CS 232: Artificial Intelligence

Fall 2023

Prof. Carolyn Anderson
Wellesley College

Recap

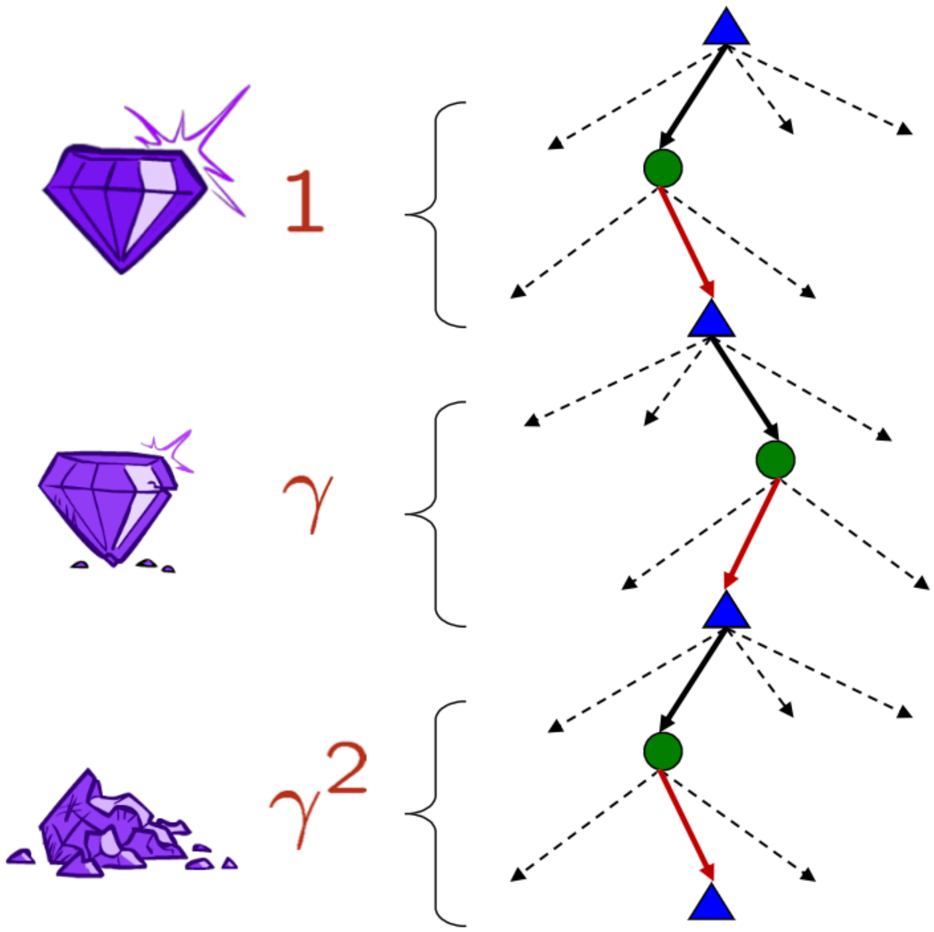
Discounting

How to discount?

- Each time we descend a level, we multiply in the discount once

Why discount?

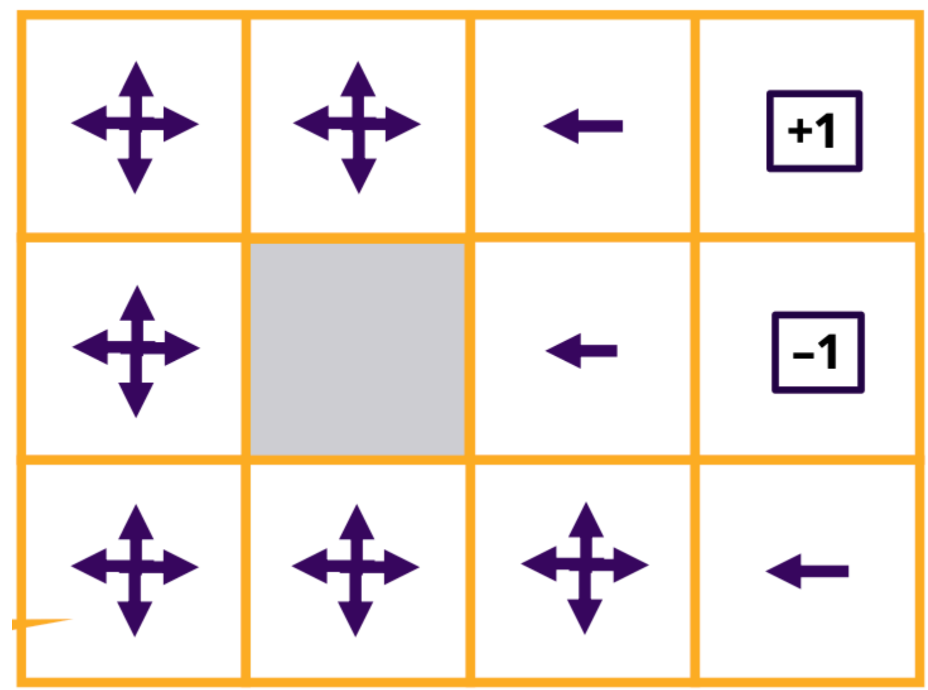
- Sooner rewards probably do have higher utility than later rewards
- Also helps our algorithms converge



Sequences of Rewards

The performance of an agent in an MDP is the sum of the rewards for the transitions it takes.

r > 0



$$Uh([s_0,a_0,s_1,a_1,\ldots, s_n]) =$$

$$R(s_0,a_0,s_1) + R(s_1,a_1,s_2) + \ldots + R(s_{n-1}, a_{n-1},s_n)$$

Slides adapted from Chris Callison-Burch

Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?

- Finite horizon (similar to depth-limited search)

- Discounting: use $0 < \gamma < 1$

  smaller γ means smaller horizon

  $$U[r_0 \ldots r_\infty] = \sum_{i=0}^{\infty} \gamma^t r_t \leq R_{max}/(1-\gamma)$$

- Terminal/Absorbing state

Recap: Defining MDPs

Markov decision processes:

- Set of states S
- Start state s_0
- Set of actions A
- Transitions $P(s'|s,a)$ (or $T(s,a,s')$)
- Rewards $R(s,a,s')$ (and discount γ)

probability returned

function notation

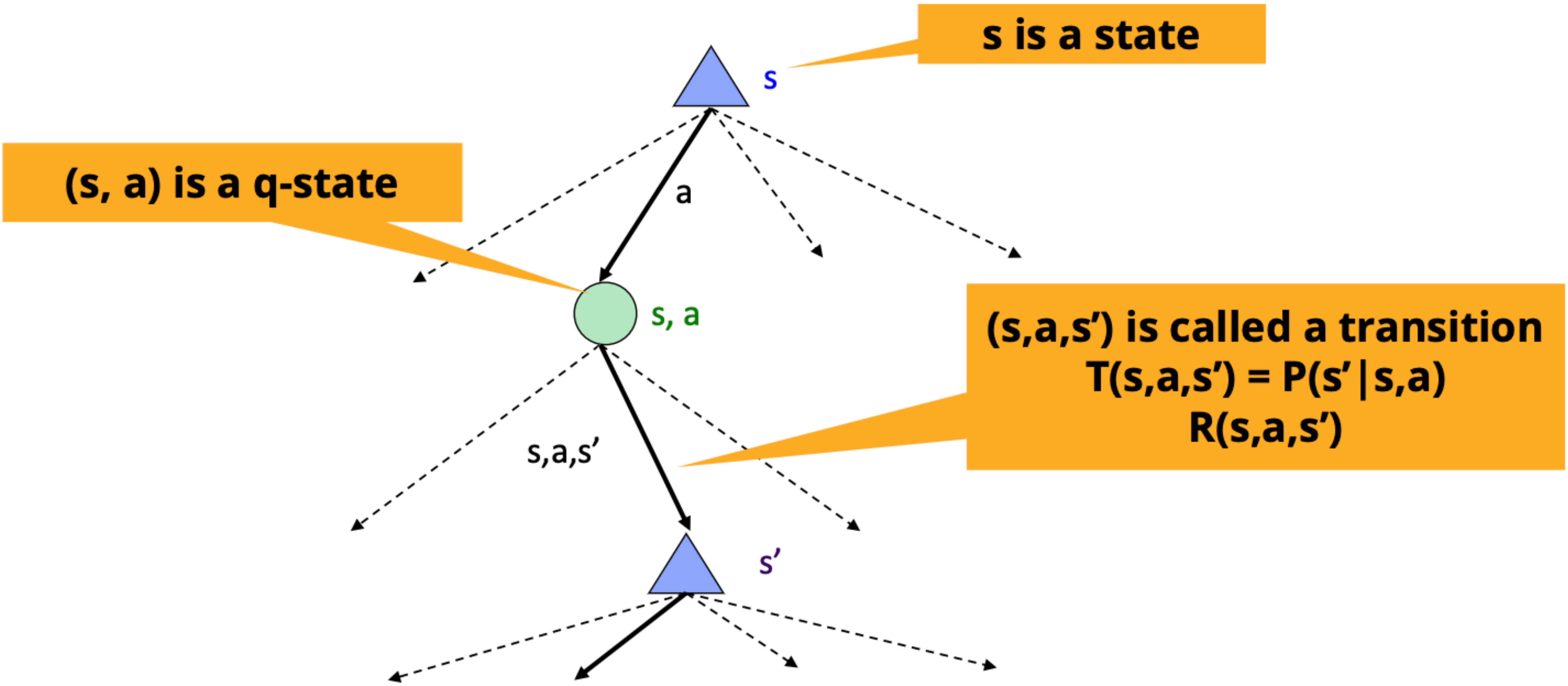
MDP quantities so far:

