

# Lecture 21: Reinforcement Learning

# Three Kinds of Learning

- **Supervised learning**

- Given labeled examples  $(x, y)$ , learn to predict  $y$  given  $x$

- **Unsupervised learning**

- Given unlabeled examples  $x$ , uncover structure in  $x$

- **Reinforcement learning**

- Learning from sequence of interactions with the environment

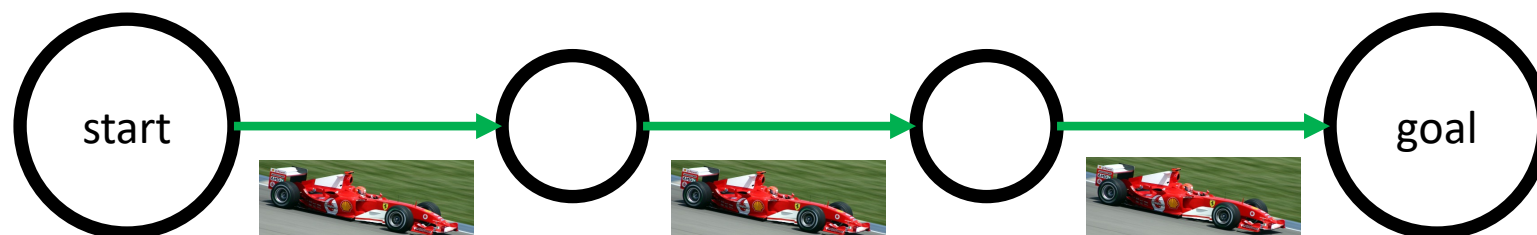
# Sequential Decision Making

- Make a sequence of decisions to maximize a real-valued reward
- **Examples**
  - Driving a car
  - Making movie recommendations
  - Treating a patient over time
  - Navigating a webpage

# Sequential Decision Making

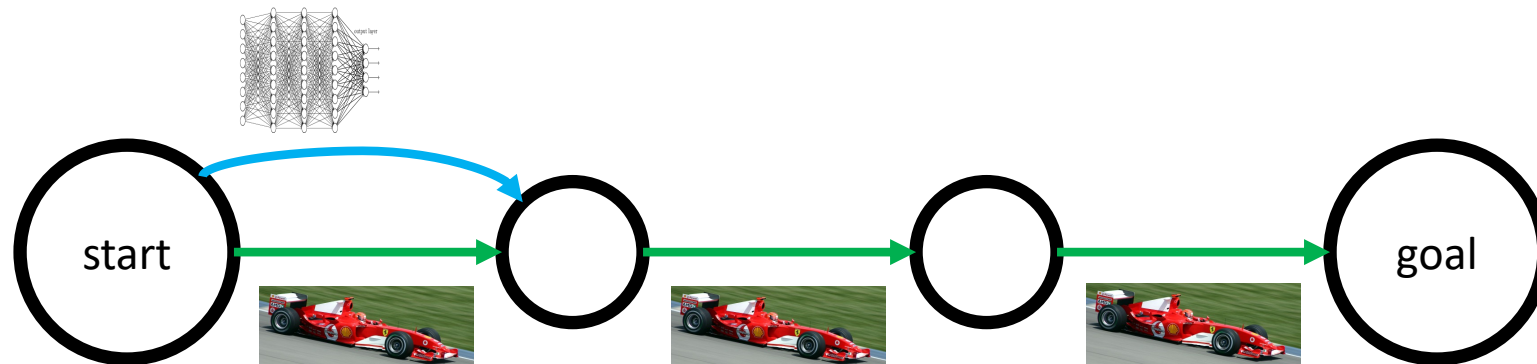
- Machine learning almost always aims to inform decision making
  - Only show user an image if it contains a pet
  - Help a doctor make a treatment decision
- Reinforcement learning is about **sequences** of decisions
- **Naïve strategy:** Predict future and optimize decisions accordingly
  - But decisions affect forecasts
  - If we show the user too many cats, they might get bored of cats!
- **Solution:** Jointly perform prediction and optimization

# What makes RL hard?



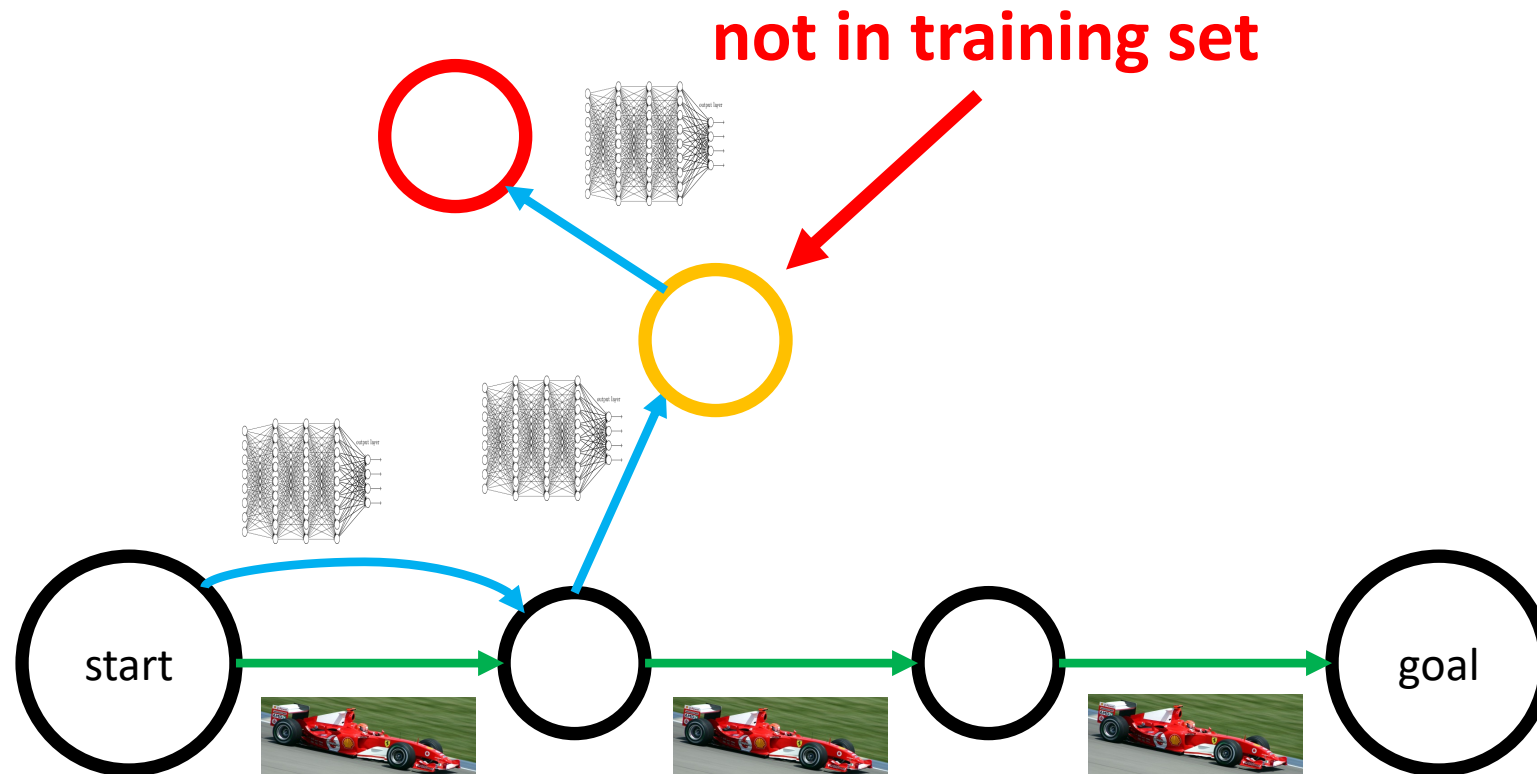
Ross & Bagnell 2011

# What makes RL hard?



Ross & Bagnell 2011

# What makes RL hard?



Ross & Bagnell 2011

# What makes RL hard?

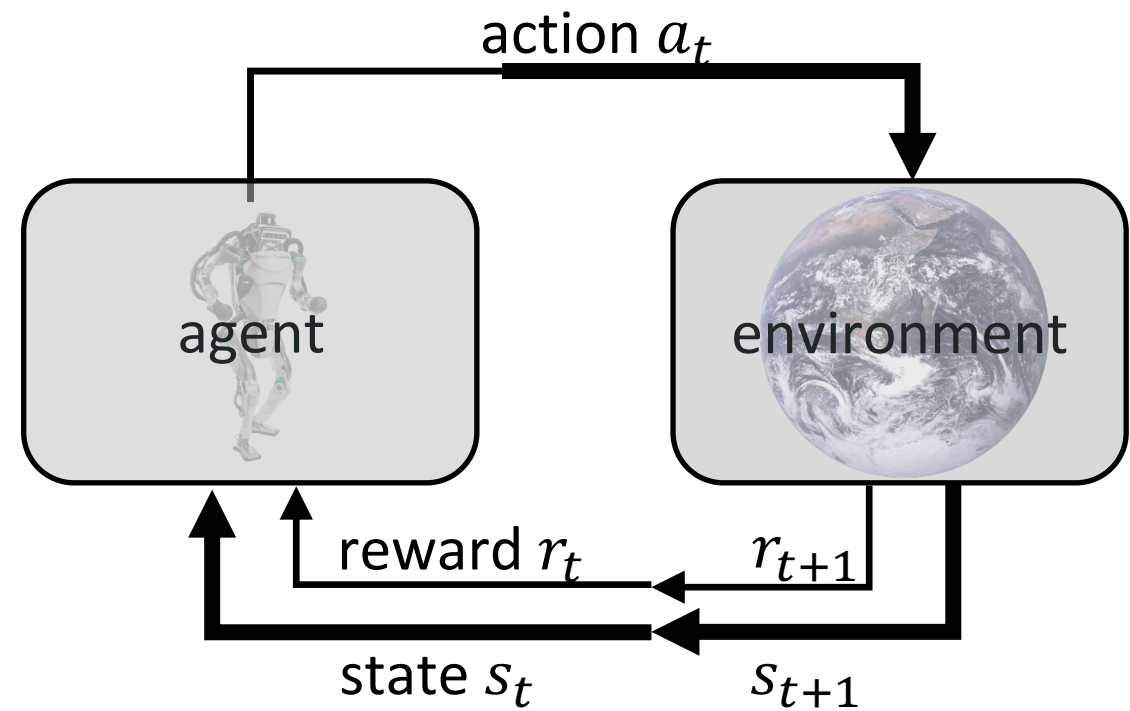
- Distribution shift is **fundamental** to the problem
  - **Repeat:** Improve policy → distribution shifts → improve policy → ...
  - This is with a human expert in the loop! Without the expert, we must start off acting randomly
- Generally, using expert data where available is promising (called “imitation learning”)
  - **Caveat:** Limited by human performance (e.g., AlphaGo Zero significantly outperforms AlphaGo, which was pretrained on expert games)



# Reinforcement Learning Problem

- **At each step**  $t \in \{1, \dots, T\}$ :
  - Observe **state**  $s_t \in S$  and **reward**  $r_t \in \mathbb{R}$
  - Take **action**  $a_t = \pi(s_t) \in A$
- **Goal:** Learn a **policy**  $\pi: S \rightarrow A$  that maximizes discounted reward sum:

$$R_T = \sum_{t=1}^T \gamma^t \cdot r_t$$



# Reinforcement Learning Problem



**state:** joint angles  
**actions:** motor torques  
**dynamics:** robot physics  
**reward:** average speed



**state:** current stock  
**actions:** how much to purchase  
**dynamics:** demand at each store  
**reward:** profit

# Reinforcement Learning Successes



Playing board games and videogames

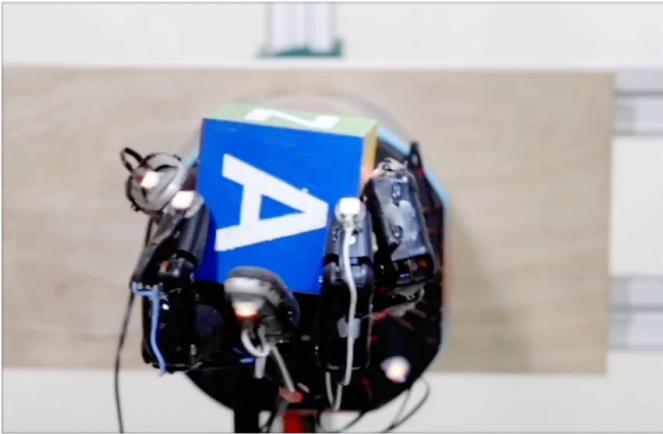
# Reinforcement Learning Successes

The figure displays four wireframe screenshots of a flight booking website, illustrating the progression of a reinforcement learning agent's navigation strategy over time. The screenshots are labeled (a) through (d).

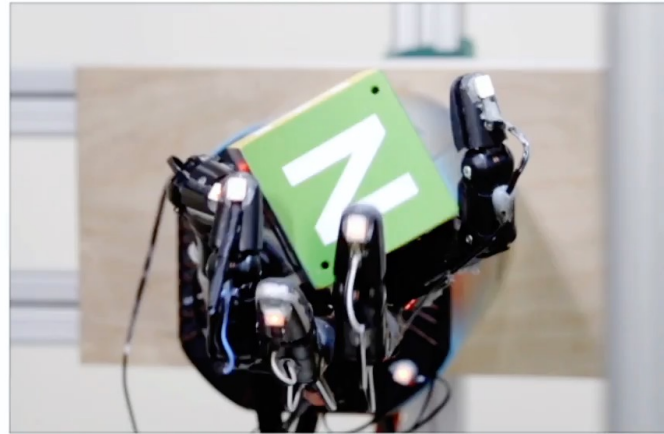
- (a) Early training:** Shows a simple form with fields for "Number of passengers", "From", and a "Continue" button. Below this, separated by a dashed line, is a "Deal of the Day" section for a "Gaming workstation" with a "Get it today!" link and a "Login and Checkout" button. A second dashed line separates this from a "Payment" section with radio buttons for "Credit Card" and "Debit Card" and a "Continue" button.
- (b) Mid training:** Shows a form with fields for "Address" and a "Continue" button. Below this, separated by a dashed line, is a form with fields for "Last Name", "To", "First Name", and a "Continue" button.
- (c) Late training:** Shows a form with fields for "To", "Last Name", "First Name", "Address", "Full name", "Payment" (with radio buttons for "Credit Card" and "Debit Card"), "From", and a "Continue" button.
- (d) Test:** Shows a "HOME" header with a hamburger menu icon. Below is a "Username" field, a "Password" field, checkboxes for "Remember me" and "Stay logged in", a "Forgot user name." link, a "Forgot password." link, and a "Continue" button.

Web navigation (e.g., book a flight)

# Reinforcement Learning Successes



**FINGER PIVOTING**



**SLIDING**

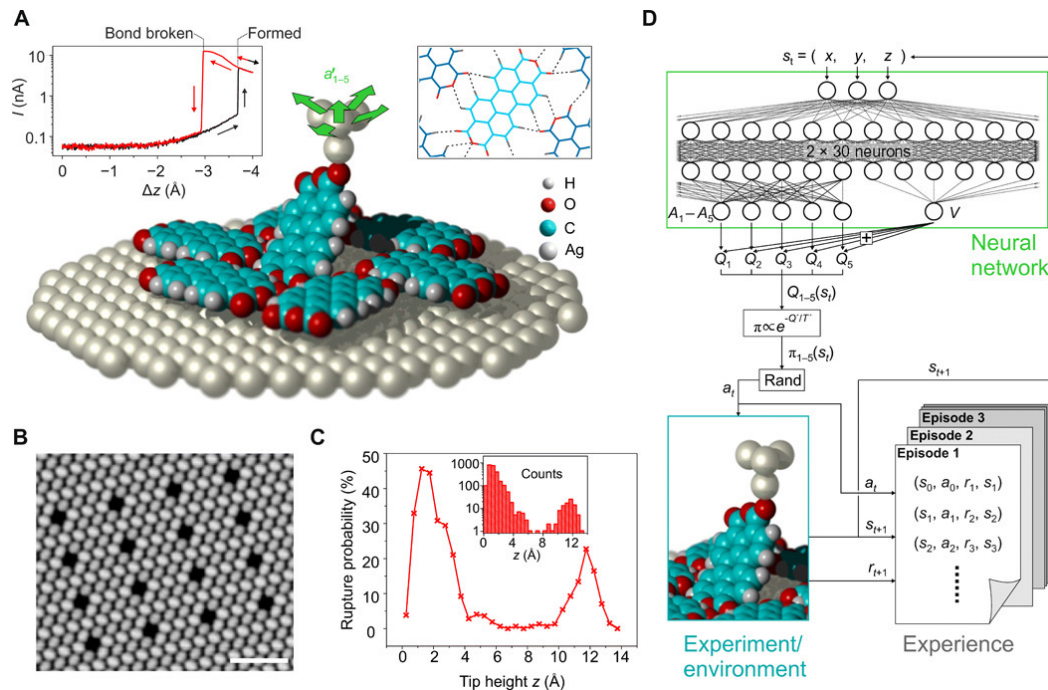


**FINGER GAITING**

Robotics (e.g., Rubik's cube manipulation)

