Announcements

- **Project Milestone 2 due tonight at 8pm**

- Homework 6 due November 29
  - You have 3 weeks!

Lecture 22: Reinforcement Learning

CIS 4190/5190

Fall 2023

Optimal Action-Value Function

- **Optimal Action-Value Function (or Q function):** Expected reward if we start in s, take action a, and then act optimally thereafter:

$$Q^*(s,a) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t \cdot r_t \mid s_0 = s, a_0 = a\right)$$

- **Bellman equation:**

$$Q^*(s,a) = \sum_{s' \in S} P(s' \mid s,a) \cdot \left(R(s,a,s') + \gamma \cdot \max_{a' \in A} Q^*(s',a')\right)$$

Q Iteration

- We have

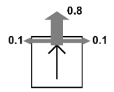
$$\pi^*(s) = \max_{a \in A} Q^*(s, a)$$

- **Strategy:** Compute Q^* and then use it to compute π^*

Q Iteration

- Initialize $Q_1(s,a) \leftarrow 0$ for all s, a
- For $i \in \{1,2,\dots\}$ until convergence:

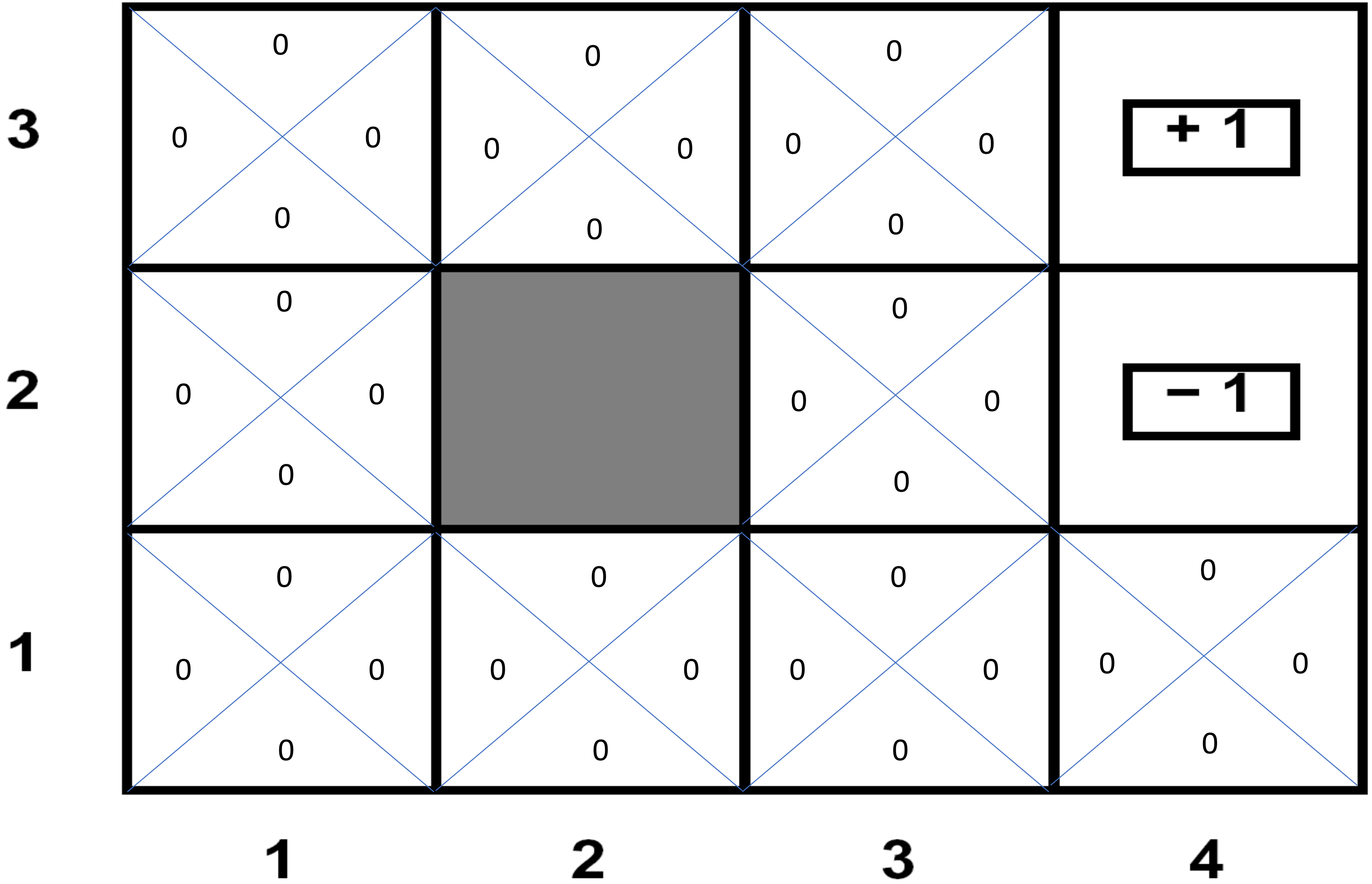
$$Q_{i+1}(s,a) \leftarrow \sum_{s' \in S} P(s' \mid s,a) \cdot \left(R(s,a,s') + \gamma \cdot \max_{a' \in A} Q_i(s',a')\right)$$

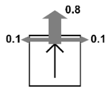


Living cost 0

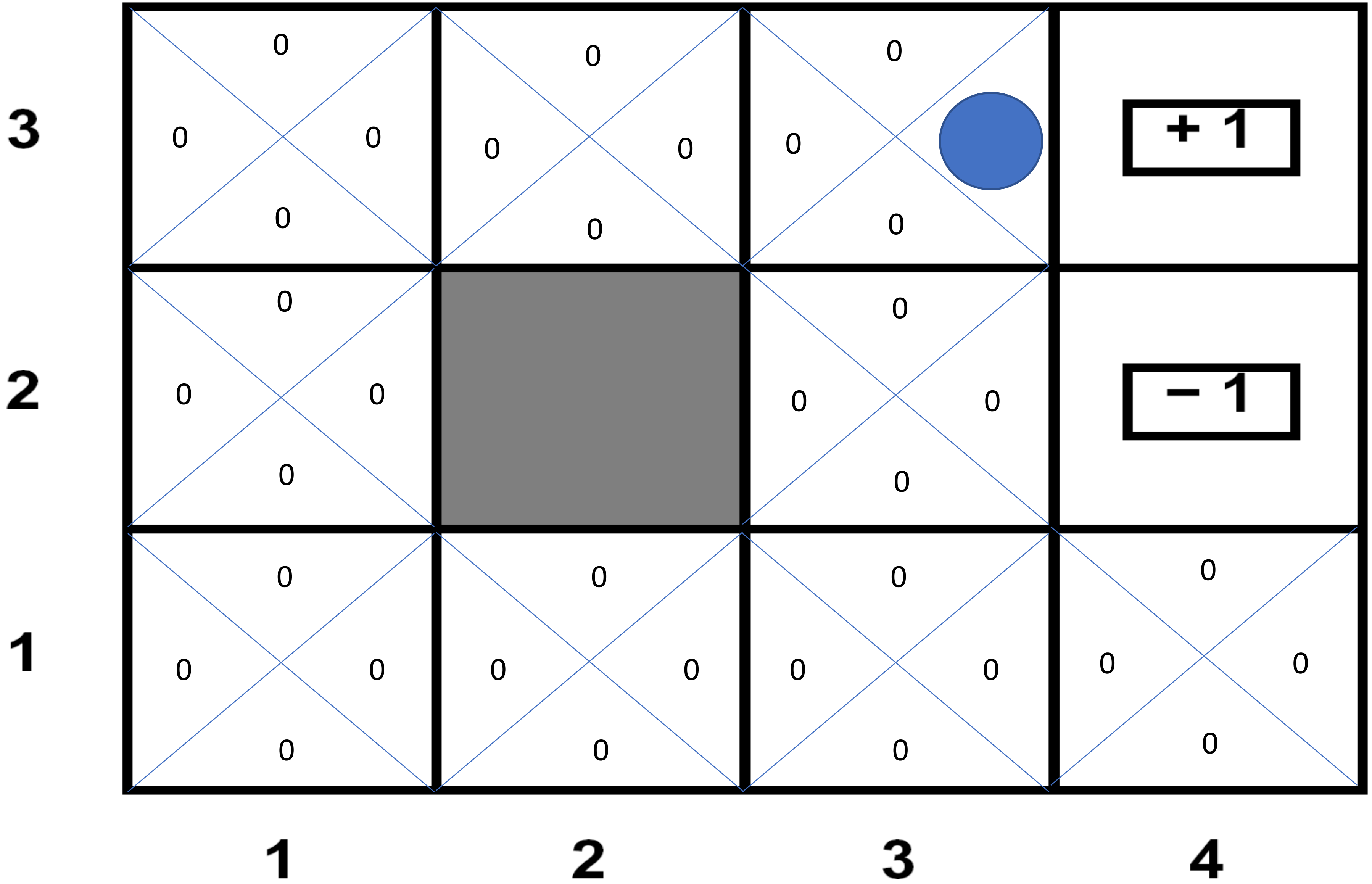
0.9

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \boxed{\max_{a'} Q_i(s',a')}\right]$$

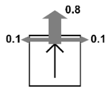




$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[\overset{0}{R(s,a,s')} + \gamma \overset{0.9}{\boxed{\max_{a'} Q_i(s',a')}}\right]$$

