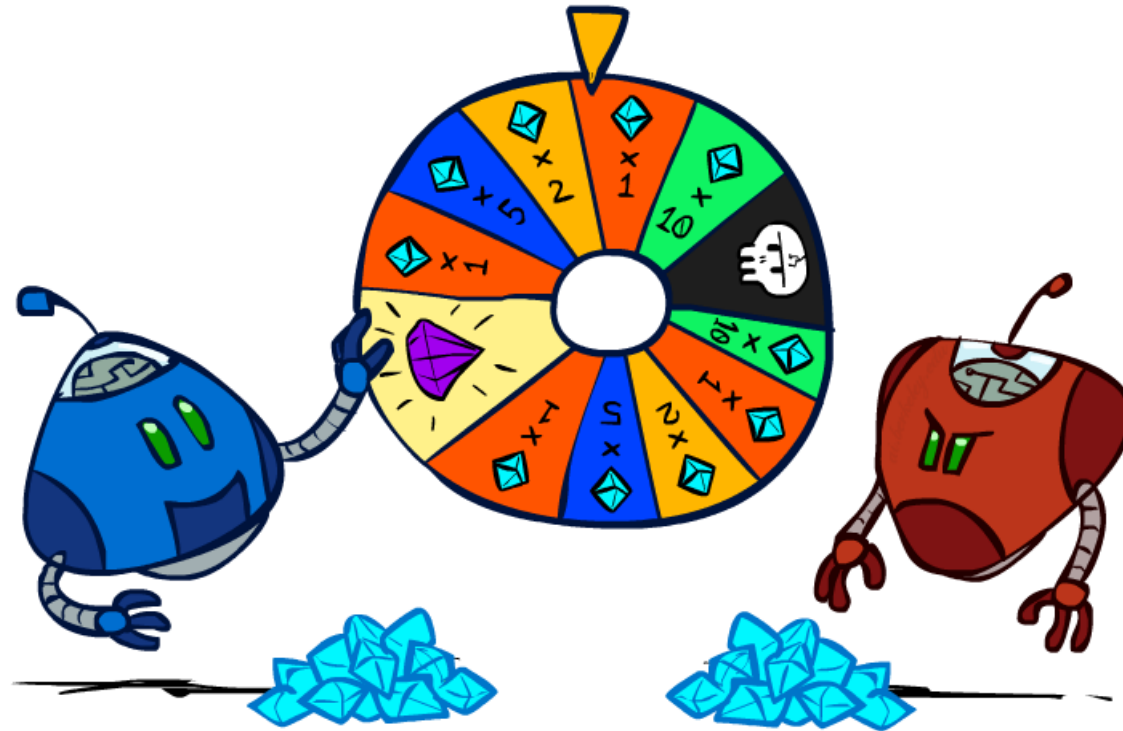


# CSE 573: Artificial Intelligence

## ExpectiMax – Complex Games

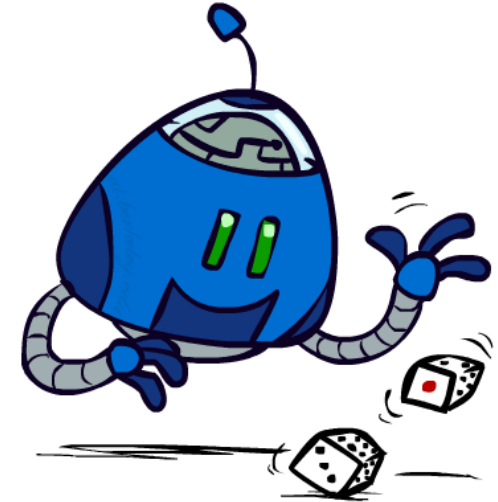
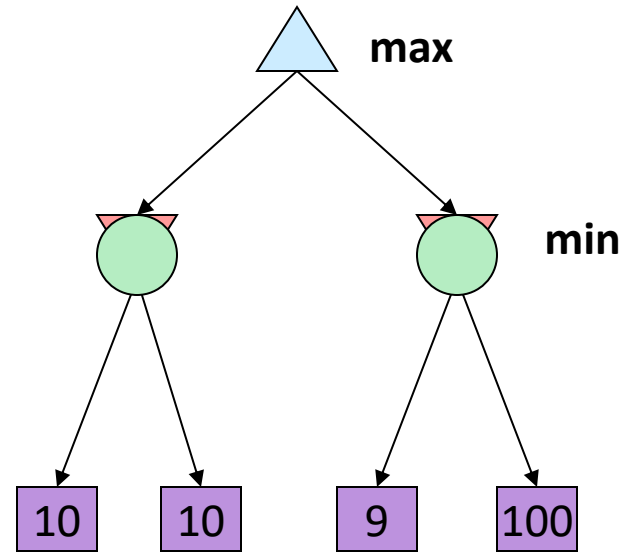
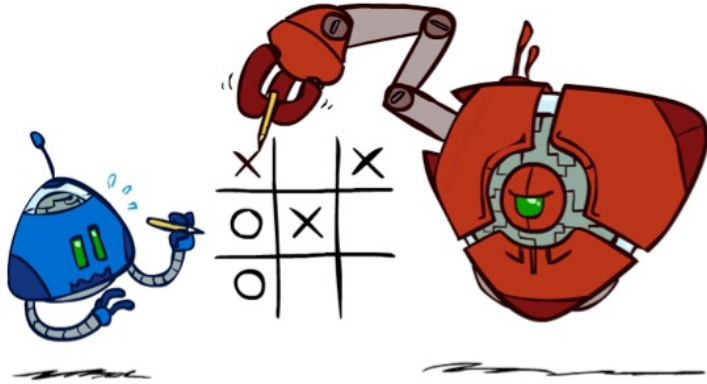


slides adapted from  
Stuart Russel, Dan Klein, Pieter Abbeel from [ai.berkeley.edu](http://ai.berkeley.edu)  
And Hanna Hajishirzi, Jared Moore, Dan Weld

