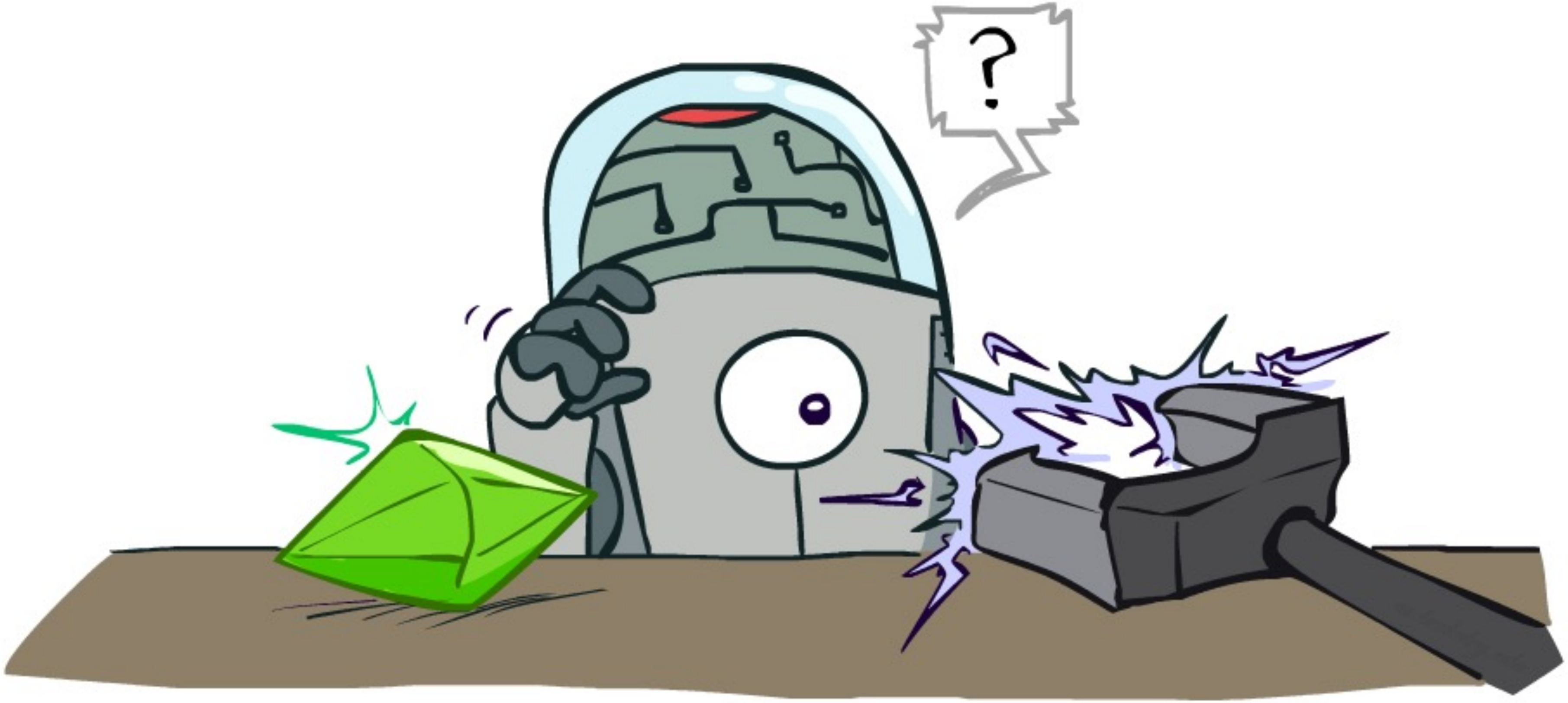
# Announcements

- **Homework 3** due **today (Sept 26)** at 11:59pm PT
- **Project 3** released and due **next Friday (Oct 6)** at 11:59pm PT

