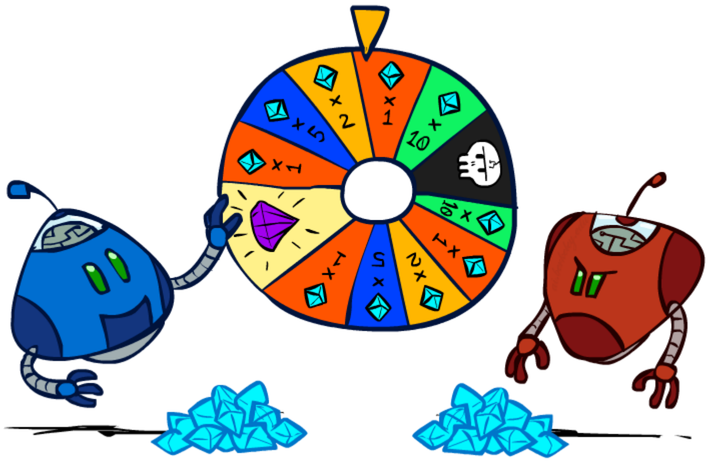
# CSE 473: Introduction to Artificial Intelligence

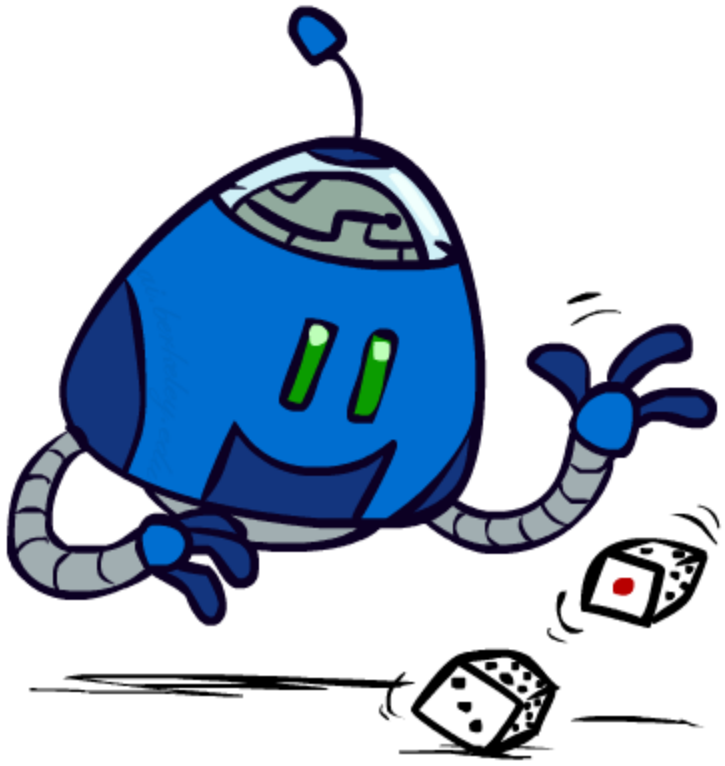
Hanna Hajishirzi

## Expectimax – Complex Games

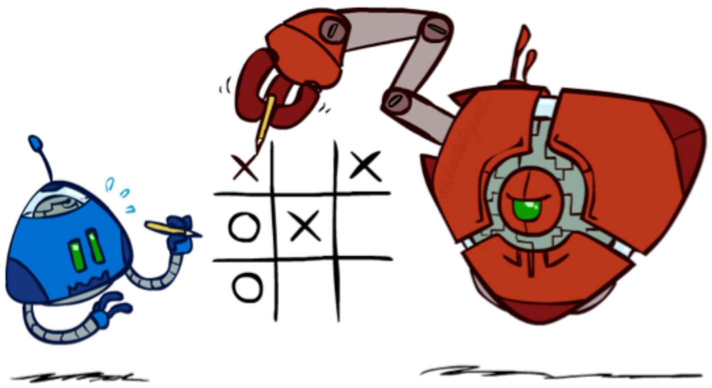


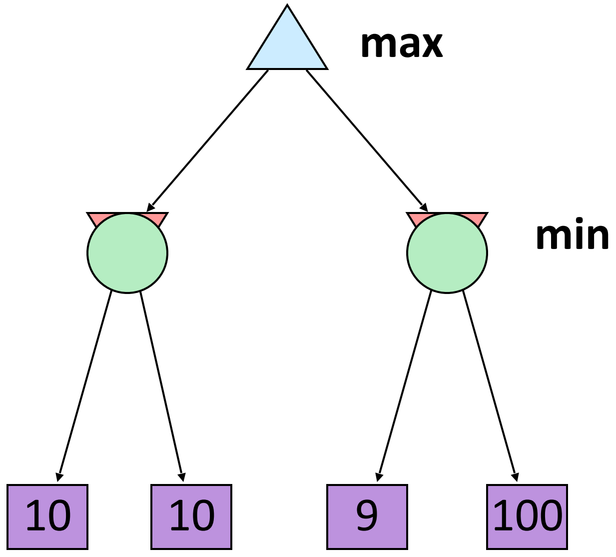
slides adapted from
Dan Klein, Pieter Abbeel ai.berkeley.edu
And Dan Weld, Luke Zettlemoyer