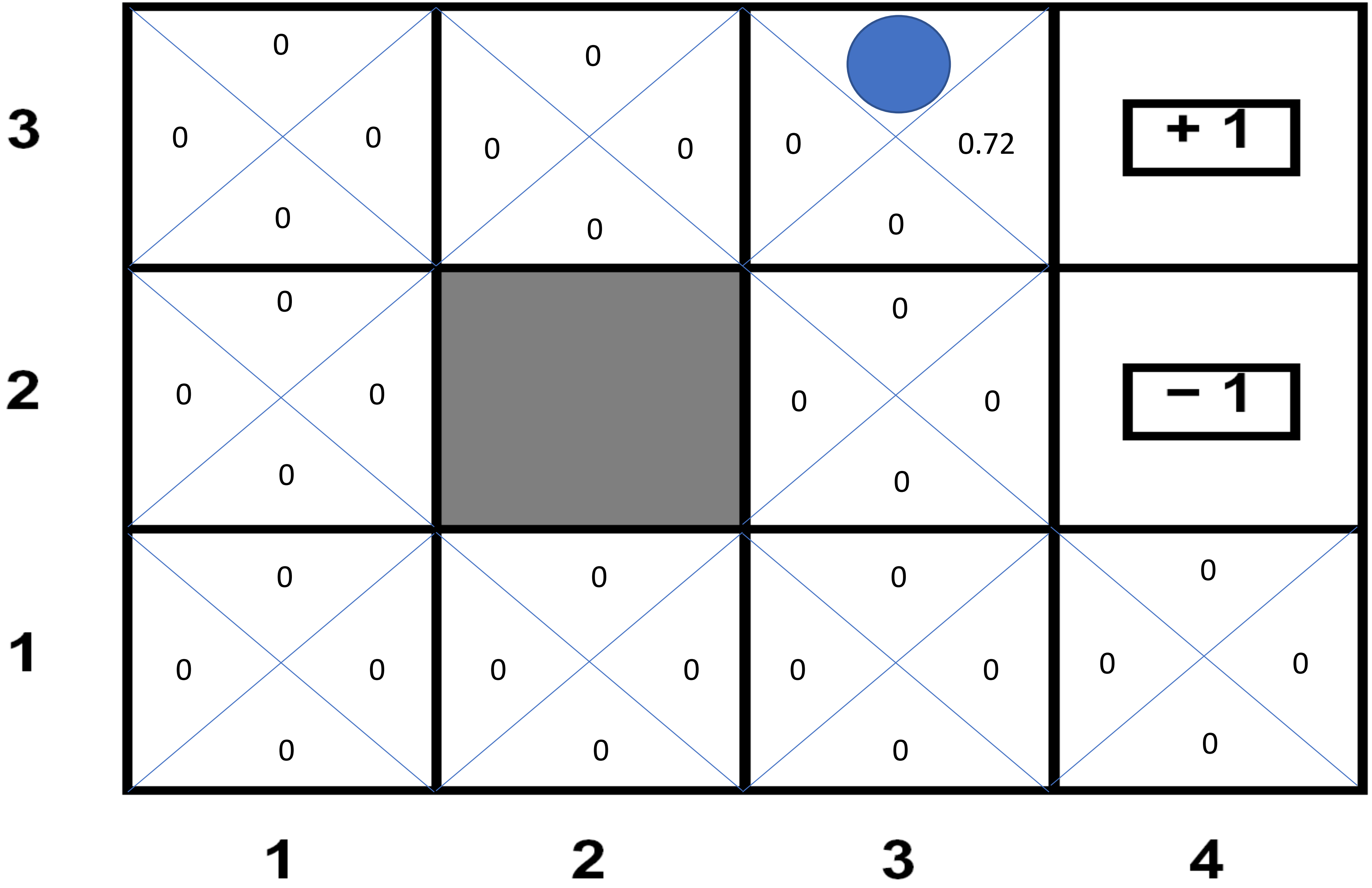


0.8x[0+0.9x1]
+ 0.1x[0 + 0]
+0.1x[0+0]
=0.72

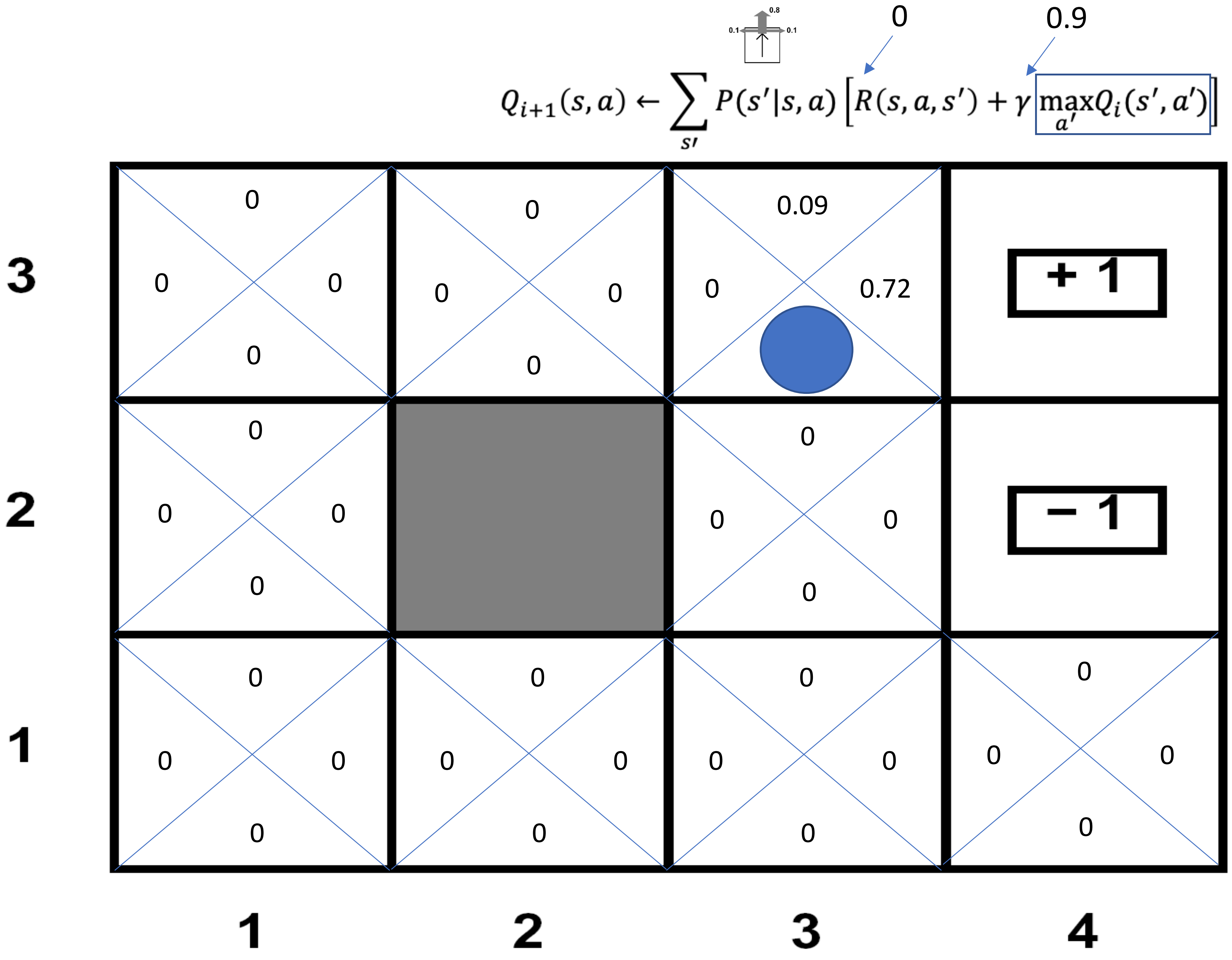


$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[\overset{0}{R(s,a,s')} + \overset{0.9}{\gamma}\, \boxed{\max_{a'} Q_i(s',a')}\right]$$

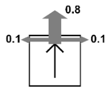
0.8x[0+0]
+ 0.1x[0+0.9x1]
+0.1x[0+0]
=0.09



$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \boxed{\max_{a'} Q_i(s',a')}\right]$$



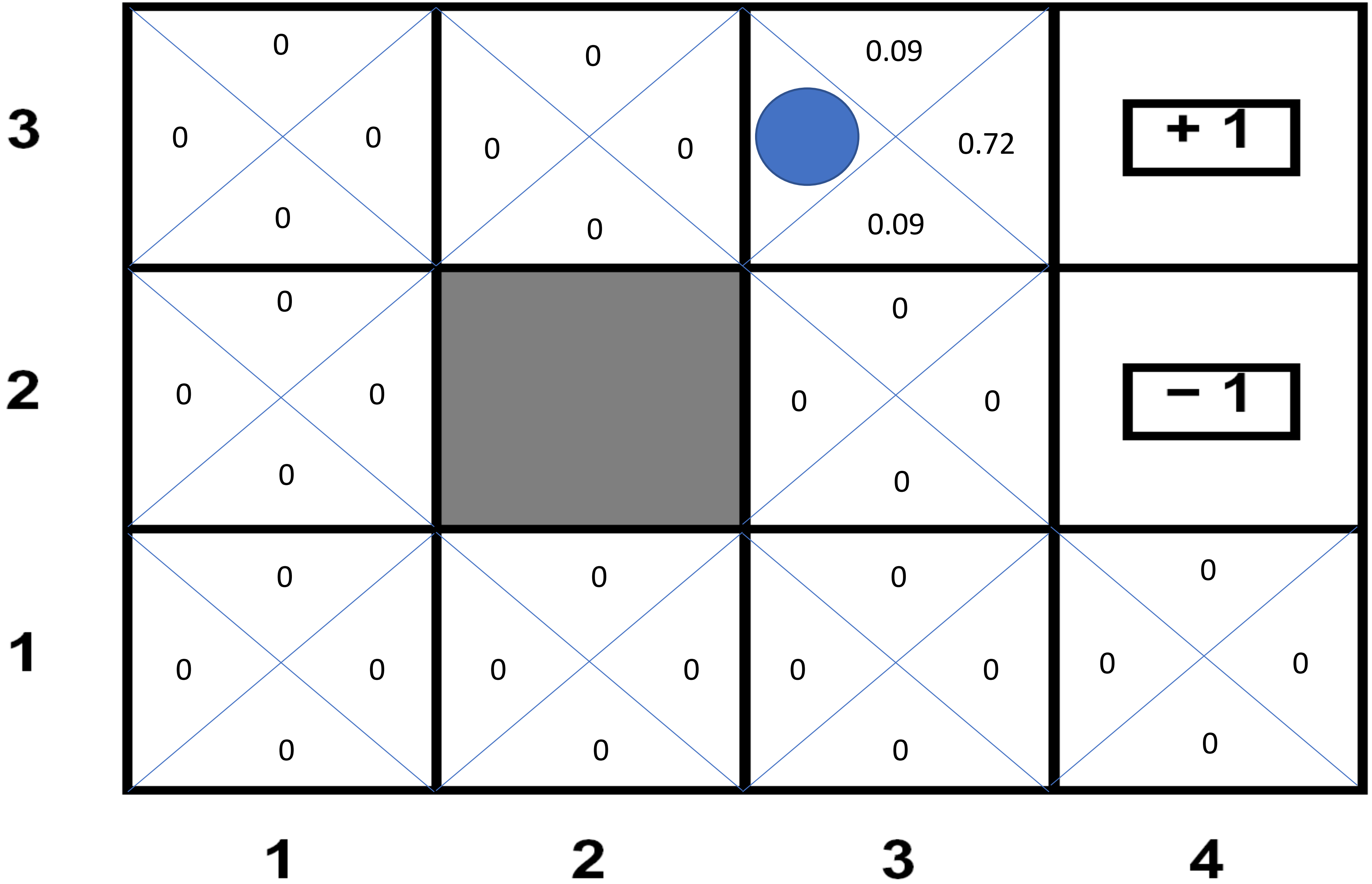
0.8x[0+0]
+ 0.1x[0+0.9x1]
+0.1x[0+0]
=0.09



0 0.9

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \boxed{\max_{a'} Q_i(s',a')}\right]$$

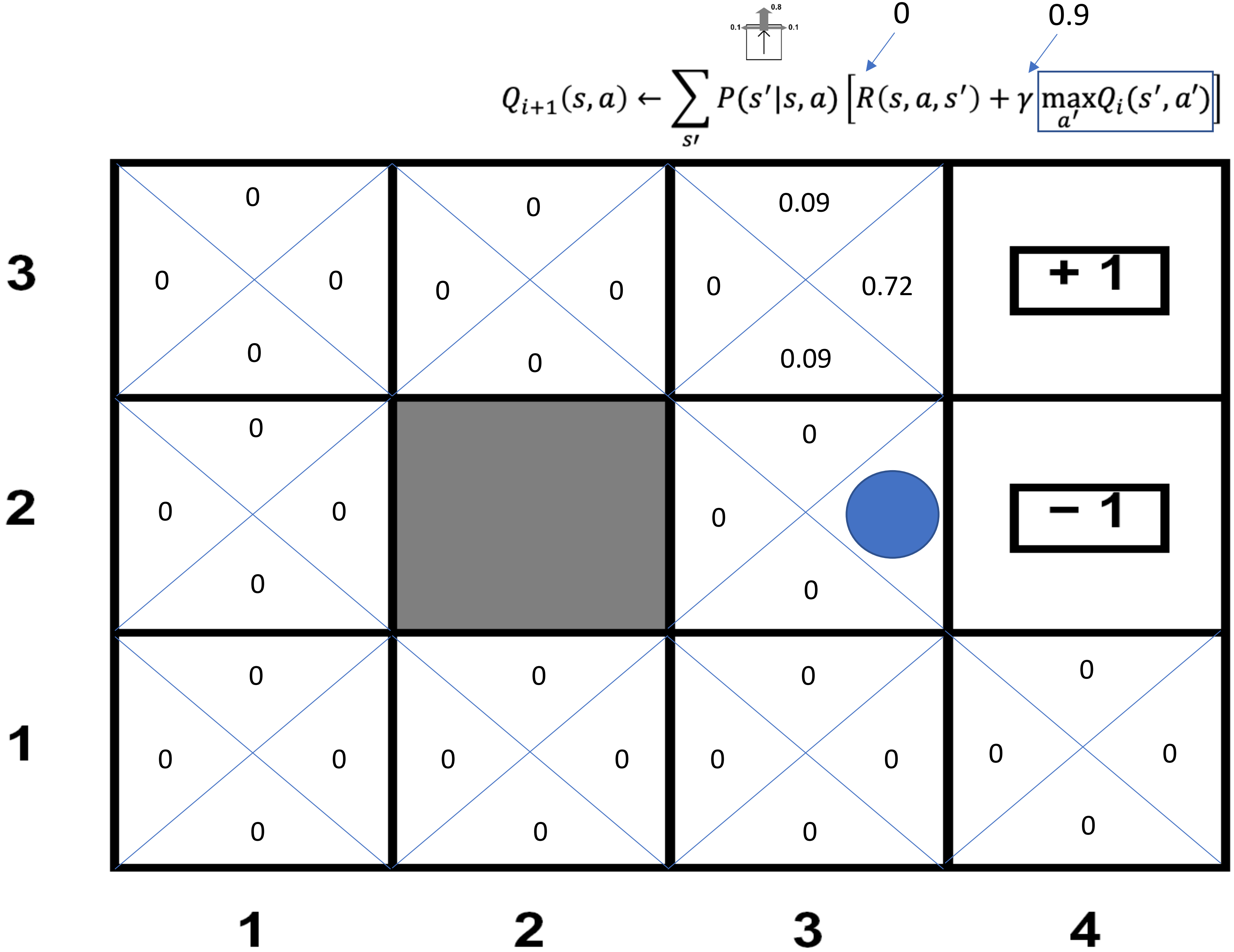
0.8x[0+0]
+ 0.1x[0+0]
+0.1x[0+0]
=0



$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \boxed{\max_{a'} Q_i(s',a')}\right]$$

0

0.9

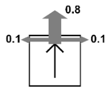


0.8x[0+0.9x-1]

+ 0.1x[0+0]

+0.1x[0+0]