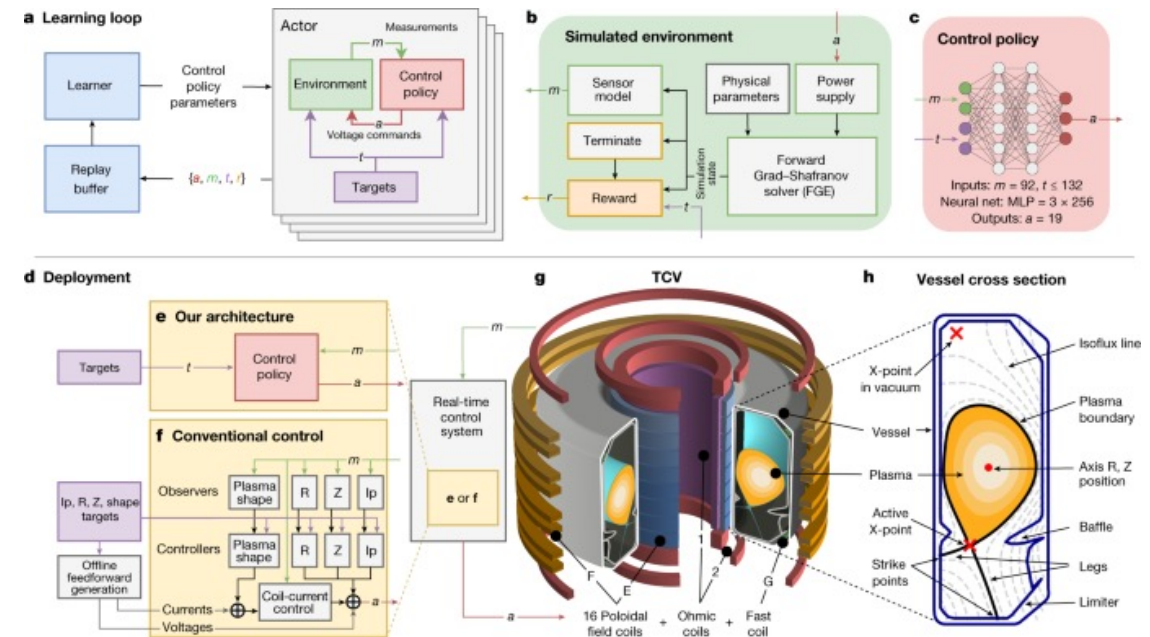


# Reinforcement Learning Successes



Steering microscope to separate molecules

<https://www.science.org/doi/10.1126/sciadv.abb6987>



Controlling magnetic fields to stabilize plasma (in simulation)

Degrave et al 2022, Magnetic control of tokamak plasmas through deep reinforcement learning

# Reinforcement Learning Successes

- **Power grids:** Reinforcement learning for demand response
  - A review of algorithms and modeling techniques, J. Vázquez-Canteli, Z. Nagy
- **Recommender systems**
  - <https://github.com/google-research/recsim>
- **Many potential applications**
  - <https://arxiv.org/abs/1904.12901>

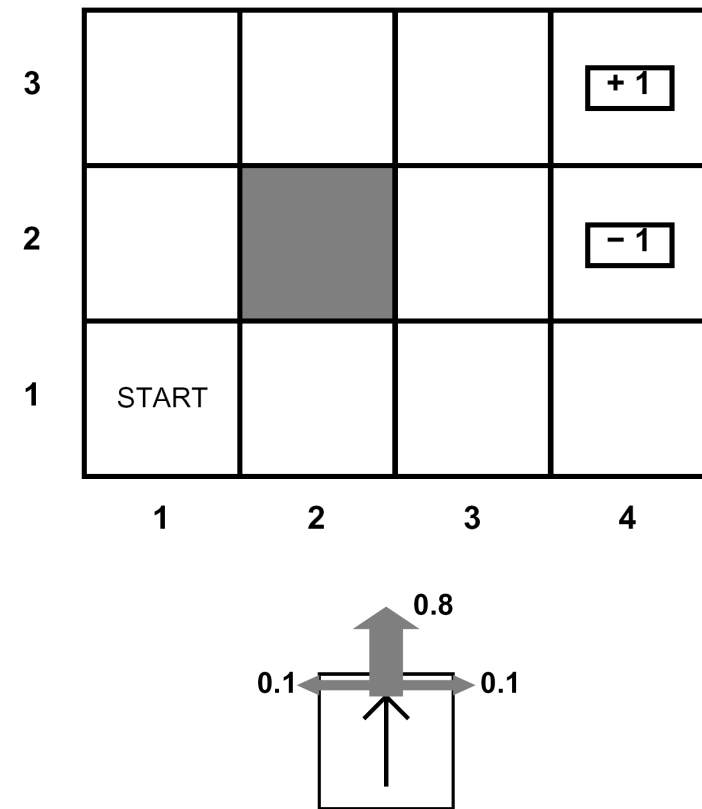
# Reinforcement Learning Problem

- At a high level, we need to specify the following:
  - **State space:** What are the observations the agent may encounter?
  - **Action space:** What are the actions the agent can take?
  - **Transitions/dynamics:** How the state is updated when taking an action
  - **Rewards:** What rewards the agent receives for taking an action in a state
- For most of today, assume state and action spaces are finite



# Toy Example

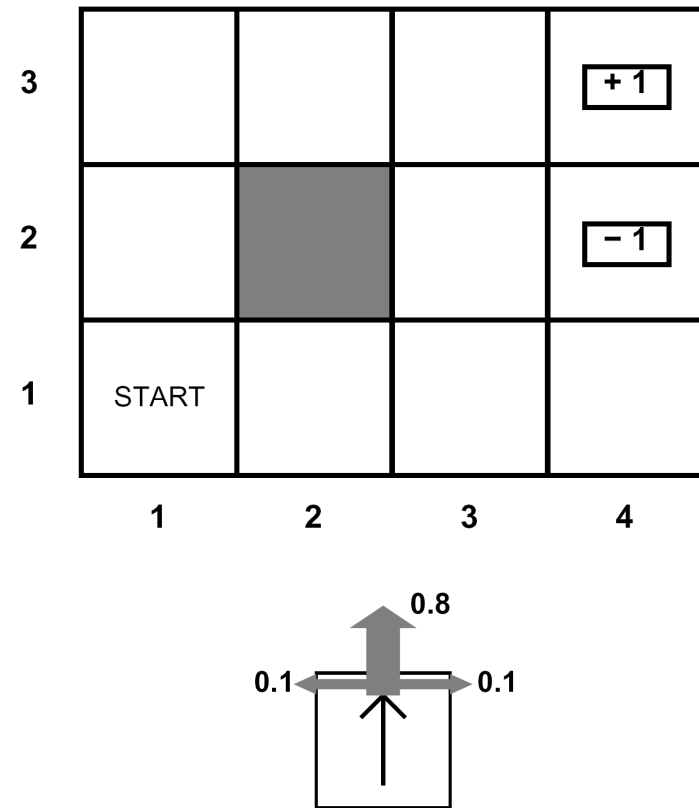
- Grid map with solid/open cells
- **State:** An open grid cell
- **Actions:** Move North, East, South, West



# Toy Example

- **Dynamics**

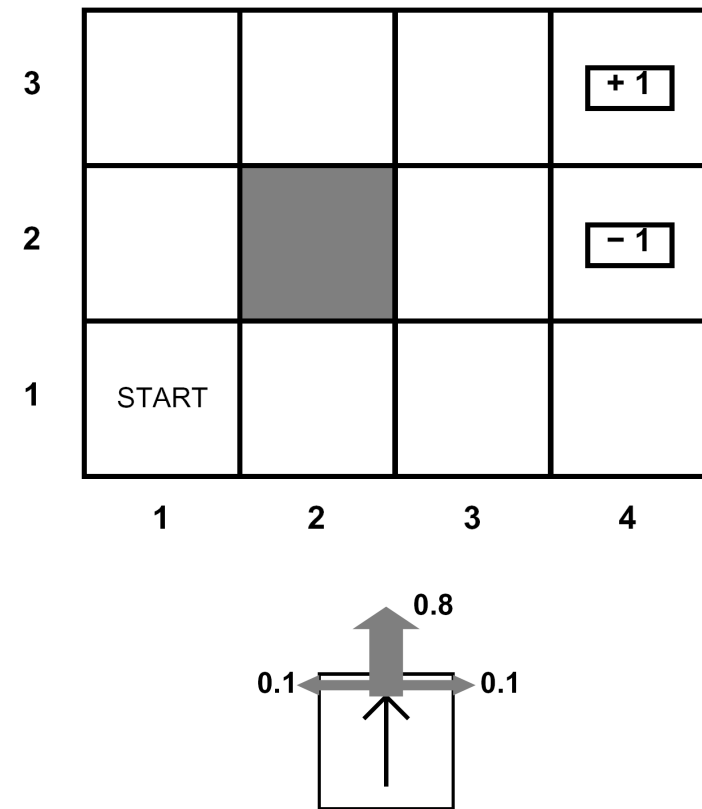
- Move in chosen direction, but **not deterministically!**
- Succeeds 80% of the time
- 10% of the time, end up  $90^\circ$  off
- 10% of the time, end up  $-90^\circ$  off
- The agent stays put if it tries to move into a solid cell or outside the world
- At terminal states, any action ends **episode** (or **rollout**)



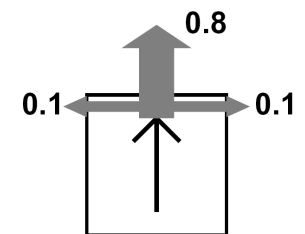
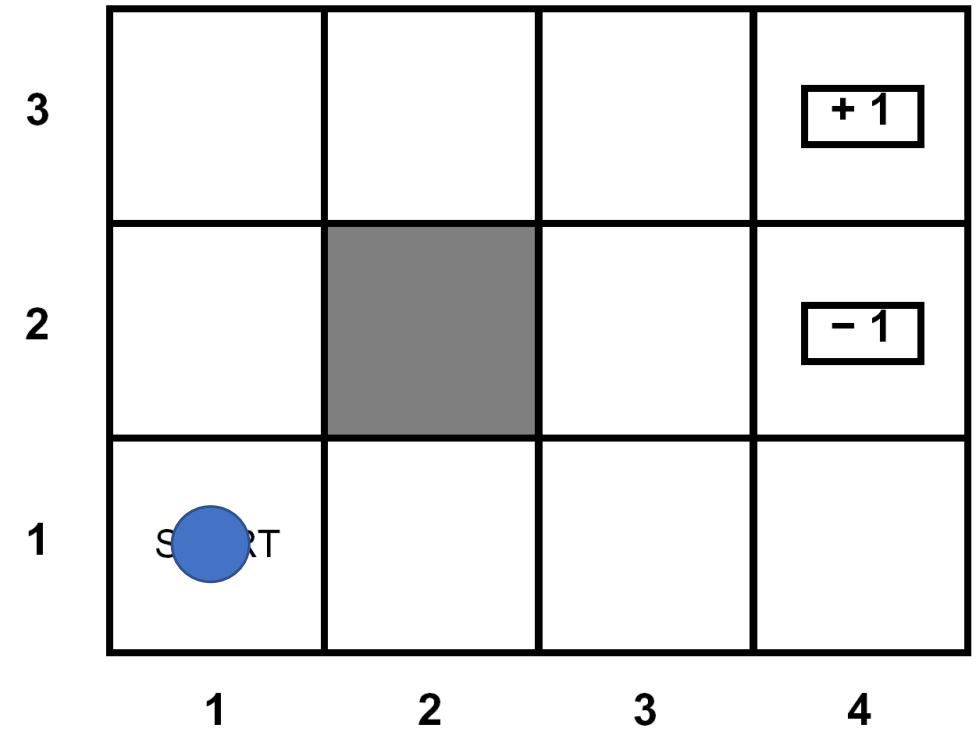
# Toy Example

- **Rewards**

- At terminal state, agent receives the specified reward
- For each timestep outside terminal states, the agent pays a small cost, e.g., a “reward” of  $-0.03$

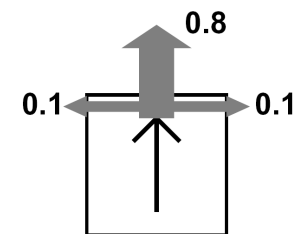
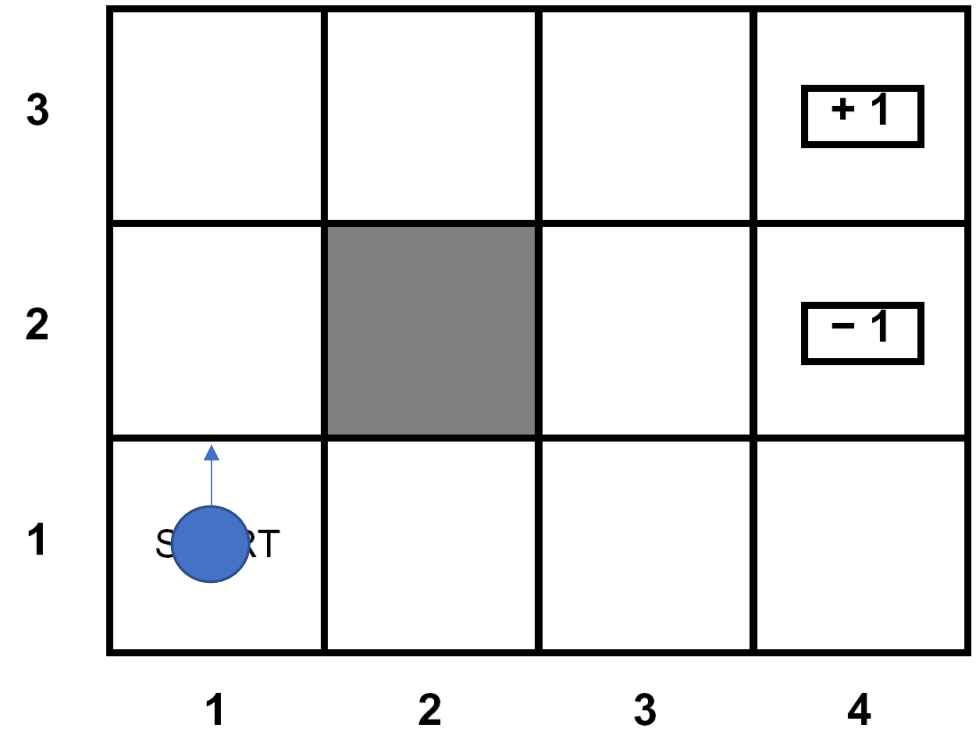


# Example Episode (Random Policy)



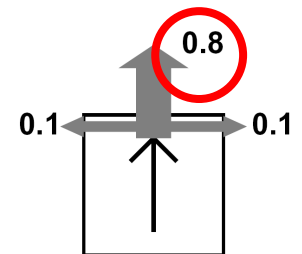
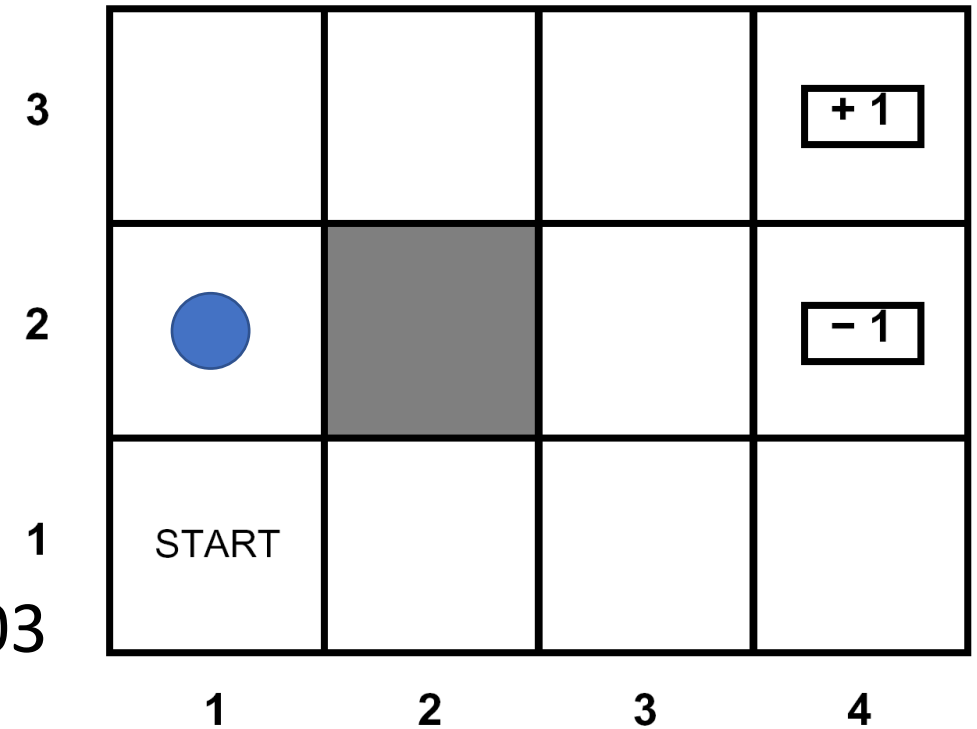
# Example Episode (Random Policy)

Action= "N"



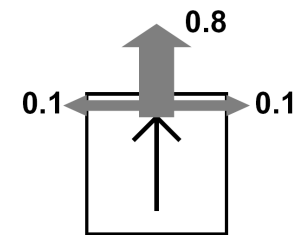
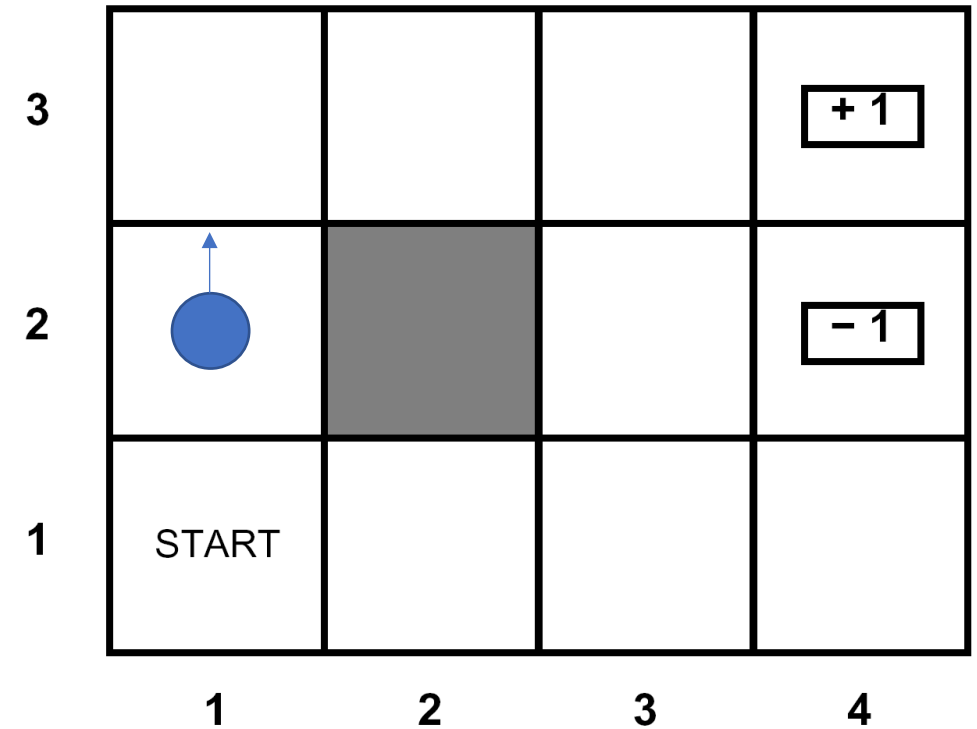
# Example Episode (Random Policy)