# CS 188: Artificial Intelligence

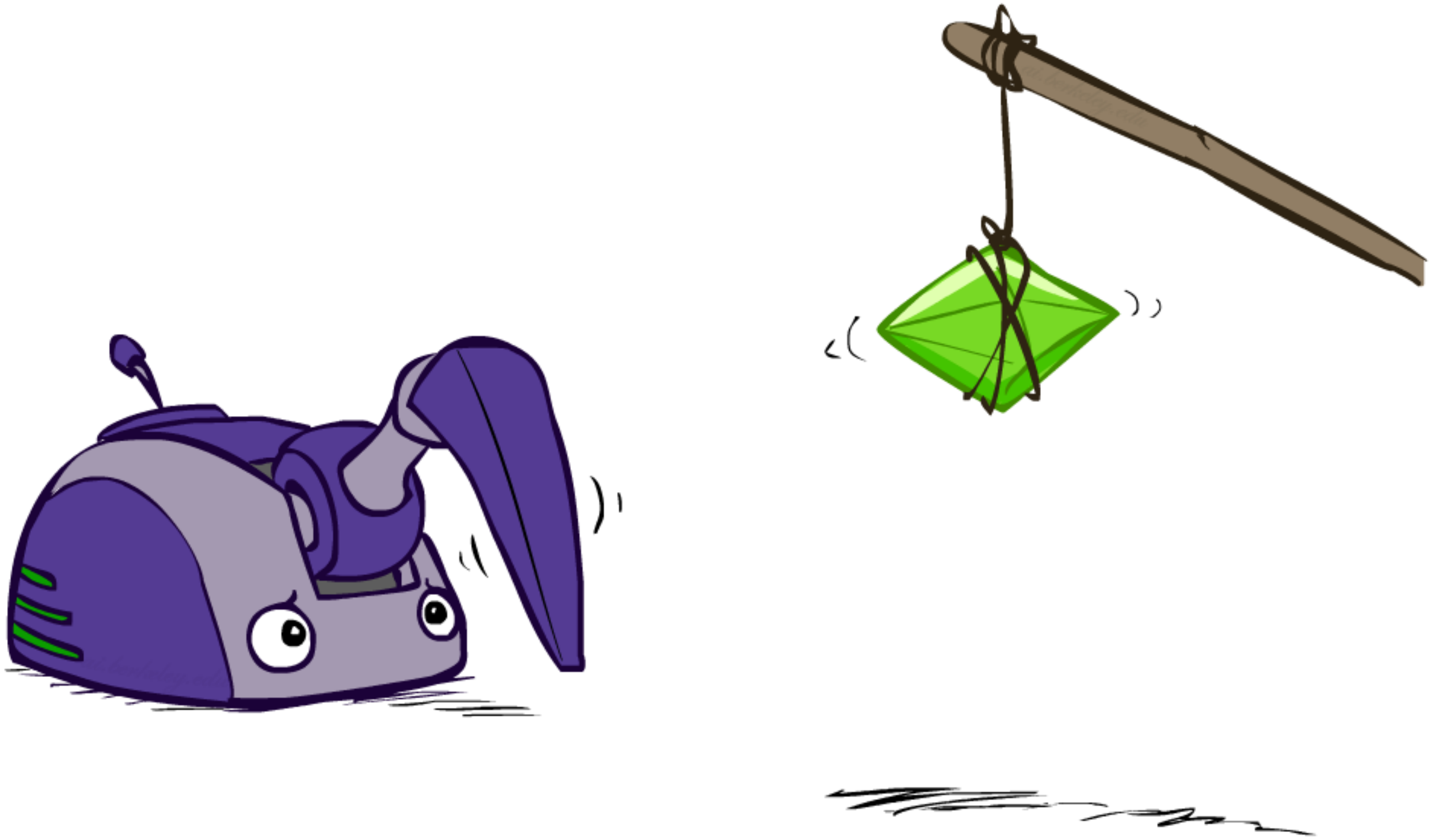
## Reinforcement Learning



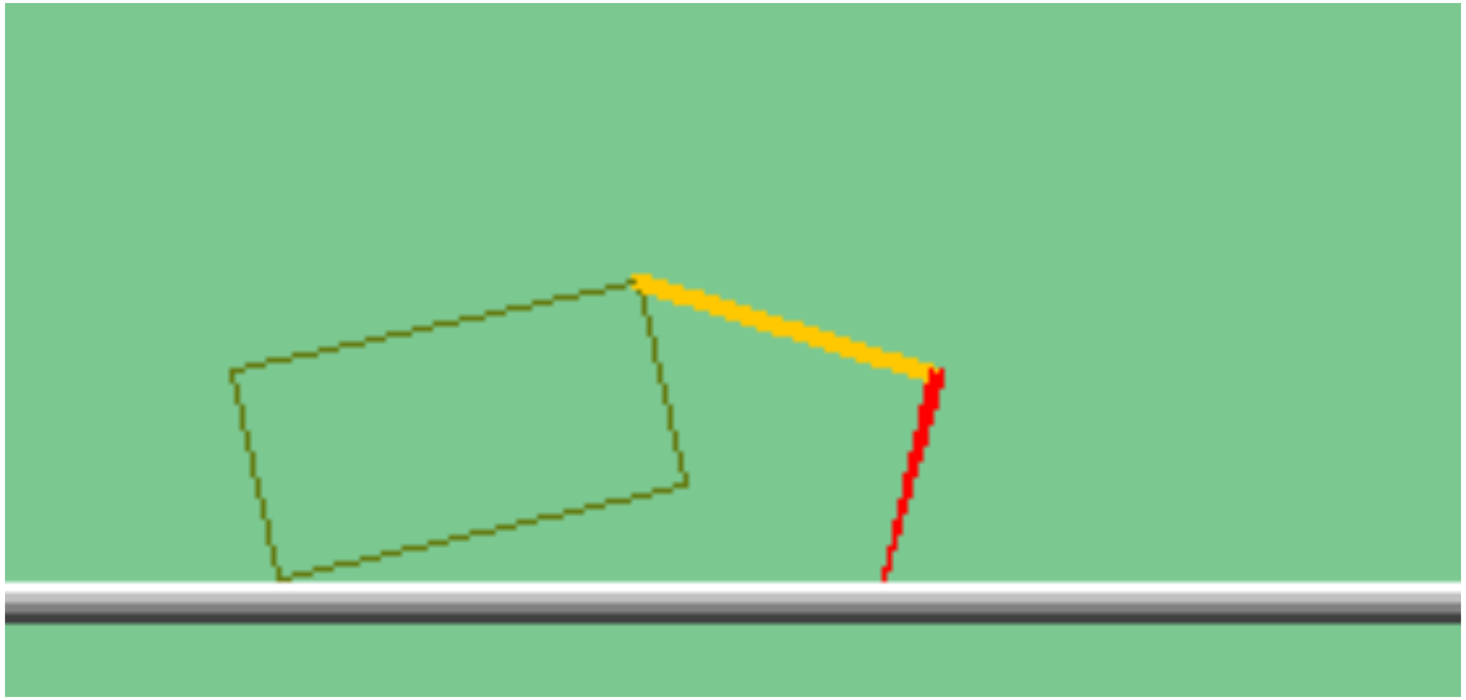

University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

# Reinforcement Learning



# The Crawler!



[Demo: Crawler Bot (L10D1)]

# Video of Demo Crawler Bot

# Quadruped Robot Learning in Berkeley Hills

[Smith et al, 2022]

# Reinforcement Learning: Overview of this week

- **Passive Reinforcement Learning:** how to learn from already given experiences
  - **Model-based:** learn the MDP model from experiences, then solve the MDP
  - **Model-free:** forego learning the MDP model, directly learn V or Q
    - Value learning: learns value of a fixed policy
      - 2 approaches: *Direct Evaluation* & *TD Learning*
    - Q learning: learns Q values of the optimal policy (Q version of TD Learning)
- **Active Reinforcement Learning:** how to collect new experiences
- **Approximate Reinforcement Learning:** to handle large state spaces
- **Case studies:** game playing, robotics, language assistants

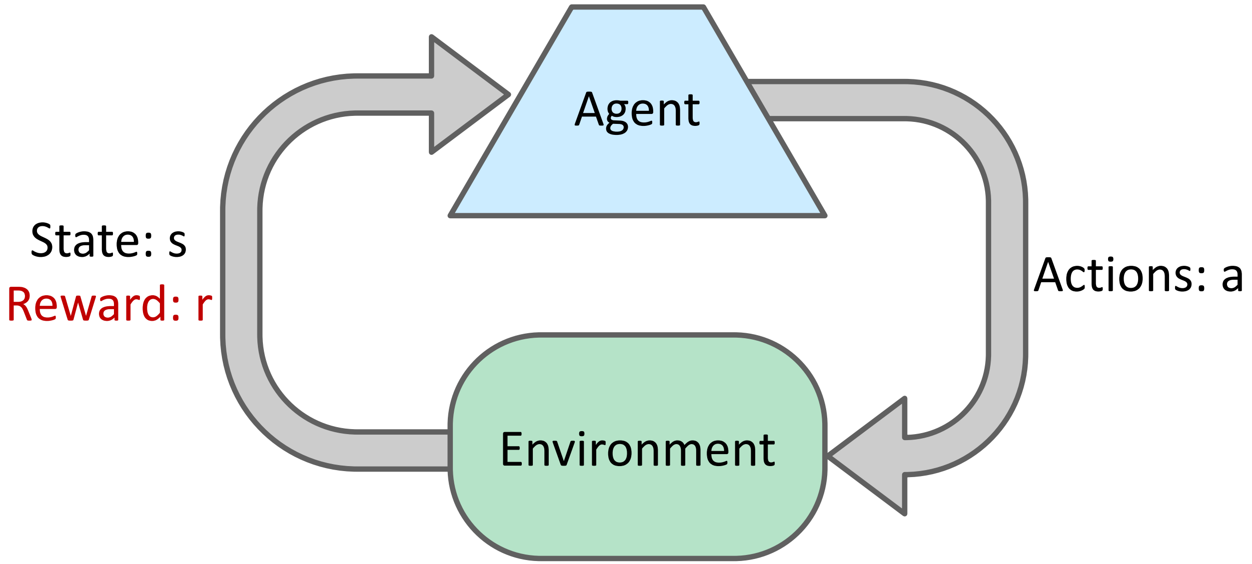
# Reinforcement Learning

- 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 out actions and states to learn

