CSE 573: Artificial Intelligence

Hanna Hajishirzi

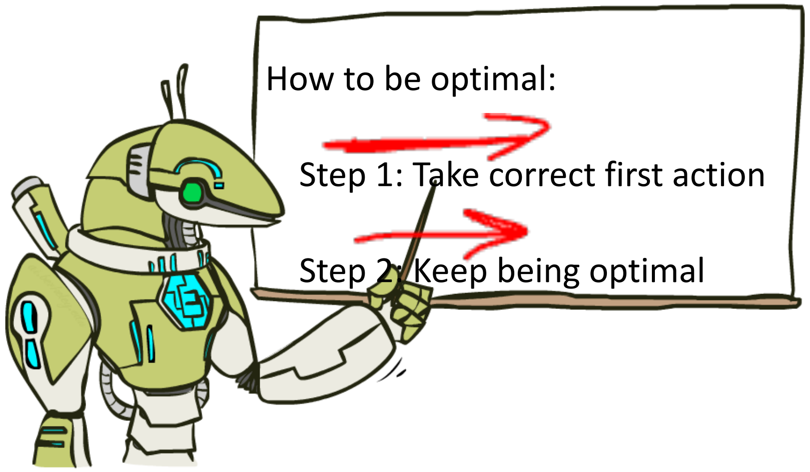
Reinforcement Learning

slides adapted from
Dan Klein, Pieter Abbeel ai.berkeley.edu
And Dan Weld, Luke Zettlemoyer

MDPs Recap

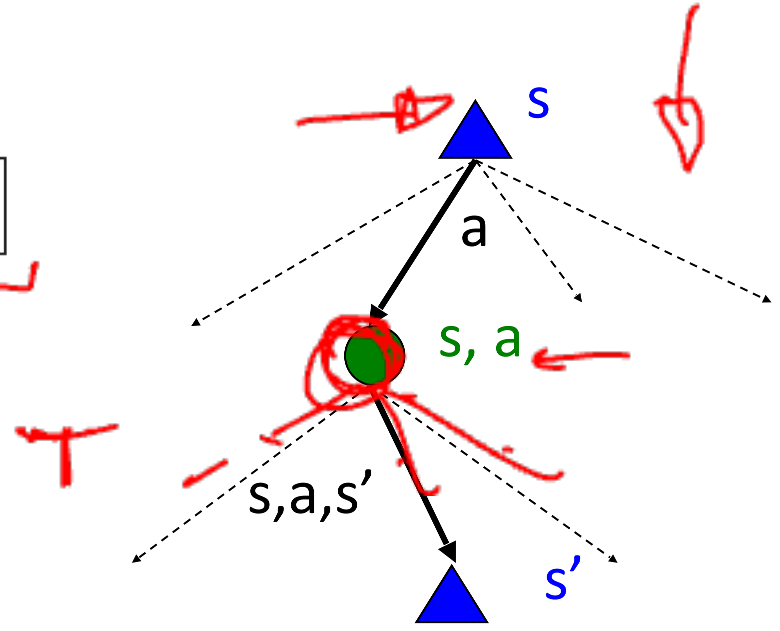
The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives one-step lookahead relationship amongst optimal utility values



$$V^*(s) = \max_a Q^*(s,a)$$

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



- These

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

values in a way we'll use over and over

Value Iteration

- Bellman equations characterize the optimal values:

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

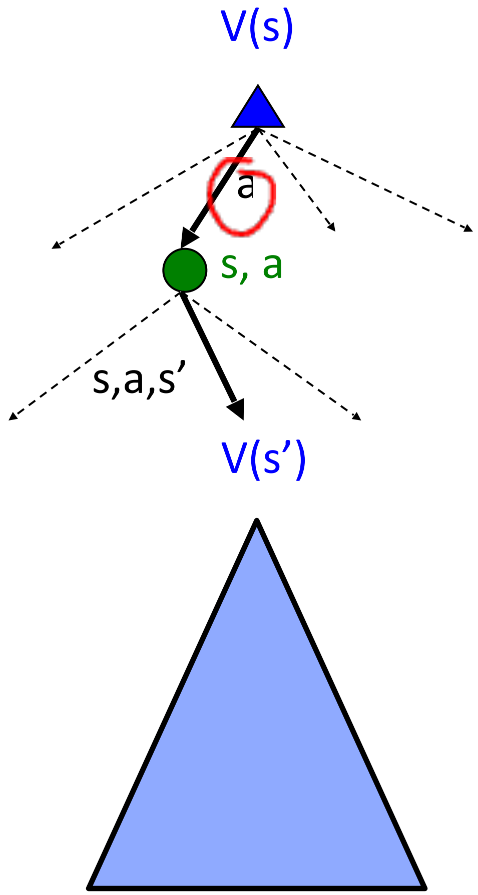
- Value iteration computes them:

$V_1 = 0$

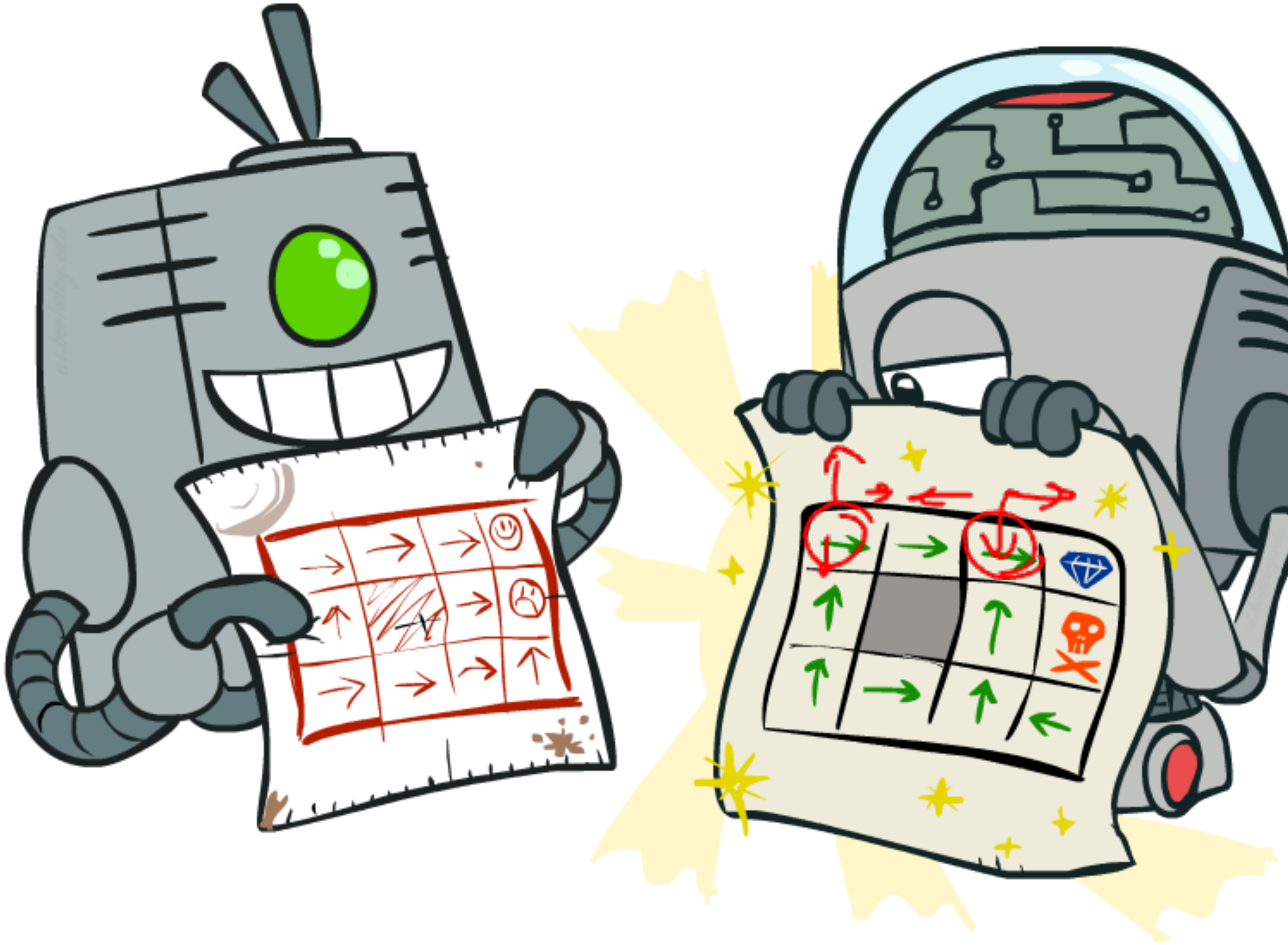
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \boxed{\gamma} V_k(s') \right]$$

$|S|^2|A|$

- Value iteration is just a fixed point solution method
  - … though the V_k vectors are also interpretable as time-limited values



Policy Methods



Policy Evaluation

- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- Define the utility of a state s, under a fixed policy π:

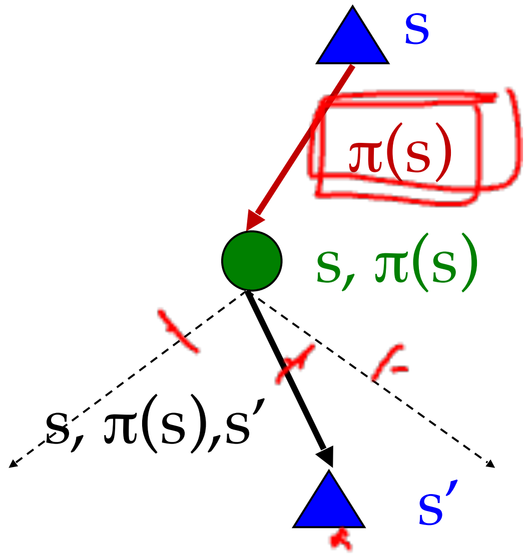
  $V^\pi(s)$ = expected total discounted rewards starting in s and following π

- Recursive relation (one-step look-ahead / Bellman equation):

max

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

+

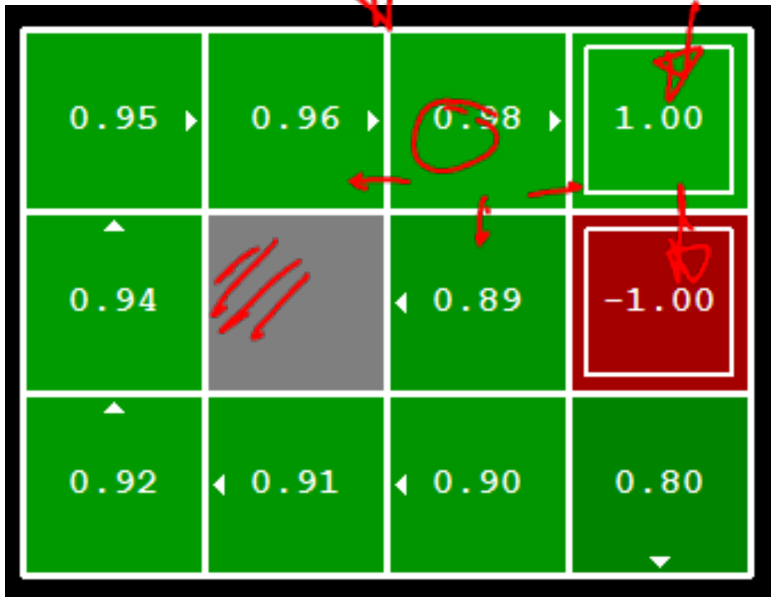


Policy Extraction

- Let's imagine we have the optimal values V*(s)

- How should we act?
  - It's not obvious!

