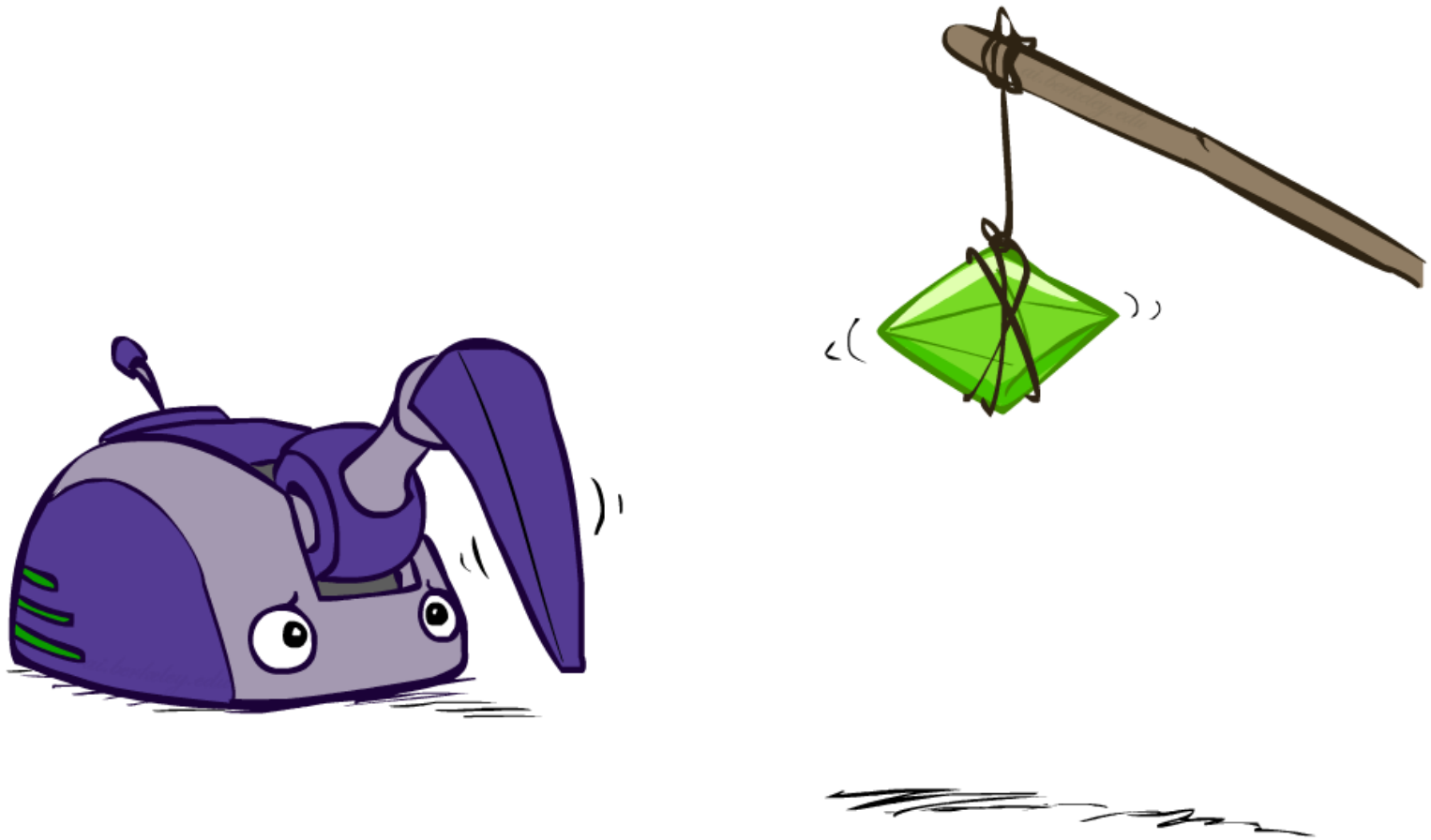
CS 188: Artificial Intelligence

Reinforcement Learning

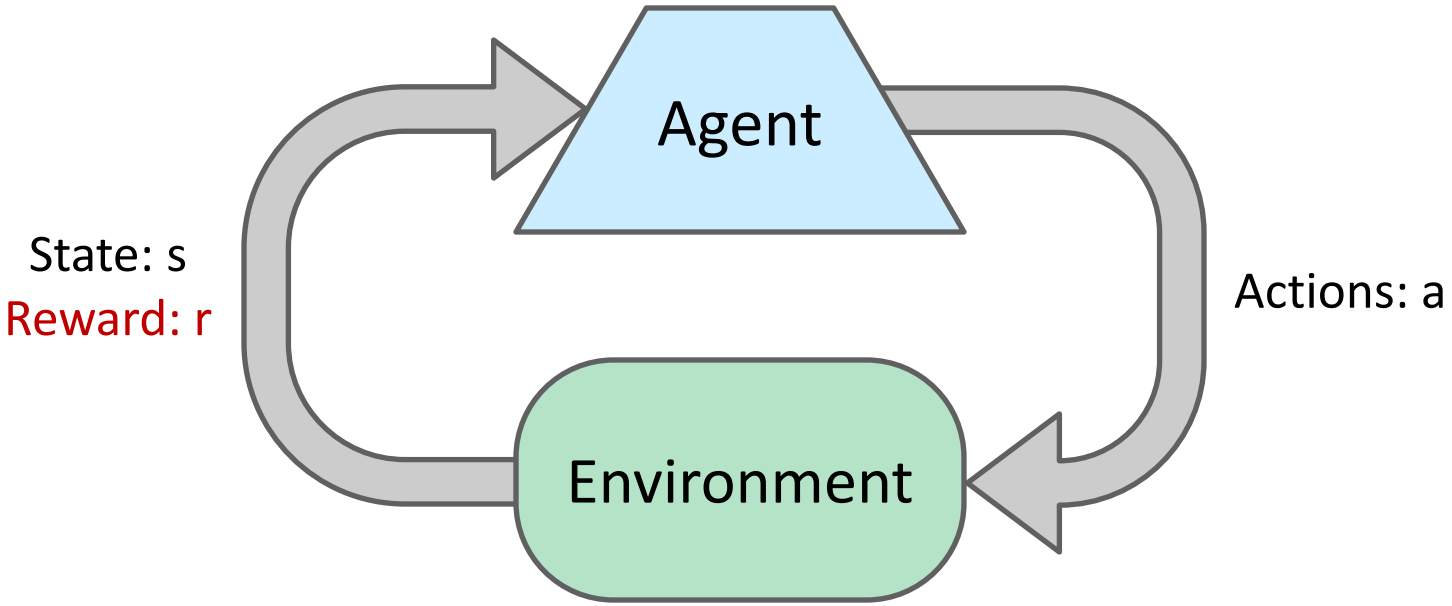
Instructors: Dan Klein and Pieter Abbeel

University of California, Berkeley

Reinforcement Learning



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]

[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]

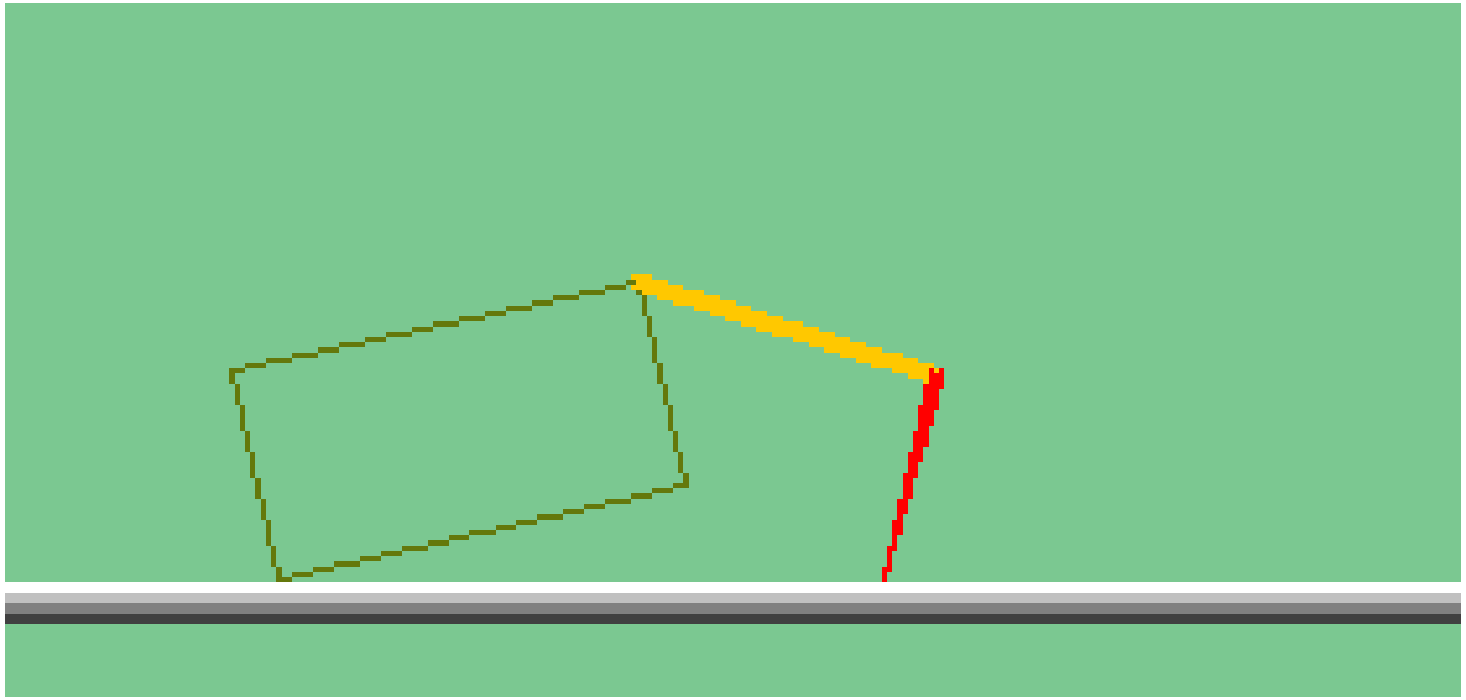
[Video: AIBO WALK – finished]

Example: Toddler Robot

[Tedrake, Zhang and Seung, 2005]

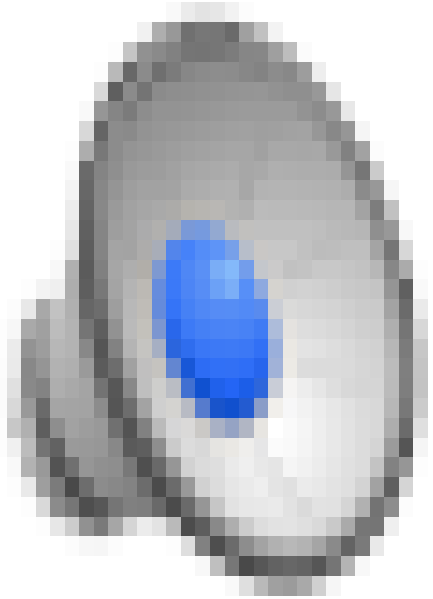
[Video: TODDLER – 40s]

The Crawler!



[Demo: Crawler Bot (L10D1)] [You, in Project 3]

Video of Demo Crawler Bot



Reinforcement learning (RL)

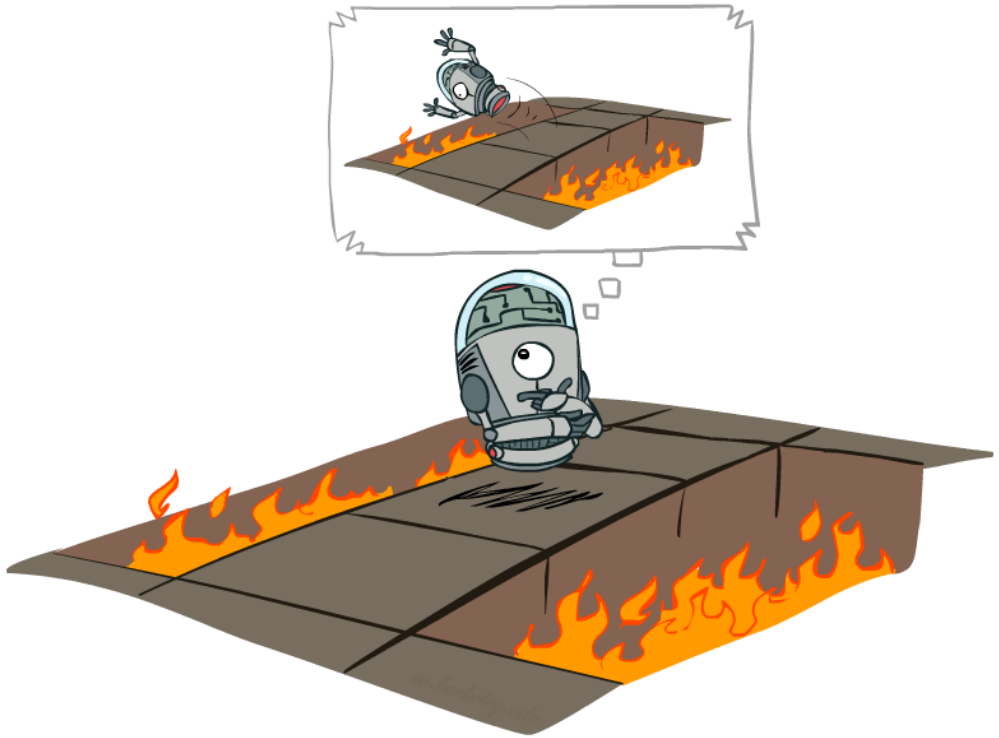
- In Reinforcement learning (RL), an agent interacts with the world and periodically receives rewards (or, in the terminology of psychology, reinforcements) that reflect how well it is doing.
- For example, in chess the reward is 1 for winning, 0 for losing, and 12 for a draw. We have already seen the concept of rewards in Chapter 16 for Markov decision processes (MDPs).
- Indeed, the goal is the same in reinforcement learning: maximize the expected sum of rewards. Reinforcement learning differs from "just solving an MDP" because the agent is not given the MDP as a problem to solve; the agent is in the MDP.
- It may not know the transition model or the reward function, and it has to act in order to learn more.

