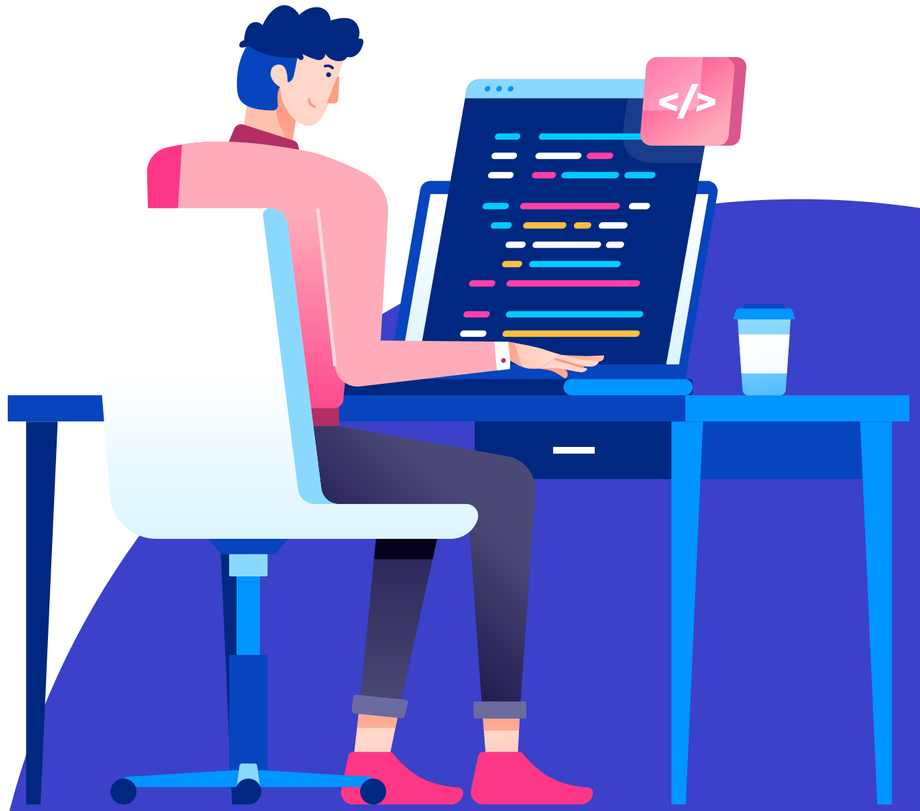
OPEN AI

Search . . .

GYM TOOLKIT

FOR STUDENTS



OpenAI Gym is a toolkit for reinforcement learning algorithms with a diverse range of environments. However, custom environments may be needed, in which case we must create our own for testing algorithms.

OPEN AI

Search . . .

WE WILL COVER THE FOLLOWING TOPICS:

01 SETTING UP OUR MACHINE

INSTALLING ANACONDA AND GYM

02 UNDERSTANDING THE GYM ENVIRONMENT

GENERATING AN EPISODE IN THE GYM ENVIRONMENT

EXPLORING MORE GYM ENVIRONMENTS

CART-POLE BALANCING WITH THE RANDOM AGENT

AN AGENT PLAYING THE TENNIS GAME

Search . . .

SETTING UP OUR MACHINE

INSTALLING ANACONDA

https://www.anaconda.com/download/

INSTALLING THE GYM TOOLKIT

pip install gym

CREATING OUR FIRST GYM ENVIRONMENT

https://github.com/baz2024/DBS_ReinforcementLearning24/tree/main

OPEN AI

Search . . .

CREATING A SIMPLE GYM ENVIRONMENT

Step 1: Install Gym

Step 2: Create a New Python File

Step 3: Import Gym and NumPy

Step 4: Define Your Environment Class

Step 5: Register Your Environment

Step 6: Test Your Environment

Read More. . .

FROZEN LAKE

S 0	F 1	F 2	F 3
F 4	H 5	F 6	H 7
F 8	F 9	F 10	H 11
H 12	F 13	F 14	G 15

ACTION SPACE:
DISCRETE(4)

- **0: MOVE LEFT**
- **1: MOVE DOWN**
- **2: MOVE RIGHT**
- **3: MOVE UP**

Observation Space Discrete(16)

Rewards

- Reach goal: +1
- Reach hole: 0
- Reach frozen: 0

OPEN AI

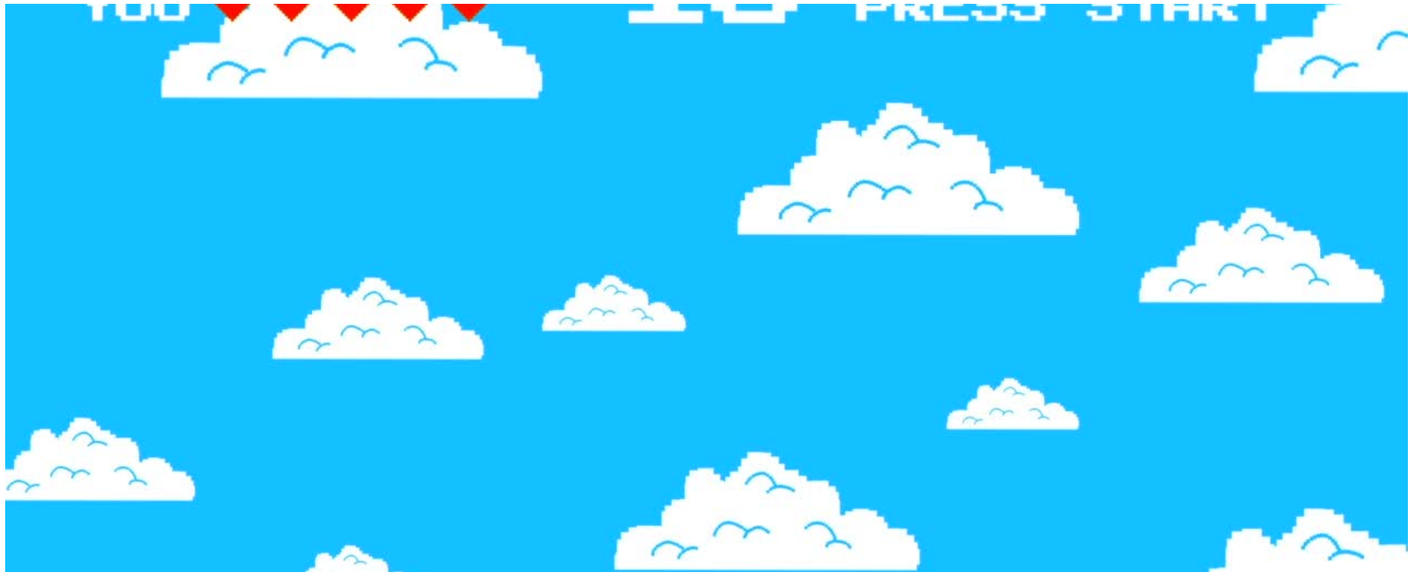
Search . . .

