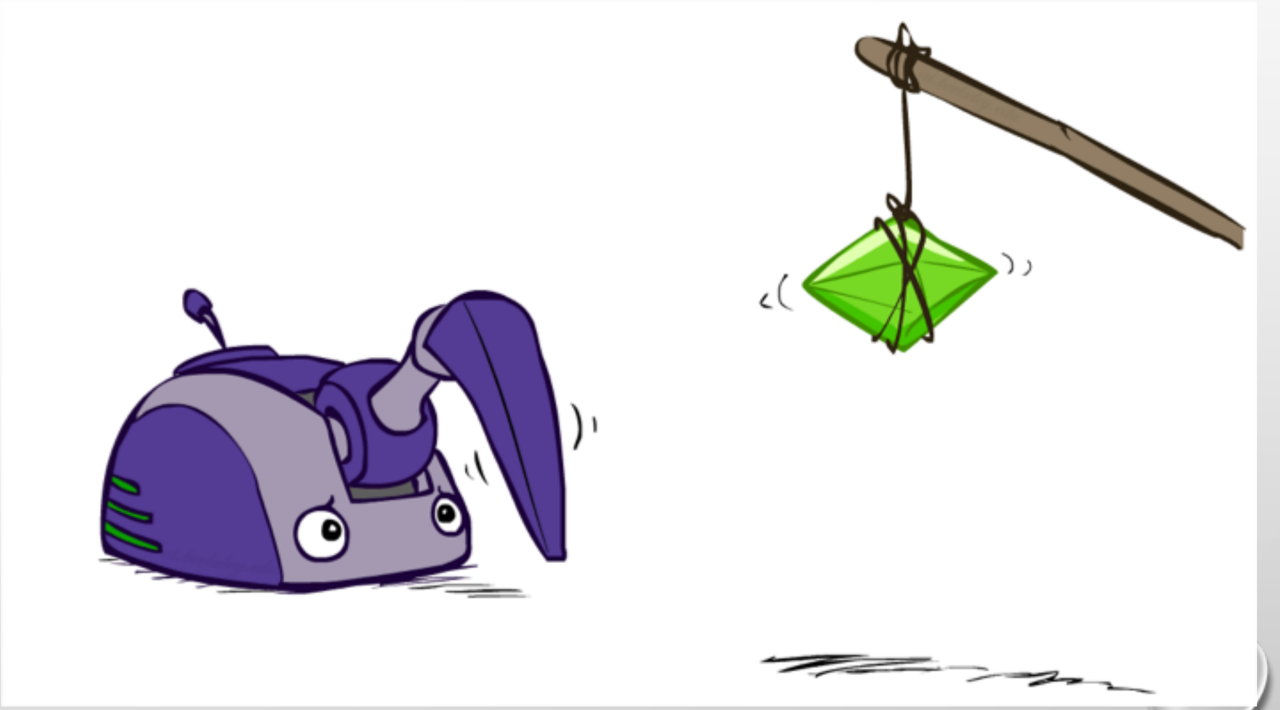
# Artificial Intelligence
# CE-417, Group 2
# Computer Eng. Department
# Sharif University of Technology

Fall 2020

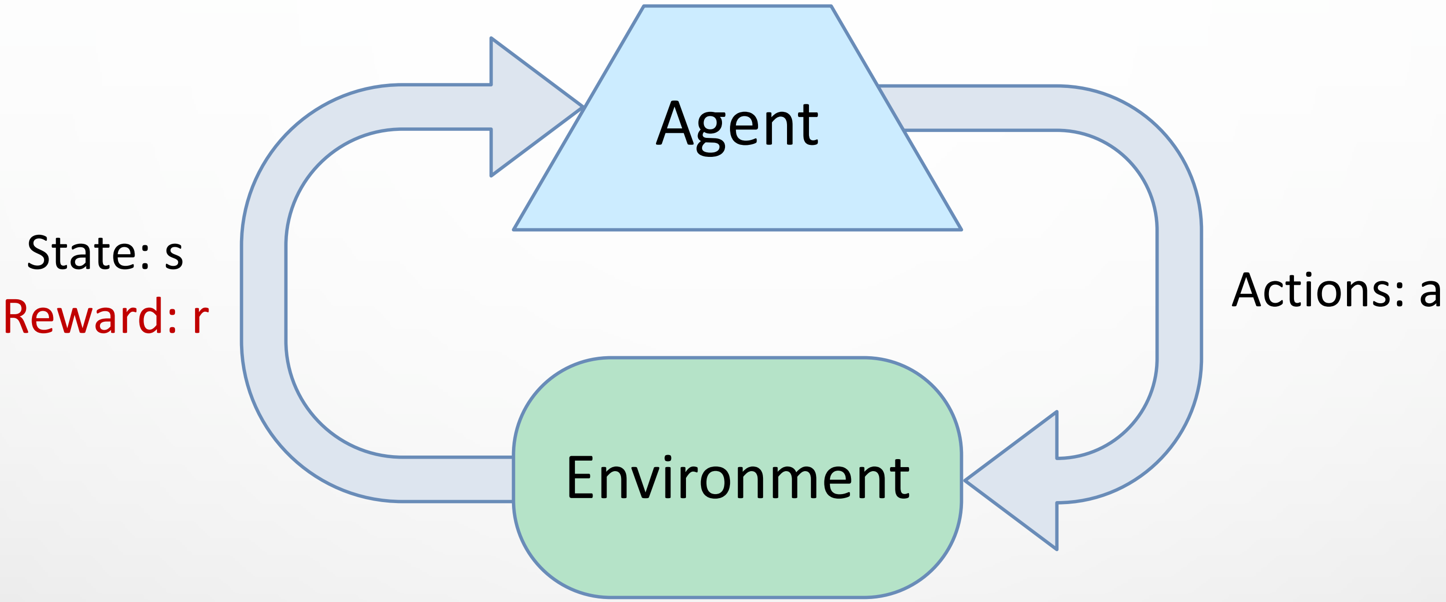
By Mohammad Hossein Rohban, Ph.D.

Courtesy: Most slides are adopted from CSE-573 (Washington U.), original slides for the textbook, and CS-188 (UC. 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]

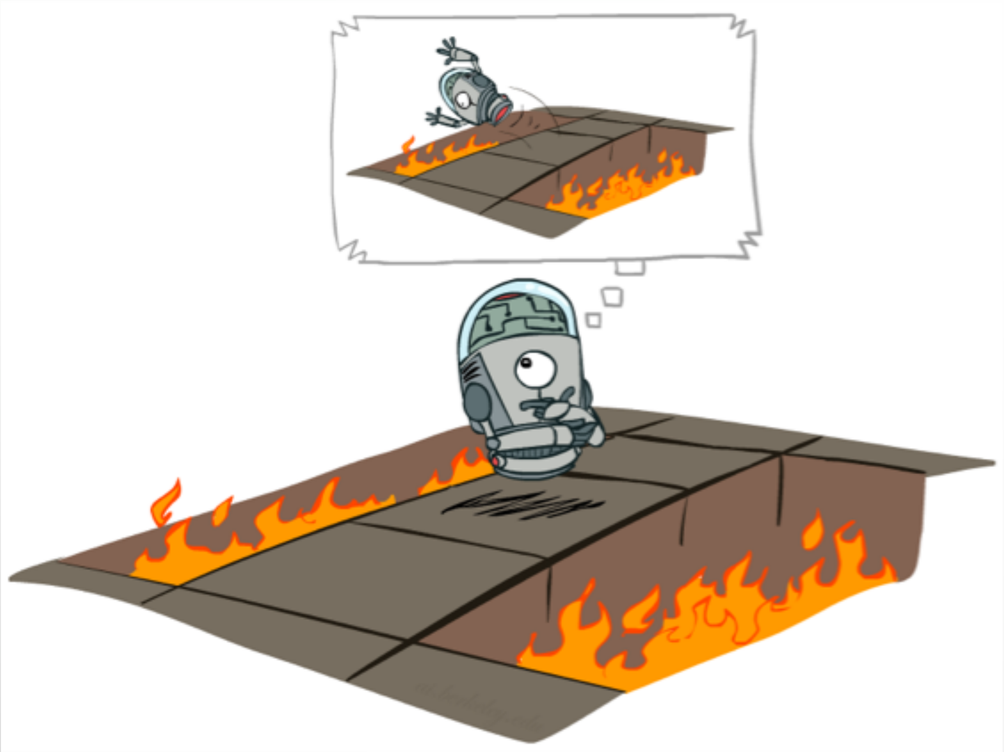
# Video of Demo Crawler Bot

# Reinforcement Learning

- Still assume a markov decision process (MDP):
  - A set of states s ∈ S
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy $\pi$(s)


- New twist: don't know T or R
  - i.e. We don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

