

# Ve492: Introduction to Artificial Intelligence

## Reinforcement Learning I

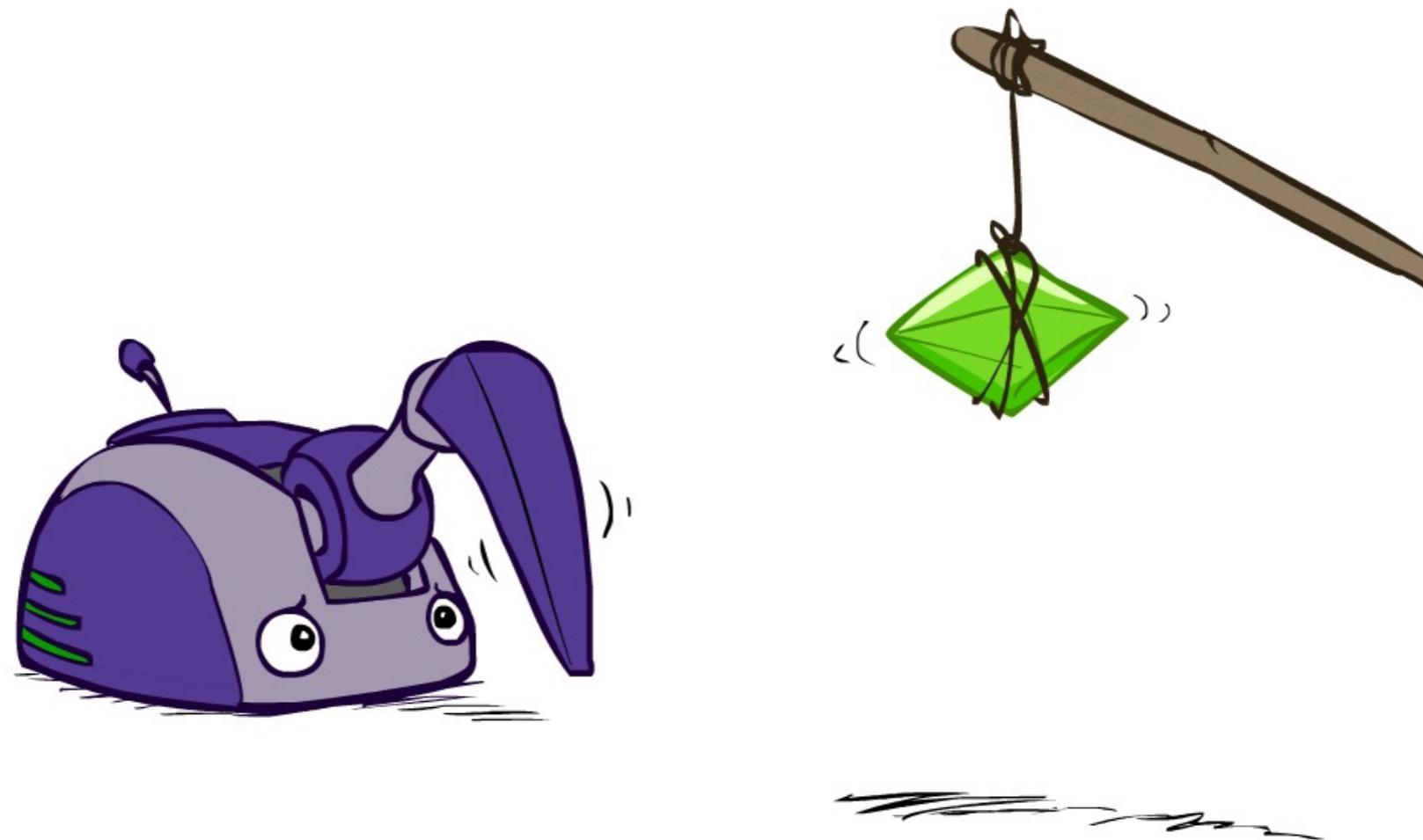


Paul Weng

UM-SJTU Joint Institute

Slides adapted from <http://ai.berkeley.edu>, AIMA, UM, CMU

# Reinforcement Learning



# Example: Prescription Problem



$P(\text{cure}) = 0.2$



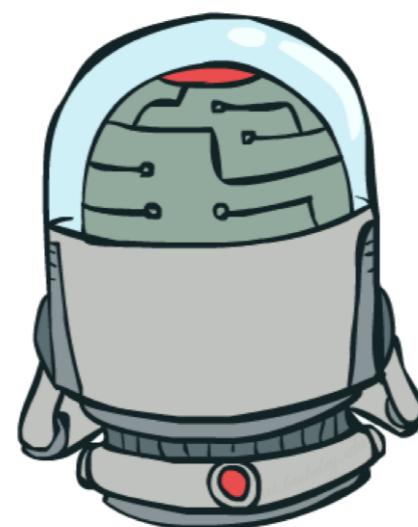
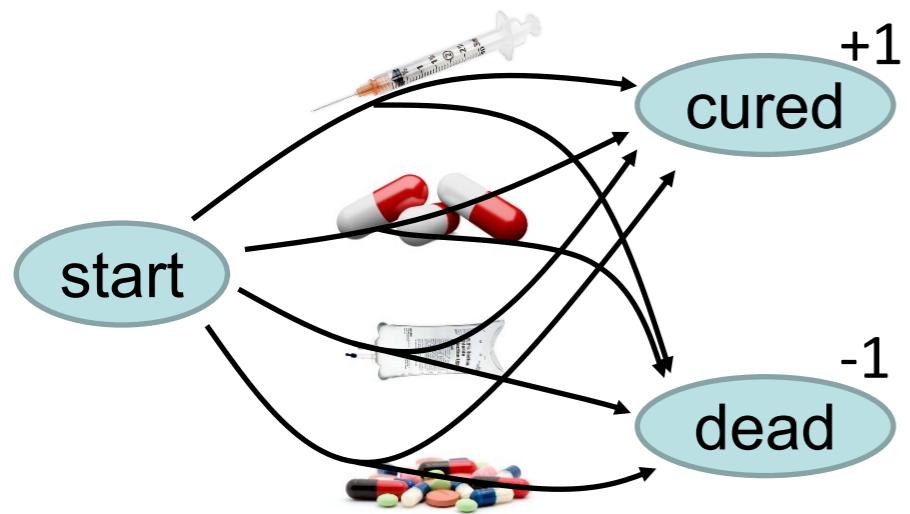
$P(\text{cure}) = 0.4$



$P(\text{cure}) = 0.9$



$P(\text{cure}) = 0.1$



# Example: Prescription Problem



$P(\text{cure}) = ?$



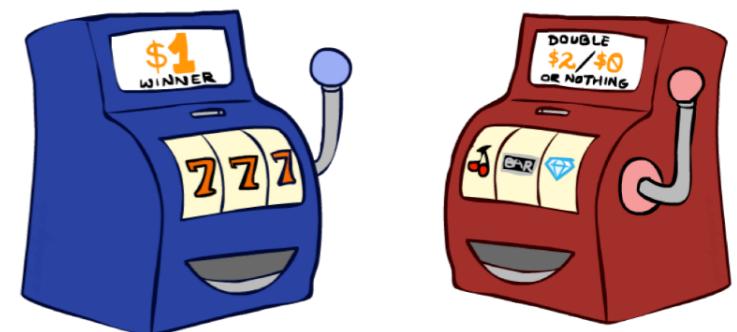
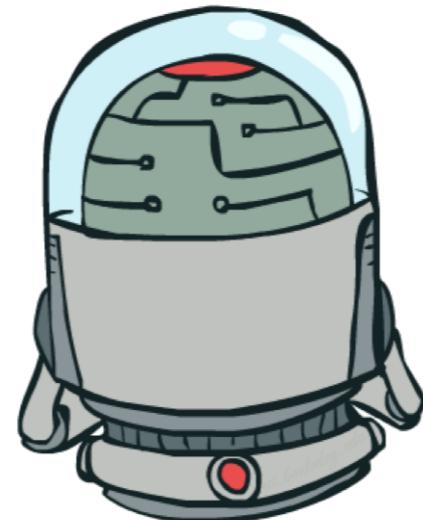
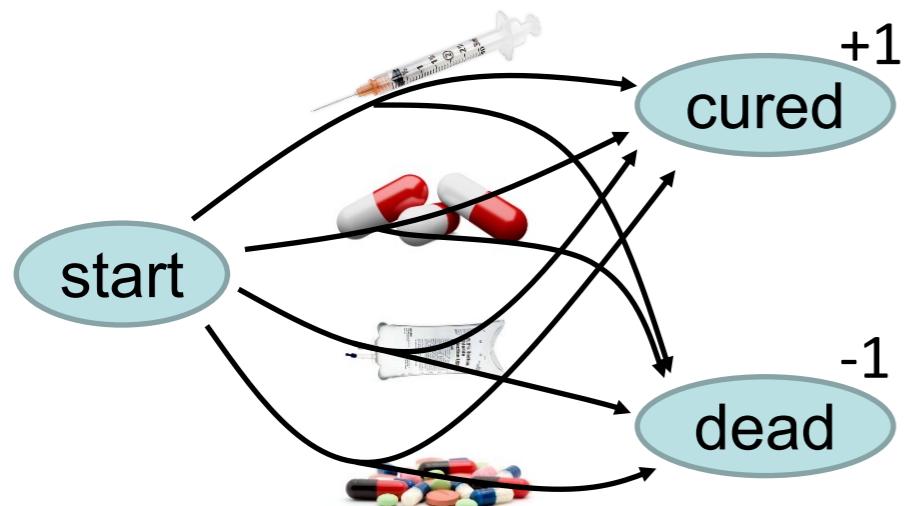
$P(\text{cure}) = ?$



