



南方科技大学

# STA303: Artificial Intelligence

## Markov Decision Processes 2

Fang Kong

<https://fangkongx.github.io/>

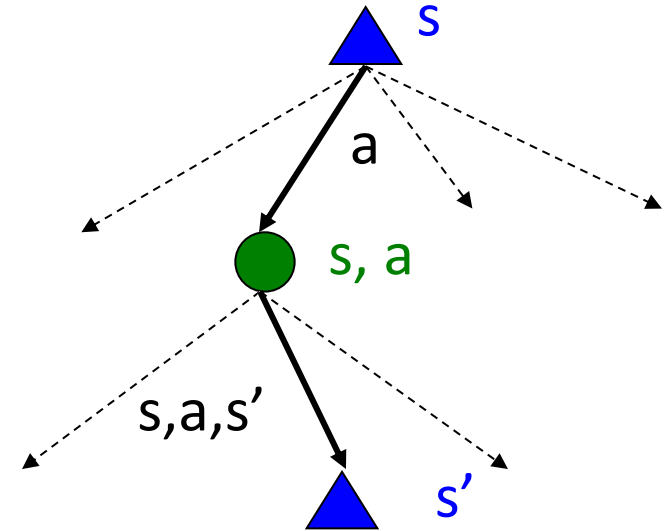
# Recap: MDPs

- Markov decision processes:

- States  $S$
- Actions  $A$
- Transitions  $P(s' | s, a)$  (or  $T(s, a, s')$ )
- Rewards  $R(s, a, s')$  (and discount  $\gamma$ )
- Start state  $s_0$

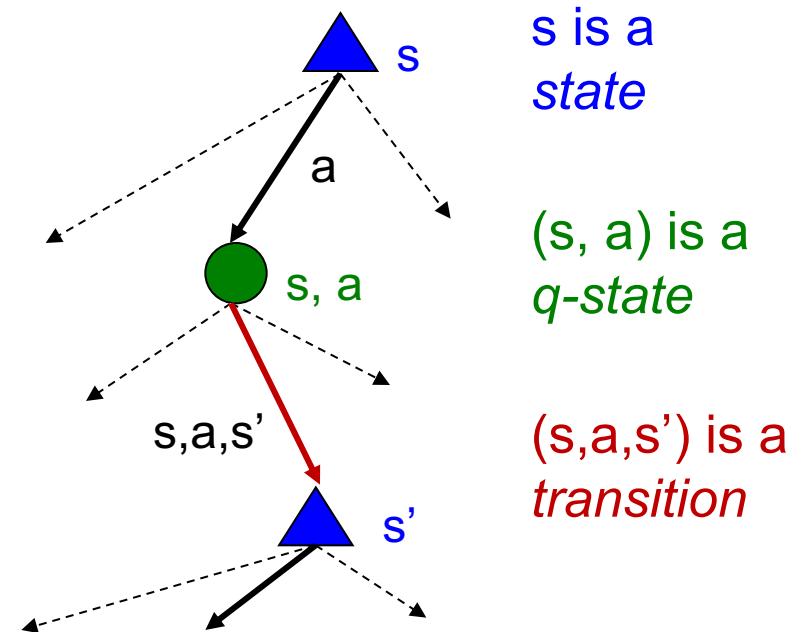
- Quantities:

- Policy = map of states to actions
- Utility = sum of discounted rewards
- Values = expected future utility from a state (max node)
- Q-Values = expected future utility from a q-state (chance node)

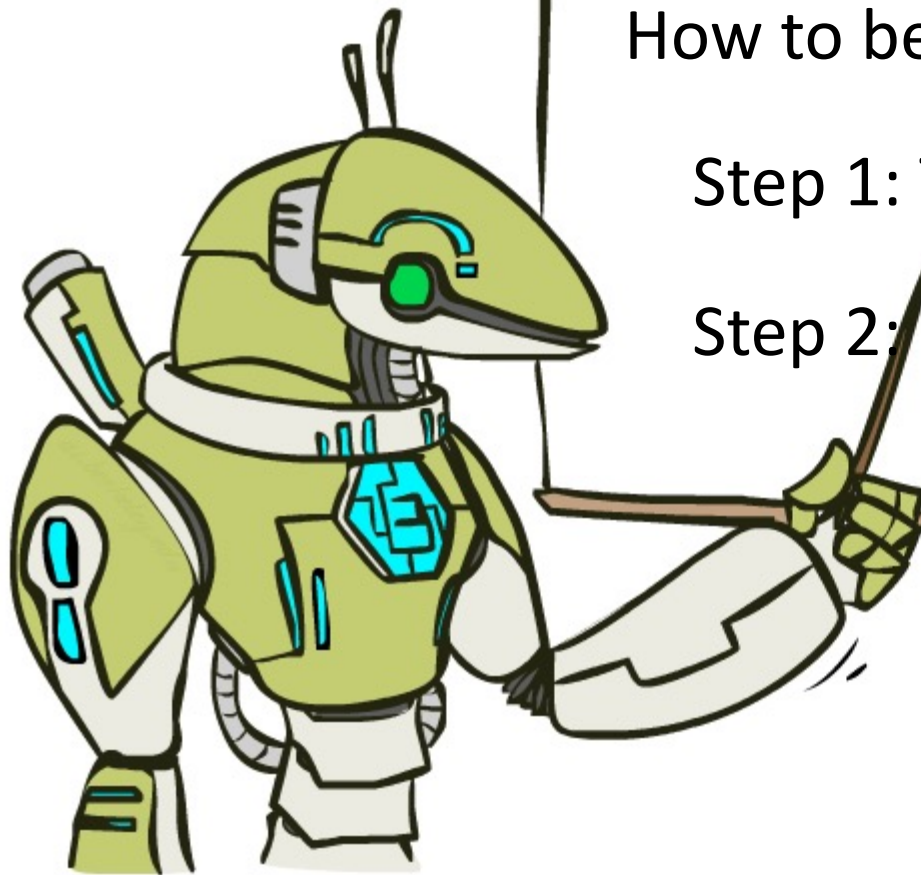


# Optimal Quantities

- The value (utility) of a state  $s$ :  
 $V^*(s)$  = expected utility starting in  $s$  and acting optimally
- The value (utility) of a q-state  $(s,a)$ :  
 $Q^*(s,a)$  = expected utility starting out having taken action  $a$  from state  $s$  and (thereafter) acting optimally
- The optimal policy:  
 $\pi^*(s)$  = optimal action from state  $s$



# The Bellman Equations



How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal

# The Bellman Equations

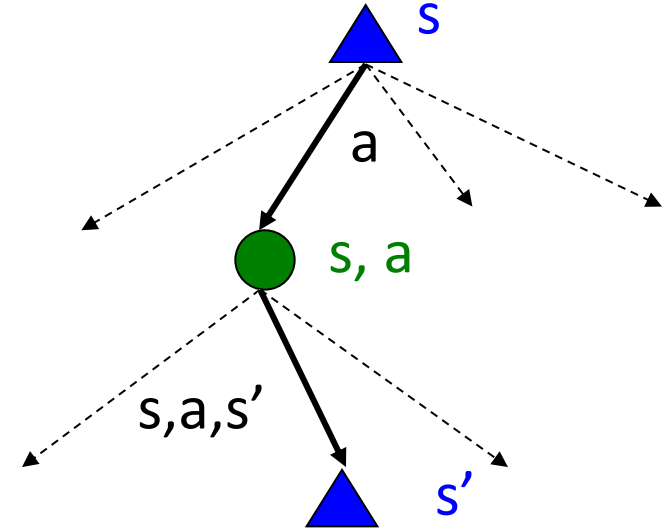
- Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

