LAB No 12

Implementation of Reinforcement Learning

In this lab, students will learn how to implement Q-Learning, a model-free Reinforcement Learning (RL) algorithm. The lab involves:

- Understanding the core concepts of RL: agent, environment, states, actions, and rewards.
- Implementing Q-Learning on a simple environment.
- Observing the learning process and how the agent optimizes its actions to maximize cumulative reward.

LAB Objectives:

1. Understand the fundamentals of reinforcement learning.
2. Implement Q-Learning for a discrete environment.
3. Analyze the convergence of the Q-table and optimal policy.
4. Visualize agent performance over episodes.

Theory

1. Reinforcement Learning (RL)

Reinforcement Learning is a type of machine learning where an agent learns to make decisions by interacting with an environment.

- **Agent:** Learner or decision maker.
- **Environment:** Where the agent acts.
- **State (s):** Representation of the environment at a point in time.
- **Action (a):** Possible moves the agent can take.
- **Reward (r):** Feedback from the environment after taking an action.
- **Policy (π):** Strategy that maps states to actions.

- **Goal:** Maximize cumulative reward over time.

2. Q-Learning

Q-Learning is an **off-policy, model-free RL algorithm** used to find the optimal policy.

- **Q-Table**: Stores the expected cumulative reward for each **state-action pair**.
- **Update Rule:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Where:

- α = learning rate ($0 < \alpha \leq 1$)
- γ = discount factor ($0 \leq \gamma \leq 1$)
- r = reward for taking action a in state s
- s' = next state after action a

Algorithm Steps:

1. Initialize Q-table with zeros.
2. For each episode:
    - Observe current state s.
    - Choose an action a using a policy (e.g., ε-greedy).
    - Take action a, observe reward r and next state s'.
    - Update Q(s,a) using the update rule.
    - Repeat until terminal state.

3. Applications of Q-Learning

- Game AI (e.g., Tic-Tac-Toe, Gridworld, Luddo).
- Robot navigation.

Python Libraries Required

```
import numpy as np
import random
import matplotlib.pyplot as plt
```

Solved Examples

Example 1: Q-Learning in a Simple Gridworld

Environment: 4x4 grid, start at top-left (0,0), goal at bottom-right (3,3). Reward = 1 at goal, 0 otherwise.

Solution:

```
# Gridworld parameters
n_states = 16
n_actions = 4  # up, down, left, right
Q = np.zeros((n_states, n_actions))
gamma = 0.9   # discount factor
alpha = 0.1   # learning rate
epsilon = 0.2 # exploration probability

# Helper functions
def step(state, action):
    row, col = divmod(state, 4)
    if action == 0: row = max(row-1, 0)   # up
    if action == 1: row = min(row+1, 3)   # down
    if action == 2: col = max(col-1, 0)   # left
    if action == 3: col = min(col+1, 3)   # right
    next_state = row*4 + col
    reward = 1 if next_state == 15 else 0
    done = next_state == 15
    return next_state, reward, done
```

```
# Q-learning algorithm
episodes = 500
for ep in range(episodes):
   state = 0
   done = False
   while not done:
      if random.uniform(0,1) < epsilon:
         action = np.random.randint(n_actions)
      else:
         action = np.argmax(Q[state])
      next_state, reward, done = step(state, action)
      Q[state, action] = Q[state, action] + alpha * (reward +
gamma*np.max(Q[next_state]) - Q[state, action])
      state = next_state

print("Learned Q-Table:\n", Q)
```

Explanation:

- The agent learns the optimal path to reach the goal.
- Over episodes, Q-values for the correct actions increase, guiding the agent.

Example 2: Q-Learning in FrozenLake (OpenAI Gym)

Environment: FrozenLake-v1 (4x4 grid, slippery surface).

Solution:

```
import gym
import numpy as np
env = gym.make("FrozenLake-v1", is_slippery=False)

# Initialize Q-table
Q = np.zeros((env.observation_space.n, env.action_space.n))
alpha = 0.1
```

```
gamma = 0.99
epsilon = 0.2
episodes = 1000

# Q-learning
for ep in range(episodes):
   state = env.reset()[0]
   done = False
   while not done:
      if np.random.rand() < epsilon:
         action = env.action_space.sample()
      else:
         action = np.argmax(Q[state])
      next_state, reward, done, _, _ = env.step(action)
      Q[state, action] = Q[state, action] + alpha * (reward +
gamma*np.max(Q[next_state]) - Q[state, action])
      state = next_state

# Display Q-table
print("Learned Q-Table:\n", Q)
```

Explanation:

- The agent learns safe paths to reach the goal without falling into holes.
- Q-Table guides action selection to maximize cumulative reward.

Key Points to Remember

1. Q-Learning is **model-free**; no prior knowledge of environment dynamics is required.
2. **Exploration vs Exploitation:** ε-greedy helps balance trying new actions vs. using learned Q-values.

3. **Discount factor γ** determines importance of future rewards.
4. Q-Learning works best for **discrete state-action spaces**.

LAB Exercise Questions

LAB Task 1:

Experiment: CartPole Environment using Gymnasium & Pygame

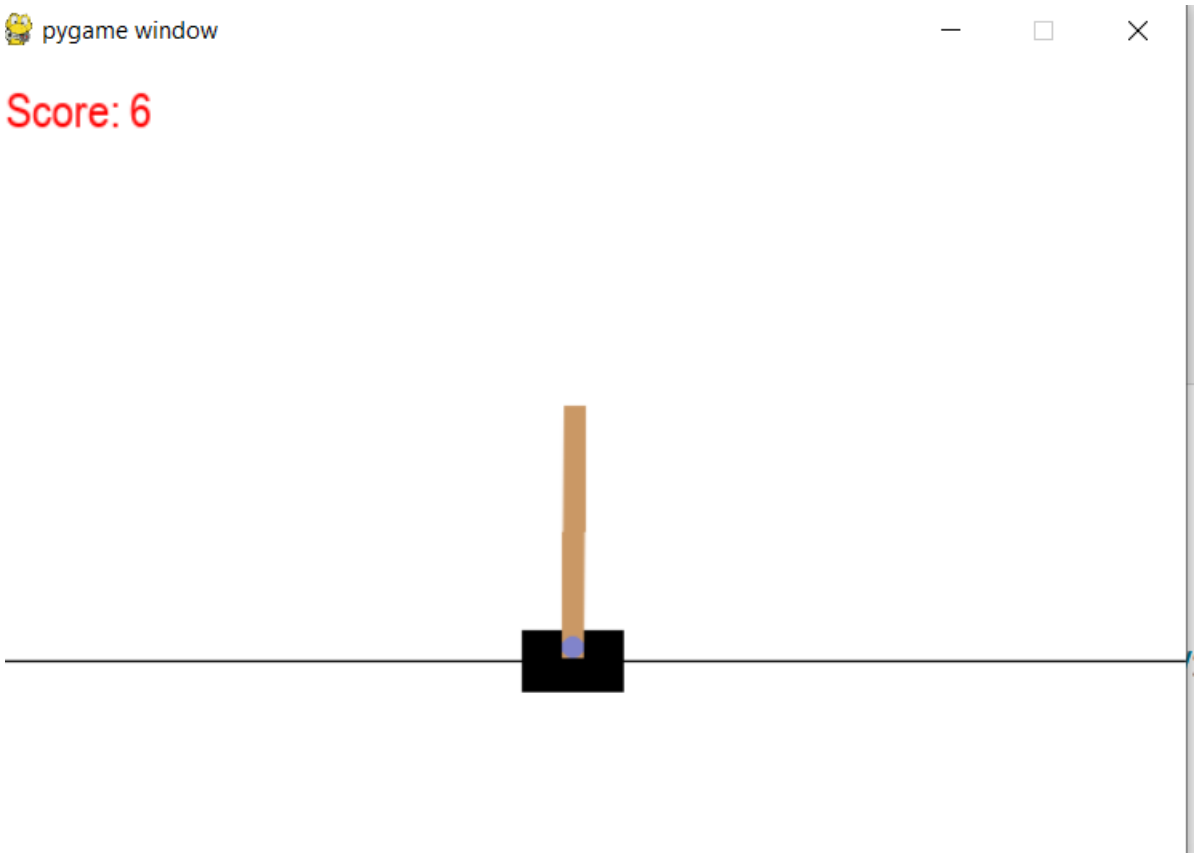
🎯 Lab Objectives

After completing this lab, students will be able to:

- Understand the **Reinforcement Learning interaction loop**
- Use **Gymnasium environments**
- Visualize agent behavior using **Pygame**
- Interpret **states, actions, rewards, and episodes**
- Modify and analyze RL environment parameters

```
import gymnasium as gym
import pygame

env = gym.make("CartPole-v1", render_mode="human")

font = None

for episode in range(1, 20):
    score = 0
    state, info = env.reset()
    done = False

    while not done:
        action = env.action_space.sample()
        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        score += reward

        if font is None:
```

```
            pygame.font.init()
            font = pygame.font.SysFont("Arial", 24)

        surface = pygame.display.get_surface()
        text = font.render(f"Score: {int(score)}", True, (255, 0, 0))
        surface.blit(text, (10, 10))
        pygame.display.update()

    print(f"Episode {episode} Score: {score}")

env.close()
pygame.quit()
```



Lab Questions (Conceptual Understanding)

Q1.