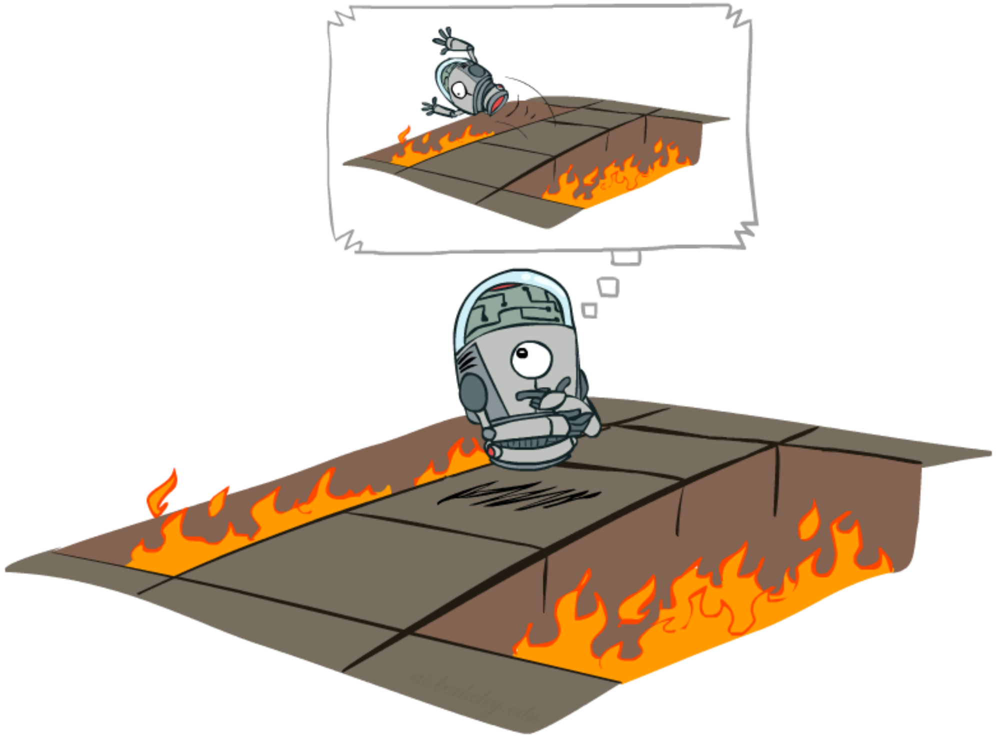
# Reinforcement Learning

- 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 out actions and states to learn