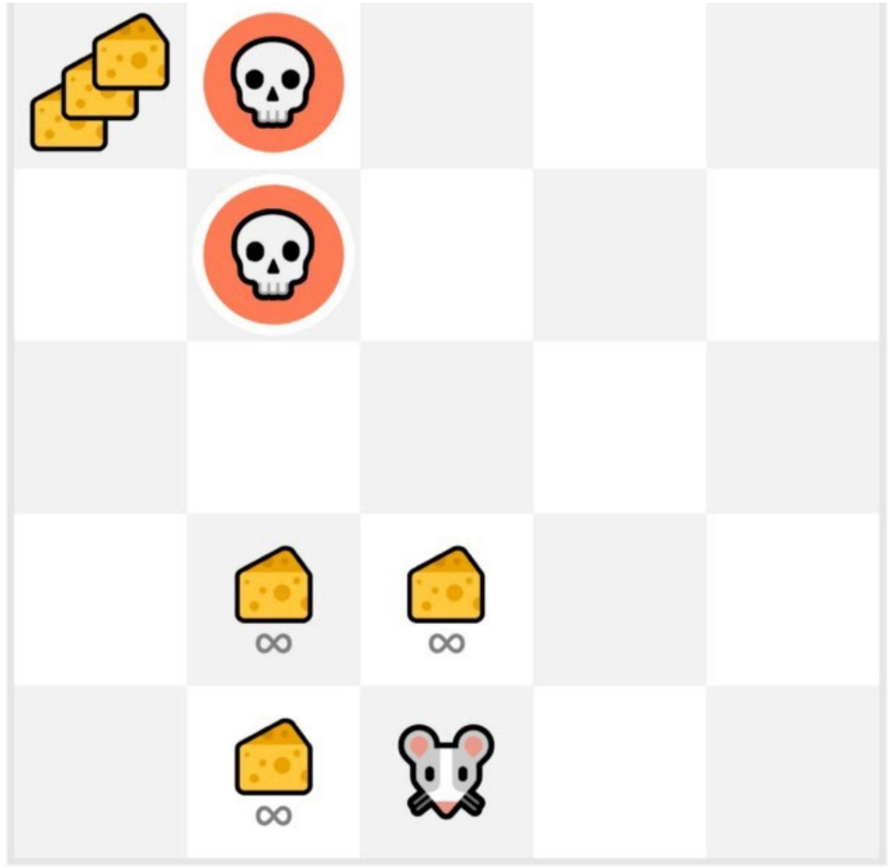
CREATING CUSTOM GYM ENV

- Defining the state space, action space, and reward function is necessary.
- The state space includes possible environment states
- The action space contains the actions the agent can take
- The reward function determines the reward for each action in a state.

The Balancing Act between Exploration and Exploitation

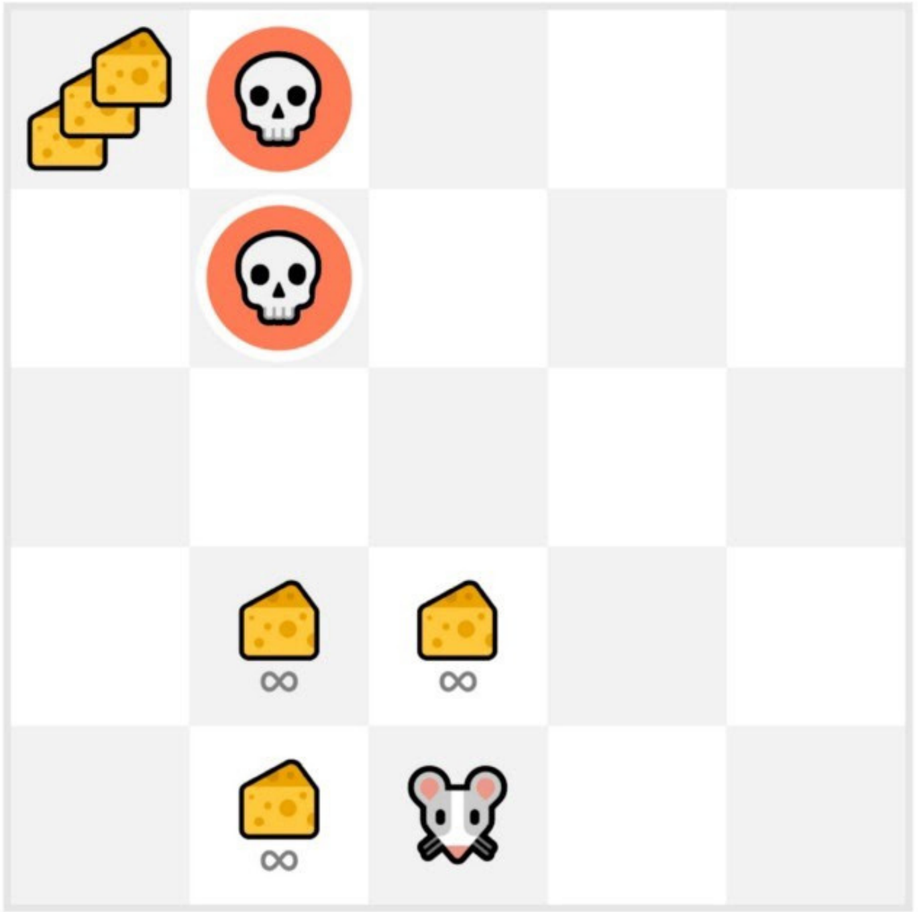
The exploration/exploitation trade-off is crucial to understand before solving problems in Reinforcement Learning.
Exploration involves trying random actions to gain information, while exploitation uses known information to maximize rewards. The objective is to maximize the expected cumulative reward, but it's important to balance exploration and exploitation.





Collecting as many small cheeses as possible is simple for the mouse (+1 for each). But to achieve an impressive score, the mouse should aim for the top of the maze where a massive chunk of cheese awaits (+1000).

1. Relying solely on exploitation may not lead to a substantial cheese yield, as the agent might only exploit the nearest reward source, even if it's small.
2. By incorporating some exploration, the agent can uncover larger rewards, such as a big pile of cheese.
3. Balancing exploration and exploitation is crucial, as there is a trade-off between the two in the environment.