What is **Reinforcement Learning**? Identify the **agent**, **environment**, **state**, **action**, and **reward** in the given code.

Answer:

Reinforcement Learning is a type of machine learning in which an agent interacts with an environment, takes actions, and learns from rewards. The objective is to learn a policy that maximizes the total reward over time.

Identification Table:

Component	Description in Code
Agent	The program that selects actions
Environment	CartPole-v1
State	Cart position, cart velocity, pole angle, pole velocity
Action	Move cart left or right
Reward	+1 for each successful step

Q2.

Explain the purpose of the following line:

env = gym.make("CartPole-v1", render_mode="human")

Answer:

This line creates the CartPole environment using Gym. The `render_mode="human"` option enables visual rendering so the environment is displayed in a window where the cart and pole movement can be observed.

Q3.

What does env.reset() return? Why are two values returned?

Answer:

`env.reset()` returns two values: the initial observation and additional environment information. The observation represents the starting state, while the second value provides extra details related to the environment setup.

Q4.

Explain the difference between:

terminated

truncated

Answer:

Aspect	terminated	truncated
Reason for ending	Episode ends because the environment's success or failure condition is met	Episode ends because an external limit such as time or steps is reached
Task status	Task logically finishes	Task does not logically finish
Agent performance	Indicates correct or incorrect performance	Does not reflect agent performance
Environment rule	Defined by environment dynamics	Defined by imposed constraints
Learning impact	Helps agent understand success or failure	Mainly limits episode length

Q5.

What is the role of the variable score? How is it calculated?

Answer:

The variable `score` stores the total reward obtained in one episode. Each time the agent takes a valid step, the reward is added to the score. A higher score indicates better performance.

Q6.

Why is action = env.action_space.sample() used?
Is this an intelligent agent? Justify your answer.

- It selects a random action from the action space
- It allows basic exploration of the environment
- No learning or decision-making logic is used

Q7.

Explain how **Pygame** is used to display the score on the screen.

Answer:

Pygame is used to create a font object that renders the score as text. This text is then drawn onto the game window at a fixed position, and the screen is refreshed continuously to show the updated score.

Q8.

What happens if the pygame.display.update() line is removed?

Answer:

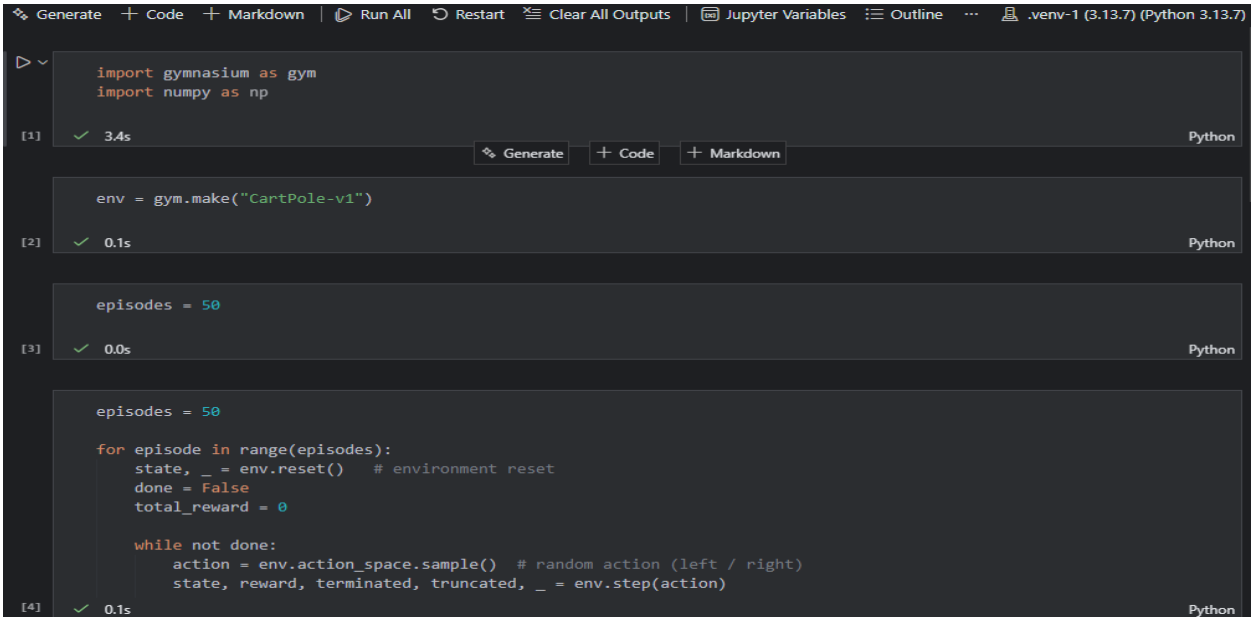
If `pygame.display.update()` is removed, changes made to the screen will not appear. The score and graphics may update internally, but nothing new will be visible to the user.

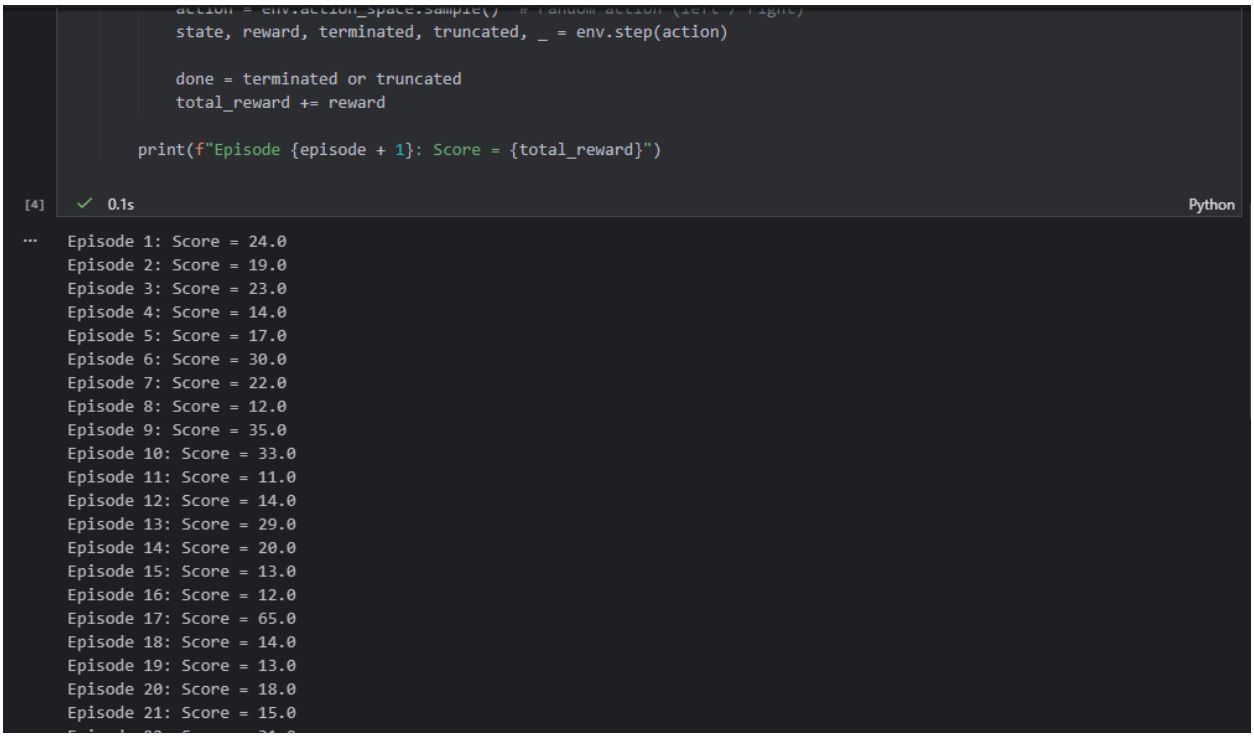
Lab Tasks

Task 1: Modify Number of Episodes

Change the number of episodes from **20 to 50** and observe:

- How the score varies across episodes
- Whether performance improves or remains random


Generate
Code
Markdown
Run All
Restart
Clear All Outputs
Jupyter Variables
Outline
.venv-1 (3.13.7) (Python 3.13.7)
import gymnasium as gym
import numpy as np
[1]
3.4s
Python
Generate
Code
Markdown
env = gym.make("CartPole-v1")
[2]
0.1s
Python
episodes = 50
[3]
0.0s
Python
episodes = 50
for episode in range(episodes):
state, _ = env.reset() # environment reset
done = False
total_reward = 0
while not done:
action = env.action_space.sample() # random action (left / right)
state, reward, terminated, truncated, _ = env.step(action)
[4]
0.1s
Python


state, reward, terminated, truncated, _ = env.step(action)
done = terminated or truncated
total_reward += reward
print(f"Episode {episode + 1}: Score = {total_reward}")
[4]
0.1s
Python
Episode 1: Score = 24.0
Episode 2: Score = 19.0
Episode 3: Score = 23.0
Episode 4: Score = 14.0
Episode 5: Score = 17.0
Episode 6: Score = 30.0
Episode 7: Score = 22.0
Episode 8: Score = 12.0
Episode 9: Score = 35.0
Episode 10: Score = 33.0
Episode 11: Score = 11.0
Episode 12: Score = 14.0
Episode 13: Score = 29.0
Episode 14: Score = 20.0
Episode 15: Score = 13.0
Episode 16: Score = 12.0
Episode 17: Score = 65.0
Episode 18: Score = 14.0
Episode 19: Score = 13.0
Episode 20: Score = 18.0
Episode 21: Score = 15.0

```
Episode 11: Score = 11.0
Episode 12: Score = 14.0
Episode 13: Score = 29.0
Episode 14: Score = 20.0
Episode 15: Score = 13.0
Episode 16: Score = 12.0
Episode 17: Score = 65.0
Episode 18: Score = 14.0
Episode 19: Score = 13.0
Episode 20: Score = 18.0
Episode 21: Score = 15.0
Episode 22: Score = 31.0
Episode 23: Score = 37.0
Episode 24: Score = 17.0
Episode 25: Score = 55.0
...
Episode 47: Score = 37.0
Episode 48: Score = 13.0
Episode 49: Score = 13.0
Episode 50: Score = 12.0
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
env.close()
```

[5] ✓ 0.0s Python

Task 2: Display Episode Number on Screen

Modify the code to show:

Episode: X | Score: Y

on the CartPole window.

```
import gymnasium as gym
import pygame
import numpy as np
```

✓ 2.4s Python

```
env = gym.make("CartPole-v1", render_mode="human")
```

✓ 0.0s Python

```
pygame.init()
font = pygame.font.SysFont("Arial", 24)
```

✓ 0.7s Python

```
episodes = 50
```

✓ 0.0s Python

```
for episode in range(episodes):
    state, _ = env.reset()
```

✓ 0.8s Python

```python
    for episode in range(episodes):
        state, _ = env.reset()
        done = False
        total_reward = 0

        while not done:
            action = env.action_space.sample()
            state, reward, terminated, truncated, _ = env.step(action)
            done = terminated or truncated
            total_reward += reward

        print(f"Episode: {episode + 1} | Score: {total_reward}")
```

[1] ✓ 0.8s Python

```
Episode: 1 | Score: 22.0
Episode: 2 | Score: 13.0
Episode: 3 | Score: 18.0
Episode: 4 | Score: 16.0
Episode: 5 | Score: 13.0
Episode: 6 | Score: 24.0
Episode: 7 | Score: 13.0
Episode: 8 | Score: 22.0
Episode: 9 | Score: 19.0
Episode: 10 | Score: 25.0
Episode: 11 | Score: 14.0
Episode: 12 | Score: 13.0
Episode: 13 | Score: 19.0
Episode: 14 | Score: 23.0
Episode: 15 | Score: 20.0
Episode: 16 | Score: 23.0
```

```
Episode: 2 | Score: 13.0
Episode: 3 | Score: 18.0
Episode: 4 | Score: 16.0
Episode: 5 | Score: 13.0
Episode: 6 | Score: 24.0
Episode: 7 | Score: 13.0
Episode: 8 | Score: 22.0
Episode: 9 | Score: 19.0
Episode: 10 | Score: 25.0
Episode: 11 | Score: 14.0
Episode: 12 | Score: 13.0
Episode: 13 | Score: 19.0
Episode: 14 | Score: 23.0
Episode: 15 | Score: 20.0
Episode: 16 | Score: 23.0
Episode: 17 | Score: 24.0
Episode: 18 | Score: 12.0
Episode: 19 | Score: 16.0
Episode: 20 | Score: 10.0
Episode: 21 | Score: 14.0
Episode: 22 | Score: 12.0
Episode: 23 | Score: 19.0
Episode: 24 | Score: 11.0
Episode: 25 | Score: 16.0
...
Episode: 47 | Score: 31.0
Episode: 48 | Score: 34.0
Episode: 49 | Score: 13.0
Episode: 50 | Score: 31.0
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Task 3: Change Text Color and Position

- Change score text color from **red to green**
- Display it at position **(200, 20)**

cartpole_score.py > ...

```python
import pygame
import sys

# Pygame initialize
pygame.init()

# Screen size
screen = pygame.display.set_mode((600, 400))
pygame.display.set_caption("Score Display Example")

# Colors
green = (0, 255, 0)
black = (0, 0, 0)

# Font
font = pygame.font.Font(None, 36)  # None = default font, 36 = font size

# Score
score = 10

# Main loop
running = True
while running:
    screen.fill(black)  # Screen ko black se fill karo

    # Score text create karo
    score_text = font.render(f"Score: {score}", True, green)

    # Score ko screen par draw karo at (200, 20)
    screen.blit(score_text, (200, 20))
```

```python
# Font
font = pygame.font.Font(None, 36)  # None = default font, 36 = font size

# Score
score = 10

# Main loop
running = True
while running:
    screen.fill(black)  # Screen ko black se fill karo

    # Score text create karo
    score_text = font.render(f"Score: {score}", True, green)

    # Score ko screen par draw karo at (200, 20)
    screen.blit(score_text, (200, 20))

    # Event handling
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    pygame.display.flip()  # Screen update karo

pygame.quit()
sys.exit()
```

Task 4: Print Maximum Score

After all episodes finish:

- Store all episode scores
- Print the **maximum score achieved**

```python
import gym
env = gym.make("CartPole-v1")
```

✓ 0.2s Python

```
Gym has been unmaintained since 2022 and does not support NumPy 2.0 amongst other critical functionality.
Please upgrade to Gymnasium, the maintained drop-in replacement of Gym, or contact the authors of your software and request
Users of this version of Gym should be able to simply replace 'import gym' with 'import gymnasium as gym' in the vast major
See the migration guide at https://gymnasium.farama.org/introduction/migration_guide/ for additional information.
```

```python
episodes = 50
all_scores = []
```

✓ 0.0s Python

```python
for episode in range(episodes):
    state, info = env.reset()  # Gym 0.26+ me reset() 2 values return karta hai
    done = False
    score = 0

    while not done:
        action = env.action_space.sample()
        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        score += reward
```

✓ 0.0s Python

```python
        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        score += reward

    all_scores.append(score)
    print(f"Episode: {episode+1} | Score: {score}")
```

✓ 0.0s Pytho

```
Episode: 1 | Score: 17.0
Episode: 2 | Score: 17.0
Episode: 3 | Score: 37.0
Episode: 4 | Score: 22.0
Episode: 5 | Score: 18.0
Episode: 6 | Score: 20.0
Episode: 7 | Score: 19.0
Episode: 8 | Score: 25.0
Episode: 9 | Score: 59.0
Episode: 10 | Score: 29.0
Episode: 11 | Score: 25.0
Episode: 12 | Score: 20.0
Episode: 13 | Score: 23.0
Episode: 14 | Score: 10.0
Episode: 15 | Score: 11.0
Episode: 16 | Score: 24.0
Episode: 17 | Score: 22.0
Episode: 18 | Score: 37.0
Episode: 19 | Score: 31.0
Episode: 20 | Score: 21.0
Episode: 21 | Score: 14.0
Episode: 22 | Score: 21.0
Episode: 23 | Score: 32.0
```

