CSE 473: Introduction to Artificial Intelligence

Hanna Hajishirzi

Reinforcement Learning
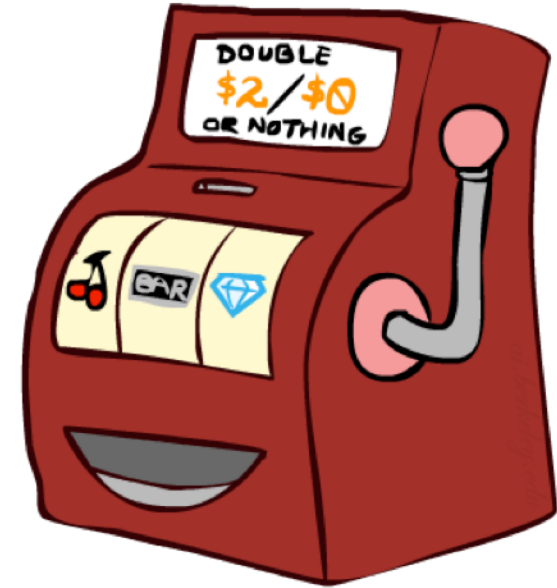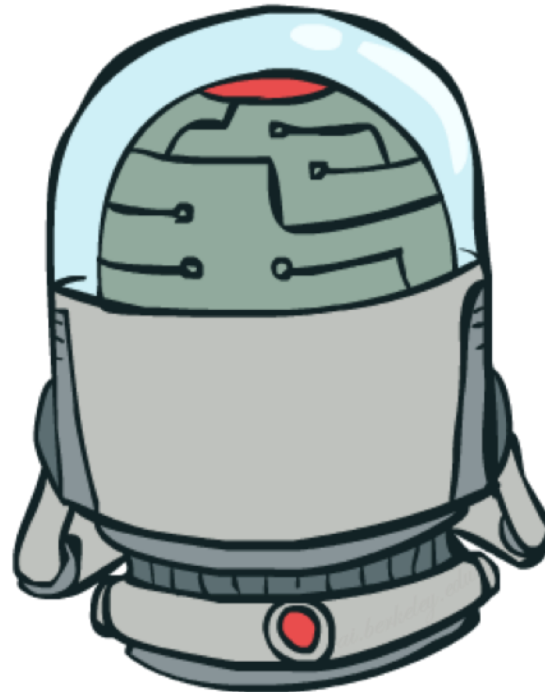
Reinforcement Learning
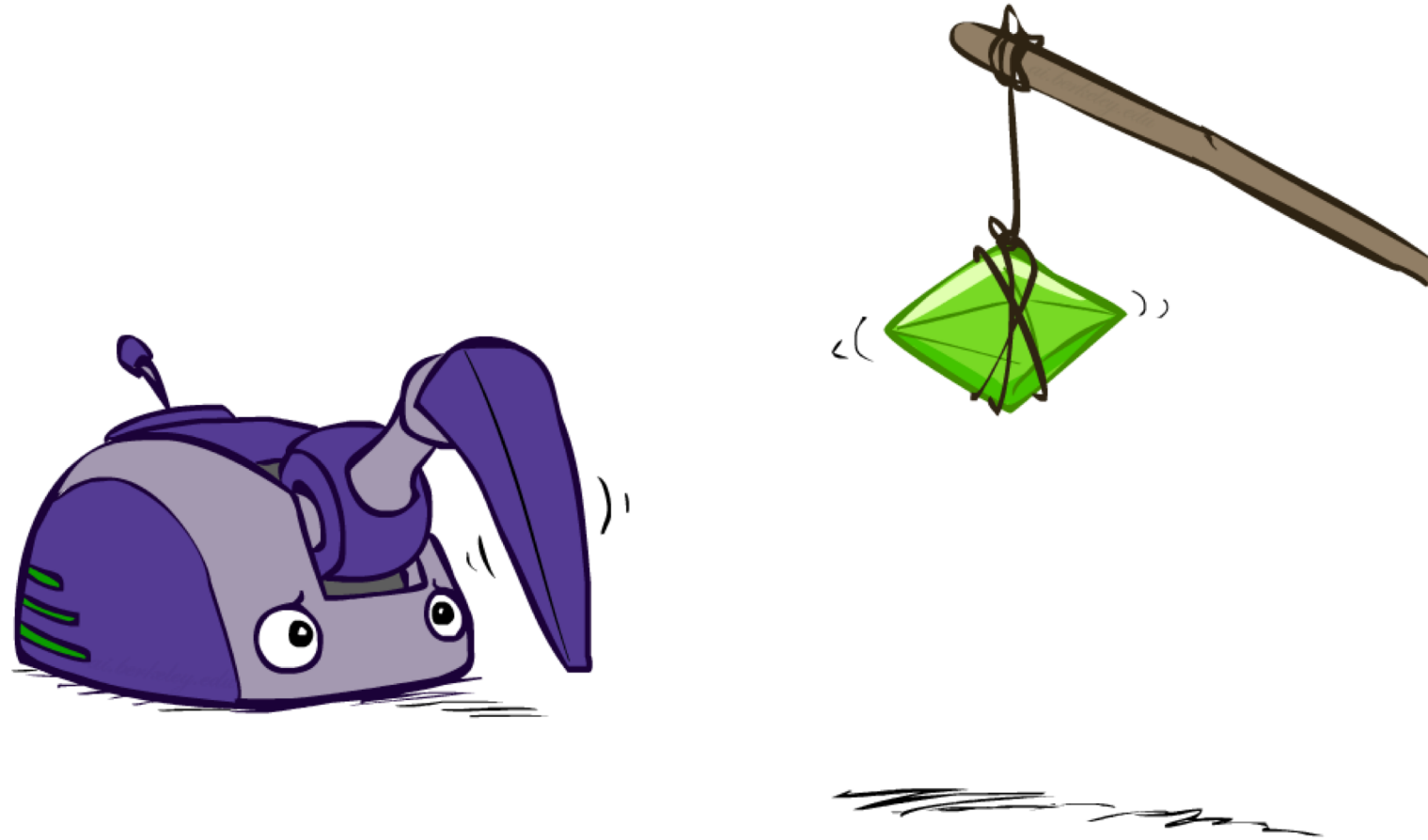
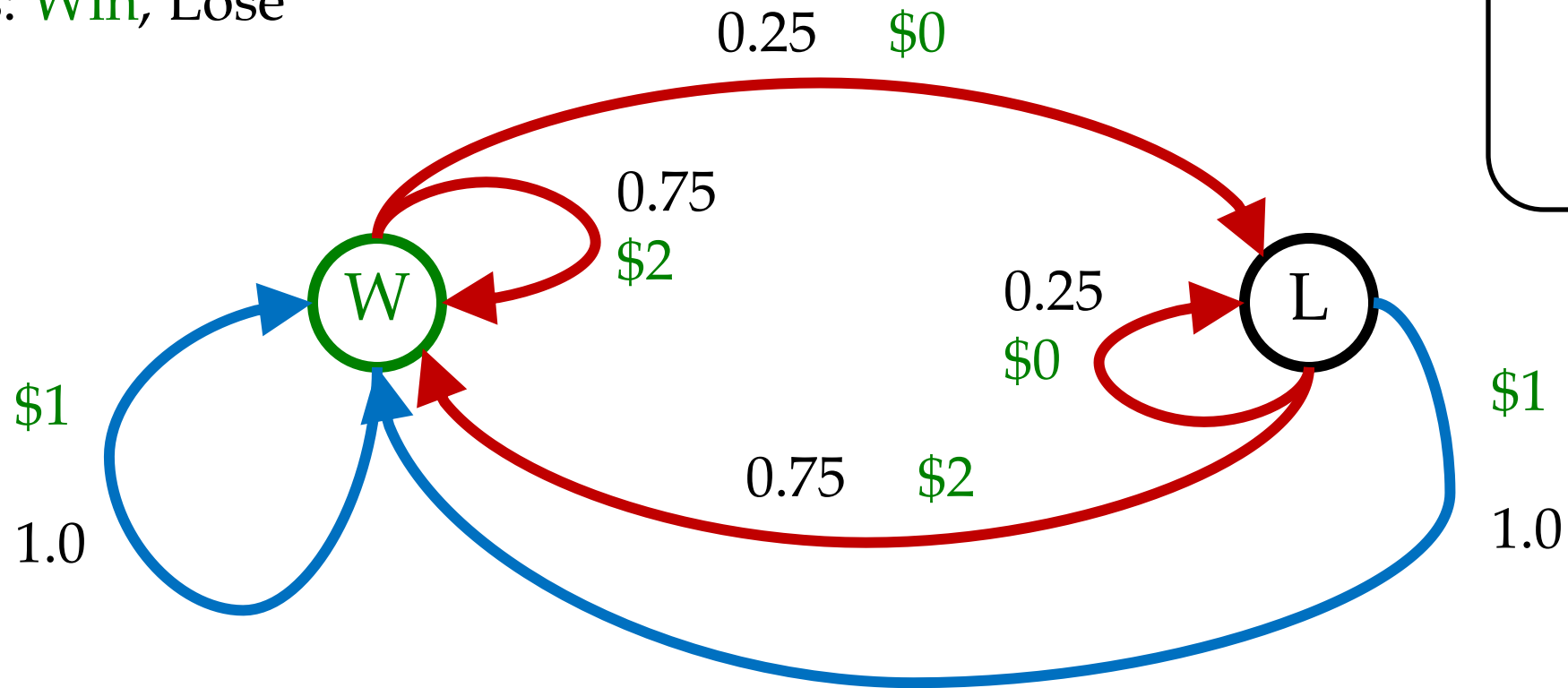# Double Bandits

