

# CS 5522: Artificial Intelligence II

## Reinforcement Learning



Instructor: Wei Xu

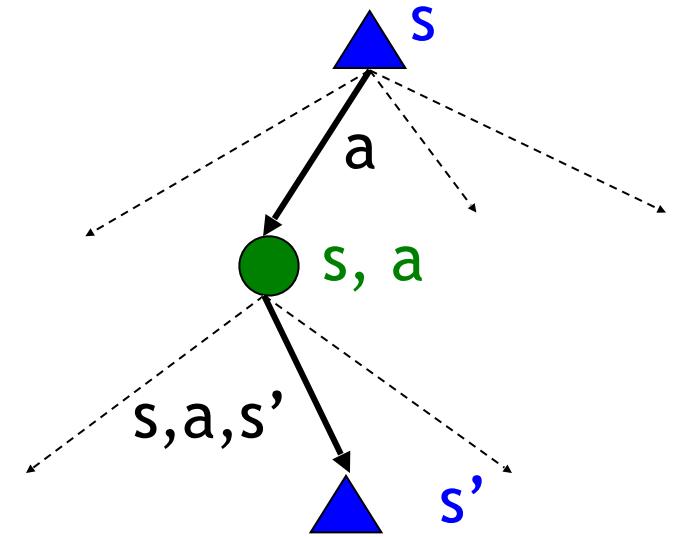
Ohio State University

[These slides were adapted from CS188 Intro to AI at UC Berkeley.]

# 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  $s_0$

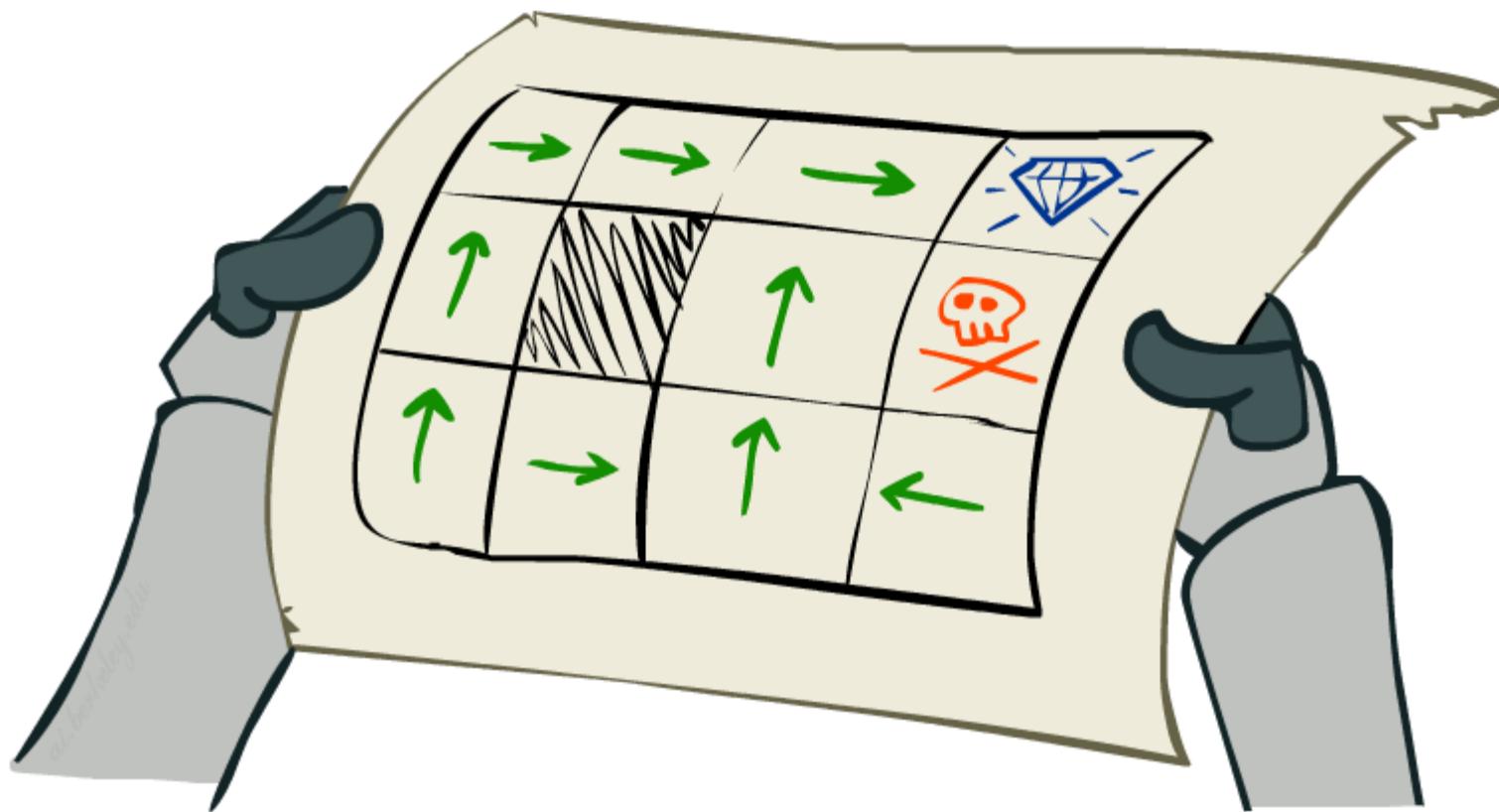


- **Quantities:**

- 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)

# Solving MDPs

---



# The Bellman Equations

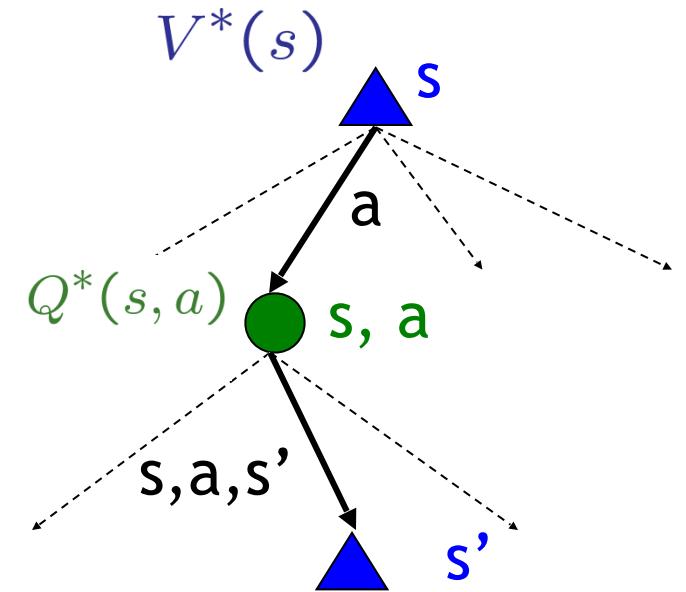
- Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

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

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

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

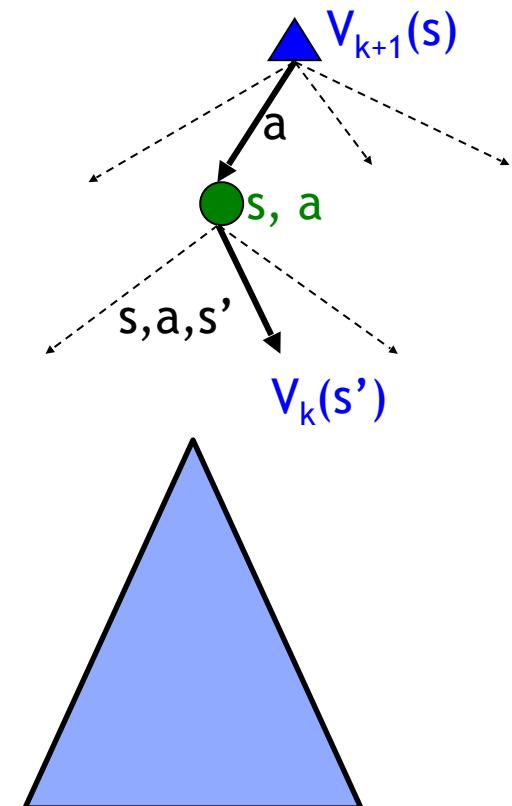


# Value Iteration

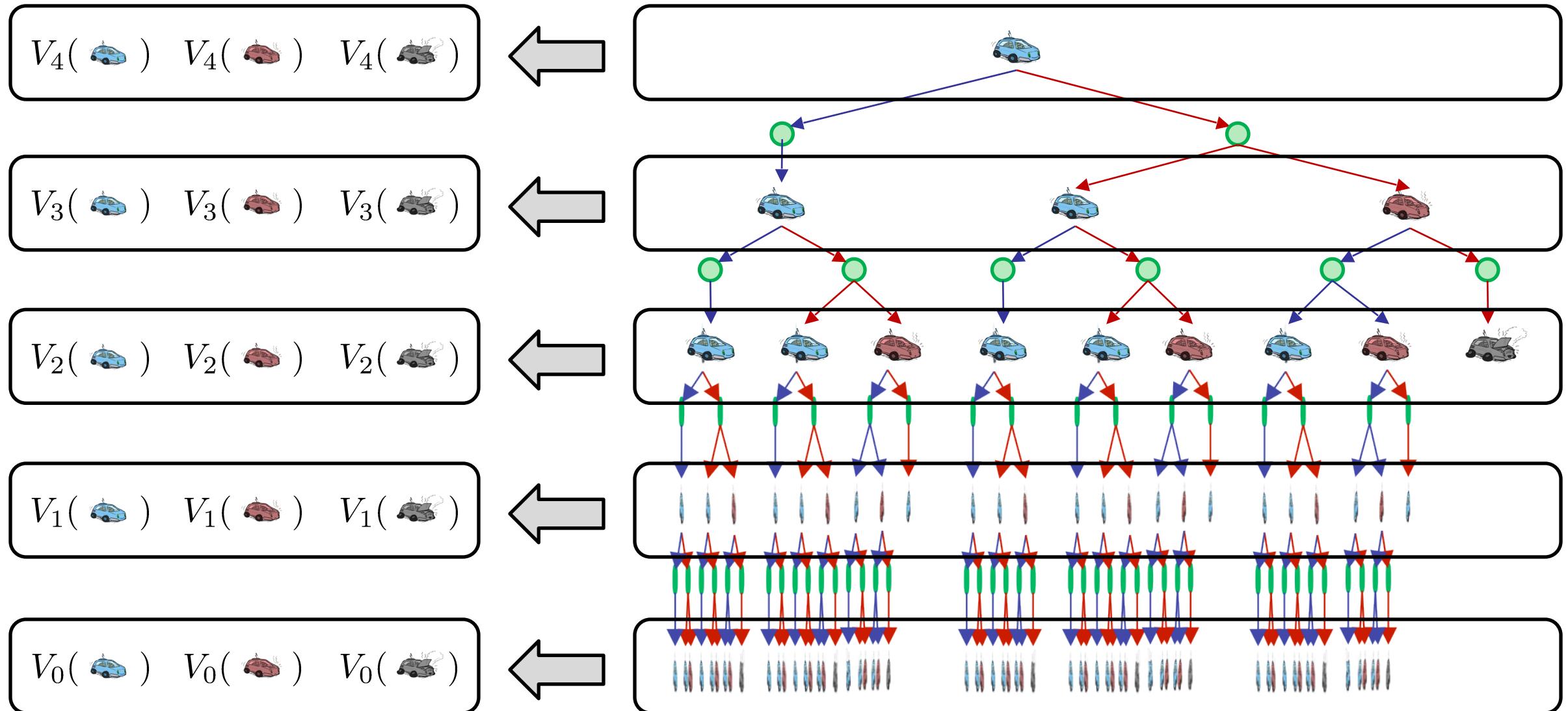
- Start with  $V_0(s) = 0$ : no time steps left means an expected reward sum of zero
- Given vector of  $V_k(s)$  values, do one ply of expectimax from each state:

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

- Repeat until convergence
- Complexity of each iteration:  $O(S^2A)$
- **Theorem: will converge to unique optimal values**
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do