- Policy = Choice of action for each state
- Utility = sum of (discounted) rewards

Q-values

Q-values

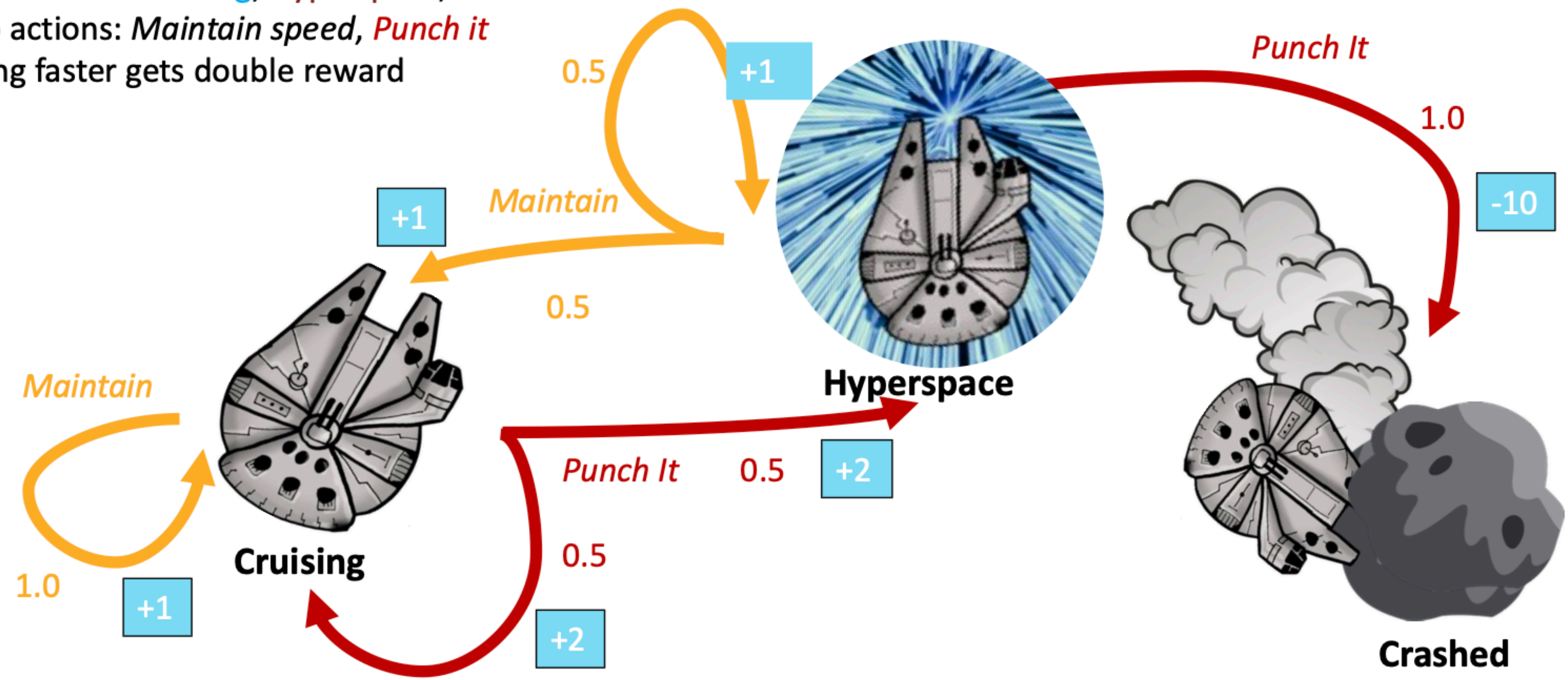
A q-state is a pair of a state and an action.



Slides adapted from Chris Callison-Burch

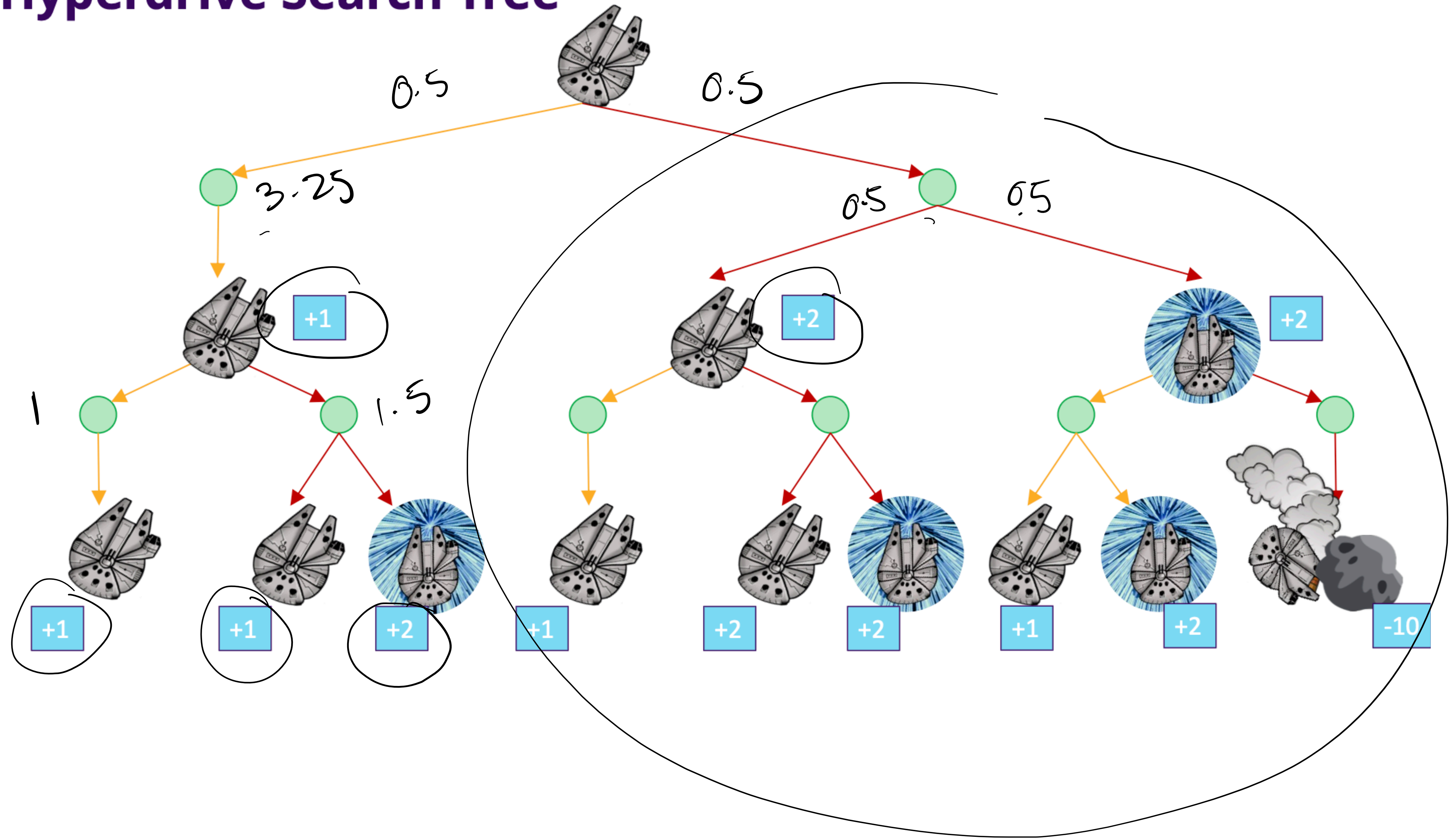
Example Hyperdrive MDP

The Millennium Falcon needs to travel far far away, quickly
Three states: Cruising, Hyperspace, Crashed
Two actions: *Maintain speed*, *Punch it*
Going faster gets double reward



Slides adapted from Chris Callison-Burch

Hyperdrive Search Tree



Slides adapted from Chris Callison-Burch

Optimal Quantities

- The value (utility) of a state s:

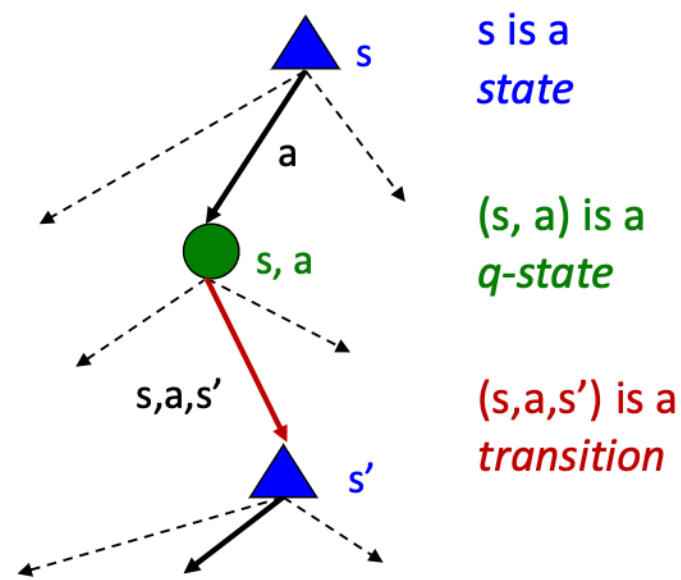
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s & acting optimally afterwards

- The optimal policy:

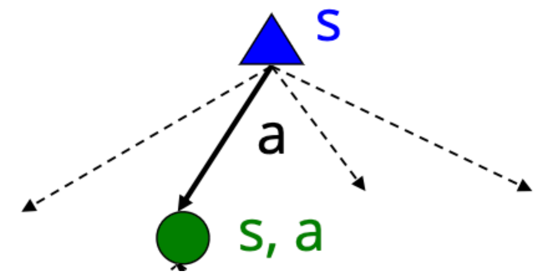
  $\pi^*(s)$ = optimal action from state s



Values of States

Fundamental operation: compute the (expectimax) value of a state

- Expected utility under optimal action
- Average sum of (discounted) rewards

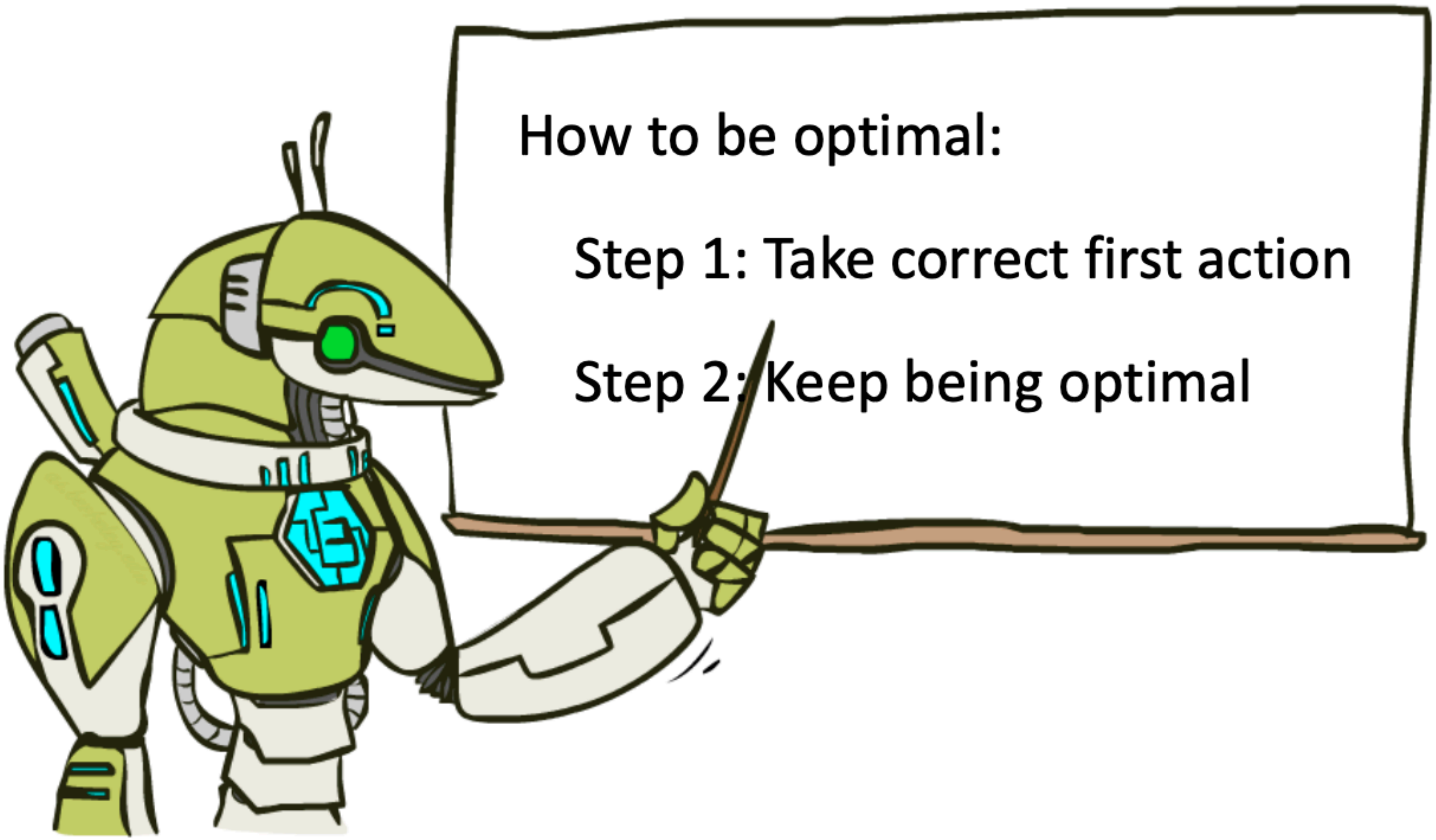


Recursive definitions:

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

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

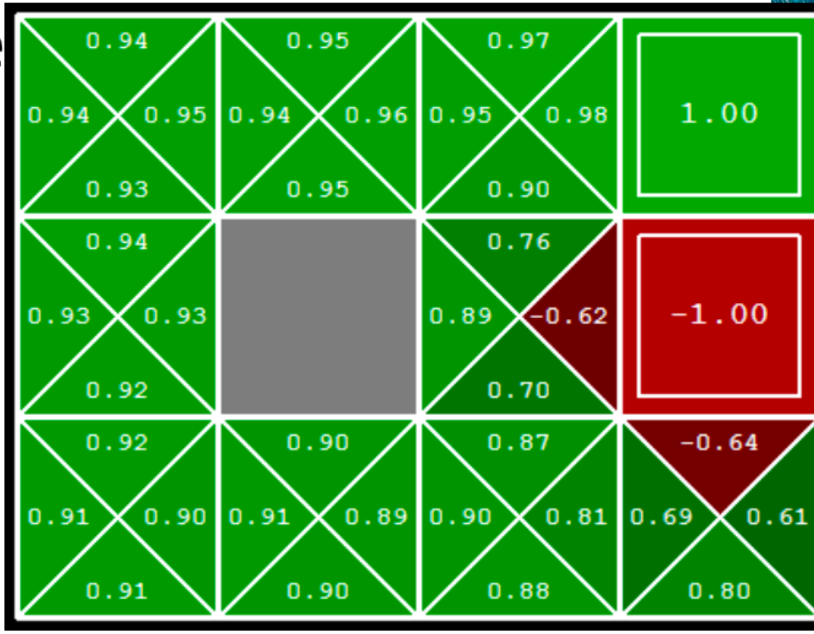
$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$



Slides adapted from Chris Callison-Burch

Computing Actions from Q-Values

Let's imagine we have the optimal q-values



How should we act?

- Completely trivial to decide!