$$V^*(s) = \max_a Q^*(s, a)$$

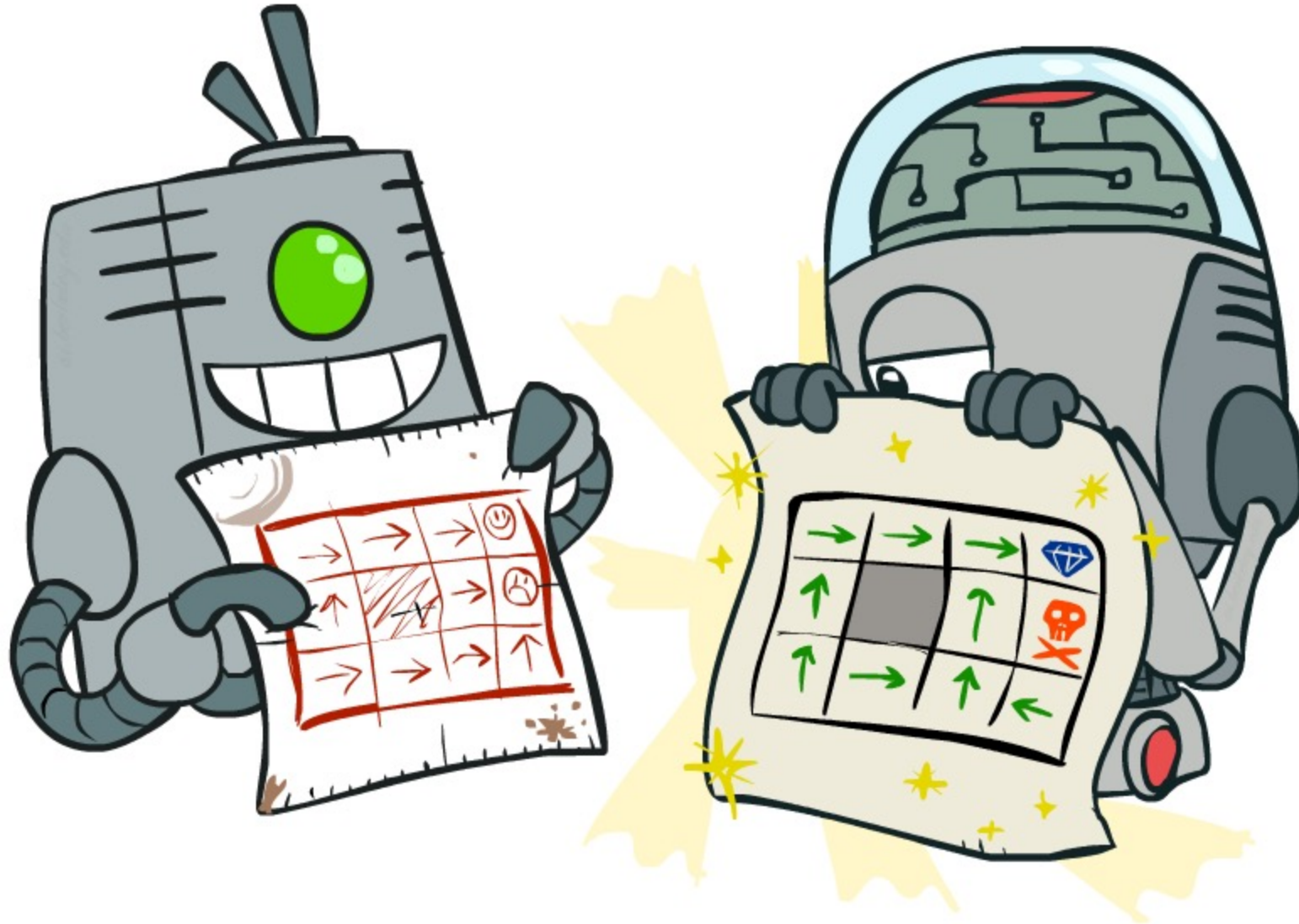
$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