# Video of Demo Min vs. Exp (Min)



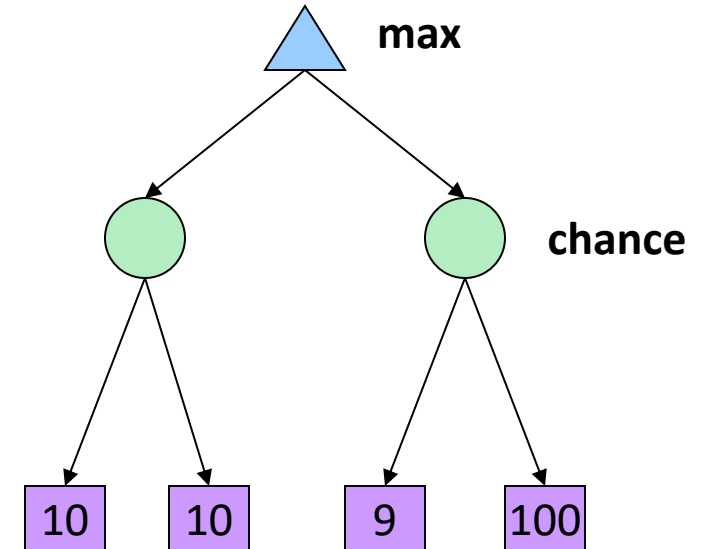
# Worst-Case vs. Average Case



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

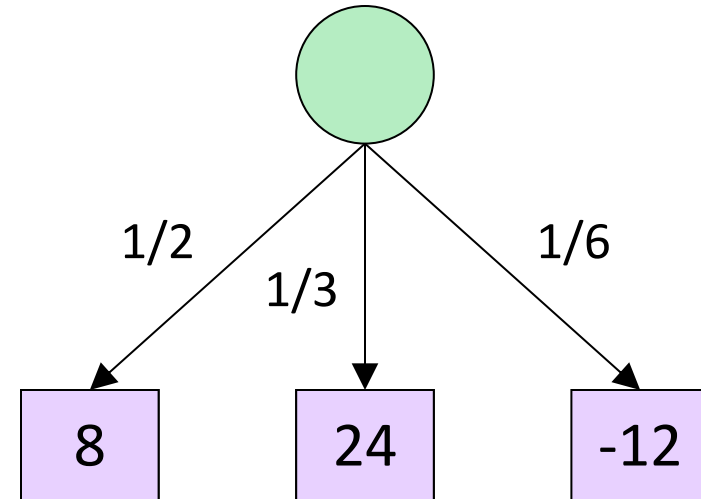
# Expectimax Search

- 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**



# Expectimax Pseudocode

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



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

# Minimax

function `decision(s)` returns an action

return the action `a` in `Actions(s)` with the highest `minimax_value(Result(s,a))`



function `minimax_value(s)` returns a value

if `Terminal-Test(s)` then return `Utility(s)`

if `Player(s) = MAX` then return `maxa in Actions(s) minimax_value(Result(s,a))`

if `Player(s) = MIN` then return `mina in Actions(s) minimax_value(Result(s,a))`

# Expectiminimax

function `decision(s)` returns an action

return the action `a` in `Actions(s)` with the highest  
`value(Result(s,a))`



function `value(s)` returns a value

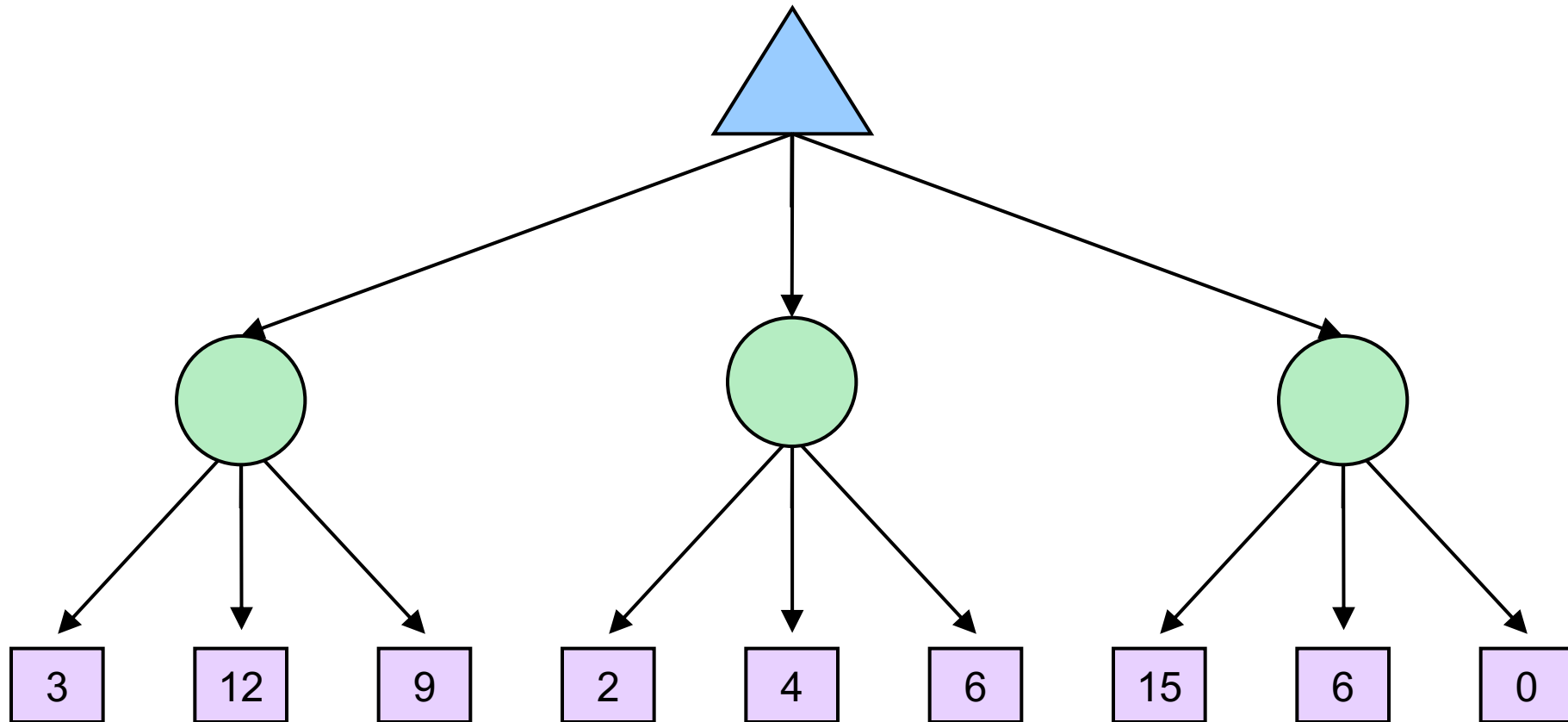
if `Terminal-Test(s)` then return `Utility(s)`

