Adversarial Search 3

A-B pruning, Evaluation Functons, ExpectiMax

R&N: Chap. 6

With some slides from Dan Klein and Stuart Russell

Alpha-Beta Pruning

- **Explore** the game tree to depth **h** in depth-first manner
- **Back** up *alpha* and *beta* values whenever possible
- **Prune** branches that can't lead to changing the final decision

Alpha-beta Algorithm

- Depth first search – only considers nodes along a single path at any time

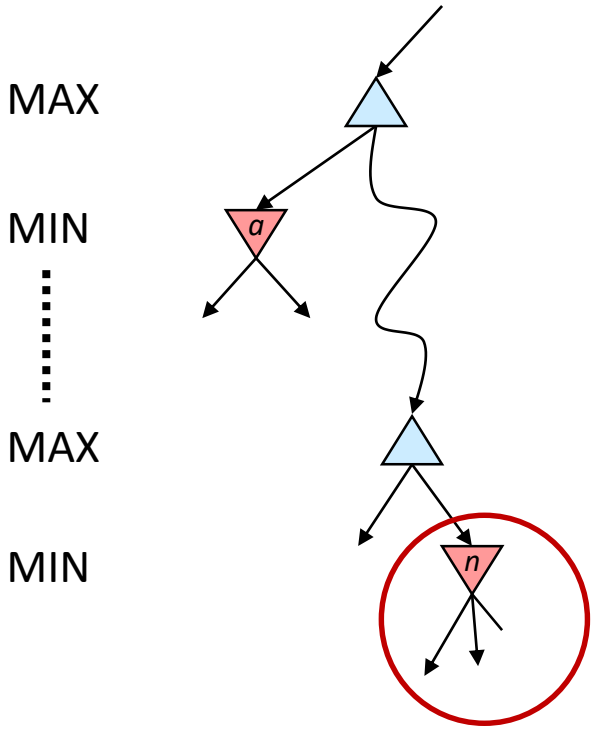
→ α = highest-value choice we have found at any choice point along the path for MAX

→ β = lowest-value choice we have found at any choice point along the path for MIN

- update values of α and β during search, and prune remaining branches when the value is worse than the current α or β value for MAX or MIN

Alpha-Beta Pruning

- General configuration (MIN version)
  - 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 at any choice point along the current path from the root
  - If *n* becomes worse than *α*, MAX will avoid it, so we can stop considering *n*'s other children (it's already bad enough that this move won't be played)