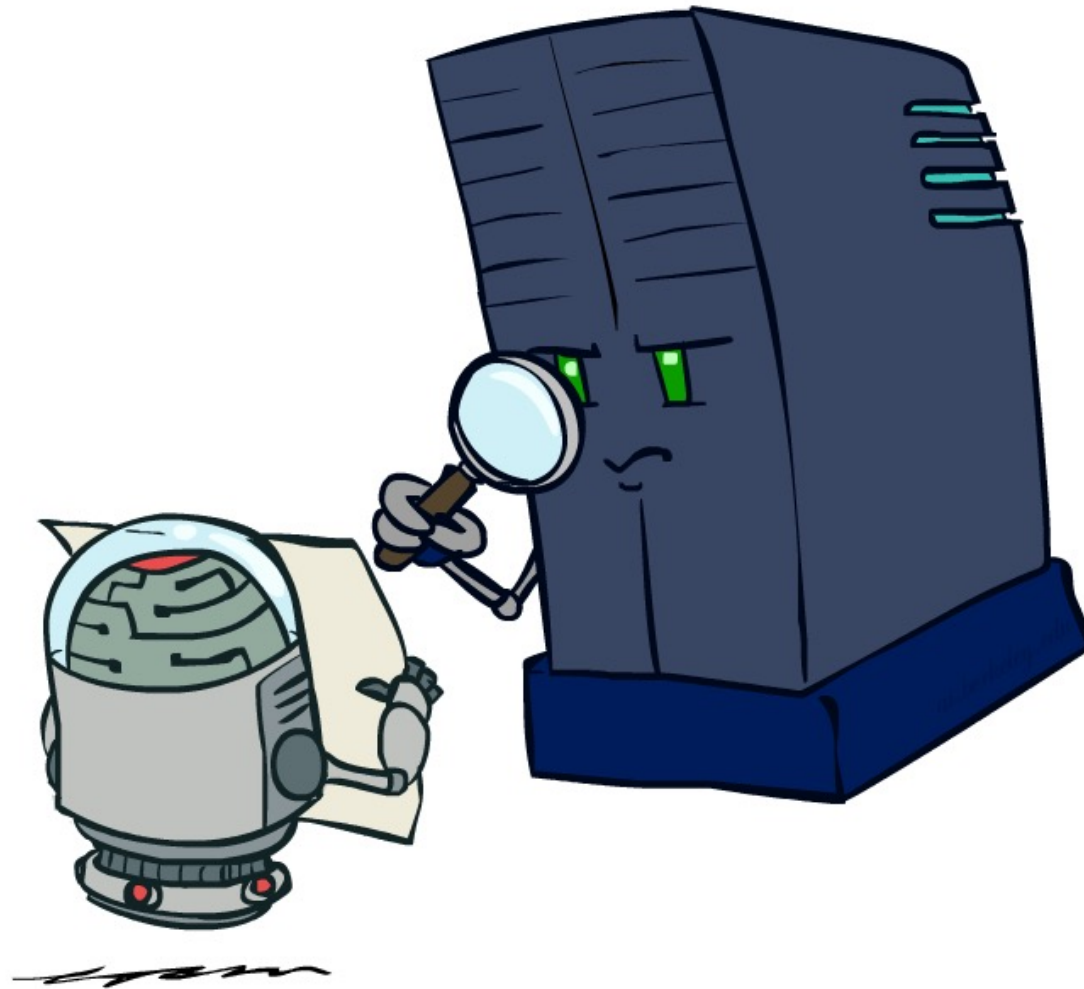


# Policy Methods



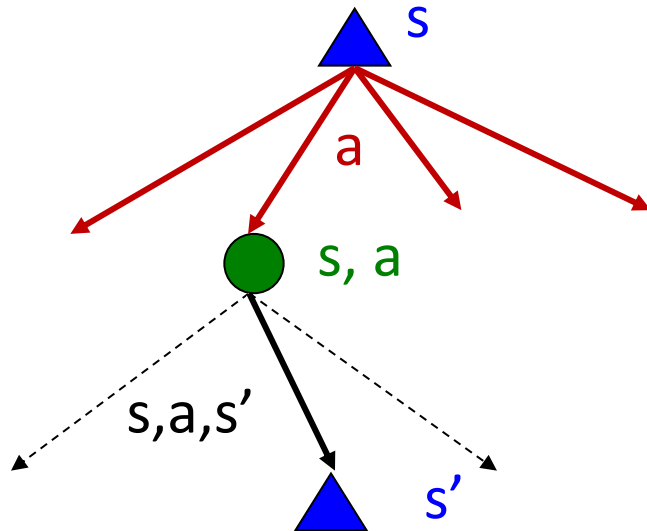
# Policy Evaluation

---

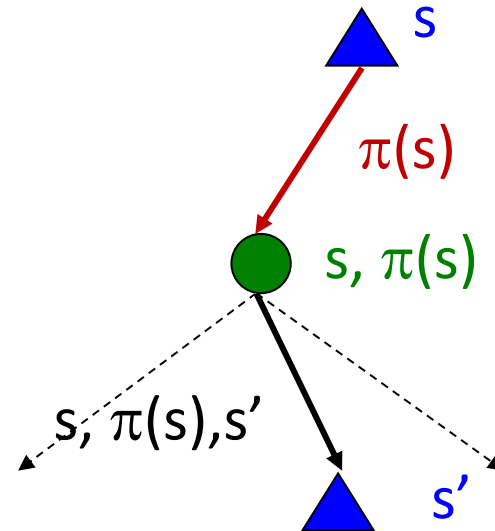


# Fixed Policies

Do the optimal action



Do what  $\pi$  says to do



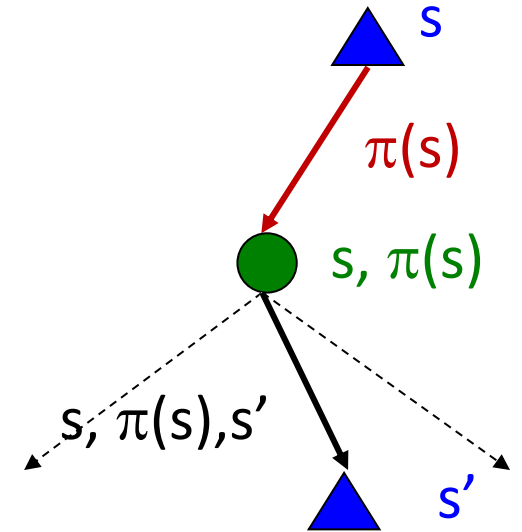
- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy  $\pi(s)$ , then the tree would be simpler – only one action per state
  - ... though the tree's value would depend on which policy we fixed



# Utilities for a Fixed Policy

- Another basic operation: compute the utility of a state  $s$  under a fixed (generally non-optimal) policy
- Define the utility of a state  $s$ , under a fixed policy  $\pi$ :  
 $V^\pi(s)$  = expected total discounted rewards starting in  $s$  and following  $\pi$
- Recursive relation (one-step look-ahead / Bellman equation):

$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

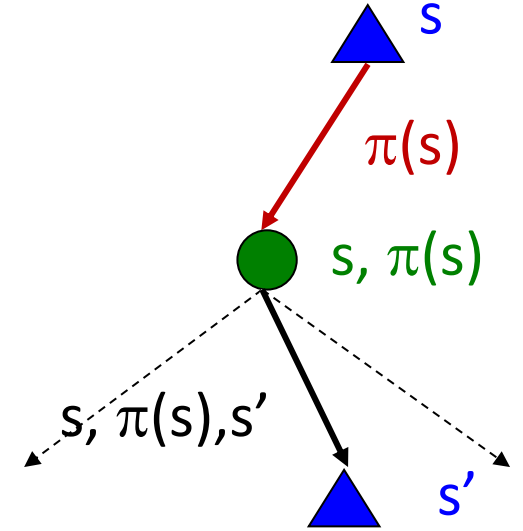


# Policy Evaluation

- How do we calculate the  $V$ 's for a fixed policy  $\pi$ ?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = 0$$

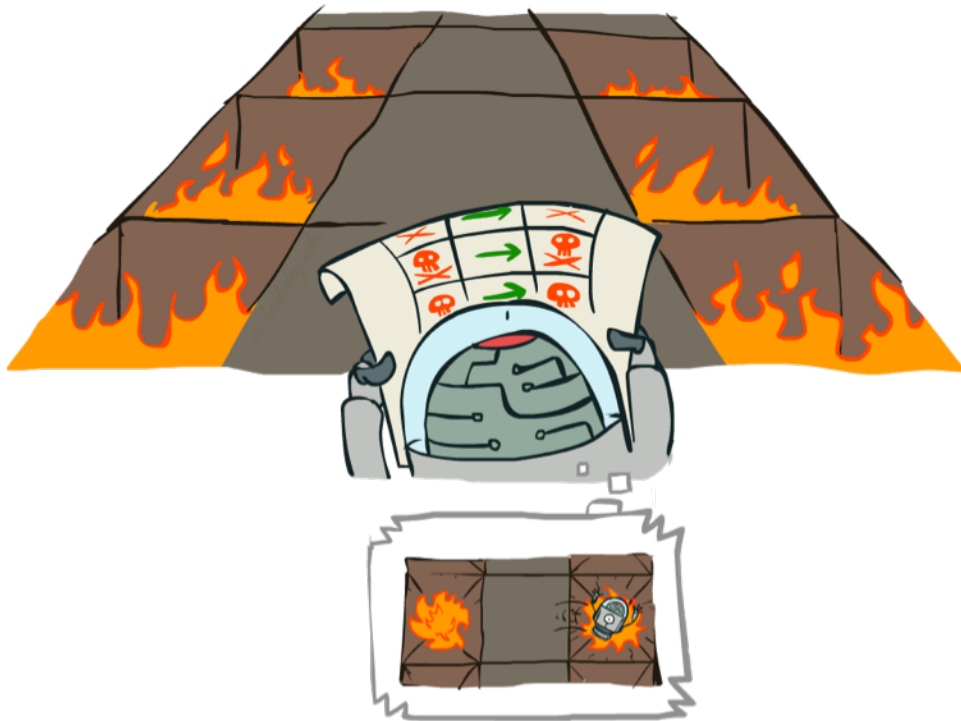
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$



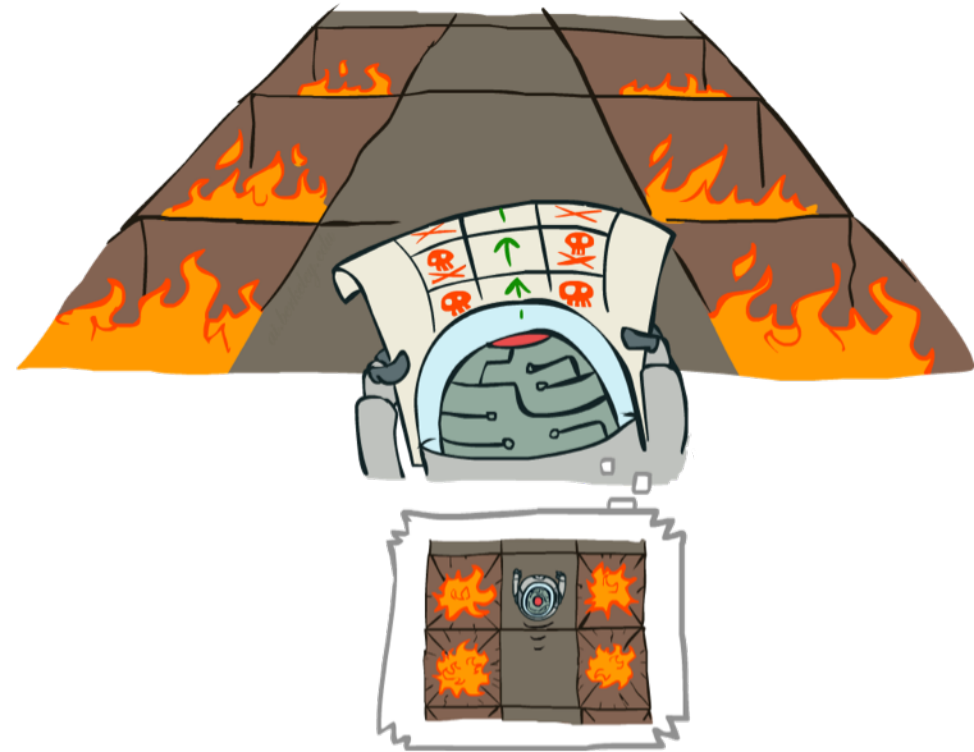
- Efficiency:  $O(S^2)$  per iteration
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