$P(\text{cure}) = ?$



$P(\text{cure}) = ?$



# Let's Play!



$P(\text{cure}) = ?$



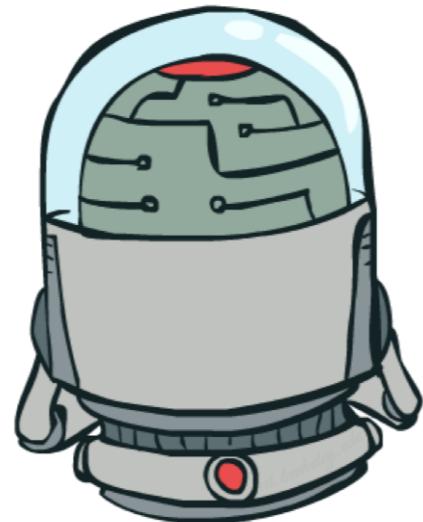
$P(\text{cure}) = ?$



$P(\text{cure}) = ?$



$P(\text{cure}) = ?$

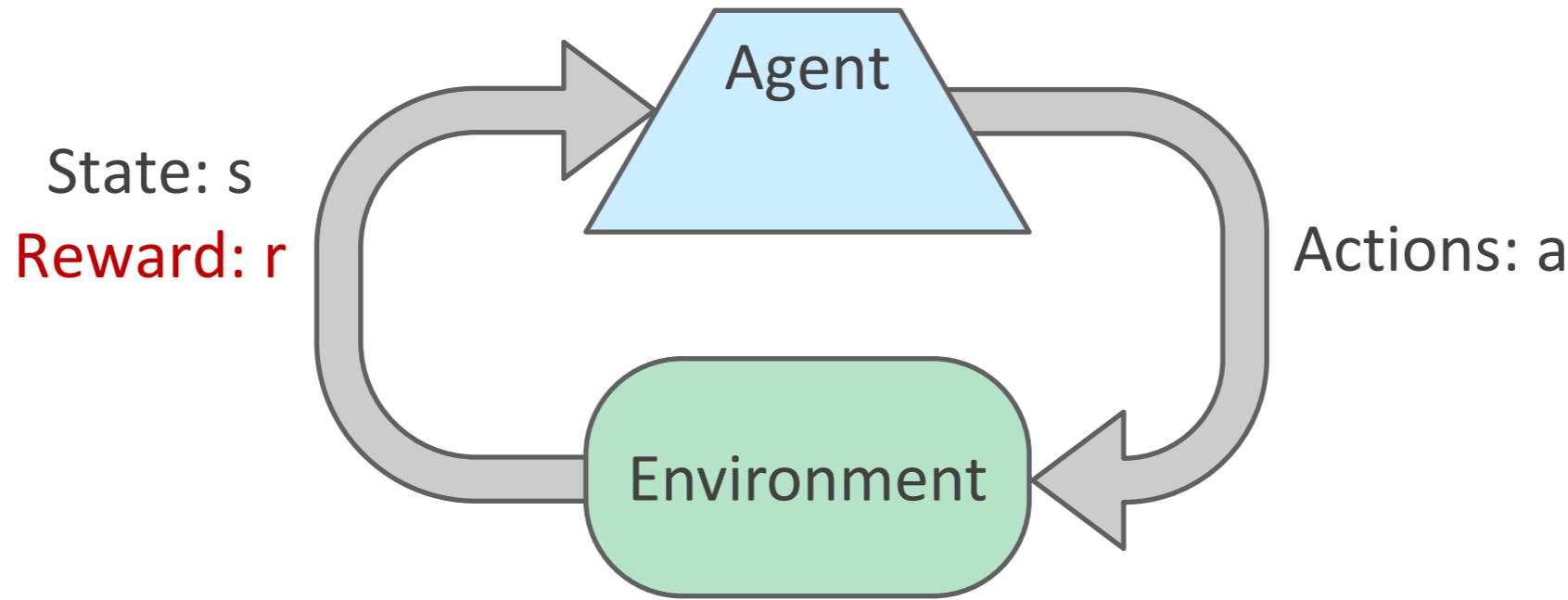


# What Just Happened?

- ❖ **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

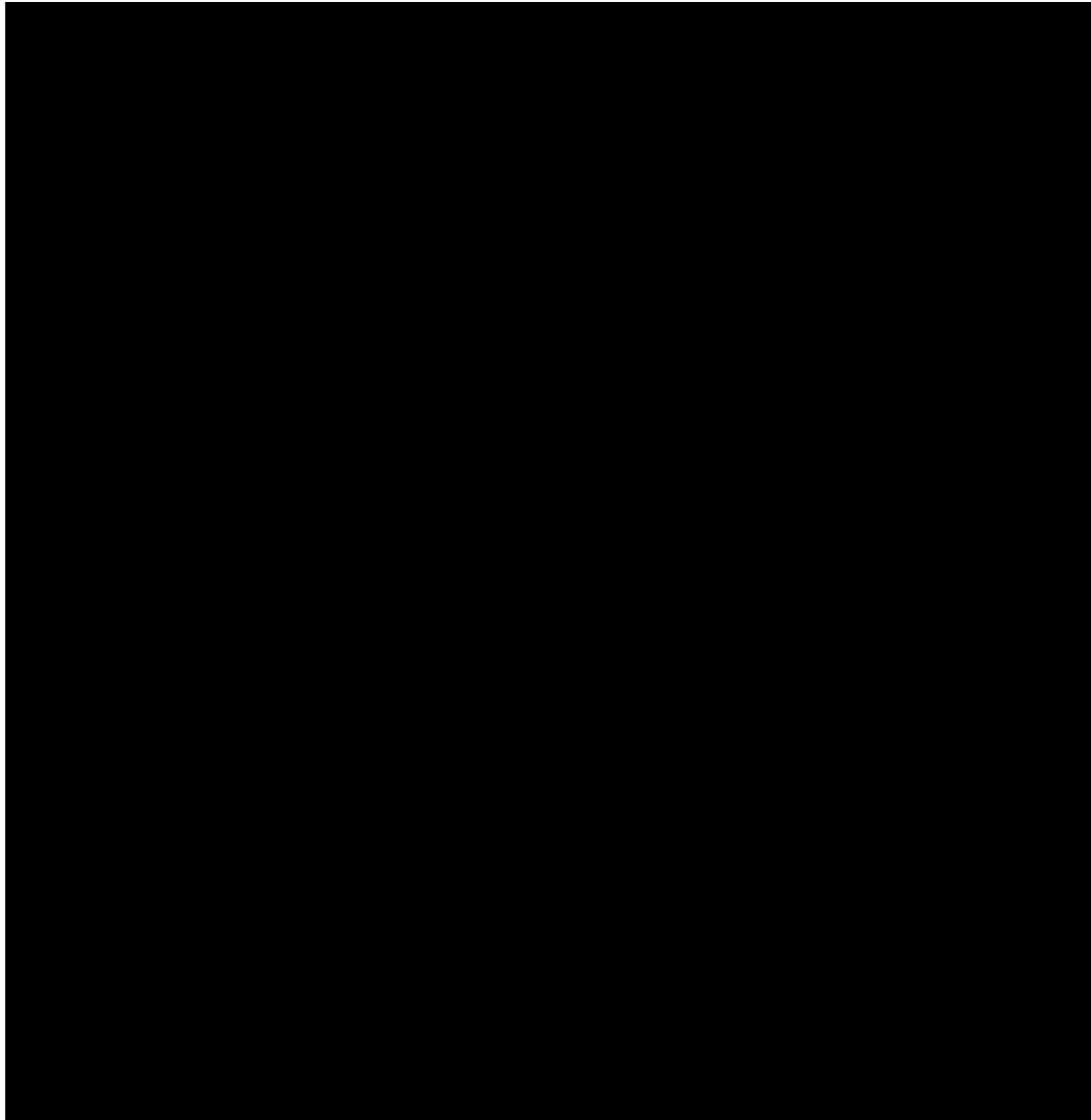


- ❖ Basic idea:
  - ❖ Receive feedback in the form of rewards
  - ❖ Agent's utility is defined by the reward function
  - ❖ Must (learn to) act so as to maximize expected rewards
  - ❖ All learning is based on observed samples of outcomes!

