LAB No 12

Implementation of Reinforcement Learning Part 1

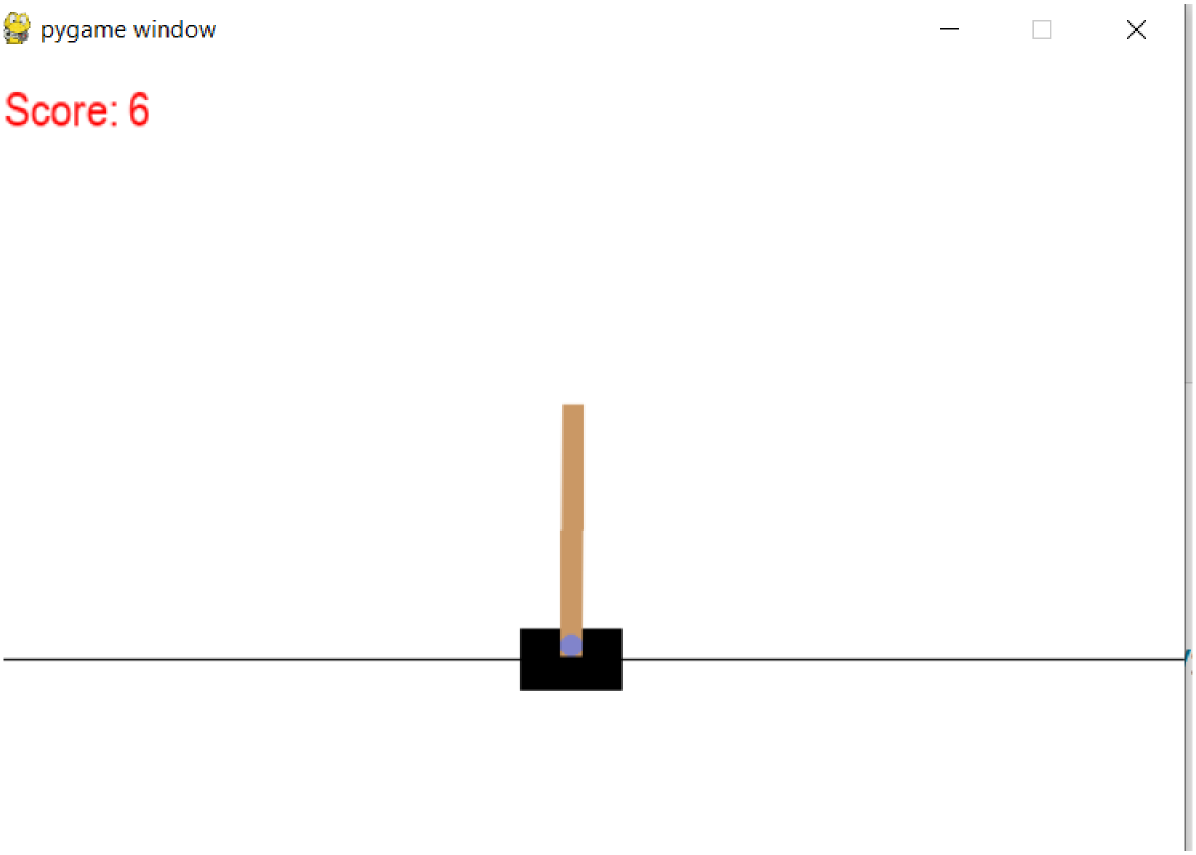
Experiment: CartPole Environment using Gymnasium & Pygame

🎯 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()
```



Score: 6

Lab Questions (Conceptual Understanding)

Q1.

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

Answer:

RL is a type of machine learning where an agent learns by **taking actions in an environment** and getting **rewards**. The goal is to maximize total rewards over time.

In the CartPole code:

Component	**Description in the Code**
Agent	The part of the code that chooses actions: action = env.action_space.sample()
Environment	The CartPole simulation created with: env = gym.make("CartPole-v1", render_mode="human")
State	The current situation of the system returned by env.reset() or env.step(action) It includes: [cart_position, cart_velocity, pole_angle, pole_angular_velocity]
Action	What the agent does at each step: 0 = push cart left, 1 = push cart right
Reward	The feedback the agent gets: reward = 1 for every step the pole stays upright

---

Q2.

Explain the purpose of the following line:

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

Answer:

This line **sets up the environment** and makes it **visible to the user** so you can watch the agent act in real time.

Q3.

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

Answer:

What it does:

- env.reset() **resets the environment** to the starting state for a new episode.
- It returns **two values:**
    1. **state** → The initial observation of the environment (CartPole variables: cart position, cart velocity, pole angle, pole angular velocity)
    2. **info** → Additional information from the environment (usually empty or extra metadata; not used in basic experiments)

Why two values:

- Gymnasium separates **the important observation (state)** from **optional metadata (info)**.