- MAX's reasoning is symmetric

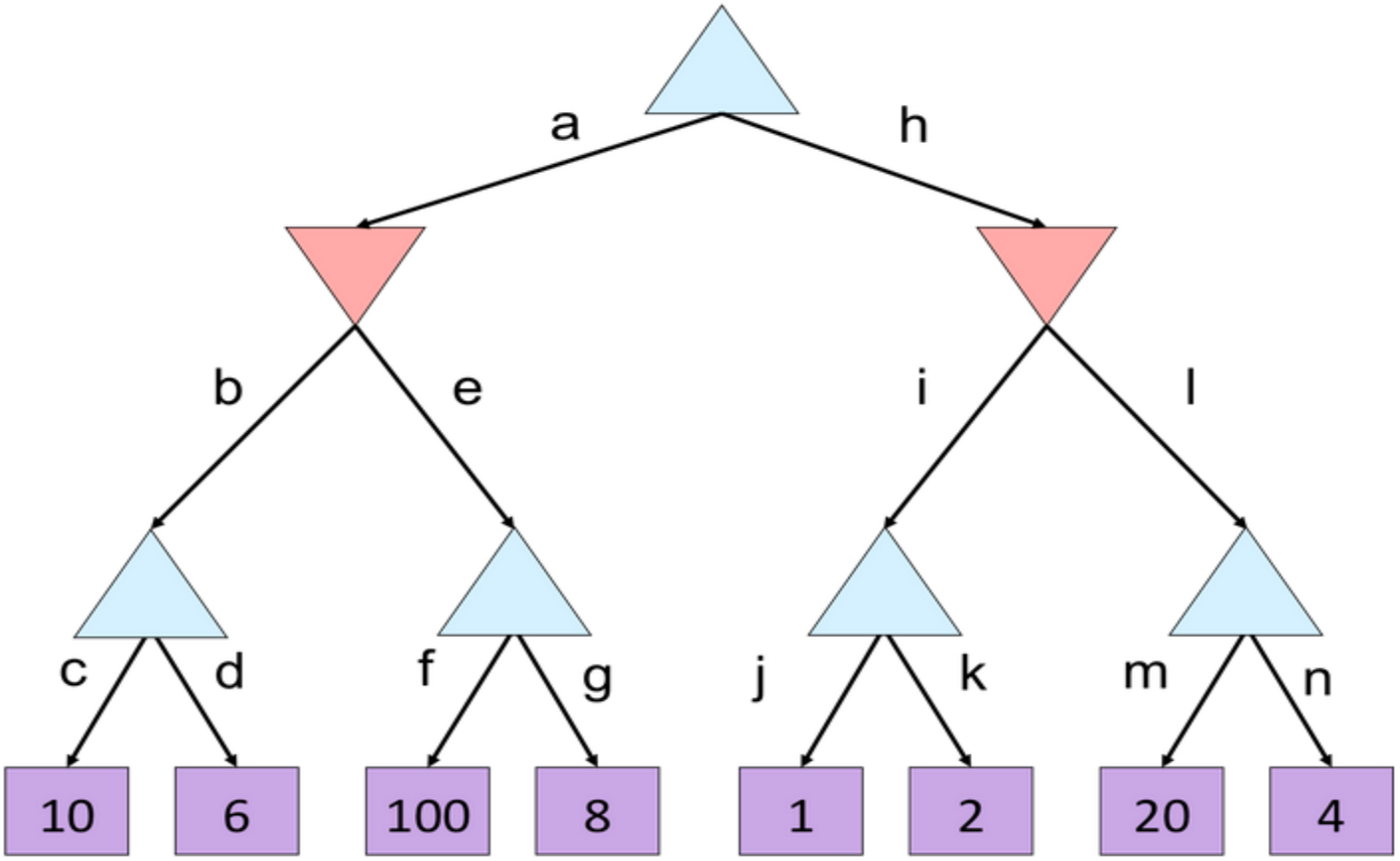
Alpha-Beta Implementation

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 Quiz 2: which nodes pruned?

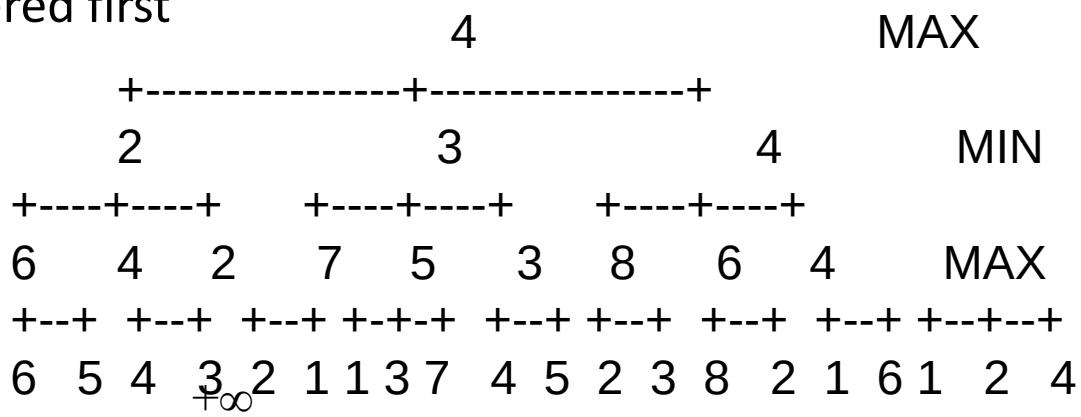


More Alpha-Beta Pruning Examples

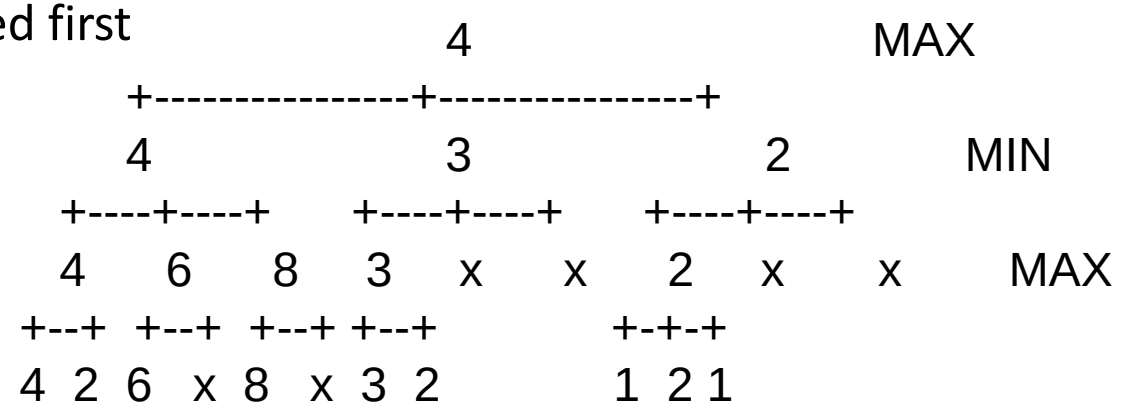
- Generic game tree visualization: http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html
- Connect Four: https://gimu.org/connect-four-js/

Bad and Good Cases for Alpha-Beta Pruning

- Bad: Worst moves encountered first



- Good: Good moves ordered first



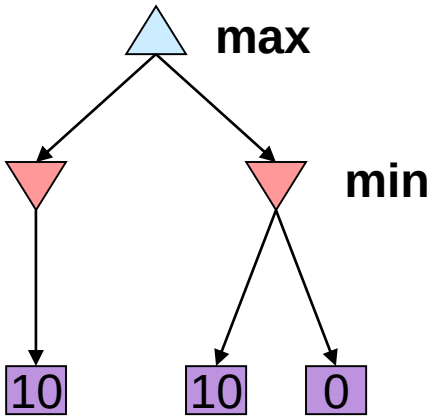
- If we can order moves, we can get more benefit from alpha-beta pruning

Analysis of Alpha-Beta Search

- Worst-Case
  - branches are ordered so that no pruning takes place. In this case alpha-beta gives no improvement over exhaustive search
- Best-Case
  - each player's best move is the left-most alternative (i.e., evaluated first)
  - in practice, performance is closer to best rather than worst-case
- In practice often get $O(b^{(d/2)})$ rather than $O(b^d)$
  - this is the same as having a branching factor of sqrt(b),
    - since $(\text{sqrt}(b))^d = b^{(d/2)}$
    - i.e., we have effectively gone from b to square root of b
  - e.g., in chess go from b ~ 35 to b ~ 6
    - this permits much deeper search in the same amount of time