```
Episode: 17 | Score: 22.0
Episode: 18 | Score: 37.0
Episode: 19 | Score: 31.0
Episode: 20 | Score: 21.0
Episode: 21 | Score: 14.0
Episode: 22 | Score: 21.0
Episode: 23 | Score: 32.0
Episode: 24 | Score: 10.0
Episode: 25 | Score: 35.0
...
Episode: 47 | Score: 28.0
Episode: 48 | Score: 21.0
Episode: 49 | Score: 10.0
Episode: 50 | Score: 15.0
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
max_score = max(all_scores)
max_episode = all_scores.index(max_score) + 1

print("\nMaximum Score Achieved:", max_score)
print("Achieved in Episode:", max_episode)
```

[14] ✓ 0.0s Python

```
Maximum Score Achieved: 60.0
Achieved in Episode: 30
```

Task 5: Slow Down the Environment

Insert a small delay using:

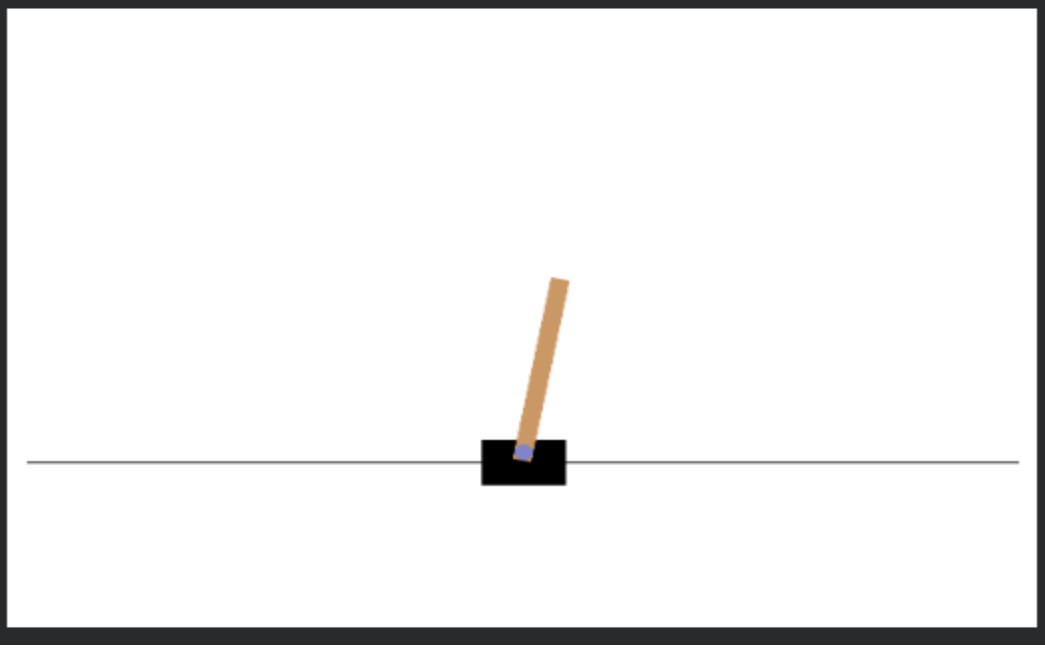
pygame.time.delay(20)

Observe the effect on visualization.

```
# Install gymnasium and matplotlib
!pip install gymnasium matplotlib

import gymnasium as gym
import matplotlib.pyplot as plt
from IPython import display as ipythondisplay
import time

# Create CartPole environment
env = gym.make("CartPole-v1", render_mode='rgb_array')

episodes = 3

for episode in range(episodes):
    observation, info = env.reset()
    done = False
    score = 0

    while not done:
        # Render frame
        img = env.render()
        plt.imshow(img)
        plt.axis('off')
        ipythondisplay.clear_output(wait=True)
        ipythondisplay.display(plt.gcf())
```

```
        done = False
        score = 0

        while not done:
            # Render frame
            img = env.render()
            plt.imshow(img)
            plt.axis('off')
            ipythondisplay.clear_output(wait=True)
            ipythondisplay.display(plt.gcf())

            # Random action
            action = env.action_space.sample()
            observation, reward, terminated, truncated, info = env.step(action)
            done = terminated or truncated
            score += reward

            # Small delay ~ simulate pygame.time.delay(20)
            time.sleep(0.01)

        print(f"Episode {episode+1} finished with score: {score}")

    env.close()
    ipythondisplay.clear_output(wait=True)
```

```
env.close()
ipythondisplay.clear_output(wait=True)
```



Task 6: Replace CartPole with MountainCar

Change the environment to:

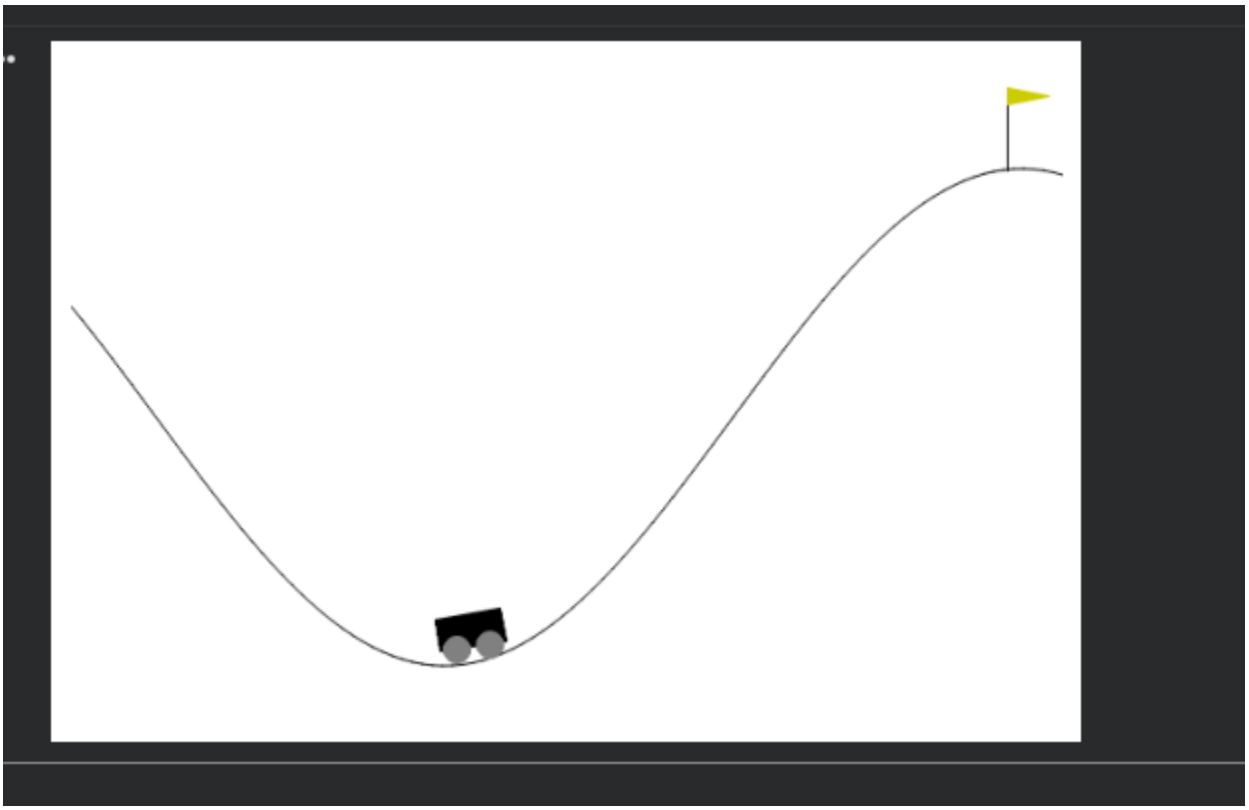
env = gym.make("MountainCar-v0", render_mode="human")

Compare:

- Reward behavior
- Episode termination condition

```
# Install required libraries
!pip install gymnasium matplotlib

import gymnasium as gym
import matplotlib.pyplot as plt
from IPython import display as ipythondisplay
import time

# Create MountainCar environment
env = gym.make("MountainCar-v0", render_mode="rgb_array")

episodes = 2

for episode in range(episodes):
    observation, info = env.reset()
    done = False
    total_reward = 0

    while not done:
        # Render frame
        img = env.render()
        plt.imshow(img)
        plt.axis("off")
```

```
    while not done:
        # Render frame
        img = env.render()
        plt.imshow(img)
        plt.axis("off")
        ipythondisplay.clear_output(wait=True)
        ipythondisplay.display(plt.gcf())

        # Random action (0 = left, 1 = no push, 2 = right)
        action = env.action_space.sample()

        observation, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        total_reward += reward

        # Small delay (speed control)
        time.sleep(0.02)

    print(f"Episode {episode+1} finished with total reward: {total_reward}")

env.close()
ipythondisplay.clear_output(wait=True)
```



Task 7: Identify State Variables

Print the state vector and answer:

- How many state variables are there?
- What does each variable represent?

