Reinforcement Learning

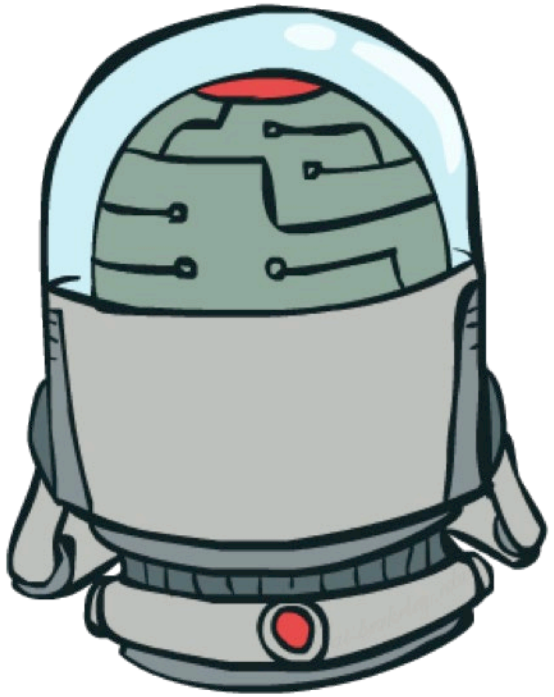
Outline

- Offline Planning vs Online Learning
  - Markov Decision Processes is a planning
  - Reinforcement Learning is a learning
- What is Reinforcement Learning
- Solving an RL problem
  - Model-Based Learning
  - Model-Free Learning
    - Temporal-Difference learning
      - Q-Learning

Double Bandits – Planning vs. Learning

1.00 $1

EU: 1.00 × 1 = 1

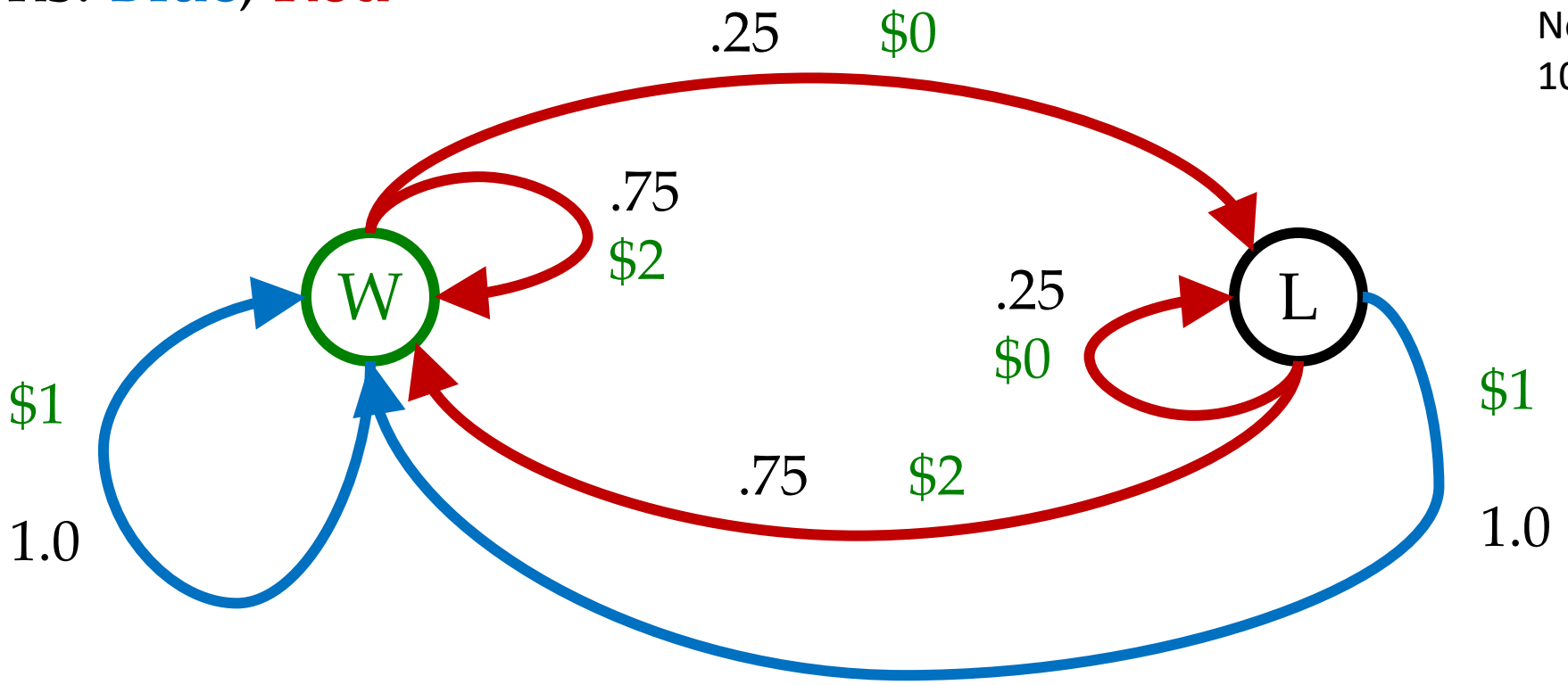


0.75 $2
0.25 $0

EU: 0.75 × 2 + 0.25 × 0 = 1.5

Offline Planning (MDP)

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



No discount (ϒ=1)
100 steps (k=100)

Let's Play!

- Solving MDP is offline learning
  - You need to know the details of the MDP
  - You determine all quantities through computation
  - You don't actually play the game.