# Computing Time-Limited Values



# Policy Iteration

---

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

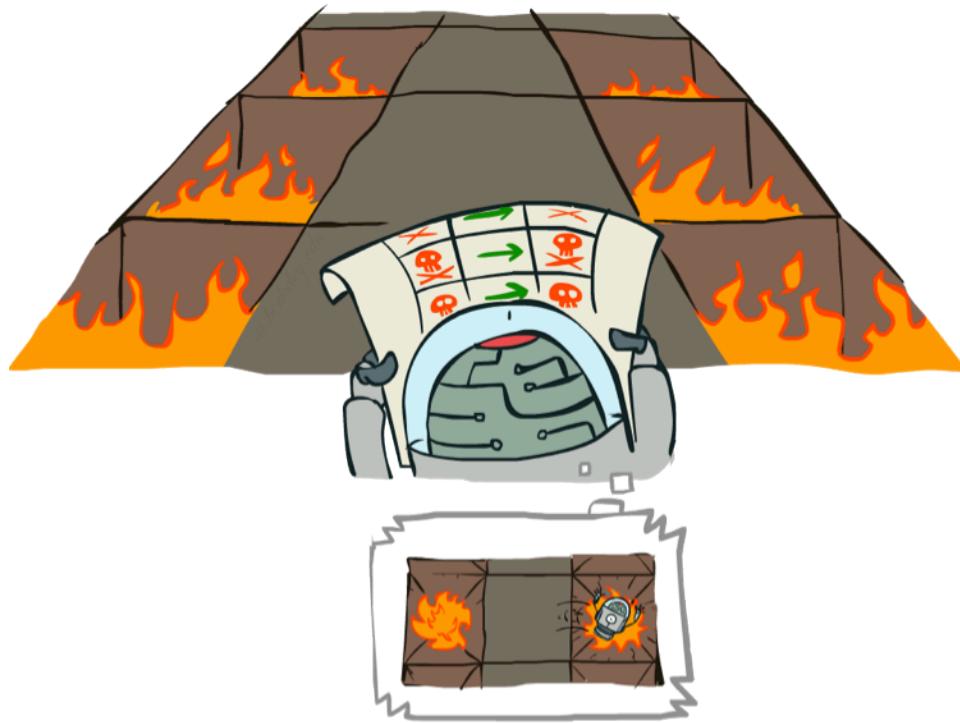
$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') [R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$

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

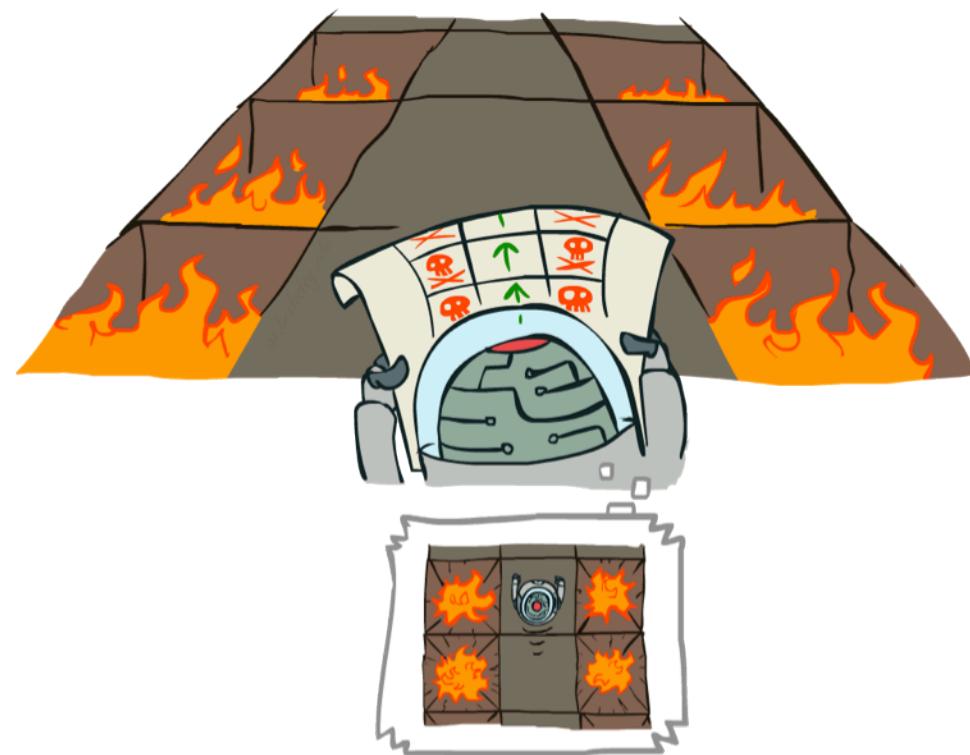
$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')]$$

# Example: Policy Evaluation

Always Go Right



Always Go Forward



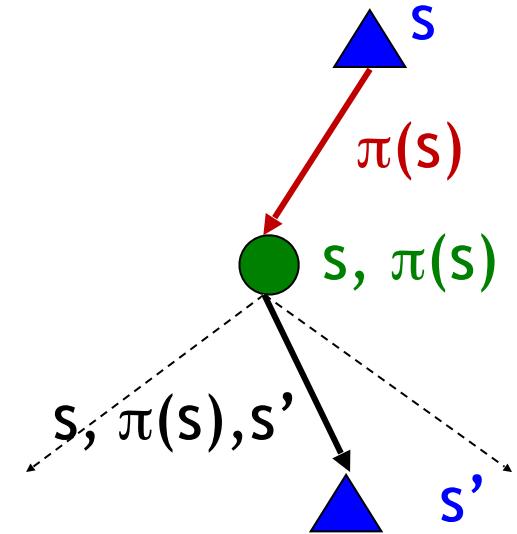
# Policy Evaluation

- 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)



# 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

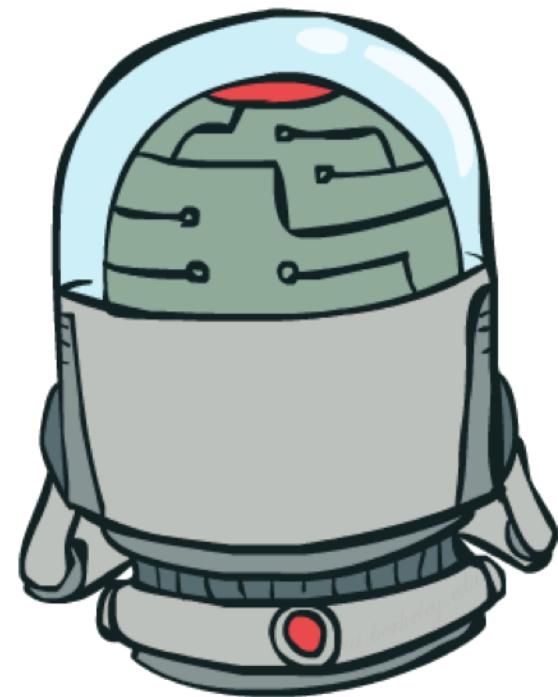
# Summary: MDP Algorithms

---

- 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

# Double Bandits

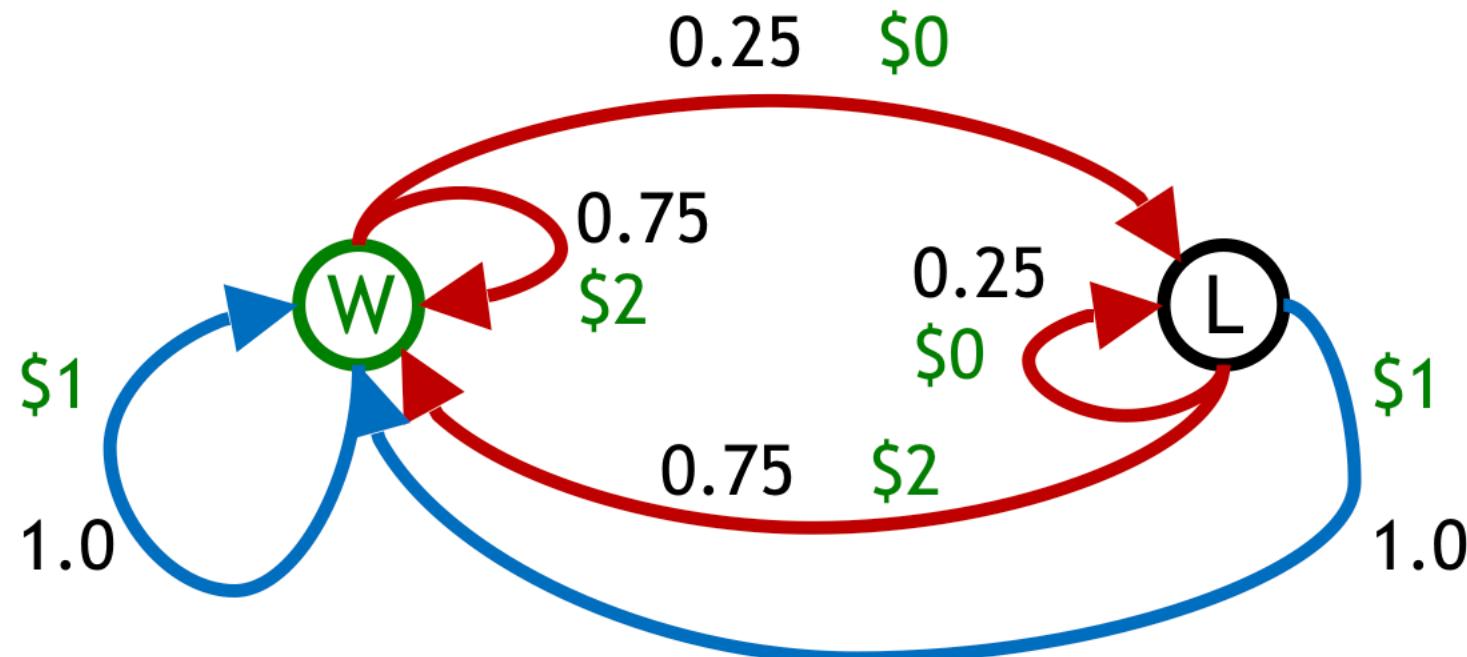
---



# Double-Bandit MDP

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

*No discount  
100 time steps  
Both states have  
the same value*



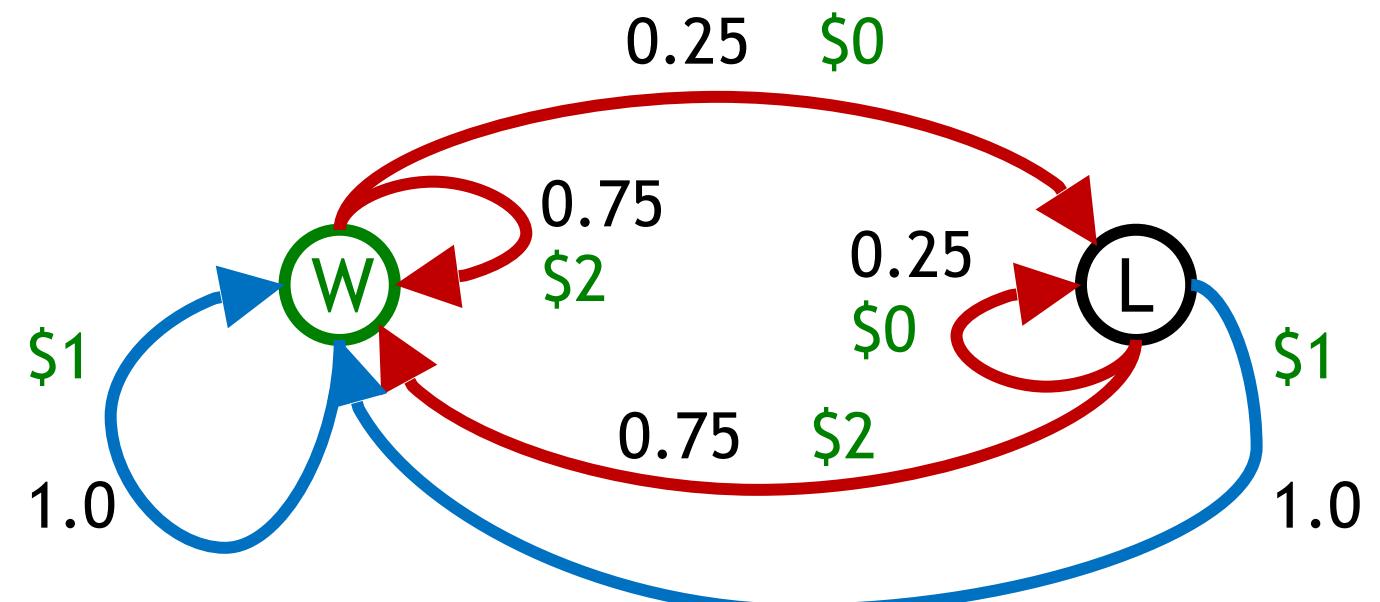
# Offline Planning

- 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!