Action= "N"  
Result = "N" 1  
Reward = -0.03



# Example Episode (Random Policy)

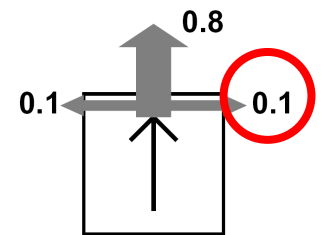
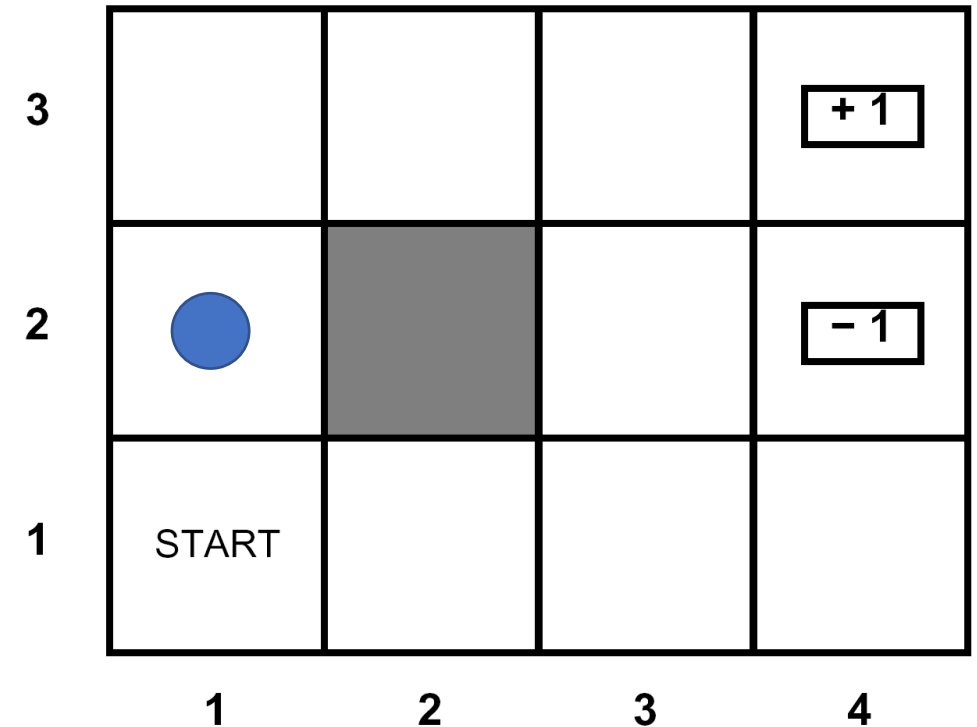
Action= "N"



# Example Episode (Random Policy)

(stays still because blocked)

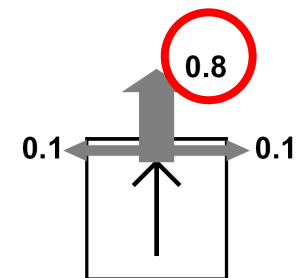
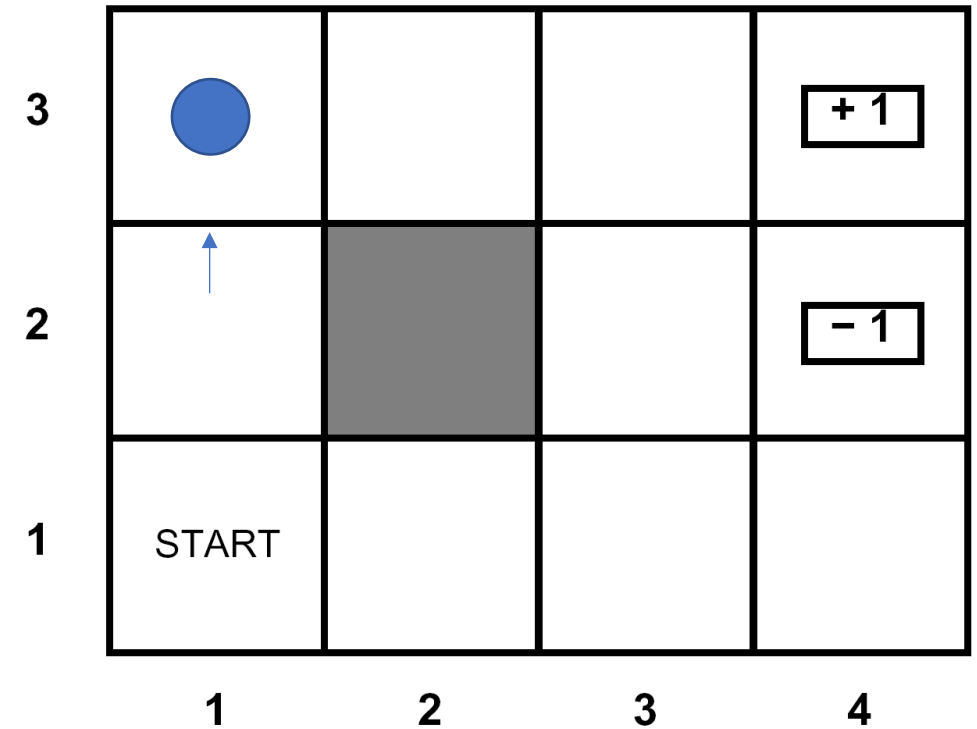
Action= "N"  
Result="E"  
Reward = -0.03





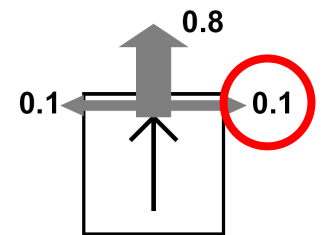
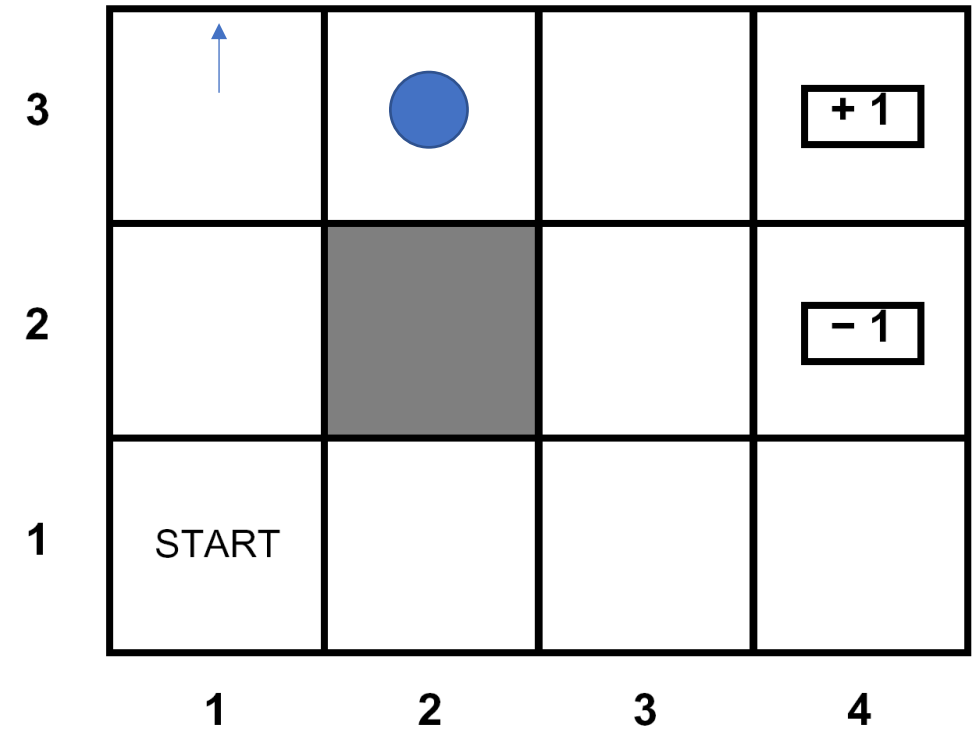
# Example Episode (Random Policy)

Action= "N"  
Result="N"  
Reward = -0.03



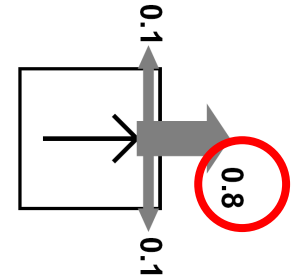
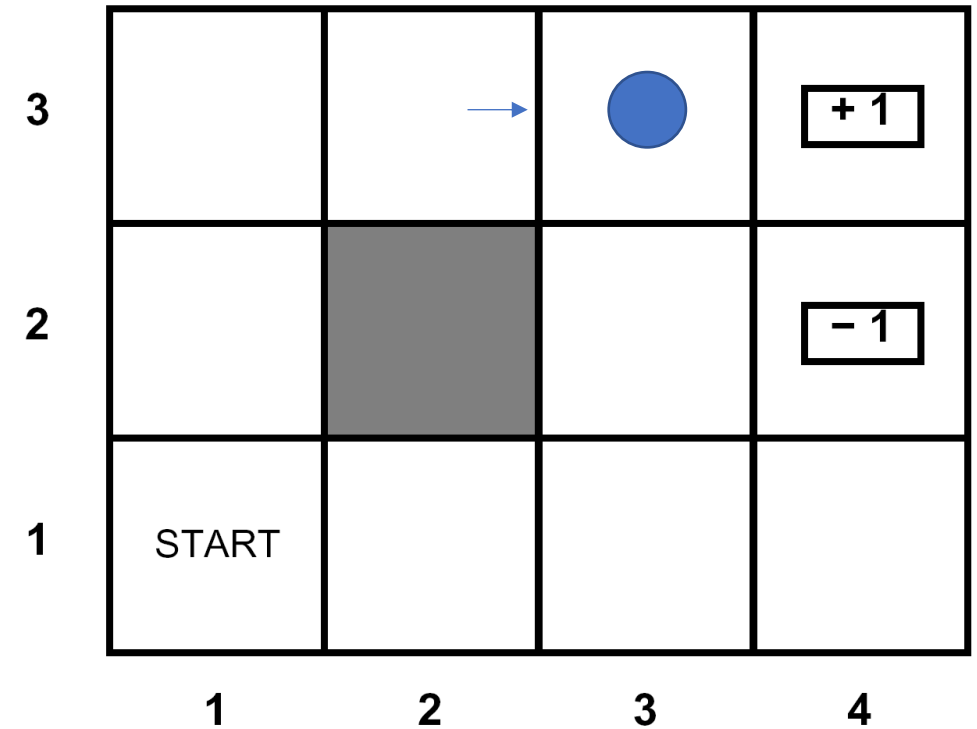
# Example Episode (Random Policy)

Action= "N"  
Result="E"  
Reward = -0.03



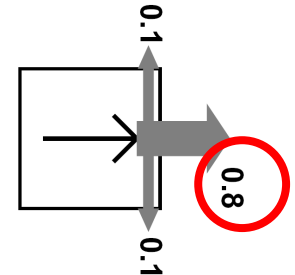
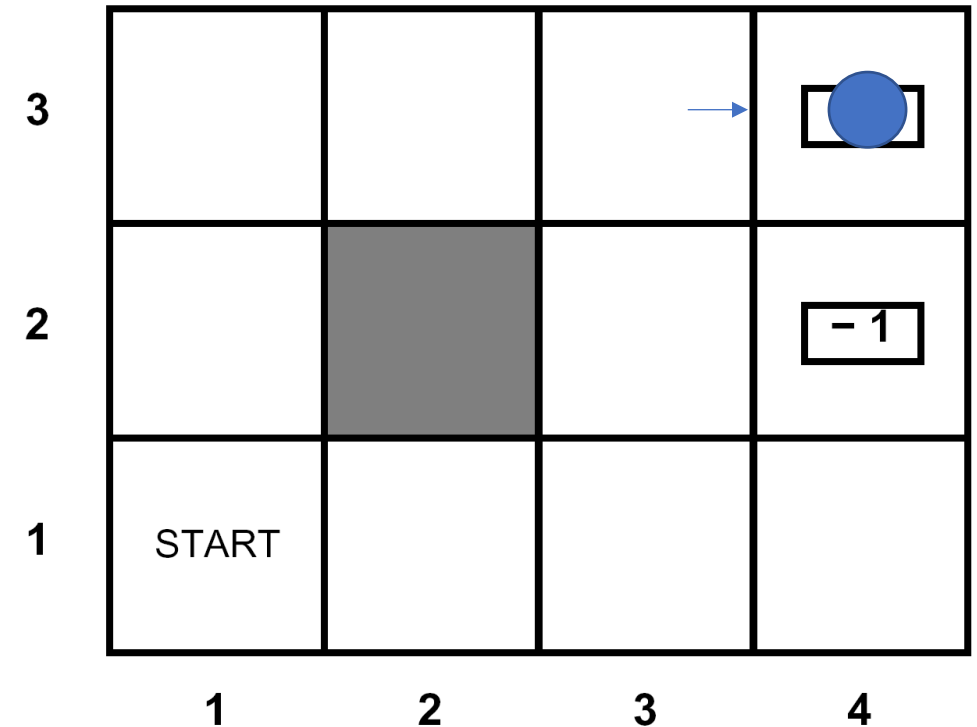
# Example Episode (Random Policy)

Action= "E"  
Result="E"  
Reward = -0.03



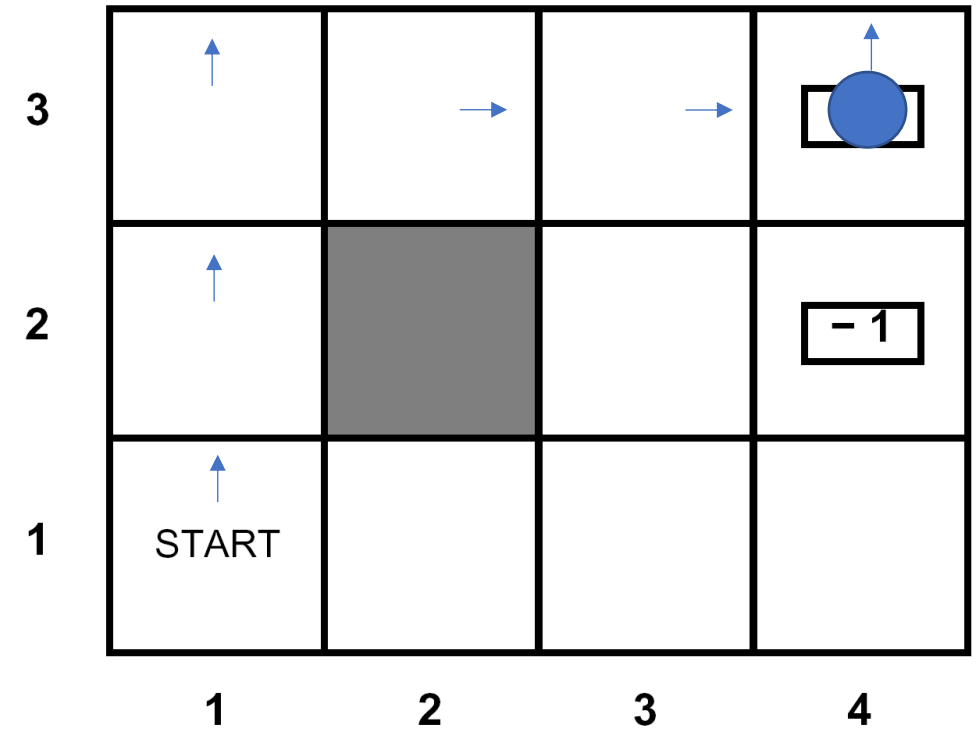
# Example Episode (Random Policy)

Action= "E"  
Result="E"  
Reward = -0.03



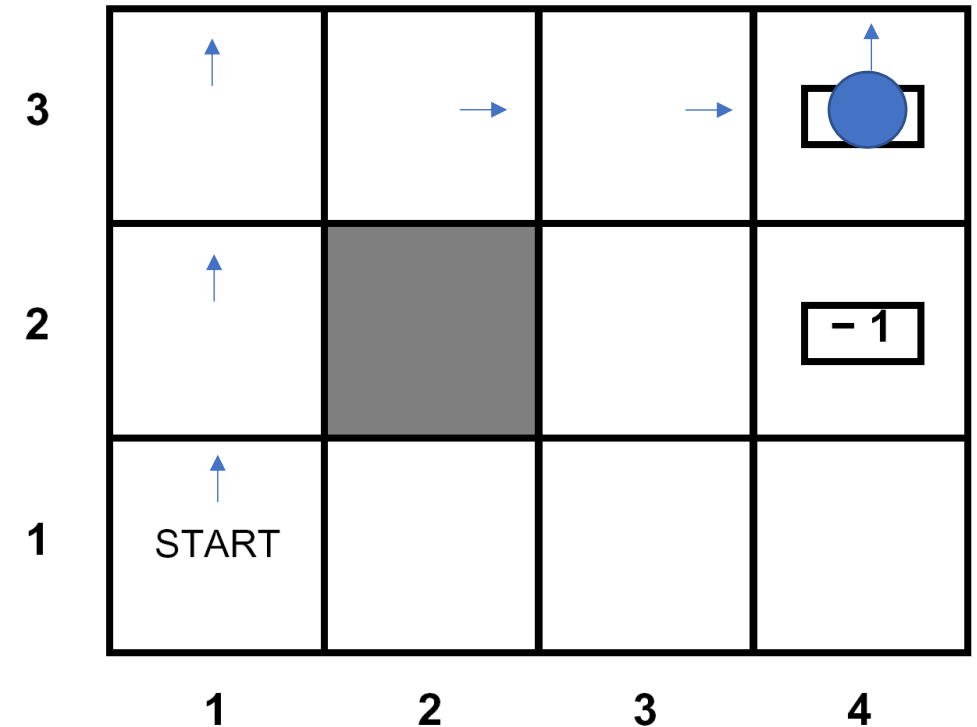
# Example Episode (Random Policy)