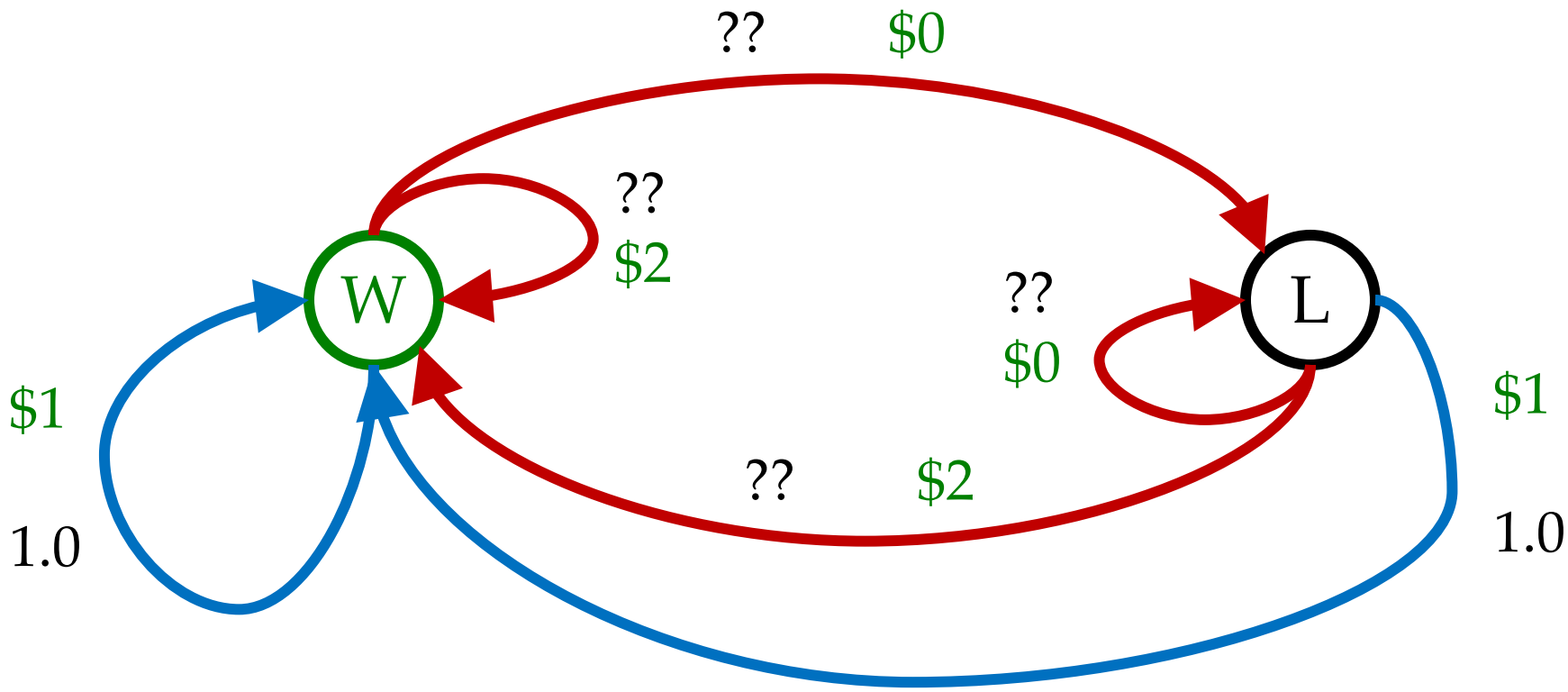
- Play Red, value = 2.
- Play Blue, value = 1.
- Optimal policy is to play Red.

$1 $1 $1 $1 $1 ……100 steps

$2 $0 $0 $0 $0 …… 100 steps

Online Learning

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



Let's Play!

$1 $1 $1 $1 $1 ……

$2 $0 $0 $0 $0 ……

Try it again and again (sampling)

What Is Reinforcement Learning?

- It isn't a planning, it is a learning!
  - Specifically, reinforcement learning.
  - There was an MDP, but you couldn't solve it with just computation because you don't know some parameters in the MDP.
  - You needed to actually act to figure it out.

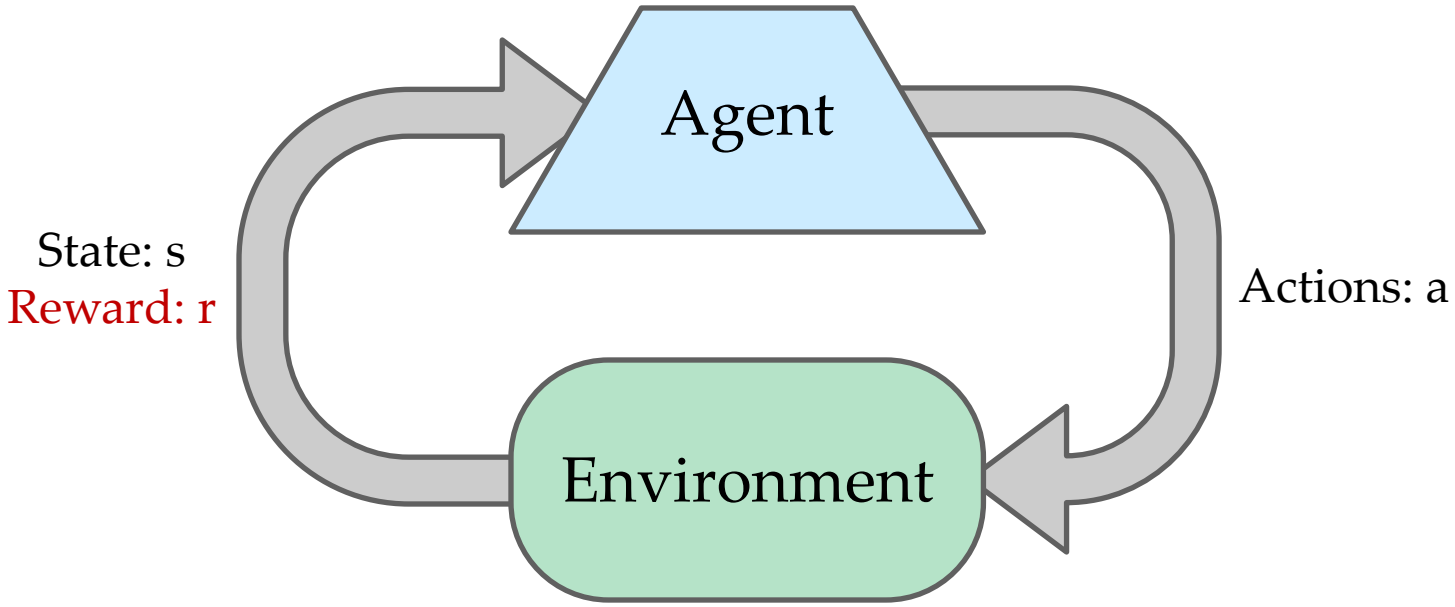
- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to find out how good/bad they are.
  - Exploitation: if you know they are good, you have to use what you know to collect more rewards from them.
  - Sampling: because of chance, you have to try things 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) $a \in A$
  - A model $T(s,a,s')$
  - A reward function $R(s,a,s')$
- Still looking for a policy $\pi(s)$