*No discount  
100 time steps  
Both states have  
the same value*

	Value
Play Red	150
Play Blue	100



# Let's Play!

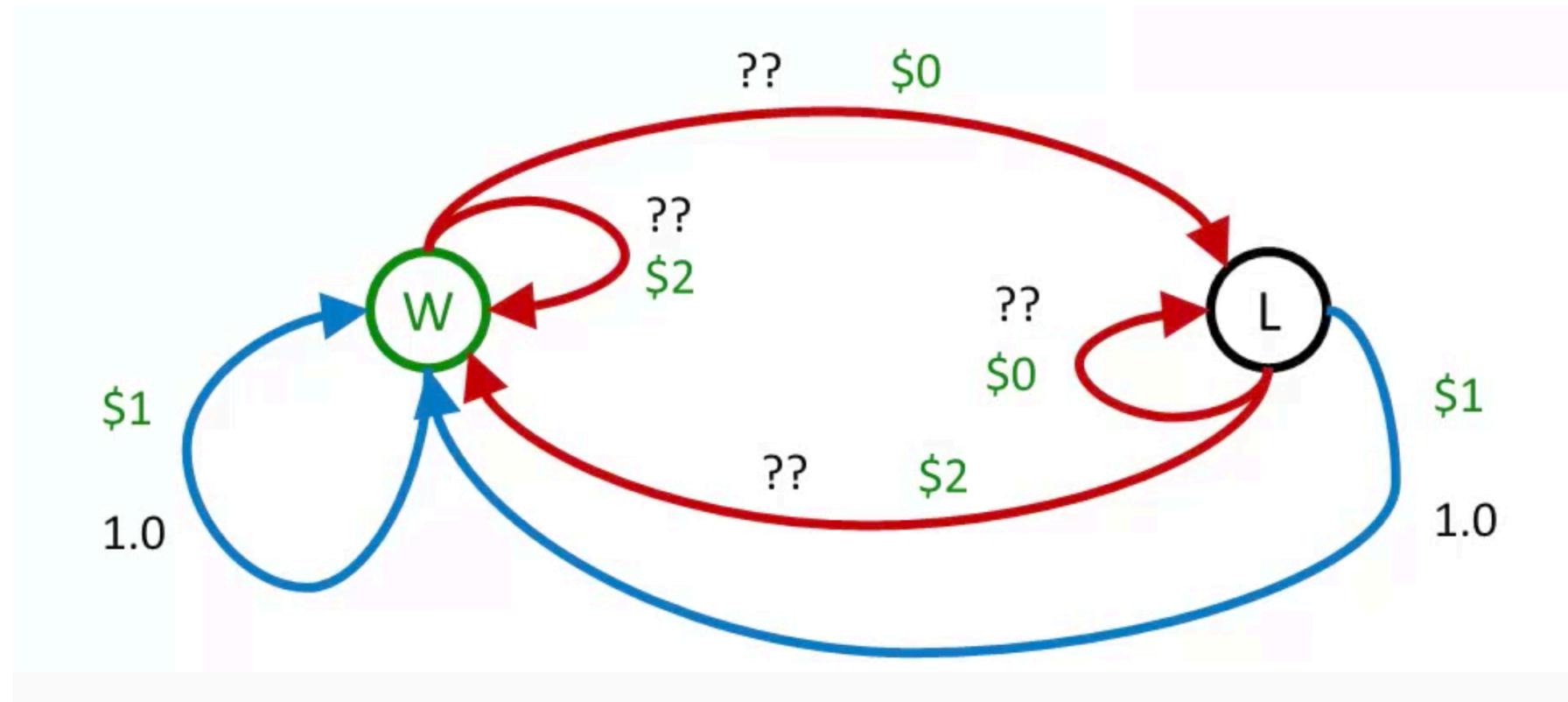
---



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

# Online Planning

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



# Let's Play!

---



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

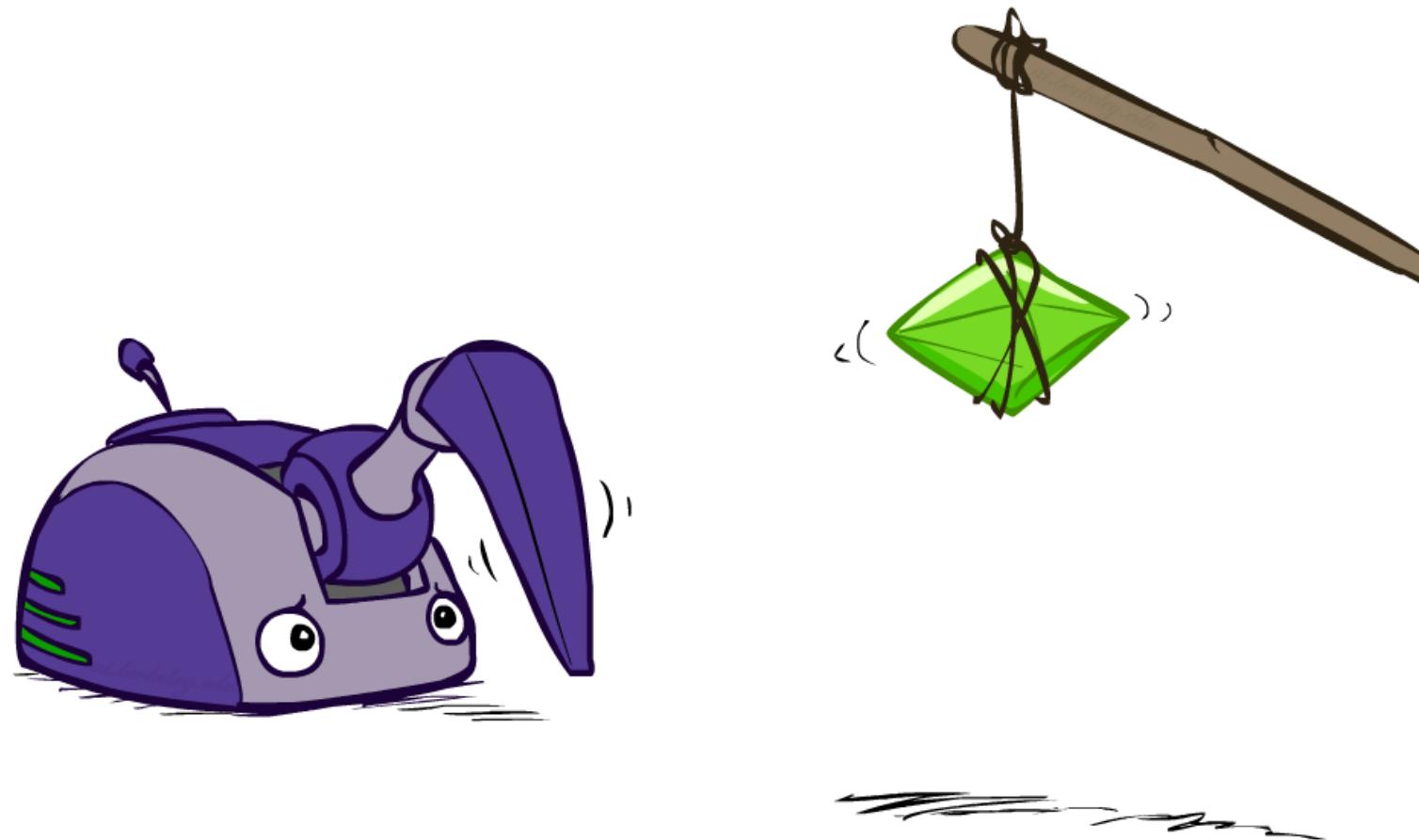
# What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, 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 reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP



# Reinforcement Learning

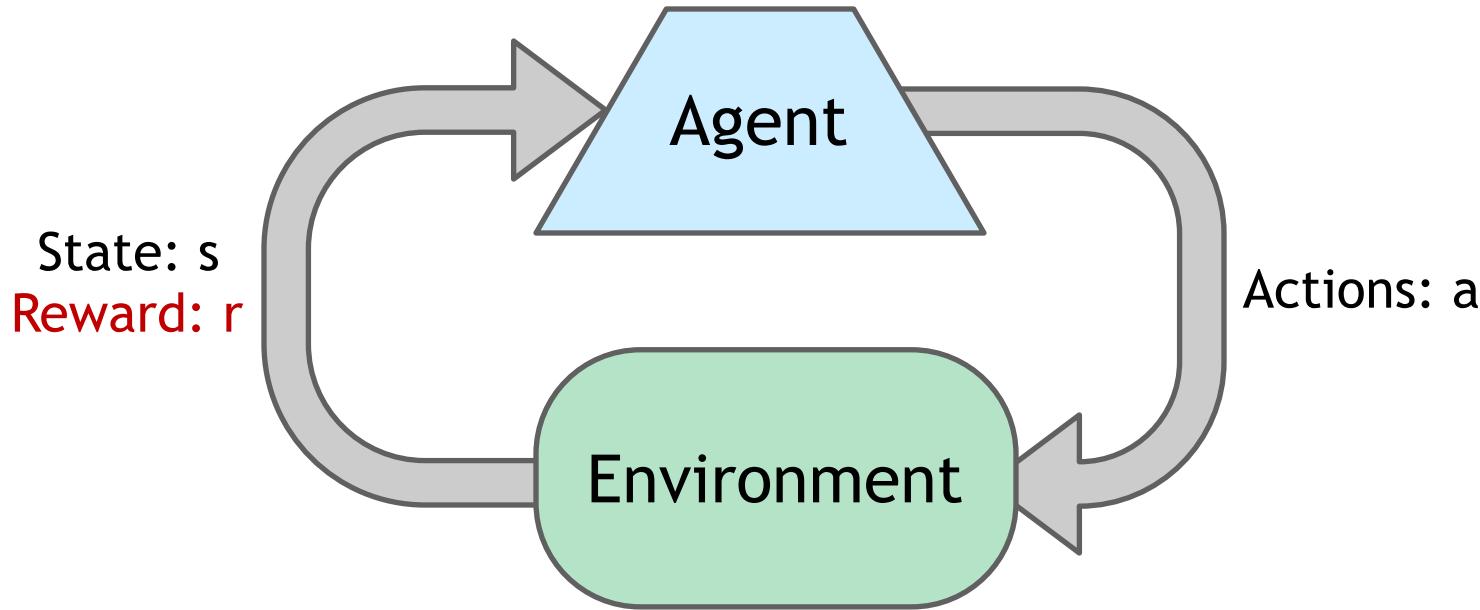
---



<https://www.youtube.com/watch?v=7a7MqMg2JPY&t=8s>

# Reinforcement Learning

---



- Basic idea:
  - Receive feedback in the form of **rewards**
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to **maximize expected rewards**
  - All learning is based on observed samples of outcomes!