if `Player(s) = MAX` then return  $\max_{a \in \text{Actions}(s)} \text{value}(\text{Result}(s,a))$

if `Player(s) = MIN` then return  $\min_{a \in \text{Actions}(s)} \text{value}(\text{Result}(s,a))$

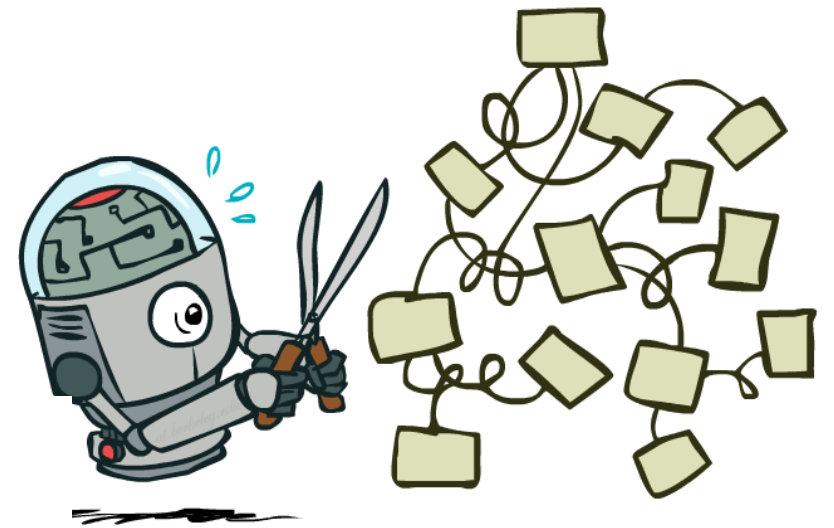
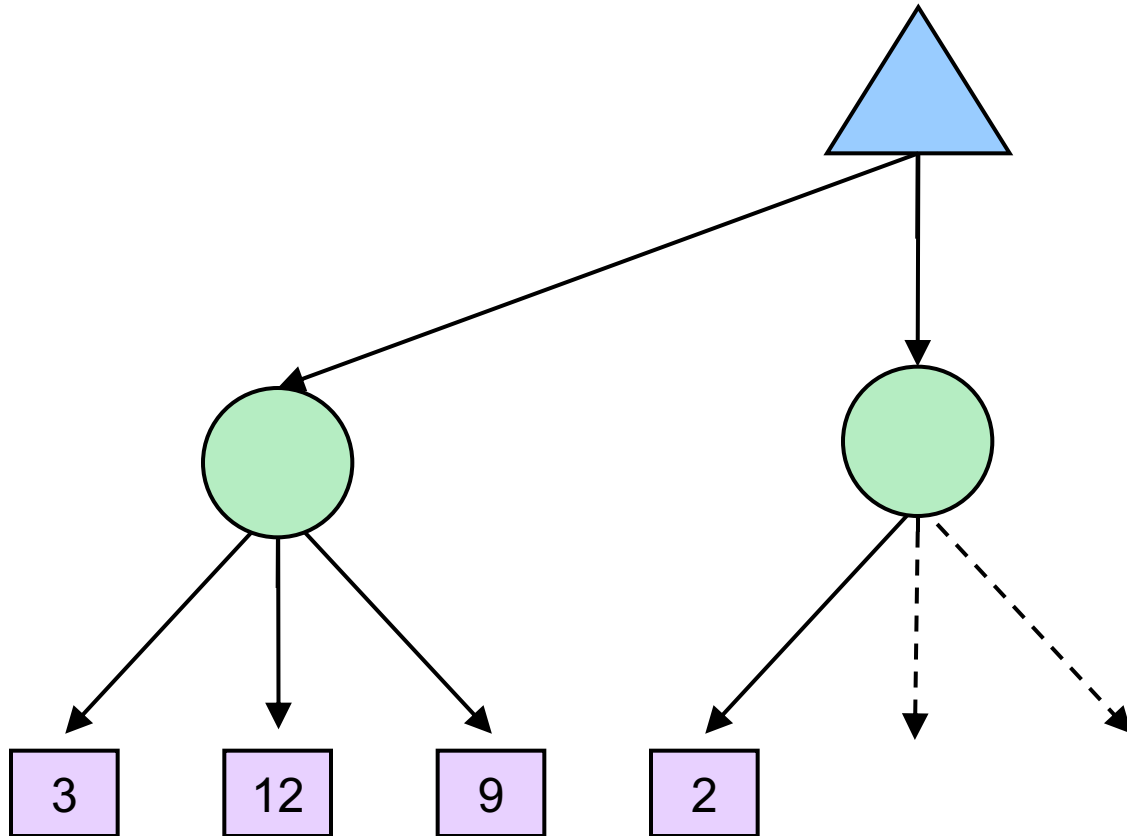
if `Player(s) = CHANCE` then return  $\sum_{a \in \text{Actions}(s)} \text{Pr}(a) * \text{value}(\text{Result}(s,a))$