Action= "N"  
Result= "the end"  
Reward = +1



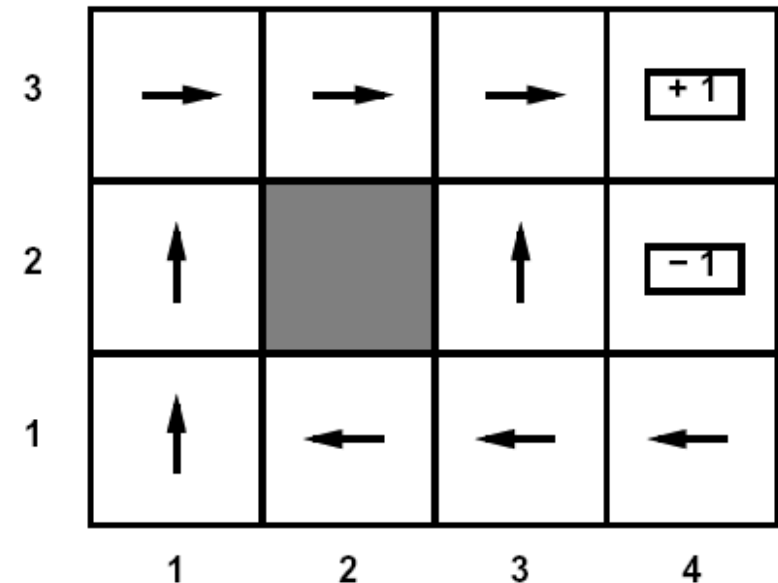
# Example Episode (Random Policy)

- Our random trajectory happened to end in the right place!
- Optimal policy? **No!**
  - Only succeeded by random chance



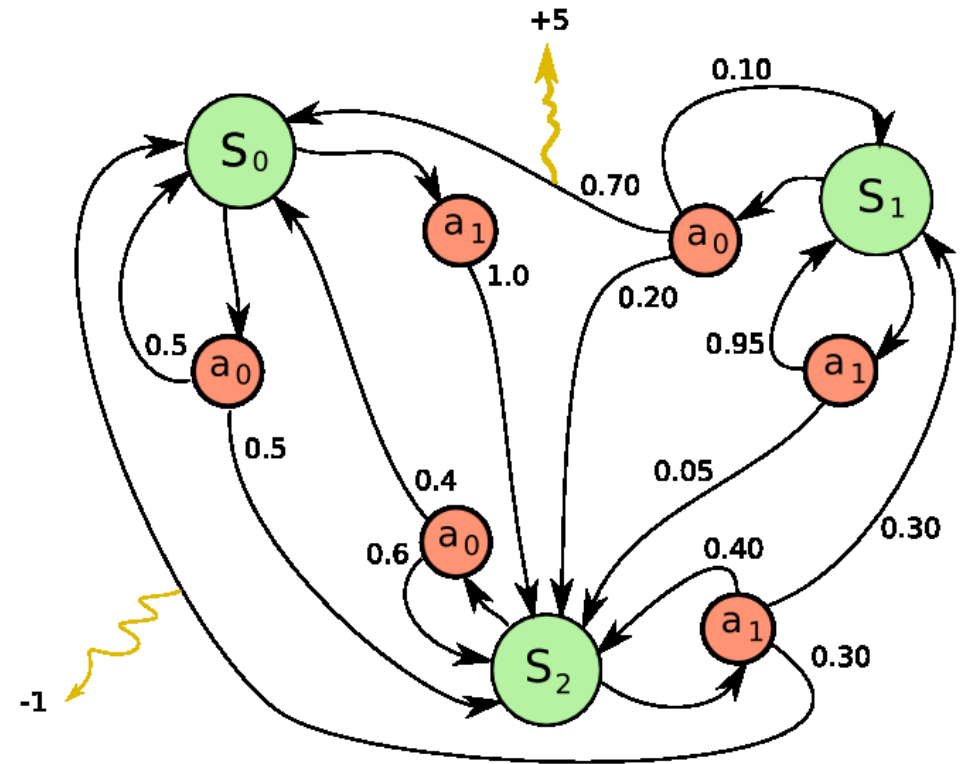
# Optimal Policy

- **Optimal policy:** Following  $\pi^*$  maximizes total reward received
  - **Discounted:** Future rewards are downweighted
  - **In expectation:** On average across randomness of environment and actions



# Markov Decision Process (MDP)

- An MDP  $(S, A, P, R, \gamma)$  is defined by:
  - Set of states  $s \in S$
  - Set of actions  $a \in A$
  - Transition function  $P(s' | s, a)$  (also called “dynamics” or the “model”)
  - Reward function  $R(s, a, s')$
  - Discount factor  $\gamma < 1$
- Also assume an initial state distribution  $D(s)$ 
  - Often omitted since optimal policy does not depend on  $D$





# Markov Decision Process (MDP)

- **Goal:** Maximize **cumulative expected discounted reward**:

$$\pi^* = \max_{\pi} J(\pi) \quad \text{where} \quad J(\pi) = \mathbb{E}_{\zeta} \left[ \sum_{t=0}^{\infty} \gamma^t \cdot r_t \right]$$

- Expectation over **episodes**  $\zeta = (s_0, a_0, r_0, s_1, \dots)$ , where
  - $s_0 \sim D$
  - $a_t = \pi(s_t)$
  - $s_{t+1} \sim P(\cdot | s_t, a_t)$
  - $r_t = R(s_t, a_t, s_{t+1})$

# Markov Decision Process (MDP)

- **Planning:** Given  $P$  and  $R$ , compute the optimal policy  $\pi^*$ 
  - Purely an optimization problem! No learning
- **Reinforcement learning:** Compute the optimal policy  $\pi^*$  without prior knowledge of  $P$  and  $R$

# Policy Value Function

- **Policy Value Function:** Expected reward if we start in  $s$  and use  $\pi$ :

$$V^\pi(s) = \mathbb{E} \left( \sum_{t=0}^{\infty} \gamma^t \cdot r_t \mid s_0 = s \right)$$

- **Bellman equation:**

$$\underbrace{V^\pi(s)}_{\text{current value}} = \sum_{s' \in \mathcal{S}} \underbrace{P(s' \mid s, \pi(s))}_{\text{expectation over next state}} \cdot \underbrace{\left( R(s, \pi(s), s') + \gamma \cdot V^\pi(s') \right)}_{\text{current reward + discounted future reward}}$$

# Optimal Value Function

- **Optimal value function:** Expected reward if we start in  $s$  and use  $\pi^*$ :

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

- **Bellman equation:**

Optimal policy selects action that maximizes future expected reward from state  $s$

$$\underbrace{V^*(s)}_{\text{current value}} = \max_{a \in A} \sum_{s' \in S} \underbrace{P(s' \mid s, a)}_{\text{expectation over next state}} \cdot \underbrace{\left( R(s, a, s') + \gamma \cdot V^*(s') \right)}_{\text{current reward + discounted future reward}}$$

# Optimal Value Function

- **Bellman equation:**

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

- Do not need to know the optimal policy  $\pi^*$ !
- **Strategy:** Compute  $V^*$  and then use it to compute  $\pi^*$ 
  - **Caveat:** Latter step requires knowing  $P$

# Policy Action-Value Function

- **Policy Action-Value Function (or Q function):** Expected reward if we start in  $s$ , take action  $a$ , and then use  $\pi$  thereafter:

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

- **Bellman equation:**

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

# 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:**

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

# Relationship

- We have

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s))$$

- Similarly, we have

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



# Q Iteration

- We have

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

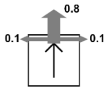
- **Strategy:** Compute  $Q^*$  and then use it to compute  $\pi^*$

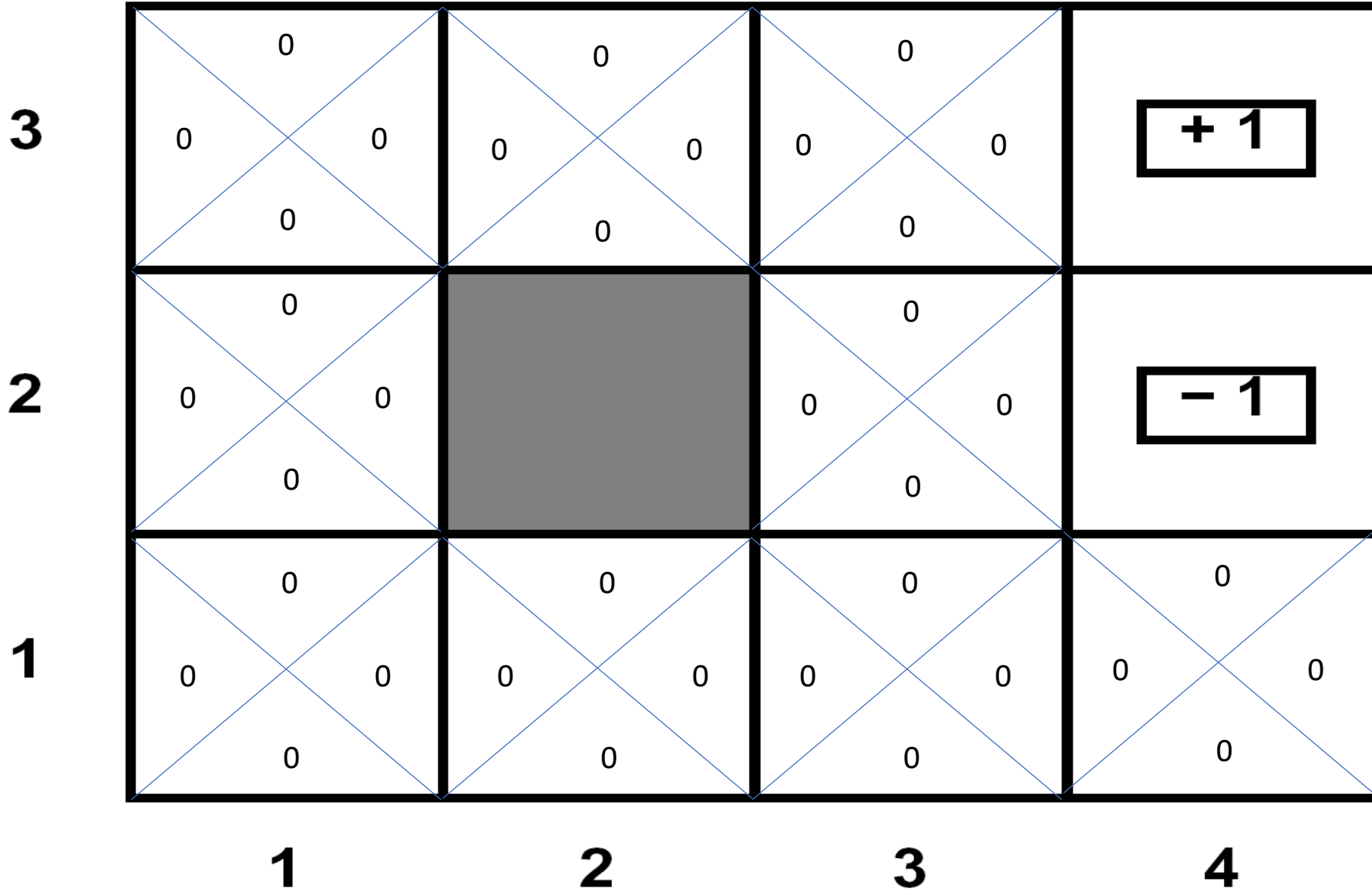
# Q Iteration

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

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

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

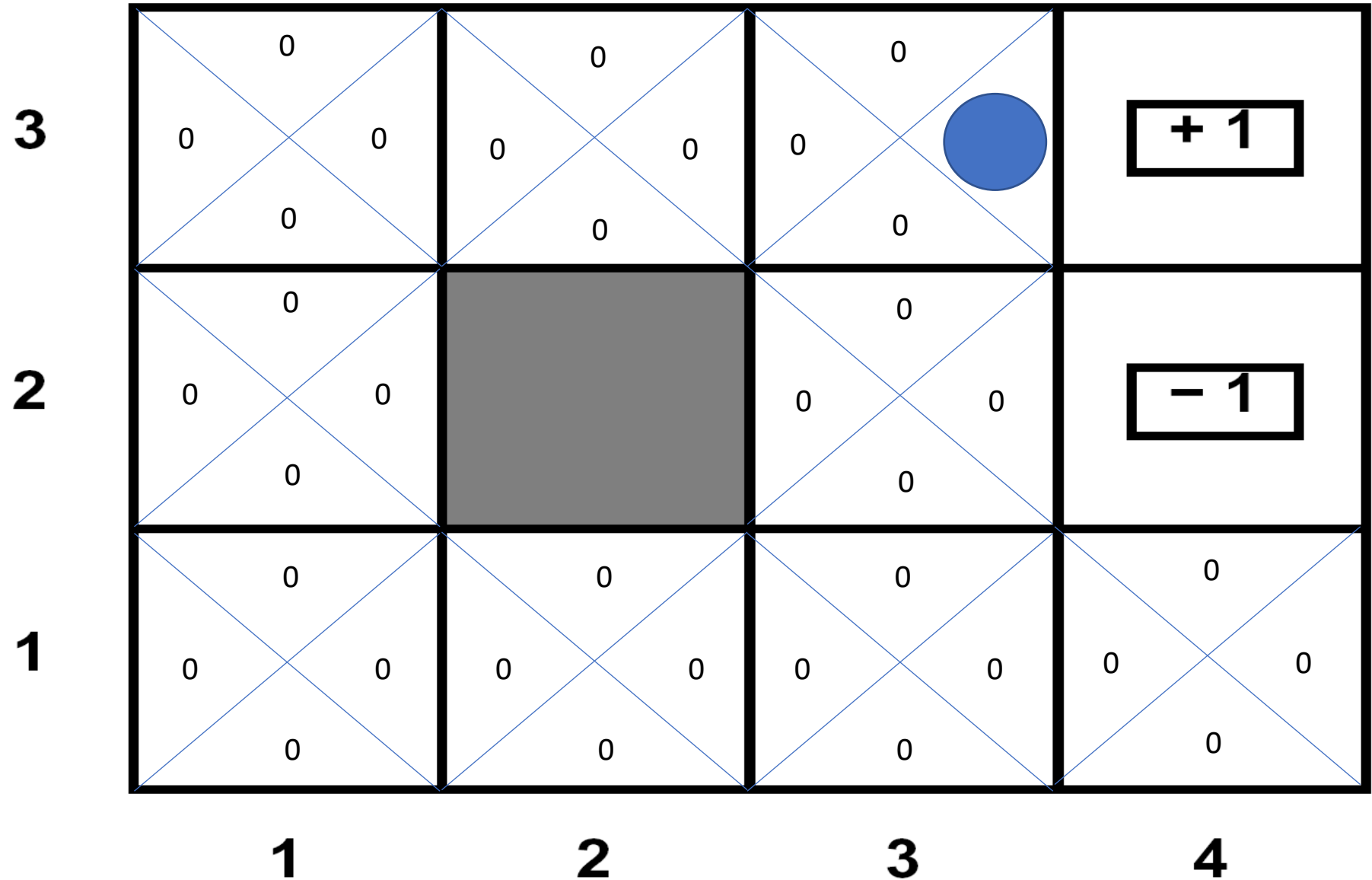

 Living cost 0
 0.9



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

Diagram illustrating the Bellman optimality equation for Q-learning. A small diagram shows a state-action pair with probabilities: 0.8 for the action leading to the next state, and 0.1 for the action leading to the current state. The equation calculates the updated Q-value for state  $s$  and action  $a$  based on the immediate reward  $R(s, a, s')$  and the maximum Q-value for the next state  $s'$  over all actions  $a'$ .