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…



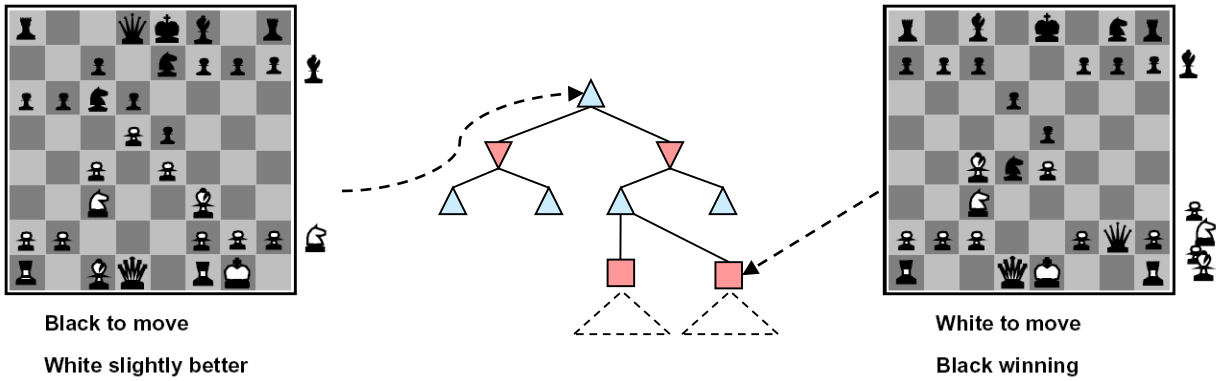
- This is a simple example of metareasoning (computing about what to compute)

Alpha Beta Properties (Summary)

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With **perfect ordering**, time complexity is $O(b^{m/2})$

(Better) 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.

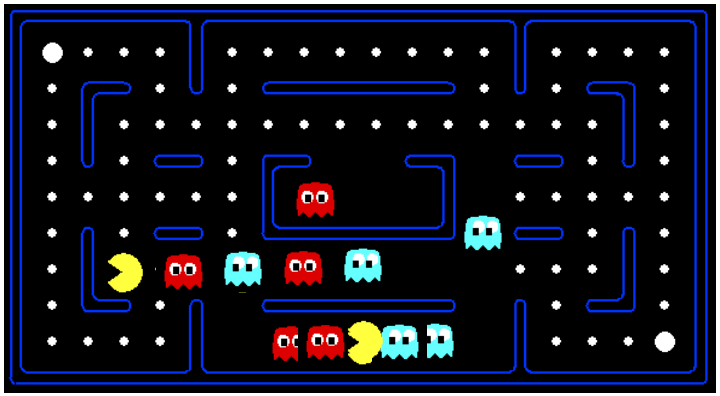
Construction of an Evaluation Function

- Usually a weighted sum of "features":

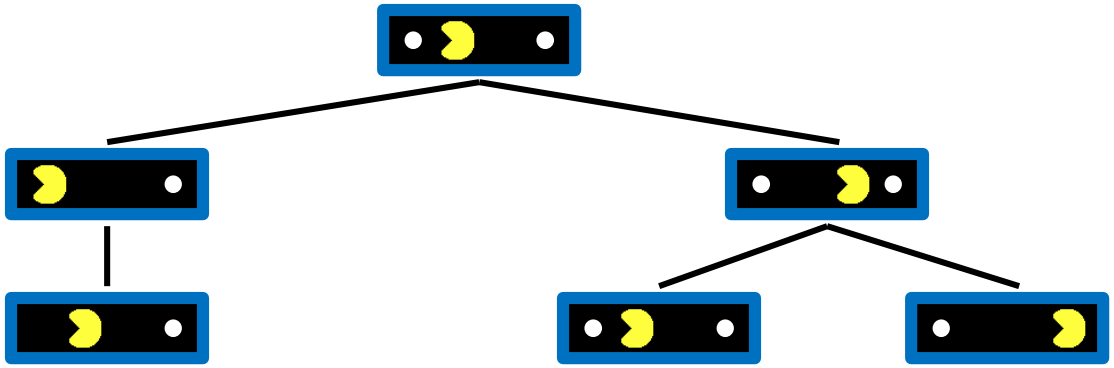
$$e(s)=\sum_{i=1}^{n} w_i f_i(s)$$

- Features may include
  - Number of pieces of each type
  - Number of possible moves
  - Number of squares controlled

Evaluation for Pacman



Why Pacman Starves



- A danger of re-planning 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!