# Example: Learning to Walk



Initial



A Learning Trial



After Learning  
[1K Trials]

# Example: Learning to Walk

---



Initial

# Example: Learning to Walk

---



Training

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK - training]

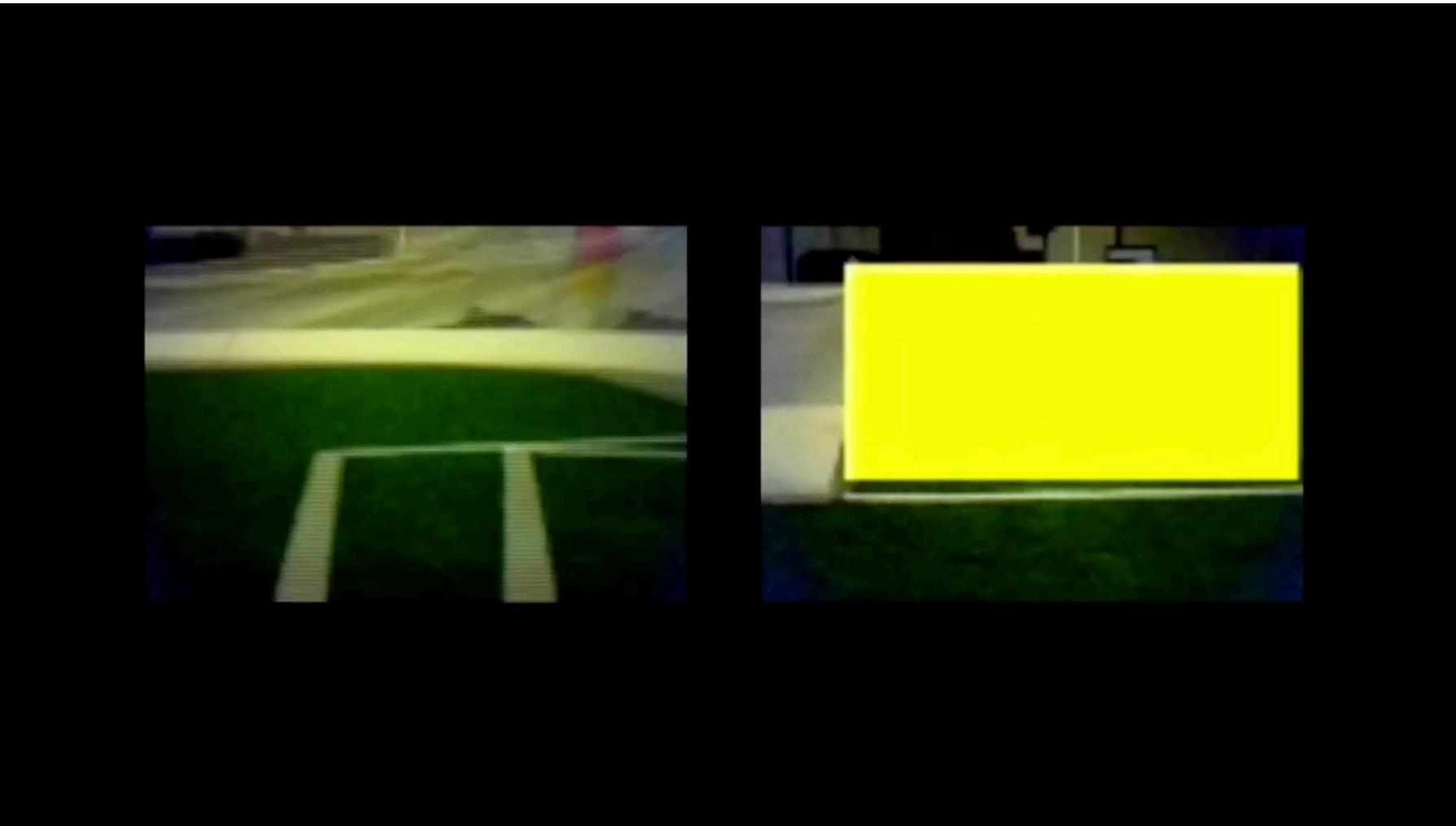
# Example: Learning to Walk

---



Finished

# Example: Learning to Walk



Robot View

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK - robot view]

# The Crawler!



[You, in Homework 3]

# Reinforcement Learning

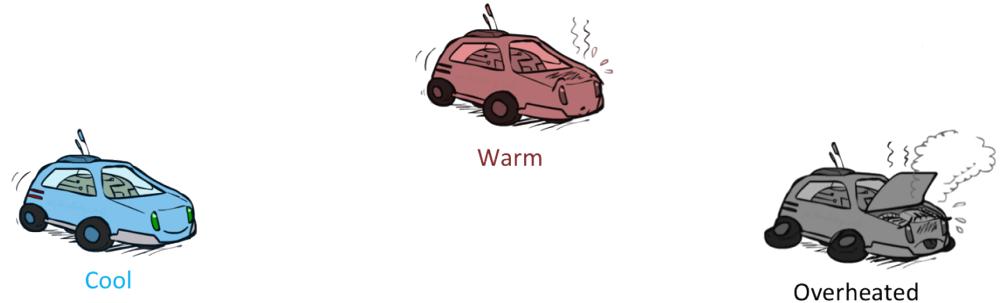
- Still assume a Markov decision process (MDP):

- A set of states  $s \in S$
  - A set of actions (per state)  $A$
  - A model  $T(s,a,s')$
  - A reward function  $R(s,a,s')$

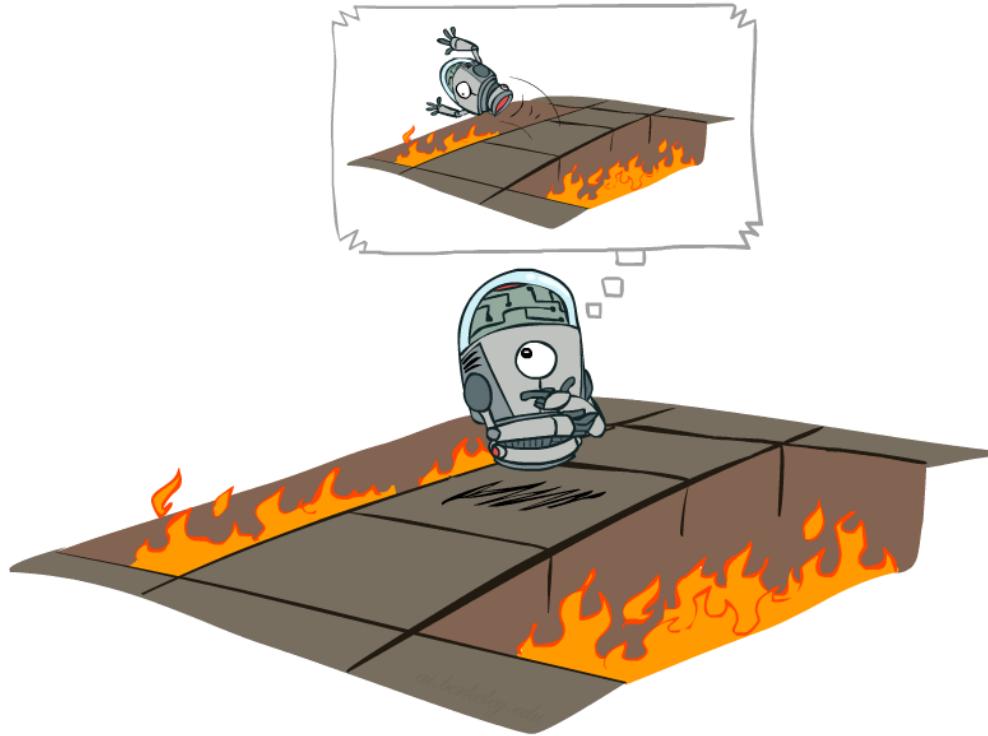
- Still looking for a policy  $\pi(s)$

- New twist: don't know  $T$  or  $R$

- I.e. we don't know which states are good or what the actions do
    - Must actually try actions and states out to learn



# Offline (MDPs) vs. Online (RL)



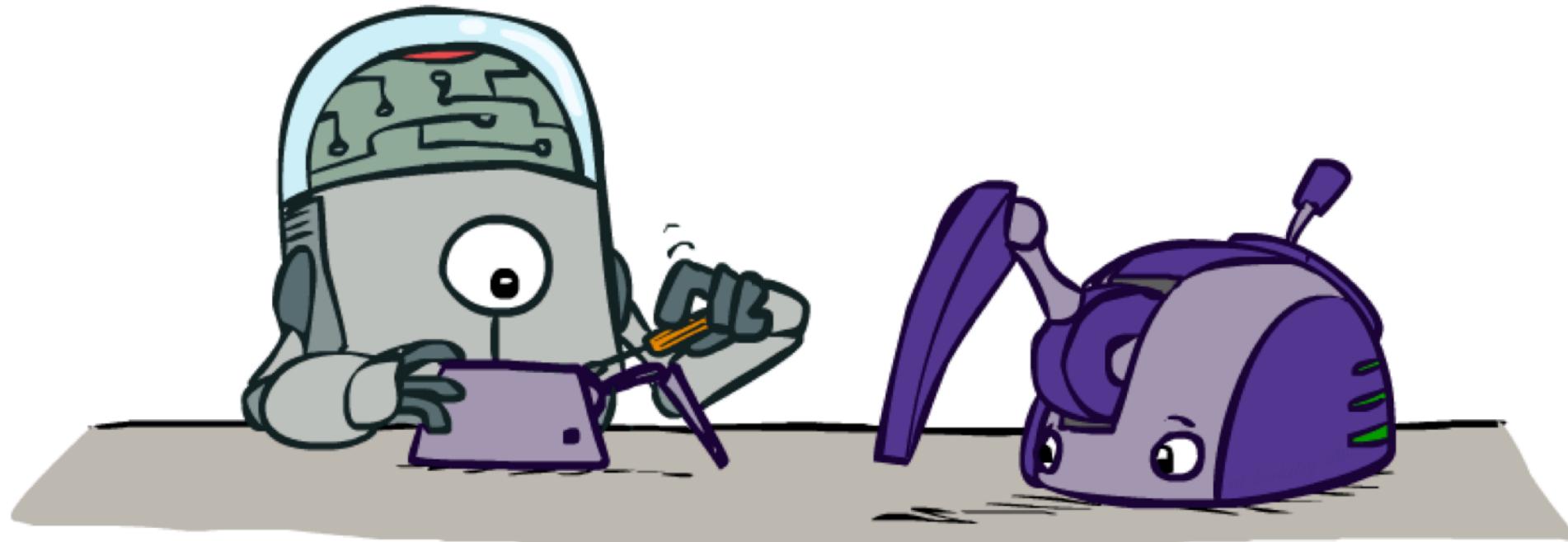
Offline Solution



Online Learning

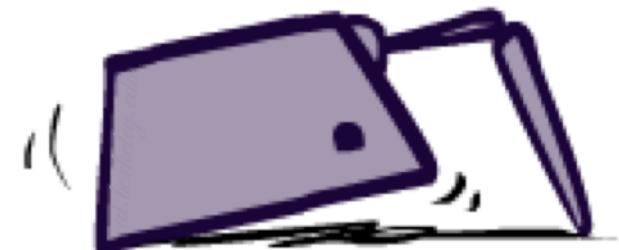
# Model-Based Learning

---



# Model-Based Learning

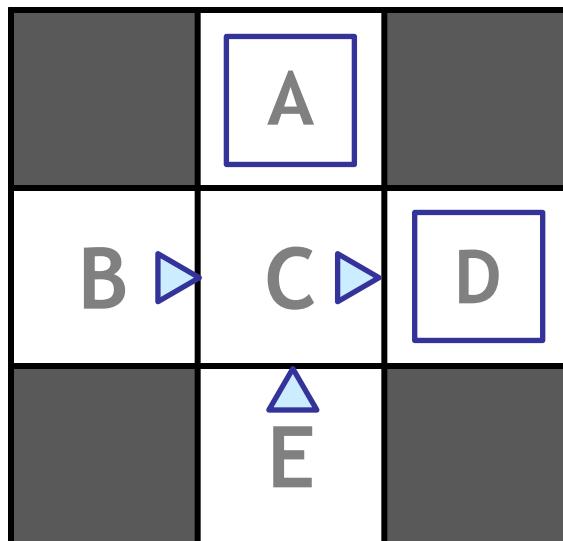
- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct
- Step 1: Learn empirical MDP model
  - Count outcomes  $s'$  for each  $s, a$
  - Normalize to give an estimate of  $\hat{T}(s, a, s')$
  - Discover each  $\hat{R}(s, a, s')$  when we experience  $(s, a, s')$
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before



# Example: Model-Based Learning

Input Policy

$\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 4

E, north, C, -1  
C, east, A, -1  
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

$T(B, \text{east}, C) = 1.00$   
 $T(C, \text{east}, D) = 0.75$   
 $T(C, \text{east}, A) = 0.25$   
...

