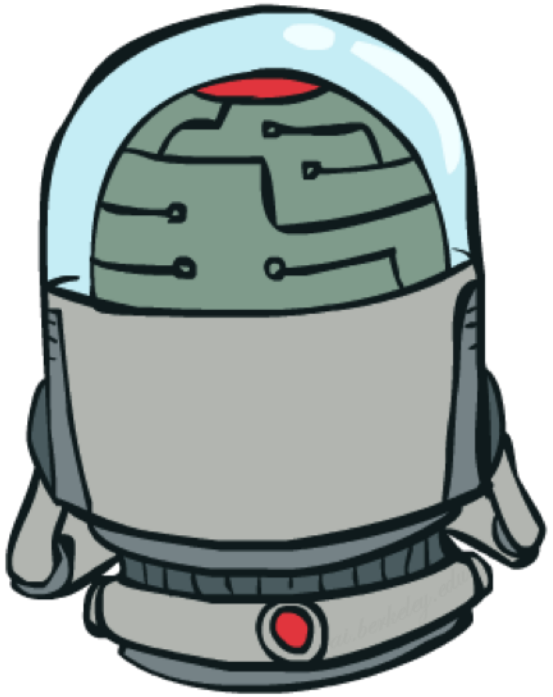
# Double Bandits

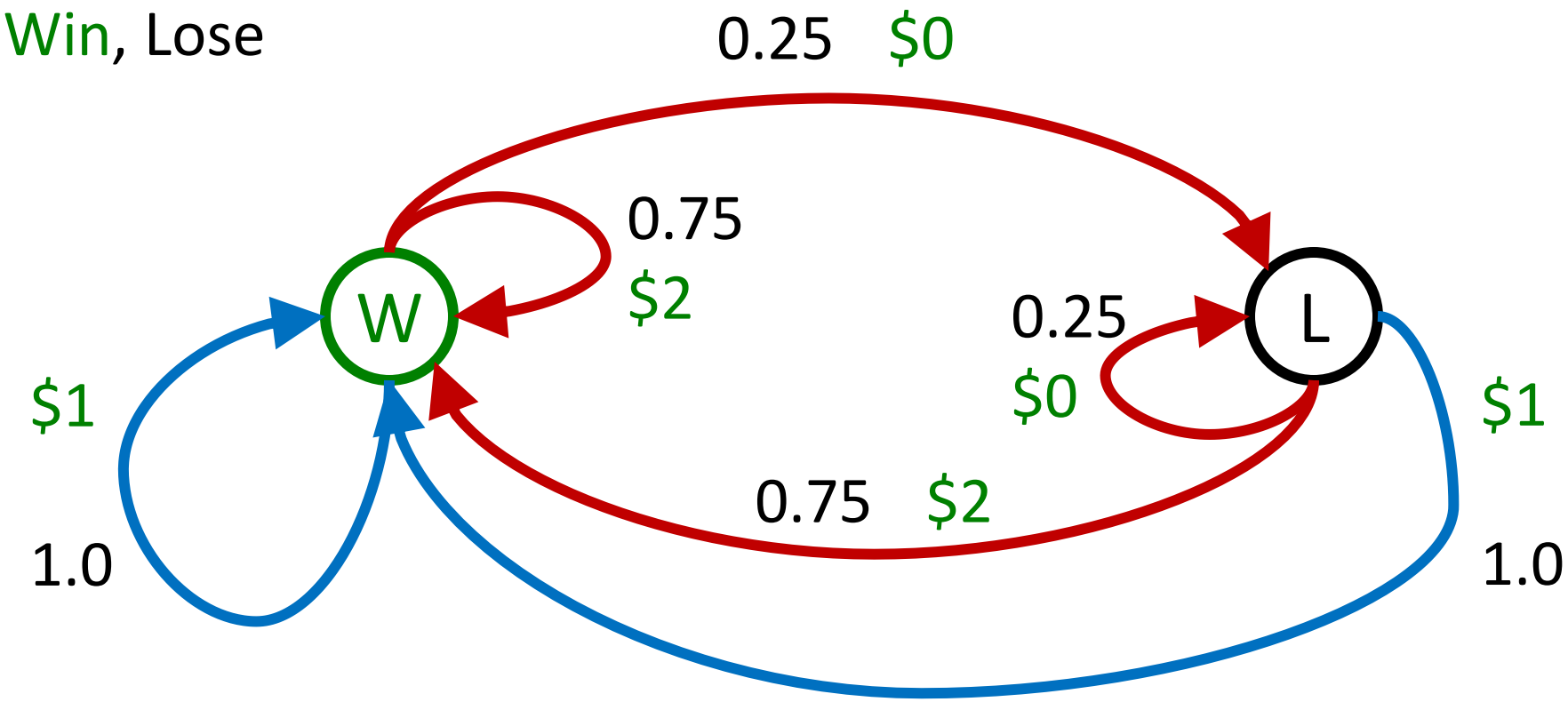




Slide: ai.berkeley.edu

# Double-Bandit MDP

Actions: *Blue, Red*

States: Win, Lose




Slide: ai.berkeley.edu

# Offline Planning

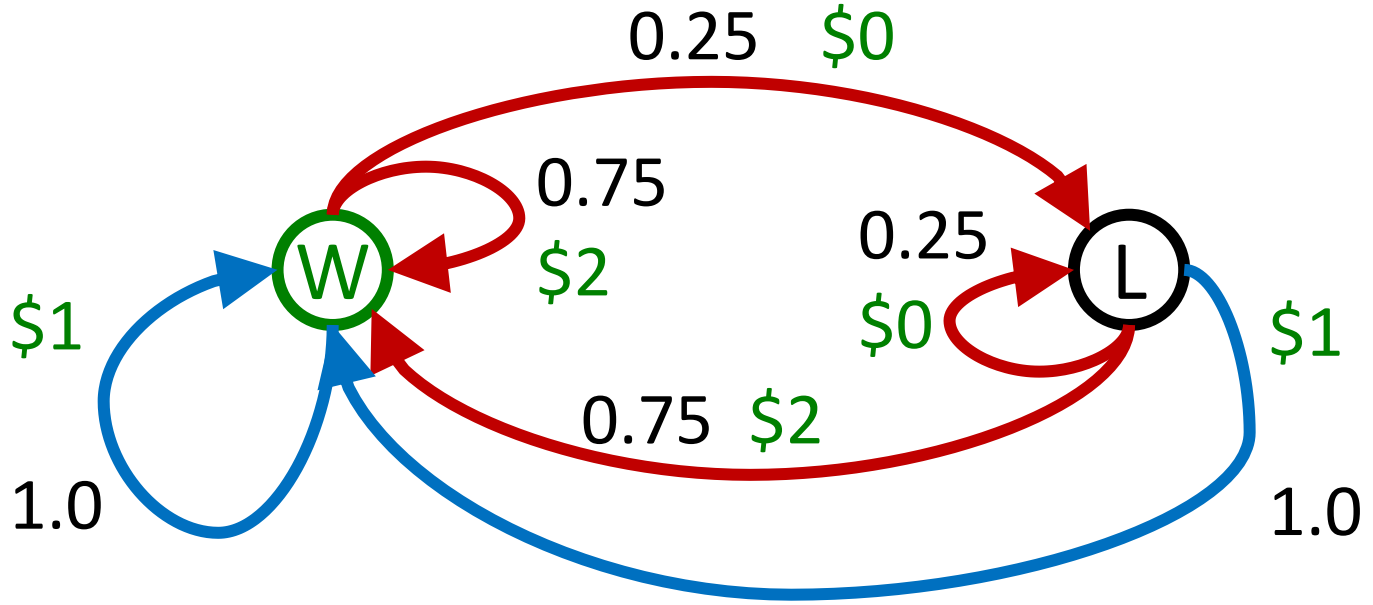