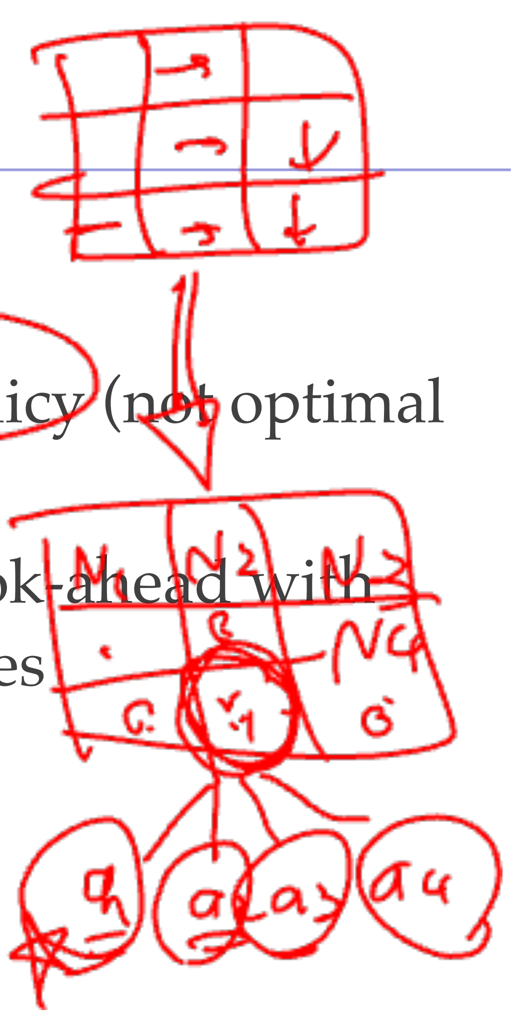
- We need to do a mini-expectimax (one step)



$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- This is called policy extraction, since it gets the policy implied by the values

Policy Iteration



- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

Policy Iteration

- ~~Evaluation:~~ For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

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

- Improvement: For fixed values, get a better policy using policy extraction
  - ~~One-step look-ahead:~~

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$$

Comparison

30 29 28 26

8.1 30 29 28

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we're done)

$\boxed{V_K} = f(V_{K-1})$

$V_{K-1} = f(V_{K-2})$

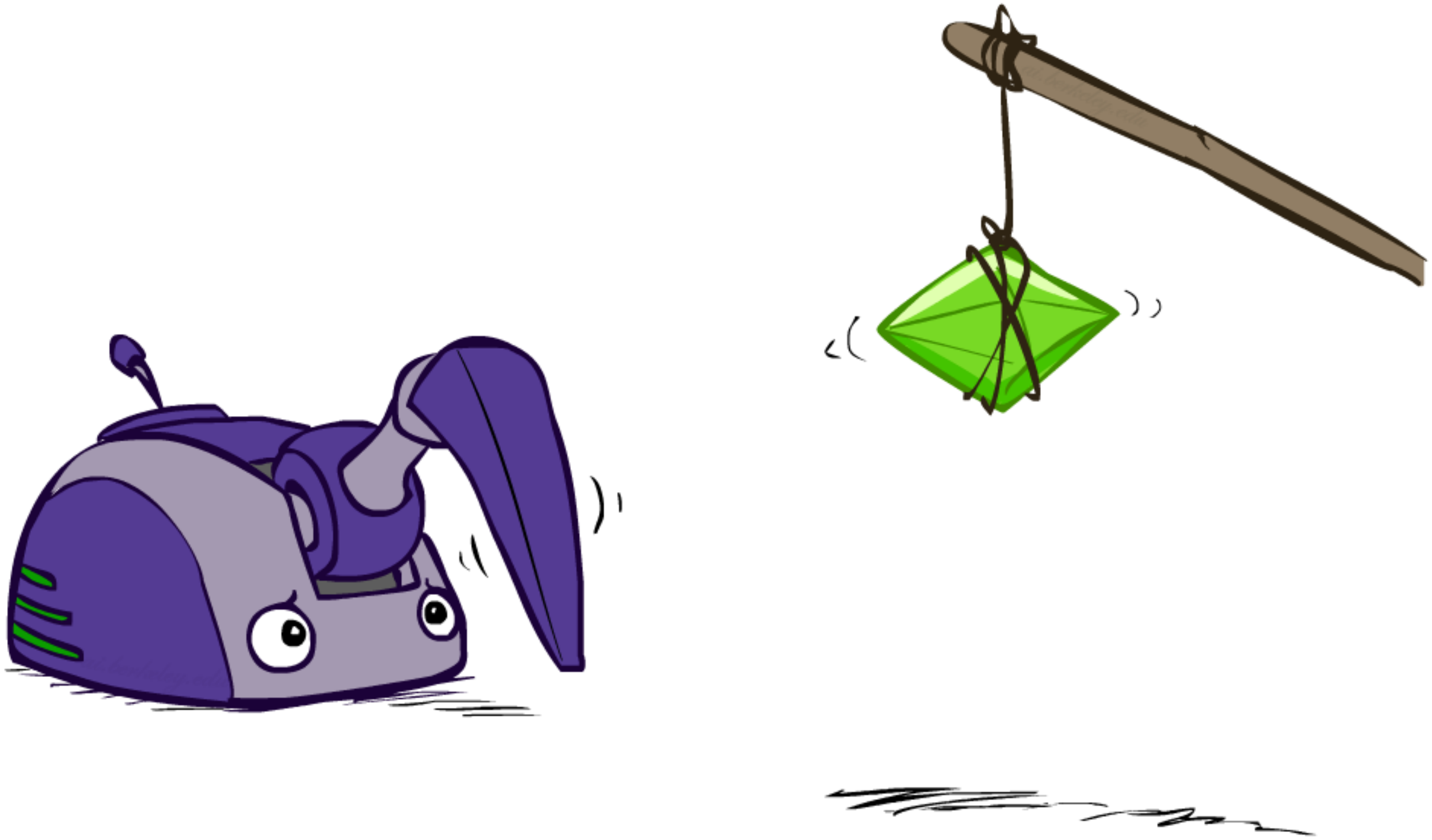
$V_0 = 0$

- Both are dynamic programs for solving MDPs

Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)
- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