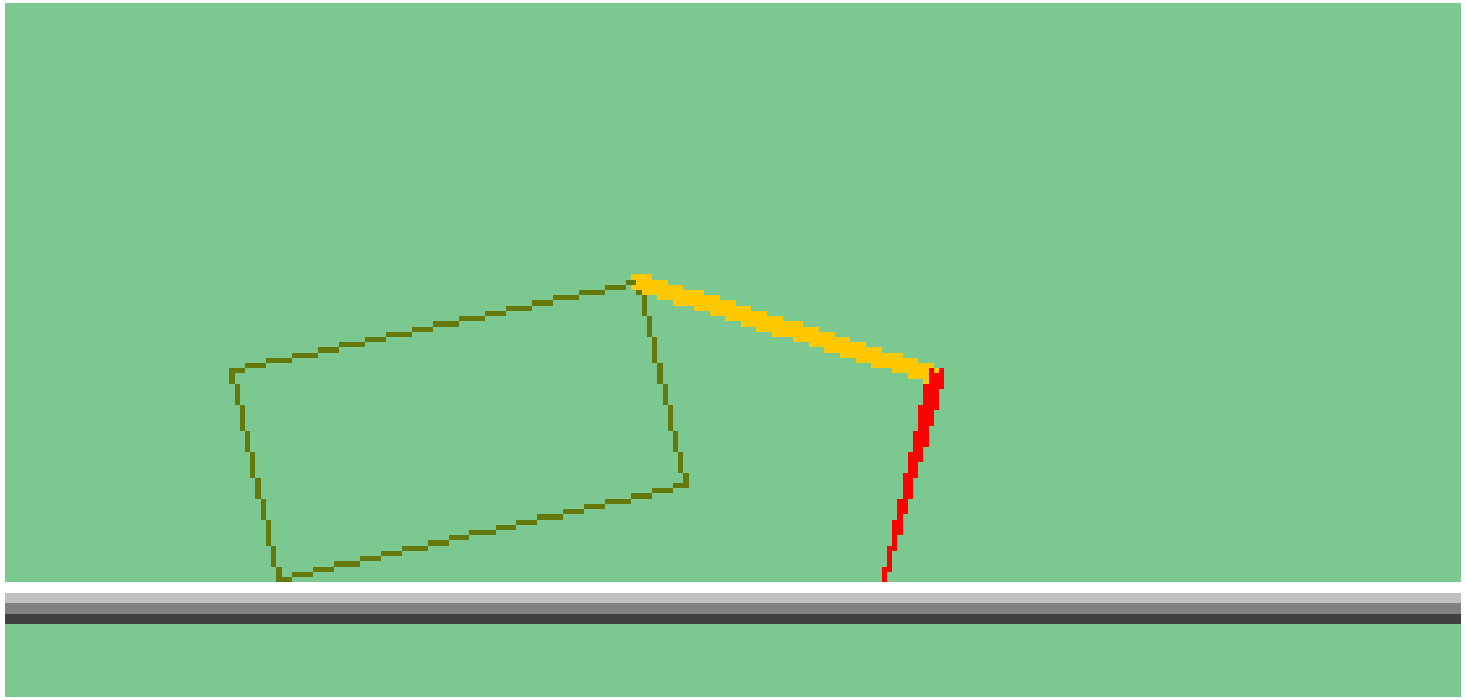
- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

Reinforcement Learning



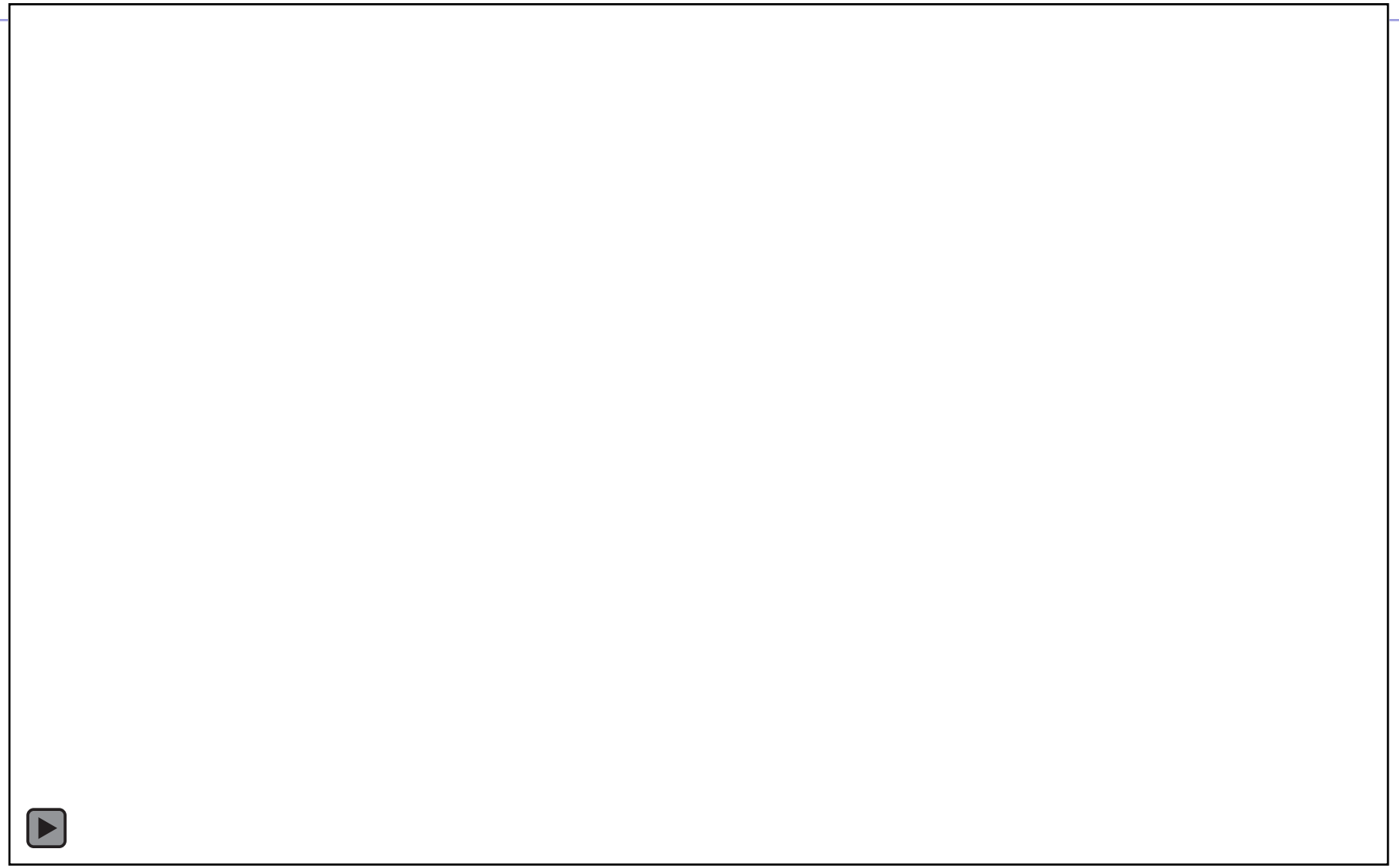
- Basic idea:
  - Receive feedback in the form of rewards.
  - Agent's utility is defined by the reward function.
  - Must (learn to) act so as to maximize expected rewards.
  - All learning is based on observed samples of outcomes!

The Crawler!