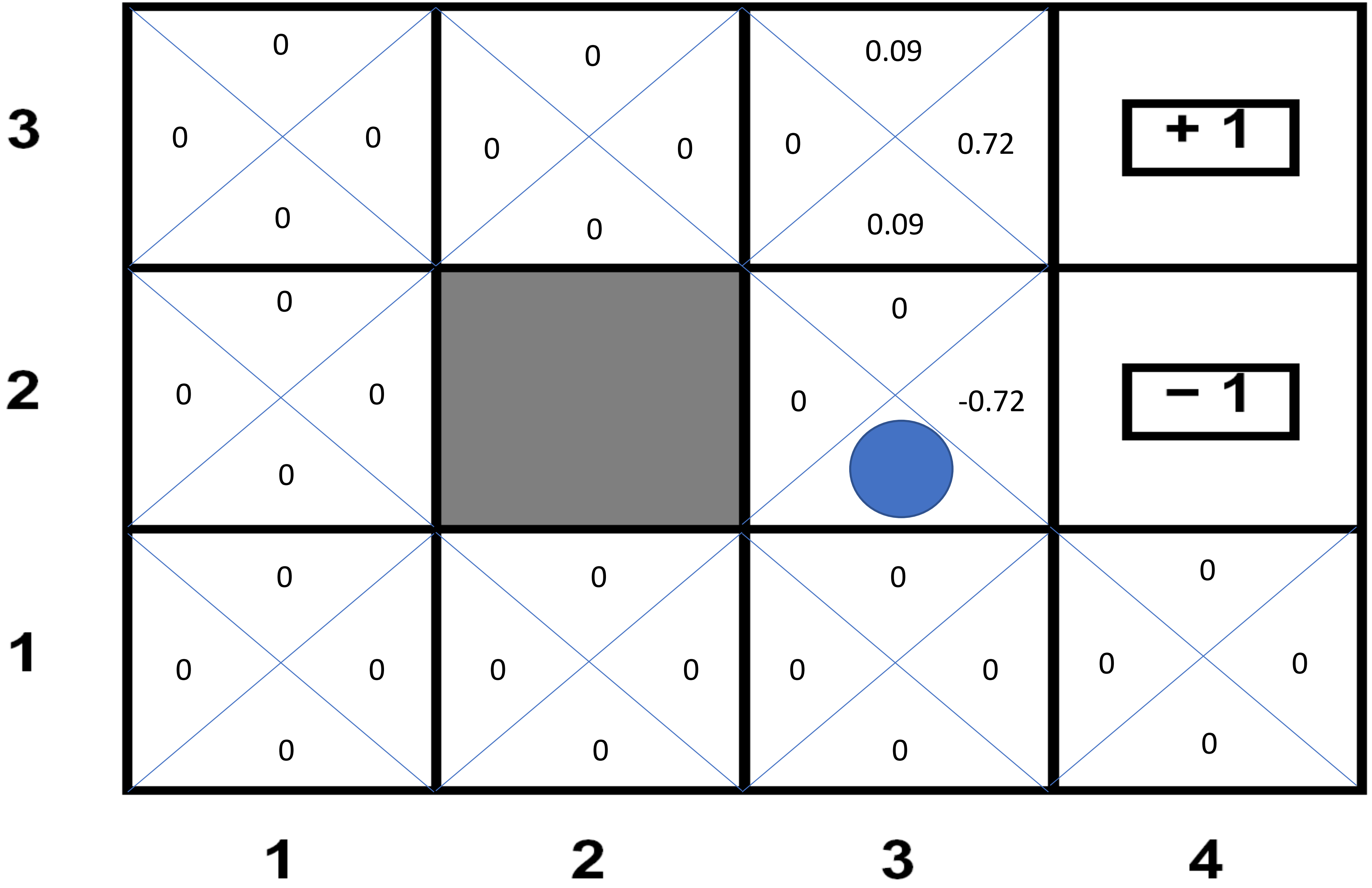
=-0.72

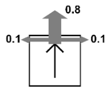


0　　0.9

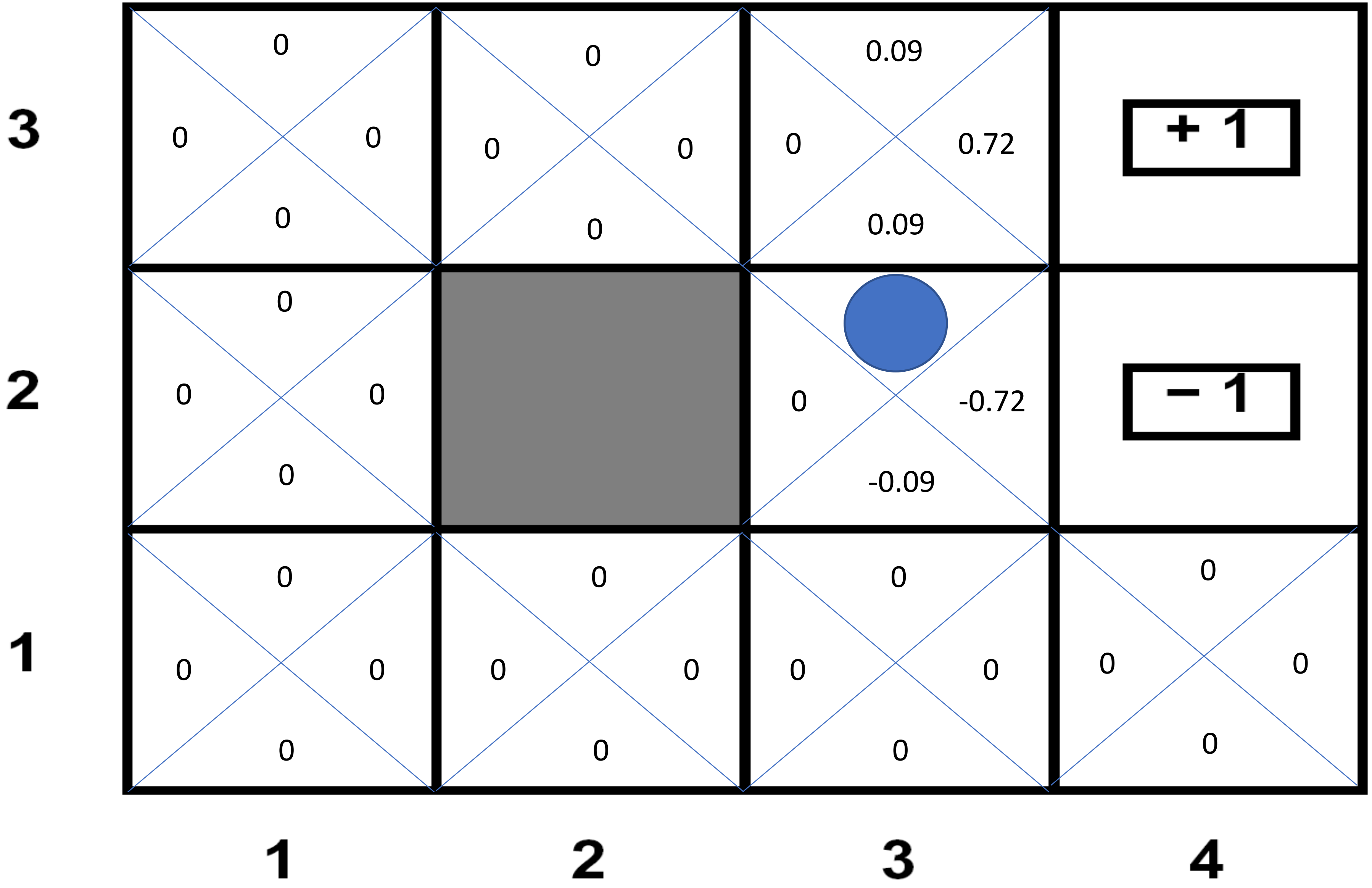
$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \boxed{\max_{a'} Q_i(s',a')}\right]$$

0.8x[0+0]
+ 0.1x[0+0]
+0.1x[0+0.9x-1]
=-0.09

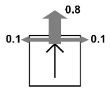




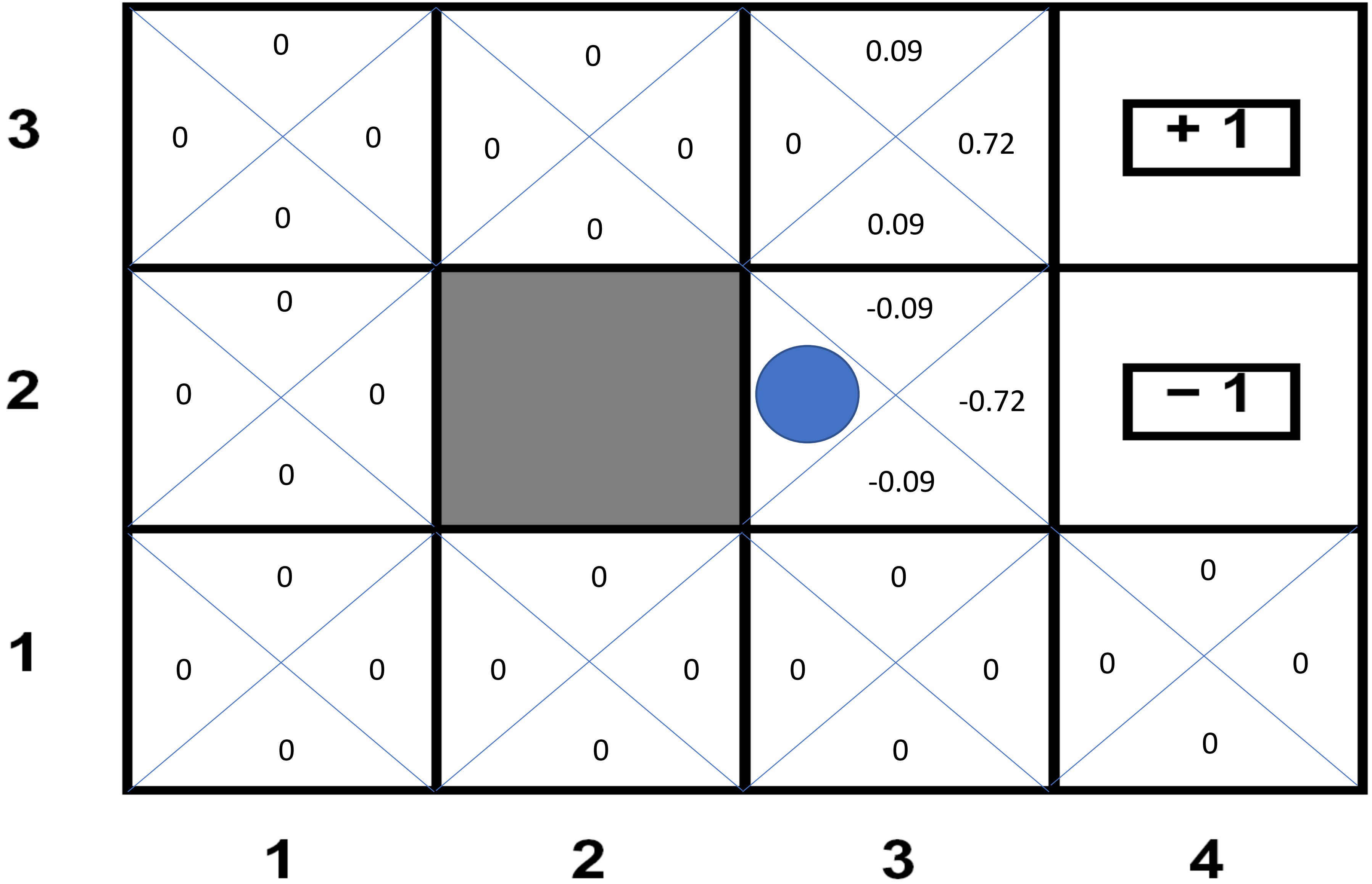
$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[\overset{0}{R(s,a,s')} + \overset{0.9}{\gamma}\, \boxed{\max_{a'} Q_i(s',a')}\right]$$



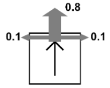
0.8x[0+0]
+ 0.1x[0+0.9x-1]
+0.1x[0+0]
=-0.09



$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[\overset{0}{R(s,a,s')} + \gamma \overset{0.9}{\boxed{\max_{a'} Q_i(s',a')}}\right]$$



