

---

# CSE 573: Artificial Intelligence

## Winter 2019

Hanna Hajishirzi  
Expectimax – Complex Games

slides from

Dan Klein, Stuart Russell, Andrew Moore, Dan Weld, Pieter Abbeel, Luke Zettlemoyer

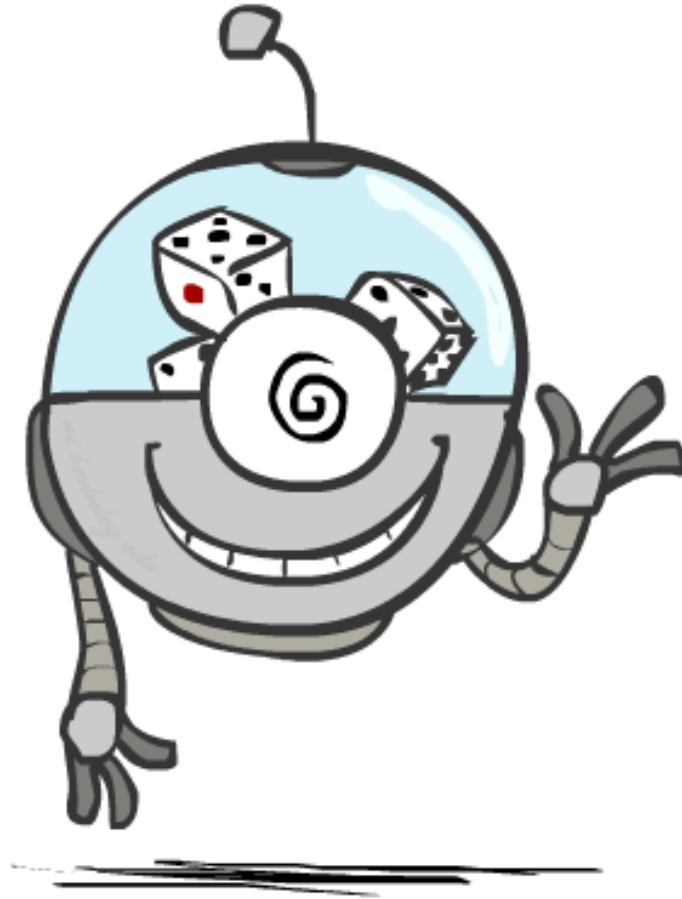
# Today's lecture

---

- Announcements:
  - PS2 is due on Wed.
  - Quiz1 is next Monday
  
- Outline:
  - Expectimax
  - Complex Games
  - MDPs

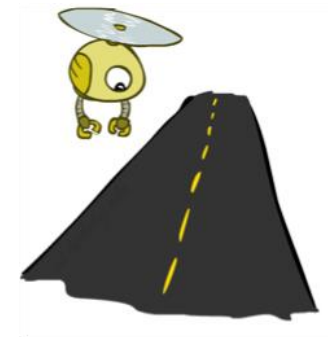
# Probabilities

---

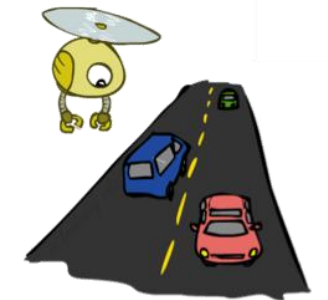


# Reminder: Probabilities

- 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:  $P(T=\text{none}) = 0.25$ ,  $P(T=\text{light}) = 0.50$ ,  $P(T=\text{heavy}) = 0.25$
- 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:
  - $P(T=\text{heavy}) = 0.25$ ,  $P(T=\text{heavy} \mid \text{Hour}=8\text{am}) = 0.60$
  - We'll talk about methods for reasoning and updating probabilities later



0.25



0.50

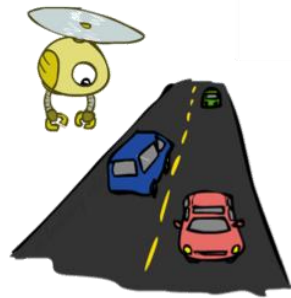
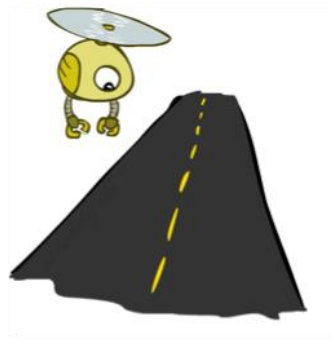


