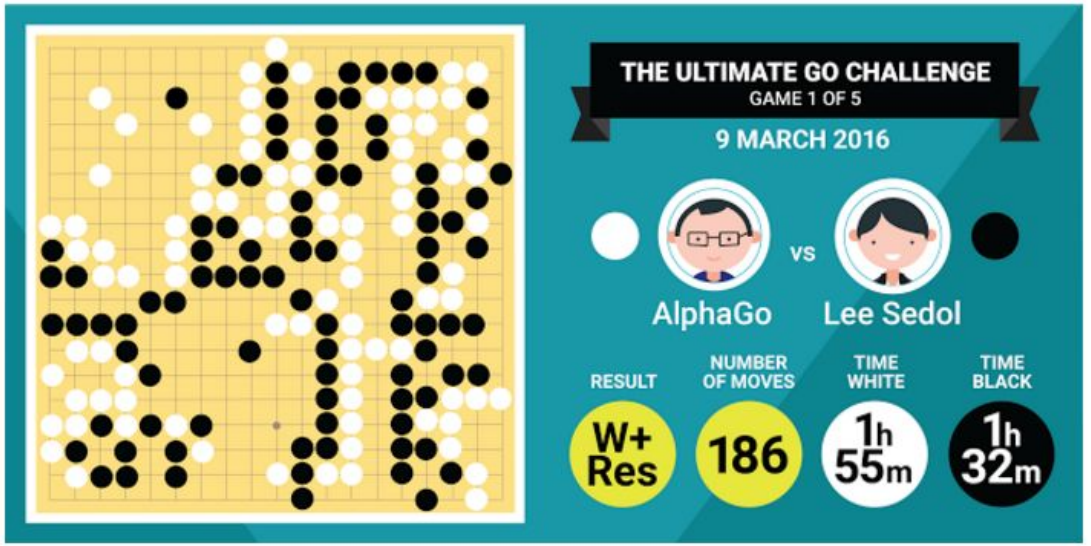
Update: http://googleasiapacific.blogspot.co.uk/

AlphaGo's ultimate challenge: a five-game match against the legendary Lee Sedol

Tuesday, March 8, 2016

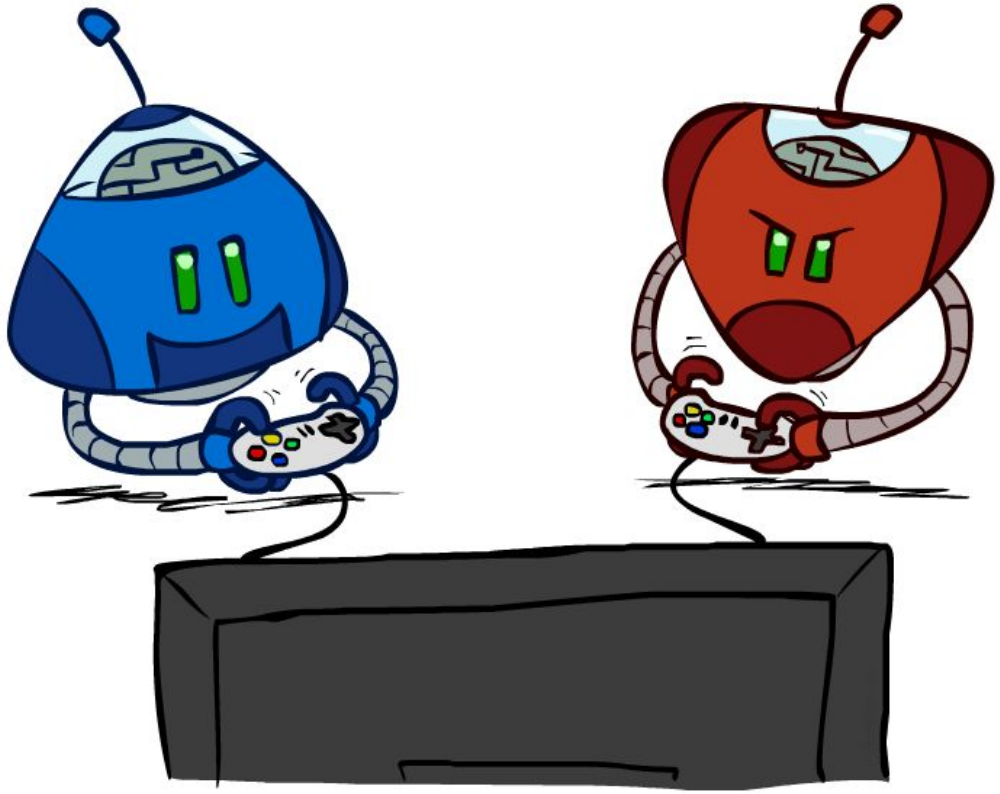


Game 1 results

Updated at 5pm KST, March 9

AlphaGo takes the first game against Lee Sedol. They were neck-and-neck for its entirety, in a game filled with complex fighting. Lee Sedol made very aggressive moves but AlphaGo did not back down from the fights. AlphaGo took almost all of its time compared to Lee Sedol who had almost 30 minutes left on the clock.

Adversarial Search



These slides are based on the slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley - http://ai.berkeley.edu.

The artwork is by Ketrina Yim.

Minimax Implementation

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the agent is MAX: return max-value(state)
    if the agent is MIN: return min-value(state)
```

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

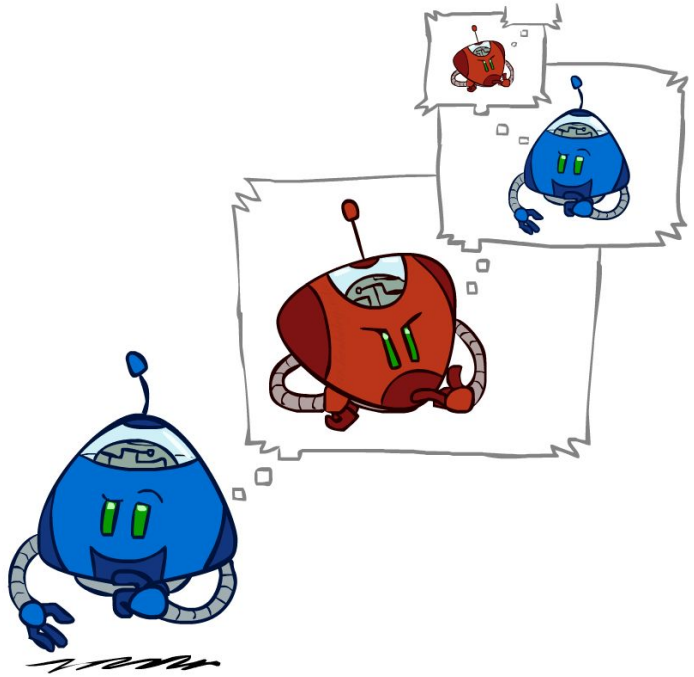
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

```
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 Properties

- Time complexity?
  - m: depth of solution - number of moves
  - Depth first search $O(b^m)$
- Space complexity?
  - O(bm) (depth-first exploration)
- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Pacman Minimax Implementation

```
def getAction(self, gameState):
  """
    Returns the minimax action from the current gameStat
    and self.evaluationFunction.
    Here are some method calls that might be useful when
    gameState.getLegalActions(agentIndex): Returns a list
      agentIndex=0 means Pacman, ghosts are >= 1
    gameState.generateSuccessor(agentIndex, action):
      Returns the successor game state after an agent takes an action
    gameState.getNumAgents(): Returns the total number of agents in the game
  """
```

Calculate the minimax value for all successor states.

Pick the action that corresponds to the maximum value of a successor state.

Similar to picking the shop with the lowest price for the order list.

Pacman Minimax Implementation

Let n be the total number of agents in the game.

(gameState.getNumAgents())

The agents take turns.

Pacman starts (agent 0) - Pacman is Max

The first ghost (agent 1) takes an action - all ghosts are min agents

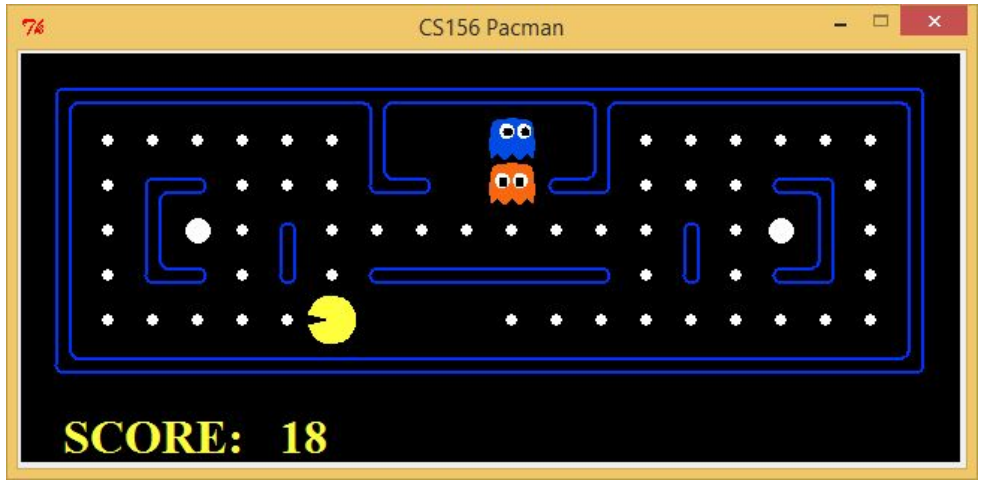
The next ghost (agent 2) takes an action - all ghosts are min agents

….