```python
import gymnasium as gym

# Create MountainCar environment
env = gym.make("MountainCar-v0")

# Reset environment
state, info = env.reset()

print("State vector:", state)
print("Number of state variables:", len(state))

env.close()
```

```
State vector: [-0.5949445  0.       ]
Number of state variables: 2
```

Task 8 (Advanced): Rule-Based Action

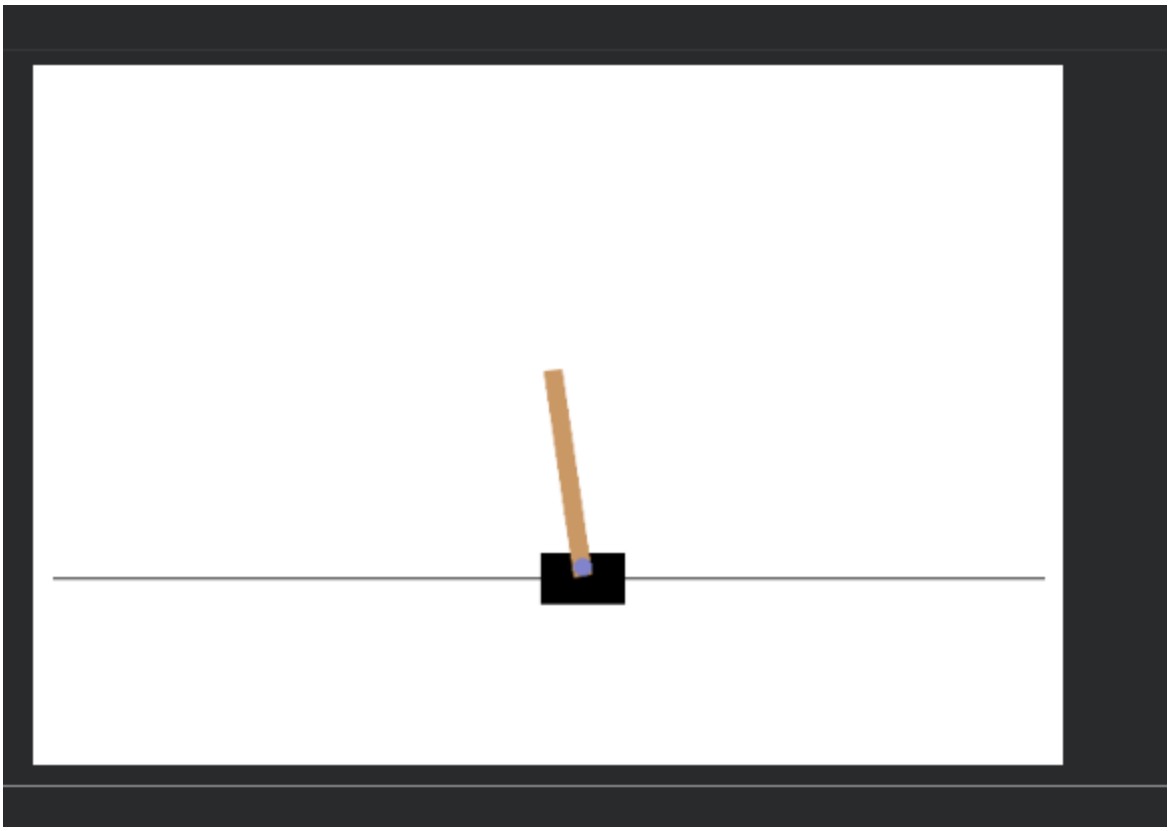
Replace random action with:

```
if state[2] > 0:
    action = 1
else:
    action = 0
```

random actions were replaced with a rule-based policy. If the pole angle is positive, the cart moves right; otherwise, it moves left. This rule helps in balancing the pole better than random actions and increases the episode score

```
# Install libraries
!pip install gymnasium matplotlib

import gymnasium as gym
import matplotlib.pyplot as plt
from IPython import display as ipythondisplay
import time

# Create CartPole environment
env = gym.make("CartPole-v1", render_mode="rgb_array")

episodes = 5   # observation table ke liye

for episode in range(episodes):
    state, info = env.reset()
    done = False
    score = 0

    while not done:
        # Render
        img = env.render()
        plt.imshow(img)
        plt.axis("off")
```

```
    while not done:
        # Render
        img = env.render()
        plt.imshow(img)
        plt.axis("off")
        ipythondisplay.clear_output(wait=True)
        ipythondisplay.display(plt.gcf())

        # 🔹 Rule-Based Action (as given by sir)
        if state[2] > 0:
            action = 1   # move right
        else:
            action = 0   # move left

        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        score += reward

        time.sleep(0.02)

    print(f"Episode {episode+1} | Score: {score}")

env.close()
ipythondisplay.clear_output(wait=True)
```



📊 Observation Table (For Students)

Episode	Score	Remarks
1		
2		
...		
20		

Observation Table For Students:

```python
# Install required packages
!pip install gymnasium pandas

import gymnasium as gym
import pandas as pd

# Create CartPole environment
env = gym.make("CartPole-v1", render_mode="rgb_array")  # Colab-safe rendering

num_episodes = 20  # Number of episodes for observation
data = []

for episode in range(1, num_episodes + 1):
    state, _ = env.reset(seed=42+episode)
    done = False
    total_reward = 0
    step_count = 0

    while not done:
        # Simple random policy for now (you can modify later)
        action = env.action_space.sample()
        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        total_reward += reward
        step_count += 1

    # Check if goal reached: max 500 steps = success
    goal_reached = "Yes" if step_count >= 500 else "No"

    # Append episode data
    data.append([episode, step_count, total_reward, goal_reached])
```

```python
    while not done:
        # Simple random policy for now (you can modify later)
        action = env.action_space.sample()
        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        total_reward += reward
        step_count += 1

    # Check if goal reached: max 500 steps = success
    goal_reached = "Yes" if step_count >= 500 else "No"

    # Append episode data
    data.append([episode, step_count, total_reward, goal_reached])

env.close()

# Create DataFrame (Observation Table)
df = pd.DataFrame(data, columns=["Episode", "Steps", "Score", "Goal Reached (Yes/No)"])
df
```

```
Requirement already satisfied: gymnasium in /usr/local/lib/python3.12/dist-packages (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

1 to 20 of 20 entries Filter

index	Episode	Steps	Score	Goal Reached (Yes/No)
0	1	27	27.0	No
1	2	18	18.0	No
2	3	12	12.0	No
3	4	18	18.0	No
4	5	14	14.0	No
5	6	10	10.0	No
6	7	14	14.0	No
7	8	21	21.0	No
8	9	27	27.0	No
9	10	36	36.0	No
10	11	15	15.0	No
11	12	10	10.0	No
12	13	20	20.0	No
13	14	15	15.0	No
14	15	59	59.0	No
15	16	23	23.0	No
16	17	29	29.0	No
17	18	13	13.0	No
18	19	11	11.0	No
19	20	38	38.0	No

Experiment: MountainCar Environment using Gymnasium & Pygame

Lab Objectives

After completing this lab, students will be able to:

- Understand the **working of a continuous control RL environment**
- Analyze **delayed reward problems**
- Use **Gymnasium MountainCar-v0**
- Visualize agent behavior and rewards using **Pygame**
- Compare MountainCar with CartPole environment

Provided Code:

```
import gymnasium as gym
import pygame

env = gym.make("MountainCar-v0", render_mode="human")

font = None
```

```
for episode in range(1, 20):
    state, info = env.reset()
    done = False
    score = 0

    while not done:
        action = env.action_space.sample()
        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        score += reward

        if font is None:
            pygame.font.init()
            font = pygame.font.SysFont("Arial", 24)

        surface = pygame.display.get_surface()
        text = font.render(f"Episode: {episode}  Score: {int(score)}", True, (255, 0, 0))
        surface.blit(text, (10, 10))
        pygame.display.update()

    print(f"Episode {episode} Score: {score}")

env.close()
pygame.quit()
```

Lab Questions (Conceptual Understanding)

Q1.