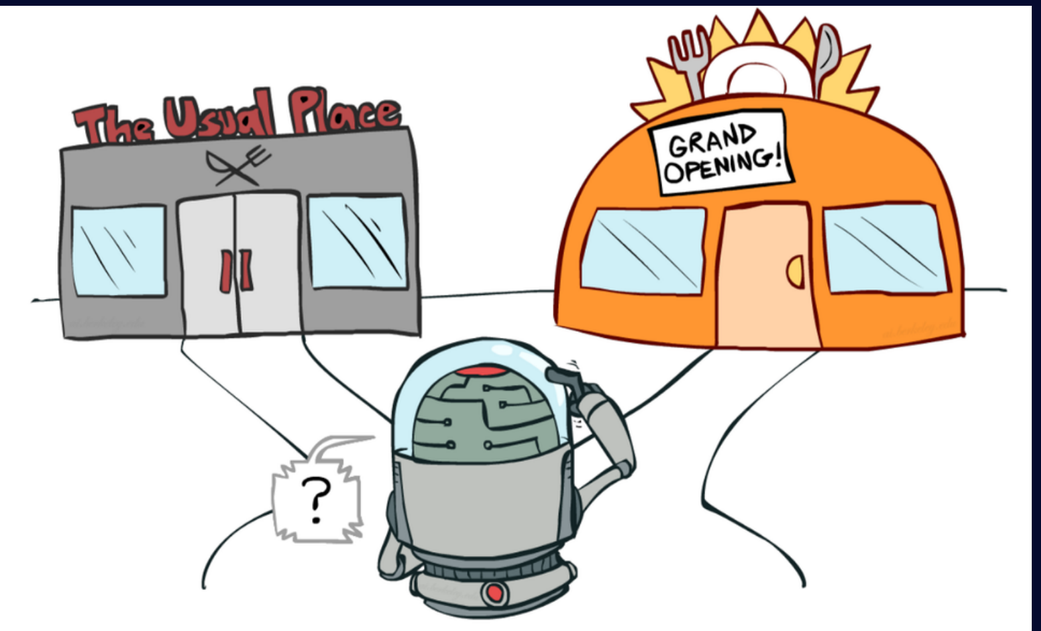


As a result, a rule must be established to manage this trade-off effectively.

In the upcoming units, we will explore various methods to address it.

Choosing a Restaurant!

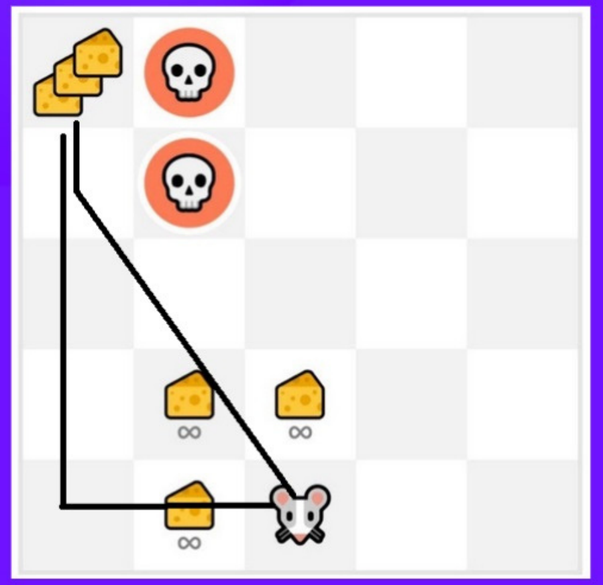
Exploitation: You go to the same one that you know is good every day and take the risk to miss another better restaurant.



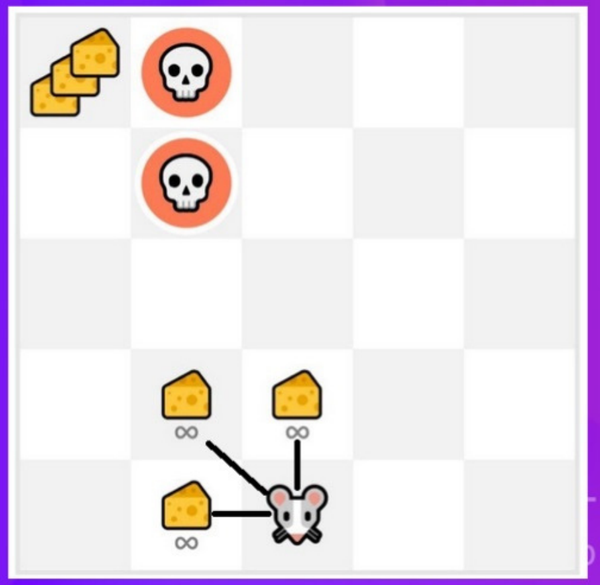
Exploration: Try restaurants you never went to before, with the risk of having a bad experience but the probable opportunity of a fantastic experience.

Exploration/ Exploitation tradeoff

Exploration: trying **random actions** in order to find **more information about the environment.**



Exploitation: using known information to **maximize the reward.**



Policy-Based Methods

RL Approches

Value-based methods

Creating an RL Agent that Prioritizes Actions for Maximum Cumulative Reward

To put it simply, our goal is to develop an RL agent that selects actions that will result in the greatest cumulative reward.

The Two Primary Methods for Addressing RL Problems.

The Policy π: the agent’s brain

- The Role of Policy π in Our Agent's Decision-Making Process
- The Policy π is the key component of our Agent, as it determines the most appropriate action to take based on the current state.
- Essentially, it outlines the behavior of the agent at any given moment.
- Policy is the brain of our agent, a function that determines the appropriate action to take based on a given state.

Our objective is to train our agent to become an exceptional policy-maker. Our primary goal is to determine the optimal policy, π^*, that generates the highest rewards for the agent when it performs the correct action.

There are two approaches to train our agent to find this optimal policy π^*:

- The process of instructing agents to determine the best course of action based on the current state is referred to as Policy-Based Methods.

- Teaching an agent to identify more valuable states and take actions that lead to them can be achieved through value-based methods.

Defining a mapping function for optimal action selection by establishing a mapping from each state to the corresponding action or creating a probability distribution over the available actions.

As evident from this, the deterministic policy explicitly specifies the action to be taken for each step.

Policy Types:

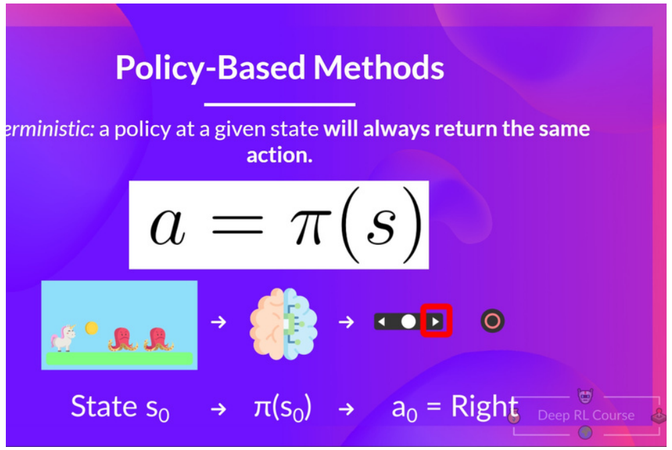
Deterministic: a policy at a given state will always return the same action.