0.25

# Reminder: Expectations

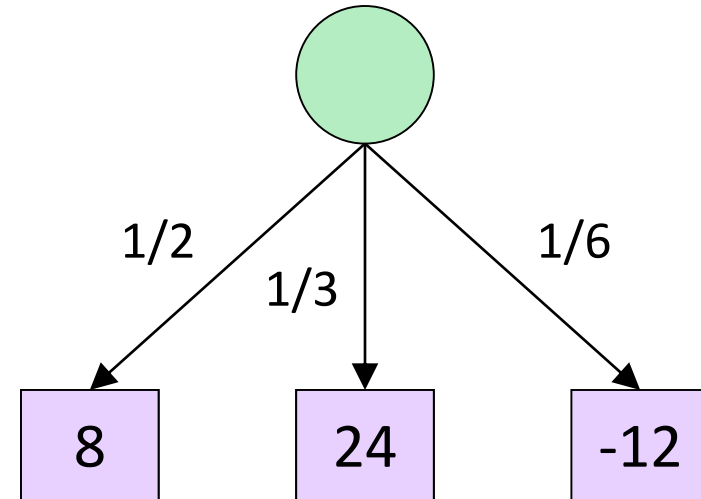
- The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes
- Example: How long to get to the airport?

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



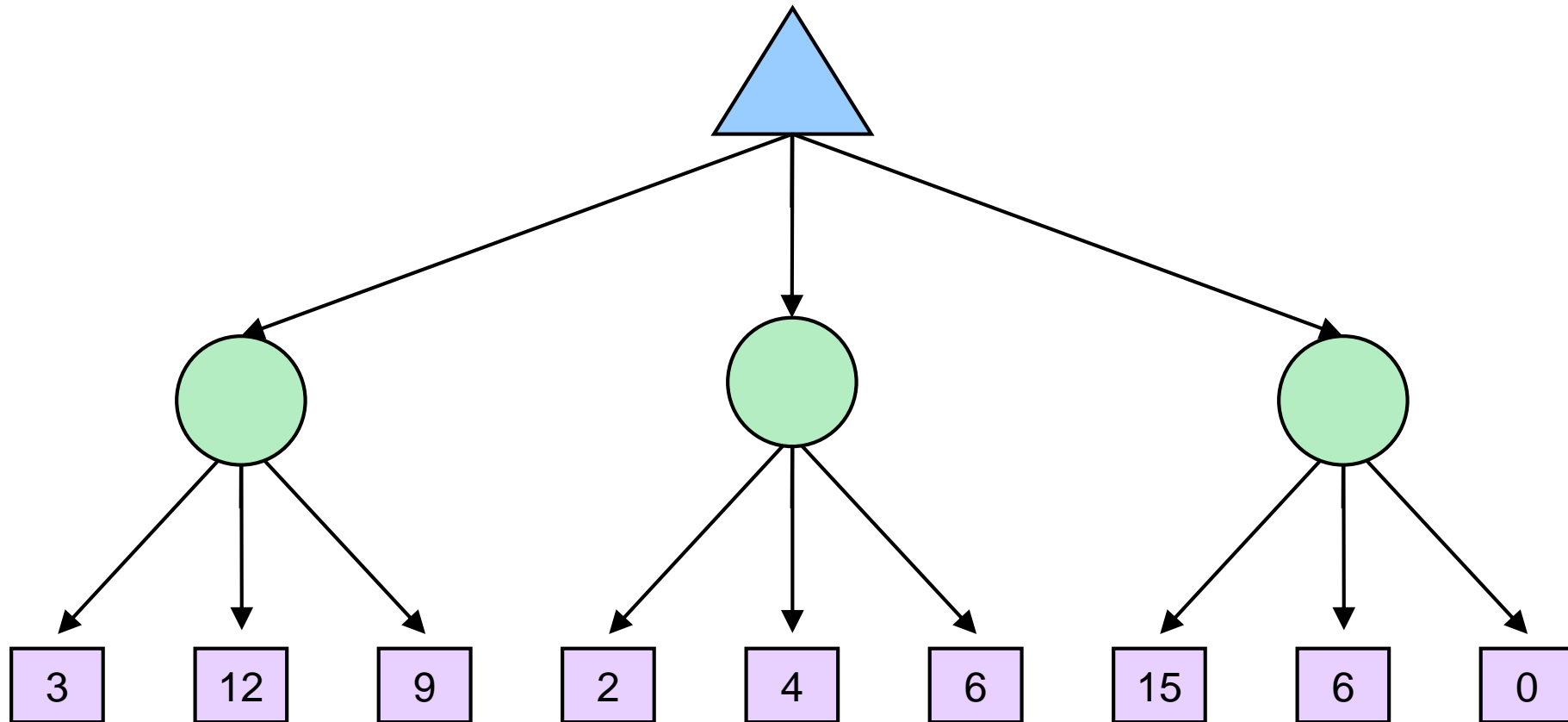