Reinforcement Learning



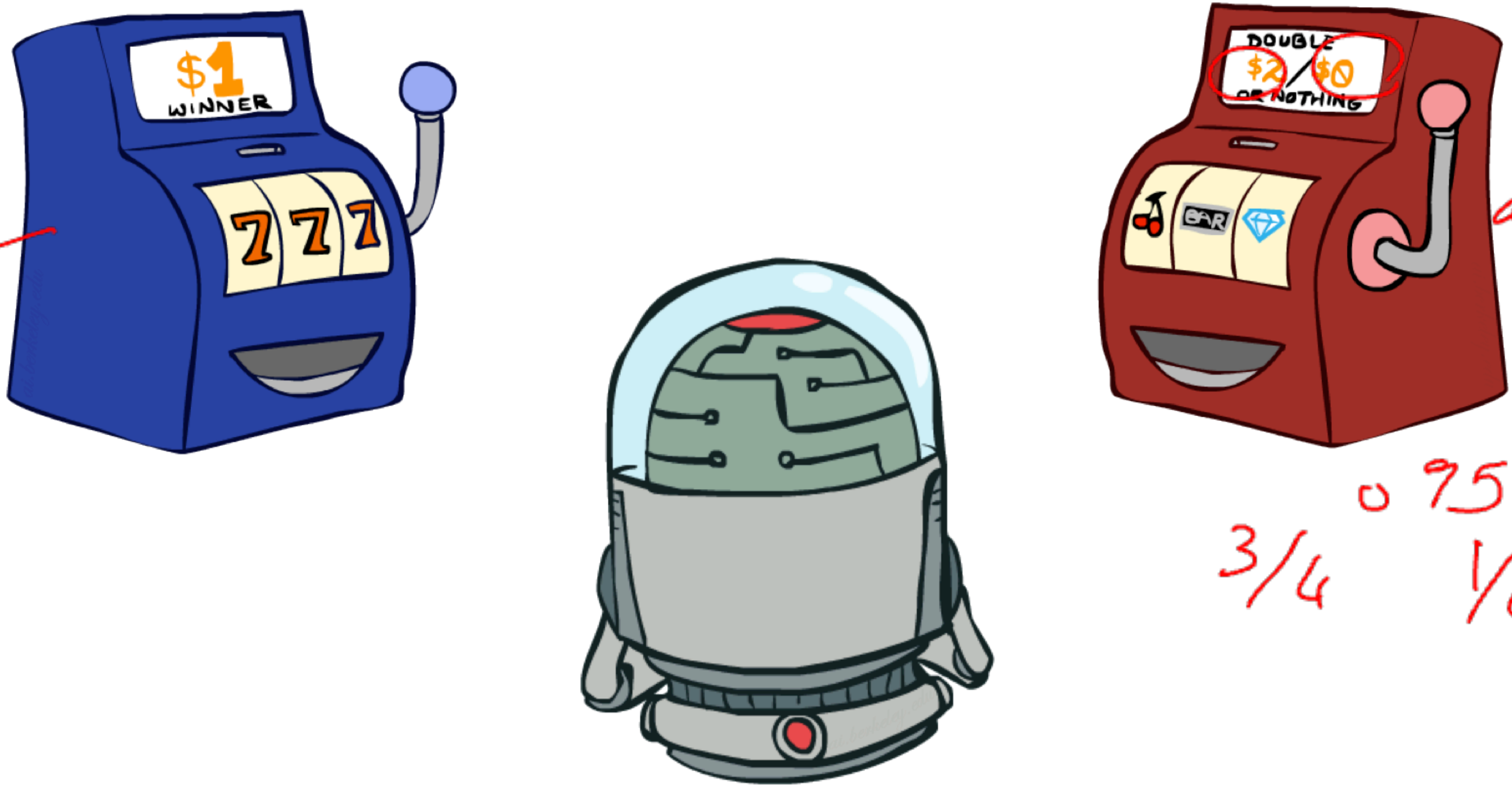
MDP

offline planning

Double Bandits

RL

online learning

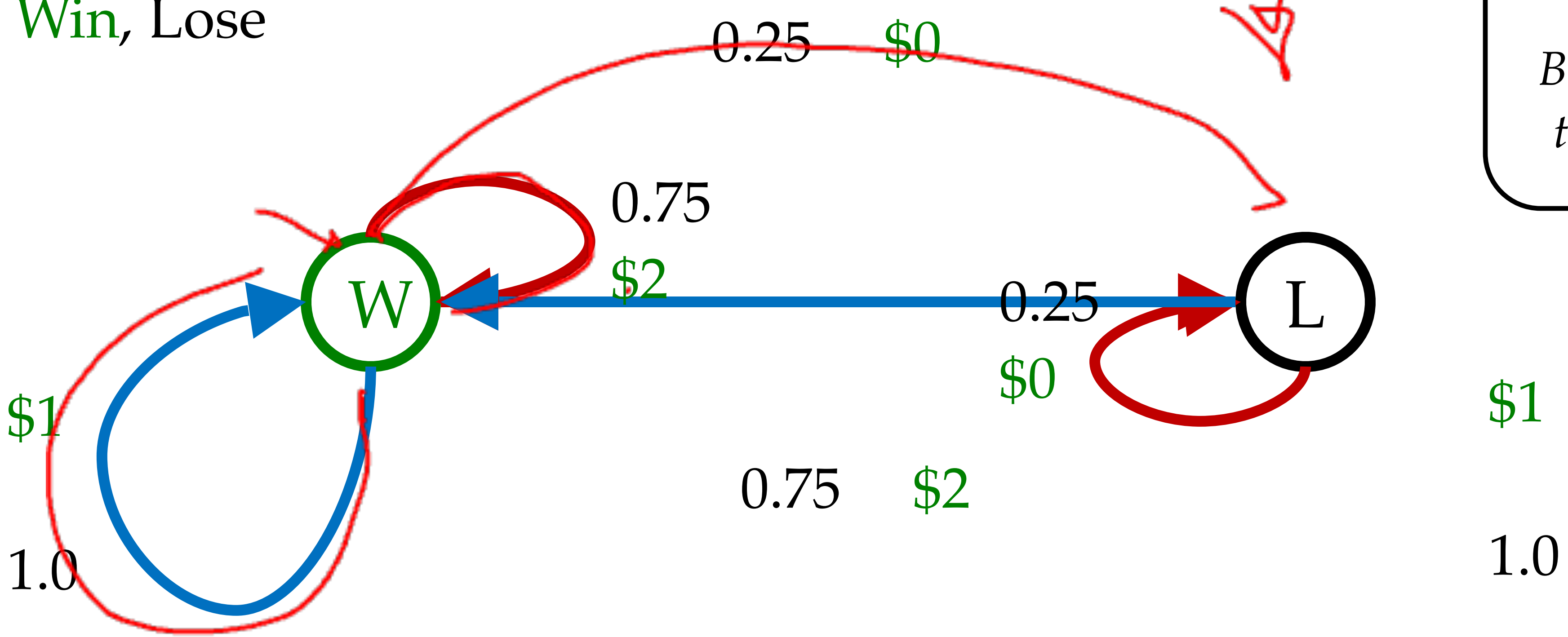


0.75

3/4 1/4

Double-Bandit MDP

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



No discount
10 time steps
Both states have the same value

$10 \times (3/4 \times 2 + 1/4 \times 0) = \$15$

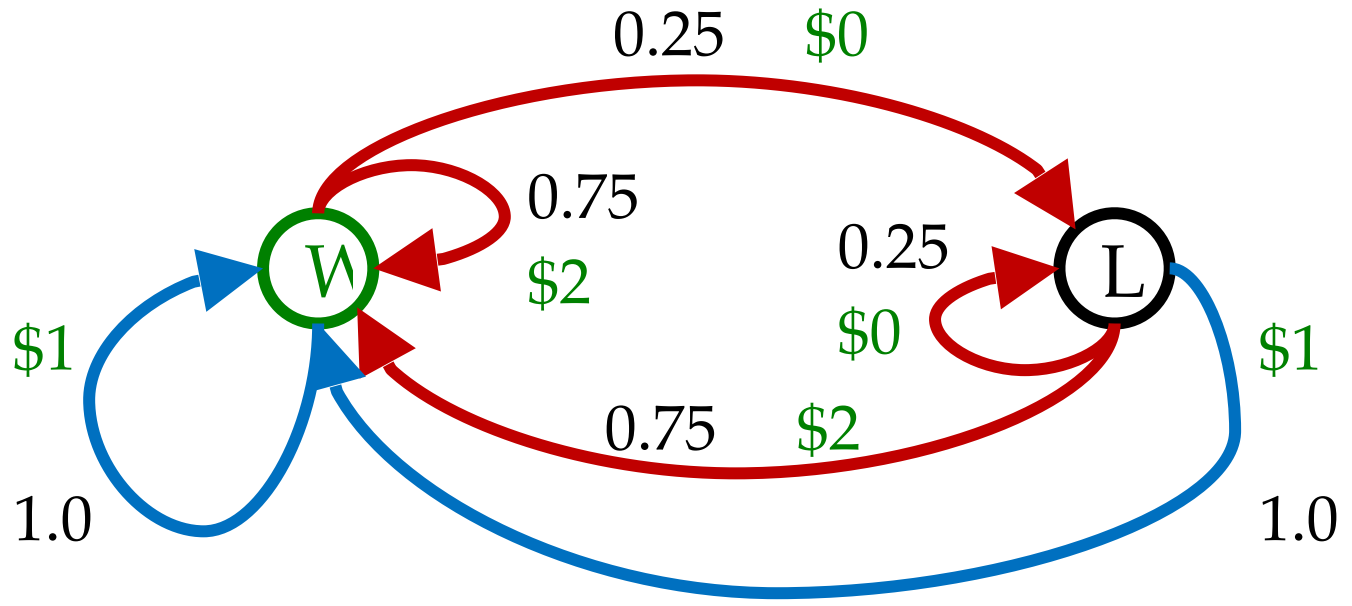
Offline Planning

- 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!

$10

10

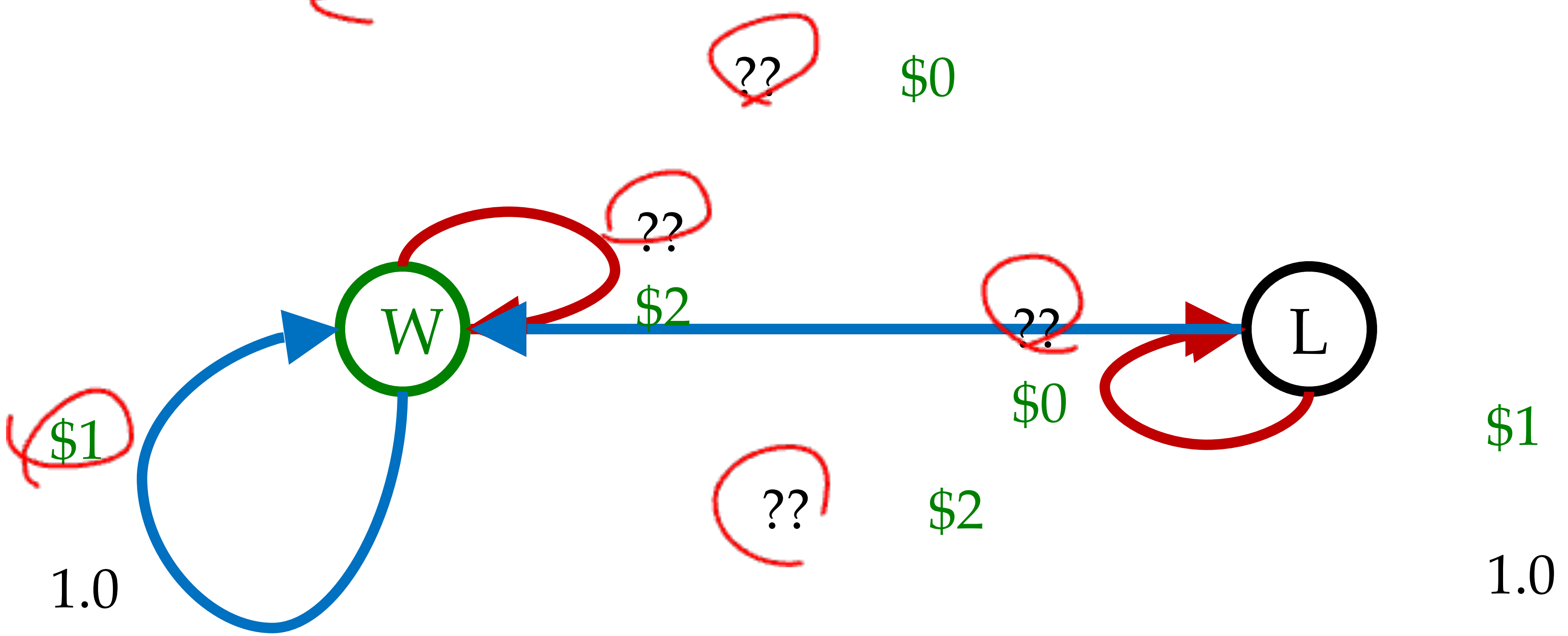
$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

$12

Online Planning

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



Let's Play!

$0 $0 $2 $0

$0 $2 $2 $0 $0

$0

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

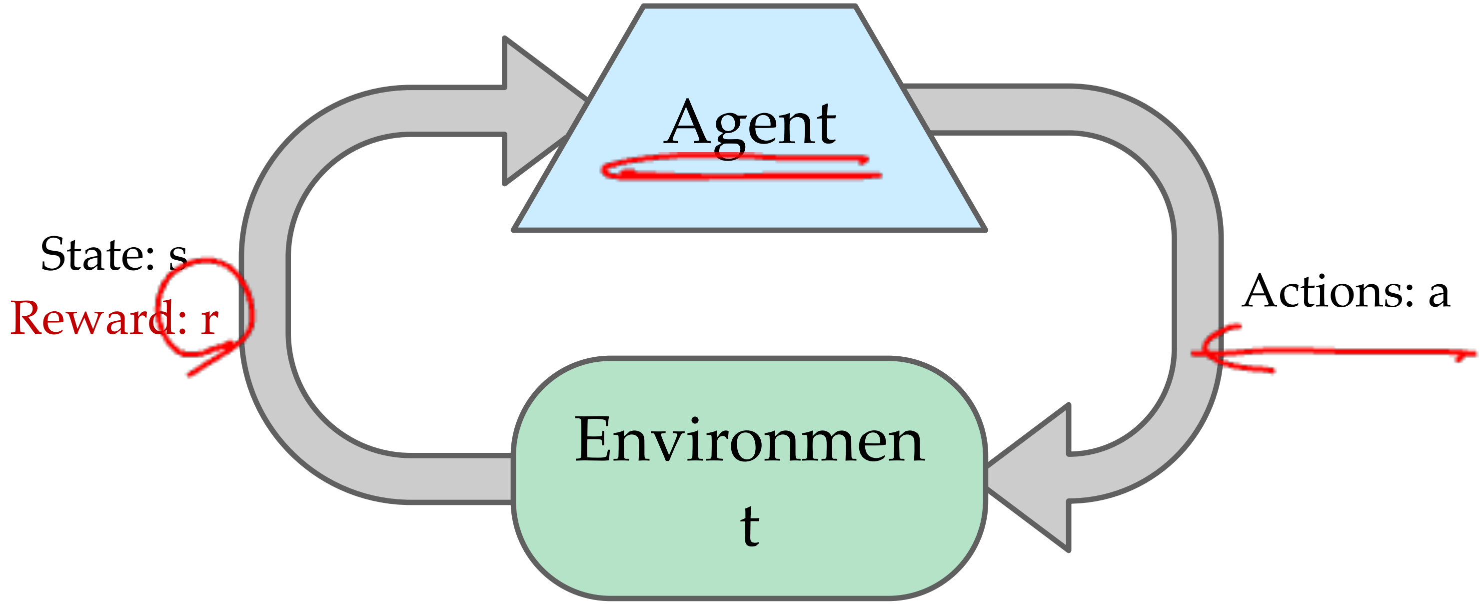
- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things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

Reinforcement Learning

- 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

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]