# Example: Policy Evaluation

Always Go Right



Always Go Forward



# Example: Policy Evaluation

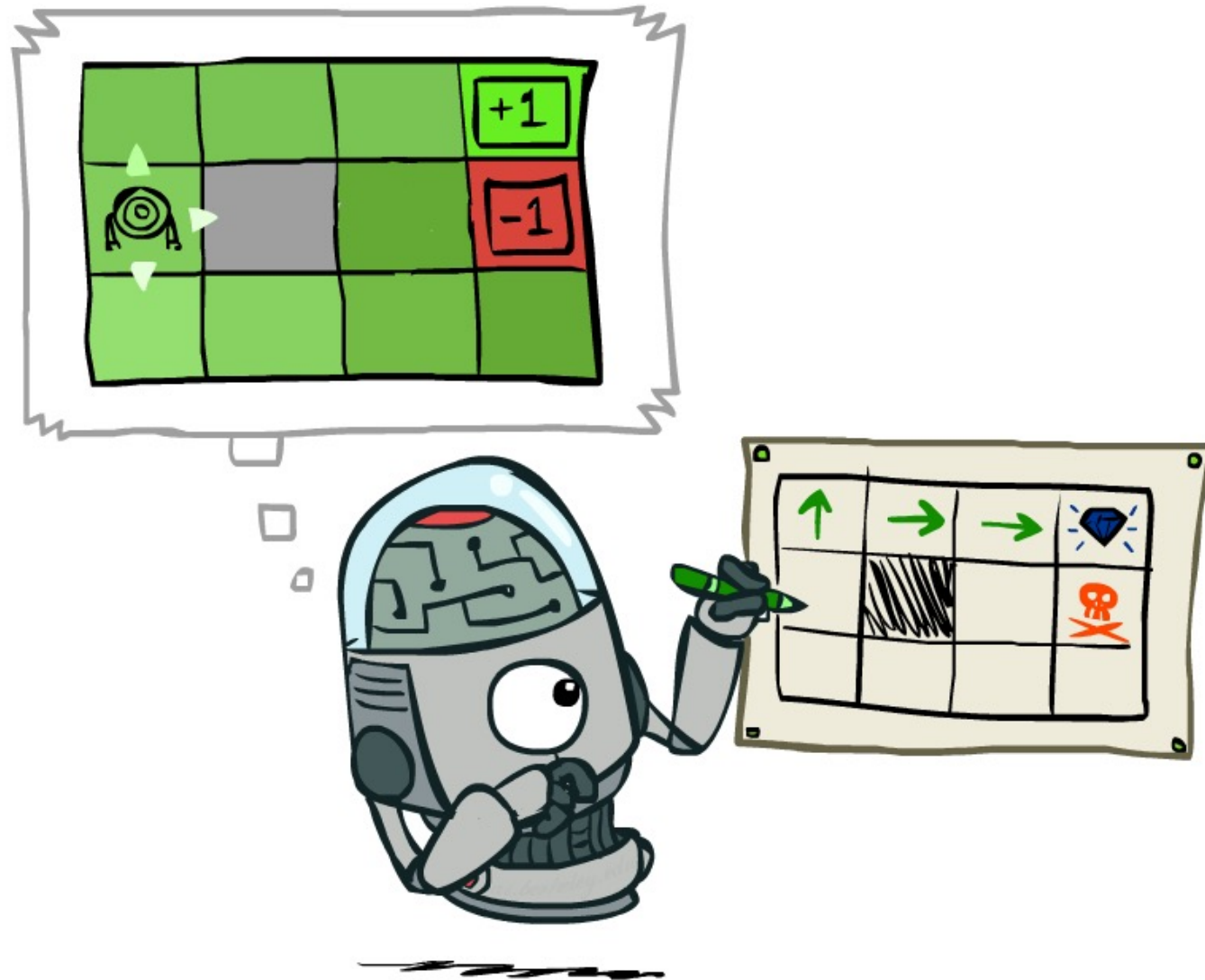
Always Go Right



Always Go Forward



# Policy Extraction



# Computing Actions from Values

- Let's imagine we have the optimal values  $V^*(s)$
- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)



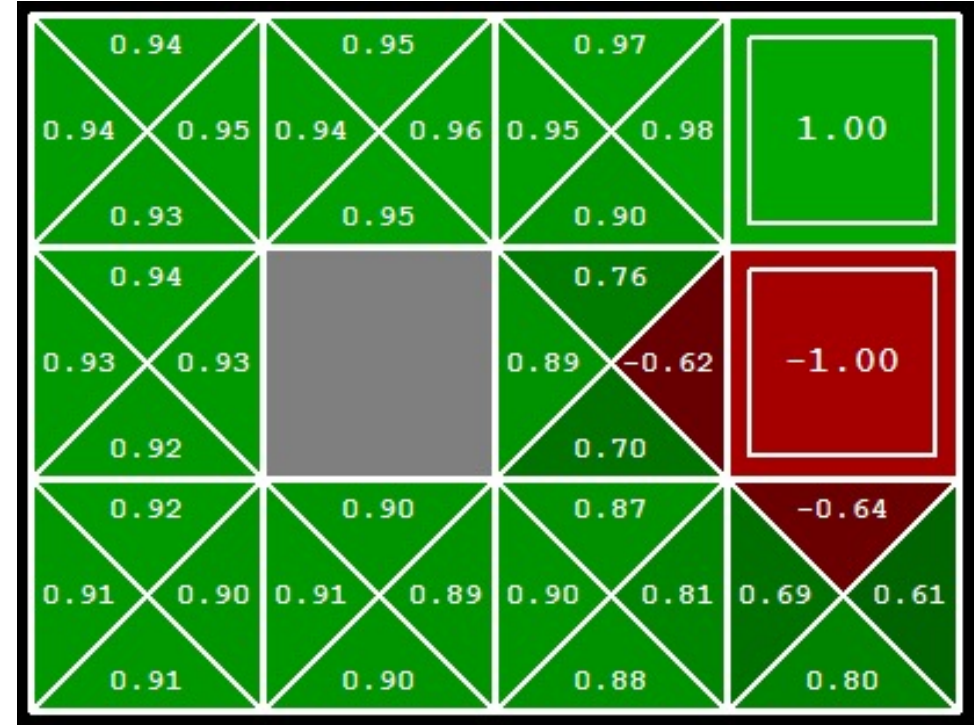
$$\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

- This is called **policy extraction**, since it gets the policy implied by the values