---

# Example: Learning to Walk

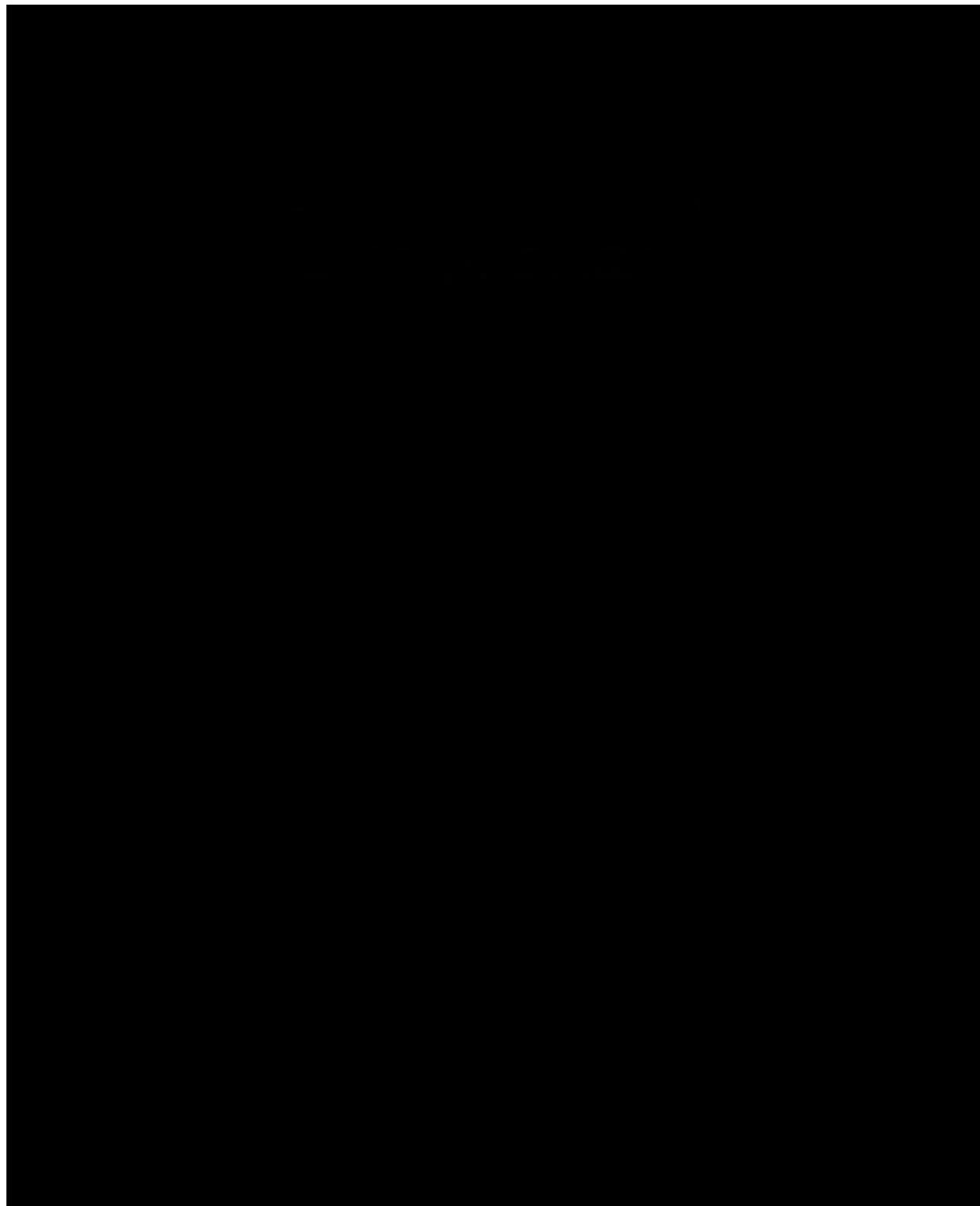
---



---

# Example: Atari

---



# Example: Robots



# Reinforcement Learning

- ❖ Still assume a Markov decision process (MDP):

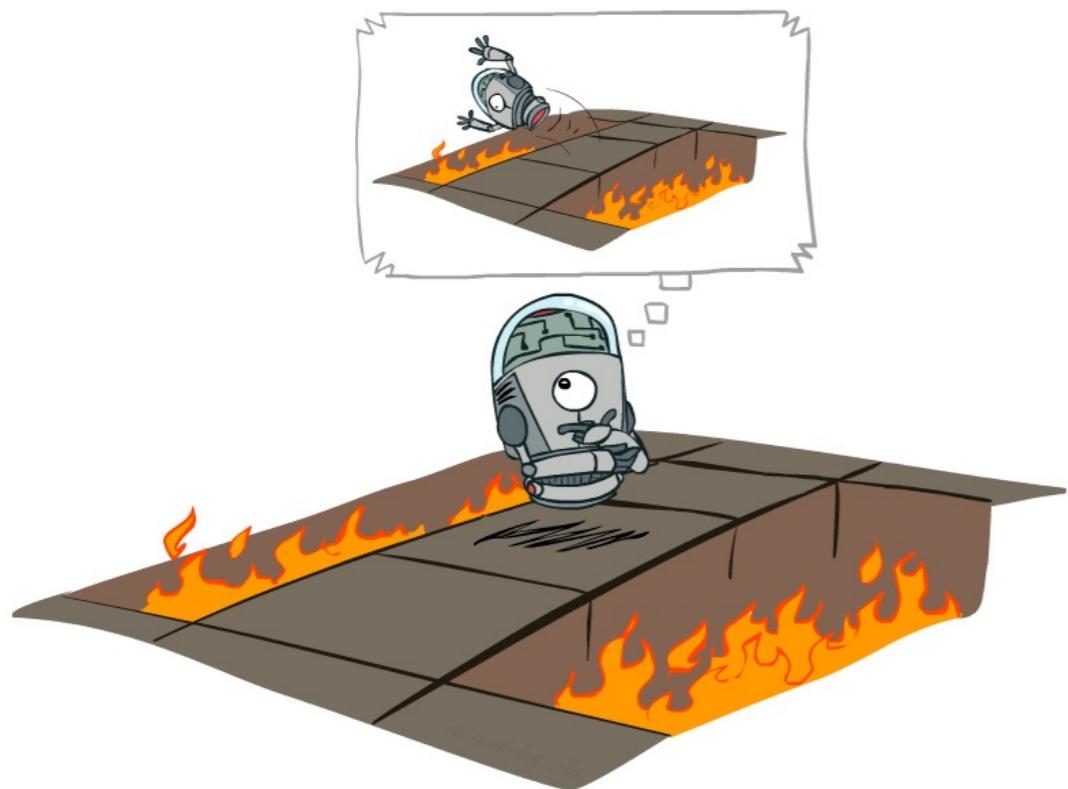
- ❖ A set of states  $s \in S$
- ❖ A set of actions (per state)  $A$
- ❖ A model  $T(s,a,s')$
- ❖ A reward function  $R(s,a,s')$

- ❖ 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)

