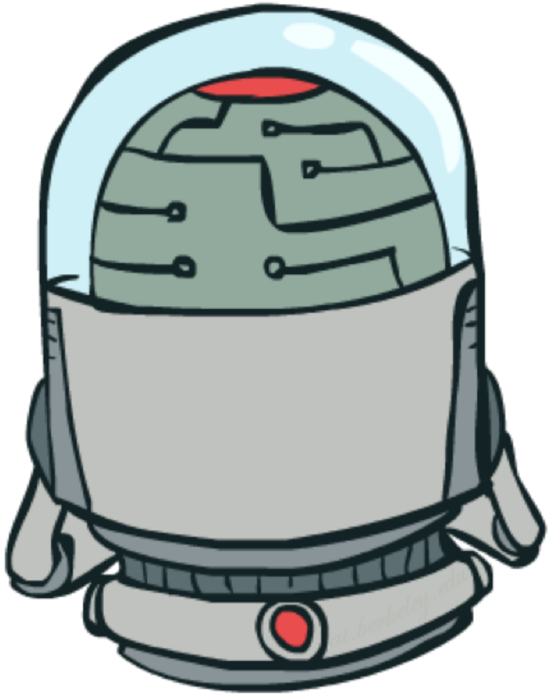
# Announcements

- HW4 will be out today

# Double Bandits

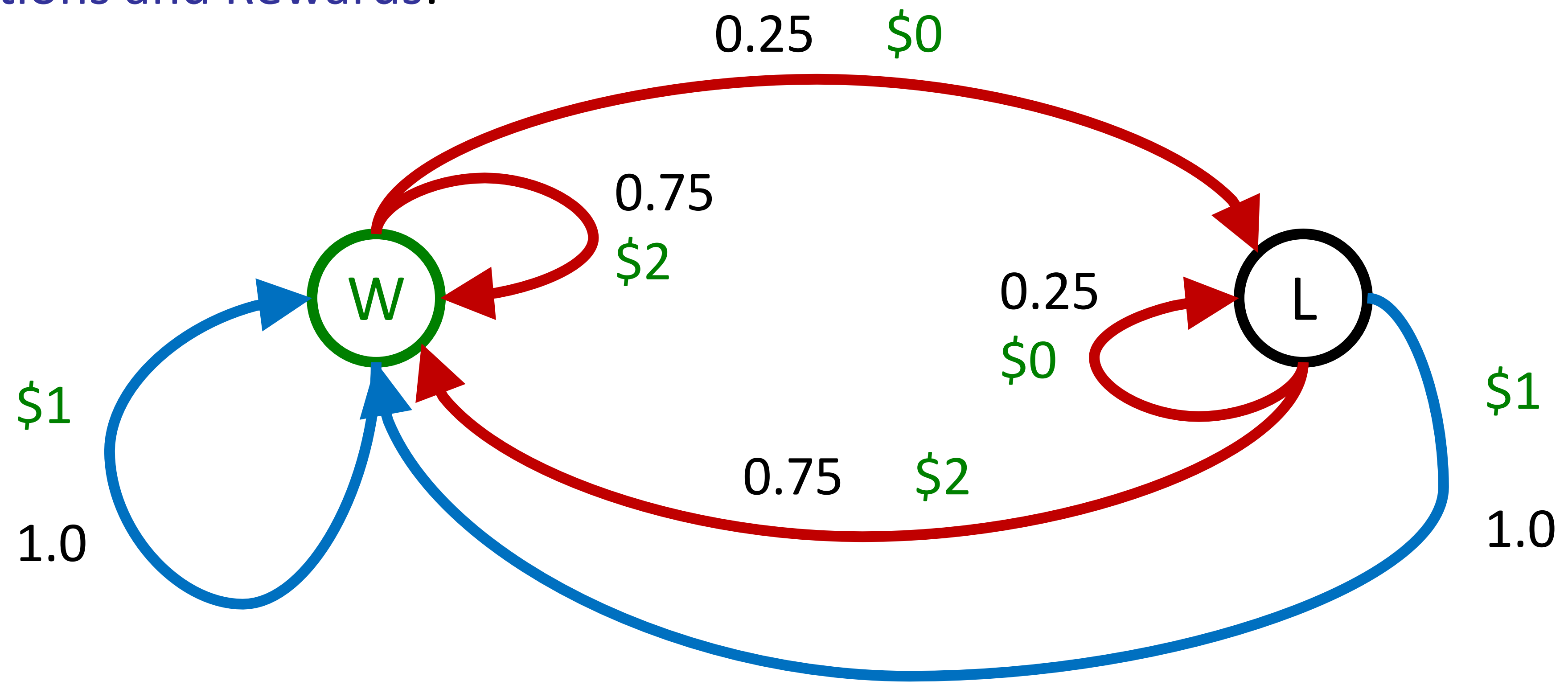



# Double-Bandit MDP

- Actions: *Blue*, *Red*
- States: Win, Lose
- Transitions and Rewards:

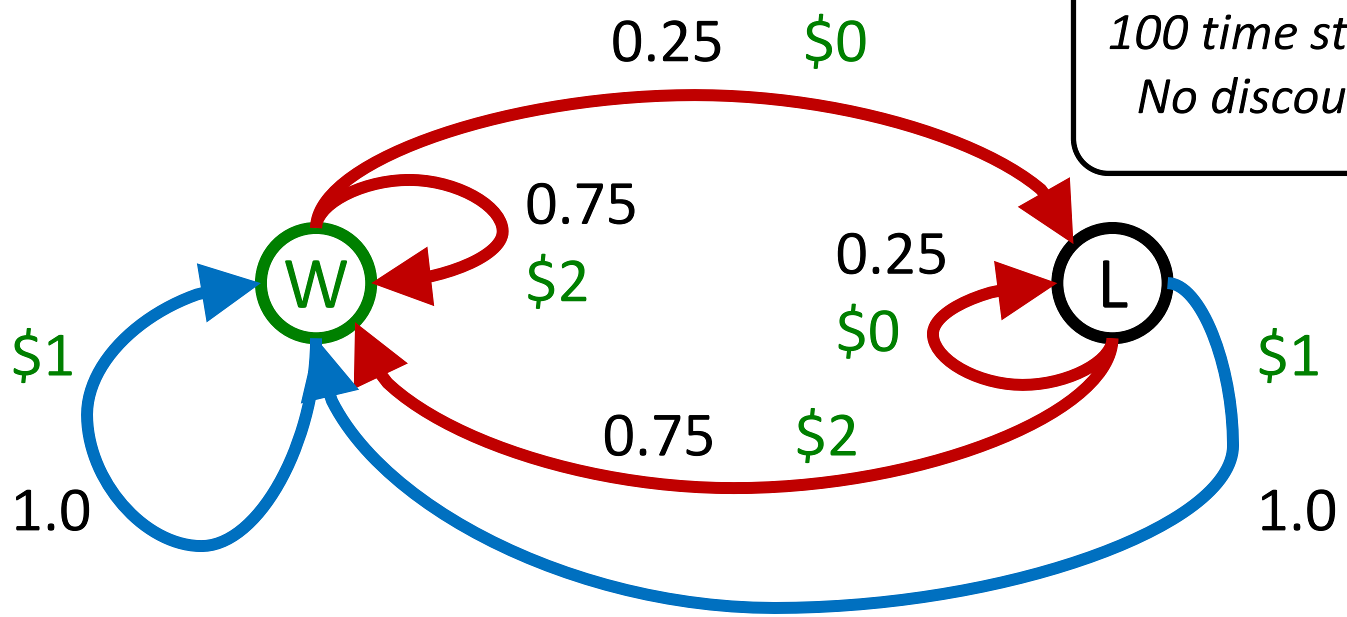
*Both states have the same value*
*(Actions have the same consequences)*

*No discount*
*100 time steps*



# Offline Planning

- Solving MDPs by offline planning
  - You know the details of the MDP
  - You determine all quantities through computation (VI/PI)
  - Note: you do **not actually** play the game!
    - Only simulating using the MDP **model**:




*100 time steps*
*No discount*

**Q-Values**

*Both states have the same value*

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

# Let's Play!


Which one to play?

