Minimax Implementation

Recursive Approach

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    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)
```

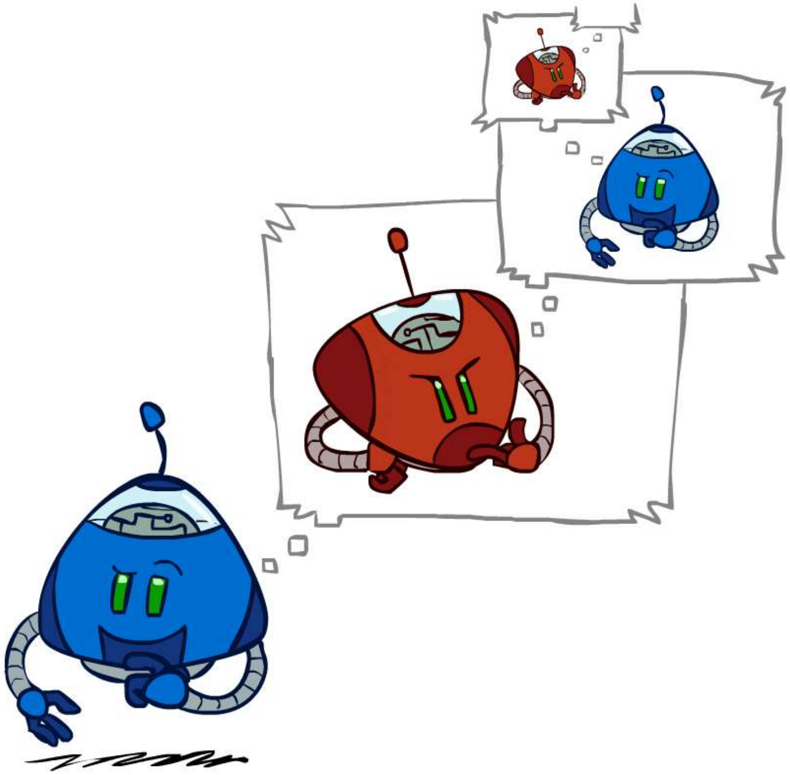
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

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

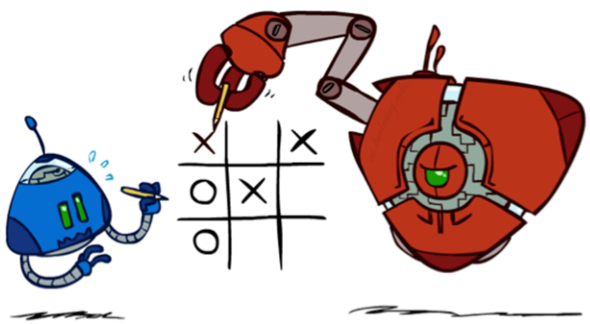
Minimax Efficiency

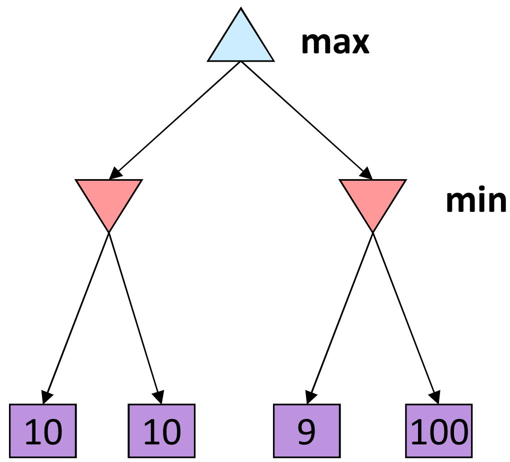
- How efficient is minimax?
  - Complete: if tree is finite
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$(why?)
  - Space: $O(bm)$(why?)

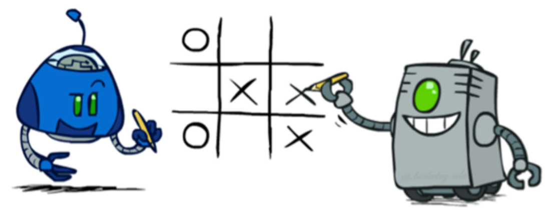
- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Minimax Properties





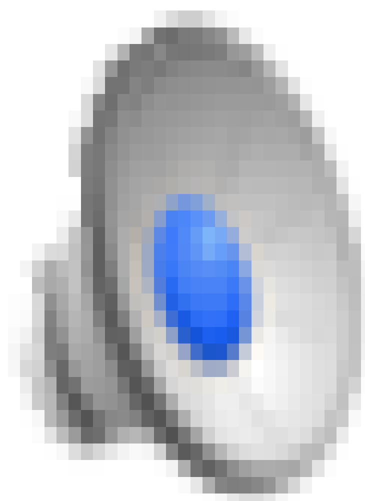


Optimal against a perfect player. Otherwise?

[Demo: min vs exp (L6D2, L6D3)]

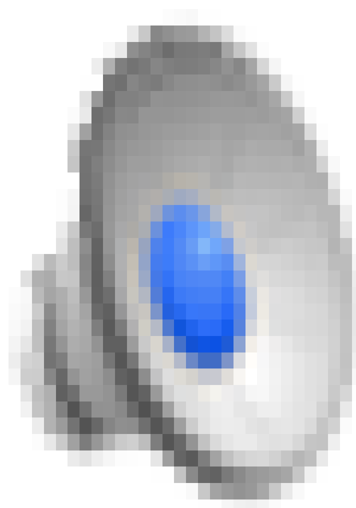
Video of Demo Min vs. Exp (Min)

Ghosts are intelligence



Video of Demo Min vs. Exp (Exp)

Ghost are (some time) in Random Action



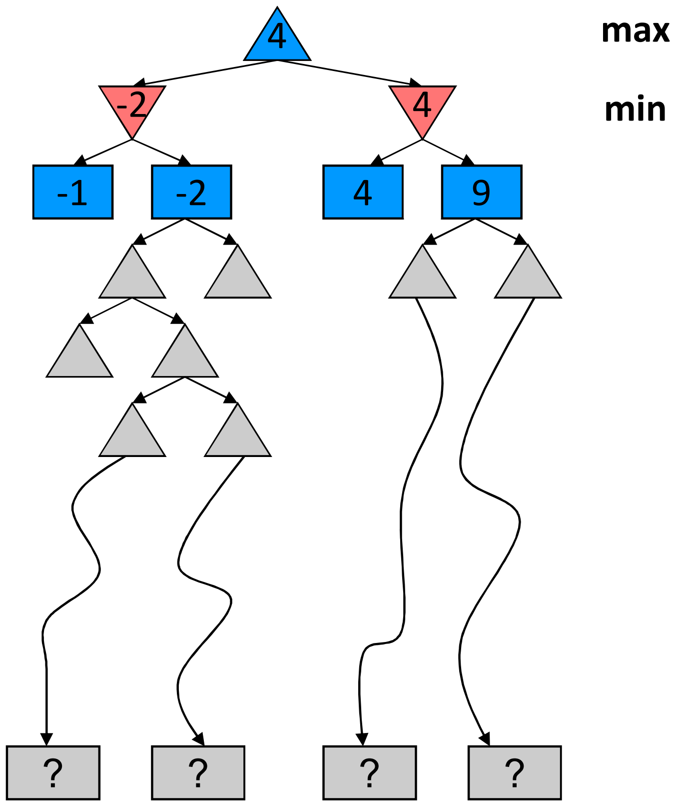
Resource Limits

Challenge: Resource Limits

- Problem: In realistic games, cannot search to leaves!

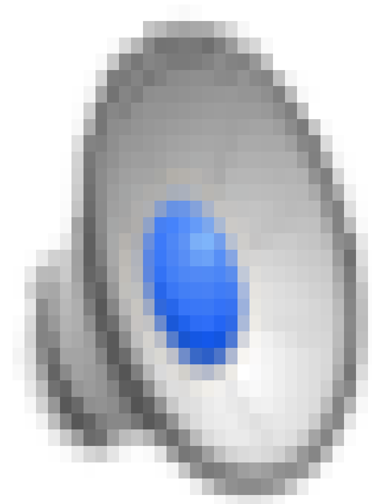
Usually search is Time limited!!

- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree (is it Ok?)