# Expectimax Pseudocode

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

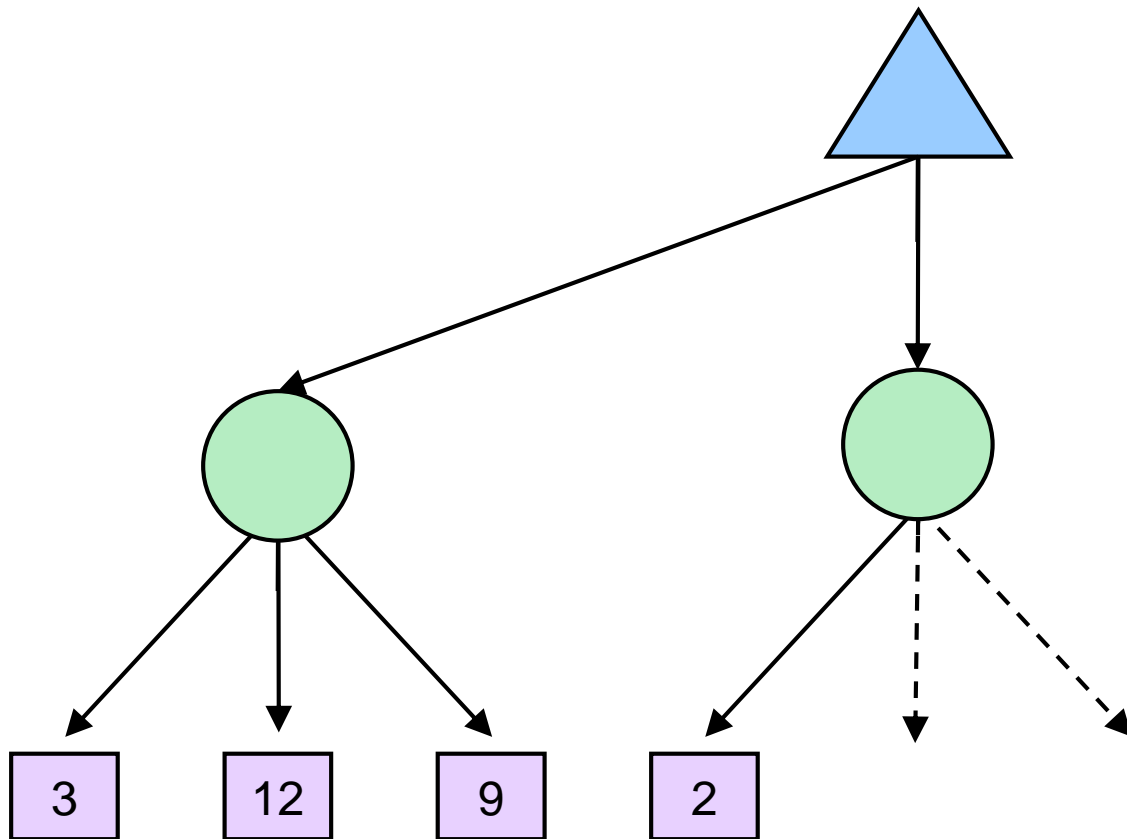


$$v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10$$

# Expectimax Example

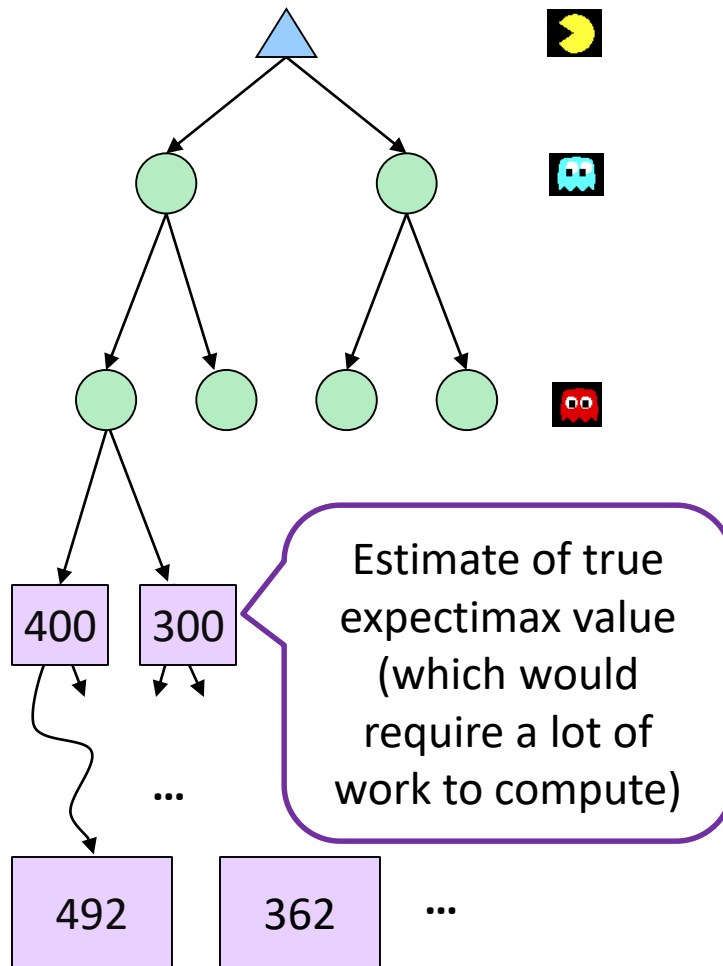


# Expectimax Pruning?



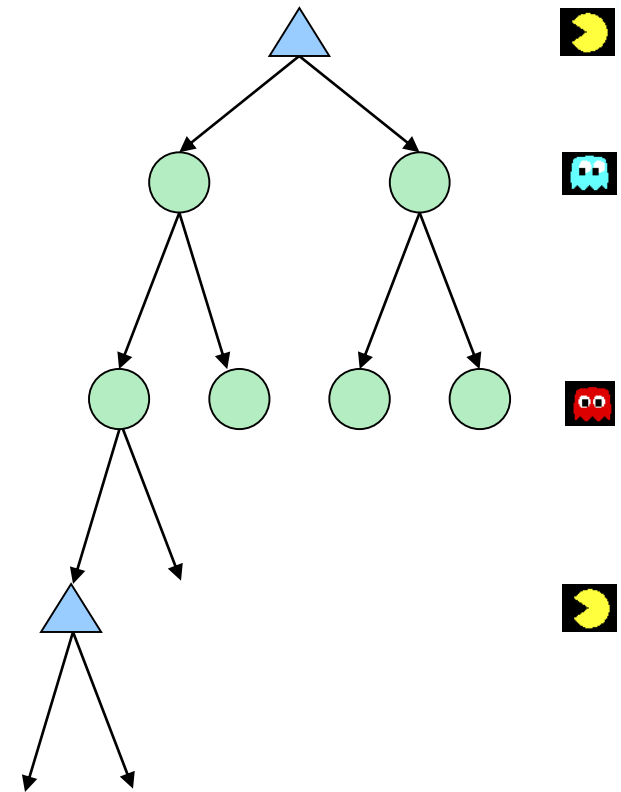