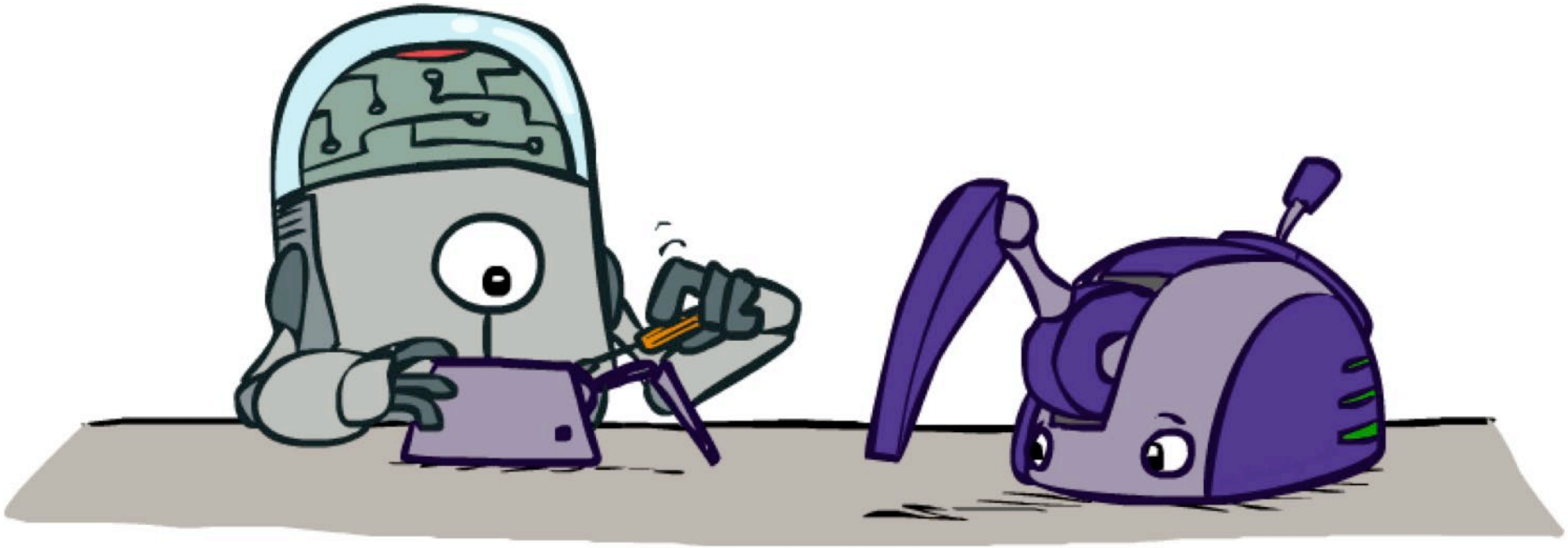


Two angles (states)
+r for moving right
-r for moving left

Video of Demo Crawler Bot



Model-Based Learning



Model-Based Learning

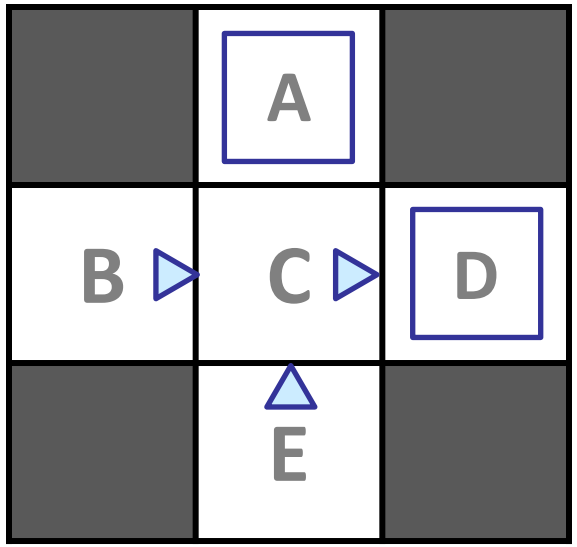
- Model-Based Idea:
  - Learn an approximate model based on experiences.
  - Solve for values as if the learned model were correct.

- Step 1: Learn empirical MDP model
  - Count outcomes s′ for each (s, a).
  - Normalize to give an estimate of $\hat{T}(s, a, s')$.
  - 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
…

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

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

Analogy: Expected Age

Goal: Compute expected age of CSCI-3344 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]

Answer

Goal: Compute expected age of CSCI-3344 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"

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

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

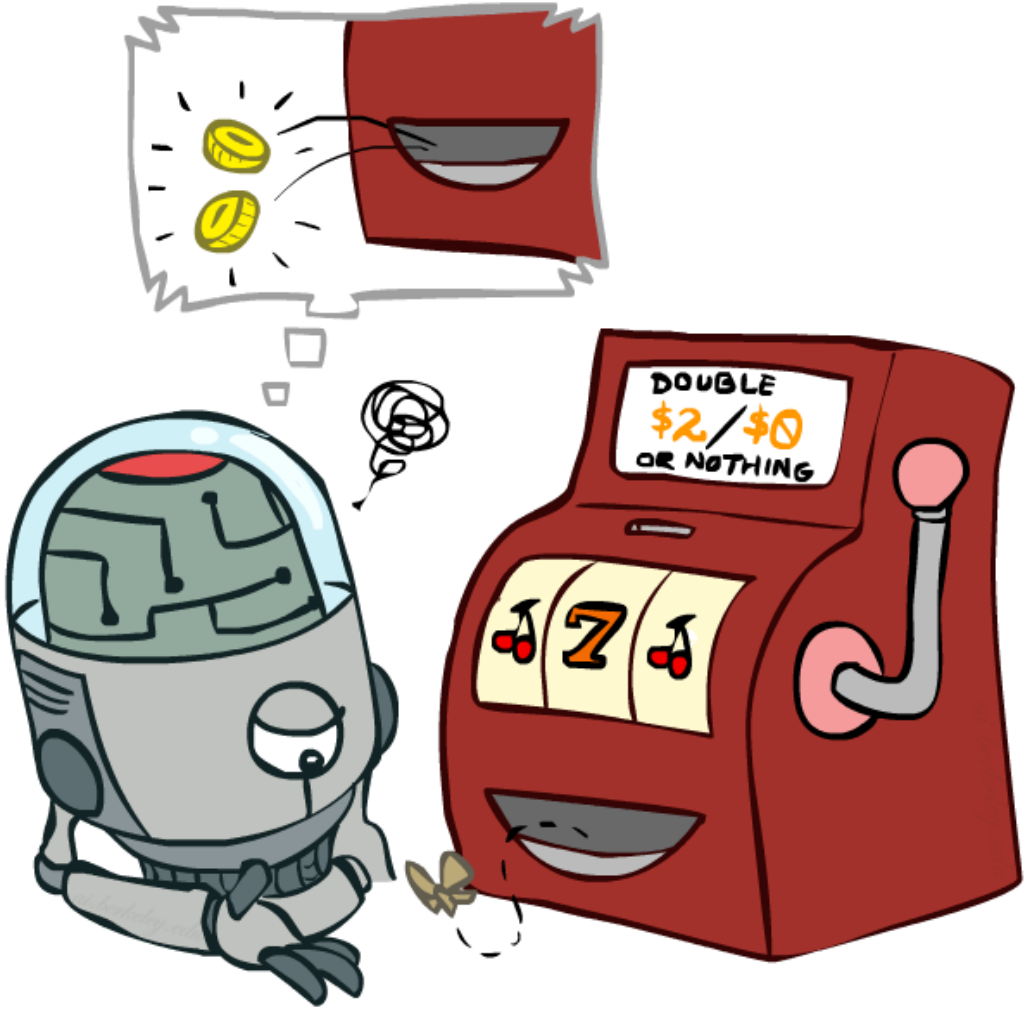
Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