What is **Reinforcement Learning**? Identify the **agent**, **environment**, **state**, **action**, and **reward** in the MountainCar code.

Answer:

Reinforcement Learning

Reinforcement Learning is a learning approach in which an agent learns by interacting with an environment. The agent takes actions, observes the resulting state, and receives rewards. The objective is to learn a strategy that maximizes cumulative reward over time.

Identification Table

Component	Description in MountainCar
Agent	The program controlling the car
Environment	MountainCar-v0
State	Car position and velocity
Action	Push left, no push, push right
Reward	−1 at every time step

Q2.

Explain the purpose of the following statement:

env = gym.make("MountainCar-v0", render_mode="human")

Answer:

This statement creates the MountainCar environment using the Gym library. The `render_mode="human"` option enables graphical visualization, allowing the user to observe the car moving between the hills on the screen.

Q3.

What are the **state variables** in MountainCar-v0? What does each state represent?

Answer:

In the MountainCar-v0 environment, the state describes the current situation of the car at any given moment. The agent uses this state information to decide which action to take next. The state consists of two continuous variables that together explain where the car is and how it is moving on the track.

State Variables Table

State Variable	What it Represents
Position	The horizontal location of the car between the two mountains
Velocity	The speed and direction at which the car is moving

Q4.

Describe the **action space** of MountainCar-v0. How many actions are available and what do they mean?

Answer:

The action space in MountainCar-v0 defines all the possible moves the agent can make to control the car. The agent can apply force in different directions or choose not to apply any force. These actions influence the car's movement and help it build momentum to eventually reach the goal at the top of the mountain.

Action Space Table

Action Value	Meaning
0	Apply force to move the car toward the left side
1	Do not apply any force and let the car move naturally
2	Apply force to move the car toward the right side

Total actions available: 3

These three actions give the agent complete control over the car's motion, allowing it to switch directions strategically to gain enough momentum to solve the task.

Q5.

Explain the reward mechanism in MountainCar-v0.
Why does the agent receive a **negative reward** at each step?

Answer:

In MountainCar-v0, the agent receives a reward of −1 at every step until it reaches the goal. This negative reward encourages the agent to solve the problem in the minimum number of steps. The environment is designed this way to penalize slow or inefficient behavior.

Q6.

What is the difference between:

terminated

truncated

in this environment?

Answer:

Aspect	**terminated**	**truncated**
Ending reason	Car reaches the goal position	Maximum step limit is reached
Goal achieved	Yes	No
Indicates success	Yes	No
Based on environment logic	Yes	No
Learning signal	Shows task completion	Shows time constraint

Q7.

Why does the agent fail to reach the goal when using action_space.sample()?

Answer:

When using `action_space.sample()`, the agent selects actions randomly. MountainCar requires a specific strategy involving momentum buildup. Random actions fail to create sufficient momentum, so the car cannot reach the goal at the top of the hill.

Q8.

Explain the role of **momentum** in solving the MountainCar problem.

Answer:

Momentum is essential because the car's engine is not powerful enough to climb the hill directly. The agent must move back and forth to build velocity. This accumulated momentum allows the car to eventually reach the goal position at the top of the hill.

Lab Tasks (Hands-on Practice)

Task 1: Increase Episodes

Modify the code to run **50 episodes** instead of 20.
Observe the score trend across episodes.

```python
# Install required libraries
!pip install gymnasium pygame

import gymnasium as gym
import numpy as np

# Create MountainCar environment
env = gym.make("MountainCar-v0")

episodes = 50   # 🔴 Changed from 20 to 50

episode_rewards = []

for episode in range(episodes):
    state, _ = env.reset()
    done = False
    total_reward = 0

    while not done:
        action = env.action_space.sample()  # Random action
        state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated
        total_reward += reward

    episode_rewards.append(total_reward)
    print(f"Episode {episode+1}/{episodes} → Score: {total_reward}")

env.close()

# Show reward trend
print("\nReward Trend:")
print(episode_rewards)
```

```
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
Episode 1/50 → Score: -200.0
Episode 2/50 → Score: -200.0
Episode 3/50 → Score: -200.0
Episode 4/50 → Score: -200.0
Episode 5/50 → Score: -200.0
Episode 6/50 → Score: -200.0
Episode 7/50 → Score: -200.0
Episode 8/50 → Score: -200.0
Episode 9/50 → Score: -200.0
Episode 10/50 → Score: -200.0
Episode 11/50 → Score: -200.0
Episode 12/50 → Score: -200.0
Episode 13/50 → Score: -200.0
Episode 14/50 → Score: -200.0
Episode 15/50 → Score: -200.0
Episode 16/50 → Score: -200.0
Episode 17/50 → Score: -200.0
Episode 18/50 → Score: -200.0
Episode 19/50 → Score: -200.0
Episode 20/50 → Score: -200.0
Episode 21/50 → Score: -200.0
Episode 22/50 → Score: -200.0
Episode 23/50 → Score: -200.0
Episode 24/50 → Score: -200.0
Episode 25/50 → Score: -200.0
Episode 26/50 → Score: -200.0
Episode 27/50 → Score: -200.0
Episode 28/50 → Score: -200.0
Episode 29/50 → Score: -200.0
Episode 30/50 → Score: -200.0
Episode 31/50 → Score: -200.0
Episode 32/50 → Score: -200.0
```

Task 2: Display State Values

Print the state array in each step and identify:

- Car position
- Car velocity

```
# Install required libraries
!pip install gymnasium pygame

import gymnasium as gym

# Create MountainCar environment
env = gym.make("MountainCar-v0")

episodes = 1   # Sirf 1 episode for clear observation

state, _ = env.reset()
done = False
step = 0

print("State Format: [Car Position, Car Velocity]\n")

while not done:
    action = env.action_space.sample()  # Random action
    state, reward, terminated, truncated, _ = env.step(action)
    done = terminated or truncated

    car_position = state[0]
    car_velocity = state[1]

    print(f"Step {step}")
    print(f"State Array  : {state}")
    print(f"Car Position : {car_position}")
    print(f"Car Velocity : {car_velocity}")
    print("-" * 40)

    step += 1

env.close()
```

```
Step 0
State Array  : [-0.45091537 -0.00155218]
Car Position : -0.4509153664112091
Car Velocity : -0.0015521758468821645
----------------------------------------
Step 1
State Array  : [-0.45300835 -0.00209299]
Car Position : -0.45300835371017456
Car Velocity : -0.002092991955578327
----------------------------------------
Step 2
State Array  : [-0.45662683 -0.00361847]
Car Position : -0.456626832485199
Car Velocity : -0.0036184717901051044
----------------------------------------
Step 3
State Array  : [-0.46074423 -0.00411739]
Car Position : -0.4607442319393158
Car Velocity : -0.004117388743907213
```

```
Step 4
State Array  : [-0.46633023 -0.00558601]
Car Position : -0.46633023023605347
Car Velocity : -0.005586009472608566
----------------------------------------
Step 5
State Array  : [-0.47234365 -0.00601341]
Car Position : -0.47234365344047546
Car Velocity : -0.006013413425534964
----------------------------------------
Step 6
State Array  : [-0.47973996 -0.00739631]
Car Position : -0.4797399640083313
Car Velocity : -0.007396313827484846
----------------------------------------
Step 7
State Array  : [-0.48746425 -0.00772431]
Car Position : -0.4874642491340637
Car Velocity : -0.007724307011812925
```

```
----------------------------------------
Step 8
State Array  : [-0.49645904 -0.00899479]
Car Position : -0.496459037065506
Car Velocity : -0.008994785137474537
----------------------------------------
Step 9
State Array  : [-0.5066572  -0.01019811]
Car Position : -0.5066571831703186
Car Velocity : -0.010198108851909637
----------------------------------------
Step 10
State Array  : [-0.51798224 -0.01132512]
Car Position : -0.5179822444915771
Car Velocity : -0.011325116269290447
----------------------------------------
Step 11
State Array  : [-0.5283495  -0.01036724]
Car Position : -0.5283495187759399
Car Velocity : -0.010367237962782383
```

