Announcements

Election Day tomorrow

If you have a vote, cast your vote!

There is literally no other place where your vote will count more than this.

Announcements

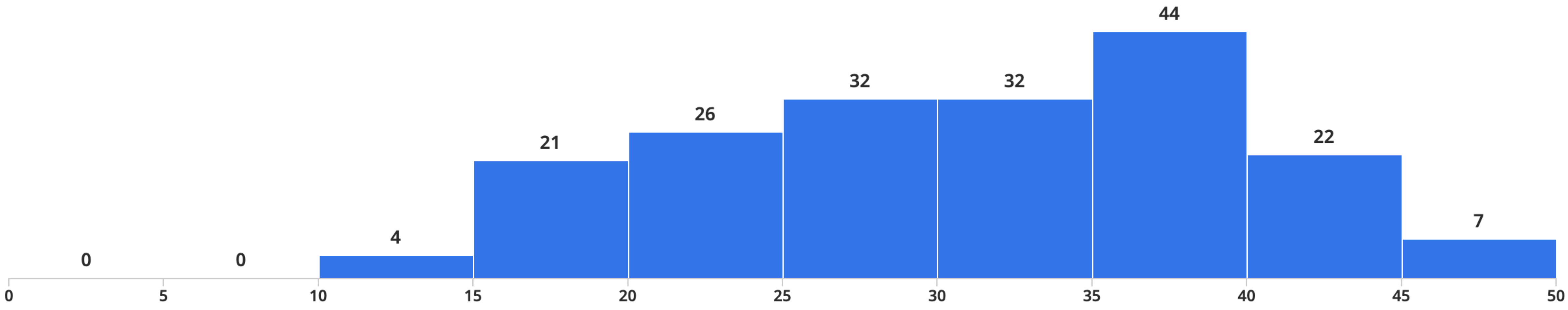
- HW3 due, HW4 out this Wednesday (last HW!)
- Project recitations last week (recordings linked on Ed #380)
    - Today: deadline to tell us your project topics!

Midterm 1 grades distribution

Remember: carries only 15% of points.

Final grade assignments are lenient: 90+: A+, 85+: A, 80+: A-, etc.

- Even if you got 0 points on midterm 1, you're still probably in line for A- if:
  - You have been turning in all your work on time and doing well on HWs.
  - You study hard to make up what you didn't learn.
- Midterm 2 will still also test the first half of class (tentatively, about a third of its points, more details closer to the exam).



CIS 4190/5190: Lec 17 Mon Nov 04, 2024

Reinforcement Learning Part 1

Machine Learning Systems Make Decisions

- For example:
  - A spam classifier might decide whether to place an email in your inbox or spam.
  - ML-based credit scoring in a financial institution might decide whether to approve a loan application.

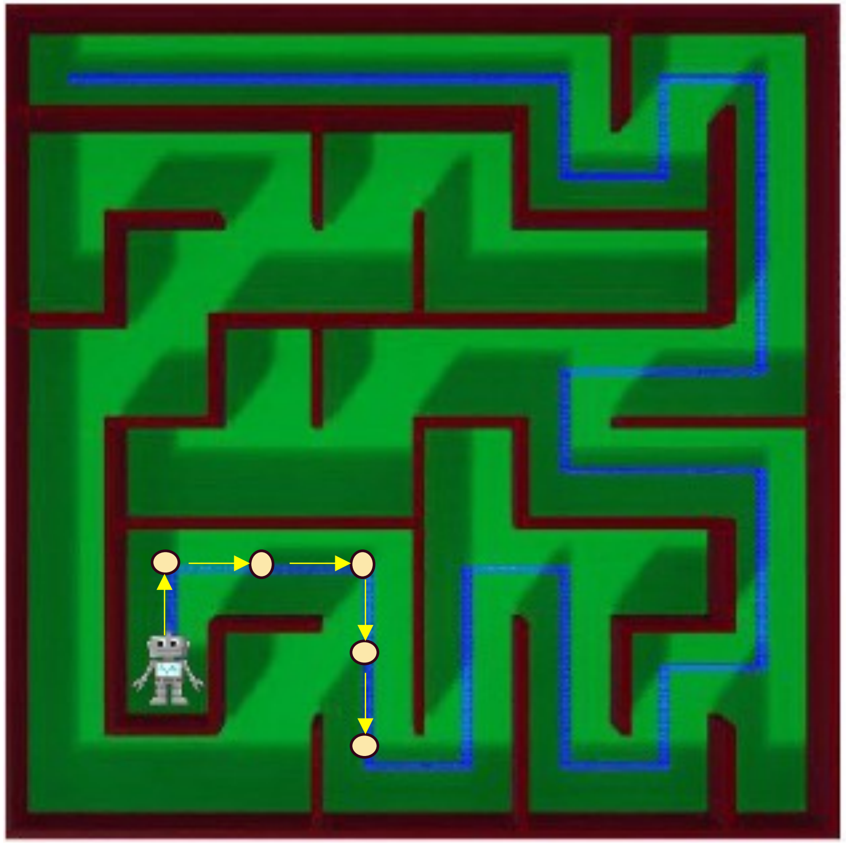
- In these and all the settings we have considered so far, the ML system makes a *one-time* decision.
  - For each loan application or each email, the system would make an independent decision. There is no reason to be influenced by the previous decision.

What if we need to make a series of interconnected decisions over time?

Problem Setting: Sequential Decision Making

- The decision-making “agent” must make a series of interconnected decisions that affect each other. The outcome of one decision affects the future decision-making process.

- Performance score is typically a function of the full sequence of states and decisions.



Problem Setting: Sequential Decision Making

Must make a sequence of decisions to
maximize some success measure/"reward",
which is a cumulative effect of the full sequence.

Actions a_t:	muscle contractions	motor current or torque	what to purchase
State s_t:	sight, smell	camera images	inventory levels
Reward r_t:	food	average speed	profit

“Policies” for Sequential Decision Making

For any input state of the system, the ML model maps it to a decision.

- This motivates the following input-output structure of the model:
  - Input: state observation, like sight and smell for the dog.
  - Output: actions, like muscle contractions.

This mapping from input states to a probability distribution over output actions (or sometimes just a single deterministic action) is called a decision-making “policy”, often denoted π.

(PS: The word “model” in this context has many meanings, so we use the word “policy” instead to avoid ambiguity.)

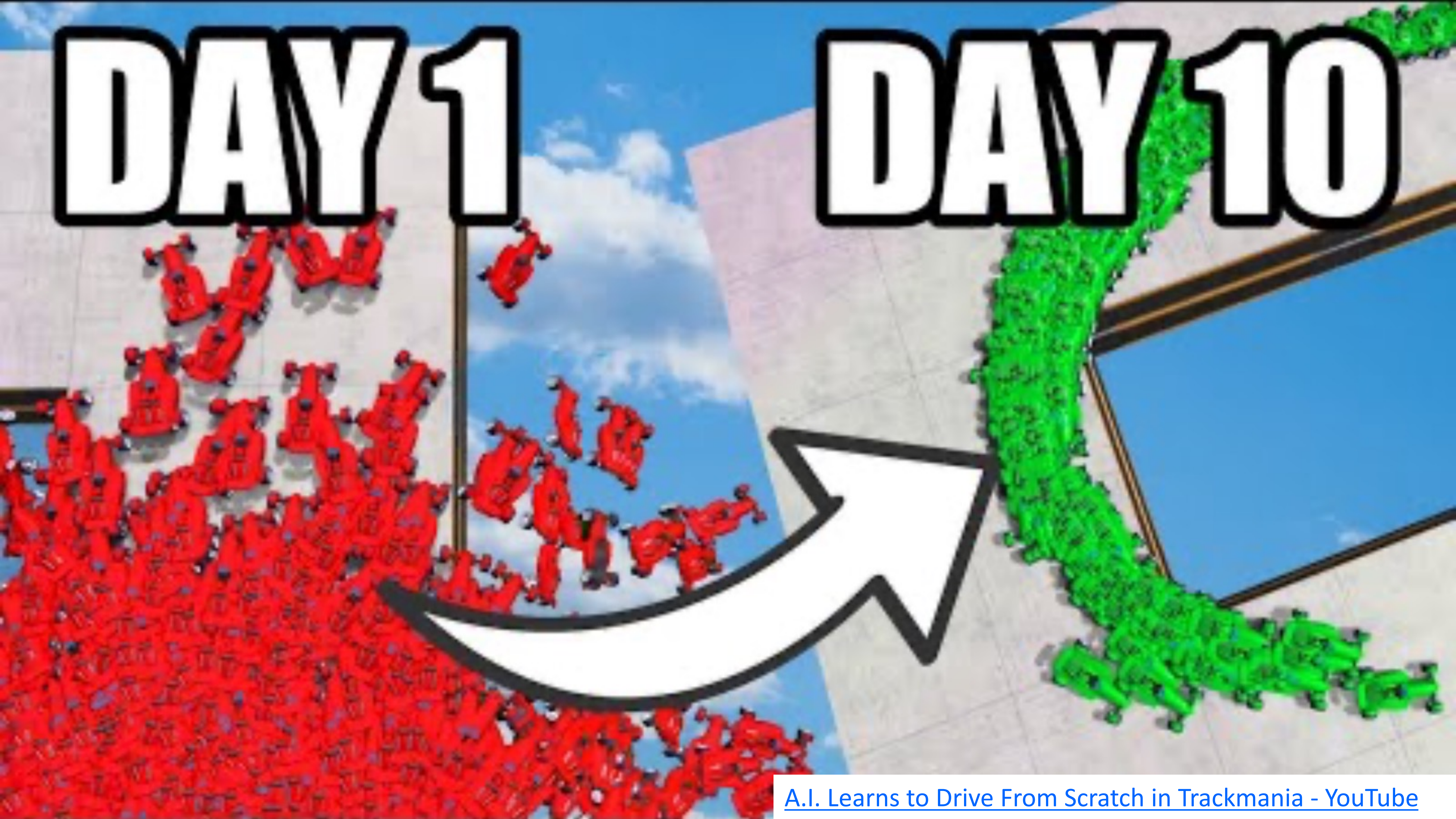
RL: Learning Through Trial and Error

The aim of RL is to learn to make sequential decisions in an environment:

- Driving a car
- Cooking
- Playing a videogame
- Controlling a power plant
- Riding a bicycle
- Making movie recommendations
- Navigating a webpage
- Treating a trauma patient

How does an RL agent learn to do these things?

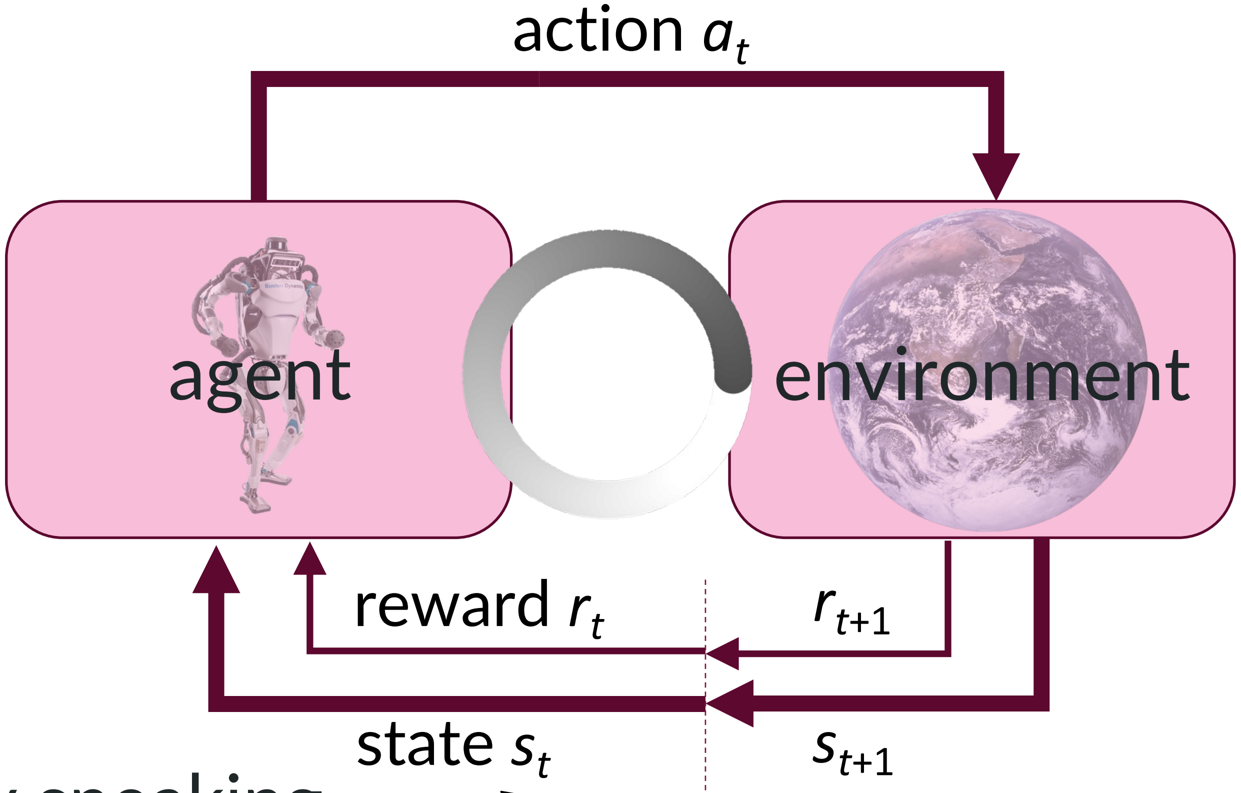
- Very little needs to be known about the task in advance.
- Assume only occasional feedback, such as a tasty meal, or a car crash, or video game points.
- Learn through trial and error.