Double-Bandit MDP

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

No discount
10 time steps

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

10 time steps

	Value
Play Red	15
Play Blue	10
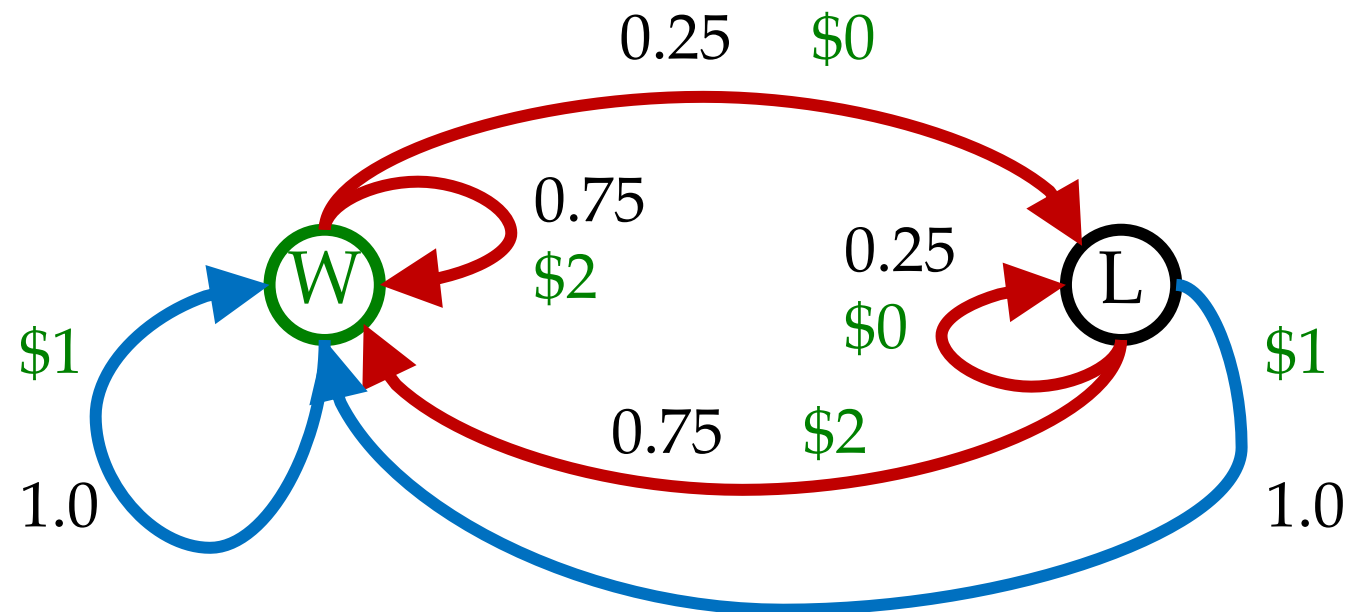
Let's Play!
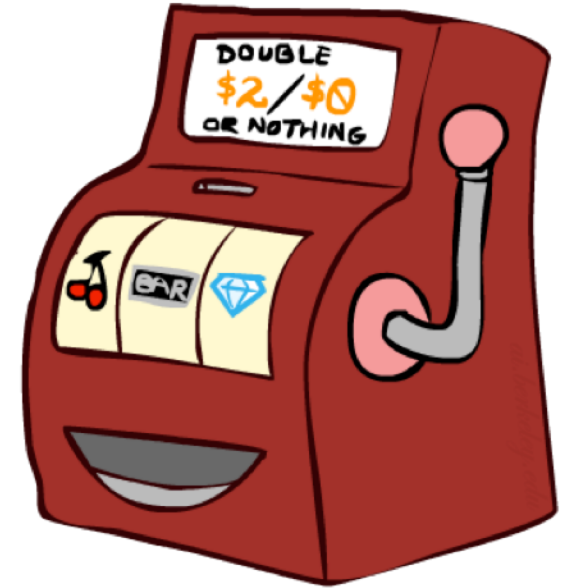
$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Online Planning

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

Let's Play!

$0 $0 $0 $0

$2 $2 $2 $0 $0

$2

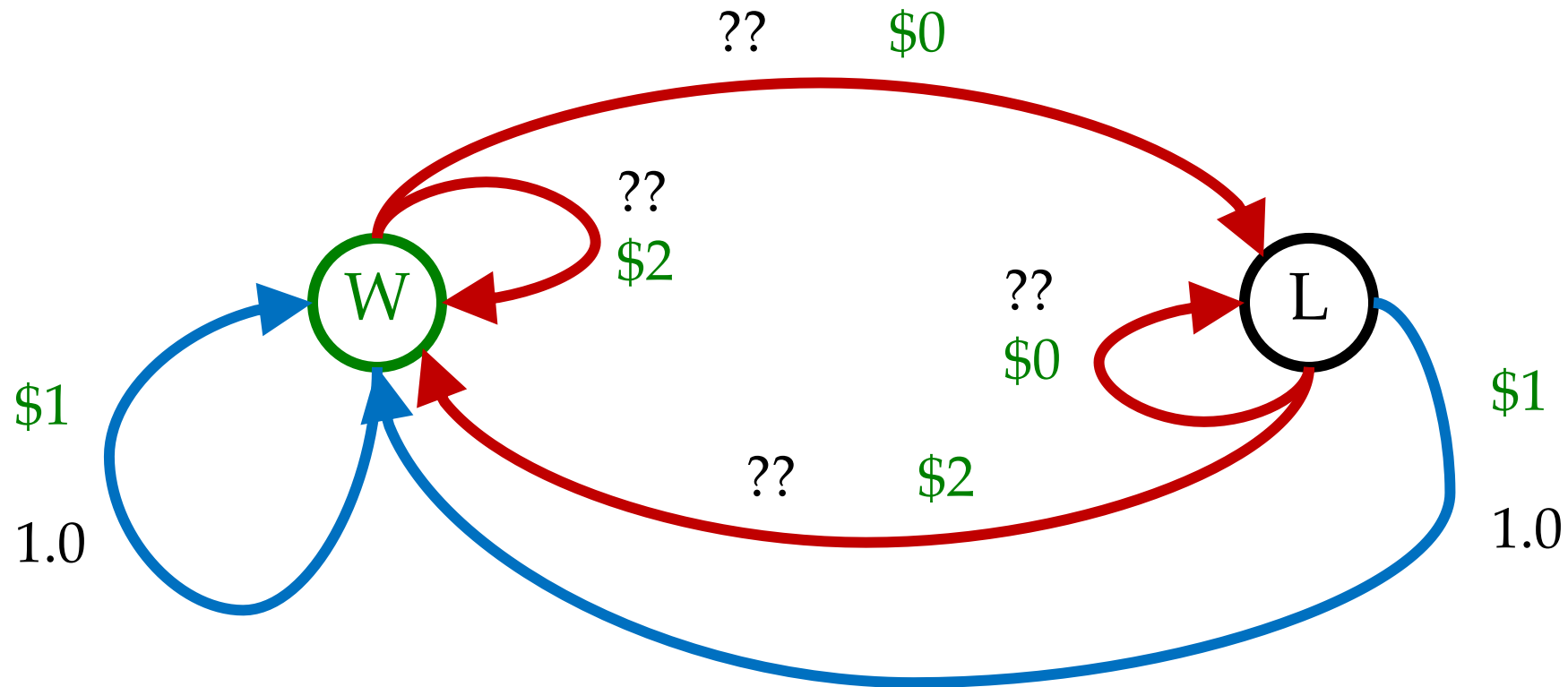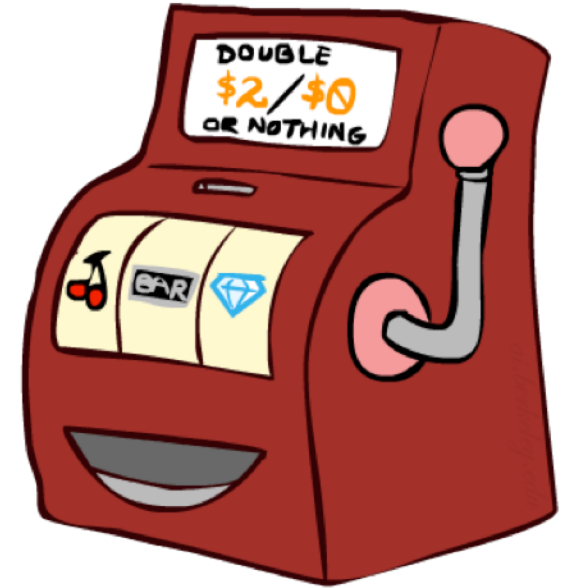# What Just Happened?