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

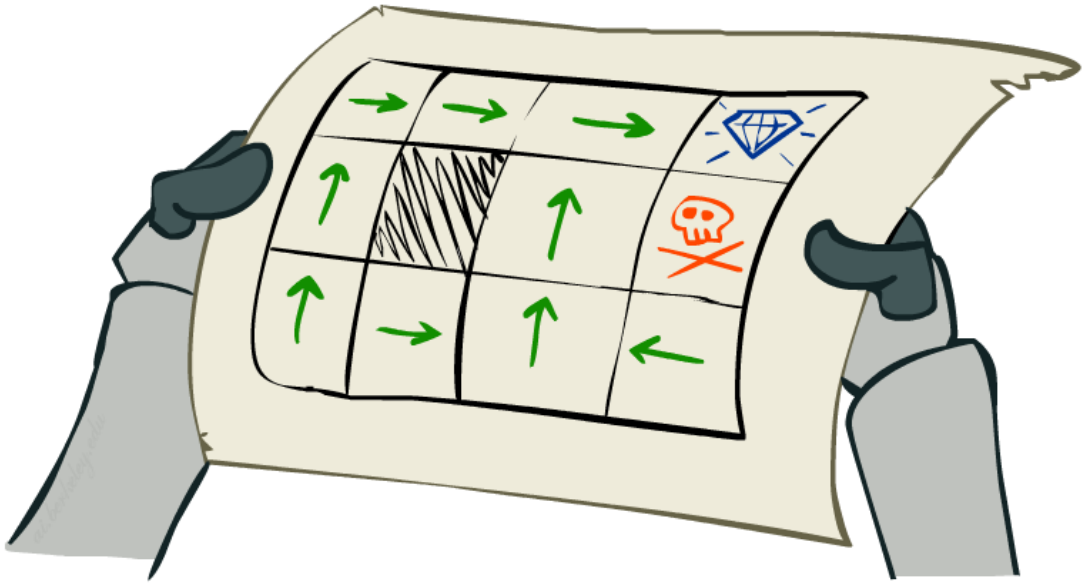
Why does this work? Because samples appear with the right frequencies.

Model-Free Learning



Passive Reinforcement Learning

- 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.



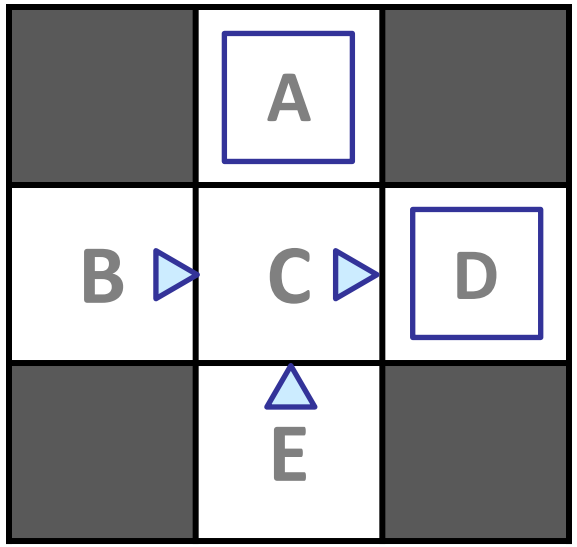
- In this case:
  - Learner is "along for the ride".
  - No choice about what actions to take.
  - Just execute the policy and learn from experience.
  - This is NOT offline planning! You learn the state values from actual experience.

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: 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

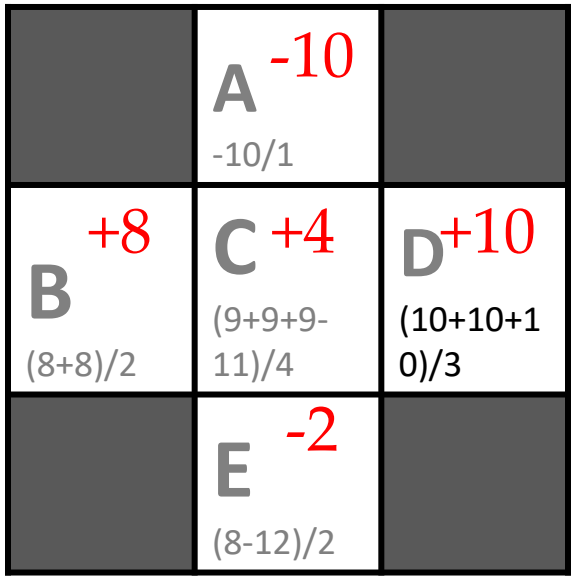
Episode 3

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

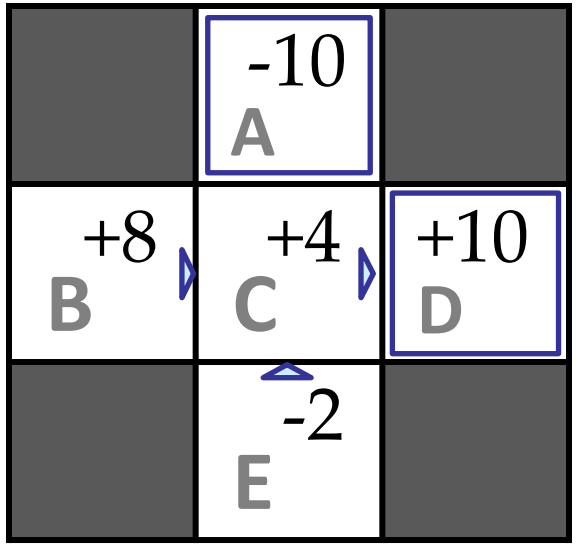
Output Values



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



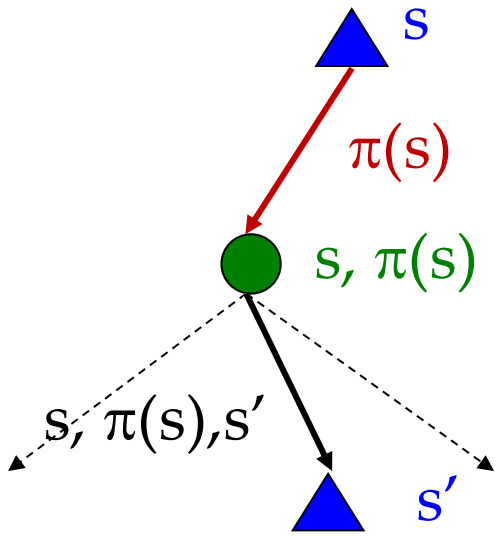
If B and E both go to C under this policy, how can their values be different?

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 do we take a weighted average without knowing the weights?

Sample-Based Policy Evaluation?