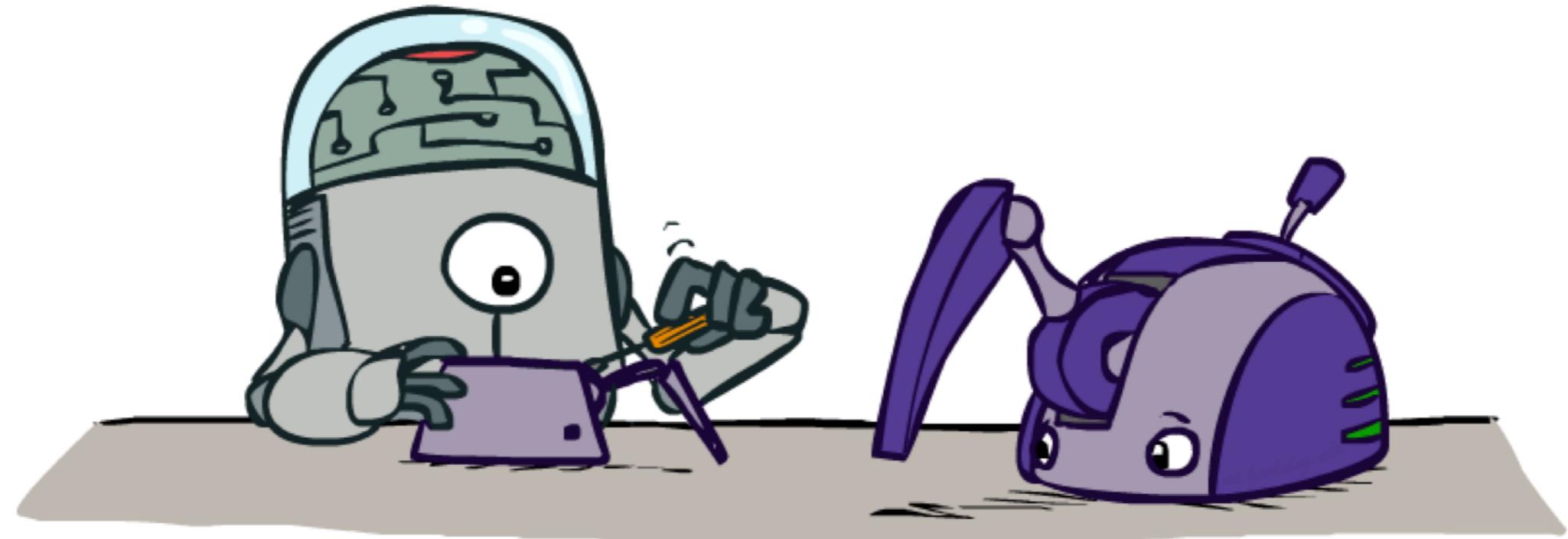


Offline  
Solution



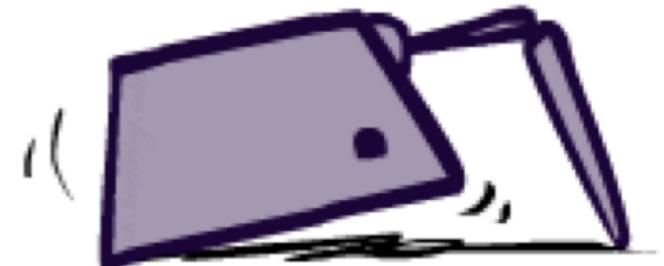
Online  
Learning

# Model-Based Learning



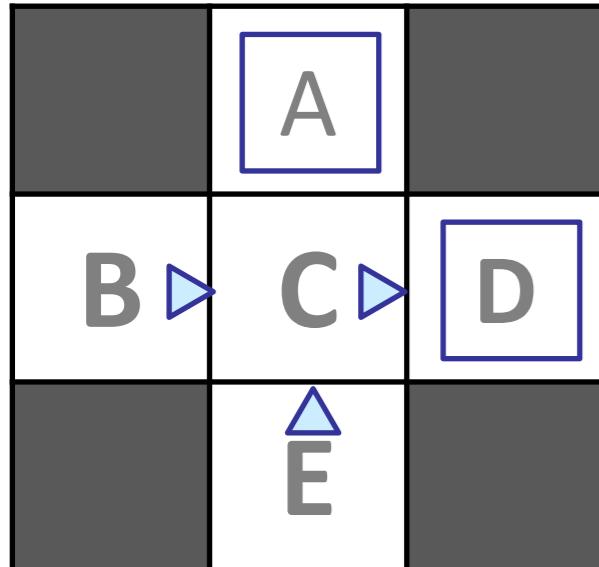
# Model-Based Learning

- ❖ 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  $T(s, a, s')$
  - ❖ Discover each  $R(s, a, s')$  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 Travel Time

Goal: Compute expected travel time from home to school

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.

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

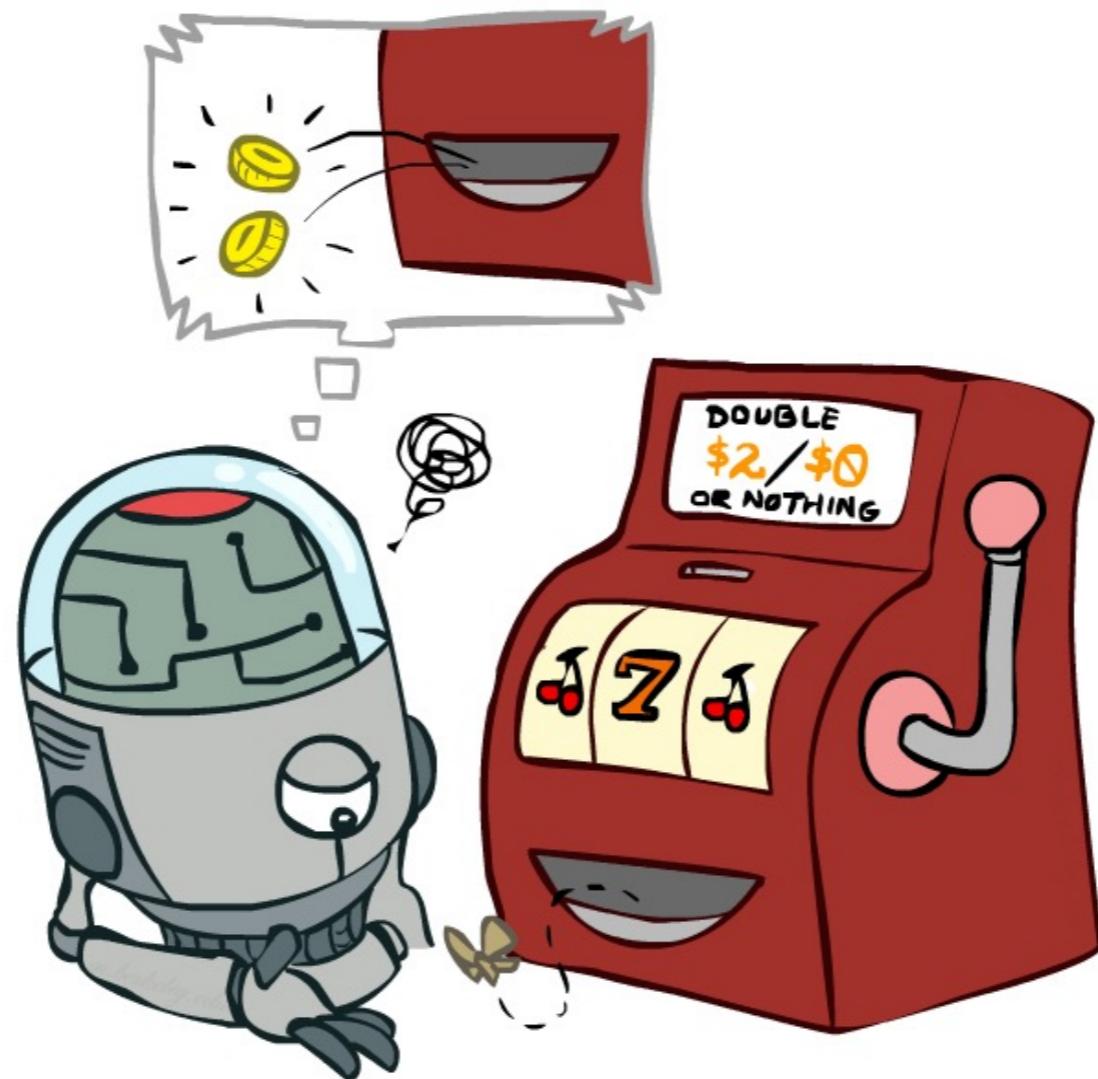
$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Unknown  $P(A)$ : “Model Free”