The last ghost (agent n - 1) takes an action

Pacman (agent 0) takes an action

The first ghost (agent 1) takes an action ...

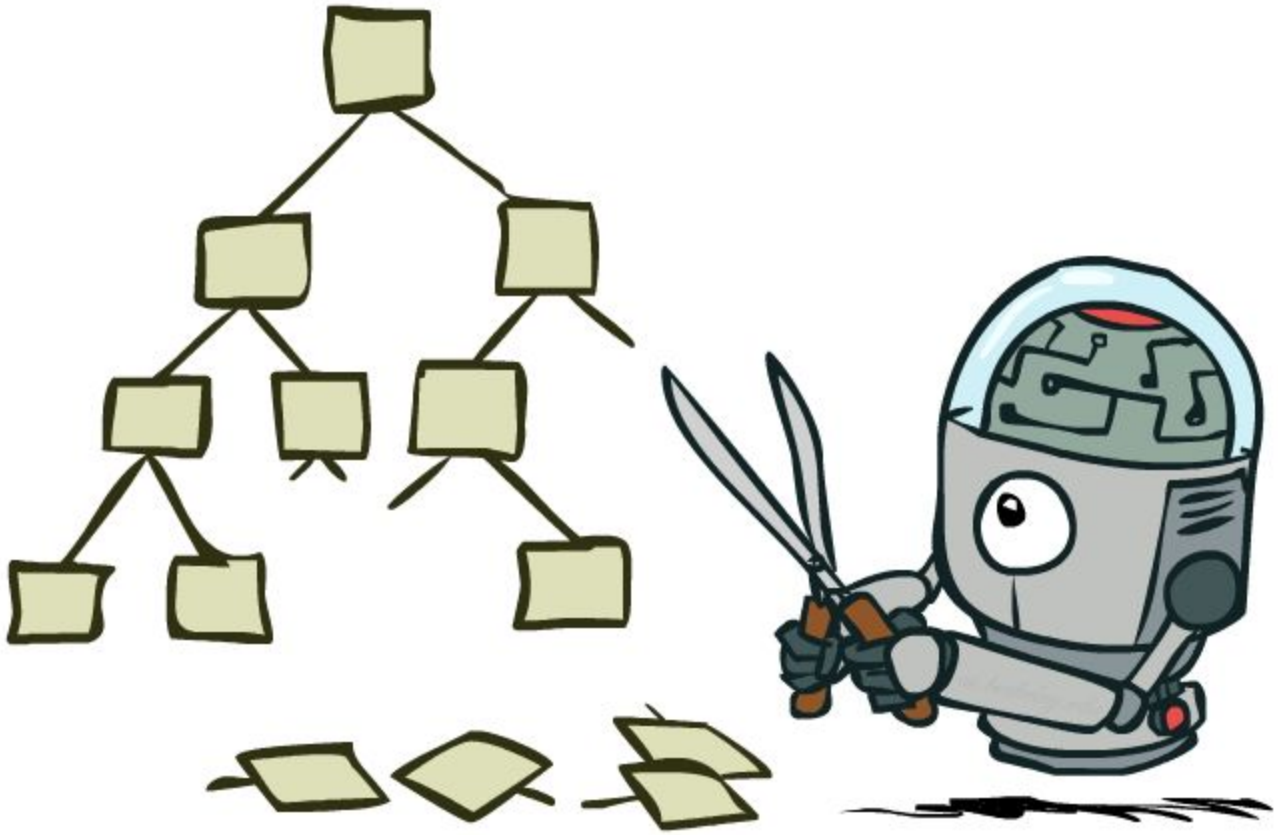


Resource Limits

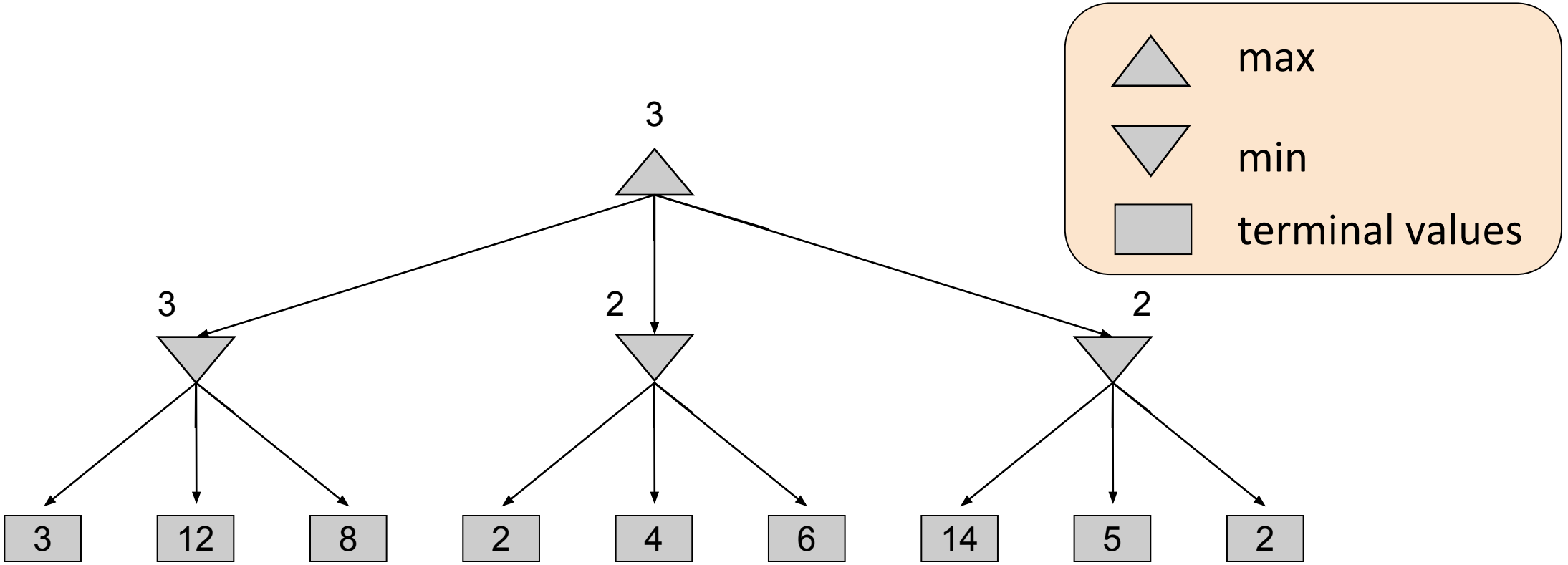
Problem: In realistic games, cannot search the whole tree!
Our resources (mostly time) are limited.