# Computing Actions from Q-Values

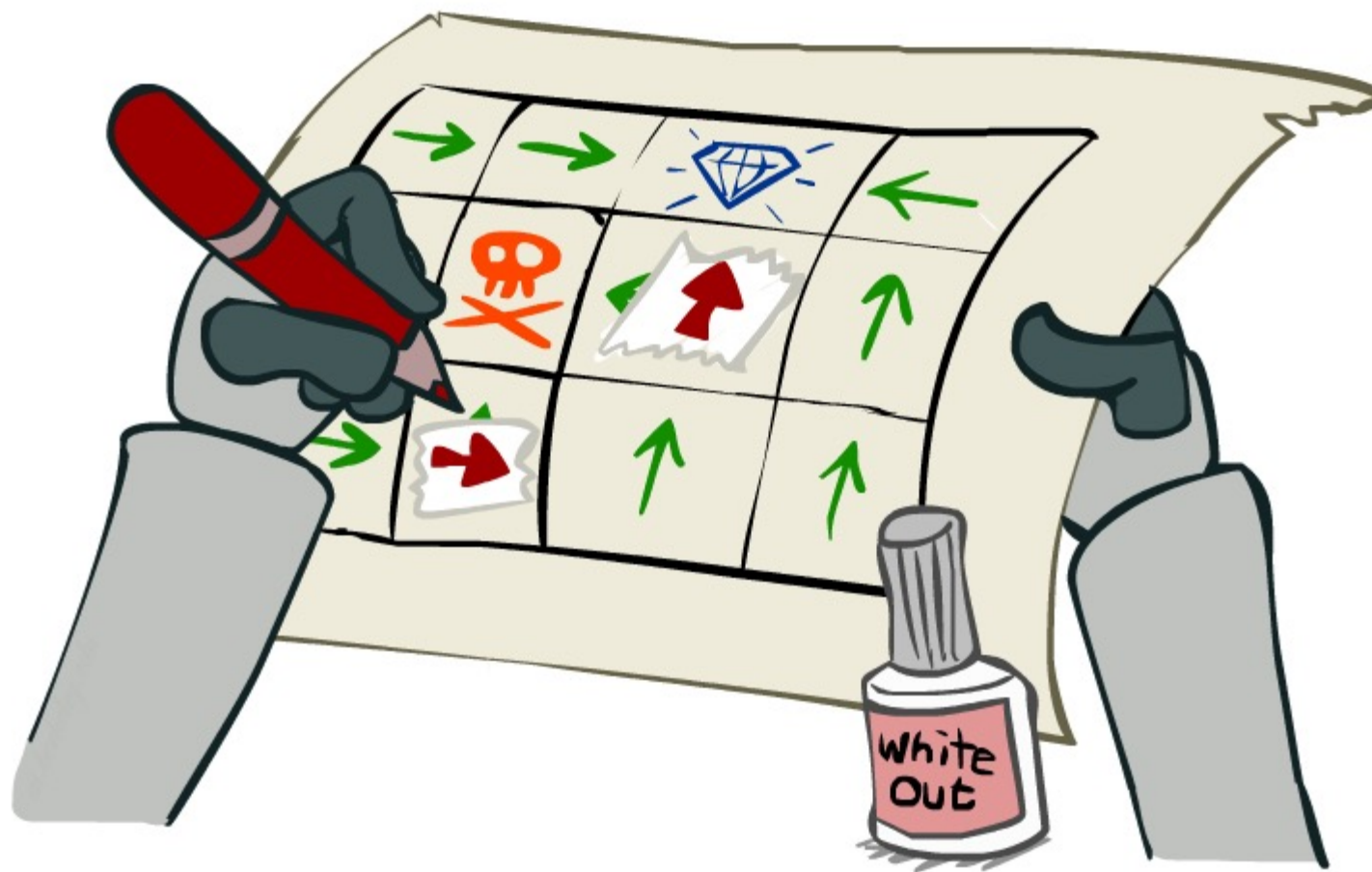
- Let's imagine we have the optimal q-values:
- How should we act?
  - Completely trivial to decide!

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$



- Important lesson: actions are easier to select from q-values than values!

# Policy Iteration



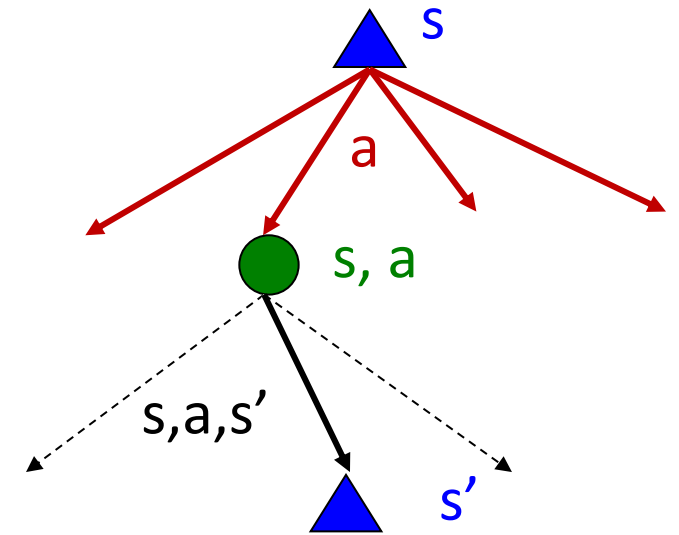


# Problems with Value Iteration

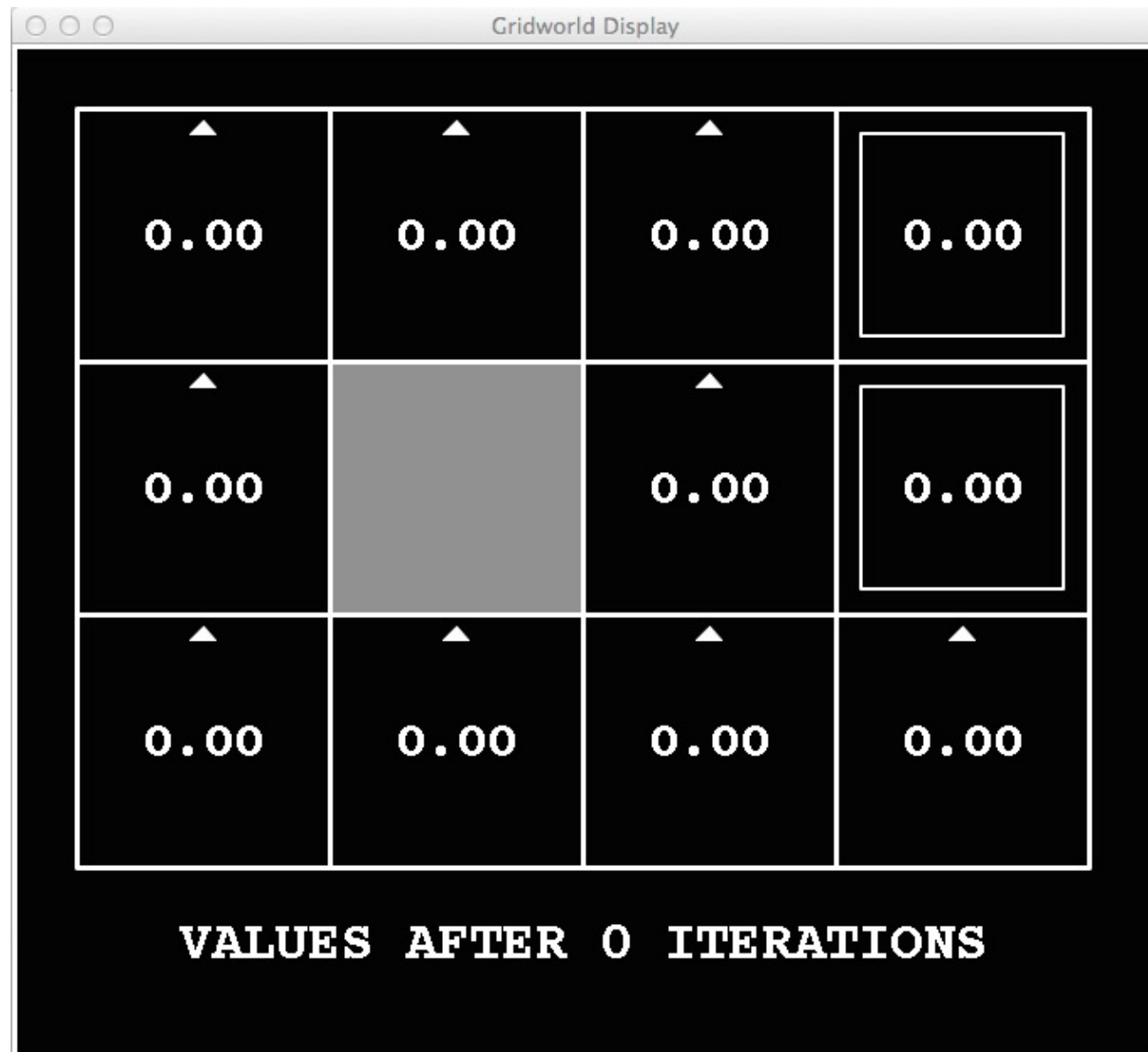
- Value iteration repeats the Bellman updates:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- Problem 1: It's slow –  $O(S^2A)$  per iteration
- Problem 2: The “arg max” at each state rarely changes
- Problem 3: The policy often converges long before the values