$\hat{R}(s, a, s')$

$R(B, \text{east}, C) = -1$   
 $R(C, \text{east}, D) = -1$   
 $R(D, \text{exit}, x) = +10$   
...

# Example: Expected Age

Goal: Compute expected age of cse5522 students

Known  $P(A)$

$$E[A] = \sum_a P(a) \cdot a = 0.35 \times 20 + \dots$$

Without  $P(A)$ , instead collect samples  $[a_1, a_2, \dots a_N]$

Unknown  $P(A)$ : “Model Based”

Why does this work? Because eventually you learn the right model.

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

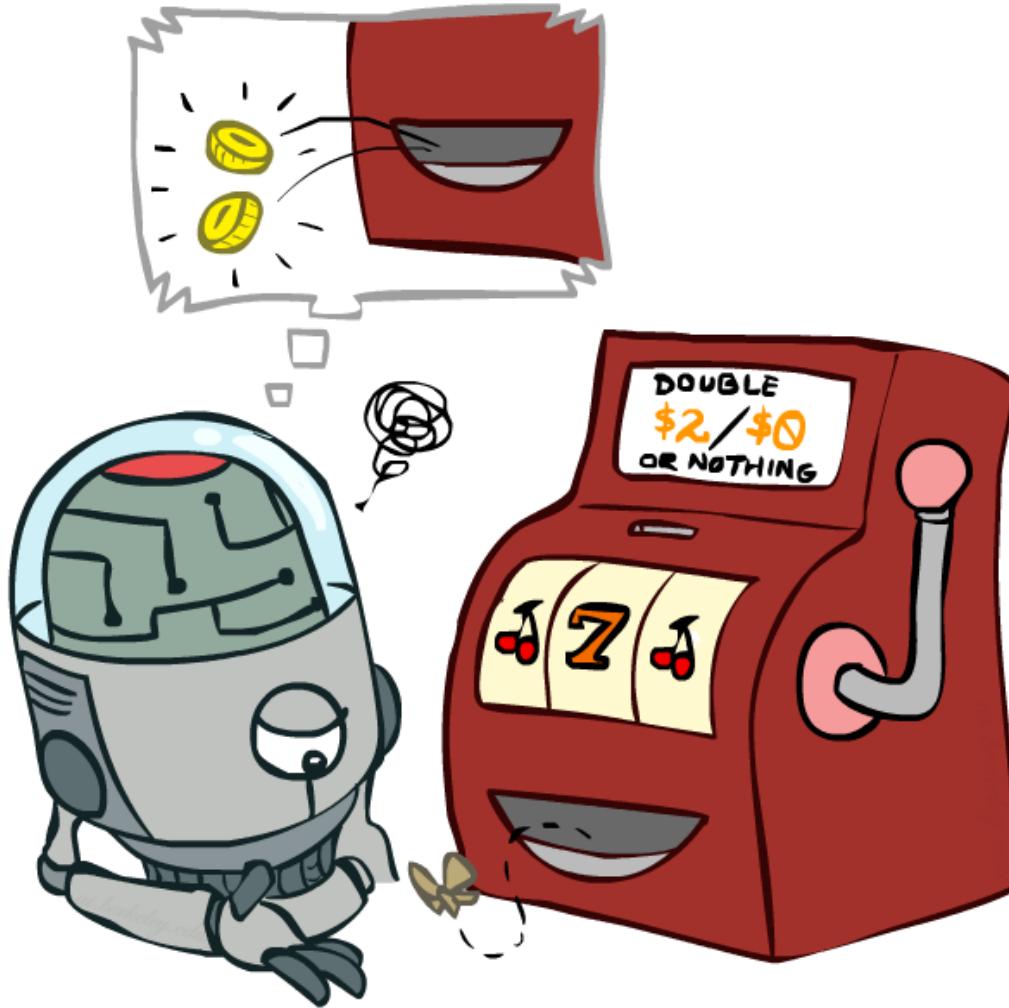
Unknown  $P(A)$ : “Model Free”

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Why does this work? Because samples appear with the right frequencies.

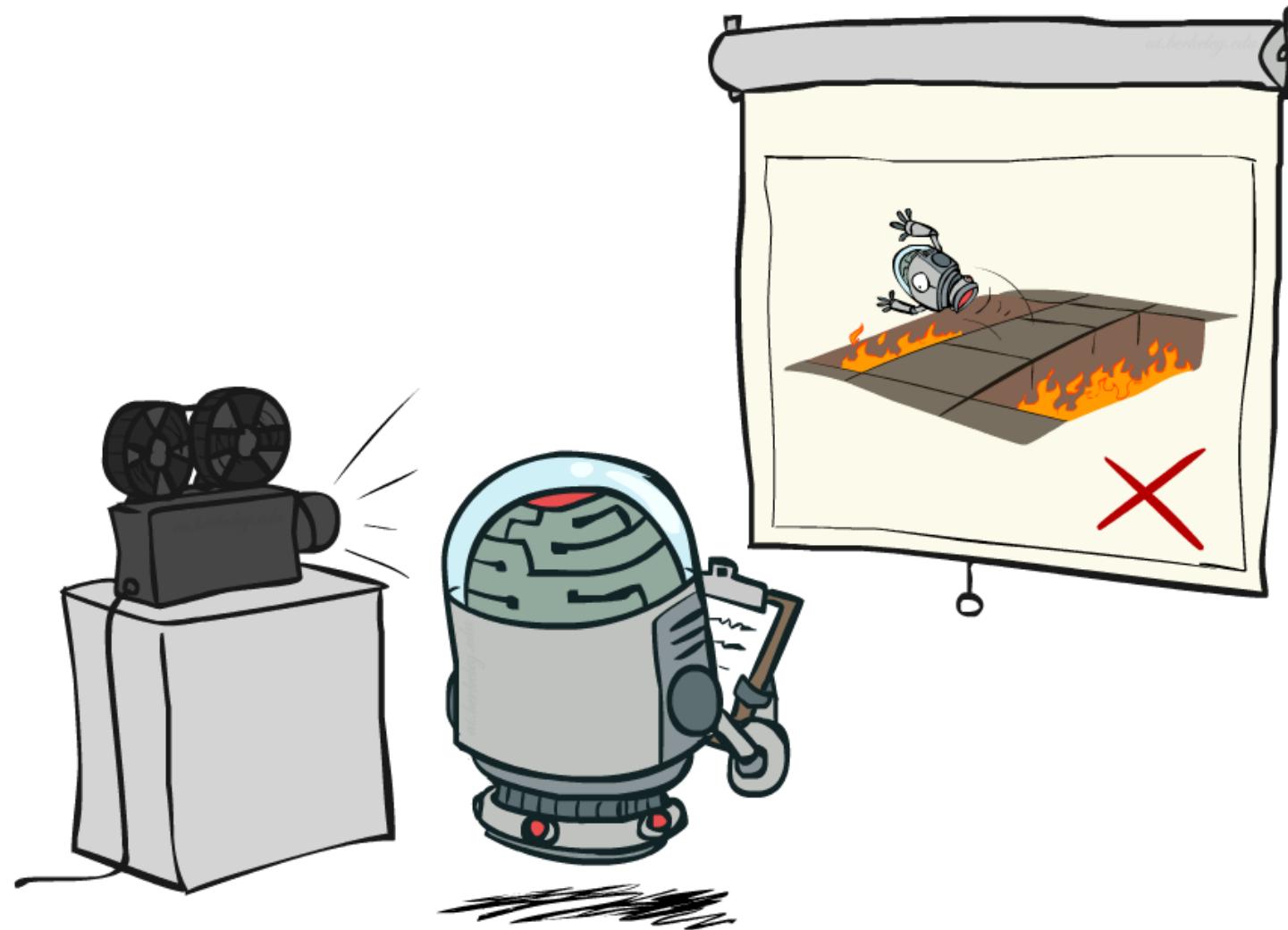
# Model-Free Learning

---



# Passive Reinforcement Learning

---



# Passive vs. Active Learning

---

- **Passive Learning:**
  - The agent has a fixed policy and tried to learn the utilities of states by observing the world go by
  - Analogous to policy evaluation
  - Often serves as a component of active learning
  - Often inspires active learning algorithm
- **Active Learning:**
  - The agent attempts to find an optimal (or at least good) policy by acting in the world
  - Analogous to solving the underlying MDP, but without first being given the MDP model

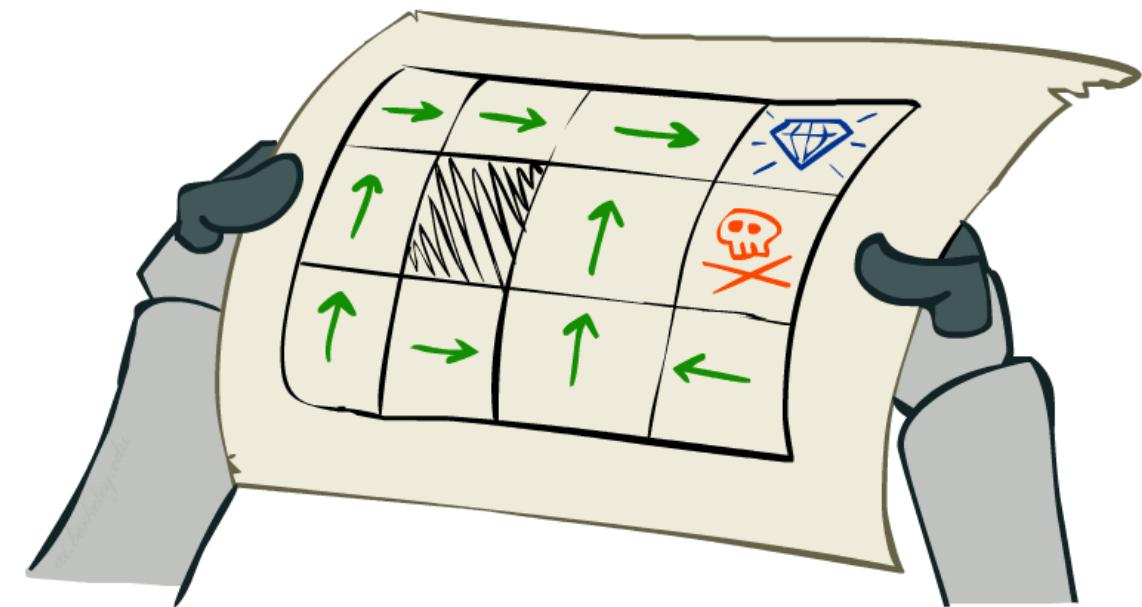
# Passive Reinforcement Learning

- Simplified task: policy evaluation

- Input: a fixed policy  $\pi(s)$
- You don't know the transitions  $T(s,a,s')$
- You don't know the rewards  $R(s,a,s')$
- Goal: learn the state values

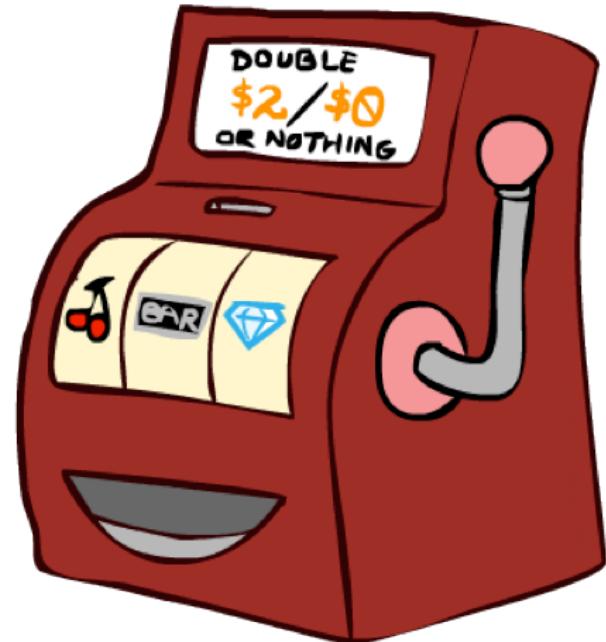
- In this case:

- Learner is “along for the ride”
- No choice about what actions to take
- Just execute the policy and learn from experience
- This is NOT offline planning! You actually take actions in the world.