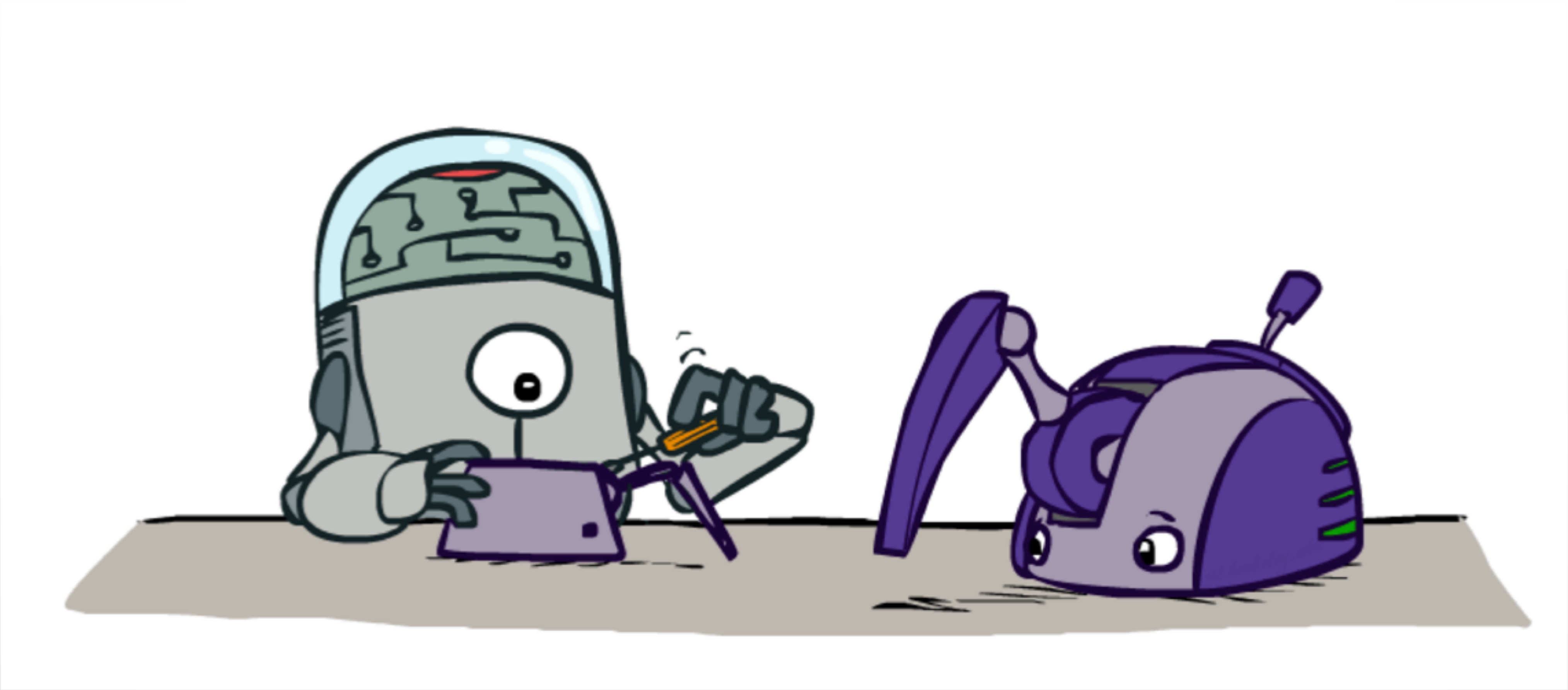
# Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

# Model-Based Learning

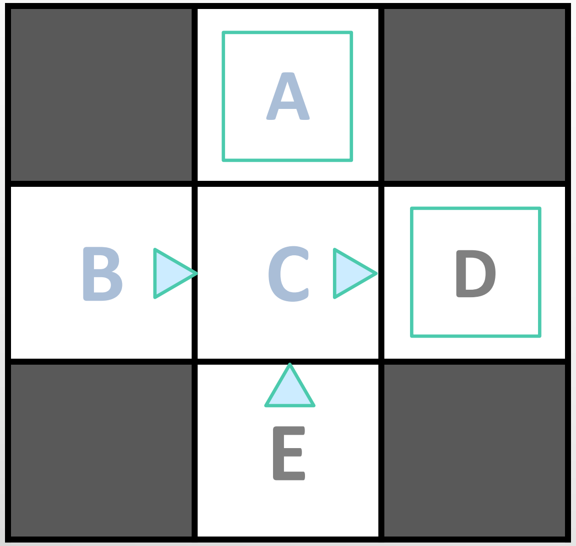


# Model-Based Learning

- Model-based idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct
- Step 1: learn empirical MDP model
  - Count outcomes s' for each s, a
  - Normalize to give an estimate of $\widehat{T}(s, a, s')$
  - 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 $\pi$




*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, 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
…

# Example: 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"**

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

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

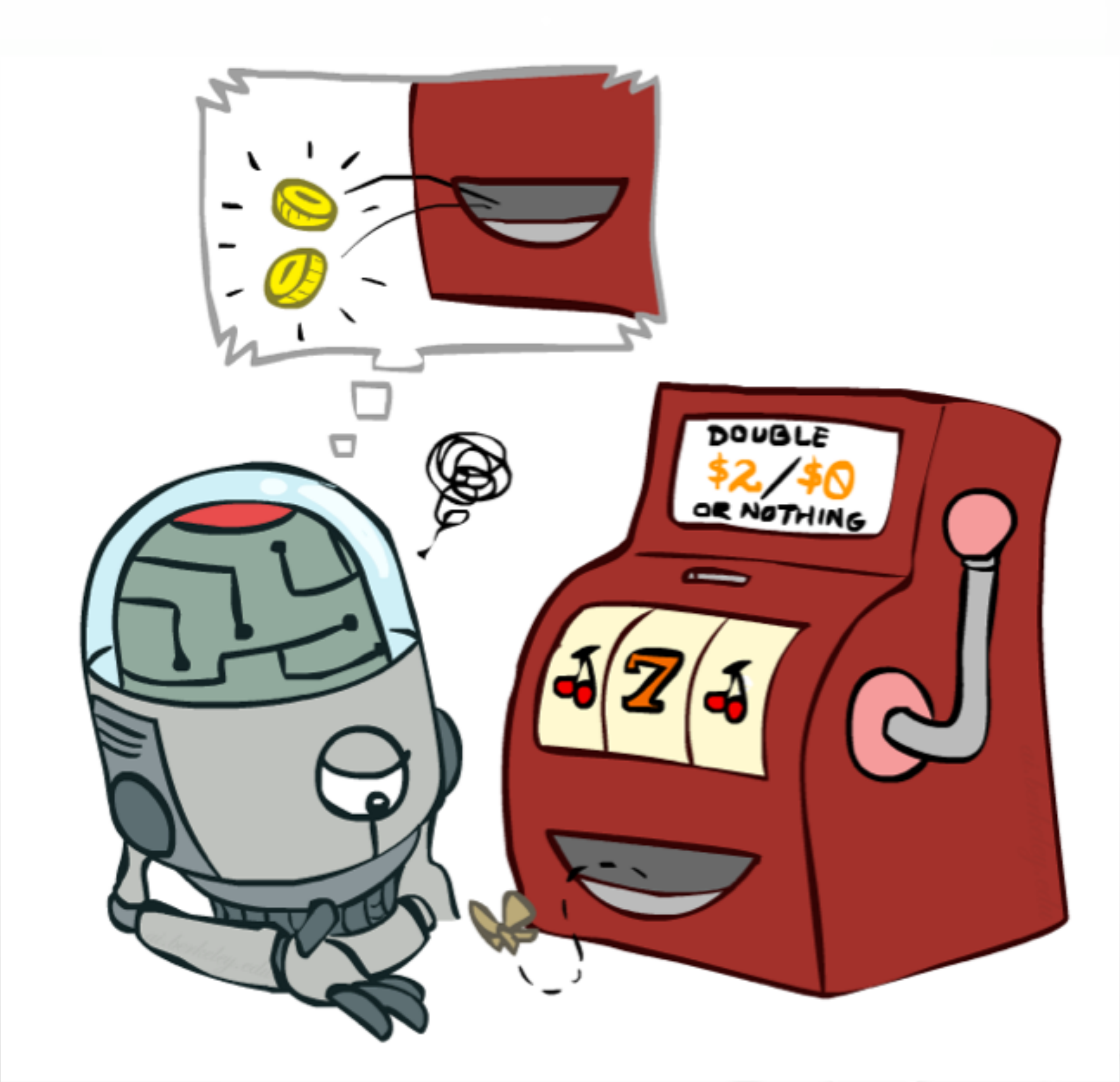
Why does this work? Because eventually you learn the right model.