A.I. Learns to Drive From Scratch in Trackmania - YouTube

The Standard Reinforcement Learning Interface

- Agent receives observations (state $s_t \in$ S) and feedback (reward r_t) from the "environment"
- Agent takes action $a_t \in$ A
- Agent receives updated state s_{t+1} and reward r_{t+1}



- Agent's goal is to maximize, loosely speaking, "expected rewards in the future".

States might have to be estimated, e.g., from images

Goal of RL is to learn a policy $\pi(s): S \rightarrow A$ for acting in the environment

e.g. state s_t = robot pose, action a_t= motor torques, r_t= running speed

Image: https://robots.ieee.org/robots/pr2/

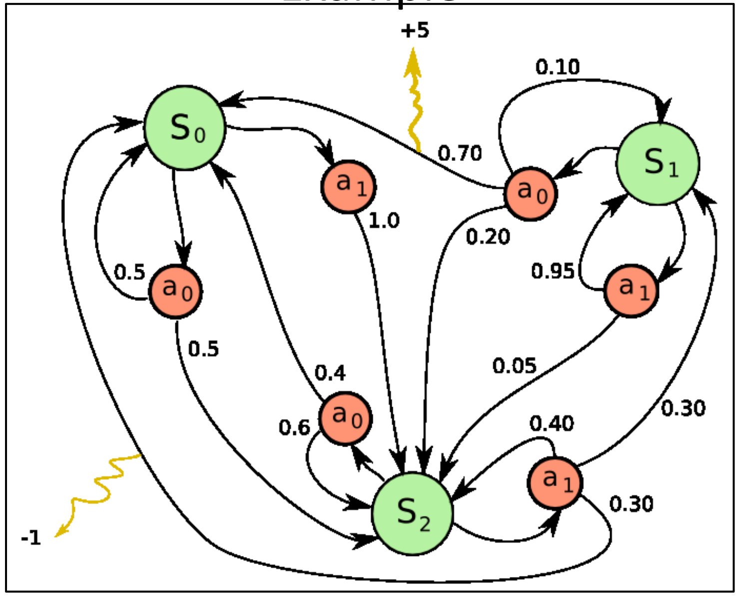
The Environment as a Markov Decision Process (MDP)

An MDP (S, A, P, R, γ) is defined by:

- Set of states $s \in S$
- Set of actions $a \in A$
- Transition function $P(s_{t+1} \mid s_t, a_t)$
  - Probability $P(s' \mid s, a)$ that a from s leads to s'
  - Also "dynamics model" / just "model"
- Reward function $r_t = R(s_t, a_t, s_{t+1})$
- Discount factor $\gamma < 1$, expressing how much we care about the future (vs. immediate rewards)
- "return" = *discounted* sum of future rewards $\sum_t \gamma^t r_{t+1}$
- Goal: find a "policy" π such that its actions $a_t = \pi(s_t)$ maximize return *in expectation*

Unknown to agent

Example



In RL, we assume no knowledge of the true functions $P(\cdot)$ or $R(\cdot)$

Image: https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690

Notation: Rewards, Returns, Utilities, Discounted …

- "reward": instantaneous reward r_t received at one time instant from the environment
- "return": sum of (future) rewards
  - Sometimes also called "cumulative reward", "utility", etc.
  - By default includes the discount factor i.e. return = $\sum_t \gamma^t r_{t+1}$
    - sometimes called the "discounted return" to make this explicit and distinguish from "(undiscounted) return" = $\sum_t r_{t+1}$.

Outside of this class, lots of confusion, beware! For example:

- Sometimes "reward" ↔"return"
- Sometimes "return" is by default undiscounted, etc.

Entering An Unknown Gridworld

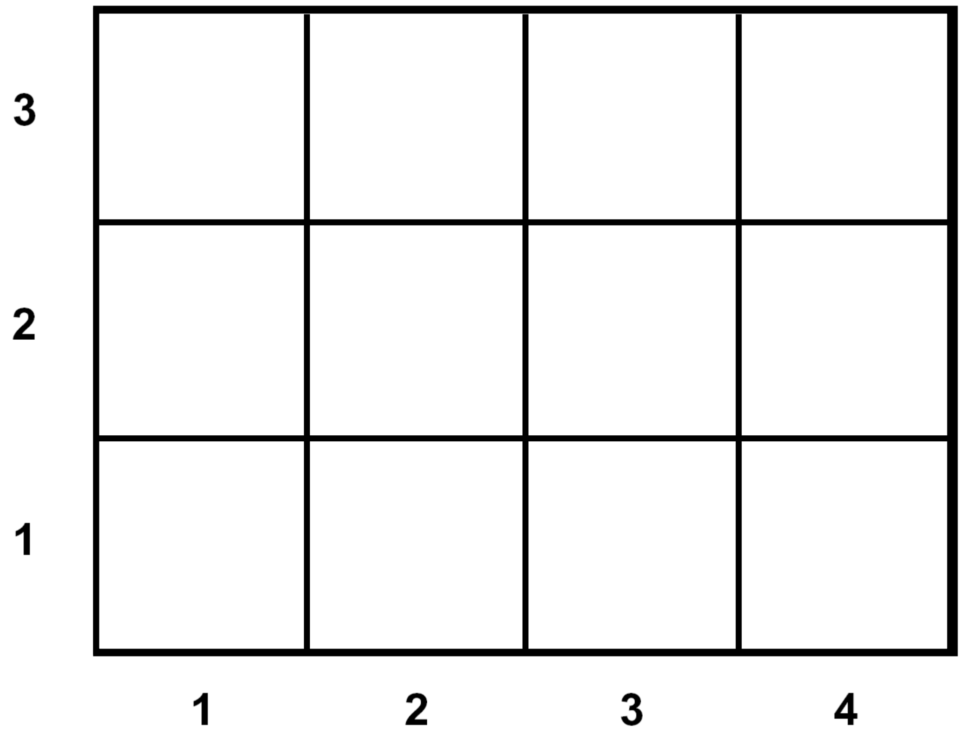
In the shoes of an RL agent

Sample RL environment: Grid World

- The agent's state is one cell $s = (x, y)$ within the grid $x \in \{1,2,3,4\}$ and $y \in \{1,2,3\}$.
- The agent can execute 4 actions: $a =$"W", "E", "S", "N"

For the moment, this is all that that the RL agent knows about the environment. In particular, it does not know:

- $P(s'|s, a)$
  - Which cell would it move to, if it executes an action from a cell? (e.g. $a =$"N" from $s = (1, 2)$)
  - The result might even be non-deterministic.
- $R(s, a, s')$
  - What is the instantaneous reward it would get if it moved from $s = (1,2)$ to $s' = (1,3)$ by executing action $a =$"N"?

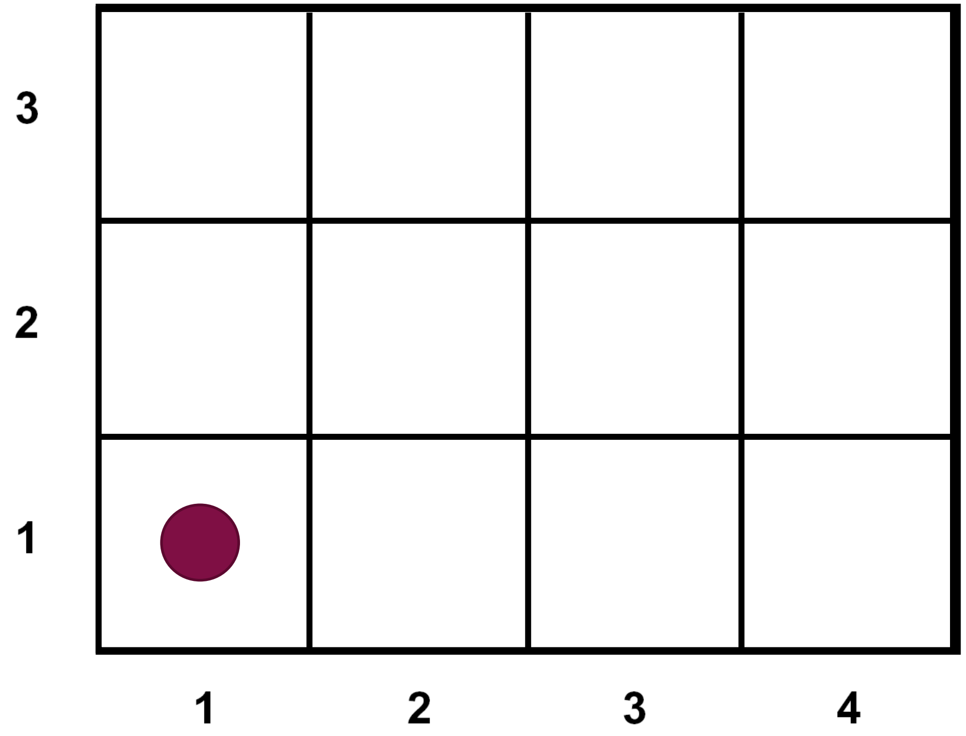


A random trajectory of an RL agent

Time t=0

s=(1,1)

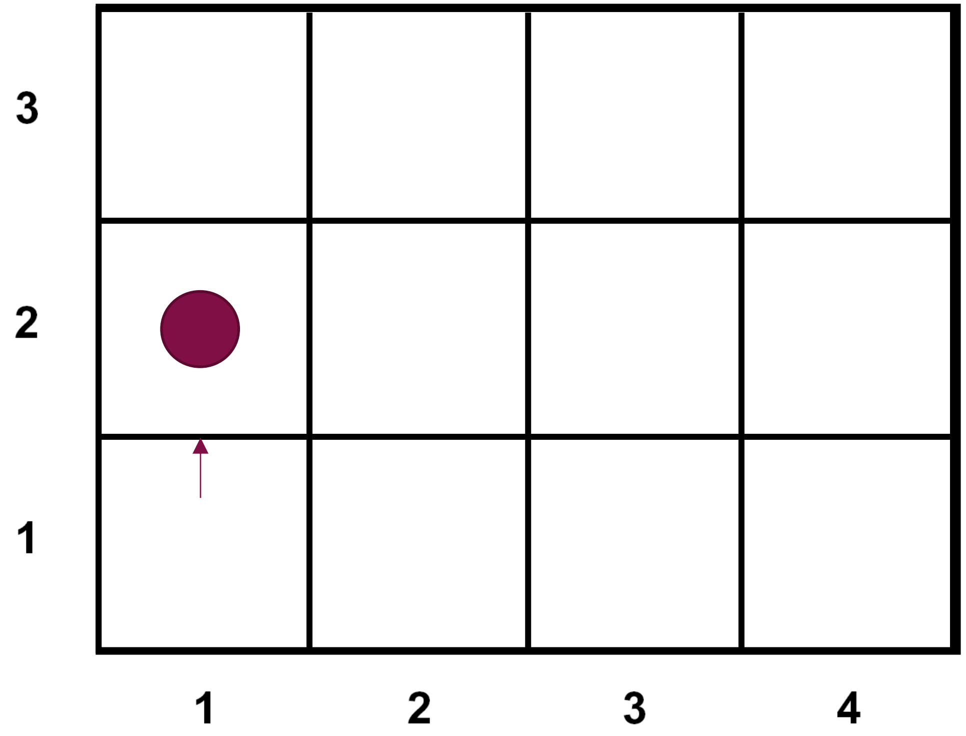
Action= “N”