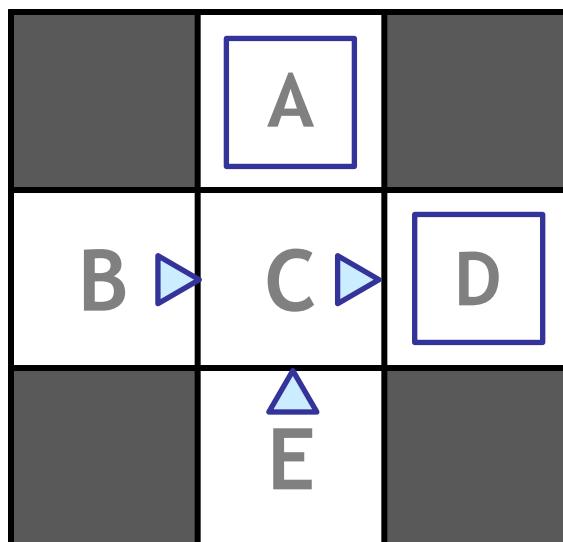
# Direct Evaluation

- Goal: Compute values for each state under  $\pi$
- Idea: Average together observed sample values
  - Act according to  $\pi$
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples
- This is called direct evaluation



# Example: Direct Evaluation

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 4

E, north, C, -1  
C, east, A, -1  
A, exit, x, -10

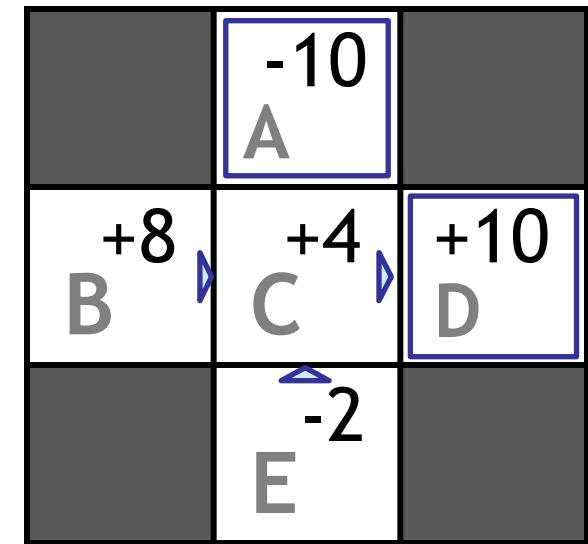
Output Values

	-10	
A	+4	+10
B	+8	
C		
D		
E	-2	

# Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions
- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

## Output Values



*If B and E both go to C under this policy, how can their values be different?*

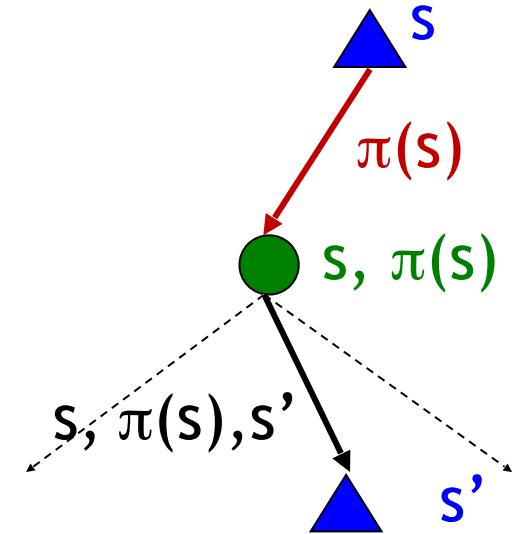
# Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate  $V$  for a fixed policy:

- Each round, replace  $V$  with a one-step-look-ahead layer over  $V$

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

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



- This approach fully exploited the connections between the states
  - Unfortunately, we need  $T$  and  $R$  to do it!
- Key question: how can we do this update to  $V$  without knowing  $T$  and  $R$ ?
  - In other words, how to we take a weighted average without knowing the weights?

# Sample-Based Policy Evaluation?

- We want to improve our estimate of  $V$  by computing these averages:

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

- Idea: Take samples of outcomes  $s'$  (by doing the action!) and average

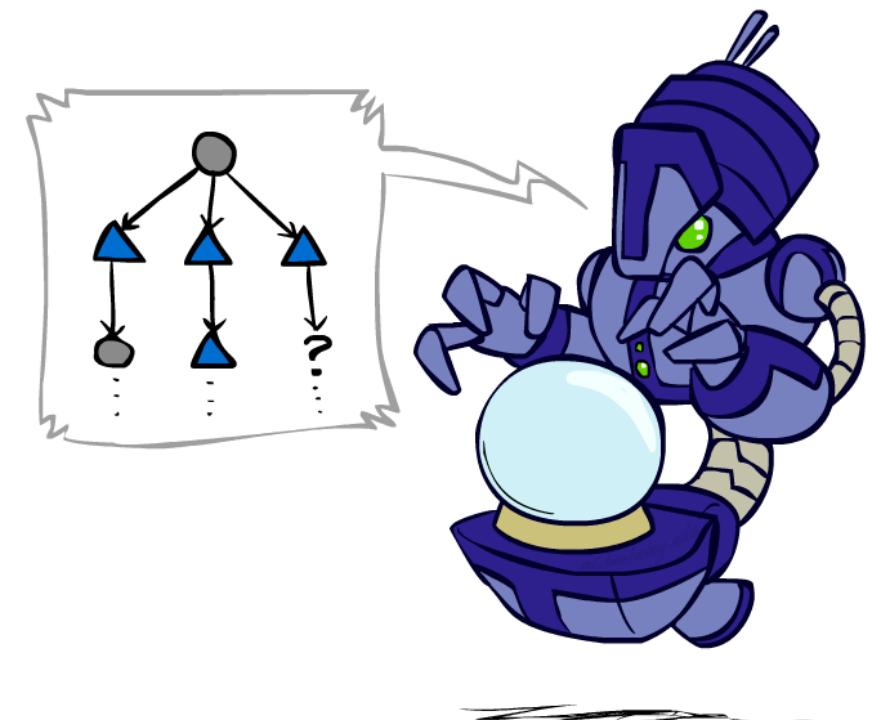
$$\text{sample}_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$

$$\text{sample}_2 = R(s, \pi(s), s'_2) + \gamma V_k^{\pi}(s'_2)$$

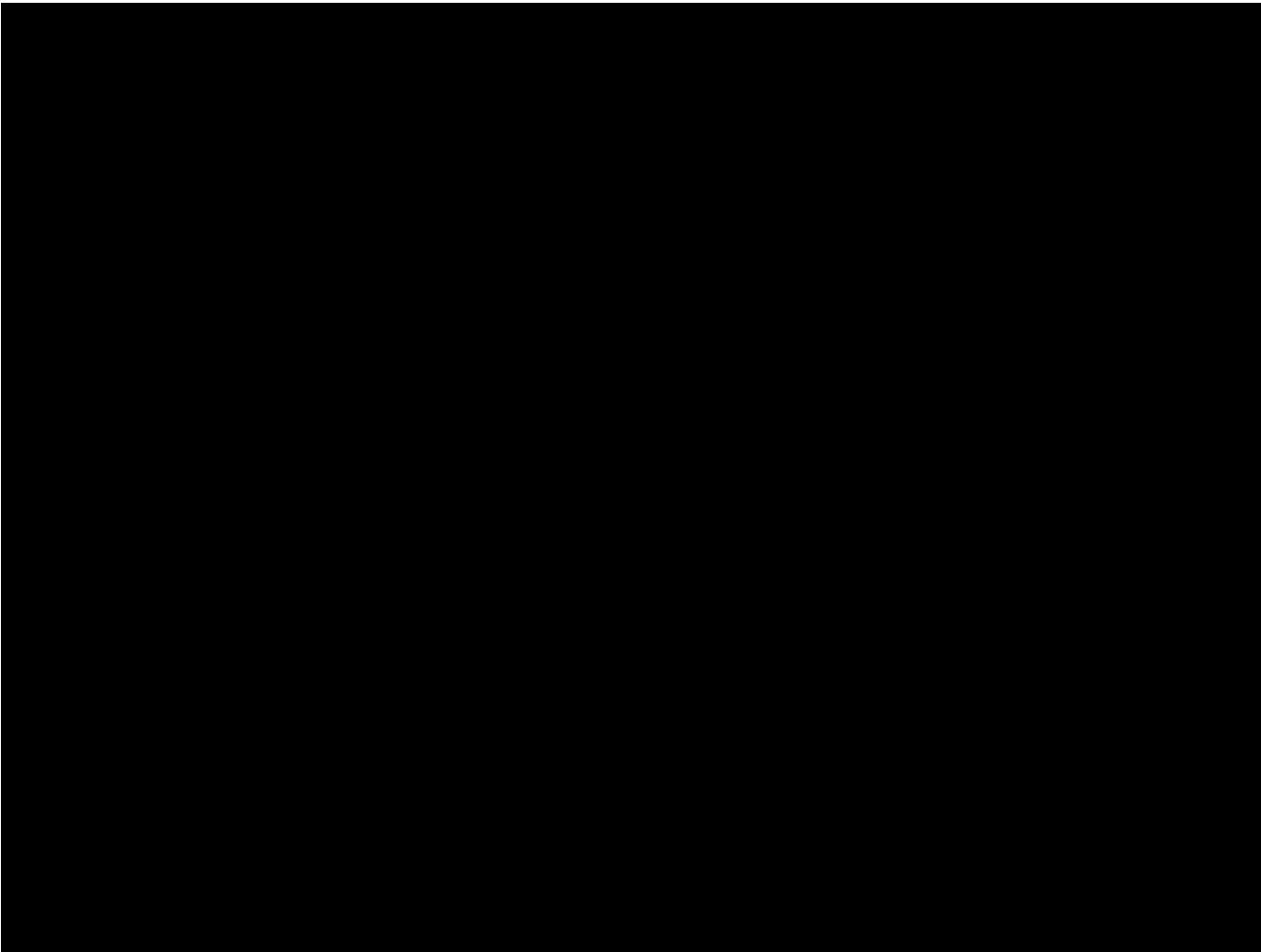
...