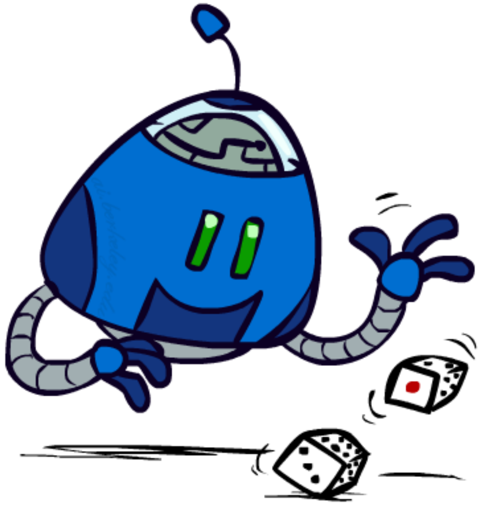
# Uncertain Outcomes



# Worst-Case vs. Average Case



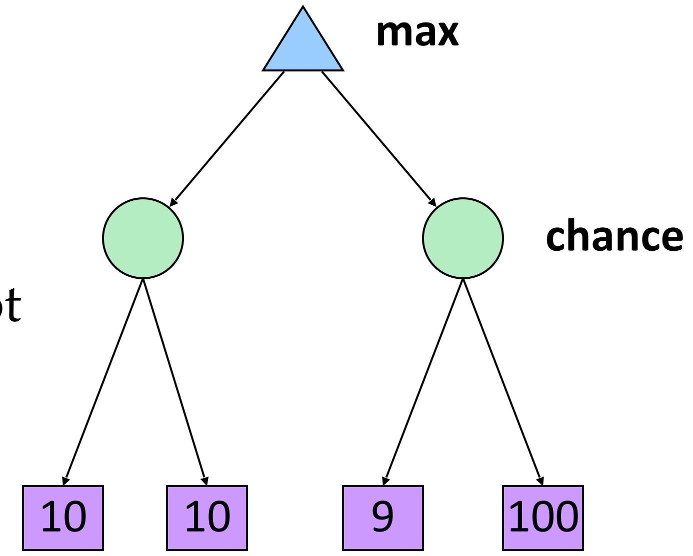


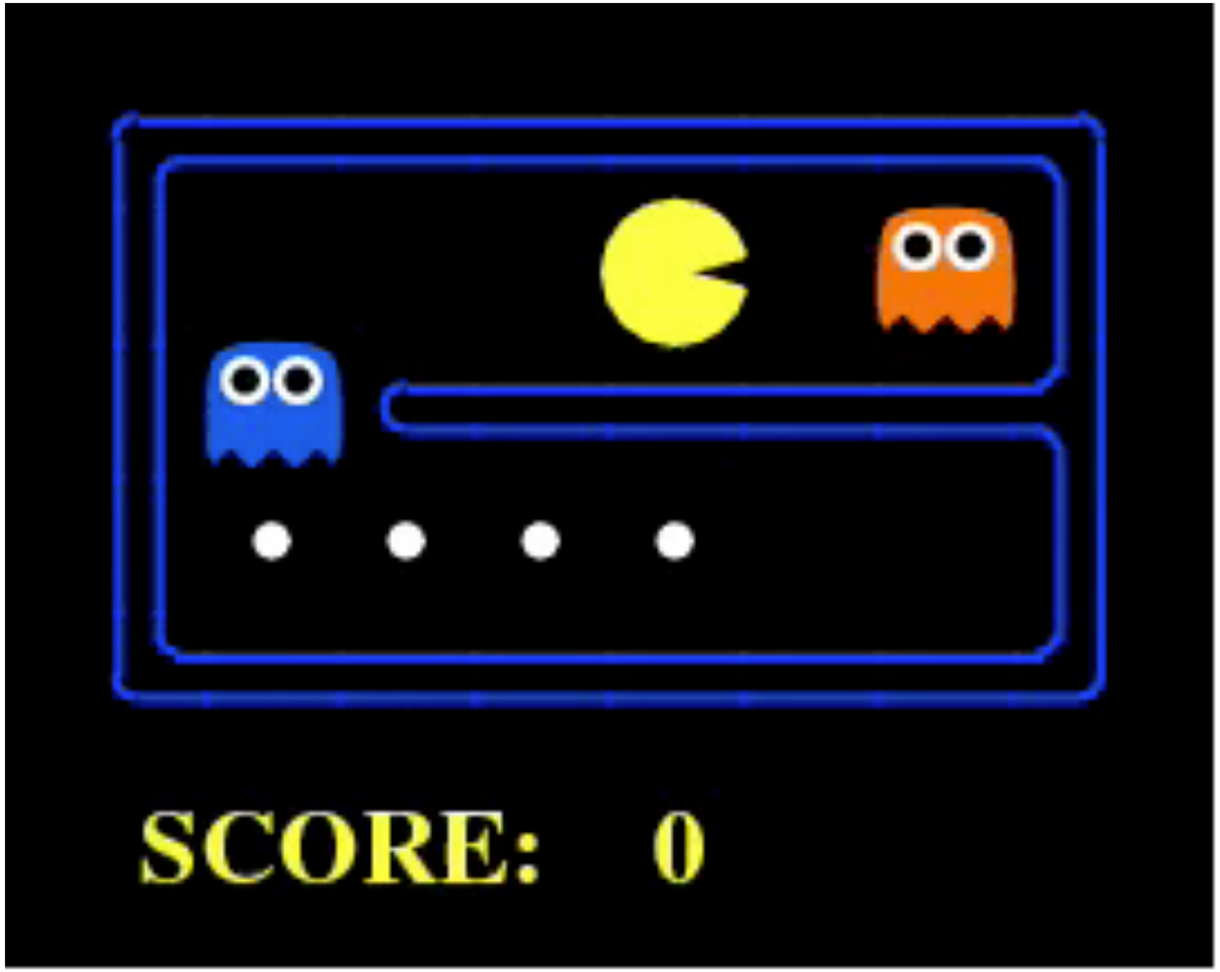
Idea: Uncertain outcomes controlled by chance, not an adversary!

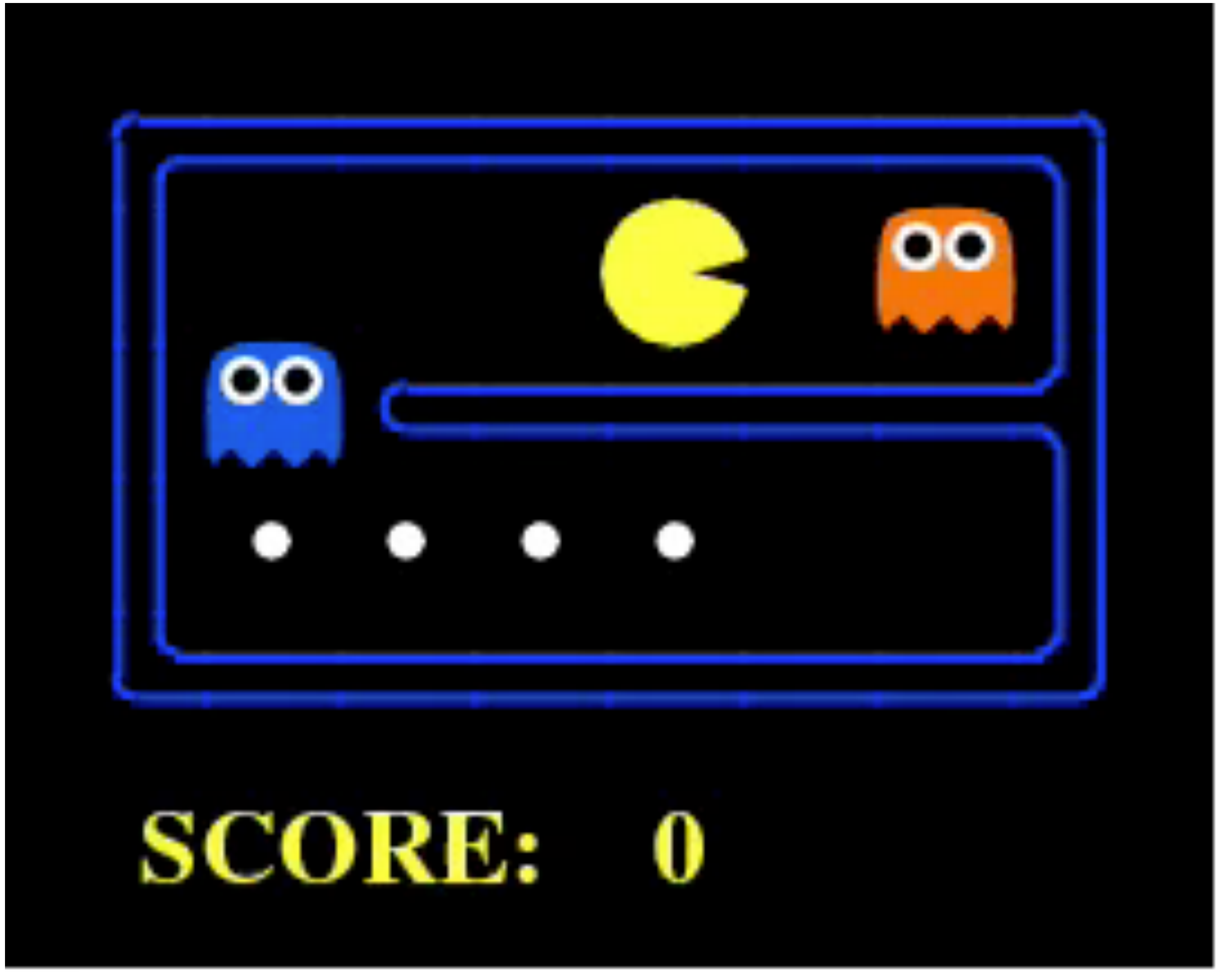
# Expectimax Search

- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Unpredictable humans: humans are not perfect
  - 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