*No discount*

*100 time steps*

## 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!

| | Value |
|---|---|
| Play Red | 150 |
| Play Blue | 100 |



# Let’s Play!



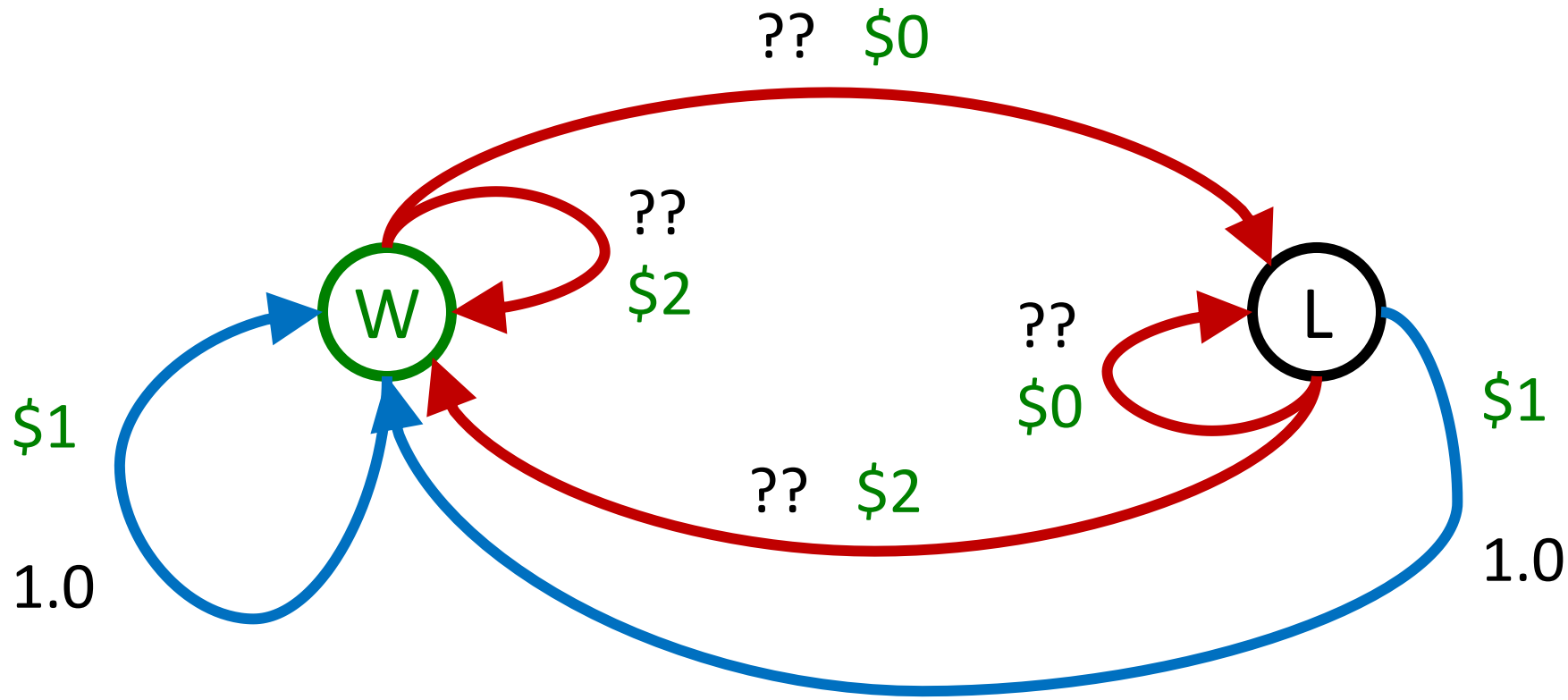
$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Slide: ai.berkeley.edu