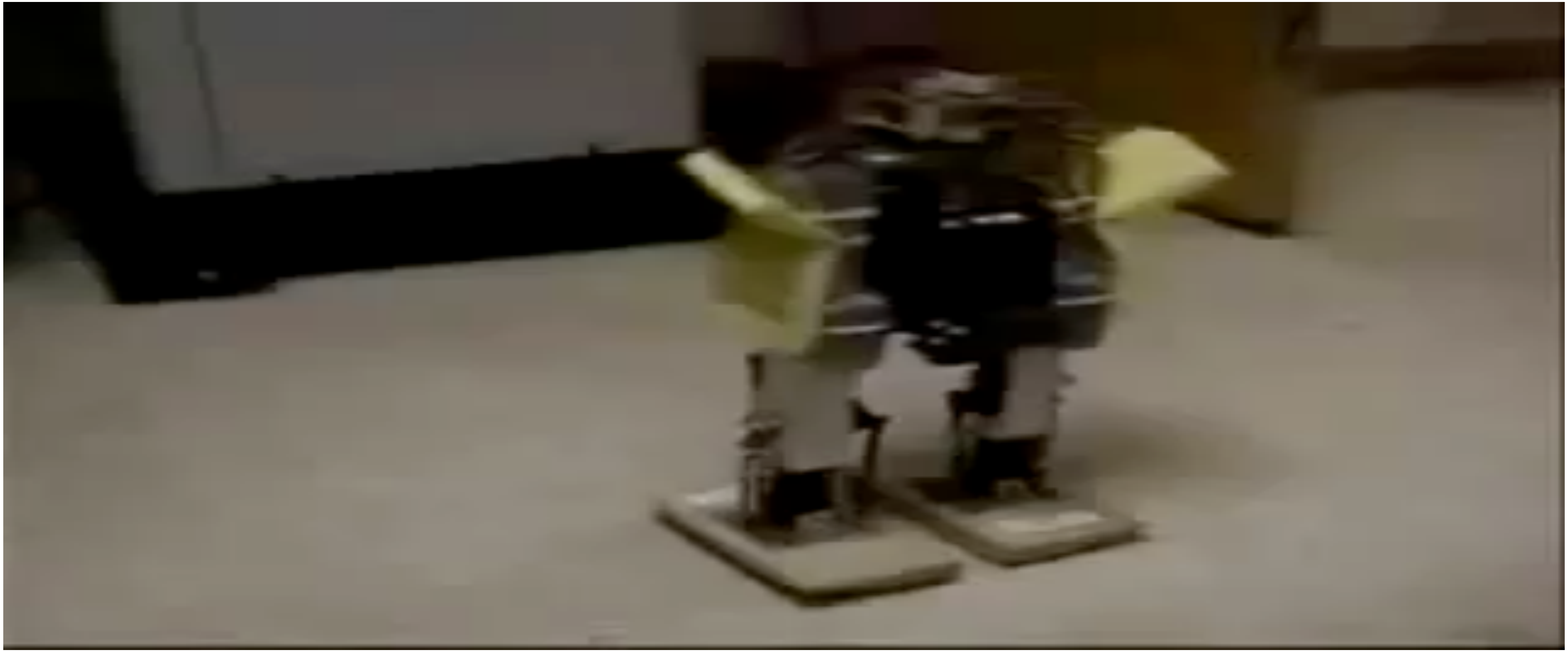
Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

Example: Toddler Robot



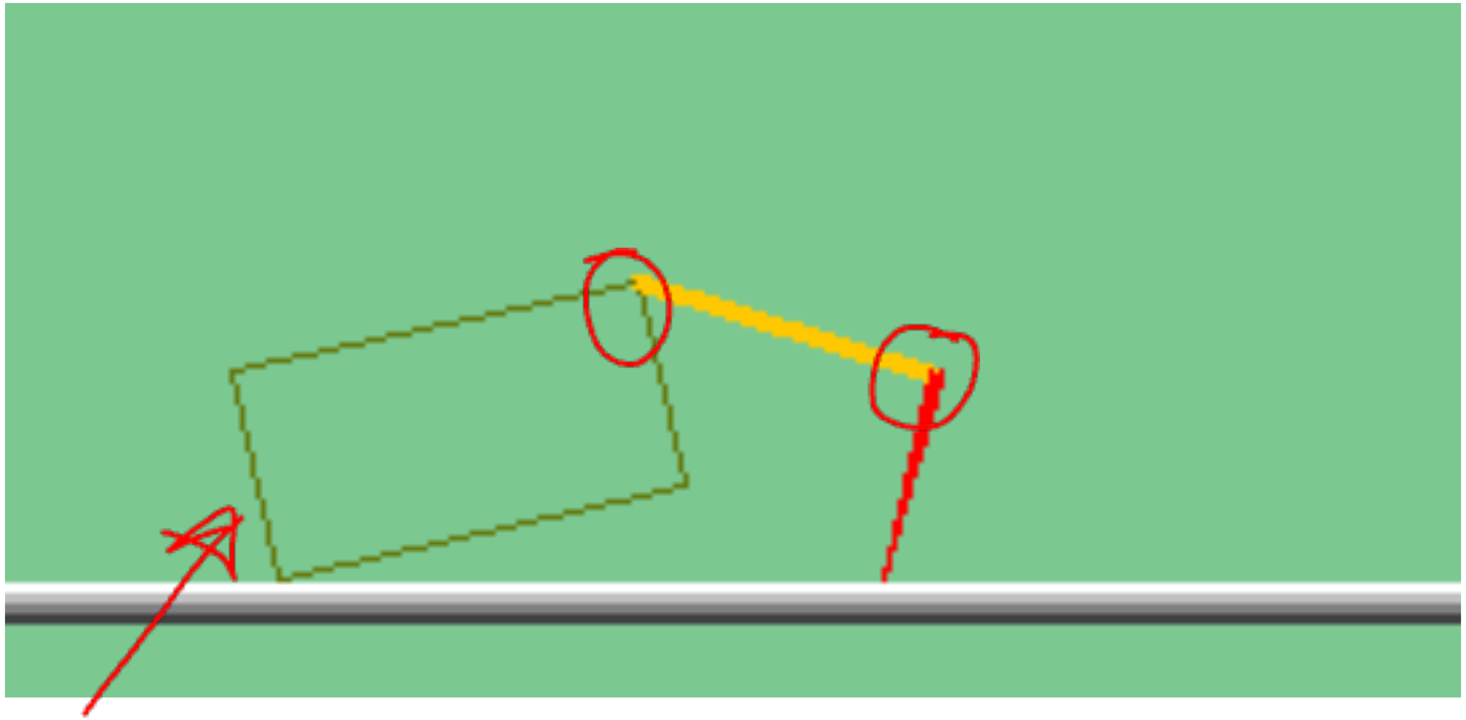
[Tedrake, Zhang and Seung, 2005]

[Video: TODDLER – 40s]

Robotics Rubik Cube

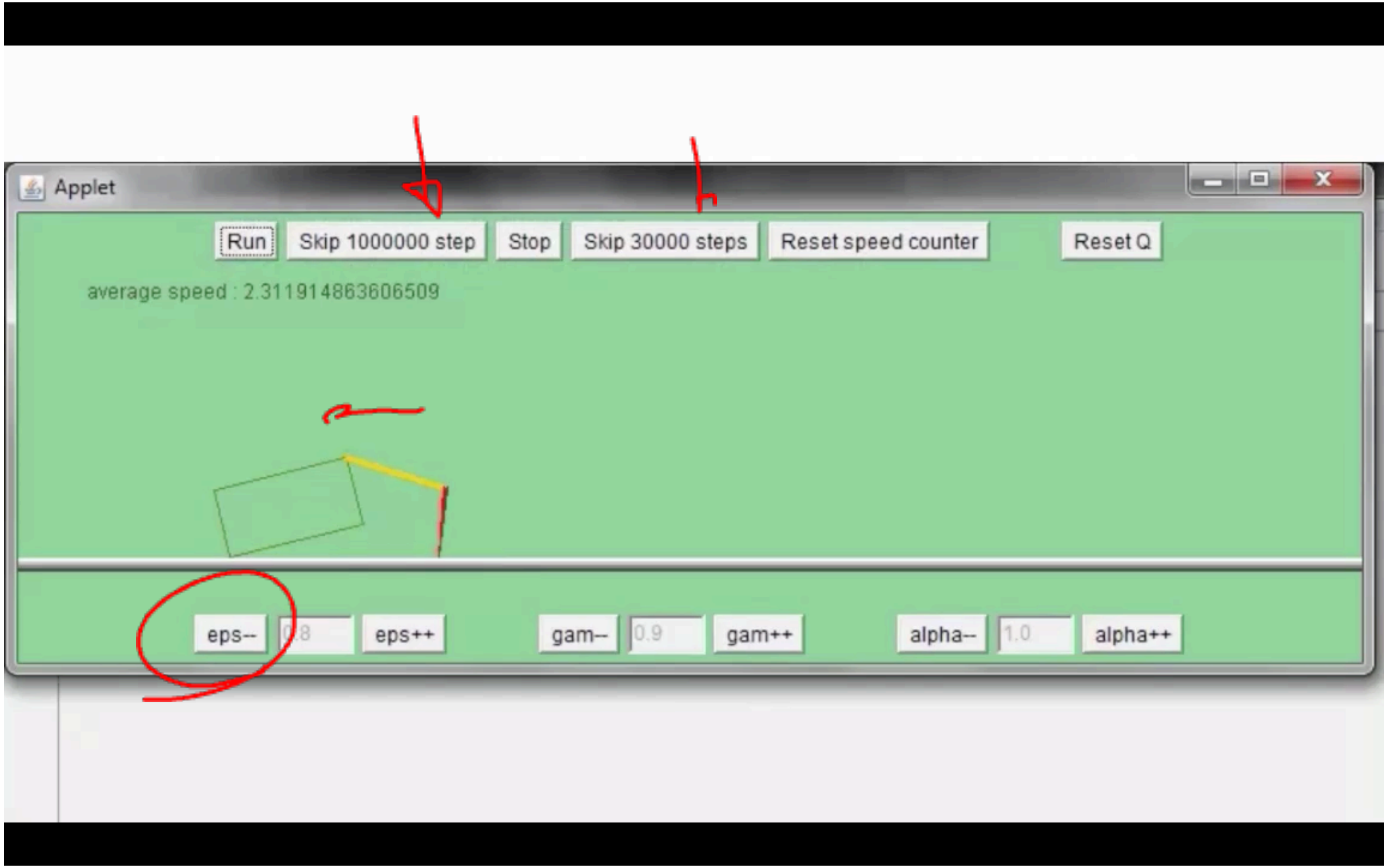
- https://www.youtube.com/watch?v=x4O8pojMF0w

The Crawler!