o **That wasn't planning, it was learning!**
  o Specifically, reinforcement learning
  o There was an MDP, but you couldn't solve it with just computa
  o You needed to actually act to figure it out

o **Important ideas in reinforcement learning that came up**
  o Exploration: you have to try unknown actions to get information
  o Exploitation: eventually, you have to use what you know
  o Regret: even if you learn intelligently, you make mistakes
  o Sampling: because of chance, you have to try things repeatedly
  o Difficulty: learning can be much harder than solving a known MDP

Reinforcement Learning
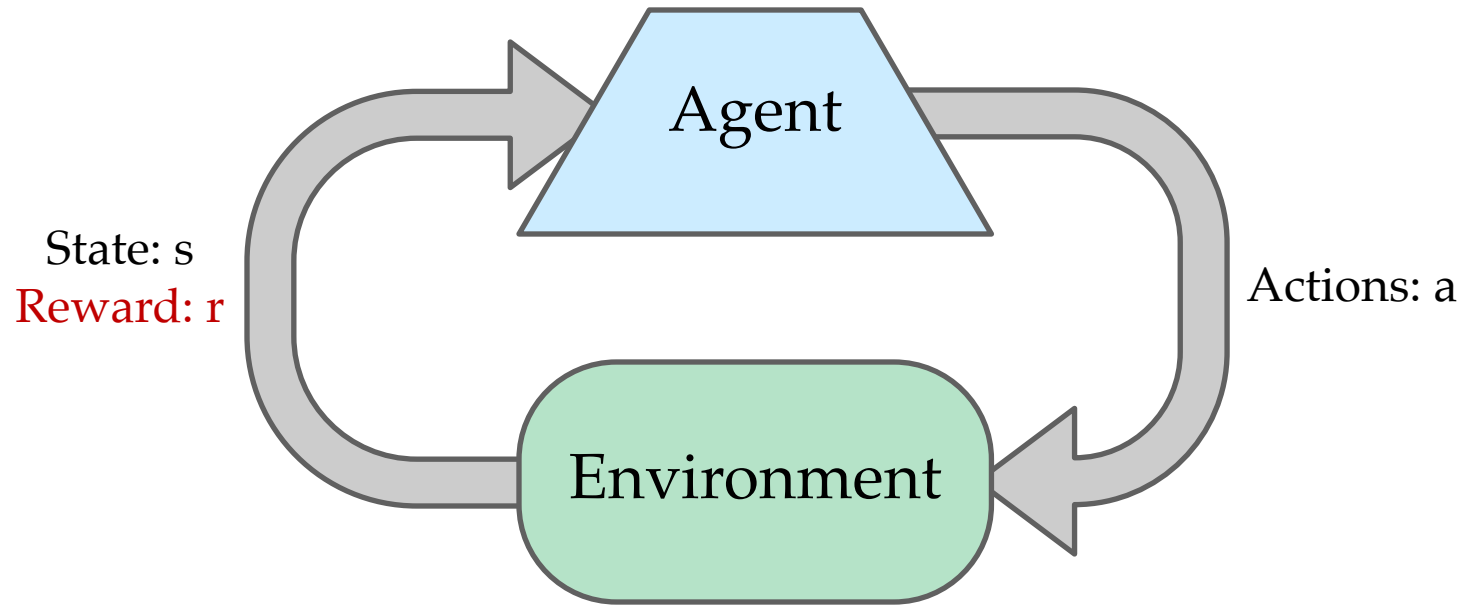
o Still assume a Markov decision process (MDP):

    o A set of states s ∈ S

    o A set of actions (per state) A

    o A model T(s,a,s')

    o A reward function R(s,a,s')

o Still looking for a policy π(s)

Warm

Cool

Overheated

o New twist: don't know T or R

    o I.e. we don't know which states are good or what the actions do

    o Must actually try actions and states out to learn

Reinforcement Learning

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

Example: Learning to Walk

Initial   A Learning Trial   After Learning [1K Trials]

[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

Example: Learning to Walk

Training

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – training]

Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

Example: Toddler Robot

[Tedrake, Zhang and Seung, 2005]   [Video: TODDLER – 40s]
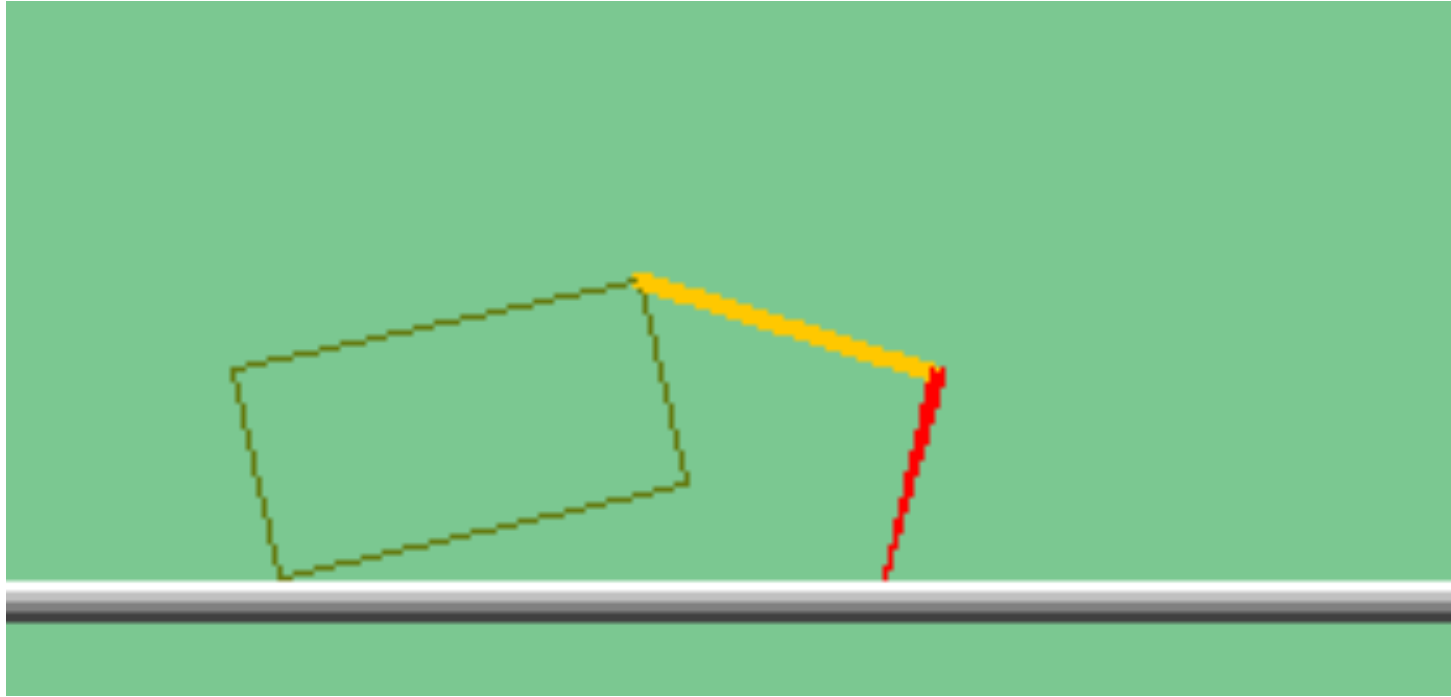
# The Crawler!