# Online Planning

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



# Let’s Play!



$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

Slide: ai.berkeley.edu

# 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

# Overview: MDPs and Reinforcement Learning

## Known MDP: Offline Solution

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, $\pi$* | Value / policy iteration |
| Evaluate a fixed policy $\pi$ | Policy evaluation |

## Unknown MDP: Model-Based

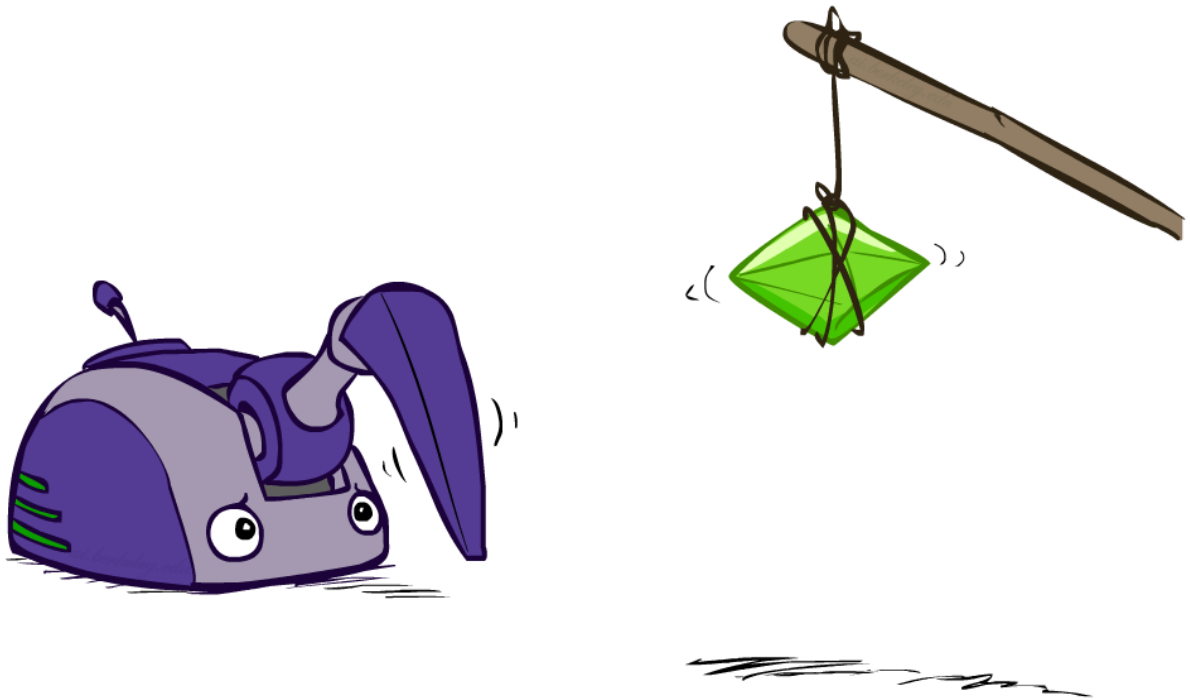
| Goal | Technique |
| --- | --- |
| Compute V*, Q*, $\pi$* | VI/PI on approx. MDP |
| Eval fixed policy $\pi$ | PE on approx. MDP |

## Unknown MDP: Model-Free

| Goal | Technique |
| --- | --- |
| Compute V*, Q*, $\pi$* | Q-learning |
| Eval fixed policy $\pi$ | TD/Value Learning |

# AI: Representation and Problem Solving

## Reinforcement Learning



Instructor: Pat Virtue

Slide credits: CMU AI and http://ai.berkeley.edu

# MDP Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

Policy evaluation: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall\, s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

# Poll 1

## Which of the following are used in policy iteration? Select all that apply.

A. Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

B. Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

C. Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

D. Policy evaluation: $V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \quad \forall\, s$

E. Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

# Poll 1

Which of the following are used in policy iteration? Select all that apply.

A. Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

B. Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

C. Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

✓ D. Policy evaluation: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall\, s$

✓ E. Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

# Poll 2

Rewards may depend on any combination of *state*, *action*, *next state*.

Which of the following are valid formulations of the Bellman equations?

Select all that apply.

*A.* $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

*B.* $V^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$

*C.* $V^*(s) = \max_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$

*D.* $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$

# Poll 2

Rewards may depend on any combination of *state*, *action*, *next state*.

Which of the following are valid formulations of the Bellman equations?

Select all that apply.