A random trajectory of an RL agent

Time t=0

s=(1,1)
Action= “N”
s’=(1,2)
Reward = -0.03

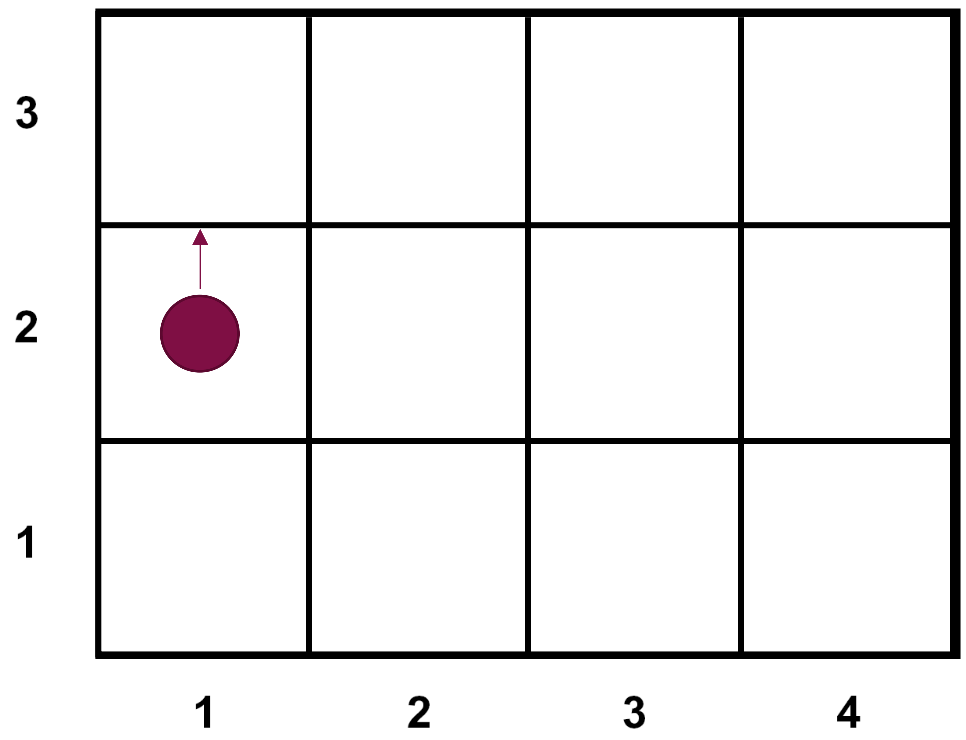


Time step t=0 over

A random trajectory of an RL agent

Time t=1

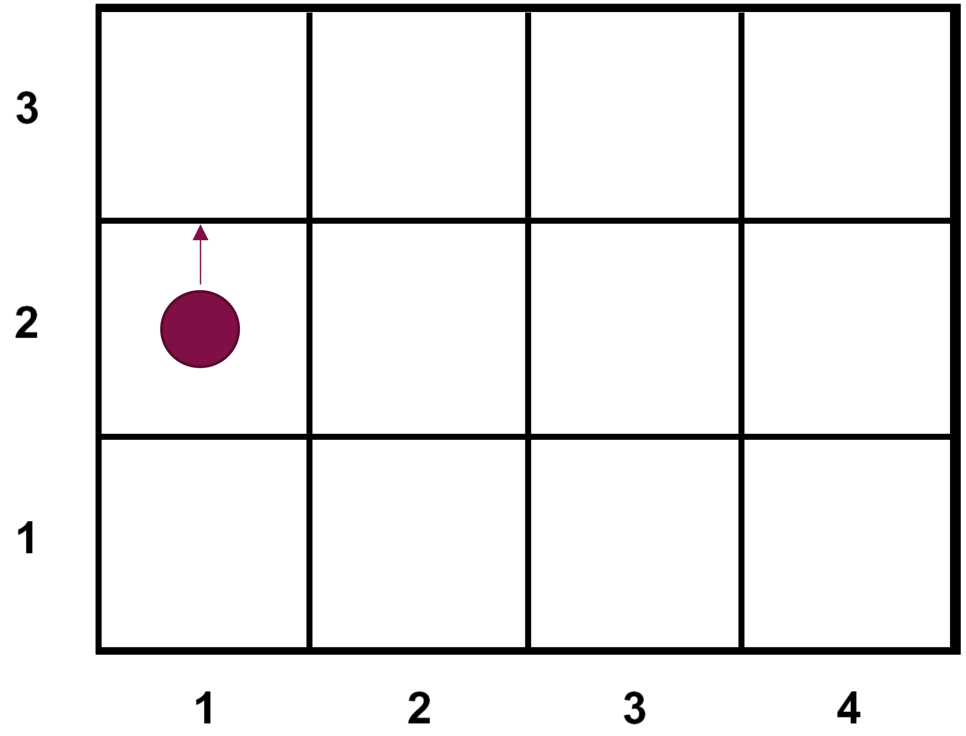
s=(1,2)
Action= “N”
s’=?
Reward = ?



A random trajectory of an RL agent

Time t=1

s=(1,2)
Action= “N”
s’=(1,2)
Reward = -0.03



Time step t=1 over

A random trajectory of an RL agent

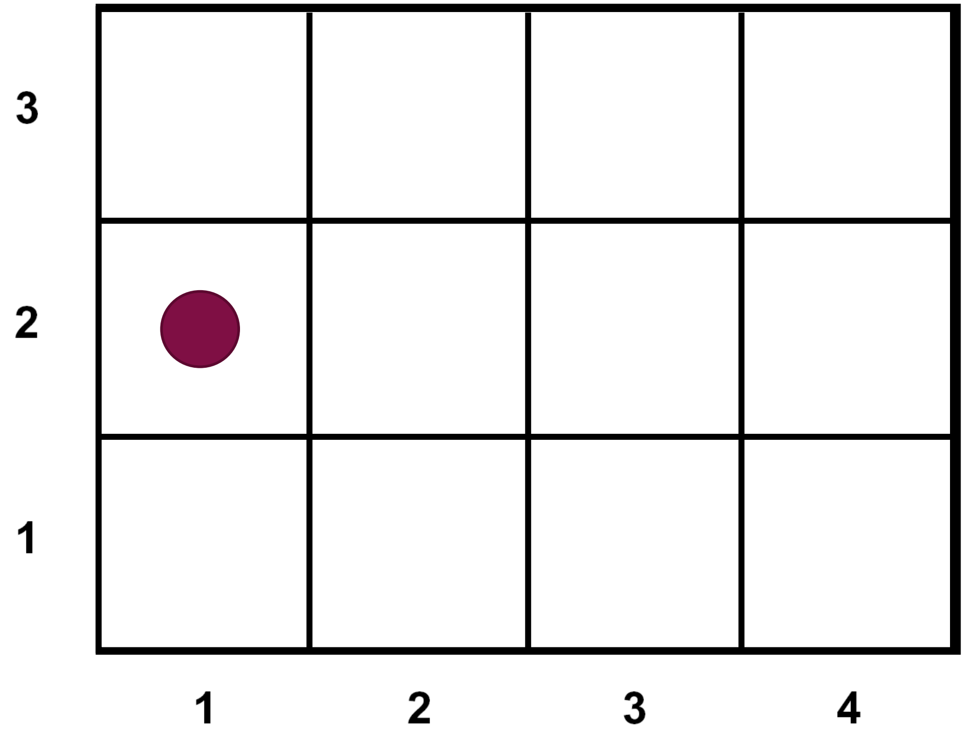
Time t=2

s=(1,2)

Action= “N”

s’=?

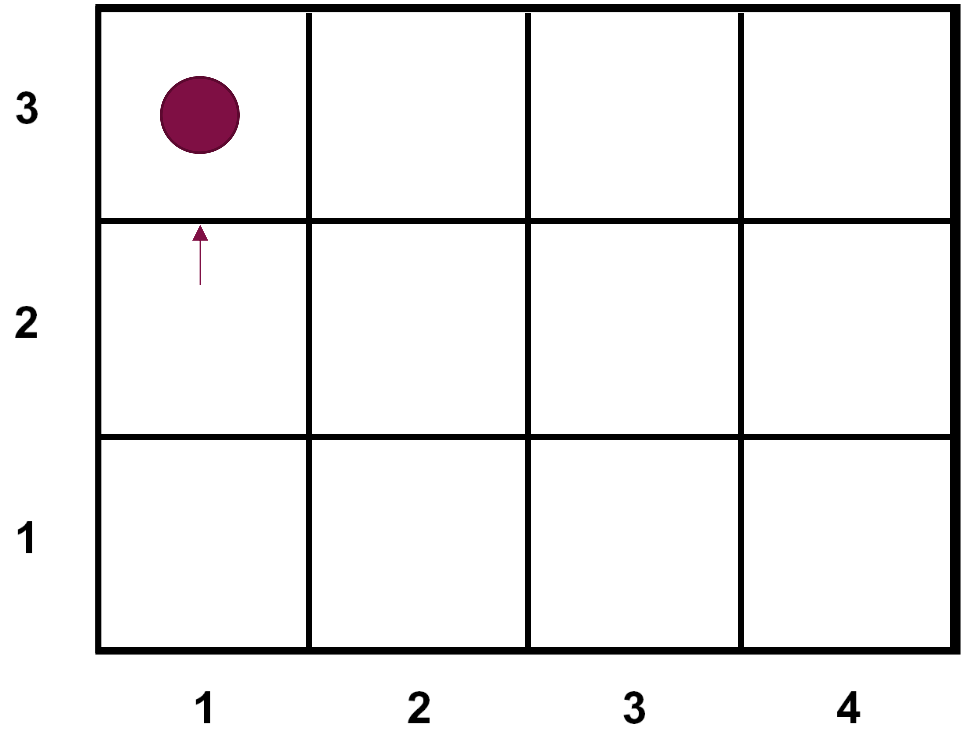
Reward = ?



A random trajectory of an RL agent

Time t=2

s=(1,2)
Action= “N”
s’=(1,3)
Reward = -0.03

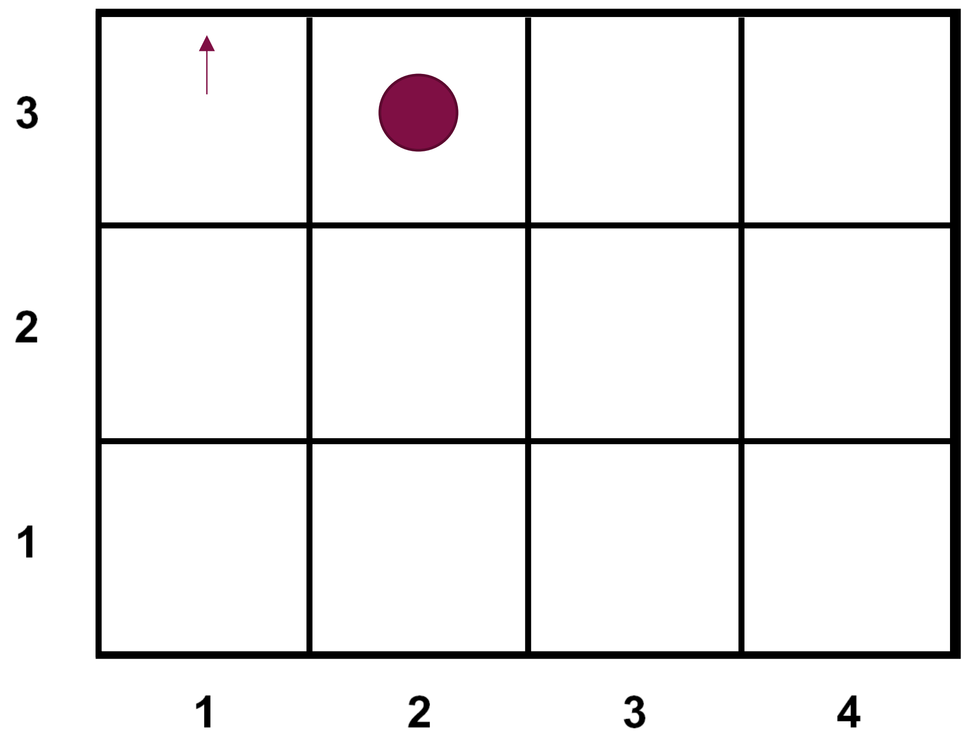


Time step t=2 over

A random trajectory of an RL agent

Time t=3

s=(1,3)
Action= “N”
s’=(2,3)
Reward = -0.03



Time step t=3 over

A random trajectory of an RL agent

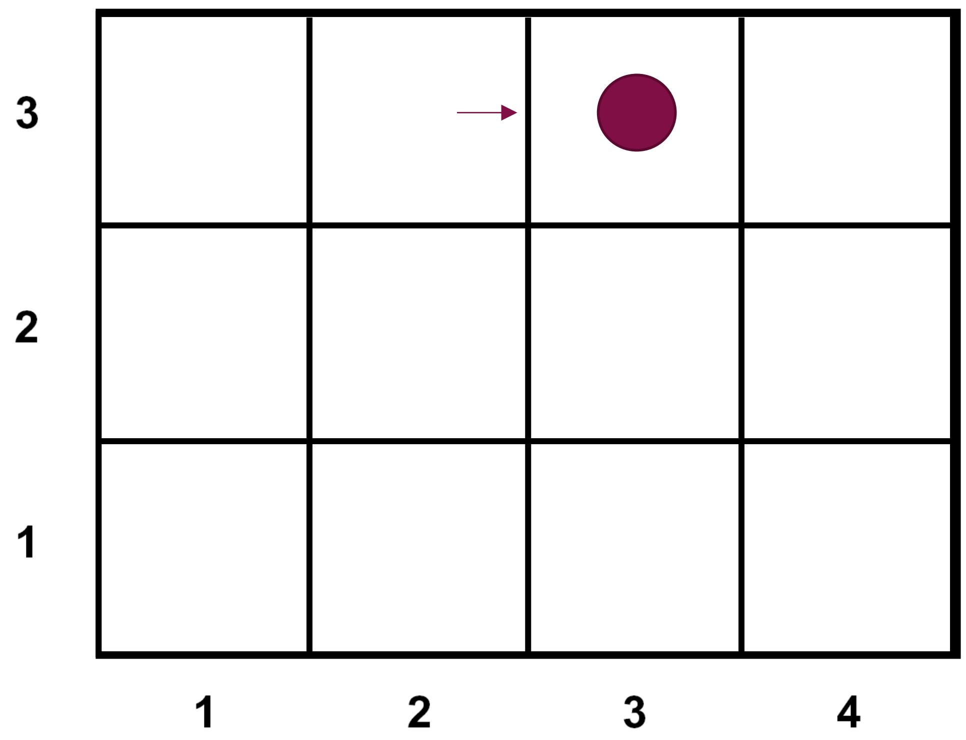
Time t=4

s=(2,3)

Action= “E”

s’=(3,3)

Reward = -0.03



Time step t=4 over

A random trajectory of an RL agent

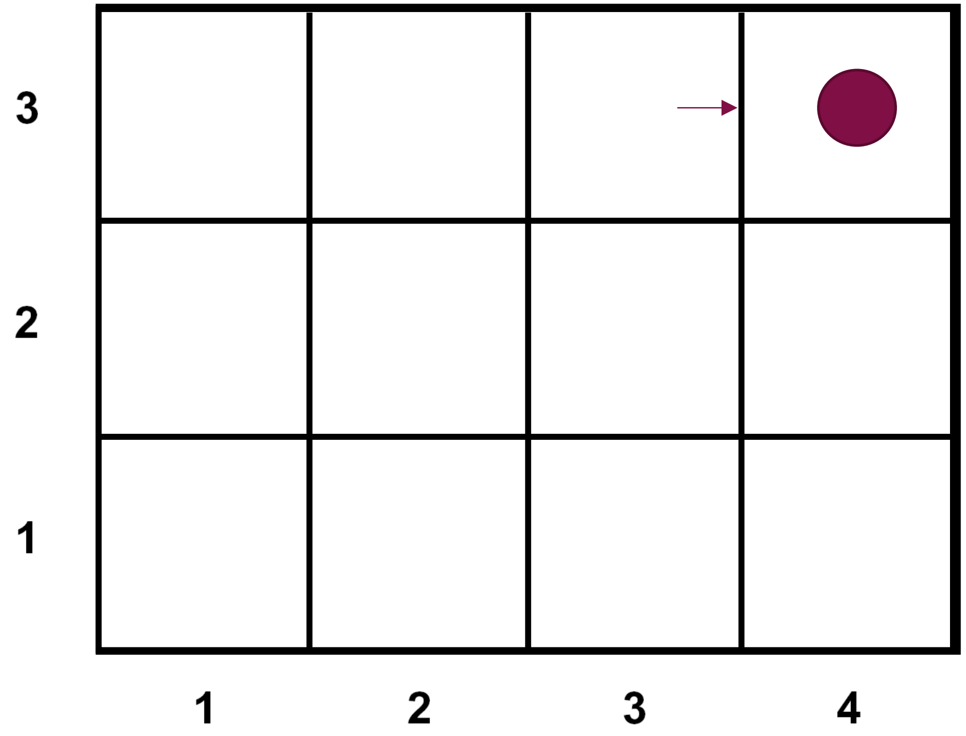
Time t=5

s=(3,3)