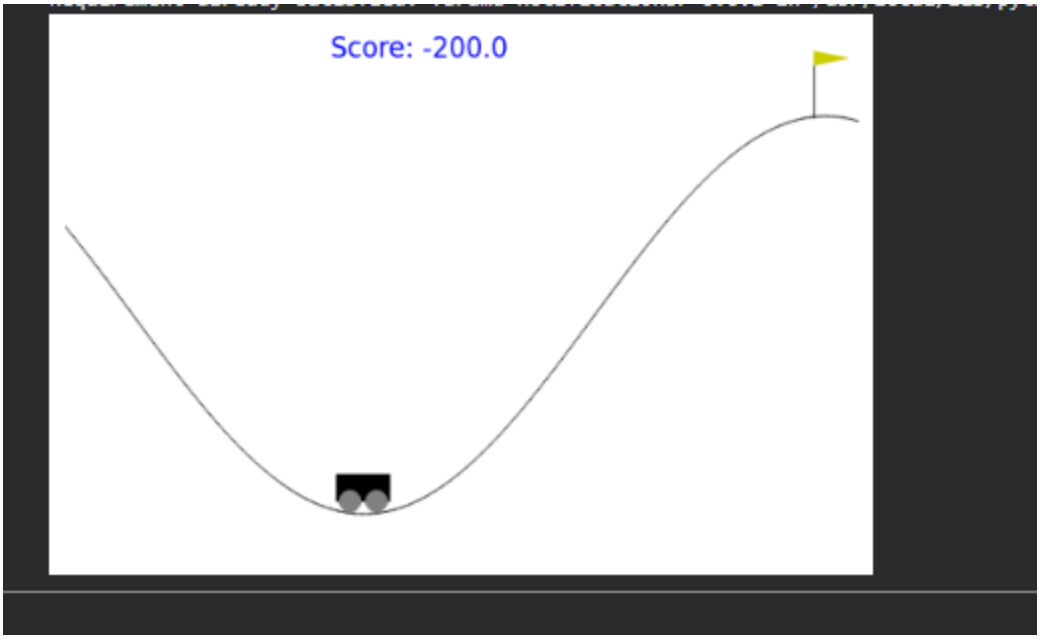
```
Car Velocity : -0.00022589651052840054
----------------------------------------
Step 193
State Array  : [-0.60279083  0.00137247]
Car Position : -0.6027908325195312
Car Velocity : 0.001372471917420625 7
----------------------------------------
Step 194
State Array  : [-0.60183     0.00096084]
Car Position : -0.601830005645752
Car Velocity : 0.0009608409600332379
----------------------------------------
Step 195
State Array  : [-0.6002878  0.0015422]
Car Position : -0.6002877950668335
Car Velocity : 0.001542203710414469 2
----------------------------------------
Step 196
State Array  : [-0.5971755   0.00311231]
Car Position : -0.5971754789352417
Car Velocity : 0.003112310776486993
----------------------------------------
Step 197
State Array  : [-0.5945158   0.00265967]
Car Position : -0.5945158004760742
Car Velocity : 0.002659666119143367
----------------------------------------
Step 198
State Array  : [-0.5923283   0.00218754]
Car Position : -0.5923283100128174
Car Velocity : 0.002187540521845221 5
----------------------------------------
Step 199
State Array  : [-0.5906289   0.00169937]
Car Position : -0.5906289219856262
Car Velocity : 0.001699367305263877
```

Task 3: Change Text Color & Location

- Change score color from **red to blue**
- Display text at position **(200, 20)**

```
!pip install gymnasium

import gymnasium as gym
import matplotlib.pyplot as plt

env = gym.make("MountainCar-v0", render_mode="rgb_array")

state, _ = env.reset()
score = 0
done = False

while not done:
    action = env.action_space.sample()
    state, reward, terminated, truncated, _ = env.step(action)
    done = terminated or truncated
    score += reward

    frame = env.render()

# Show frame with score text
plt.imshow(frame)
plt.text(200, 20, f"Score: {score}", color="blue", fontsize=12)
plt.axis("off")
plt.show()

env.close()
```

```
Requirement already satisfied: gymnasium in /usr/local/lib/python3.12/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
```



Task 4: Track Best Performance

- Store the score of each episode
- Print the **best (highest) score** at the end

```
# Install library
!pip install gymnasium

import gymnasium as gym

# Create environment
env = gym.make("MountainCar-v0")

episodes = 20   # aap 50 bhi kar sakti ho
episode_scores = []

for episode in range(episodes):
    state, _ = env.reset()
    done = False
    score = 0

    while not done:
        action = env.action_space.sample()   # random action
        state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated
        score += reward

    episode_scores.append(score)
    print(f"Episode {episode+1} Score: {score}")

# 🔹 Best performance
best_score = max(episode_scores)

print("\nAll Episode Scores:", episode_scores)
print("Best (Highest) Score:", best_score)

env.close()
```

```
Requirement already satisfied: gymnasium in /usr/local/lib/python3.12/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
Episode 1 Score: -200.0
Episode 2 Score: -200.0
Episode 3 Score: -200.0
Episode 4 Score: -200.0
Episode 5 Score: -200.0
Episode 6 Score: -200.0
Episode 7 Score: -200.0
Episode 8 Score: -200.0
Episode 9 Score: -200.0
Episode 10 Score: -200.0
Episode 11 Score: -200.0
Episode 12 Score: -200.0
Episode 13 Score: -200.0
Episode 14 Score: -200.0
Episode 15 Score: -200.0
Episode 16 Score: -200.0
Episode 17 Score: -200.0
Episode 18 Score: -200.0
Episode 19 Score: -200.0
Episode 20 Score: -200.0

All Episode Scores: [-200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0, -200.0]
Best (Highest) Score: -200.0
```

Task 5: Slow Down Visualization

Add the following line inside the loop:

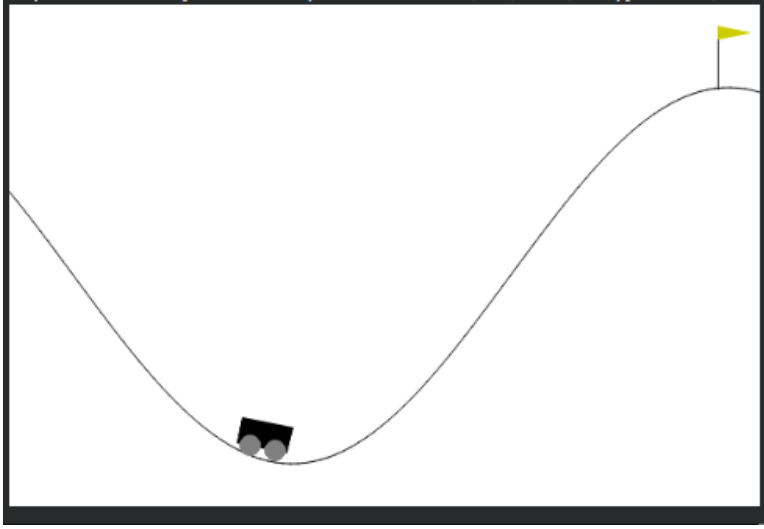
pygame.time.delay(20)

Observe the change in animation speed.

```python
# Install required packages
!pip install gymnasium pygame imageio

import gymnasium as gym
import imageio
import numpy as np

# Create environment
env = gym.make("MountainCar-v0", render_mode="rgb_array")
state, _ = env.reset(seed=42)

frames = []
done = False
total_reward = 0

while not done:
    action = env.action_space.sample()  # random action
    state, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated
    total_reward += reward

    # Capture frame for GIF
    frame = env.render()
    frames.append(frame)

env.close()

# Save frames as a GIF
gif_path = "mountaincar_slow.gif"
# Slow down animation: 50ms per frame (~20 FPS)
imageio.mimsave(gif_path, frames, duration=0.05)
```

```
from IPython.display import Image
Image(filename=gif_path)
```

```
Requirement already satisfied: gymnasium in /usr/local/lib/python3.12/dist-packages (1.2.2)
Requirement already satisfied: pygame in /usr/local/lib/python3.12/dist-packages (2.6.1)
Requirement already satisfied: imageio in /usr/local/lib/python3.12/dist-packages (2.37.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.12/dist-packages (from imageio) (11.3.0)
```



Task 6: Compare with CartPole

Replace the environment with:

env = gym.make("CartPole-v1", render_mode="human")