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

Video of Demo Crawler Bot

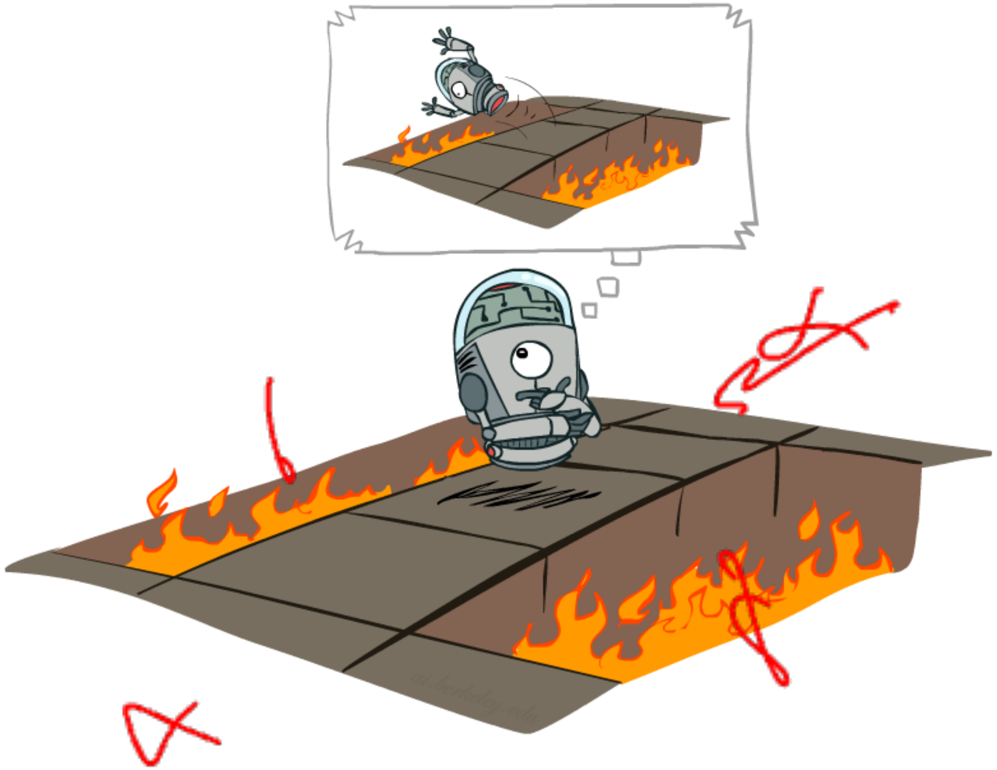


Reinforcement Learning

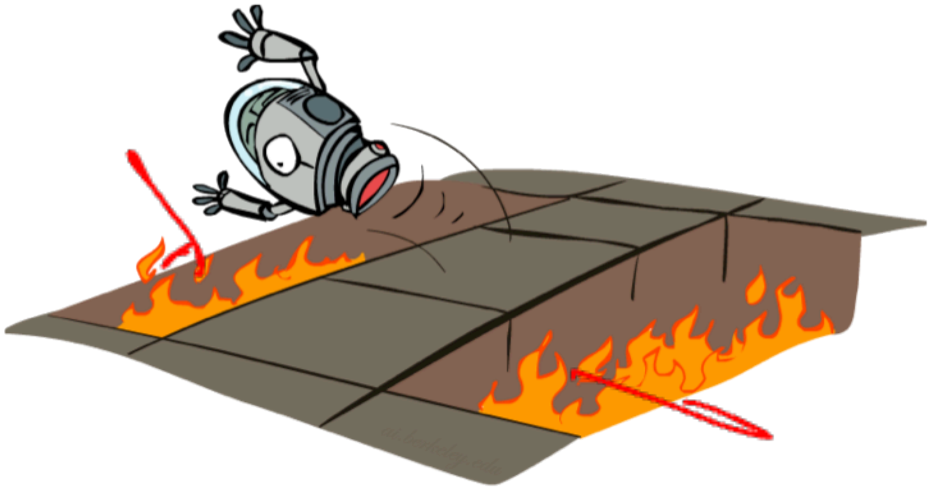
- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model $T(s,a,s')$
  - A reward function $R(s,a,s')$
- Still looking for a policy $\pi(s)$

- 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)

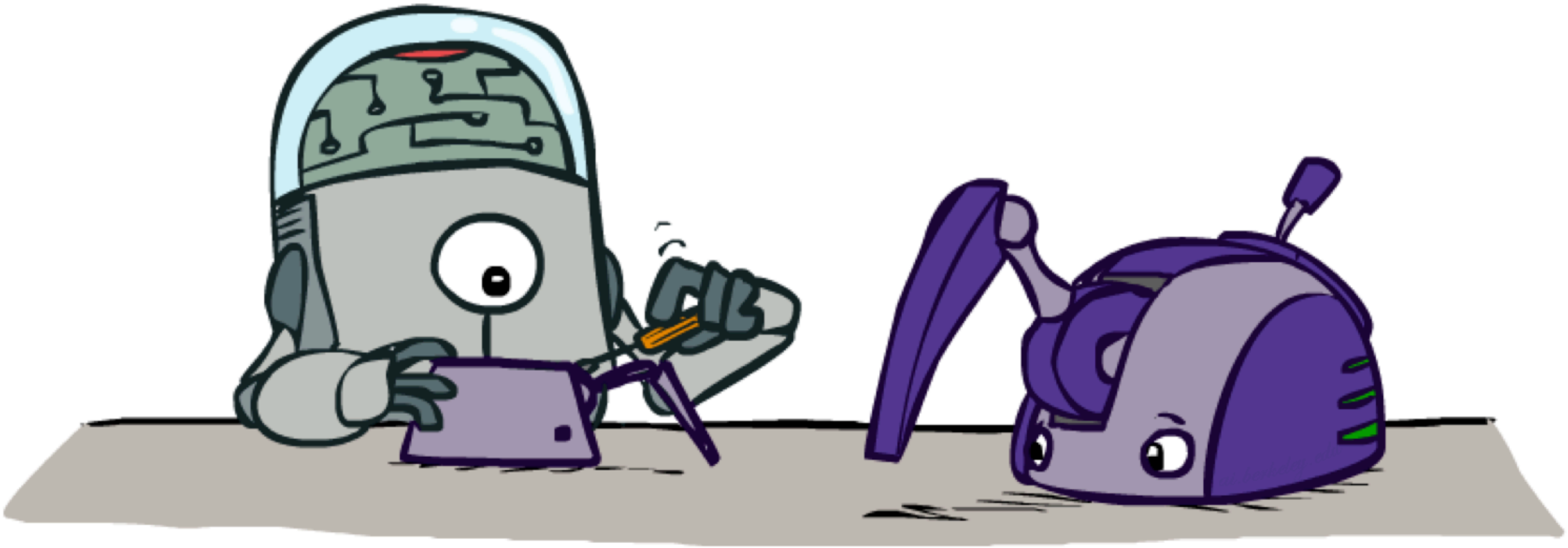


Offline Solution



Online Learning

Model-Based Learning



Model-Based Learning

$P(s,a)$ $s'/|N|$

- 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 $\hat{T}(s,a,s')$
  - Normalize to g $\hat{R}(s,a,s')$ imate of
  - Discover each when we experience (s, a, s′)

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

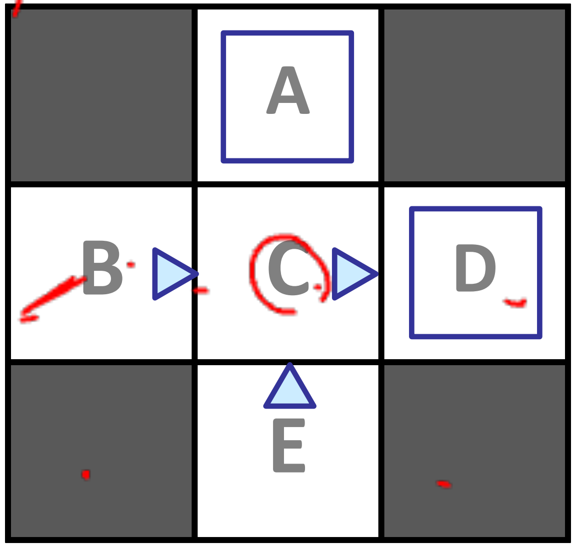
T

R

Example: Model-Based Learning

$\hat{T}$ $\hat{R}$

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

$$\widehat{T}(s, a, s')$$

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

$\hat{R}$(B, east, C) =

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

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

Model based → Model free

Analogy: Expected Age

40% 22
20% 23

Goal: Compute expected age of cse573 students

Known P(A)

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

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

4, 4, 4, 5, 5

Unknown P(A): "Model Based"

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

$\hat{P}(4) = \frac{3}{5}$

$\hat{P}(5) = \frac{2}{5}$

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

$\boxed{\frac{3}{5}} \times 4 + \boxed{\frac{2}{5}} \times 5$

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

Unknown P(A): "Model Free"

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

$\frac{1}{5}(4 + 4 + 4 + 5 + 5)$

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

CSE 573: Artificial Intelligence

Hanna Hajishirzi

Reinforcement Learning

slides adapted from
Dan Klein, Pieter Abbeel ai.berkeley.edu
And Dan Weld, Luke Zettlemoyer

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
- Big Idea: Compute all averages over T using sample outcomes

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

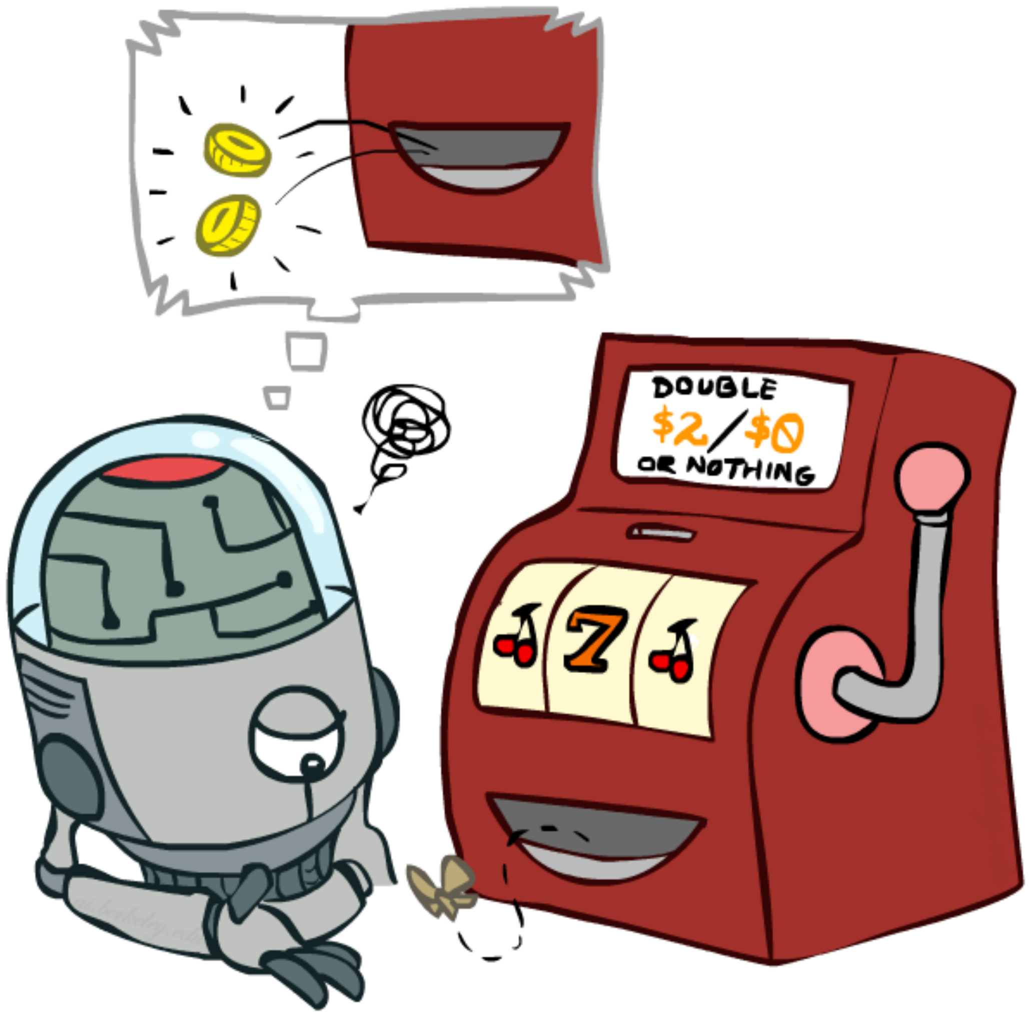
Unknown MDP: Model-Based

$\hat{T}$ $\hat{R}$

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

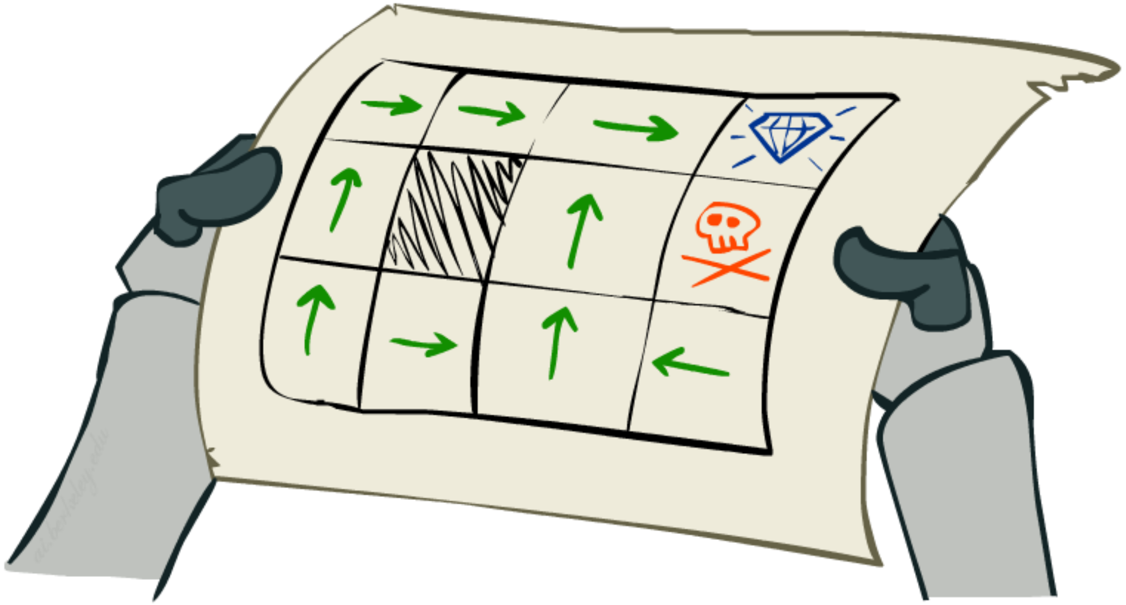
Unknown MDP: Model-Free

Model-Free Learning



Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy π(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