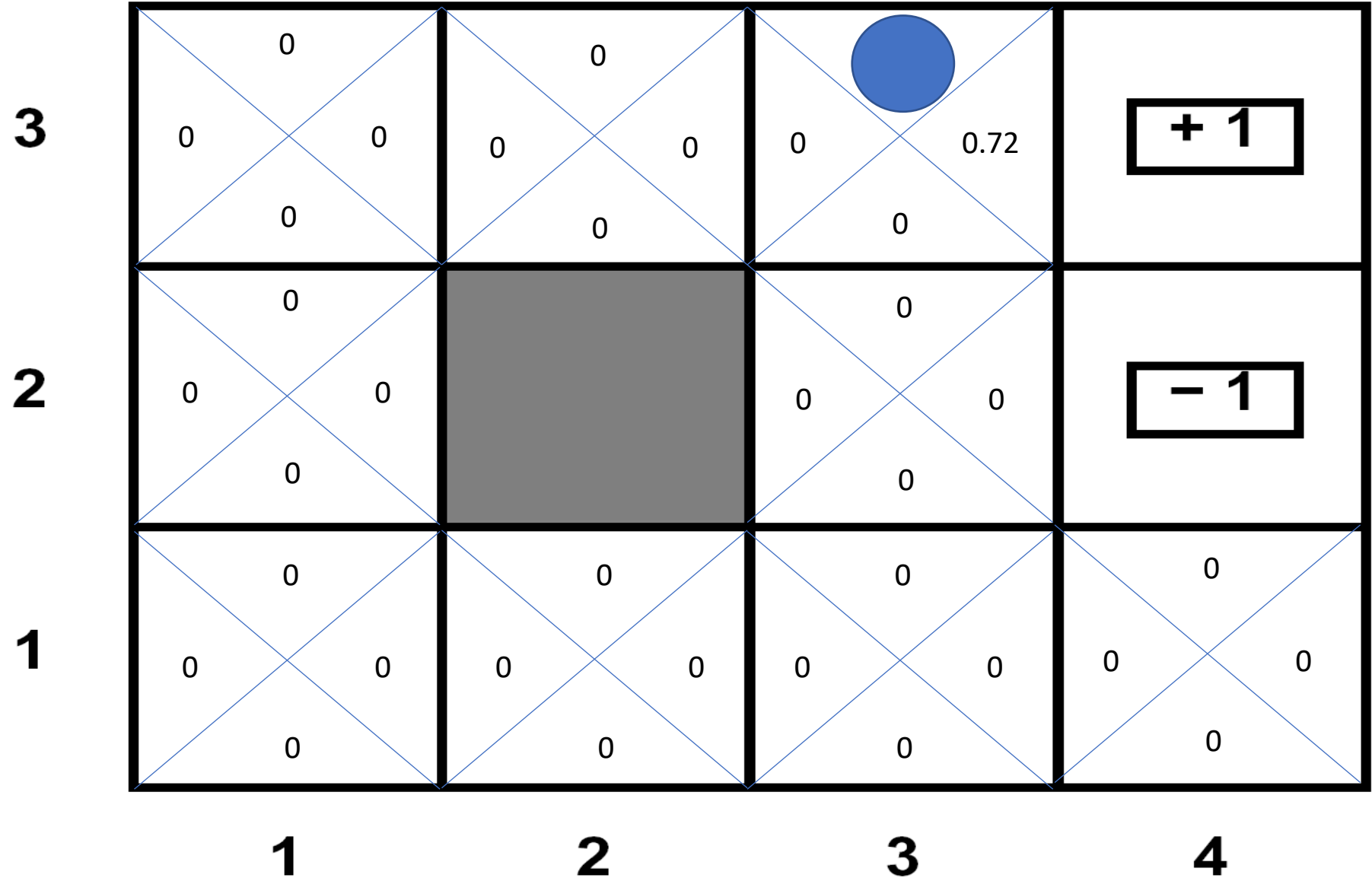
$$\begin{aligned} &0.8 \times [0 + 0.9 \times 1] \\ &+ 0.1 \times [0 + 0] \\ &+ 0.1 \times [0 + 0] \\ &= 0.72 \end{aligned}$$



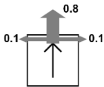
$$Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right]$$

Diagram illustrating the Bellman optimality equation for Q-learning. A small diagram shows a state-action pair with probabilities: 0.8 for action 'a' and 0.1 for action 'a''.

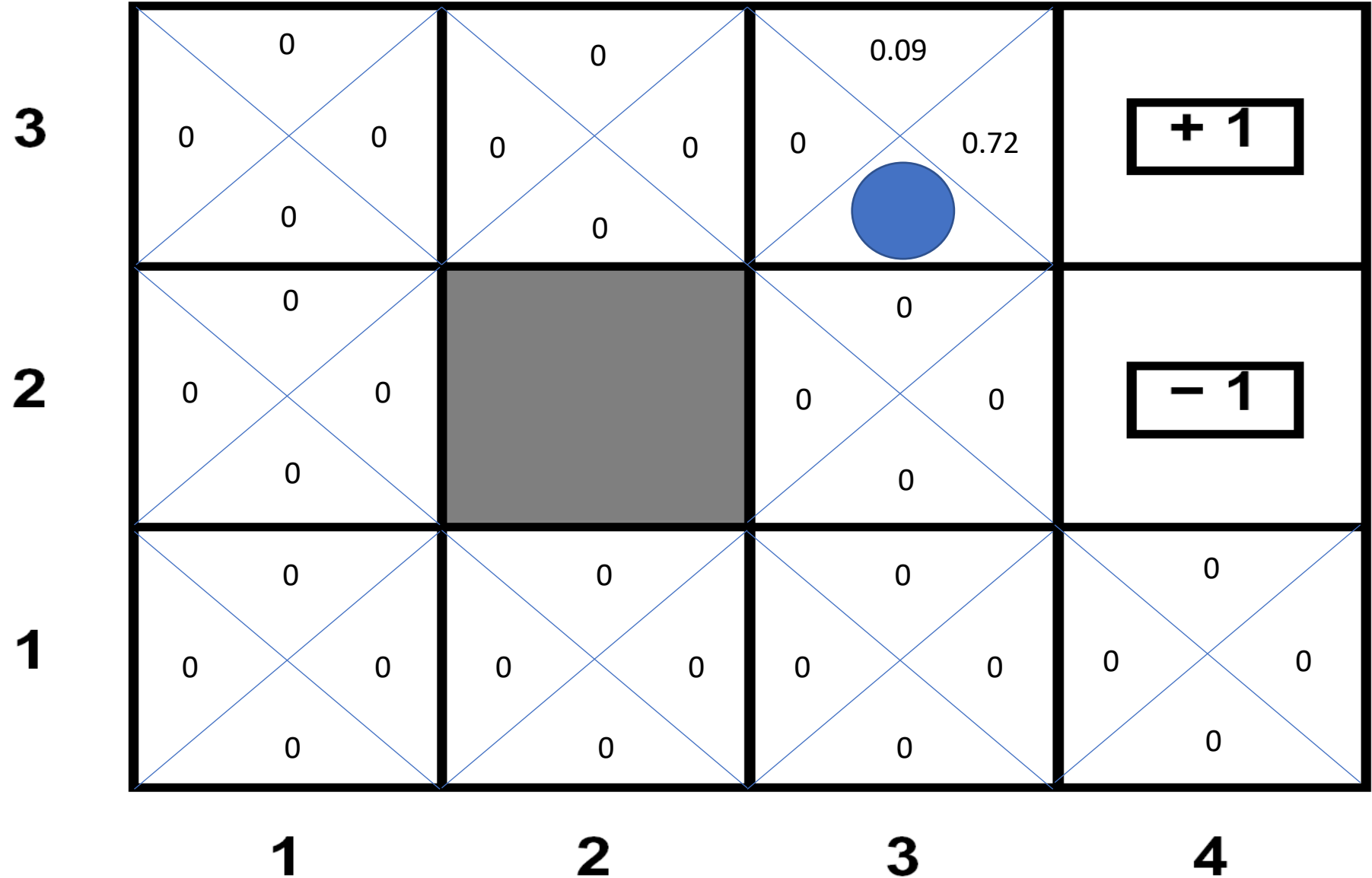
$$\begin{aligned} &0.8 \times [0 + 0] \\ &+ 0.1 \times [0 + 0.9 \times 1] \\ &+ 0.1 \times [0 + 0] \\ &= 0.09 \end{aligned}$$



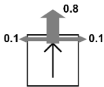
$$Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right]$$


0
0.9

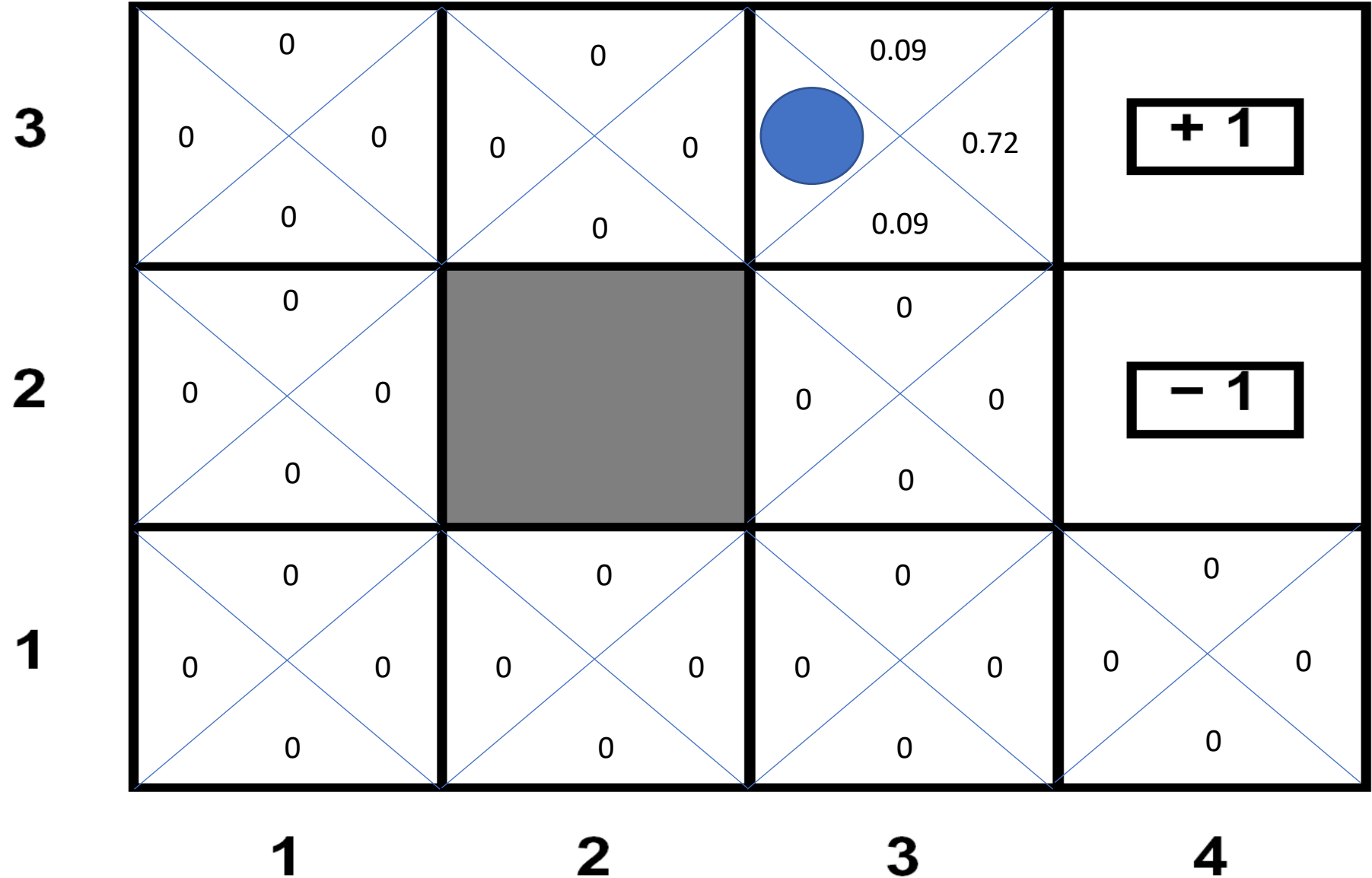
$$\begin{aligned}
 &0.8 \times [0 + 0] \\
 &+ 0.1 \times [0 + 0.9 \times 1] \\
 &+ 0.1 \times [0 + 0] \\
 &= 0.09
 \end{aligned}$$



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


0
0.9

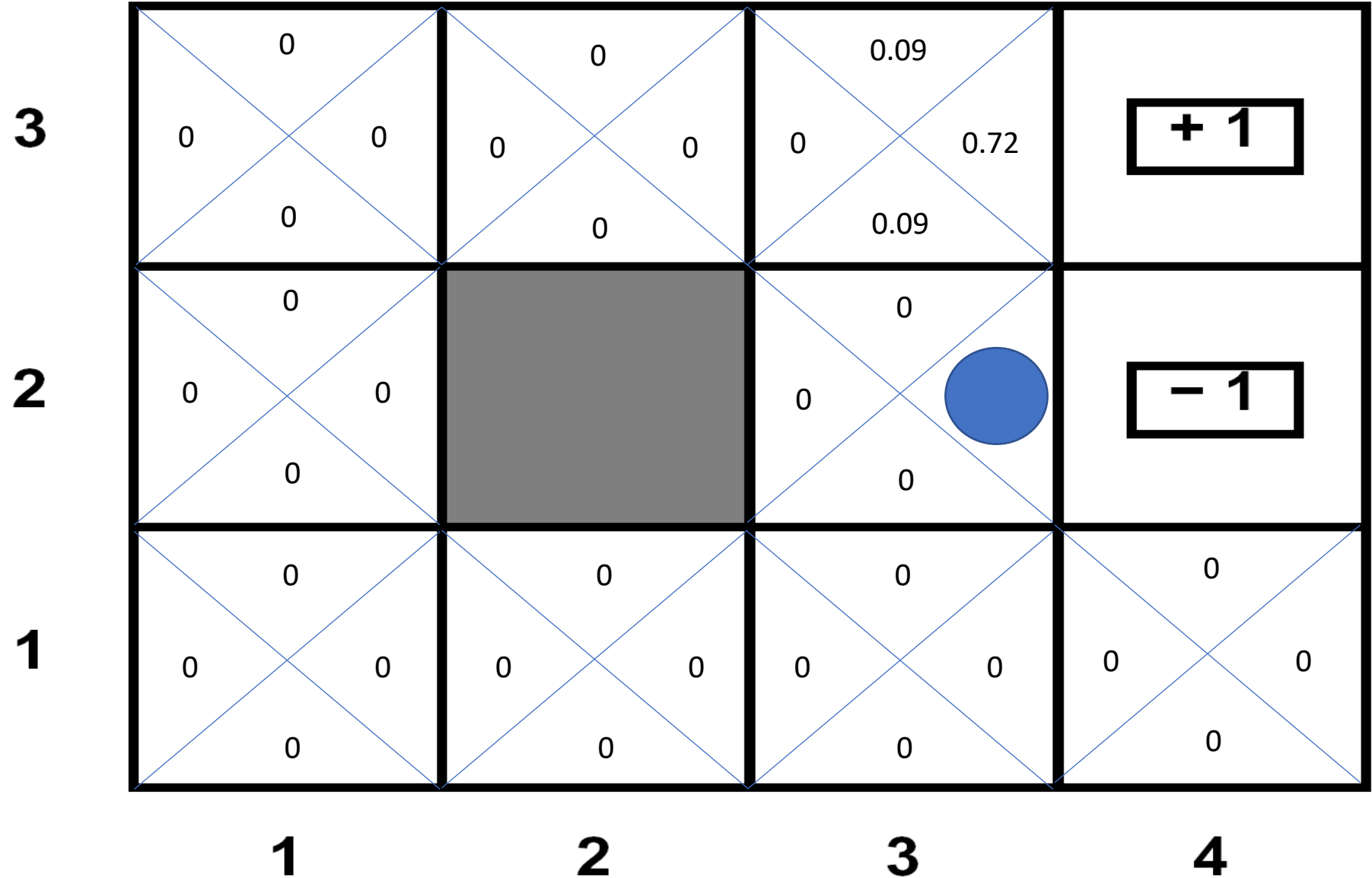
$$\begin{aligned}
 &0.8 \times [0+0] \\
 &+ 0.1 \times [0+0] \\
 &+ 0.1 \times [0+0] \\
 &= 0
 \end{aligned}$$



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

Diagram illustrating the Bellman optimality equation for Q-learning. A small diagram shows a state-action pair (s, a) with a transition probability of 0.1 to a next state s'. The equation calculates the updated Q-value for (s, a) based on the immediate reward R(s, a, s') and the discounted maximum Q-value of the next state s'.

$$\begin{aligned} &0.8 \times [0 + 0.9 \times -1] \\ &+ 0.1 \times [0 + 0] \\ &+ 0.1 \times [0 + 0] \\ &= -0.72 \end{aligned}$$



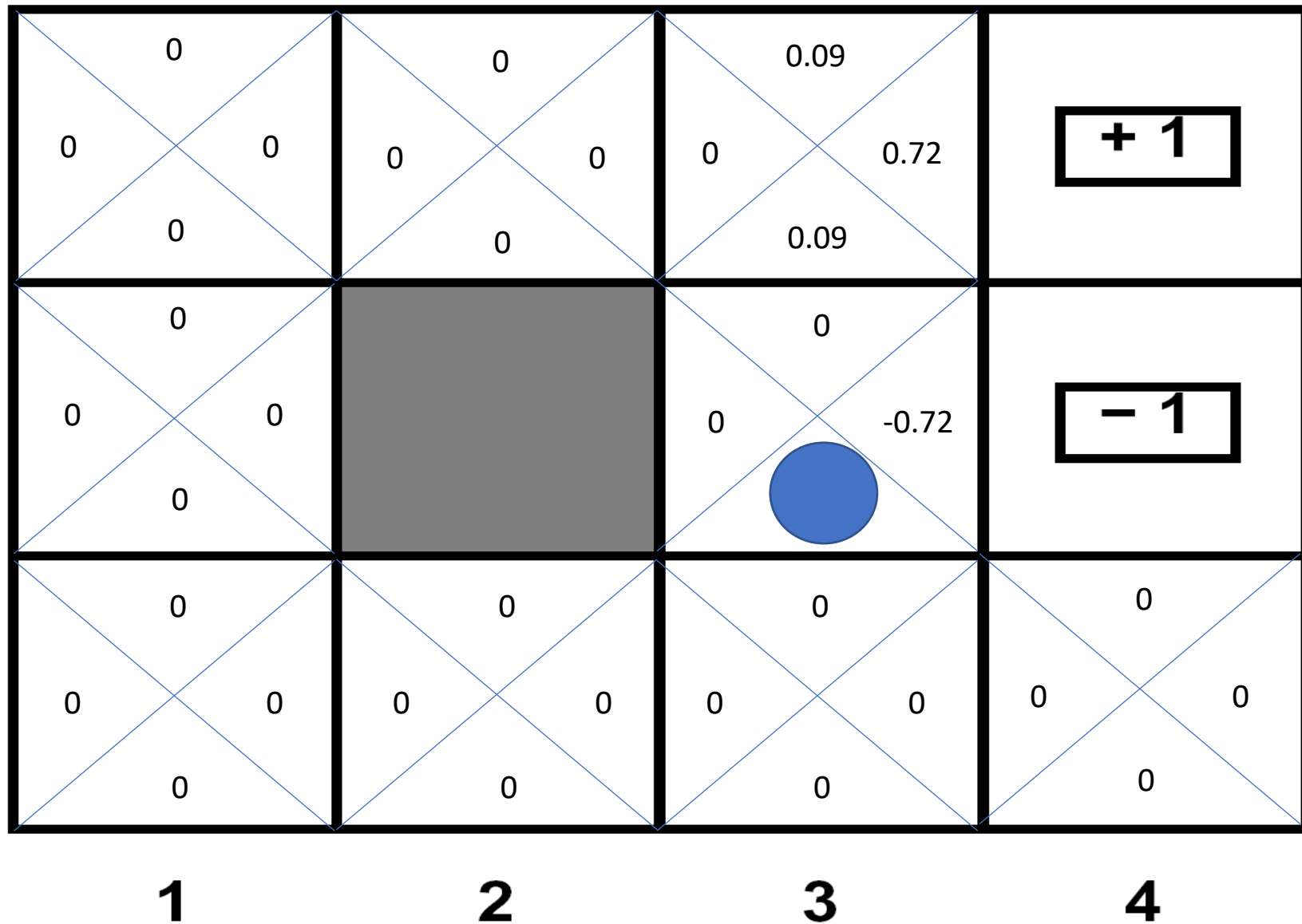


$$\begin{aligned}
 &0.8 \times [0 + 0] \\
 &+ 0.1 \times [0 + 0] \\
 &+ 0.1 \times [0 + 0.9 \times -1] \\
 &= -0.09
 \end{aligned}$$