# Video of Demo Min vs. Exp (Min)



# Expectimax Pseudocode

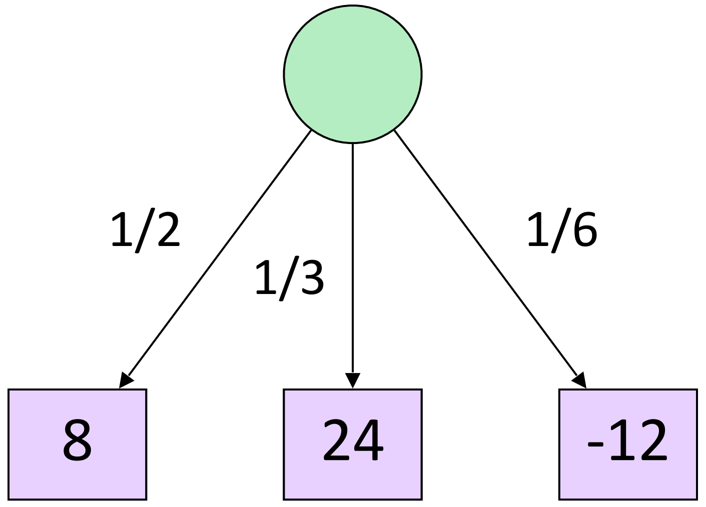
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

```
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
```