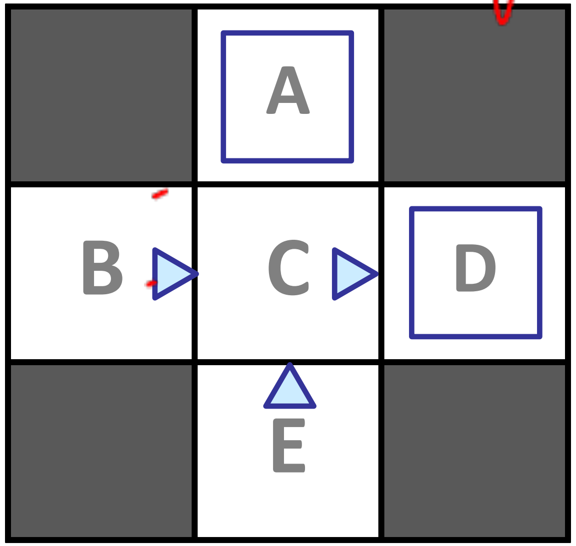
- Goal: Compute values for each state under π

- 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

+8 C: 9

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

+8 C: 9

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

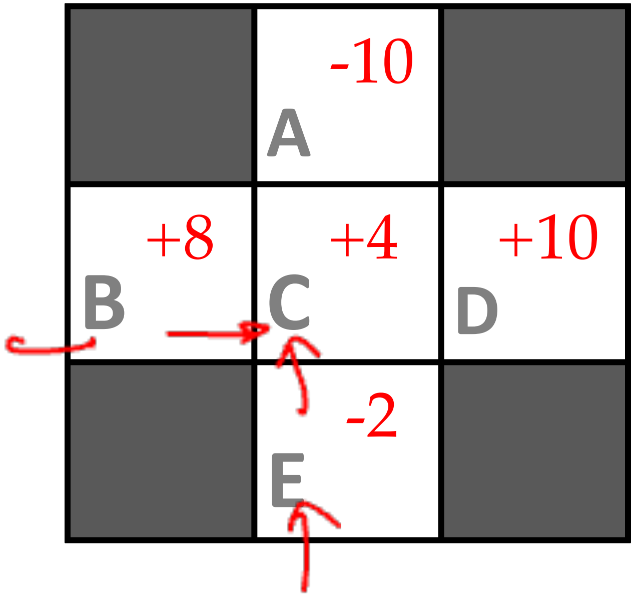
C: +9

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

C: -11

Output Values



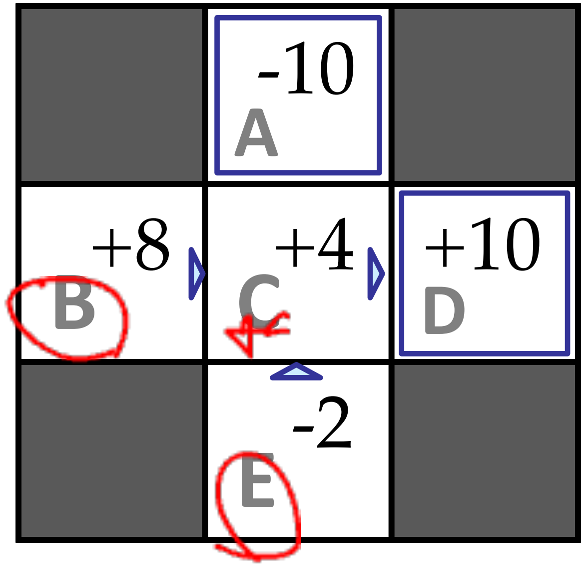
If B and E both go to C under this policy, how can their values be different?

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



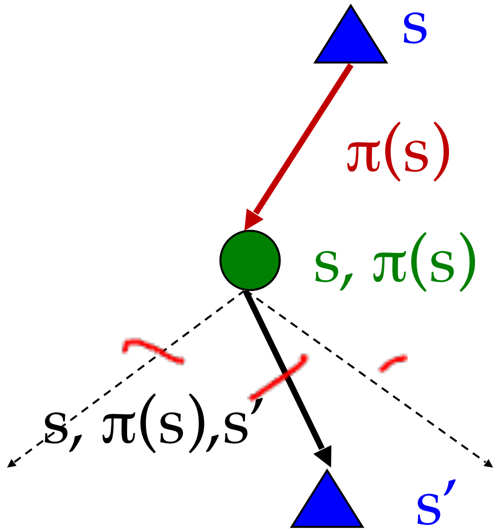
If B and E both go to C under this policy, how can their values be different?

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:

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

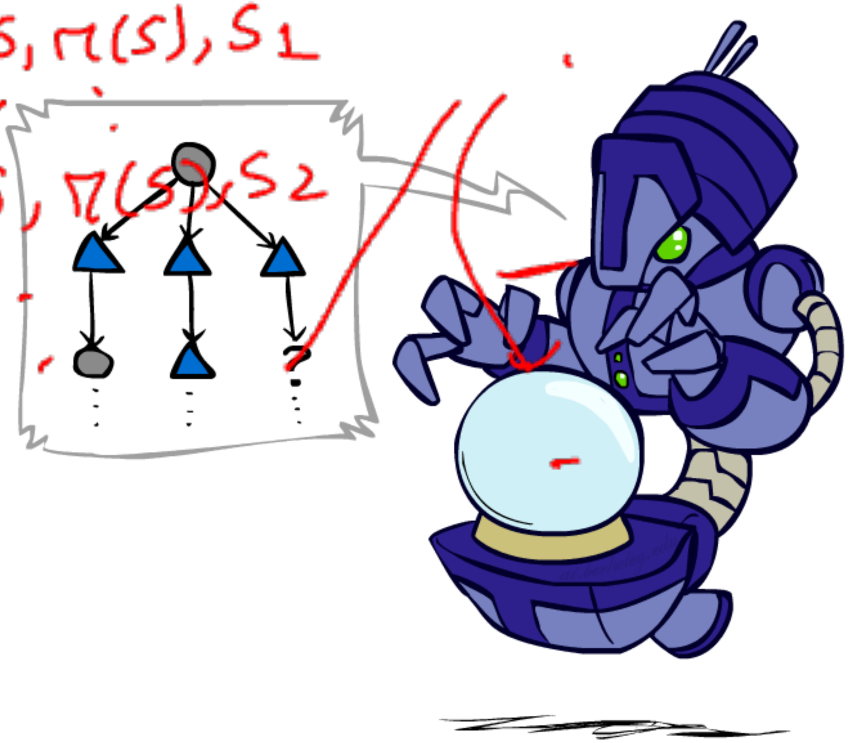
- Idea: Take samples of outcomes s' (by doing the action!) and average

$$sample_1 = \boxed{R(s,\pi(s),s'_1) + \gamma V^{\pi}_{k}(s'_1)}$$ s, π(s), s1

$$sample_2 = R(s,\pi(s),s'_2) + \gamma V^{\pi}_{k}(s'_2)$$ s, π(s), s2

$$\dots$$

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

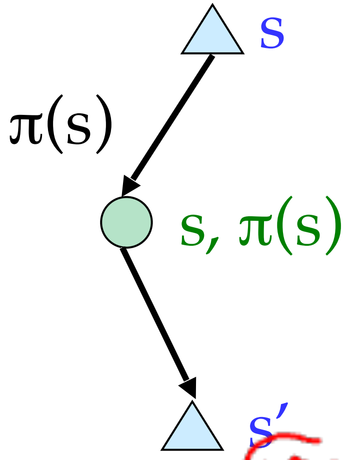


$$V^{\pi}_{k+1}(s) \leftarrow \frac{1}{n}\sum_{i} sample_i$$

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

V(s)