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

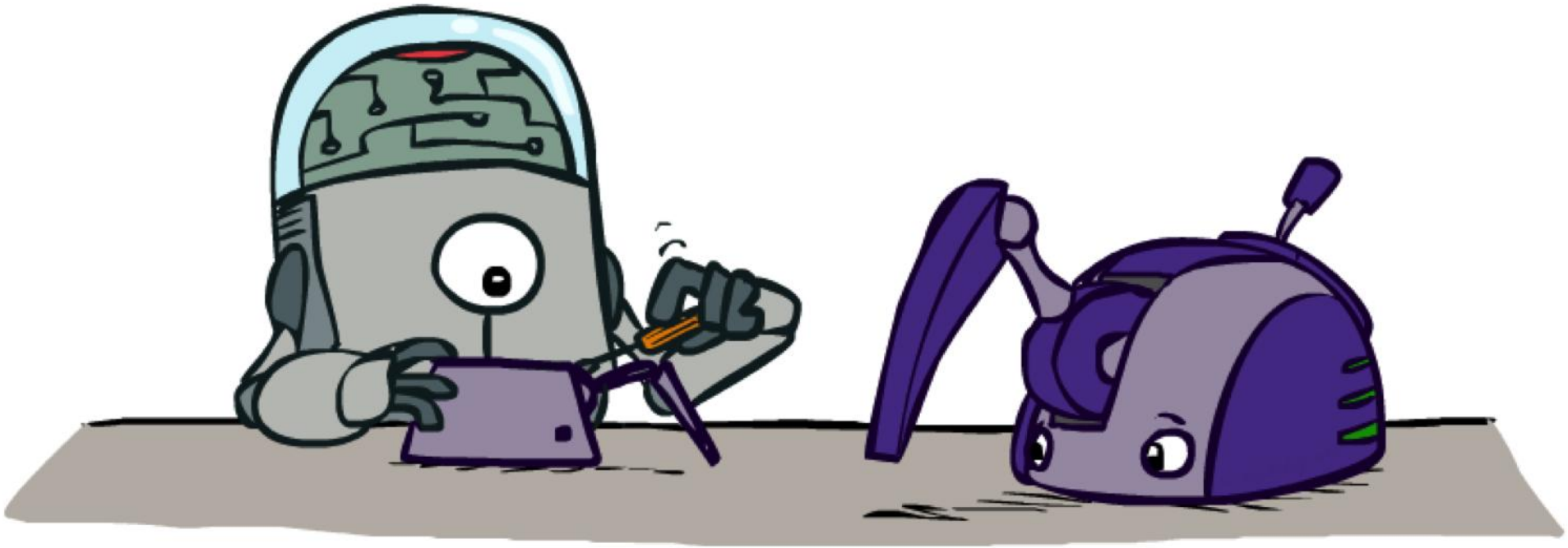
Offline (MDPs) vs. Online (RL)



Offline Solution:
Given an MDP, you have to find the optimal solution (Go always straight).

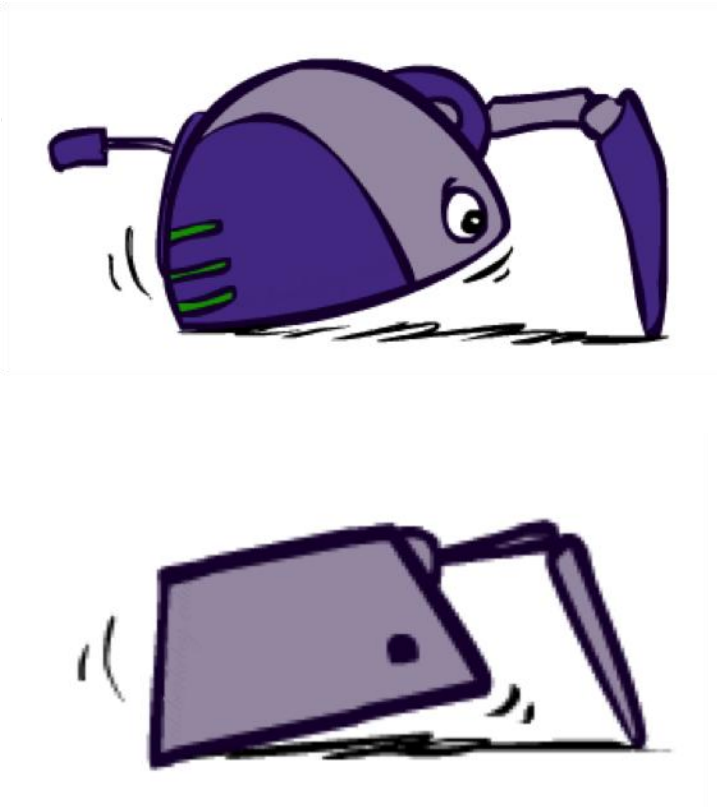
Online Learning: You have to do learning. Choose an action and see what happens and learn about the world.

Model-Based Learning



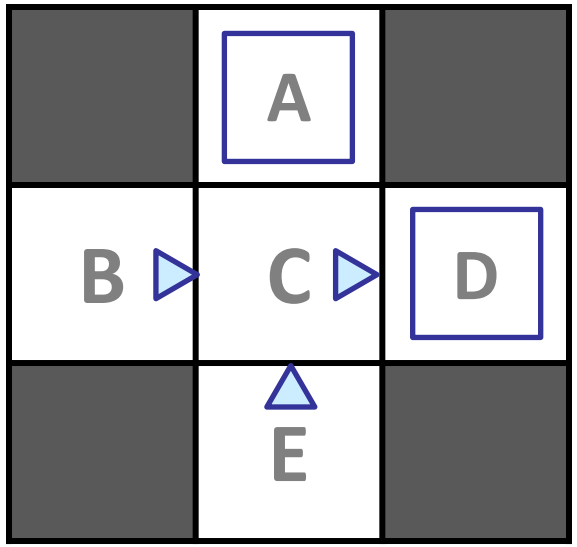
Model-Based Learning

- Model-Based Idea:
  - You assume a model and learn the parameters of that model
  - 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 π



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

2/2
3/4
1/4

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

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

The estimated probablity of C to D = 3 transitions / in 4 episodes

Model Based Learning vs Model Free Learning
Example: Expected Age

Goal: Compute expected age of cs188 students

Known distribution 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"

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

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

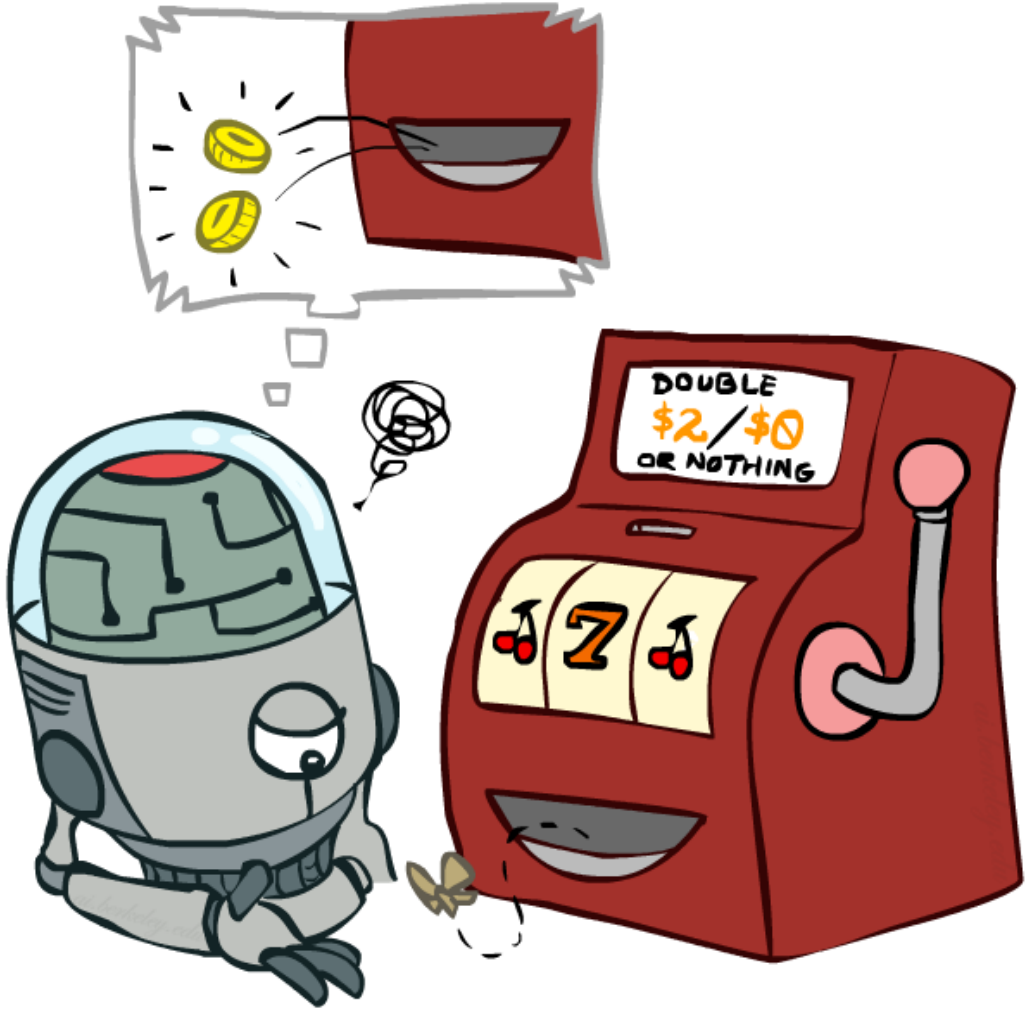
Why does this work? Because eventually you learn the right model.

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



- In reinforcement learning (RL), a model-free algorithm (as opposed to a model-based one) is an algorithm which does not use the *transition probability distribution* (and the *reward function*) associated with the Markov decision process (MDP),[1] which, in RL, represents the problem to be solved.
- The **transition probability distribution** (or transition model) and the **reward function** are often collectively called the "**model**" of the environment (or MDP), hence the name "model-free".
- A model-free RL algorithm can be thought of as an "explicit" trial-and-error algorithm

Learning from Rewards

- Hundreds of different reinforcement learning algorithms have been devised, and many of them can employ as tools a wide range of learning methods
- These approaches can be categorized as follows:
  - Model-based reinforcement learning: In these approaches the agent uses a transition learning model of the environment to help interpret the reward signals and to make decisions about how to act. Model-based reinforcement learning systems often learn a utility (also called value) function U(s), defined in terms of the sum of rewards from state s onward. (learn the MDP model (T and R), or an approximation of it.)
  - *Model-free reinforcement learning: In these approaches the agent neither knows nor learns a transition model for the environment. Instead, it learns a more direct representation of how to behave. (*– derive the optimal policy without explicitly learning the model.)

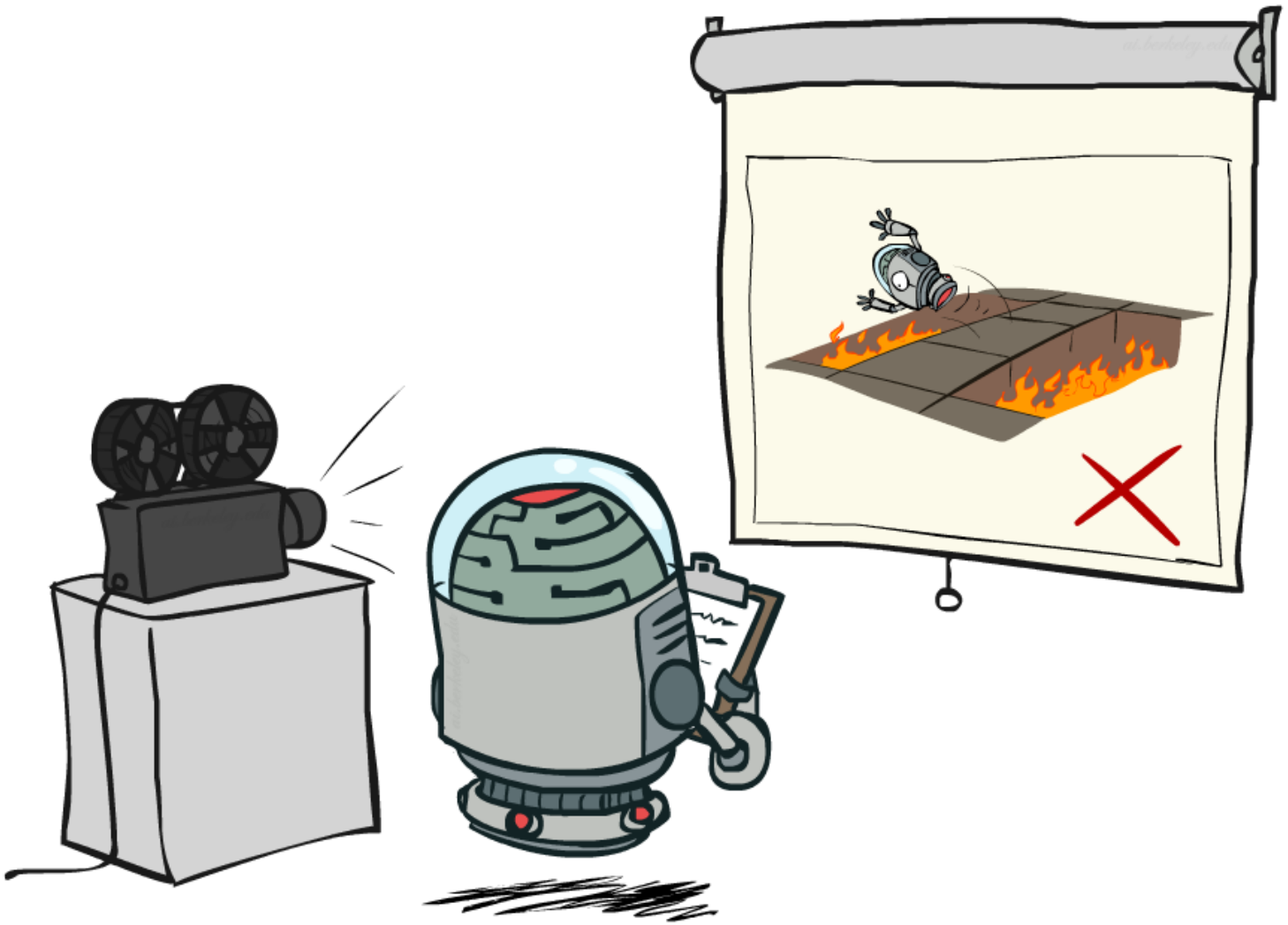
Passive vs. Active learning

- Passive learning
  - The agent acts based on a fixed policy π and tries to learn how good the policy is by observing the world go by
  - Analogous to policy evaluation in policy iteration
- Active learning
  - The agent attempts to find in an optimal (or at least good) policy by exploring different actions in the world
  - Analogous to solving the underlying MDP