# k=0



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# k=1



Noise = 0.2  
Discount = 0.9  
Living reward = 0

$k=2$



Noise = 0.2  
Discount = 0.9  
Living reward = 0

$k=3$



Noise = 0.2  
Discount = 0.9  
Living reward = 0

$k=4$



Noise = 0.2  
Discount = 0.9  
Living reward = 0

k=5



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# k=6



Noise = 0.2  
Discount = 0.9  
Living reward = 0



$k=7$



Noise = 0.2  
Discount = 0.9  
Living reward = 0

k=8



Noise = 0.2  
Discount = 0.9  
Living reward = 0

k=9



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# k=10



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# k=11



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# k=12



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# k=100



Noise = 0.2  
Discount = 0.9  
Living reward = 0

# Policy Iteration

---

- Alternative approach for optimal values:
  - **Step 1: Policy evaluation:** calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - **Step 2: Policy improvement:** update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges
- This is **policy iteration**
  - It's still optimal!
  - Can converge (much) faster under some conditions



# Policy Iteration (PI)

- Evaluation: For fixed current policy  $\pi$ , find values with policy evaluation:
  - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') [R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')]$$

# Convergence of PI

---

- 1. Improvement: Does each policy improvement step produce a better policy?
- 2. Convergence: Does PI converge to an optimal policy?

# Comparison

---

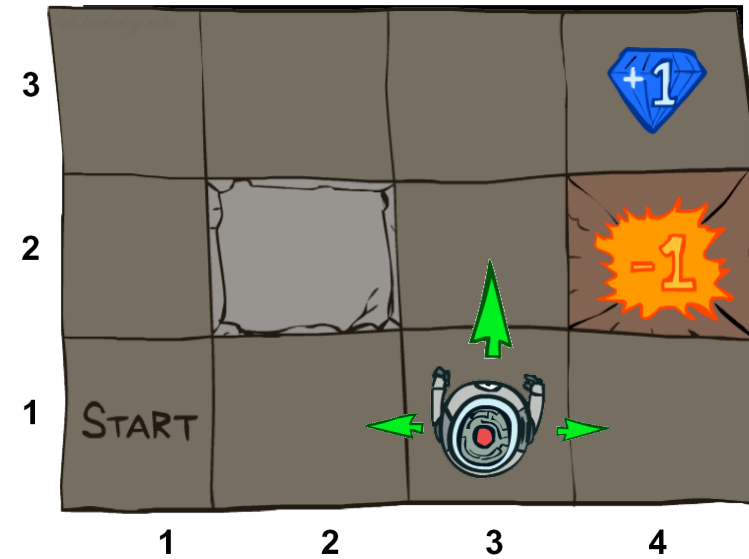
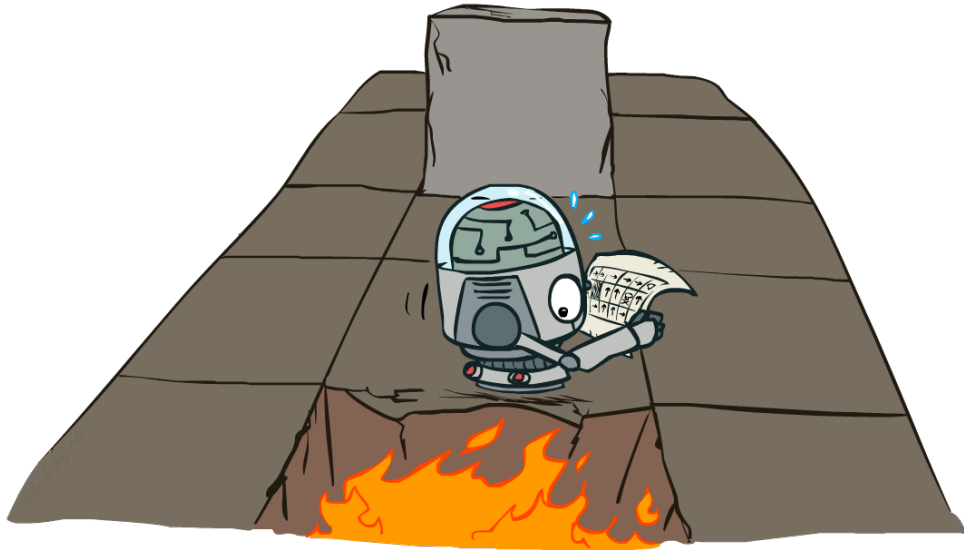
- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we're done)
- Both are dynamic programs for solving MDPs

# Recap: MDP Algorithms

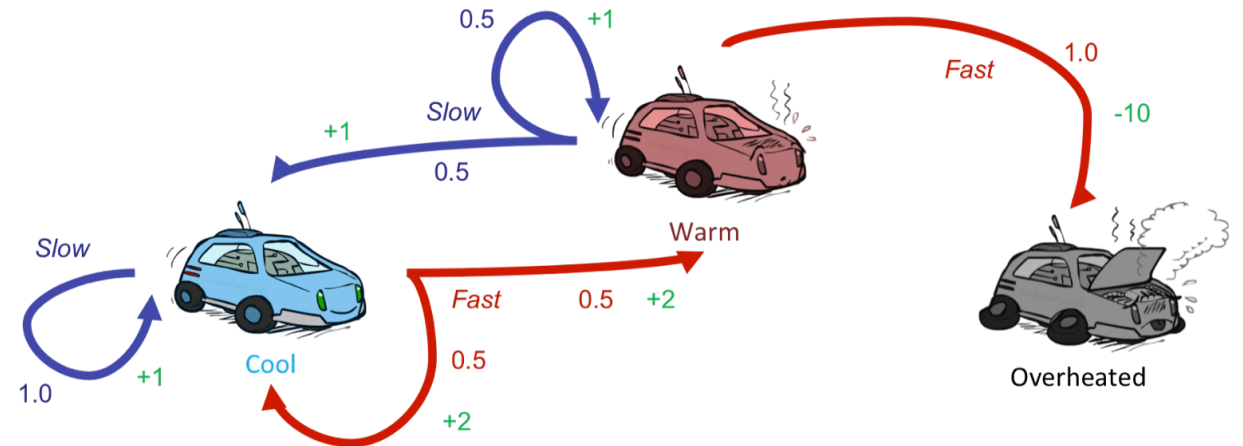
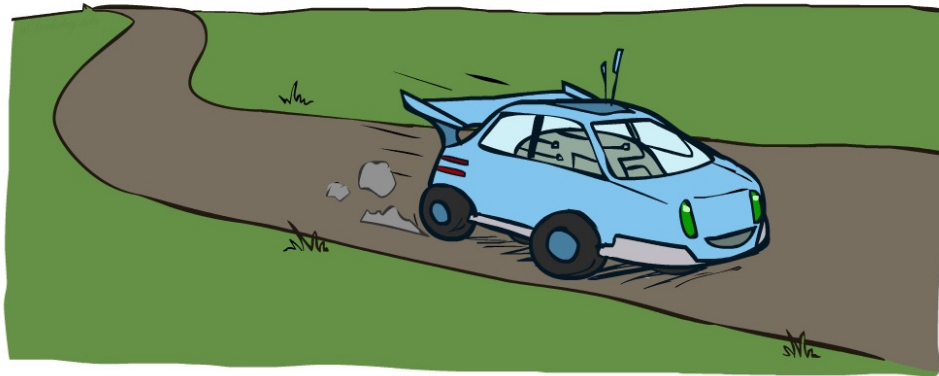
---