- Pruning
- Depth-limited search

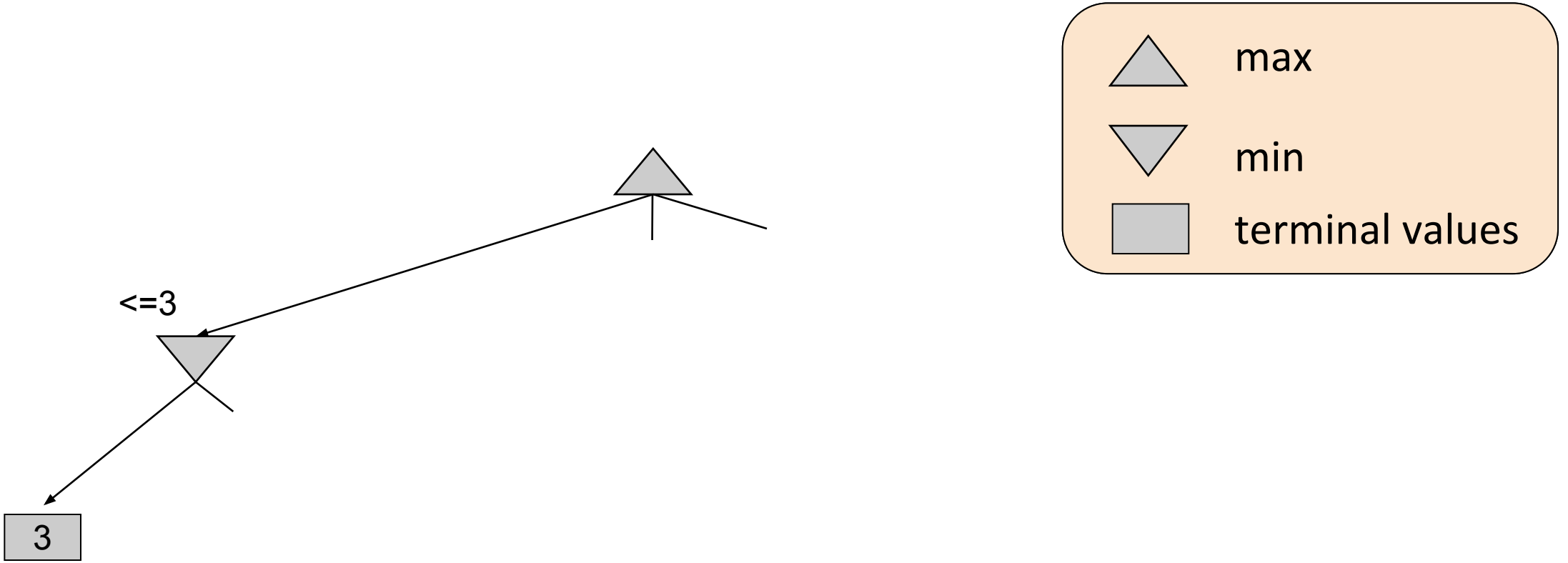
Game Tree Pruning



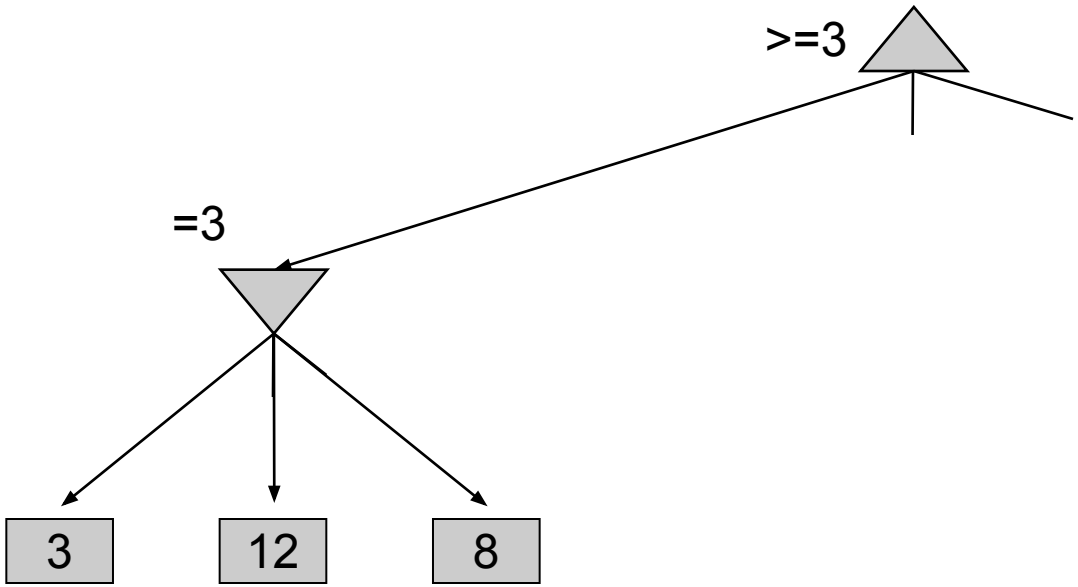
Minimax Example

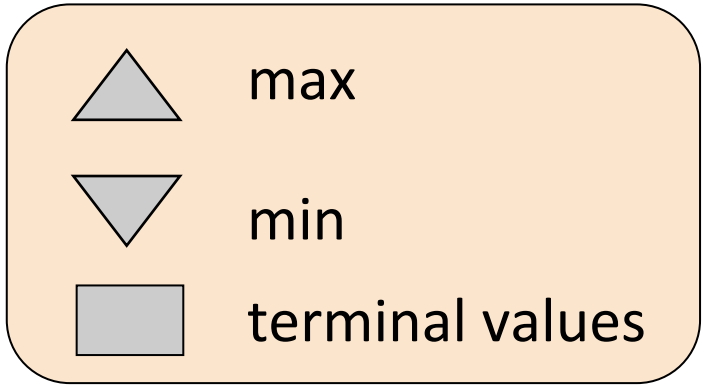


Alpha-Beta Pruning

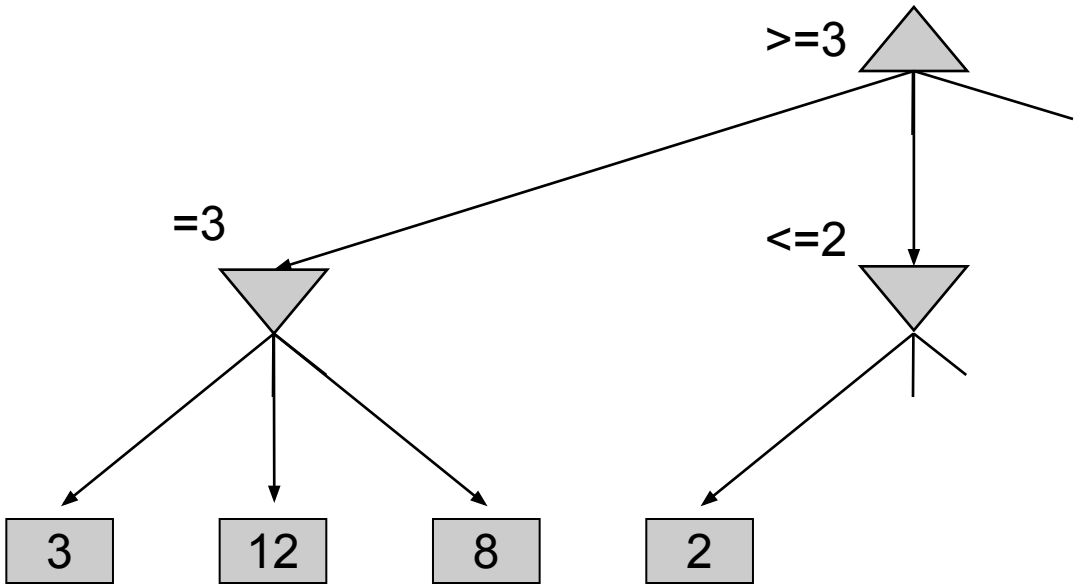


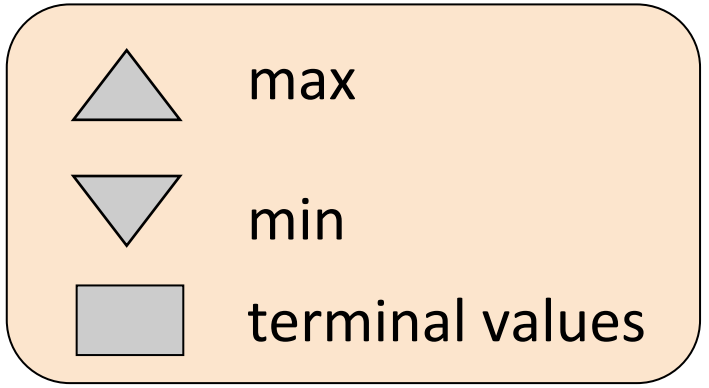
Alpha-Beta Pruning



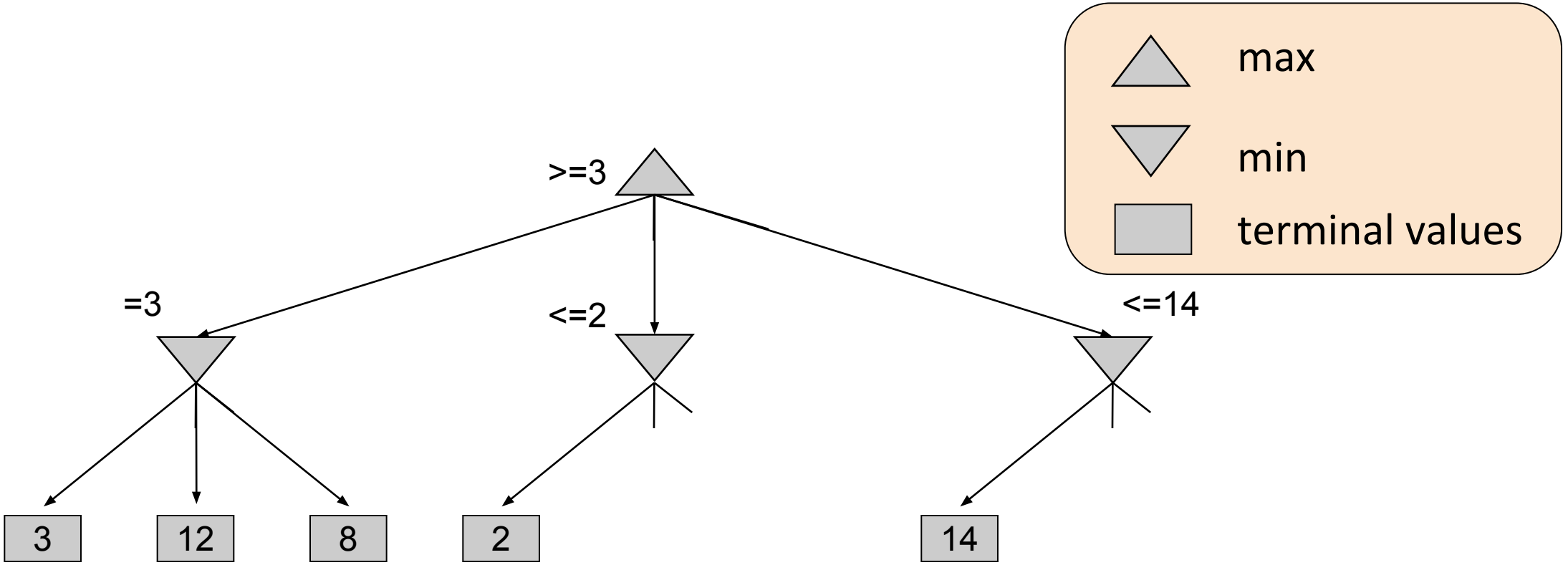


Alpha-Beta Pruning

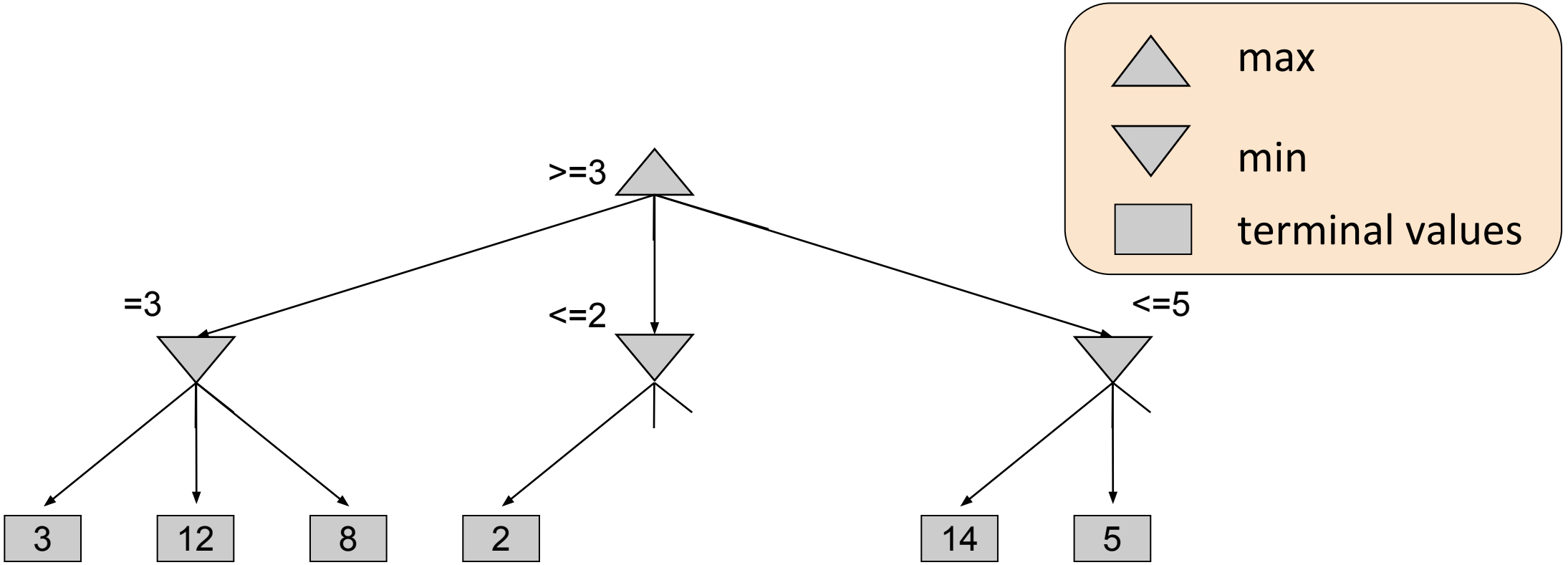




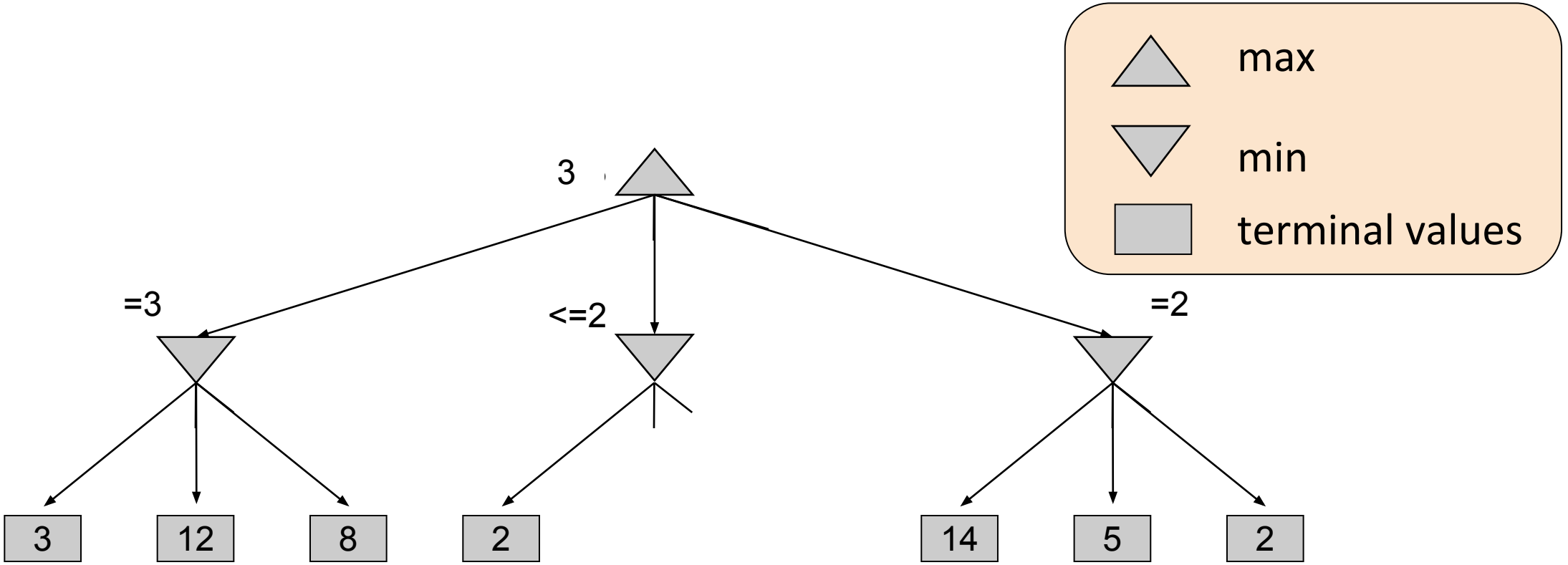
Alpha-Beta Pruning



Alpha-Beta Pruning



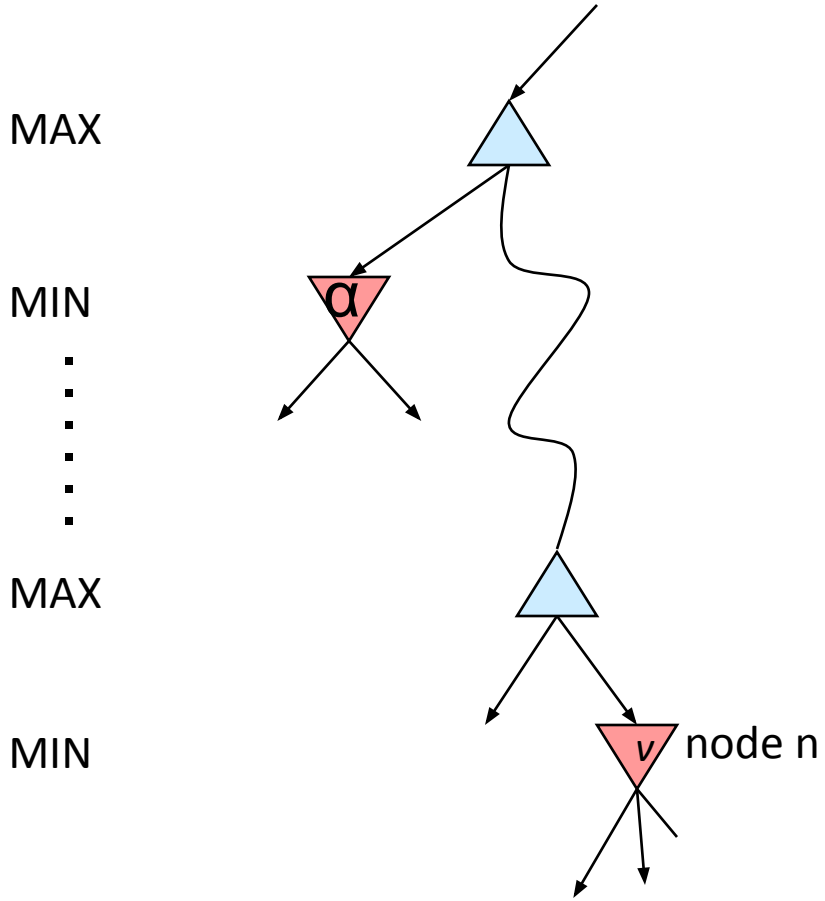
Alpha-Beta Pruning



Alpha-Beta Pruning

From MIN's perspective:

- 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 (v) is dropping
- Who cares about *n*'s value? MAX
- Let α be the best value that MAX can get at any choice point along the current path from the root
- If *n's* value (v) becomes worse than α, MAX will avoid it, so we can stop considering *n*'s other children (it's already bad enough that it won't be played)