  - 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(if we have time search in new depth)



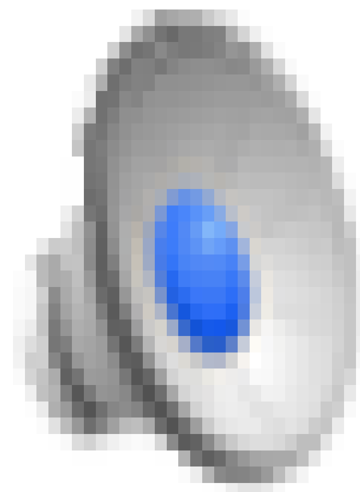
Video of Demo Limited Depth (2) Low value

What happen if we
search just 2 level

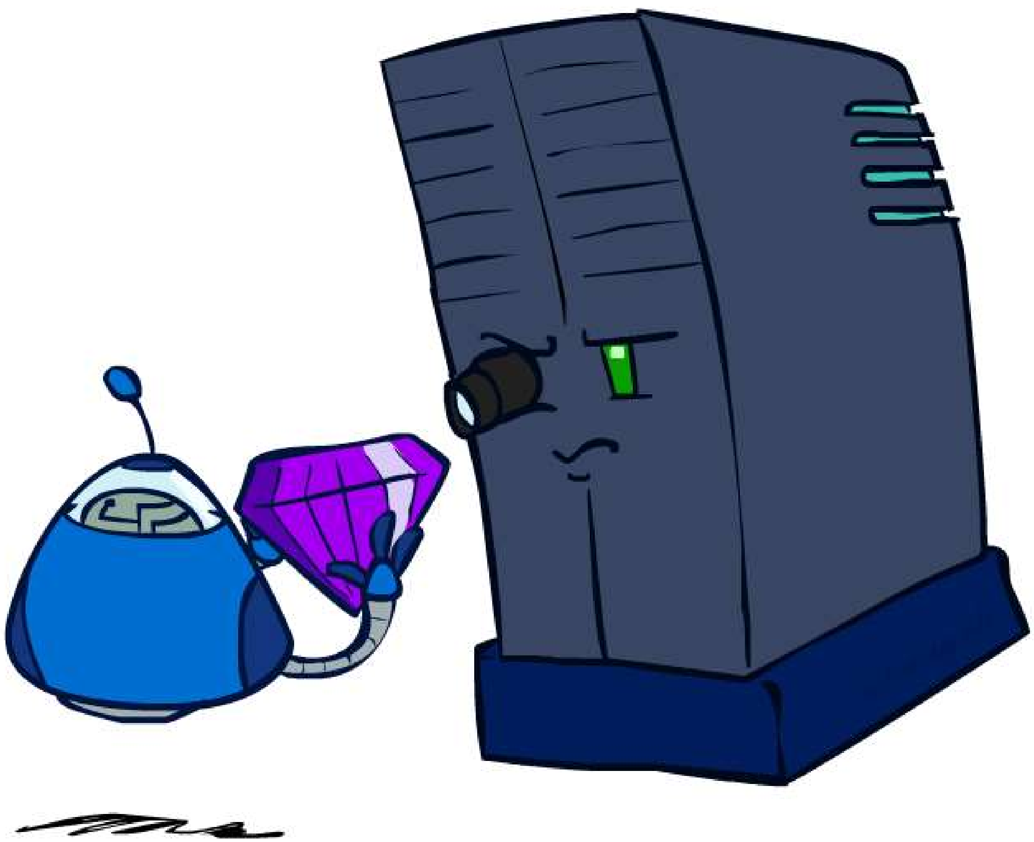


Video of Demo Limited Depth (10) Good value

What happen if we
search all state

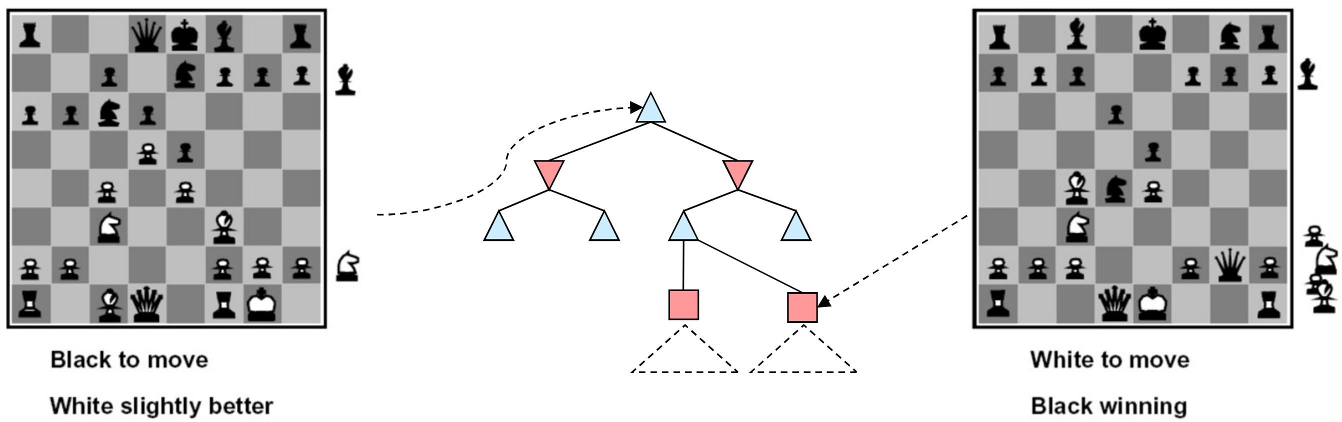


Evaluation Functions



Evaluation Functions

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



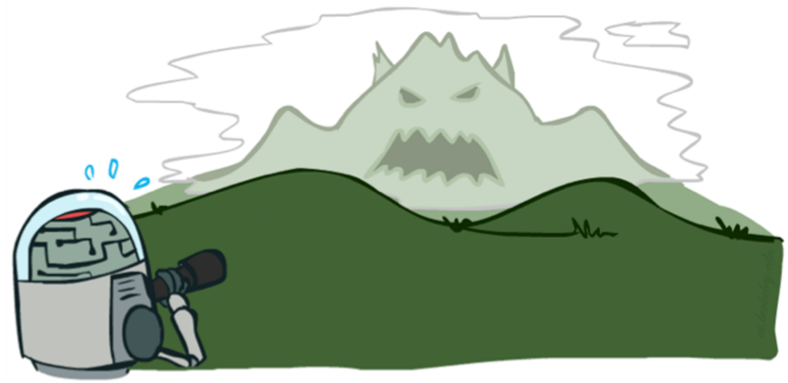
- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

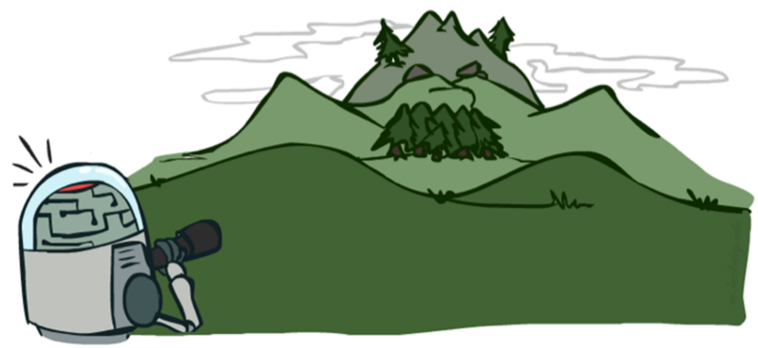
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Depth Matters

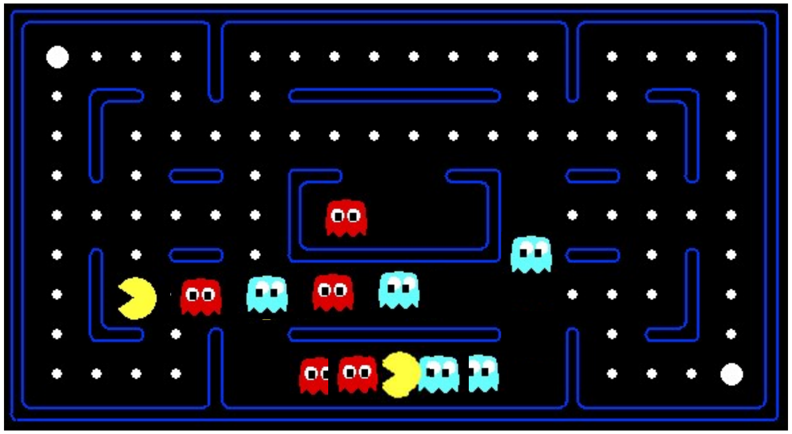
- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation





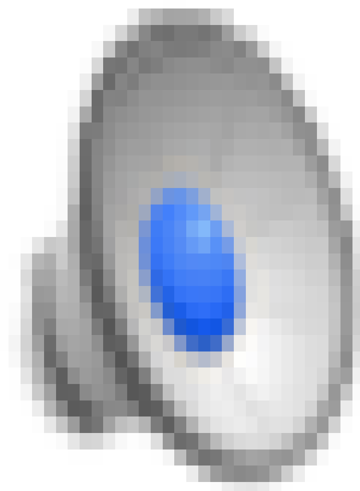
[Demo: depth limited (L6D4, L6D5)]

Evaluation for Pacman

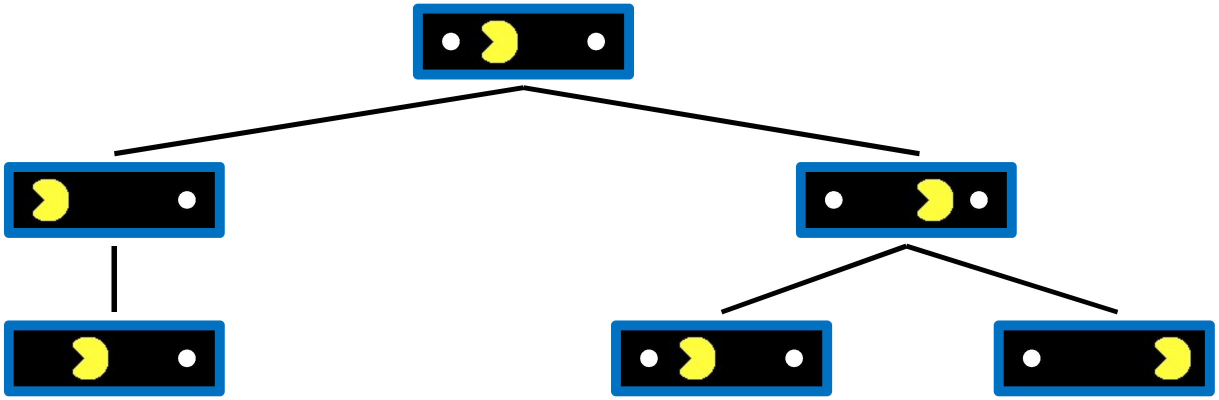


[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function), smart ghosts coordinate (L6D6,7,8,10)]

Video of Demo Thrashing (d=2)

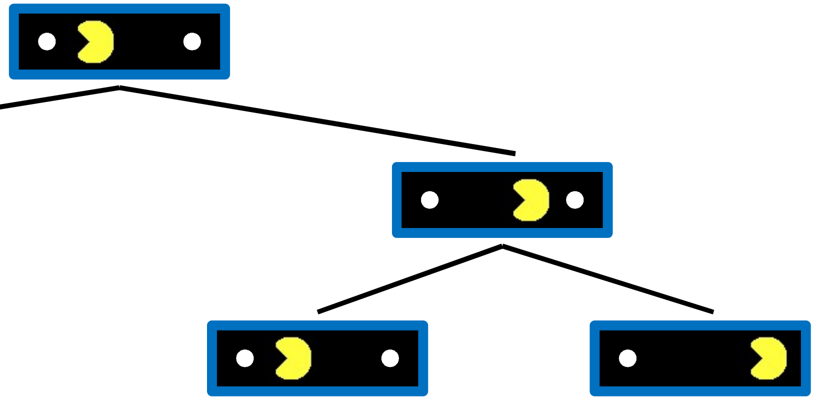


Why Pacman Starves



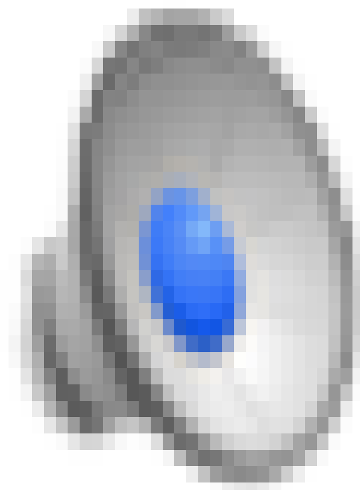
