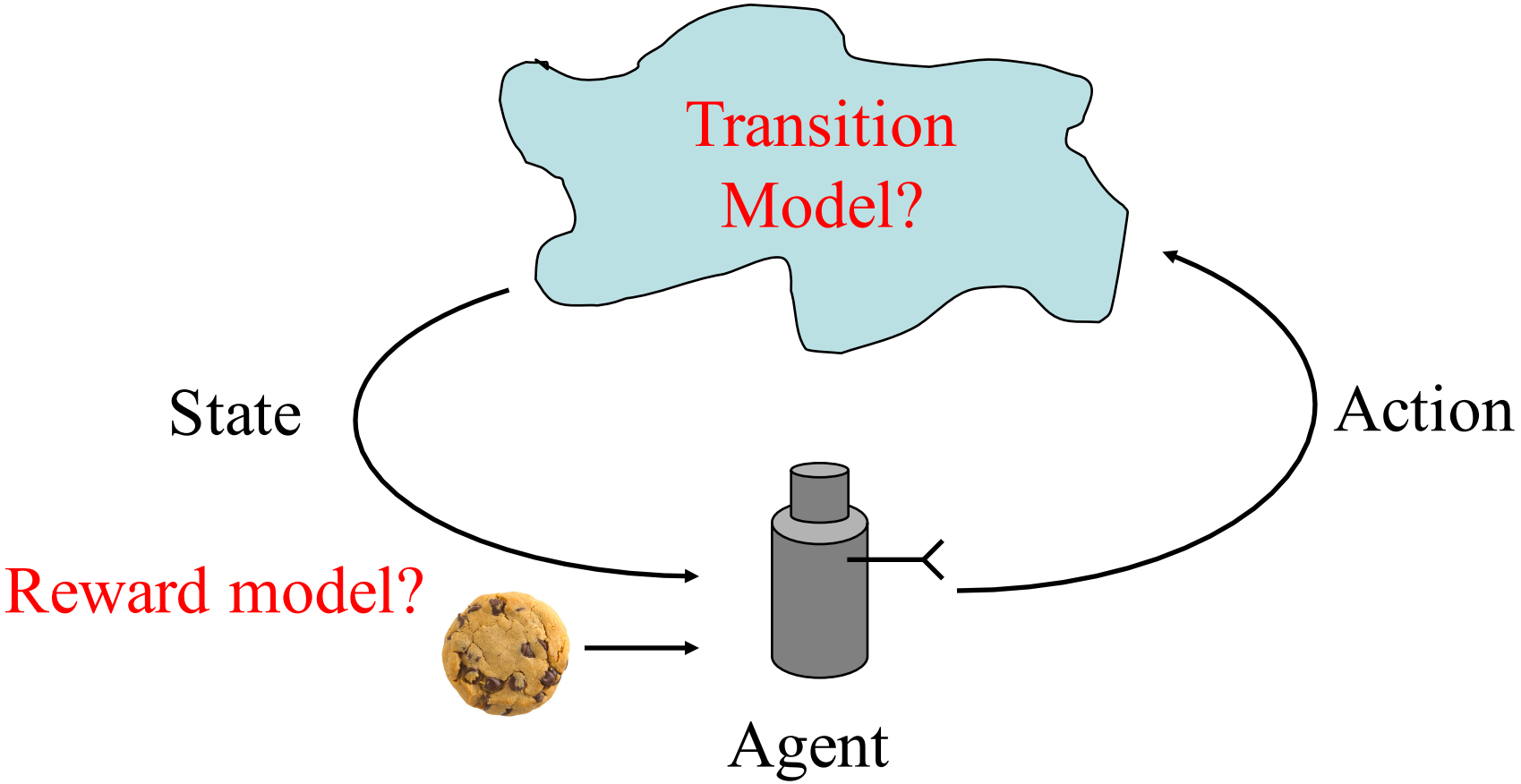
CMU 15-781

Lecture 12:
Reinforcement Learning

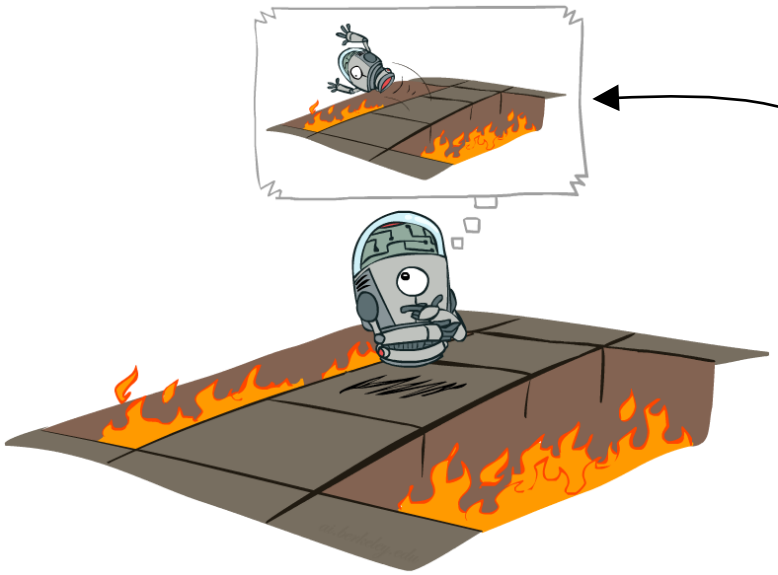
Teacher:
Gianni A. Di Caro

Reinforcement Learning



Goal: Maximize expected sum of future rewards

MDP Planning vs Reinforcement Learning



Don't have a **simulator**! Have to actually learn what happens if take an action in a state

Drawings by Ketrina Yim

Reinforcement learning problem

- ✓ The agents can "sense" the environment (it knows the state) and has goals
- ✓ Learning from interaction with the environment

- **Trial and error search**
- (***Delayed***) **Rewards** (Advisory signals ≠ errors signals)
- **What actions to take?**
  → Exploration- exploitation dilemma

Passive Reinforcement Learning

- Before figuring out how to act, let's first just try to figure out how good a particular policy is
- **Passive learning:** agent's policy is fixed (i.e., in state s it always execute action $\pi(s)$) and the task is to policy's value → Learn state values ($V(s)$) or state-action values ($Q(s,a)$)

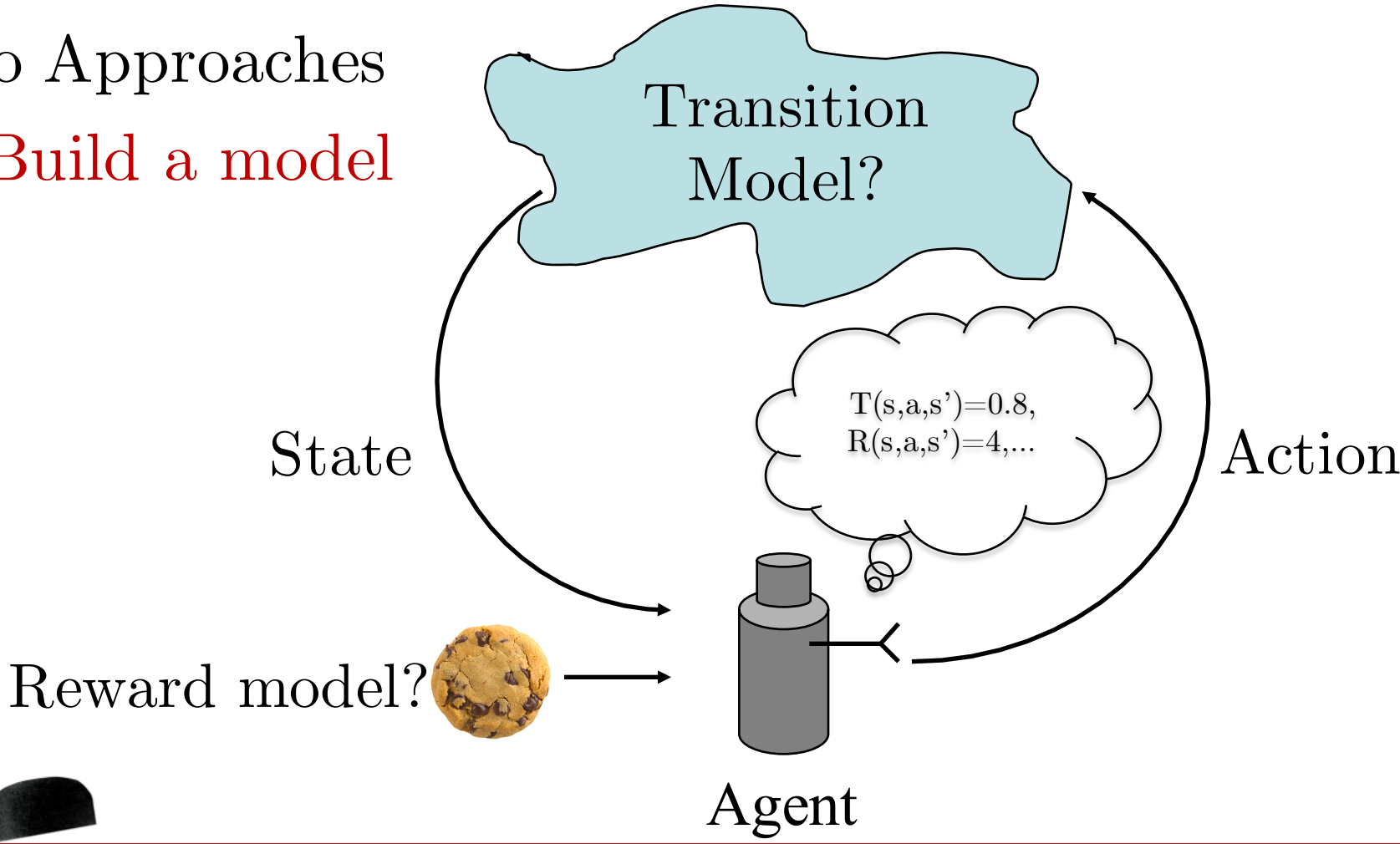
Policy evaluation in MDPs ~ Passive RL

(T,R) Model	~~(T,R) Model~~
Bellman eqs.	Learning

Passive Reinforcement Learning

Two Approaches

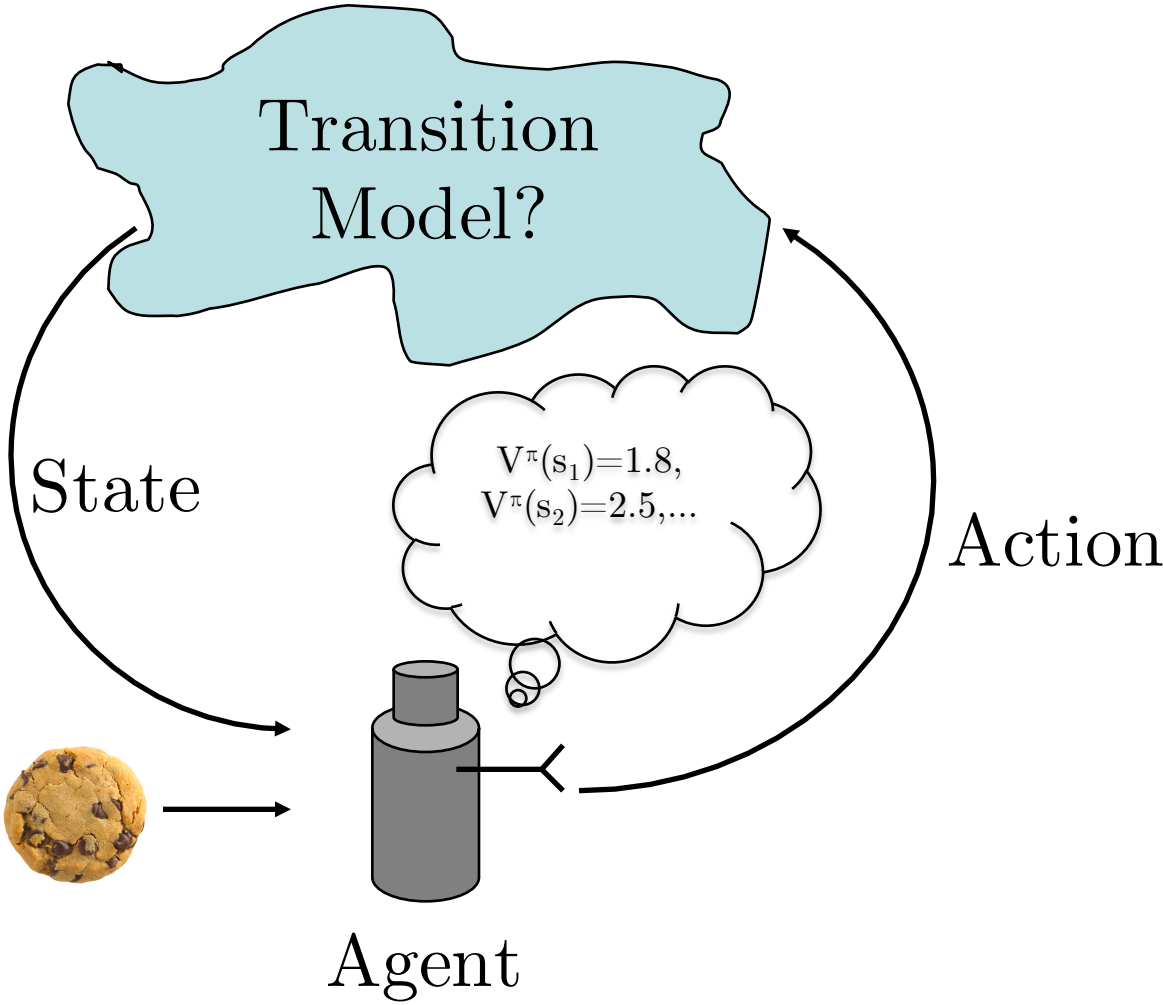
1. Build a model



Passive Reinforcement Learning

Two Approaches:

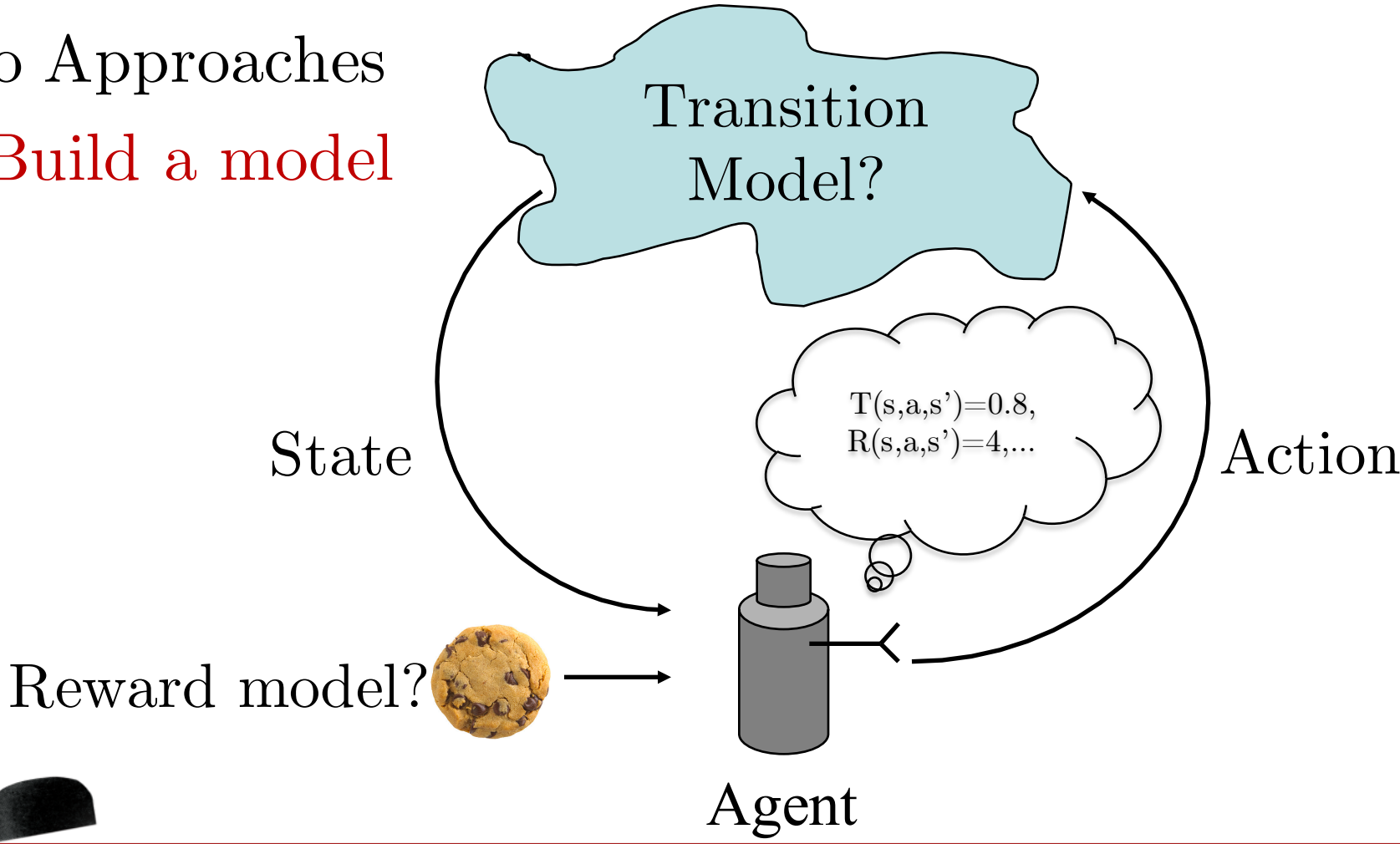
1. Build a model
2. Model-free: directly estimate V^{π}



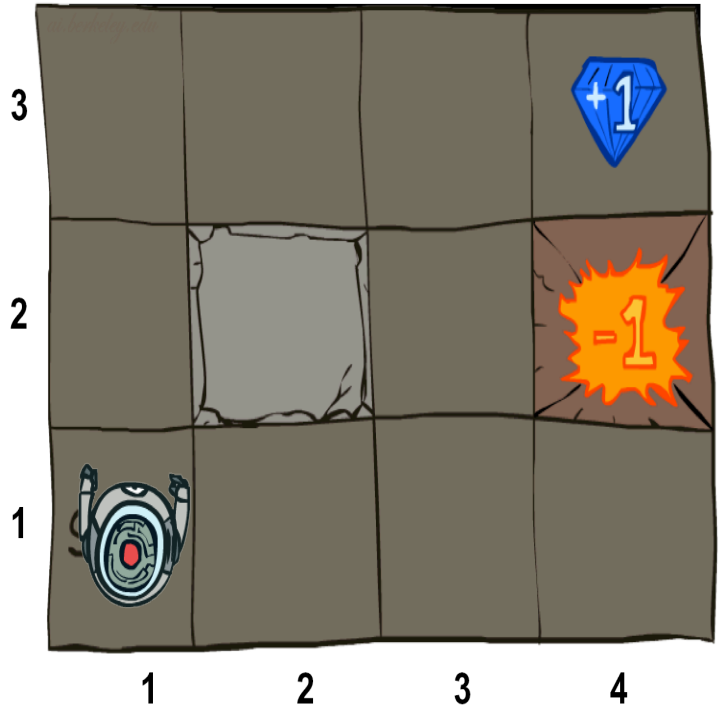
Passive Reinforcement Learning

Two Approaches

1. Build a model



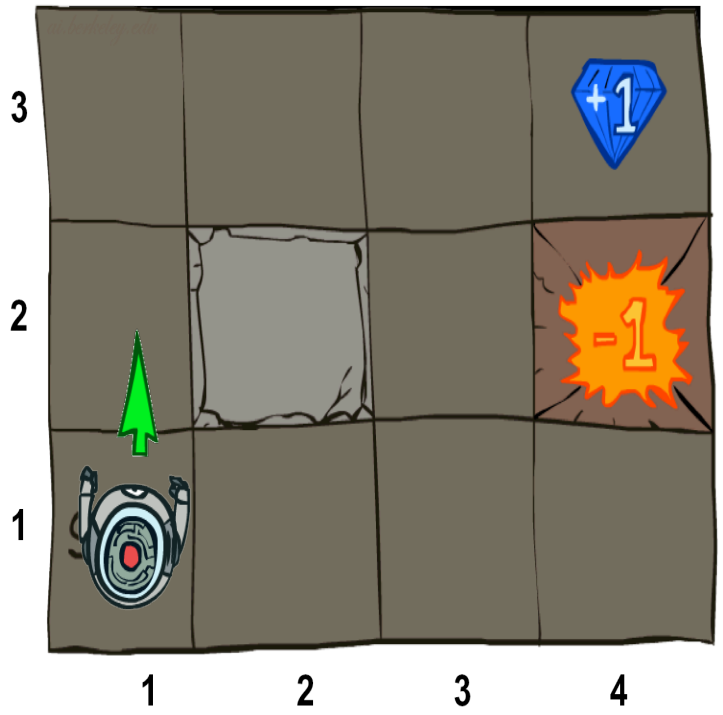
Start at (1,1)



Adaption of drawing by Ketrina Yim

Start at (1,1)

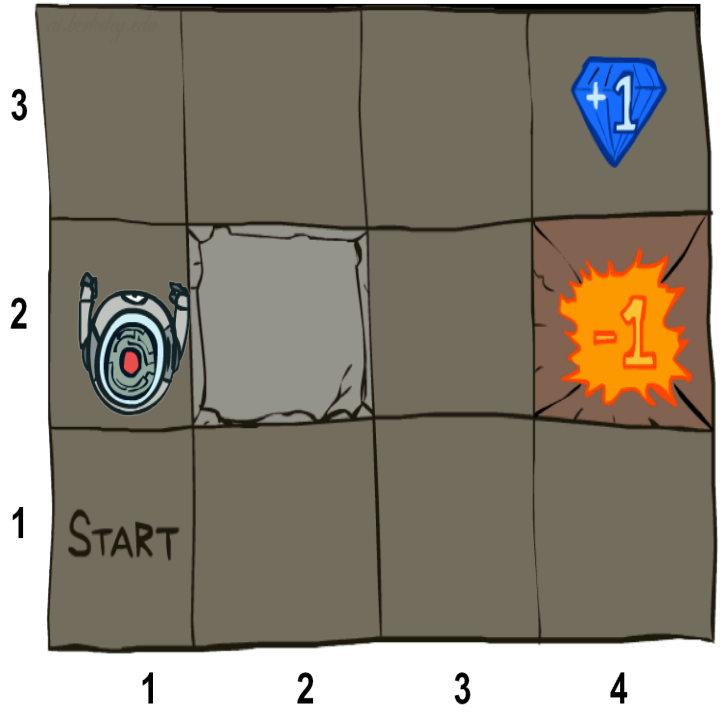
s=(1,1) action= tup *try* up



Adaption of drawing by Ketrina Yim

Start at (1,1)

s=(1,1) action= tup, s'=(1,2), r = -.01

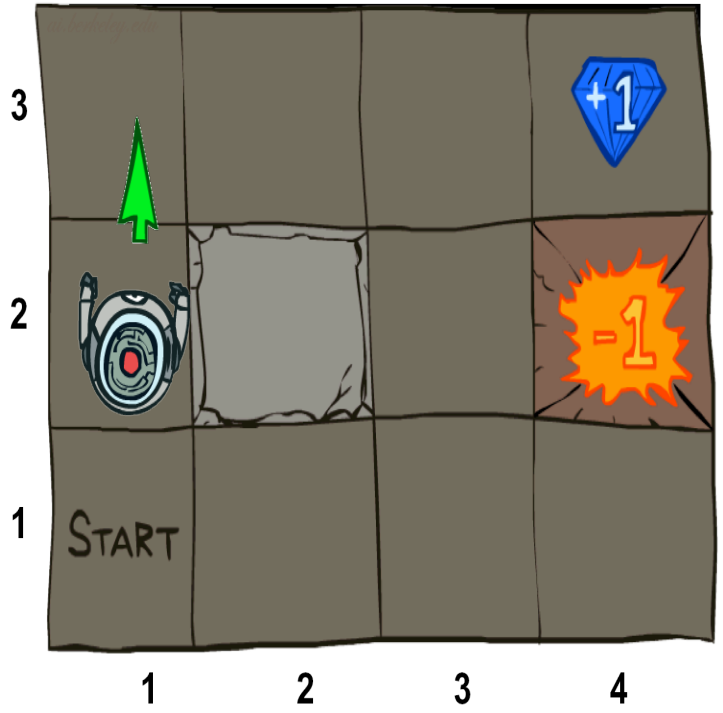


Adaption of drawing by Ketrina Yim

Start at (1,1)

s=(1,1) action= tup, s'=(1,2), r = -.01

s=(1,2) action= tup

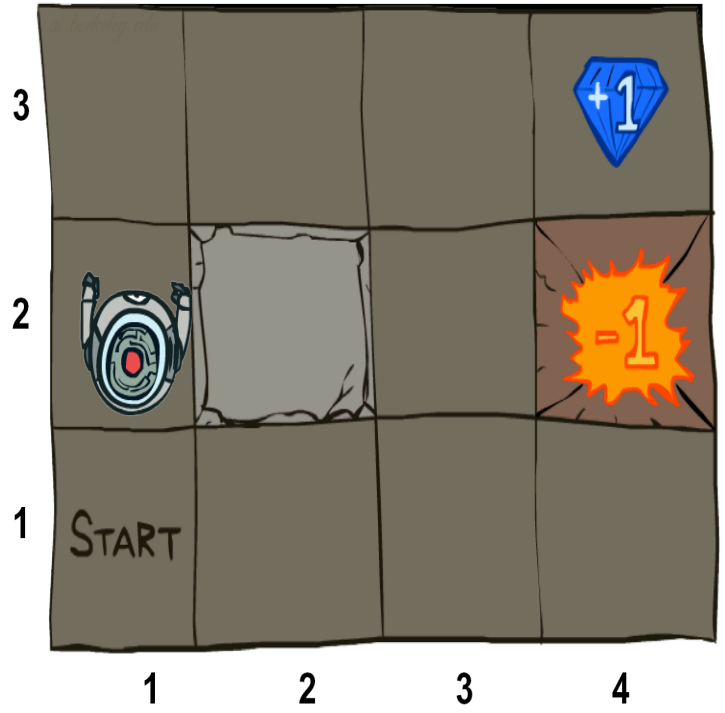


Adaption of drawing by Ketrina Yim

Start at (1,1)

s=(1,1) action= tup, s'=(1,2), r = -.01

s=(1,2) action= tup, s'=(1,2), r = -.01



Start at (1,1)

s=(1,1) action= tup, s'=(1,2), r = -.01

s=(1,2) action= tup, s'=(1,2), r = -.01

s=(1,2) action=tup, s'=(1,3), r = -.01

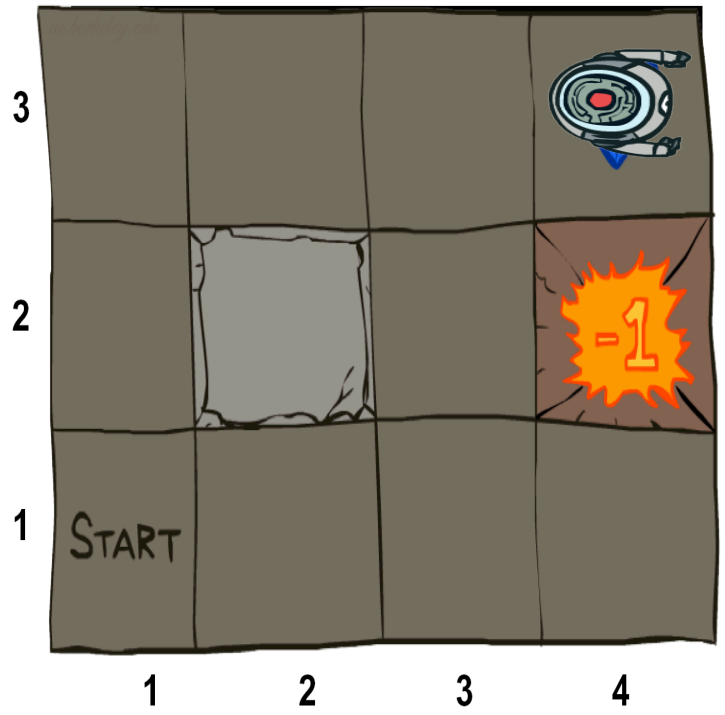
s=(1,3) action=tright, s'=(2,3), r = -.01

s=(2,3) action=tright, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(3,2), r = -.01

s=(3,2) action=tup, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(4,3), r = 1



Start at (1,1)

s=(1,1) action= tup, s'=(1,2), r = -.01

s=(1,2) action= tup, s'=(1,2), r = -.01

s=(1,2) action=tup, s'=(1,3), r = -.01

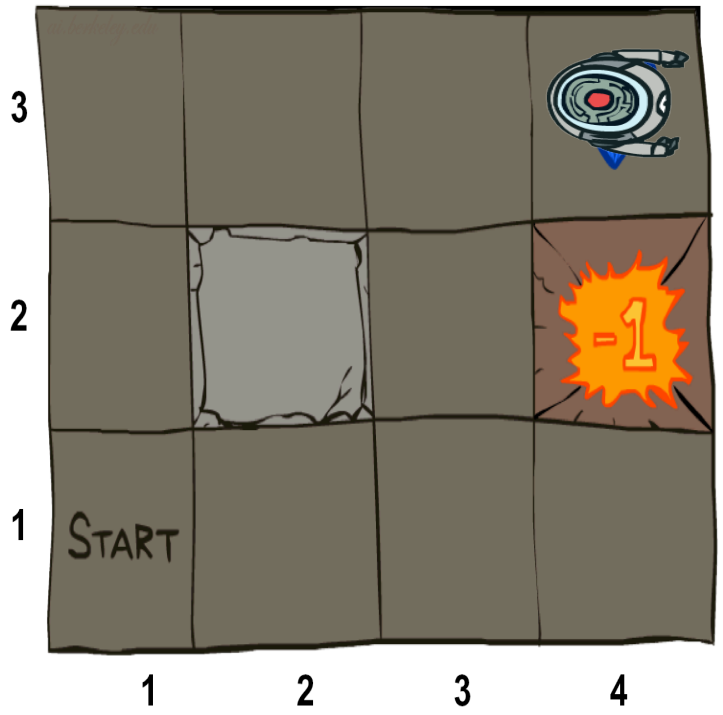
s=(1,3) action=tright, s'=(2,3), r = -.01

s=(2,3) action=tright, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(3,2), r = -.01

s=(3,2) action=tup, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(4,3), r = 1



The gathered experience can be used to estimate MDP's T and R models

Start at (1,1)

s=(1,1) action= tup, s'=(1,2), r = -.01

s=(1,2) action= tup, s'=(1,2), r = -.01

s=(1,2) action=tup, s'=(1,3), r = -.01

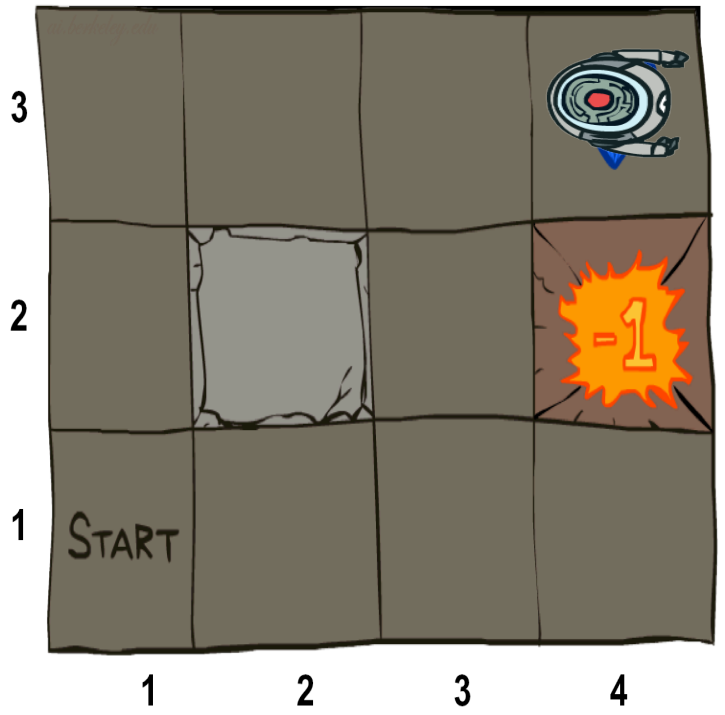
s=(1,3) action=tright, s'=(2,3), r = -.01

s=(2,3) action=tright, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(3,2), r = -.01

s=(3,2) action=tup, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(4,3), r = 1



The gathered experience can be used to estimate MDP's T and R models

Estimate of T(<1,2>,tup,<1,3>) = 1/2

Adaption of drawing by Ketrina Yim

Model-Based Passive Reinforcement Learning

1. Follow policy π
2. Estimate MDP model parameters T and R given observed transitions and rewards
    1. If finite set of states and actions, can just count and average counts
3. Use estimated MDP to do policy evaluation of π

Does this give us all the parameters for an MDP?

Start at (1,1)

s=(1,1) action= tup, s'=(1,2), r = -.01

s=(1,2) action= tup, s'=(1,2), r = -.01

s=(1,2) action=tup, s'=(1,3), r = -.01

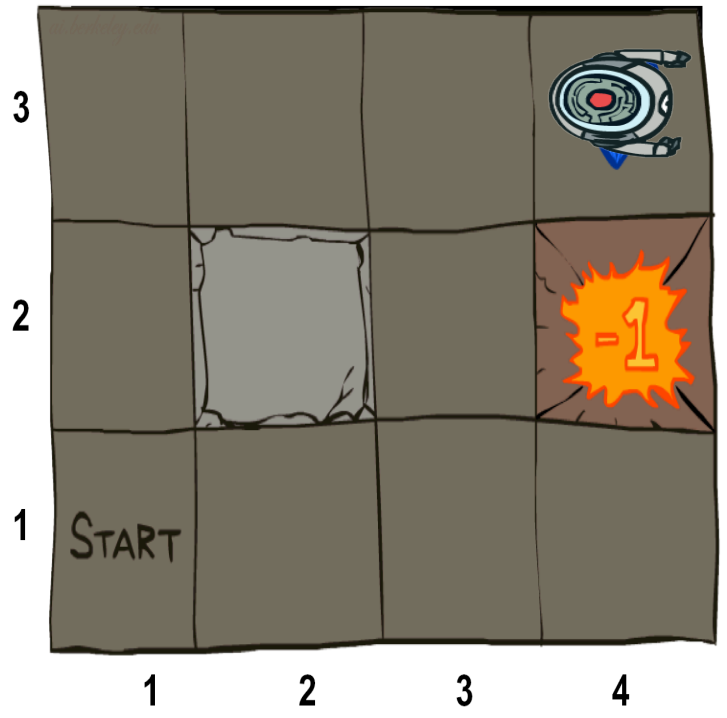
s=(1,3) action=tright, s'=(2,3), r = -.01

s=(2,3) action=tright, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(3,2), r = -.01

s=(3,2) action=tup, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(4,3), r = 1



Estimate of T(<1,2>,tright,<1,3>)? No idea! Never tried this action...

Adaption of drawing by Ketrina Yim

Does This Give Us All the Parameters for a MDP?
No.
But Does That Matter for Computing Policy Value?
No.
Have all Parameters We Need!
(After We Visit All States at Least Once)

- Follow policy π
- Estimate MDP model parameters given observed transitions and rewards
  - If finite set of states and actions, can just count and average counts
- Use estimated MDP to do policy evaluation of π

Start at (1,1)

s=(1,1) action= tup, s'=(1,2), r = -.01

s=(1,2) action= tup, s'=(1,2), r = -.01

s=(1,2) action=tup, s'=(1,3), r = -.01

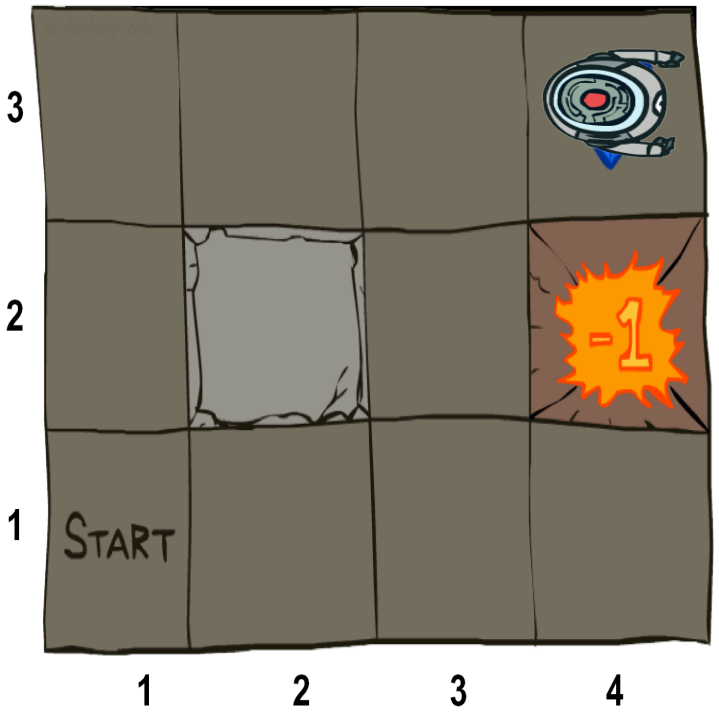
s=(1,3) action=tright, s'=(2,3), r = -.01

s=(2,3) action=tright, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(3,2), r = -.01

s=(3,2) action=tup, s'=(3,3), r = -.01

s=(3,3) action=tright, s'=(4,3), r = 1



s=(1,1) action= tup, s'=(2,1), r = -.01

s=(2,1) action= tright, s'=(3,1), r = -.01

s=(3,1) action= tup, s'=(4,1), r = -.01

s=(4,1) action= tleft, s'=(3,1), r = -.01

s=(3,1) action= tup, s'=(3,2), r = -.01

s=(3,2) action= tup, s'=(4,2), r = -1

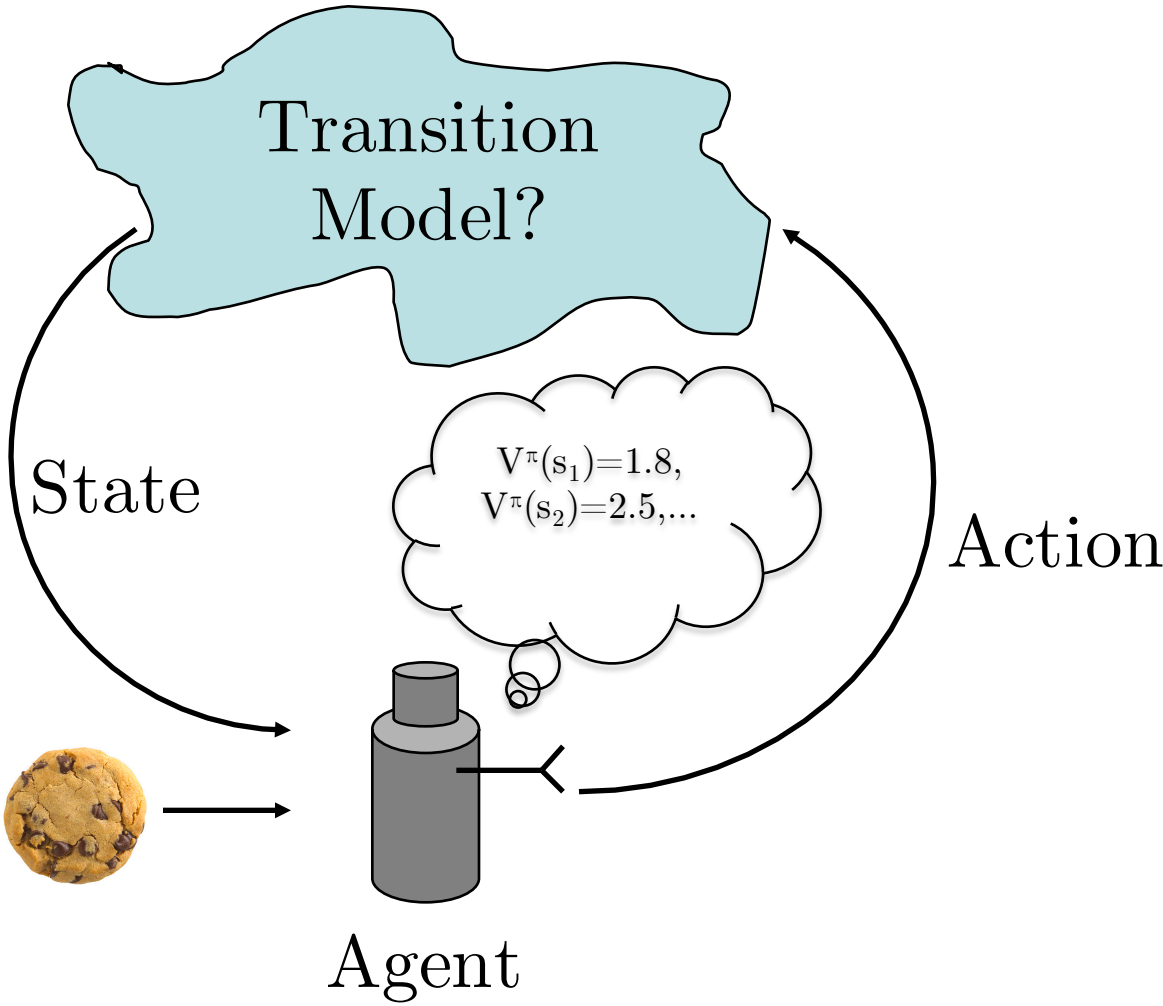
2 episodes of experience in MDP.
Use to estimate MDP parameters & evaluate π
Is the computed policy value likely to be correct?
(1) Yes (2) No (3) Not sure

Adaption of drawing by Ketrina Yim

Passive Reinforcement Learning

Two Approaches:

1. Build a model
2. Model-free: directly estimate V^{π}



Direct utility estimation: Monte Carlo Policy Evaluation

First-visit MC policy evaluation (returns $V \approx v_\pi$)

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

Generate an episode using π

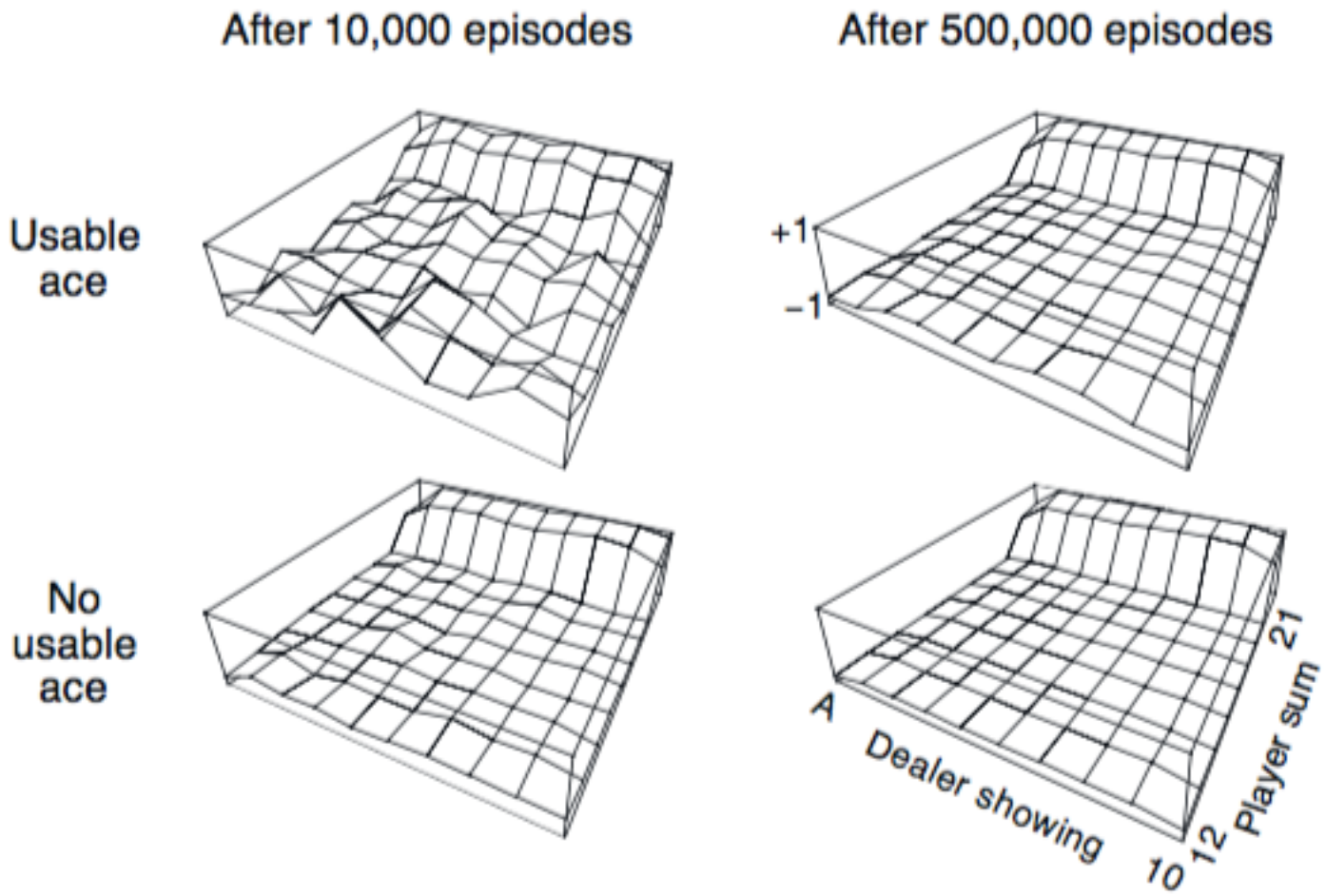
For each state s appearing in the episode:

$G \leftarrow$ return following the first occurrence of s

Append G to $Returns(s)$

$V(s) \leftarrow$ average($Returns(s)$)

Monte Carlo Policy Evaluation



Monte Carlo Policy Evaluation

What information is *missing* in the way we are proceeding performing policy evaluation for our MDP? (it's an MDP, we just do not know its parameters ...)

The recursive Bellman relations among states!

$$V^{\pi}(s) = \sum_{s' \in S} p\Big(s' \mid s, \pi(s)\Big)\Big[R(s, \pi(s), s') + \gamma V^{\pi}(s')\Big] \qquad \forall s \in S$$

We should exploit this structure in the MDP...

Temporal Differences (TD) Policy Evaluation

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
    $S \leftarrow S'$
  until S is terminal

Update at each step!

Sample + Bellman

TD *bootstraps* on available state information

Temporal Difference Learning

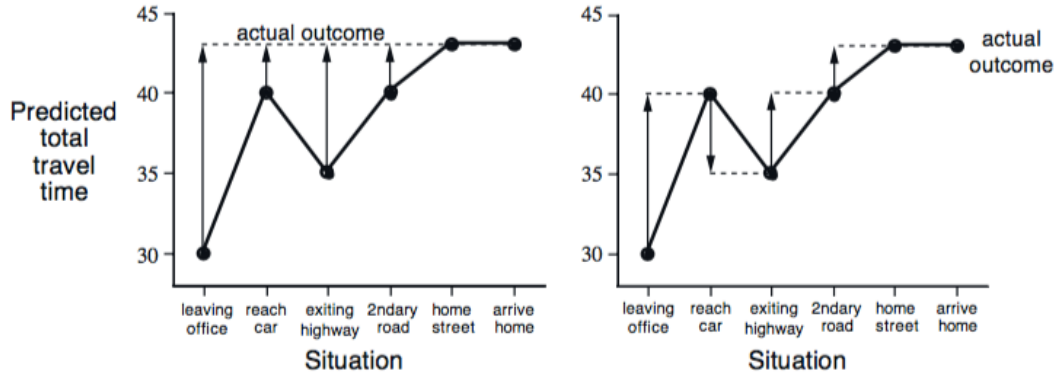
- No explicit model of T or R!
- Estimate V and expectation **through samples**
- Update from **every experience**
  - Update $V(s)$ after each state transition $(s,\ a,\ s',\ r)$
  - Likely outcomes s' will contribute updates more often
- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward sample of V: **running average**

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

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

TD EXAMPLE: GOING HOME

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43



Exponential Moving Avg

- Exponential moving average
  - The running interpolation update:

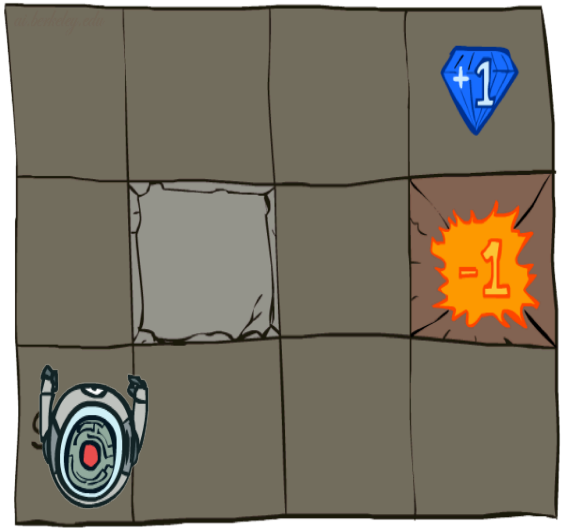
$$\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

  - 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
- Decreasing learning rate (α) can give converging avgs

TD Learning Example

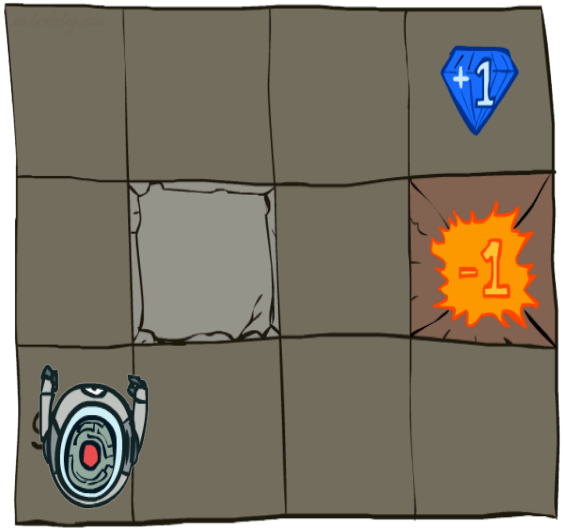


Initialize all $V^{\pi}(s)$ values: $V^{\pi}(s) = 0$

State	$V^{\pi}(s)$
(1,1)	0
(1,2)	0
(1,3)	0
(4,1)	0
(2,1)	0
(2,3)	0
(3,1)	0
(3,2)	0
(3,3)	0

TD Learning Example $\alpha=0.1, \gamma=1$

s=(1,1) action= tup, s'=(1,2), r = -.01



Update $V^{\pi}((1,1))$

$$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

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

sample = -0.01 + V^{π}((1,2)) = -0.01

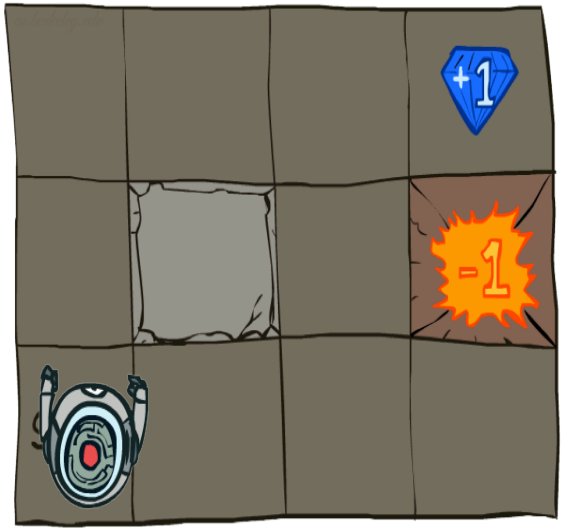
V^{π}((1,1)) = (1-α) V^{π}((1,1)) + α*sample

= .9*0 + 0.1*-0.01 = -0.001

State	$V^{\pi}(s)$
(1,1)	0
(1,2)	0
(1,3)	0
(4,1)	0
(2,1)	0
(2,3)	0
(3,1)	0
(3,2)	0
(3,3)	0

TD Learning Example $\alpha=0.1, \gamma=1$

s=(1,1) action= tup, s'=(1,2), r = -.01



Update $V^{\pi}((1,1))$

$$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

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

sample = -0.01 + V^{π}((1,2)) = -0.01

V^{π}((1,1)) = (1-α) V^{π}((1,1)) + α*sample

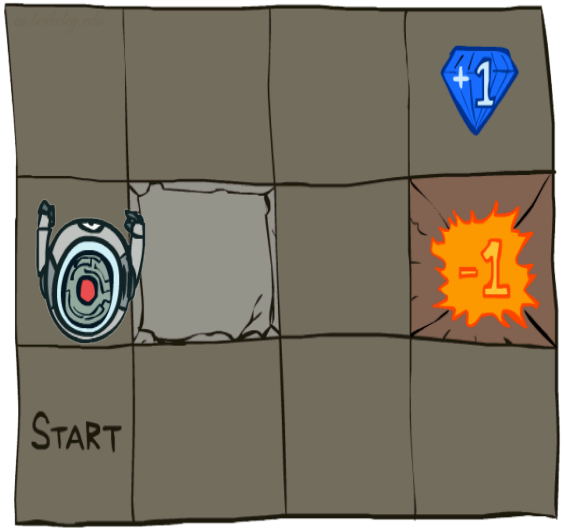
= .9*0 + 0.1*-0.01 = -0.001

State	$V^{\pi}(s)$
(1,1)	-0.001
(1,2)	0
(1,3)	0
(4,1)	0
(2,1)	0
(2,3)	0
(3,1)	0
(3,2)	0
(3,3)	0

TD Learning Example $\alpha=0.1, \gamma=1$

s=(1,1) action= tup, s'=(1,2), r = -.01

s=(1,2) action= tup, s'=(1,2), r = -.01



State	$V^{\pi}(s)$
(1,1)	-0.001
(1,2)	0
(1,3)	0
(4,1)	0
(2,1)	0
(2,3)	0
(3,1)	0
(3,2)	0
(3,3)	0

Problems with Passive Learning

- Agent wants to ultimately learn to *act* to gather high reward in the environment.
- Using a deterministic policy, gives no experience for other actions (not included in the policy)

Active reinforcement learning: the agent decides what action to take with the goal of learning an optimal policy

Active RL: Exploration issues

- How the actions are selected for the purpose of efficient learning?

- Consider acting **randomly** in the world
- Can such experience allow the agent to learn the optimal values and policy?

Recall Model-Based Passive Reinforcement Learning

- **Follow policy π**
- Estimate MDP model parameters given observed transitions and rewards
  - If finite set of states and actions, can just count and average counts
- Use estimated MDP to do policy evaluation of π

Model-Based active RL w/Random Actions

- **Choose actions randomly**
- Estimate MDP model parameters given observed transitions and rewards
  - If finite set of states and actions, can just count and average counts
- Use estimated MDP to compute estimate of optimal values and policy

Will the computed values and policy converge to the true optimal values and policy in the limit of infinite data?

Reachability

- When acting randomly forever, still need to be able to **visit each state and take each action many times**
- Want all states to be reachable from any other state
- Quite mild assumption but doesn't always hold

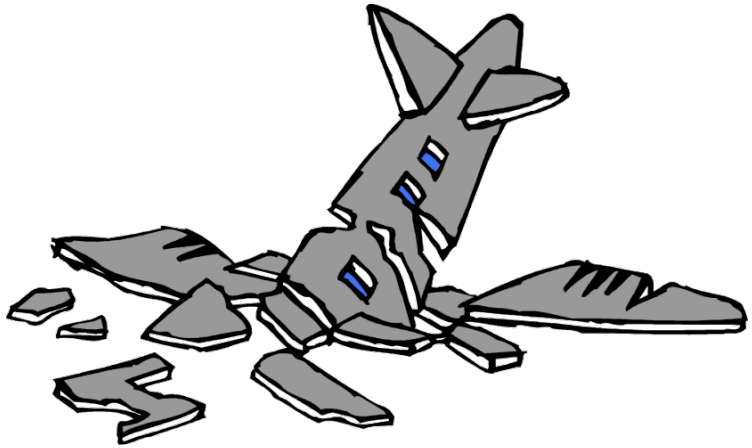


Image source: http://ancient-heritage.blogspot.com/2014/05/crash-course-on-flying-in-face-of-logic.html

On-Policy / Off-policy RL learning

An Active RL agent can have two (different) policies:

- **Behavior policy** → Used to generate actions (↔ Interact with environment to gather sample data)
- **Learning policy** → Target action policy to learn (the "good"/optimal policy the agent eventually aims to discover through interaction)
- If Behavior policy = Learning policy → **On-policy learning**
- If Behavior policy ≠ Learning policy → **Off-policy learning**

On-Policy / Off-policy

- **On-policy learning:** the behavior policy used to generate samples is *the same* as the target policy → Agent learns the value of the policy being used, including exploration actions.
  - The used policy is usually "soft" and non-deterministic, to ensure there is always exploration.
- **Off-policy learning:** the behavior and the target policy are different. The target policy is learned regardless (independently) of the actions chosen for exploring the environment. The agent follows a policy but learns the value of a different policy.
  → Easier converge proofs and analysis
  - In ***Q-learning***, the action-value function $Q(s,a)$ is learned using a greedy action policy, such that it directly estimates Q^*

Example of On-Policy Learning

- To gather sample data, the agent applies an *ε-greedy behavior policy* π_ε , such that with a probability (1-ε)=0.9 the **optimal** action (as currently estimated from Q) is chosen, and with probability ε=0.1 a **random** action is chosen.
- The actions defined by the behavior policy are also used to update the target policy (i.e., to update $Q(s,a)$ estimates)
- Since also random actions are being used, the target policy will learn the value of the policy that includes *both* optimal and random actions, selected with a probability ε
- In SARSA (next slide), each $Q(s,a)$ is updated with a sample Q value that can refer to a random action (i.e, the "value" of action a in state s is the exponential average of the observed rewards associated to both optimal and random actions)

SARSA: On-Policy TD State-Action Learning

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Update Q estimate with the sample data generated according to the behavior policy

or keep acting forever,
or termination criterion

Q-Learning: Off-Policy TD State-Action Learning

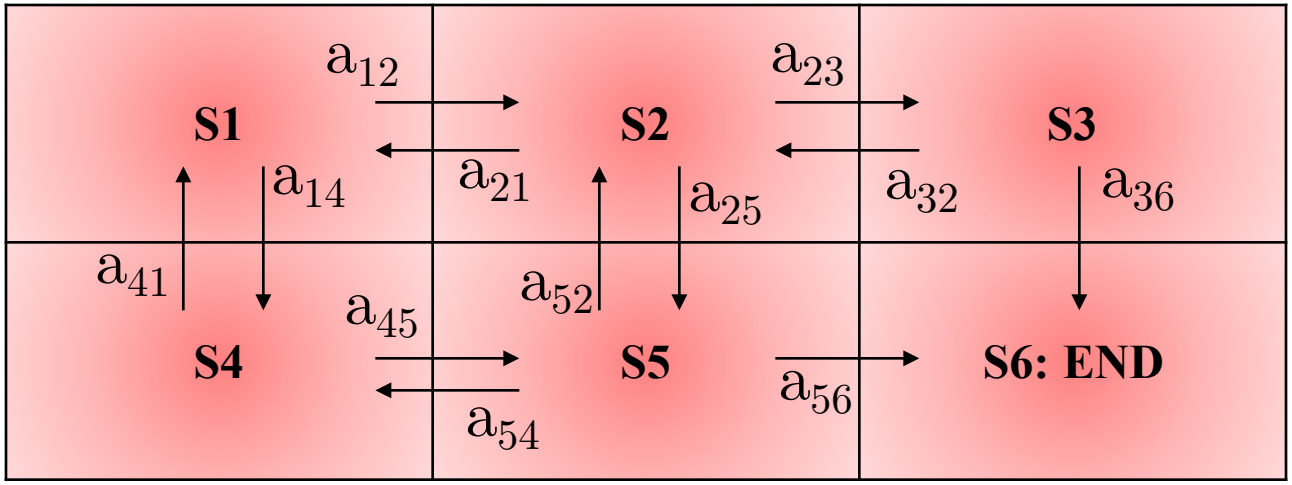
Any exploration policy

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal

or keep acting forever,
or termination criterion

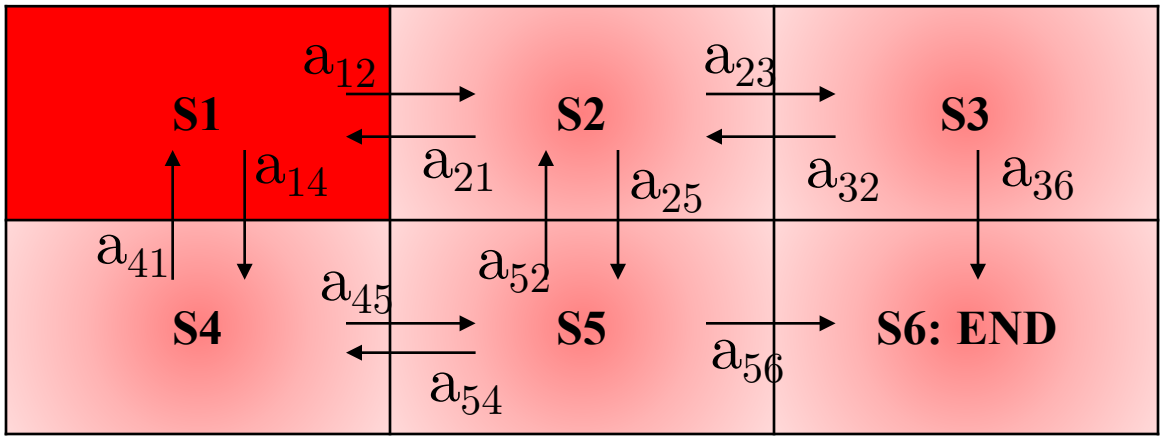
Update Q estimate with the sample data but according to a greedy policy for action selection (take the max) ≠ from behavior policy

Q-Learning Example



- 6 states, S1,..S6
- 12 actions a_{ij} for state transitions, deterministic
- R=100 in S6 (terminal state), R=0 otherwise
- γ=0.5, $\alpha = 1$
- Random behavior policy

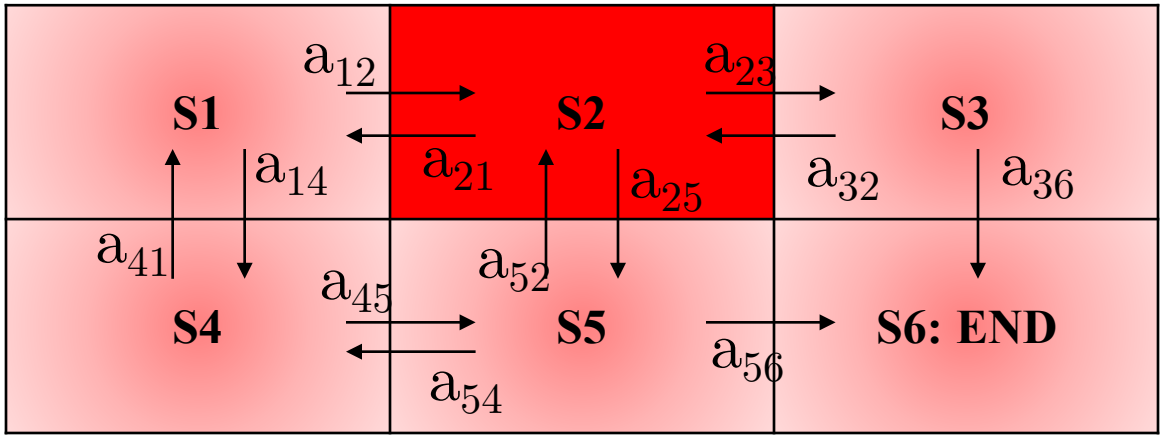
Initial state



$Q(S1, a_{12})$	0
$Q(S1, a_{14})$	0
$Q(S2, a_{21})$	0
$Q(S2, a_{25})$	0
$Q(S2, a_{23})$	0
$Q(S3, a_{32})$	0
$Q(S3, a_{36})$	0
$Q(S4, a_{41})$	0
$Q(S4, a_{45})$	0
$Q(S5, a_{52})$	0
$Q(S5, a_{54})$	0
$Q(S5, a_{56})$	0

- Available actions: a_{12}, a_{14}
- Choose a_{12}

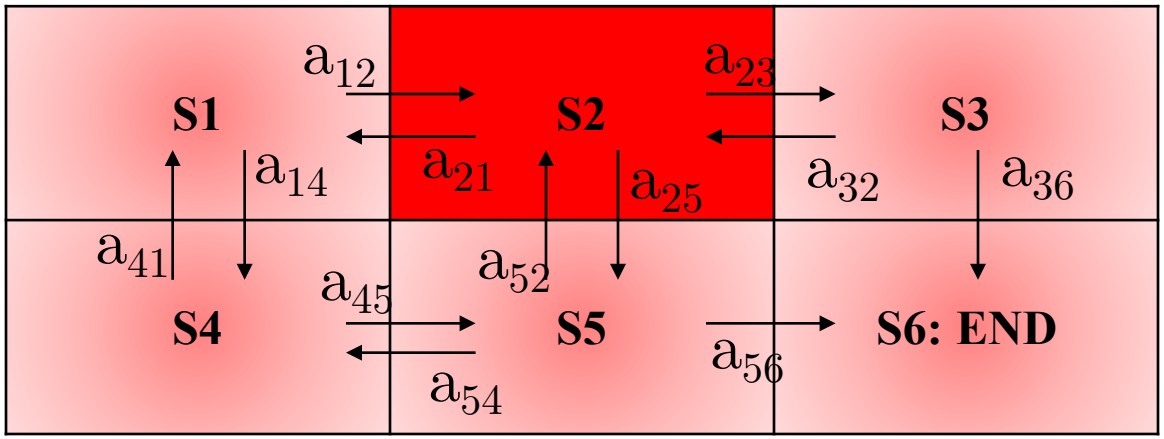
New state, Update



$Q(S1, a_{12})$	0
$Q(S1, a_{14})$	0
$Q(S2, a_{21})$	0
$Q(S2, a_{25})$	0
$Q(S2, a_{23})$	0
$Q(S3, a_{32})$	0
$Q(S3, a_{36})$	0
$Q(S4, a_{41})$	0
$Q(S4, a_{45})$	0
$Q(S5, a_{52})$	0
$Q(S5, a_{54})$	0
$Q(S5, a_{56})$	0

- Reward: 0; New state: S2
- Update state-action:
  $Q(S1, a_{12}) \leftarrow Q(S1, a_{12}) +$
  $\big(0 + 0.5 \cdot \max\{Q(S2, a_{21}), Q(S2, a_{23}), Q(S2, a_{25})\} - Q(S1, a_{12})\big) = 0$

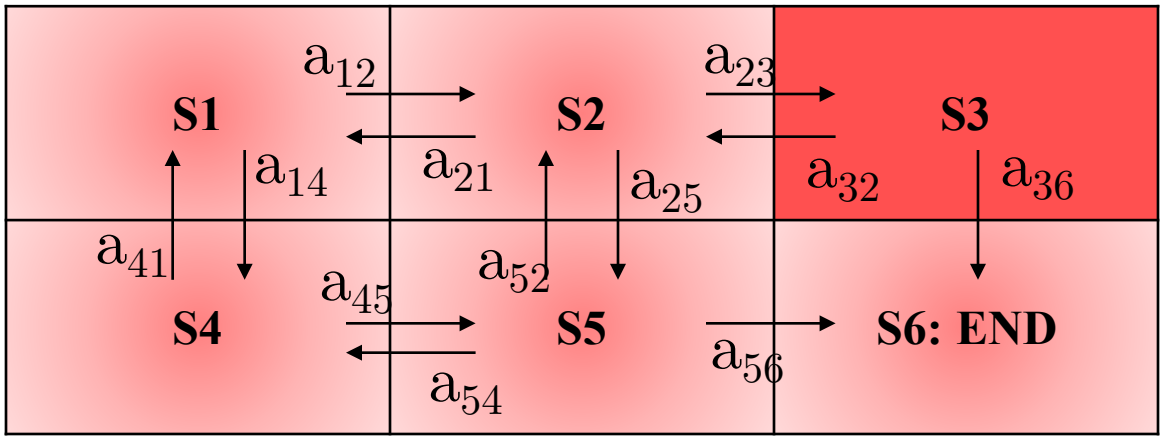
New Action



$Q(S1, a_{12})$	0
$Q(S1, a_{14})$	0
$Q(S2, a_{21})$	0
$Q(S2, a_{25})$	0
$Q(S2, a_{23})$	0
$Q(S3, a_{32})$	0
$Q(S3, a_{36})$	0
$Q(S4, a_{41})$	0
$Q(S4, a_{45})$	0
$Q(S5, a_{52})$	0
$Q(S5, a_{54})$	0
$Q(S5, a_{56})$	0

- Available actions: a_{21}, a_{23}, a_{25}
- Choose a_{23}

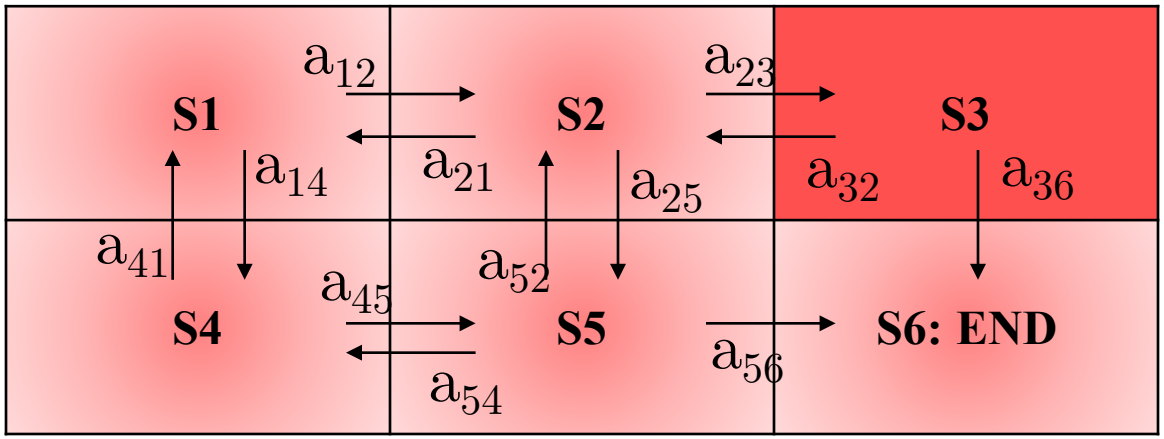
New State, Update



$Q(S1, a_{12})$	0
$Q(S1, a_{14})$	0
$Q(S2, a_{21})$	0
$Q(S2, a_{25})$	0
$Q(S2, a_{23})$	0
$Q(S3, a_{32})$	0
$Q(S3, a_{36})$	0
$Q(S4, a_{41})$	0
$Q(S4, a_{45})$	0
$Q(S5, a_{52})$	0
$Q(S5, a_{54})$	0
$Q(S5, a_{56})$	0

- Reward: 0; New state: S3
- Update state-action:
  $Q(S2, a_{23}) \leftarrow Q(S2, a_{23}) +$
  $\big(0 + 0.5 \cdot \max\{Q(S3, a_{32}), Q(S3, a_{36})\} - Q(S2, a_{23})\big) = 0$

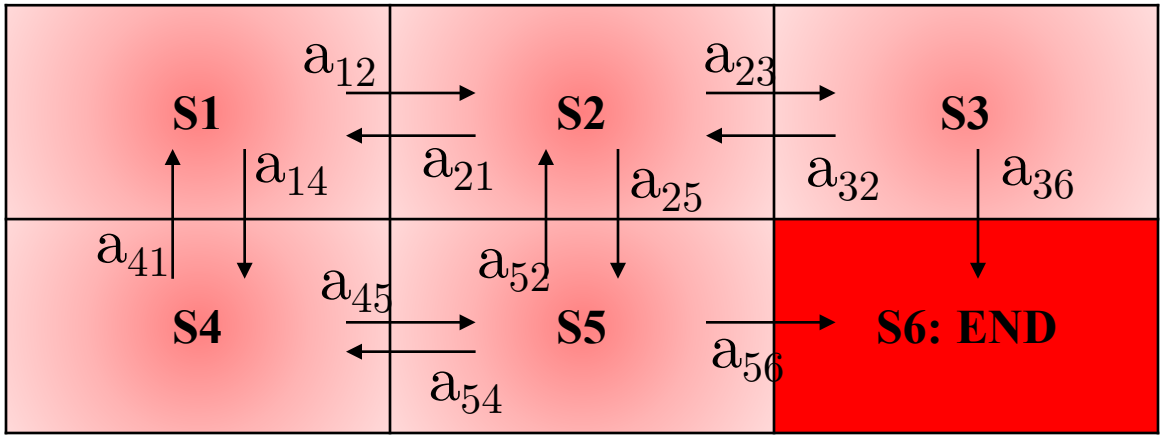
New Action



$Q(S1, a_{12})$	0
$Q(S1, a_{14})$	0
$Q(S2, a_{21})$	0
$Q(S2, a_{25})$	0
$Q(S2, a_{23})$	0
$Q(S3, a_{32})$	0
$Q(S3, a_{36})$	0
$Q(S4, a_{41})$	0
$Q(S4, a_{45})$	0
$Q(S5, a_{52})$	0
$Q(S5, a_{54})$	0
$Q(S5, a_{56})$	0

- Available actions: a_{36}, a_{32}
- Choose a_{36}

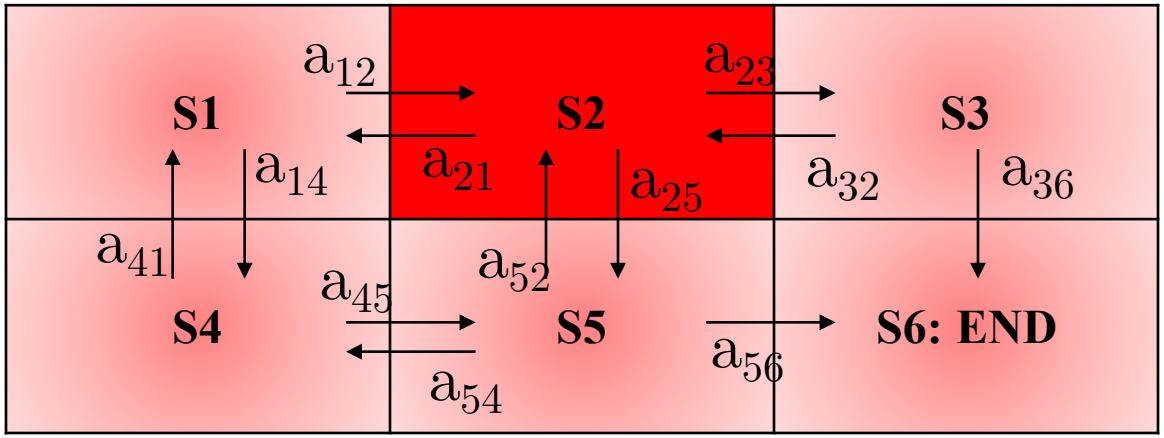
New State, Update



$Q(S1, a_{12})$	0
$Q(S1, a_{14})$	0
$Q(S2, a_{21})$	0
$Q(S2, a_{25})$	0
$Q(S2, a_{23})$	0
$Q(S3, a_{32})$	0
$Q(S3, a_{36})$	100
$Q(S4, a_{41})$	0
$Q(S4, a_{45})$	0
$Q(S5, a_{52})$	0
$Q(S5, a_{54})$	0
$Q(S5, a_{56})$	0

- Reward: 100; New state S6
- Update state-action:
  $Q(S3, a_{36}) \leftarrow Q(S3, a_{36}) + \big(100 + 0.5 \cdot \max\{\} - Q(S3, a_{36})\big) = 100$

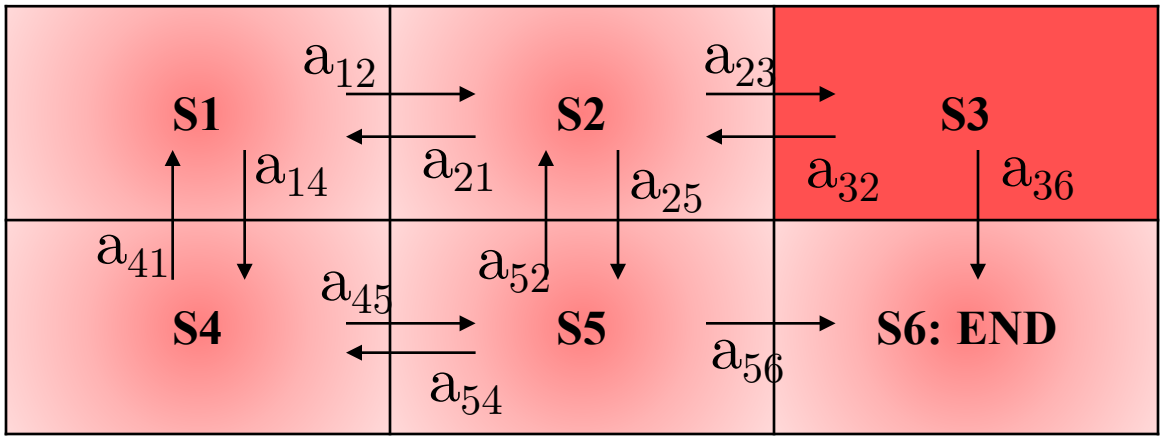
New Episode



$Q(S1, a_{12})$	0
$Q(S1, a_{14})$	0
$Q(S2, a_{21})$	0
$Q(S2, a_{25})$	0
$Q(S2, a_{23})$	0
$Q(S3, a_{32})$	0
$Q(S3, a_{36})$	100
$Q(S4, a_{41})$	0
$Q(S4, a_{45})$	0
$Q(S5, a_{52})$	0
$Q(S5, a_{54})$	0
$Q(S5, a_{56})$	0

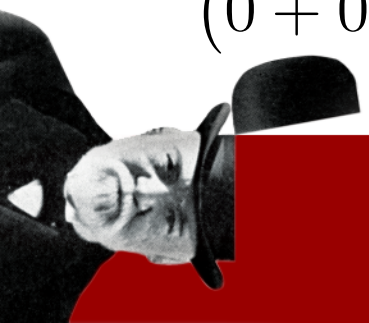
- Available actions: a_{21}, a_{23}, a_{25}
- Choose a_{23}

New State, Update



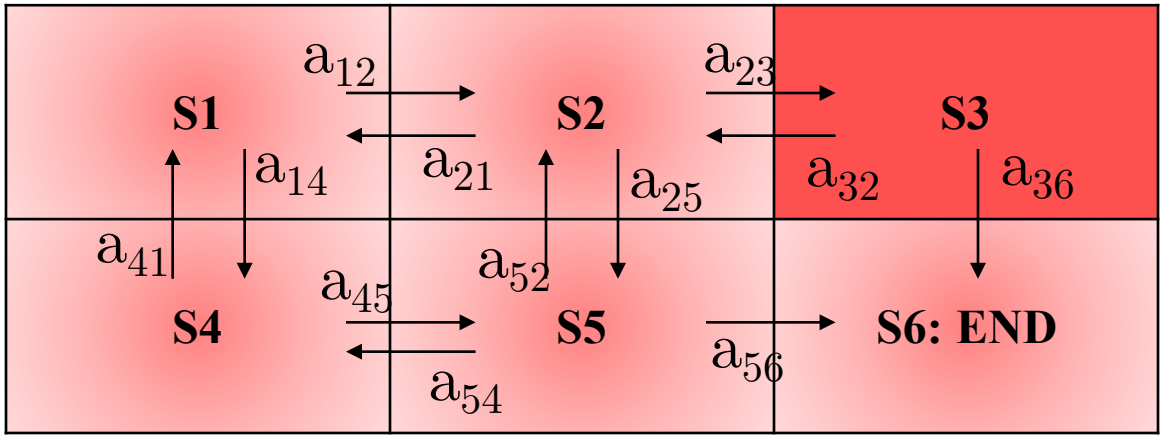
$Q(S1, a_{12})$	0
$Q(S1, a_{14})$	0
$Q(S2, a_{21})$	0
$Q(S2, a_{25})$	0
$Q(S2, a_{23})$	50
$Q(S3, a_{32})$	0
$Q(S3, a_{36})$	100
$Q(S4, a_{41})$	0
$Q(S4, a_{45})$	0
$Q(S5, a_{52})$	0
$Q(S5, a_{54})$	0
$Q(S5, a_{56})$	0

- Reward: 0; New state: S3
- Update state-action:
  $Q(S2, a_{23}) \leftarrow Q(S2, a_{23}) +$
  $\big(0 + 0.5 \cdot \max\{Q(S3, a_{32}), Q(S3, a_{36})\} - Q(S2, a_{23})\big) = 50$



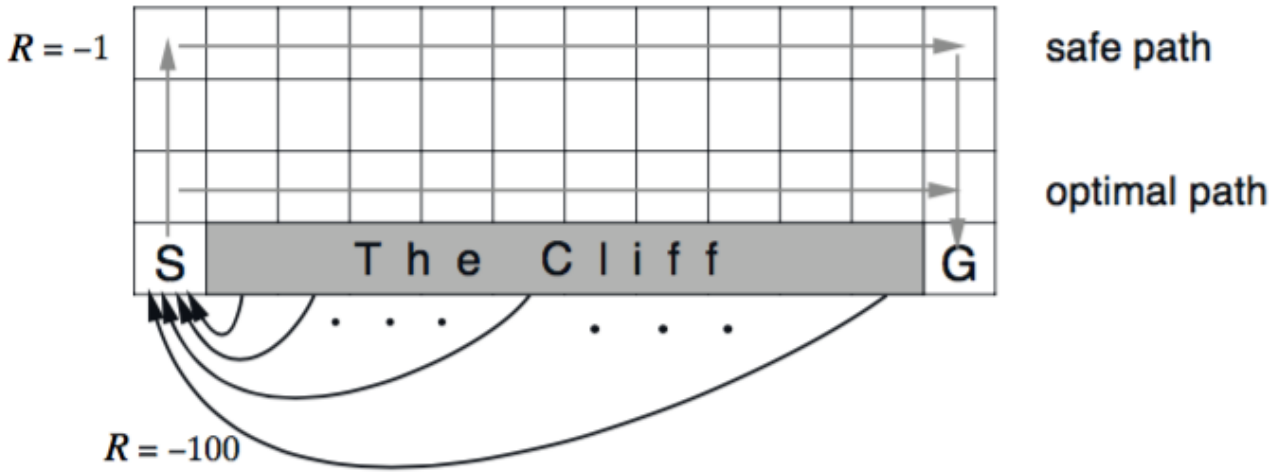
After many episodes ...

The optimal Q-values for the discount factor γ=0.5



$Q(S1, a_{12})$	25
$Q(S1, a_{14})$	25
$Q(S2, a_{21})$	12.5
$Q(S2, a_{25})$	25
$Q(S2, a_{23})$	50
$Q(S3, a_{32})$	25
$Q(S3, a_{36})$	100
$Q(S4, a_{41})$	12.5
$Q(S4, a_{45})$	50
$Q(S5, a_{52})$	25
$Q(S5, a_{54})$	25
$Q(S5, a_{56})$	100

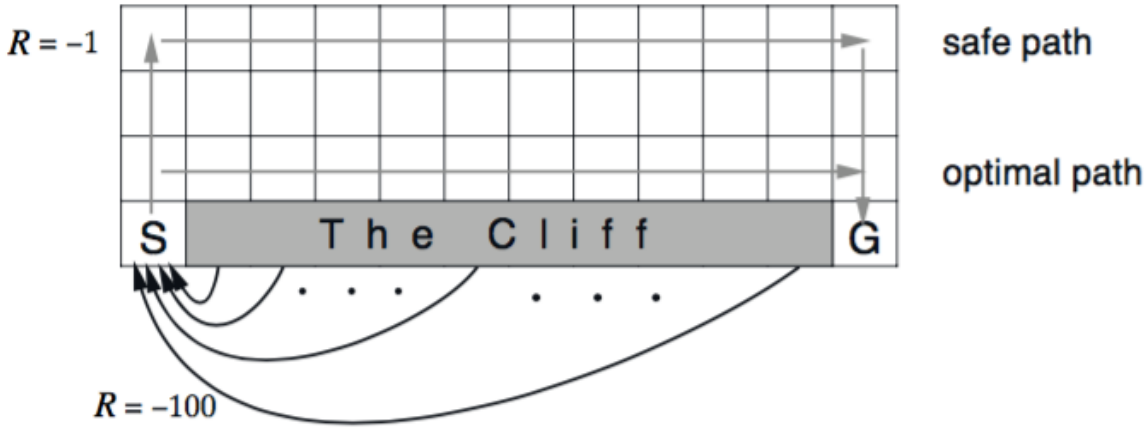
SARSA vs. Q-learning



- Episodic task in a grid world with a start state S and a goal state G
- R = -1 on all state transitions except when stepping on the *Cliff*, where R=-100 and the agent is sent back to S
- Up, down left, right deterministic actions are available at each state
- Both algorithms employ an *ε-greedy behavior policy*, ε=0.1
- The optimal policy is that moving along the edge of the cliff

Example from Sutton and Barto

SARSA vs. Q-learning

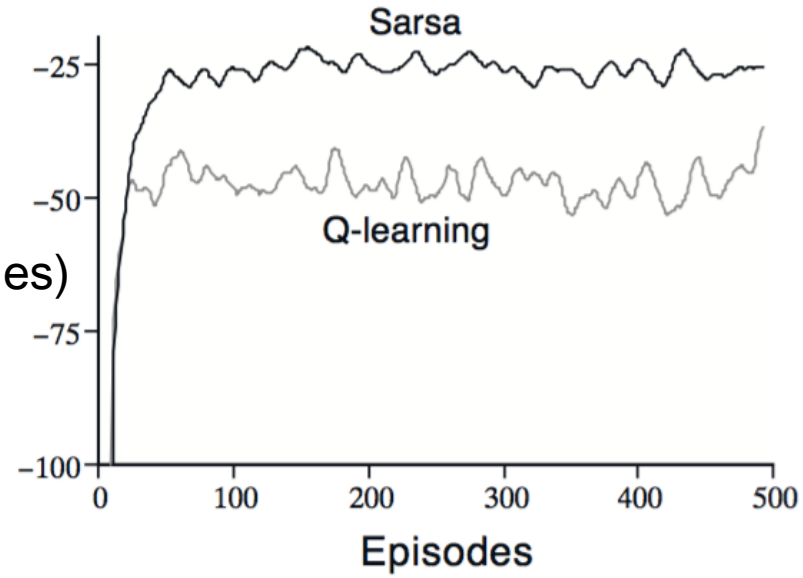


- **Q-learning learns the optimal policy**, however, the *online* (i.e., while learning) performance is not that good since the agent occasionally falls off the cliff because of the ε-greedy action selection
- **SARSA does not learn the optimal policy,** but rather the action values that accounts for ε-greedy action selections, and therefore it learns the longer but safer path that minimizes the risk of R=-100

Example from Sutton and Barto

SARSA vs. Q-learning

Sum of rewards during episode (avg over 10 episodes)



After an initial transient, Q-learning learns the optimal Q-values, corresponding to traveling along the edge of the cliff

Q-learning *online* performance is lower than SARSA because it occasionally falls into the cliff, while rarely happens for SARSA

Example from Sutton and Barto

Q-Learning Properties

- Acting randomly, Q-learning asymptotically converges to optimal state-action values → Finds optimal policy
- More in general, any behavior policy that guarantees that *all state-action pairs continue to be updated* guarantees asymptotic convergence
- In stochastic environments, optimality also requires that the **learning rate** α decreases to zero over time. In deterministic environments a constant α=1 suffices

- We are "free" to choose the behavior policy that is best to perform an effective exploration of the environment

Towards Gathering High Rewards

- Fortunately, acting randomly is sufficient, but not necessary, to learn the optimal values and policy

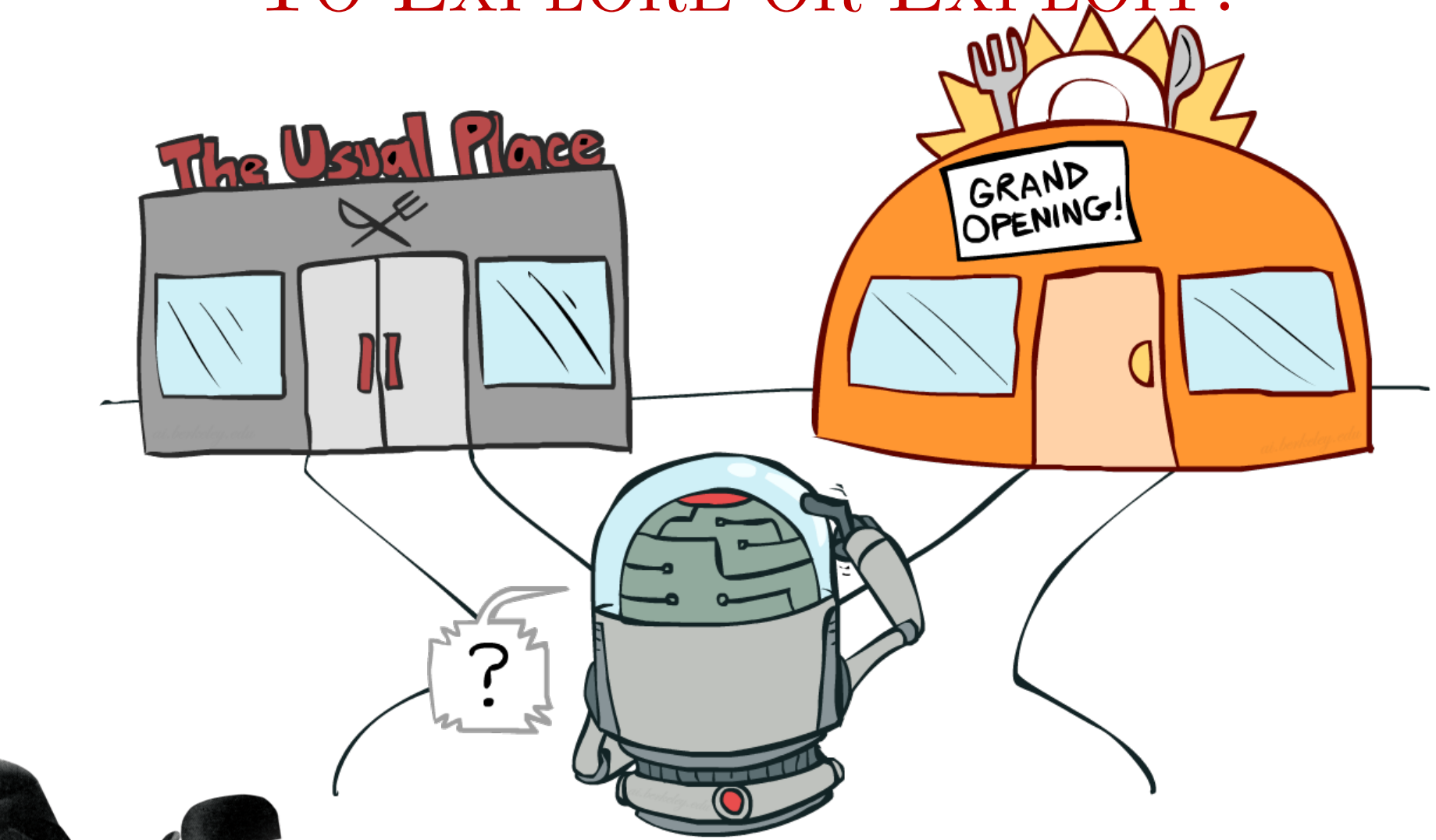
How to Act?

- Initialize s to a starting state
- Initialize $Q(s,a)$ values
- For t=1,2,...
  - Choose a = argmax $Q(s,a)$
  - Observe s',r(s,a,s')
  - Update/Compute Q values (using model-based or Q-learning approach)

- Is this guaranteed to find the optimal policy?

  1. Yes 2. No 3. Not sure

To Explore or Exploit?



Drawing by Ketrina Yim

Simple Approach: ε-greedy

- With probability 1-ε
  - **Choose argmax_a Q(s,a)**
- With probability ε
  - **Select random action**

- Guaranteed to compute optimal policy
- But even after millions of steps still won't always be following policy computed (the argmax Q(s,a))

Greedy in Limit of Infinite Exploration (GLIE)

- ε-greedy approach
- But decay ε over time
- Eventually will be following optimal policy almost all the time

Performance evaluation

How should we evaluate the performance of an algorithm?

- Computational efficiency
- How much reward gathered under algorithm?

Objectives for RL Algorithms

- **Asymptotic guarantees**
  - In *limit* converge to a policy identical to the optimal policy if knew unknown model parameters
  - Q-learning! (under what conditions?)
- **Probably Approximately Correct (PAC)**
  - On all but *finite* number of samples, choose action whose expected reward is close to expected reward of action take if knew model parameters
  - E^3 (Kearns & Singh), R-MAX (Brafman & Tennenholtz)

Sample complexity and PAC learning

- **Sample complexity** of exploration of a reinforcement-learning algorithm: the number of steps in which the algorithm does *not* choose near-optimal actions

- **A PAC algorithm** is such that the sample complexity is bounded by some function **polynomial** in the states, action, and error quantities, with a certain (high) **probability**

General idea of PAC learning

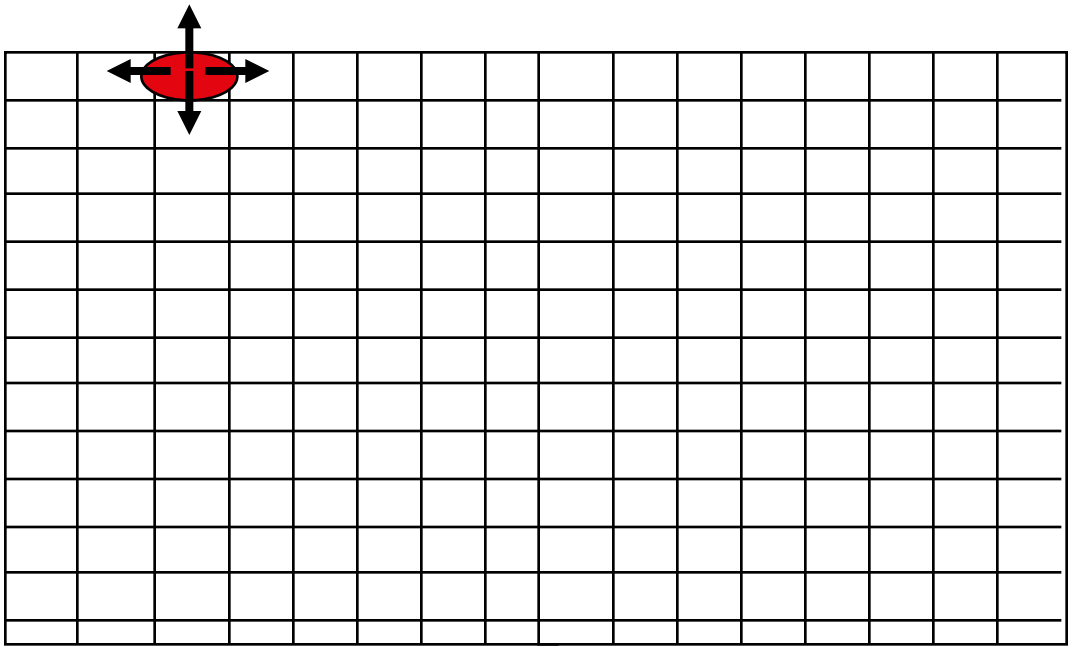
- **Give up (a bit) being exact**
  - We may spare the learner from learning the exact function/model/concept, an approximation would be fine.
- **Allow a small probability that the learning algorithm fails**
  - This would be in some worst cases (e.g., samples are badly chosen). Try to reduce the probability of getting such unlucky examples by considering sufficient #examples
  - A PAC learning strategy places an upper bound on the probability of committing an error by placing a min bound on the number of examples/samples it takes to learn a target

Recall Model-Based RL

- Given data seen so far
- Build an explicit model of the MDP
- Compute policy for it
- Select action for current state given policy, observe next state and reward
- Repeat

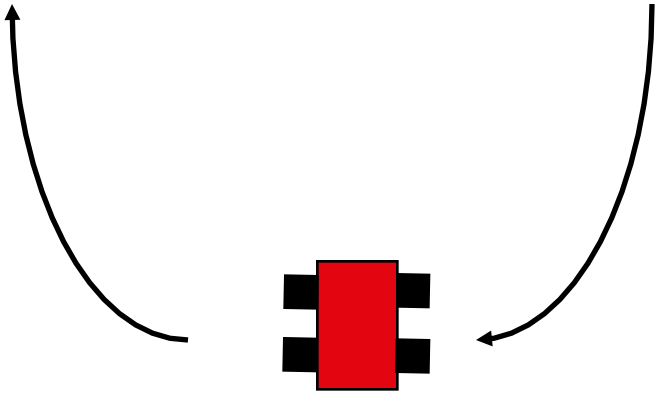
R-max (Brafman & Tennenholtz)

S1 S2 …



R-max is Model-based RL

Think hard: estimate models & compute policies



Act in world

R-max leverages optimism under uncertainty!

If I don't know about s, I assume I can get a good reward there $\rightarrow$ I will go there $\rightarrow$ I will explore it!

R-max Algorithm

- The key idea in R-max is to track if we have enough experience to **trust model estimates** for rewards, R, and dynamics, T
- Set up T and R models in order to favor exploration as long as no enough evidence has been gathered
- Initially mark all state-action pairs as *Unknown*, since no experience has been gathered yet
- All unknown dynamics are *self loops*
- Set reward to ***R-max*** to encourage exploration
- Transition counts are 0 $\rightarrow$ Transition dynamics is flat

R-max Algorithm: Initialize: Define "Known" MDP

(S,A)

Known/ Unknown

	S1	S2	S3	S4	…
↑	U	U	U	U	
→	U	U	U	U	
↓	U	U	U	U	
←	U	U	U	U	

T

Transition Counts

	S1	S2	S3	S4	…
↑	0	0	0	0	
→	0	0	0	0	
↓	0	0	0	0	
←	0	0	0	0	

R

Reward

	S1	S2	S3	S4	…
↑	R_{max}	R_{max}	R_{max}	R_{max}	
→	R_{max}	R_{max}	R_{max}	R_{max}	
↓	R_{max}	R_{max}	R_{max}	R_{max}	
←	R_{max}	R_{max}	R_{max}	R_{max}	

In the "known" MDP, any unknown (s,a) pair has its dynamics set as a self loop & reward = Rmax

R-max Algorithm

Plan in known MDP

R-max: Planning

- Compute optimal policy π_{known} for "known" MDP

Exercise: What Will Initial Value of Q(s,a) be for each (s,a) Pair in the Known MDP? What is the Policy?

Known/ Unknown

	S1	S2	S3	S4	…
↑	U	U	U	U	
→	U	U	U	U	
↓	U	U	U	U	
←	U	U	U	U	

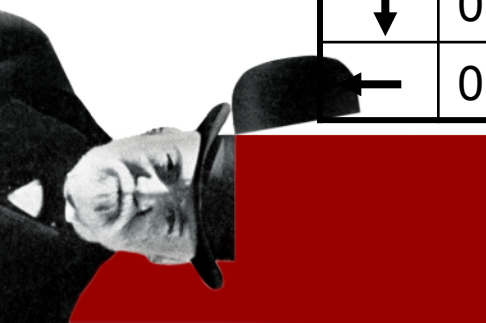
Reward

	S1	S2	S3	S4	…
↑	R_{max}	R_{max}	R_{max}	R_{max}	
→	R_{max}	R_{max}	R_{max}	R_{max}	
↓	R_{max}	R_{max}	R_{max}	R_{max}	
←	R_{max}	R_{max}	R_{max}	R_{max}	

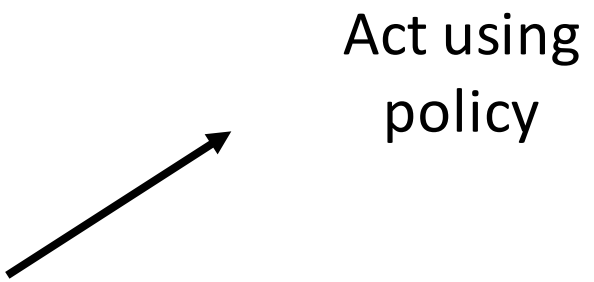
Transition Counts

	S1	S2	S3	S4	…
↑	0	0	0	0	
→	0	0	0	0	
↓	0	0	0	0	
←	0	0	0	0	

In the "known" MDP, any unknown (s,a) pair has its dynamics set as a self loop & reward = Rmax



R-max Algorithm



- Given optimal policy π_{known} for "known" MDP
- Take best action for current state $\pi_{\text{known}}(s)$, transition to new state s' and get reward r

R-MAX ALGORITHM

Act using policy

Plan in known MDP

Update state-action counts

Update Known MDP

Known/ Unknown

	S2	S2	S3	S4	…
↑	U	U	U	U	
→	U	U	U	U	
↓	U	U	U	U	
←	U	U	U	U	

Reward

	S2	S2	S3	S4	…
↑	R_{max}	R_{max}	R_{max}	R_{max}	
→	R_{max}	R_{max}	R_{max}	R_{max}	
↓	R_{max}	R_{max}	R_{max}	R_{max}	
←	R_{max}	R_{max}	R_{max}	R_{max}	

Transition Counts

	S2	S2	S3	S4	…
↑	0	0	0	0	
→	0	0	**1**	0	
↓	0	0	0	0	
←	0	0	0	0	

Increment counts for state-action tuple

Update Known MDP

Known/ Unknown

	S2	S2	S3	S4	…
↑	U	U	U	U	
→	U	U	**K**	U	
↓	U	U	U	U	
←	U	U	U	U	

Reward

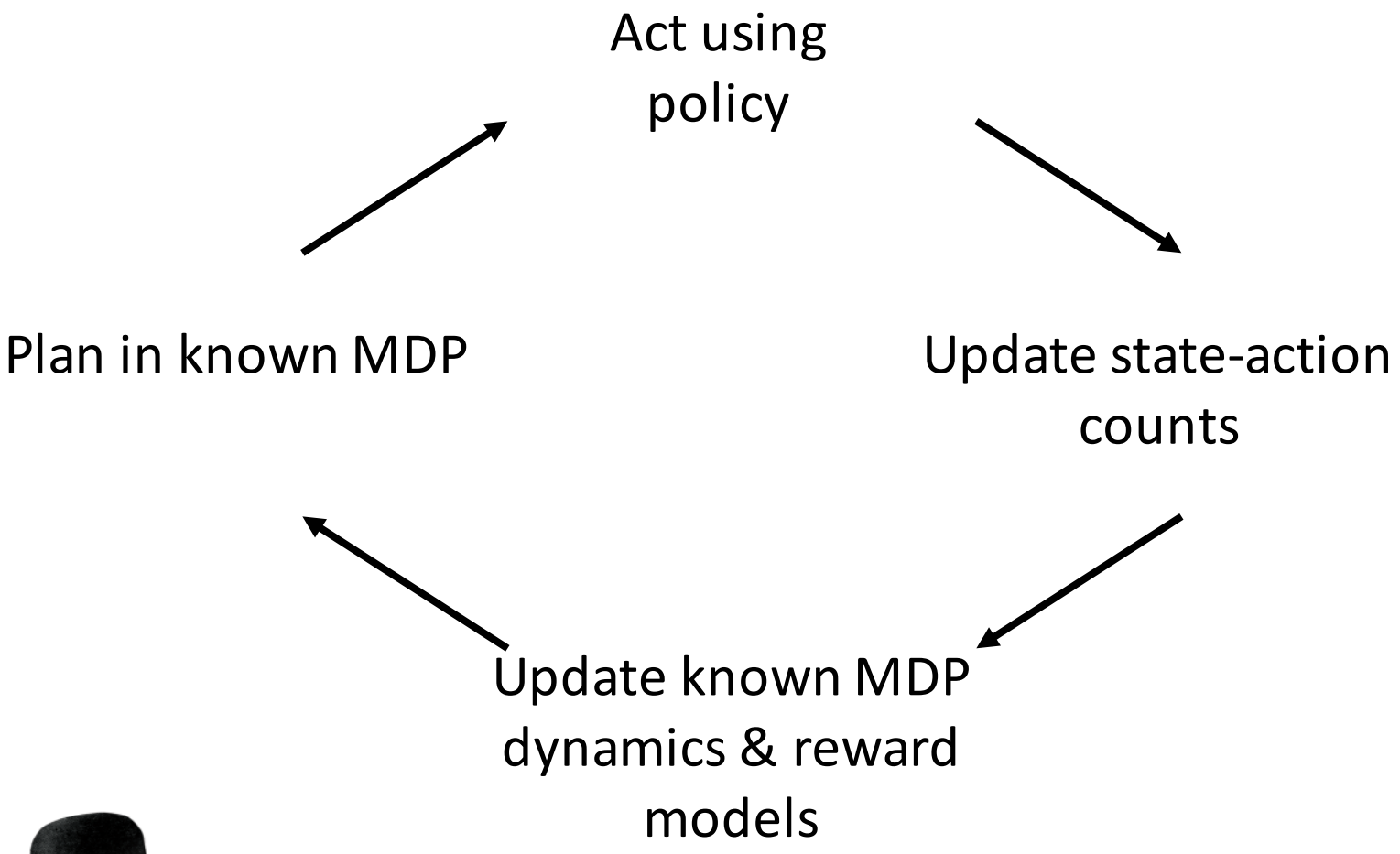
	S2	S2	S3	S4	…
↑	R_{max}	R_{max}	R_{max}	R_{max}	
→	R_{max}	R_{max}	**R**	R_{max}	
↓	R_{max}	R_{max}	R_{max}	R_{max}	
←	R_{max}	R_{max}	R_{max}	R_{max}	

Transition Counts

	S2	S2	S3	S4	…
↑	3	3	4	3	
→	2	4	**5**	0	
↓	4	0	4	4	
←	2	2	4	1	

If counts for (s,a) > N, (s,a) becomes *known*: **use observed data to estimate transition & reward model** for (s,a) when planning

R-MAX ALGORITHM



Estimating MDP Model for a (s,a) Pair Given Data

- Transition model estimation

- Reward model estimation

Sample Complexity of R-max

$$\tilde{O}\left(\frac{SA}{\varepsilon(1-\gamma)^2}\underbrace{\frac{S}{\varepsilon^2(1-\gamma)^4}}\right)$$

samples need per (s,a) pair

PAC guarantees: On all but the above number of steps, chooses action whose expected reward is close to expected reward of action take if knew model parameters, with high probability

Common Idea of Model-Based PAC RL Algorithms:

Quantify how **many samples** do we need to build a **good model** that we can use to **act well** in the world?

"Good" RL Models

- Estimate model parameters from experience
- More experience means our estimated model parameters will closer be to the true unknown parameters, with high probability

Acting Well in the World

$p(s' \mid s,a)$ known → Compute ε-optimal policy

Bound $\left(\tilde{p}(s' \mid s,a) - p(s' \mid s,a)\right)$ → Bound error in policy calculated using $\tilde{p}(s' \mid s,a)$

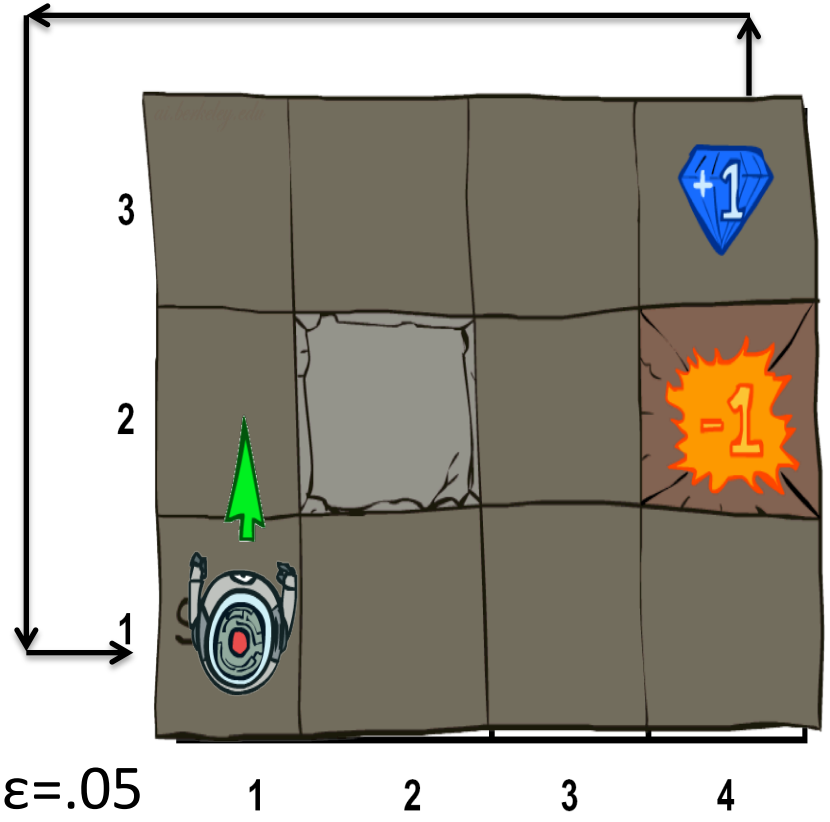
How many samples do we need to build a good model that we can use to act well in the world?

Sample complexity = # steps on which may not act well (could be far from optimal)

(R-MAX and E^3) = Poly(# of states)

Is this a small number of steps?

$$\tilde{O}\left(\frac{SA}{\varepsilon(1-\gamma)^2}\,\frac{S}{\varepsilon^2(1-\gamma)^4}\right)$$



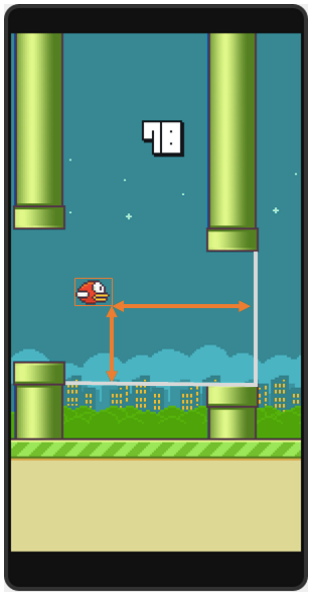
γ=.99, ε=.05

What You Should Know About RL

- Define
  - Exploration vs exploitation tradeoff
  - Model free and model based RL
- Implement Q-learning and R-max
- Describe evaluation criteria for RL algorithms
  - empirical performance, computational complexity sample/data efficiency
  - Contrast strengths & weaknesses of Q-learning vs R-max in terms of these criteria. Give examples of where each might be preferred

Some (interesting) RL Fun

- RL in the "real" world: Flappy Bird game
- http://sarvagyavaish.github.io/FlappyBirdRL/



- DeepMind's Deep Q-learning playing Atari games at human expert level
- https://www.youtube.com/watch?v=V1eYniJ0Rnk