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

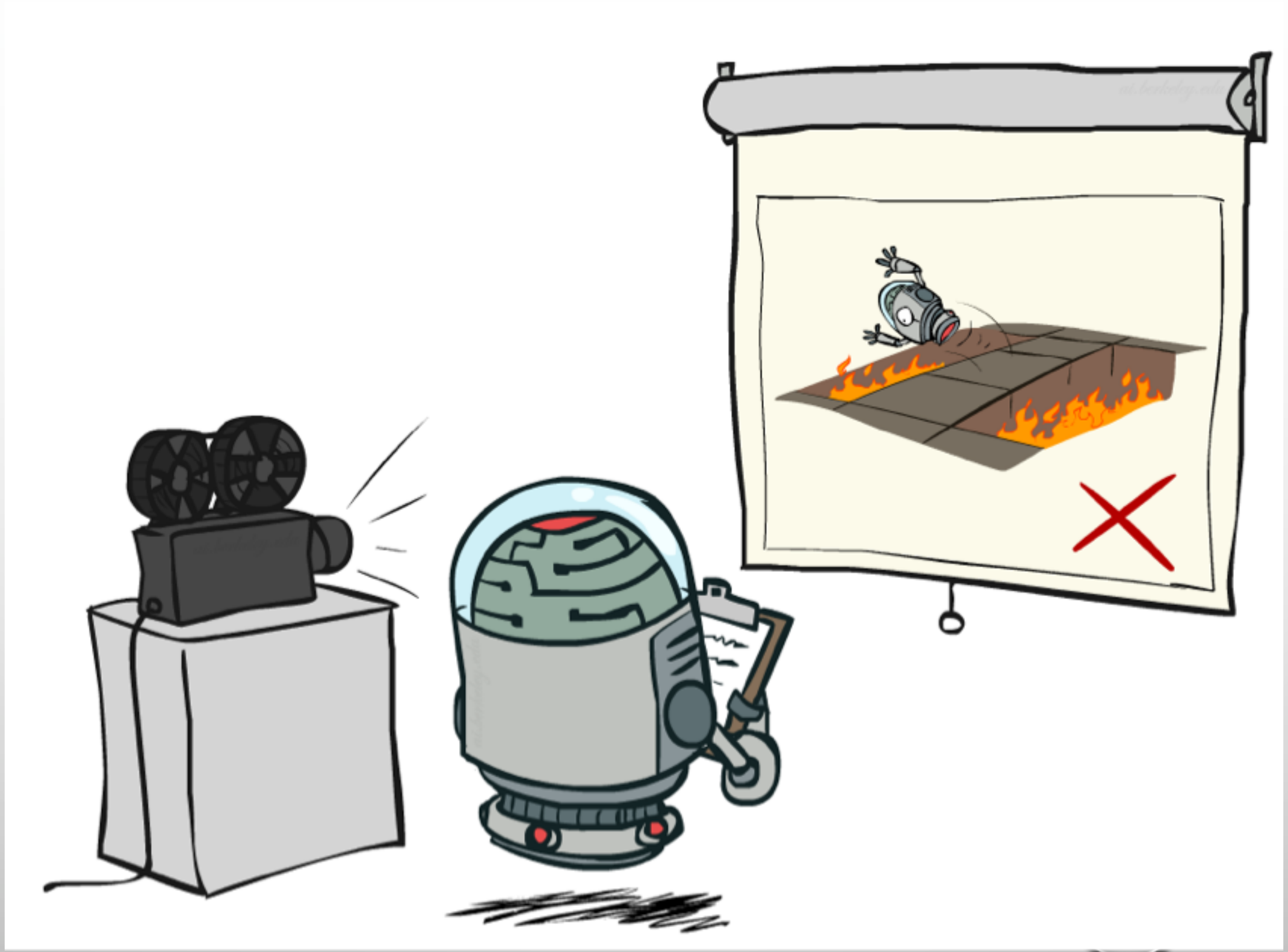
$$E[A] \approx \frac{1}{N} \sum_i a_i$$

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

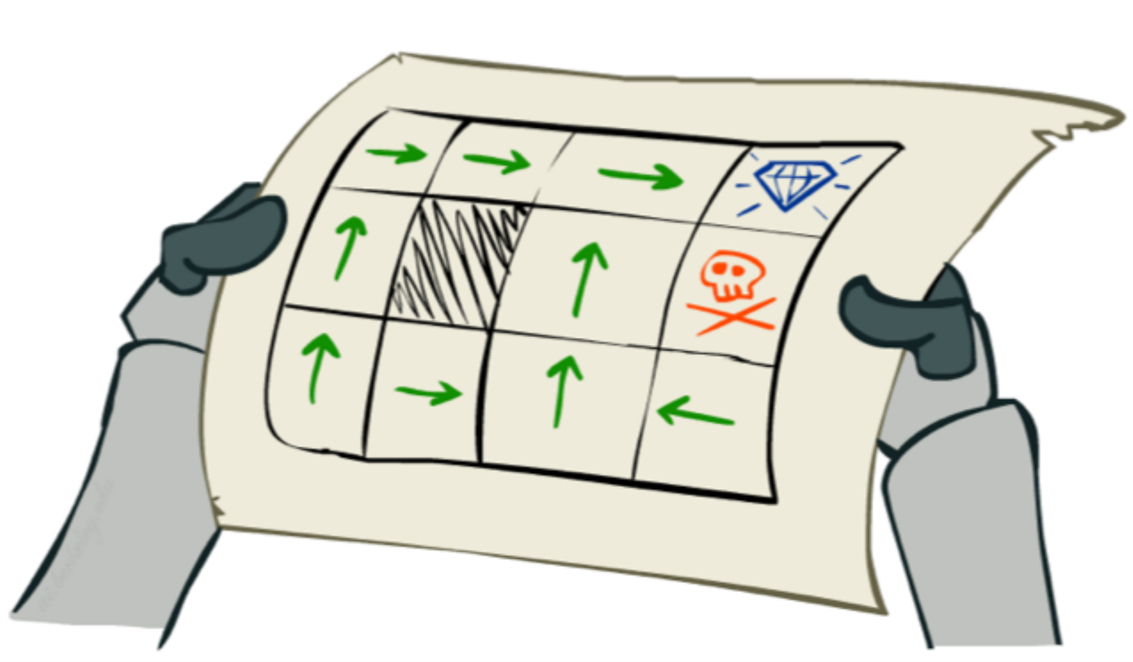
# Model-Free Learning



# Passive Reinforcement Learning



# Passive Reinforcement Learning



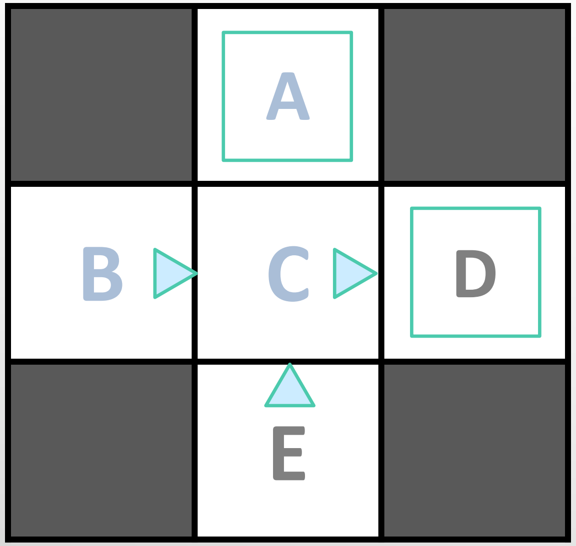
- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - Goal: learn the state values
- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute 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

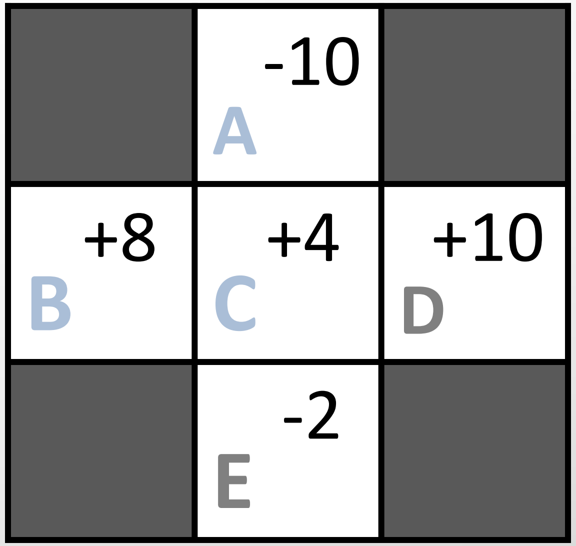
**Episode 3**

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

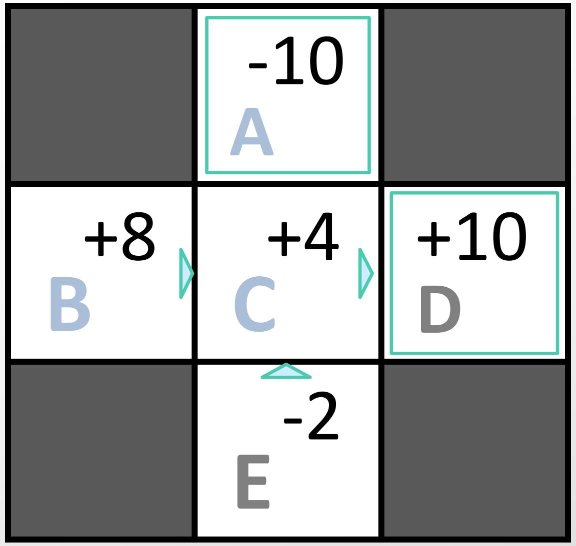


# Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions

- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

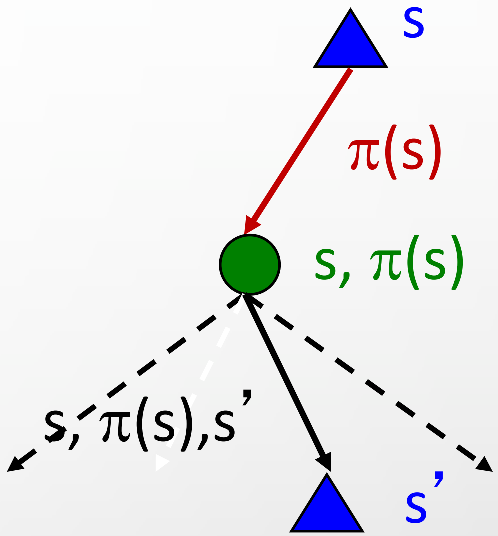
Output Values




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

# Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V




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

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

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

- Key question: how can we do this update to V without knowing T and R?
  - In other words, how to we take a weighted average without knowing the weights?

# Sample-Based Policy Evaluation?

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

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

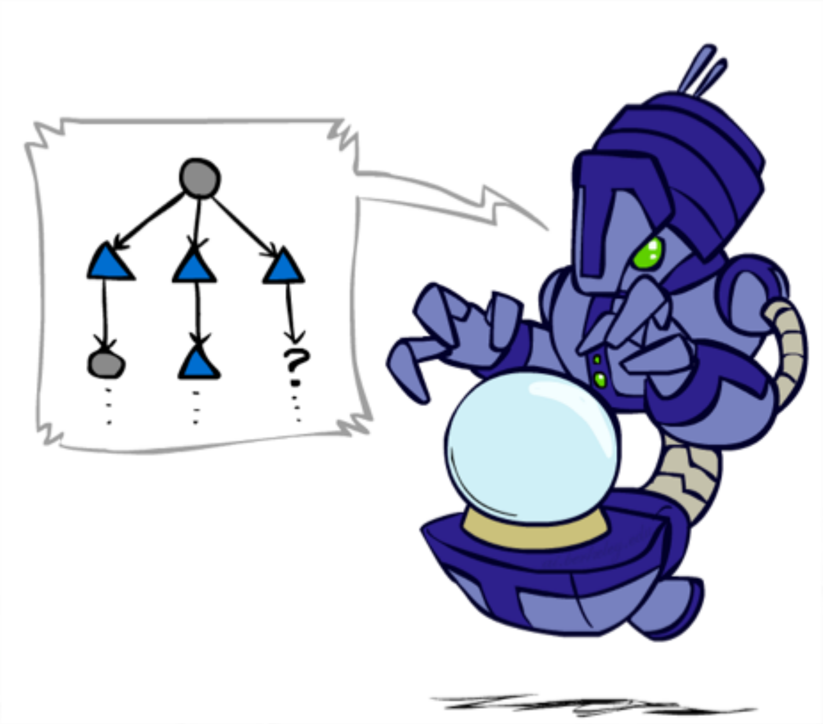
- Idea: take samples of outcomes s' (by doing the action!) and average