- assume
  - evaluation function -> 10 * number of eaten dots
  - Search in depth 2
  - What is Value of Root
  - If select right
    - Start New Search
    - what is value of root? (What happen If select left)
- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, 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!

Why Pacman Starves

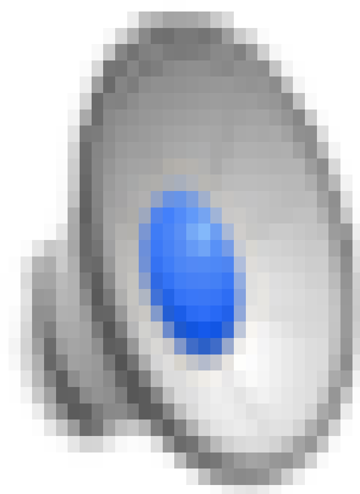


- assume
  - evaluation function -> 10 * number of eaten dots
  - Search in depth 2
  - What is Value of Root
  - If select right
    - Start New Search
    - what is value of root? (What happen If select left)
- Solution
  - Distance to dot

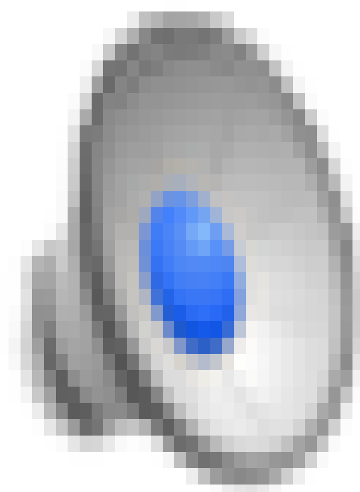
Video of Demo Thrashing -- Fixed (d=2)



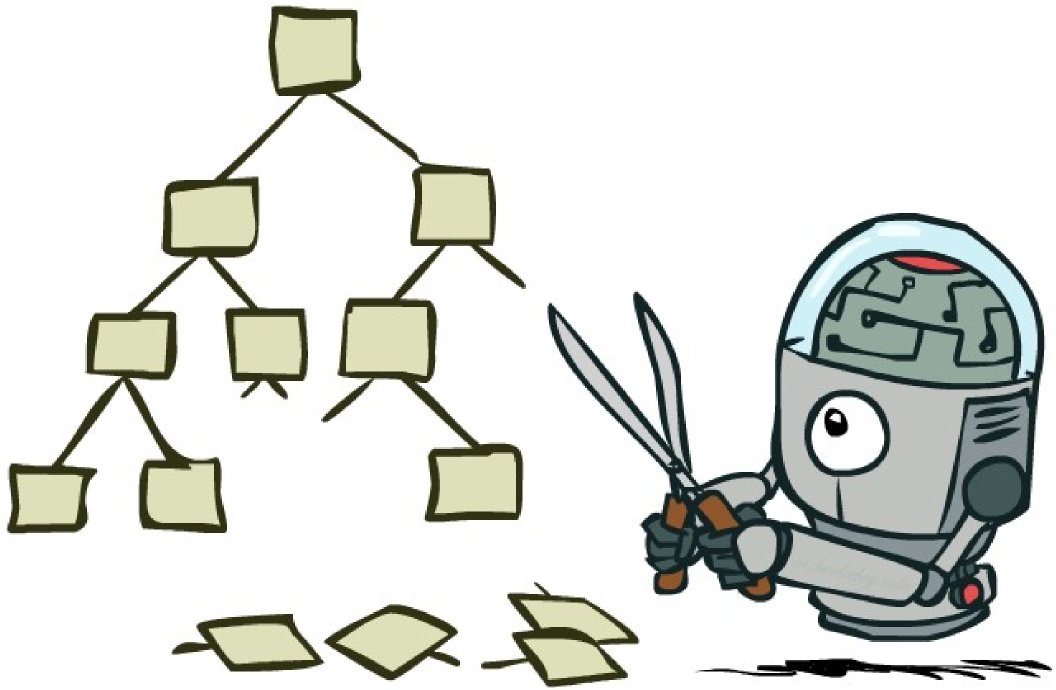
Video of Demo Smart Ghosts (Coordination)



Video of Demo Smart Ghosts (Coordination) – Zoomed In

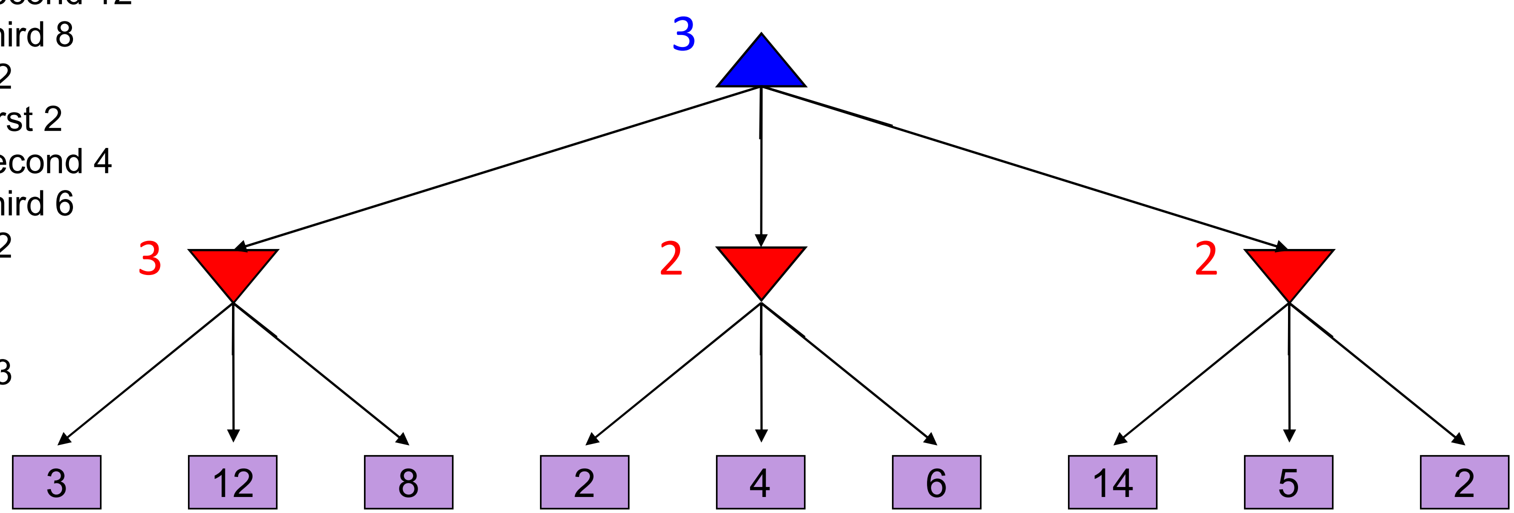


Game Tree Pruning



Minimax Example

1. Min Search
    1. Select 3
        1. First 3
        2. Second 12
        3. Third 8
    2. Select 2
        1. First 2
        2. Second 4
        3. Third 6
    3. Select 2
        …
2. Max Search
    1. Select 3