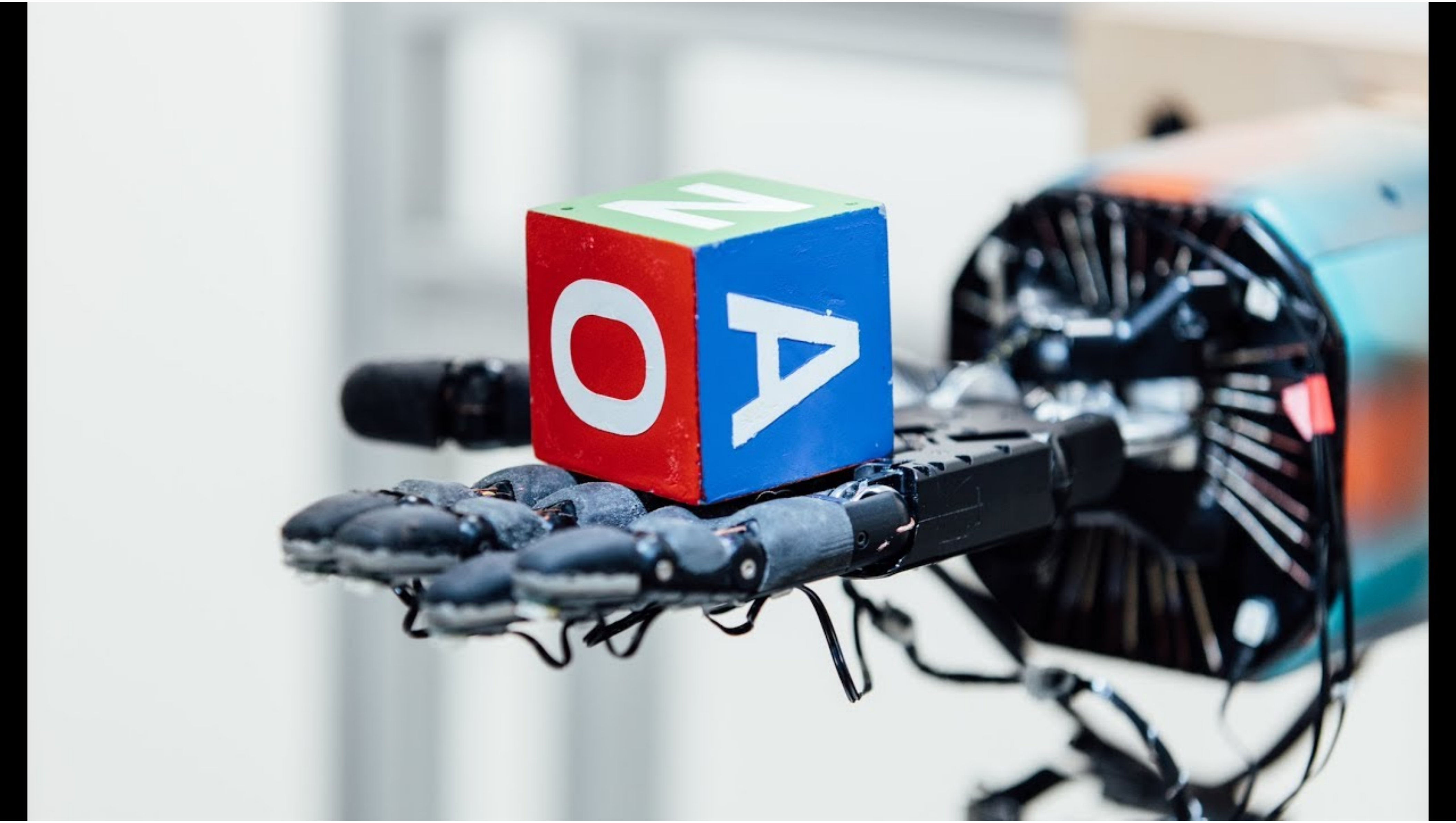
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

Important lesson: actions are easier to select from q-values than values!

Slides adapted from Chris Callison-Burch

Reinforcement Learning

Exploitation versus Exploration


N
O
A

Offline Planning

- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

New Scenario: Unknown States and Rewards

Which of these slot machines should we play?

Let's play a while and find out!

$ 3/5

Bet:	Reward:
$0.8	$1
$0.8	$0
$0.8	$1
$0.8	$1
$0.8	$0

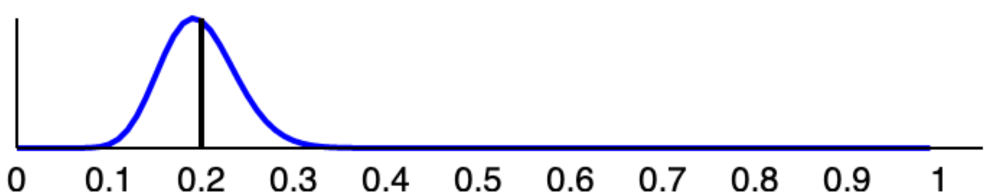
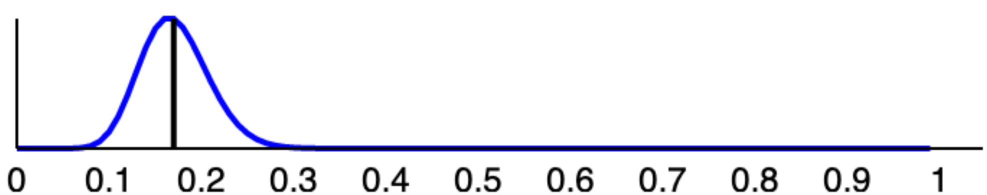
Bet:	Reward:
$0.8	$0
$0.8	$0
$0.8	$0
$0.8	$2
$0.8	$0

$ 2/5

What Did We Learn?

First of all, gambling is a bad idea.

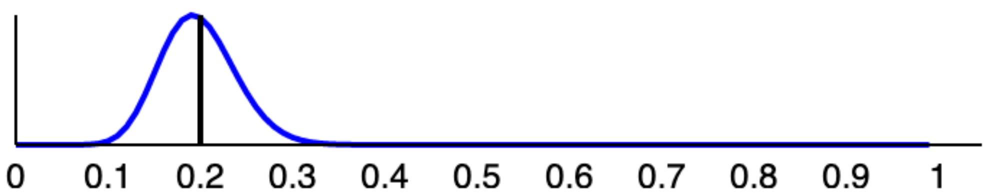
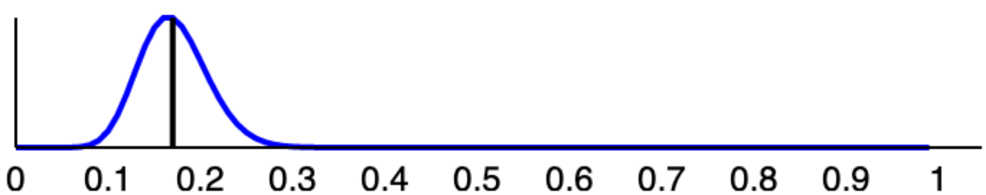
Second, it looks like Slot A has slightly better rewards.

Posterior Distribution *Black bar is the bandit's actual probability of success*	**Hits**	**Misses**	**Total Pulls**
	18 (19%)	76	94
	17 (17%)	86	103

What Did We Learn?

Wait, why are there more pulls for Slot A?

Our player favored **previously successful** actions. But some percent of the time, our player picked randomly to **gain experience** with all the slots.

Posterior Distribution *Black bar is the bandit's actual probability of success*	**Hits**	**Misses**	**Total Pulls**
 	18 (19%)	76	94
 	17 (17%)	86	103

What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out
- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP

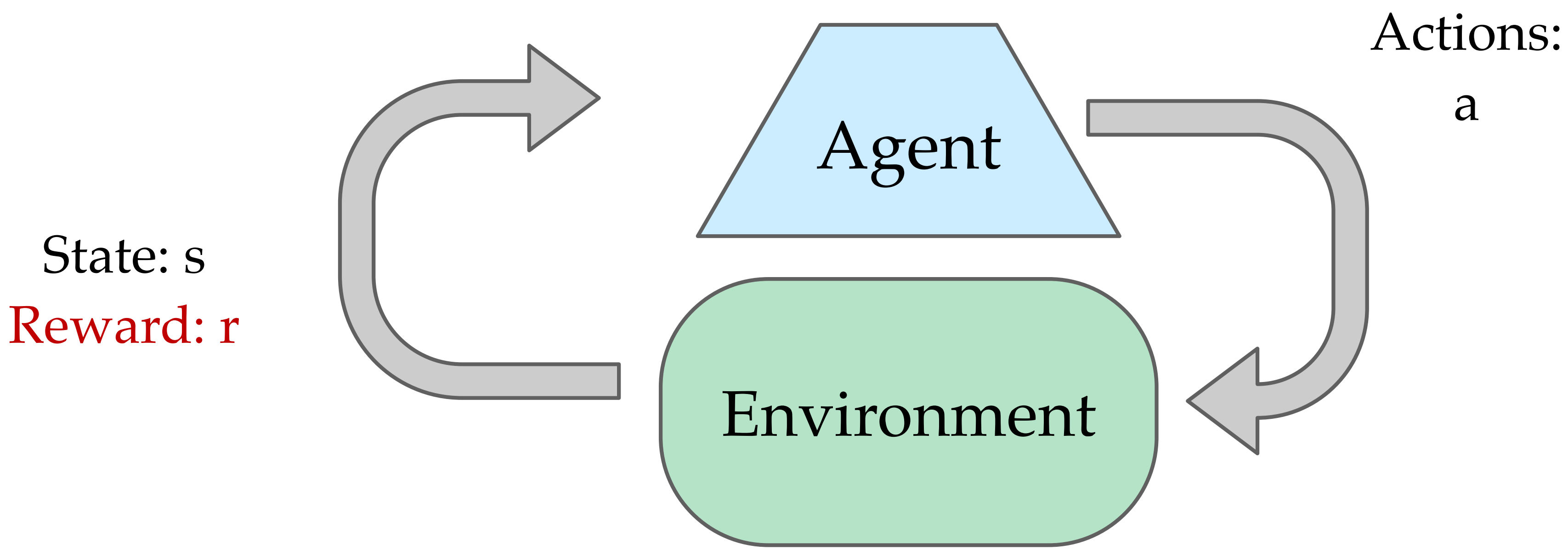
Reinforcement Learning

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

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

Reinforcement Learning

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



Example: Learning to Walk

Initial

A Learning Trial

After Learning
[1K Trials]