Video of Demo Crawler Bot

Reinforcement Learning

o Still assume a Markov decision process (MDP):

    o A set of states $s \in S$

    o A set of actions (per state) A

    o A model T(s,a,s')

    o A reward function R(s,a,s')

o Still looking for a policy π(s)

o New twist: don't know T or R

    o I.e. we don't know which states are good or what the actions do

    o Must actually try actions and states out to learn

Warm

Cool

Overheated

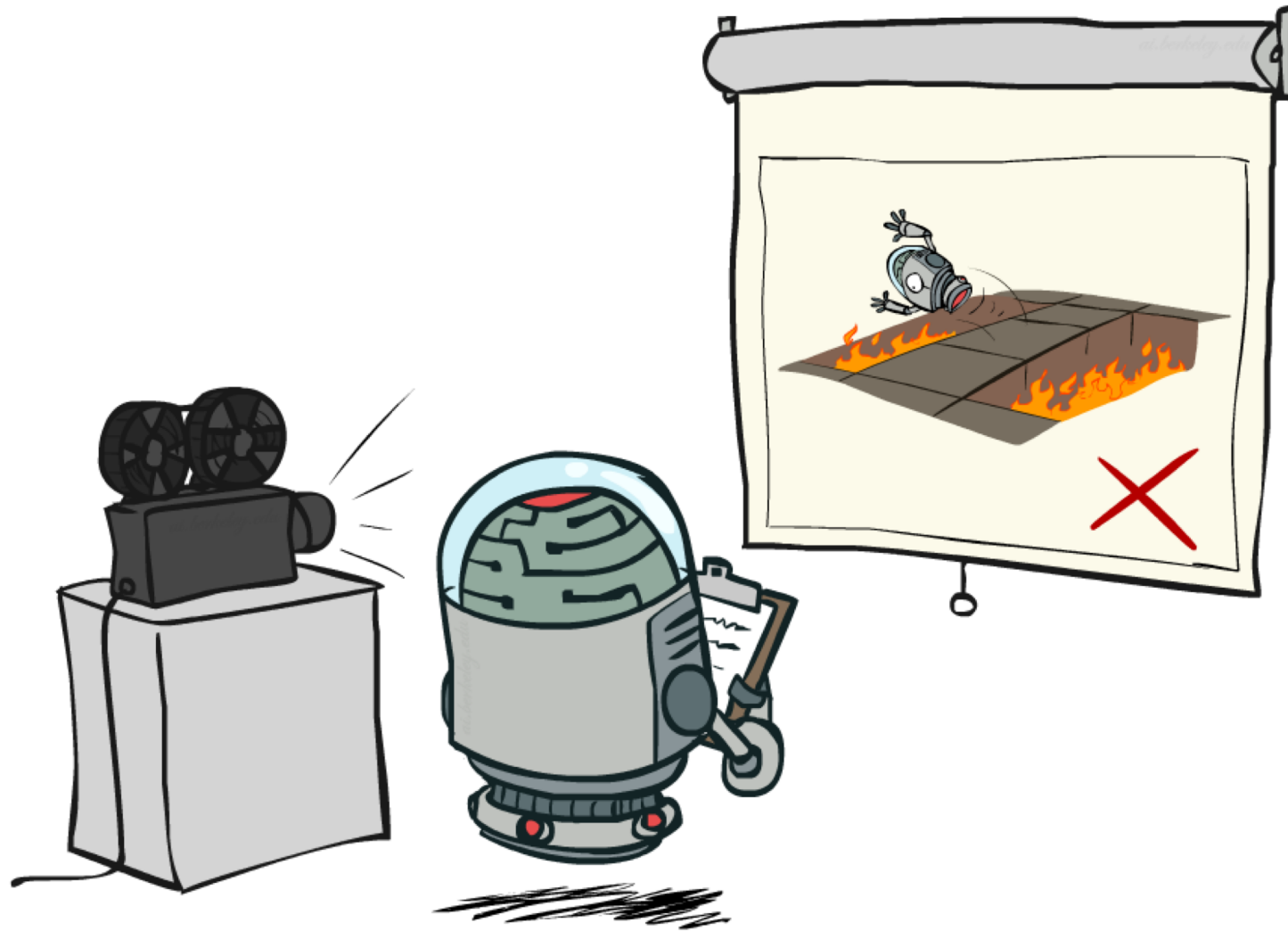# Offline (MDPs) vs. Online (RL)
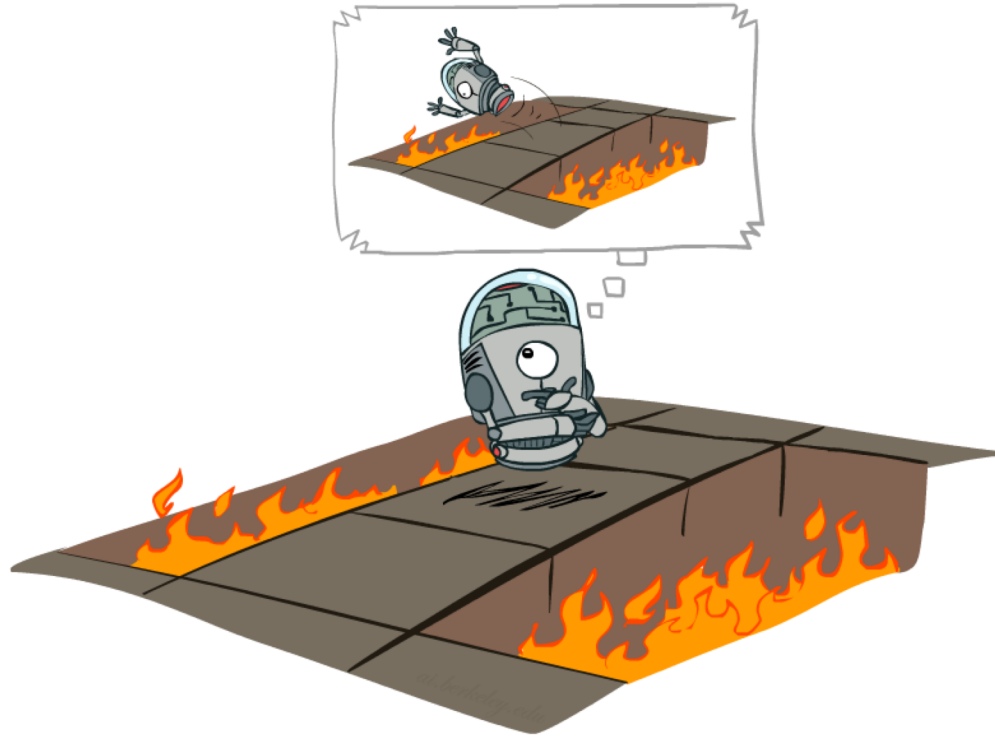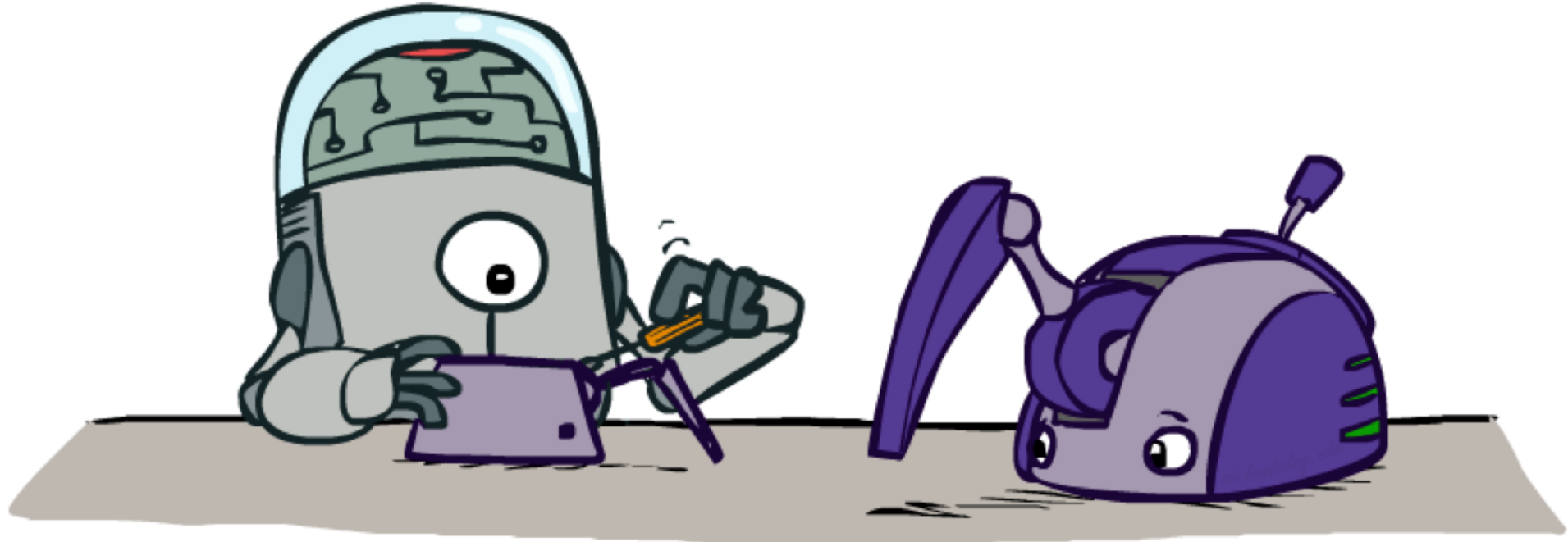