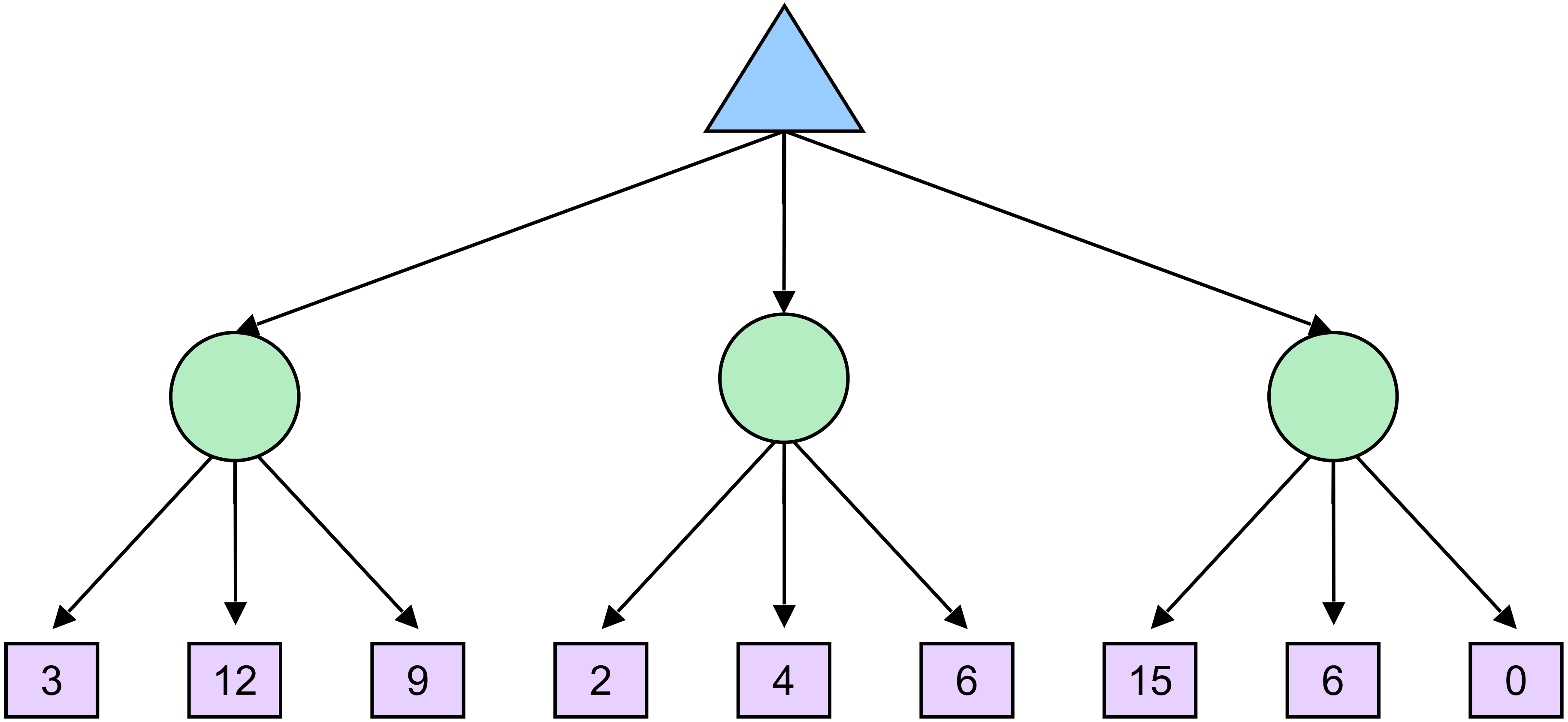
# Expectimax Pseudocode

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

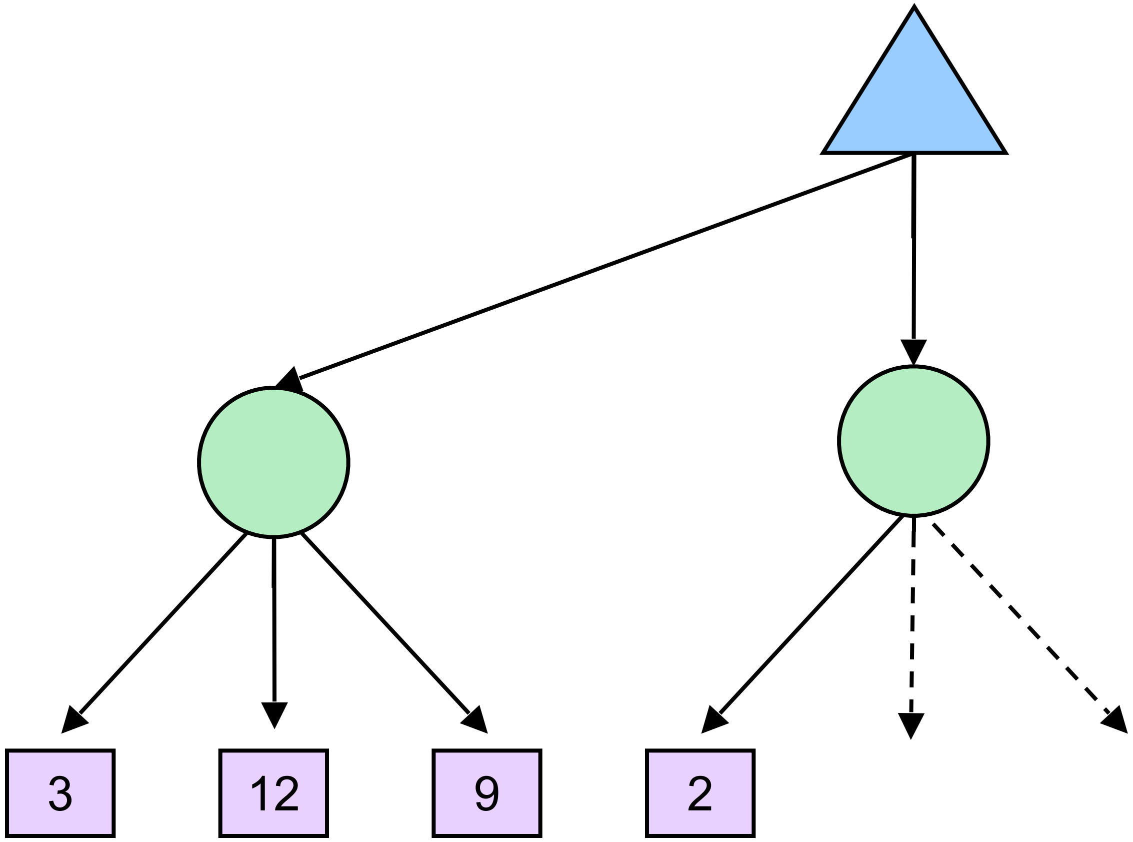


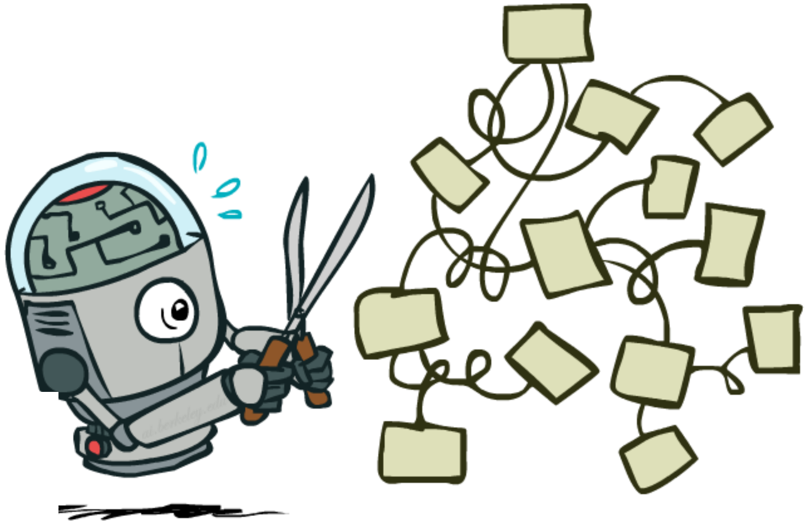

v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

# Expectimax Example

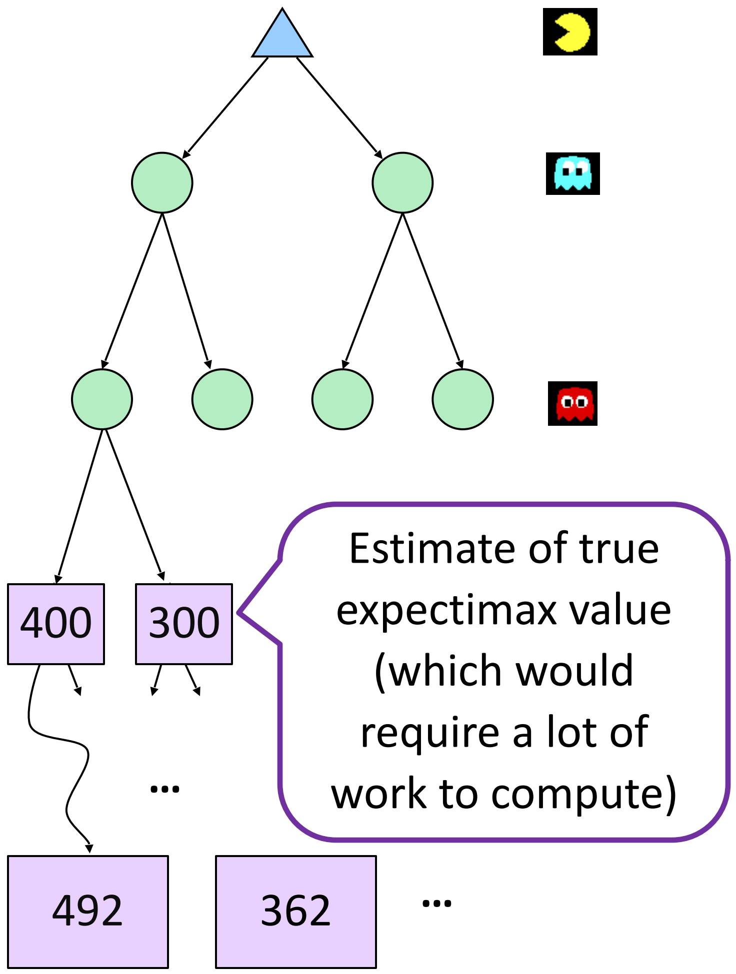


# Expectimax Pruning?

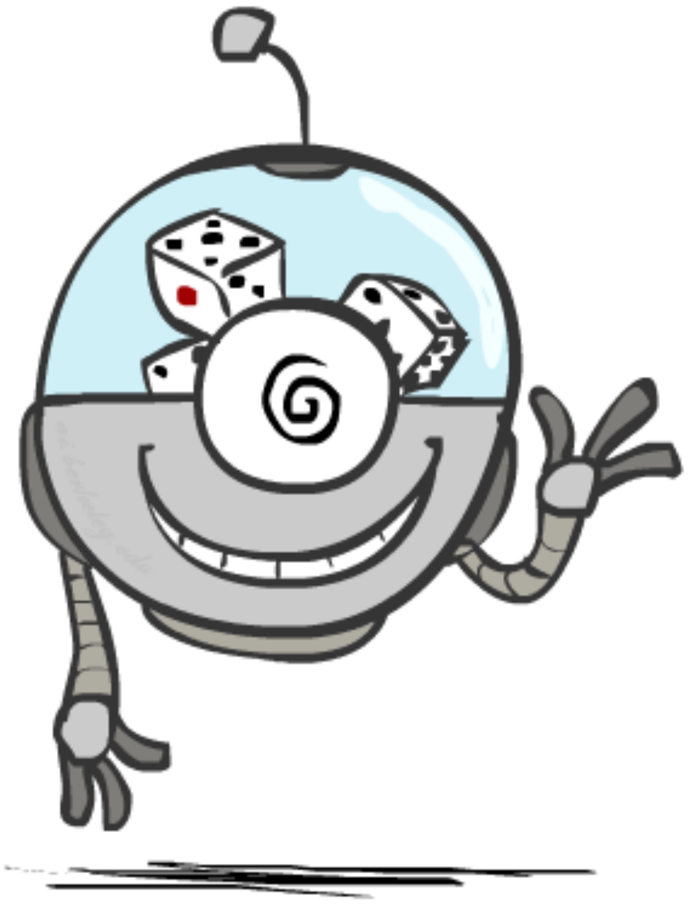


# Depth-Limited Expectimax



# Probabilities



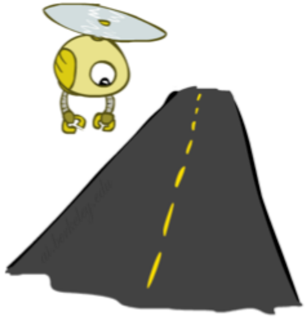
# Reminder: Probabilities

- 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



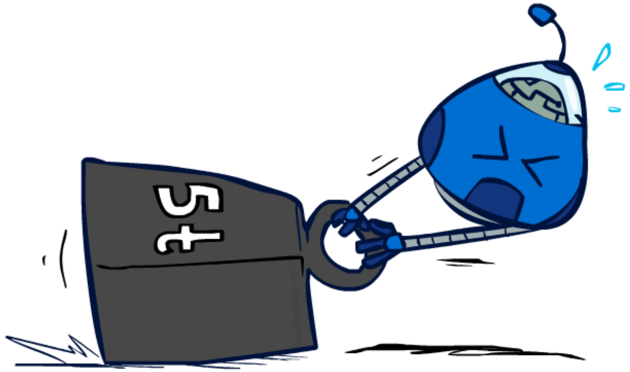
0.25

0.50

0.25

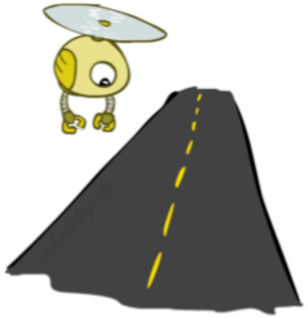
# Reminder: Expectations

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




- Example: How long to get to the airport?