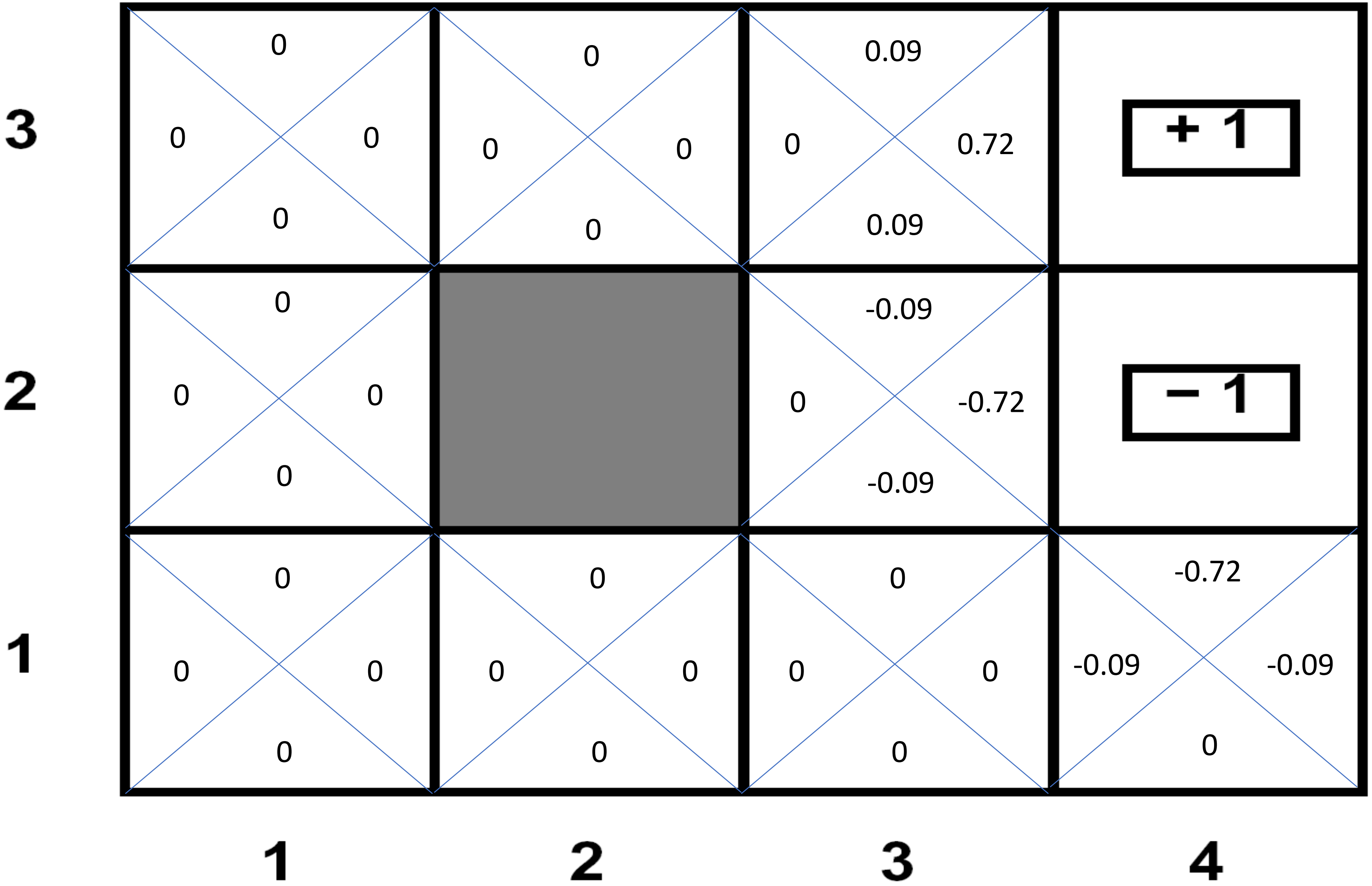
0.8x[0+0]
+ 0.1x[0+0]
+0.1x[0+0]
=0

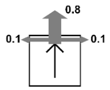


0 0.9

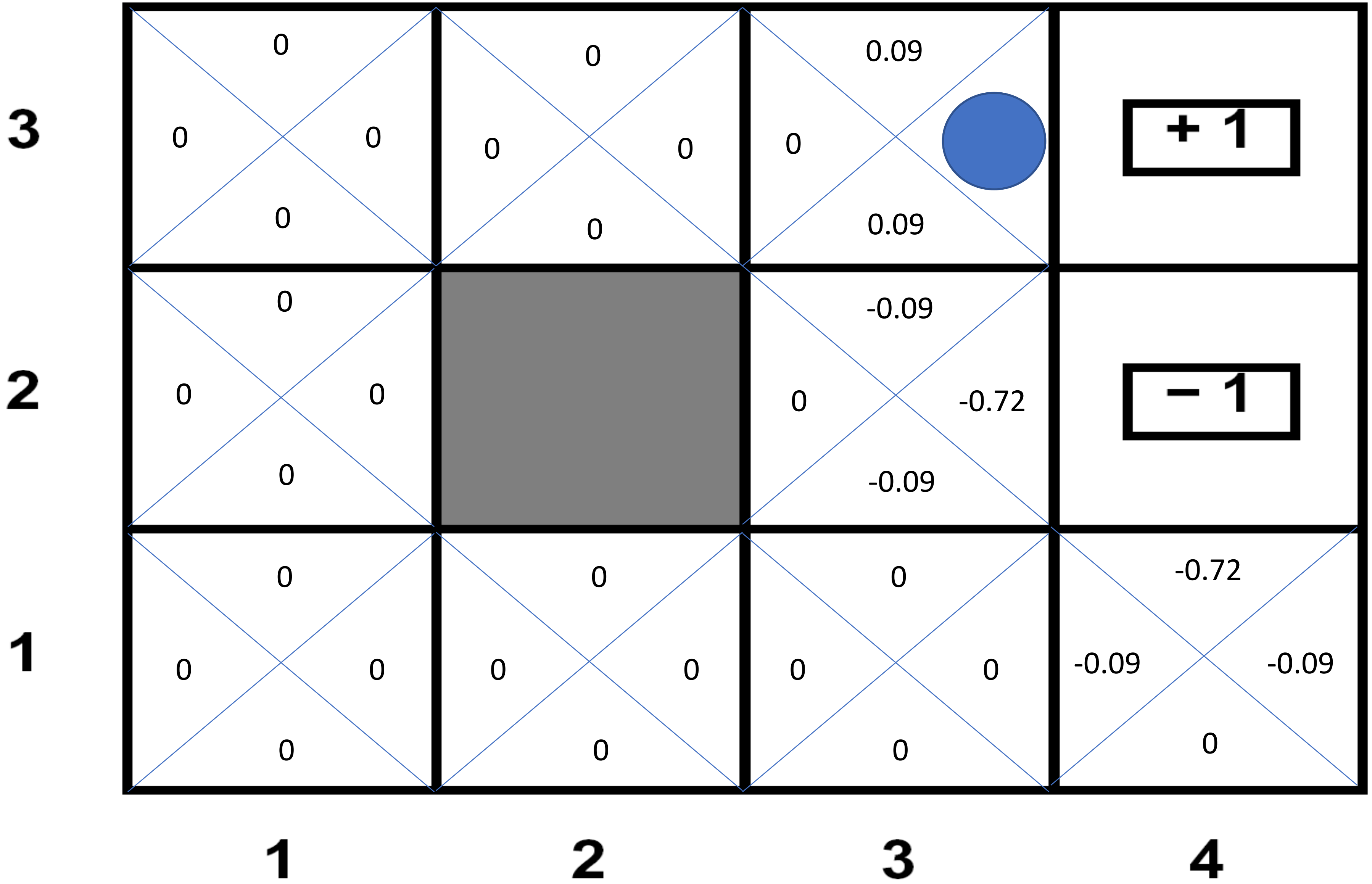
$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \boxed{\max_{a'} Q_i(s',a')}\right]$$

Now we have
$Q_1(s,a)$ for all (s,a)

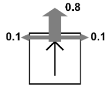




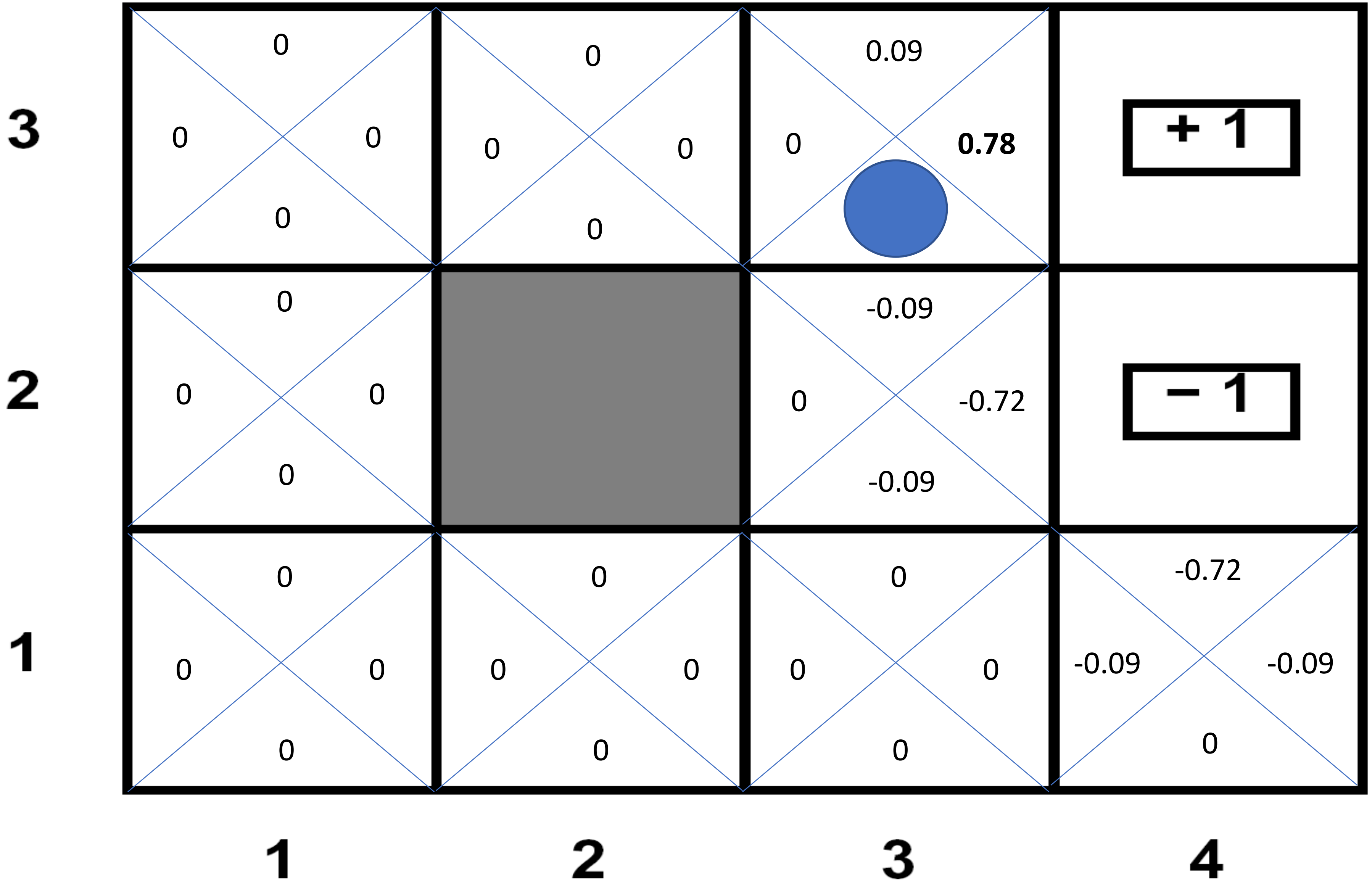
$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[\overbrace{R(s,a,s')}^{0} + \overset{0.9}{\gamma}\boxed{\max_{a'} Q_i(s',a')}\right]$$



0.8x[0+0.9x1]
+ 0.1x[0+0.9x0.72]
+0.1x[0+0]
=0.7848

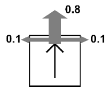


$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[\overbrace{R(s,a,s')}^{0} + \overbrace{\gamma}^{0.9} \boxed{\max_{a'} Q_i(s',a')}\right]$$



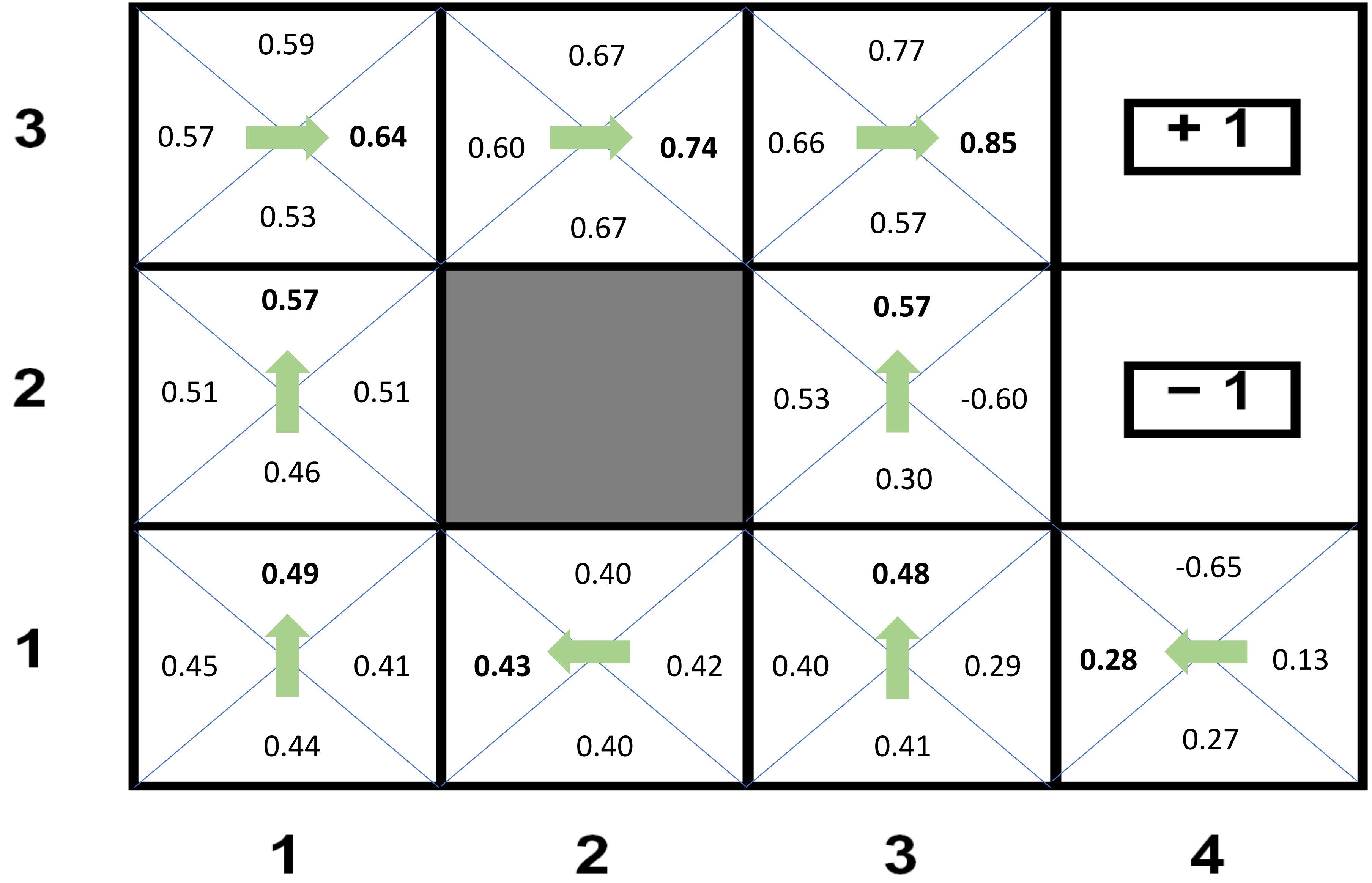
0.8x[0+0]
+ 0.1x[0+0.9x1]
+0.1x[0+0]
=0.09

After 1000 iterations:



0 0.9

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \boxed{\max_{a'} Q_i(s',a')}\right]$$



Reinforcement Learning

- Q iteration can be used to compute the optimal Q function when P and R are **known**

- How can we adapt it to the setting where these are unknown?
  - **Observation:** Every time you take action a from state s, you obtain one sample $s' \sim P(\cdot \mid s, a)$ and observe $R(s, a, s')$
  - Use single sample instead of full P

Q Learning

- Can we learn π^* without explicitly learning P and R?

$$Q_{i+1}(s,a) \leftarrow \sum_{s' \in S} P(\, s' \mid s, a \,) \cdot \left(R(s,a,s') + \gamma \cdot \max_{a' \in A} Q_i(s', a') \right)$$

Q Learning

- Can we learn π^* without explicitly learning P and R?

$$Q_{i+1}(s, a) \leftarrow \mathbb{E}_{s' \sim P(\cdot|s, a)}\left[R(s, a, s') + \gamma \cdot \max_{a' \in A} Q_i(s', a')\right]$$