**3**

**2**

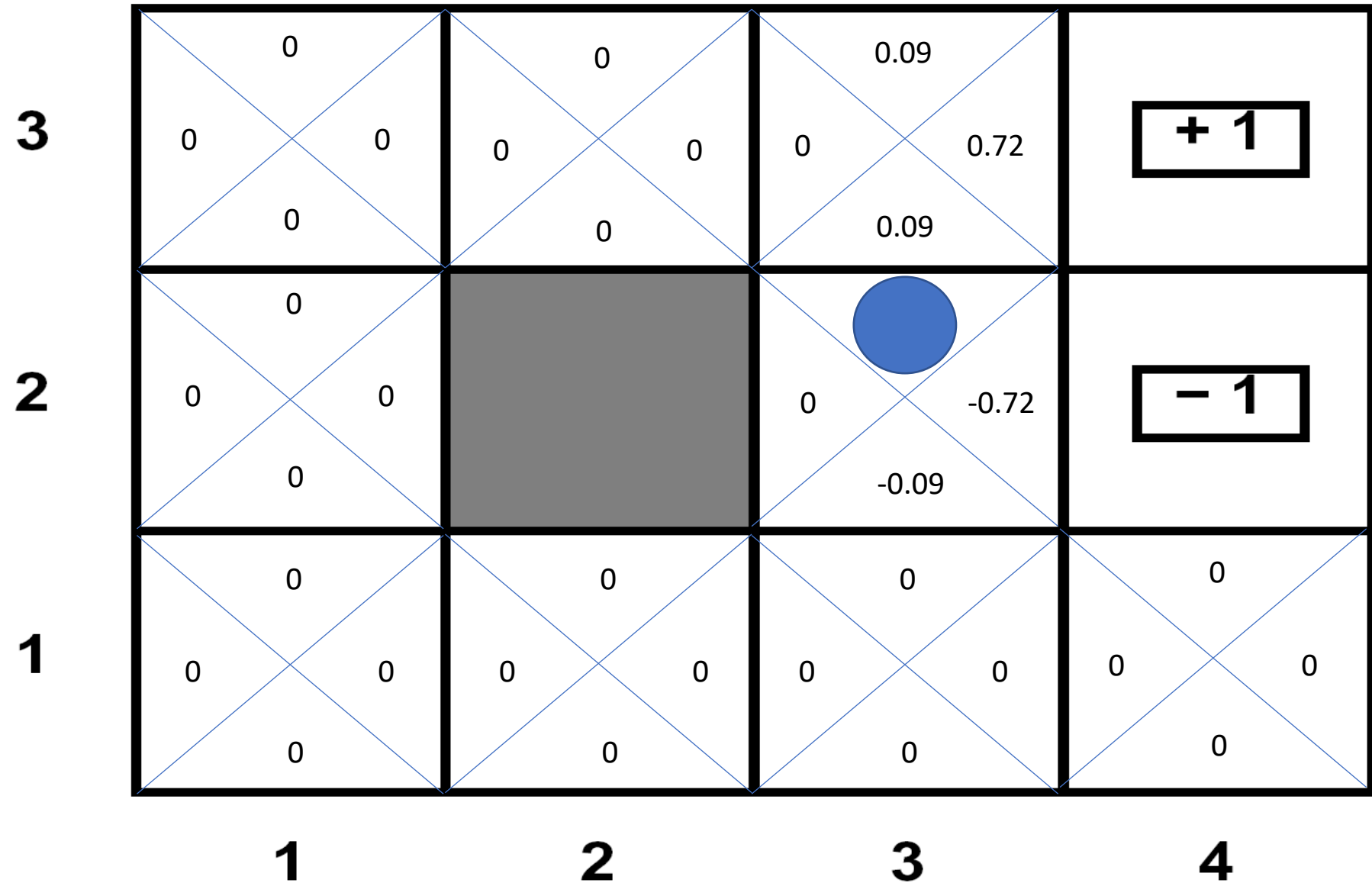
**1**



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



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

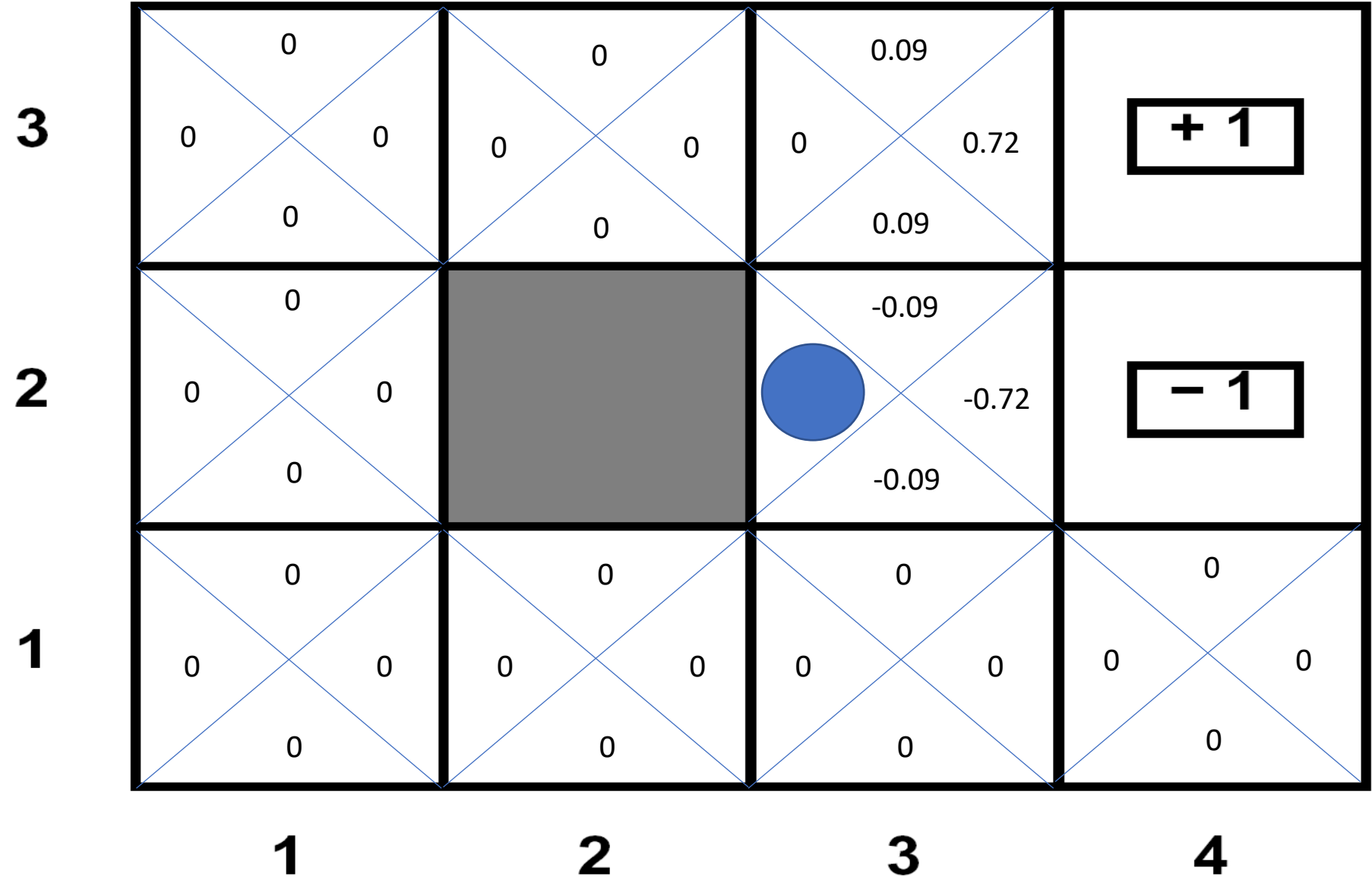

$$\begin{aligned} & 0.8x[0+0] \\ & + 0.1x[0+0.9x-1] \\ & + 0.1x[0+0] \\ & = -0.09 \end{aligned}$$

1

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

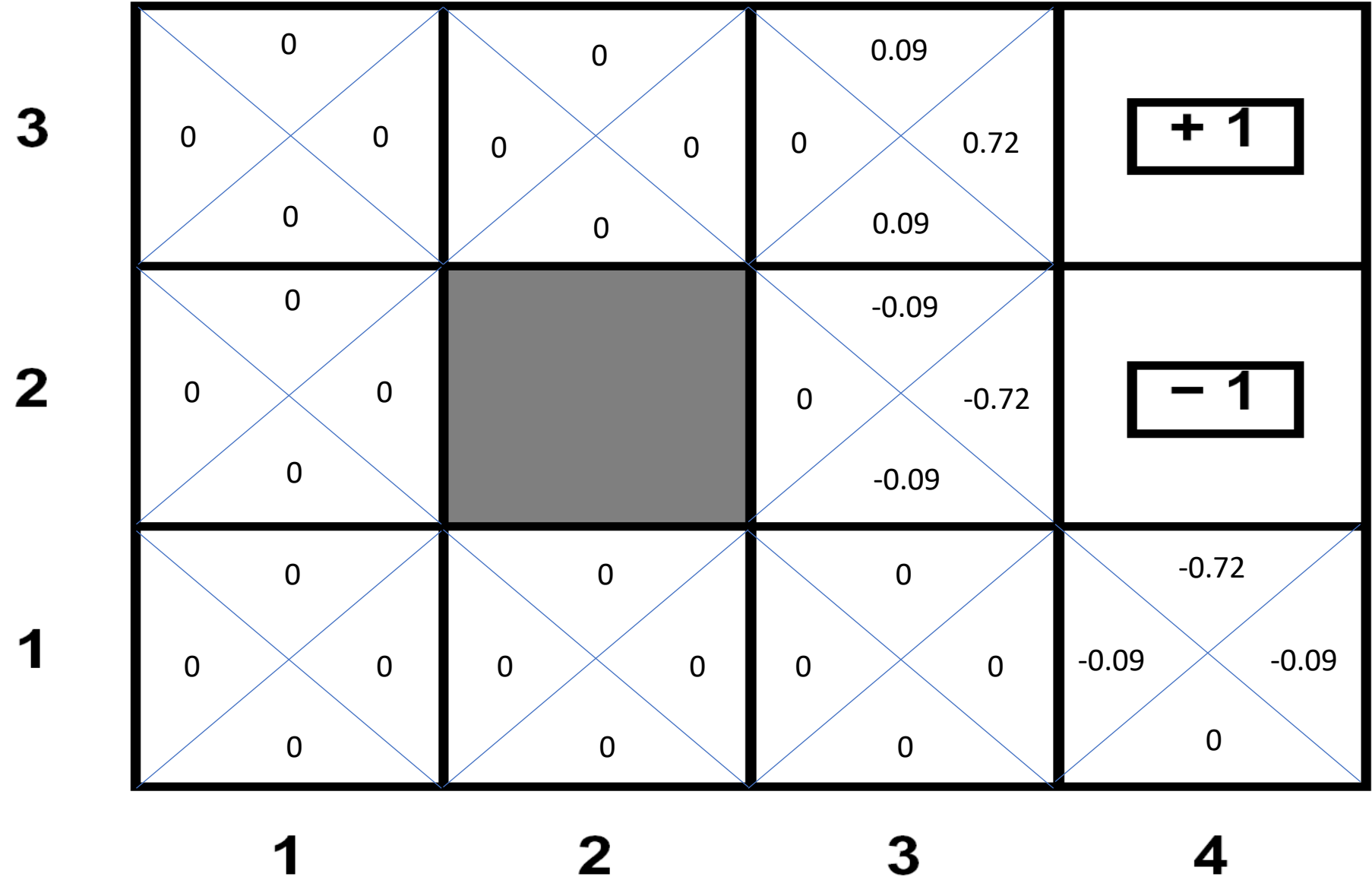
Diagram illustrating the Bellman optimality equation for Q-learning. A small diagram shows a state-action pair with a transition probability of 0.1 to a new state and a reward of 0.8. The equation shows the update of the Q-value for state  $s$  and action  $a$  based on the expected future rewards. The maximum Q-value over all actions  $a'$  is highlighted in a box.

$$\begin{aligned} &0.8 \times [0 + 0] \\ &+ 0.1 \times [0 + 0] \\ &+ 0.1 \times [0 + 0] \\ &= 0 \end{aligned}$$




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

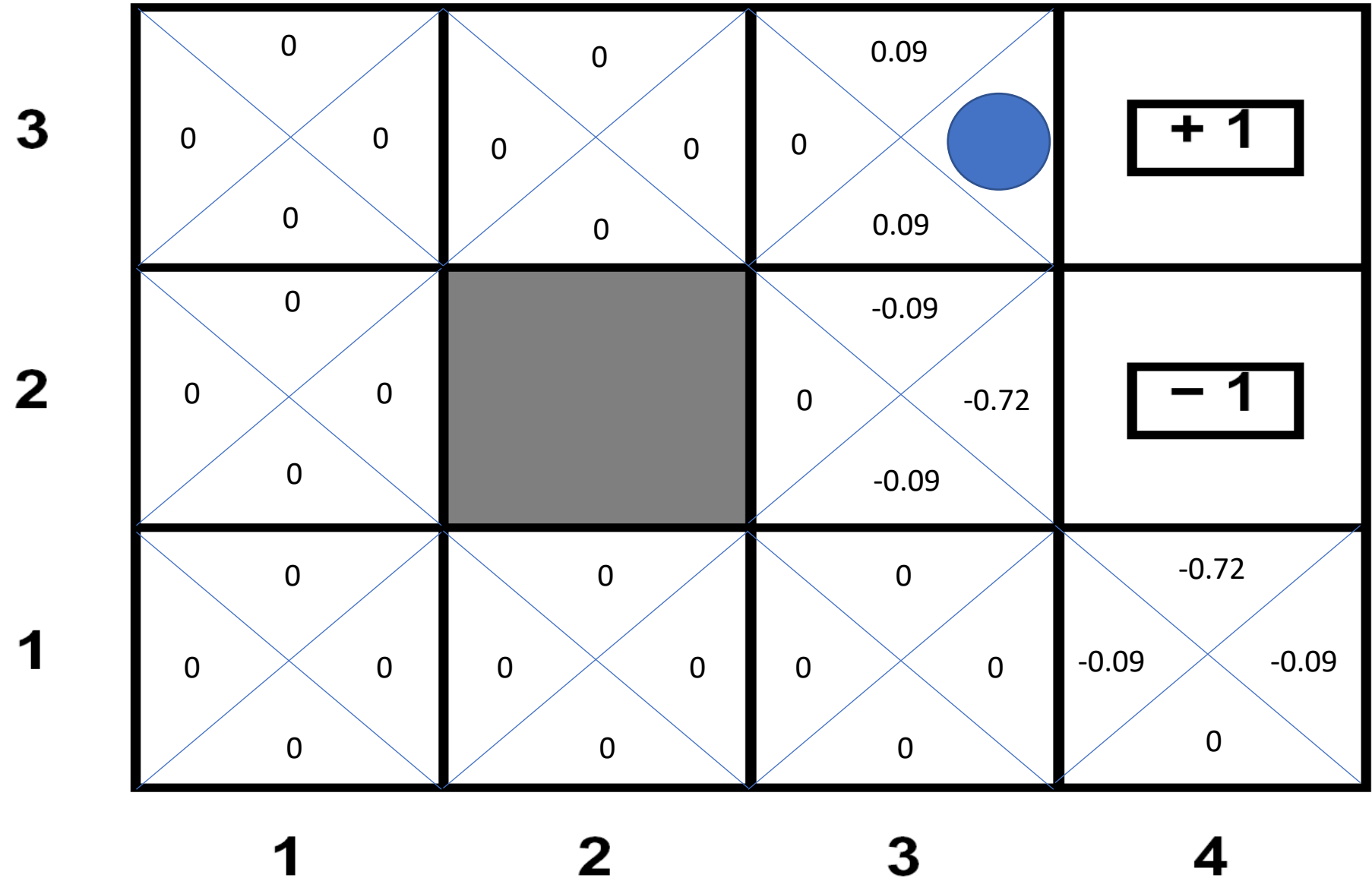
Diagram illustrating the Bellman optimality update equation. A small diagram shows a state-action pair (s, a) with a transition probability of 0.8 to a next state s'. The current state s is labeled 0, and the next state s' is labeled 0.9. The action a is labeled 0. The equation shows the update of the Q-value for (s, a) based on the expected reward and the maximum Q-value of the next state.