Sample

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

0.9 0.1

V 0.1

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

    $$\bar{x}_n = \frac{\alpha\cdot x_n + \alpha(1-\alpha)x_{n-1} + \cdots}{Z}$$

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

- ~~Decreasing learning rate~~ (alpha) can give converging averages

$R(s,a,s')$

Example: Temporal Difference Learning

$(1-\alpha) \cdot V^\pi + \alpha \text{ sample}$

$0 + 0.5 \times -2$

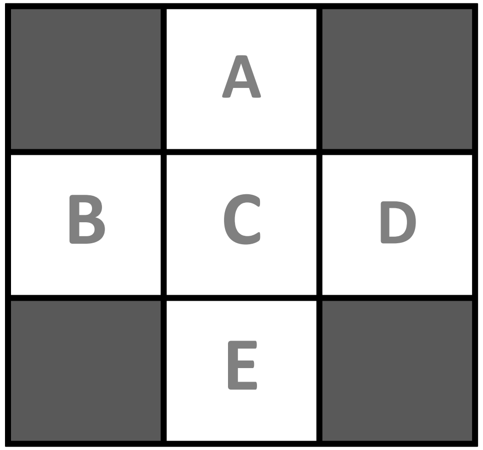
$0 + 0.5 \times 6$

$R(s,a,s') + \gamma V^\pi(s')$

Sample: $-2 + 0$

$-2 + 1 \times 8 = 6$

States

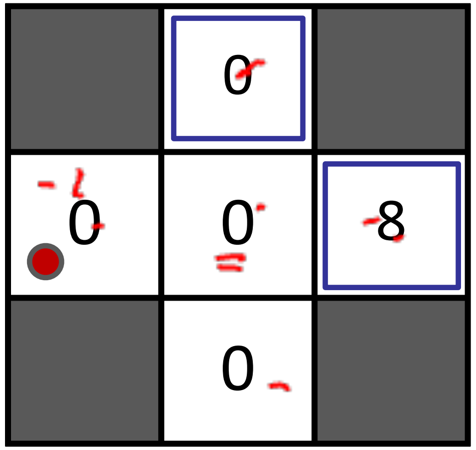


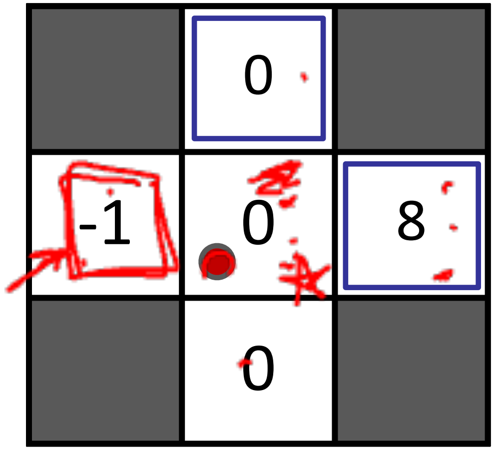
Assume: $\gamma = 1$, $\alpha = 1/2$

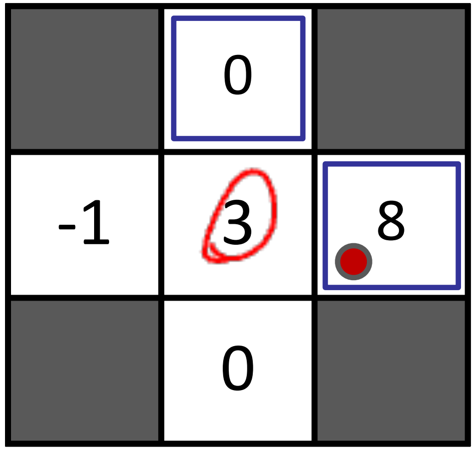
Observed Transitions

B, east, C, -2

C, east, D, -2







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

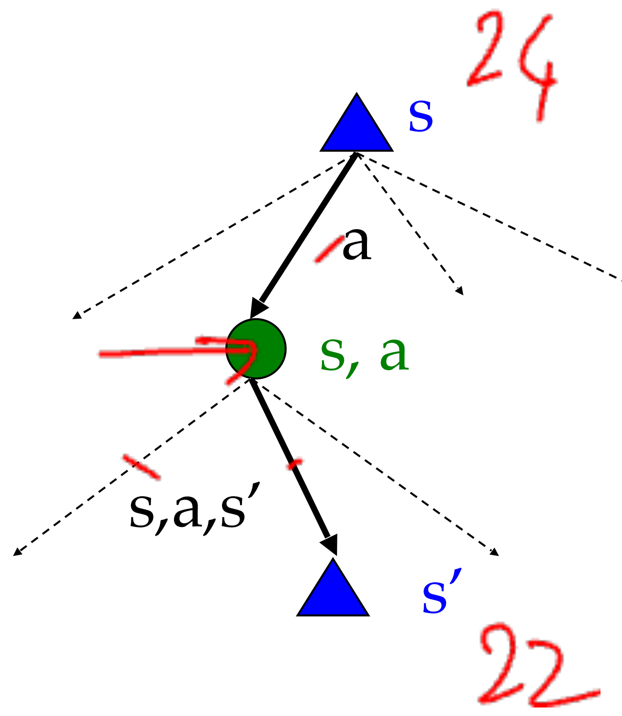
sample

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:

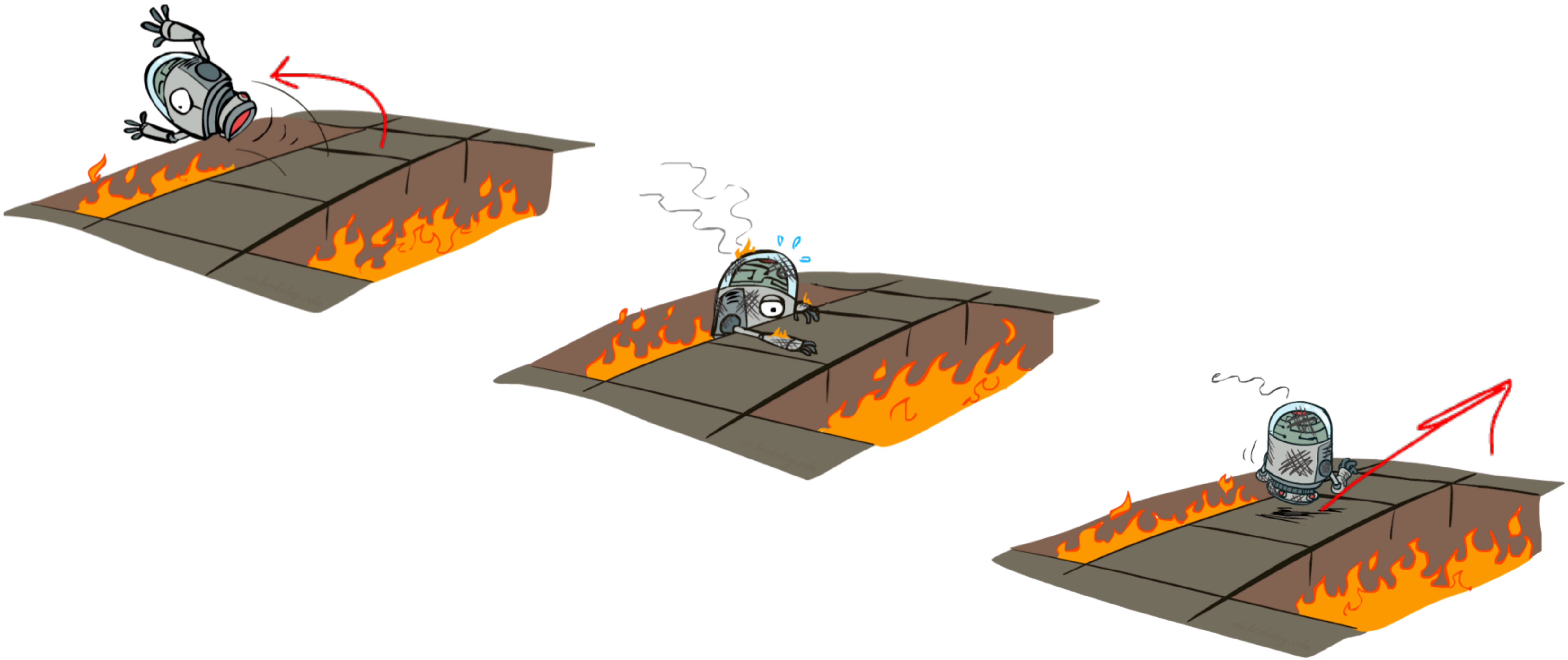
$$\pi(s) = \arg\max_a Q(s,a)$$

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



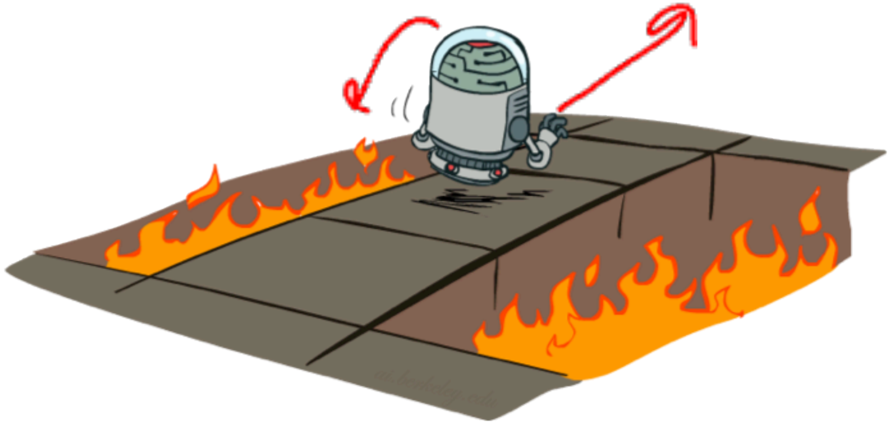
- Idea: learn ~~Q-values,~~ not values
- Makes action selection model-free too!

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:

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

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

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given Q_k, calculate the depth k+1 q-values for all q-states:

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

Q-Learning

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

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

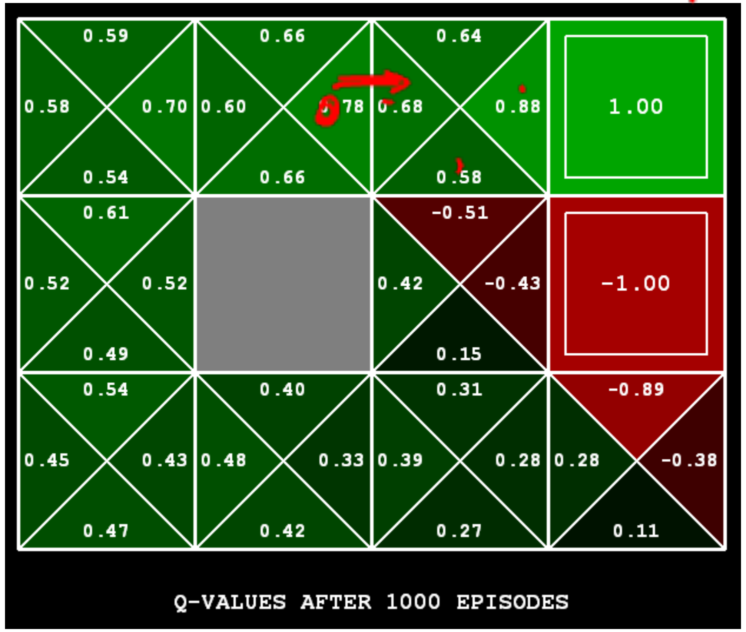
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

no longer policy evaluation!

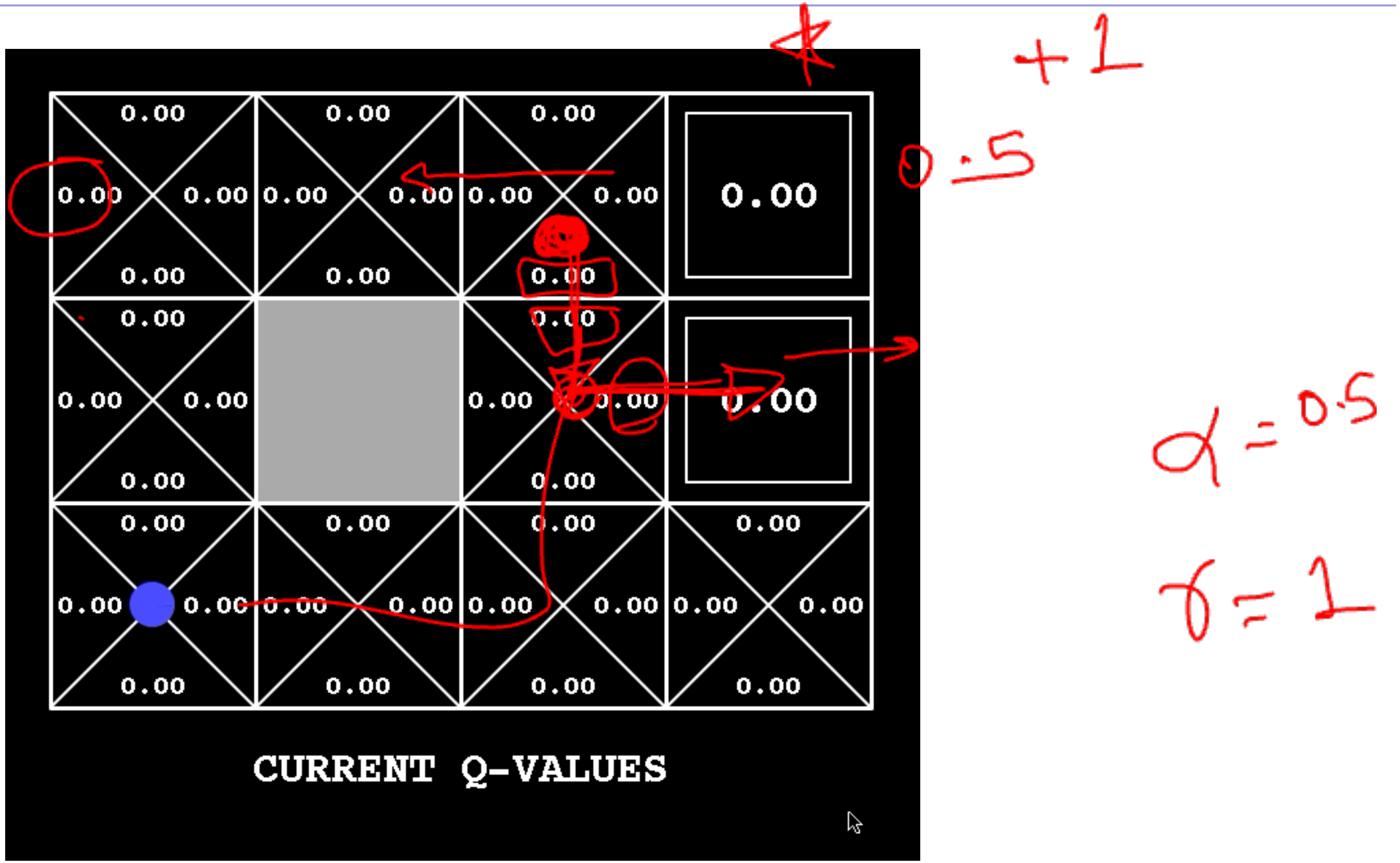
  - Incorporate the new estimate into a running average:

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

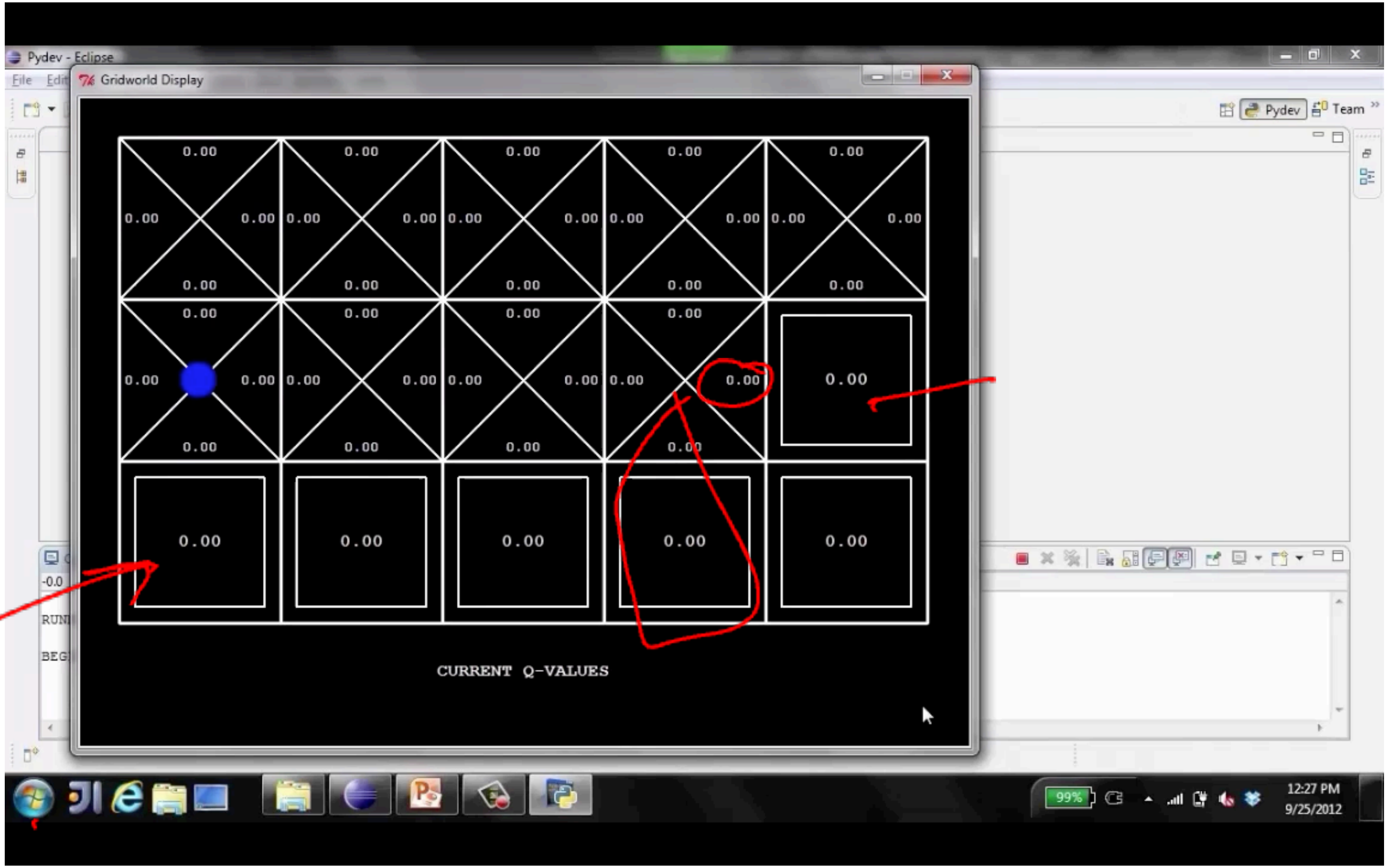


[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

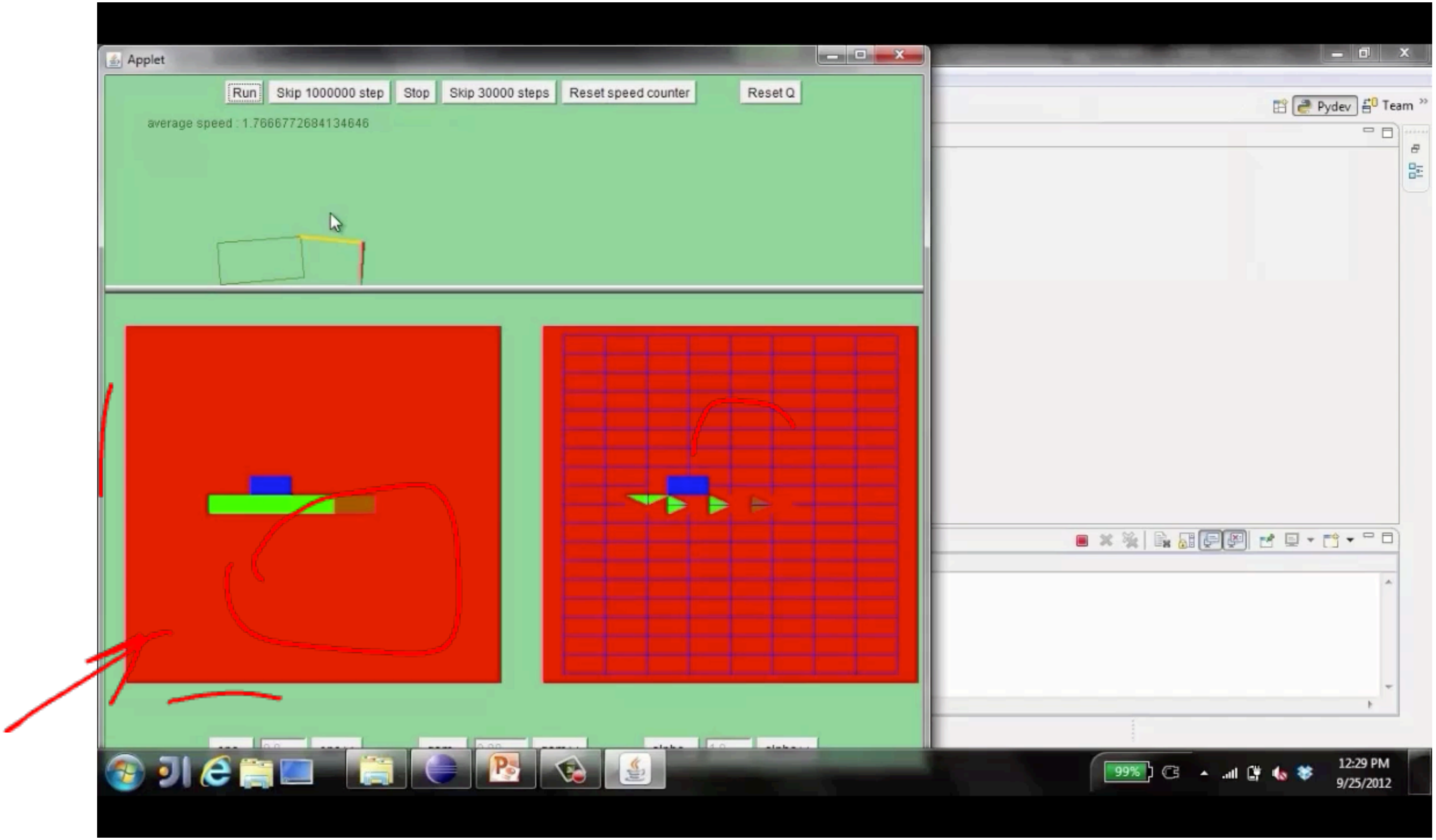
Q-Learning Demo



Video of Demo Q-Learning -- Gridworld

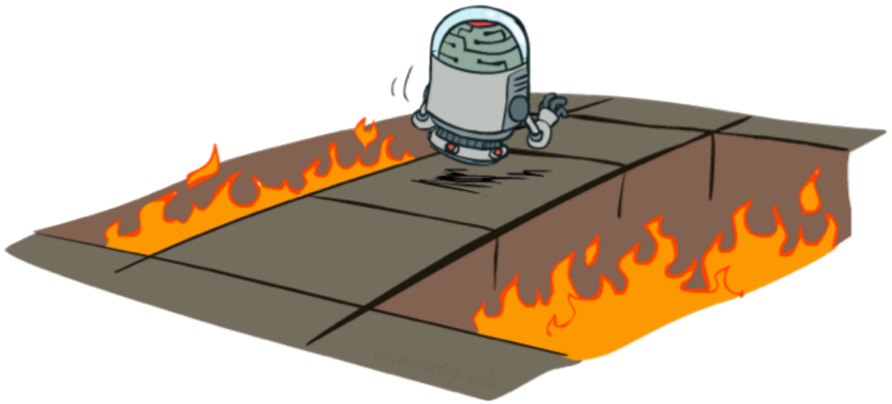


Video of Demo Q-Learning -- Crawler



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

- 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



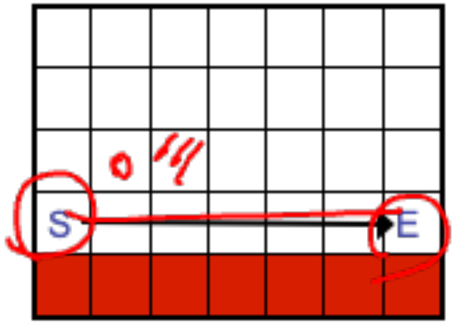
- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: expl~~oration vs. exploitation~~
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

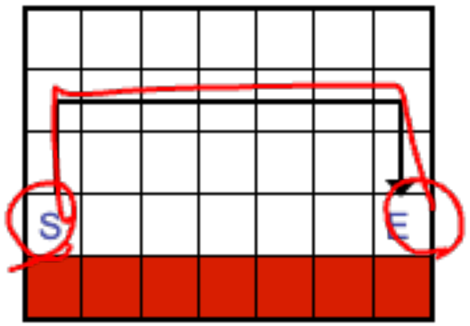
Q-Learning Properties

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

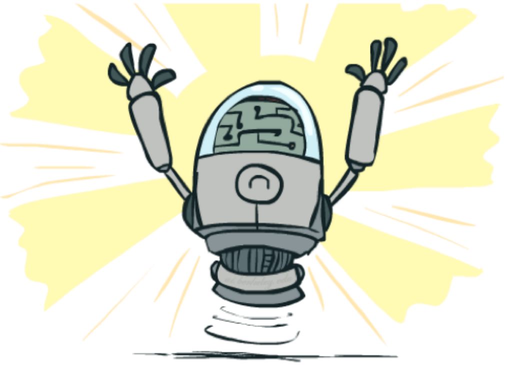
$R(\ ) + \max_{a'} Q(s, a')$

- 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 (!)



Discussion: Model-Based vs Model-Free RL