Q Learning

- **Q Learning update:**

$$Q_{i+1}(s,a) \leftarrow R(s,a,s') + \gamma \cdot \max_{a' \in A} Q_i(s',a')$$

- **Q Iteration:** Update for all (s,a,s') at each step

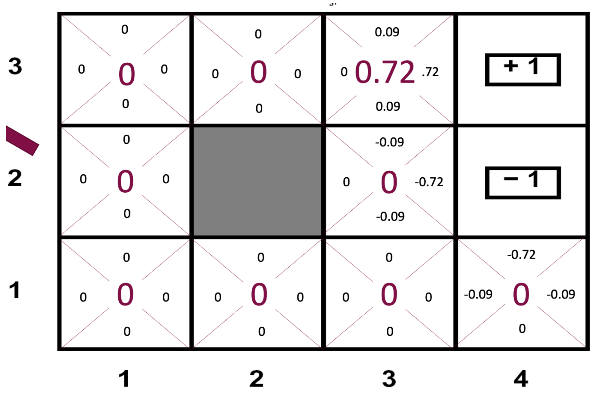
- **Q Learning:** Update just for current (s,a), and approximate with the state s' we actually reached (i.e., a single sample $s' \sim P(\cdot \mid s,a)$)

Q Learning

- **Problem:** Forget everything we learned before (i.e., $Q_i(s,a)$)
- **Solution:** Incremental update:

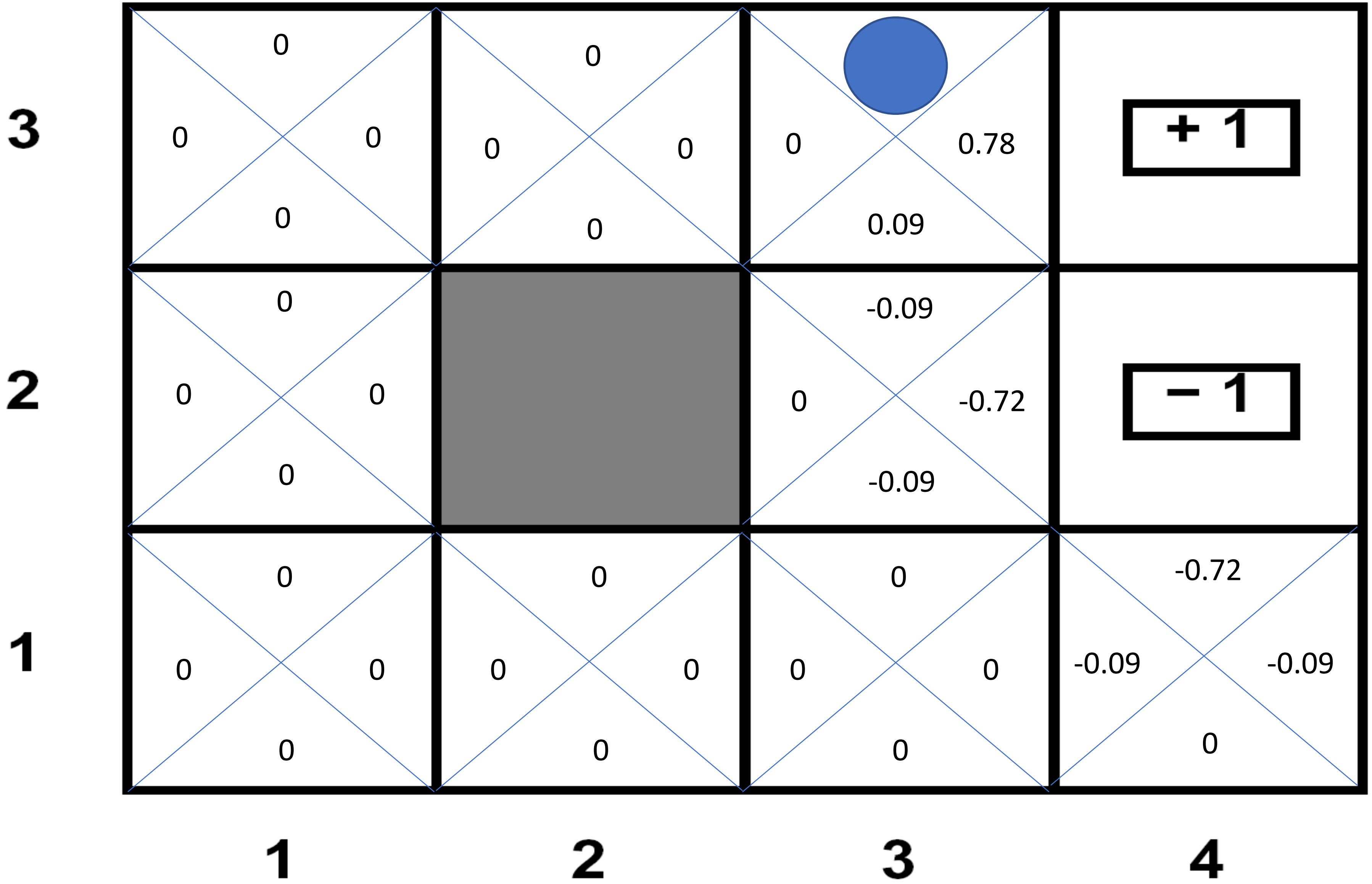
$$Q_{i+1}(s,a) \leftarrow (1-\alpha) \cdot Q_i(s,a) + \alpha \cdot \left(R(s,a,s') + \gamma \cdot \max_{a' \in A} Q_i(s',a')\right)$$

$$Q(s,a) \leftarrow Q(s,a) + \overset{0.1}{\alpha}\left(R(s,a,s') + \overset{0.9}{\gamma} \max_{a'} Q(s',a') - Q(s,a)\right)$$



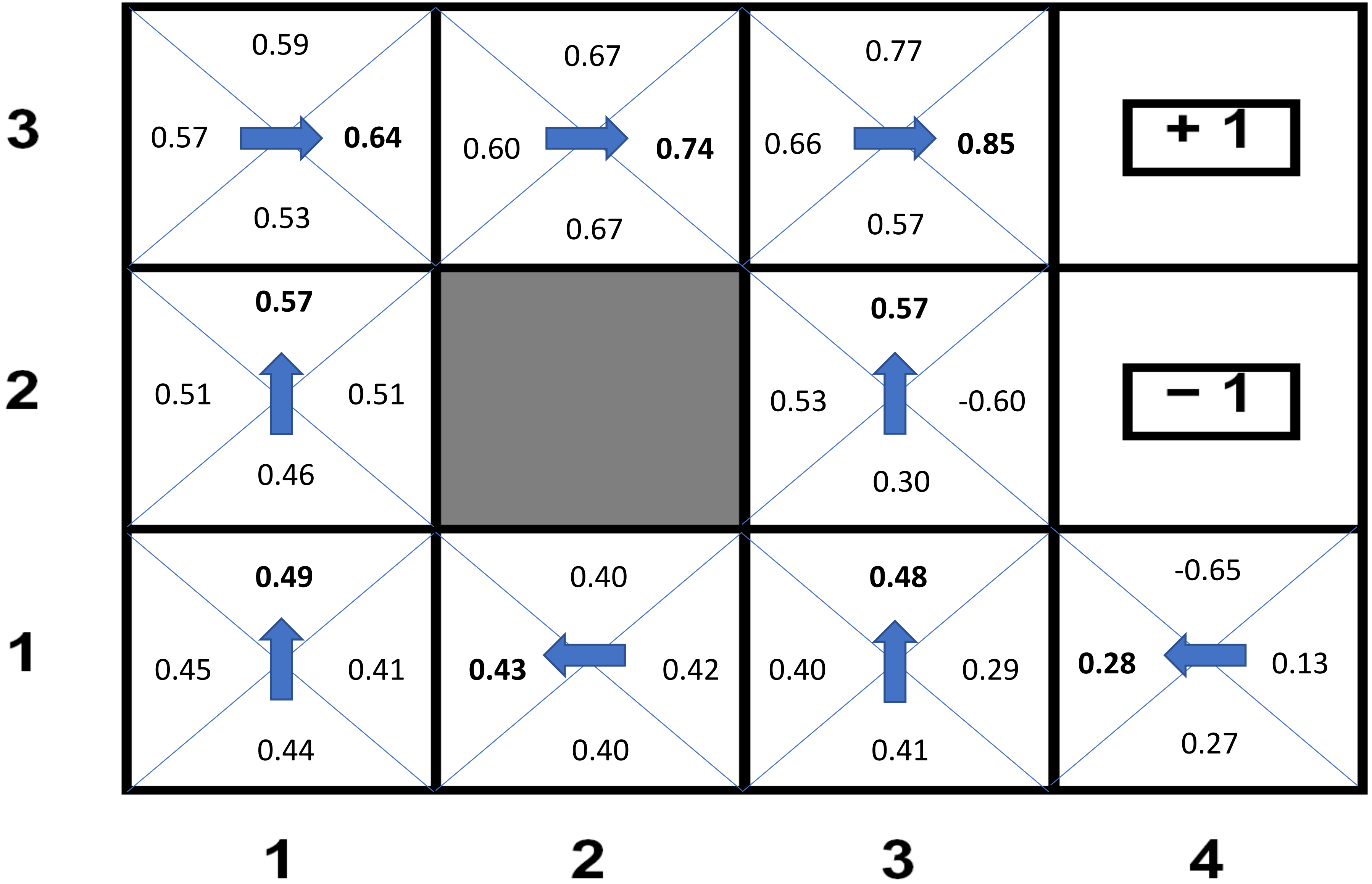
Sample $R + \gamma \max Q$ =
0+0.9x0.78 = 0.702

New Q =
0.09+0.1X(0.702-0.09)
= 0.1512



After 100,000 actions:

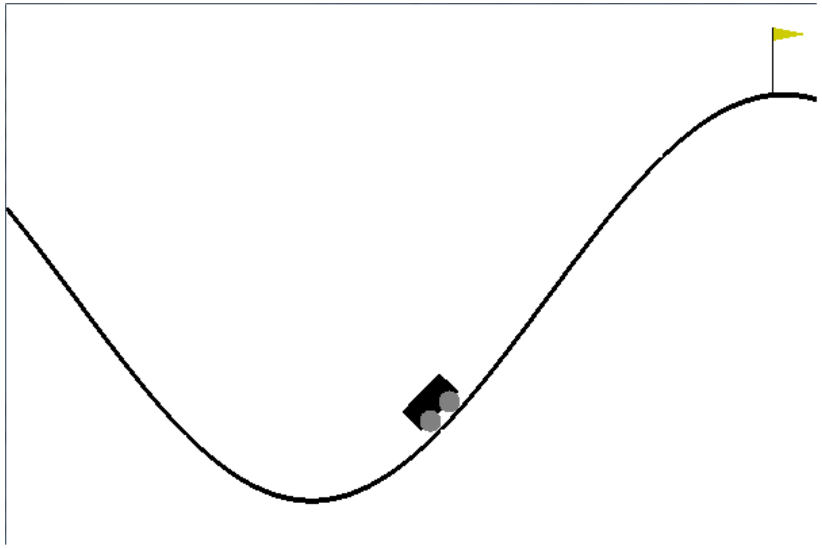
$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$



Policy for Gathering Data

- **Strategy 1:** Randomly explore all (s, a) pairs
  - Not obvious how to do so!
  - E.g., if we act randomly, it may take a very long time to explore states that are difficult to reach

- **Strategy 2:** Use current best policy
  - Can get stuck in local minima
  - E.g., we may never discover a shortcut if it sticks to a known route to the goal



Policy for Gathering Data

- **ϵ-greedy:**
  - Play current best with probability $1 - \epsilon$ and randomly with probability ϵ
  - Can reduce ϵ over time
  - Works okay, but exploration is undirected
- **Visitation counts:**
  - Maintain a count $N(s, a)$ of number of times we tried action a in state s
  - Choose $a^* = \arg\max_{a \in A} \left\{ Q(s, a) + \frac{1}{N(s,a)} \right\}$, i.e., inflate less visited states

Summary

- **Q iteration:** Compute optimal Q function when the transitions and rewards are known

- **Q learning:** Compute optimal Q function when the transitions and rewards are unknown

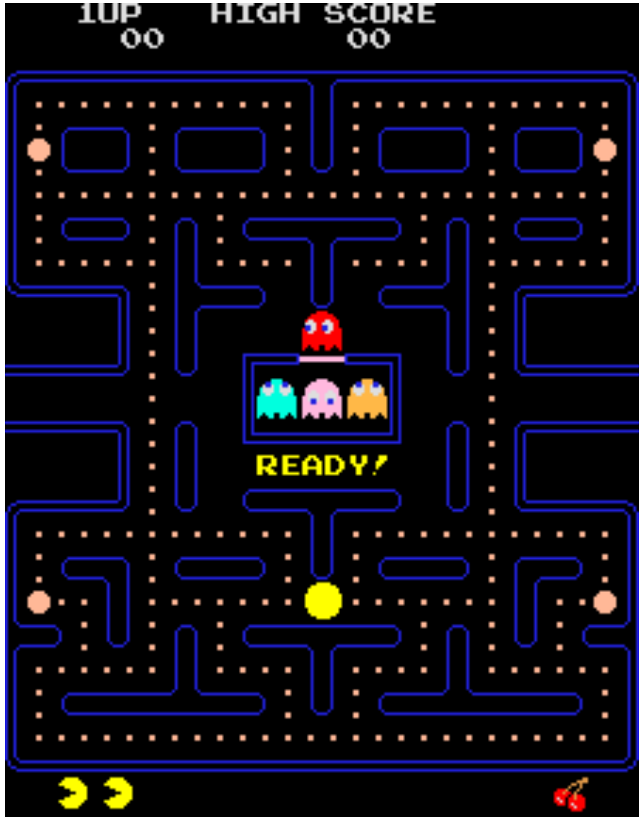
- **Extensions**
  - Various strategies for exploring the state space during learning
  - Handling large or continuous state spaces

Curse of Dimensionality

- How large is the state space?
  - **Gridworld:** One for each of the n cells
  - **Pacman:** State is $(\text{player}, \text{ghost}_1, \ldots, \text{ghost}_k)$, so there are n^k states!