# Expectimax Example

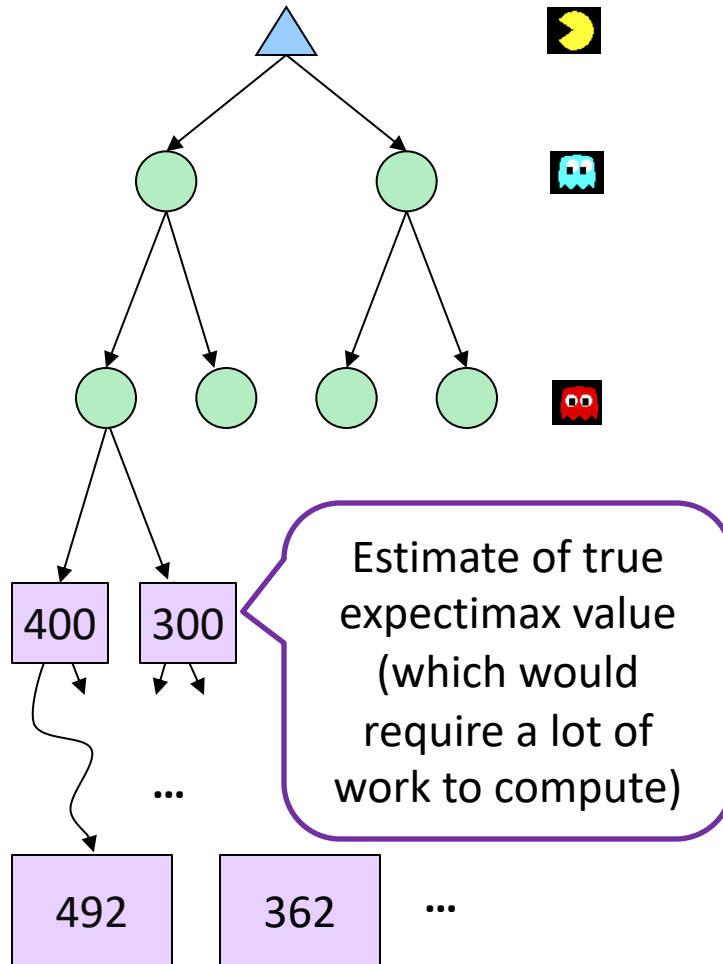




# Expectimax Pruning?

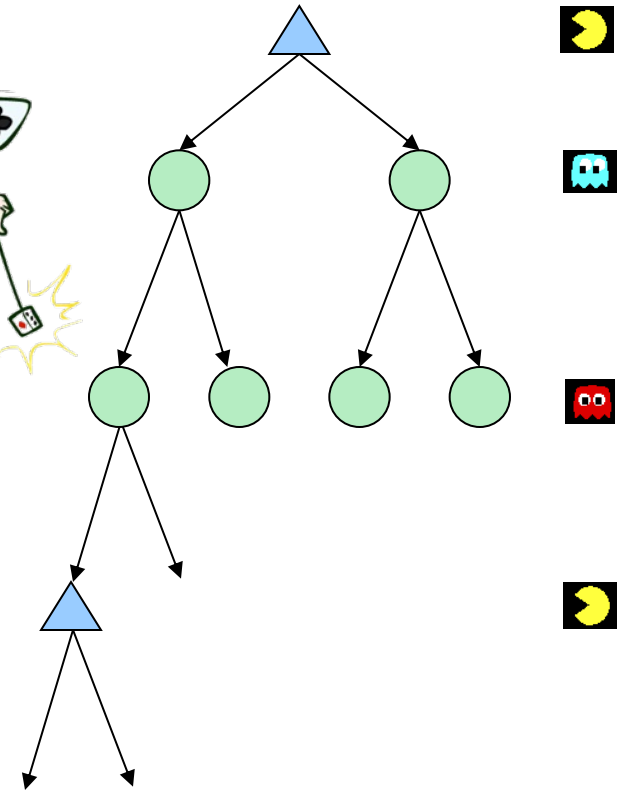


# Depth-Limited Expectimax



# What Probabilities to Use?

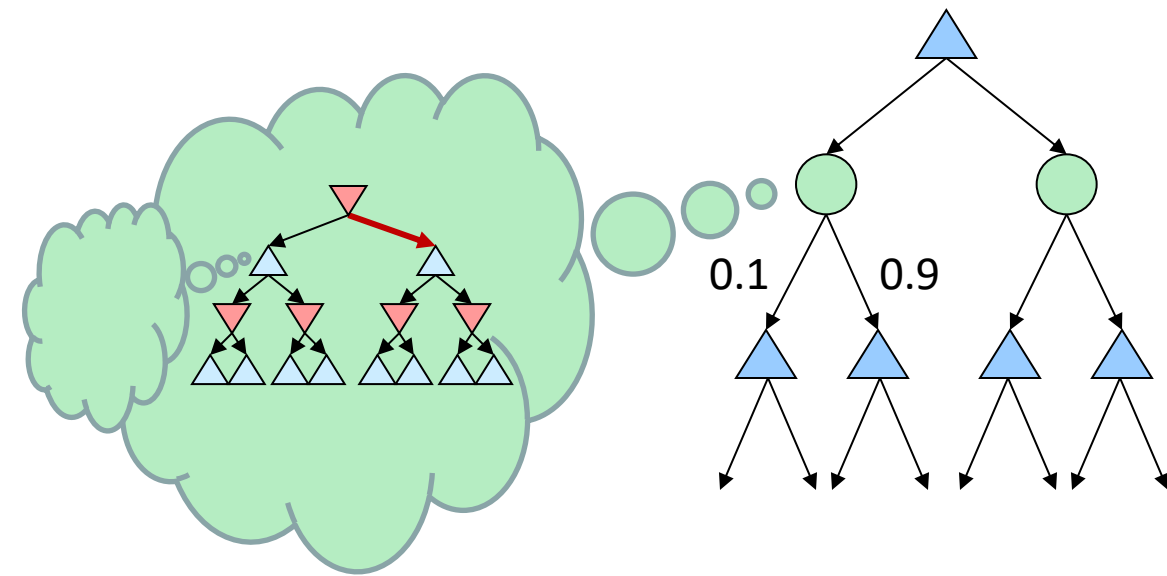
- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our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