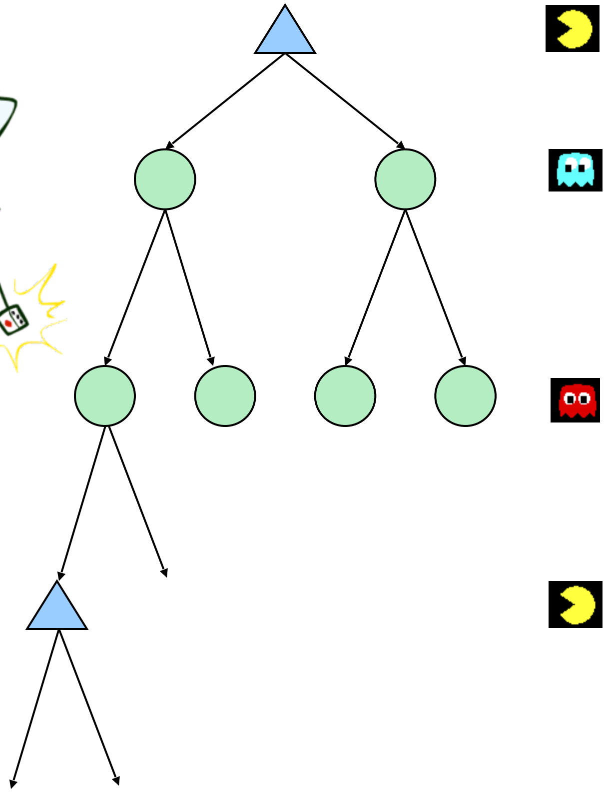
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Time: | 20 min | + | 30 min | + | 60 min | | |
| | x | | x | | x | → | 35 min |
| Probability: | 0.25 | | 0.50 | | 0.25 | | |



# What Probabilities to Use?

- In expectimax search, we have a probabilist  model of how the opponent (or environment) behave in any state
  - Model could be a simple uniform distribution (roll a
  - Model could be sophisticated and require a great deal 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!

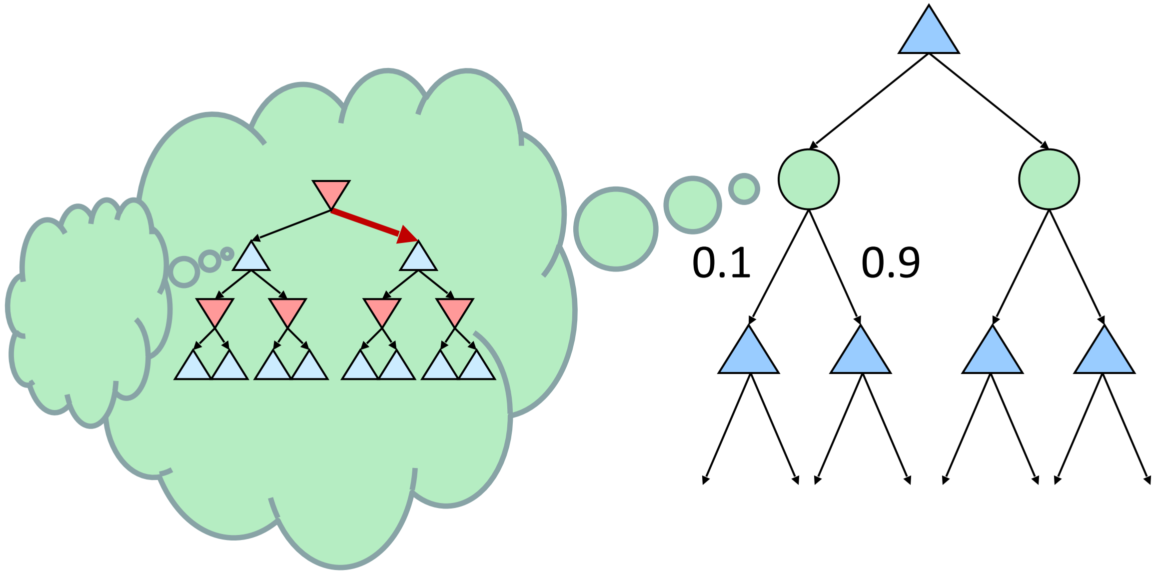
- For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes



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

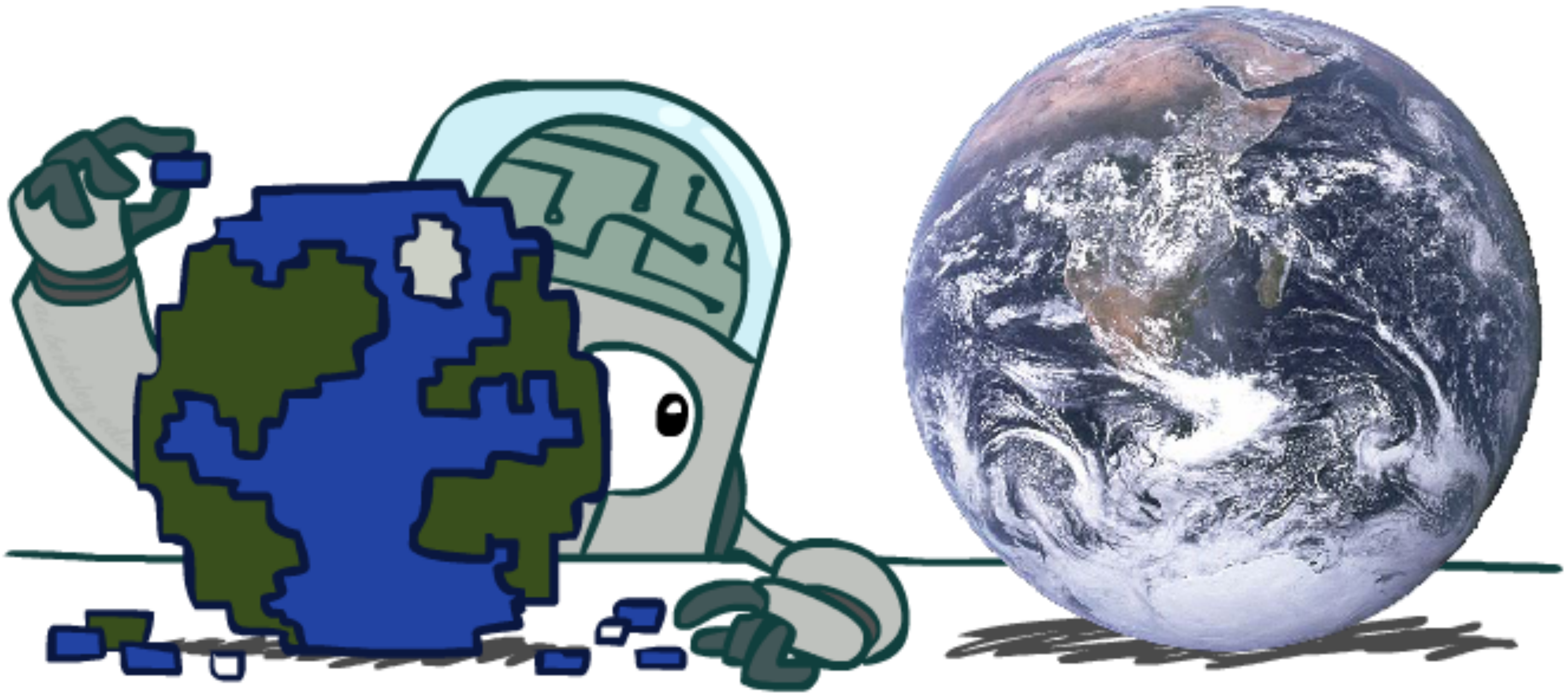
# Quiz: Informed Probabilities

- 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 and maximax, which have the nice property that it all collapses into one game tree

# Modeling Assumptions



# The Dangers of Optimism and Pessimism

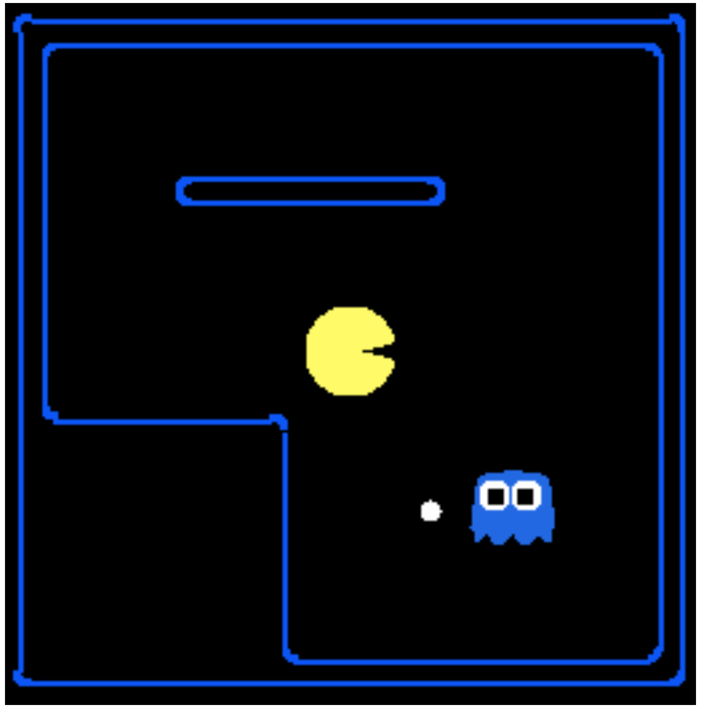
## Dangerous Optimism

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 | | |
| Expectimax Pacman | | |