More Examples of Evaluation Functions

- Reversi
  - Number squares held?
  - Better: number of squares held that cannot be flipped
  - Prefer valuable squares
    - NxN array w[i,j] of position values
    - Highest value: corners, edges
    - Lowest value: next to corner or edge
    - s[i,j] = +1 player, 0 empty, -1 opponent

$$score = \sum_{i,j} w[i,j]s[i,j]$$

Eval Function Approximation

Key idea: parameterized evaluation functions

$\text{Eval}(s; \mathbf{w})$ depends on weights $\mathbf{w} \in \mathbb{R}^d$

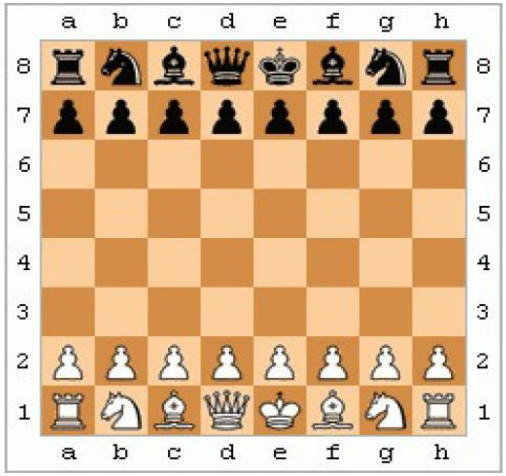
Feature vector: $\phi(s) \in \mathbb{R}^d$

$$\phi_1(s) = K - K'$$
$$\phi_2(s) = Q - Q'$$
$$\ldots$$

Linear evaluation function:

$$\text{Eval}(s; \mathbf{w}) = \mathbf{w} \cdot \phi(s)$$

Chess Example: Revisited



Example: chess

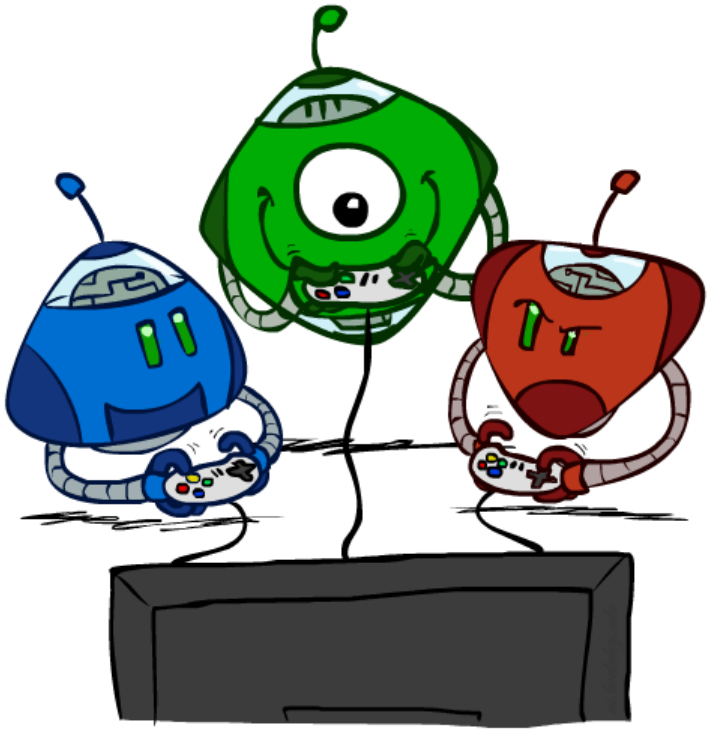
$$\text{Eval}(s) = \text{material} + \text{mobility} + \text{king-safety} + \text{center-control}$$

$$\text{material} = 10^{100}(K - K') + 9(Q - Q') + 5(R - R') + 3(B - B' + N - N') + 1(P - P')$$

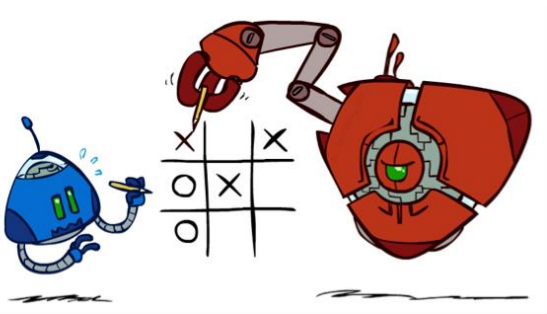
$$\text{mobility} = 0.1(\text{num-legal-moves} - \text{num-legal-moves}')$$

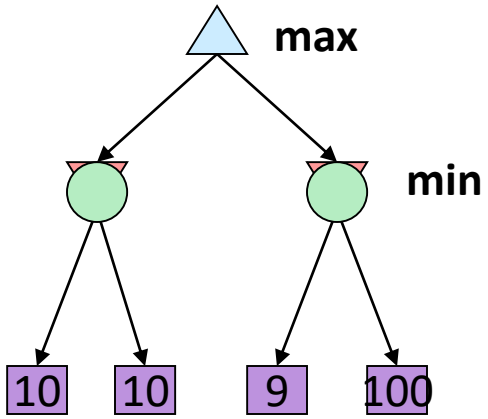
...

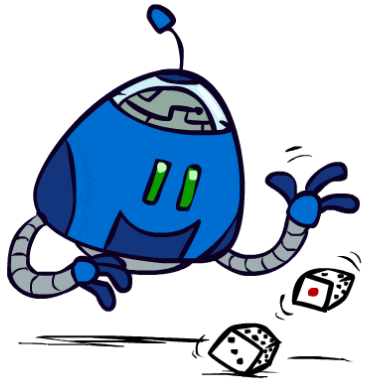
Games with Uncertainty



Worst-Case vs. Average Case



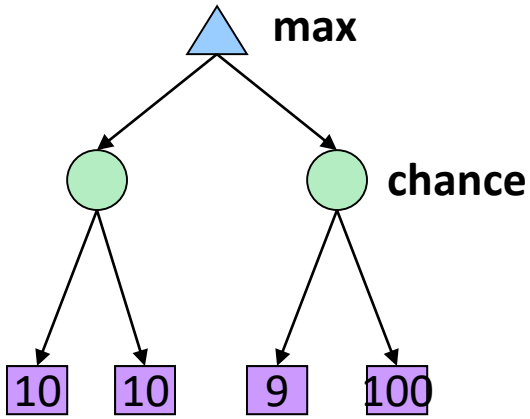




Idea: Uncertain outcomes controlled by chance, not an adversary!