Alpha-Beta Pruning

From MIN's perspective:

α is the best value (to max) found so far off the current path

If v is worse than α ($v < \alpha$), max will avoid it

$\Rightarrow$ prune that branch

From MAX's perspective:

β is the best value (to min) found so far off the current path

If v is worse (to min) than β ($v > \beta$), min will avoid it

$\Rightarrow$ prune that branch

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 Implementation - Homework 5

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
```

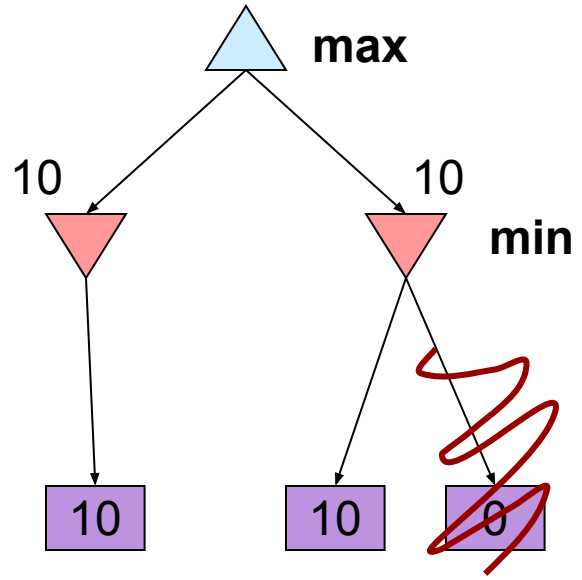
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v < α return v
        β = min(β, v)
    return v
```

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 us do action selection



Alpha-Beta Pruning Properties

Good child ordering improves effectiveness of pruning

With "perfect ordering":

- Time complexity drops to $O(b^{m/2})$
- Doubles solvable depth!
- Full search of chess, is still hopeless… 35^{50} is still impossible!