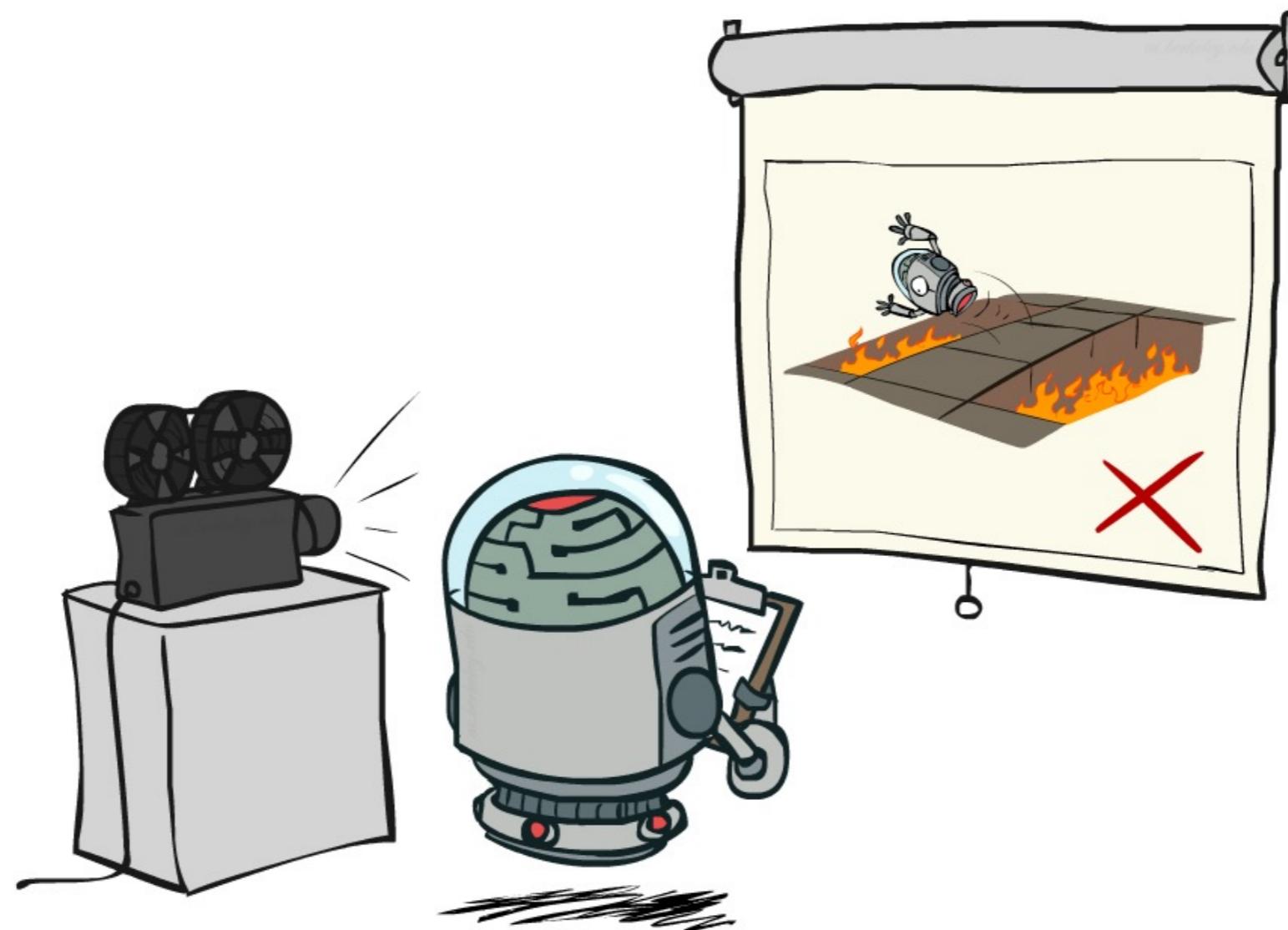
$$E[A] \approx \frac{1}{N} \sum_i a_i$$

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

# Model-Free Learning



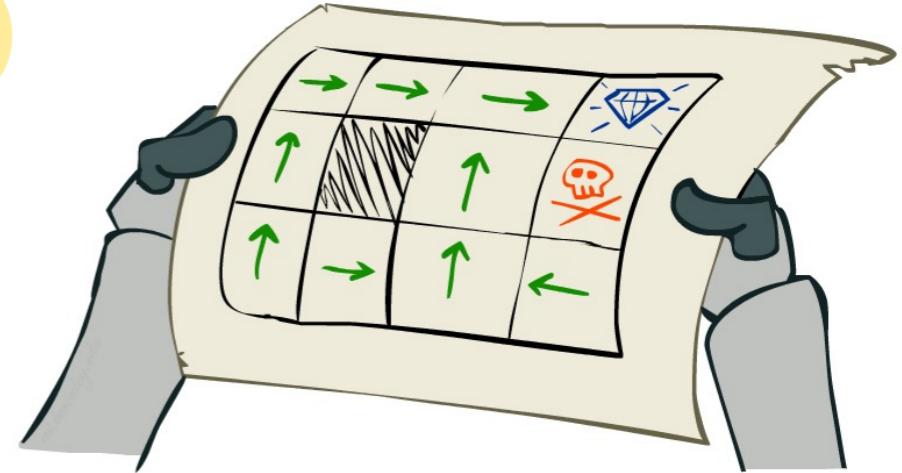
# Passive Reinforcement Learning



# Passive 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.

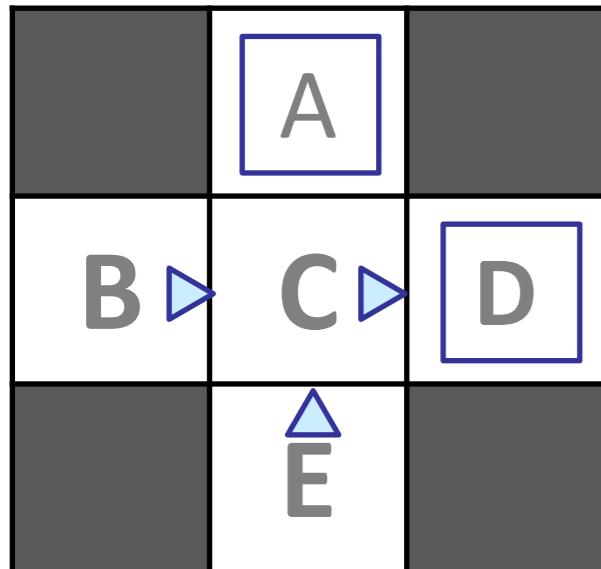
# Direct Evaluation

- ❖ 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<br>A |          |
| +8<br>B | +4<br>C  | +10<br>D |
|         | -2<br>E  |          |

# 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's 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

|         |          |          |
|---------|----------|----------|
|         | -10<br>A |          |
| +8<br>B | +4<br>C  | +10<br>D |
|         | -2<br>E  |          |

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

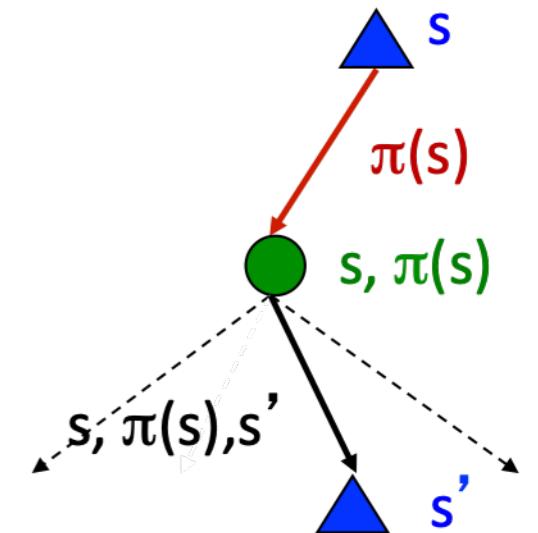
# Why Not Use Policy Evaluation?

- ❖ 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 do 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

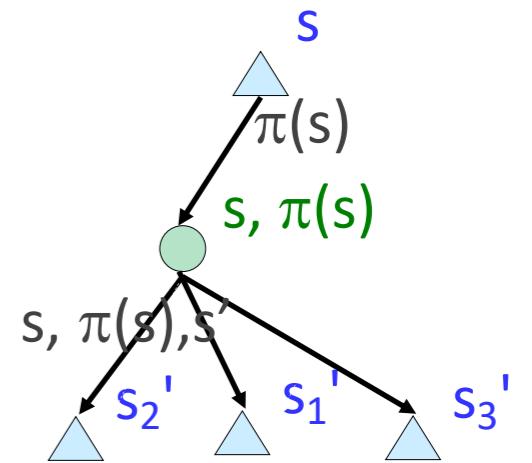
$$\text{sample}_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$

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

...

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

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



Almost! But we  
can't rewind time to  
get sample after  
sample from state  $s$ .