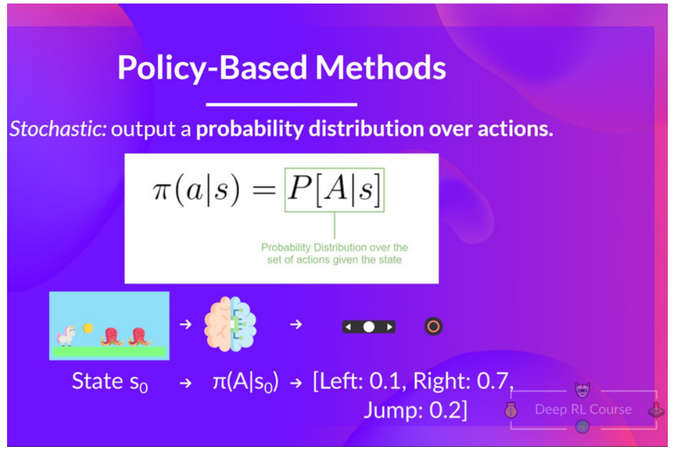
action = policy(state)

$$a = \pi(s)$$

$$\pi(a|s) = P[A|s]$$

Probability Distribution over the set of actions given the state





The stochastic policy produces an output that offers a probability distribution among various actions.

policy(actions | state) = probability distribution over the set of actions given the current state

Value-based methods

- Rather than learning a policy function, value-based methods rely on a value function that connects a state with the anticipated value of being in that state.
- In reinforcement learning, the value of a state is calculated based on the expected return an agent can receive if it begins in that state and follows our policy. This value is discounted to reflect the time delay between taking an action and receiving a reward.

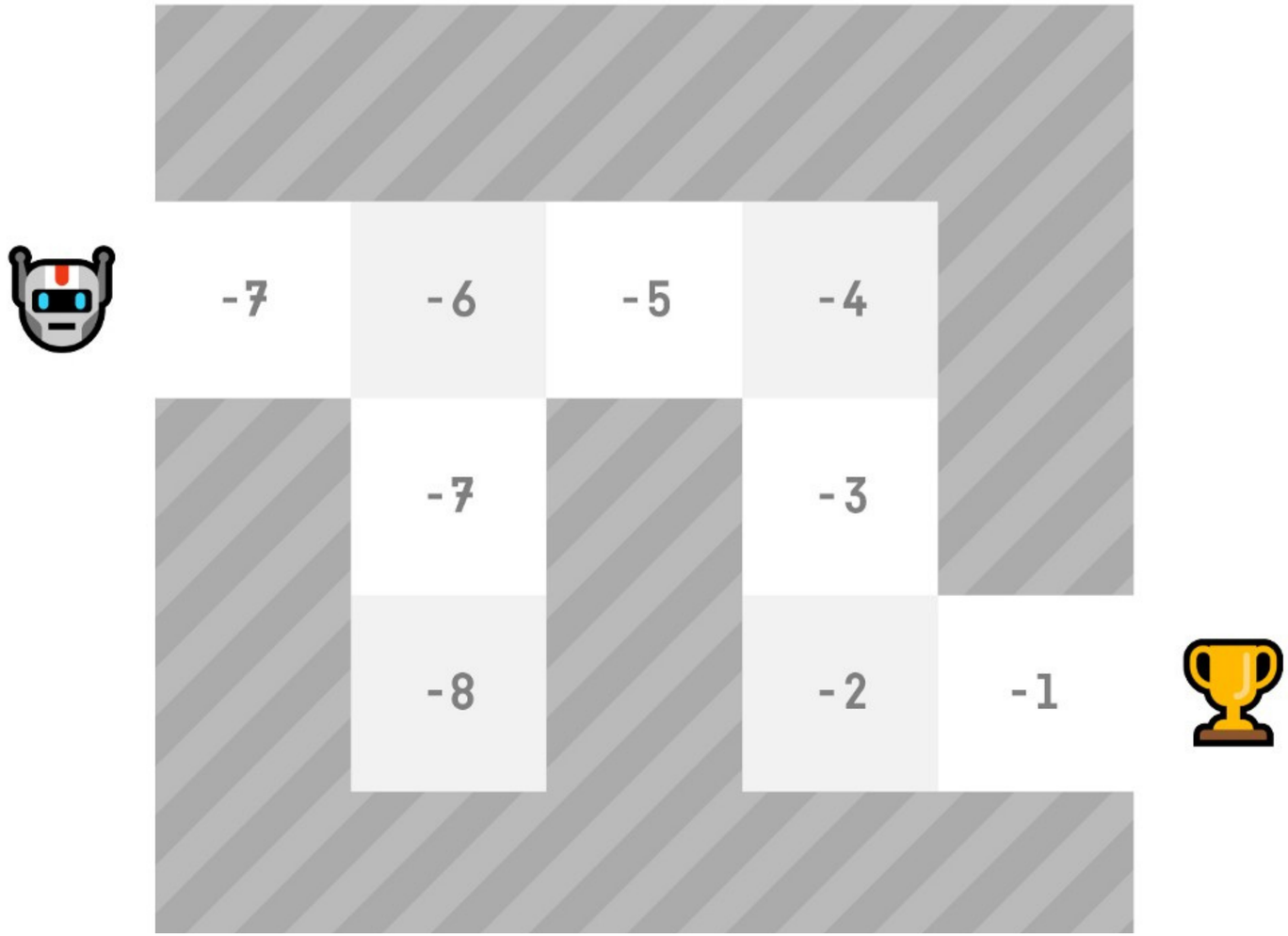
$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s\right]$$

Value function — Expected discounted return — Starting at state s

The statement "Act according to our policy" simply means that our policy involves pursuing the option with the greatest value.

The value function has assigned values to every possible state, demonstrating its usefulness through state values.

- Our policy prioritizes selecting the state with the highest value determined by our value function at each step.
- The objective is achieved by prioritizing states with values like -7, -6, -5, and so on.