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



Expectimax Pseudocode

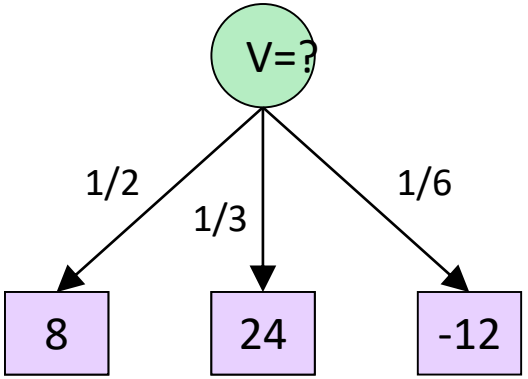
```
def value(state):
        if the state is a terminal state: return the state's utility
        if the next agent is MAX: return max-value(state)
        if the next agent is EXP: return exp-value(state)
```

```
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

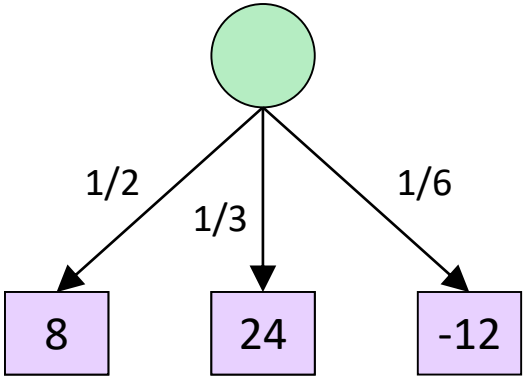
```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```



v = ?