Offline Solution

Online Learning

Passive Reinforcement Learning

Model-Based Learning

Model-Based Learning

o Model-Based Idea:
 o Learn an approximate model based on experiences
  o Solve for values as if the learned model were correct

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

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

Example: Model-Based Learning

Input Policy π
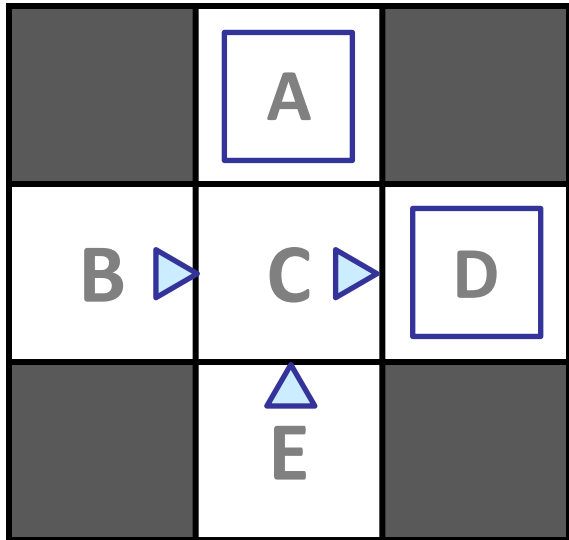
Assume: γ = 1

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, 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 cs473 students

Known P(A)

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

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

Unknown P(A): "Model Based"

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

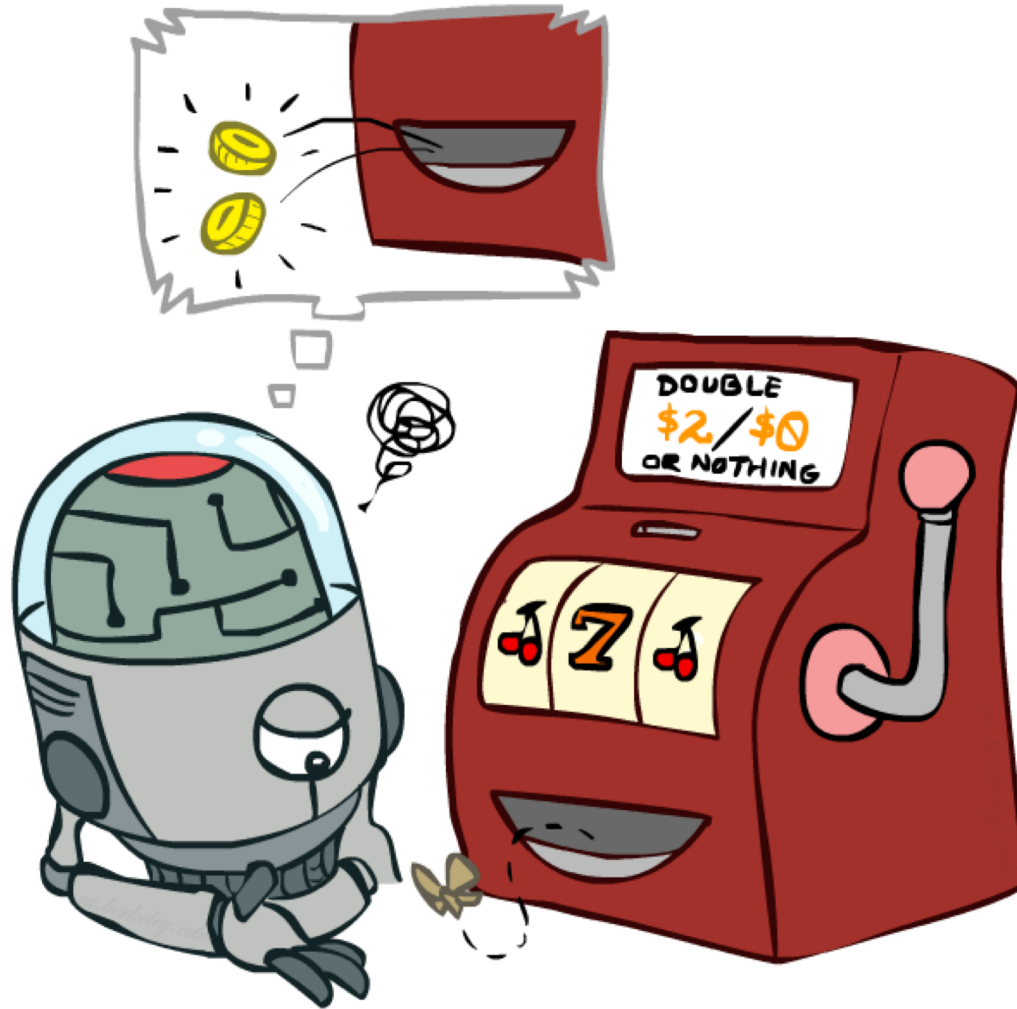$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

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

Passive Reinforcement Learning

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

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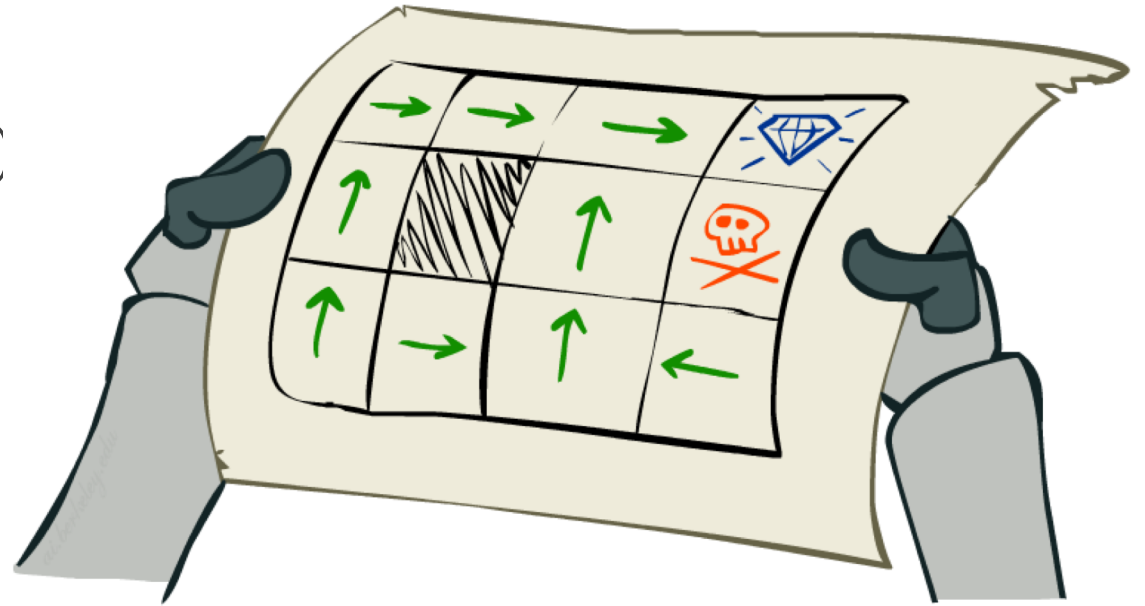
Direct Evaluation

o Goal: Compute values for each state under π

o Idea: Average together observed sample values

   o Act according to π

   o Every time you visit a state, write down what the sum of discounted rewards turned out to be

   o Average those samples

o This is called direct evaluation

Example: Direct Evaluation

Input Policy π
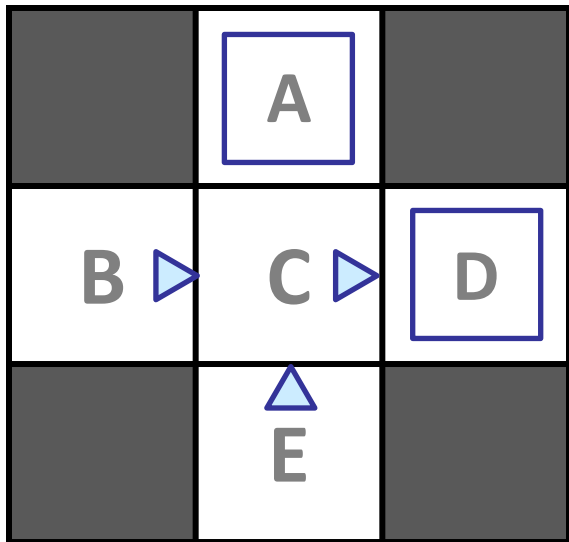
Assume: γ = 1

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

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

Problems with Direct Evaluation

o What's good about direct evaluation?