# Offline (MDPs) vs. Online (RL)



**Offline Solution:**

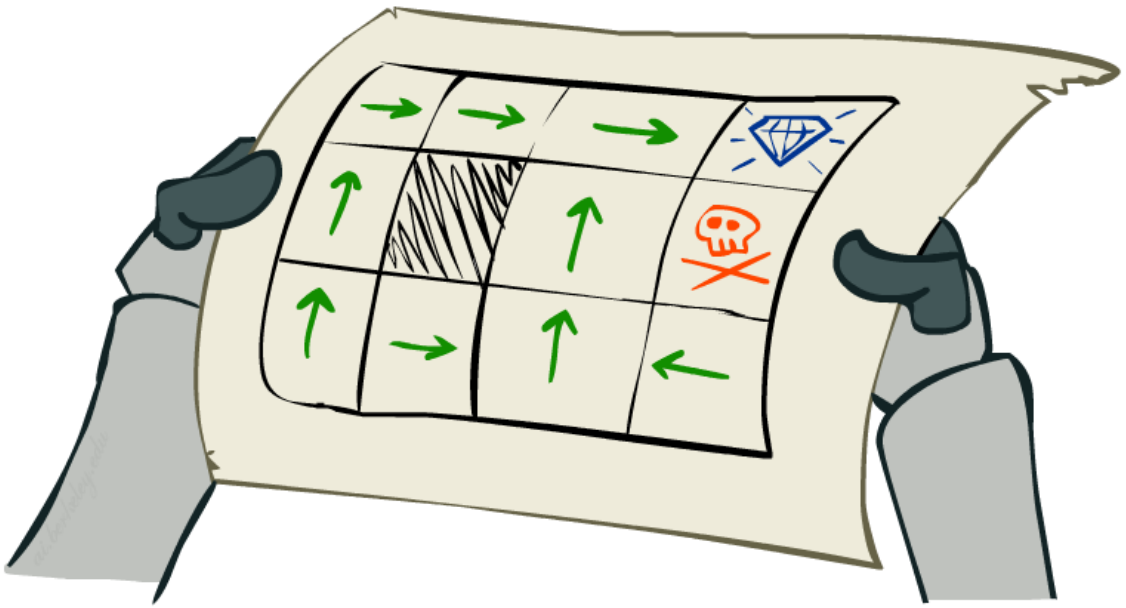
Compute policy ahead of time

**Online Learning:**

Compute policy as experience comes in

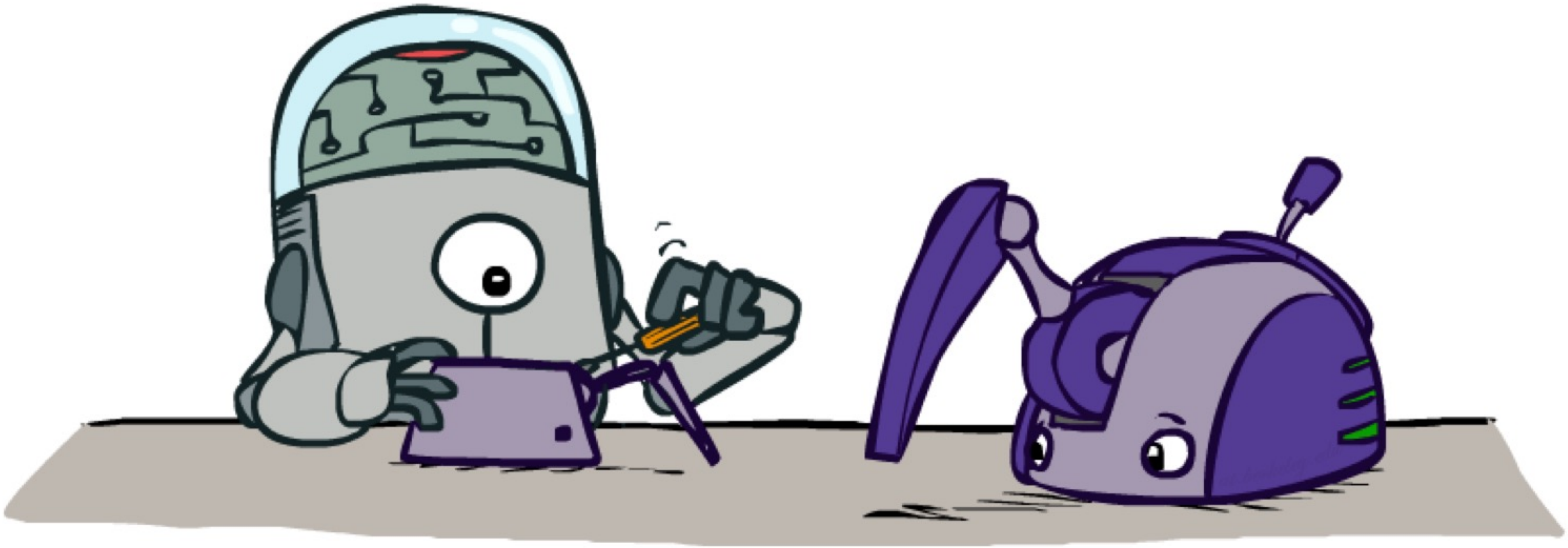
# Passive Reinforcement Learning

- 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



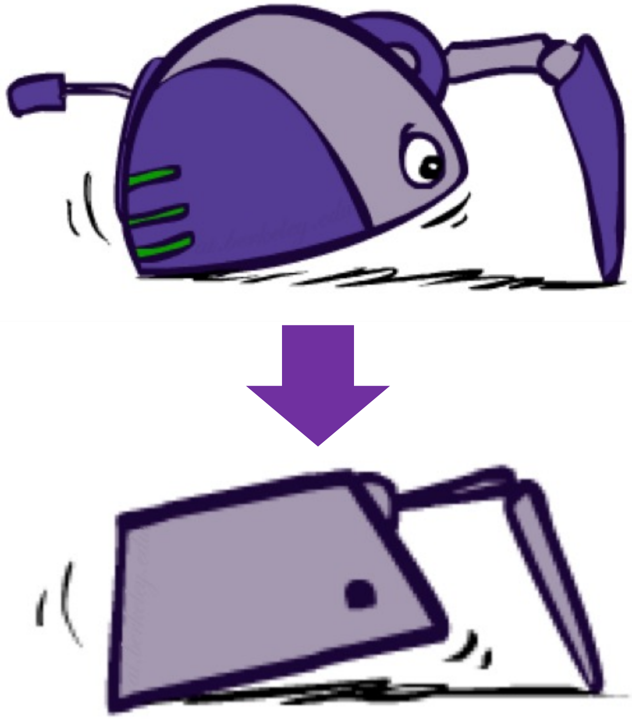
- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning!  You actually take actions in the world.

# 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

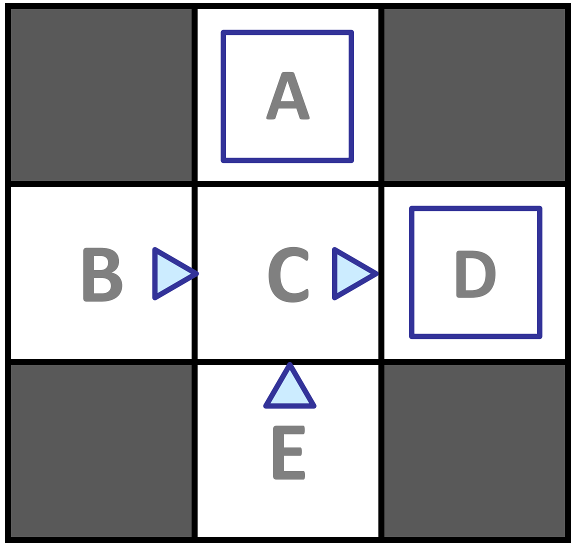


- **Step 1:** Learn empirical MDP model
  - Count outcomes s' for each s, a
  - Normalize to give an estimate of $\widehat{T}(s, a, s')$
  - Discover each $\widehat{R}(s, a, s')$ when we experience (s, a, s')

- **Step 2:** Solve the learned MDP
  - For example, use value iteration, as before

# Example: Model-Based Learning

## Input Policy π




*Assume:* $\gamma = 1$

## Observed (s, a, s', R) Transitions

**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, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
…

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

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
…

# Analogy: Expected Age

Goal: Compute expected age of cs188 students

**Known P(A)**

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

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

**Unknown P(A): "Model Based"**

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

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

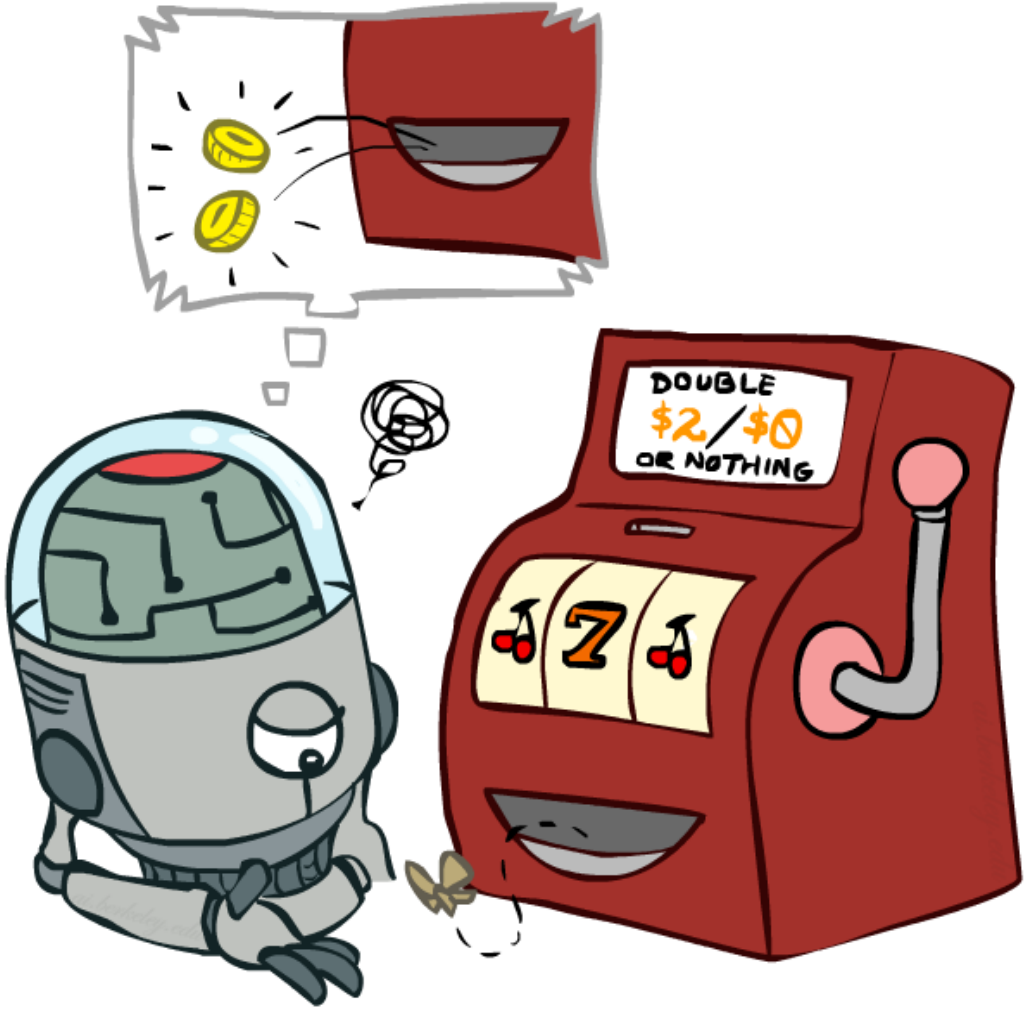
Why does this work? Because eventually you learn the right model.