✓ *A.* $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

✓ *B.* $V^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$

✓ *C.* $V^*(s) = \max_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$

✓ *D.* $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$

# Reinforcement learning

## What if we didn't know $P(s'|s,a)$ and $R(s,a,s')$?

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

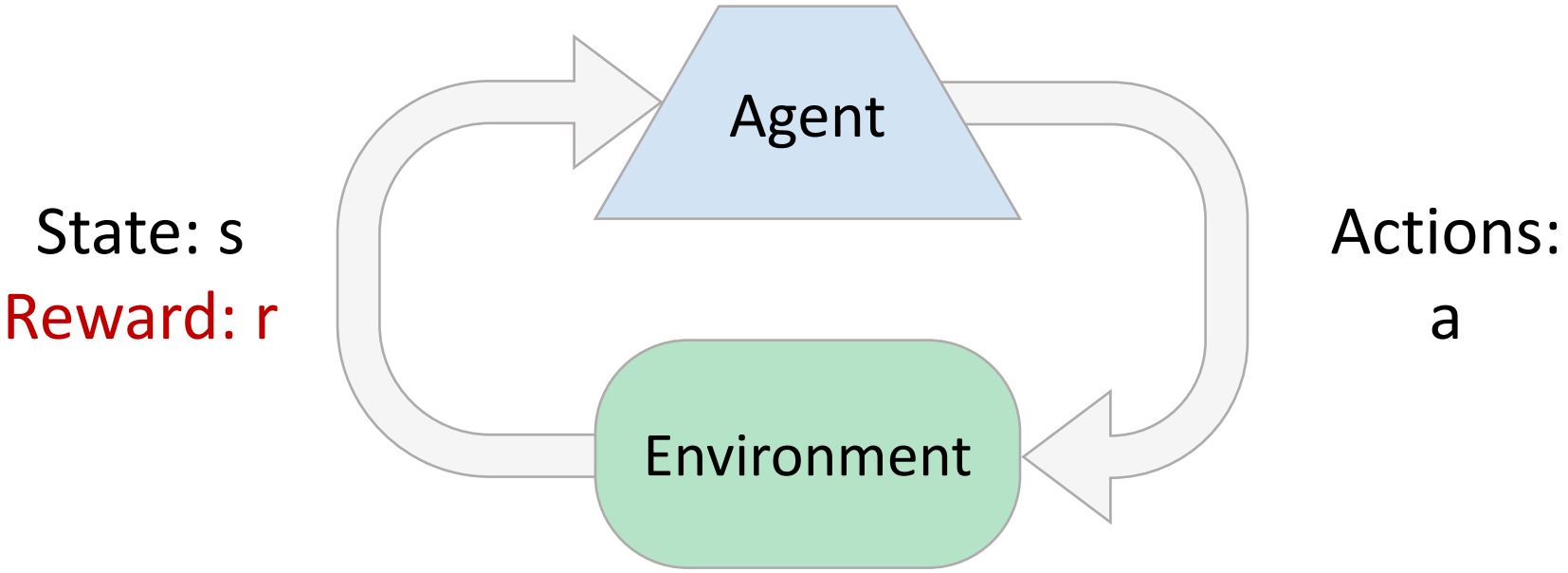
Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

Policy evaluation: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall\, s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

# Reinforcement Learning




Basic idea:

- All learning is based on observed samples of rewards and next states!
- Receive feedback in the form of rewards
- Must (learn to) act so as to maximize expected rewards

# Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

# Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

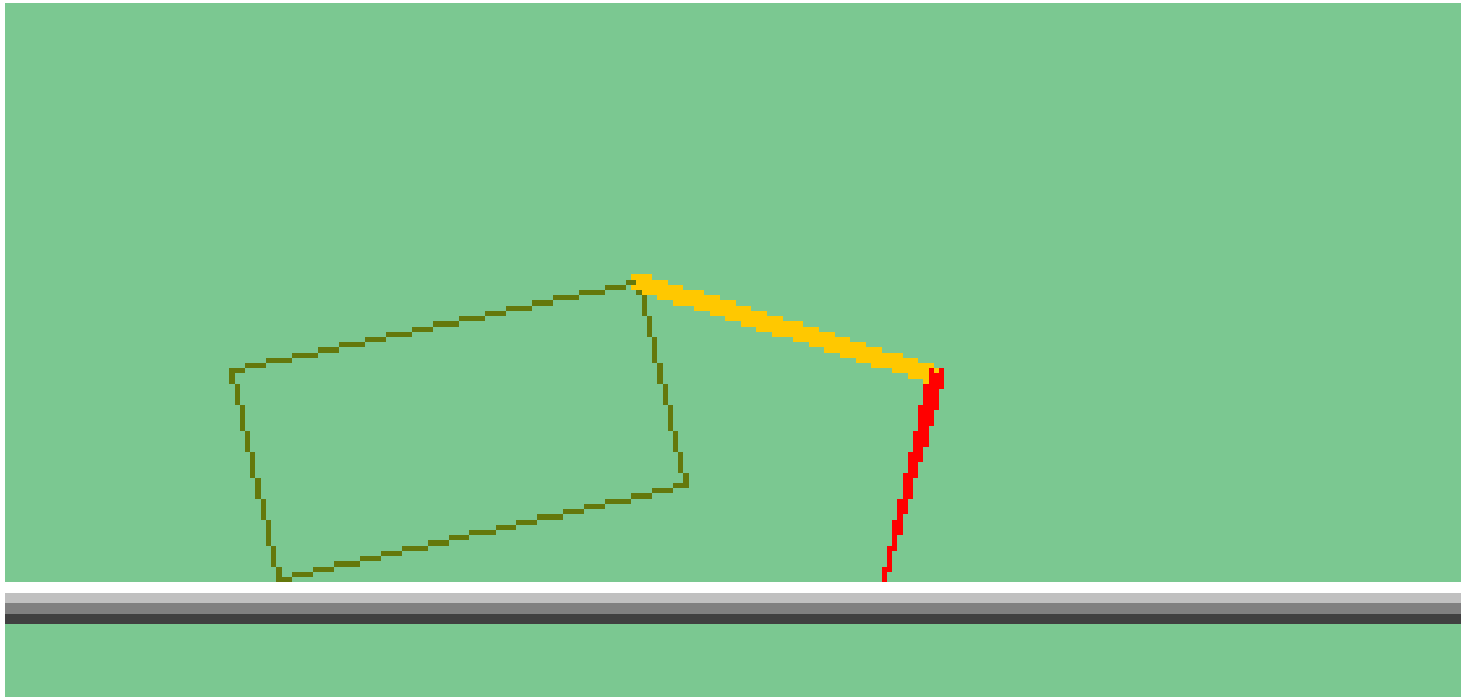
[Video: AIBO WALK – finished]