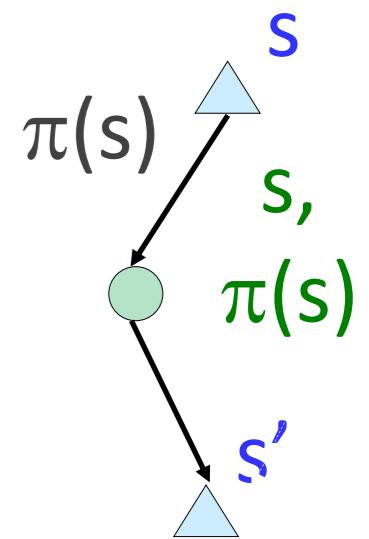
# Temporal Difference Learning



# 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

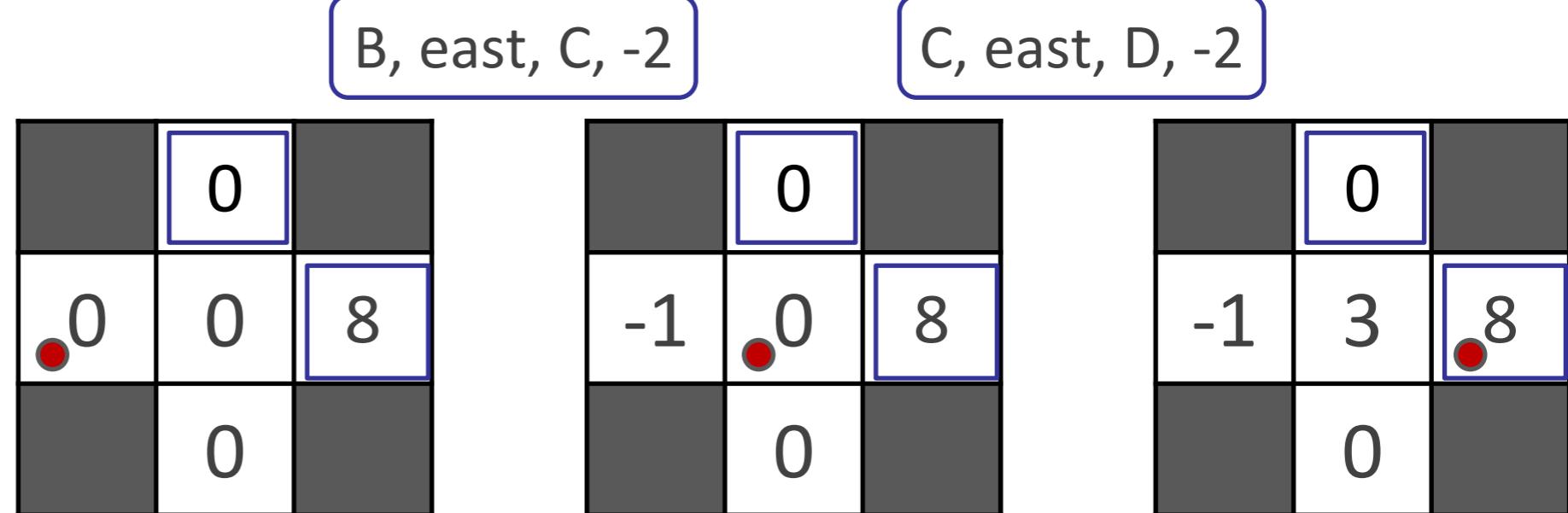
# Example: Temporal Difference Learning

States

|   |   |   |
|---|---|---|
|   | A |   |
| B | C | D |
|   | E |   |

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

Observed Transitions



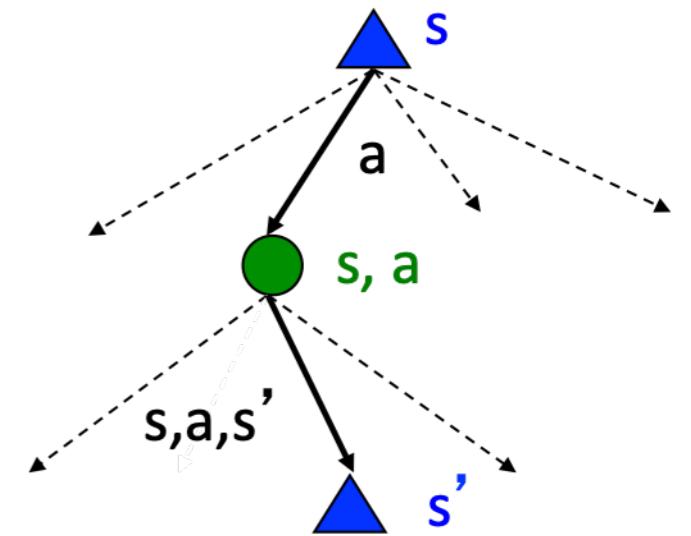
$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

# 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:

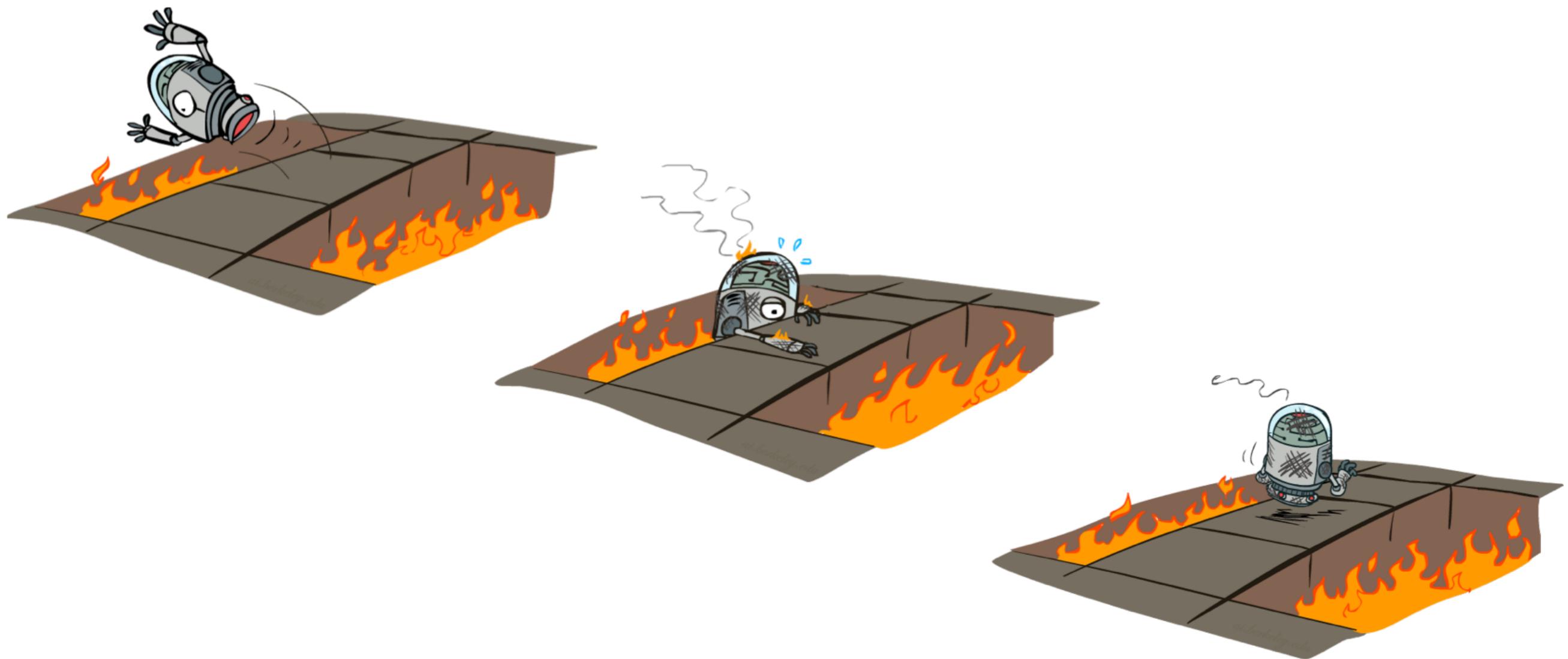
$$\pi(s) = \arg \max_a Q(s, a)$$

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



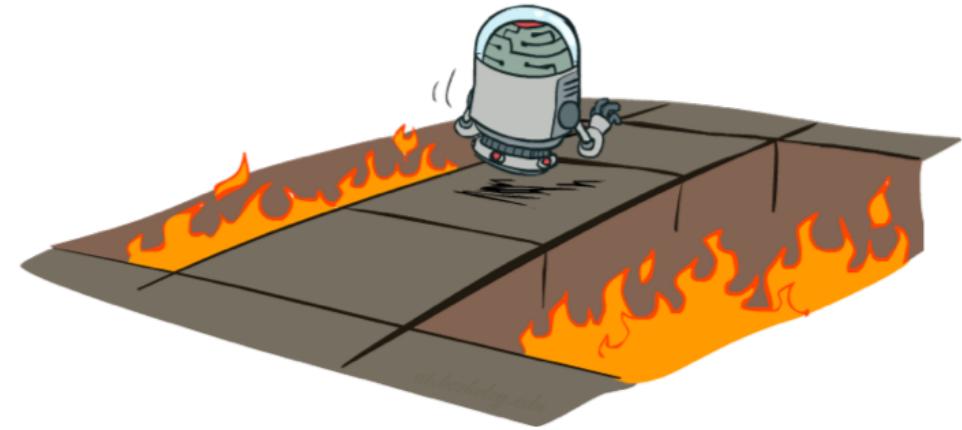
- ❖ Idea: learn Q-values, not values
- ❖ Makes action selection model-free too!