# Depth-Limited Expectimax



# What Probabilities to Use?

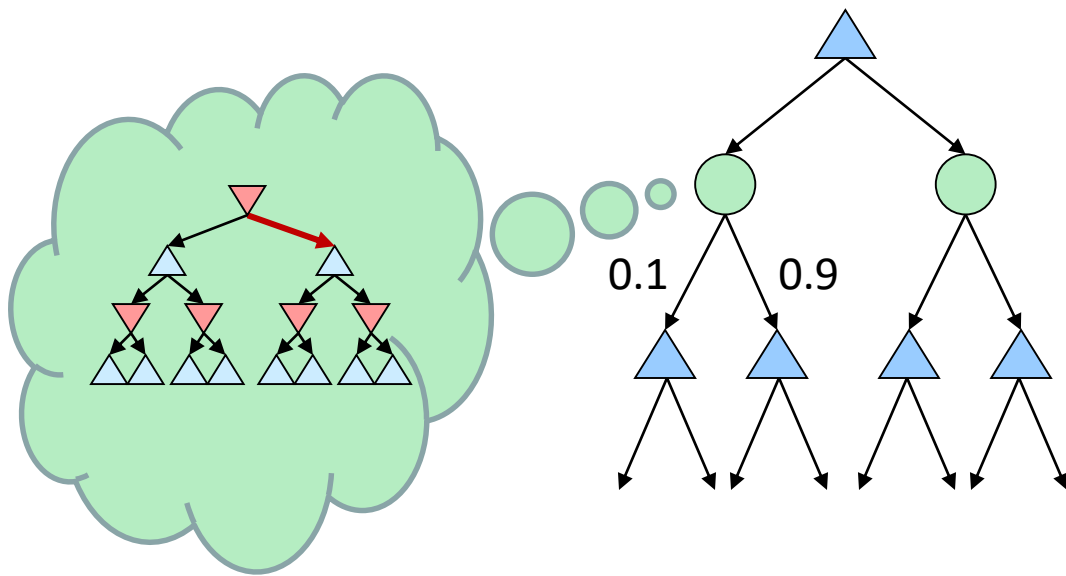
- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our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



*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, which has 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