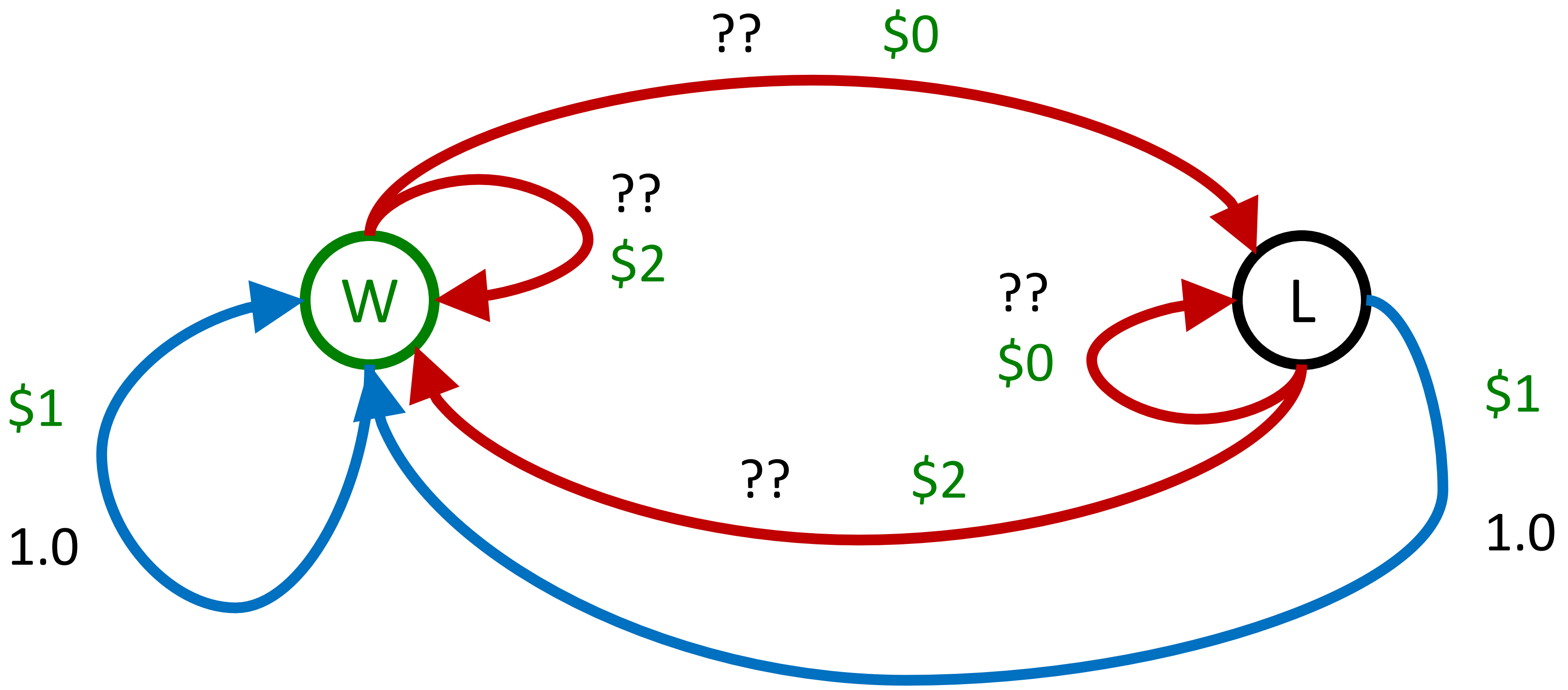

\$2 \$2 \$0 \$2 \$2

\$2 \$2 \$0 \$0 \$0

*\$12 is indeed higher than \$10*

# Online Planning

- Red bandit changed! Red's win chance is different and unknown.



# Let’s Play!


Which one to play?


$0  $0  $0  $2  $0

$2  $0  $0  $0  $0

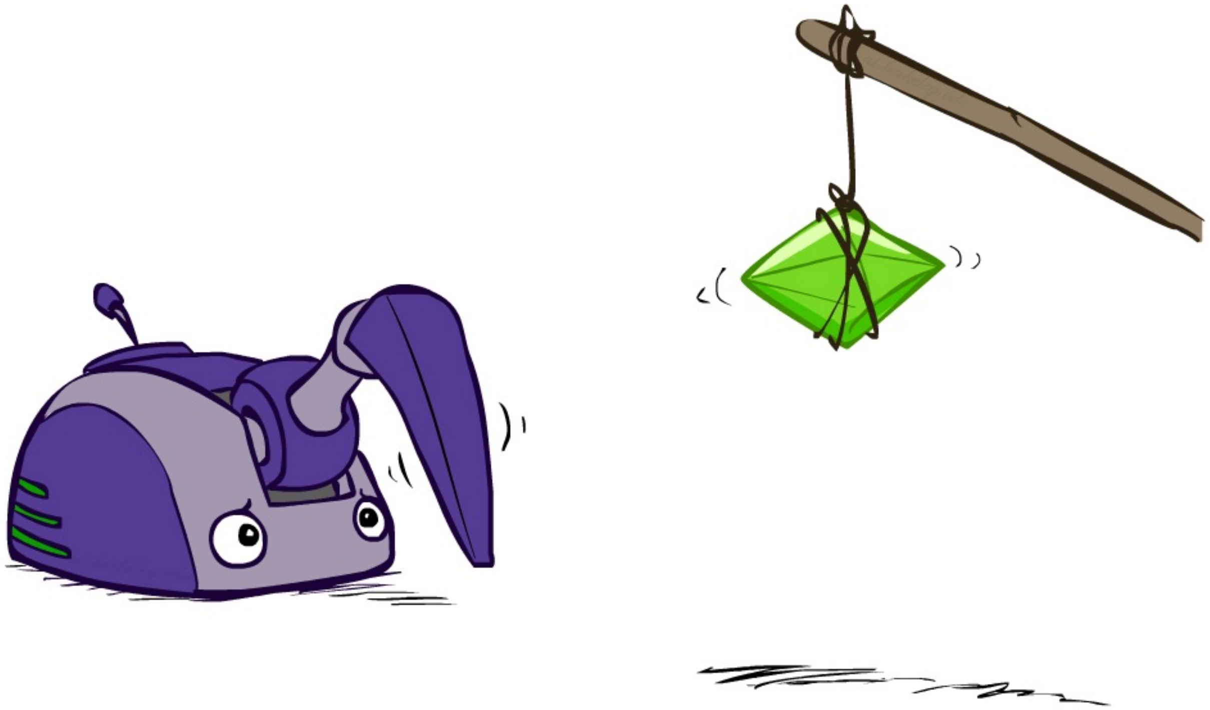
# What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you don't have full information on it
  - So, you couldn't solve it with just computation
  - You needed to **actually act** to figure things out
    - Previously for planning, you didn't need to actually act


- Learning can be much harder than solving a known MDP!
  - Critical information missing

# CS 3317: Artificial Intelligence

## Reinforcement Learning I



Instructors: **Cai Panpan**

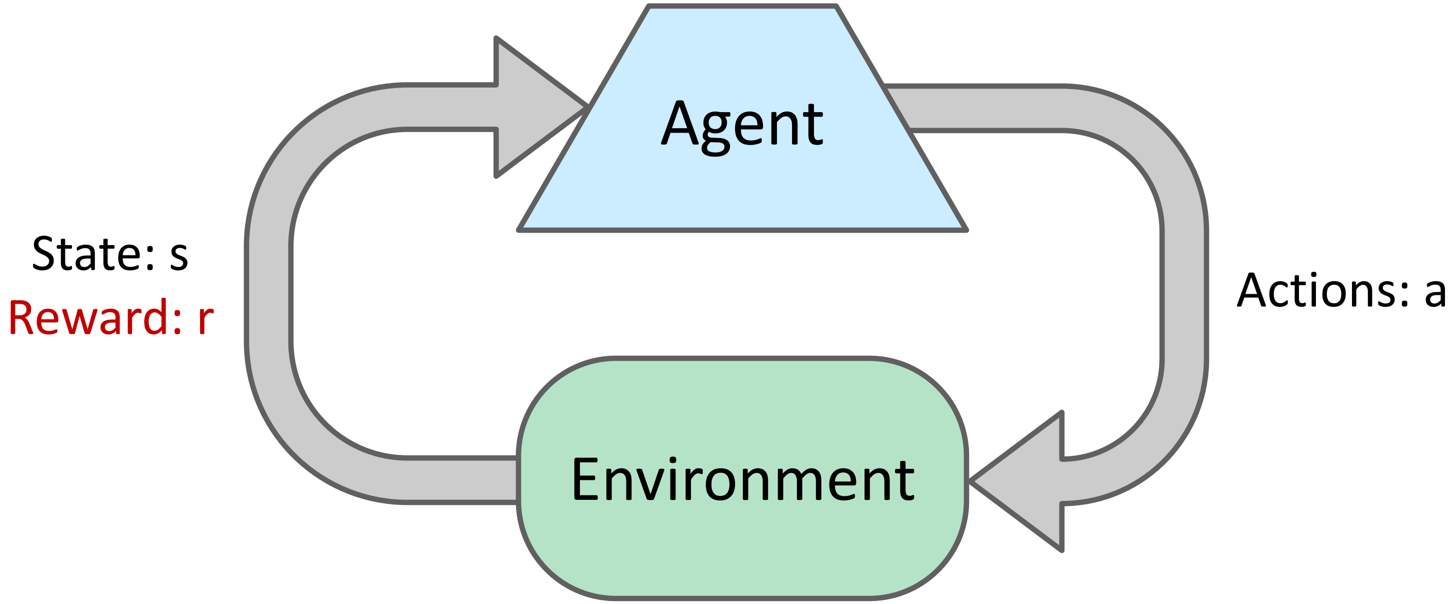
Shanghai Jiao Tong University

(slides adapted from UC Berkeley CS188)