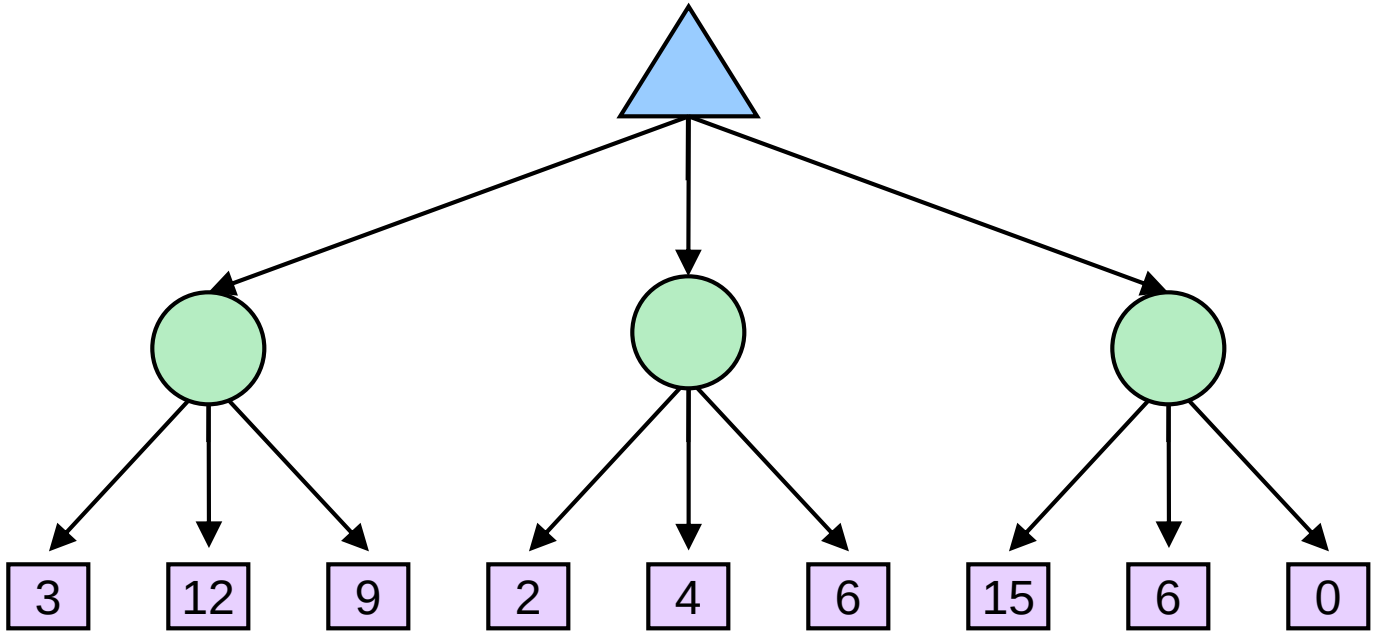
Expectimax Pseudocode

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

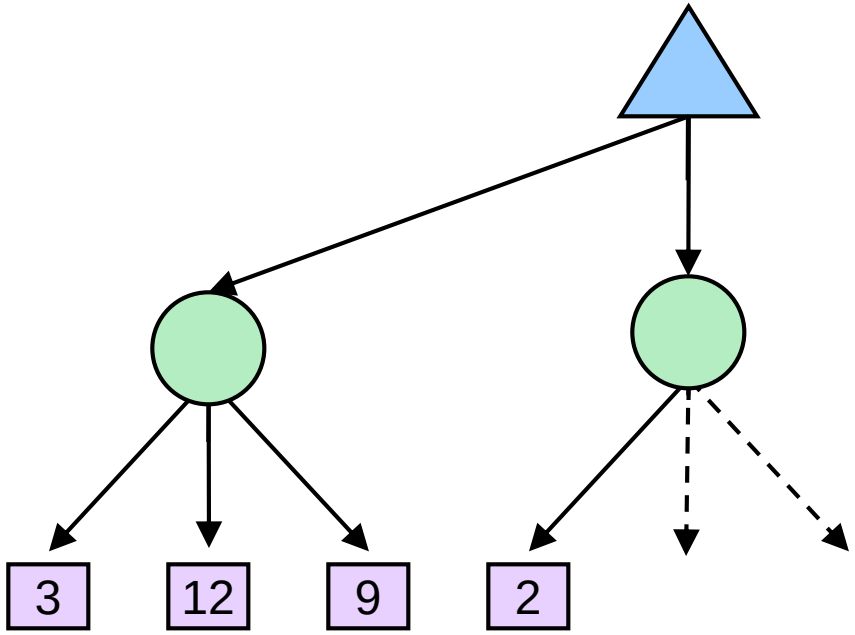


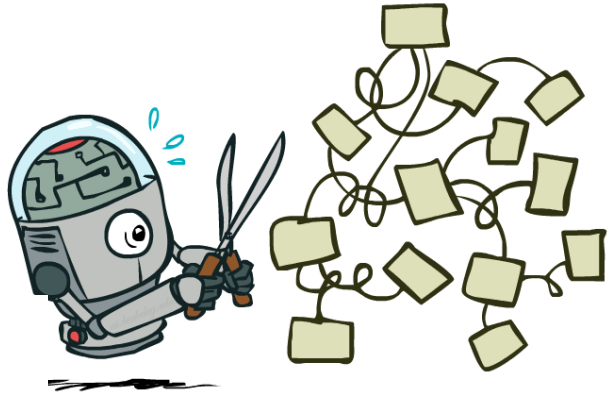
v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

Expectimax Example

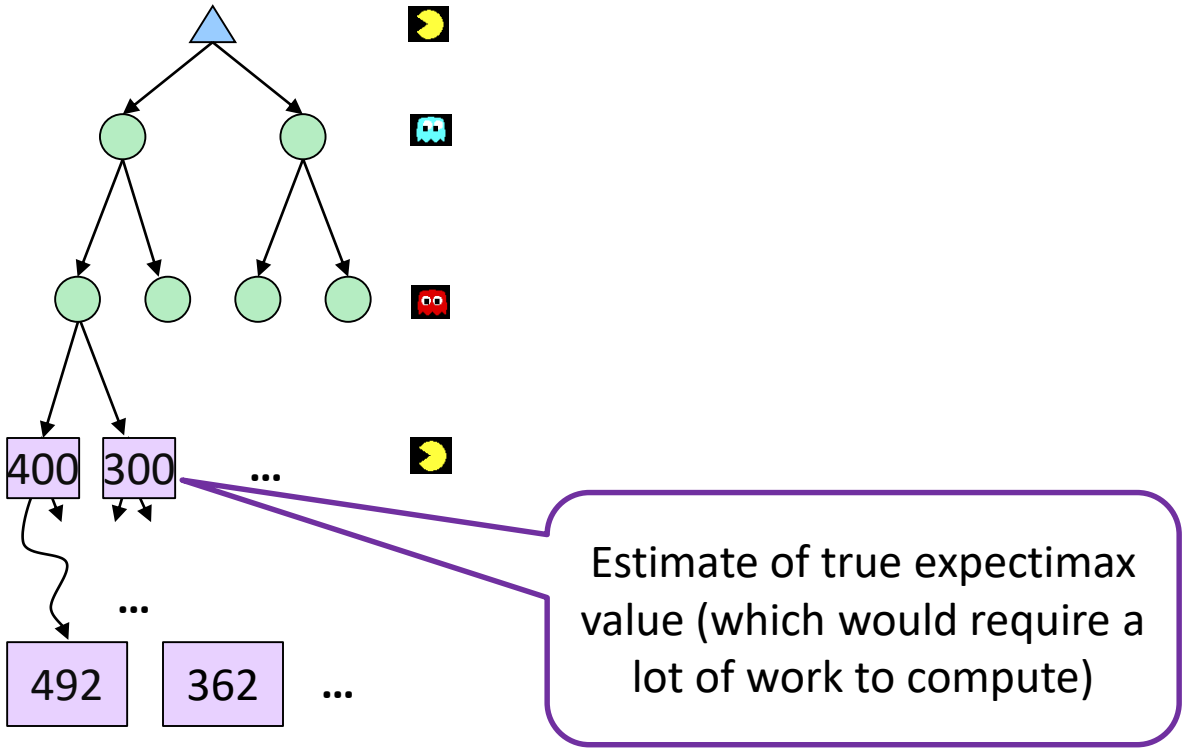


Expectimax Pruning?