    o It's easy to understand

    o It doesn't require any knowledge of T, R

    o It eventually computes the correct average values, using just sample transitions

o What bad about it?

    o It wastes information about state connections

    o Each state must be learned separately

    o So, it takes a long time to learn
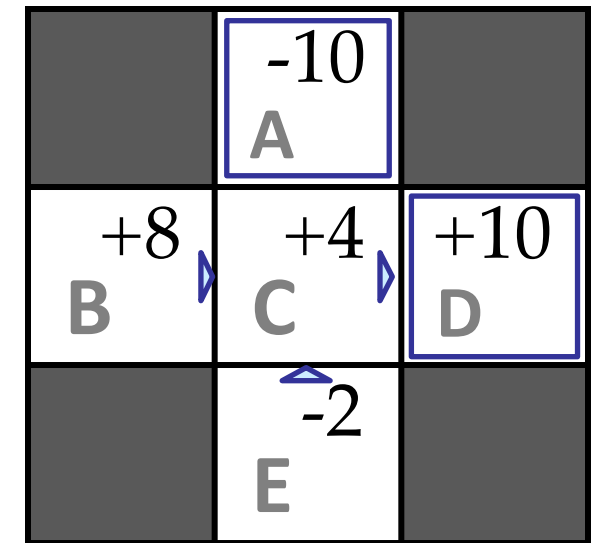
If B and E both go to C under this policy, how can their values be different?

Why Not Use Policy Evaluation?

o Simplified Bellman updates calculate V for a fixed policy:

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

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

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

   o This approach fully exploited the connections between the states

   o Unfortunately, we need T and R to do it!

o Key question: how can we do this update to V without knowing T and R?

   o In other words, how to we take a weighted average without knowing the weights?

s

$\pi(s)$

s, $\pi(s)$

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

s'
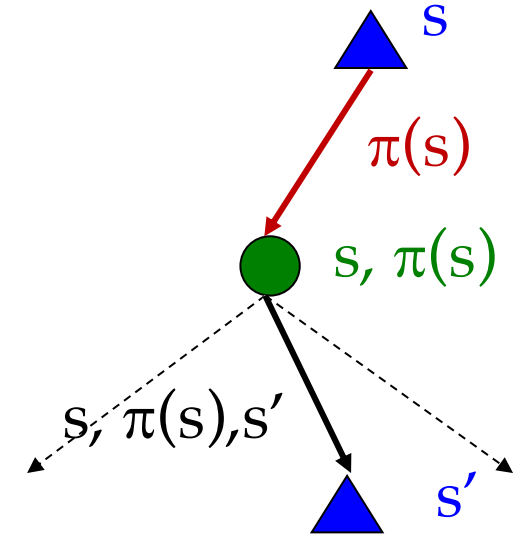
Sample-Based Policy Evaluation?

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

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

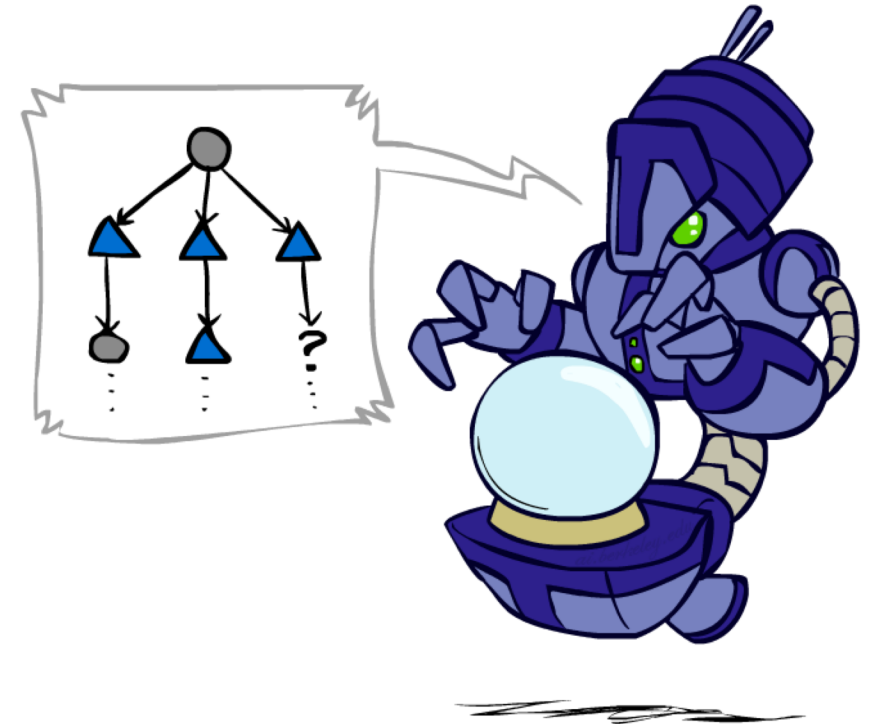o Idea: Take samples of outcomes s' (by doing the action!) and average

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

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

$$\dots$$

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

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

Temporal Difference Learning

o Big idea: learn from every experience!

   o Update V(s) each time we experience a transition (s, a, s', r)
   o Likely outcomes s' will contribute updates more often

o Temporal difference learning of values

   o Policy still fixed, still doing evaluation!
   o Move values toward value of whatever successor occurs: running average
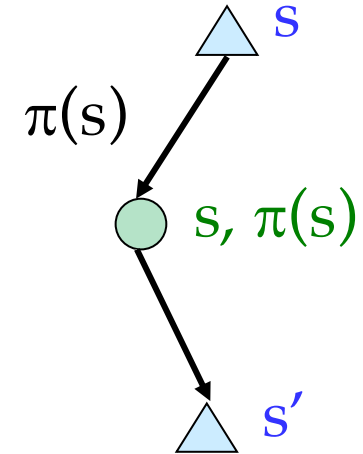
Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

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

  - 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

B, east, C, -2

C, east, D, -2

	0	
•0	0	8
	0	

	0	
-1	•0	8
	0	

	0	
-1	3	•8
	0	

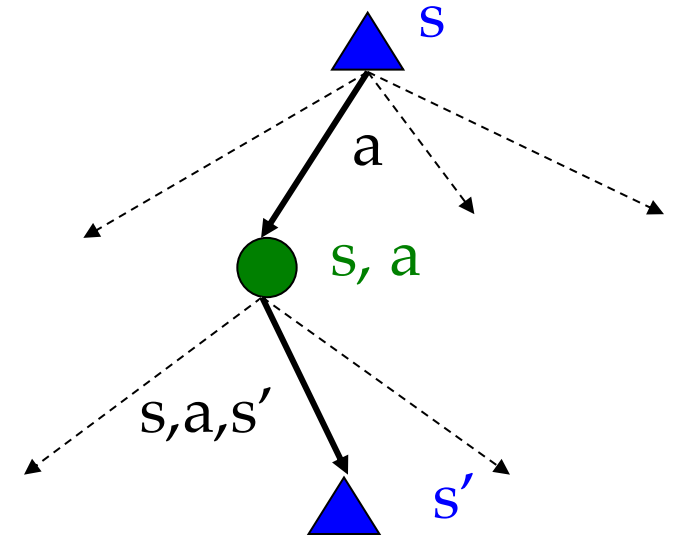$$V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

Problems with TD Value Learning

o TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages

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

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

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

o Idea: learn Q-values, not values

o Makes action selection model-free too!