# Reinforcement learning

- Core question: *What if the MDP is initially unknown?*
- New ideas coming up!
  - ***Exploration***: gather information
    - you have to ***try unknown actions*** to get information
  - ***Exploitation***: get rewards
    - eventually, you have to use what you know to ***make decisions***
  - ***Regret***: initially, you inevitably "make mistakes" and lose reward
  - ***Sampling***: you may need to repeat many times to get good estimates
  - ***Generalization***: what you learn in one state may apply to others too

# Reinforcement Learning




- Basic idea:
  - Treat the environment as a black box
  - Learn how to ***maximize expected rewards*** based on observed samples of transitions

# A Basic Problem Faced by All Living Things

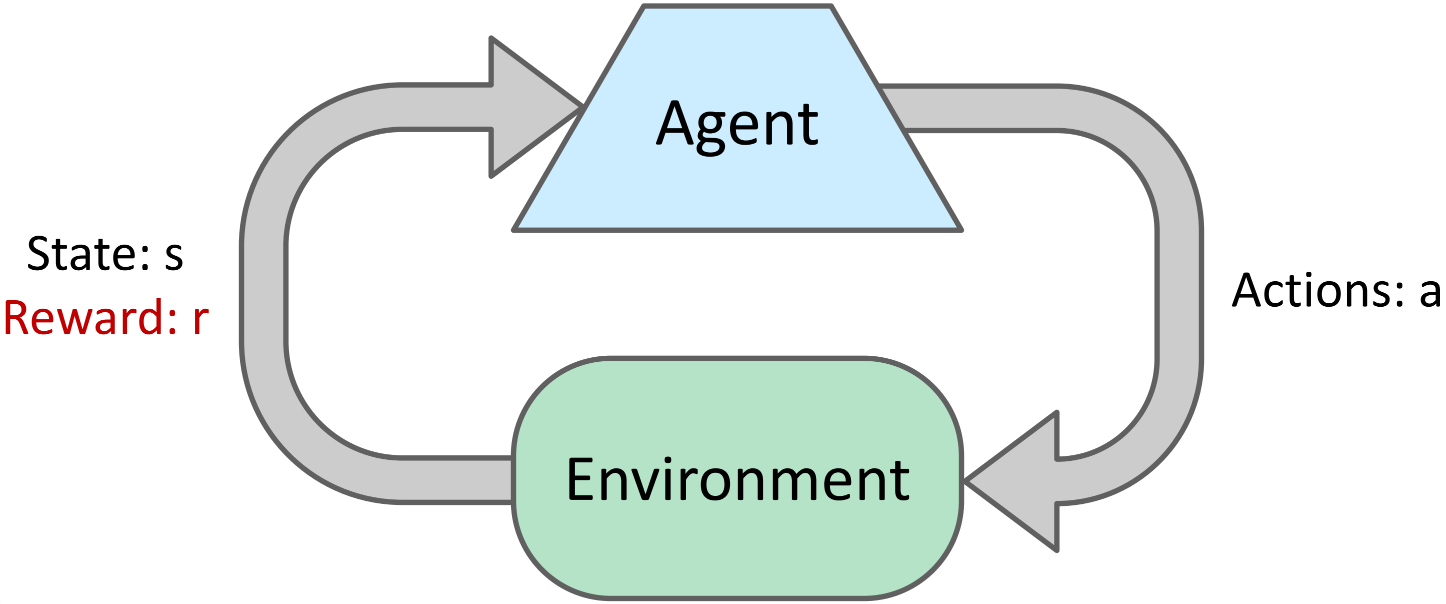



Image credit: GPT4-turbo

- Example: baby learning to walk:
  - Action: moving arms and legs
  - State: configuration of the entire body
  - Reward: praise from parents
  - Learning by trying different actions and observing outcomes -> reinforcement learning!

# AI Example: Samuel’s checker player (1956-67)

# Robotics Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

# Example: Learning to Walk

Finished

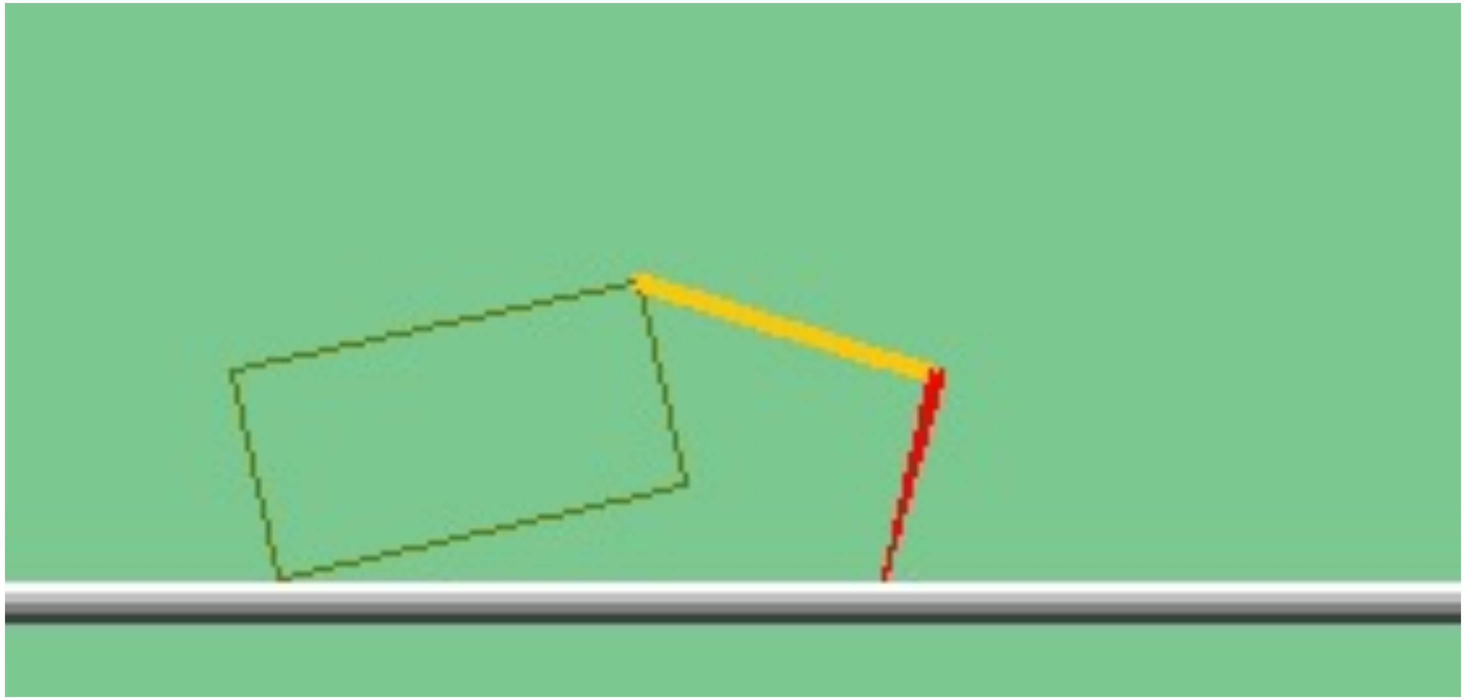
[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

# Example: Breakout (DeepMind)

# Recent Example: AlphaGo (2016)

# Our Lecture: The Crawler!



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

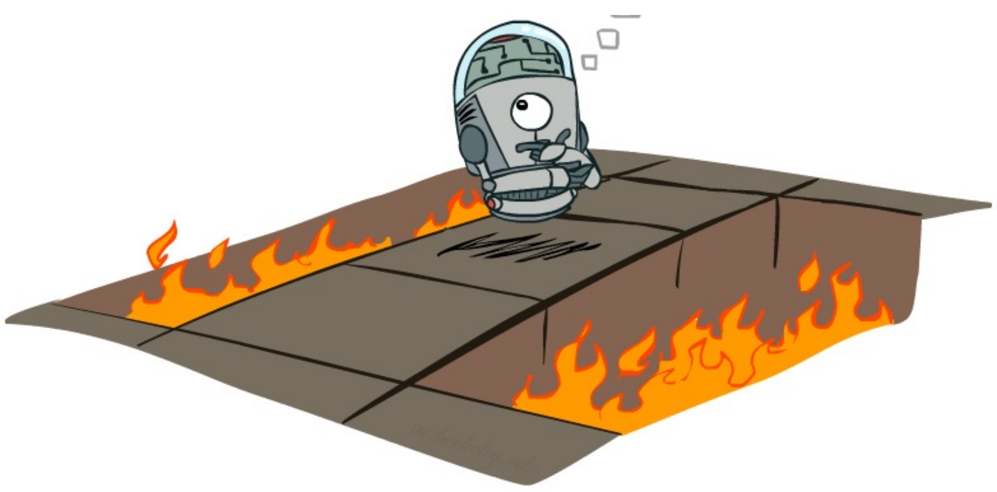
# Video of Demo Crawler Bot

# Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) $A(s)$
  - A transition 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 the consequence of actions and goodness of states
  - Must explore new states and actions
    - -- to bravely go where no robot has gone before