Action= “E”

s’=(4,3)

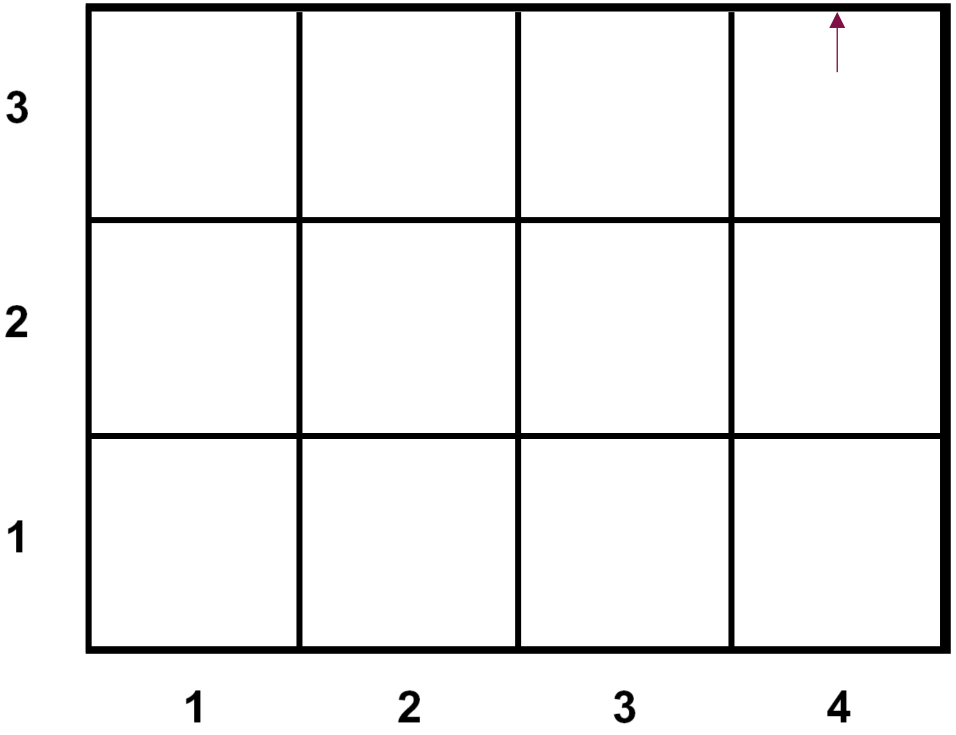
Reward = -0.03



Time step t=5 over

A random trajectory of an RL agent

s=(4,3)
Action= “N”
s’= special state “END”
Reward = +1



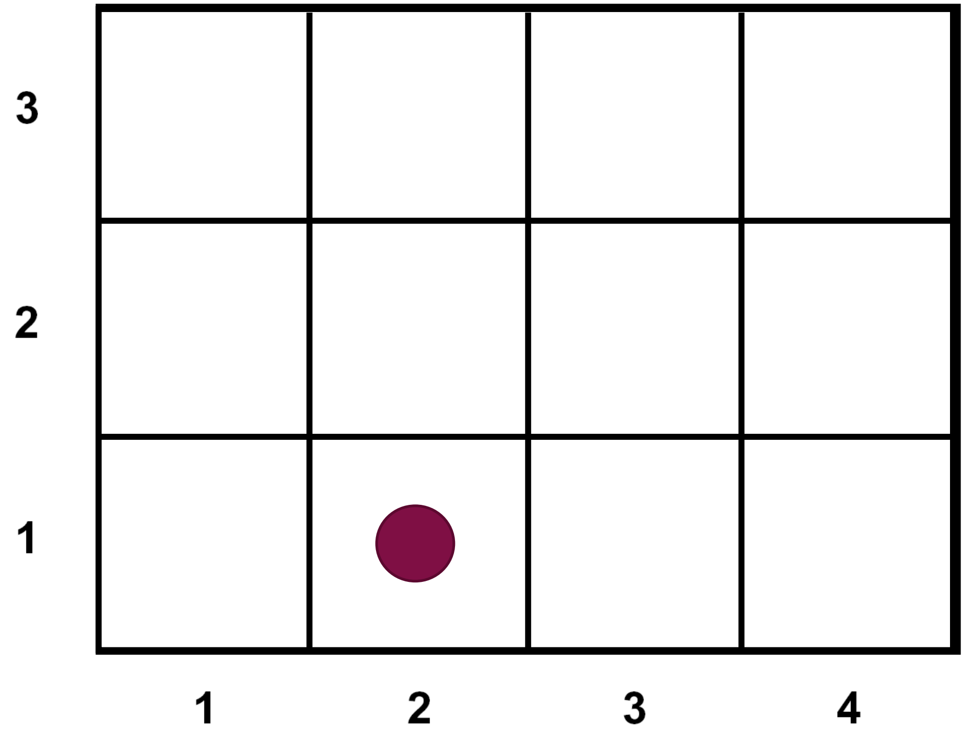
One “episode”/“trial” of our “episodic task” is over.

Next, the agent respawns in the environment. “Reset”

Reset

Another episode begins!
Time t=0

s=(2,1)
Action=?
s'= ?
Reward = ?



Note that we have started at a different point in the grid than last time.
In addition to (S, A, P, R, γ), there may also be an "initial state probability distribution" μ over states that the agent is spawned into.

So, can we maximize rewards in this environment?

- What have we learned about this environment after having acquired this experience?
  - Do we know something about P, R?
  - Do we know how to act optimally now?

We have learned some things, but there is still far too much ambiguity.

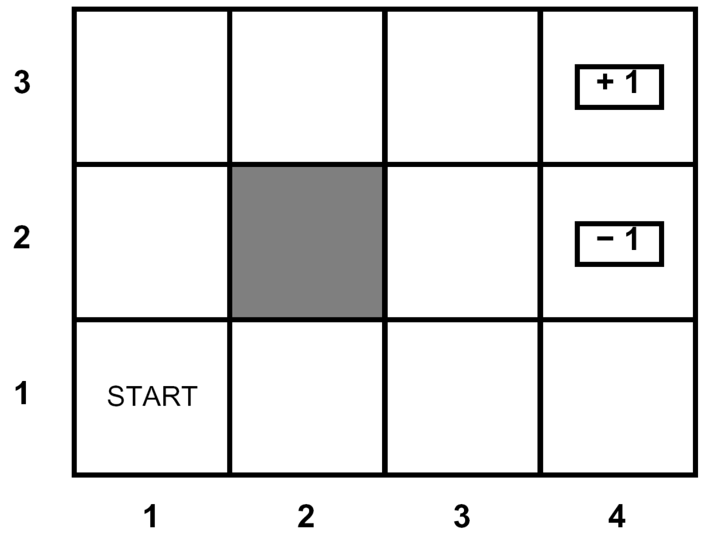
Perhaps with more experience …

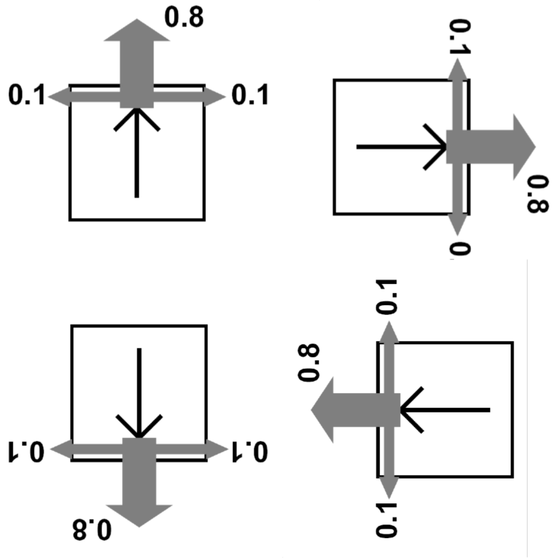
Provided sufficient experience, RL algorithms can learn optimal policies!

Gridworld Revealed

Behind The Scenes: The Full Environment

- A grid map with solid / open cells. Agent('s dot) moves between open cells.
- From terminal states $(4{,}3)$ and $(4{,}2)$, any action ends the episode, and results in a +1/-1 reward respectively.
- For each timestep outside terminal states , the agent pays a small "living" cost (negative reward): -0.03
- The agent actions N, E, S, W correspond to North, East, South, West
  - **But the outcomes of actions are not deterministic!**
    - The dot obeys the commanded motion direction 80% of the time
    - 10% of the time, the dot instead executes a different direction 90° off from the agent command. Another 10% of the time, -90° off.
    - E.g. if dot surrounded by open cells and executing action N, will end up in the northern cell 80% of the time, in the eastern cell 10% of the time, and in the western cell 10% of the time.
  - **The dot stays put if it attempts to move into a solid cell** or outside the world. (Imagine the map is surrounded by solid cells)
- Goal: As always, maximize the sum of discounted future rewards within an episode

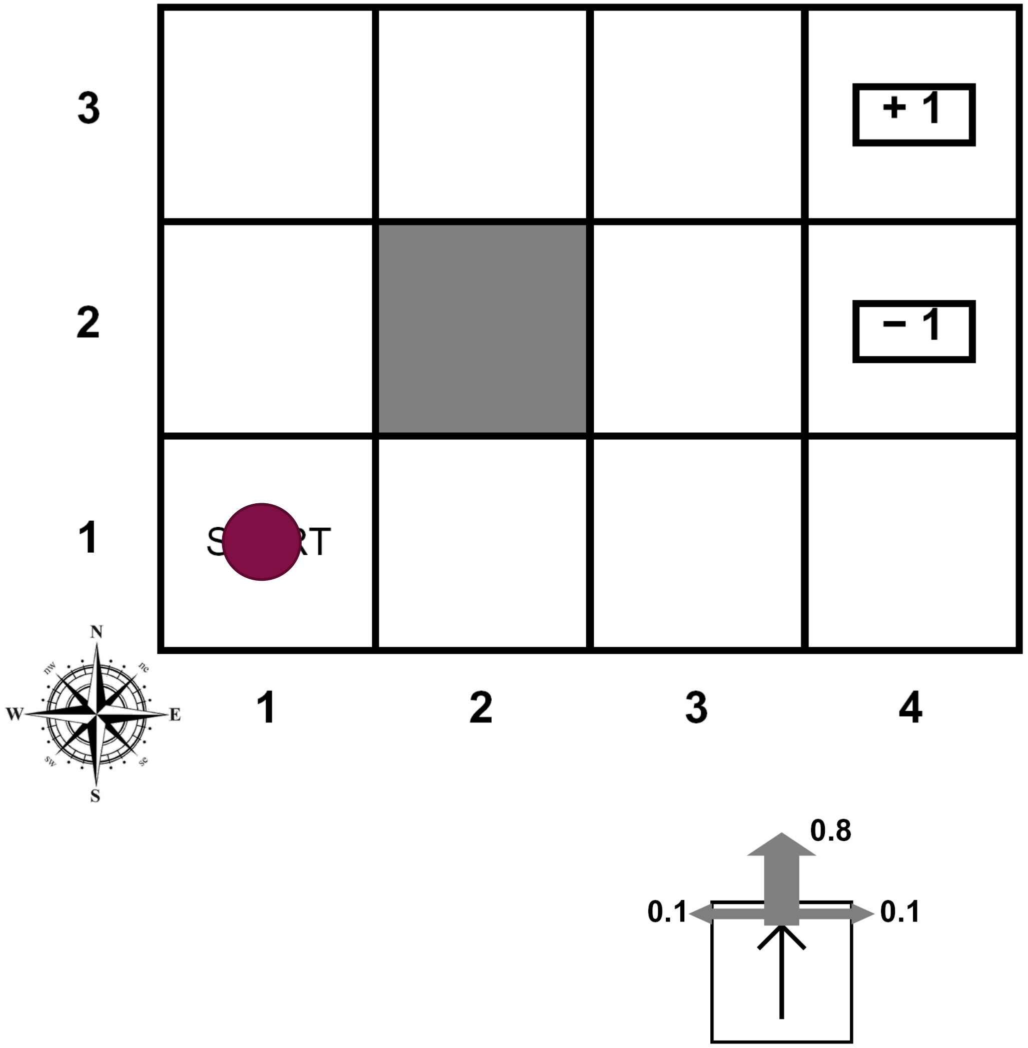




What actually happened in that episode?