- So you want to....
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)
- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

# Planning Requires a Model!

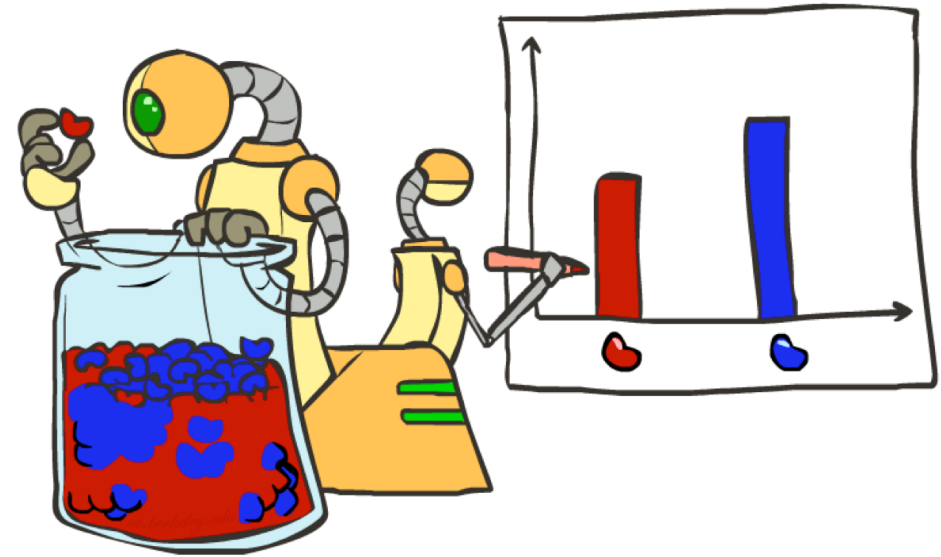


# Planning Requires a Model!



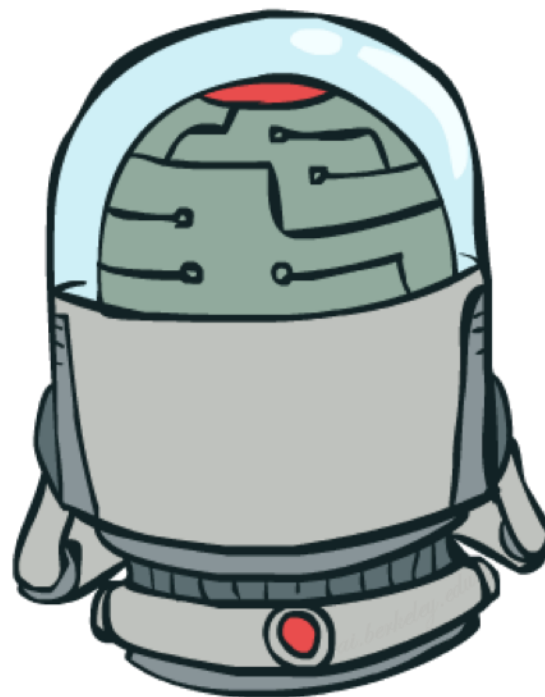
# Planning vs. Learning

- Markov decision processes:
  - States  $S$
  - Actions  $A$
  - Transitions  $T(s,a,s') = P(s' | s,a)$
  - Rewards  $R(s,a,s')$
  - Discount  $\gamma$
  - Start state  $s_0$



How can we learn these quantities?