# Example: Toddler Robot

[Tedrake, Zhang and Seung, 2005]

[Video: TODDLER – 40s]

# The Crawler!



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

# 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')

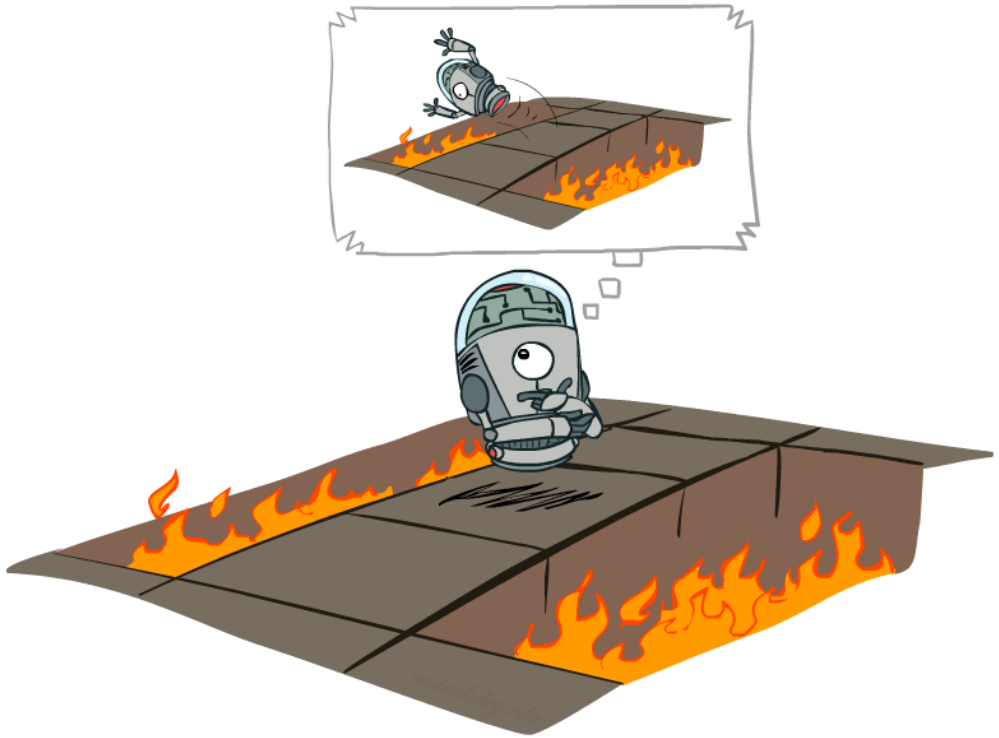
## Still looking for a policy $\pi(s)$




## 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
(Known MDP)

Online Learning
(Unknown MDP)

# Overview: MDPs and Reinforcement Learning

Known MDP: Offline Solution

Value Iteration / Policy Iteration

Unknown MDP: Online Learning

Model-Based

Estimate MDP T(s,a,s') and R(s,a,s') from samples of environment

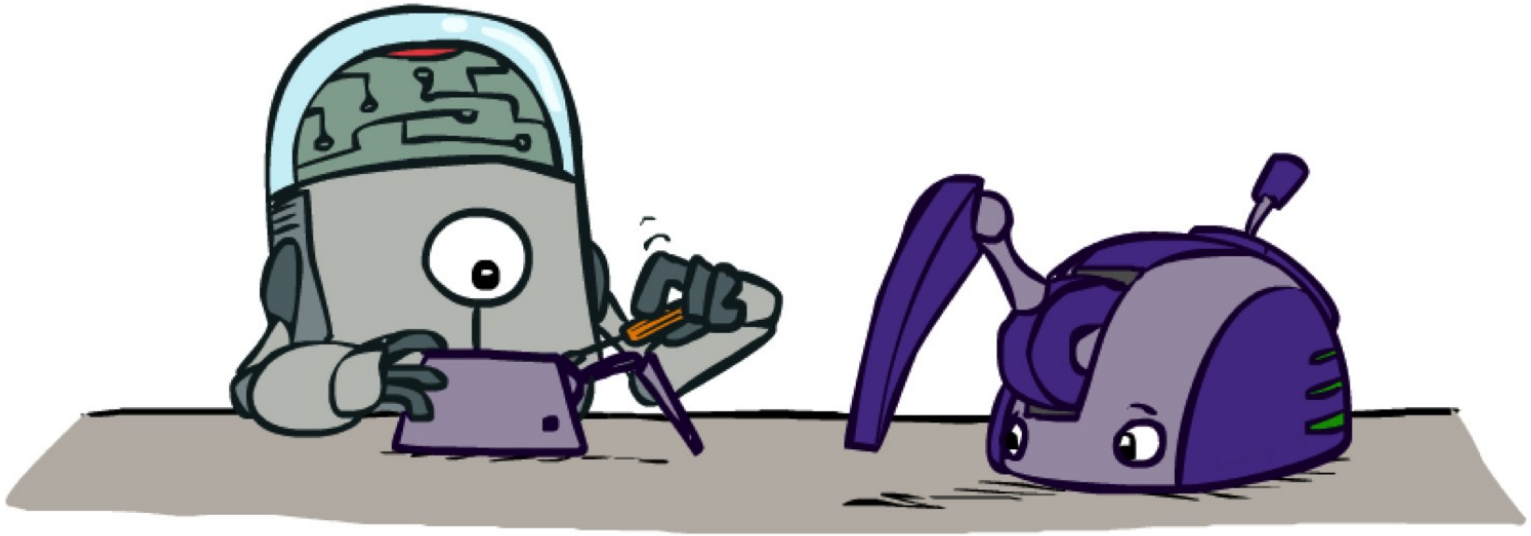
Model-Free

Passive Reinforcement Learning

- Direct Evaluation (simple)
- TD Learning

Active Reinforcement Learning

- Q-Learning



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 $\hat{T}(s, a, s')$
- Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s’)