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

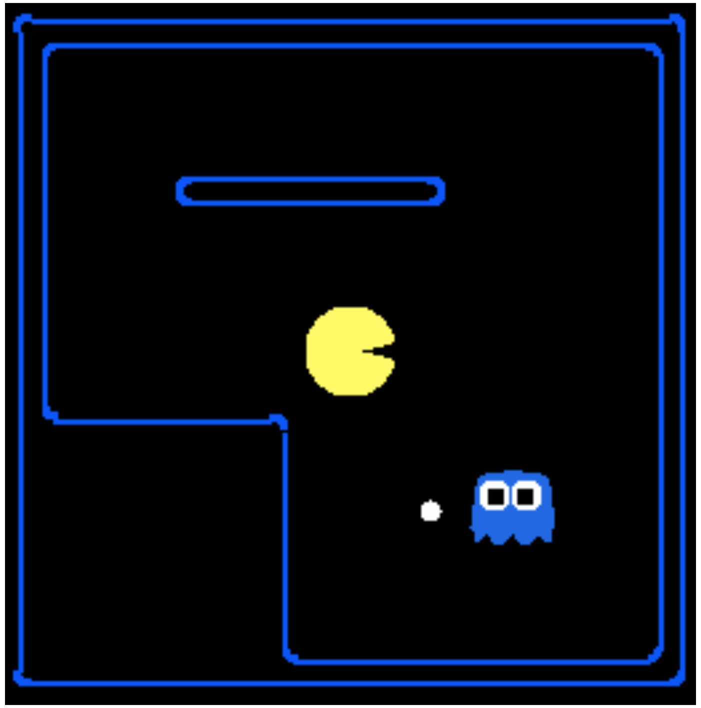
# Video of Demo World Assumptions
# Random Ghost – Expectimax Pacman

# Video of Demo World Assumptions
# Adversarial Ghost – Minimax Pacman

# Video of Demo World Assumptions
# Random Ghost – Minimax Pacman

# Video of Demo World Assumptions
## Adversarial Ghost – Expectimax Pacman

# Assumptions vs. Reality



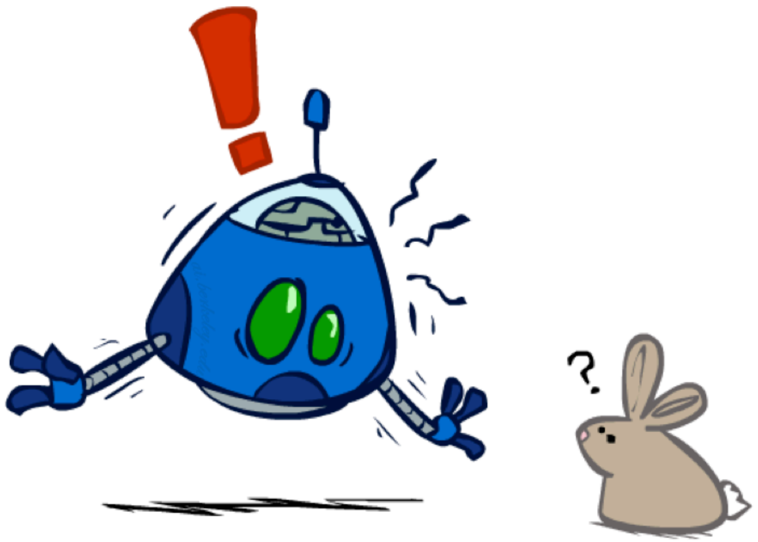
| | Adversarial Ghost | Random Ghost |
|---|---|---|
| Minimax Pacman | Won 5/5<br>Avg. Score: 483 | Won 5/5<br>Avg. Score: 493 |
| Expectimax Pacman | Won 1/5<br>Avg. Score: -303 | Won 5/5<br>Avg. Score: 503 |

Results from playing 5 games

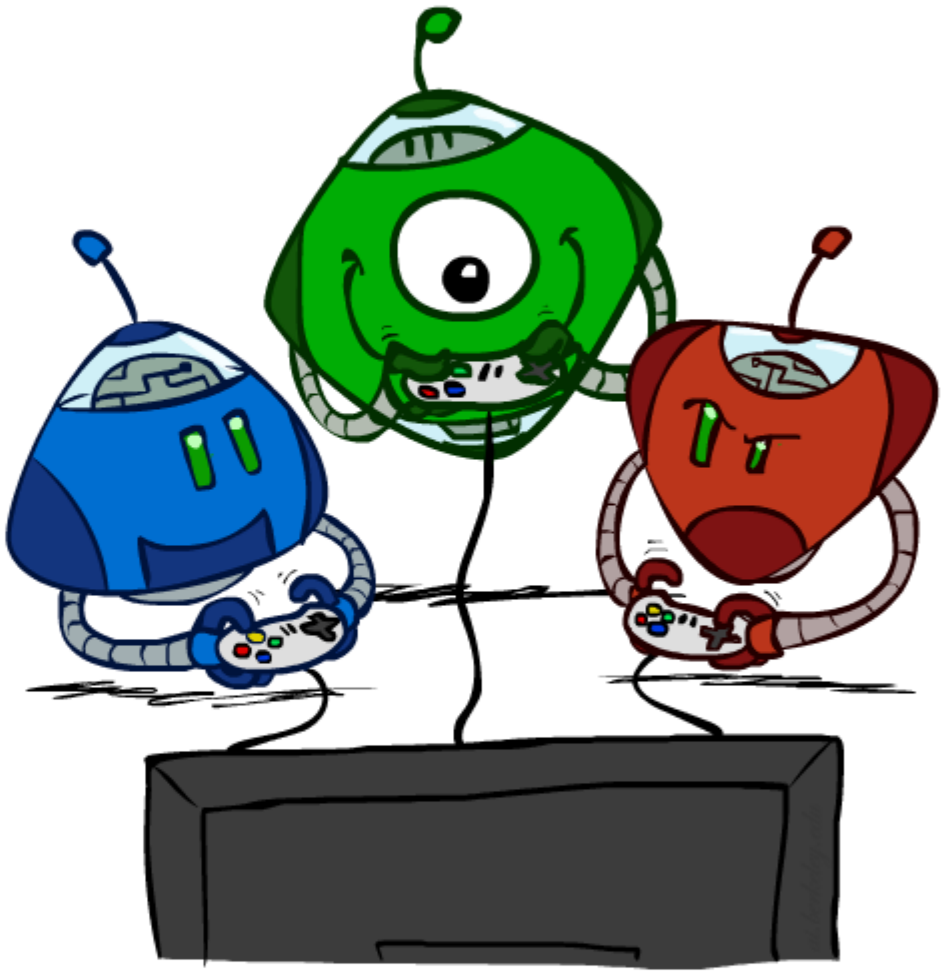
Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