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

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

$$\ldots$$

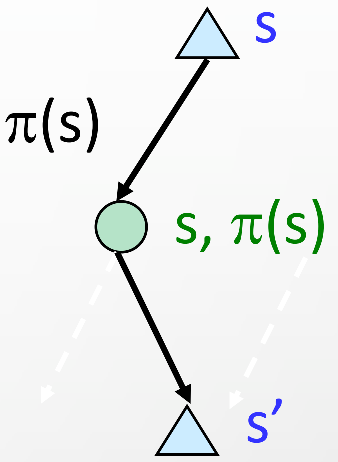
$$sample_n = R(s,\pi(s),s_n') + \gamma V_k^{\pi}(s_n')$$




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

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

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:

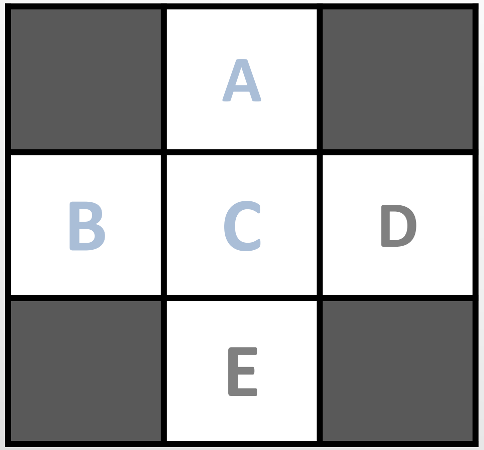
$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

  - Forgets about the past (distant past values were wrong anyway)

- Decreasing learning rate (alpha) can give converging averages

# Example: Temporal Difference Learning

## States

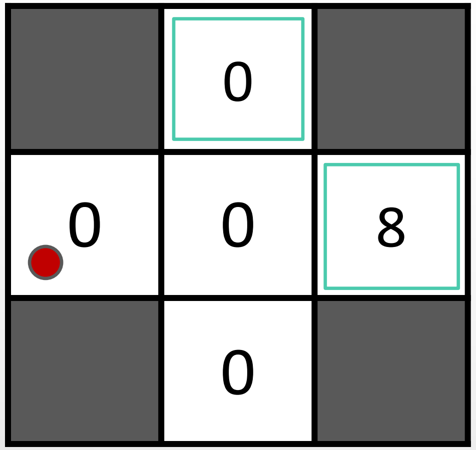


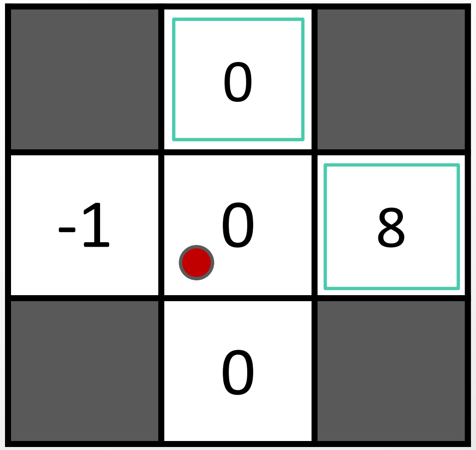

*Assume:* $\gamma = 1$, $\alpha = 1/2$

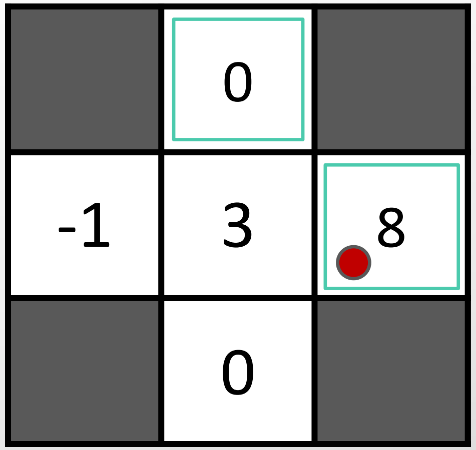
## Observed Transitions

B, east, C, -2

C, east, D, -2



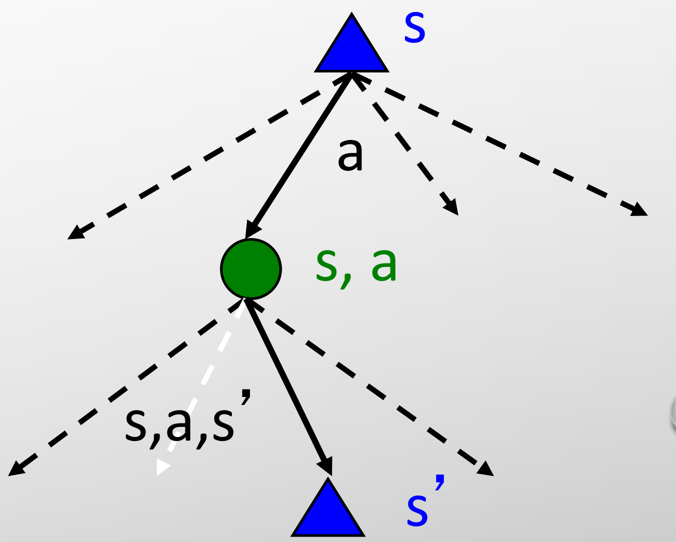

$$V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

# Problems with TD Value Learning

- TD value leaning 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:

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

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




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

# 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

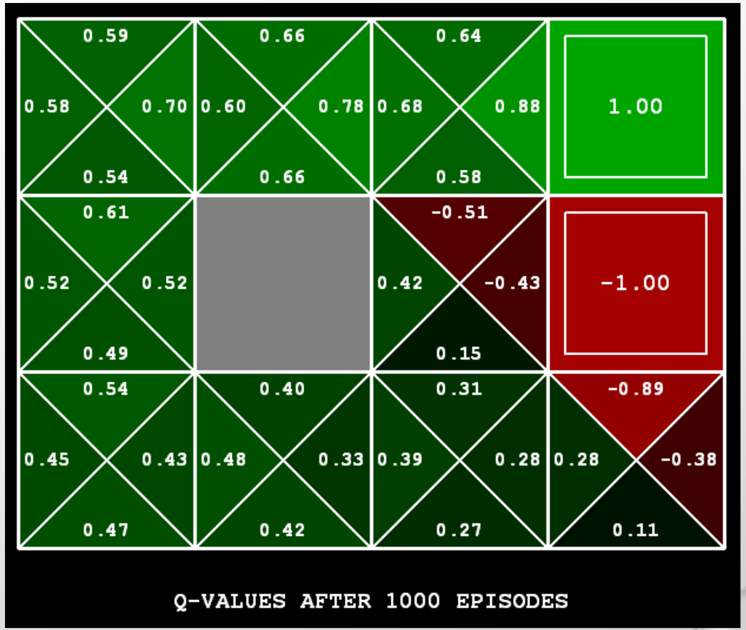
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- 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:

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





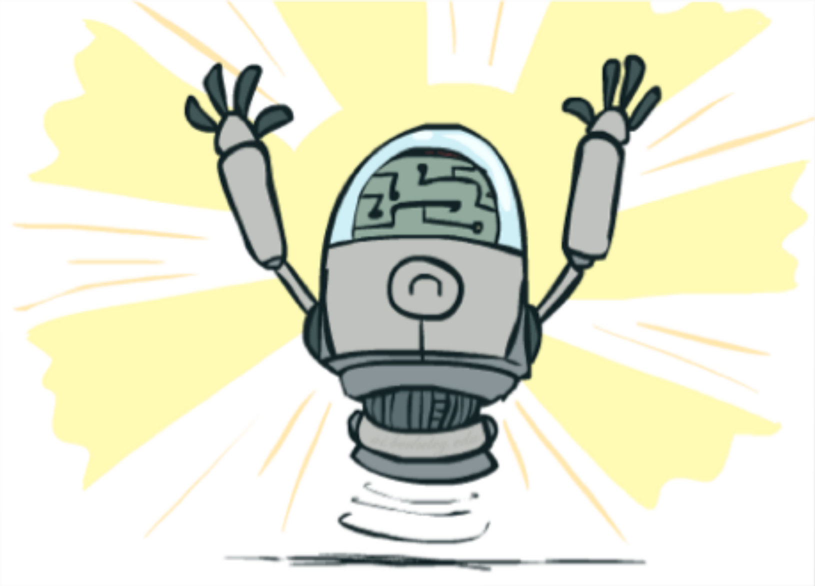

[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

# Video of Demo Q-Learning -- Gridworld

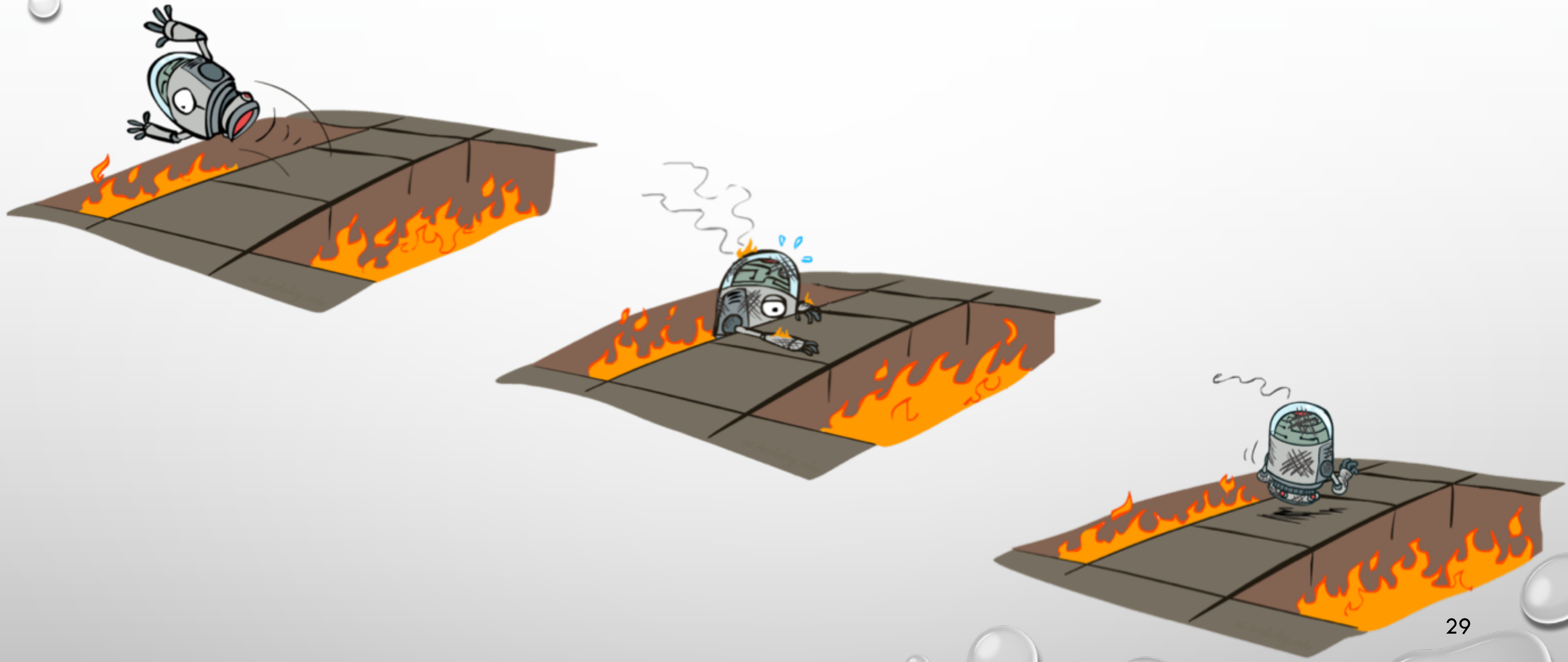
# Video of Demo Q-Learning -- Crawler

# Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting sub-optimally!
- 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 (!)

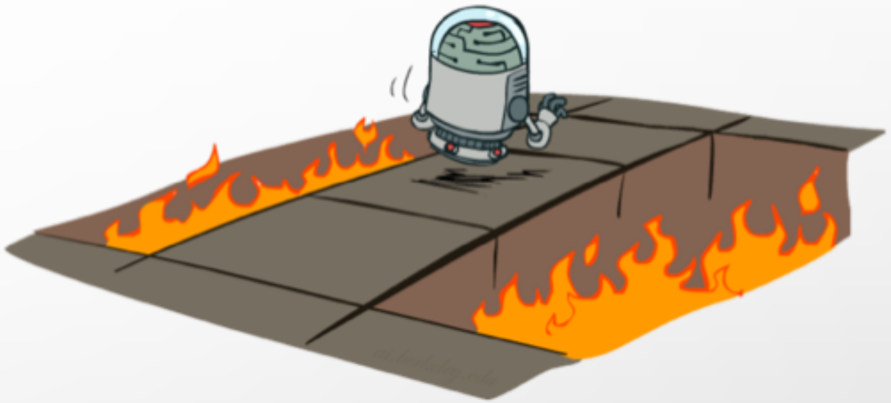


# Active Reinforcement Learning



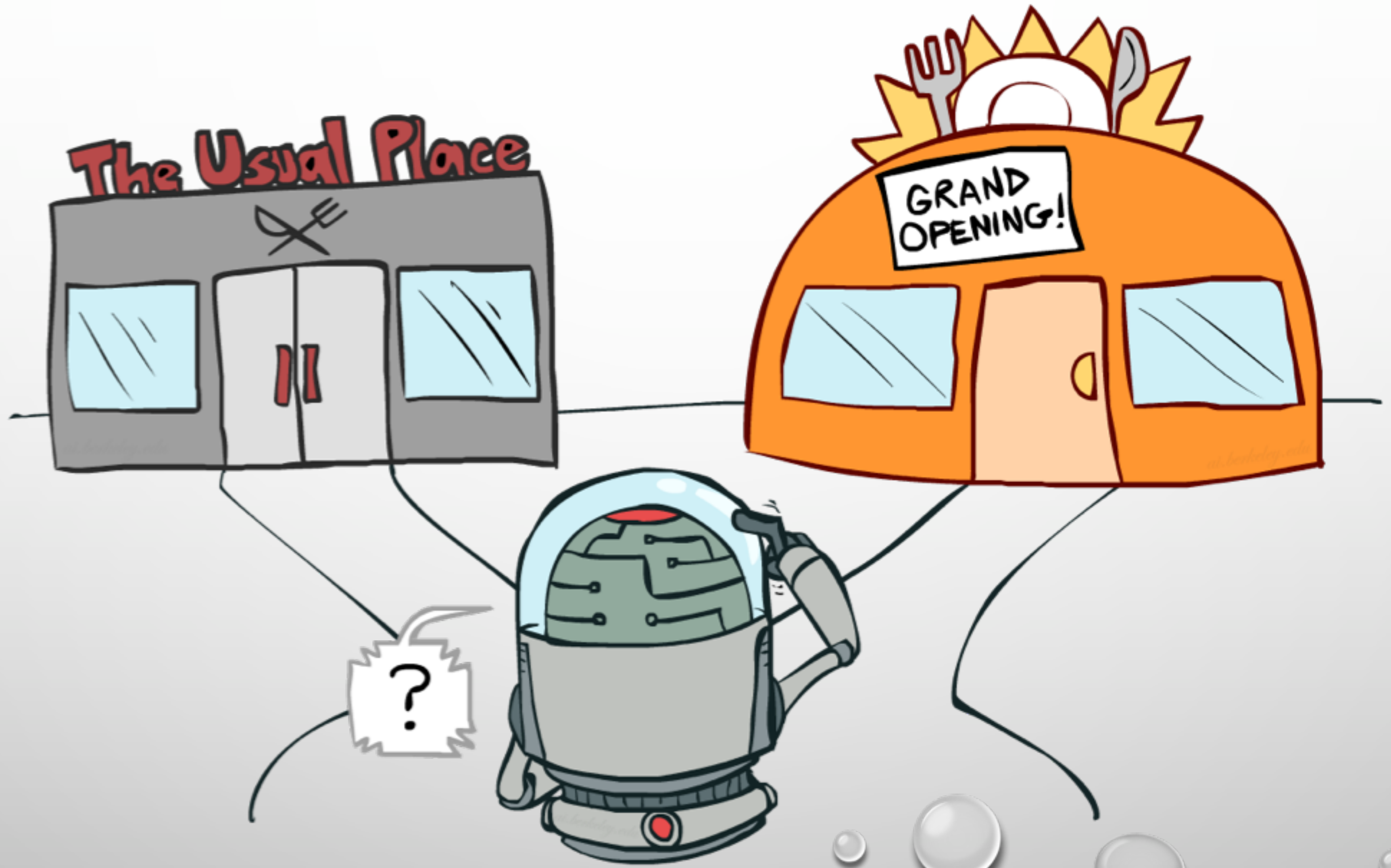
# Active Reinforcement Learning

- 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…

# Exploration vs. Exploitation



# How to Explore?

- Several schemes for forcing exploration
  - Simplest: random actions ($\varepsilon$-greedy)
    - Every time step, flip a coin
    - With (small) probability $\varepsilon$, act randomly
    - With (large) probability $1-\varepsilon$, 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 $\varepsilon$ over time
    - Another solution: exploration functions