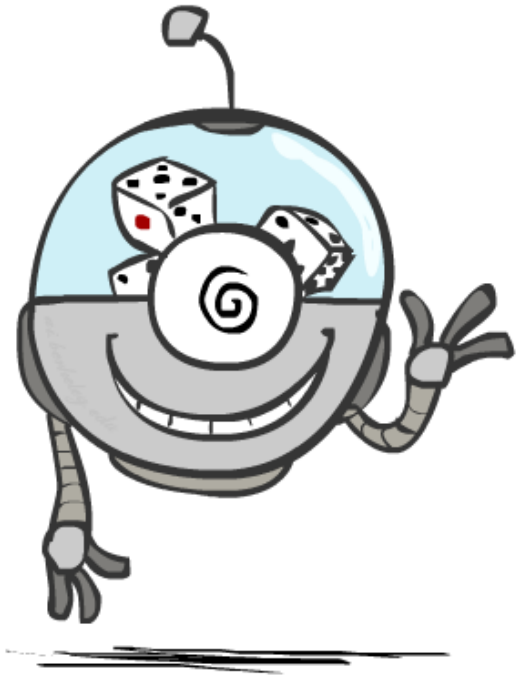




Depth-Limited Expectimax



Probabilities



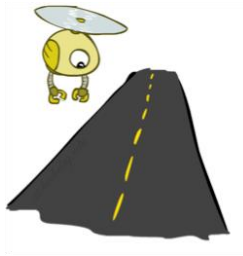
Reminder: Probabilities

- 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

What are Probabilities?

- Objectivist / frequentist answer:
  - Averages over repeated *experiments*
  - E.g. empirically estimating P(rain) from historical observation
  - Assertion about how future experiments will go (in the limit)
  - New evidence changes the *reference class*
  - Makes one think of *inherently random* events, like rolling dice

- Subjectivist / Bayesian answer:
  - Degrees of belief about unobserved variables
  - E.g. an agent's belief that it's raining, given the temperature
  - E.g. pacman's belief that the ghost will turn left, given the state
  - Often *learn* probabilities from past experiences (more later)
  - New evidence *updates beliefs* (more later)

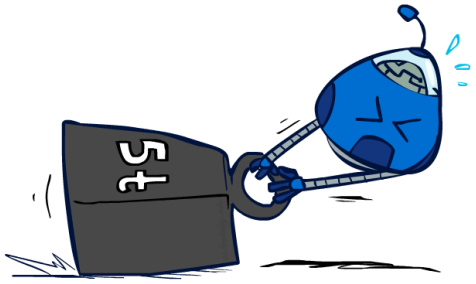
Uncertainty Everywhere

- Not just for games of chance!
  - I'm sick: will I sneeze this minute?
  - Email contains "FREE!": is it spam?
  - Tooth hurts: have cavity?
  - 60 min enough to get to the airport?
  - Robot rotated wheel three times, how far did it advance?
  - Safe to cross street? (Look both ways!)

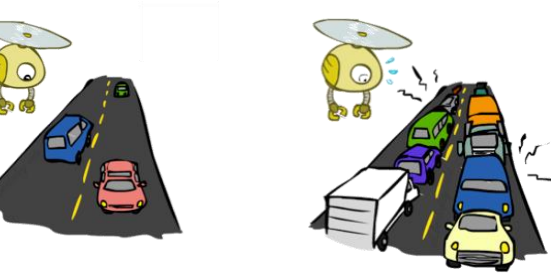
- Sources of uncertainty in random variables:
  - Inherently random process (dice, etc)
  - Insufficient or weak evidence
  - Ignorance of underlying processes
  - Unmodeled variables
  - The world's just noisy – it doesn't behave according to plan!

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
Probability:	0.25		0.50		0.25

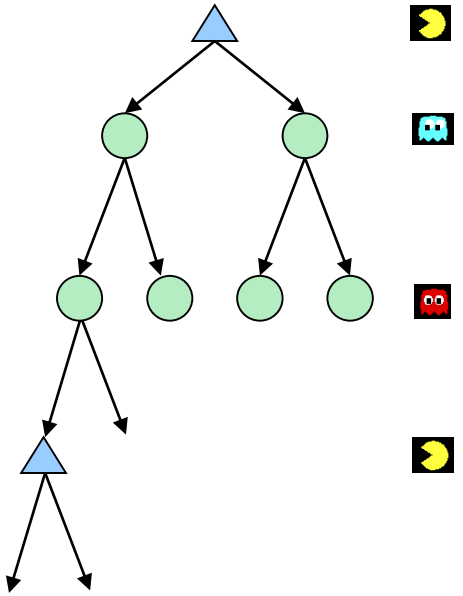


35 min

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
  - Model might say that adversarial actions are more likely!