Value-Based Methods

The value of a state is the **expected discounted return** the agent can get if it **starts in that state, and then act according to our policy.**

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s\right]$$

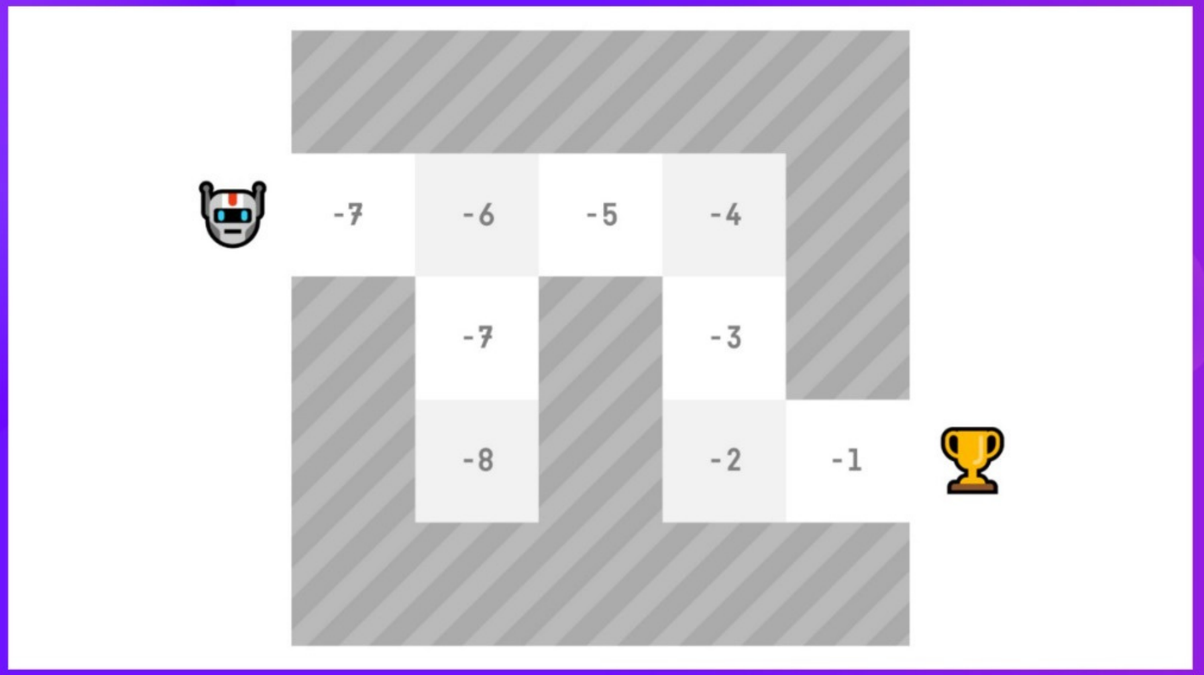
Value function

Expected discounted return

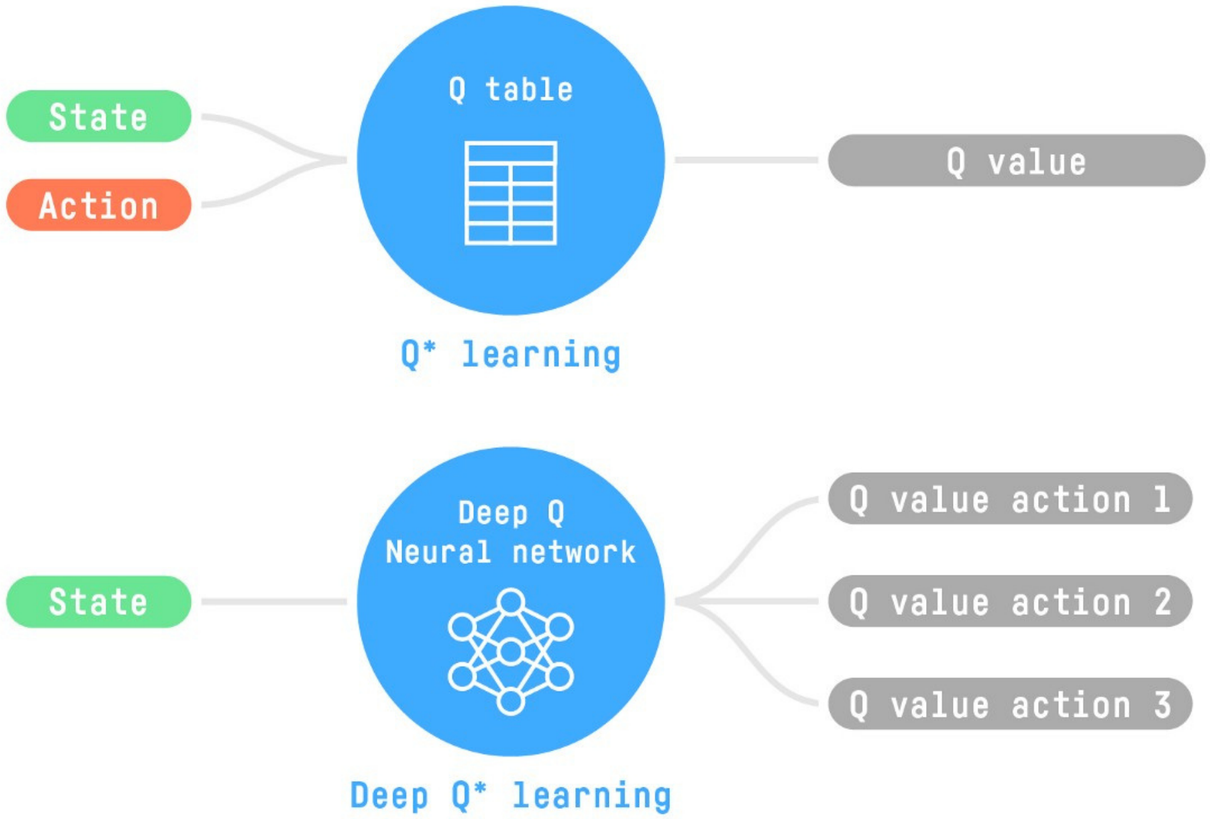
Starting at state s

"Act according to our policy" just means that our policy is **"going to the state with the highest value".**

Value-Based Methods



Thanks to our value function, at each step our policy will select the state with the **biggest value defined by the value function**: -7, then -6, then -5 (and so on) to attain the goal.



The “Deep” in Reinforcement Learning

Deep Reinforcement Learning introduces deep neural networks to solve Reinforcement Learning problems — hence the name “deep”.

For instance, in the next unit, we’ll learn about two value-based algorithms: Q-Learning (classic Reinforcement Learning) and then Deep Q-Learning.

You’ll see the difference is that, in the first approach, we use a traditional algorithm to create a Q table that helps us find what action to take for each state.

In the second approach, we will use a Neural Network (to approximate the Q value).

Summary

- Reinforcement Learning is a computational approach where an agent learns from the environment through trial and error by receiving feedback in the form of rewards or penalties.

- The aim of reinforcement learning (RL) is to maximize the expected cumulative reward, based on the reward hypothesis stating that all goals can be described as maximizing the expected cumulative reward.

- The RL process is a cyclical sequence that produces state, action, reward, and next state as its output.

Summary

- To calculate expected cumulative reward, rewards are discounted since those that occur sooner are more predictable than long-term future rewards.

- To solve a Reinforcement Learning (RL) problem, it's crucial to identify the optimal policy that provides actions resulting in the highest expected return. The policy directs the appropriate action based on the state and serves as the agent's "brain."