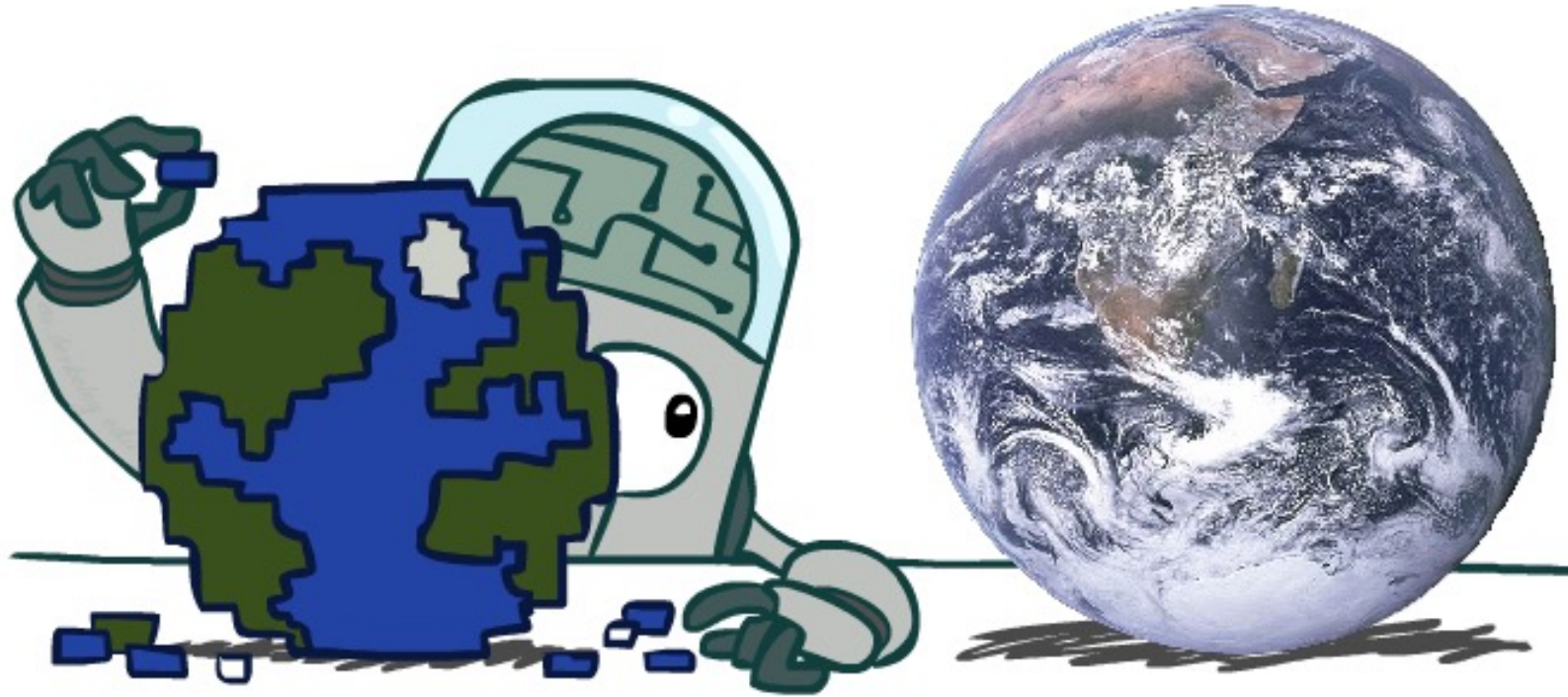
# Quiz: Informed Probabilities

- 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

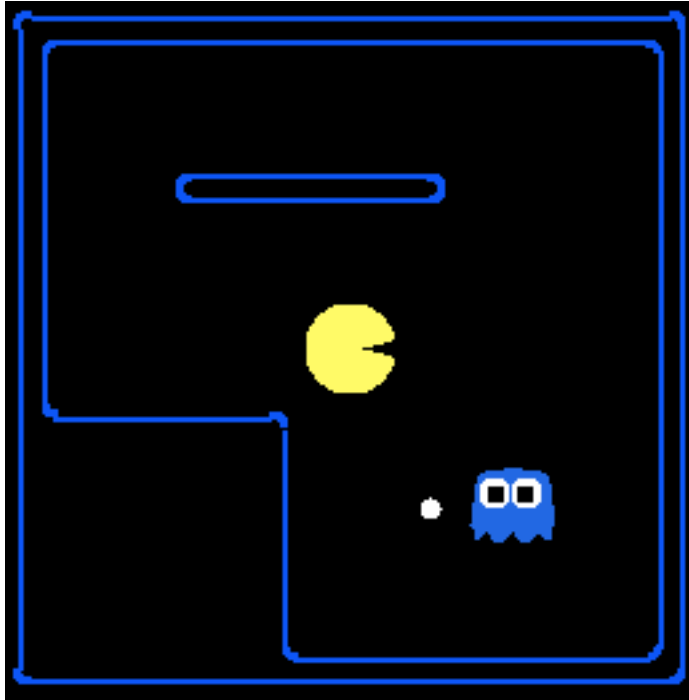


## Dangerous Pessimism

Assuming the worst case when it's not likely