# 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

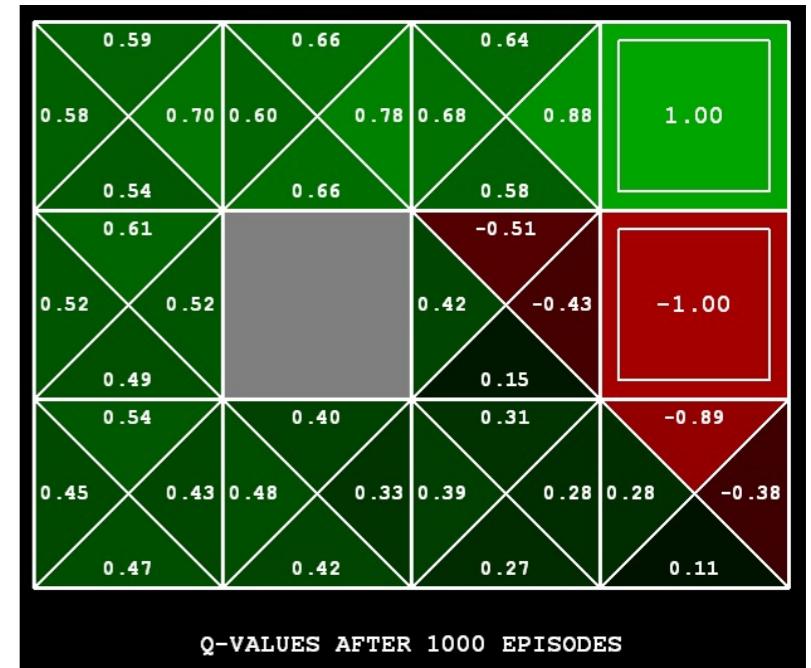
- ❖ Q-Learning: sample-based Q-value iteration

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

- ❖ 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:

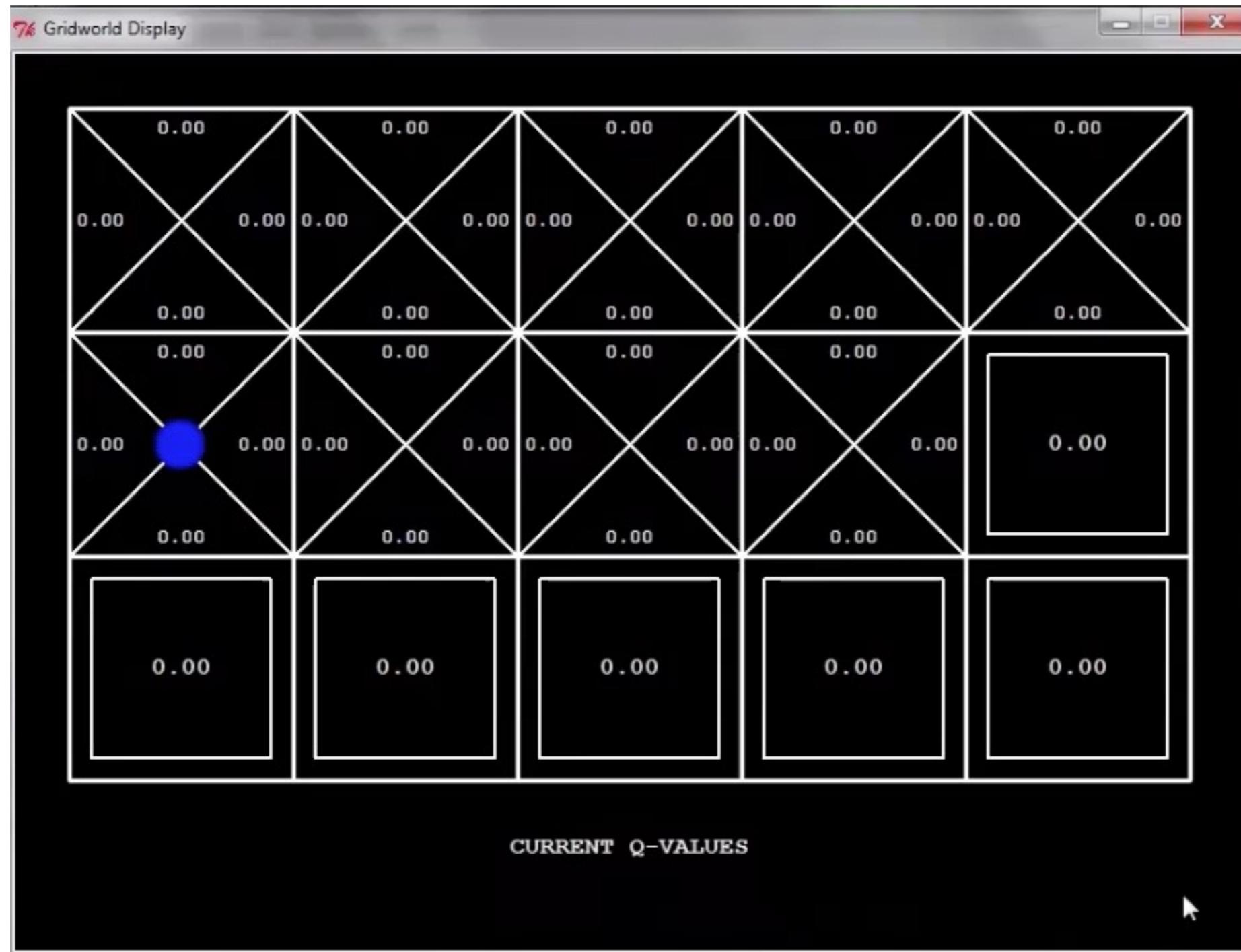
$$\text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$



- ❖ Incorporate the new estimate into a running average:

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

# Video of Demo Q-Learning -- Gridworld



# Q-Learning Principles

- ❖ Most classic model-free RL algorithm
- ❖ Learn  $Q^*(s, a)$  from observed transitions  $s, a, r, s'$

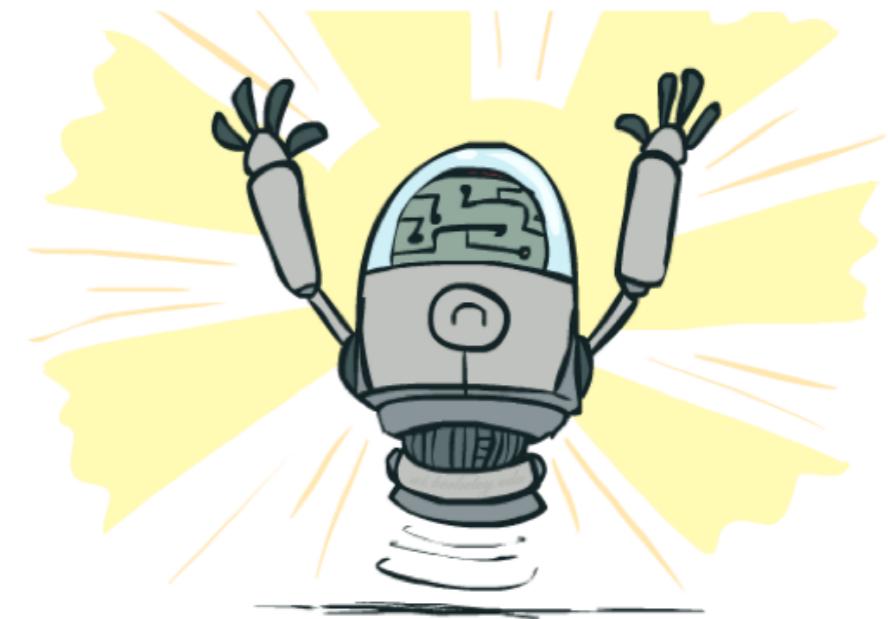
$$sample = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