Detour: Q-Value Iteration

o Value iteration: find successive (depth-limited) values
  o Start with $V_0(s) = 0$, which we know is right
  o Given V_k, calculate the depth $k+1$ values for all states:

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

o But Q-values are more useful, so compute them instead
  o Start with $Q_0(s,a) = 0$, which we know is right
  o 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

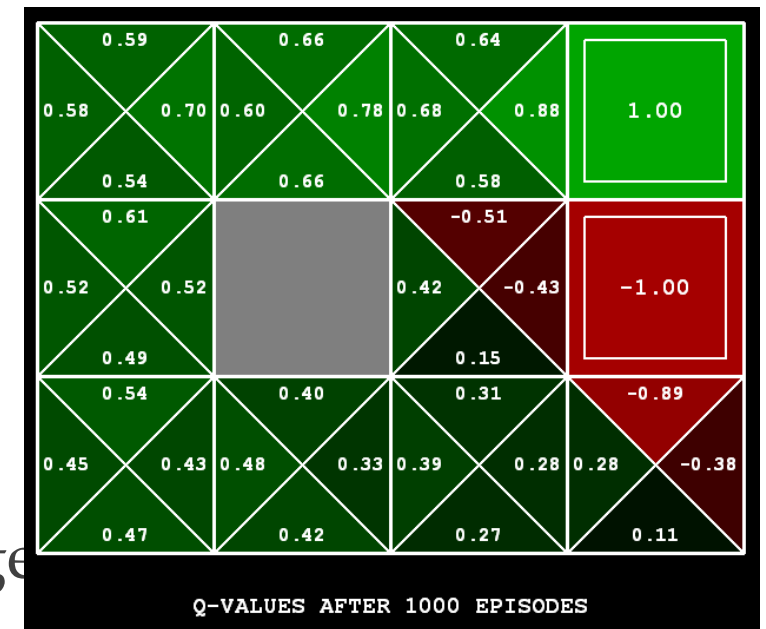$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

○ 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!
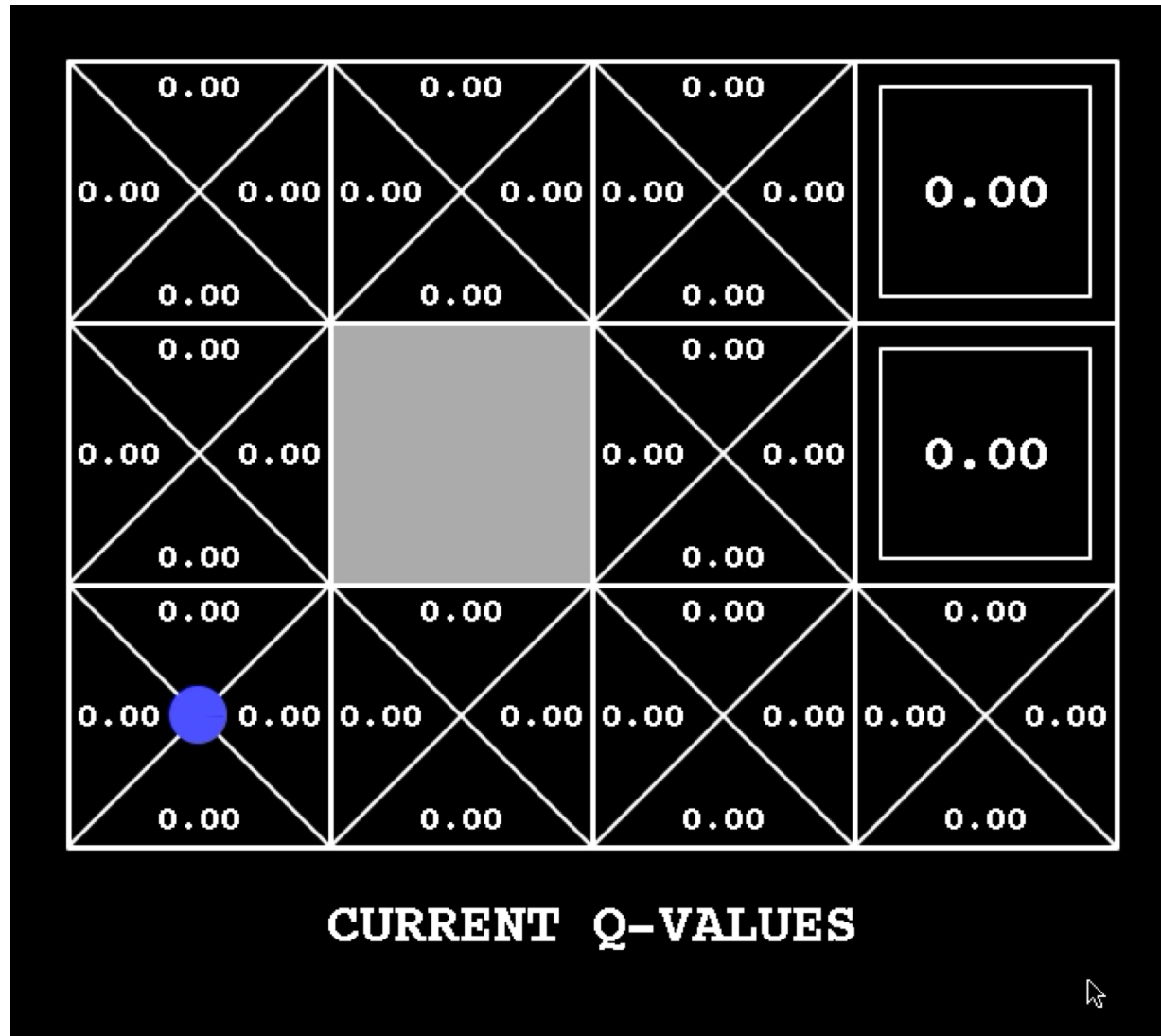
   ○ Incorporate the new estimate into a running average

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

Q-VALUES AFTER 1000 EPISODES

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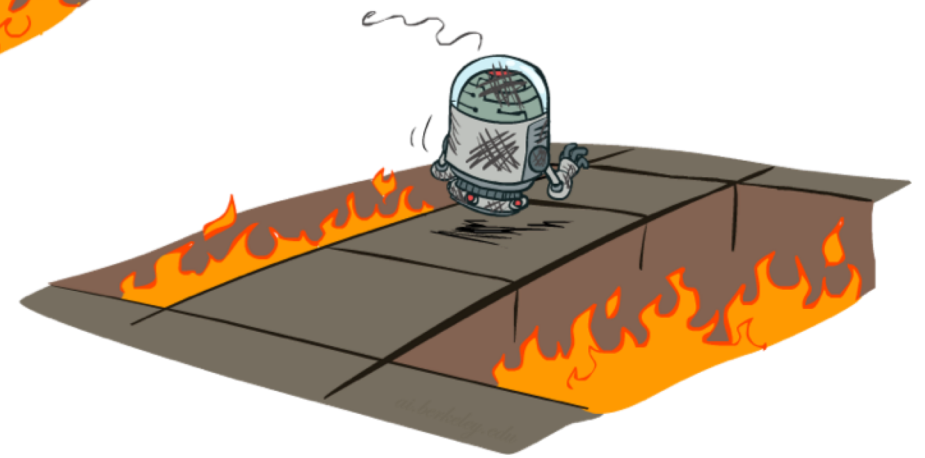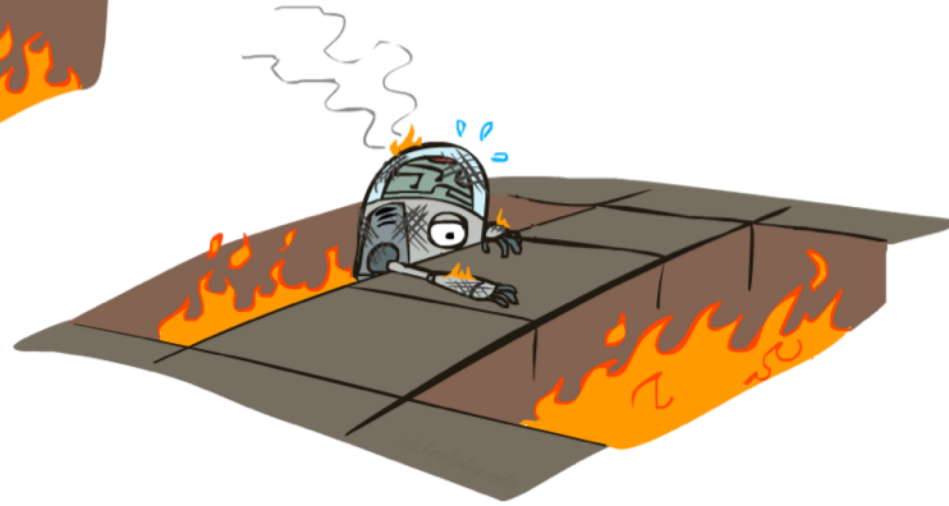
Q-Learning Demo

Video of Demo Q-Learning -- Gridworld

Video of Demo Q-Learning -- Crawler

Active Reinforcement Learning

Q-Learning:
# act according to current optimal (and also explore...)