There are two ways to find your optimal policy:

By training your policy directly: policy-based methods.

Utilizing a value function that predicts the expected return of an agent at each state to establish our policy is known as value-based methods.

Deep RL uses neural networks to estimate the action to take or the value of a state, hence the name "deep".

Bellman Equation

Dynamic Programming

Bellman Equation

- Reinforcement learning aims to find the optimal policy, which selects the correct action in each state to maximize the agent's return and achieve its goal.
- This class introduces two classic reinforcement learning algorithms: value iteration and policy iteration, used to find the optimal policy.
- Before delving into these algorithms, we will first learn about the Bellman equation, a fundamental concept in reinforcement learning.
- The Bellman equation is crucial for finding the optimal value and Q functions, which are essential for determining the optimal policy in reinforcement learning.

The Bellman equation

- The Bellman equation, named after Richard Bellman, is essential for solving Markov decision processes (MDPs) and finding the optimal policy in reinforcement learning.
- It is widely used to recursively compute the optimal value and Q functions, which can then be used to derive the optimal policy.
- We will explore how the Bellman equation can be used to compute the optimal value and Q functions.

The Bellman equation of the value function

- An Introduction to the Bellman Equation and Value Function
- Here's a quick overview of the Bellman equation and how it relates to the value of a state. The equation suggests that the value of a state can be calculated by combining the immediate reward and the discounted value of the next state. For example, if you take action a in state s and move to state s' to receive reward r, the Bellman equation for the value function is expressed as follows:

$$V(s) = R(s, a, s') + \gamma V(s')$$

- R(s,a,s') represents the immediate reward obtained from performing an action a in state s and transitioning to the next state s'.
- y is the discount factor
- V(s’) implies the value of the next state

Return and discount factor

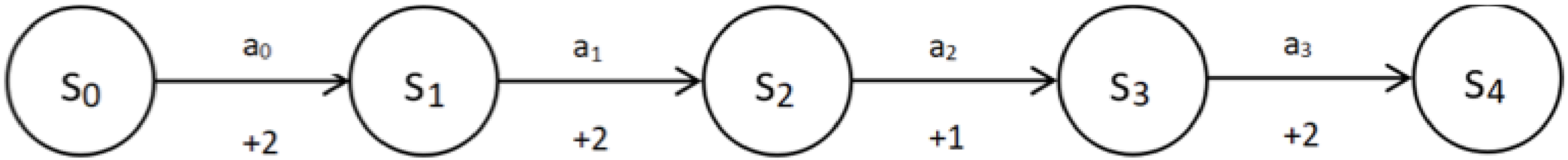
A return can be defined as the sum of the rewards obtained by the agent in an episode. The return is often denoted by *R*. Say the agent starts from the initial state at time step *t* = 0 and reaches the final state at time step *T*, then the return obtained by the agent is given as:

$$R(\tau) = r_0 + r_1 + r_2 + \ldots + r_T$$

$$R(\tau) = \sum_{t=0}^{T} r_t$$

Return and discount factor

Let's understand this with an example; consider the trajectory (episode) τ:



$$R(\tau) = 2 + 2 + 1 + 2 = 7.$$

- The agent's goal in reinforcement learning is to maximize the return, which is the sum of rewards obtained over an episode.
- Maximizing the return requires performing the correct action in each state.
- Achieving the correct actions in each state is facilitated by following the optimal policy.
- The optimal policy is the one that enables the agent to attain the maximum return by selecting the correct actions in each state.

The Bellman equation of the value function

Let us define the return for continuous tasks?

$$R(\tau) = r_0 + r_1 + r_2 + \ldots + r_\infty \qquad R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

How can we maximize the return that just sums to infinity?

how is this discount factor helping us?

Discount Factor

- The discount factor in reinforcement learning prevents the return from reaching infinity by balancing the importance of future rewards and immediate rewards.
- Its value ranges from 0 to 1, where lower values prioritize immediate rewards, and higher values prioritize future rewards.
- A small discount factor (close to 0) emphasizes immediate rewards, while a high discount factor (close to 1) emphasizes future rewards.
- An example with different discount factor values can illustrate how this balance impacts decision-making in reinforcement learning.