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



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

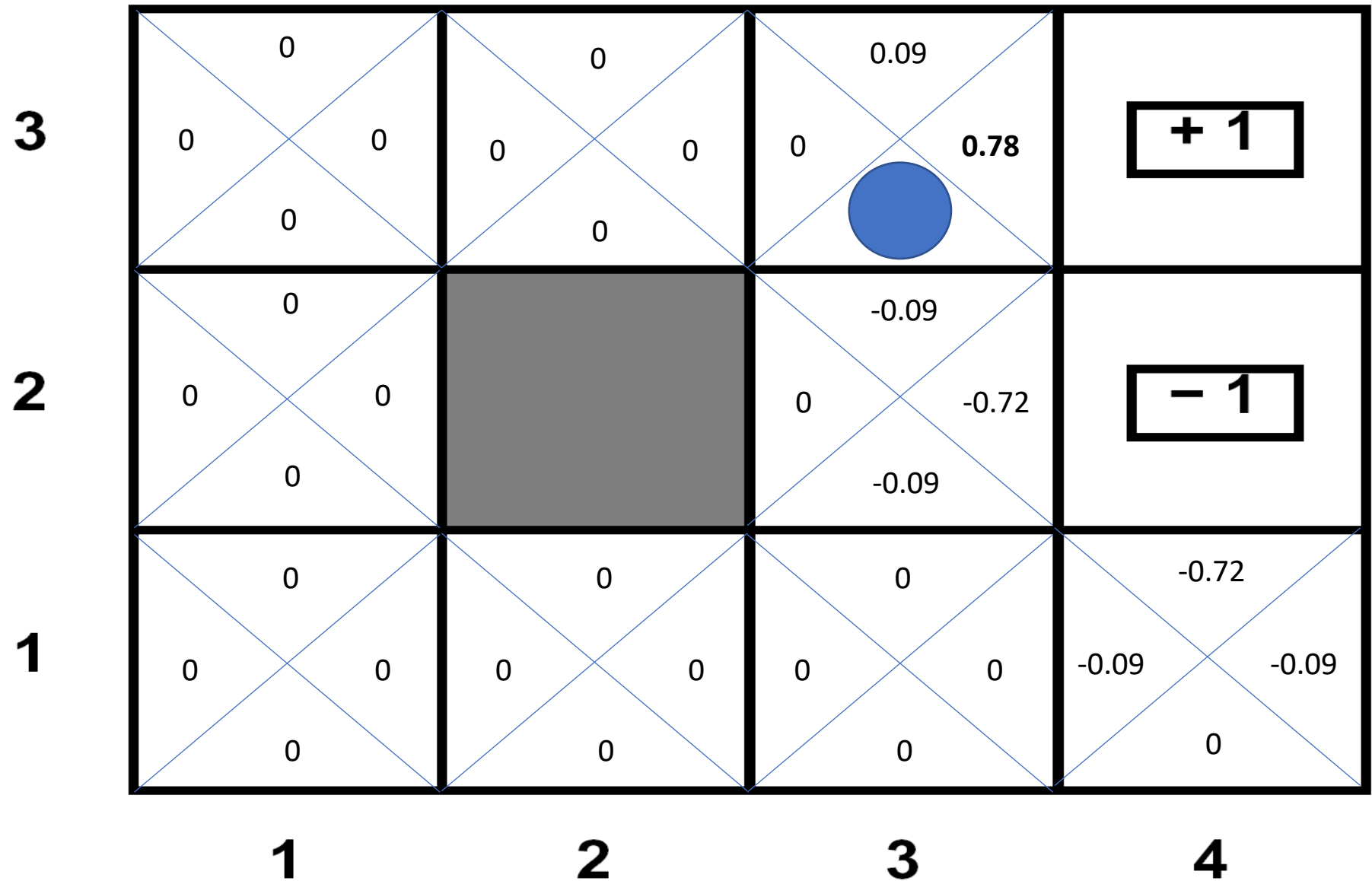

$$\begin{aligned} & 0.8x[0+0.9x1] \\ & + 0.1x[0+0.9x0.72] \\ & + 0.1x[0+0] \\ & = 0.7848 \end{aligned}$$

1

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

Diagram illustrating the Bellman optimality equation for Q-learning. A small diagram shows a state-action pair (s, a) with a transition to state s' with probability 0.1, and a reward of 0.8. The equation shows the update of the Q-value for state s and action a based on the expected reward and the maximum Q-value of the next state s'.

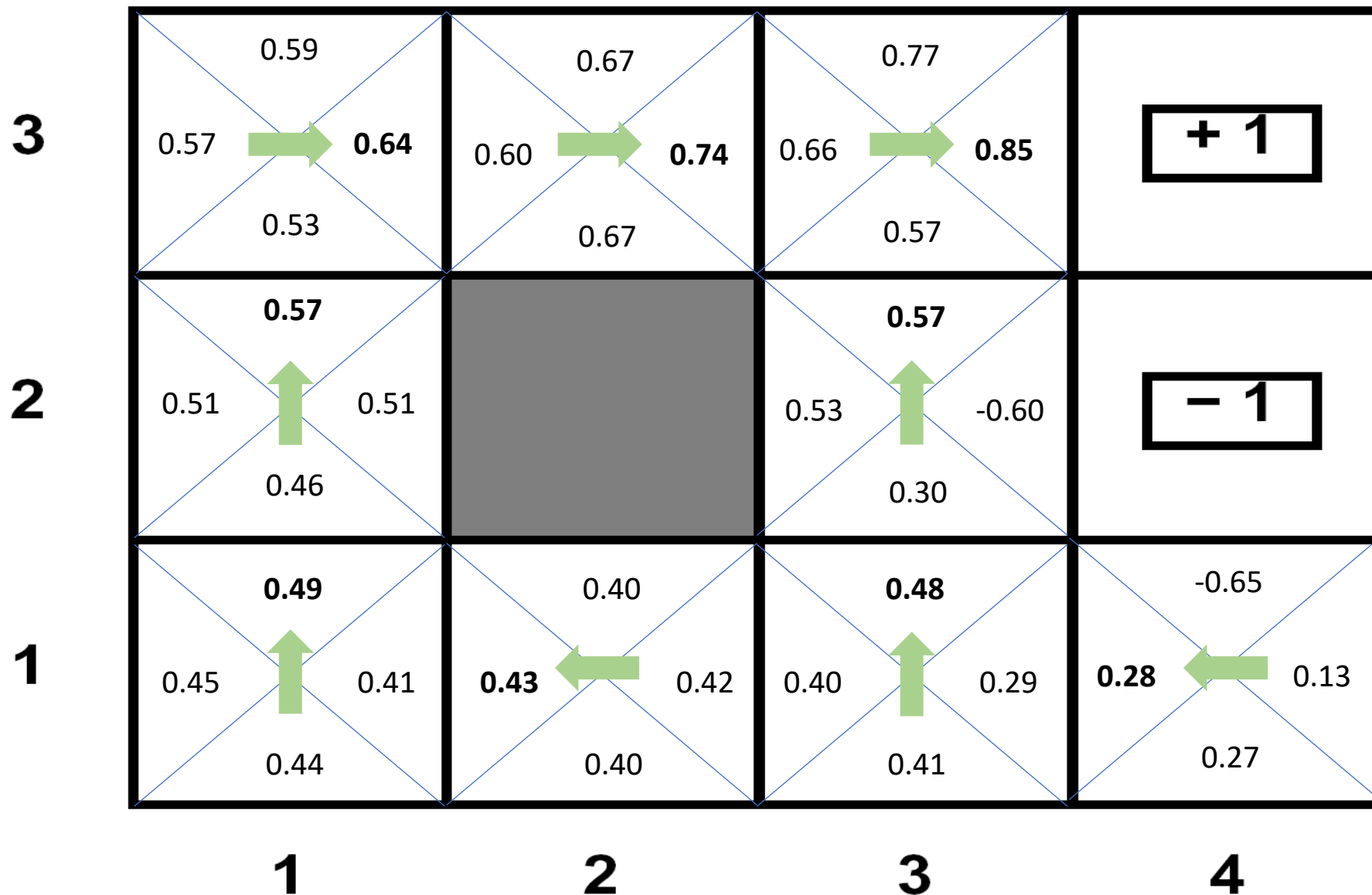
$$\begin{aligned} &0.8 \times [0 + 0] \\ &+ 0.1 \times [0 + 0.9 \times 1] \\ &+ 0.1 \times [0 + 0] \\ &= 0.09 \end{aligned}$$



After 1000 iterations:

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

Diagram illustrating the Q-learning update rule. A small grid shows a state-action pair (up, left) with a value of 0.1. A larger grid shows a state-action pair (up, right) with a value of 0.8. A blue arrow points from the value 0.8 to the term  $\max_{a'} Q_i(s', a')$  in the equation. Another blue arrow points from the value 0.9 to the term  $\max_{a'} Q_i(s', a')$ .



# Q Iteration

- Information propagates outward from terminal states
- Eventually all state-action pairs converge to correct Q-value estimates



# Aside: Value Iteration

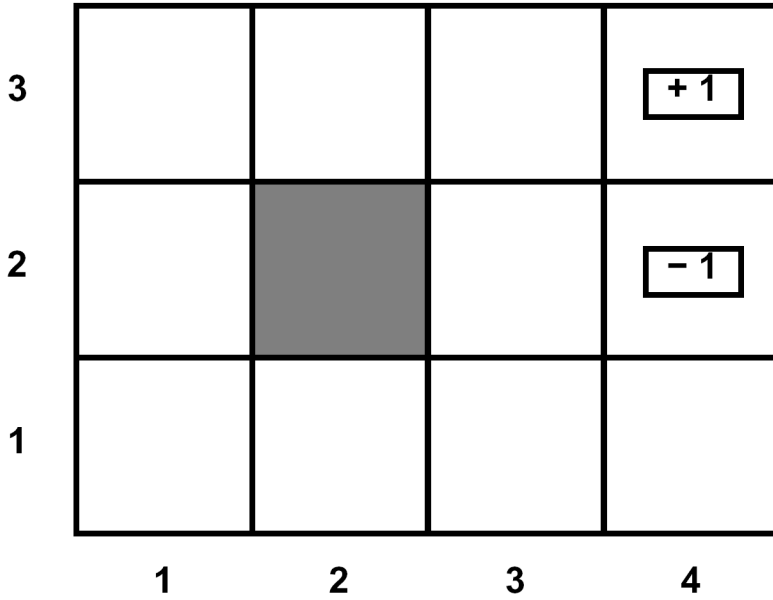
- Analogous to Q-Policy iteration but for computing the value function
- Initialize  $V_1(s) \leftarrow 0$  for all  $s$
- For  $i \in \{1, 2, \dots\}$  until convergence:

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

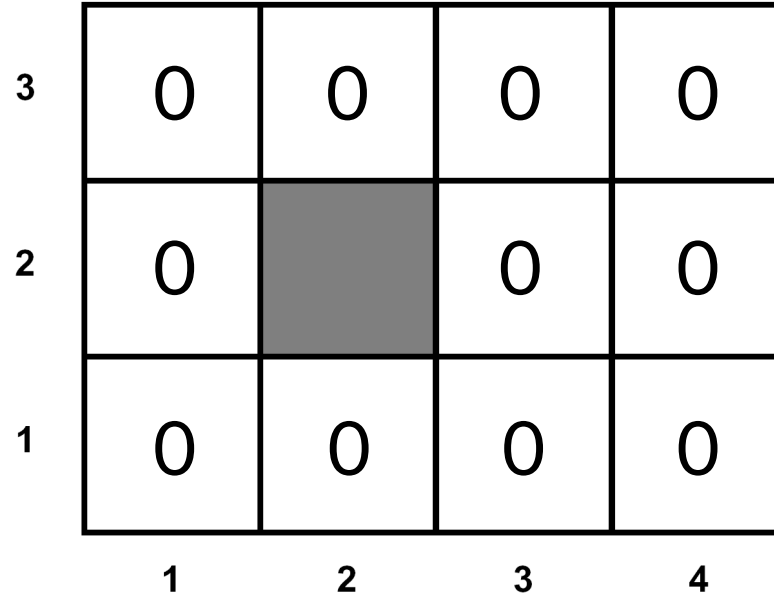
$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s, a) [R(s, a, s') + \gamma V_i(s')]$$

↙ 0      ↙ 0.9

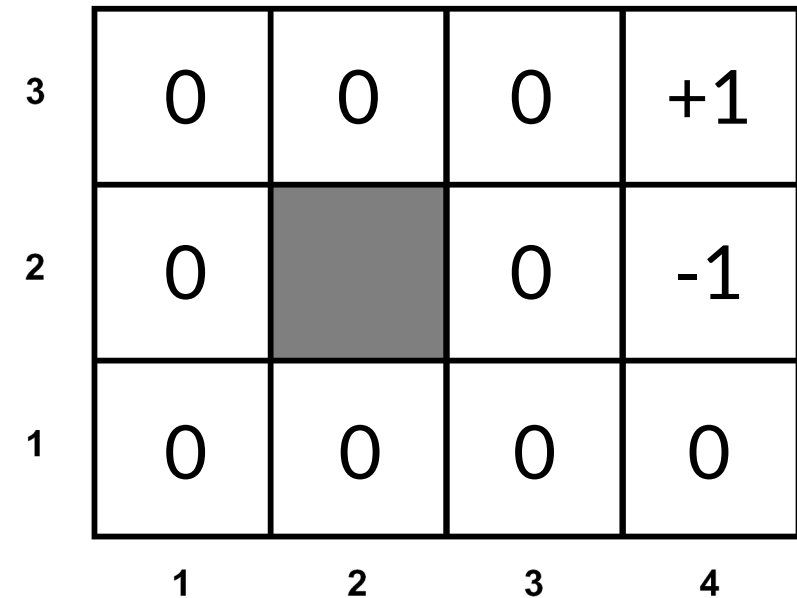
Example MDP



$V_0$



$V_1$



$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s, a) [R(s, a, s') + \gamma V_i(s')]$$

↙ 0
↙ 0.9

Example MDP

3				<div style="border: 1px solid black; padding: 2px;">+1</div>
2				<div style="border: 1px solid black; padding: 2px;">-1</div>
1				
	1	2	3	4

$$V_2(\langle 4, 3 \rangle) \leftarrow 1$$

$V_1$

3	0	0	0	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

$$V_2(\langle 4, 2 \rangle) \leftarrow -1$$

$V_2$

3	0	0	0.72	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s, a) [R(s, a, s') + \gamma V_i(s')]$$

↙ 0
↙ 0.9

Example MDP

3				<div style="border: 1px solid black; padding: 2px;">+1</div>
2				<div style="border: 1px solid black; padding: 2px;">-1</div>
1				
	1	2	3	4

$V_2$

3	0	0	0.72	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

$V_3$

3	0	0.52	0.78	+1
2	0		0.43	-1
1	0	0	0	0
	1	2	3	4

# 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?

# Model-Based Reinforcement Learning

- **Step 1:** Estimate  $\hat{P} \approx P$  and  $\hat{R} \approx R$  from samples
  - What policy to use to gather data?
  - Need to take action  $a$  in state  $s$  to obtain an observation of  $P(\cdot | s, a)$ !
  - More on this later

	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)	(4,1)	(4,2)	(4,3)
(1,1), N	0.1	0.8	0	0.1	0	0	0	0	0	0	0	0
(1,1), E	0.1	0.1	0	0.8	0	0	0	0	0	0	0	0
(1,1), S	0.9	0	0	0.1	0	0	0	0	0	0	0	0
...												

- **Step 2:** Compute optimal policy  $\hat{\pi} \approx \pi^*$  for  $\hat{P}$  and  $\hat{R}$

# Model-Free Reinforcement Learning

- Can we learn  $\pi^*$  without explicitly learning  $P$  and  $R$ ?
- **Q Learning**
  - Can we extend Q Iteration to the setting where  $P$  and  $R$  are unknown?
  - **Observation:** Every time you take action  $a$  from state  $s$ , you obtain one sample  $s' \sim P(\cdot | s, a)$  and observe  $R(s, a, s')$
  - Use single sample instead of full  $P$

# Q Learning

- Can we learn  $\pi^*$  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  $\pi^*$  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:**

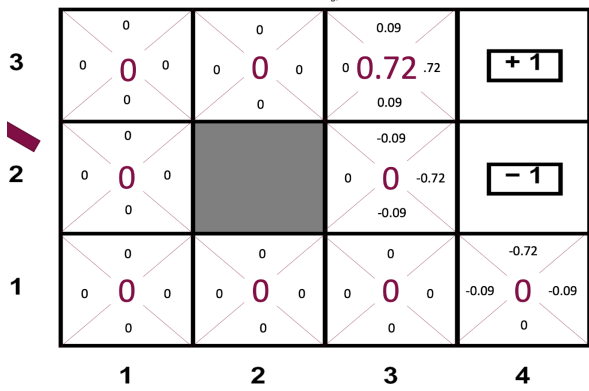
$$Q_{i+1}(s, a) \leftarrow R(s, a, s') + \gamma \cdot \max_{a' \in A} Q_i(s', a')$$

- **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  $s' \sim P(\cdot | s, a)$ )