Summary of Key Ideas for Learning

- Online vs. Batch
  - Learn while exploring the world, or learn from fixed batch of data
- Active vs. Passive
  - Does the learner actively choose actions to gather experience? or, is a fixed policy provided?
- Model based vs. Model free
  - Do we estimate T(s,a,s') and R(s,a,s'), or just learn values/policy directly

Passive Reinforcement Learning

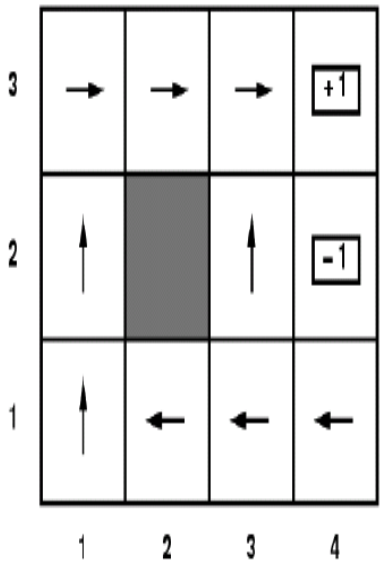


- The agent doesn't know the transition model or the reward function.
- observe the reward in each state it is in.
- Like watching a video and learning from that video
- You don't have any control
- Ex: A robot can jump in the fire pit and observe the reward it gets

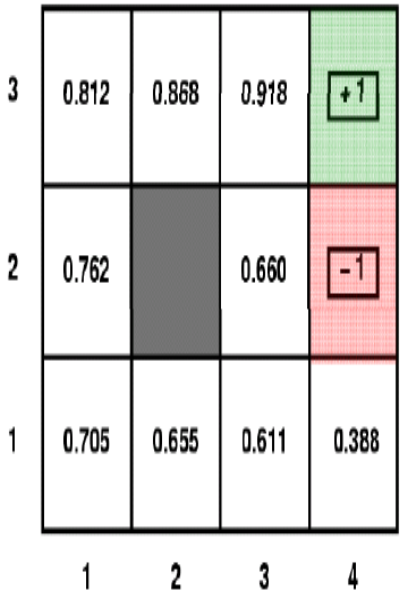
Passive Reinforcement Learning

- Suppose we are given a policy
- Want to determine how good it is

Given π:



Need to learn $U_\pi(S)$:



The agent executes a set of trials

of **trials** in the environment using its policy π. In each trial, the agent starts in state (1,1) and experiences a sequence of state transitions until it reaches one of the terminal states, (4,2) or (4,3). Its percepts supply both the current state and the reward received for the transition that just occurred to reach that state. Typical trials might look like this:

$$(1,1)\xrightarrow[Up]{.04}(1,2)\xrightarrow[Up]{.04}(1,3)\xrightarrow[Right]{.04}(1,2)\xrightarrow[Up]{.04}(1,3)\xrightarrow[Right]{.04}(2,3)\xrightarrow[Right]{.04}(3,3)\xrightarrow[Right]{+1}(4,3)$$

$$(1,1)\xrightarrow[Up]{.04}(1,2)\xrightarrow[Up]{.04}(1,3)\xrightarrow[Right]{.04}(2,3)\xrightarrow[Right]{.04}(3,3)\xrightarrow[Right]{.04}(3,2)\xrightarrow[Up]{.04}(3,3)\xrightarrow[Right]{+1}(4,3)$$

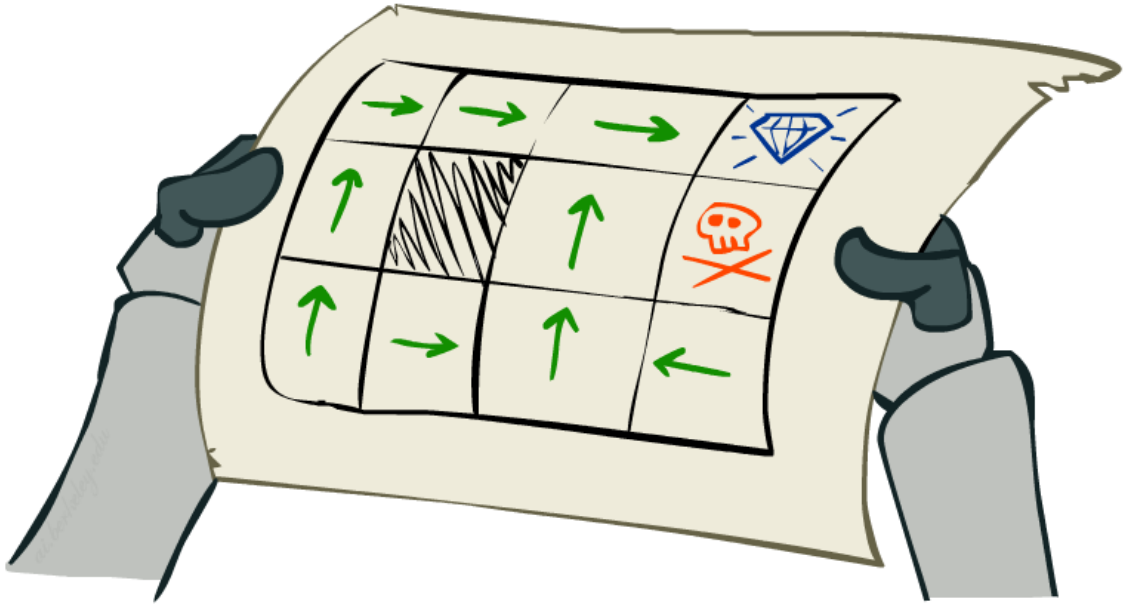
$$(1,1)\xrightarrow[Up]{.04}(1,2)\xrightarrow[Up]{.04}(1,3)\xrightarrow[Right]{.04}(2,3)\xrightarrow[Right]{.04}(3,3)\xrightarrow[Right]{.04}(3,2)\xrightarrow[Up]{-1}(4,2)$$

Note that each transition is annotated with both the action taken and the reward received at the next state. The object is to use the information about rewards to learn the expected utility $U^\pi(s)$ associated with each nonterminal state s. The utility is defined to be the expected sum of (discounted) rewards obtained if policy π is followed. As in Equation (16.2) on page 557, we write

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty}\gamma^t R(S_t,\pi(S_t),S_{t+1})\right], \qquad (23.1)$$

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



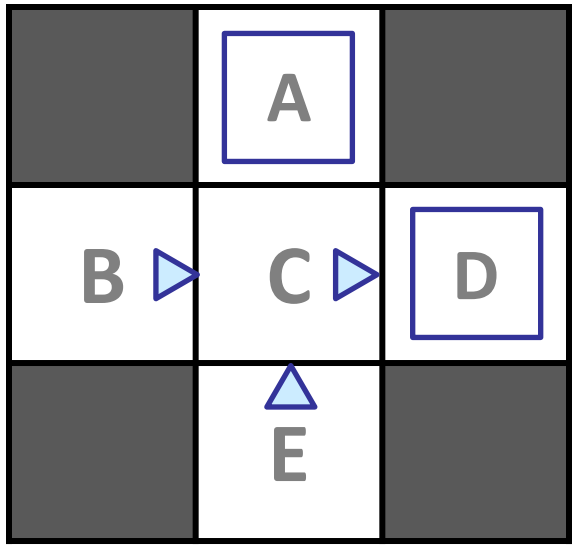
- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

Direct Evaluation

- Goal: Compute values for each state under π
- Idea: Average together observed sample values
  - Act according to π
  - 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 π



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

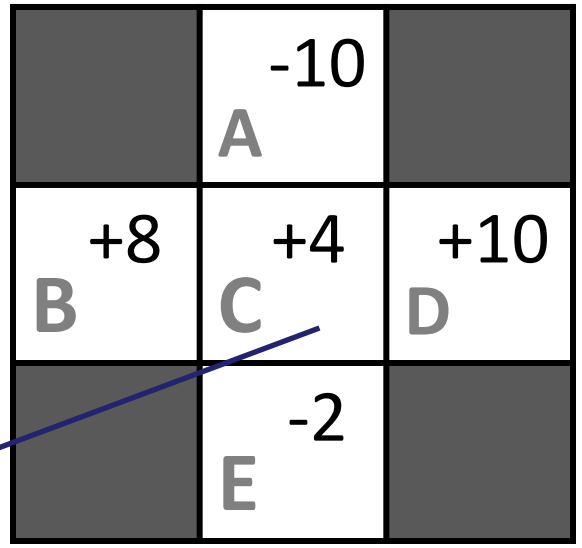
Episode 3

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



- Take the average of rewards from all episodes
- $V^{\pi}(c) = (9+9+9-11)/4$

Episode 1:
From c to d: -1 (one move reward) + 10 (reward in d) = 9

Example: Direct Evaluation

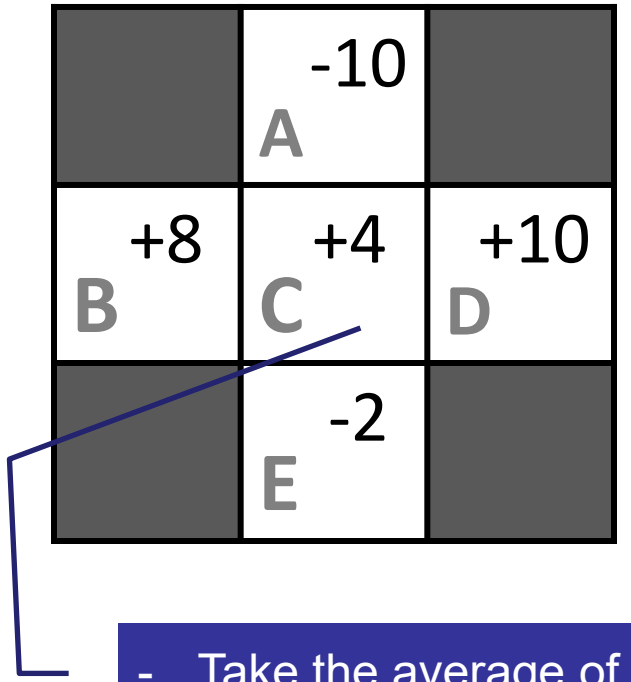
Walking through the first episode, we can see that
- from state D to termination we acquired a total reward of 10,
- from state C we acquired a total reward of (−1)+10=9,
- and from state B we acquired a total reward of (−1)+(−1)+10=8.

Completing this process yields the total reward across episodes for each state and the resulting estimated values as follows:

Though direct evaluation eventually learns state values for each state, it's often unnecessarily slow to converge because it wastes information about transitions between states.

s	Total Reward	Times Visited	Vπ(s)
A	−10	1	−10
B	16	2	8
C	16	4	4
D	30	3	10
E	−4	2	−2

Output Values

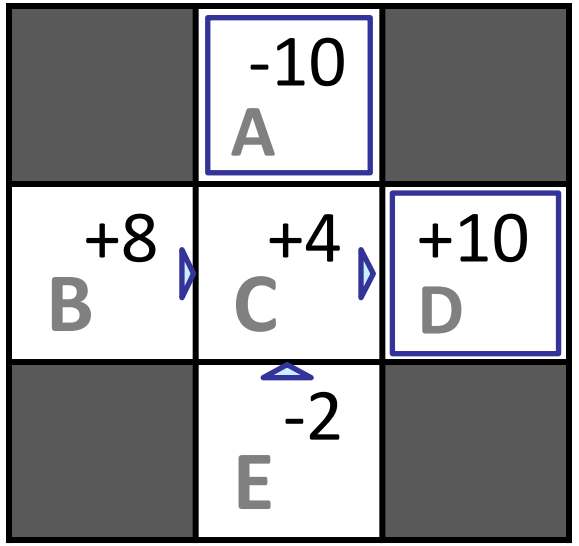


- Take the average of rewards from all episodes
- $V^{\pi}(c) = 9+9+9-11)/4$

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



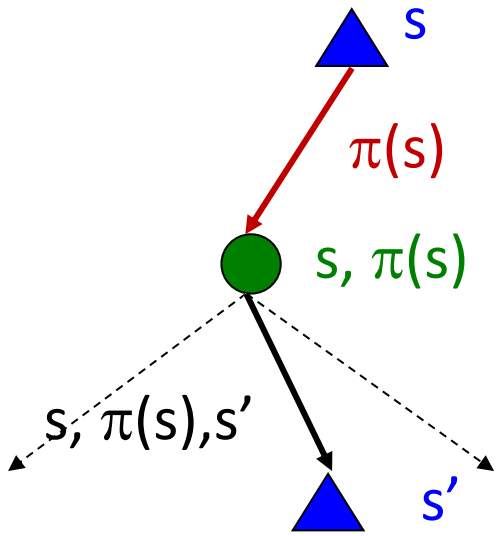
If B and E both go to C under this policy, how can their values be different?

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

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

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



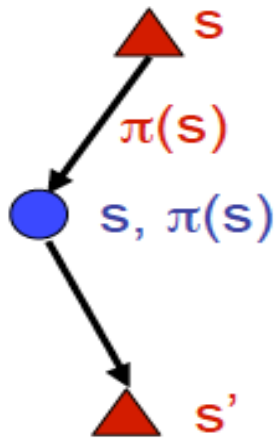
  - This approach fully exploited the connections between the states
  - Unfortunately, we need T and R to do it!

- 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?