- **Problem:** Learning in one state does not tell us anything about the other states!

- Many states → learn very slowly



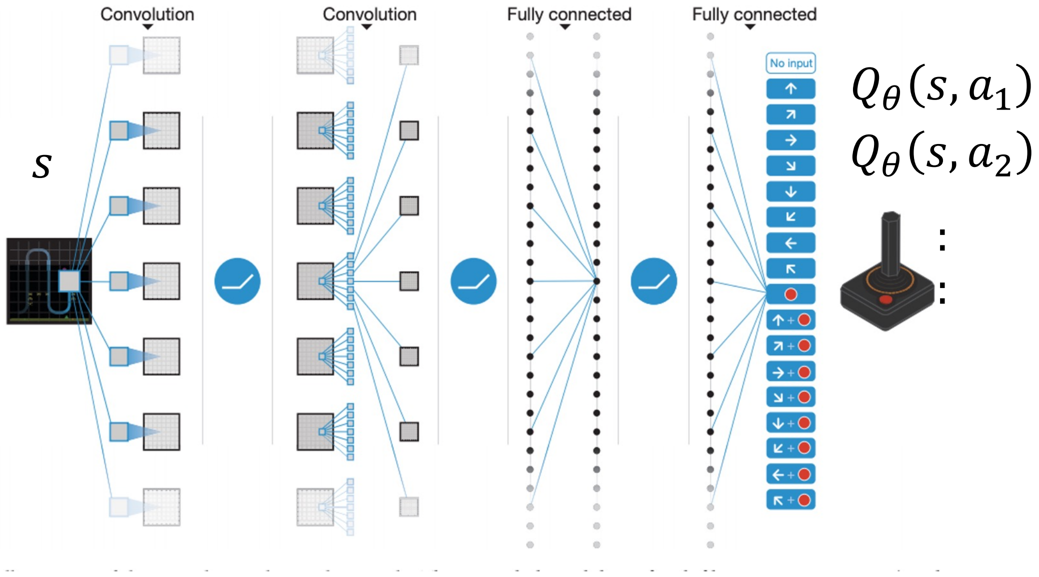
State-Action Features

- Can we learn **across** state-action pairs?

- Yes, use features!
  - $\phi(s, a) \in \mathbb{R}^d$
  - Then, learn to predict $Q^*(s, a) \approx Q_\theta(s, a) = f_\theta\big(\phi(s, a)\big)$
  - Enables generalization to similar states

Neural Network Q Function

- **Examples:** Distance to closest ghost, distance to closest dot, etc.
  - Can also use neural networks to **learn** features (e.g., represent Pacman game state as an image and feed to CNN)!



Deep Q Learning

- **Learning:** Gradient descent with the squared Bellman error loss:

$$\Bigg(\underbrace{\Big(R(s,a,s') + \gamma \cdot \max_{a'} Q_\theta(s',a')\Big)}_{\text{“Label” } y} - Q_\theta(s,a)\Bigg)^2$$

Deep Q Learning

- **Iteratively perform the following:**
  - Take an action a_i and observe (s_i, a_i, s_{i+1}, r_i)
  - $y_i \leftarrow r_i + \gamma \cdot \max_{a' \in A} Q_\theta(s_{i+1}, a')$
  - $\phi \leftarrow \phi - \alpha \cdot \frac{d}{d\theta}(Q_\theta(s_i, a_i) - y_i)^2$

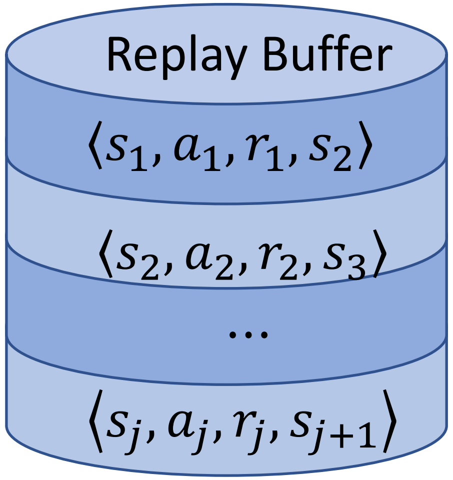
- **Note:** Pretend like y_i is constant when taking the gradient

- For finite state setting, recover incremental update if the "parameters" are the Q values for each state-action pair

Based on slide by Sergey Levine

Experience Replay Buffer

- **Problem**
  - Sequences of states are highly correlated
  - Tend to overfit to current states and forget older states
- **Solution**
  - Keep a **replay buffer** of observations (as a priority queue)
  - Gradient updates on samples from replay buffer instead of current state
- **Advantages**
  - Breaks correlations between consecutive samples
  - Can take multiple gradient steps on each observation

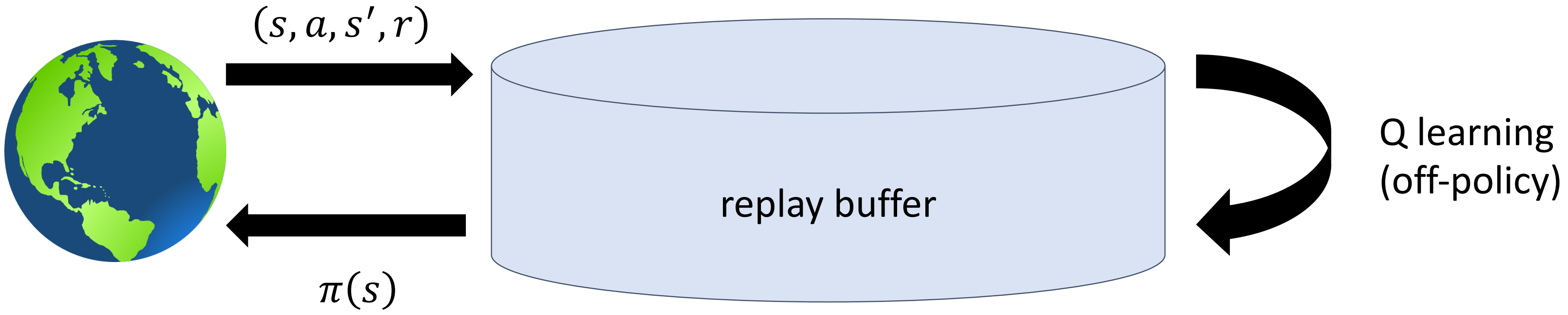


Priority Queue

Based on slide by Sergey Levine

Deep Q Learning with Replay Buffer

- **Iteratively perform the following:**
  - Take an action a_i and add observation (s_i, a_i, s_{i+1}, r_i) to replay buffer D
  - For $k \in \{1, \ldots, K\}$:
    - Sample $(s_{i,k}, a_{i,k}, s_{i+1,k}, r_{i,k})$ from D
    - $y_{i,k} \leftarrow r_{i,k} + \gamma \cdot \max_{a' \in A} Q_\theta(s_{i+1,k}, a')$
    - $\theta \leftarrow \theta - \alpha \cdot \frac{d}{d\theta}\left(Q_\theta(s_{i,k}, a_{i,k}) - y_{i,k}\right)^2$



Based on slide by Sergey Levine

Target Q Network

- **Problem**
  - Q network occurs in the label y_i!
  - $\theta \leftarrow \theta - \alpha \cdot \frac{d}{d\theta}\left(Q_\theta(s_i, a_i) - r_i + \gamma \cdot \max_{a' \in A} Q_\theta(s_{i+1}, a')\right)^2$
  - Thus, labels change as Q network changes (distribution shift)

- **Solution**
  - Use a separate **target Q network** for the occurrence in y_i
  - Only update target network occasionally
  - $\theta \leftarrow \theta - \alpha \cdot \frac{d}{d\theta}\left(\underbrace{Q_\theta(s_i, a_i)}_{\text{Original Q Network}} - r_i + \gamma \cdot \max_{a' \in A} \underbrace{Q_{\theta'}(s_{i+1}, a')}_{\text{Target Q Network}}\right)^2$

Based on slide by Sergey Levine

Deep Q Learning with Target Q Network

- **Iteratively perform the following:**
  - Take an action a_i and add observation (s_i, a_i, s_{i+1}, r_i) to replay buffer D
  - For $k \in \{1, \ldots, K\}$:
    - Sample $(s_{i,k}, a_{i,k}, s_{i+1,k}, r_{i,k})$ from D
    - $y_{i,k} \leftarrow r_{i,k} + \gamma \cdot \max_{a' \in A} Q_{\theta'}(s_{i+1,k}, a')$
    - $\theta \leftarrow \theta - \alpha \cdot \frac{d}{d\theta}\left(Q_\theta(s_{i,k}, a_{i,k}) - y_{i,k}\right)^2$
  - Every N steps, $\theta' \leftarrow \theta$

Based on slide by Sergey Levine

Deep Q Learning for Atari Games

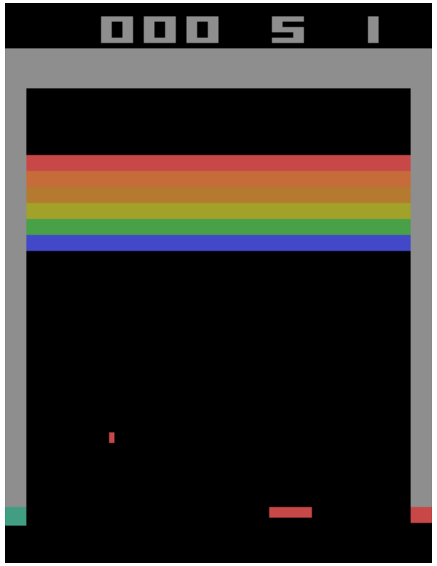


Image Sources:
https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756
https://deepmind.com/blog/going-beyond-average-reinforcement-learning/
https://jaromiru.com/2016/11/07/lets-make-a-dqn-double-learning-and-prioritized-experience-replay/

Aside: Policy Gradient Algorithm

- Directly train policy $\pi_\theta(\, a \mid s\,)$ mapping states to action distributions

- Policy gradient theorem gives the gradient update:

$$\theta \leftarrow \theta + \eta \cdot \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(a_{i,t} \middle| s_{i,t}\right) \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \right)$$

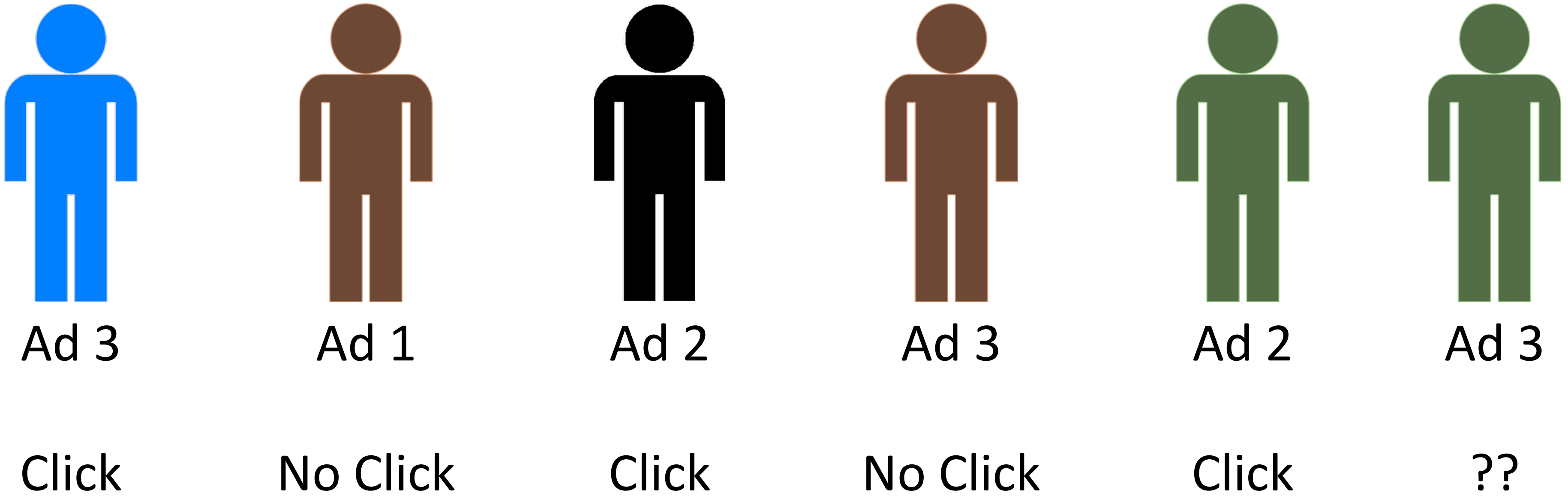
- Can be combined with Q learning to form “actor-critic algorithms”

Multi-Armed Bandits

- **State:** None! (To be precise, a single state $S = \{s_0\}$)
- **Action:** Item to recommend (often called **arms**)
- **Transitions:** Just stay in the same state
- **Rewards:** Random payoff for each arm
  - Denote $R(a) = R(s_0, a)$, where a is the chosen action

Example: Ad Targeting

- **Setting**
  - Google wants to show the most popular ad for a search term (e.g., “lawyer”)
  - There are a fixed number of ads to choose from



Multi-Armed Bandits

- **Many applications**
  - Cold-start for news/ad/movie recommendations
  - A/B testing
  - Flagging potentially harmful content on a social media platform
  - Prioritizing medical tests

- Learning dynamically

- Many practical RL problems are multi-armed bandits

Exploration-Exploitation Tradeoff

- For $t \in \{1,2,\dots,T\}$
  - Compute reward estimates $r_{t,a} = \frac{\sum_{i=1}^{t-1} r_i \cdot 1(a_i = a)}{\sum_{i=1}^{t-1} 1(a_i = a)}$
  - Choose action a_t based on reward estimates
  - Add (a_t, r_t) to replay buffer

- **Question:** How to choose actions?
  - **Exploration:** Try actions to better estimate their rewards
  - **Exploitation:** Use action with the best estimated reward to maximize payoff

Multi-Armed Bandit Algorithms

- **Naïve strategy:** ϵ-Greedy
  - Choose action $a_t \sim \mathrm{Uniform}(A)$ with probability ϵ
  - Choose action $a_t = \arg\max_{a \in A} r_{t,a}$ with probability $1 - \epsilon$

- Can we do better?