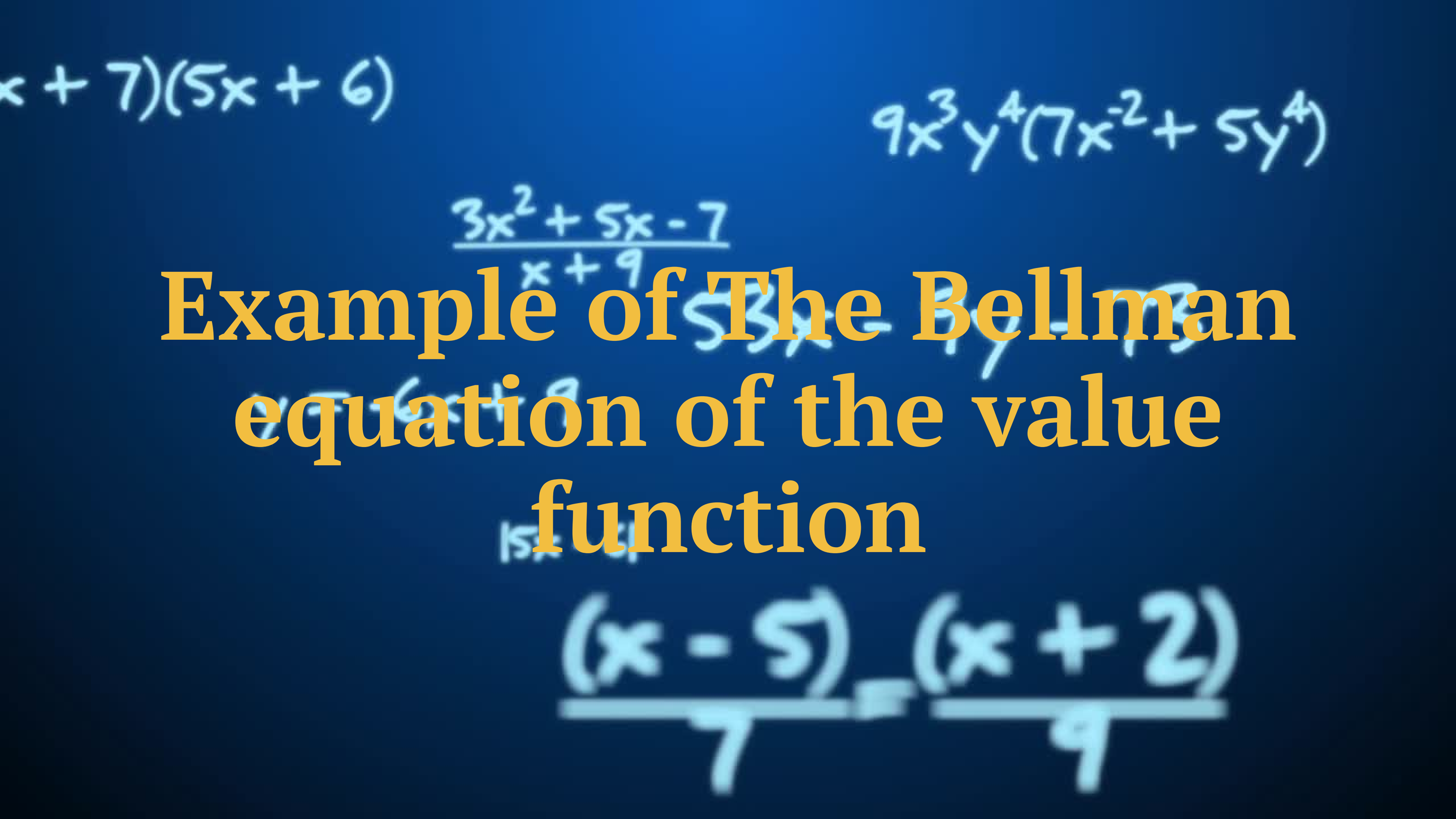
Small vs. large discount factor

Let's set the discount factor to a small value, say 0.2

$$\begin{aligned} R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \ldots \\ &= (0.2)^0 r_0 + (0.2)^1 r_1 + (0.2)^2 r_2 + \ldots \\ &= (1) r_0 + (0.2) r_1 + (0.04) r_2 + \ldots \end{aligned}$$

When we set the discount factor to 0, that is , it implies that we consider only the immediate reward r_0 and not the reward obtained from the future time steps.

$$\begin{aligned} R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \ldots \\ &= (0)^0 r_0 + (0)^1 r_1 + (0)^2 r_2 + \ldots \\ &= (1) r_0 + (0) r_1 + (0) r_2 + \ldots \\ &= r_0 \end{aligned}$$



Example of The Bellman equation of the value function

Let's consider a simple grid world environment with three states (S1, S2, and the goal state G) and two actions (left and right).
The agent receives a reward of -1 for each step taken and a reward of +10 for reaching the goal state. The discount factor γ is set to 0.9.
The Bellman equation for the value functio V(s) in this case can be expressed as:

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right)$$

Example 1:The Bellman equation of the value function

Let's calculate the value of state S1 using the Bellman equation. Assuming the agent is in state S1 and takes action left, it moves to state S2 with a reward of -1. Using the Bellman equation:

V (S1) = max (-1 + 0.9 × V (S2), -1 + 0.9 × V (S2))

Since both actions lead to the same state S2, we can simplify the equation:

V (S1) = -1 + 0.9 × V (S2)

Similarly, for state S2, the agent receives a reward of -1 for each action, and both actions lead to the goal state G with a reward of +10. Using the Bellman equation:

V (S2) = max (-1 + 0.9 × V (G), -1 + 0.9 × V (G))

Again, since both actions lead to the same state G, we simplify the equation:

V (S2) = -1 + 0.9 × V (G)

Finally, for the goal state G, the value is simply the reward:

V(G) = 10

Now, we can substitute the value of V (G) into the equation for V (S2), and then substitute the value of V (S2) into the equation for V (S1) to find the value of V (S1):

V (S2) = -1 + 0.9 × 10 = -1 + 9 = 8

V (S1) = -1 + 0.9 × 8 = -1 + 7.2 = 6.2

Therefore, the value of state S1 is 6.2.

The Bellman equation of the Q function

In the equation, the following applies:
R(s, a, s`) implies the immediate reward obtained while performing an action a in state s and moving to the next state
γ is the discount factor, which determines the importance of future rewards relative to immediate rewards.
Q(s`,a`) is the Q value of the next state-action pair"

- The Bellman equation expresses the Q-value of a state-action pair as the sum of the immediate reward and the discounted maximum Q-value of the next state. This reflects the agent's goal of maximizing cumulative rewards over time.
- By iteratively updating Q-values using the Bellman equation, an agent can learn an optimal policy that maximizes its expected cumulative reward in the long run. The optimal policy selects actions with the highest Q-values for each state.
- The Bellman equation is fundamental in reinforcement learning algorithms, such as Q-learning, which use it to iteratively update Q-values based on observed rewards and transitions. This process allows the agent to learn an optimal policy through trial and error.

$$Q(s,a) = R(s,a,s') + \gamma Q(s',a')$$

EXAMPLE 2: THE BELLMAN EQUATION OF THE Q FUNCTION

ADD MORE TEXT

Consider a simple grid world where an agent can move left, right, up, or down. The grid has a reward of -1 for each step, and the agent receives a reward of +10 for reaching the goal state. The discount factor γ is set to 0.9.

Let's consider a specific example where the agent is in state S and can take actions to move left, right, up, or down. The rewards for each action are as follows:

- **Moving left or right: -1**
- **Moving up or down: -1**

The goal state (state G) has a reward of +10. Since the grid is deterministic, the agent moves to the desired state with probability 1.

We can calculate the Q-values for each state-action pair using the Bellman equation and the given rewards. Let's start with the initial Q-values:

Q(S, left) = 0 Q(S, right) = 0 Q(S, up) = 0 Q(S, down) = 0

To update the Q-values, we apply the Bellman equation for each state-action pair. For example, to update Q(S, left):

Q(S, left) = -1 + 0.9 × max(Q(next state, all actions))

= -1 + 0.9 × max(Q(S, left), Q(S, right), Q(S, up), Q(S, down)) = -1 + 0.9 × max(0, 0, 0, 0)

= -1

Similarly, we can update Q(S,right), Q(S,up), and Q(S,down). After up- dating, the Q-values become:

Q(S, left) = -1 Q(S, right) = -1 Q(S, up) = -1 Q(S, down) = -1

These updated Q-values reflect the expected cumulative rewards the agent can achieve from each state-action pair following an optimal policy in the grid world environment.

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right)$$

THE BELLMAN OPTIMALITY EQUATION

The Bellman optimality equation is a key concept in reinforcement learning that describes the relationship between the optimal value function and the optimal policy in a Markov decision process (MDP). It defines the optimal value of a state as the maximum expected return achievable by following an optimal policy from that state onward.
The Bellman optimality equation for the optimal value function V∗(s) of a state s in an MDP is given by:

γ is the discount factor, which determines the importance of future rewards relative to immediate rewards.

R(s,a) is the immediate reward received after taking action a in state s.
P(s′ | s,a) is the probability of transitioning to state s′ from state s after taking action a.

max denotes the maximum over all possible actions a in state s.