**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



# 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 from that state until the end of the episode:

$$sample_i(s) = R(s) + \gamma R(s') + \gamma^2 R(s'') + \ \ldots$$

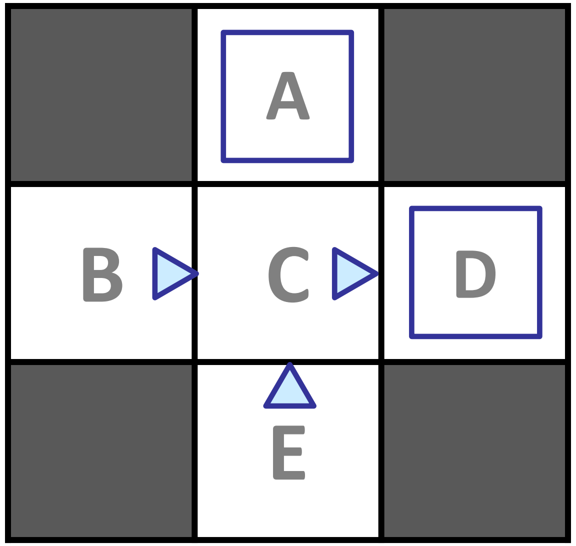
  - Average those samples:

$$V(s) \leftarrow \frac{1}{N} \sum_i sample_i(s)$$

- This is called *direct* or *Monte-Carlo evaluation*

# Example: Direct Evaluation

### Input Policy π




*Assume:* $\gamma = 1$

### Observed (s, a, s', R) Transitions

**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

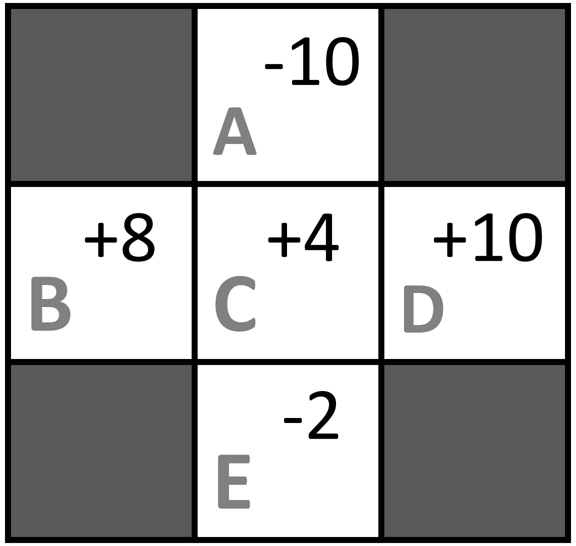
**Episode 3**

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



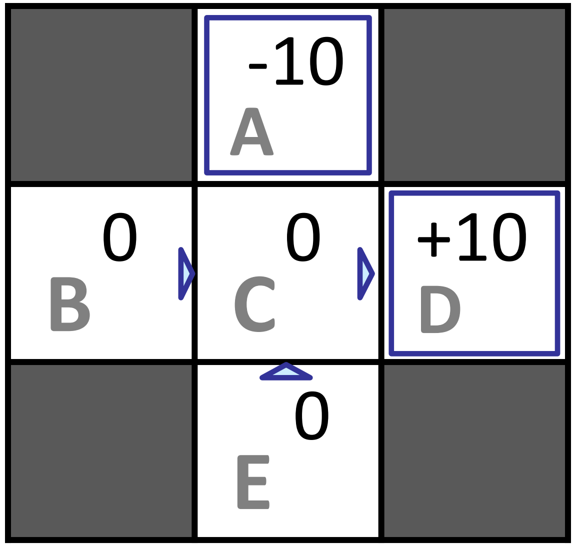

*V(s)* is sum of discounted rewards from *s* until the end, averaged over all encounters of *s*

# 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

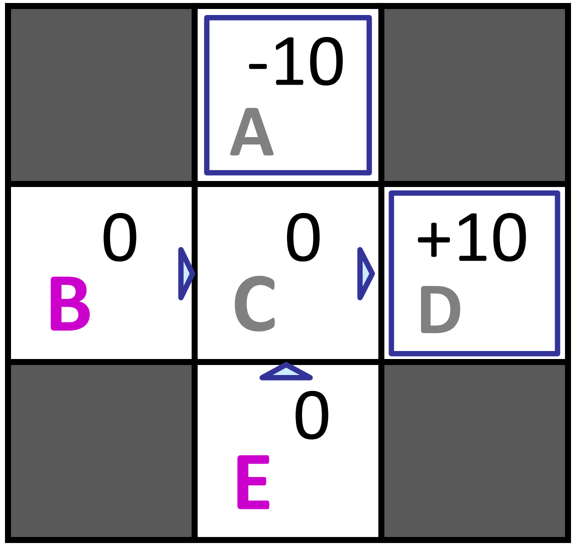
- What bad about it?
  - It wastes information about state connections
  - Need to have all episodes ahead of time (cannot "stream" in transitions)

Output Values




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

# Problems with Direct Evaluation




Observed (s, a, s', R) Transitions:

Episode 1

E, north, C, 0
C, east, D, 0
D, exit, x, +10

Episode 2

B, east, C, 0
C, east, A, 0
A, exit, x, -10

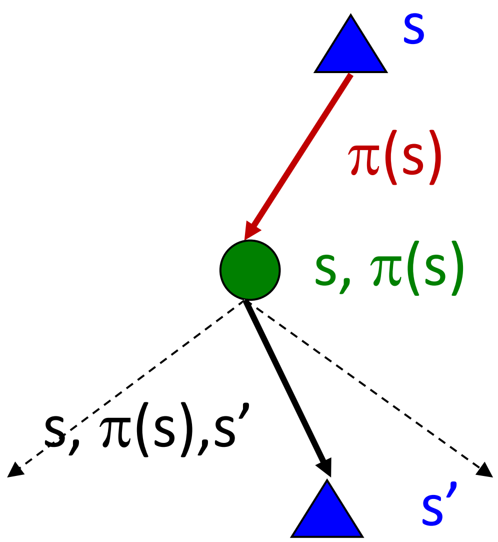
Is B a bad state?