# Why not minimax?

- Worst case reasoning is too conservative
- Need average case reasoning

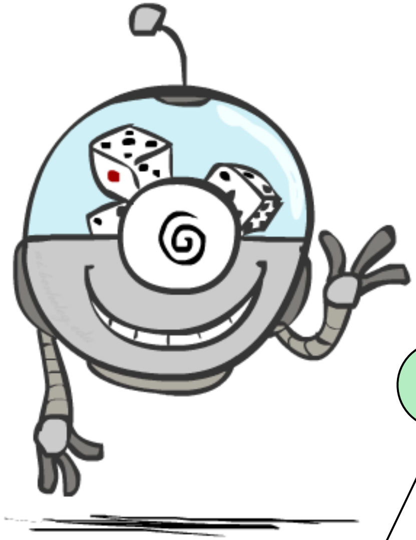


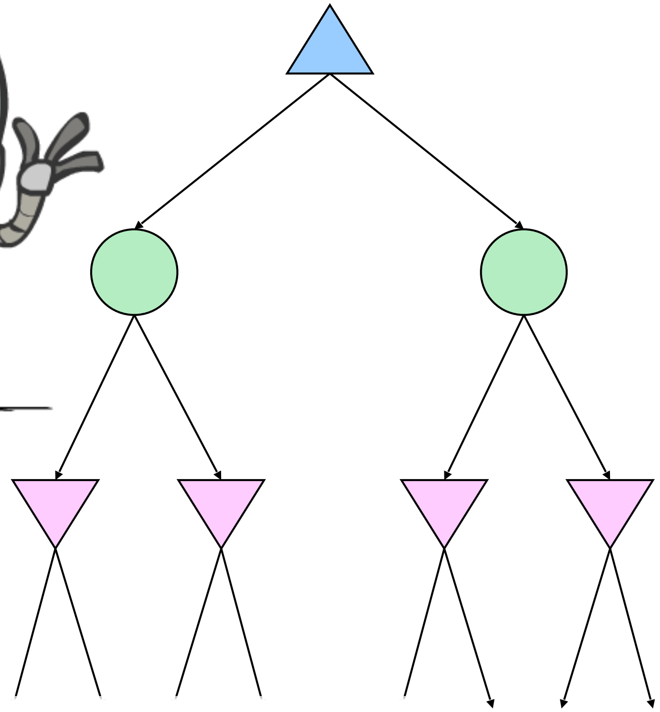
# Other Game Types



# Mixed Layer Types

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





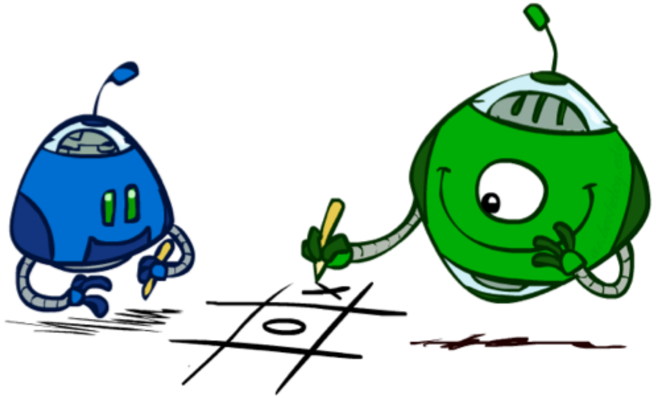
**if** *state* is a MAX node **then**
    **return** the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
**if** *state* is a MIN node **then**
    **return** the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
**if** *state* is a chance node **then**
    **return** average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

# Example: Backgammon

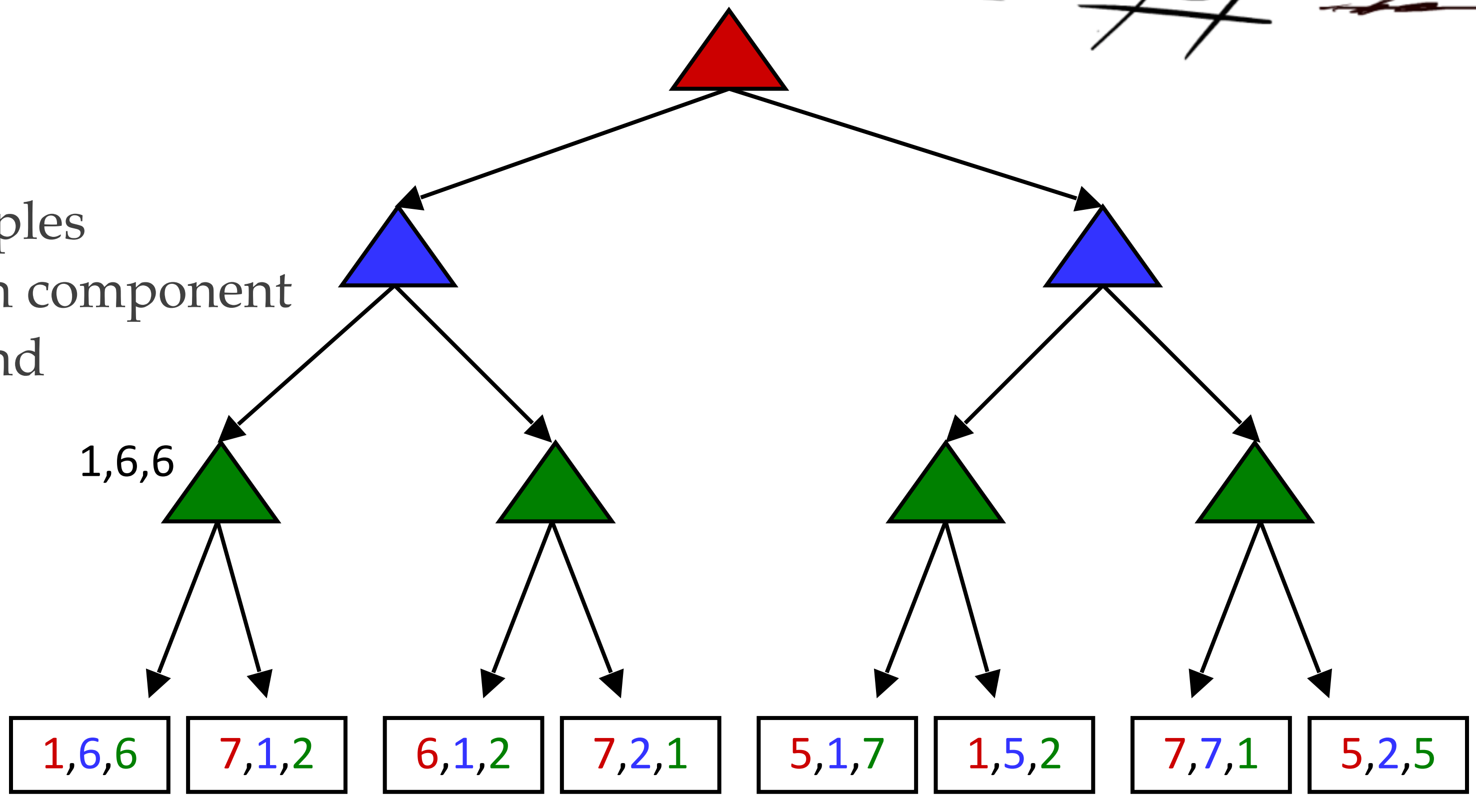
- Dice rolls increase *b*: 21 possible rolls with 2 dice
  - Backgammon ≈ 20 legal moves
  - Depth 2 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$
- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier…
- Historic AI: TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play
- 1st AI world champion in any game!