[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

Example: Learning to Walk

Training

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – training]

Example: Learning to Walk

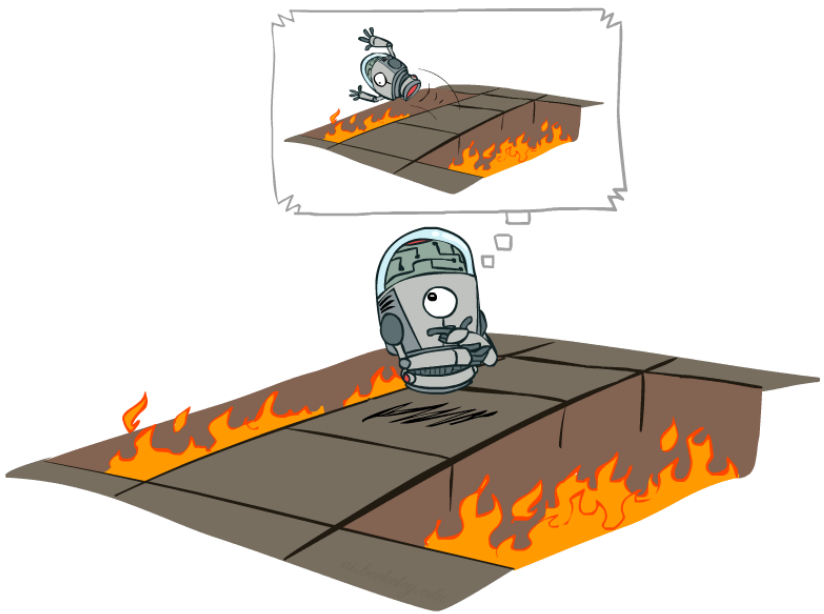
Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

Video of Demo Crawler Bot

Offline (MDPs) vs. Online (RL)

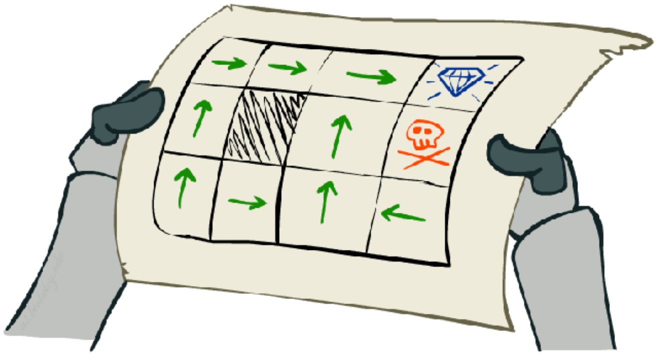


Offline
Solution

Online
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



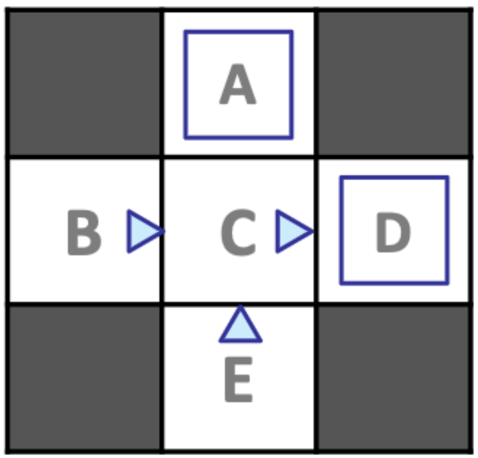
- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

Direct Evaluation

- Goal: Compute values for each state under π
- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples
- This is called direct evaluation

Example: Direct Evaluation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

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

Episode 2

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

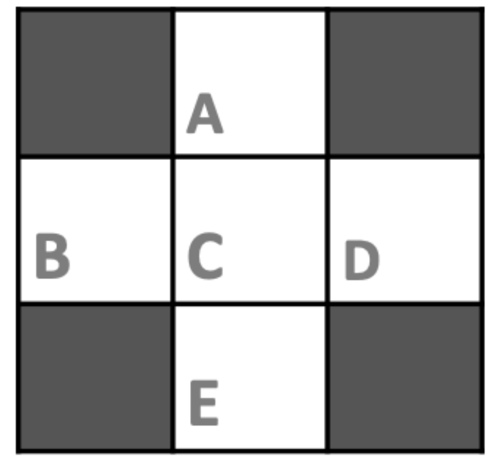
Episode 3

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



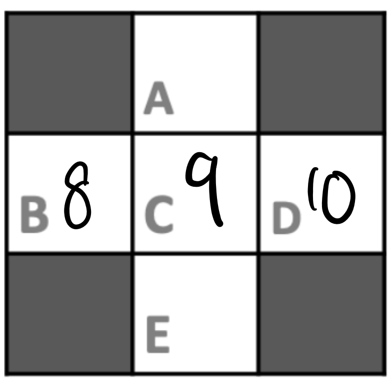
Example: Direct Evaluation

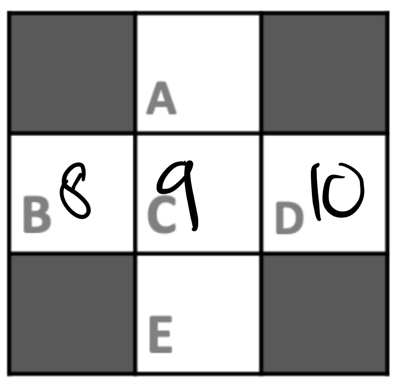
Episode 1	Episode 2	Episode 3	Episode 4
B, east, C, -1 C, east, D, -1 D, exit, x, +10	B, east, C, -1 C, east, D, -1 D, exit, x, +10	E, north, C, -1 C, east, D, -1 D, exit, x, +10	E, north, C, -1 C, east, A, -1 A, exit, x, -10

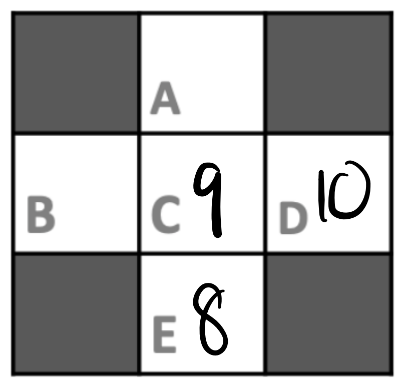
$B = -1 + \gamma -1 + \gamma 10$
$= 8$
$C = -1 + \gamma 10$

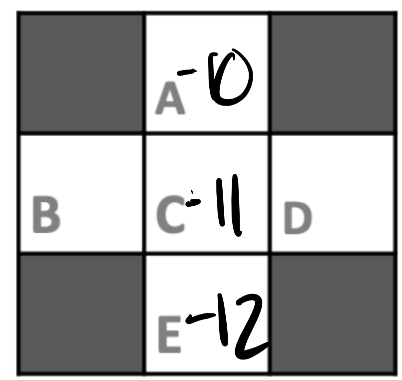
$E = -1 + \gamma -1 + \gamma 10$
$C = -1 + \gamma 10$

$E = -1 + \gamma -1 + \gamma -10$
$C = -1 + \gamma -10$





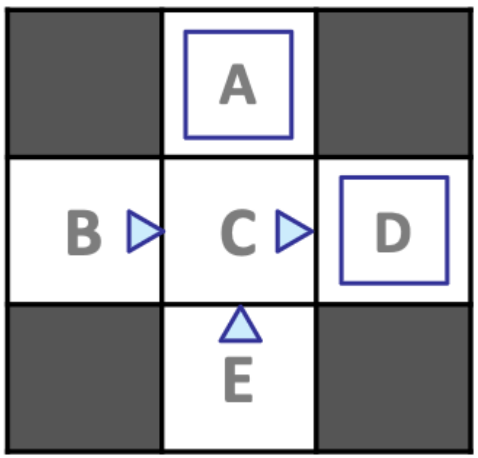




Assume $\gamma = 1$

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

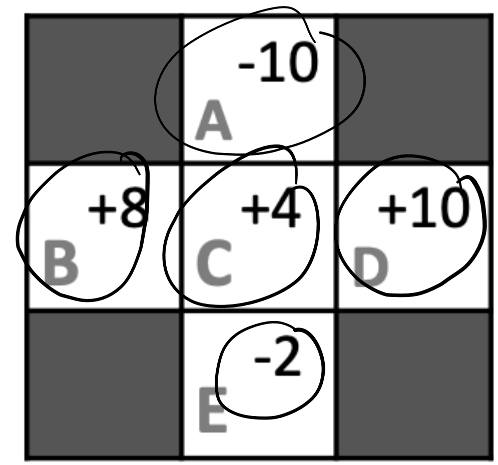
Episode 3

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

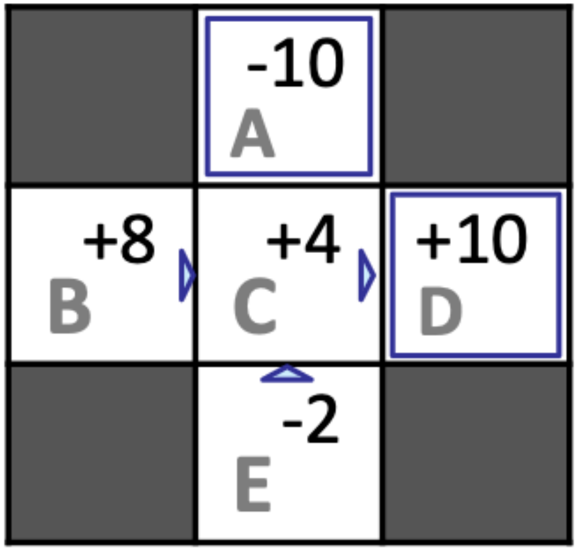
Output Values



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?