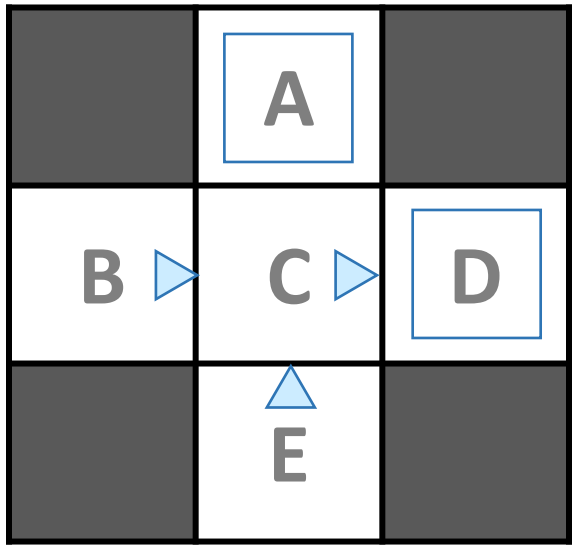
## Step 2: Solve the learned MDP

- For example, use value iteration, as before

# Example: Model-Based Learning

Episode: a sequence of states actions and rewards sampled from the environment

## 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

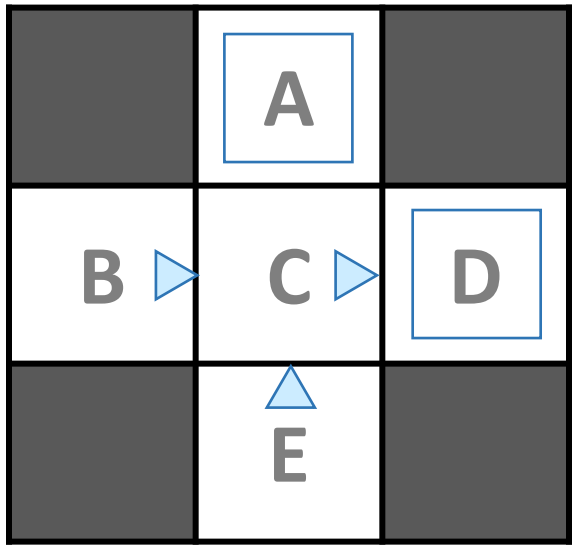
## Learned Model

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

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

# 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 15-281 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"**

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

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

**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.

# Overview: MDPs and Reinforcement Learning

Known MDP: Offline Solution

Value Iteration / Policy Iteration

Unknown MDP: Online Learning

Model-Based

Estimate MDP T(s,a,s') and R(s,a,s') from samples of environment

Model-Free

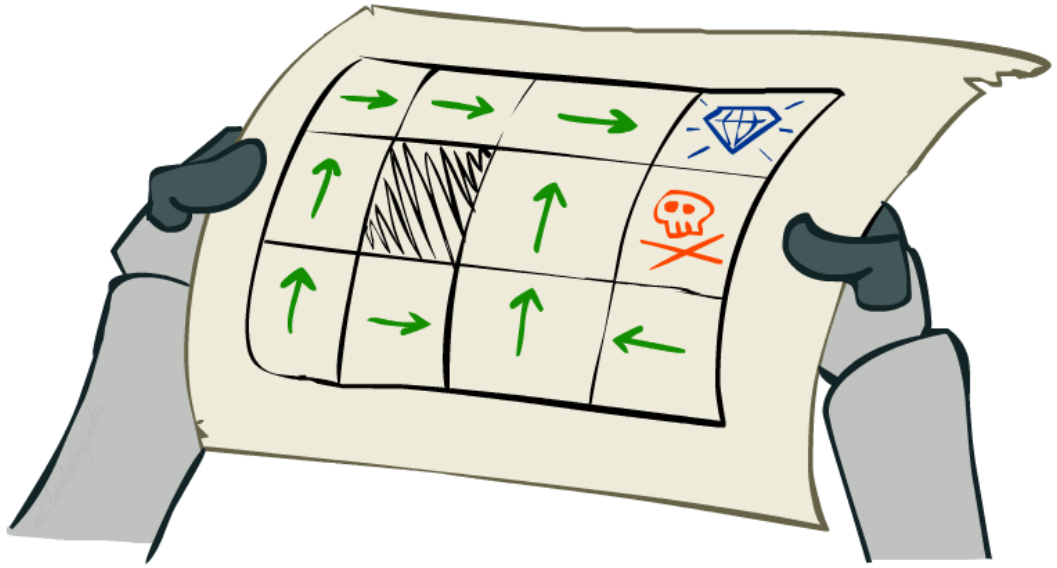