- This allows the code to use the state for decision-making while still having access to extra info if needed.

Q4.

Explain the difference between: Terminated and truncated

Answer:

Term	**Meaning in Gymnasium / CartPole**
terminated	The episode ended because the **goal was reached** or **failure occurred**. In CartPole: the pole fell too far or cart moved out of bounds.
truncated	The episode ended because it **reached the maximum allowed steps**. This is not due to failure, just a **time limit**.

Q5.

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

Answer:

Role:

- score keeps track of the **total reward** the agent receives during an episode.
- It measures **how well the agent is performing—**higher score means the pole stayed upright longer.

How it is calculated:

score += reward

- At each step, the environment gives a **reward** (in CartPole, reward = 1 per step).
- The code **adds this reward to score** until the episode ends.
- At the end of the episode, score represents the **total steps the pole stayed balanced**.

Q6.

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

Answer:

action = env.action_space.sample() is used to **choose a random action** at each step.

This is not an intelligent agent because it **does not learn** from rewards or past experience; it acts **randomly**.

Q7.

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

Answer:

Pygame is used to **draw the score on the simulation window**.

Steps in the code:

1. Initialize Pygame font: pygame.font.SysFont("Arial", 24)
2. Create a surface (the window) using pygame.display.get_surface()
3. Render the score as text: font.render(f"Score: {int(score)}", True, (255,0,0))
4. Draw it on the window at a position: surface.blit(text, (10, 10))
5. Update the display: pygame.display.update()

Q8.

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

Answer:

- If pygame.display.update() is removed, the **score will not appear or refresh** on the screen.
- The **Pygame** window **won't show changes**, so the score text won't be visible while the simulation runs.

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

Answer:

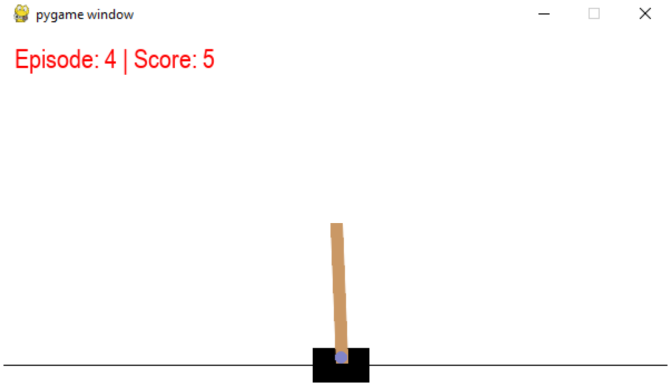
- The score **changes a lot from episode to episode**.
- Some episodes have **low scores** (around 9–15).
- Some episodes have **higher scores** (around 40–58).
- There is **no fixed pattern** in the scores

🔷 Task 2: Display Episode Number on Screen

Modify the code to show:

Episode: X | Score: Y

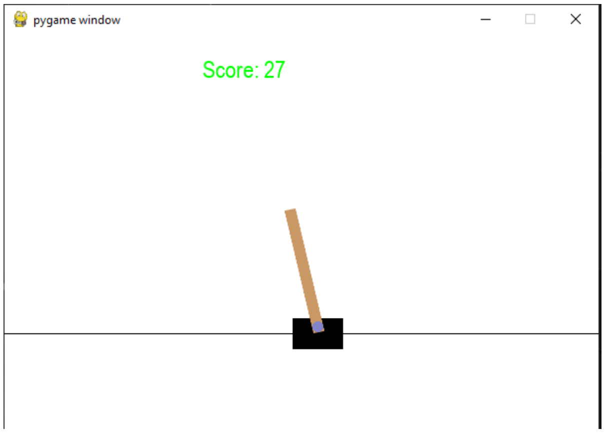
on the CartPole window.



Episode: 4 | Score: 5

Task 3: Change Text Color and Position

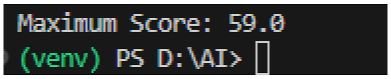
- Change score text color from **red to green**
- Display it at position **(200, 20)**



🔷 Task 4: Print Maximum Score

After all episodes finish:

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



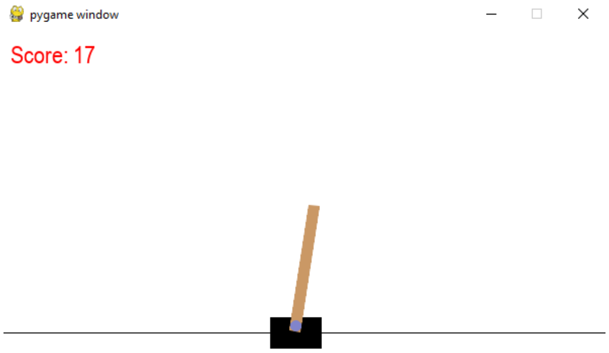
```
Maximum Score: 59.0
(venv) PS D:\AI>
```

🔷 Task 5: Slow Down the Environment

Insert a small delay using:

pygame.time.delay(20)

Observe the effect on visualization.