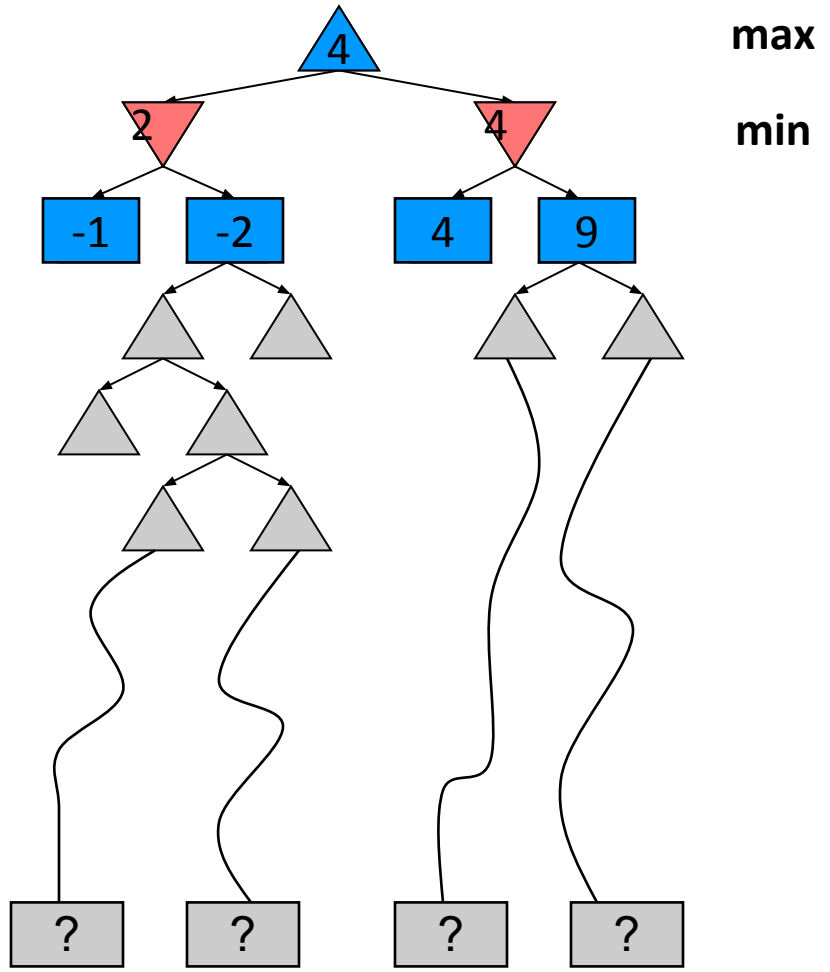
- This is a simple example of metareasoning (computing about what to compute)

Depth Limited Search

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



Alpha-Beta Pruning Properties

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

Alpha-Beta Pruning Properties

- Guarantee of optimal play is gone
- More plies (going deeper in the tree) makes a BIG difference
- Use iterative deepening for an anytime algorithm
  - start with depth 1, then depth 2, etc…
  - when we run out of time, we have a move
  - the more time we get, the better our move gets

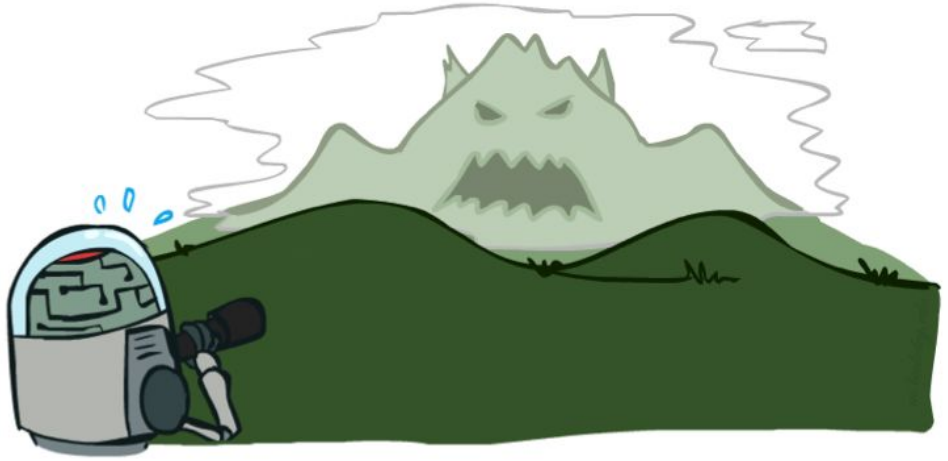
Minimax Revisited - Depth Limited

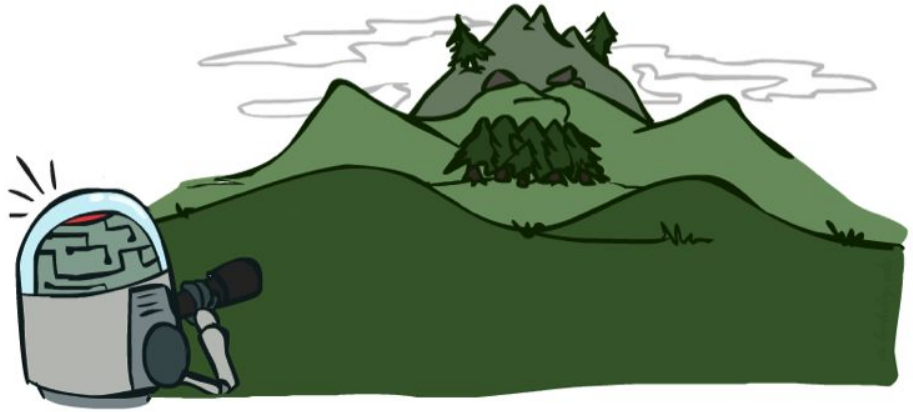
```
def value(state):
    if the state is a terminal state or maximum depth exceeded:
        return the state's utility
    if the agent is MAX: return max-value(state)
    if the agent is MIN: return min-value(state)
```

Need to add some bookkeeping to keep track of the depth.