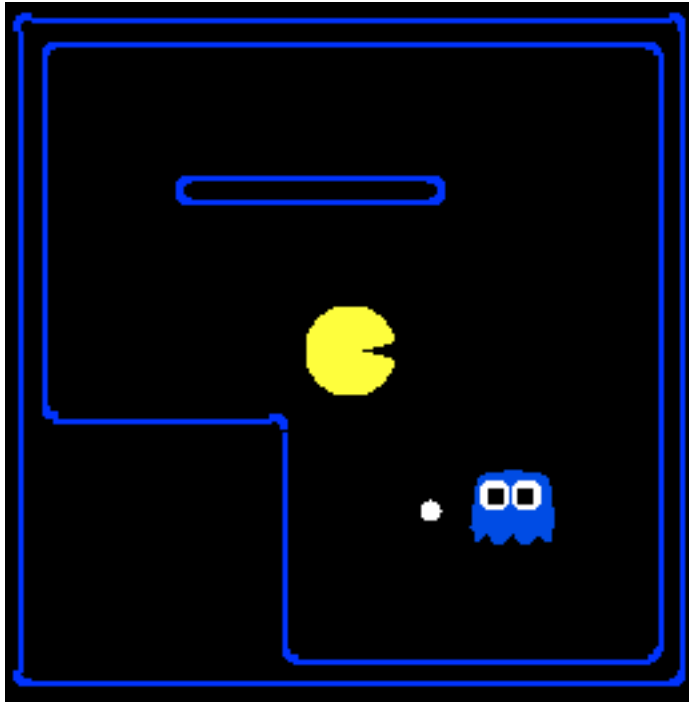


## Dangerous Pessimism

Assuming the worst case when it's not likely