Compare:

- Reward behavior
- Episode termination
- Learning difficulty

Feature	MountainCar-v0	CartPole-v1
Reward behavior	Sparse: -1 per step until reaching the goal → harder to learn	Dense: +1 per step while pole is balanced → easier feedback
Episode termination	Ends when car reaches top of hill or max steps	Ends if pole falls too far or max steps reached

Learning difficulty	Harder: requires momentum, trial-and-error	Easier: straightforward control, dense rewards
Visualization	Slow, physics-based	Pole moves quickly, requires fast balancing

```
# Install required packages
!pip install gymnasium pygame imageio

import gymnasium as gym
import imageio
import numpy as np
from IPython.display import Image

# 🔷 Replace environment with CartPole
env = gym.make("CartPole-v1", render_mode="rgb_array")  # rgb_array works in Colab
state, _ = env.reset(seed=42)

frames = []
done = False
total_reward = 0
step_count = 0

while not done:
    action = env.action_space.sample()  # Random action
    state, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated
    total_reward += reward
    step_count += 1

    # Capture frame for GIF
    frame = env.render()
    frames.append(frame)

env.close()
```

```
env.close()

# Save frames as GIF (slow down with duration=0.05)
gif_path = "cartpole_slow.gif"
imageio.mimsave(gif_path, frames, duration=0.05)

# Display GIF in Colab
Image(filename=gif_path)

# Print some stats for comparison
print("CartPole episode finished")
print("Total reward:", total_reward)
print("Total steps:", step_count)
```

```
Requirement already satisfied: gymnasium in /usr/local/lib/python3.12/dist-packages (1.2.2)
Requirement already satisfied: pygame in /usr/local/lib/python3.12/dist-packages (2.6.1)
Requirement already satisfied: imageio in /usr/local/lib/python3.12/dist-packages (2.37.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.12/dist-packages (from imageio) (11.3.0)
CartPole episode finished
Total reward: 22.0
Total steps: 22
```

Task 7: Simple Rule-Based Policy (Intermediate)

Replace random actions with:

if state[1] > 0:

    action = 2

else:

    action = 0

Observe whether the agent reaches the hilltop.

```
# Install required packages
!pip install gymnasium imageio

import gymnasium as gym
import imageio
from IPython.display import Image

# Create MountainCar environment
env = gym.make("MountainCar-v0", render_mode="rgb_array")
state, _ = env.reset(seed=42)

frames = []
done = False
total_reward = 0
step_count = 0

while not done:
    # 🔹 Task 7: Simple rule-based policy
    # state[1] is velocity
    if state[1] > 0:
        action = 2  # accelerate right
    else:
        action = 0  # accelerate left

    state, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated
    total_reward += reward
    step_count += 1

    # Capture frame for GIF
    frame = env.render()
    frames.append(frame)

env.close()

# Save GIF to visualize in Colab
gif_path = "mountaincar_rule_based.gif"
imageio.mimsave(gif_path, frames, duration=0.05)  # duration=0.05 slows animation

# Display GIF in Colab
Image(filename=gif_path)

# Print episode stats
print("Episode finished")
print("Total reward:", total_reward)
print("Total steps:", step_count)

# Check if hilltop was reached
if state[0] >= 0.5:
    print("✅ Hilltop reached!")
else:
    print("❌ Hilltop NOT reached")
```

```
# Print episode stats
print("Episode finished")
print("Total reward:", total_reward)
print("Total steps:", step_count)

# Check if hilltop was reached
if state[0] >= 0.5:
    print("✅ Hilltop reached!")
else:
    print("❌ Hilltop NOT reached")
```

```
Requirement already satisfied: gymnasium in /usr/local/lib/python3.12/dist-packages (1.2.2)
Requirement already satisfied: imageio in /usr/local/lib/python3.12/dist-packages (2.37.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
Requirement already satisfied: pillow>=8.3.2 in /usr/local/lib/python3.12/dist-packages (from imageio) (11.3.0)
Episode finished
Total reward: -91.0
Total steps: 91
✅ Hilltop reached!
```

Task 8 (Advanced): Episode Length Analysis

Print the **number of steps per episode** and analyze:

- Why some episodes last longer
- Relation between steps and score

```
# Install required packages
!pip install gymnasium

import gymnasium as gym
import numpy as np

# Create environment
env = gym.make("MountainCar-v0")  # No render needed for analysis

num_episodes = 10  # You can increase to 50 or 100 for better stats
episode_lengths = []
episode_rewards = []

for episode in range(num_episodes):
    state, _ = env.reset(seed=42+episode)
    done = False
    total_reward = 0
    step_count = 0

    while not done:
        # Simple rule-based policy
        if state[1] > 0:
            action = 2
        else:
            action = 0

        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        total_reward += reward
        step_count += 1
```

```python
    episode_lengths.append(step_count)
    episode_rewards.append(total_reward)
    print(f"Episode {episode+1}: Steps = {step_count}, Total Reward = {total_reward}")

env.close()

# Analysis
print("\n=== Episode Summary ===")
print("Average steps per episode:", np.mean(episode_lengths))
print("Average reward per episode:", np.mean(episode_rewards))
print("Max steps:", np.max(episode_lengths), "Min steps:", np.min(episode_lengths))

# Observations
print("\nObservations:")
print("• Episodes last longer when the car struggles to build enough momentum to reach the hilltop.")
print("• Shorter episodes occur if the car reaches the goal faster.")
print("• Reward is negatively correlated with steps (MountainCar gives -1 per step).")
print("  → More steps → lower total reward; fewer steps → higher total reward.")
```

```
Requirement already satisfied: gymnasium in /usr/local/lib/python3.12/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
Episode 1: Steps = 91, Total Reward = -91.0
Episode 2: Steps = 99, Total Reward = -99.0
Episode 3: Steps = 153, Total Reward = -153.0
Episode 4: Steps = 117, Total Reward = -117.0
Episode 5: Steps = 87, Total Reward = -87.0
Episode 6: Steps = 93, Total Reward = -93.0
Episode 7: Steps = 159, Total Reward = -159.0
Episode 8: Steps = 159, Total Reward = -159.0
```

```
Requirement already satisfied: gymnasium in /usr/local/lib/python3.12/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
Episode 1: Steps = 91, Total Reward = -91.0
Episode 2: Steps = 99, Total Reward = -99.0
Episode 3: Steps = 153, Total Reward = -153.0
Episode 4: Steps = 117, Total Reward = -117.0
Episode 5: Steps = 87, Total Reward = -87.0
Episode 6: Steps = 93, Total Reward = -93.0
Episode 7: Steps = 159, Total Reward = -159.0
Episode 8: Steps = 159, Total Reward = -159.0
Episode 9: Steps = 91, Total Reward = -91.0
Episode 10: Steps = 87, Total Reward = -87.0

=== Episode Summary ===
Average steps per episode: 113.6
Average reward per episode: -113.6
Max steps: 159 Min steps: 87

Observations:
• Episodes last longer when the car struggles to build enough momentum to reach the hilltop.
• Shorter episodes occur if the car reaches the goal faster.
• Reward is negatively correlated with steps (MountainCar gives -1 per step).
  → More steps → lower total reward; fewer steps → higher total reward.
```

📊 Observation Table

Episode	Steps	Score	Goal Reached (Yes/No)
1			
2			
...			
20			

```
# Install required packages
!pip install gymnasium pandas

import gymnasium as gym
import pandas as pd

# Create environment
env = gym.make("MountainCar-v0")  # No render needed for analysis

num_episodes = 20  # Match your table rows
data = []

for episode in range(1, num_episodes + 1):
    state, _ = env.reset(seed=42+episode)
    done = False
    total_reward = 0
    step_count = 0

    while not done:
        # Simple rule-based policy
        if state[1] > 0:
            action = 2
        else:
            action = 0

        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        total_reward += reward
        step_count += 1
```

```
            action = 0

        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        total_reward += reward
        step_count += 1

    # Check if goal reached
    goal_reached = "Yes" if state[0] >= 0.5 else "No"

    # Append to table
    data.append([episode, step_count, total_reward, goal_reached])

env.close()

# Create DataFrame
df = pd.DataFrame(data, columns=["Episode", "Steps", "Score", "Goal Reached (Yes/No)"])
df
```

```
Requirement already satisfied: gymnasium in /usr/local/lib/python3.12/dist-packages (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (2.0.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (3.1.2)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (4.15.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from gymnasium) (0.0.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```