# 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

$$sample_1 = R(s,\pi(s),s_1') + \gamma V_k^{\pi}(s_1')$$

$$sample_2 = R(s,\pi(s),s_2') + \gamma V_k^{\pi}(s_2')$$

$$\ldots$$

$$sample_n = R(s,\pi(s),s_n') + \gamma V_k^{\pi}(s_n')$$

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

Known P(A):

$$E[A] = \sum_a P(a) \cdot a$$

Unknown P(A): "Model Free"

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

# Sample-Based Policy Evaluation?

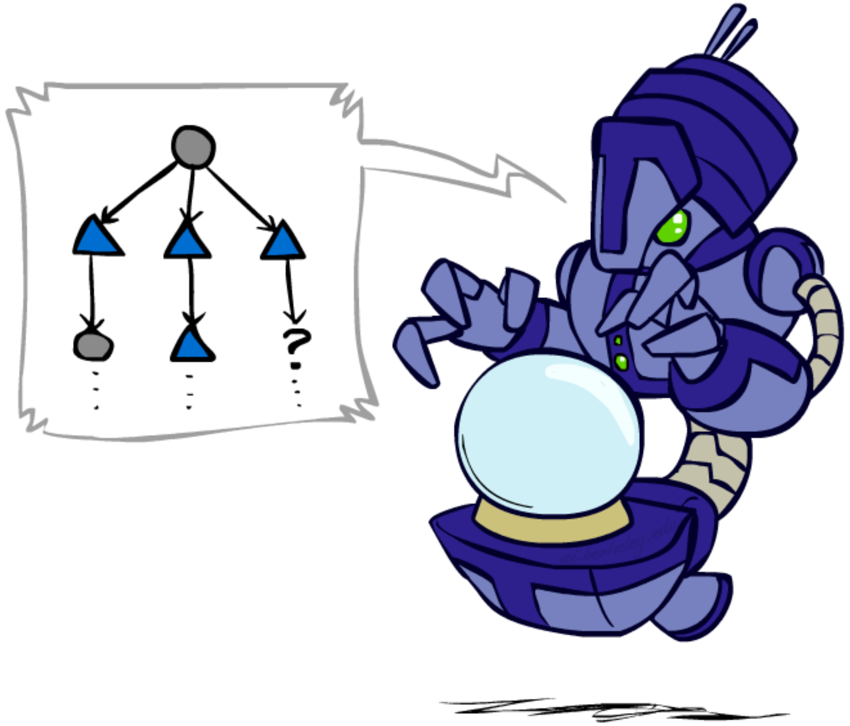
- We want to improve our estimate of V by computing these averages:

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

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

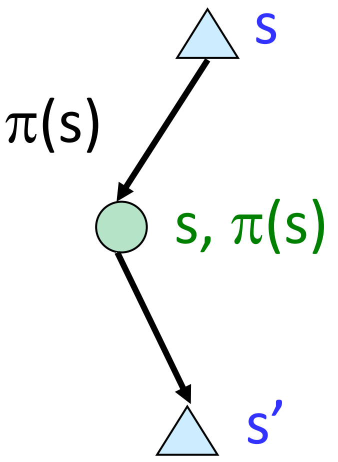
$$sample_1 = R(s,\pi(s),s_1') + \gamma V_k^{\pi}(s_1')$$
$$sample_2 = R(s,\pi(s),s_2') + \gamma V_k^{\pi}(s_2')$$
$$\ldots$$
$$sample_n = R(s,\pi(s),s_n') + \gamma V_k^{\pi}(s_n')$$

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



# 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): $V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + (\alpha) sample$ $\quad 0 < \alpha < 1$

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

# Exponential Moving Average

- Traditional Average: $AVG(x) = \frac{1}{N}\sum_n x_n$
  - Need to have all N samples at once (cannot “stream” in samples)

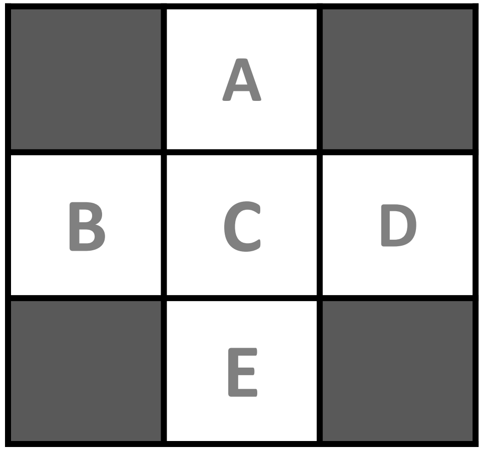
- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1-\alpha)\cdot\bar{x}_{n-1} + \alpha\cdot x_n$ $\quad 0 < \alpha < 1$
  - Makes recent samples more important:

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

  - Forgets about the past samples (how quickly depends on α)
- Decreasing learning rate α can give converging averages