# Assumptions vs. Reality



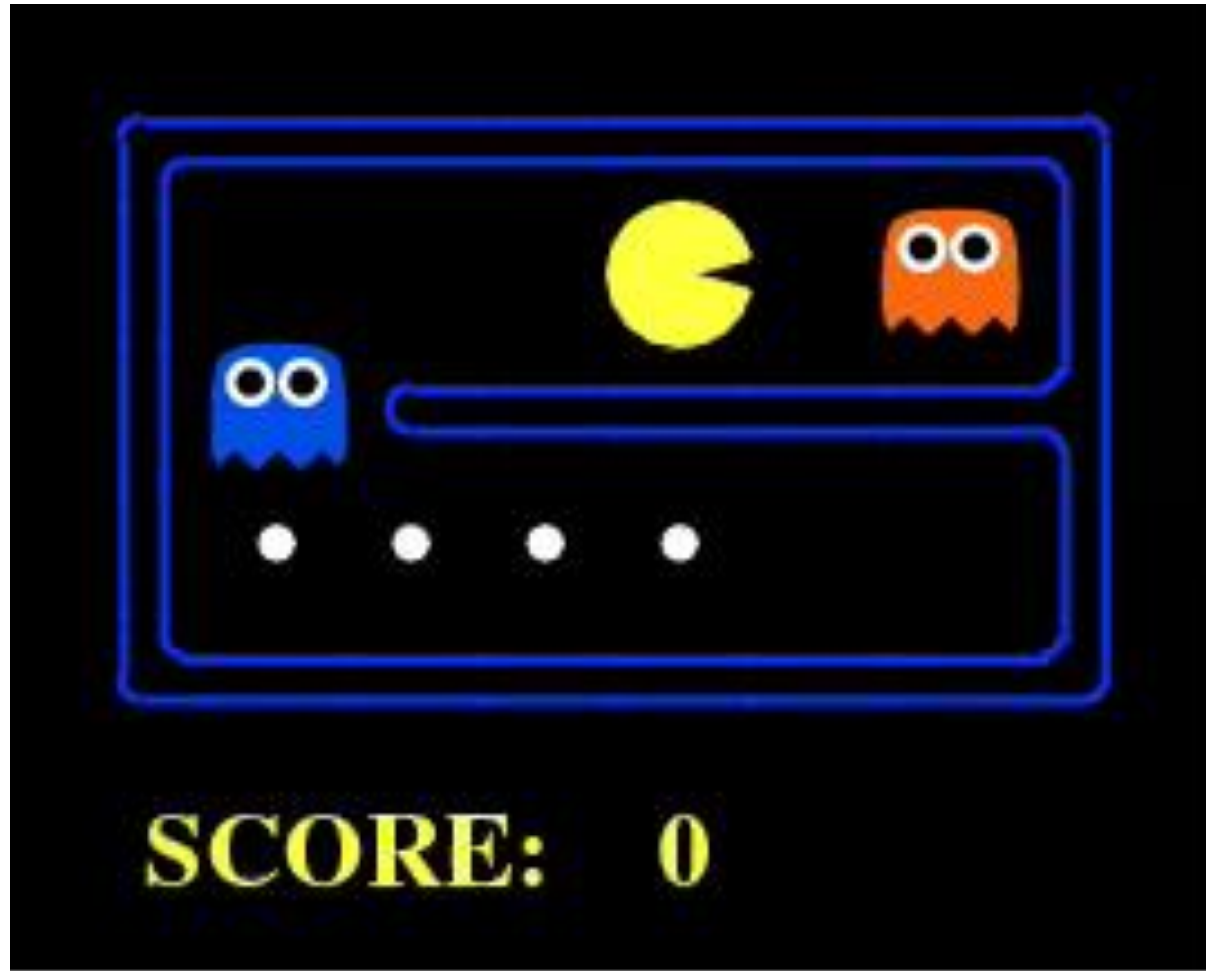
	Adversarial Ghost	Random Ghost
Minimax Pacman		
Expectimax Pacman		