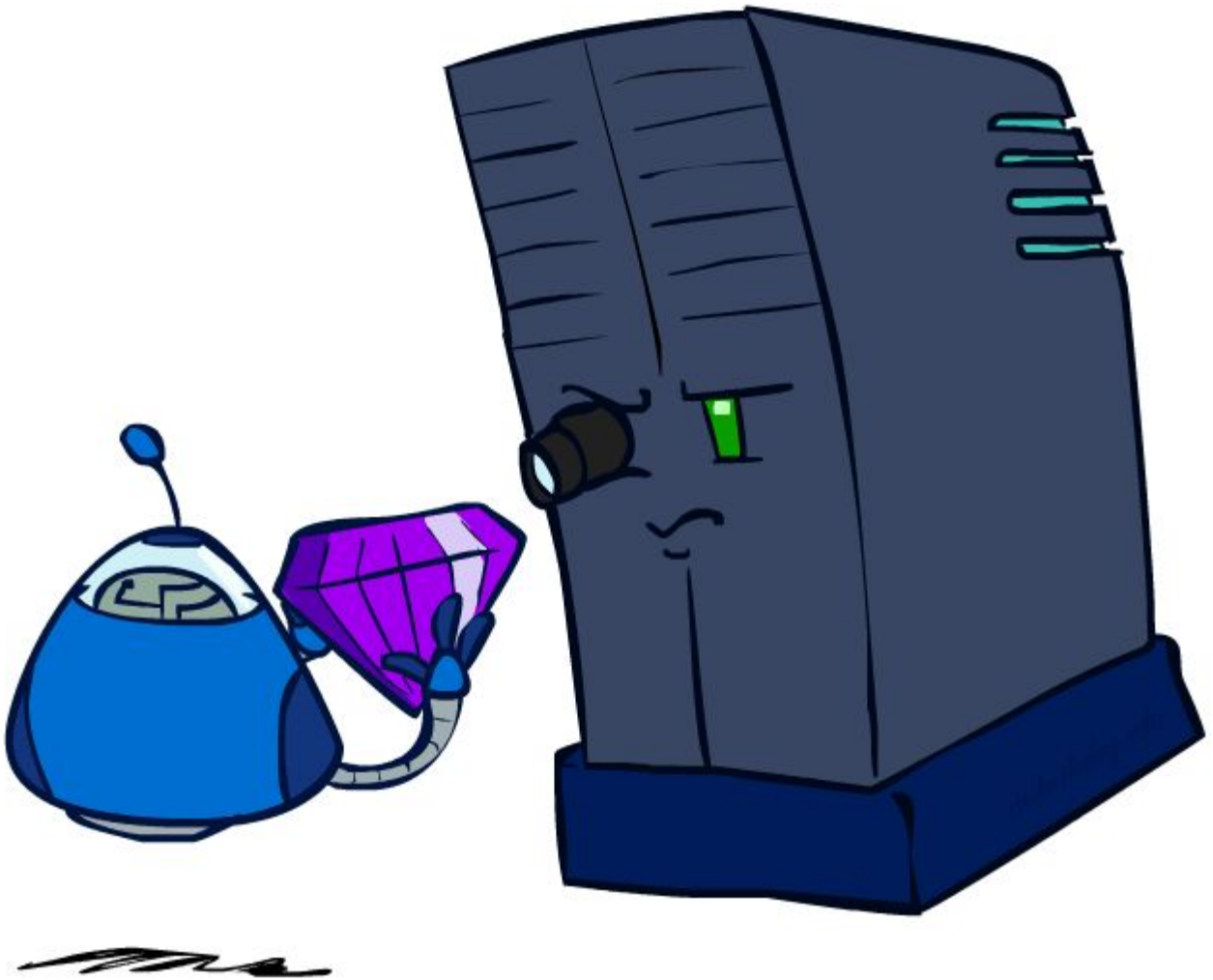
Depth Matters

- 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





Evaluation Functions

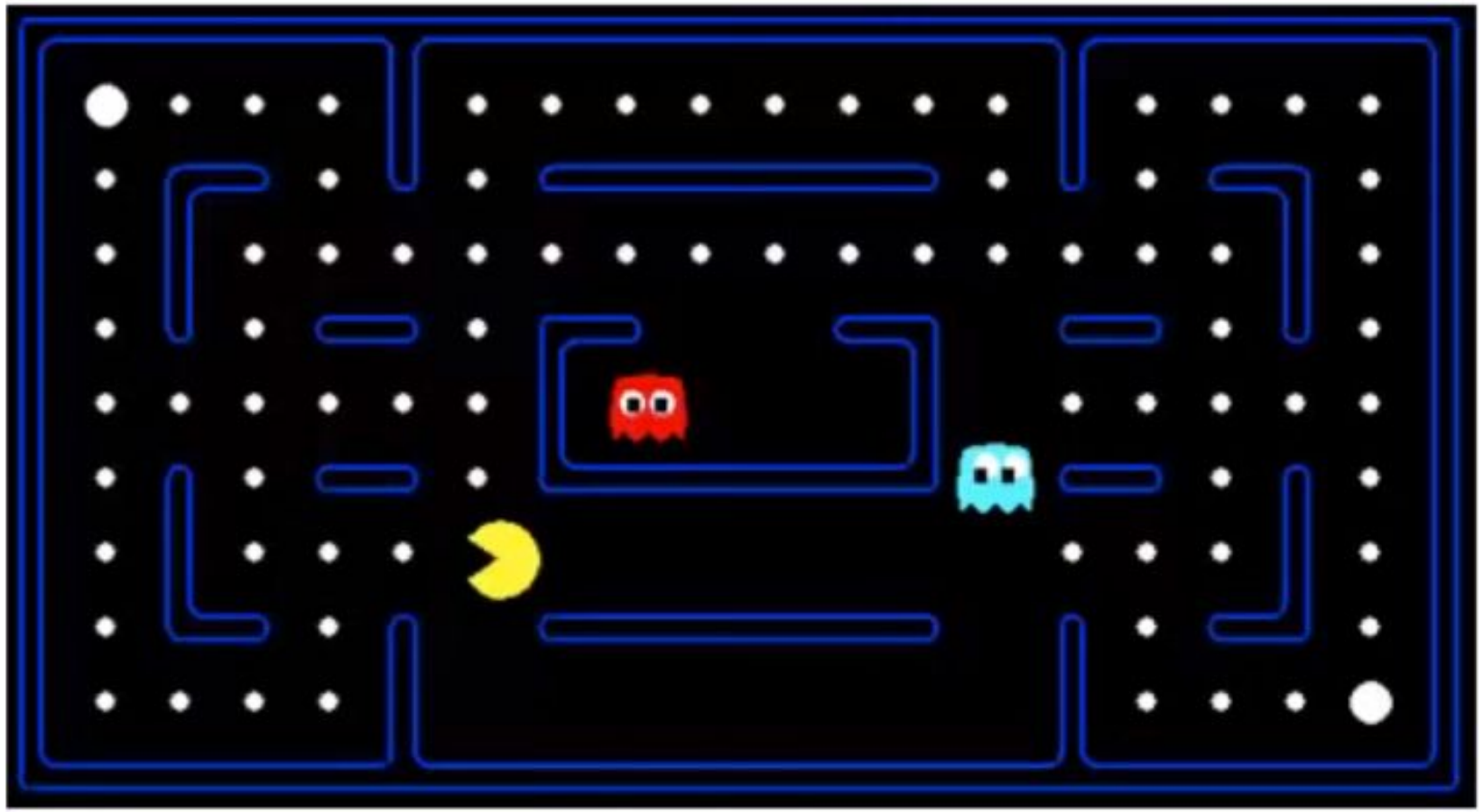


Evaluation Functions

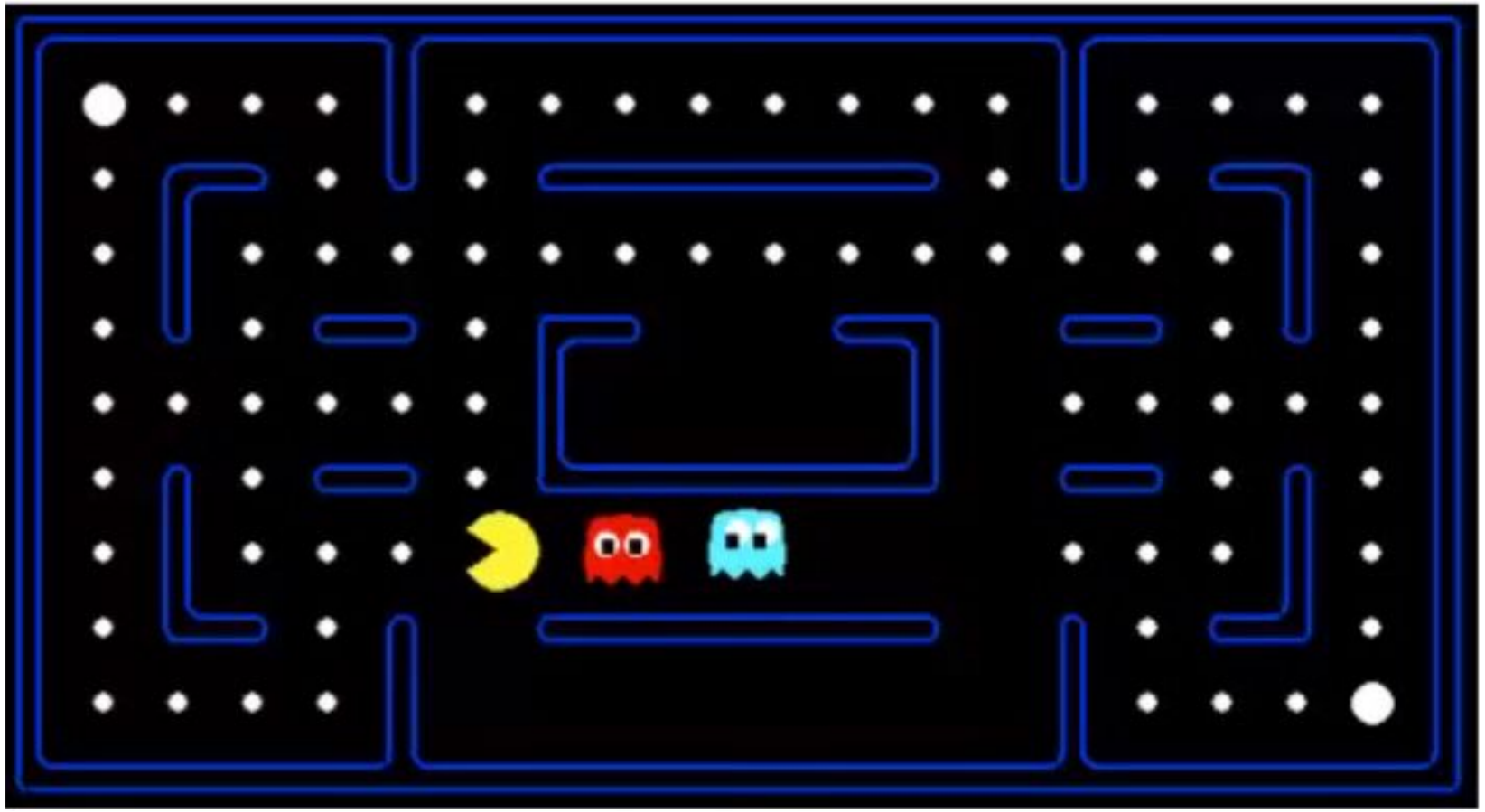
- Evaluation functions score non-terminals in depth-limited search
  - an evaluation function takes a state and returns a number that is an estimate of the minimax value.
- Ideal function: returns the actual minimax value of the state
- In practice: typically weighted linear sum of features:

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

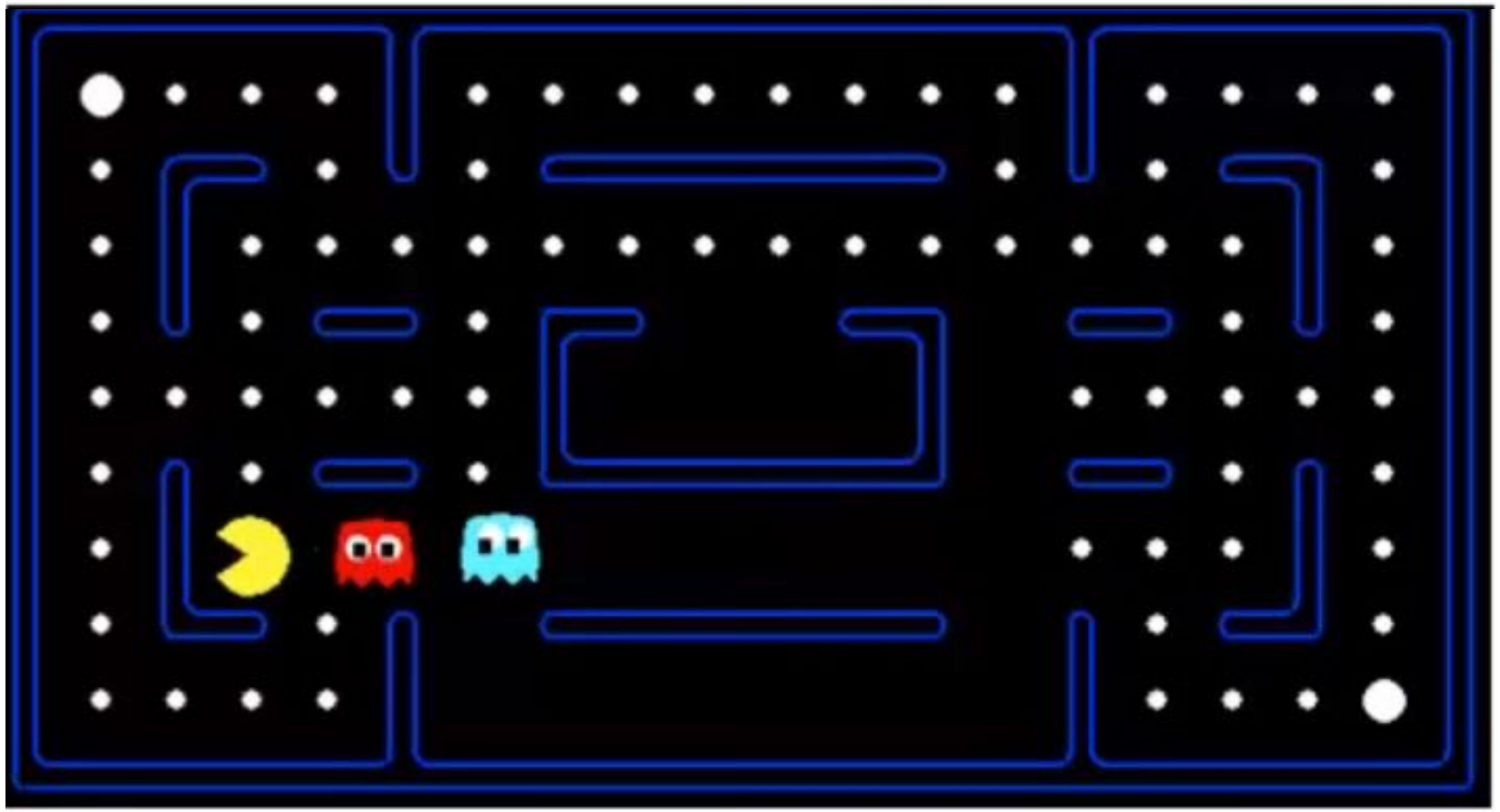
Evaluation for Pacman



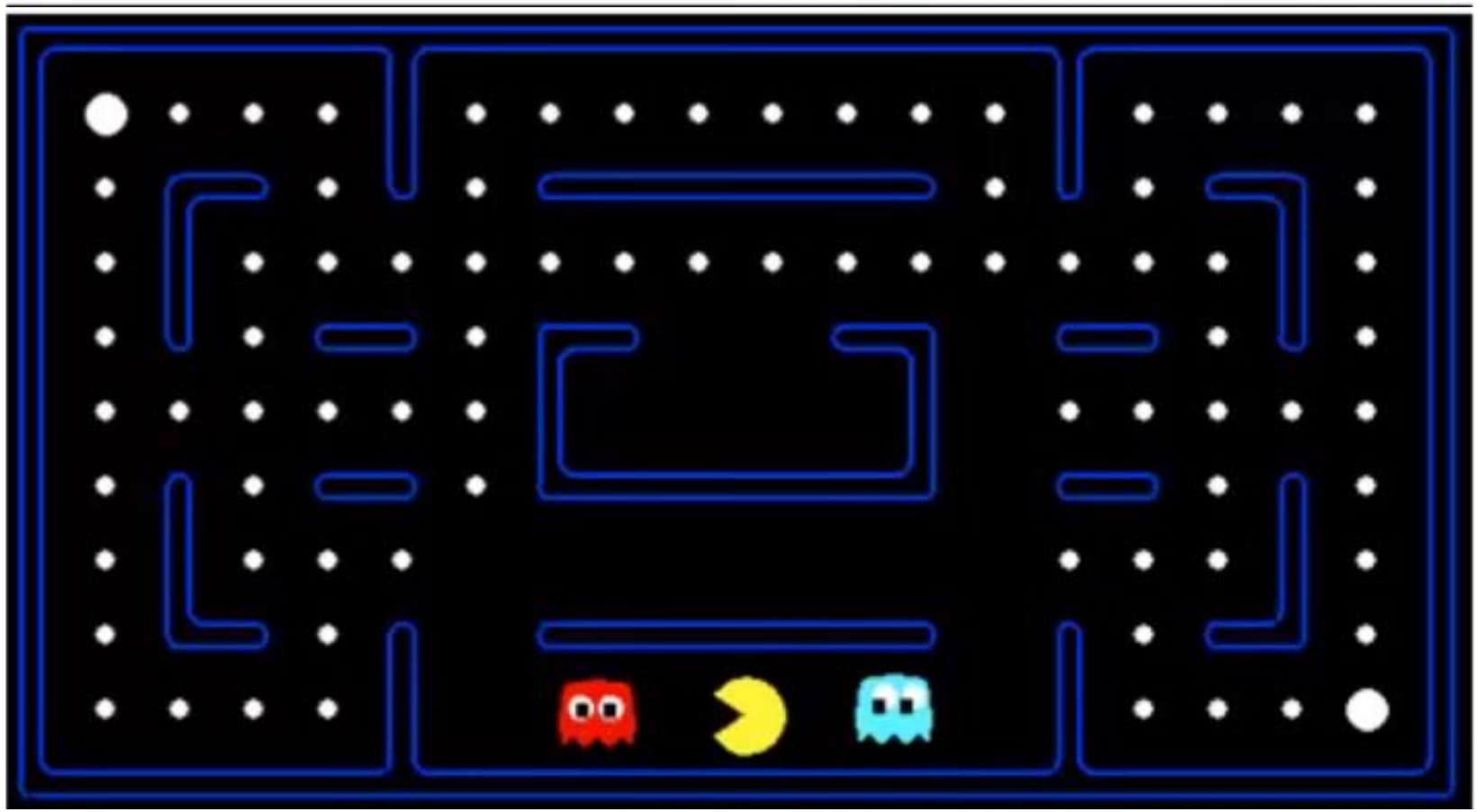
Evaluation for Pacman



Evaluation for Pacman



Evaluation for Pacman

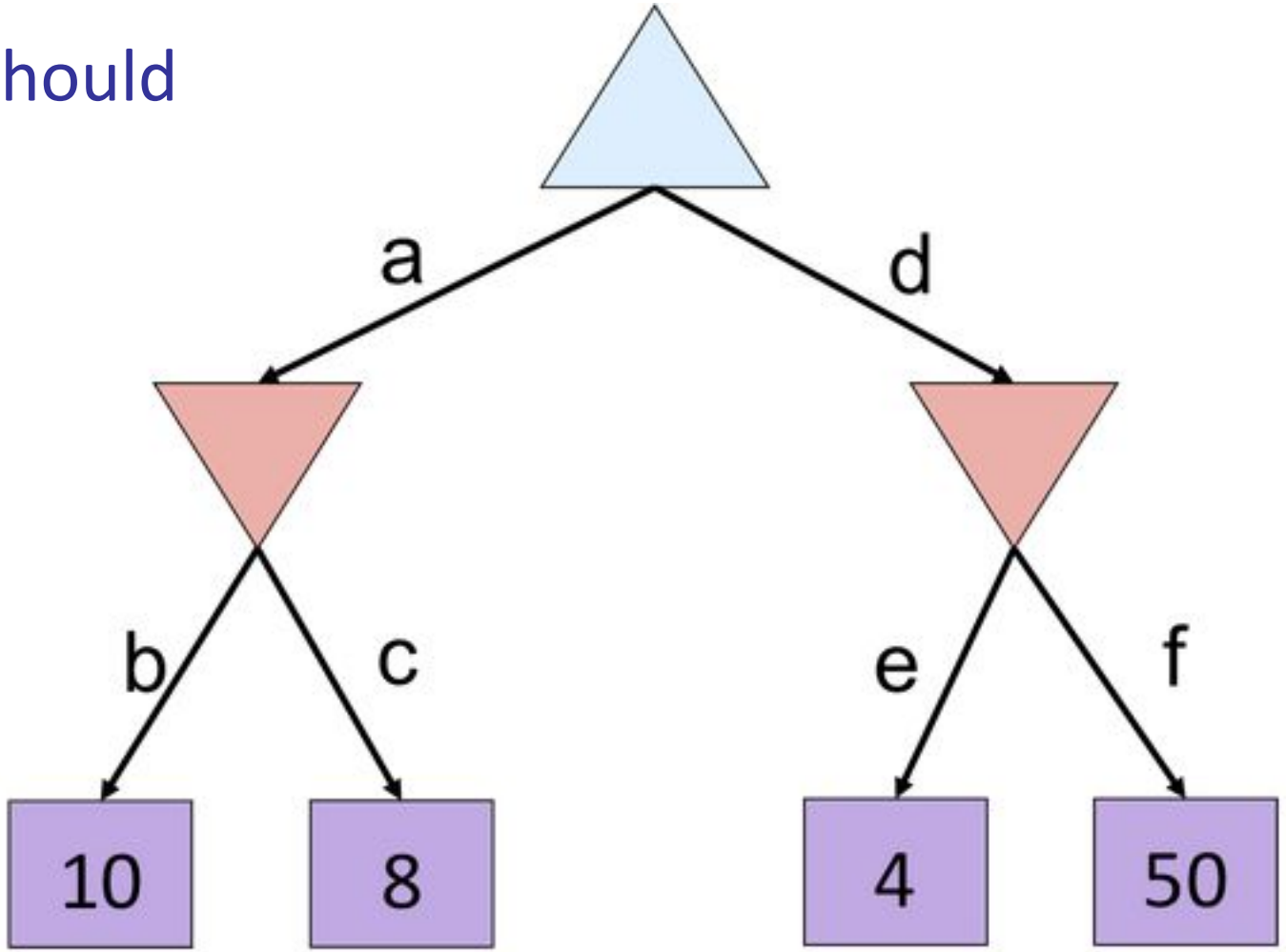


Minimax

Which action should max choose?

a.