# Example: Temporal Difference Learning

## States

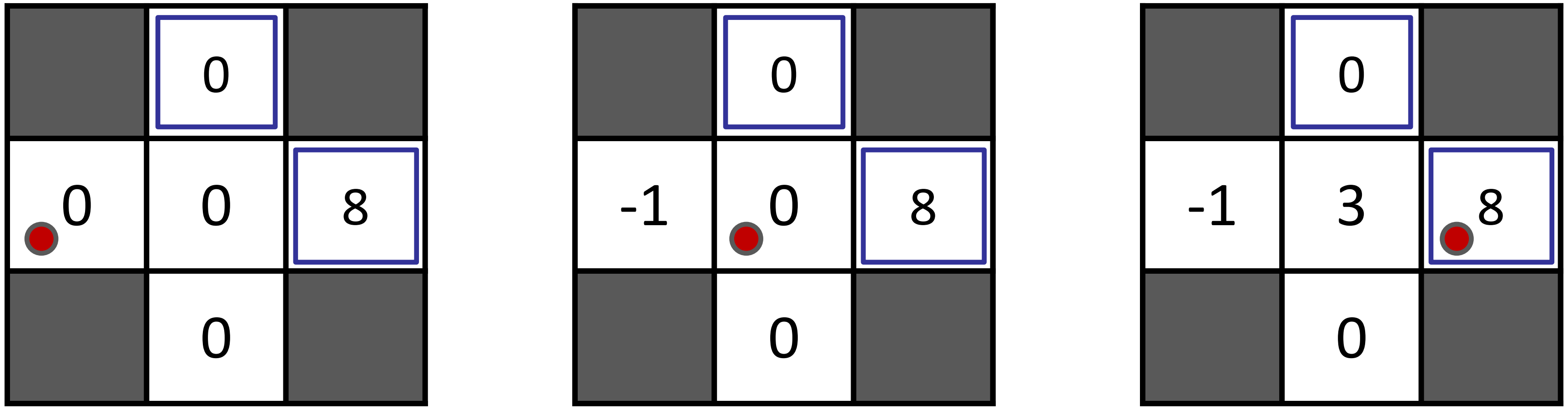



*Assume:* $\gamma$ = 1, $\alpha$ = 1/2

## Observed Transitions

B, east, C, -2

C, east, D, -2

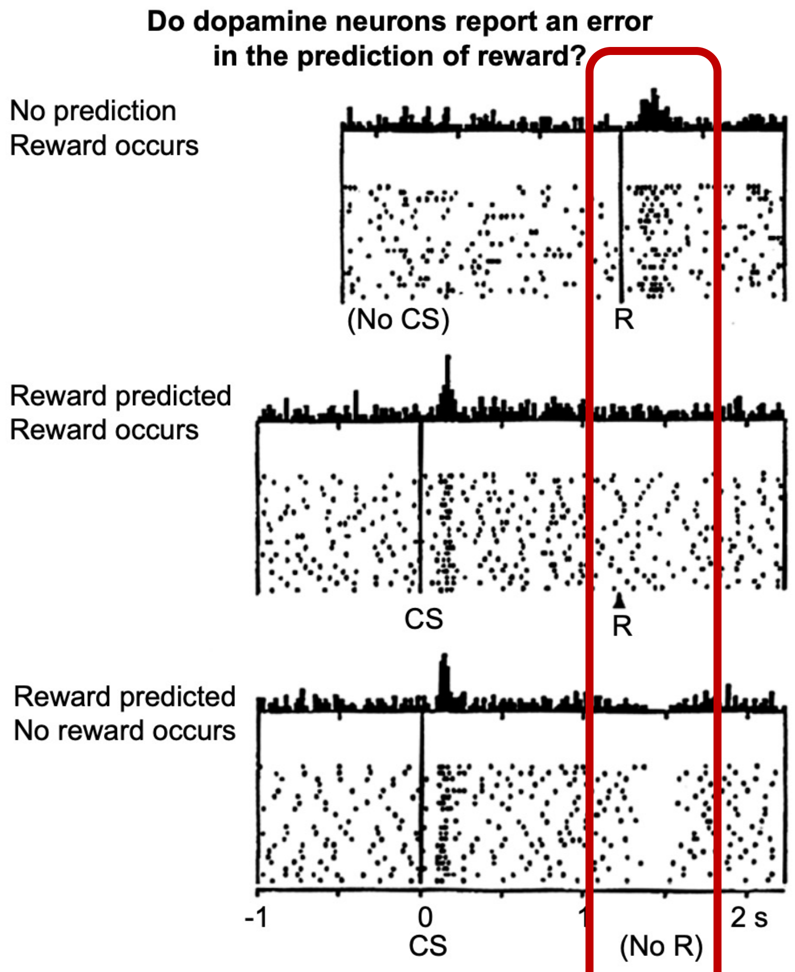



$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

# TD Learning Happens in the Brain!

- Neurons transmit *Dopamine* to encode reward or value prediction error
  $$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\boxed{(\textcolor{green}{sample} - V^{\pi}(s))}$$

- Example of Neuroscience & AI informing each other




positive

zero

negative

[A Neural Substrate of Prediction and Reward. Schultz, Dayan, Montague. 1997]

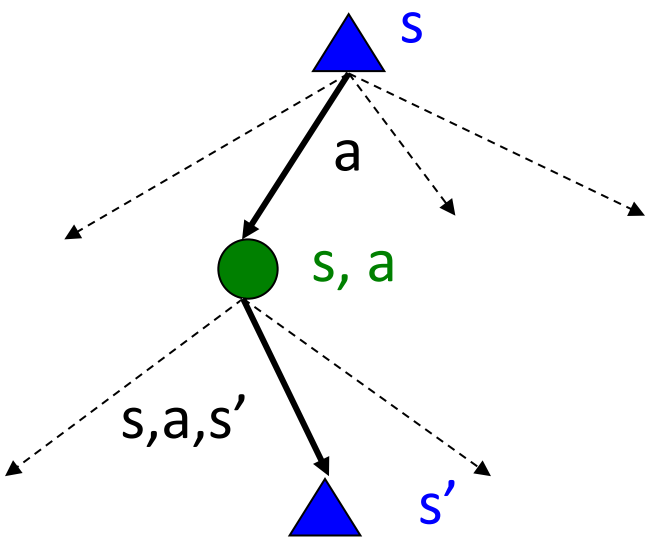
# Problems with TD Value Learning

- TD value leaning is a model-free way to do **policy evaluation**

- However, if we want to turn values into a (new) policy, we're stuck:

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

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




- What can we do?
  - Learn Q-values, not values
  - Makes action selection model-free too!

# 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:

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

- 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:

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

# Q-Learning

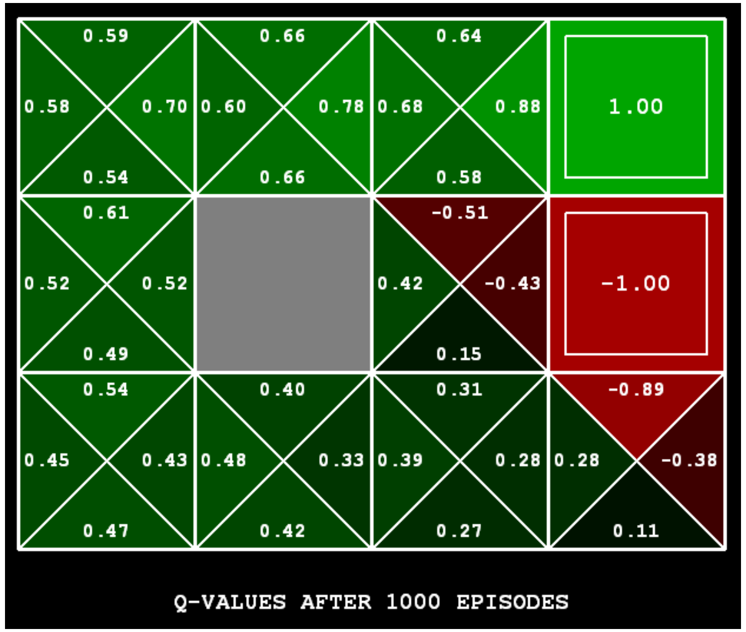
- Q-Learning: sample-based Q-value iteration

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

- 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:

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

no longer policy evaluation!