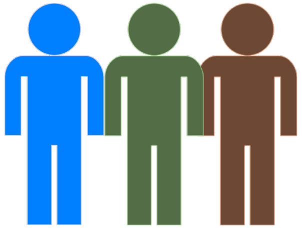
Multi-Armed Bandit Algorithms

- **Upper confidence bound (UCB)**
  - Choose action $a_t = \underset{a \in A}{\arg\max} \left\{ r_{t,a} + \frac{\text{const}}{\sqrt{N_t(a)}} \right\}$
  - $N_t(a) = \sum_{i=1}^{t-1} 1(a_i = a)$ is the number of times action a has been played

- **Thompson sampling**
  - Choose action $a_t = \underset{a \in A}{\arg\max} \{ r_{t,a} + \epsilon_{t,a} \}$, where $\epsilon_{t,a} \sim N\left(0, \frac{\text{const}}{\sqrt{N_t(a)}}\right)$

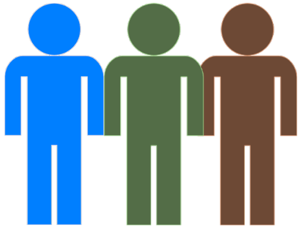
- Both come with theoretical guarantees

Application: Targeted COVID-19 Testing



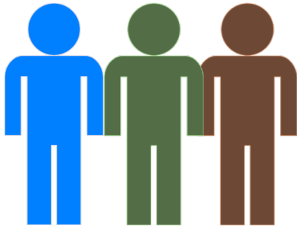
Test Blue

Negative



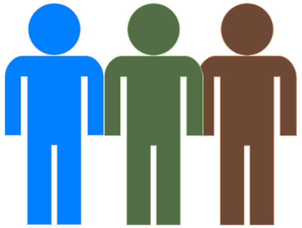
Test Green

Positive



Test Green

Negative



Test Brown

Negative

H. Bastani, K. Drakopoulos, V. Gupta, et al. Efficient and Targeted COVID-19 Border Testing via Reinforcement Learning.

EVA

30k-100k passengers

24 hours prior to travel

PLF form

Travelers report:

- Origin
- Demographics
- Destination
- Contact

EVA

no test

test (6k-8k)

QR code scanned to link sample with PLF info

Labs submit positive results to central database with ~2-day delay

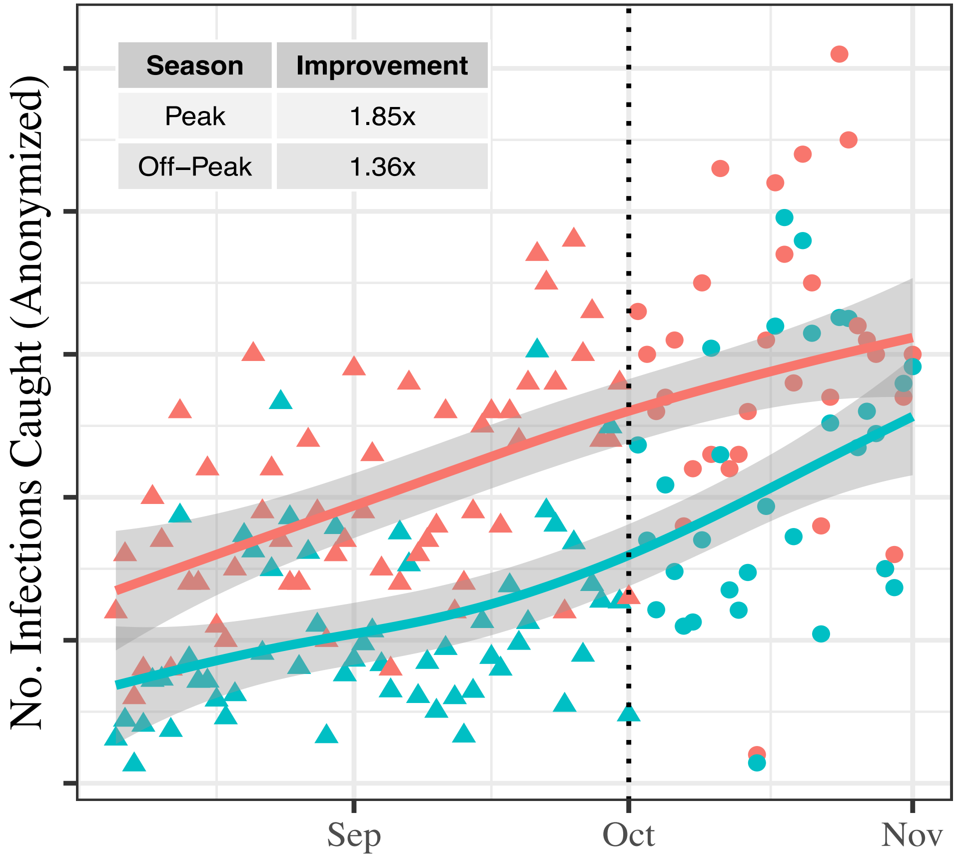
Use prior testing results to allocate tests efficiently at every point of entry

Why Bandits?

- **Bandit feedback**
  - Only observe positive/negative if the traveler is tested
  - Technically “semi-bandit feedback”
- **Nonstationarity**
  - Infection rate for different passenger types changes over time
  - Need to continue to explore and collect data over time

Cases Caught

- 1.85× improvement compared to random testing
- 1.25-1.45× improvement vs. targeting based on public data



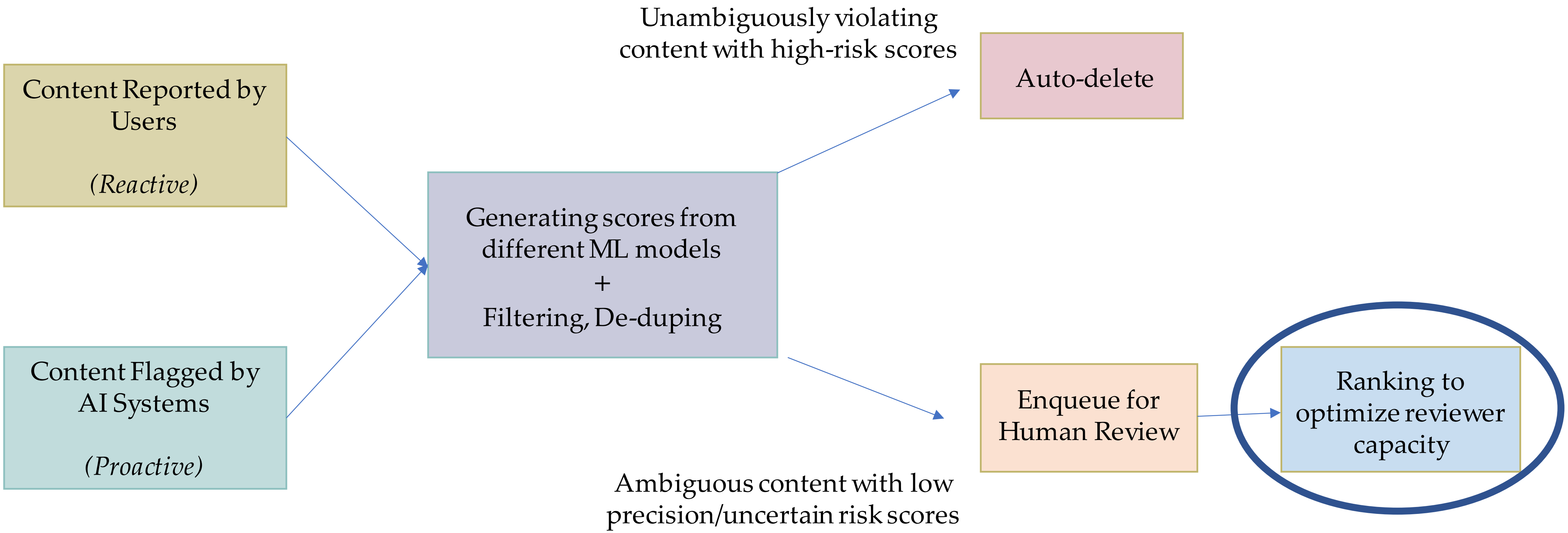
Application: Content Moderation

- **Problem**
  - Millions of pieces of content are posted on Meta platforms each day
  - Too much to manually review all content
  - How to moderate to make sure no harmful?

- **Solution**
  - ML to prioritize potentially harmful content for manual review
  - Featurize content and predict likelihood that it is harmful

V. Avadhanula, O. Baki, H. Bastani, O. Bastani, et al. Bandits for Online Calibration: An Application to Content Moderation on Social Media Platforms

Application: Content Moderation



Application: Content Moderation

- What about new “types” of content?
  - E.g., new kind of racial slur
  - Cold start problem!

- Use multi-armed bandits!

Application: Content Moderation

- Multi-armed bandit
  - Each “step” corresponds to one piece of content

- **Action:** Whether to manually review content

- **Reward:** 1 if content is harmful, 0 otherwise
  - **Intuition:** Goal is to maximize amount of harmful content caught
  - Include an α penalty for flagging content to avoid flagging everything

Application: Training ChatGPT

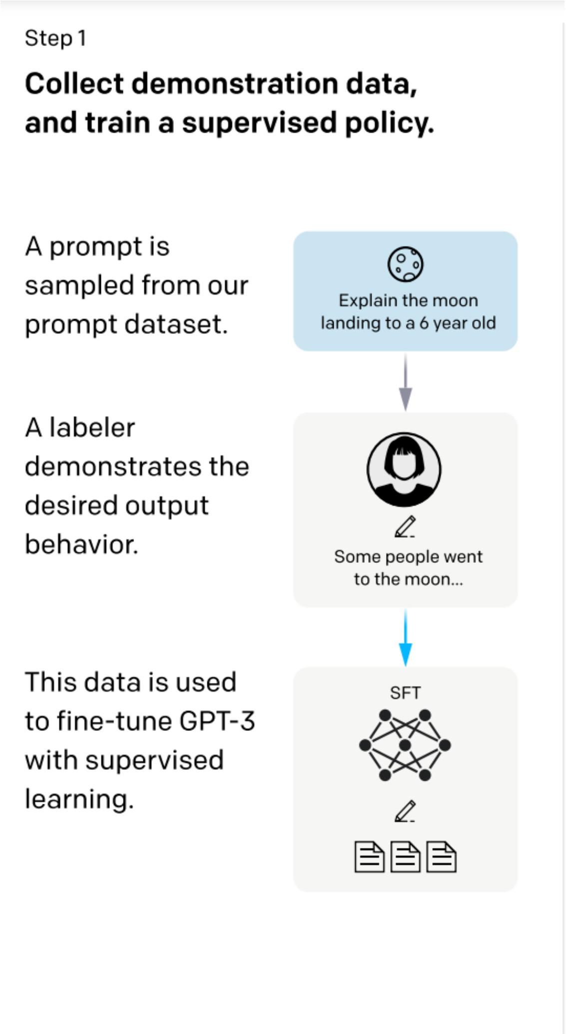
- **Problem**
  - Language models are trained using **unsupervised learning**
  - Generating from these models mimics training data rather than human preferences

- **Solution**
  - **Step 1:** Predict human preferences over possible generations (the reward)
  - **Step 2:** Finetune GPT using reinforcement learning, where it is rewarded for generating content preferred by humans

Application: Training ChatGPT

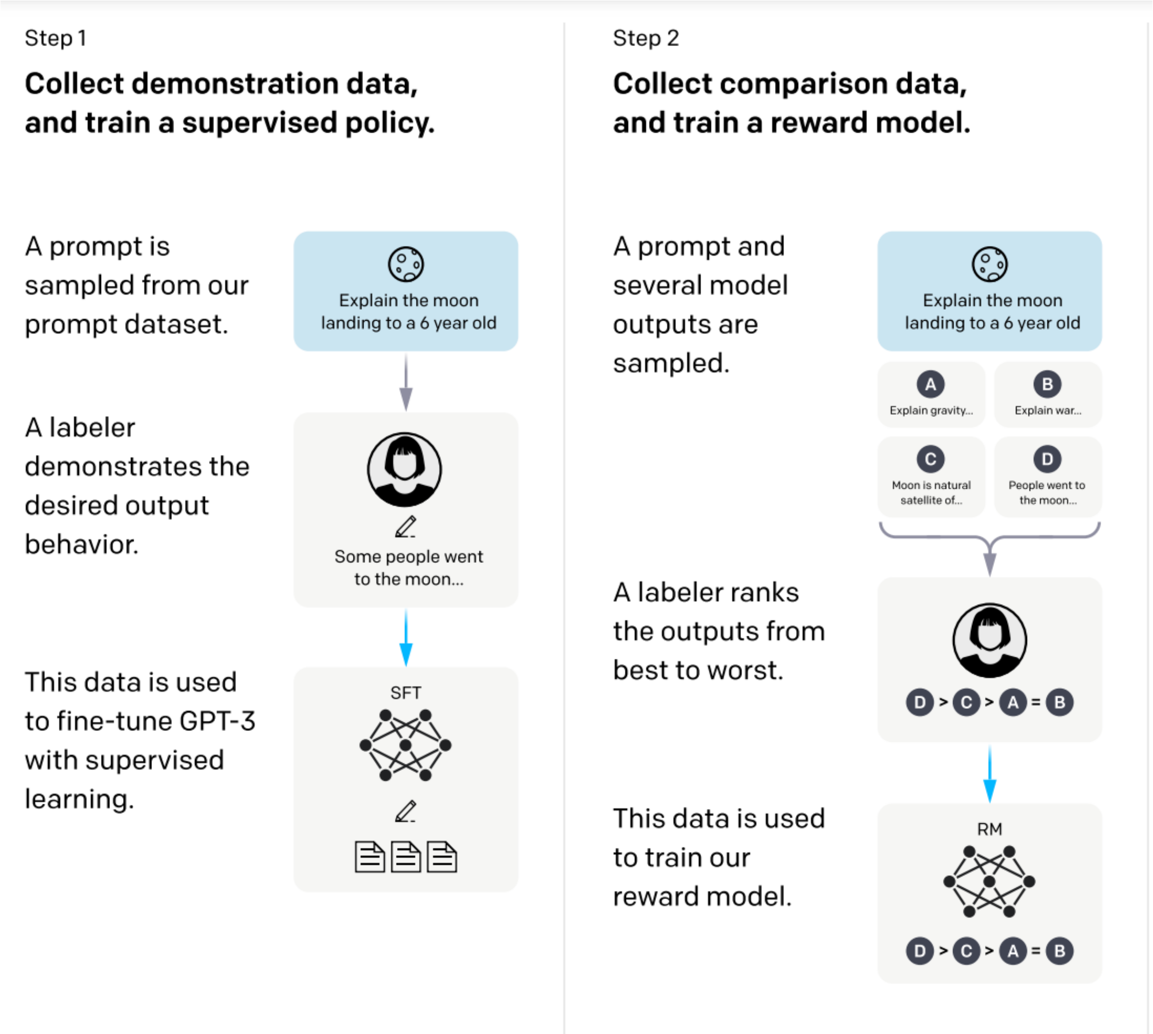
Source: Ouyang et al., Training language models to follow instructions with human feedback.

