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 < α ≤ 1)
- γ = discount factor (0 ≤ γ ≤ 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

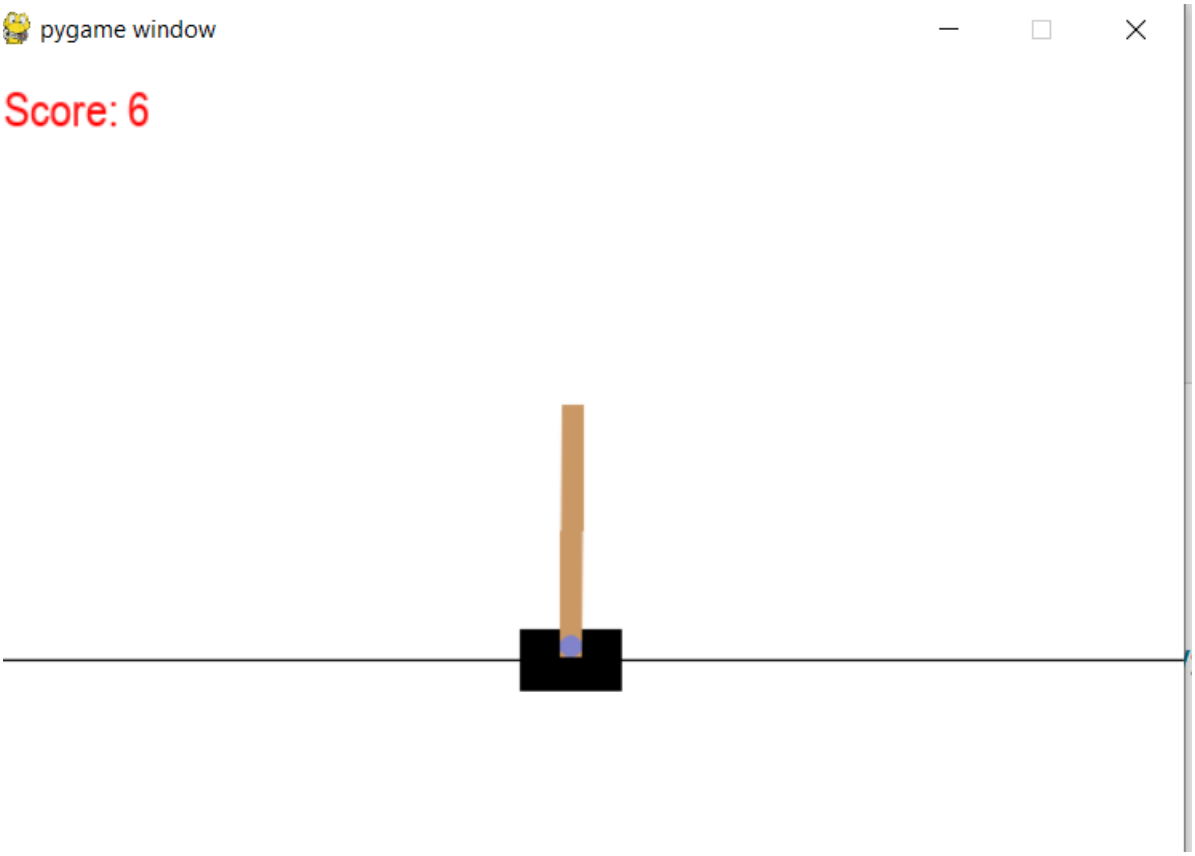
🎯 Lab Objectives

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.

Q2.

Explain the purpose of the following line:

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

Q3.

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

Q4.

Explain the difference between:

terminated

truncated

Q5.

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

Q6.

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

Q7.

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

Q8.

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

Lab Tasks (Hands-on Practice)

◈ 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

◈ Task 2: Display Episode Number on Screen

Modify the code to show:

Episode: X | Score: Y

on the CartPole window.

Task 3: Change Text Color and Position

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

◈ Task 4: Print Maximum Score

After all episodes finish:

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

◈ Task 5: Slow Down the Environment

Insert a small delay using:

pygame.time.delay(20)

Observe the effect on visualization.

◈ Task 6: Replace CartPole with MountainCar

Change the environment to:

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

Compare:

- Reward behavior
- Episode termination condition

Task 7: Identify State Variables

Print the state vector and answer:

- How many state variables are there?

- What does each variable represent?

◈ Task 8 (Advanced): Rule-Based Action

Replace random action with:

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

📊 Observation Table (For Students)

Episode	Score	Remarks
1		
2		
...		
20		

LAB Task 2:

Experiment: MountainCar Environment using Gymnasium & Pygame

Lab Objectives

After completing this lab, students will be able to:

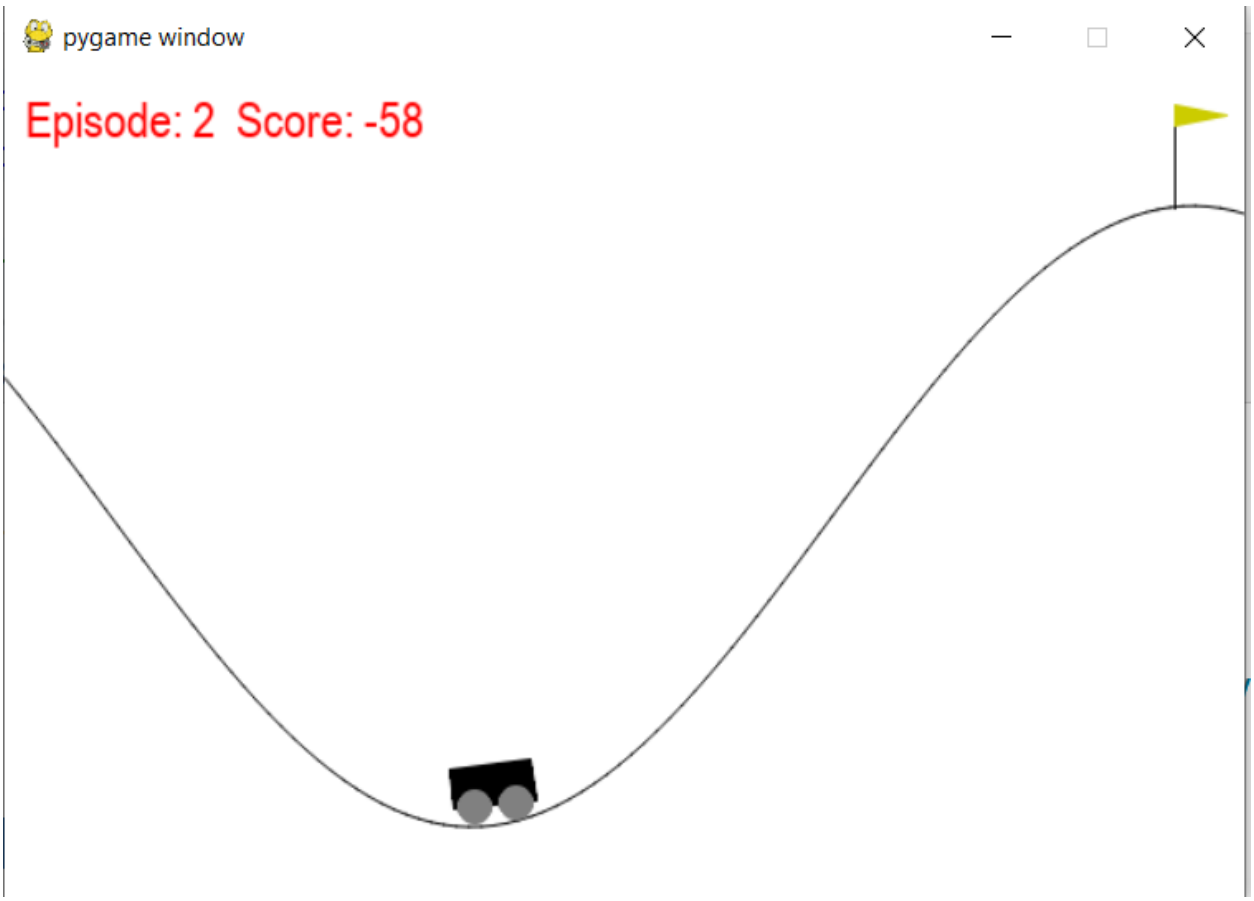
- Understand the **working of a continuous control RL environment**
- Analyze **delayed reward problems**
- Use **Gymnasium MountainCar-v0**
- Visualize agent behavior and rewards using **Pygame**

- Compare MountainCar with CartPole environment

Provided Code:

```
import gymnasium as gym
import pygame

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

font = None
best_score = -float('inf')

# We only need a few episodes to prove it works with a better policy
NUM_EPISODES = 5

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

    while not done:
        # Task 7/8: Advanced Rule-Based Action
        # state[1] is velocity. If velocity is moving right (>0), push right (2).
        # If moving left (<0), push left (0). This builds momentum rapidly.
        if state[1] > 0:
            action = 2
        else:
            action = 0

        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, (0, 0, 255))
        surface.blit(text, (200, 20))

        # Reduced delay for faster execution
        pygame.time.delay(5)
        pygame.display.update()
```

```
    print(f"Episode {episode} Score: {score}")
    if score > best_score:
        best_score = score

env.close()
pygame.quit()

print(f"\nOptimization Results:")
print(f"Best Score Achieved: {best_score}")
```



Lab Questions (Conceptual Understanding)

Q1.

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

Q2.

Explain the purpose of the following statement:

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

Q3.

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

Q4.

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

Q5.

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

Q6.

What is the difference between:

terminated

truncated

in this environment?

Q7.

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

Q8.

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

🛠️ 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.

◈ Task 2: Display State Values

Print the state array in each step and identify:

- Car position
- Car velocity

◈ Task 3: Change Text Color & Location

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

◈ Task 4: Track Best Performance

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

◈ Task 5: Slow Down Visualization

Add the following line inside the loop:

pygame.time.delay(20)

Observe the change in animation speed.

◈ Task 6: Compare with CartPole

Replace the environment with:

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

Compare:

- Reward behavior

- Episode termination
- Learning difficulty

◈ 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.

◈ 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

📊 Observation Table

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

LAB Assessment

Student Name		**LAB Rubrics**	CLO3 , P5, PLO5
		Total Marks	10
Registration No		**Obtained Marks**	

		Teacher Name	Dr. Syed M Hamedoon
Date		**Signature**	