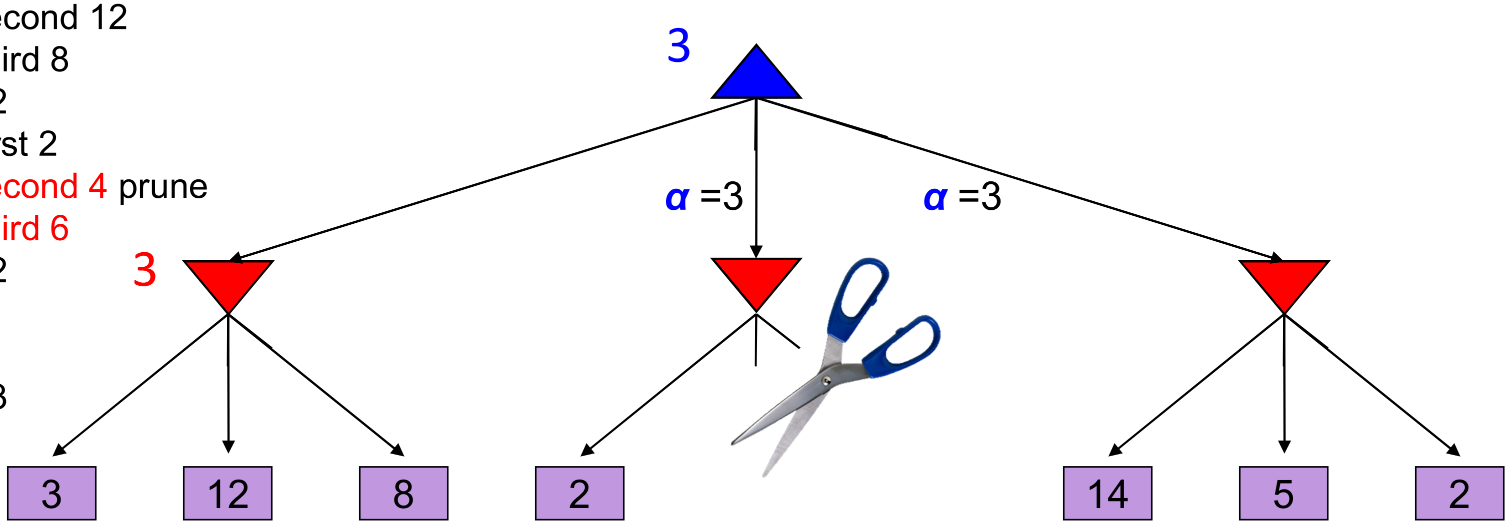


Is it efficient (computational Cost) ? Why?

Alpha-Beta Example

1. Min Search
   1. Select 3
      1. First 3
      2. Second 12
      3. Third 8
   2. Select 2
      1. First 2
      2. Second 4 prune
      3. Third 6
   3. Select 2
      …
2. Max Search
   1. Select 3

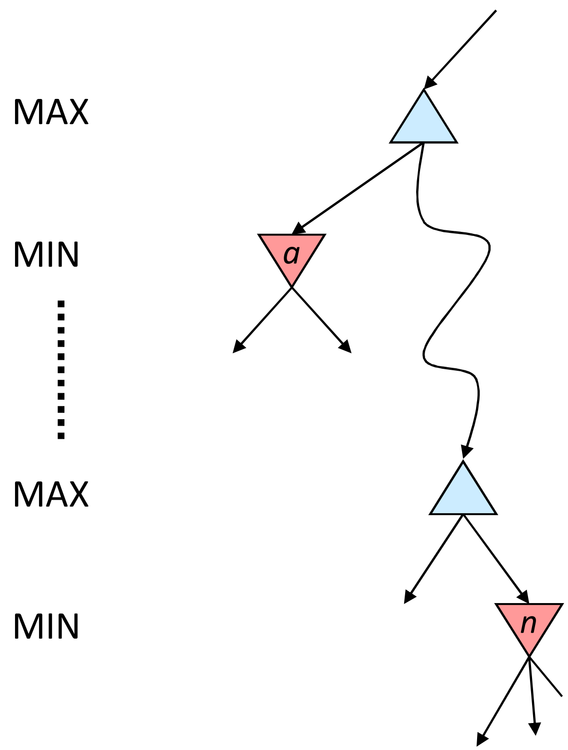
α = best option so far from any MAX node on this path



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

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



Alpha-Beta Pruning

- There are two rules for terminating search:
  - Search can be stopped below any MIN node having a beta value less than or equal to the alpha value of any of its MAX ancestors.
  - Search can be stopped below any MAX node having an alpha value greater than or equal to the beta value of any of its MIN ancestors.

Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

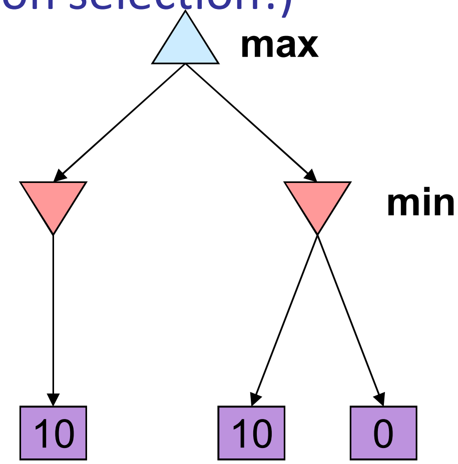
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
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(ambiguity in action selection!)
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
  - How solve?