Temporal Difference Learning

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

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

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

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

Exponential Moving Average

- Exponential moving average
  - The running interpolation update:
  - Makes recent samples more important: $\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$

$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots}$$

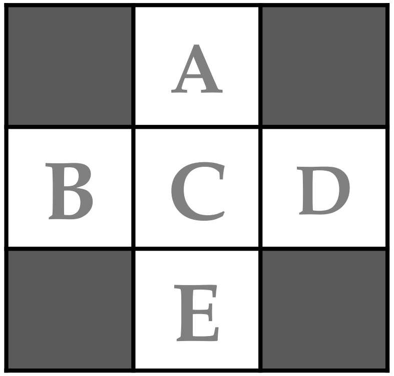
  - Forgets about the past (distant past values were wrong anyway)

- Decreasing learning rate (alpha) can give converging averages

α = learning rate

Example: Temporal Difference Learning

States



Assume:

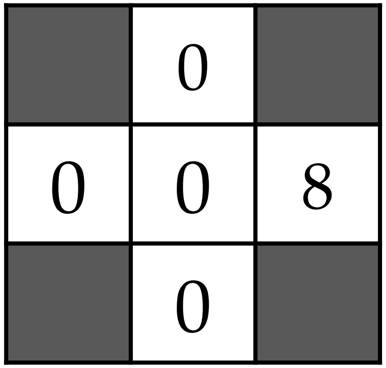
$\gamma = 1,$

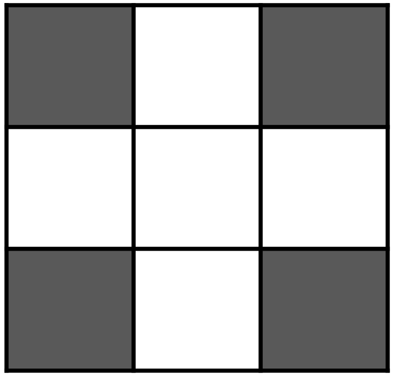
$\alpha = 1/2$

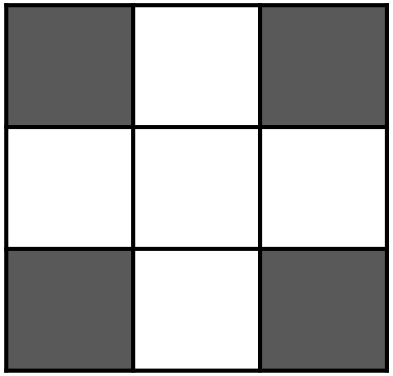
Observed Transitions

B, east, C, -2

C, east, D, -2







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

Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:

$\gamma = 1,$

$\alpha = 1/2$

Observed Transition:

B, east, C, -2

	0	
-1 ~~0~~	0	8
	0	

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

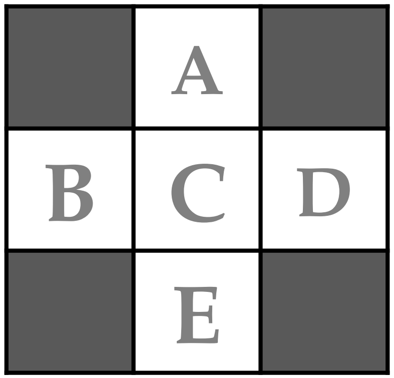
$$V^\pi(B) \leftarrow (1-0.5)\,0 + 0.5[-2 + 1\cdot 0]$$

$$\leftarrow -0.5\cdot 0 + 0.5(-2)$$

$$\leftarrow \underline{-1}$$

Example: Temporal Difference Learning

States



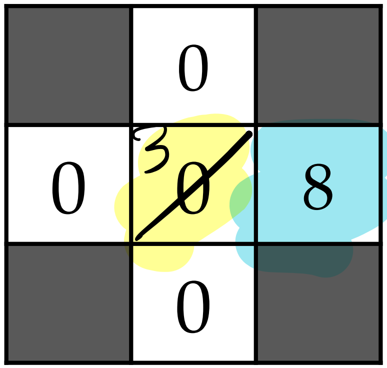
Assume:

$\gamma = 1,$

$\alpha = 1/2$

Observed Transition:

C, east, D, -2



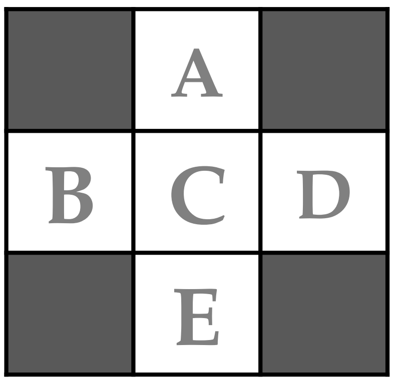
$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

$$\leftarrow 0.5[0] + 0.5[-2+8]$$

$$\leftarrow 3$$

Example: Temporal Difference Learning

States



Assume:

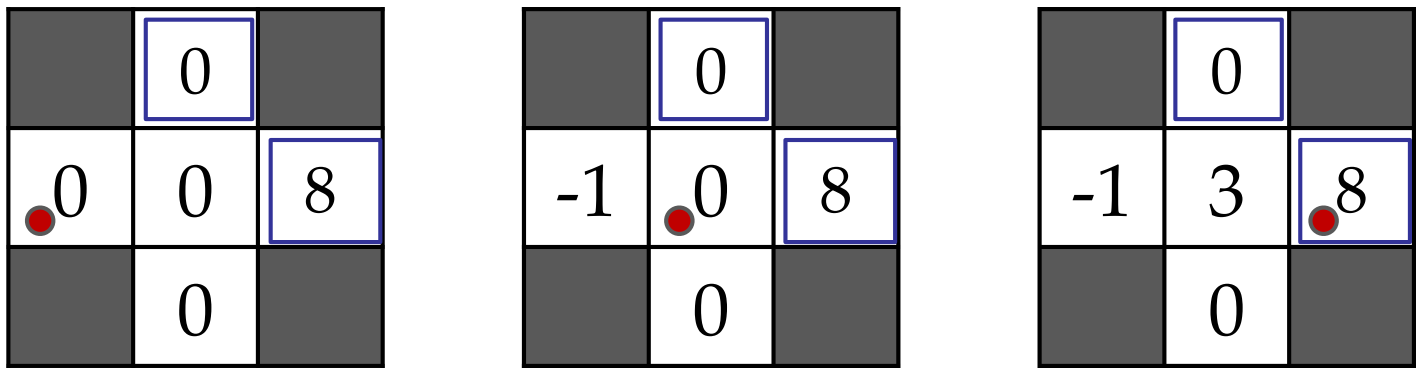
$\gamma = 1,$

$\alpha = 1/2$

Observed Transitions

B, east, C, -2

C, east, D, -2



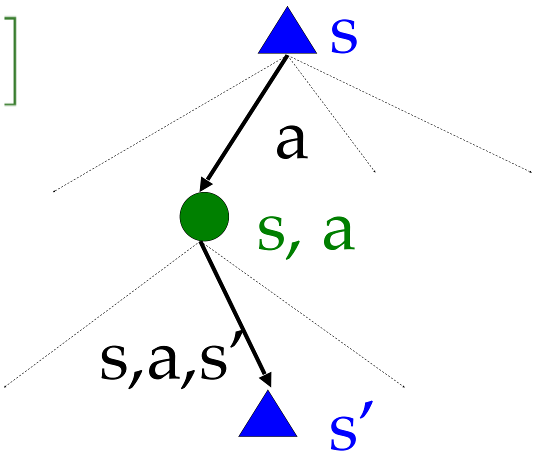
$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation
- However, if we want to turn values into a (new) policy, we're sunk:

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

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



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

Q-Learning

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

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

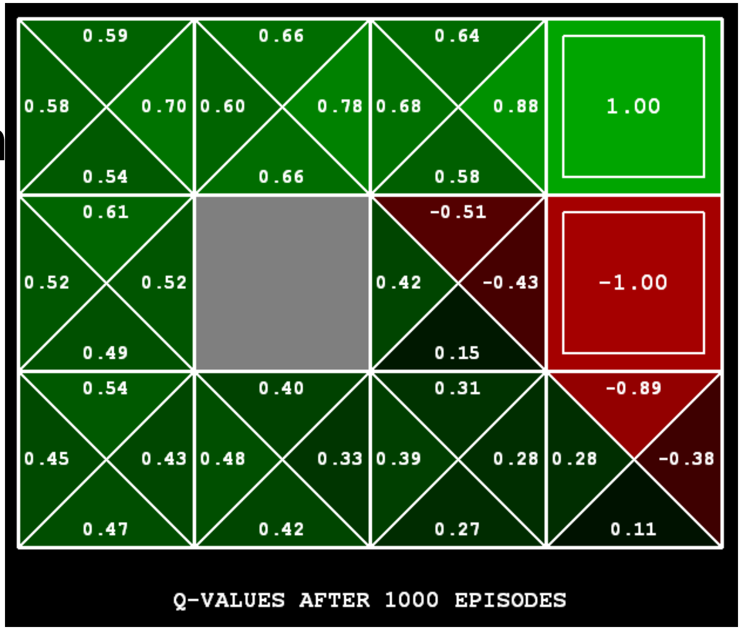
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate:
  - Consider your new sample estimate:

$$Q(s,a)$$

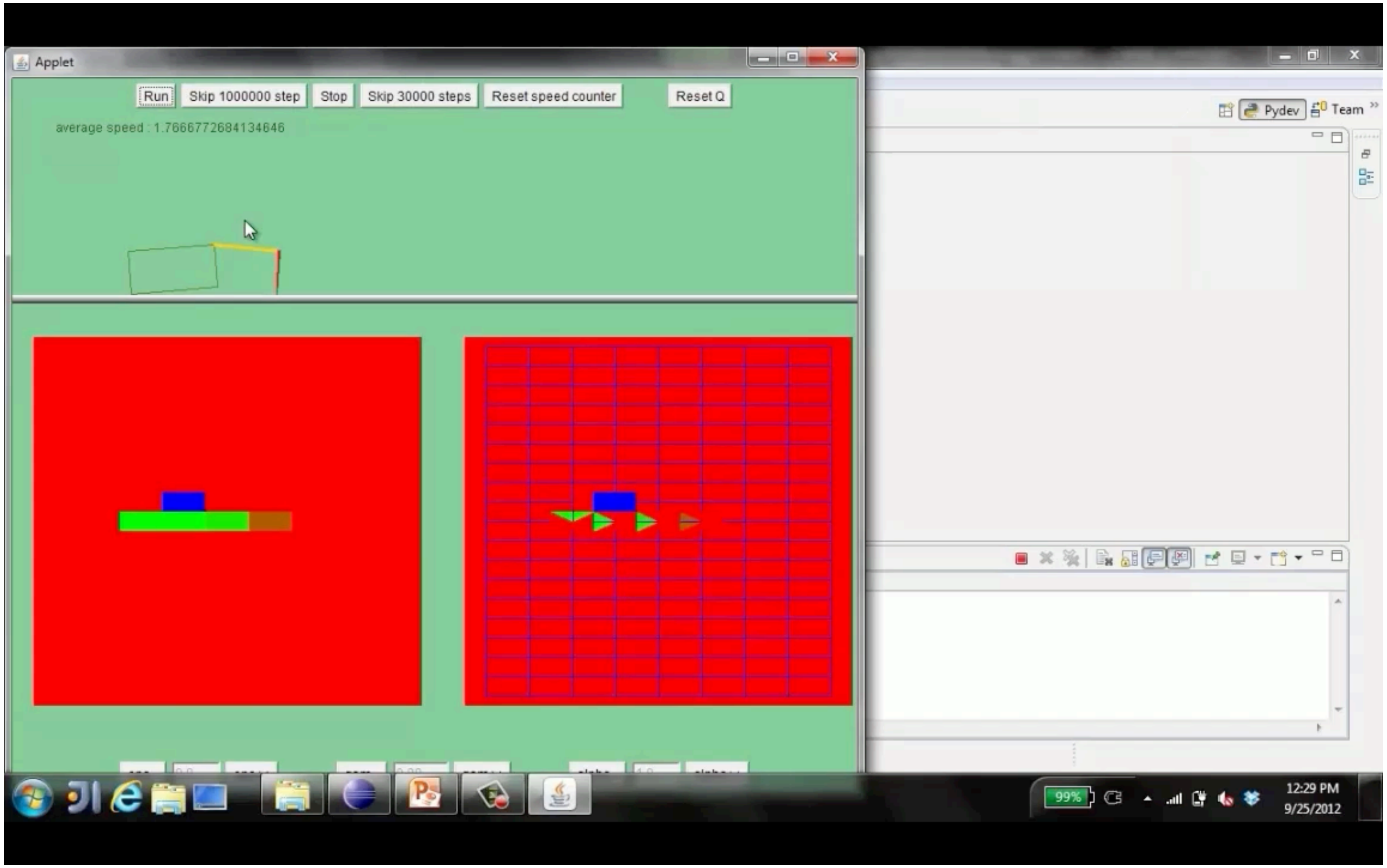
  - Incorporate the new estimate into a runnin