Application: Training ChatGPT



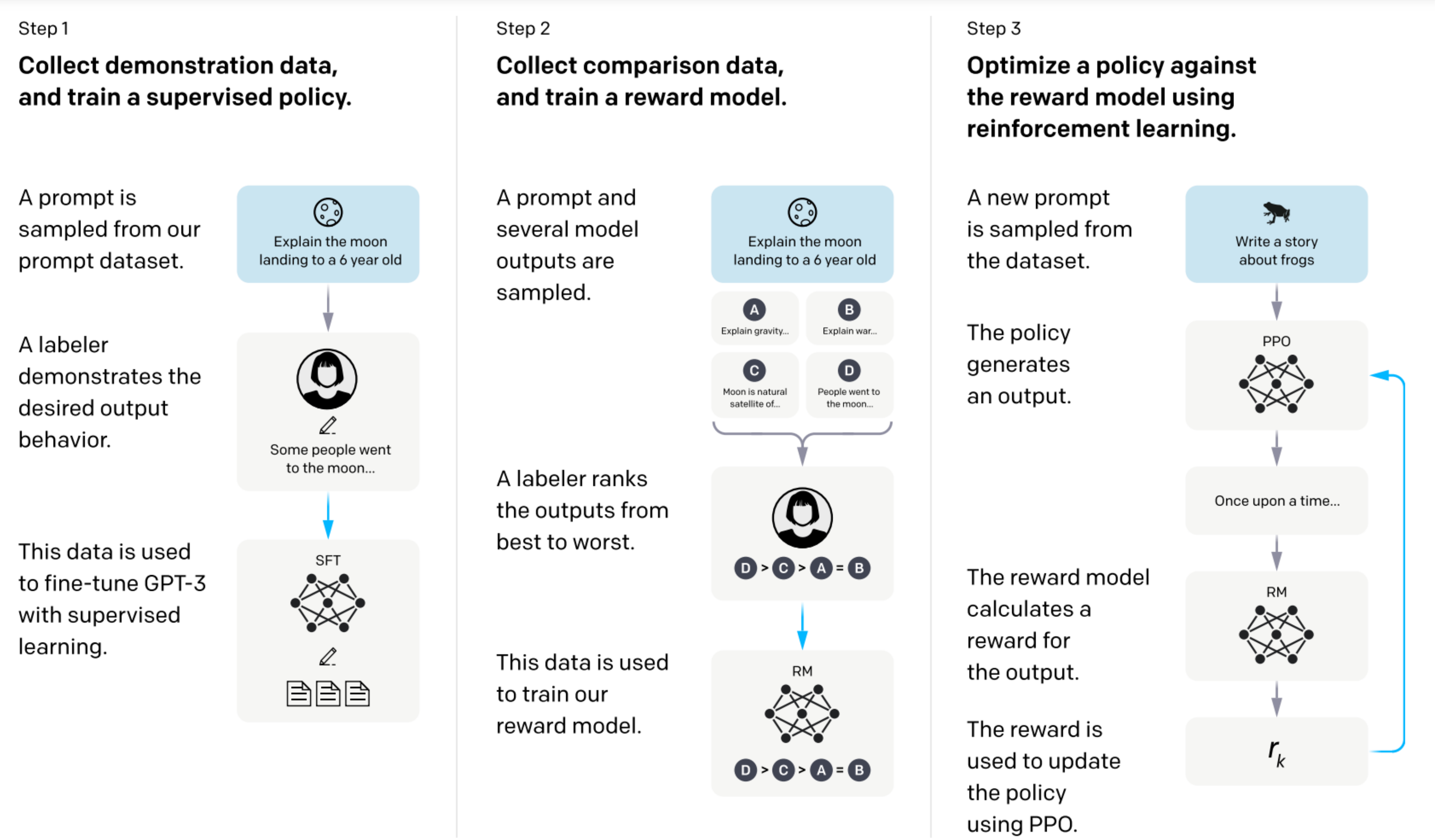
Source: Ouyang et al., Training language models to follow instructions with human feedback.

Application: Training ChatGPT



Source: Ouyang et al., Training language models to follow instructions with human feedback.

Application: Training ChatGPT

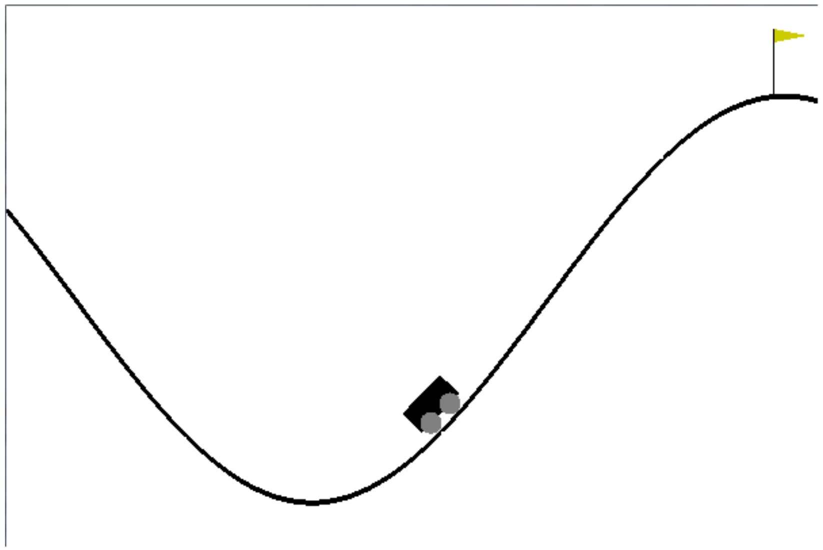


Source: Ouyang et al., Training language models to follow instructions with human feedback.

Exploration in Reinforcement Learning

- ϵ-greedy suffers additional issues due to state space

- Policy learning is an effective practical solution
  - No theoretical guarantees due to local minima



Exploration in Finite MDPs

- **Upper confidence bound (UCB)**
  - Choose action $a_t = \arg\max_{a \in A} \left\{ Q_t(s,a) + \frac{\text{const}}{\sqrt{N_t(s,a)}} \right\}$
  - $N_t(s,a) = \sum_{i=1}^{t-1} 1(s_i = s, a_i = a)$ is the number of times action a has been played in state s

- **Thompson sampling**
  - Choose action $a_t = \arg\max_{a \in A} \{ Q_t(s,a) + \epsilon_{t,s,a} \}$, where $\epsilon_{t,s,a} \sim N\left(0, \frac{\text{const}}{\sqrt{N_t(s,a)}}\right)$

- Both come with theoretical guarantees

Exploration in Continuous MDPs

- Can we adapt these ideas to continuous MDPs?
  - Thompson sampling is more suitable

- **Bootstrap DQN**
  - Train ensemble of k different Q-function estimates $Q_{\theta_1}, \ldots, Q_{\theta_k}$ in parallel
  - Original idea was to use online bootstrap, but training from different random initial θ's worked as well
  - In each episode, act optimally according to Q_{θ_i} for $i \sim \mathrm{Uniform}(\{1, \ldots, k\})$

Exploration in Continuous MDPs

- Can we adapt these ideas to continuous MDPs?
  - Thompson sampling is more suitable
- **Soft Q-learning**
  - Sample actions according to $a \sim \mathrm{Softmax}\left(\left[\beta \cdot \hat{Q}_\theta(s,a)\right]_{a \in A}\right)$

Curiosity

- **Intuition:** Rather than focus on optimism with respect to reward, focus on exploring where we are uncertain

- **How to determine uncertainty?**

- **Candidate strategy**
  - Train a **dynamics model** to predict $s' = f(s, a)$
  - Take actions where $f(s, a)$ has high variance (e.g., use bootstrap)

- **Problems?**
  - What if s' includes spurious components, like a TV screen playing a movie?

Curiosity

- Learn a feature map $\phi(s) \in \mathbb{R}^d$

- **Model 1:** Train a model to predict state transitions:

$$\hat{\phi}(s') = f_\theta(\phi(s), a)$$

  - Feature map lets the model "ignore" spurious components of s such as a TV
  - **Problem:** We could just learn $\phi(s) = \vec{0}$?

Curiosity

- Learn a feature map $\phi(s) \in \mathbb{R}^d$

- **Model 1:** Train a model to predict state transitions:

$$\hat{\phi}(s') = f_\theta(\phi(s), a)$$

- **Model 2:** Train a model to predict action to achieve a transition:

$$\hat{a} = g_\theta\big(\phi(s), \phi(s')\big)$$

  - "Inverse dynamics model" that avoids collapsing ϕ

Curiosity

- Curiosity reward is

$$R(s, a, s') = \left\| \hat{\phi}(s') - \phi(s') \right\|_2^2$$

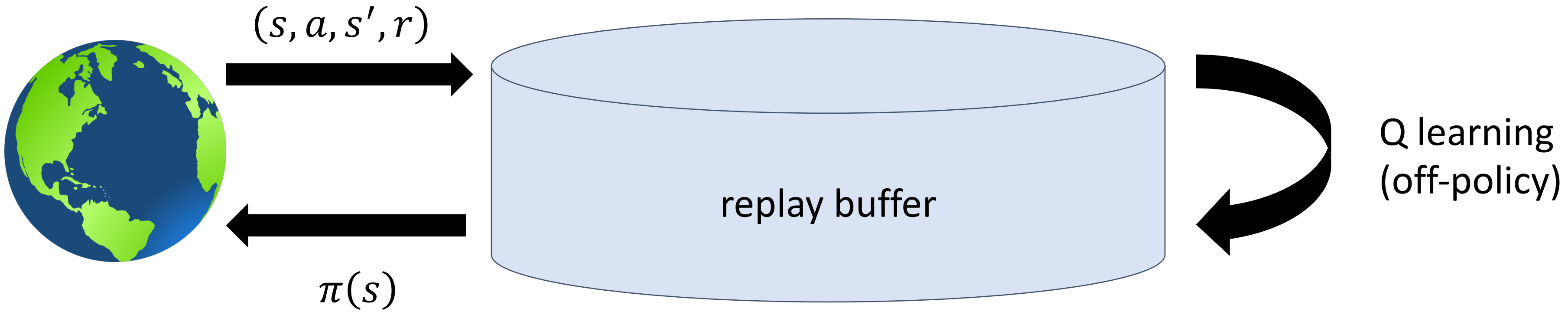
- In other words, reward agent for exercising transitions that f cannot yet predict accurately

Offline Reinforcement Learning

- **Offline reinforcement learning:** How can we learn **without** actively gathering new data?
  - E.g., learn how to perform a task from videos of humans performing the task
  - Also known as **off-policy** or **batch** reinforcement learning
- **Recall:** Drawback of Q learning was we need an exploration strategy
- However, this also enables us to use Q learning with offline data!

Offline Reinforcement Learning

- **Iteratively perform the following:**
  - Take an action a_i and add observation (s_i, a_i, s_{i+1}, r_i) to replay buffer D
  - For $k \in \{1, \ldots, K\}$:
    - Sample $(s_{i,k}, a_{i,k}, s_{i+1,k}, r_{i,k})$ from D
    - $y_{i,k} \leftarrow r_{i,k} + \gamma \cdot \max_{a' \in A} Q_\theta(s_{i+1,k}, a')$
    - $\phi \leftarrow \phi - \alpha \cdot \frac{d}{d\theta}\left(Q_\theta(s_{i,k}, a_{i,k}) - y_{i,k}\right)^2$



Offline Reinforcement Learning

- **Iteratively perform the following:**
  - ~~Take an action a_i and add observation (s_i, a_i, s_{i+1}, r_i) to replay buffer D~~
  - For $k \in \{1, \dots, K\}$:
    - Sample $(s_{i,k}, a_{i,k}, s_{i+1,k}, r_{i,k})$ from D
    - $y_{i,k} \leftarrow r_{i,k} + \gamma \cdot \max_{a' \in A} Q_\theta(s_{i+1,k}, a')$
    - $\phi \leftarrow \phi - \alpha \cdot \frac{d}{d\theta}\left(Q_\theta(s_{i,k}, a_{i,k}) - y_{i,k}\right)^2$