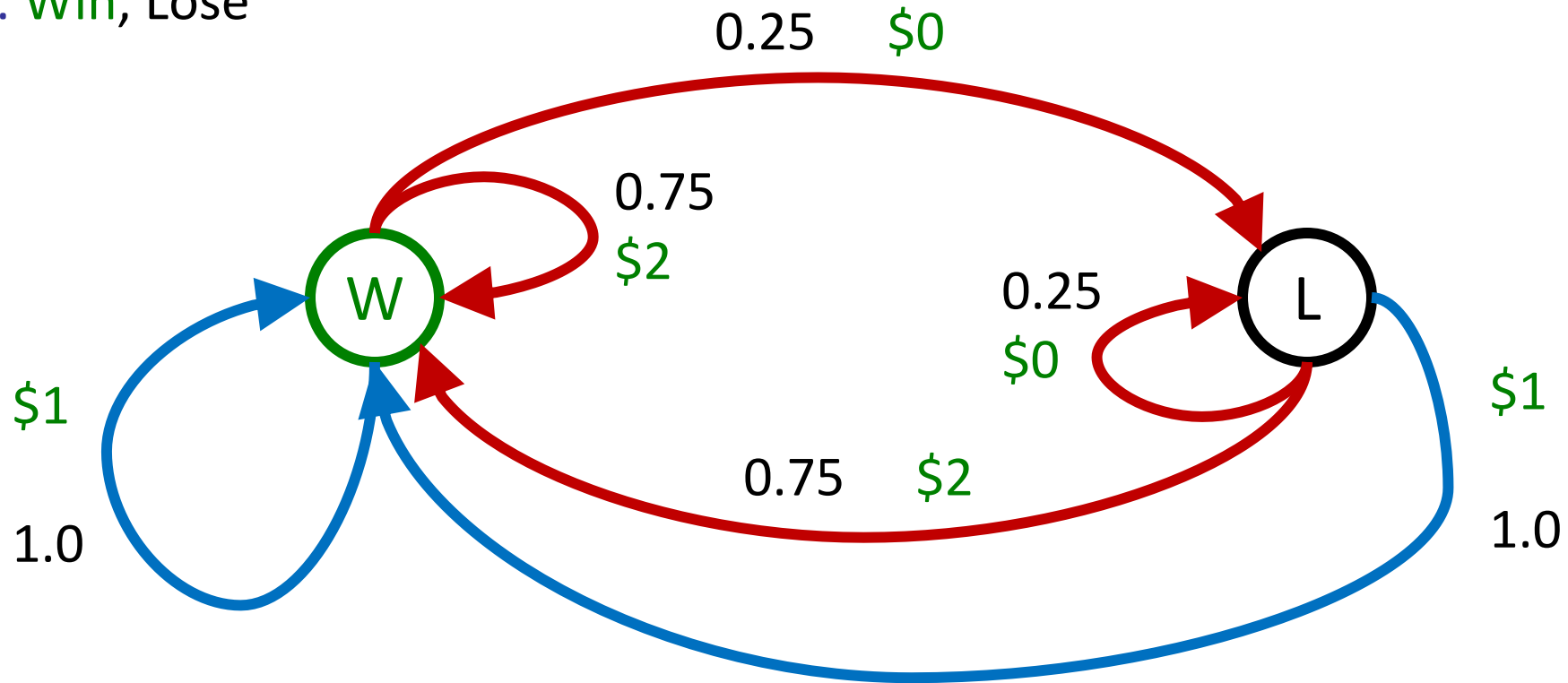
# One-Arm Bandits





# Double-Bandit MDP

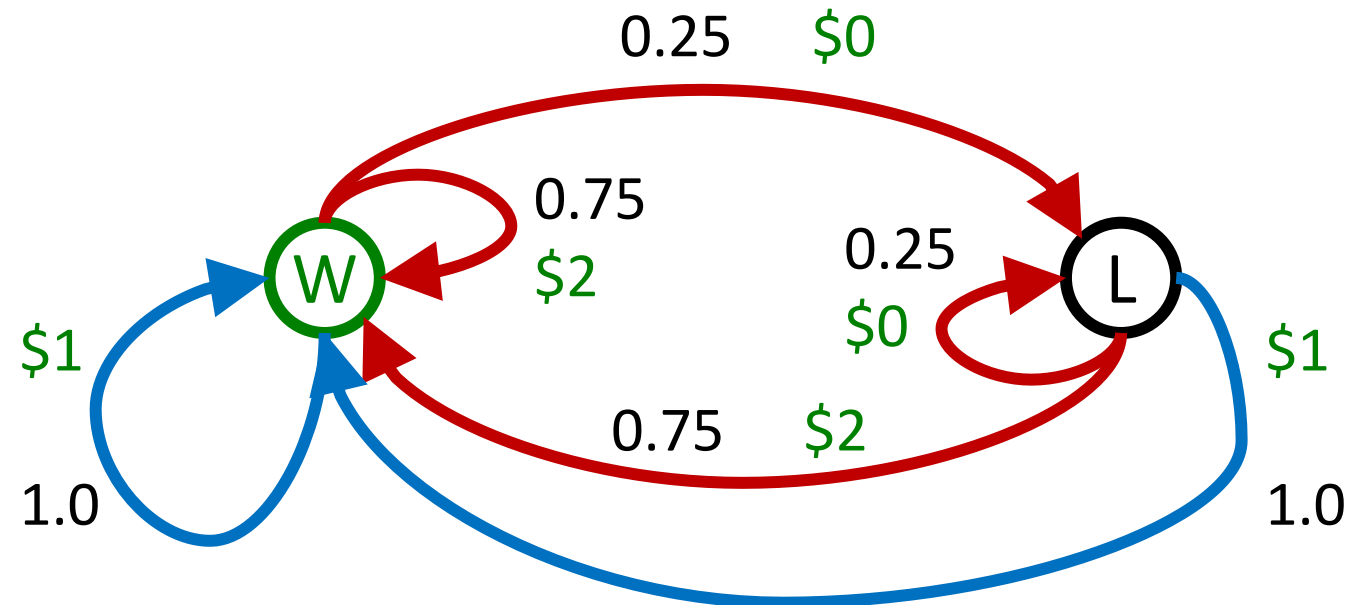
- Actions: *Blue*, *Red*
- States: *Win*, Lose



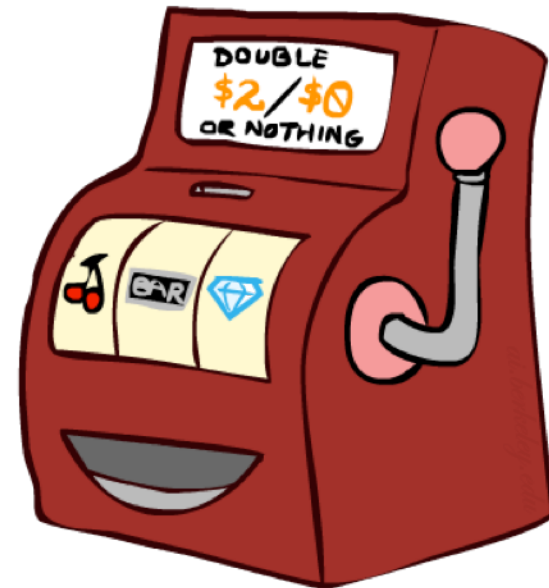
# MDP Planning

- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

	Value
Play Red	150
Play Blue	100



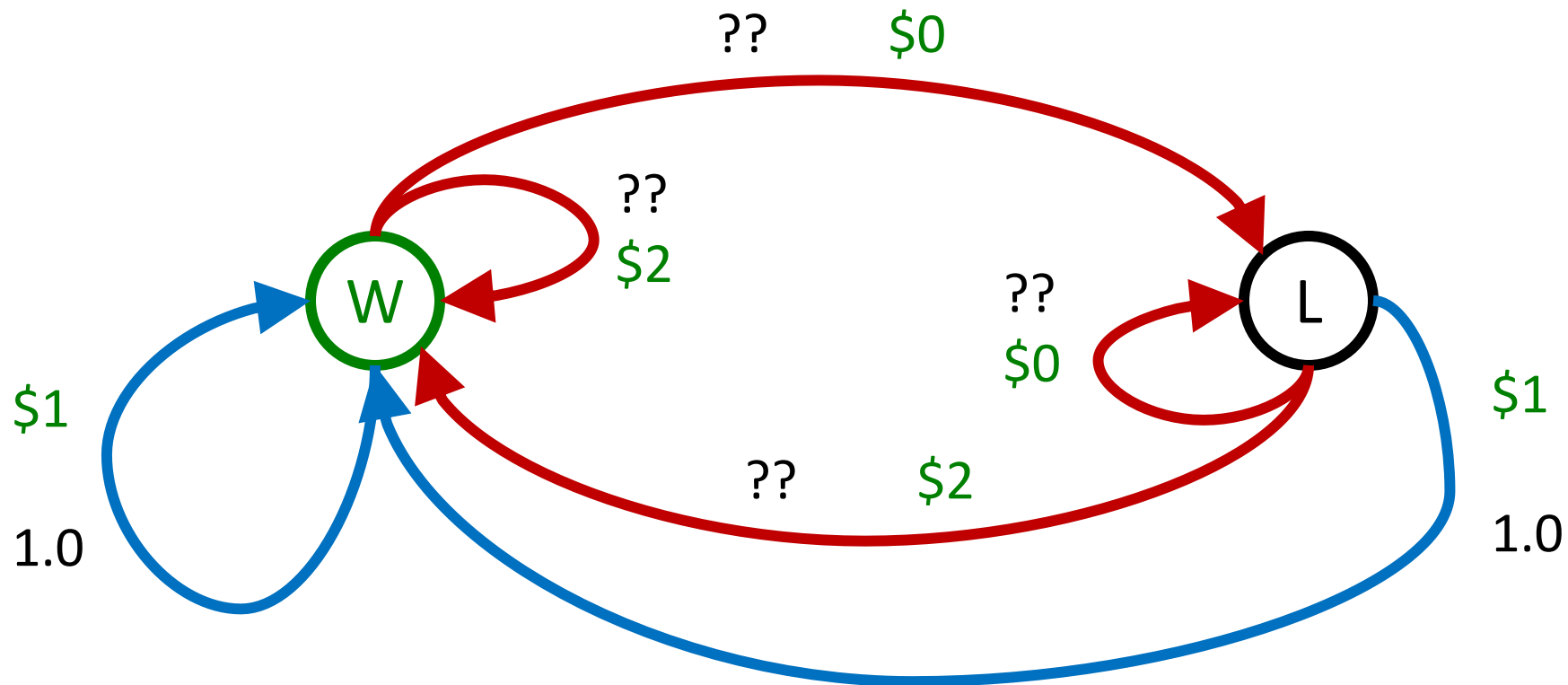
# Let's Play!



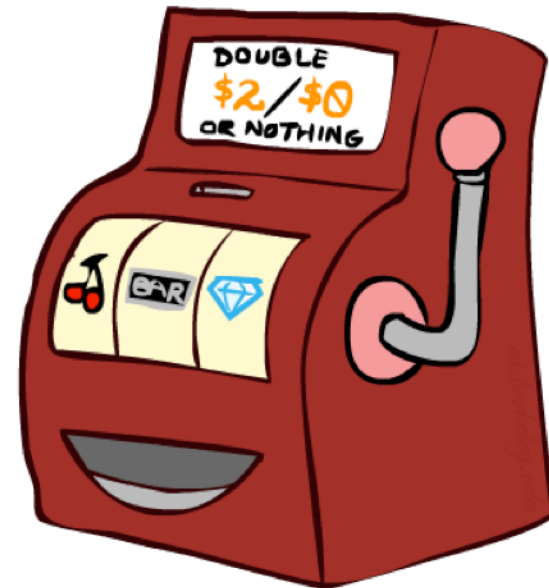
\$2 \$2 \$0 \$2 \$2  
\$2 \$2 \$0 \$0 \$0

# Planning With an Unknown Model

- Rules changed! Red's win chance is different.



# Let's Play!



\$0 \$0 \$0 \$2 \$0  
\$2 \$0 \$0 \$0 \$0

# What Just Happened?

- That wasn't planning, it was learning!
  - Technically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out
- Important ideas in machine learning that came up
  - Sampling: because of chance, you have to try things repeatedly
  - Parameter estimation: what is the most likely explanation of the data?
  - More data → better estimates