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

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

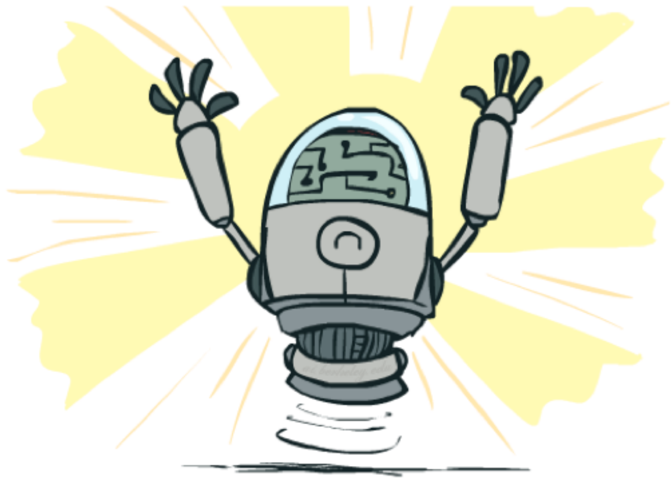


Video of Demo Q-Learning -- Crawler

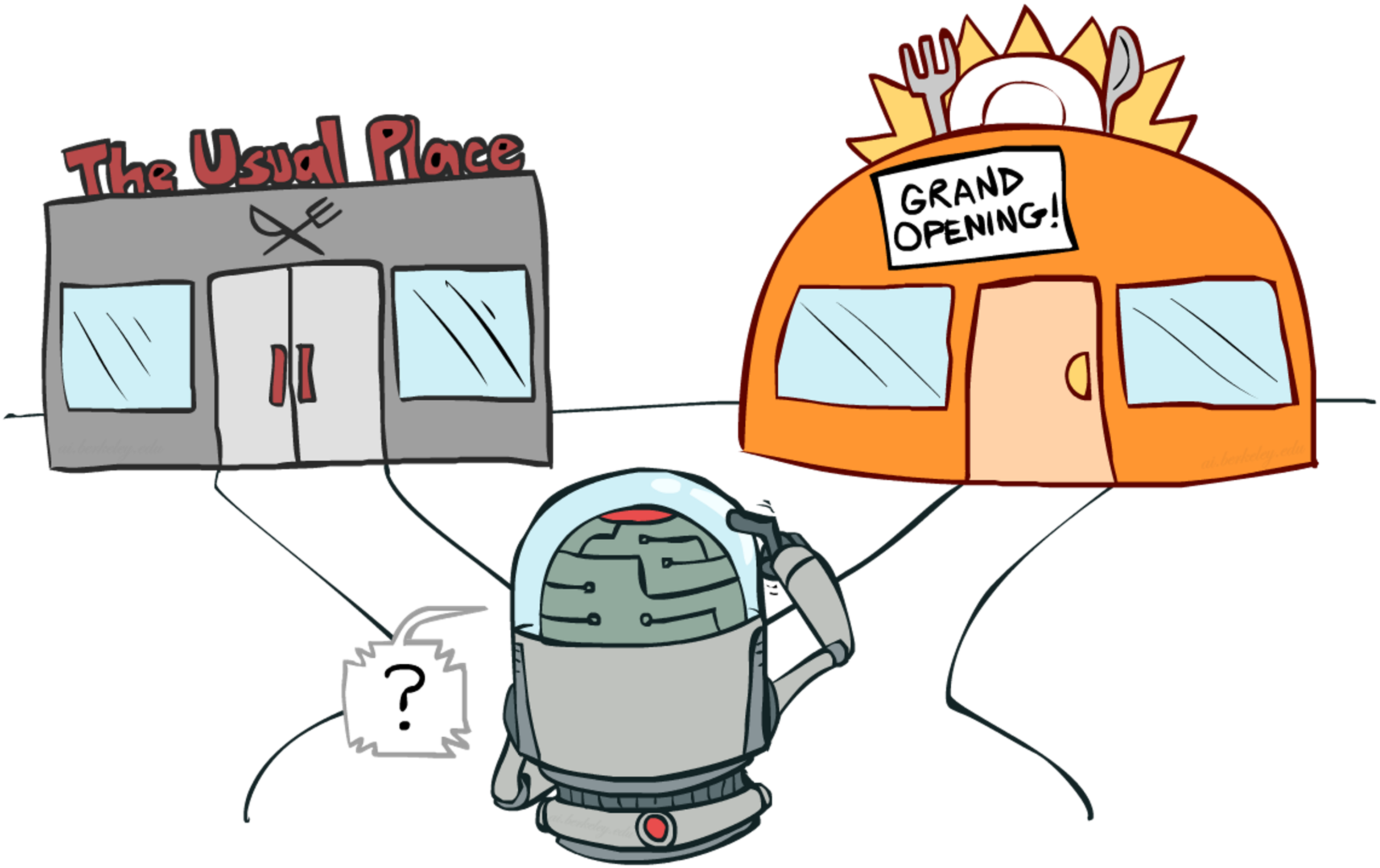


Q-Learning Properties

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



Exploration vs. Exploitation



How to Explore?

Several schemes for forcing exploration

- Simplest: random actions (ε-greedy)
  - Every time step, flip a coin
  - With (small) probability ε, act randomly
  - With (large) probability 1-ε, act on current policy

- Problems with random actions?
  - You do eventually explore the space, but keep thrashing around once learning is done
  - One solution: lower ε over time
  - Another solution: exploration functions

Exploration Functions

When to explore?

Random actions: explore a fixed amount

Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

Exploration function

Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$

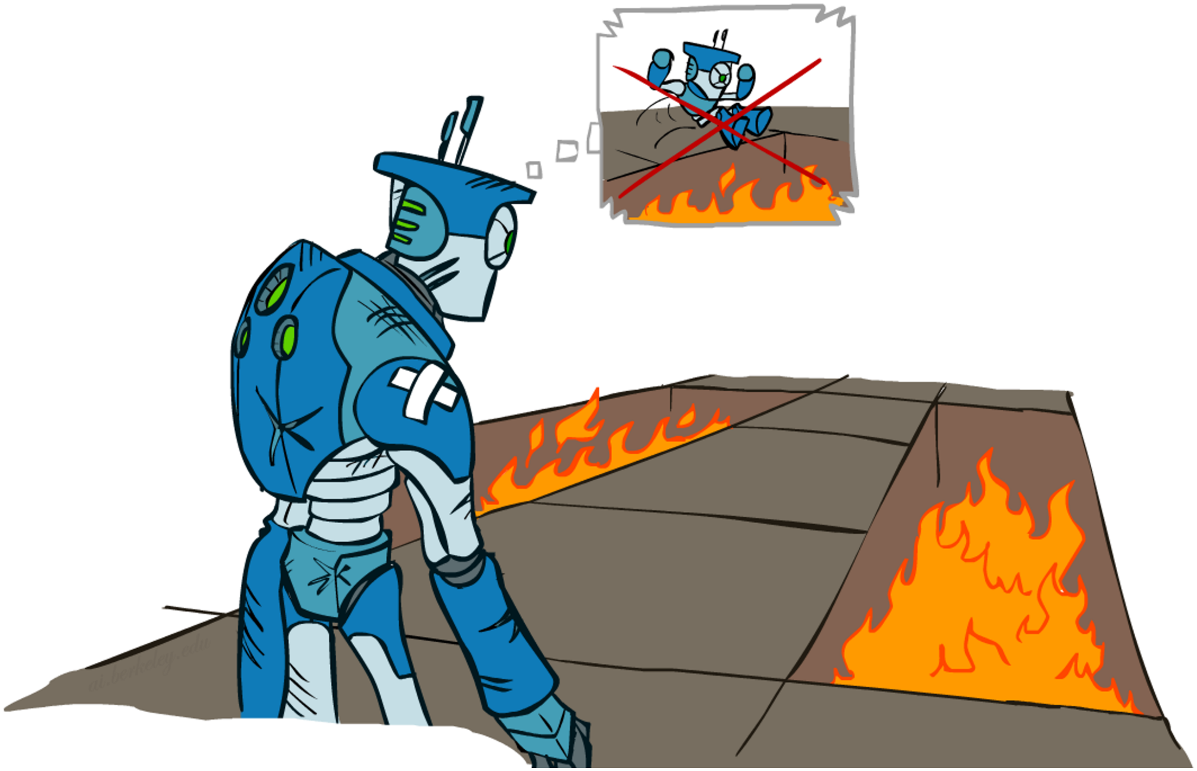
Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

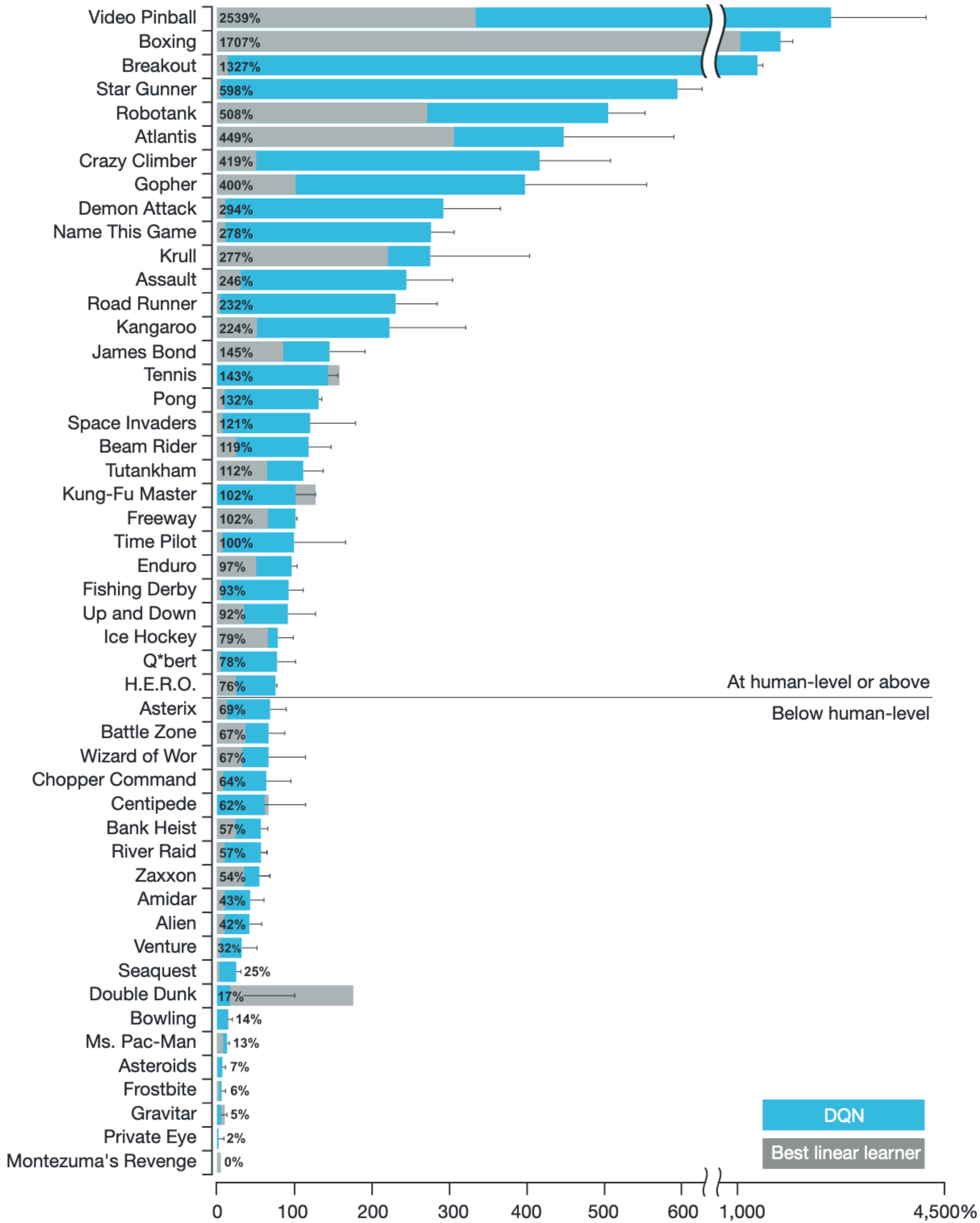
Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

Note: this propagates the "bonus" back to states that lead to unknown states as well!

Regret

- Even if you learn the optimal policy, you still make mistakes along the way
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret





Mnih et al. 2015