# MDP Planning vs. Reinforcement Learning



**MDP**
Planning with a model

**RL**
Learning from trial and error

# Approaches to reinforcement learning

1. Model-based learning

   *Learn the model, solve it, execute the solution*

2. Value-based methods

   *Learn values from experiences, use them to make decisions*

   a. Direct evaluation

   b. Temporal difference learning

   c. Q-learning

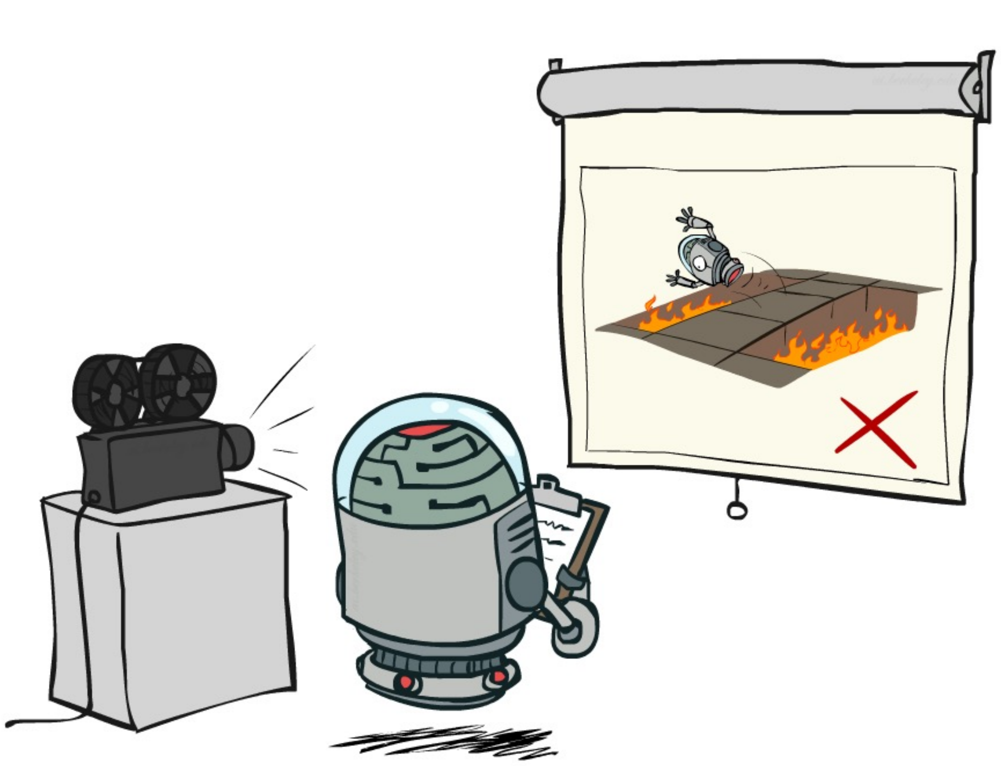
3. Policy-based methods

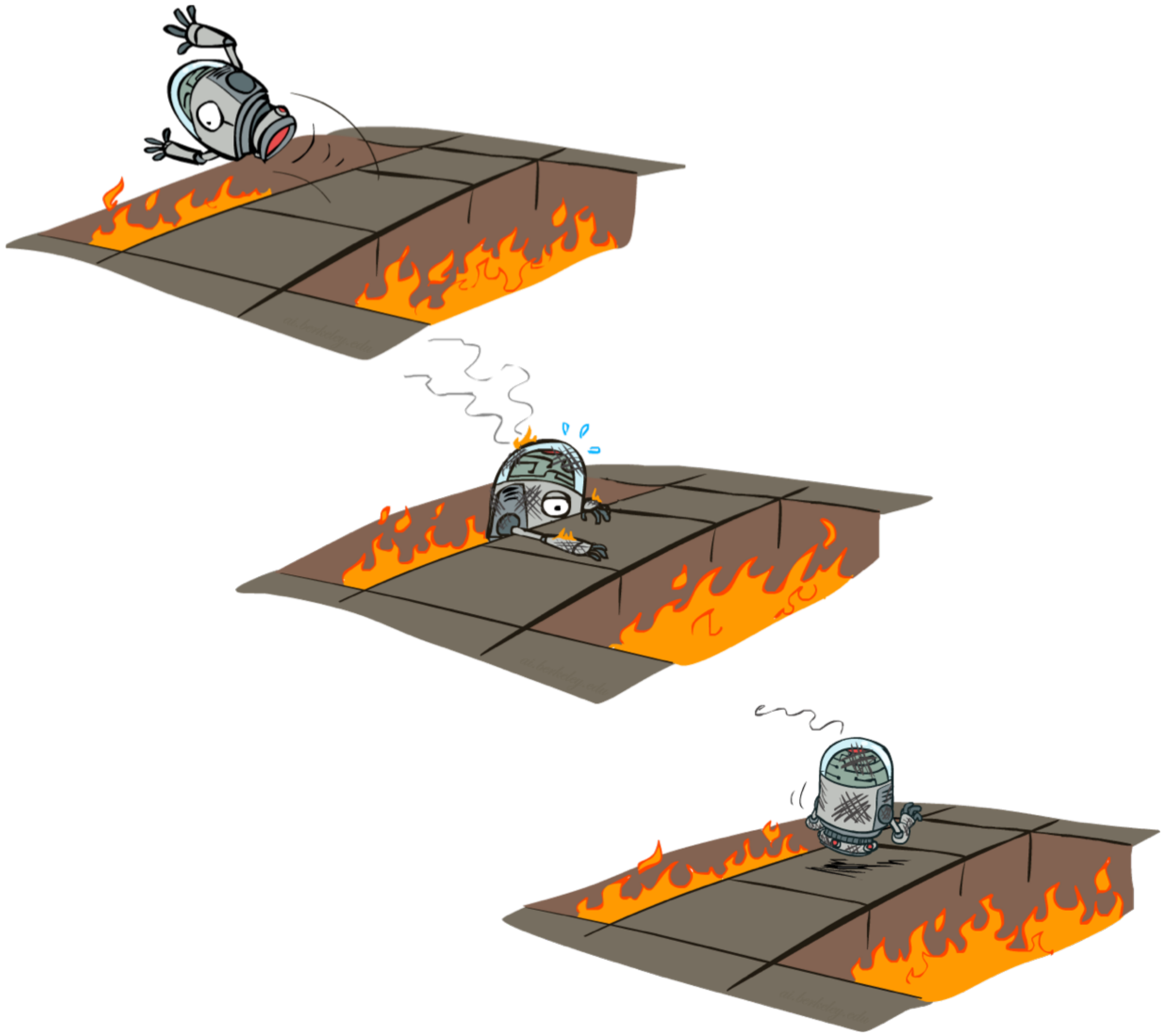
   *Directly learn policies from experiences*

   - See Sutton&Barto’s book Chapter 13

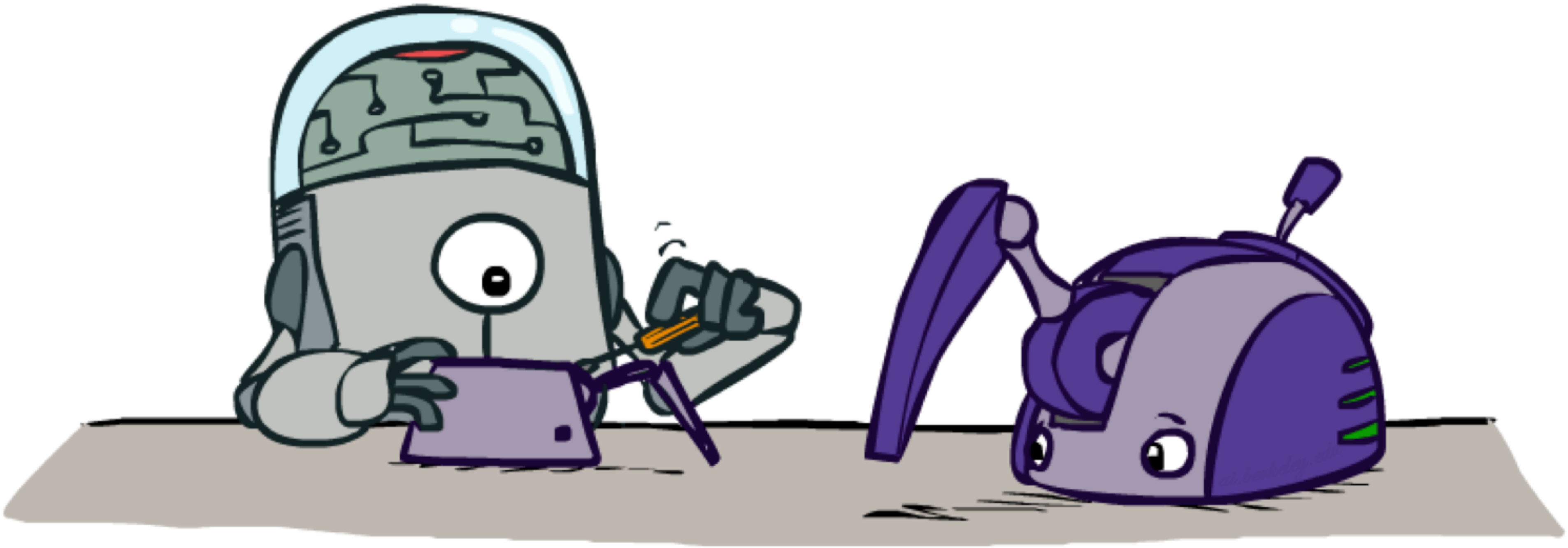
Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. (available on Canvas)