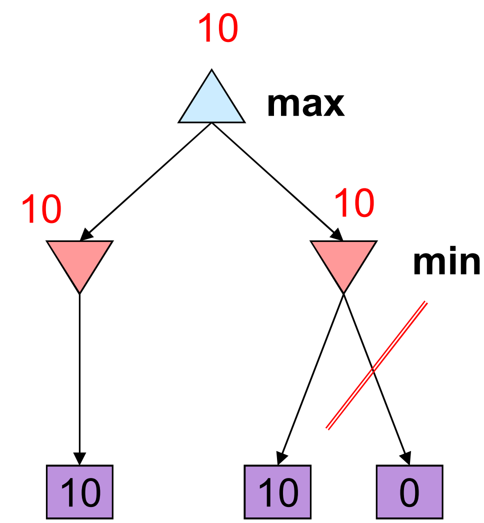
- Good child ordering improves effectiveness of pruning

- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…

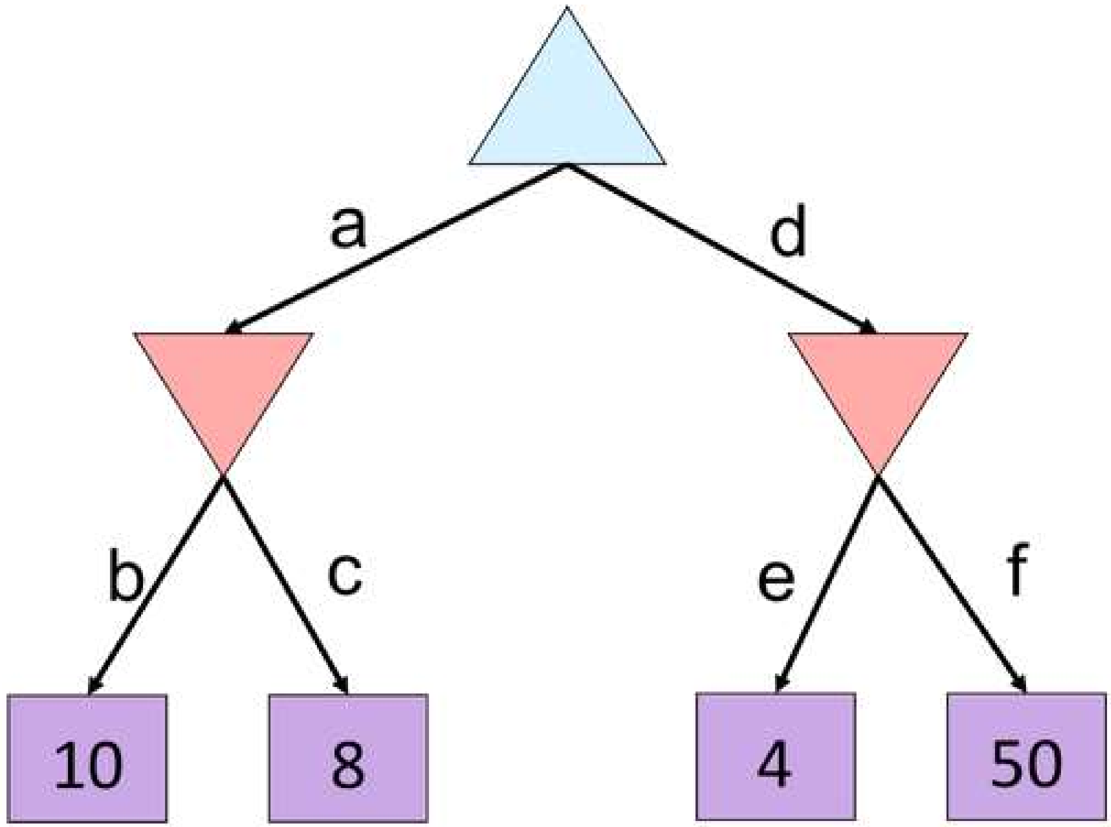
- This is a simple example of meta reasoning (computing about what to compute)

Alpha-Beta Pruning Properties

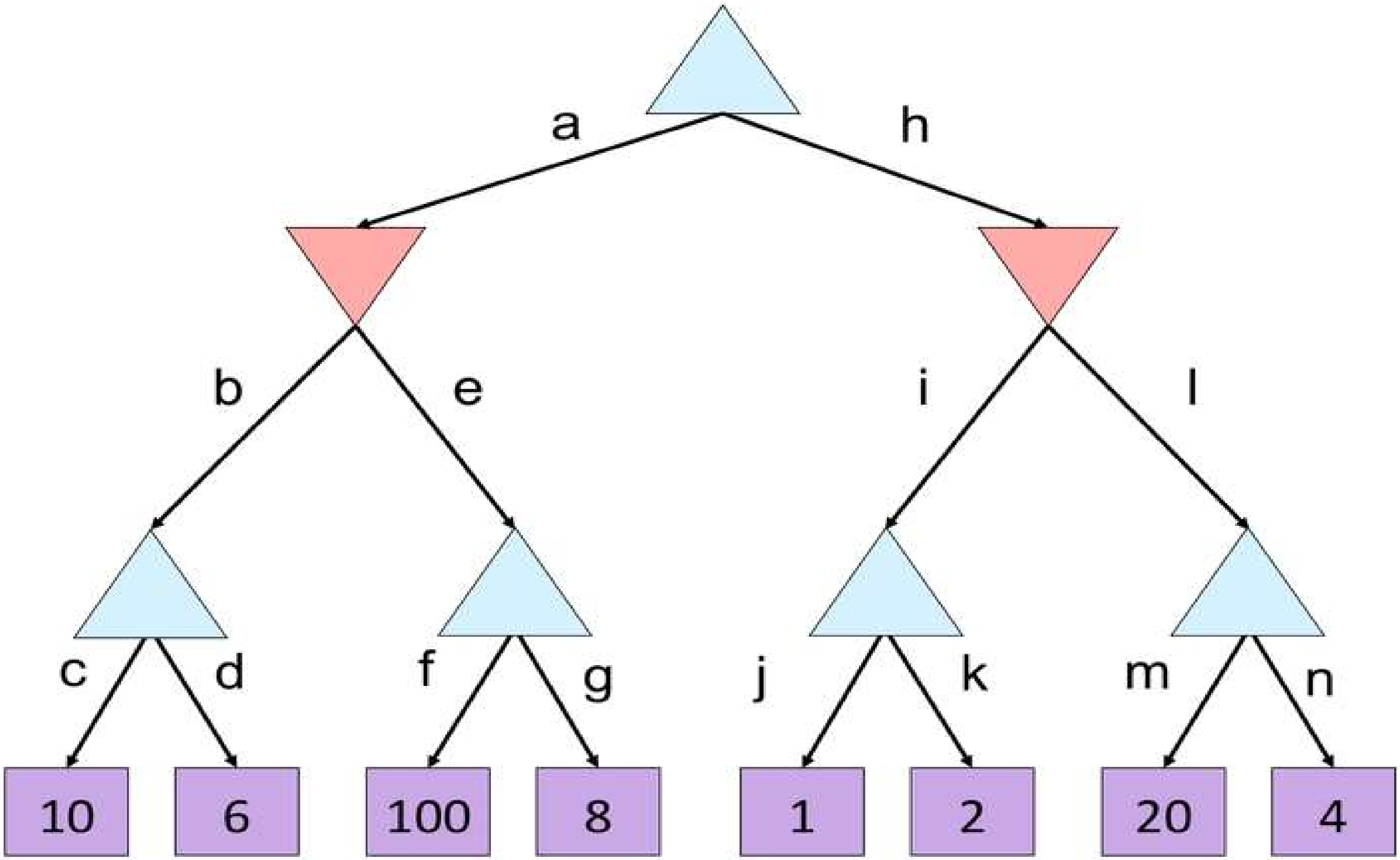
- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With "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)



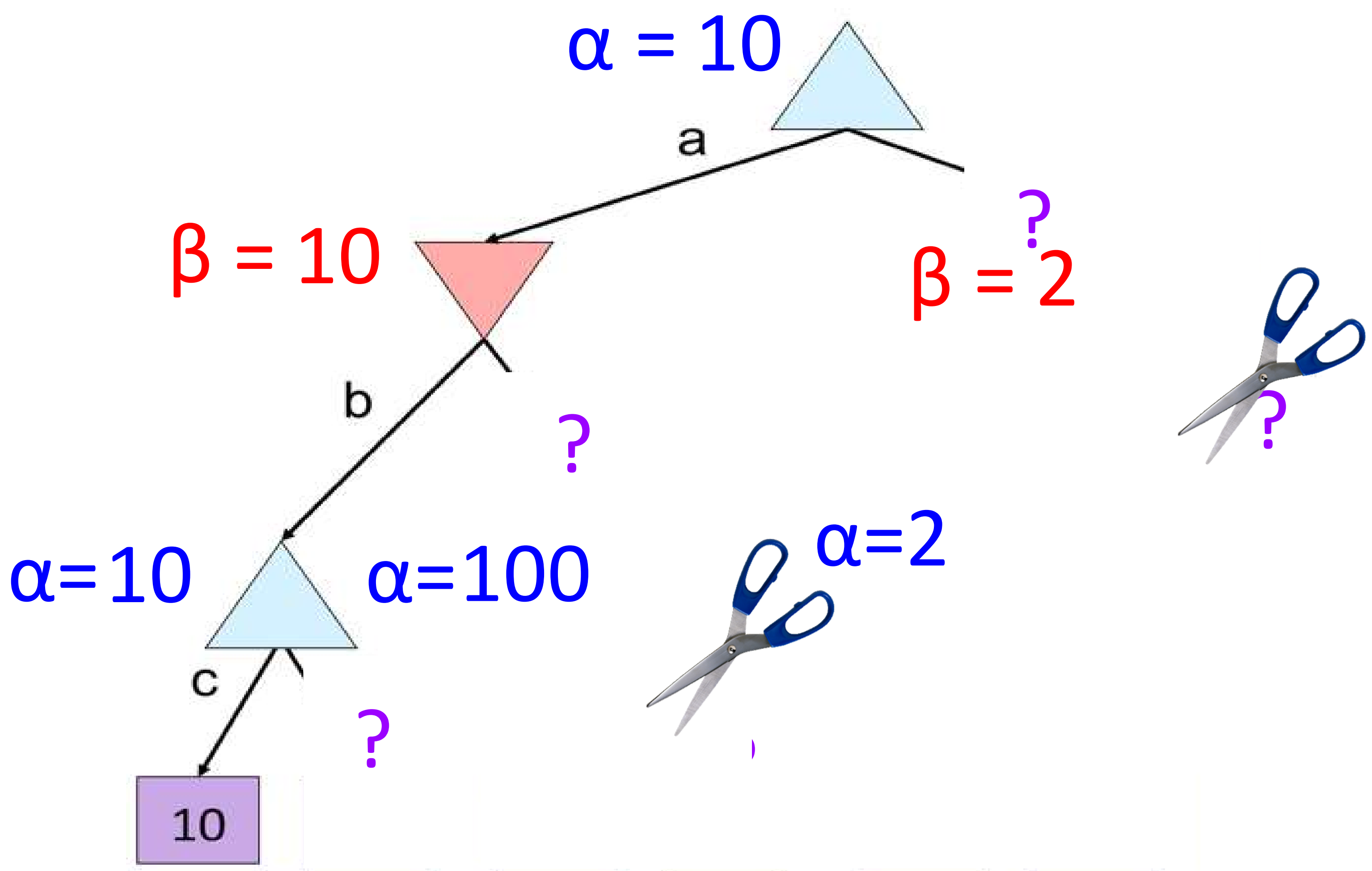
Alpha-Beta Quiz



Alpha-Beta Quiz 2



Alpha-Beta Quiz 2



Alpha-Beta algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
  return the a in ACTIONS(*state*) maximizing MIN-VALUE(RESULT($a, state$))