# Assumptions vs. Reality

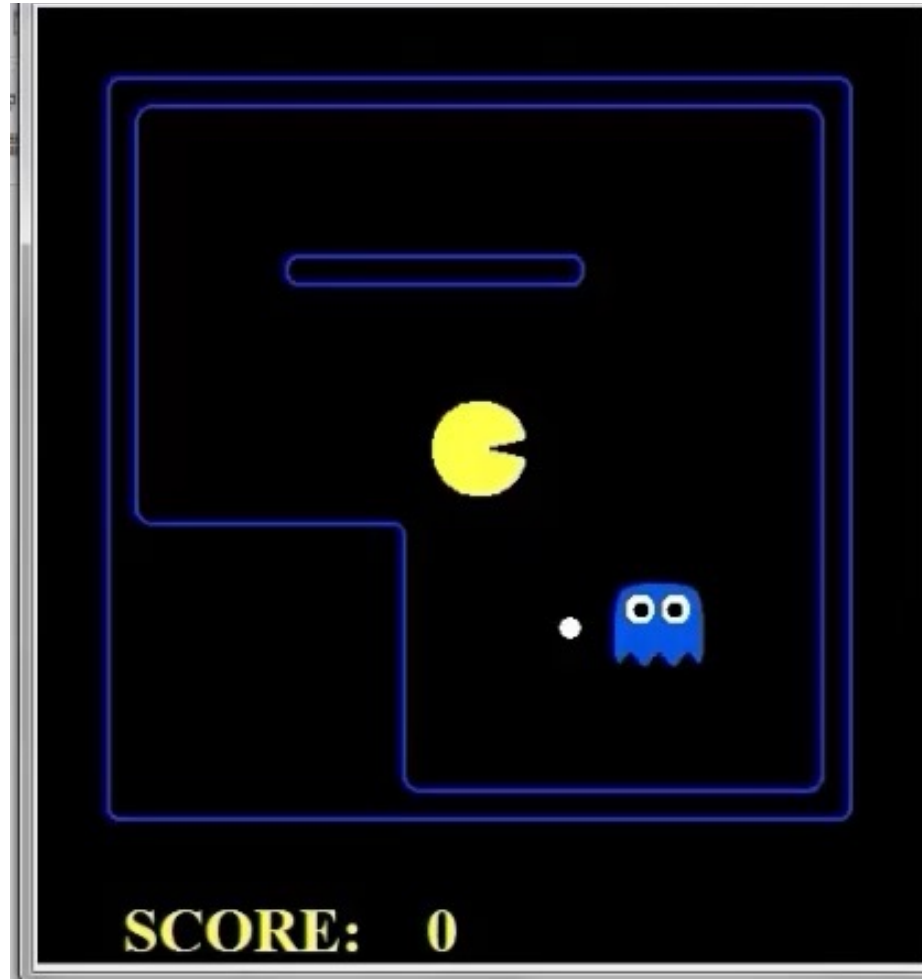


	Adversarial Ghost	Random Ghost
Minimax Pacman		
Expectimax Pacman		

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

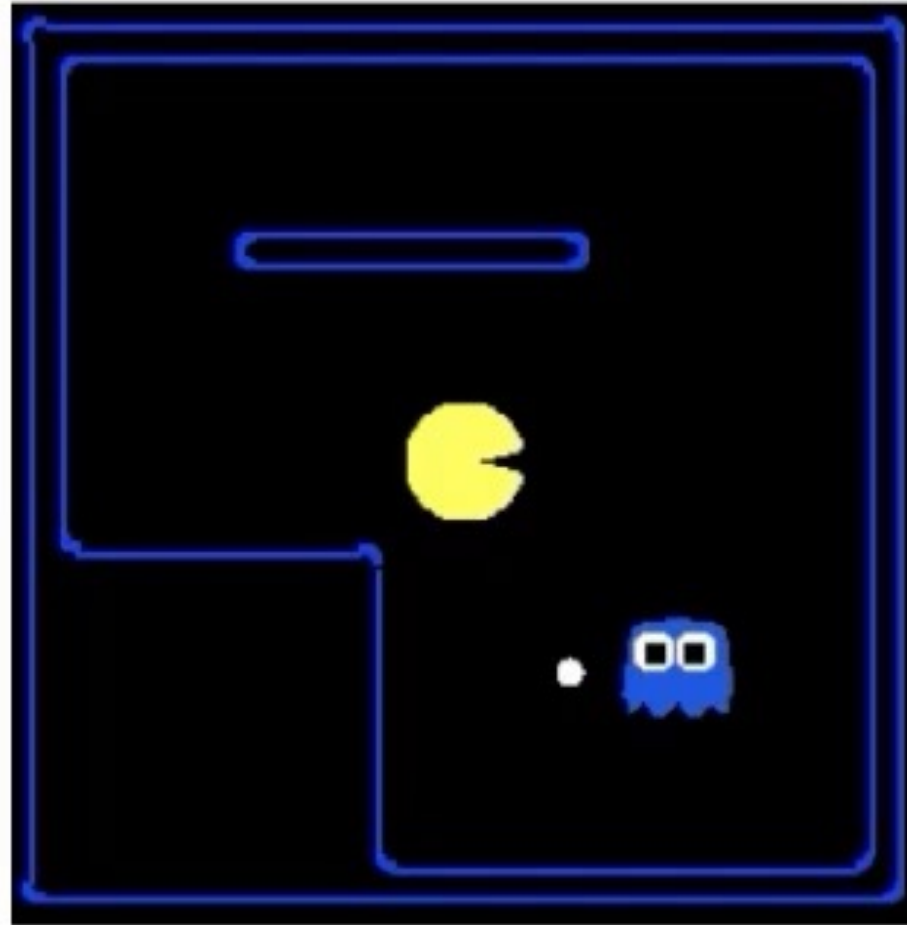


# Video of Demo World Assumptions Random Ghost – Minimax Pacman

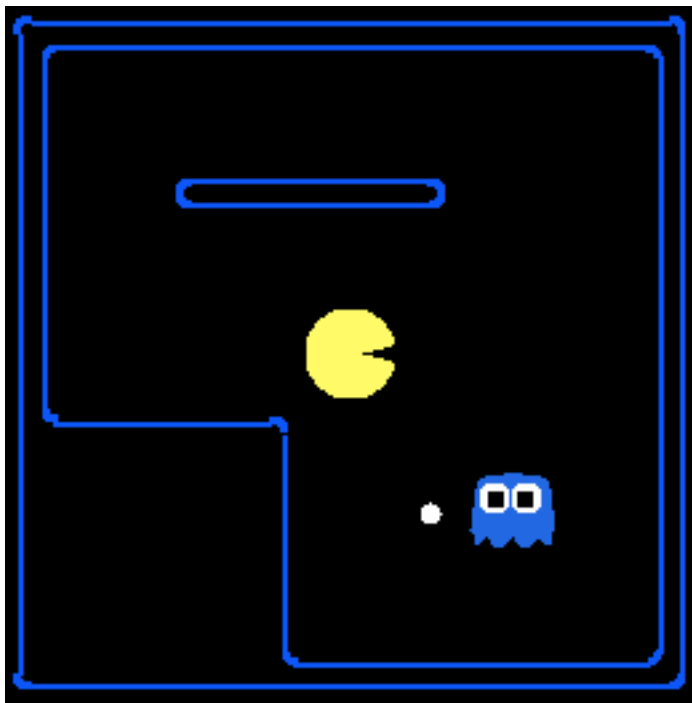


# Video of Demo World Assumptions Adversarial Ghost – Expectimax Pacman

---



# Assumptions vs. Reality



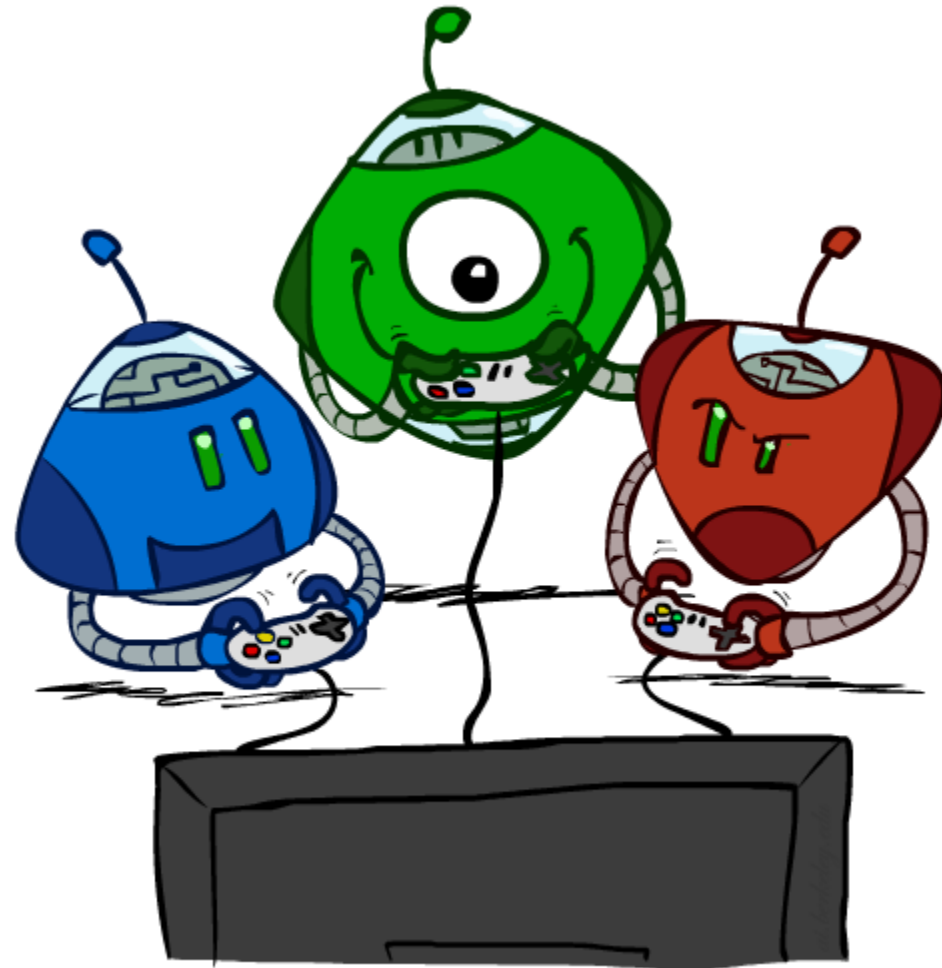
	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