Passive Reinforcement Learning

- Direct Evaluation (simple)
- TD Learning

Active Reinforcement Learning

- Q-Learning

Online Learning



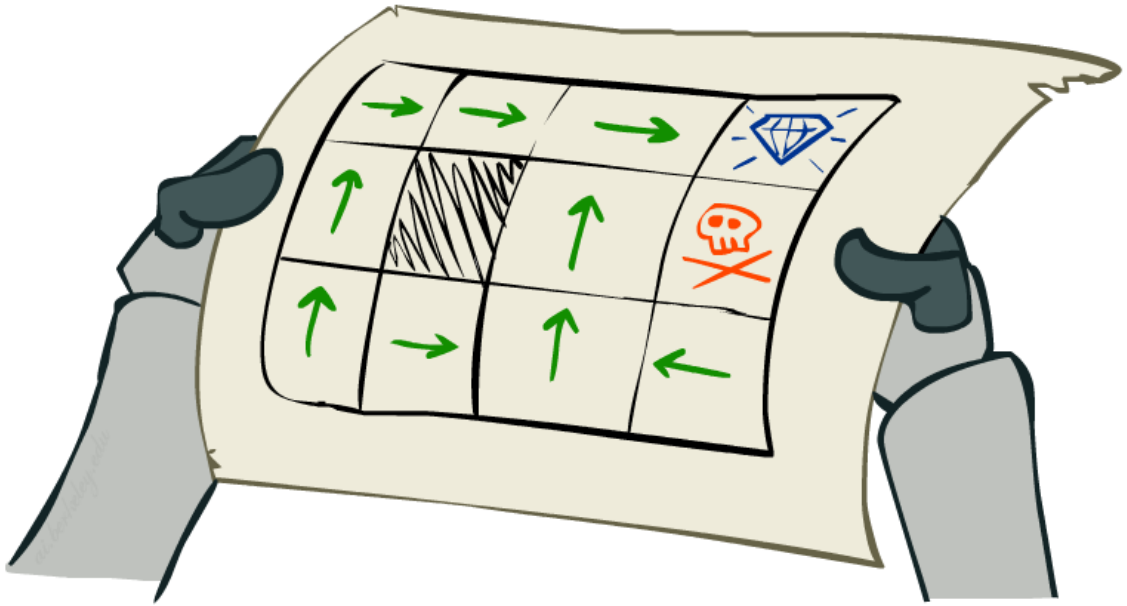
# Model-free Learning

# 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

# Simple Passive Learning: Direct Evaluation

Goal: Compute values for each state under $\pi$

Idea: Average together observed sample values

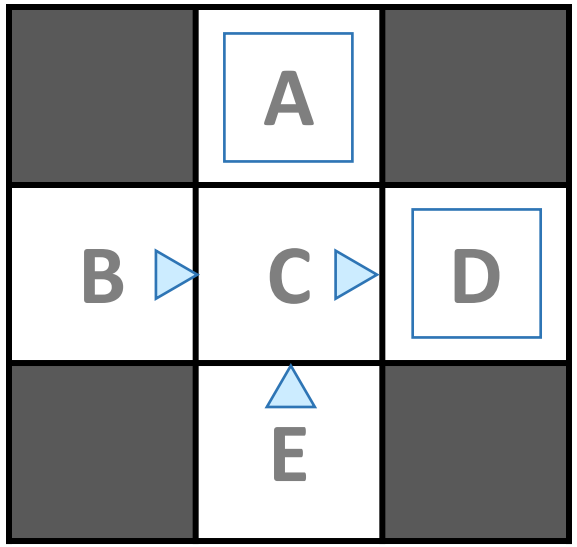
- Act according to $\pi$
- Every time you visit a state, write down what the sum of discounted rewards turned out to be
- Average those samples

This is called direct evaluation

| Pieces Available | Take 1 | Take 2 |
|---|---|---|
| 2 | 0 | **100** |
| 3 | 2 | 0 |
| 4 | **75** | 2 |
| 5 | 4 | **68** |
| 6 | 5 | 6 |
| 7 | **60** | 5 |

# 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

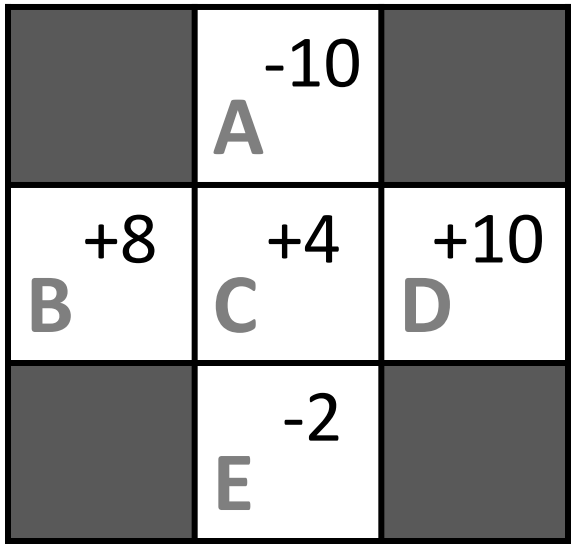
**Episode 3**

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

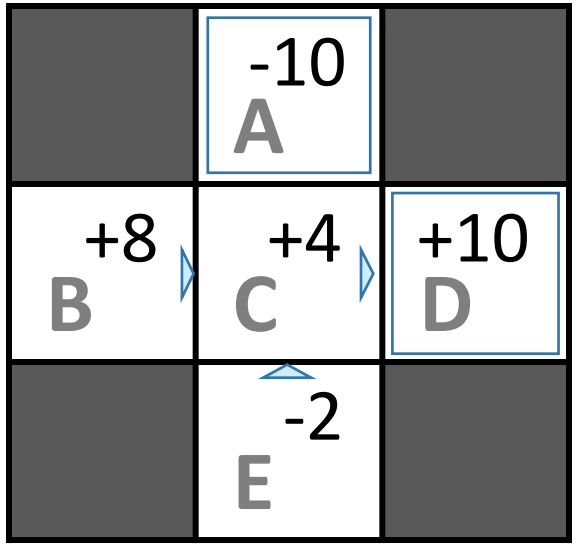
## What's good about direct evaluation?