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

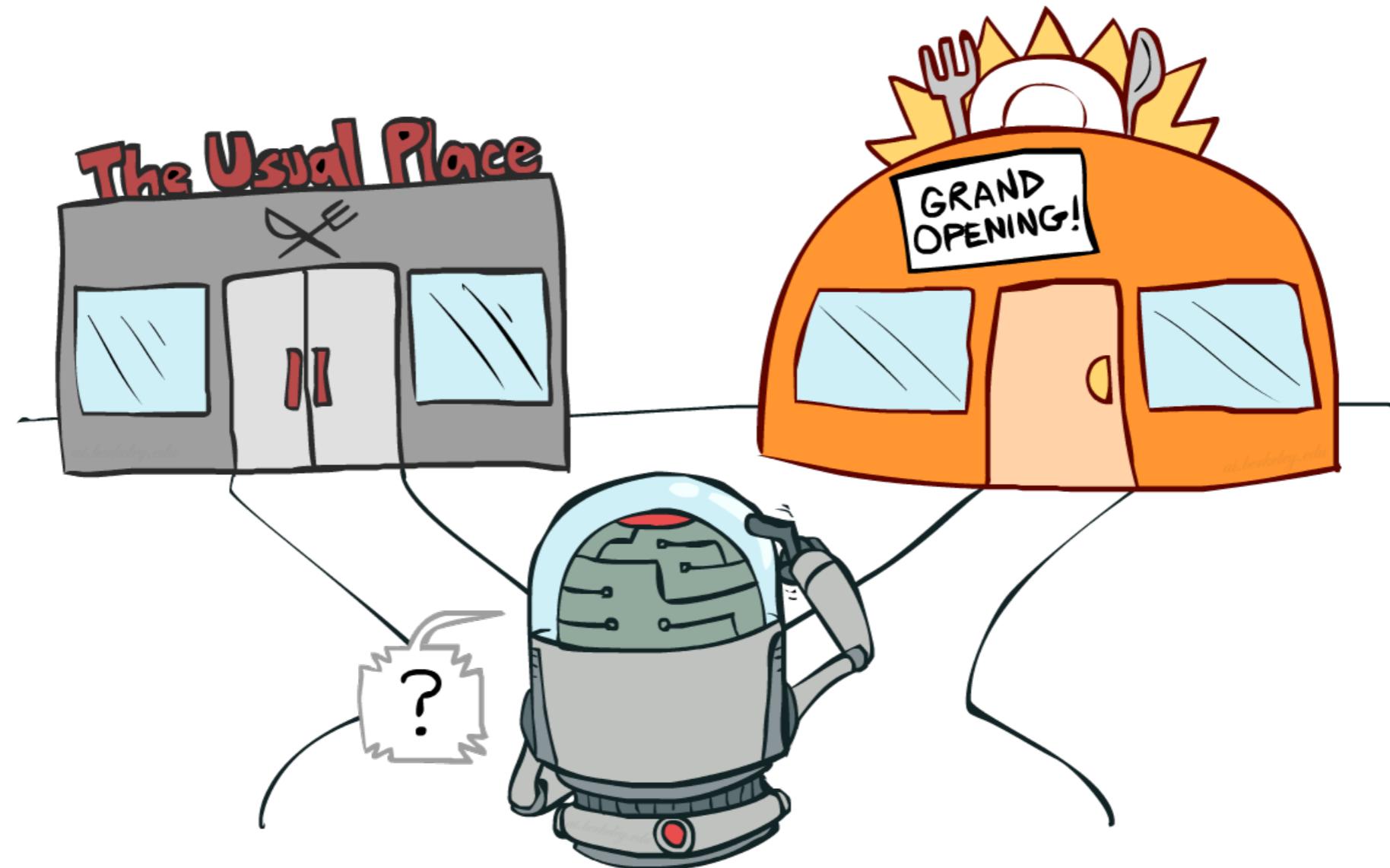
- ❖ Exploit by using greedy policy from  $Q(s, a)$
- ❖ Need to explore as  $T, R$  are unknown

# 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 (!)



# Exploration vs. Exploitation

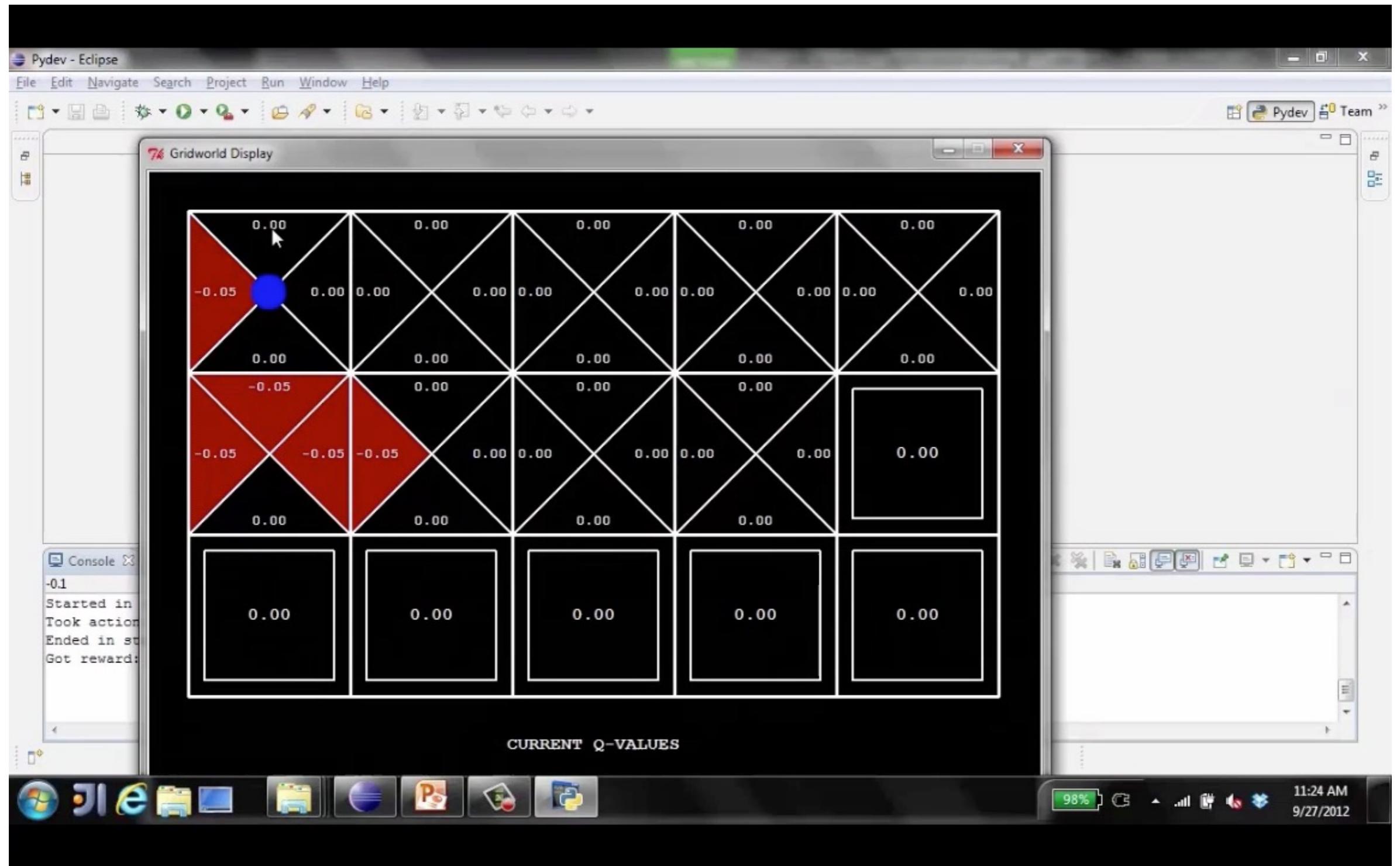


# How to Explore?

- ❖ Several schemes for forcing exploration
  - ❖ Simplest: random actions ( $\epsilon$ -greedy)
    - ❖ Every time step, flip a coin
    - ❖ With (small) probability  $\epsilon$ , act randomly
    - ❖ With (large) probability  $1-\epsilon$ , act on current policy
  - ❖ Problems with random actions?
    - ❖ You do eventually explore the space, but keep thrashing around once learning is done
    - ❖ One solution: lower  $\epsilon$  over time



# Video of Demo Q-Learning Auto Cliff Grid



# Quiz: Convergence of Q-learning

---

- ❖ Would Q-learning converge if random actions are chosen 10% of the times?
- ❖ Would Q-learning converge if random actions are chosen 50% of the times?
- ❖ Would Q-learning converge if random actions are chosen 100% of the times?

# Exploration Functions

## ❖ When to explore?

- ❖ Random actions: explore a fixed amount
- ❖ Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

## ❖ Exploration function

- ❖ Takes a value estimate  $u$  and a visit count  $n$ , and returns an optimistic utility, e.g.  $f(u, n) = u + k/n$

Regular Q-Update:  $Q(s, a) \leftarrow_{\alpha} R(s, a, s') + \gamma \max_{a'} Q(s', a')$

Modified Q-Update:  $Q(s, a) \leftarrow_{\alpha} R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s', a'))$

- ❖ Note: this propagates the “bonus” back to states that lead to unknown states as well!



# Regret

- ❖ Even if you learn the optimal policy, you still make mistakes along the way!
- ❖ Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- ❖ Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- ❖ Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret