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

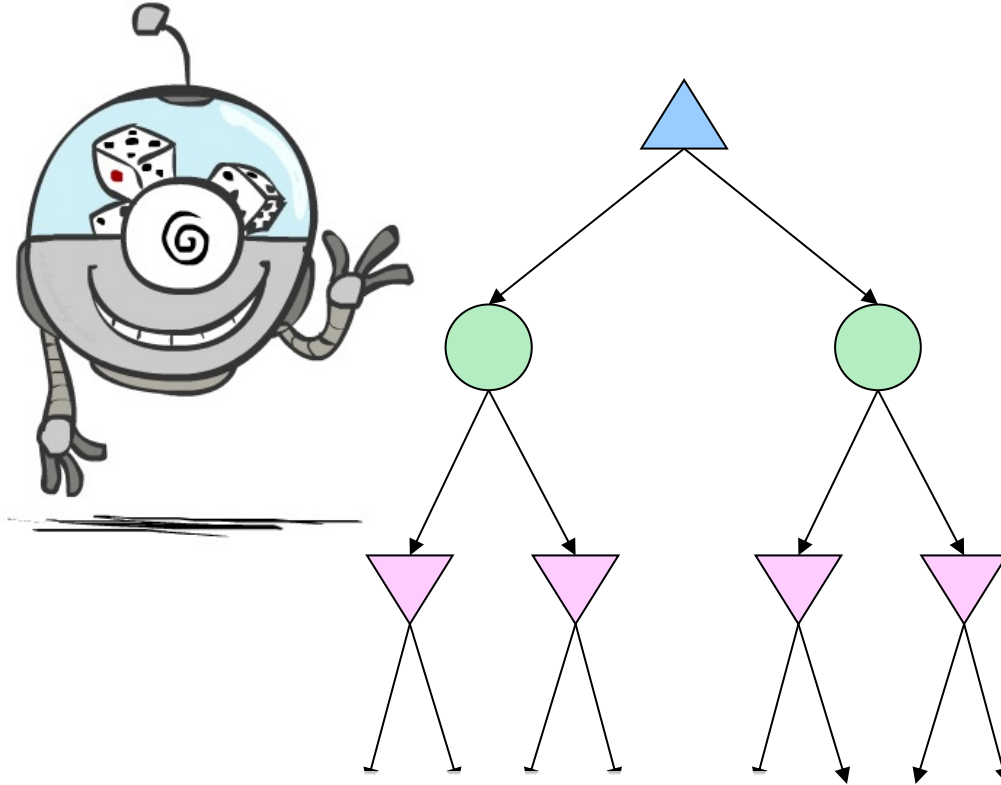
# Other Game Types

---



# 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**

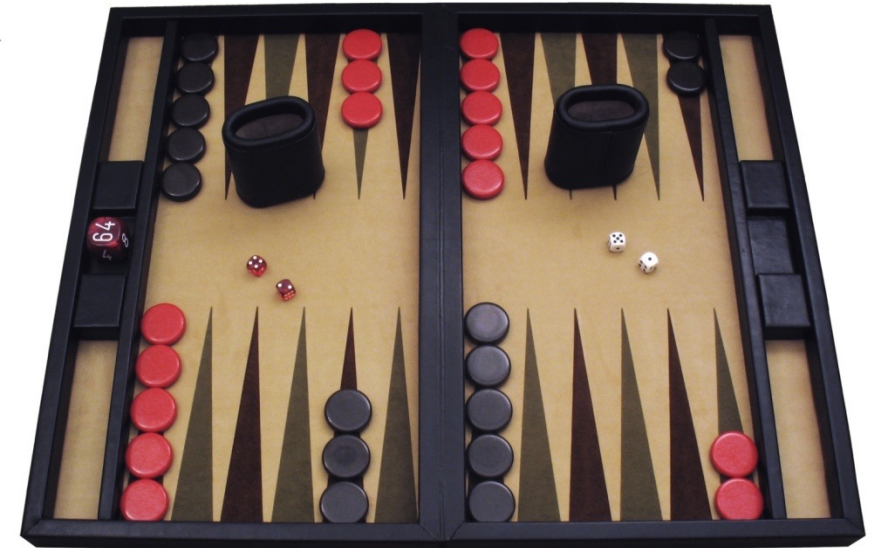
**return** the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

**if** *state* is a chance node **then**

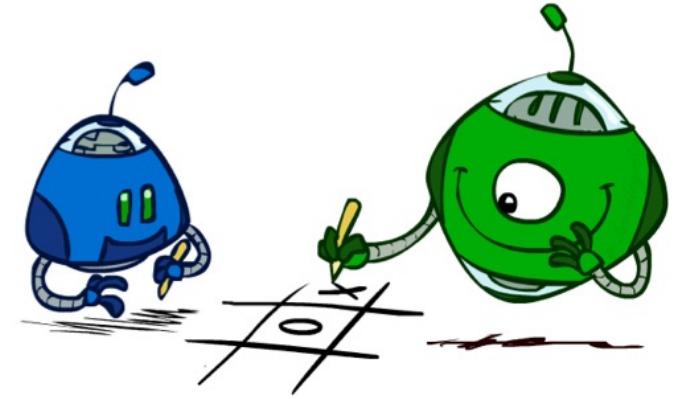
**return** average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

# Example: Backgammon

- Dice rolls increase  $b$ : 21 possible rolls with 2 dice
  - Backgammon  $\approx 20$  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
- 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...