# Passive vs Active Reinforcement Learning





# Approach 1: Model-Based RL



# Model-Based Learning

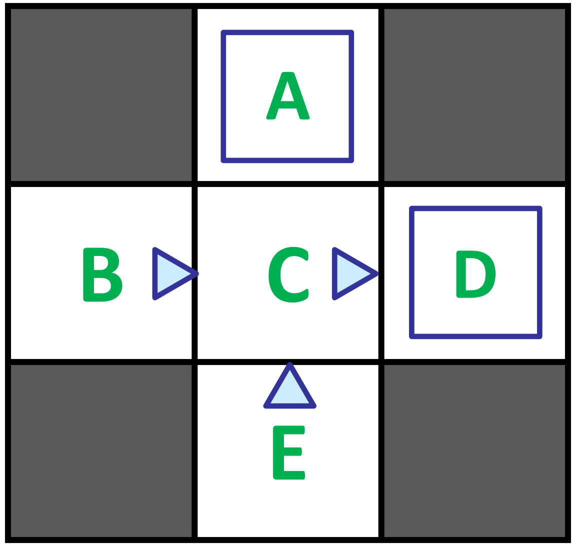
- Core ideas:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- Step 1: Learn empirical MDP model
  - Given a set of experiences {…, (*s*,*a*,*s'*,r), …}
  - Estimate each probability in *T*(*s*,*a*,*s'*) from counts
    - Count the frequency of visiting *s'* for each (*s*,*a*) pair
    - Fill number in transition table
  - Discover each *R*(*s*,*a*,*s'*) when we experience the transition
    - Fill number in reward table

- Step 2: Solve the learned MDP
  - Use, e.g., value or policy iteration, as before

# Example: Model-Based Learning

## Input Policy $\pi$




*Assume:* $\gamma = 1$

## Observed Episodes (Training)

### Episode 1

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

### Episode 2

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

### Episode 3

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

### Episode 4

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

## Learned Model

### $T(s,a,s')$

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

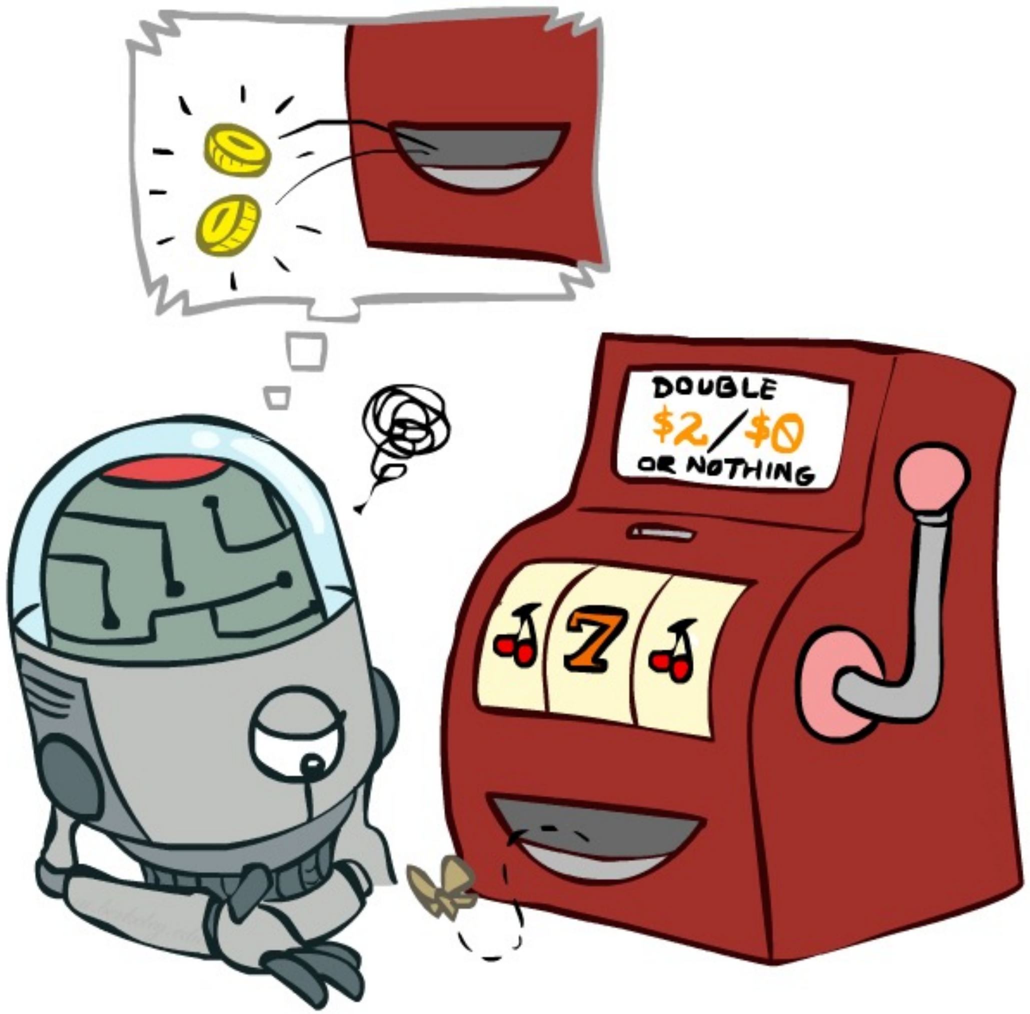
### $R(s,a,s')$

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

# Pros and cons

- Pro:
  - Makes efficient use of experiences (low ***sample complexity***)
- Con:
  - May not scale to large state spaces
    - Needs T and R tables of size $|A||S|^2$
    - Learns one (s,a) pair at a time (fixable later on)
    - Cannot solve MDP for very large $|S|$ (also somewhat fixable)
  - Much harder with partially observability
    - Need to learn an additional observation model M(a,s',o)

# Approach 2: Model-Free Learning



# Core ideas behind model-free methods

- Mathematical insight:

  To approximate expectations with respect to a distribution, you can either:

  - First estimate distribution from samples, then compute expectation
  - Or, directly estimate the expectation from samples

# Example: Expected Age

Goal: Compute expected age of CS3317 students

Known P(A)

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

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

"Model Based": estimate P(A):

$$\hat{P}(A=a) = N_a/N$$

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

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

=

"Model Free": estimate expectation

$$E[A] \approx 1/N \sum_i a_i$$

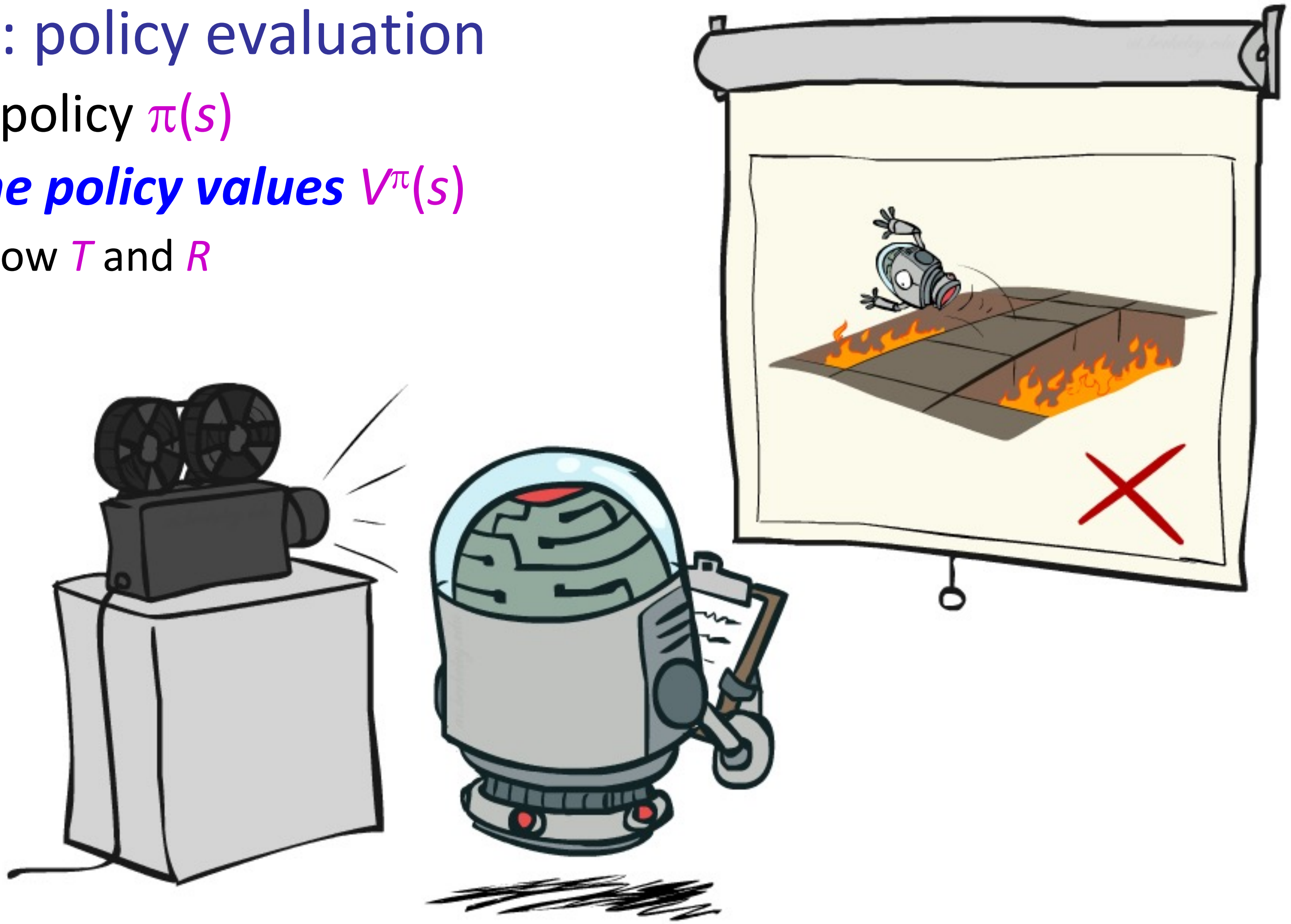
Why does this work? Because samples appear with the right frequencies.

# Value-based methods

- Value is an expectation over utilities
- We can estimate the expectation directly from samples, without learning a model

# Setup: Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - ***Goal: learn the policy values*** $V^{\pi}(s)$
    - You don't know $T$ and $R$



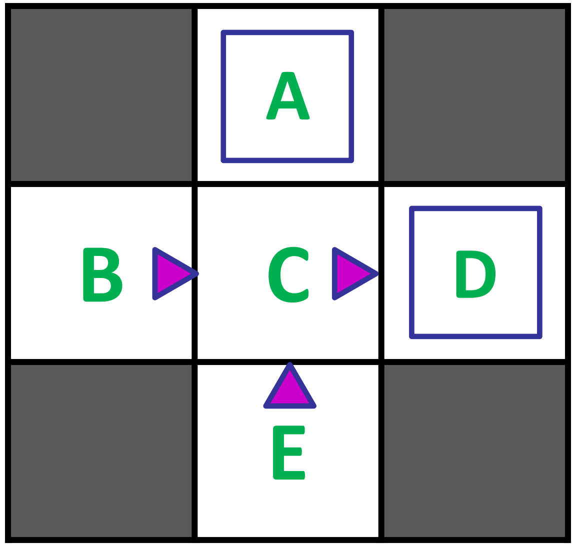
# Direct evaluation

- Goal: Estimate $V^{\pi}(s)$, i.e., expected total discounted reward from $s$ onwards

- Idea:
  - ***Returns***: the *actual* sums of discounted rewards from $s$
  - Get average returns over multiple trials and visits to $s$
    - Use the average in $V^{\pi}(s)$
- This is called ***direct evaluation*** (or direct utility estimation)

# 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

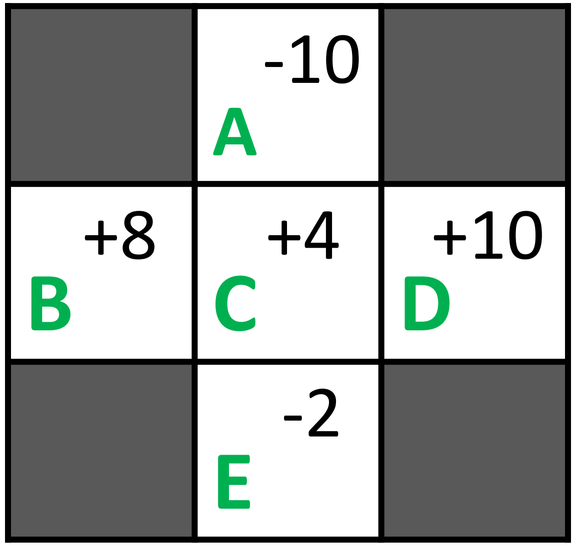
**Episode 3**

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

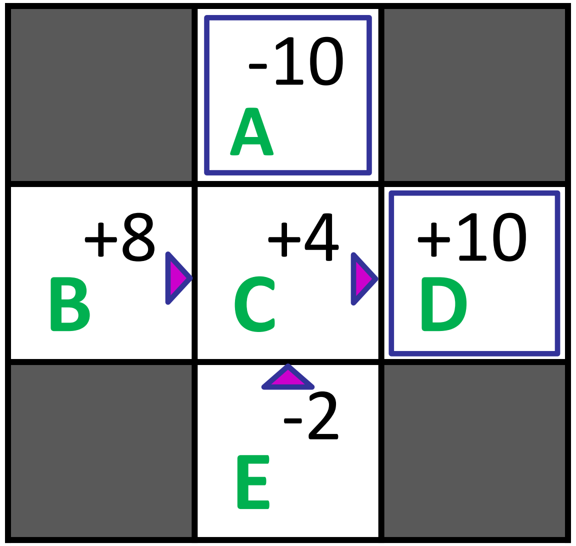


# Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of *T* and *R*
  - It converges to the right answer in the limit

- What's bad about it?
  - Each state must be learned separately (fixable)
  - It ***ignores information about state connections***
  - So, it takes a long time to learn

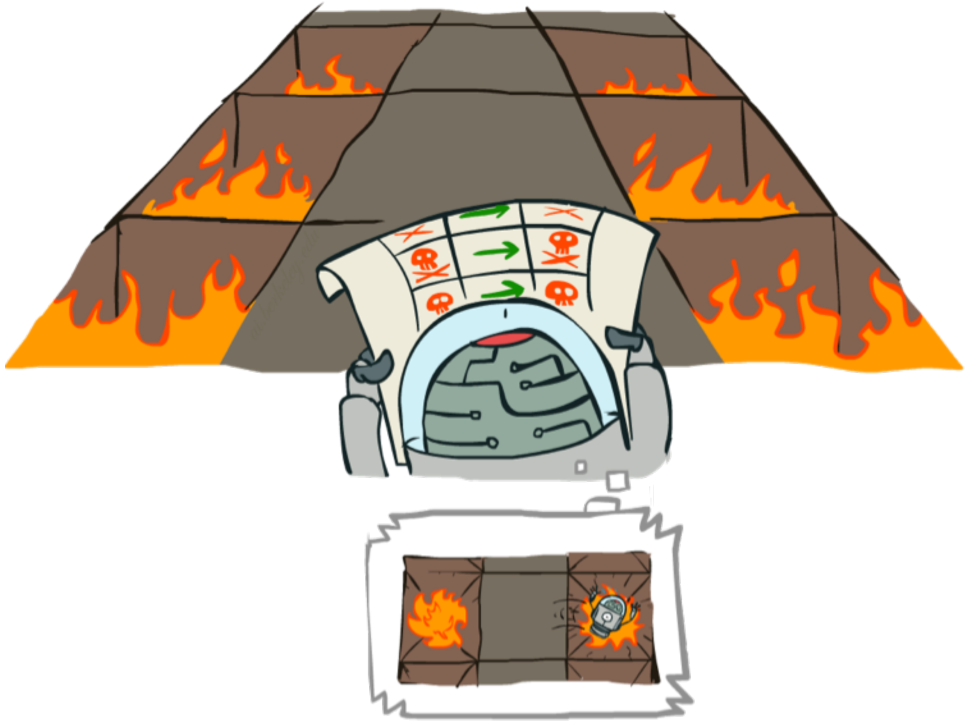
Output Values

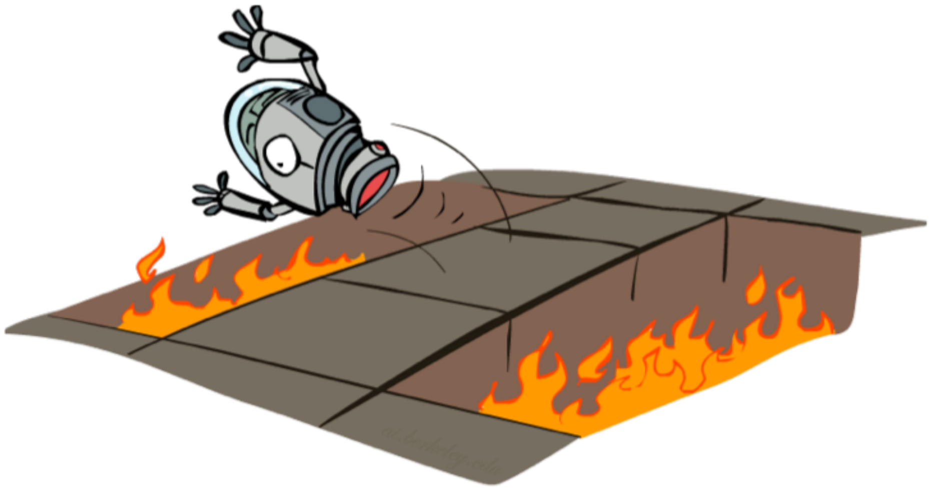



*If B and E both go to C under this policy, how can their values be different?*

*E.g., B=at home, study hard*
*E=at library, study hard*
*C=know material, go to exam*

# Temporal difference (TD) learning

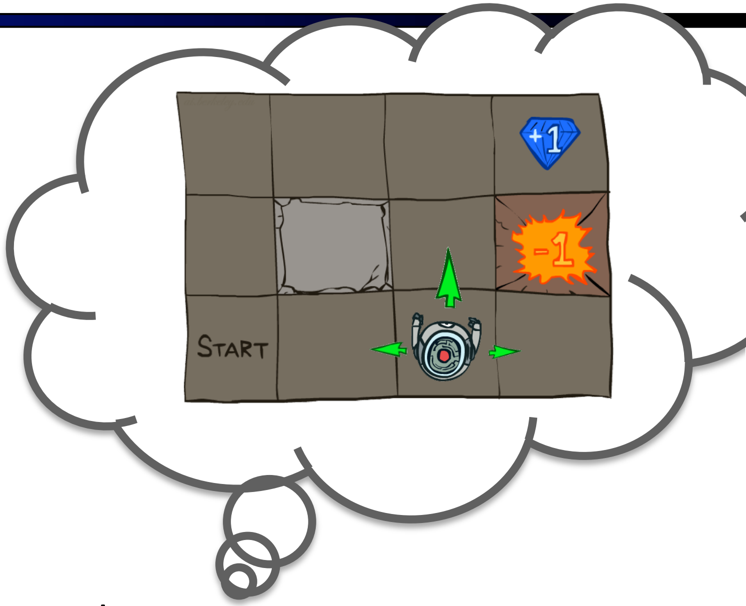




State s is known to be very bad now;
Propagate this information to
connected states.

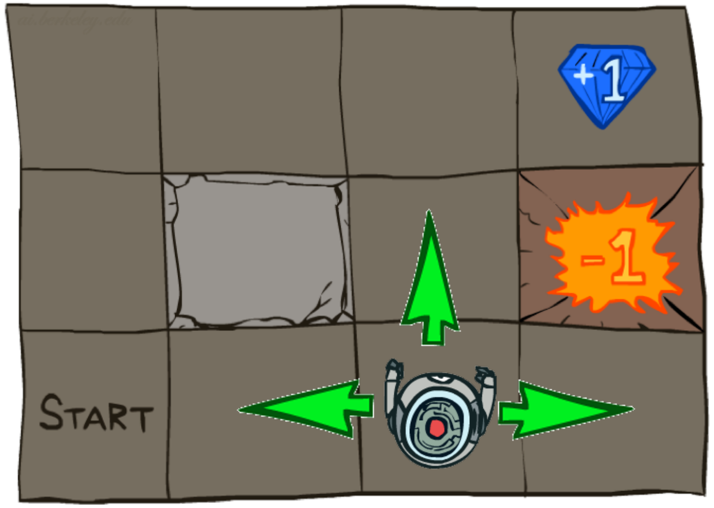
# TD as approximate Bellman update

- Given a fixed policy, the value of a state is:
  - $V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s') [R(s,\pi(s),s') + \gamma V^{\pi}(s')]$
  - TD learning: approximate this using samples!




- Idea 1: Use actual samples to estimate the expectation:
  - $\text{sample}_1 = R(s,\pi(s),s_1') + \gamma V^{\pi}(s_1')$
  - $\text{sample}_2 = R(s,\pi(s),s_2') + \gamma V^{\pi}(s_2')$
  - …
  - $\text{sample}_N = R(s,\pi(s),s_N') + \gamma V^{\pi}(s_N')$
  - $V^{\pi}(s) \leftarrow 1/N \sum_i \text{sample}_i$
  - Need to reset the robot after every transition!

Grid-world example:

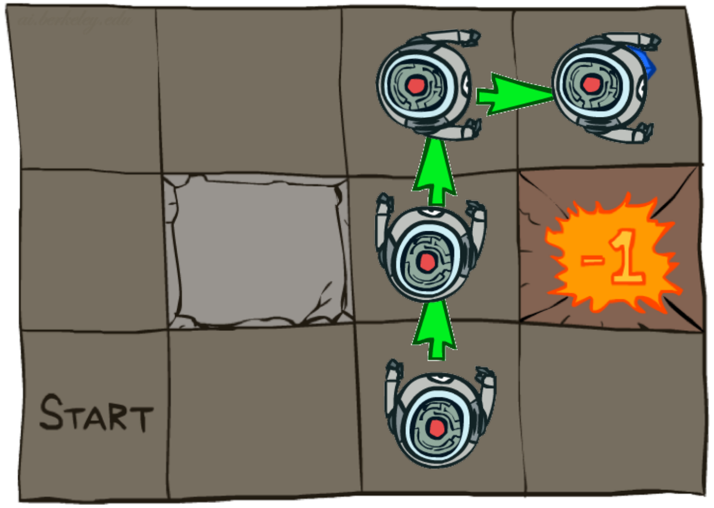


# TD as approximate Bellman update

- Idea 2: Update value of s after seeing each transition s,a,s',r :

- Update $V^{\pi}([3,1])$ based on $R([3,1],up,[3,2])$ and $\gamma V^{\pi}([3,2])$
- Update $V^{\pi}([3,2])$ based on $R([3,2],up,[3,3])$ and $\gamma V^{\pi}([3,3])$
- Update $V^{\pi}([3,3])$ based on $R([3,3],right,[4,3])$ and $\gamma V^{\pi}([4,3])$

- No need to reset the robot now!
- But how to perform the update?



# TD as approximate Bellman update

- Idea 3: Update values by maintaining a ***running average***

# Running averages

- How do you compute the average of 1, 4, 7?
- Method 1: add them up and divide by N
  - $1+4+7 = 12$
  - average $= 12/N = 12/3 = 4$
- Method 2: keep a running average $\mu_n$ and a running count $n$
  - $n=0 \quad \mu_0=0$
  - $n=1 \quad \mu_1 = (0 \cdot \mu_0 + x_1)/1 = (0 \cdot 0 + 1)/1 = 1$
  - $n=2 \quad \mu_2 = (1 \cdot \mu_1 + x_2)/2 = (1 \cdot 1 + 4)/2 = 2.5$
  - $n=3 \quad \mu_3 = (2 \cdot \mu_2 + x_3)/3 = (2 \cdot 2.5 + 7)/3 = 4$
  - General formula: $\mu_n = ((n-1) \cdot \mu_{n-1} + x_n)/n$
  - $= [(n-1)/n]\, \mu_{n-1} + [1/n]\, x_n$ (weighted average of old mean, new sample)

# Running averages contd.

- What if we use a weighted average with a fixed weight?
  - $\mu_n = (1-\alpha)\,\mu_{n-1} + \alpha\, x_n$
  - n=1 $\quad \mu_1 = x_1$
  - n=2 $\quad \mu_2 = (1-\alpha)\cdot\mu_1 + \alpha x_2 = (1-\alpha)\cdot x_1 + \alpha x_2$
  - n=3 $\quad \mu_3 = (1-\alpha)\cdot\mu_2 + \alpha x_3 = (1-\alpha)^2\cdot x_1 + \alpha(1-\alpha)x_2 + \alpha x_3$
  - n=4 $\quad \mu_4 = (1-\alpha)\cdot\mu_3 + \alpha x_4 = (1-\alpha)^3\cdot x_1 + \alpha(1-\alpha)^2 x_2 + \alpha(1-\alpha)x_3 + \alpha x_4$
- I.e., ***exponential forgetting*** of old values
- Running average with fixed weight is still ***unbiased***!
  - $\mu_n$ is a convex combination of sample values (weights sum to 1)
  - $E[\mu_n]$ is also a convex combination of $E[X_i]$ values
  - Know $E[X_i]=E[X]$, so $E[\mu_n]=E[X]$, hence unbiased