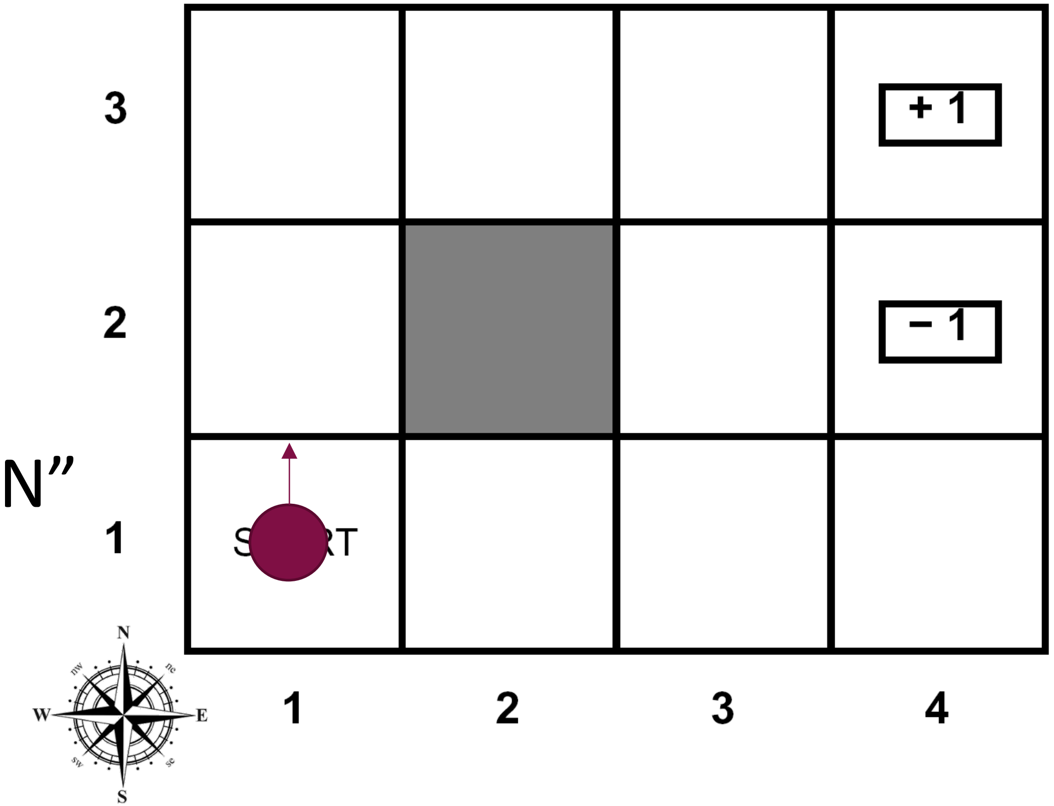
- Now that we have seen the full environment, let’s view a replay with all this extra information to see what actually happened during that one episode of experience we saw before.

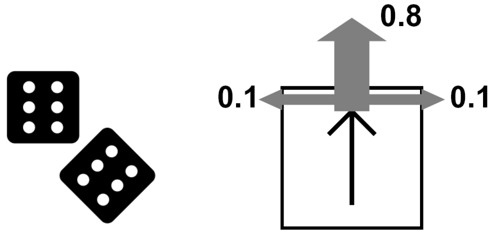
What actually happened in that episode?



What actually happened in that episode?

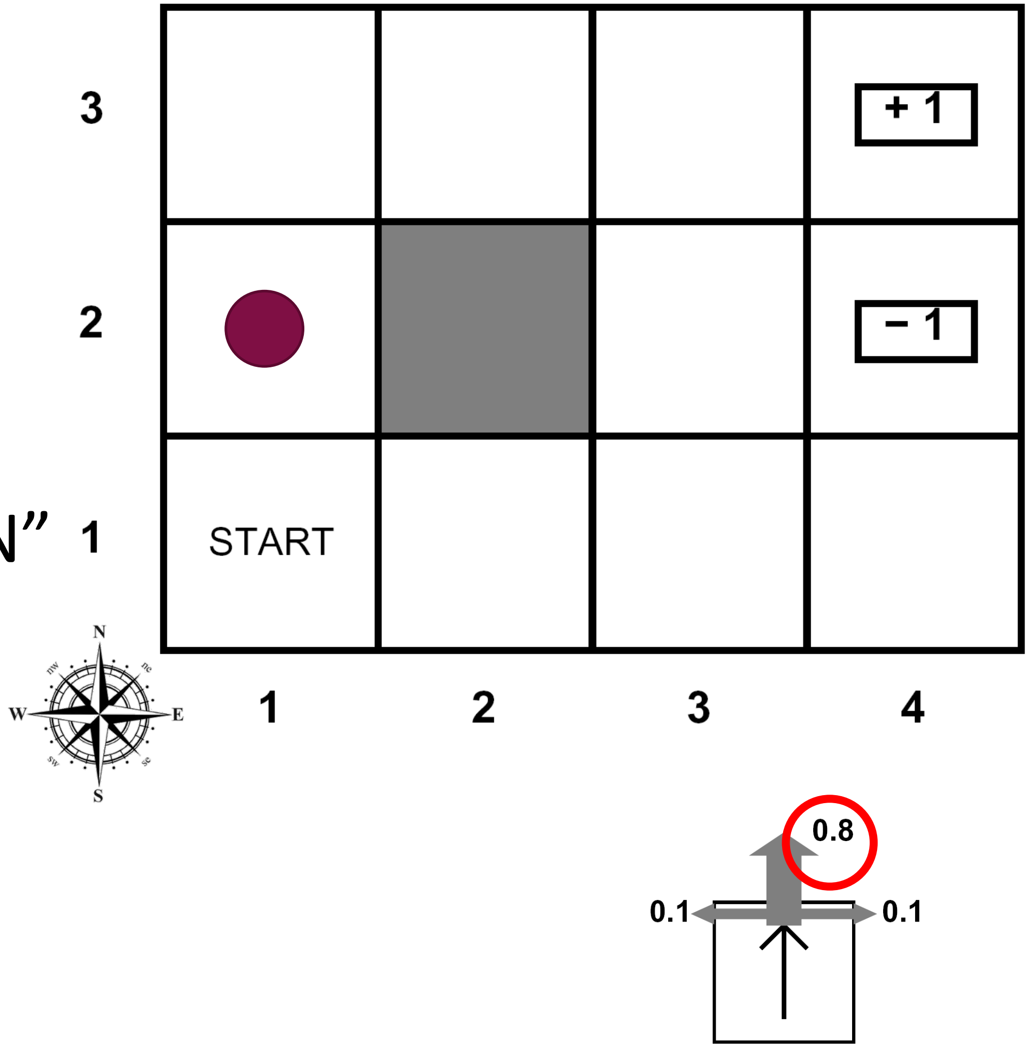
Action= “N”





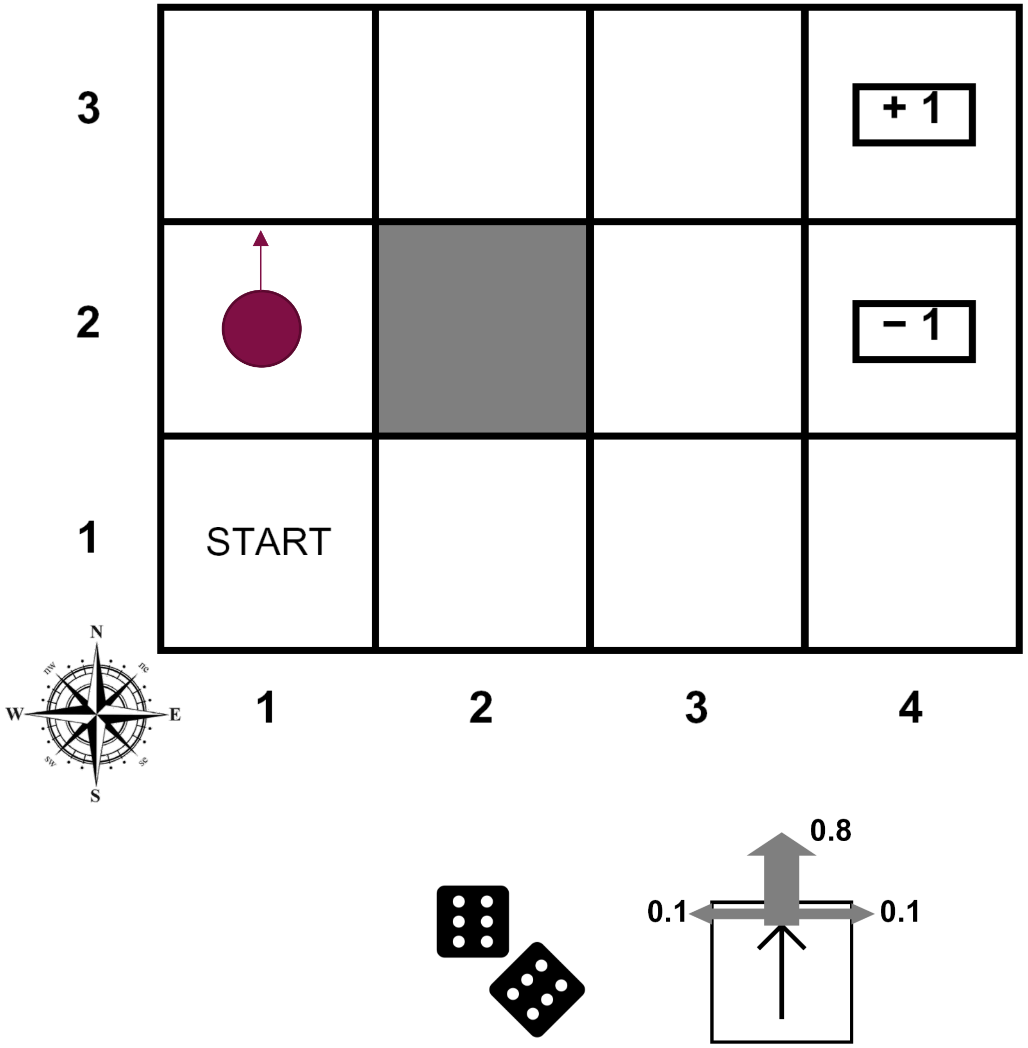
What actually happened in that episode?

Action= “N”
Attempted motion = “N”
Reward = -0.03



What actually happened in that episode?

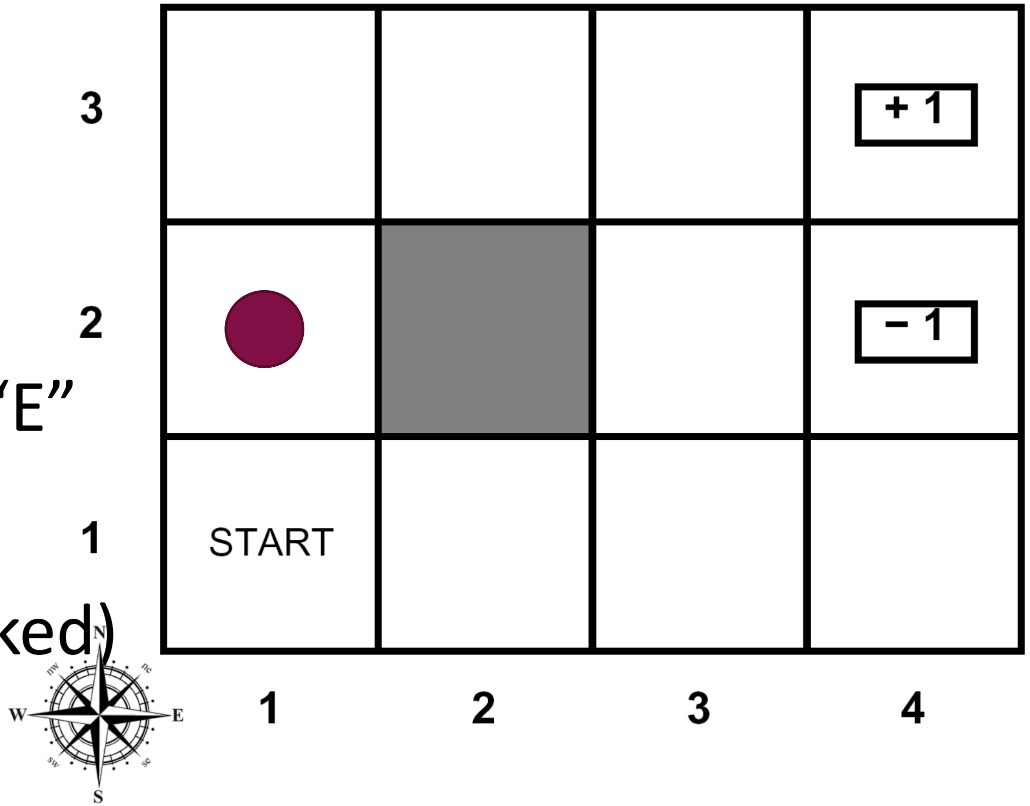
Action= “N”

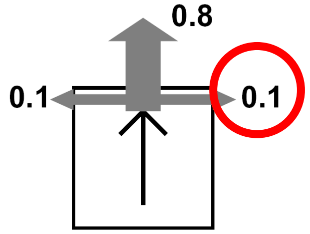


What actually happened in that episode?