$$\text{sample}_n = R(s, \pi(s), s'_n) + \gamma V_k^{\pi}(s'_n)$$

$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i \text{sample}_i$$



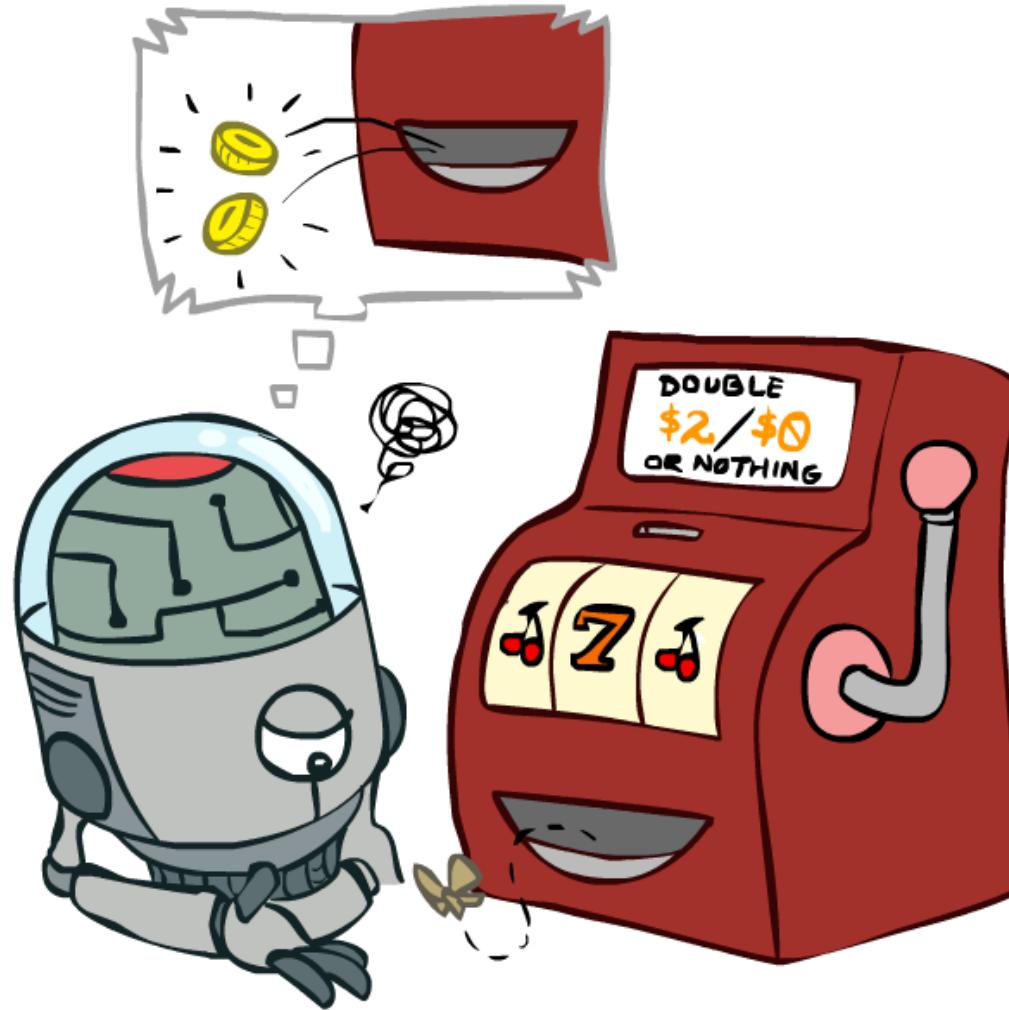
# Crawler in Real-world



**[Francis Waffels 2010]** An example of Q-learning using 16 states (4 positions for each servo's) and 4 possible actions (moving 1 position up or down for on of the servos). The robot is build with lego components, two servo's, a wheel encoder and a dwengo-board with PIC18F4550 microcontroller. The robot learns using the Q-learning algorithm and its reward is related with the moved distance.

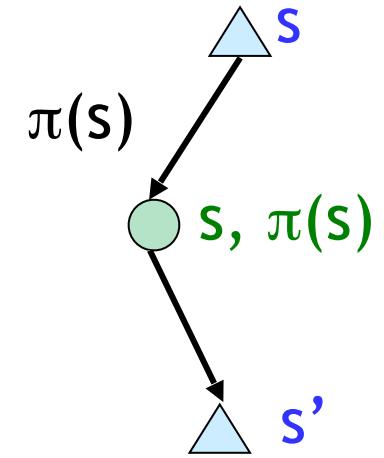
# Temporal Difference Learning

---



# Temporal Difference Learning

- Big idea: learn from every experience!
  - Update  $V(s)$  each time we experience a transition  $(s, a, s', r)$
  - Likely outcomes  $s'$  will contribute updates more often
- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average



Sample of  $V(s)$ :  $sample = R(s, \pi(s), s') + \gamma V^\pi(s')$

Update to  $V(s)$ :  $V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample$

Same update:  $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

# Exponential Moving Average

---

- Exponential moving average

- The running interpolation update:  $\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$
- Makes recent samples more important:

$$\bar{x}_n = \frac{x_n + (1 - \alpha) \cdot x_{n-1} + (1 - \alpha)^2 \cdot x_{n-2} + \dots}{1 + (1 - \alpha) + (1 - \alpha)^2 + \dots}$$

- Forgets about the past (distant past values were wrong anyway)
- Decreasing learning rate (alpha) can give converging averages

# Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Observed Transitions

$$R(s, \pi(s), s')$$

B, east, C, -2

C, east, D, -2

$V^\pi(s)$	0	0
0	0	8
	0	

$V^\pi(s) \leftarrow$	-1	0
-1	0	8
	0	

-1	3	8
-1	3	8
0	0	

Assume:  $\gamma = 1$ ,  
 $\alpha = 1/2$

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

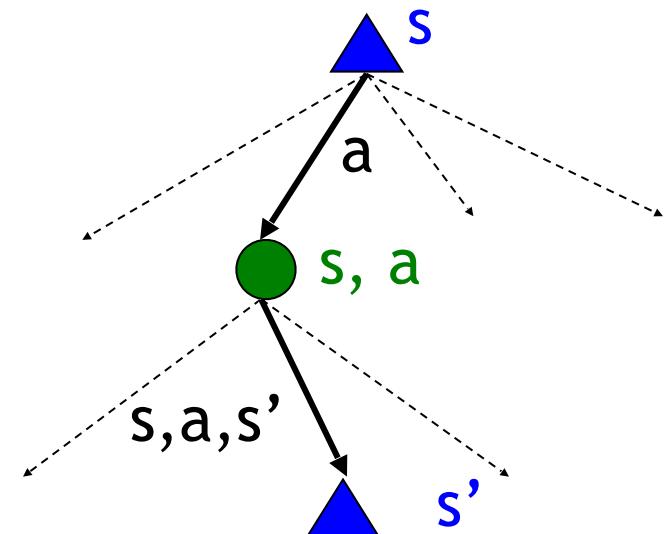
# Problems with TD Value Learning

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, if we want to turn values into a (new) policy, we're sunk:

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

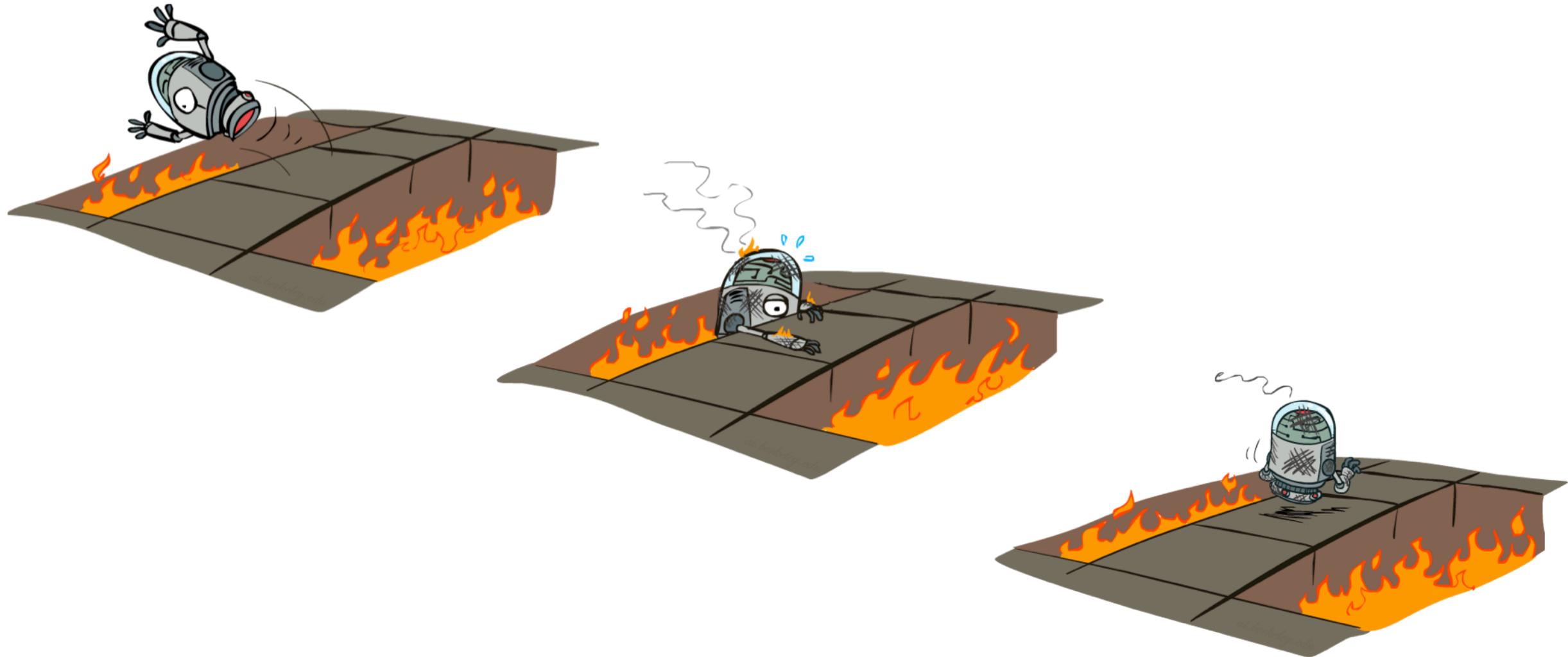
$$Q(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V(s')]$$

- Idea: learn Q-values, not values
- Makes action selection model-free too!



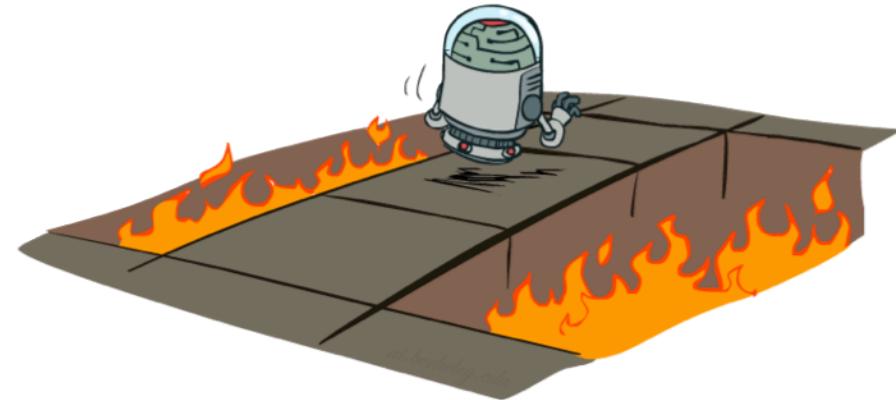
# Active Reinforcement Learning

---



# Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions  $T(s,a,s')$
  - You don't know the rewards  $R(s,a,s')$
  - You choose the actions now
  - Goal: learn the optimal policy / values
- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens...



# Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values

- Start with  $V_0(s) = 0$ , which we know is right

- Given  $V_k$ , calculate the depth  $k+1$  values for all states:

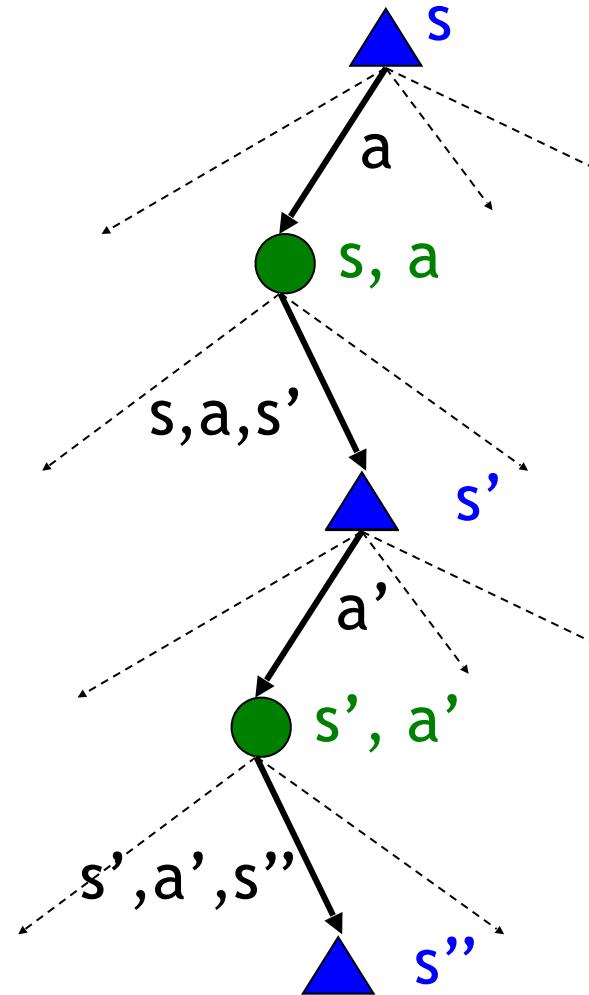
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- But Q-values are more useful, so compute them instead