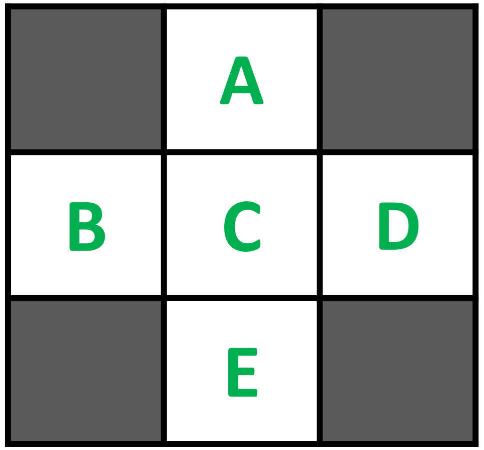
# TD as approximate Bellman update

- Idea 3: Update values by maintaining a ***running average***
  - sample = $R(s,\pi(s),s') + \gamma V^{\pi}(s')$
  - $V^{\pi}(s) \leftarrow (1-\alpha) \cdot V^{\pi}(s) + \alpha \cdot$ sample *(running average)*
  - $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha \cdot [\text{sample} - V^{\pi}(s)]$ *(rewritten)*

- This is the ***temporal difference learning rule***
  - $[\text{sample} - V^{\pi}(s)]$ is the “TD error”
  - $\alpha$ is the ***learning rate***
  - Intuitively:
    - observing a sample,
    - move $V^{\pi}(s)$ a little bit to make it more consistent with the new sample,
    - thus with its neighbor $V^{\pi}(s')$

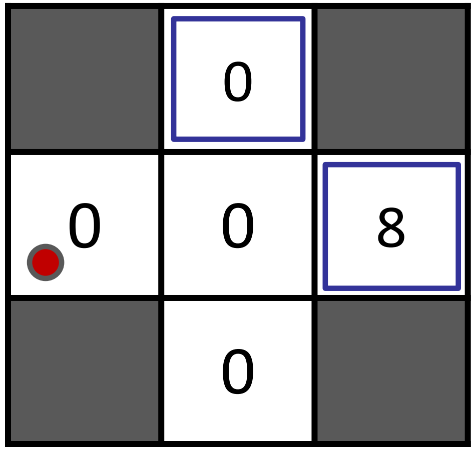
# Example: Temporal Difference Learning

## States

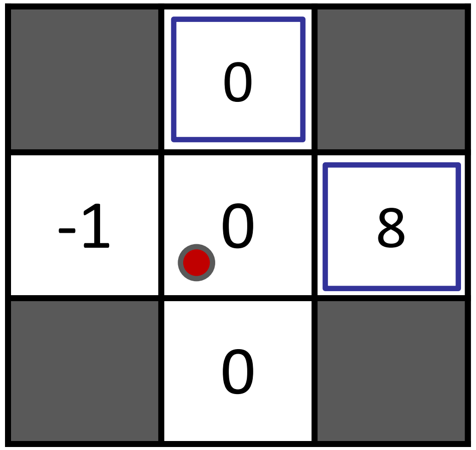



*Assume:* $\gamma = 1$, $\alpha = 1/2$

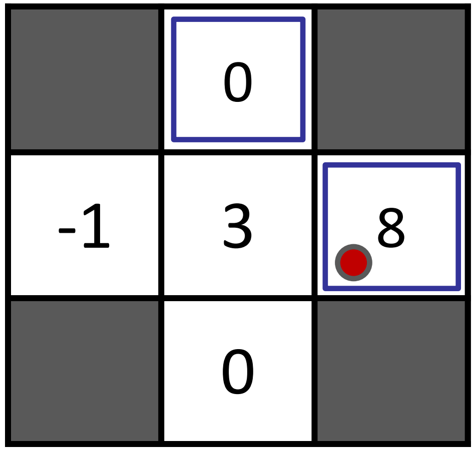
## Observed Transitions




**B**, east, **C**, -2




**C**, east, **D**, -2




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

# Problems with TD Value Learning

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
  - An efficient way to learn policy values
- But we can’t use the value function or extract the policy from it.
  - Policy extraction requires transition model to do one-step expectimax!
- TD learning can not be used alone!

# Q-learning as approximate Q-iteration

- Recall the definition of Q values:
  - $Q^*(s,a)$ = expected utility of taking a in s and behaving optimally thereafter
- Bellman equation for Q values:
  - $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$
- Approximate Bellman update for Q values:
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a')]$
- We obtain a policy from learned Q(s,a), with no model!
  - $\pi^*(s) = \text{argmax}_a Q^*(s,a)$
  - (No free lunch: Q(s,a) table is |A| times bigger than V(s) table)

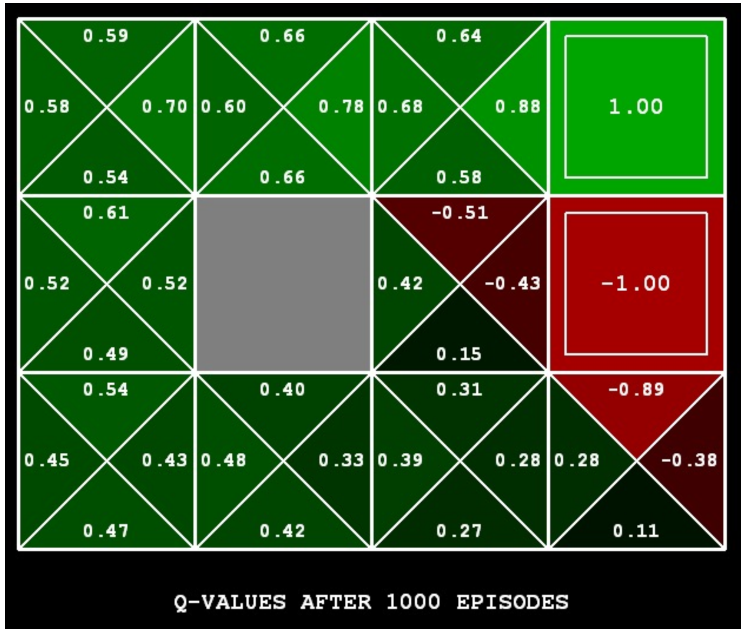
# Q-Learning Algorithm

- 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:

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

  - Incorporate the new estimate into a running average:

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



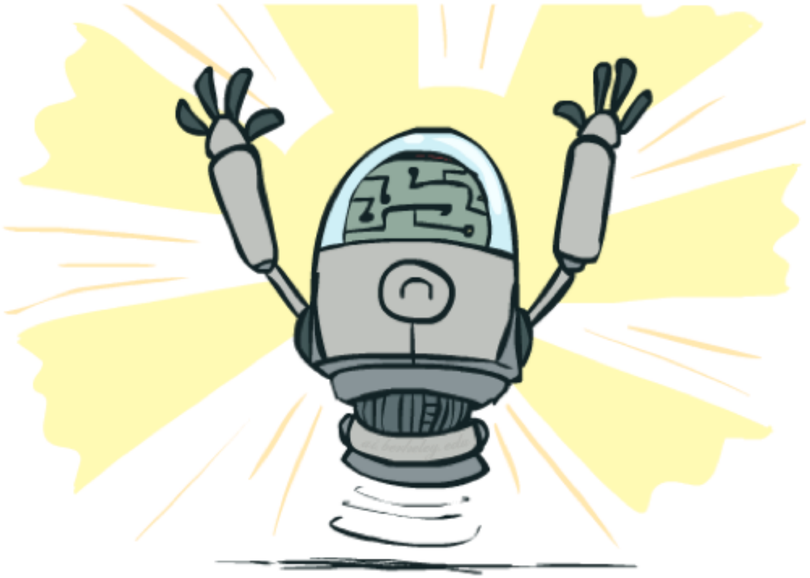

[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

# Video of Demo Q-Learning -- Gridworld

# Video of Demo Q-Learning -- Crawler

# Q-Learning Properties

- Theoretical guarantee: Q-learning converges to optimal policy -- even if samples are generated from a suboptimal policy!
- This is called ***off-policy learning***
- Caveats:
  - You have to explore sufficiently
    - Eventually try every state/action pair infinitely often
  - You have to decrease the learning rate appropriately
    - Requirements: $\sum_t \alpha(t) = \infty$ , $\sum_t \alpha^2(t) < \infty$
    - Satisfied by: $\alpha(t) = 1/t$ or (better) $\alpha(t) = K/(K+t)$



# Summary

- RL solves MDPs via direct experience of transitions and rewards
- There are several approaches:
  - Learn the MDP model and solve it
  - Learn V directly from sums of rewards, or by TD updates
    - Still need a model to make decisions by lookahead
  - Learn Q by local Q updates
    - Can directly pick actions
- Big missing pieces:
  - How to explore without too much regret?
  - How to scale this up to Tetris ($10^{60}$), Go ($10^{172}$), StarCraft ($|A|=10^{26}$)?