  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$

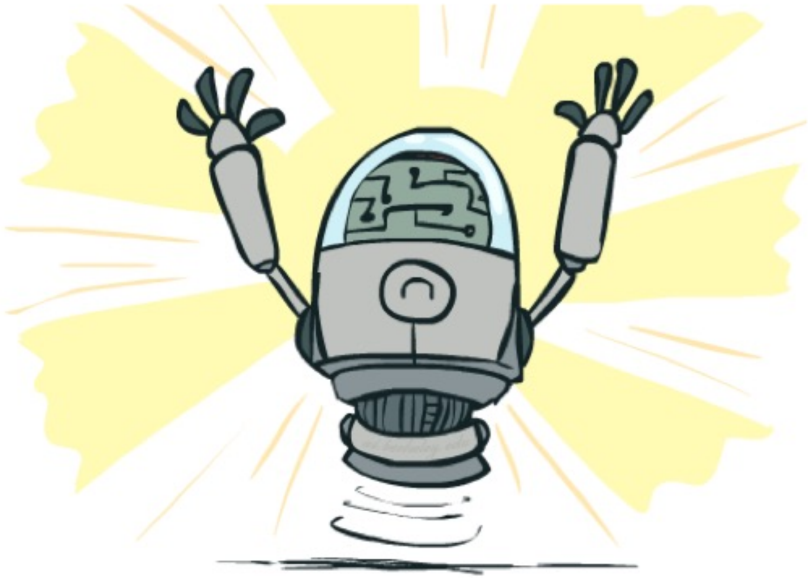
[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

# Video of Demo Q-Learning -- Gridworld

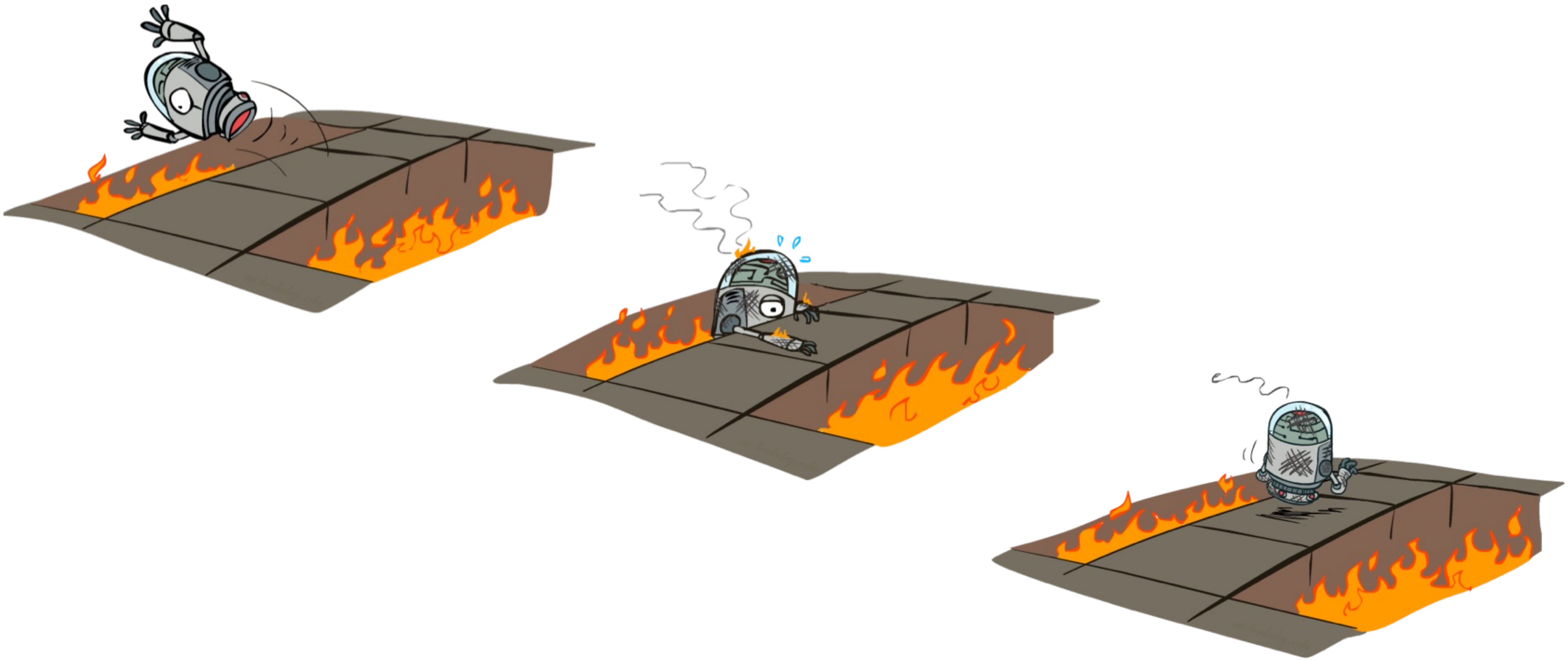
# Video of Demo Q-Learning -- Crawler

# 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 (!)

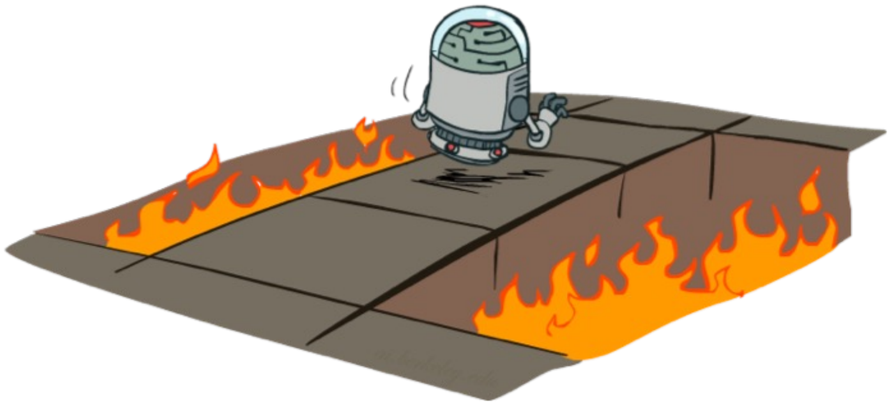


# Active Reinforcement Learning



# 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…

# What we did today (a lot!)

- Focused on *Passive* Reinforcement Learning problem
  - How to learn from already given experiences when we don't know T and R
- Saw distinction between *model-based* and *model-free* approaches to RL
  - *Model-Based:* Learn a model of T and R from experiences, then solve MDP
  - *Model-Free:* Learn from experience *samples* without building a model
- *Direct evaluation* was our first attempt at model-free value learning
  - Estimate values from samples of discounted sums of rewards: $\text{sample} = R(s) + \gamma R(s') + \gamma^2 R(s'') + \ \ldots$
  - **Issue 1:** Does not take advantage of state connections
  - **Issue 2:** Needs to see all transitions at once
- Introduced *TD Learning* as a way to address two issues above
  - **Solution 1:** Use V(s) when calculating value samples: $\text{sample} = R(s) + \gamma V^{\pi}(s')$
  - **Solution 2:** Use *Exponential Moving Average* to build up averages one transition at a time
  - **New issue:** TD Learning only learns state values – can't use it to pick optimal actions!
- Solution is *Q-Learning:* learn Q values instead of V with TD-like update
  - Now can pick *optimal* actions, so get an optimal model-free policy

# Next Time: Active & Approximate RL!