Action= “N”
Attempted Motion=“E”
Reward = -0.03

(stays still because blocked)



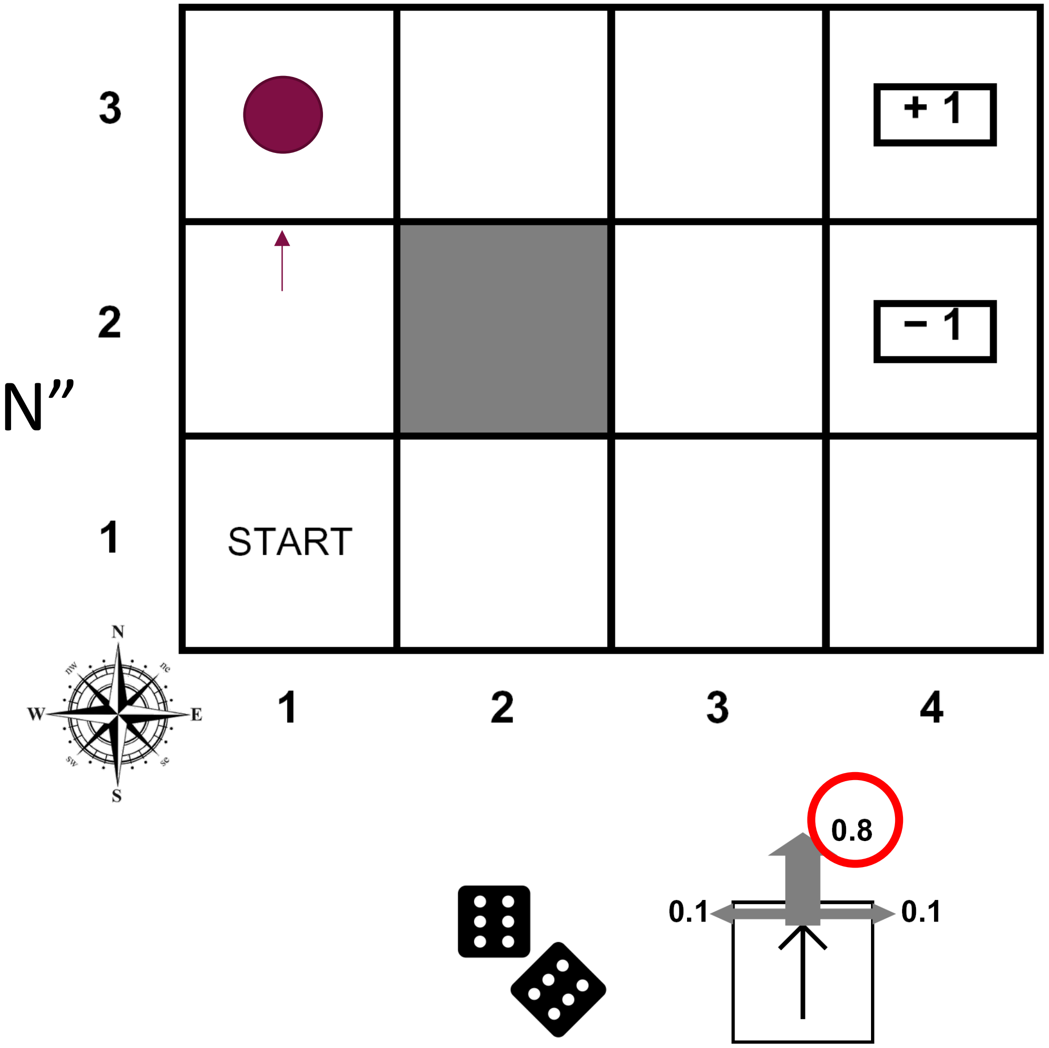


What actually happened in that episode?

Action= “N”

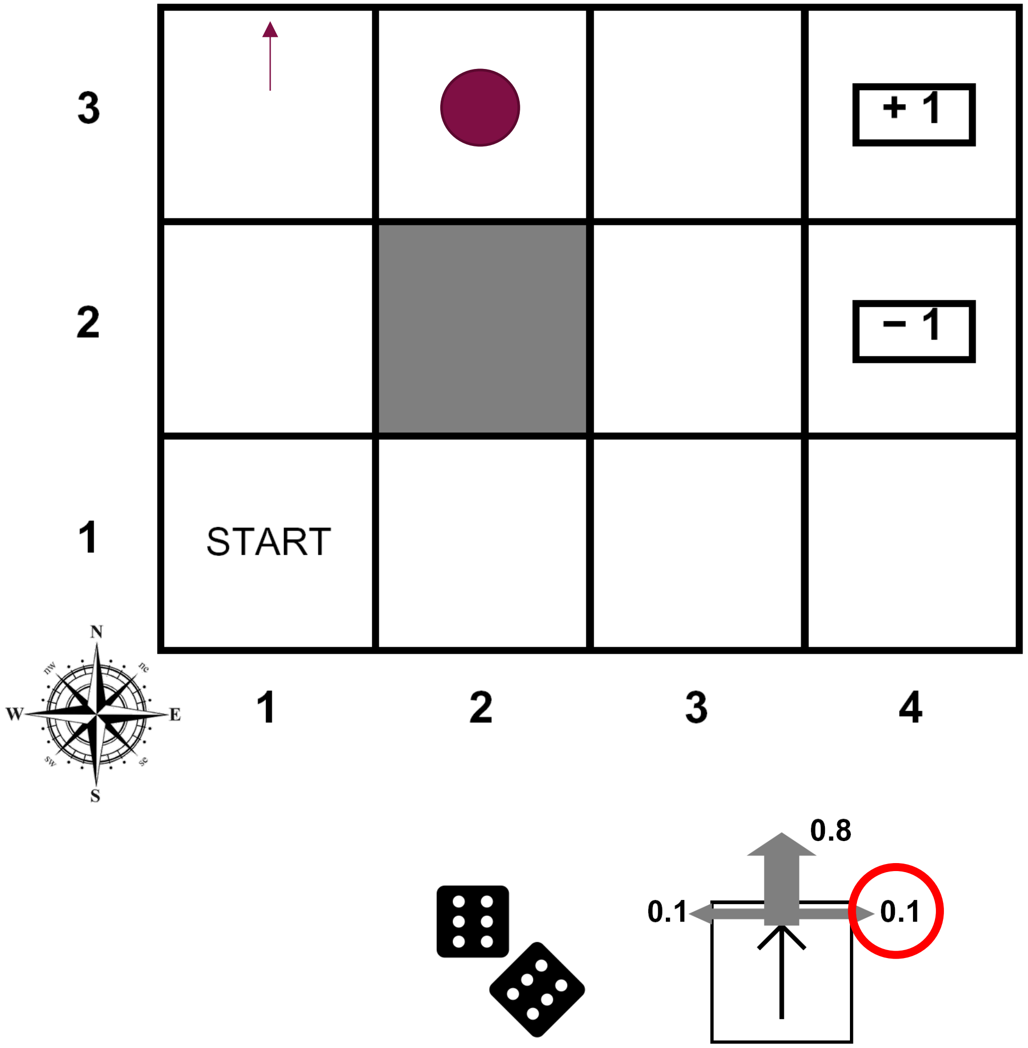
Attempted Motion=“N”

Reward = -0.03



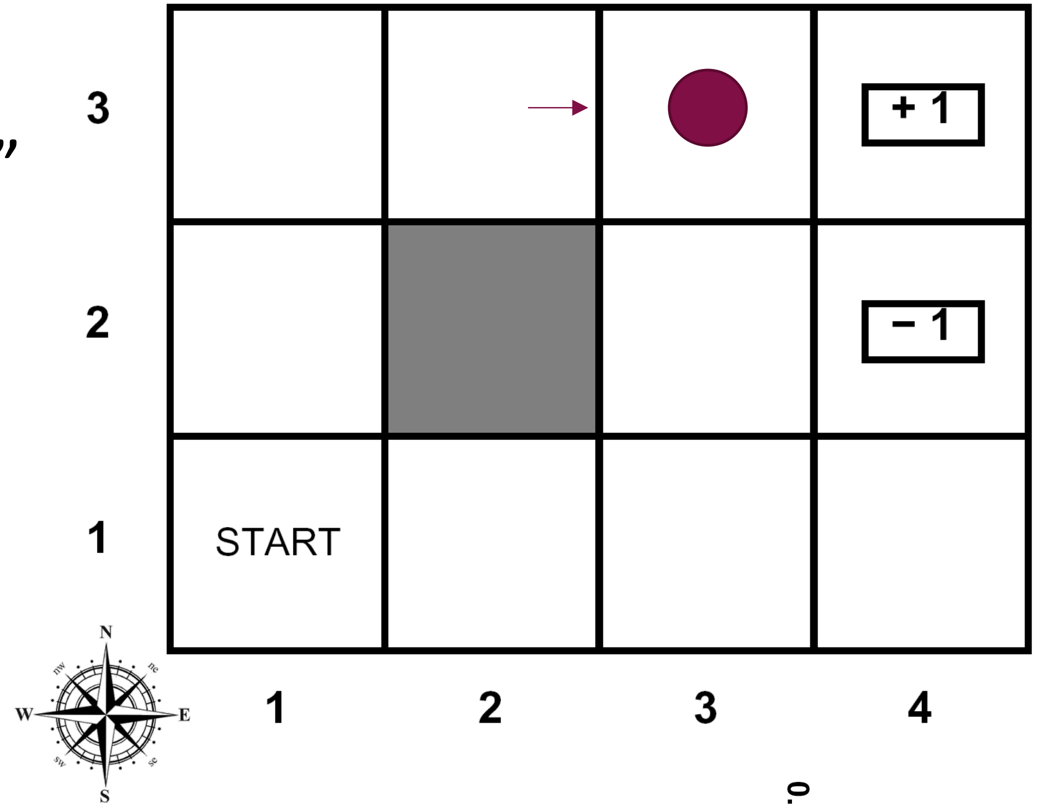
What actually happened in that episode?

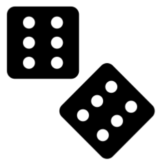
Action= “N”
Attempted Motion=“E”
Reward = -0.03

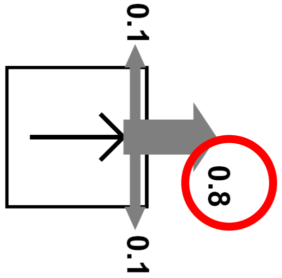


What actually happened in that episode?

Action= “E”
Attempted Motion=“E”
Reward = -0.03

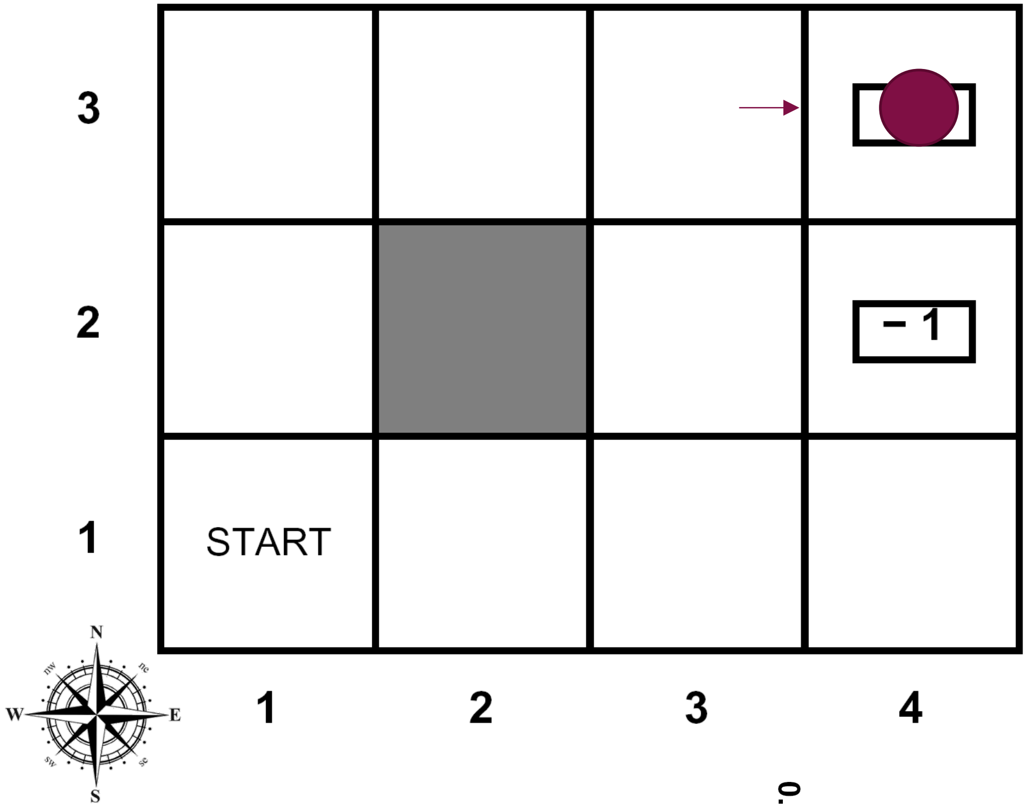


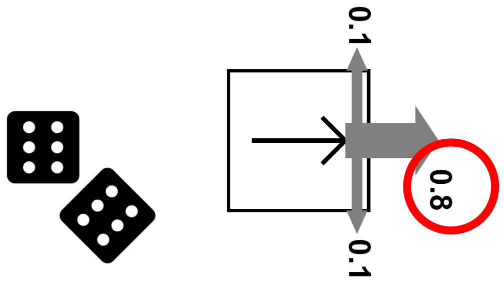




What actually happened in that episode?