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

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

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

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

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

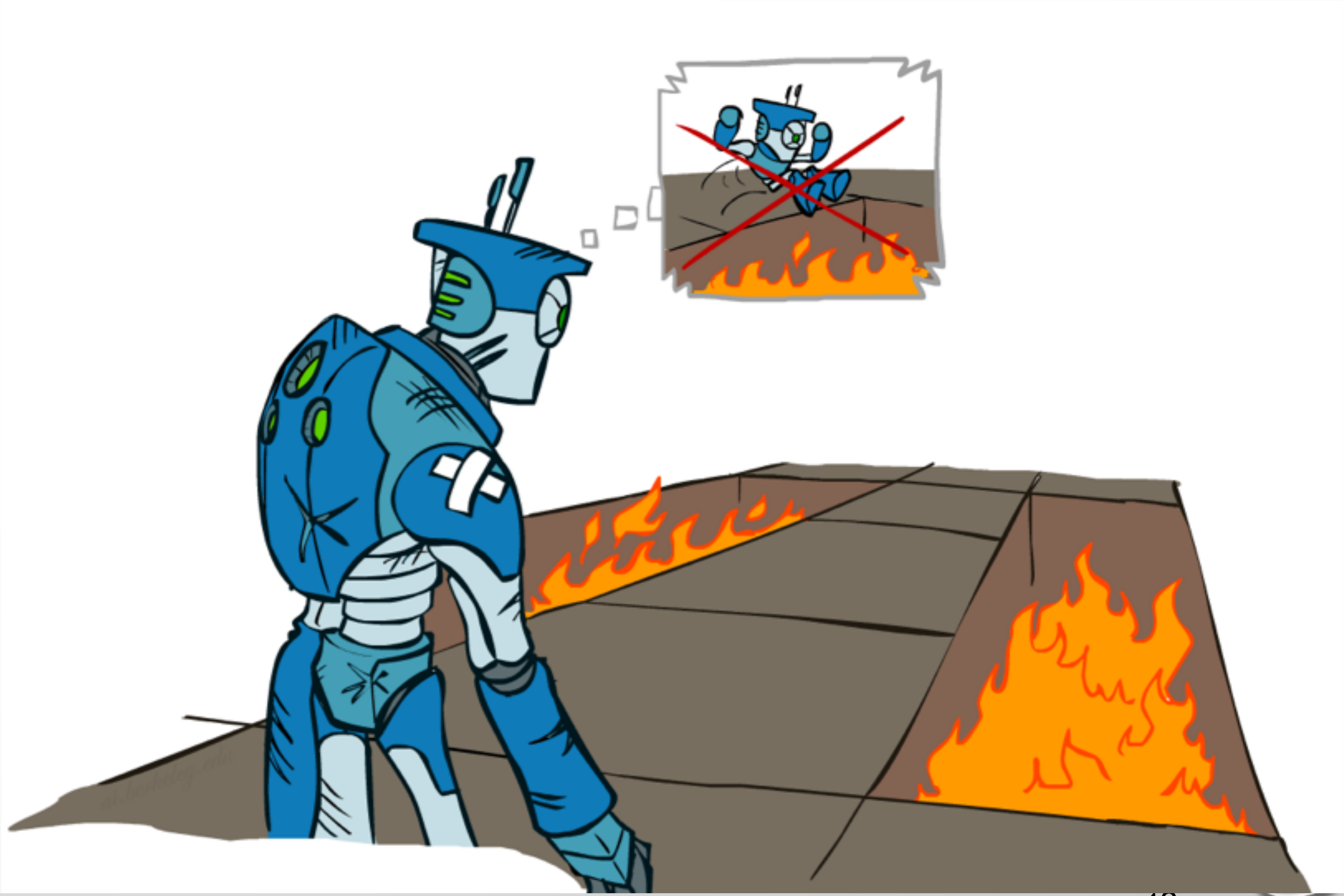


[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

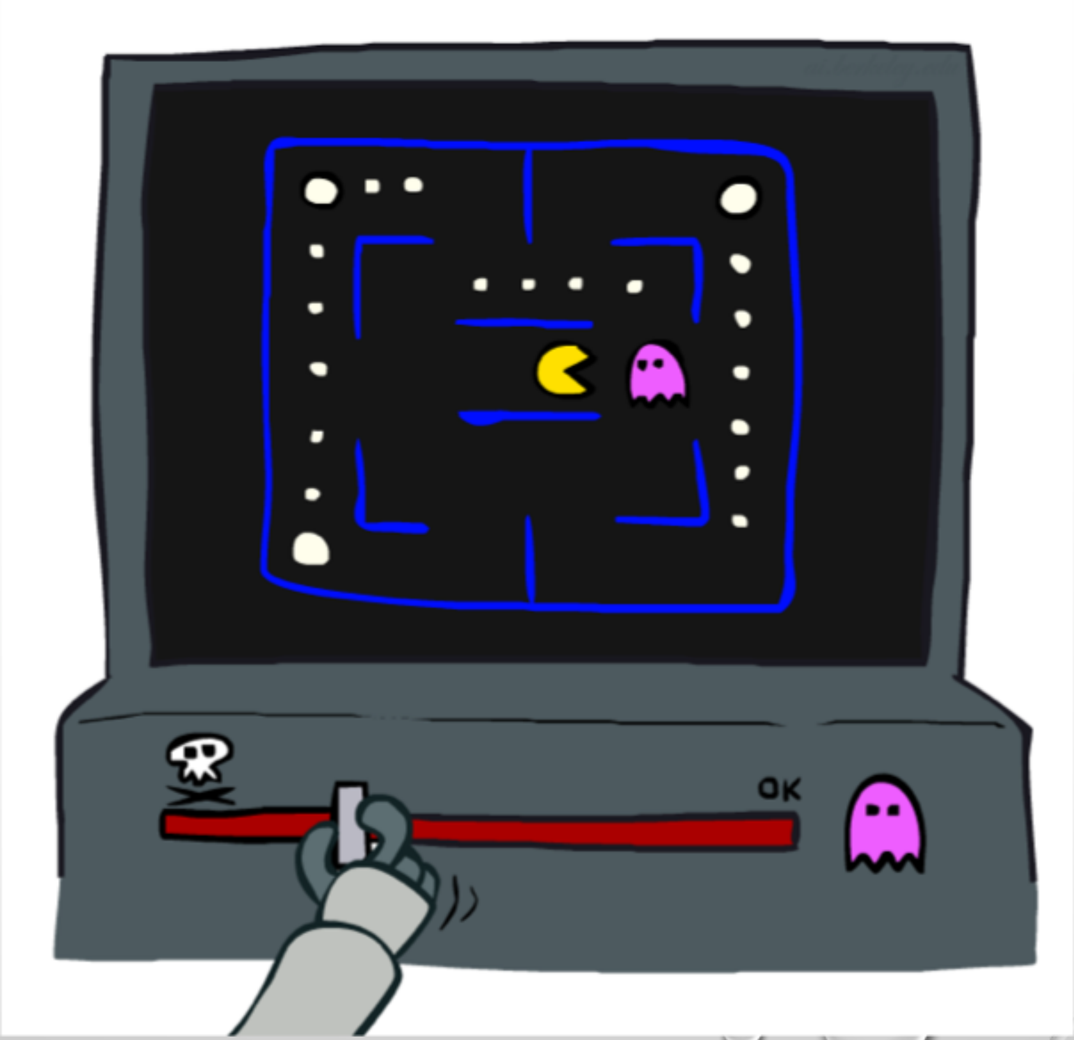
# Video of Demo Q-learning – Exploration Function – Crawler

# 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

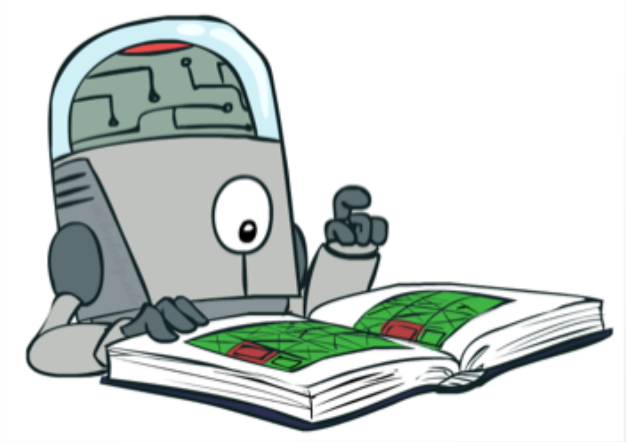


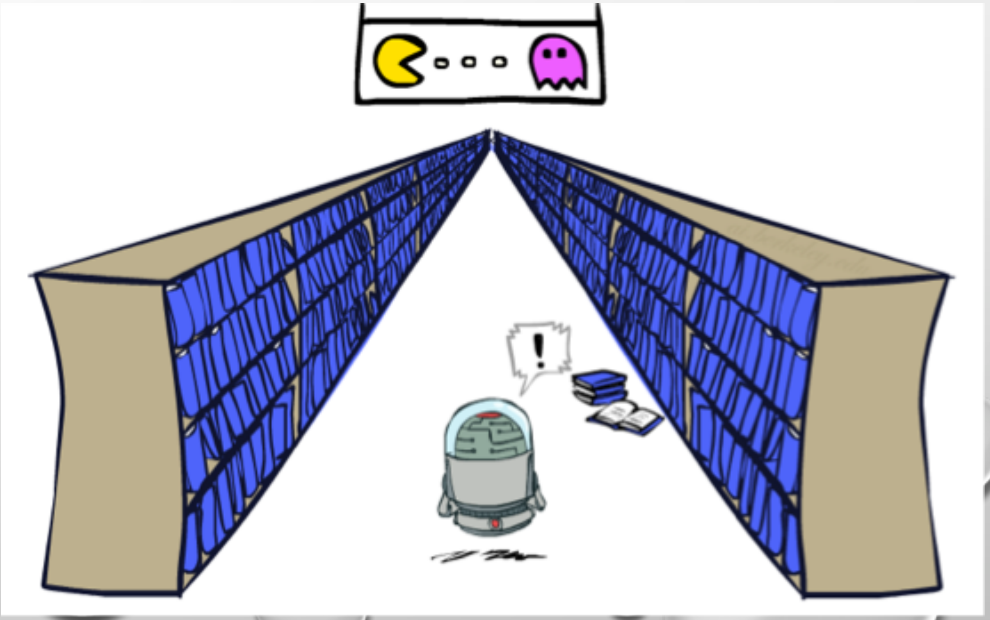
# Generalizing Across States

- 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

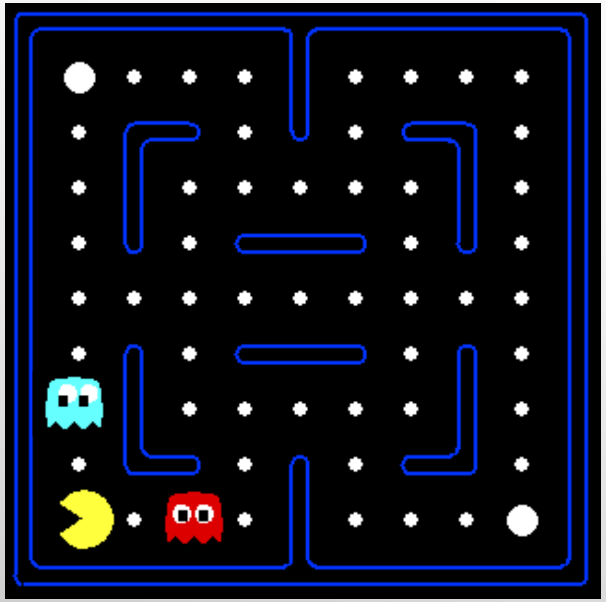




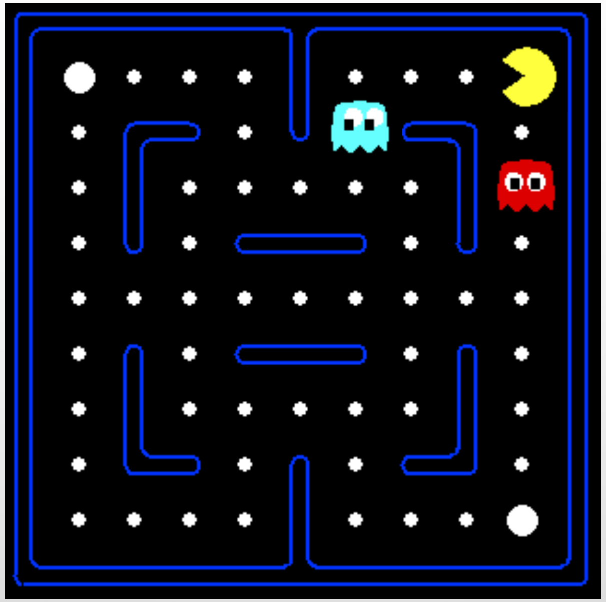

[demo – RL pacman]