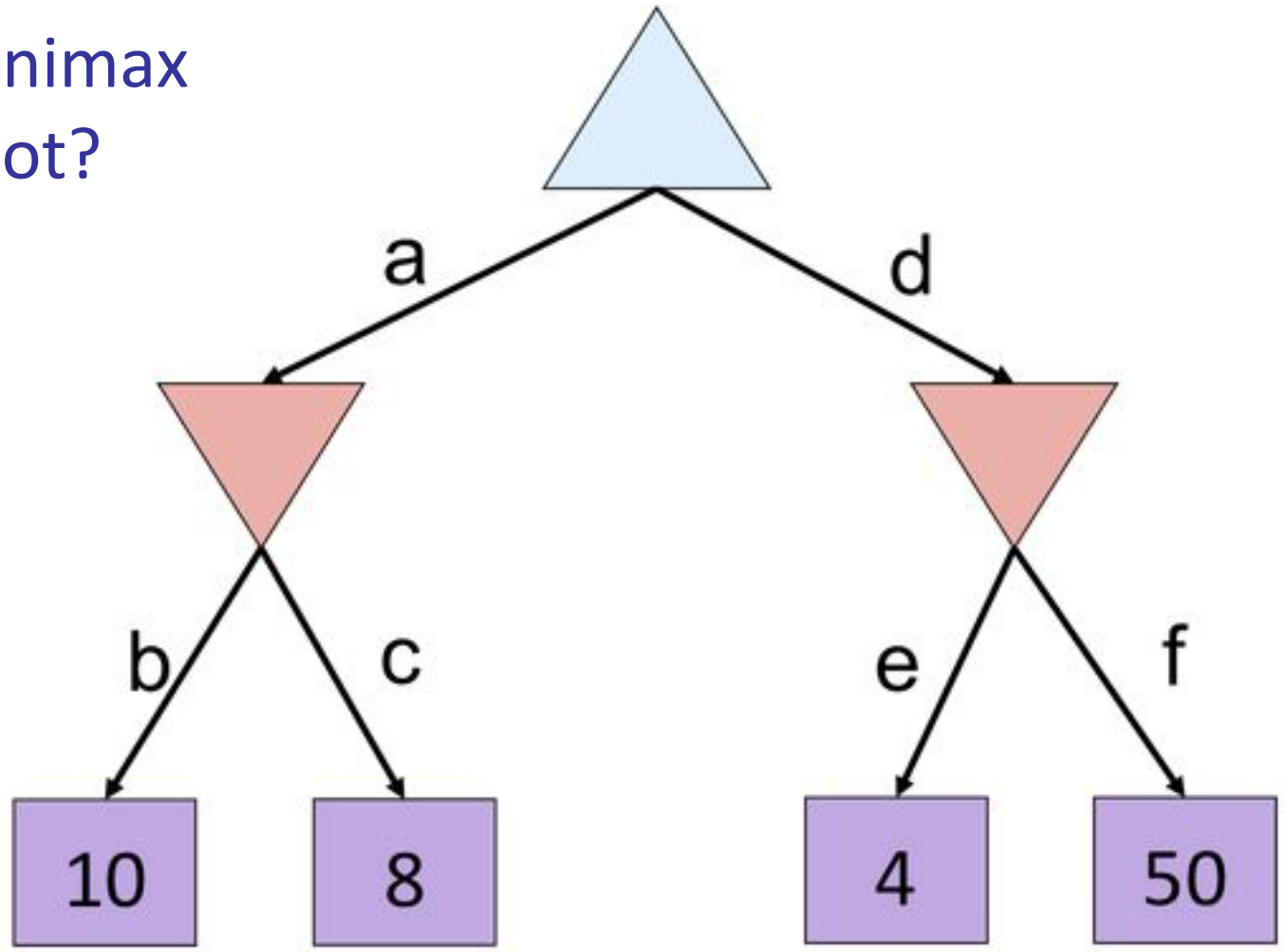
d.



Minimax

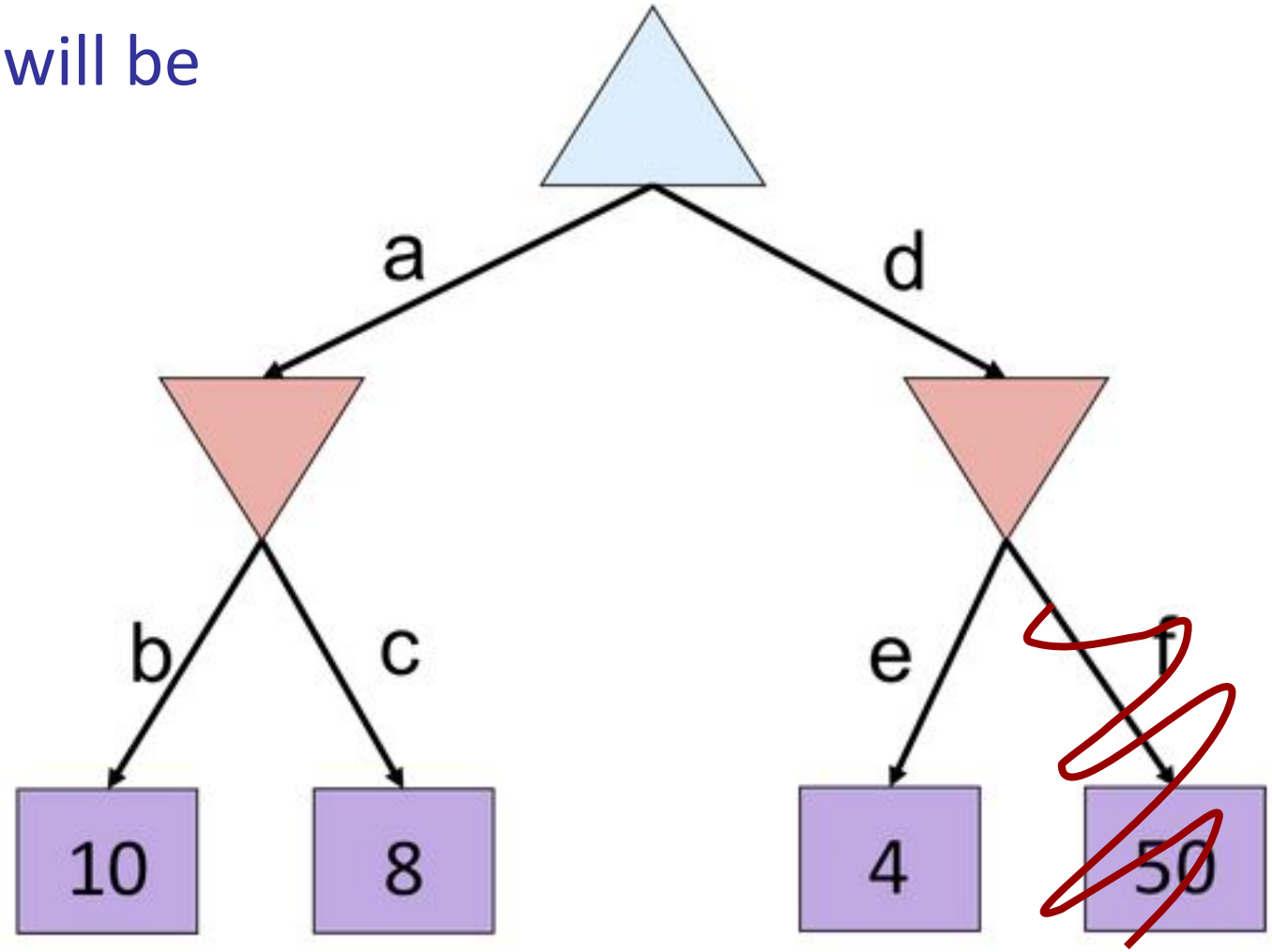
What is the minimax value at the root?

A. 10
B. 8
C. 4
D. 50



Alpha-Beta

Which branch will be pruned?



Why Games?

Games illustrate several important points about AI

- perfection is unattainable ⇒ must approximate
- good idea to think about what to think about