o Full reinforcement learning: optimal policies (like value iteration)

   o You don't know the transitions T(s,a,s')

   o You don't know the rewards R(s,a,s')

   o You choose the actions now

   o Goal: learn the optimal policy / values

o In this case:

   o Learner makes choices!

   o Fundamental tradeoff: exploration vs. exploitation

   o This is NOT offline planning! You actually take actions in the world and find out what happens...
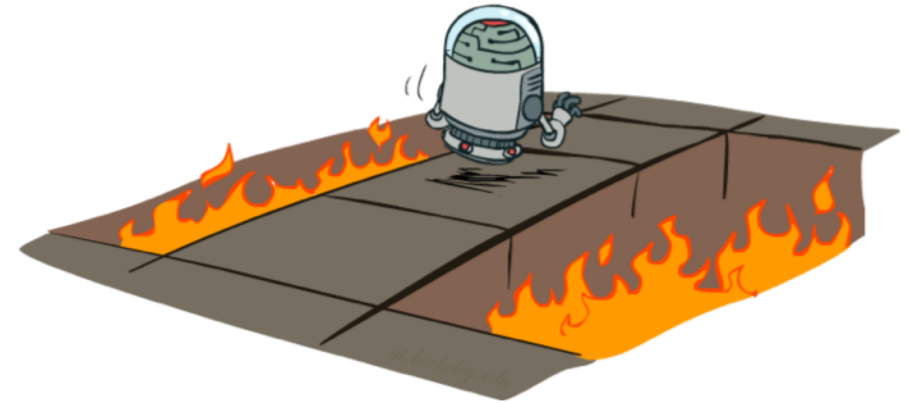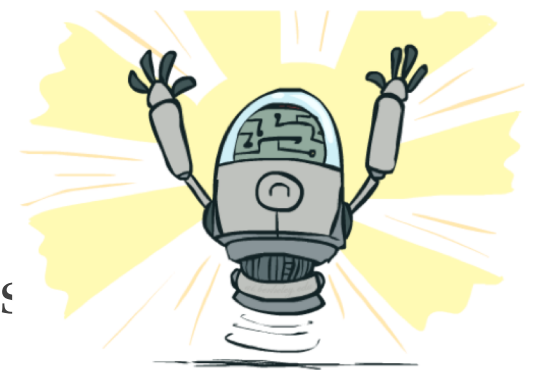
Q-Learning Properties

o Amazing result: Q-learning converges to optimal policy --
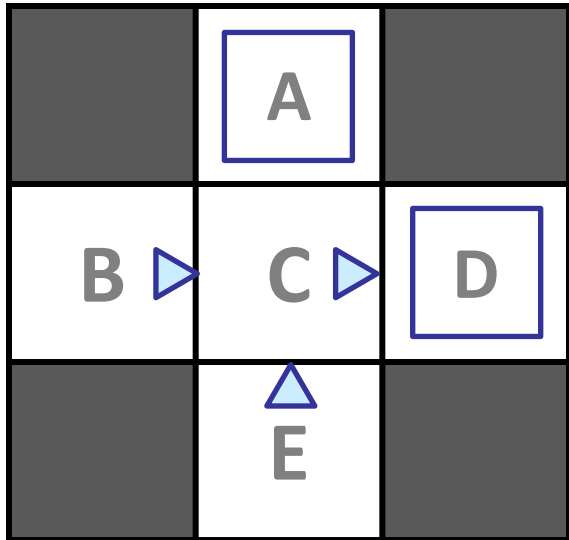 even if you're acting suboptimally!

o This is called off-policy learning
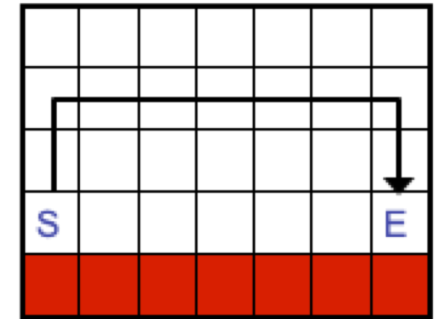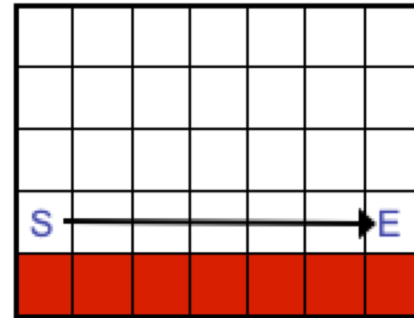
o Caveats:

   o You have to explore enough

   o You have to eventually make the learning rate
  small enough

   o … but not decrease it too quickly

   o Basically, in the limit, it doesn't matter how you select actions

# Model-Based Learning

act according to current optimal
also explore!

How to Explore?

o Several schemes for forcing exploration

    o Simplest: random actions (ε-greedy)

        o Every time step, flip a coin

        o With (small) probability ε, act randomly

        o With (large) probability 1-ε, act on current policy

    o Problems with random actions?

        o You do eventually explore the space, but keep thrashing around once learning is done

        o One solution: lower ε over time

        o Another solution: exploration functions

Q-Learn Epsilon Greedy
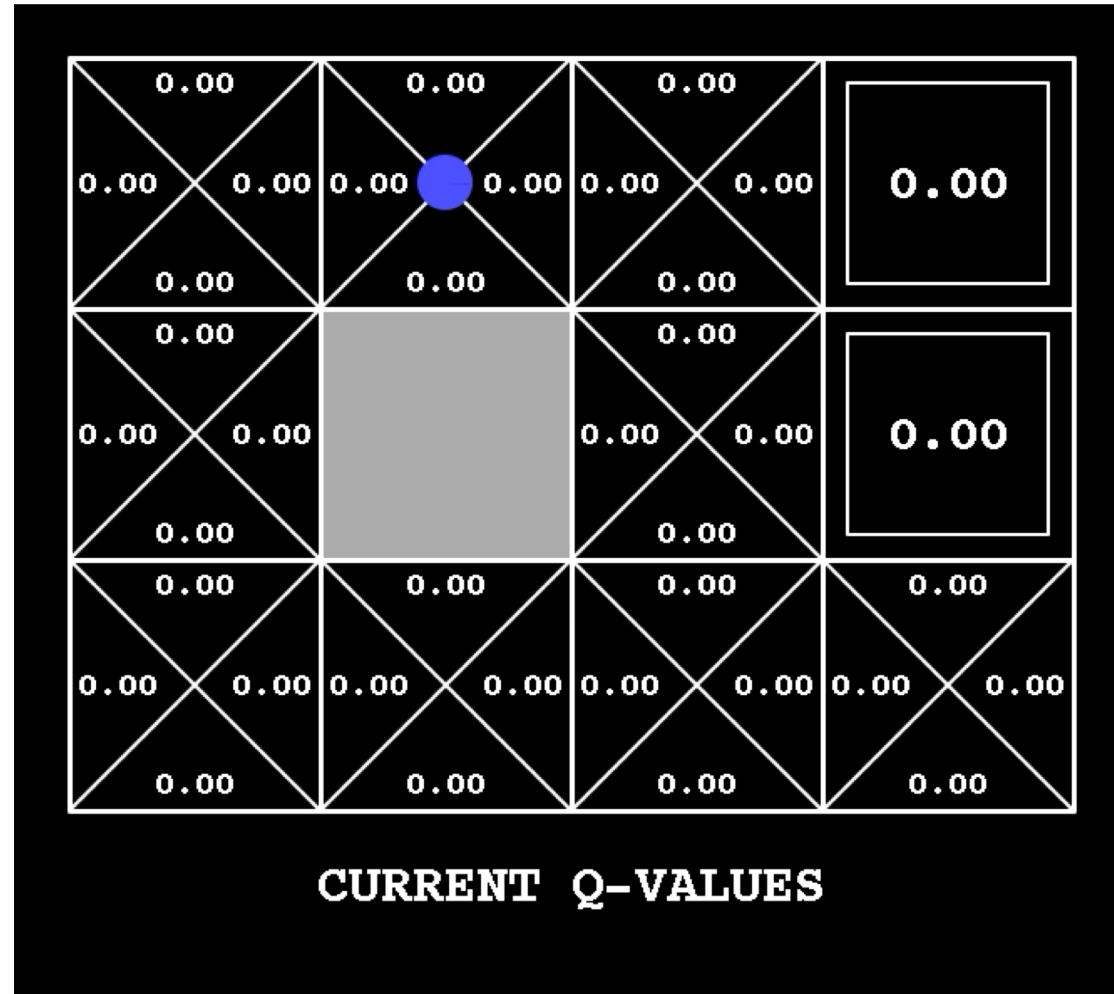
Discussion: Model-Based vs Model-Free RL