# Example: Pacman

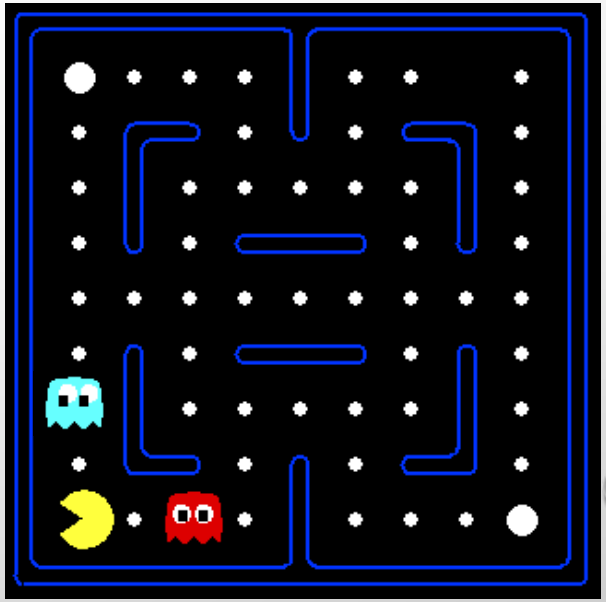
Let's say we discover through experience that this state is bad:



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



Or even this one!





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

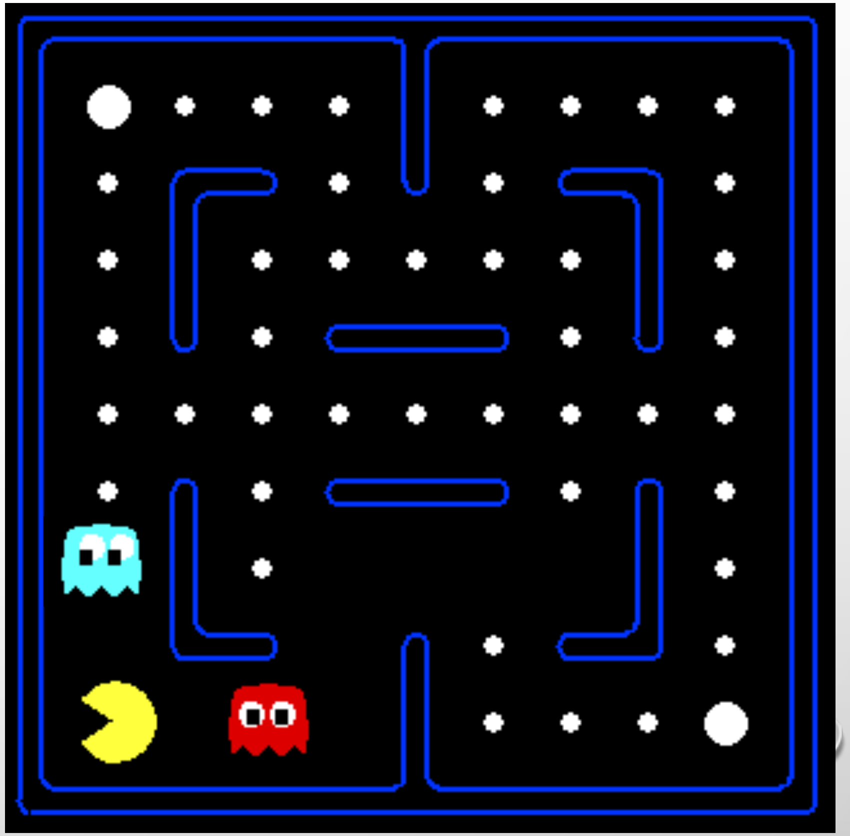
# Video of Demo Q-Learning Pacman – Tiny – Watch All

# Video of demo Q-learning Pacman – tiny – silent train

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

# Feature-Based Representations

- 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 / (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

- 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 different in value!

# Approximate Q-Learning

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

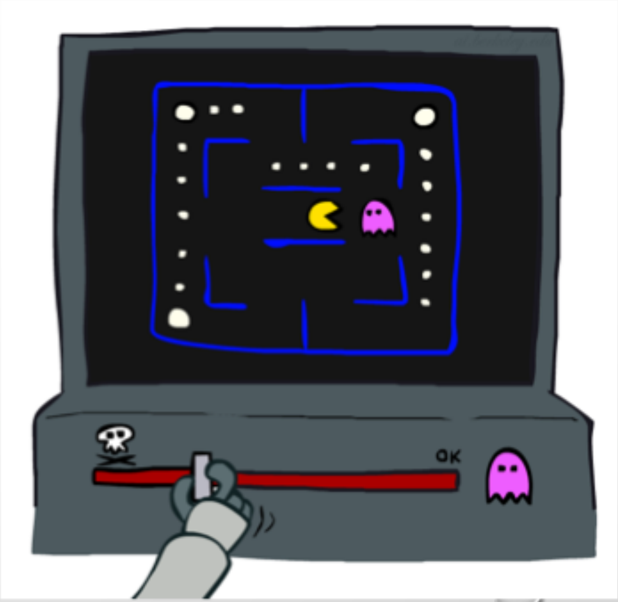
- Q-learning with linear q-functions:

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

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

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

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

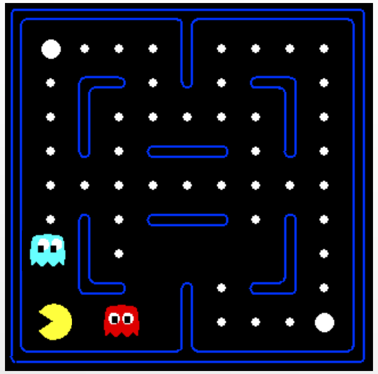


- Intuitive interpretation:
  - Adjust weights of active features
  - e.g., If something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$