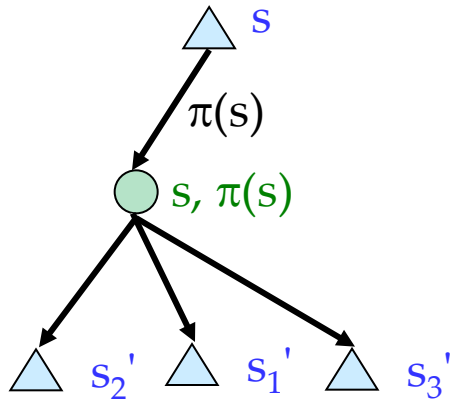
- We want to improve our estimate of V by computing these averages:

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

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

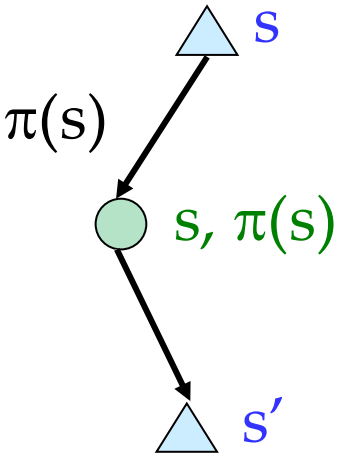
$$sample_1 = R(s, \pi(s), s'_1) + \gamma V^{\pi}_{k}(s'_1)$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V^{\pi}_{k}(s'_2)$$
$$\cdots$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V^{\pi}_{k}(s'_n)$$

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



Temporal Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s′, r)
  - The later samples will contribute updates more often b/c they are more accurate)



- 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: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(\underbrace{sample - V^{\pi}(s)}_{\text{Error (temporal difference)}})$

α is called the *learning rate*, determining how fast/slow to learn from the temporal difference

Learning Rate α

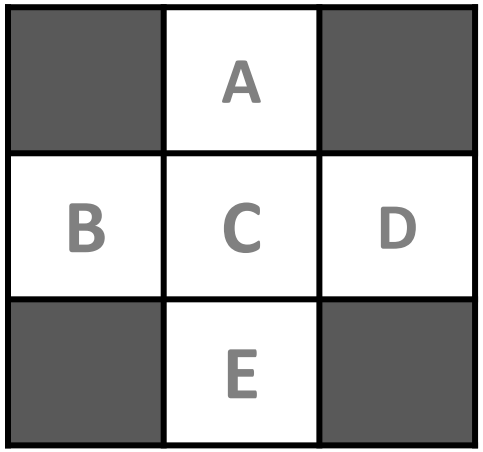
- Learning rate (α) determines how much new information overrides old knowledge.
- It controls the weight given to new experiences in reinforcement learning.
  - If α=1: You fully trust the most recent experience and completely overwrite the old value.
  - If α=0: You ignore the new information and keep the value unchanged.
  - If $\alpha \in (0,1)$: You take a weighted average between the old value and the new estimate.

Learning Rate α

- Small α (e.g. 0.01): Slow learning, stable updates, but takes longer to converge.
- Large α (e.g. 0.9): Fast learning, reacts quickly to new data, but can be unstable or noisy.
- A decaying learning rate (e.g. $\alpha_t = \frac{1}{1+t}$) is often used to learn quickly at first and become more stable over time. Need careful tuning.

Example: Temporal Difference Learning

States



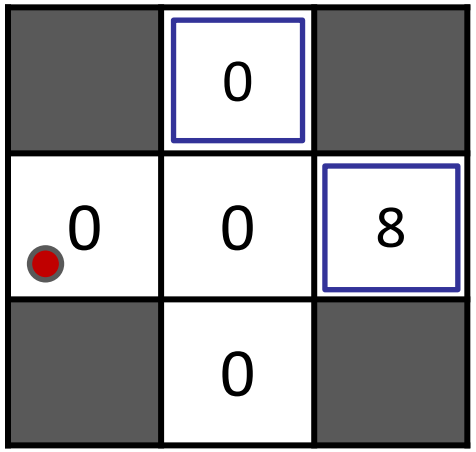
Assume: $\gamma = 1$, $\alpha = 0.5$

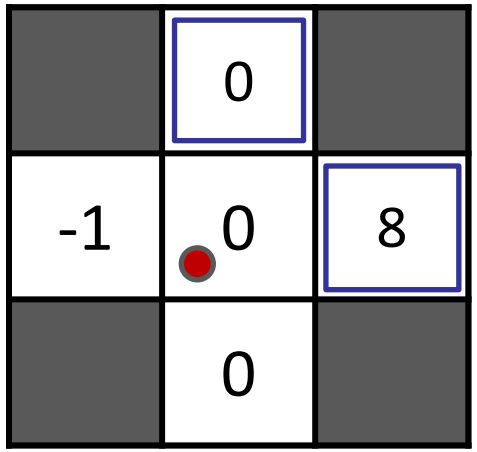
Initial Values

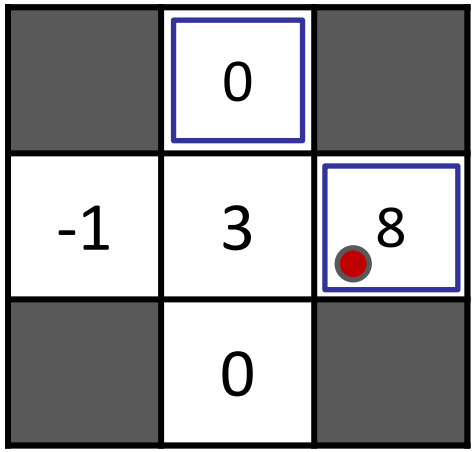
Observed Transitions

B, east, C, -2

C, east, D, -2







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

$V^{East}(B) = (1 - 0.5) \times 0 + 0.5[-2 + 1 \times 0] = -1$

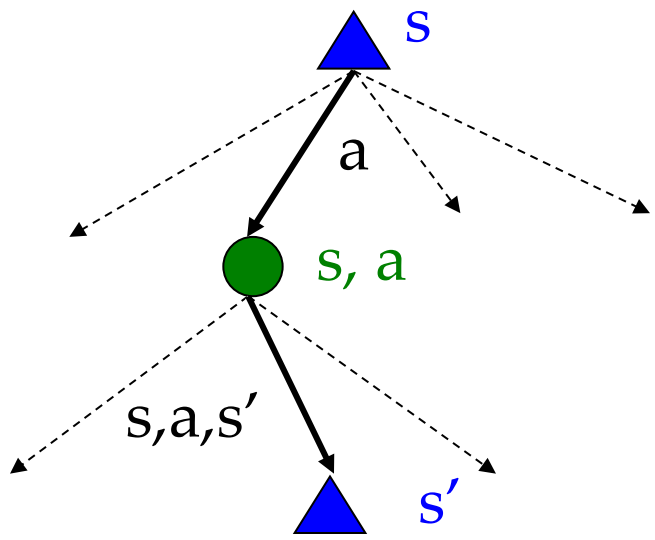
$V^{East}(C) = (1 - 0.5) \times 0 + 0.5[-2 + 1 \times 8] = 3$

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 and better) policy, we're sunk:

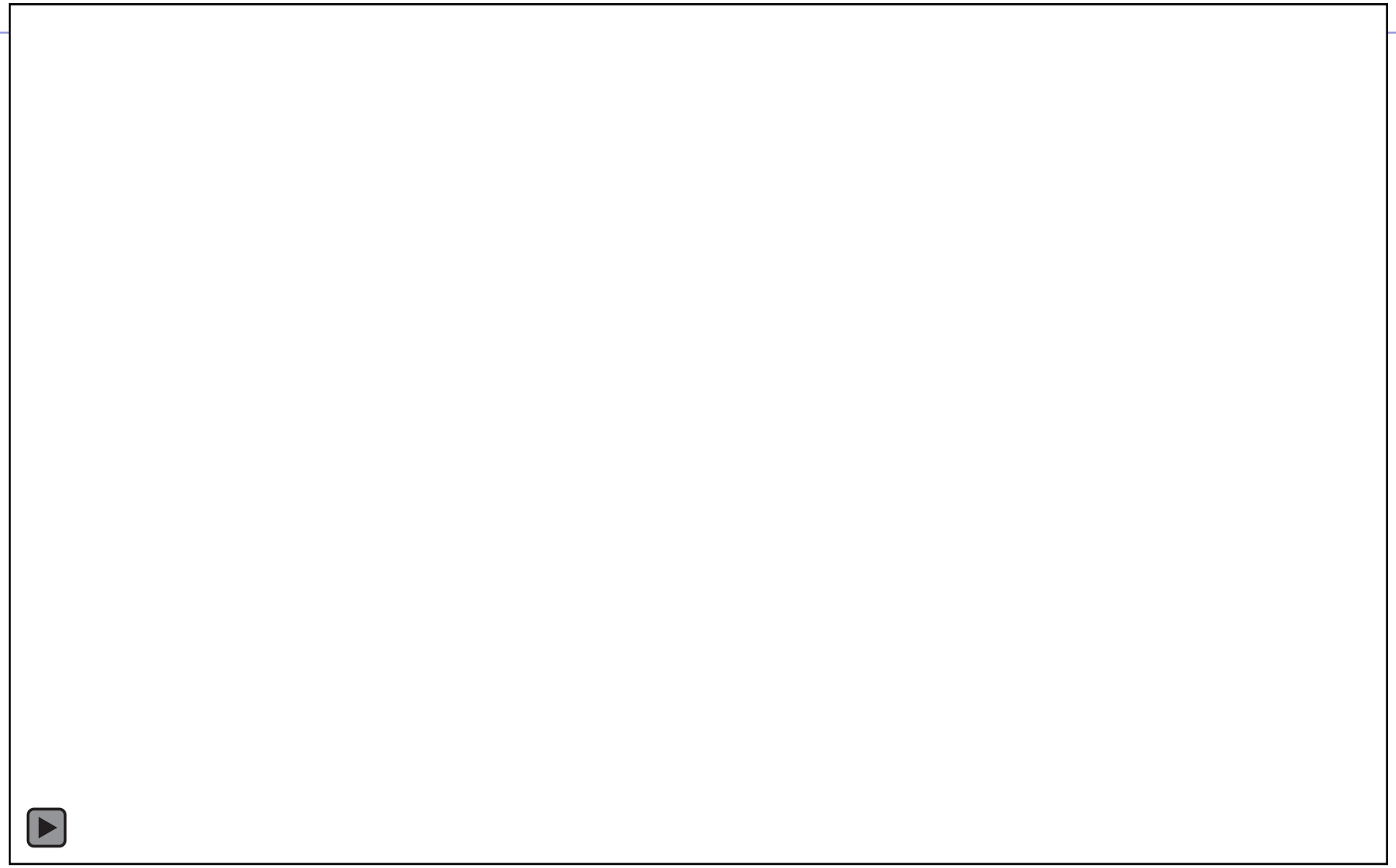
$$\pi(s) = \arg\max_a Q(s,a)$$

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



- Idea: learn Q-values, not V values.
- Makes action selection model-free too!
- This is called Q Learning.

Video of Demo Q-Learning -- Gridworld



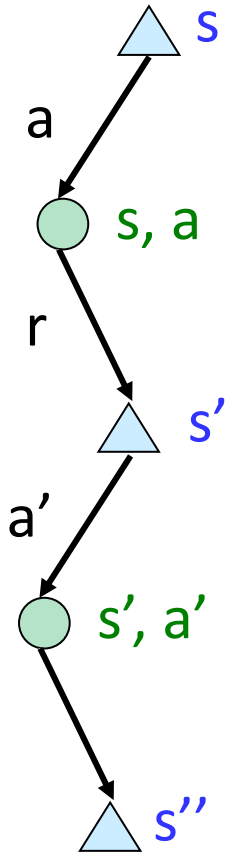
Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values.
  - Start with $V_0(s) = 0$
  - 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 $Q_0(s,a) = 0$.
  - 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.

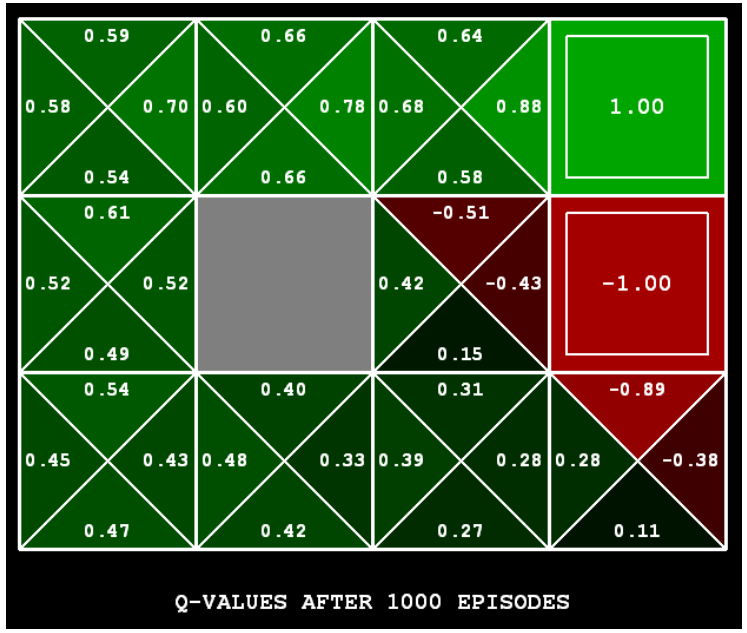
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- 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:

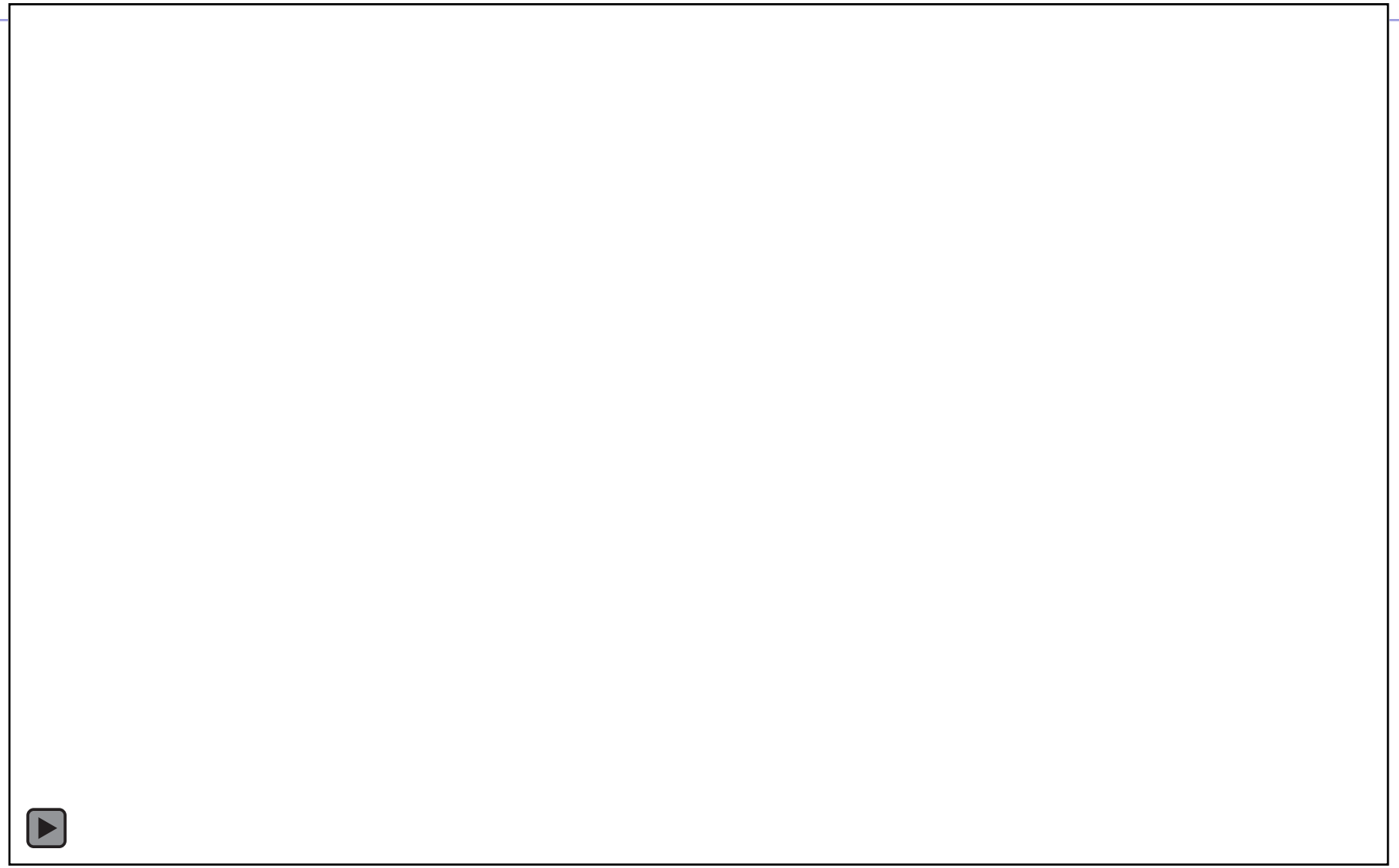
$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

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

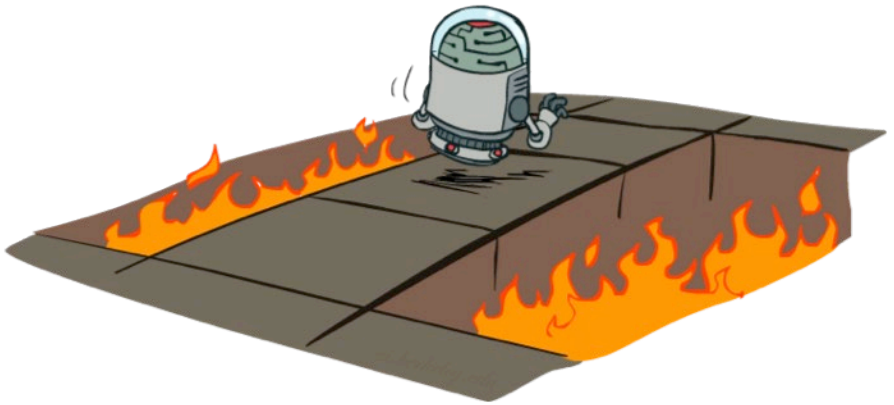


Video of Demo Q-Learning -- Crawler



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.

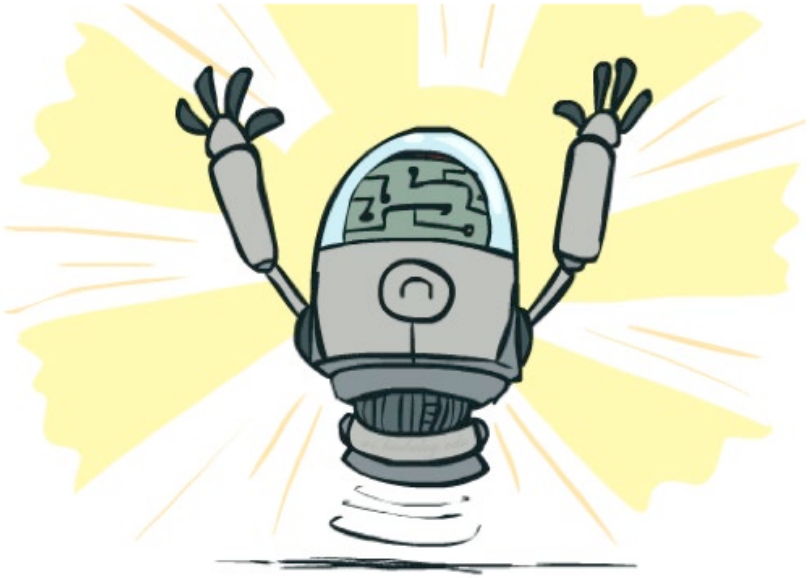


- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation.
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally (you just need to act frequently enough)!

- 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 (!)

Q-Learning Algorithm

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

ε-Greedy Action Selection

Exploration allows an agent to improve its current knowledge about each action, hopefully leading to long-term benefit. Improving the accuracy of the estimated action-values, enables an agent to make more informed decisions in the future.

Exploitation on the other hand, chooses the greedy action to get the most reward by exploiting the agent's current action-value estimates. But by being greedy with respect to action-value estimates, may not actually get the most reward and lead to sub-optimal behaviour.

When an agent explores, it gets more accurate estimates of action-values. And when it exploits, it might get more reward. It cannot, however, choose to do both simultaneously, which is also called the exploration-exploitation dilemma.

ε-Greedy Action Selection
Epsilon-Greedy is a simple method to balance exploration and exploitation by choosing between exploration and exploitation randomly. For example, if ε=0.9, 90% of the chance to choose Exploitation (max Q) and 10% to choose Exploration (another action).