- 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



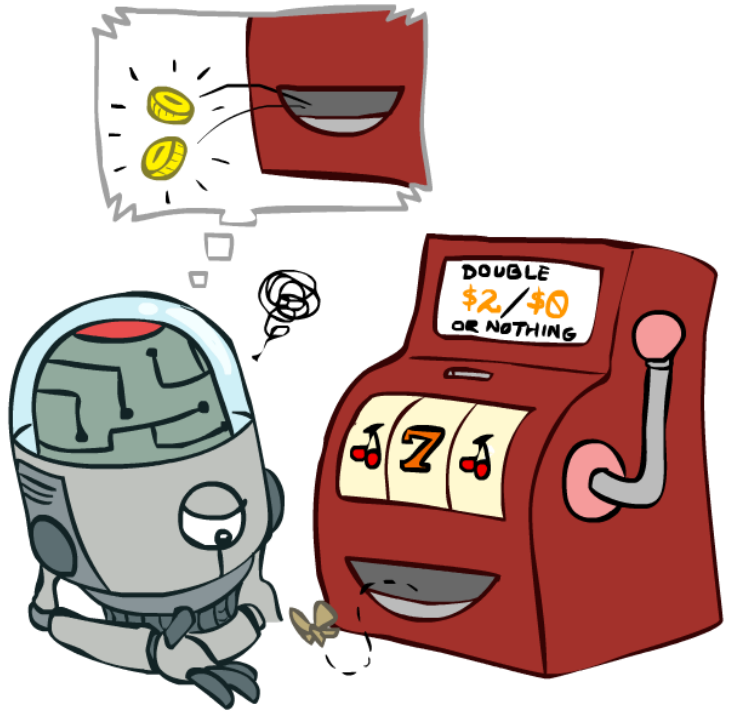

*If B and E both go to C under this policy, how can their values be different?*

Online Learning

Model-free Learning

# Passive Reinforcement Learning

## Temporal Difference Learning



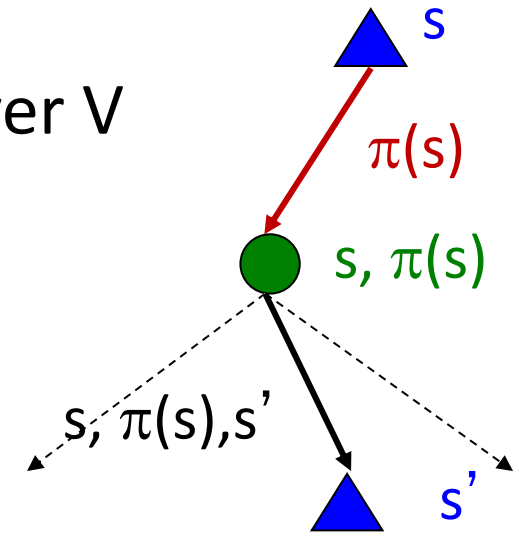
# Why Not Use Policy Evaluation?

## 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?

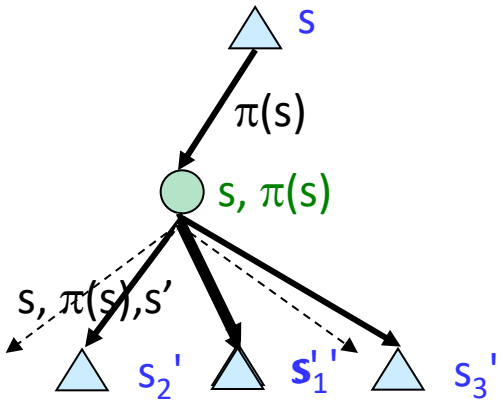
We want to improve our estimate of V by computing these averages:

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

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

$$sample_1 = r_1 + \gamma V_k^{\pi}(s_1')$$
$$sample_2 = r_2 + \gamma V_k^{\pi}(s_2')$$
$$\ldots$$
$$sample_n = r_n + \gamma V_k^{\pi}(s_n')$$

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

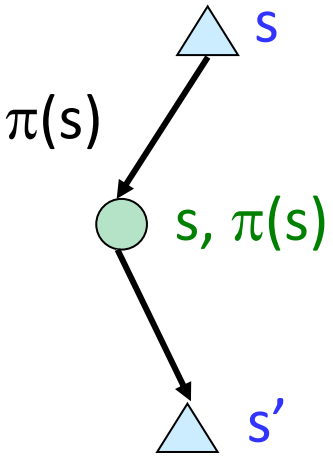



*Almost! But we can't rewind time to get sample after sample from state s.*

# 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

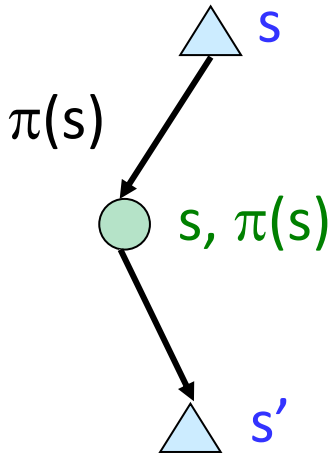
Sample of V(s): $sample = r + \gamma V_k^{\pi}(s')$

Update to V(s):

# 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

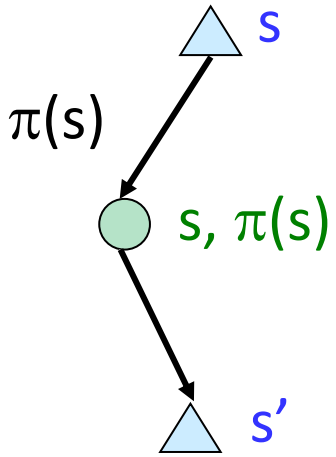
Sample of V(s): $sample = r + \gamma V_k^{\pi}(s')$

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

# 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): $sample = r + \gamma V_k^{\pi}(s')$

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

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