# Q Learning

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

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



Sample  $R + \gamma \max Q =$   
 $0 + 0.9 \times 0.72 = 0.648$

New  $Q =$   
 $0.09 + 0.1 \times (0.648 - 0.09)$   
 $= 0.1458$

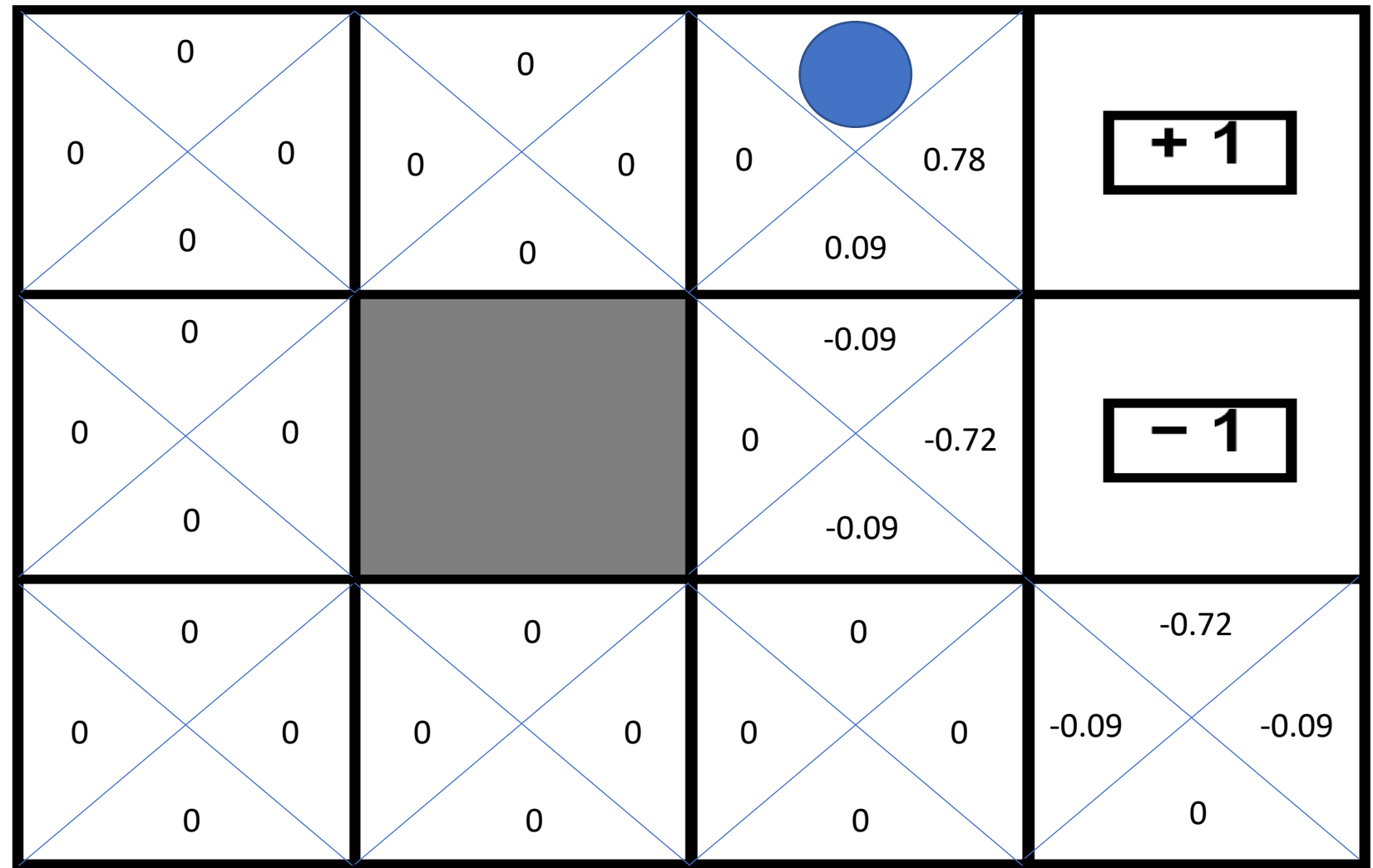
**3**

**2**

**1**

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

0.1      0.9



**1**

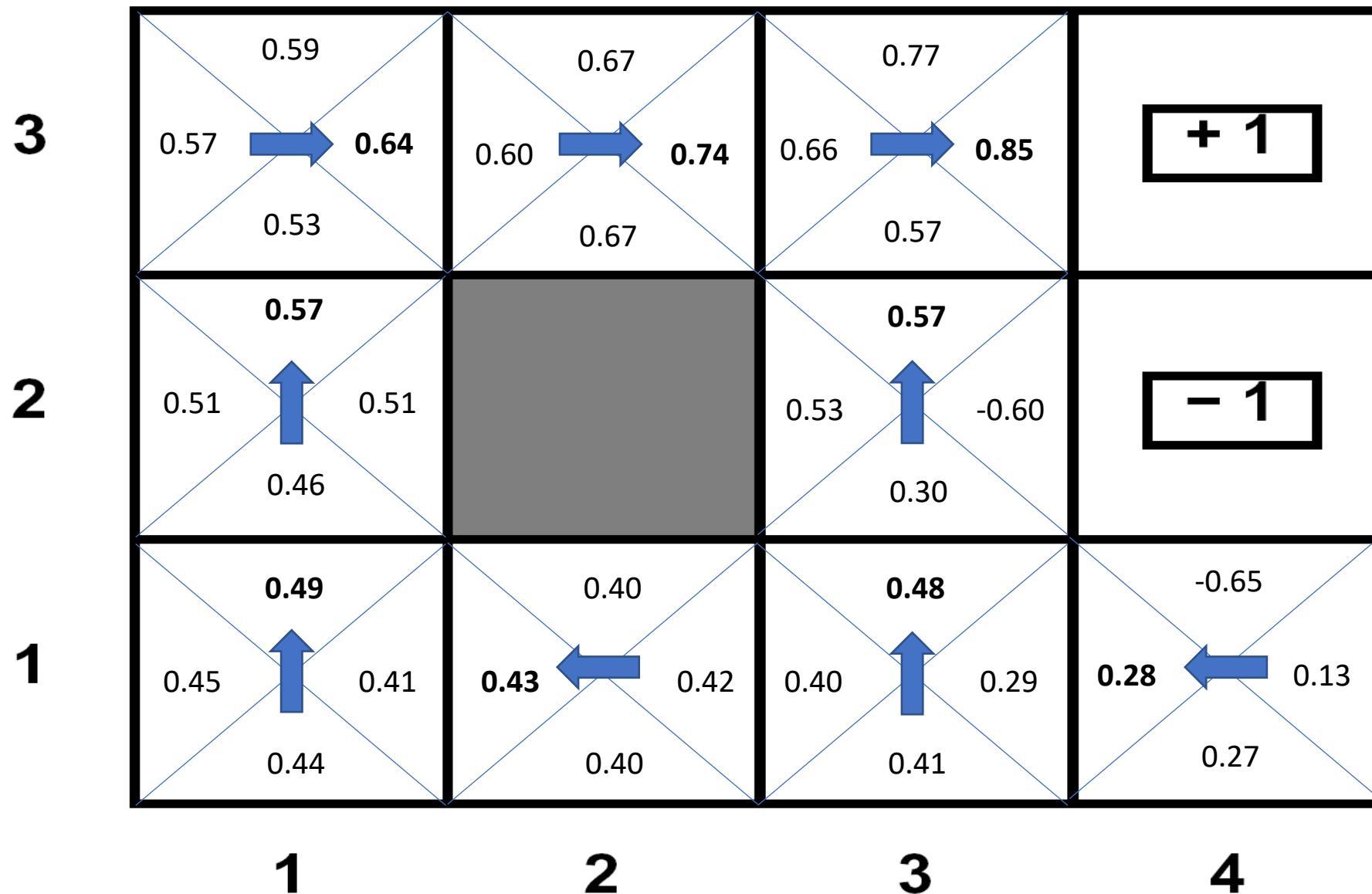
**2**

**3**

**4**

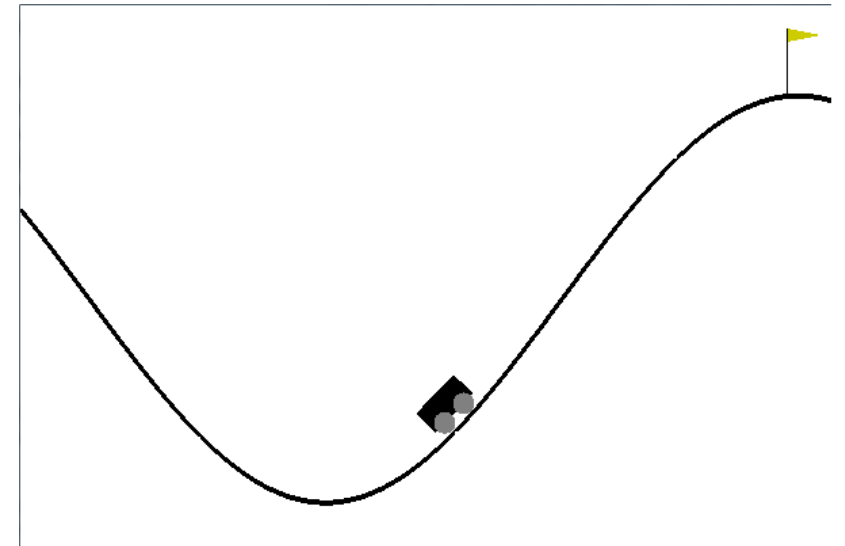
After 100,000 actions:

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



# 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

- **$\epsilon$ -greedy:**

- Play current best with probability  $1 - \epsilon$  and randomly with probability  $\epsilon$
- Can reduce  $\epsilon$  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

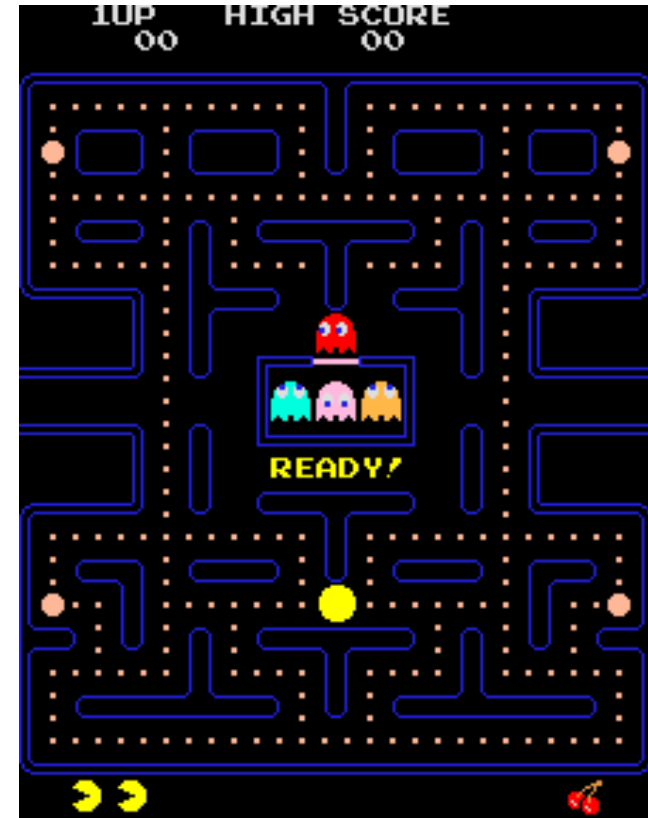
# Summary

- **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
  - **Next time:** 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 (player, ghost<sub>1</sub>, ..., ghost<sub>k</sub>), 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

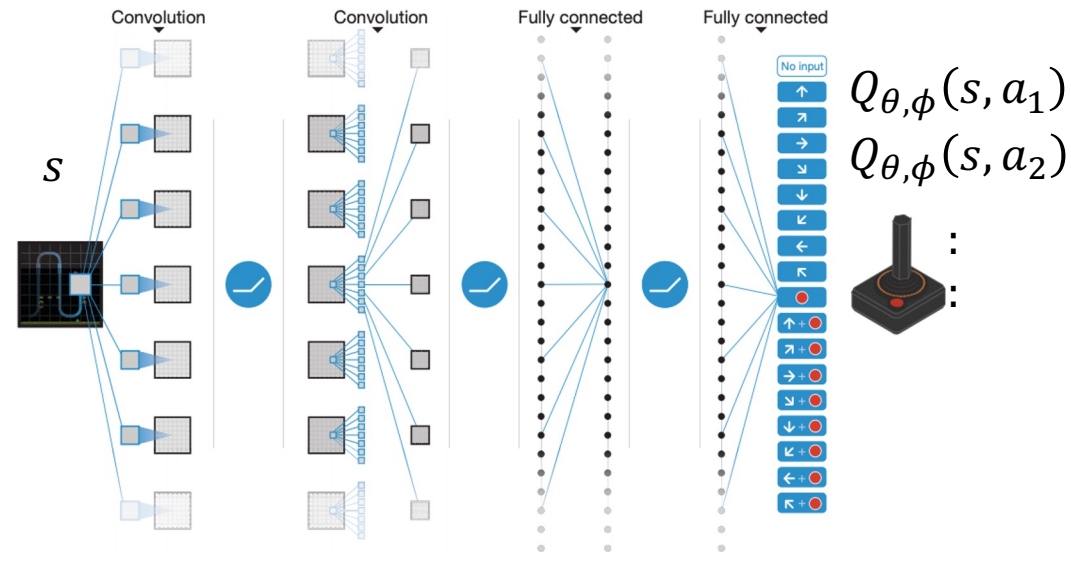


# State-Action Features

- Can we learn **across** state-action pairs?
- Yes, use features!
  - $\phi(s, a) \in \mathbb{R}^d$
  - Then, learn to predict  $Q^*(s, a) \approx Q_\theta(s, a) = f_\theta(\phi(s, a))$
  - 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:

$$\left( \underbrace{\left( R(s, a, s') + \gamma \cdot \max_{a'} Q_{\theta}(s', a') \right)}_{\text{"Label" } y} - Q_{\theta}(s, a) \right)^2$$

# 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

# Experience Replay Buffer

- **Problem**

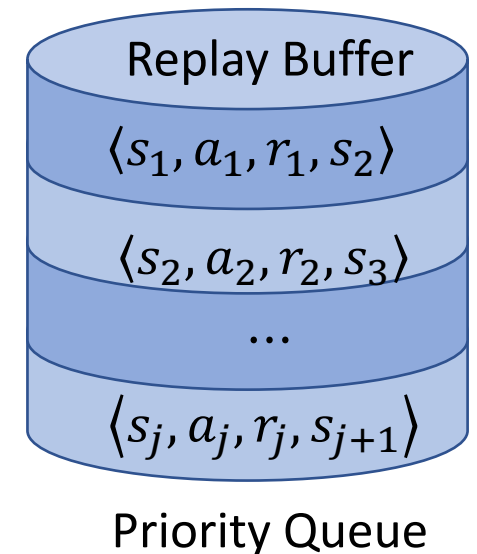
- Sequences of states are highly correlated
- Tend to overfit to current states and forget older states

- **Solution**

- Keep a **replay buffer** of observations (as a priority queue)
- Gradient updates on samples from replay buffer instead of current state

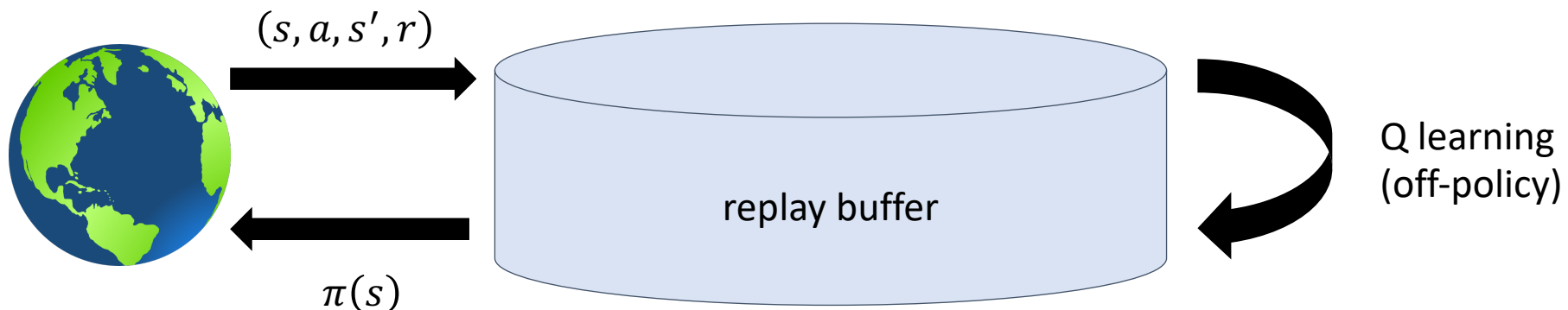
- **Advantages**

- Breaks correlations between consecutive samples
- Can take multiple gradient steps on each observation



# 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  $k \in \{1, \dots, K\}$ :
    - Sample  $(s_{i,k}, a_{i,k}, s_{i+1,k}, r_{i,k})$  from  $D$
    - $y_{i,k} \leftarrow r_{i,k} + \gamma \cdot \max_{a' \in A} Q_{\theta}(s_{i+1,k}, a')$
    - $\phi \leftarrow \phi - \alpha \cdot \frac{d}{d\theta} (Q_{\theta}(s_{i,k}, a_{i,k}) - y_{i,k})^2$



# Target Q Network

- **Problem**

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

- **Solution**

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



# 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  $k \in \{1, \dots, K\}$ :
    - Sample  $(s_{i,k}, a_{i,k}, s_{i+1,k}, r_{i,k})$  from  $D$
    - $y_{i,k} \leftarrow r_{i,k} + \gamma \cdot \max_{a' \in A} Q_{\theta'}(s_{i+1,k}, a')$
    - $\phi \leftarrow \phi - \alpha \cdot \frac{d}{d\theta} (Q_{\theta}(s_{i,k}, a_{i,k}) - y_{i,k})^2$
  - Every  $N$  steps,  $\theta' \leftarrow \theta$

# Deep Q Learning for Atari Games

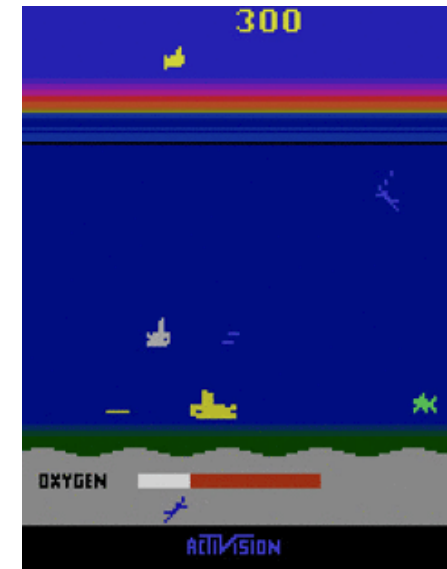
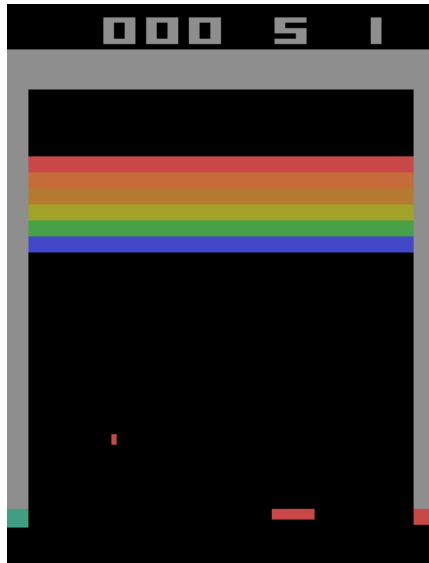


Image Sources:

<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/>