Better model-free Learning using sampling

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

- Big idea: why bother learning T?
- Question: how can we compute V if we don't know T?
  - Use direct estimation to sample complete trials, average rewards at end
  - Use sampling to approximate the Bellman updates, compute new values during each learning step



Sample-Based Policy Evaluation?

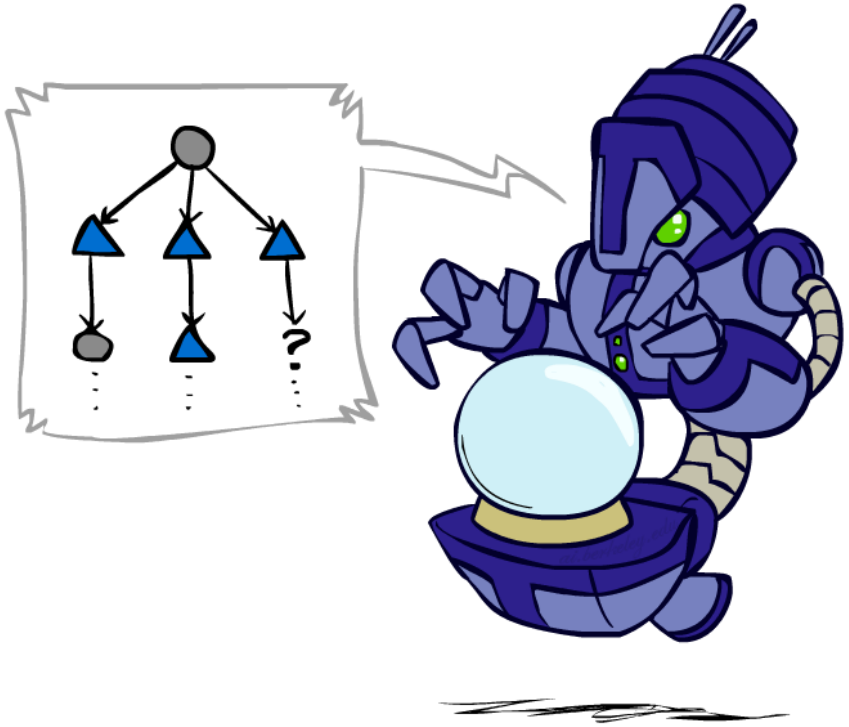
- We want to improve our estimate of V by computing these averages:

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

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

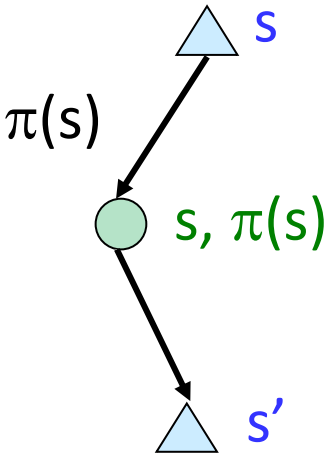
$$sample_1 = R(s, \pi(s), s'_1) + \gamma V^{\pi}_{k}(s'_1)$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V^{\pi}_{k}(s'_2)$$
$$\cdots$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V^{\pi}_{k}(s'_n)$$

$$V^{\pi}_{k+1}(s) \leftarrow \frac{1}{n} \sum_{i} sample_i$$



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

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

This is a running average calculation.

Exponential Moving Average

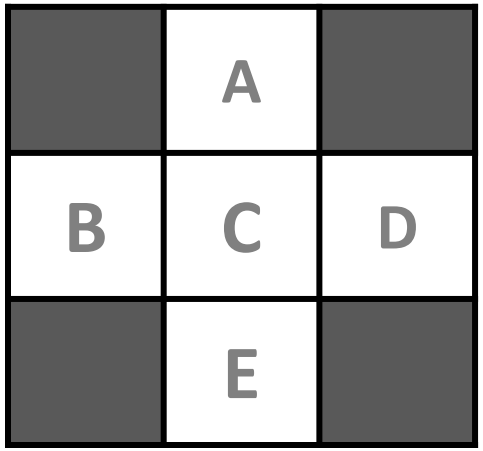
- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$
  - Makes recent samples more important:

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

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

Example: Temporal Difference Learning

States

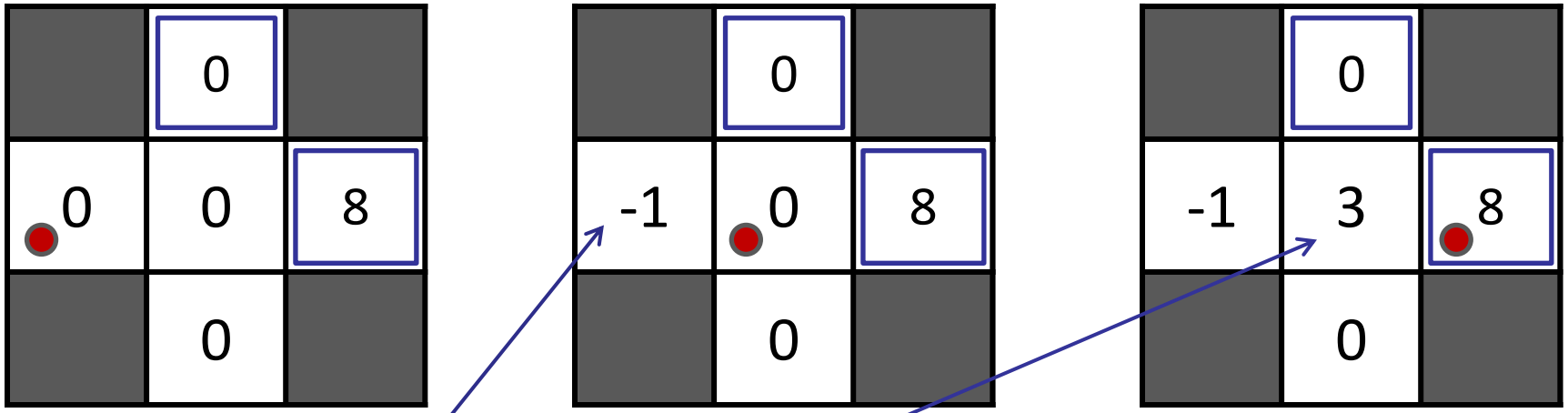


Assume: γ = 1, α = 1/2

Observed Transitions

B, east, C, -2

C, east, D, -2



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

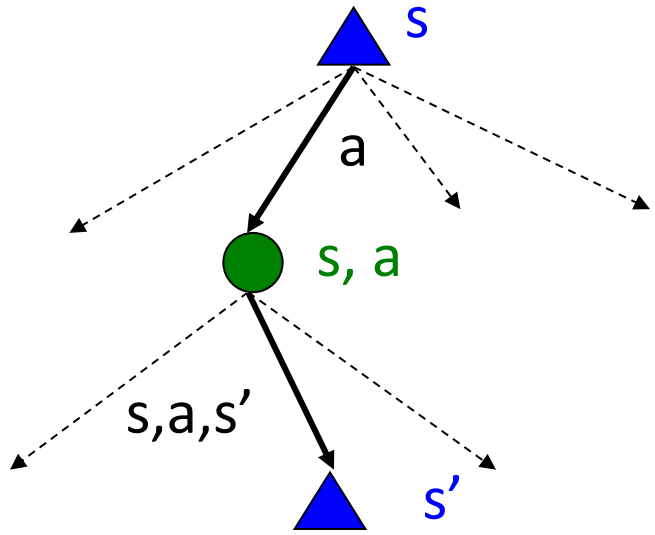
B,east C,-2:	-1	0	1/2	-2	0
C,east D,-2:	3	0	1/2	-2	8

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 V values into a (new) policy, we're sunk. We don't know how to update our policy that maximizes it:

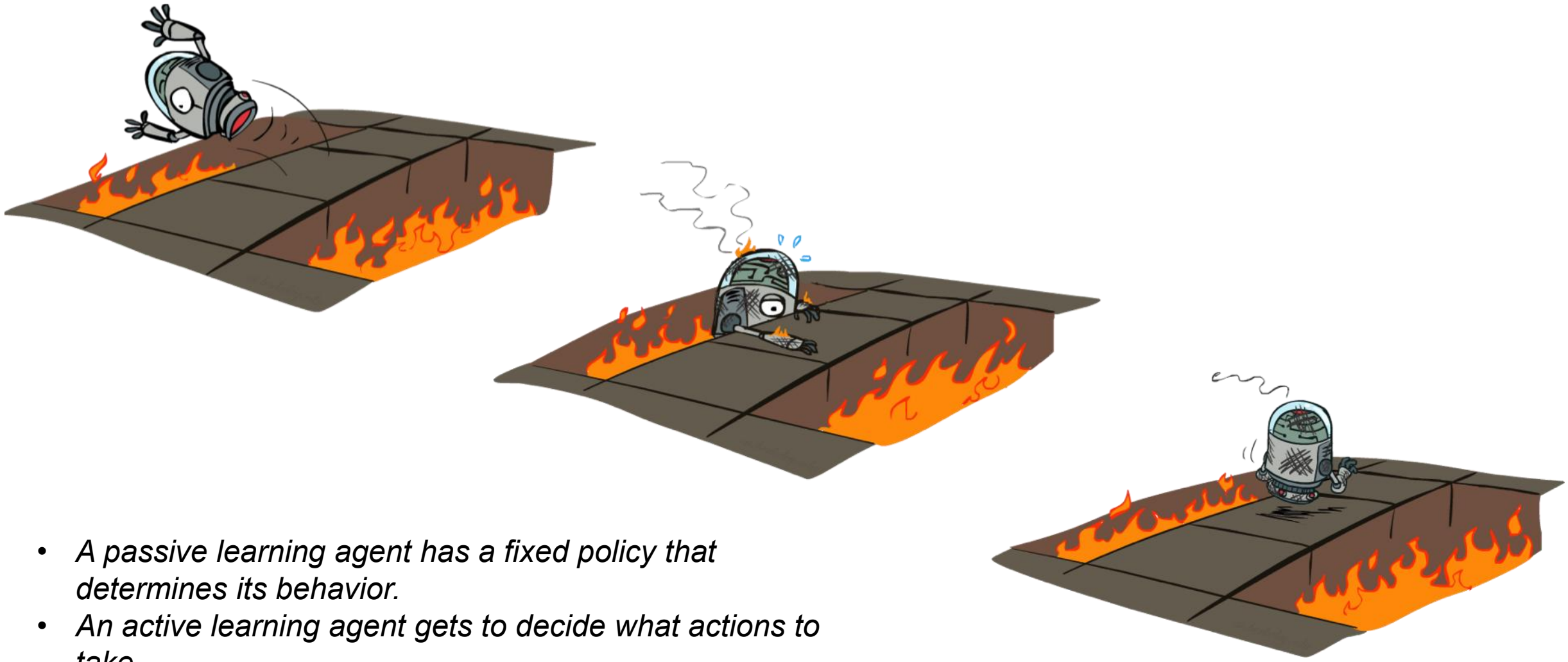
$$\pi(s) = \arg\max_a Q(s,a)$$

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



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

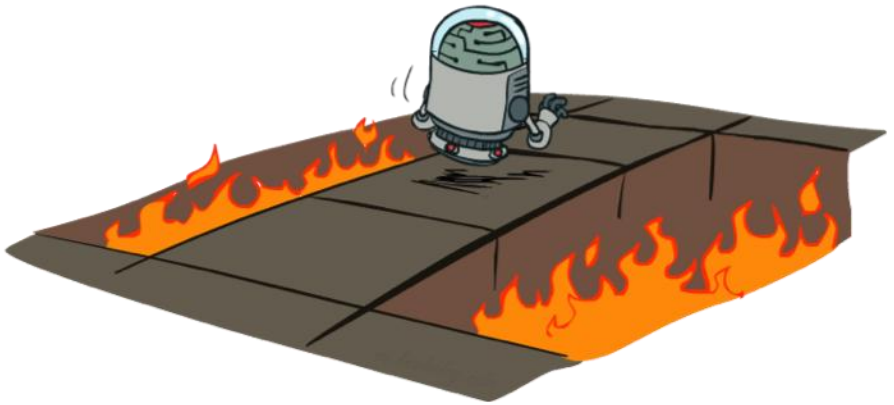
Active Reinforcement Learning



- *A passive learning agent has a fixed policy that determines its behavior.*
- *An active learning agent gets to decide what actions to take*

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…

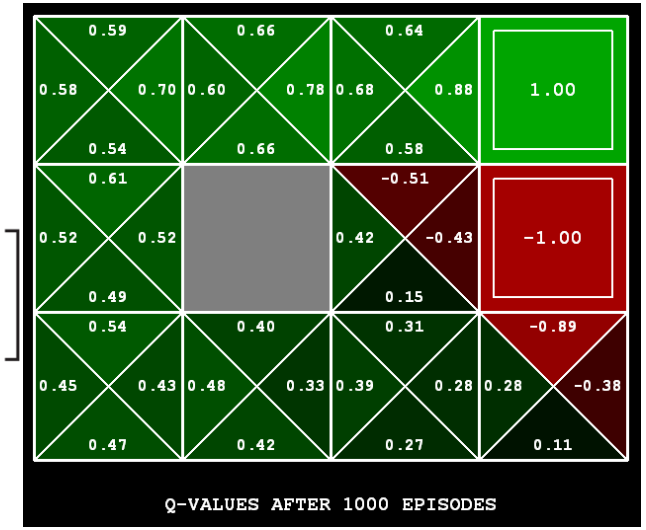
Detour: Q-Value Iteration

- 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:

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

- 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

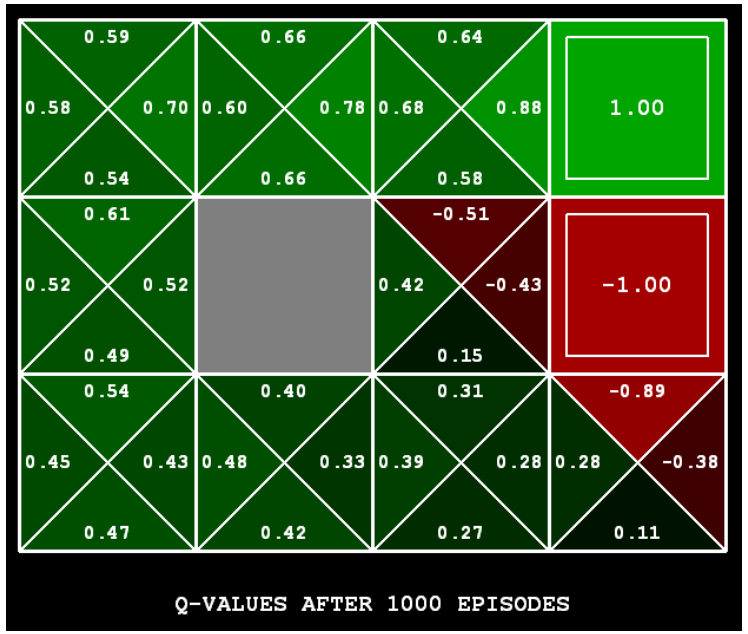
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- 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:

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

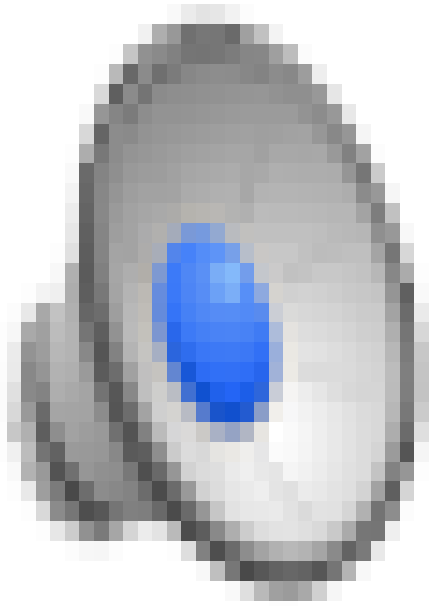
  - Incorporate the new estimate into a running average:

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

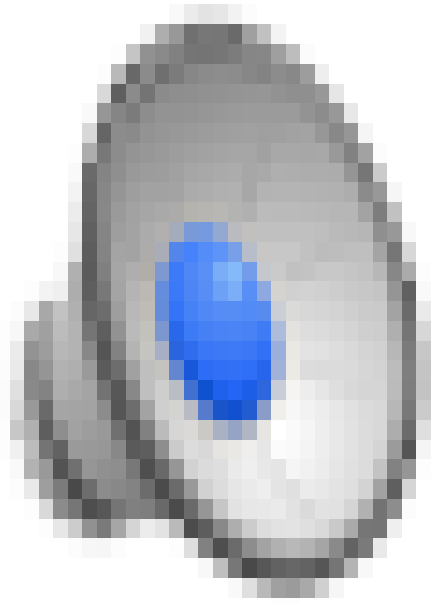


[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

Video of Demo Q-Learning -- Gridworld



Video of Demo Q-Learning -- Crawler



The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

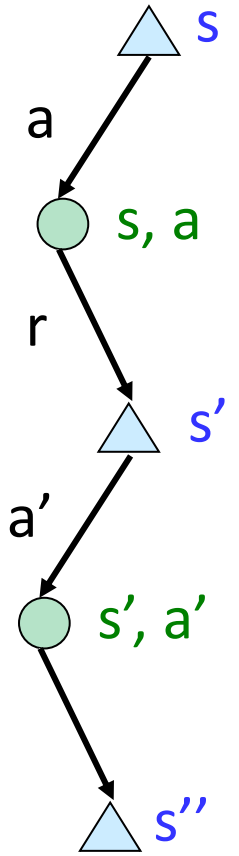
Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	Value Learning

Model-Free Learning

- Model-free (temporal difference) learning
  - Experience world through episodes (experiences)

$$(s, a, r, s', a', r', s'', a'', r'', s'''' \ldots)$$

  - Update estimates each transition (s, a, r, s')
  - Over time, updates will mimic Bellman updates



Stream of Experiences

Q-Learning

- We'd like to do Q-value updates to each Q-state (Bellman equation for Q-values):

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

  - But can't compute this update without knowing T, R

- Instead, compute average as we go
  - Receive a sample transition (s,a,r,s')
  - This sample suggests

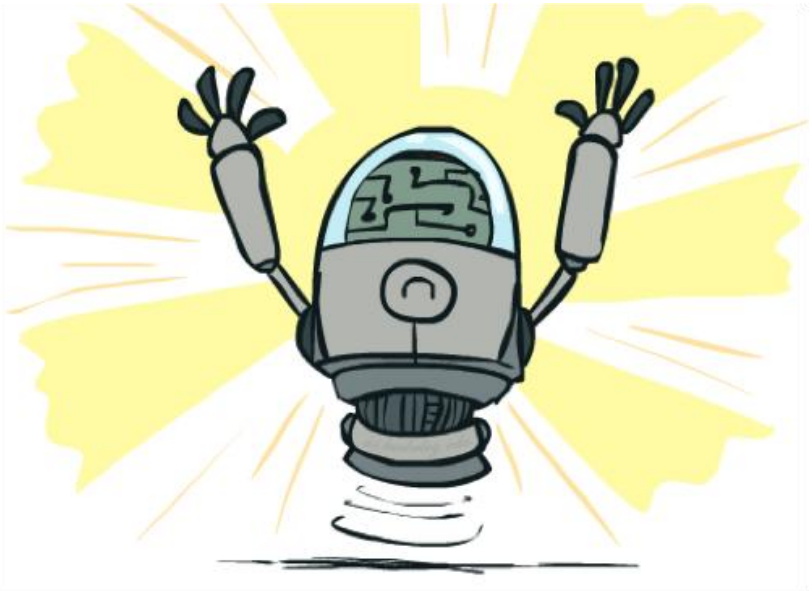
$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

  - But we want to average over results from (s,a) (Why?)
  - So keep a running average

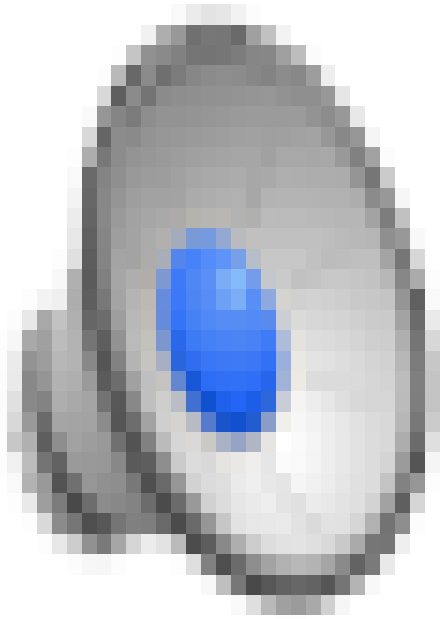
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha) \left[r + \gamma \max_{a'} Q(s',a') \right]$$

Q-Learning Properties

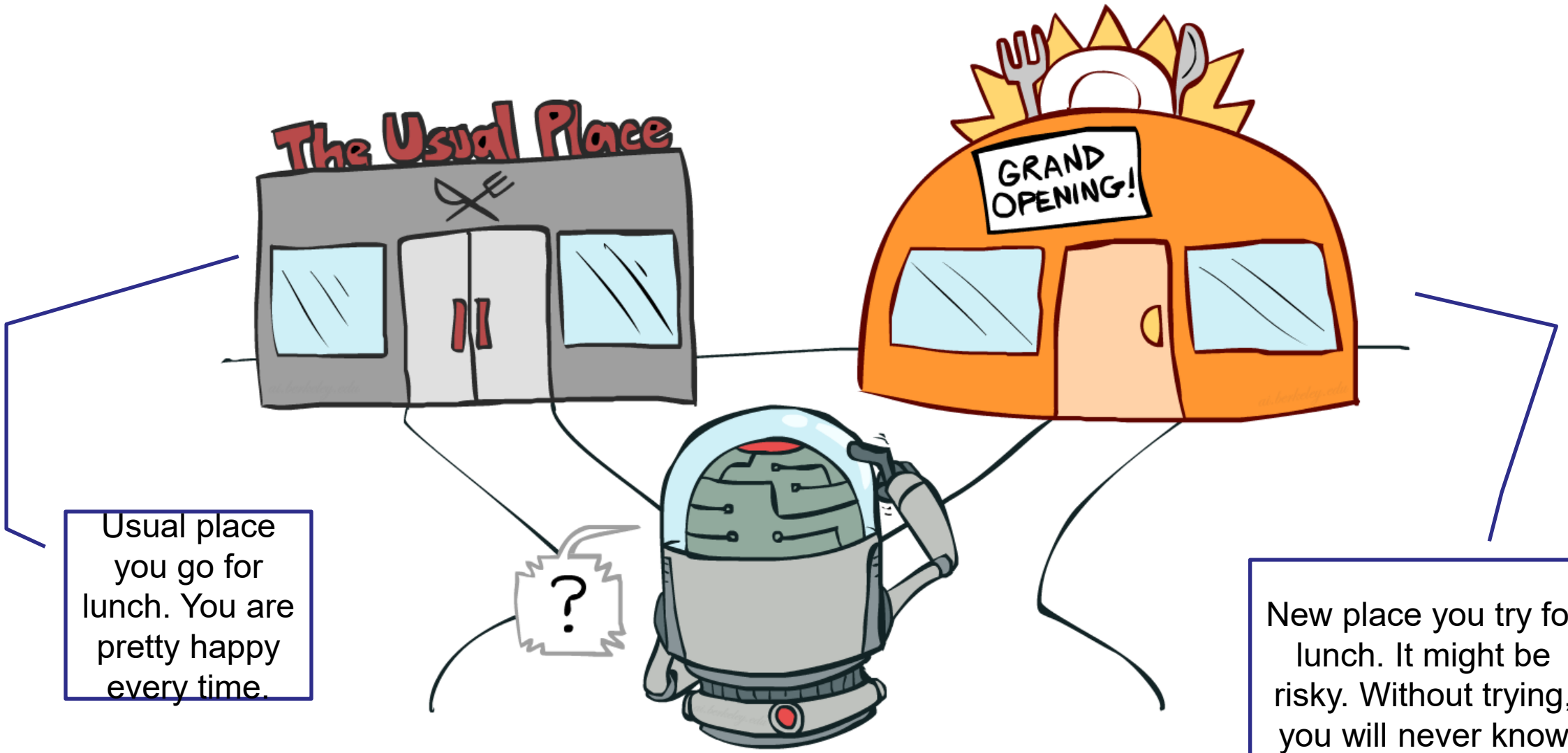
- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit,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



Video of Demo Q-Learning Auto Cliff Grid



Exploration vs. Exploitation

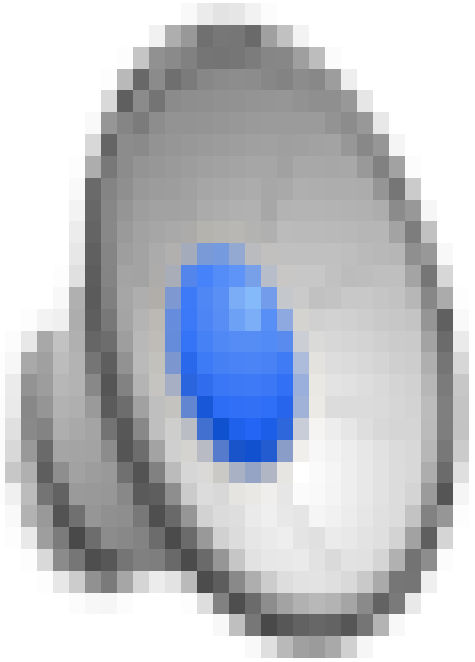


How to Explore?

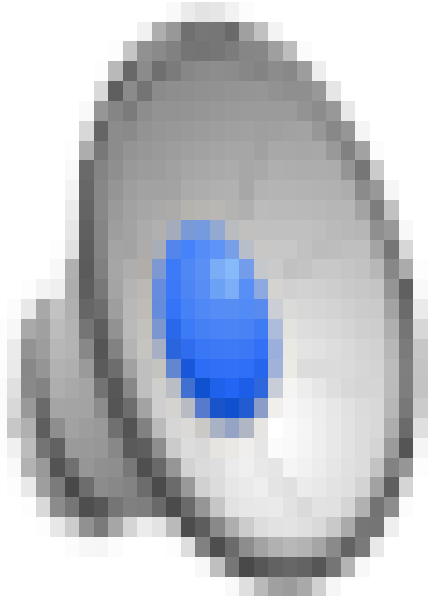
- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy
  - Problems with random actions?
    - You do eventually explore the space, but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

[Demo: Q-learning – manual exploration – bridge grid (L11D2)]
[Demo: Q-learning – epsilon-greedy -- crawler (L11D3)]

Video of Demo Q-learning – Epsilon-Greedy – Crawler



Video of Demo Q-learning – Manual Exploration – Bridge Grid



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 functions help us decay the exploration automatically.
- Exploration function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u, n) = u + k/n$

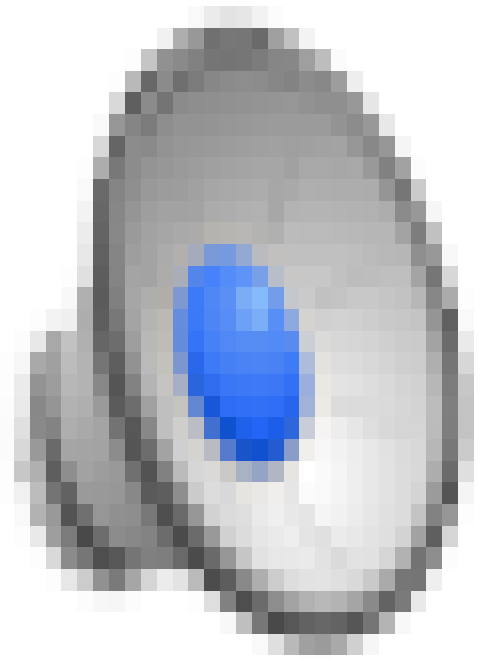
Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

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

- The term k/n (bonus) will decrease when you visit a state more.
- Note: this propagates the "bonus" back to states that lead to unknown states as well!