BELLMAN OPTIMALITY EQUATION

- The Bellman optimality equation states that the optimal value of a state is equal to the maximum expected return that can be achieved by taking the best action a in that state and then following the optimal policy thereafter. It provides a recursive way to compute the optimal value function, which represents the maximum expected return from each state under the optimal policy.
- The optimal policy π^* can be derived from the optimal value function V^* by selecting the action a in each state s that maximizes the expression inside the max operator in the Bellman optimality equation. This policy is optimal in the sense that it maximizes the expected cumulative reward over time in the MDP.
- The Bellman optimality equation is fundamental in reinforcement learning because it provides a formal way to compute the optimal value function and derive the optimal policy in MDPs. Algorithms like value iteration and policy iteration use the Bellman optimality equation to iteratively improve the value function and converge to the optimal policy.

Dynamic programming

Dynamic programming (DP) is a problem-solving technique that breaks down complex problems into smaller, simpler sub-problems. For each sub-problem, DP computes and stores the solution instead of solving the problem as a whole. When the same subproblem arises, DP uses the already computed solution instead of re-computing it, which significantly reduces computation time. DP is widely used in computer science, mathematics, bioinformatics, and other fields.

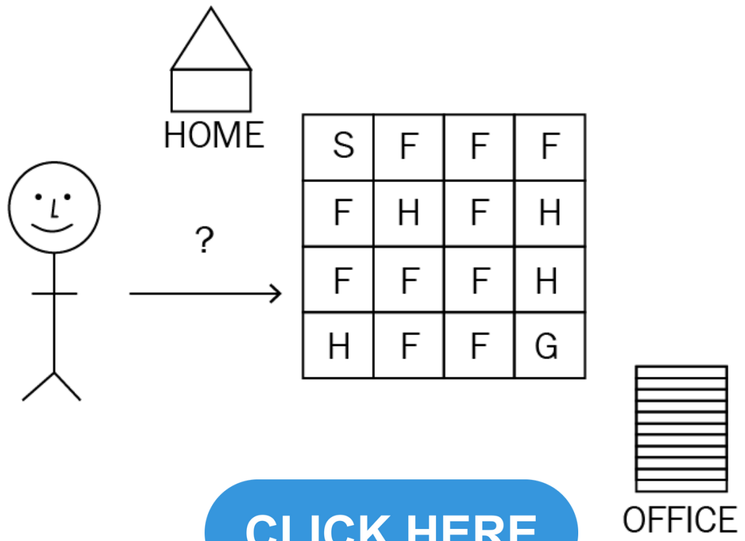
- Value iteration
- Policy iteration

Algorithm 1 Value Iteration

Initialize $V(s) = 0$ for all states s
Initialize $\epsilon > 0$ as the convergence threshold
Initialize Δ as a large value
Initialize iteration counter $k = 0$
while $\Delta > \epsilon$ **do**
    $\Delta \leftarrow 0$
    for each state s **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')\right)$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
    Increment iteration counter: $k \leftarrow k + 1$
    Output current value function V and iteration number k
end while

Value iteration

In order to determine the best course of action for an agent in each state, the value iteration method focuses on identifying the optimal policy. To do so, we first need to calculate the optimal value function, which can then be used to derive the optimal policy.
So how do we calculate the optimal value function?
The answer lies in the optimal Bellman equation of the value function. According to this equation, the optimal value function can be computed as follows:



CLICK HERE

Policy iteration

In the value iteration method, the optimal value function is computed by repeatedly taking the maximum Q function (Q values). This process is used to extract the optimal policy. Conversely, the policy iteration method aims to compute the optimal value function through iterative policy changes. Once the optimal value function is determined, it can be used to extract the optimal policy.

Bellman Equation for Policy Iteration

The Bellman equation for policy iteration is given by:

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^{\pi}(s')] \tag{1}$$

In this equation:

- $V^{\pi}(s)$ represents the value of state s under policy π.
- $\pi(a|s)$ is the probability of taking action a in state s under policy π.
- $p(s', r|s, a)$ is the transition probability from state s to state s' with reward r after taking action a.
- γ is the discount factor.
- $V^{\pi}(s')$ is the value of the next state s' under policy π.

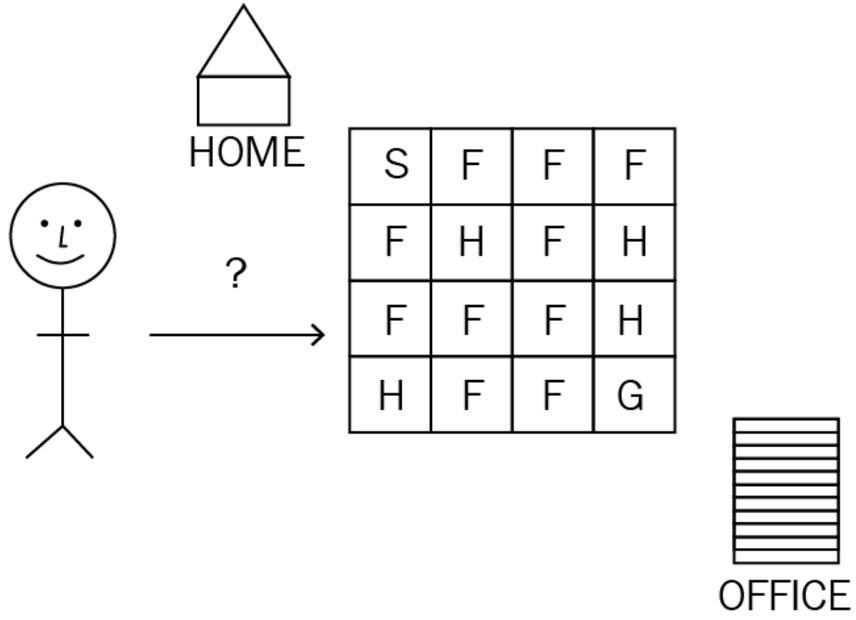
This equation describes how the value of a state under a policy is the sum of the expected immediate reward and the discounted value of the next state, weighted by the transition probabilities and the policy's action probabilities.

Algorithm – policy iteration

Policy Iteration Algorithm

Algorithm 1 Policy Iteration

Initialize a random policy π
repeat
  Policy Evaluation:
  Initialize $V(s) = 0$ for all states s
  repeat
    for all states s **do**
      $V_{\text{new}}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    end for
    $V \leftarrow V_{\text{new}}$
  until convergence
  Policy Improvement:
  Policy-stable $\leftarrow$ **true**
  for all states s **do**
    $old\_action \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    if $old\_action \neq \pi(s)$ **then**
      Policy-stable $\leftarrow$ **false**
    end if
  end for
until Policy-stable



SOLVING FROZEN LAKE PROBLEM USING POLICY ITERATION

Lab 3: Solving a Gym Environment using the Bellman Equation

Thank you!