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for $a,\ s$ in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX($\alpha,\ v$)
  return v

---

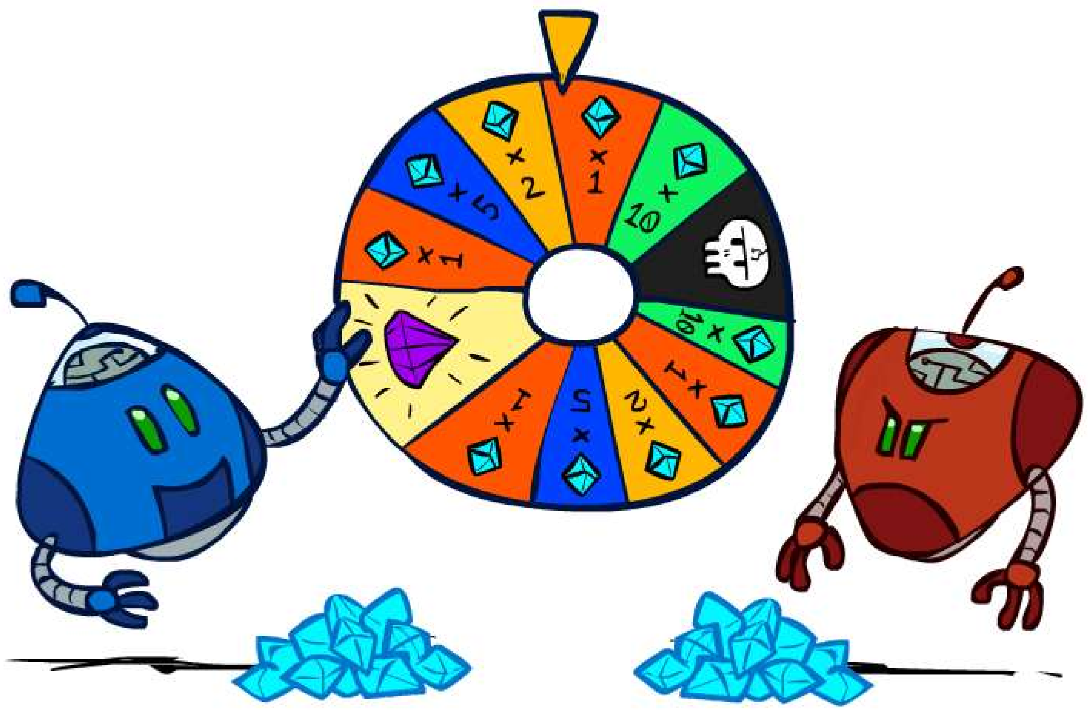
function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

Alpha-Beta algorithm

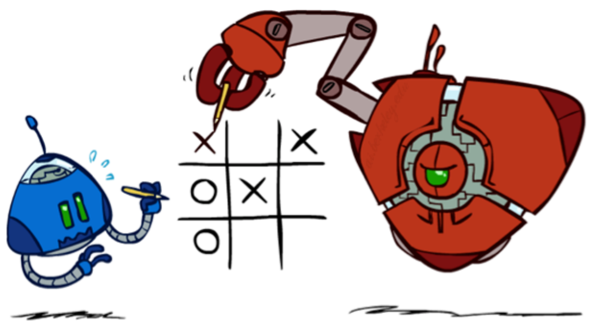
function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

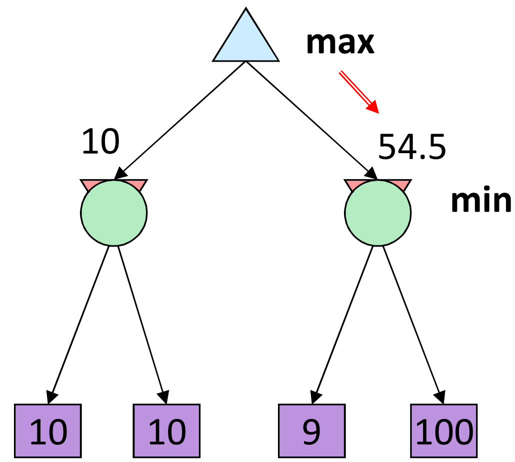
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

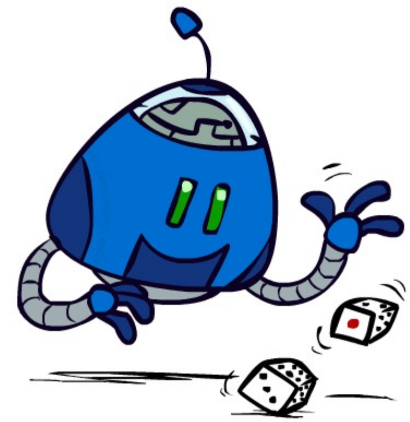
Games with uncertain outcomes



Worst-Case vs. Average Case





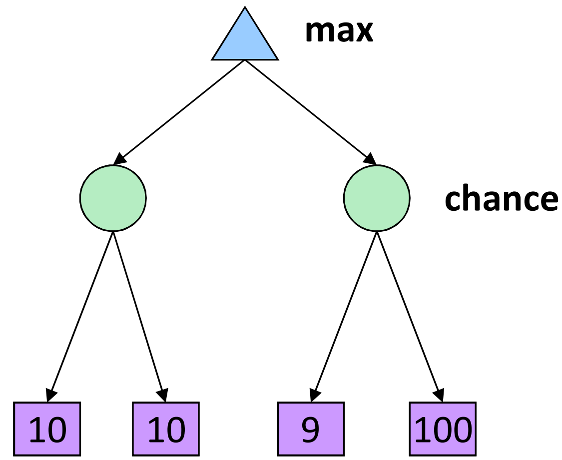


Idea: Uncertain outcomes controlled by chance, not an adversary!