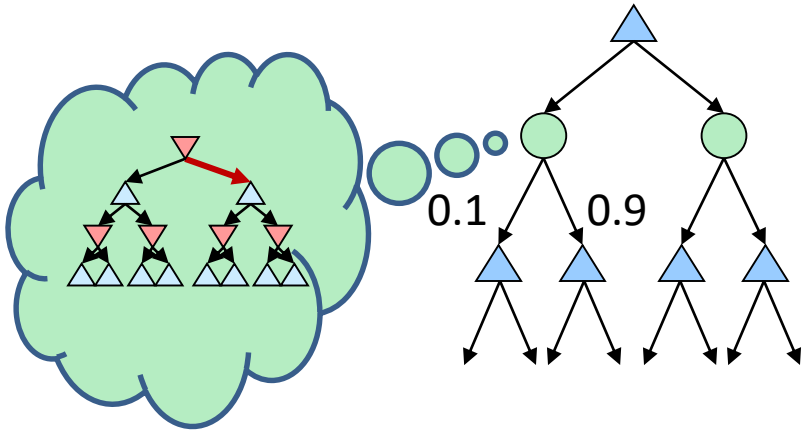
- For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes



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

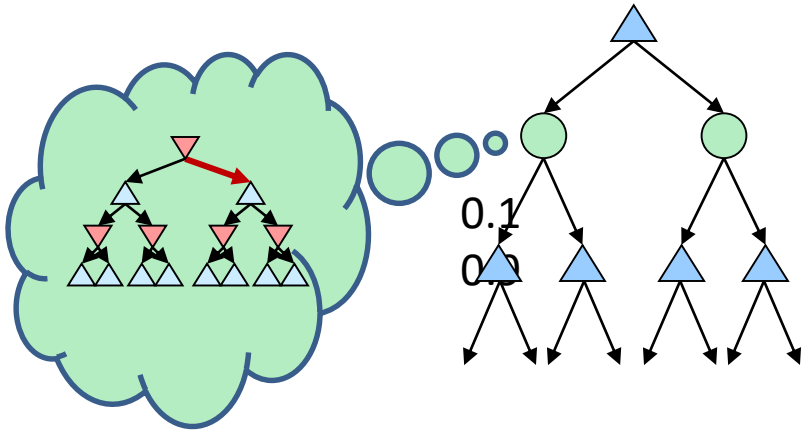
Quiz: Informed Probabilities

- 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?



Quiz: 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?



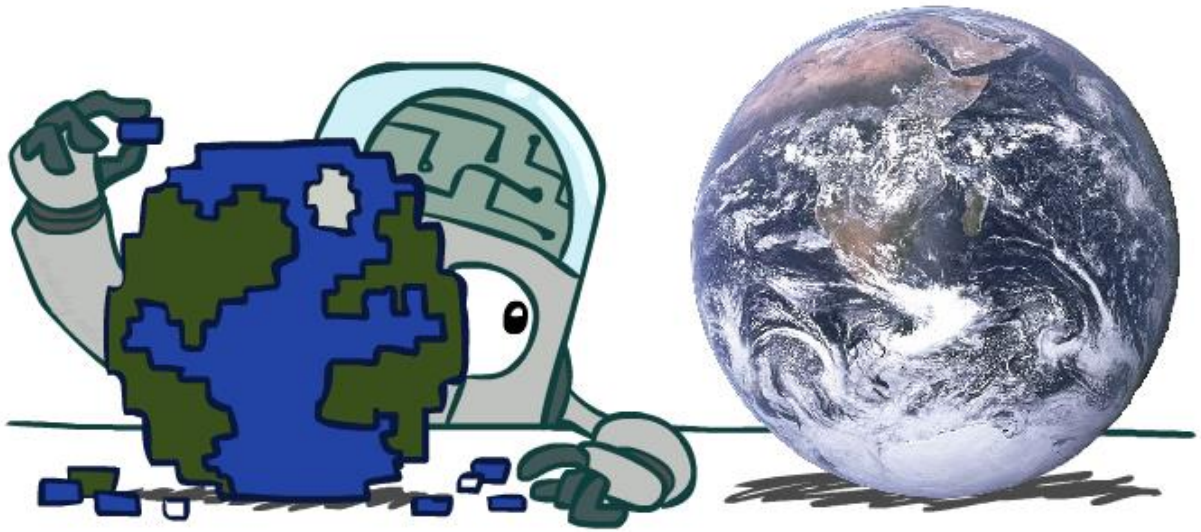
- Answer: Expectimax!
  - To figure out EACH chance node's probabilities, you have to run a simulation of your opponent
  - This kind of thing gets very slow very quickly
  - Even worse if you have to simulate your opponent simulating you…
  - … except for minimax, which has the nice property that it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

Project 2: Fightin' Pacman

- Due: **Wednesday Feb 28, at 8pm**

http://www.mathcs.emory.edu/~eugene/cs425/p2/

Modeling Assumptions

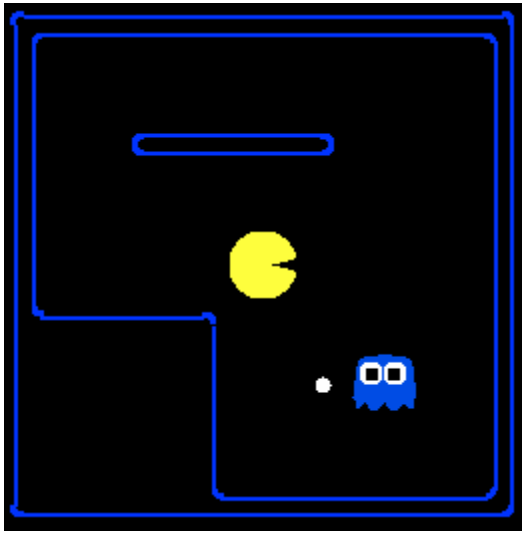


The Dangers of Optimism and Pessimism

Dangerous Optimism
Assuming 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

Next: Other Games and Rationality