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



# Minimax vs Expectimax (Min)

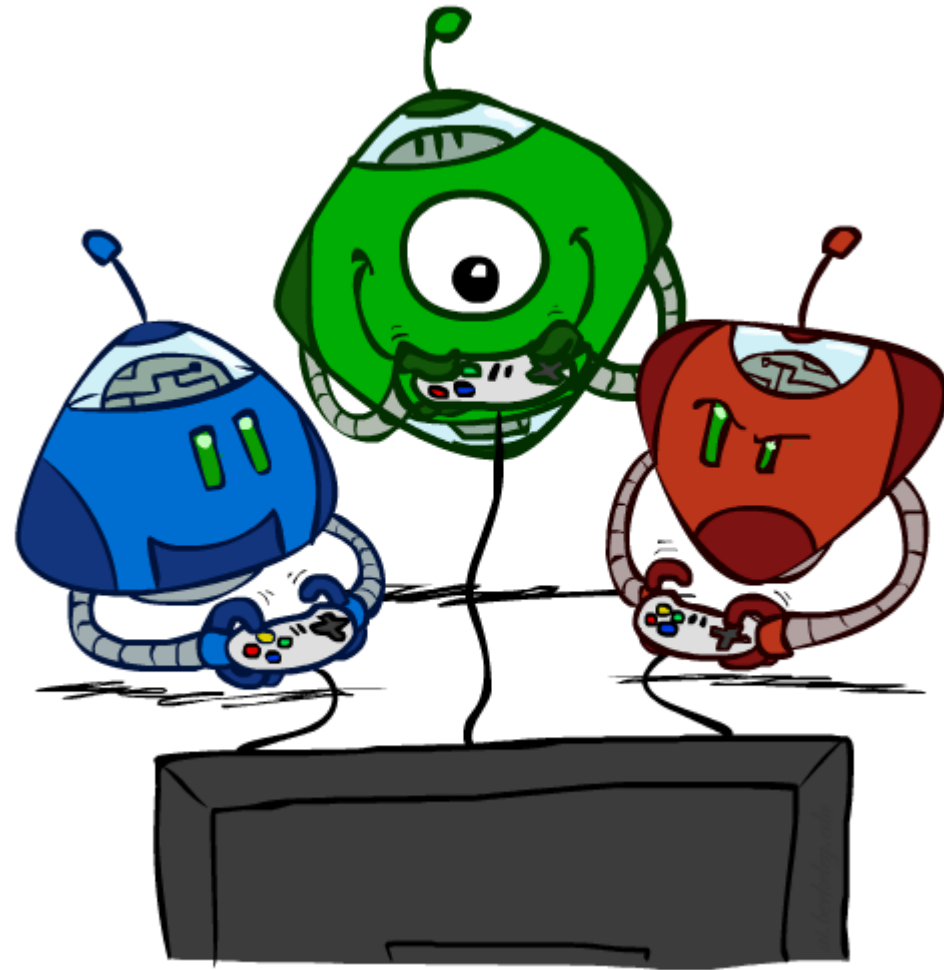


No point in trying



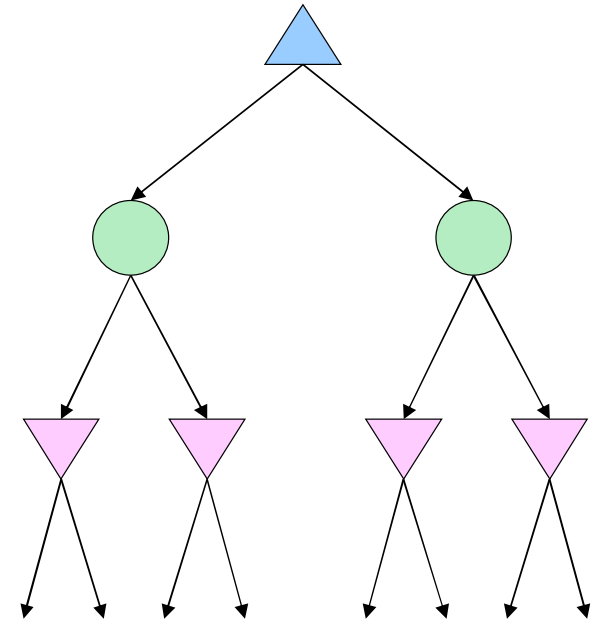
# Other Game Types

---



# Mixed Layer Types

- E.g. Backgammon
- Expectiminimax
  - 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*)

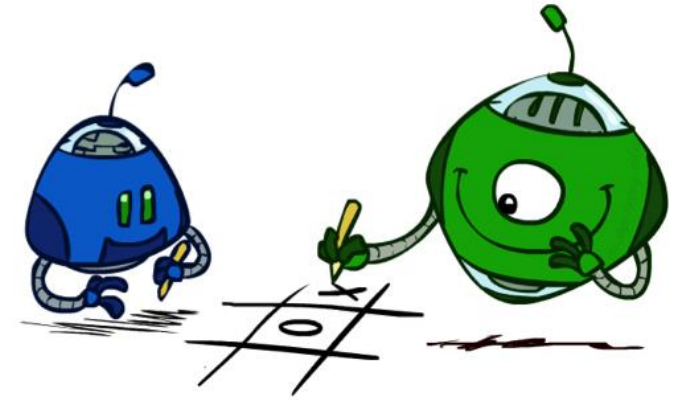
**if** *state* is a chance node **then**  
    **return** average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