Note: Change MIN node shape

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



[Demo: min vs exp (L7D1,2)]

Expectimax Pseudocode

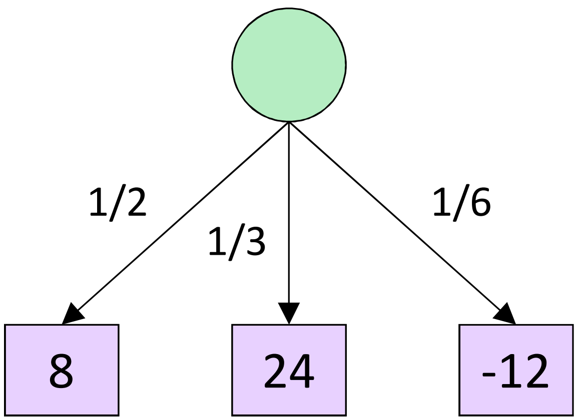
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

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

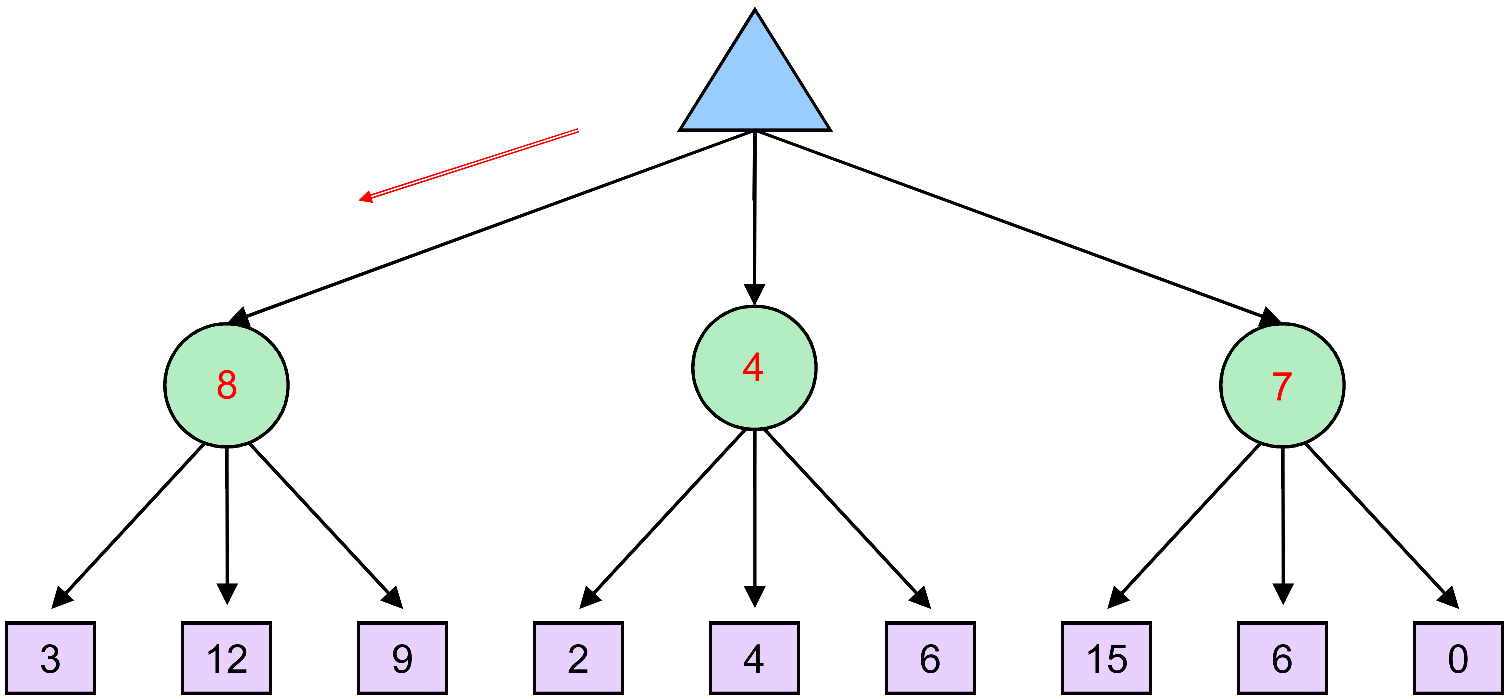
Expectimax Pseudocode

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

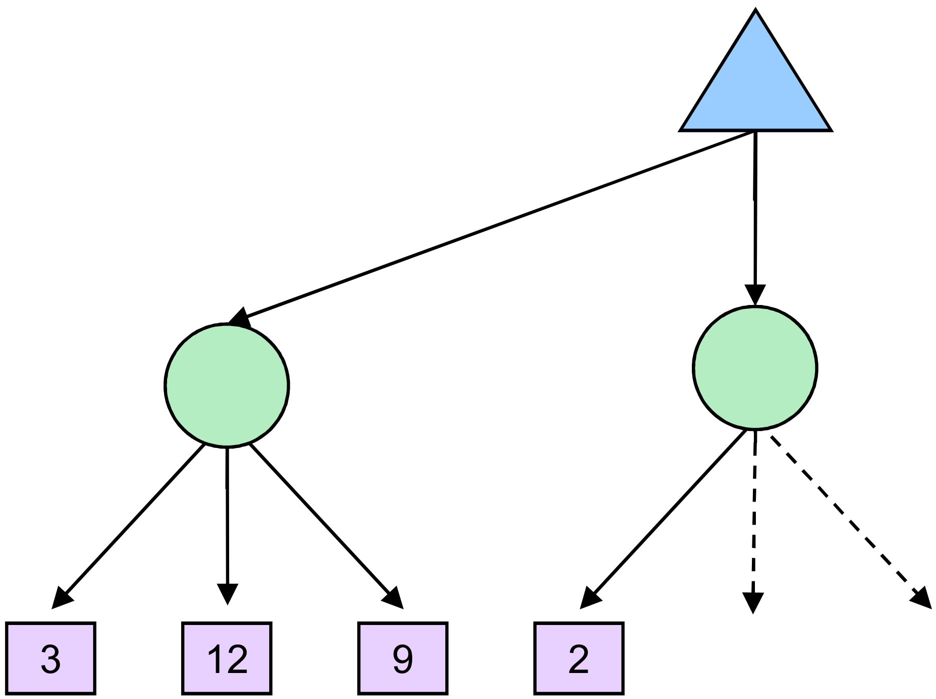


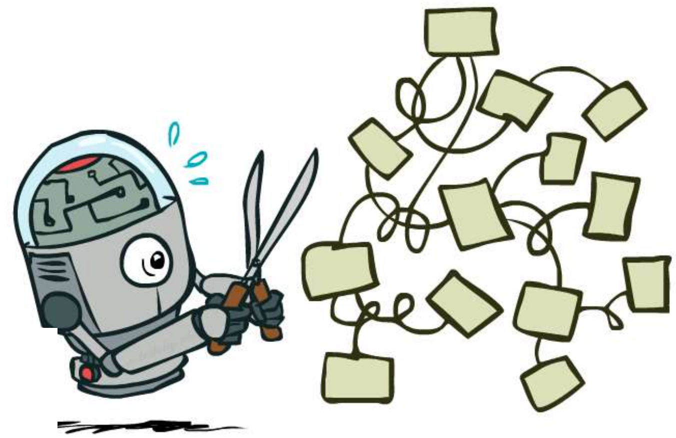
$$v = (1/2)\ (8) + (1/3)\ (24) + (1/6)\ (-12) = 10$$

Expectimax Example



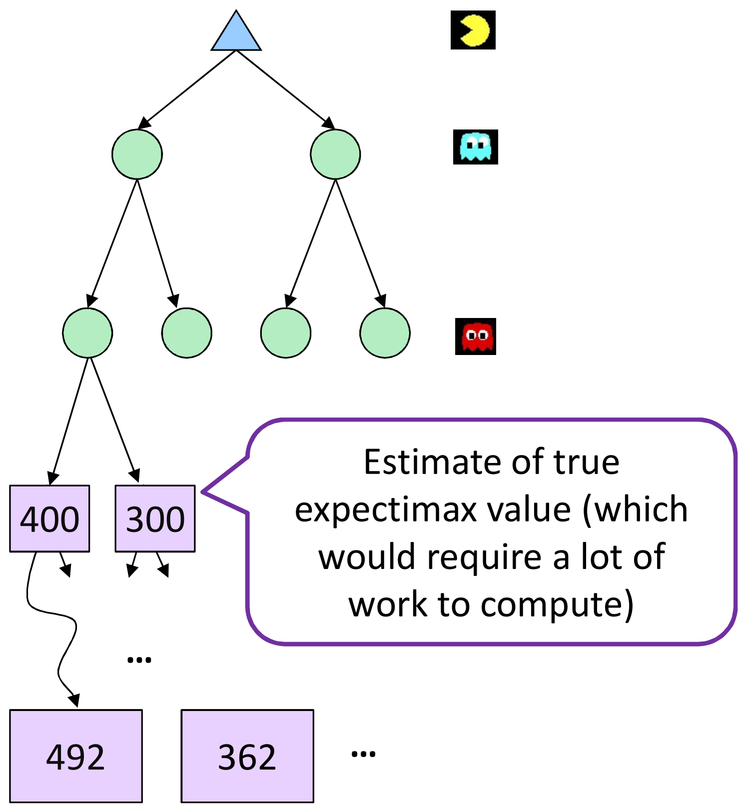
Expectimax Pruning?



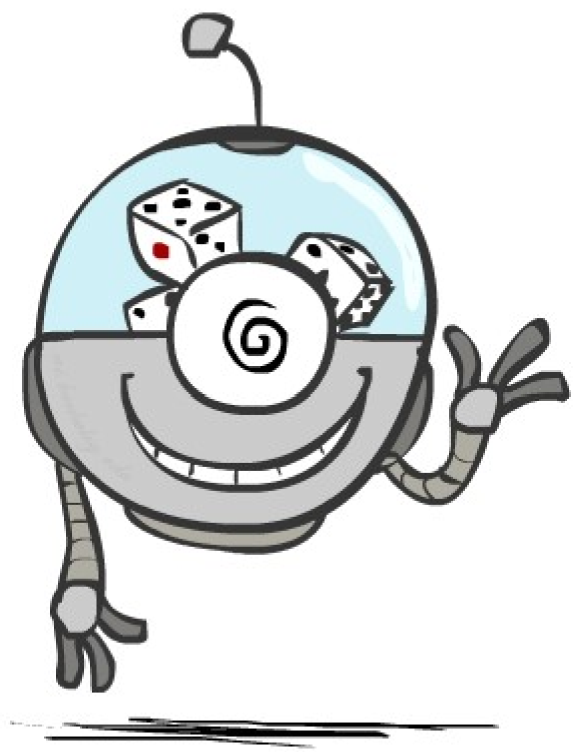


Generally Expectimax Pruning is not true.
But good, if There is upper bound for pruning node
compute upper bound for Expectation and prune based on this value

Depth-Limited Expectimax?