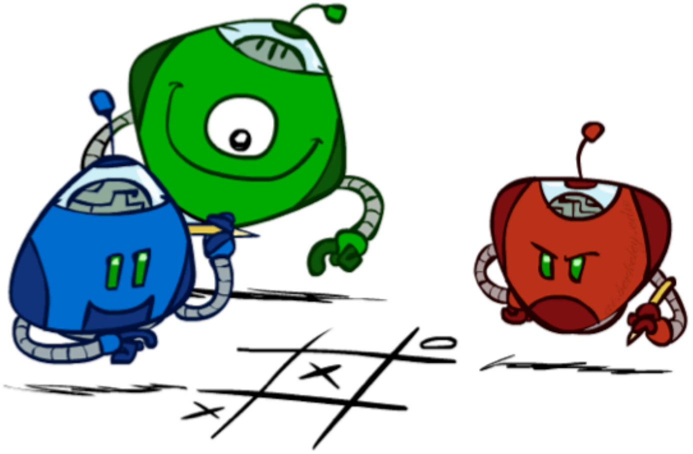
Image: Wikipedia

# Multi-Agent Utilities



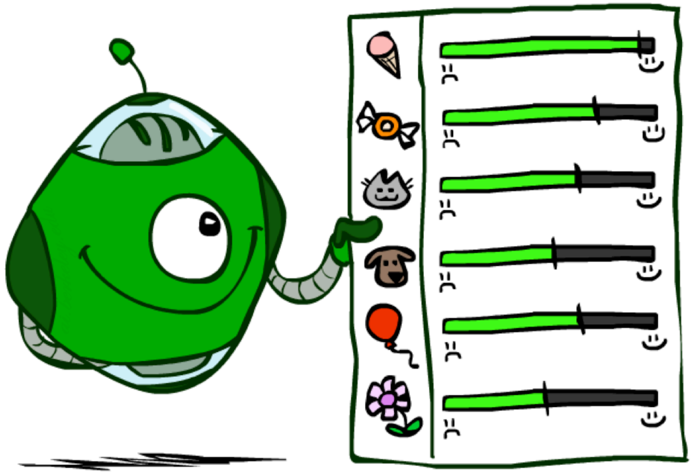
- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…



# Utilities

- Utilities: values that we assign to every state



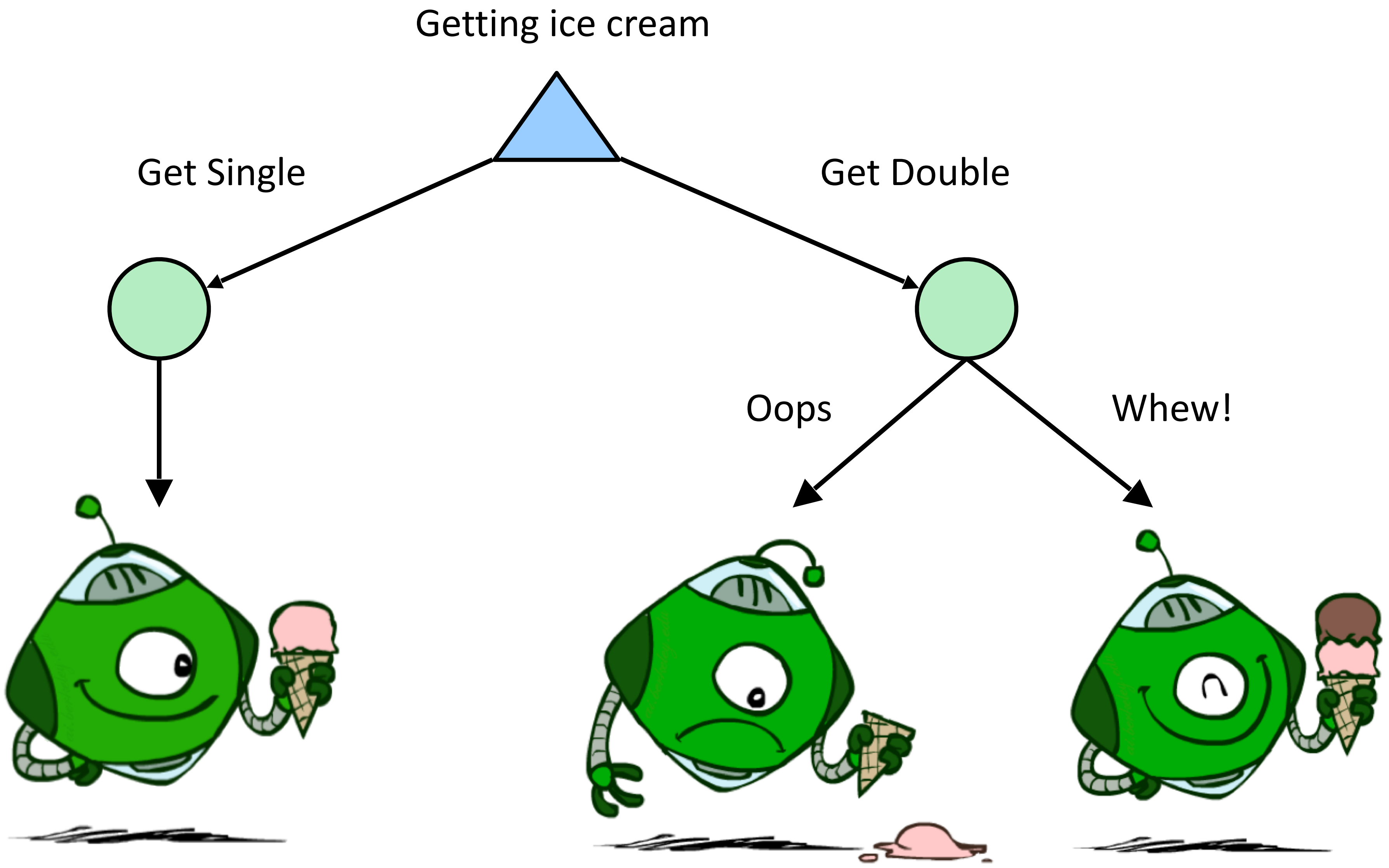
- Why should we average utilities?  Why not minimax?

- Principle of maximum expected utility:
  - A rational agent should choose the action that maximizes its expected utility, given its knowledge

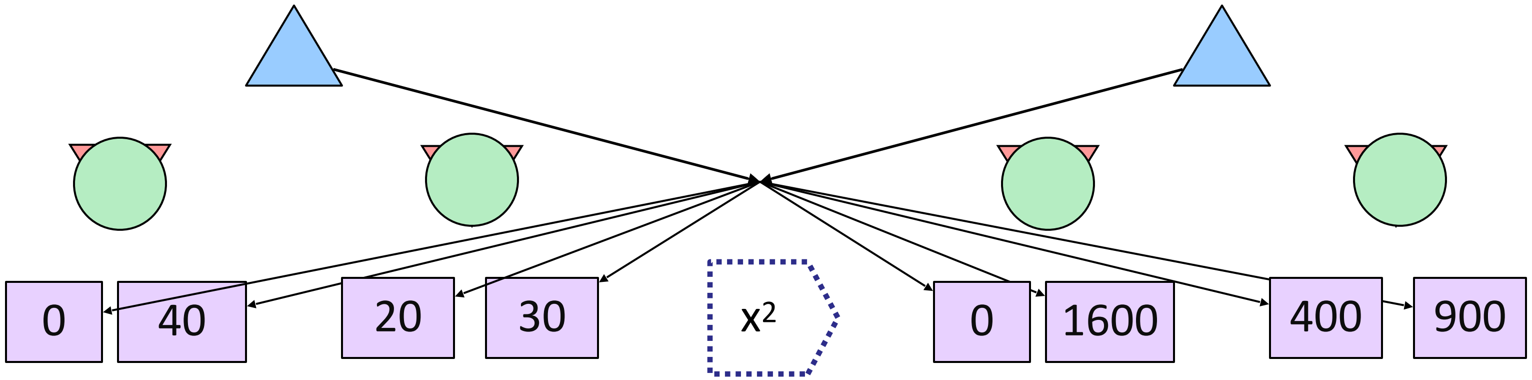
# Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent's preferences
- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent's goals
- We hard-wire utilities and let behaviors emerge

# Utilities: Uncertain Outcomes



# What Utilities to Use?




- For worst-case minimax reasoning, terminal function scale doesn't matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
- For average-case expectimax reasoning, we need *magnitudes* to be meaningful

# Next Time: MDPs!