# Example: Backgammon

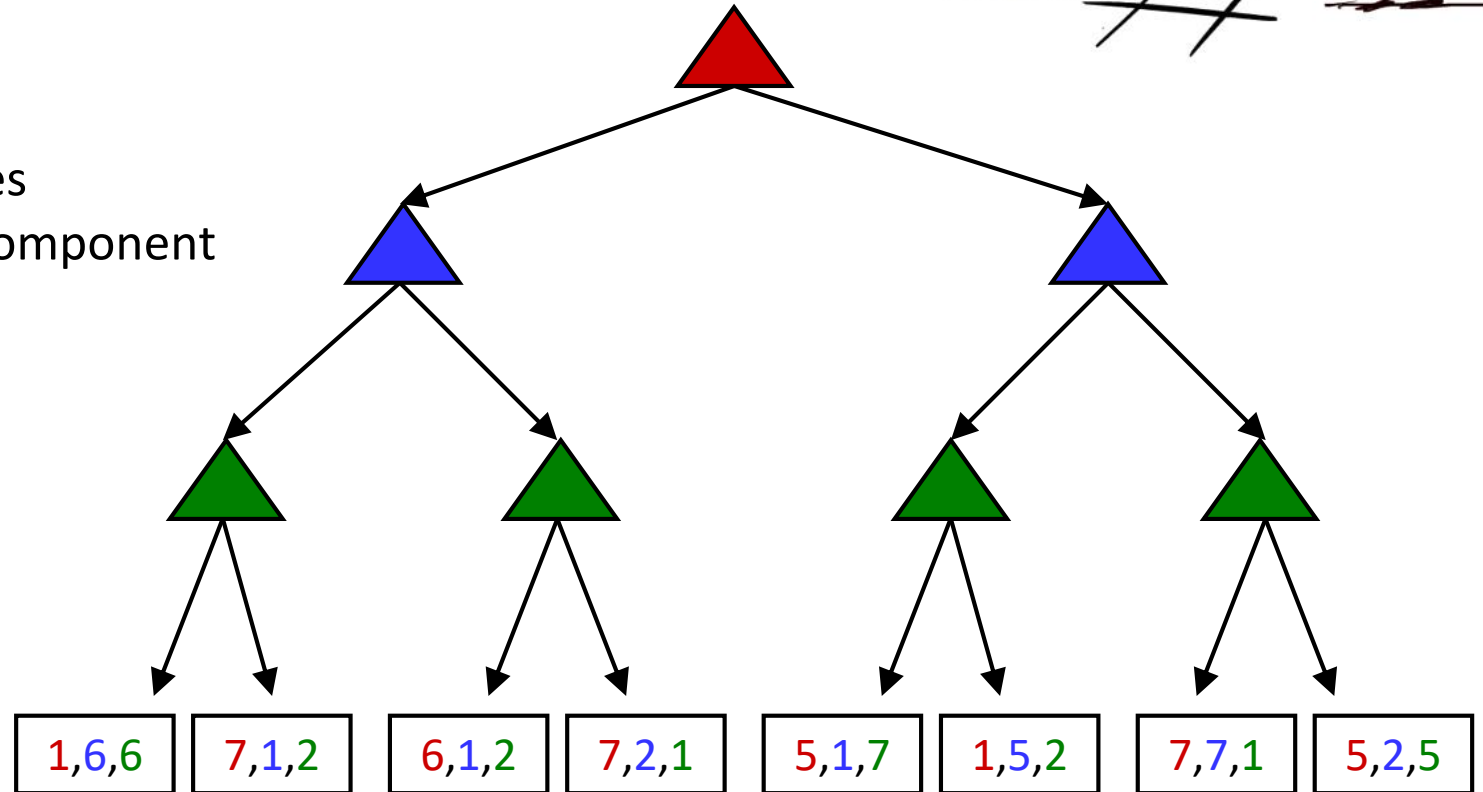
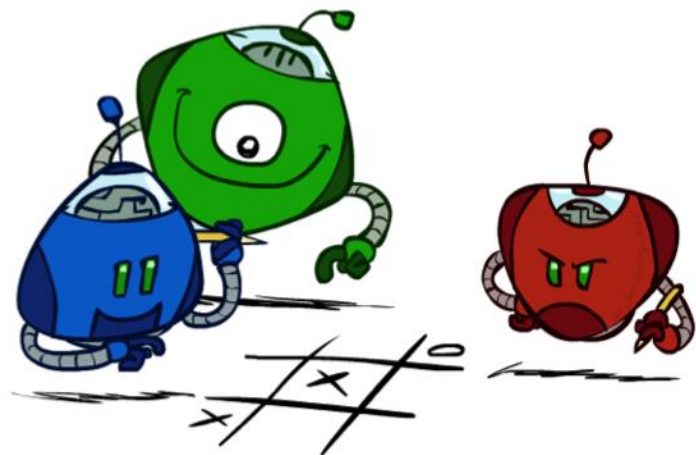
- Dice rolls increase  $b$ : 21 possible rolls with 2 dice
  - Backgammon  $\approx$  20 legal moves
  - Depth 2 =  $20 \times (21 \times 20)^3 = 1.2 \times 10^9$
  - Depth 2  $\rightarrow$  2 rounds adversary plays
- As depth increases, probability of reaching a given search node shrinks
  - So usefulness 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
- 1<sup>st</sup> AI world champion in any game!



# 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...



# Next Time: MDPs!

---