Probabilities



Reminder: Probabilities

- A random variable represents an event whose outcome is unknown
- A probability distribution is an assignment of weights to outcomes

- Example: Traffic on freeway
  - Random variable: T = whether there's traffic
  - Outcomes: T in {none, light, heavy}
  - Distribution: P(T=none) = 0.25, P(T=light) = 0.50, P(T=heavy) = 0.25

- Some laws of probability (more later):
  - Probabilities are always non-negative
  - Probabilities over all possible outcomes sum to one

- As we get more evidence, probabilities may change:
  - P(T=heavy) = 0.25, P(T=heavy | Hour=8am) = 0.60
  - We'll talk about methods for reasoning and updating probabilities later

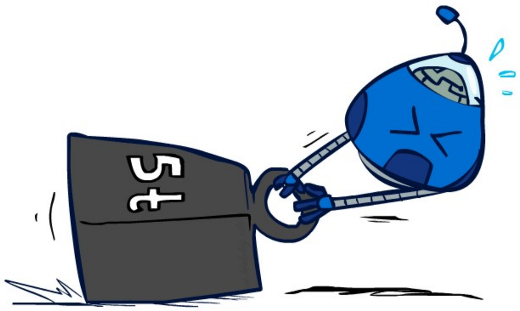
0.25

0.50

0.25

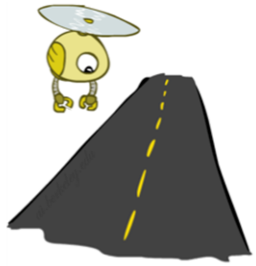
Reminder: Expectations

- The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes



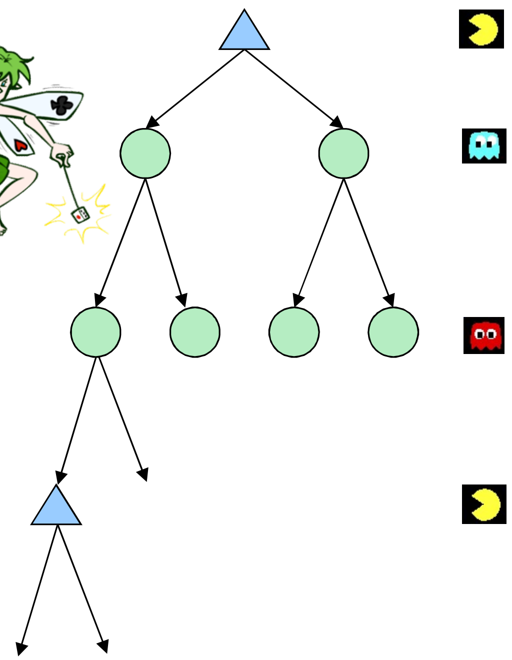
- Example: How long to get to the airport?

Time:	20 min		30 min		60 min		
	x	+	x	+	x	⇒	35 min
Probability:	0.25		0.50		0.25		



What Probabilities to Use?

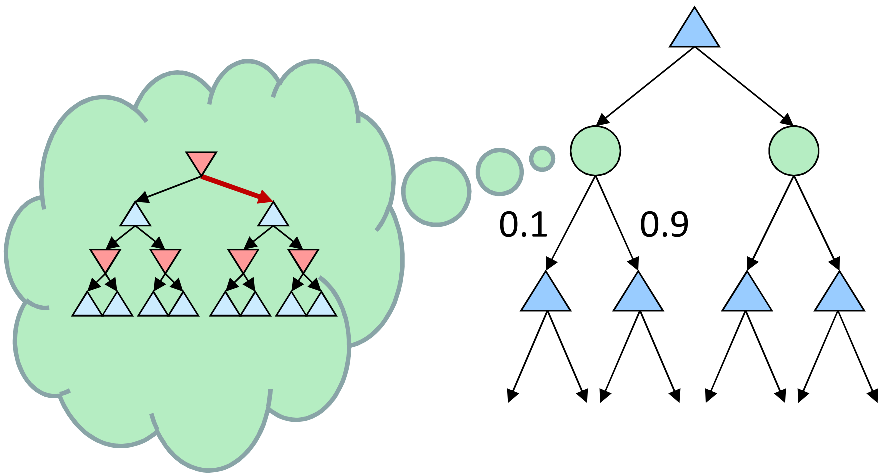
- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely!
- For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes



Having a probabilistic belief about another agent's action does not mean that the agent is flipping any coins!

Informed Probabilities

- Let’s say you know that your opponent is actually running a depth 2 minimax, using the result 80% of the time, and moving randomly otherwise
- Question: What tree search 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

Modeling Assumptions

The Dangers of Optimism and Pessimism

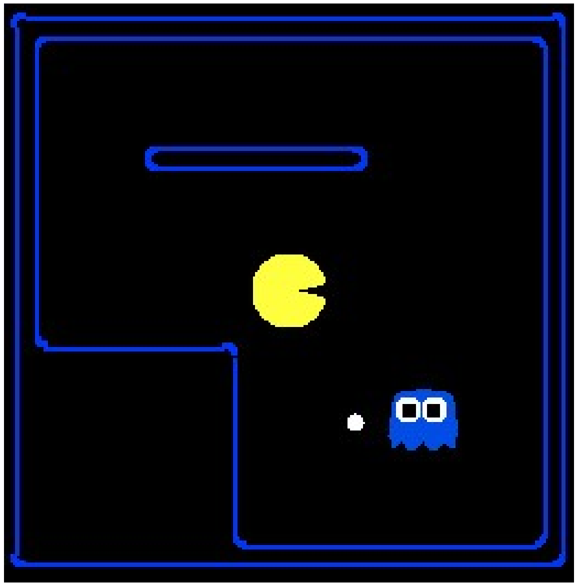
Dangerous Optimism

Assuming (random) chance when the world is adversarial

Dangerous Pessimism

Assuming the worst case when 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

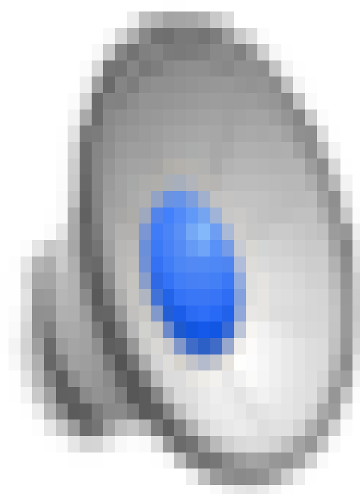
Results from playing 5 games

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

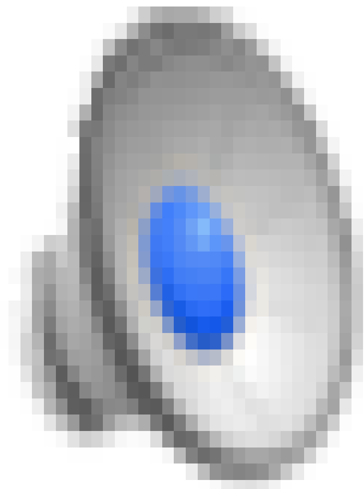
Notice: Use Expectimax if you are sure the ghost is Random

[Demos: world assumptions (L7D3,4,5,6)]

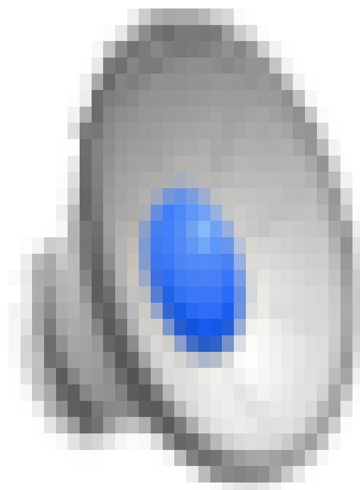
Video of Demo World Assumptions
Random Ghost – Expectimax Pacman



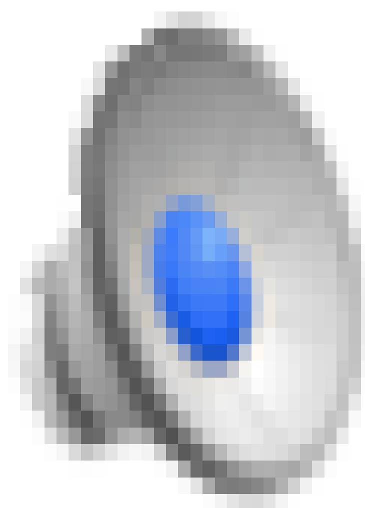
Video of Demo World Assumptions
Adversarial Ghost – Minimax Pacman



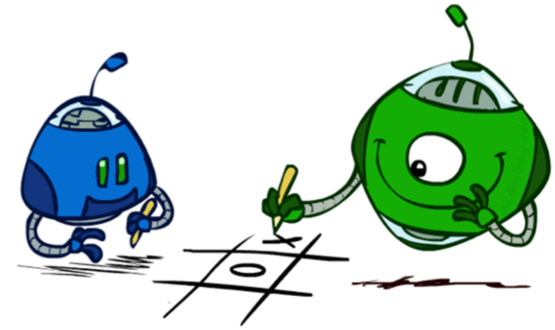
Video of Demo World Assumptions
Adversarial Ghost – Expectimax Pacman



Video of Demo World Assumptions
Random Ghost – Minimax Pacman



Multi-Agent Utilities



- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…