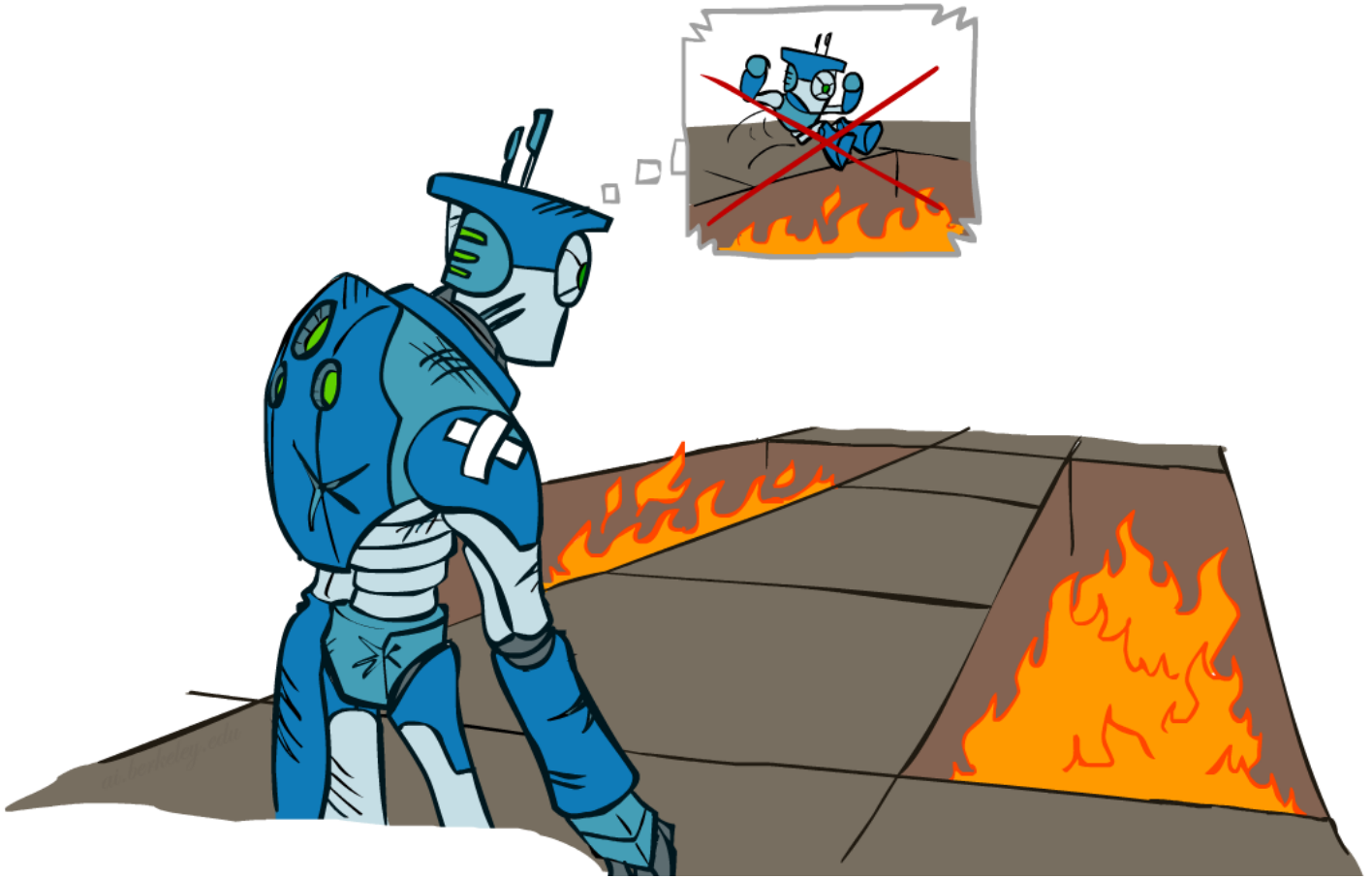
[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

Video of Demo Q-learning – Exploration Function – Crawler



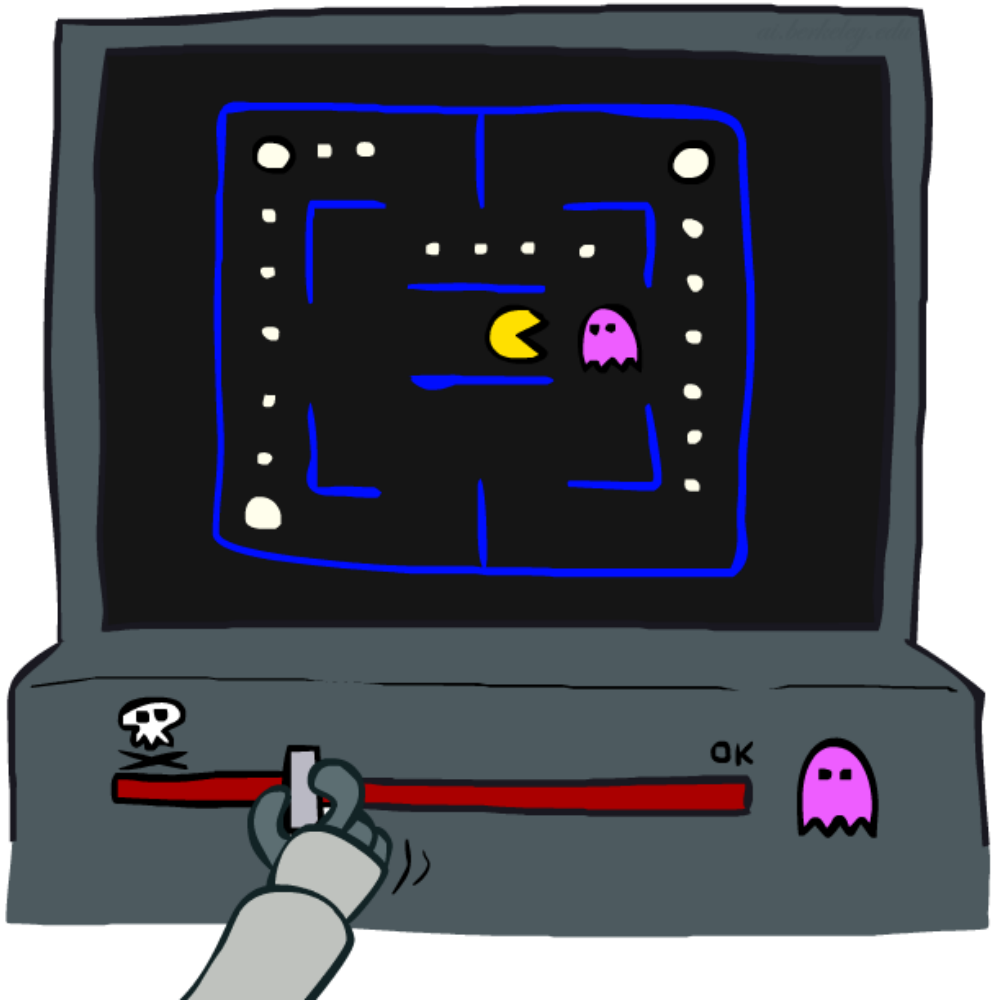
Regret

- 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



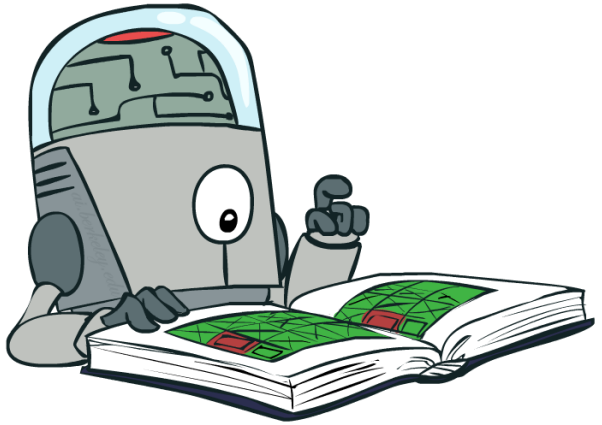
Approximate Q-Learning

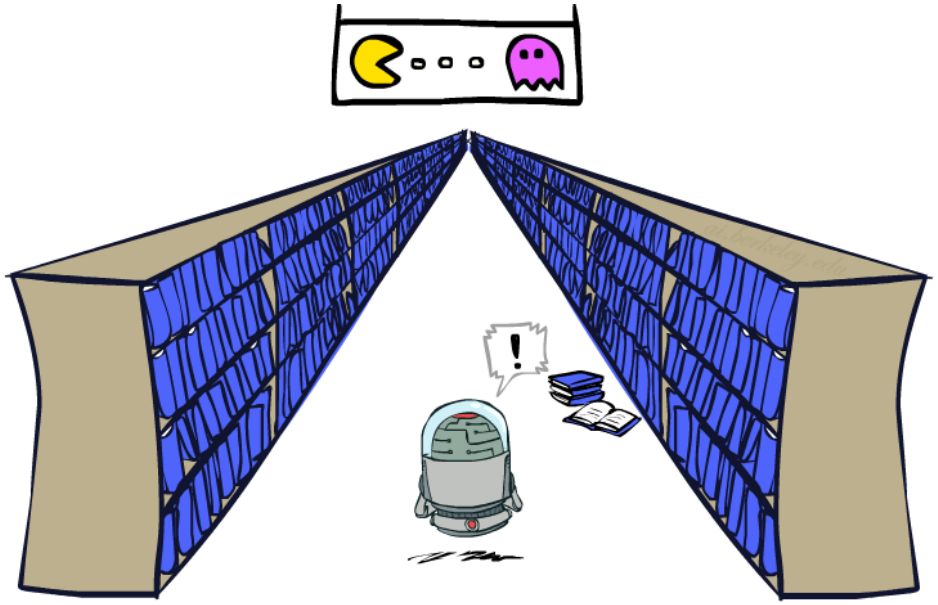
- So far we've looked at exact methods-- that is methods that give us the exact solution we need if we run it long enough.
- Now we're going to look at ways of approximating the solution.



Generalizing Across States

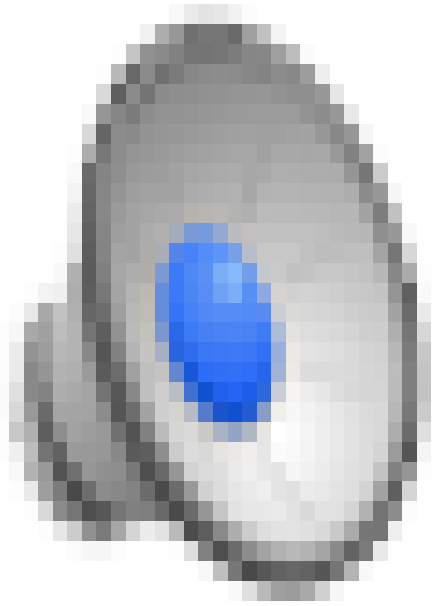
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again.



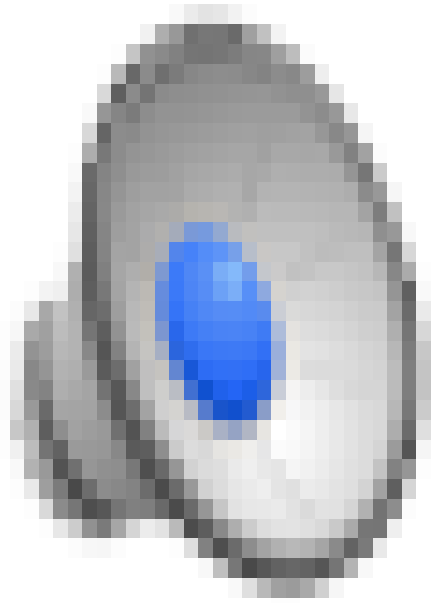


[demo – RL pacman]

Video of Demo Q-Learning Pacman – Tiny – Watch All

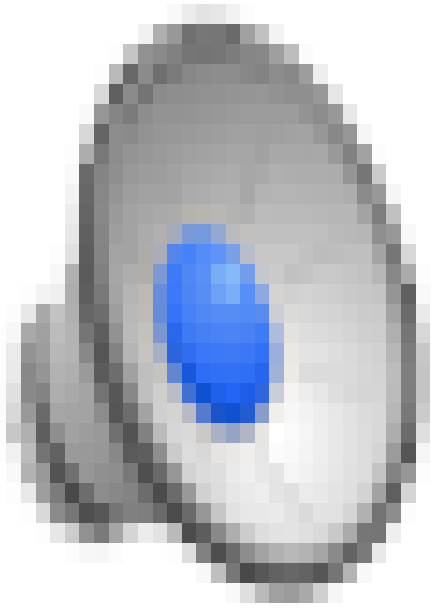


Video of Demo Q-Learning Pacman – Tiny – Silent Train



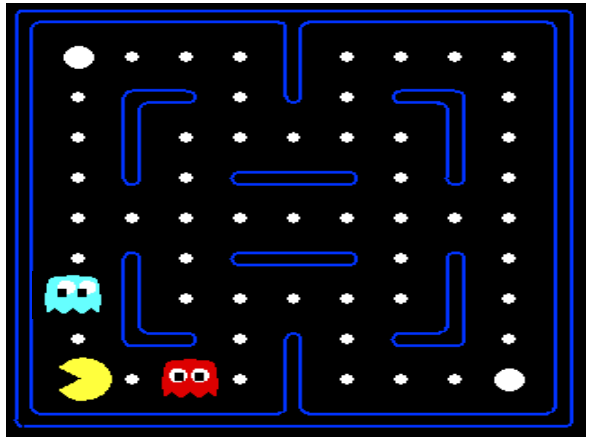
Video of Demo Q-Learning Pacman – Tricky – Watch All

-You have discovered that some of states are bad.
-You have learned nothing about bad entry from your past experience, if you use regular tabular Q-learning

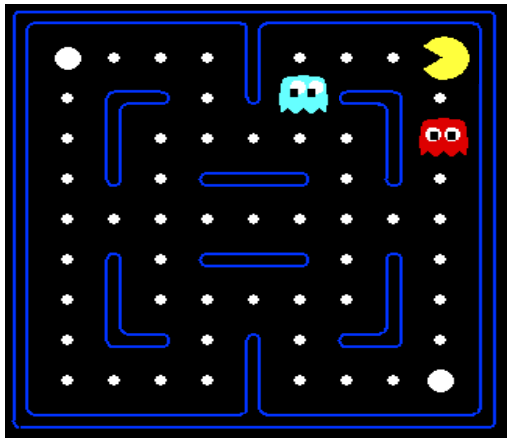


Example: Pacman

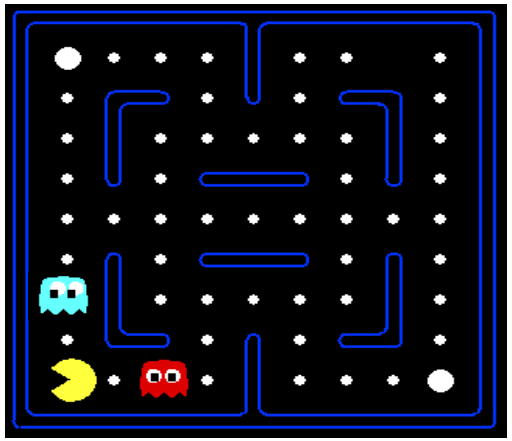
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:



Or even this one!