- Start with  $Q_0(s, a) = 0$ , which we know is right

- Given  $Q_k$ , calculate the depth  $k+1$  q-values for all q-states:

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q_k(s', a')]$$



# Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

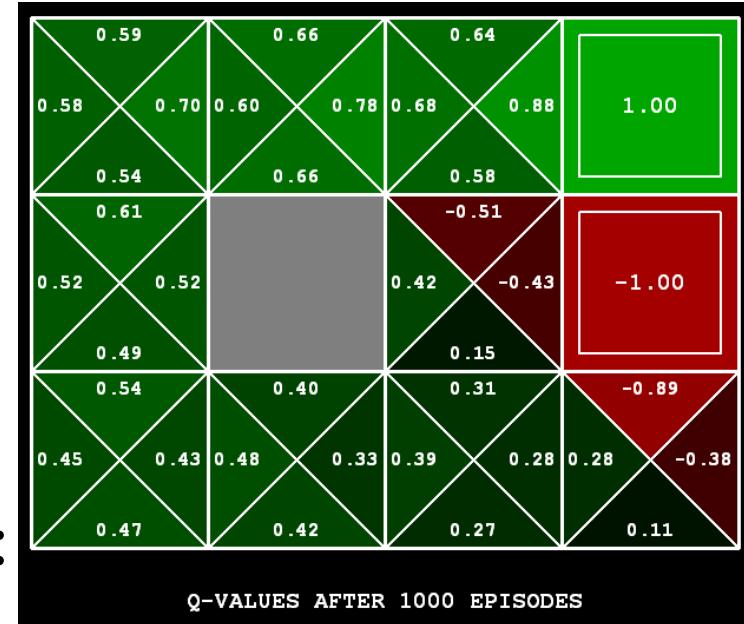
- Learn  $Q(s, a)$  values as you go

- Receive a sample  $(s, a, s', r)$
- Consider your old estimate:  $Q(s, a)$
- Consider your new sample estimate:

$$\text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

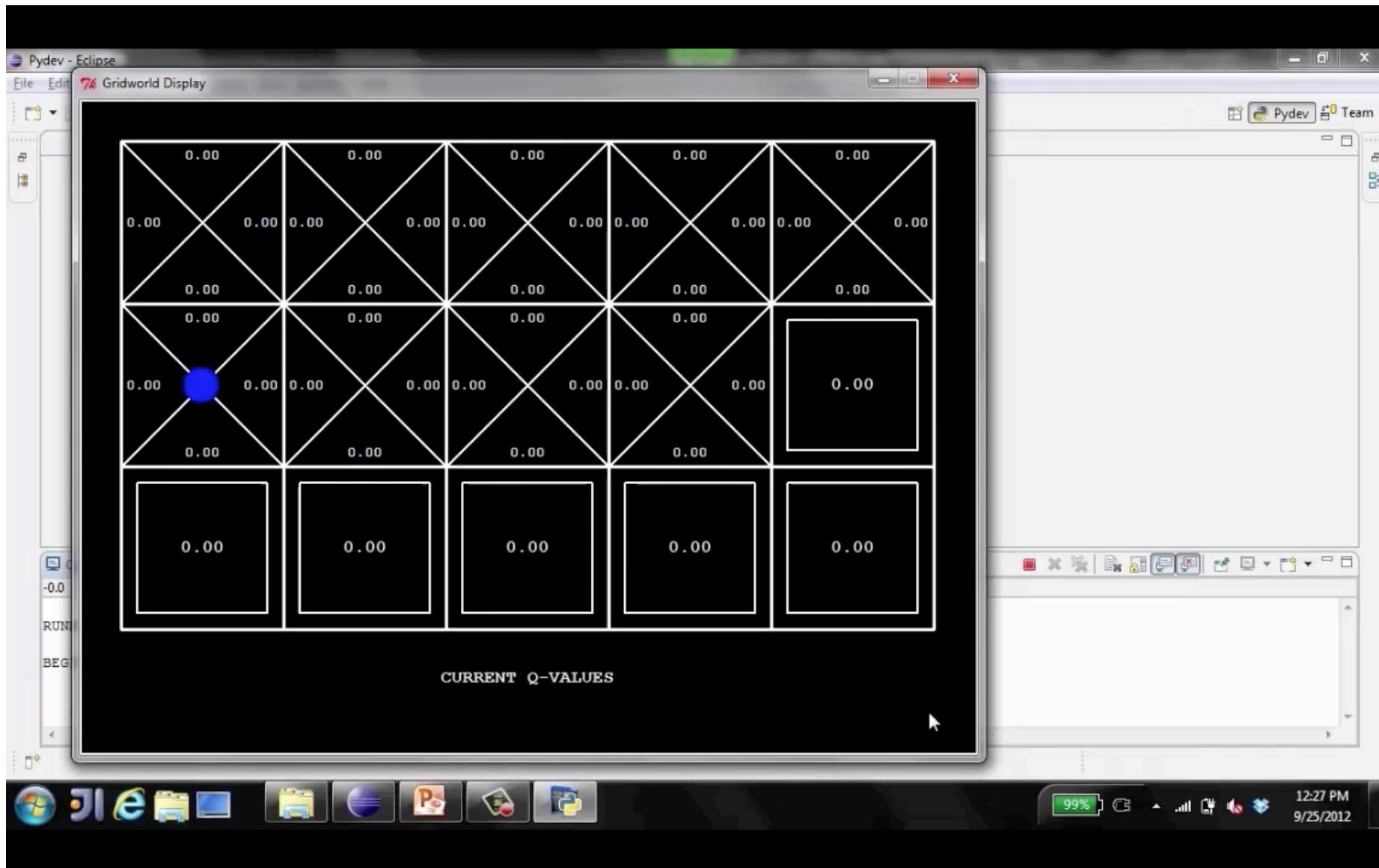
- Incorporate the new estimate into a running average:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) [\text{sample}]$$



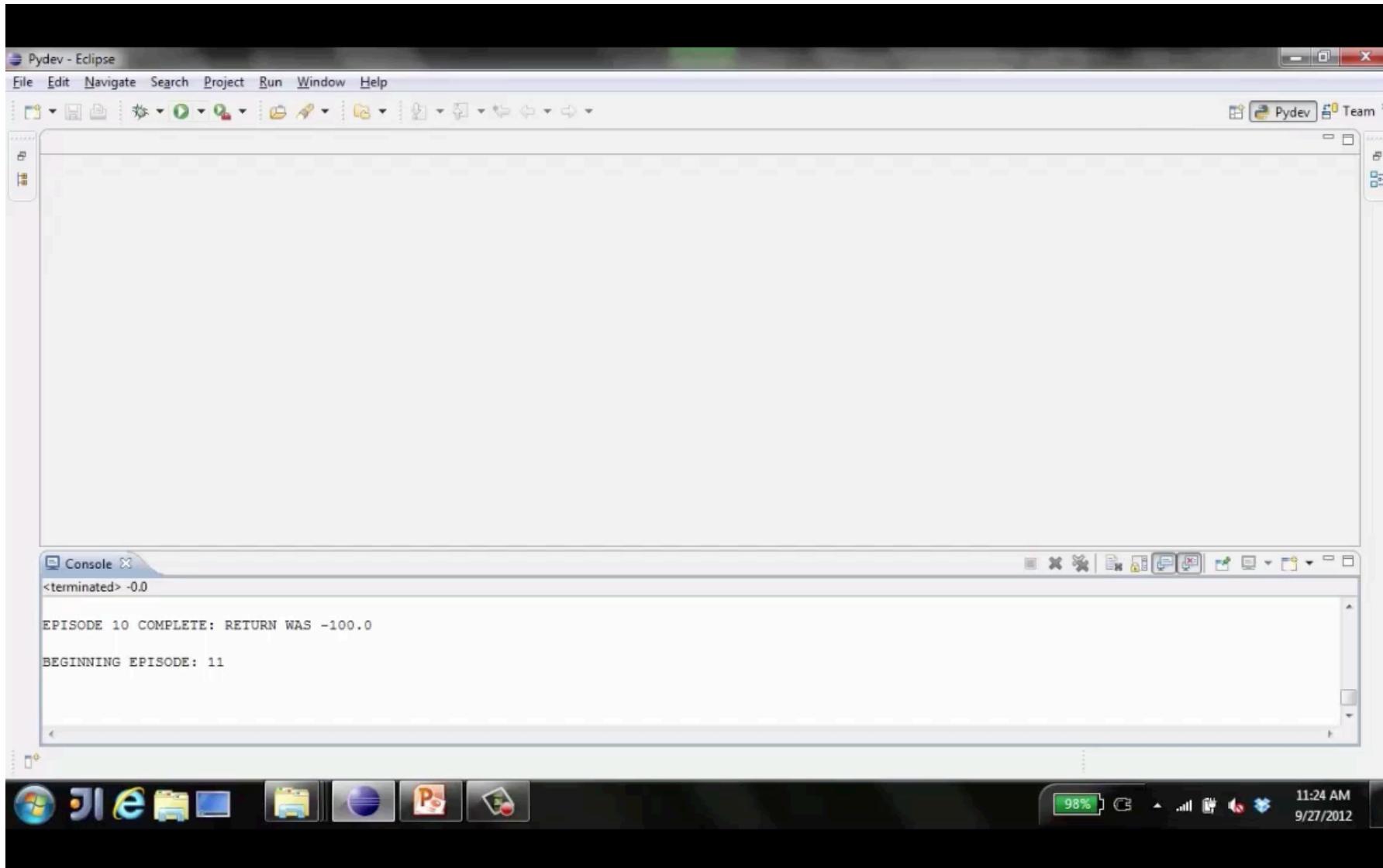
[Demo: Q-learning - gridworld (L10D2)]

# Video of Demo Q-Learning -- Gridworld



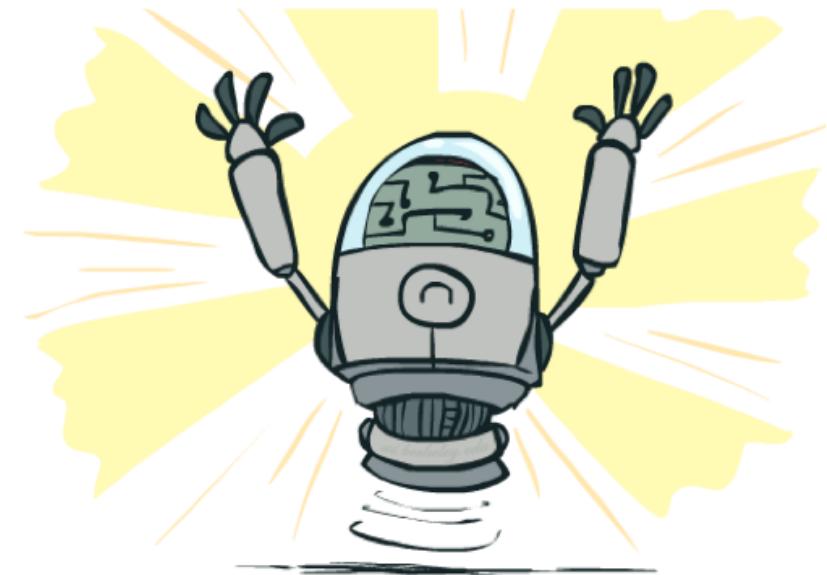
$\alpha = 0.5$

# Video of Demo Q-Learning Auto Cliff Grid



# Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called **off-policy learning**
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - ... but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)



# The Story So Far: MDPs and RL

---

Things we know how to do:

- If we know the MDP
  - Compute  $V^*$ ,  $Q^*$ ,  $\pi^*$  exactly
  - Evaluate a fixed policy  $\pi$
- If we don't know the MDP:
  - We can estimate the MDP then solve it
  - We can estimate  $V$  for a fixed policy  $\pi$
  - We can estimate  $Q^*(s,a)$  for the optimal policy while executing exploration policies

Techniques:

- Offline MDP Solution
  - Value Iteration / Policy Iteration
  - Policy evaluation
- Reinforcement Learning:
  - Model-based RL
  - Model-free: Value learning
  - Model-free: Q-learning

# Example: Sidewinding



# Example: Ducklings

---



[Video: Ducklings vs. Stairs]