Action= “E”
Attempted Motion=“E”
Reward = -0.03





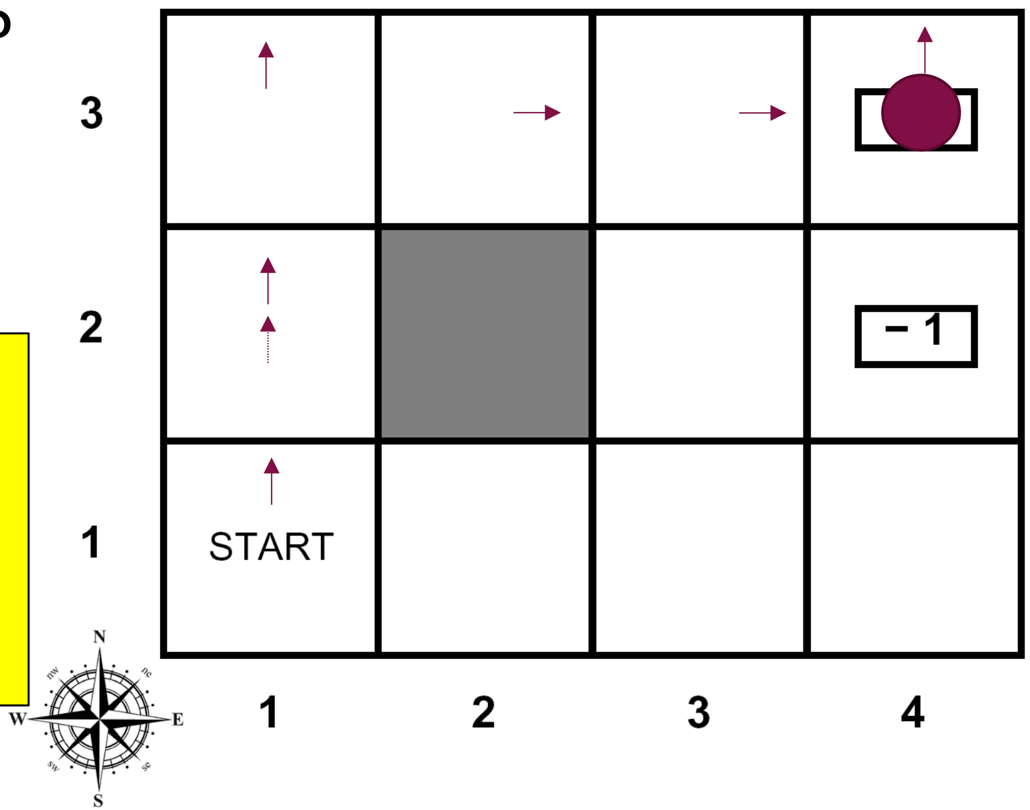
What actually happened in that episode?

Action= “N”

Attempted Motion: ?

Result=“**the end**”

Reward = +1



It so happened that our random trajectory did end up at the right place!

Was this action sequence “optimal?” No

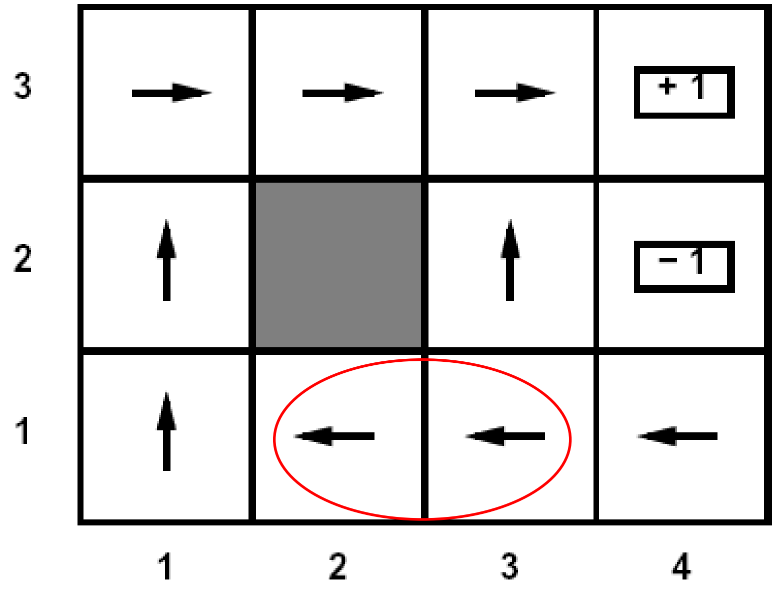
Note: this corresponds to saying: “when $s = (4{,}3)$, for any a, the reward is $R(s, a, s') = R(s) = +1$”.
This is meaningfully different from: “when $s' = (4{,}3)$, the reward is $R(s, a, s') = R(s') = +1$ for any s, a.”

Desired Outcome of RL: Optimal Policies

Goal: given some environment, find the optimal policy $\pi^*(s): S \to A$

- "Optimal" $\Longrightarrow$ Following π^* maximizes expected return $\sum_t \gamma^t r_{t+1}$

Example optimal policy π^*



Optimal policy when living cost is
$R(s, a, s') = R(s) = -0.03, \gamma = 1.0$
for all non-terminal states s

Q: What's going on here?

Why discounts?

Idea: future rewards are worth exponentially less than current rewards.

- They are less certain

Future rewards are discounted by $0 < \gamma < 1$:

$$\sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

discounted cumulative future reward / "return"

Future rewards matter less to the decision than more recent rewards

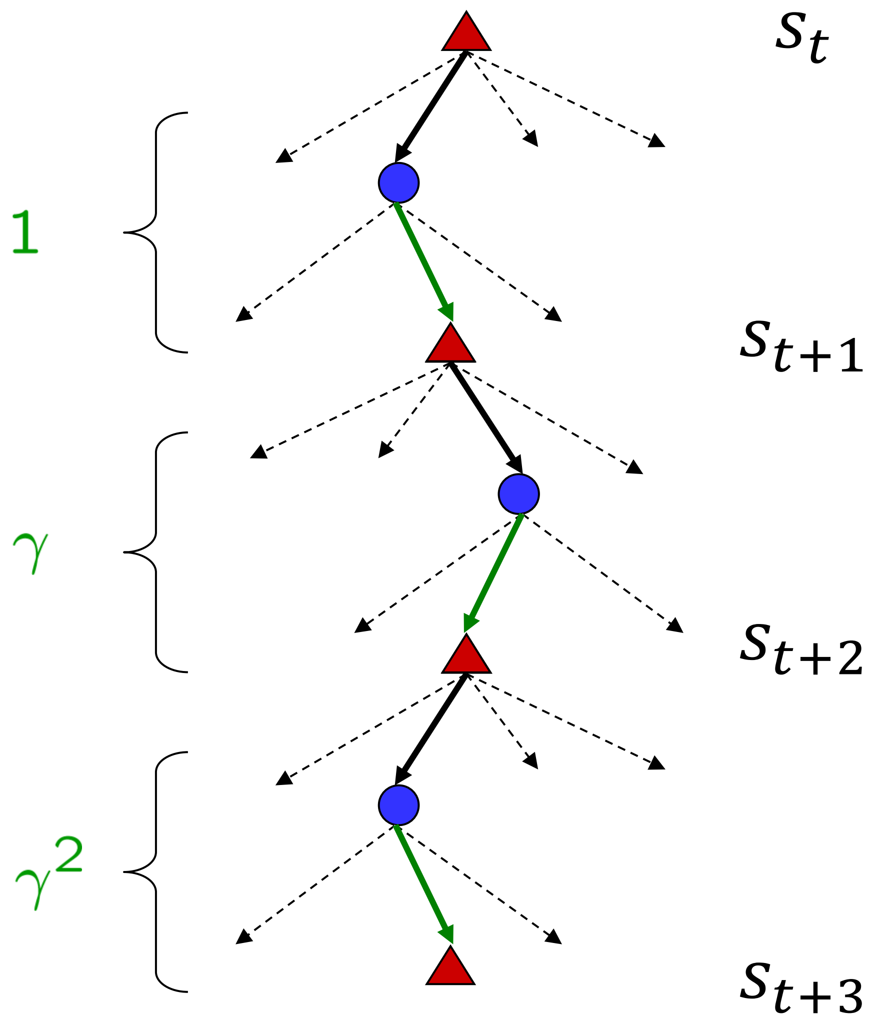
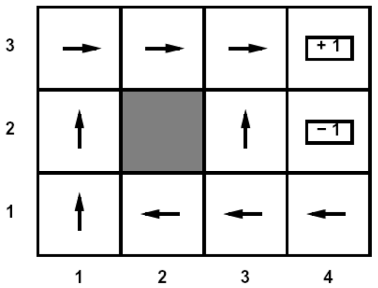


Image by Dan Klein

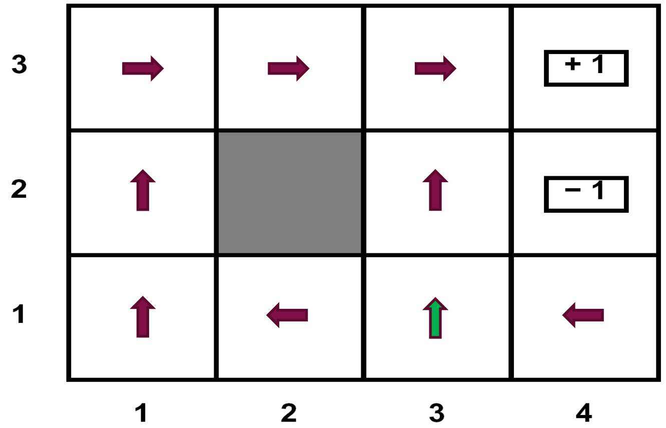
Also very useful for theoretical analysis

Sensitivity of Optimal Policy To R And γ

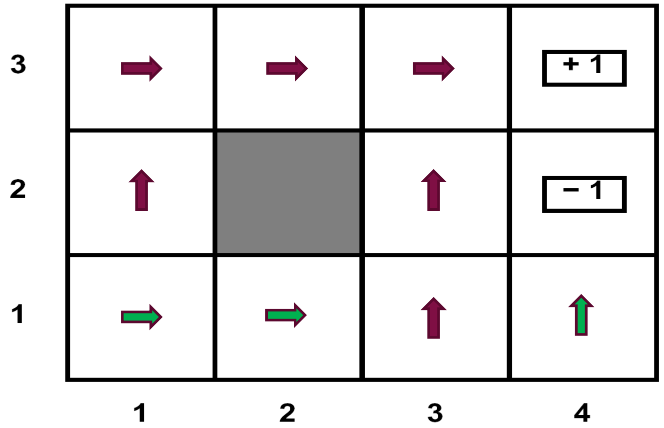


The task specification through R (and γ) is critical!

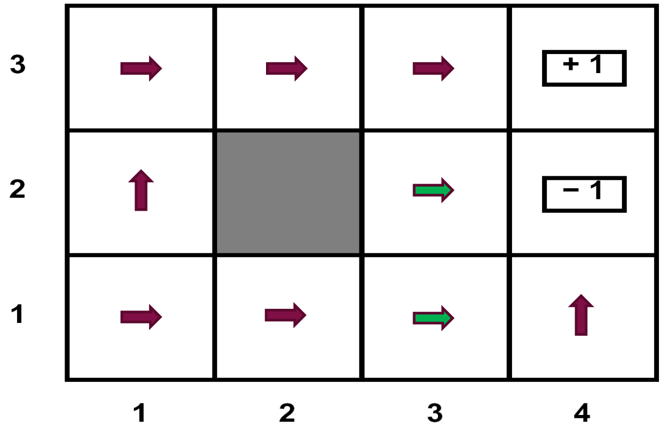
Living cost $R(s, a, s') = 0$



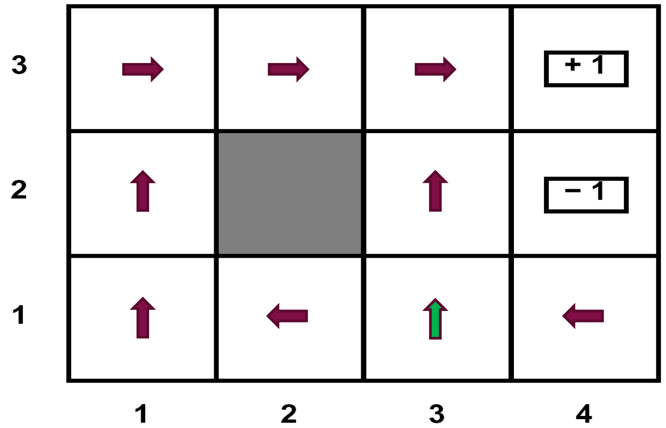
$R(s, a, s') = -1$



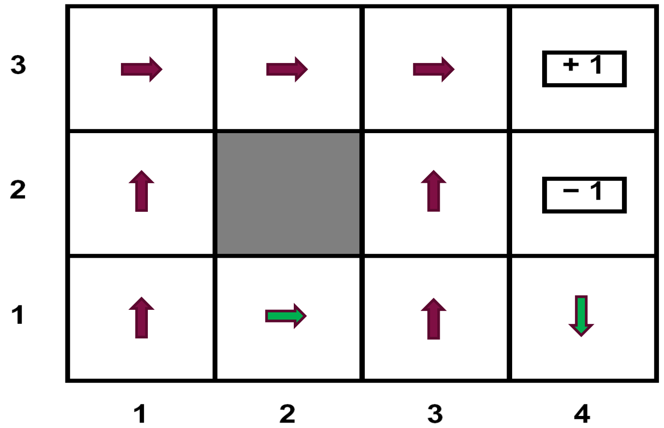
$R(s, a, s') = -2$



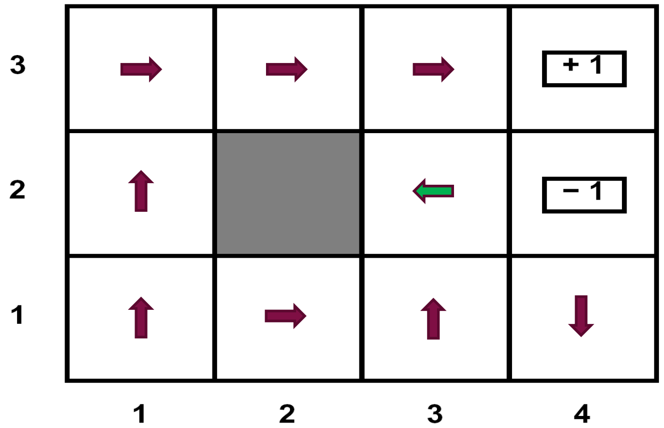
$\gamma = 0.9$ at 0.03 living cost



$\gamma = 0.5$



$\gamma = 0.1$



How is RL Different from Supervised Learning (SL)?

SL: Find $h(x): X \rightarrow Y$, that minimizes a loss L over training (x, y) pairs

RL: Find $\pi(s): S \rightarrow A$ that maximizes expected return

Supervised Learning

- Target labels for h are directly available in the training data
- Train to map (regress/classify) from x to y in the training data

Reinforcement Learning

- Optimal action labels a for states s are not given to us. No predefined solutions!
- Train by trying various action sequences in an environment, and observing which ones produce good rewards over time.