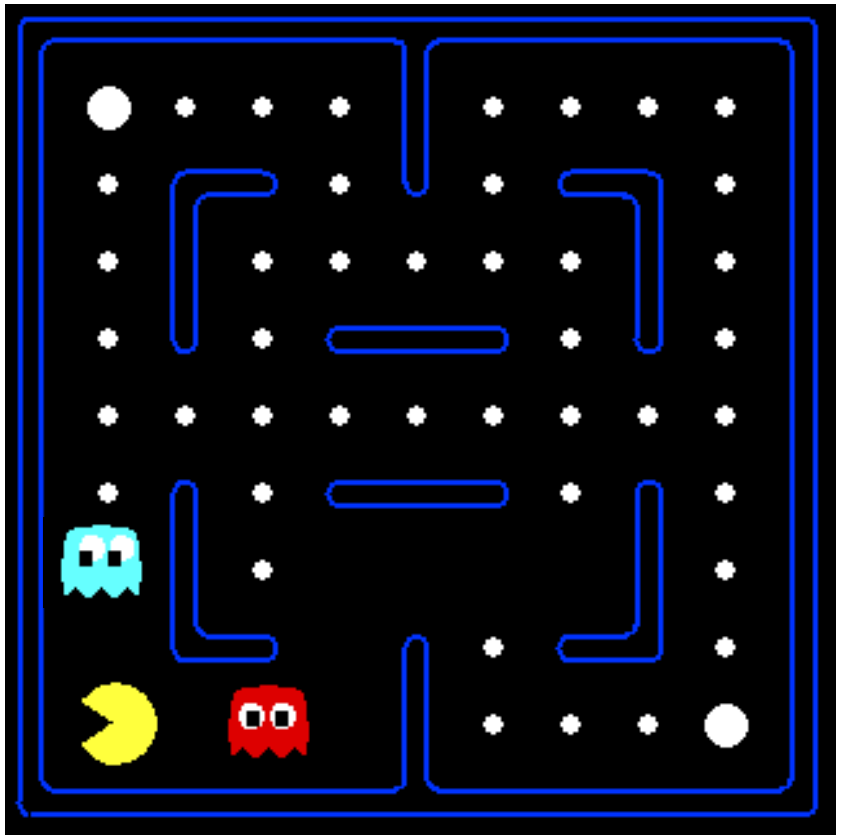


- Above, if Pacman learned that Figure 1 is unfavorable after running vanilla Q-learning, it would still have no idea that Figure 2 or even Figure 3 are unfavorable as well.
- **Approximate Q-learning** tries to account for this by learning about a few general situations and extrapolating to many similar situations.
- The key to generalizing learning experiences is the **feature-based representation** of states, which represents each state as a vector known as a **feature vector**.

[Demo: Q-learning – pacman – tiny – watch all (L11D5)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D7)]

Feature-Based Representations

- You have to learn from experiences.
- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - $1 / (\text{dist to dot})^2$
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value Functions

- Our V or Q function is a weighted sum of feature values.
- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very few and different in value!

Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

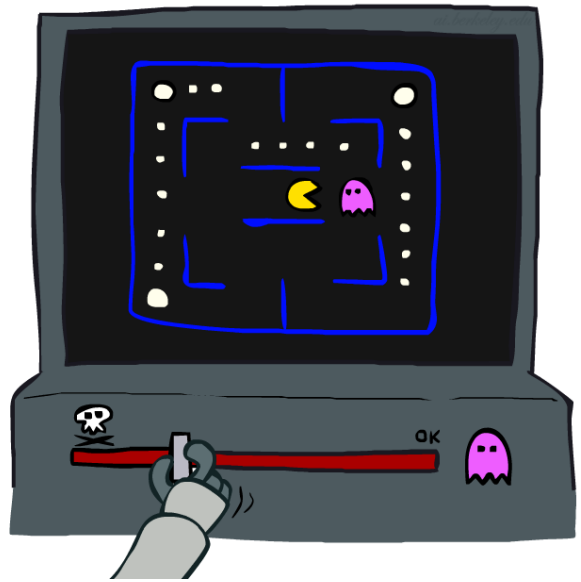
- Q-learning with linear Q-functions:

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}] \qquad \text{Exact Q's}$$

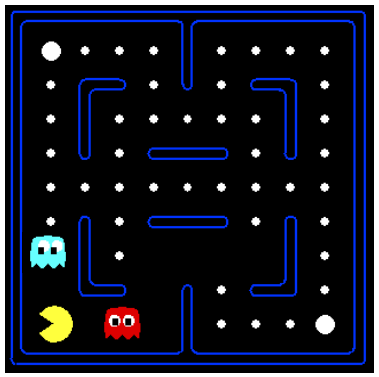
$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a) \qquad \text{Approximate Q's}$$



- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features
- Formal justification: online least squares

Example: Q-Pacman

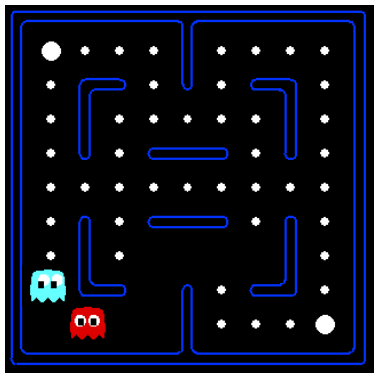
$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$ 2 features: dots and distance to ghost (SGT)

s 

$$f_{DOT}(s, \text{NORTH}) = 0.5$$

$$f_{GST}(s, \text{NORTH}) = 1.0$$

$$a = \text{NORTH}$$
$$r = -500$$

s' 

$$Q(s, \text{NORTH}) = +1$$

$$Q(s', \cdot) = 0$$

$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

difference $= -501$

$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

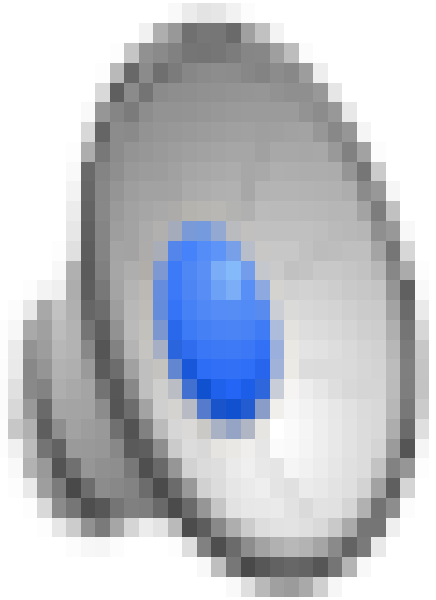
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

[Demo: approximate Q-learning pacman (L11D10)]

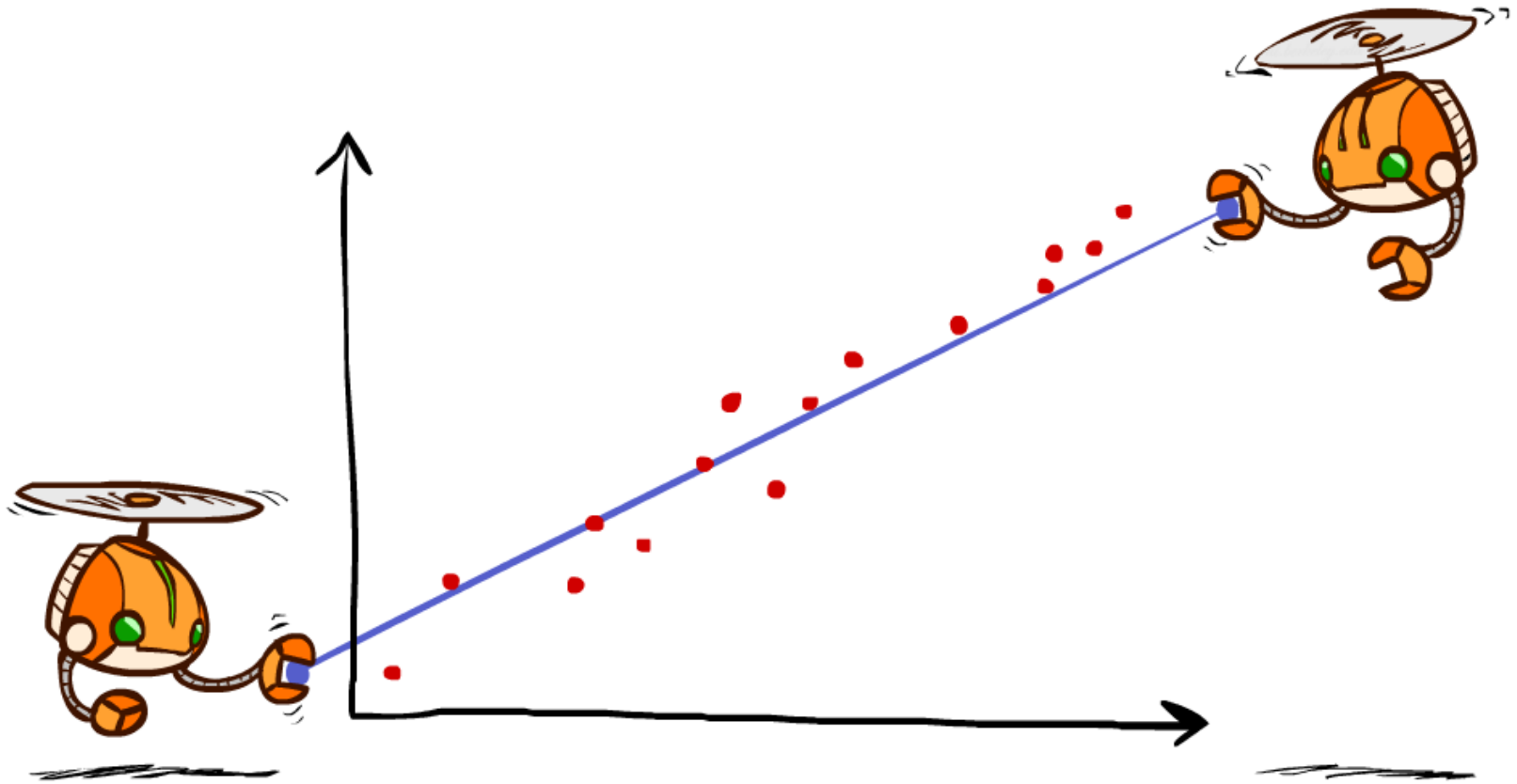
Video of Demo Approximate Q-Learning -- Pacman

-We see that maybe with about 10 episodes so far, it's already playing pretty well.

-Compare this with tabular Q-learning where we need hundreds if not thousands of episodes before we learned anything.

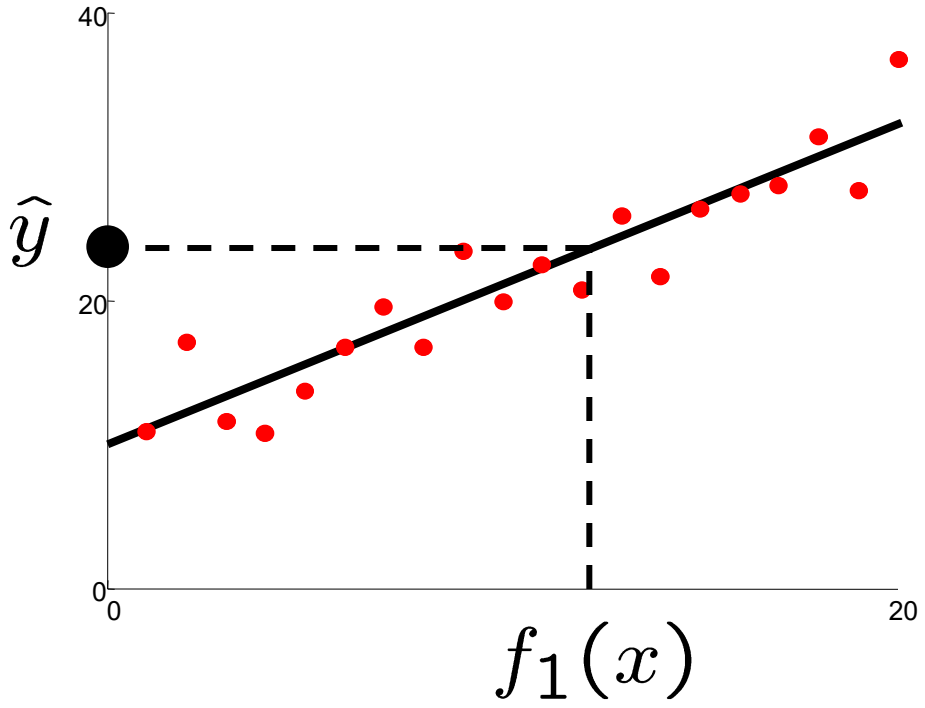


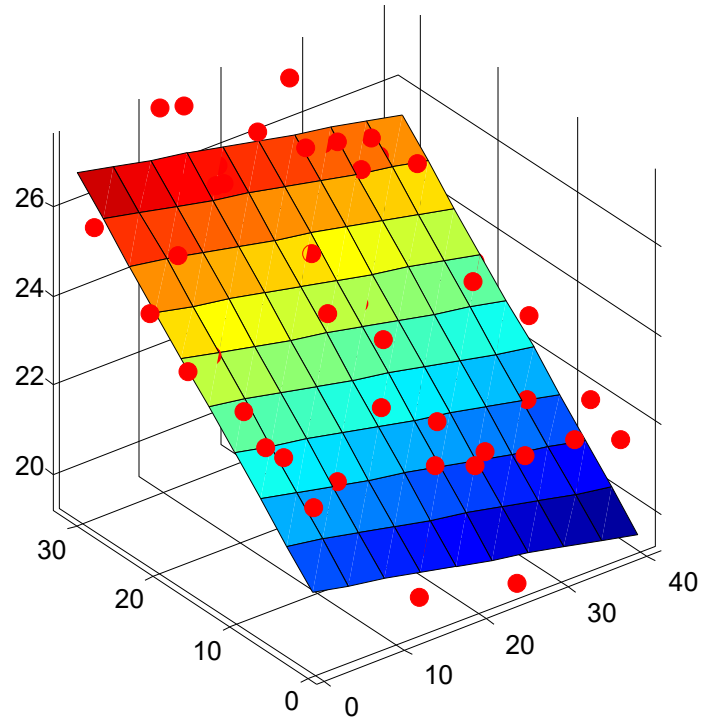
Q-Learning and Least Squares



- Q-Learning is a *reinforcement learning algorithm* used to train agents through trial-and-error interactions with an environment,
- while the Least Squares method is a *mathematical optimization technique* used in regression analysis to find the best-fit line or curve by minimizing the sum of squared errors between observed and predicted values