$s$



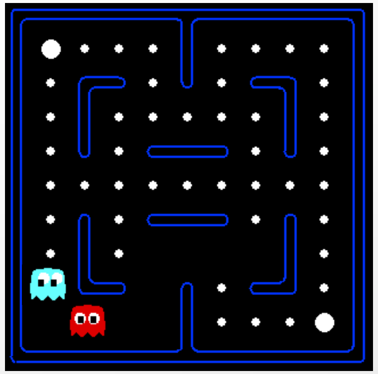
$f_{DOT}(s, \text{NORTH}) = 0.5$

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

$a = \text{NORTH}$

$r = -500$

$s'$



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

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

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

$\text{difference} = -501$

$w_{DOT} \leftarrow 4.0 + \alpha \left[-501\right] 0.5$

$w_{GST} \leftarrow -1.0 + \alpha \left[-501\right] 1.0$

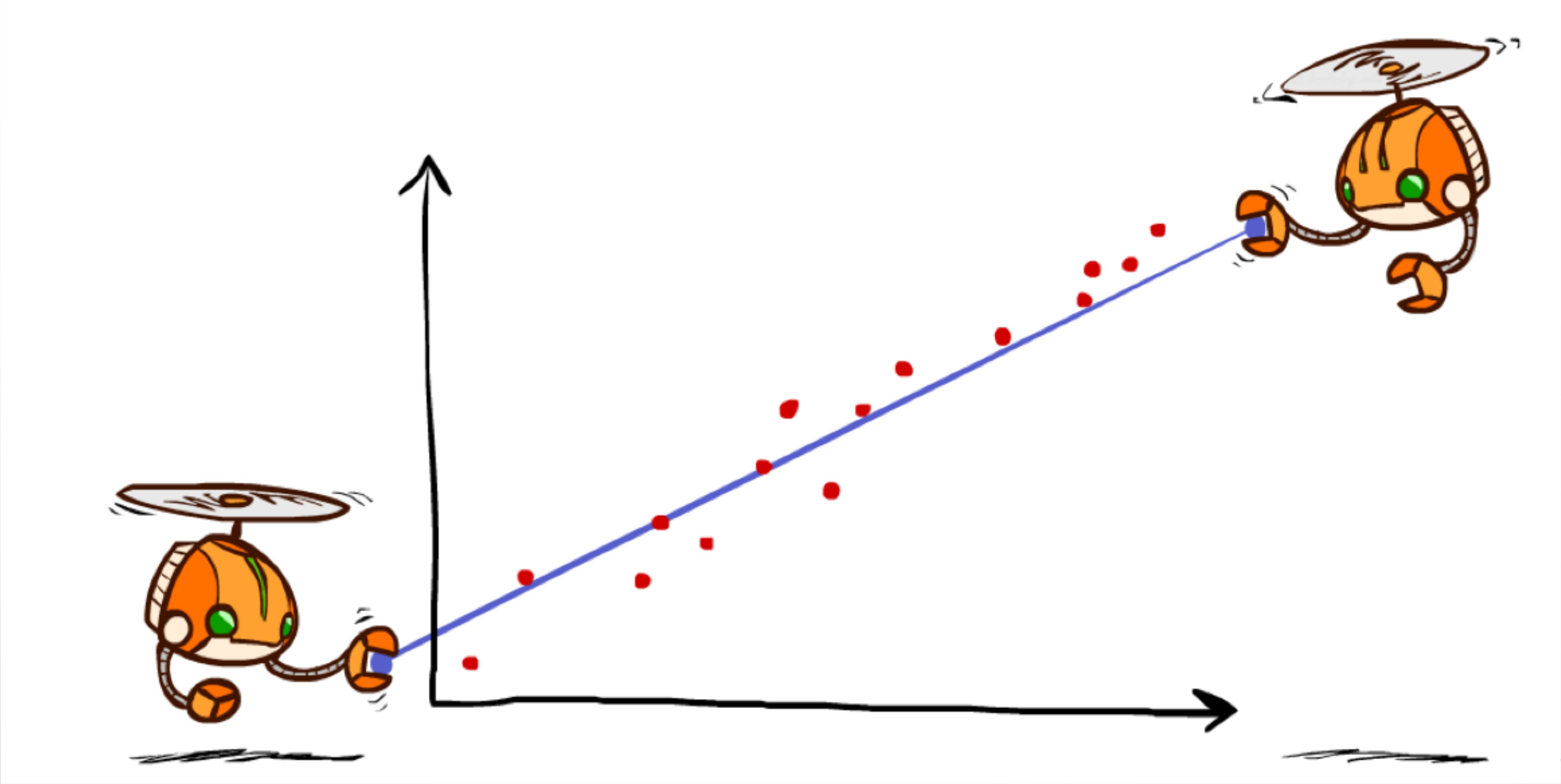


$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

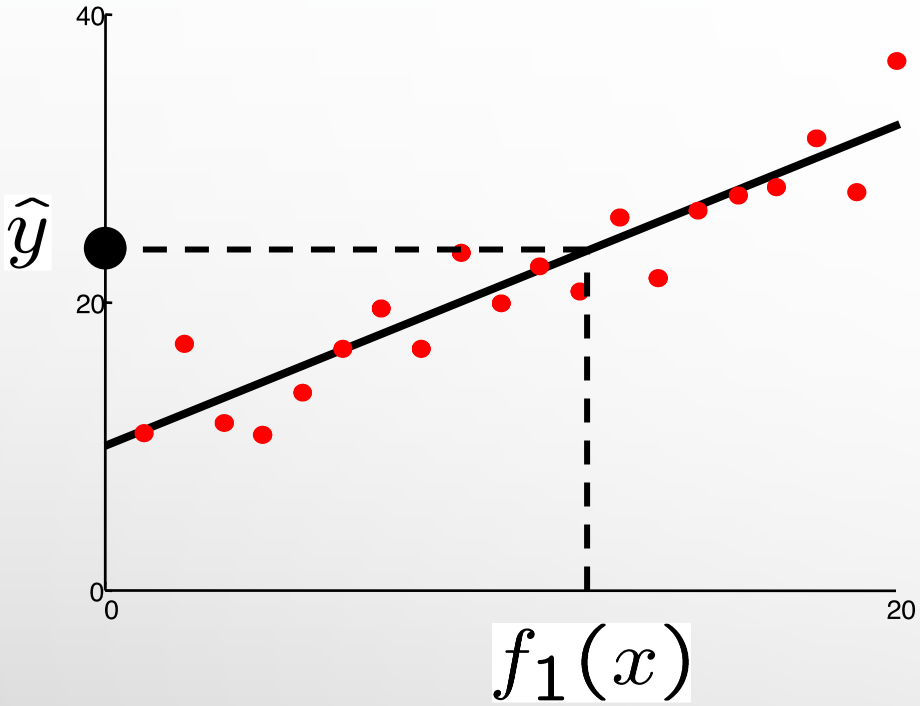
[Demo: approximate Q-learning pacman (L11D10)]

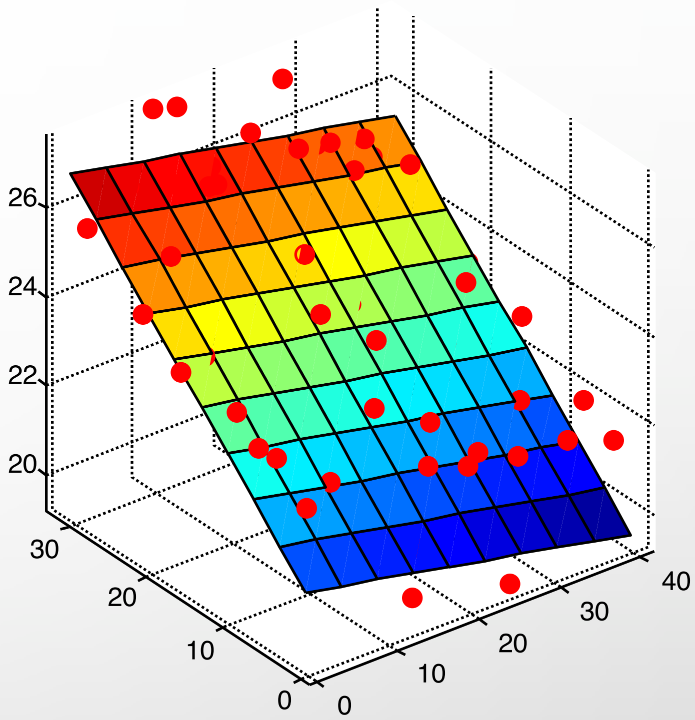
# Video of Demo Approximate Q-Learning -- Pacman

# Q-Learning and Least Squares



# Linear Approximation: Regression*




Prediction:

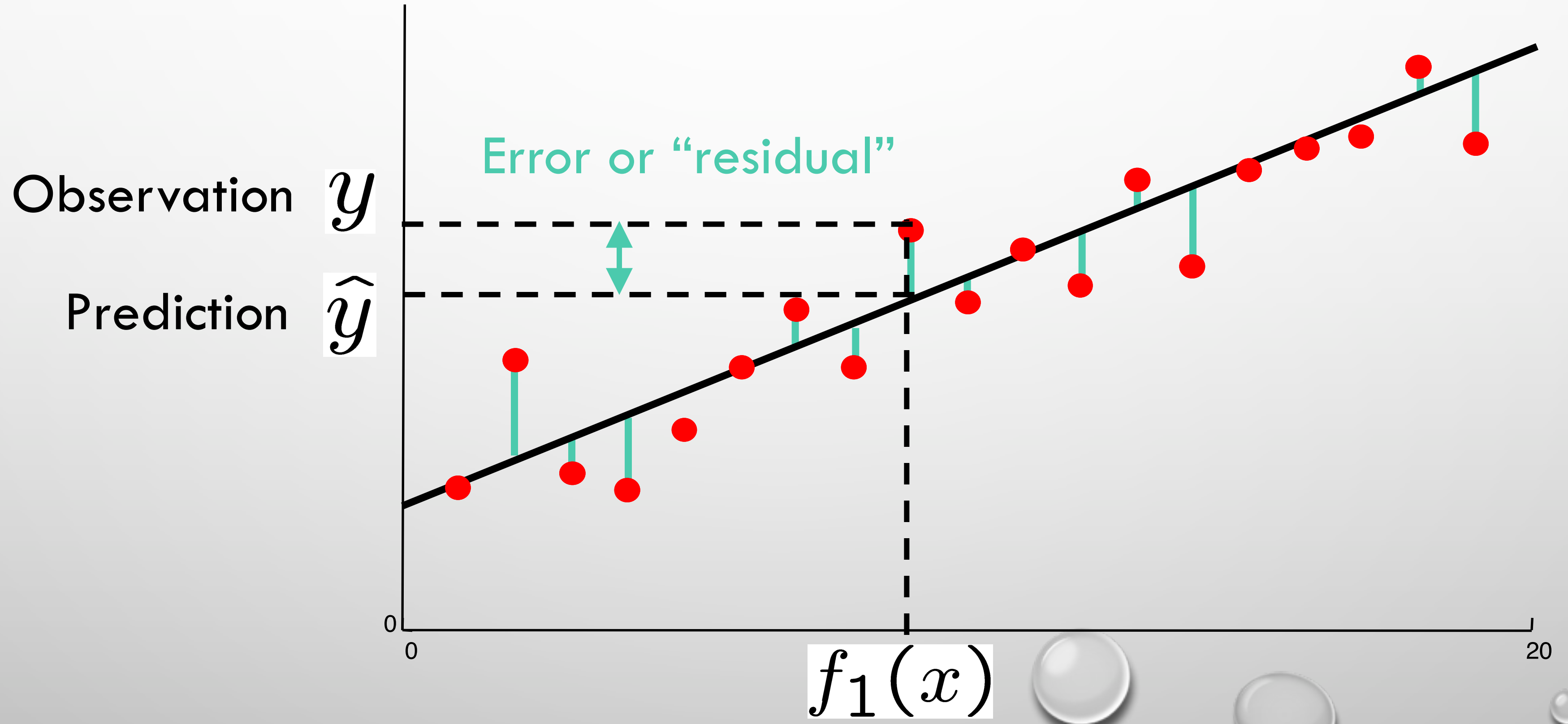
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

# Optimization: Least Squares*

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



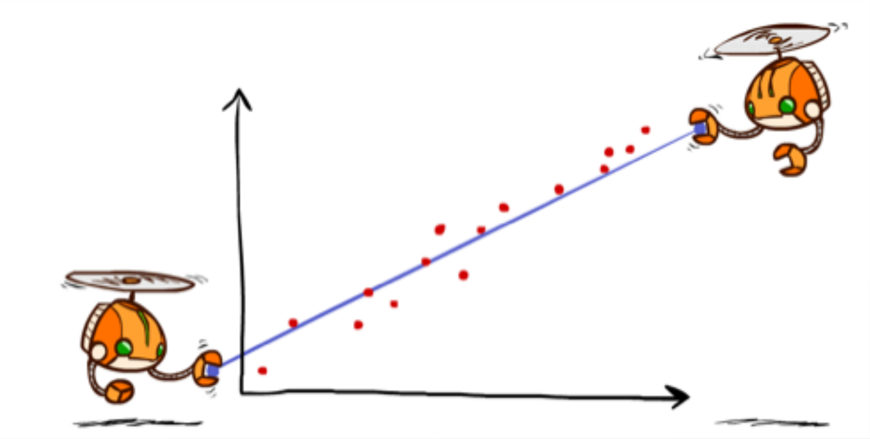
# Minimizing Error*

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”

# 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