Unlike supervised learning, RL can **find solutions that the problem specifier did not already know**!

Warning: “Reward Hacking”

- Reward functions as task specifications can be surprisingly hard to get right!

```
def reward_function(params):
    '''
    A complex reward function for a robot arm reaching a specific target position and
orientation.
    '''
    # Set up the target position and orientation
    target_pos = [0.5, 0.5, 0.5]
    target_orient = [0.0, 0.0, 0.0, 1.0]

    # Get the current position and orientation of the robot arm
    robot_pos = params['position']
    robot_orient = params['orientation']

    # Calculate the distance to the target position and orientation
    pos_diff = math.sqrt((robot_pos[0] - target_pos[0])**2 + (robot_pos[1] -
target_pos[1])**2 + (robot_pos[2] - target_pos[2])**2)
    orient_diff = np.linalg.norm(np.subtract(robot_orient, target_orient))

    # Penalize the robot for being too far away from the target position or orientation
    if pos_diff > 0.1 or orient_diff > 0.1:
        reward = -1.0
    else:
        # Calculate a reward based on the proximity to the target position and orientation
        pos_reward = (1.0 - pos_diff) ** 2
        orient_reward = (1.0 - orient_diff) ** 2

        # Penalize the robot for moving too much
        m

        #
        r

    retur
```

Reward hacking is the flip side of the cool thing about RL: it can **find solutions that the problem specifier did not already know**!

Key Problems Specific to RL

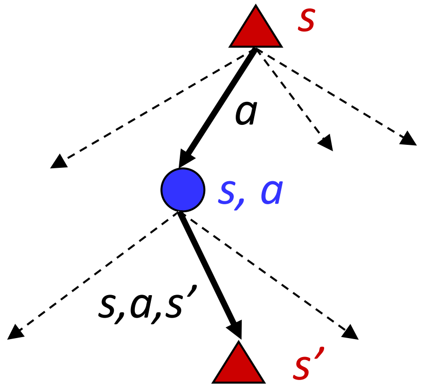
- Credit assignment: Which actions in a sequence were the good/bad ones?
- Exploration vs Exploitation: Yes, trial-and-error, but smartly pick what to try?

Value Functions and Bellman Equations

The Laws Governing Expected Future Rewards

State Value Functions $V(s)$ of Policies

Given MDP (S, A, P, R, γ):



Value of a state s under policy π :

$V^{\pi}(s)$ = expected return when starting in s and acting according to π

$$V^{\pi}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s\right)$$

Rewards generated by following π

Optimal value of a state s :

$V^{*}(s)$ = expected return when starting in s and acting optimally

$$V^{*}(s) = V^{\pi^*}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s\right)$$

Rewards generated by following optimal policy π^*

Note: The optimal policy π^* must maximize expected return $V^{\pi}(s)$

Bellman Equation #1: for (arbitrary) V^{π} functions

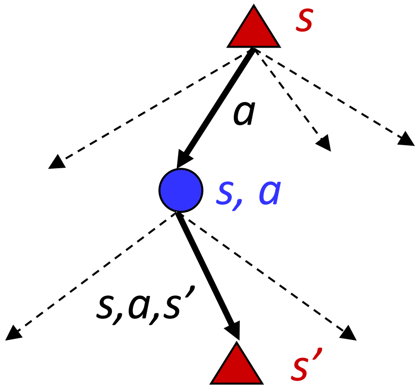
Value of a state s under policy π :

$V^{\pi}(s)$ = expected utility when starting in s and acting according to π

$$V^{\pi}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t \, r_{t+1} | S_0 = s\right)$$

Rewards generated by following π

- The Bellman equations connect value functions at consecutive timesteps:

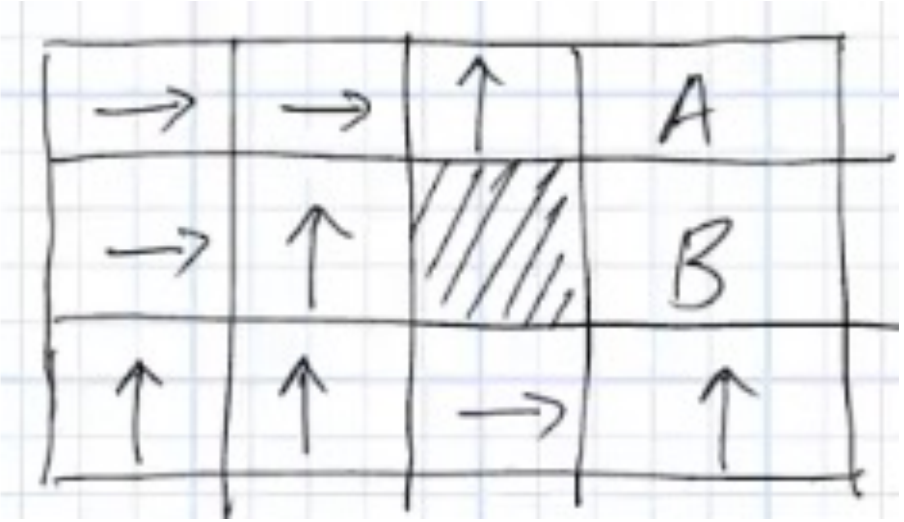


$$V^{\pi}(s) = \sum_{\substack{s' \in S \\ a = \pi(s)}} \underbrace{P(s'|s,a)}_{\text{expected value over successor state s'}} \underbrace{[R(s,a,s') + \gamma V^{\pi}(s')]}_{\text{current reward + discounted future reward}}$$

$$V^{\pi}(s) = \sum_{s' \in S} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

- Expectation (under X) of a random variable $Y(X)$
  - $\mathbb{E}_X[Y(X)] = \sum_x y(x) \, P_X(x)$

(Scratch page)



Value of a state s under policy π :

$V^{\pi}(s)$ = expected utility when starting in s and acting according to π

$$V^{\pi}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^{t}\, r_{t+1} | S_0 = s\right)$$

Rewards generated by following π

Bellman equation for arbitrary V^{π}:

$$V^{\pi}(s) = \sum_{\substack{s' \in S \\ a=\pi(s)}} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

Bellman Equation #2: for optimal V^* functions

$$V^\pi(s) = \sum_{s' \in S} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

What if we followed the optimal policy π^*? Could just plug $\pi = \pi^*$ into the above expression, but even without knowing π^*, you can say:

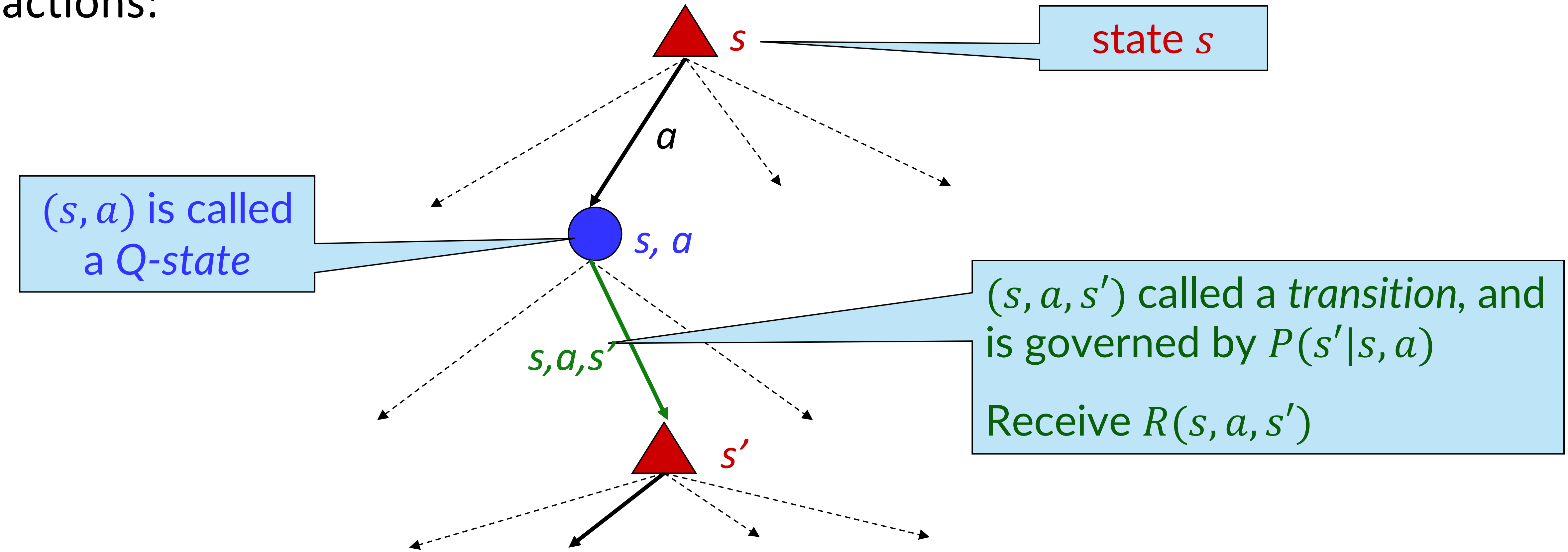
$$V^*(s) = V^{\pi^*}(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

Optimal state value of s is what we get by following the optimal policy, i.e., picking the optimal action. i.e., Optimal policy selects actions that maximize expected return, i.e. actions that maximize value.

Note that this is defined without assuming that you already know the optimal policy π^*. Indeed, we can first find the optimal value function and then derive the policy from it. Coming up soon!

"Q-States"

- Each MDP state has an associated tree of future outcomes from various actions:

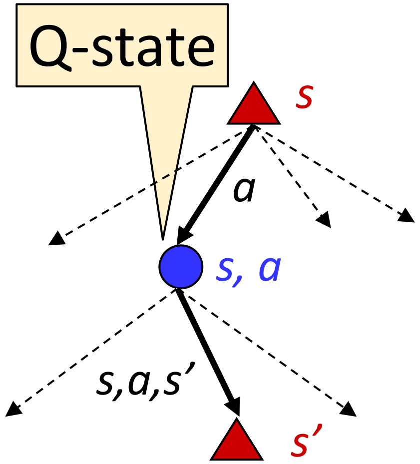


Based on slide by Dan Klein

Action Value Functions $Q(s,a)$ of Policies

$$V^\pi(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s\right)$$

- It is also helpful to define action-value functions, because they are helpfully connected to policies



Q-value of taking action a in state *s* then following policy π :

$Q^\pi(s,a)$ = expected return when taking *a* in *s* and then following π

$$Q^\pi(s,a) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s, A_0 = a\right)$$

Optimal Q-value: $Q^*(s,a) = Q^{\pi^*}(s,a)$

Given Q*, can you select optimal actions?

Yes, π^* can be **greedily** determined from Q^*: $\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$

In other words, knowing/learning Q^* would be sufficient to act optimally (assuming you can solve the argmax)!

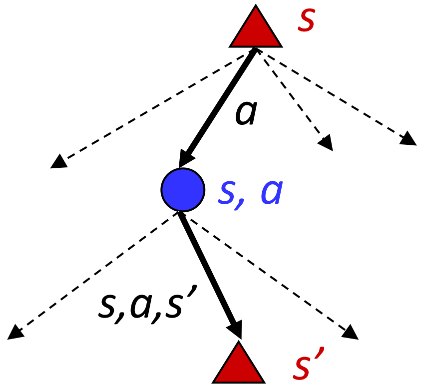
Bellman Equation #3: for (arbitrary) Q^π functions

Q-value of taking action a in state s then following policy π :

$Q^\pi(s,a)$ = expected utility taking a in s and then following π

$$Q^\pi(s,a) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s, A_0 = a\right)$$

- Action-value functions also have their own Bellman equations:



$$Q^\pi(s,a) = \underbrace{\sum_{s' \in S} P(s'|s,a)}_{\text{expected value over successor state s'}} \underbrace{[R(s,a,s') + \gamma Q^\pi(s', \pi(s'))]}_{\text{current reward + discounted future reward}}$$

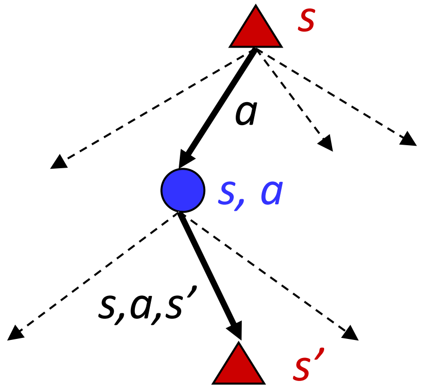
$$Q^\pi(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q^\pi(s', \pi(s'))]$$

- Expectation (under X) of a random variable $Y(X)$
  - $\mathbb{E}_X[Y(X)] = \sum_x y(x)\, P_X(x)$

Bellman Equation #4: for optimal Q functions

$$\boxed{Q^{\pi}(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma Q^{\pi}(s',\pi(s'))]}$$

Optimal values are what we get by picking the optimal action



$$Q^*(s,a) = Q^{\pi^*}(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$$

Recall:
$\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$

Recap

- Markov Decision Processes (S, A, P, R, γ)
- RL wants to find the optimal policy π^* to maximize $\sum_t \gamma^t r_{t+1}$
- One way to do this is to find the optimal Q function, $Q^*(s, a)$
- $Q^*(s, a)$ satisfies a recursive equation, called the Bellman equation

Coming up next:

- How to compute Q^* if we knew the full MDP (S, A, P, R, γ)?
  - "Q-policy and Q-value iteration". (Also briefly V-value iteration)
- How to *learn* Q^* from experience if we only had (S, A, γ) and didn't know P, R?
  - "Q learning", a widely used RL algorithm