Linear Approximation: Regression*





Prediction:

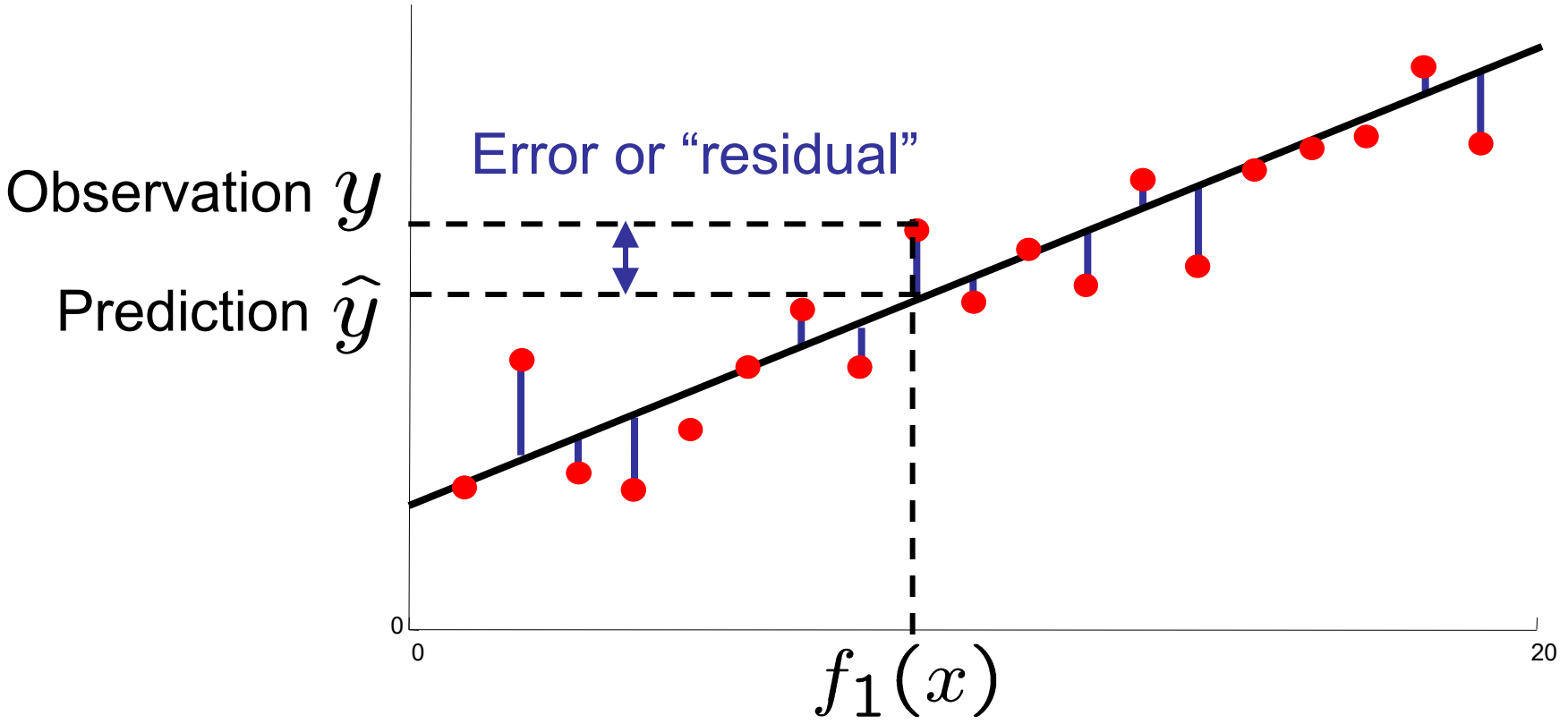
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

$$\hat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$$

Optimization: Least Squares*

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$



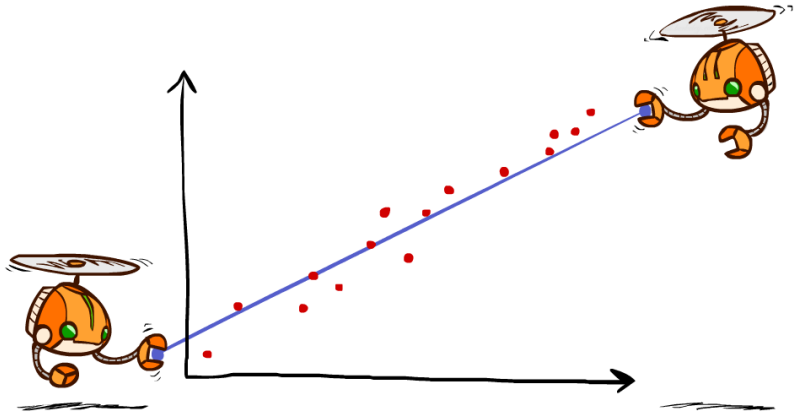
Minimizing Error*

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$



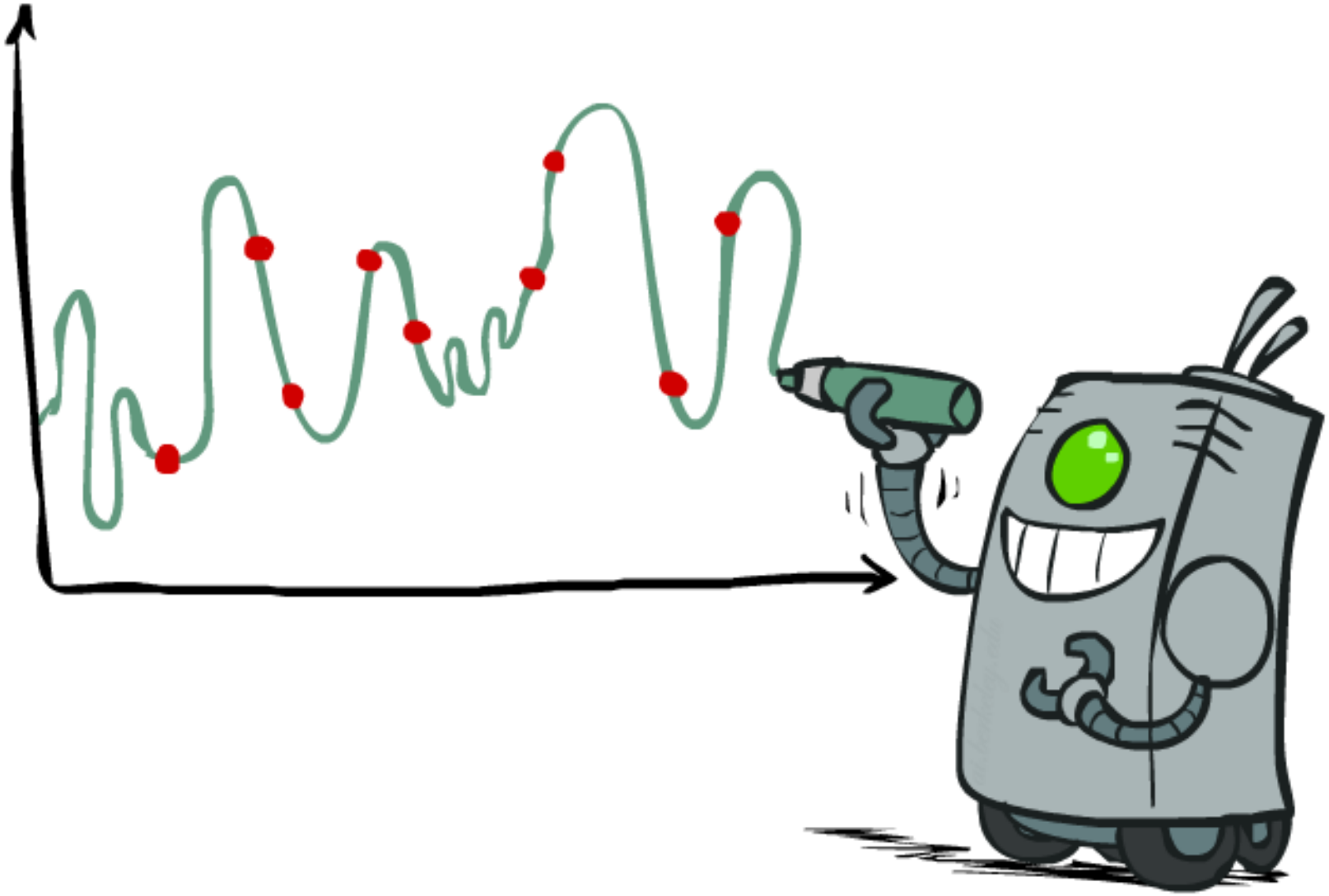
Approximate q update explained:

$$w_m \leftarrow w_m + \alpha\left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

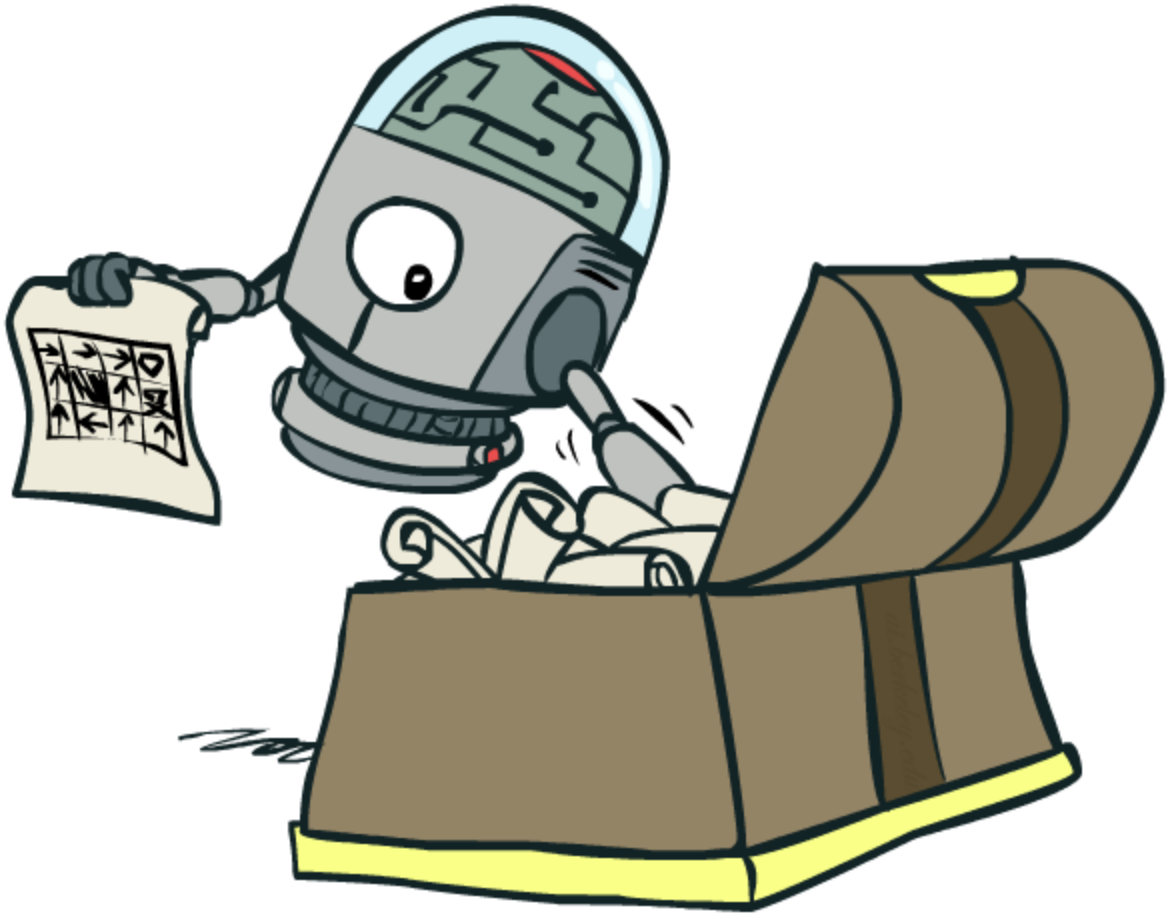
"target" "prediction"

- In essence, Q-Learning is about learning how to act, while Least Squares is about learning how to predict.
- Error minimization is also central to Q-Learning: every update step tries to reduce error

Overfitting: Why Limiting Capacity Can Help*



Policy Search



Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
  - We'll see this distinction between modeling and prediction again later in the course

- Solution: learn policies that maximize rewards, not the values that predict them

- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical
- Better methods exploit lookahead structure, sample wisely, change multiple parameters…

Conclusion

- We're done with Part I: Search and Planning!
- We've seen how AI methods can solve problems in:
  - Search
  - Constraint Satisfaction Problems
  - Games
  - Markov Decision Problems
  - Reinforcement Learning