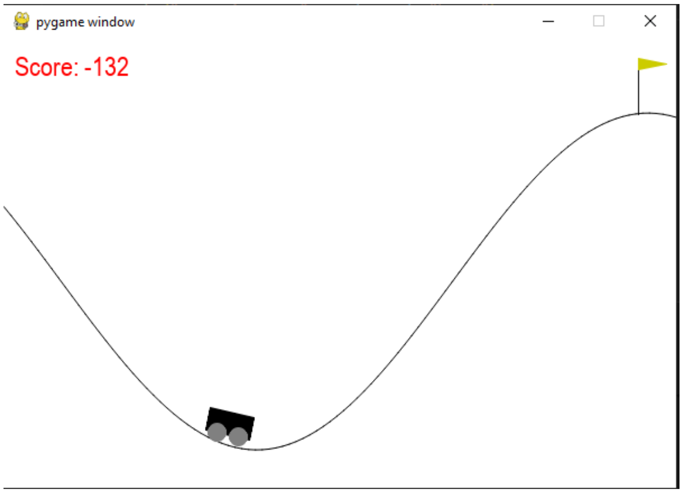
Task 6: Replace CartPole with MountainCar

Change the environment to:

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

Compare:

- Reward behavior
- Episode termination condition

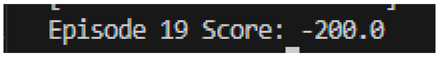


```
Score: -132
```

Task 7: Identify State Variables

Print the state vector and answer:

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

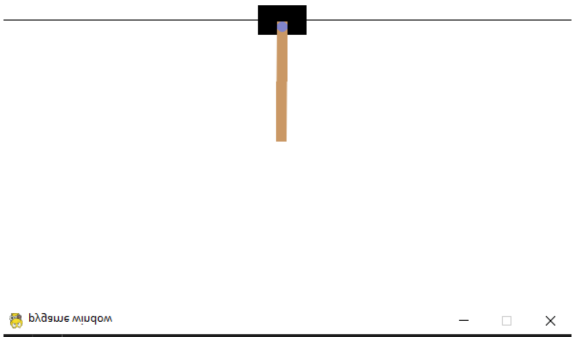


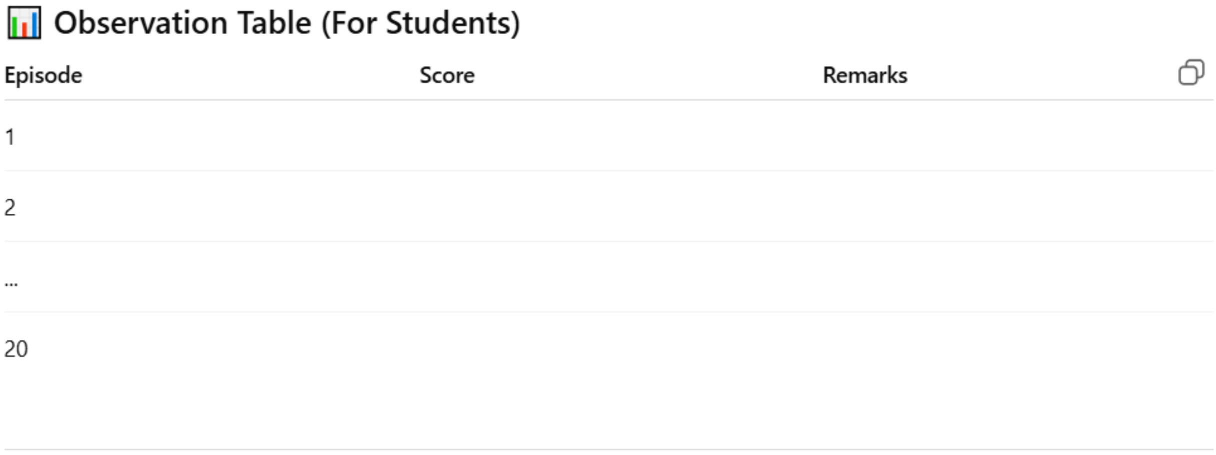
```
Episode 19 Score: -200.0
```

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		

Experiment: MountainCar Environment using Gymnasium & Pygame

Lab Objectives

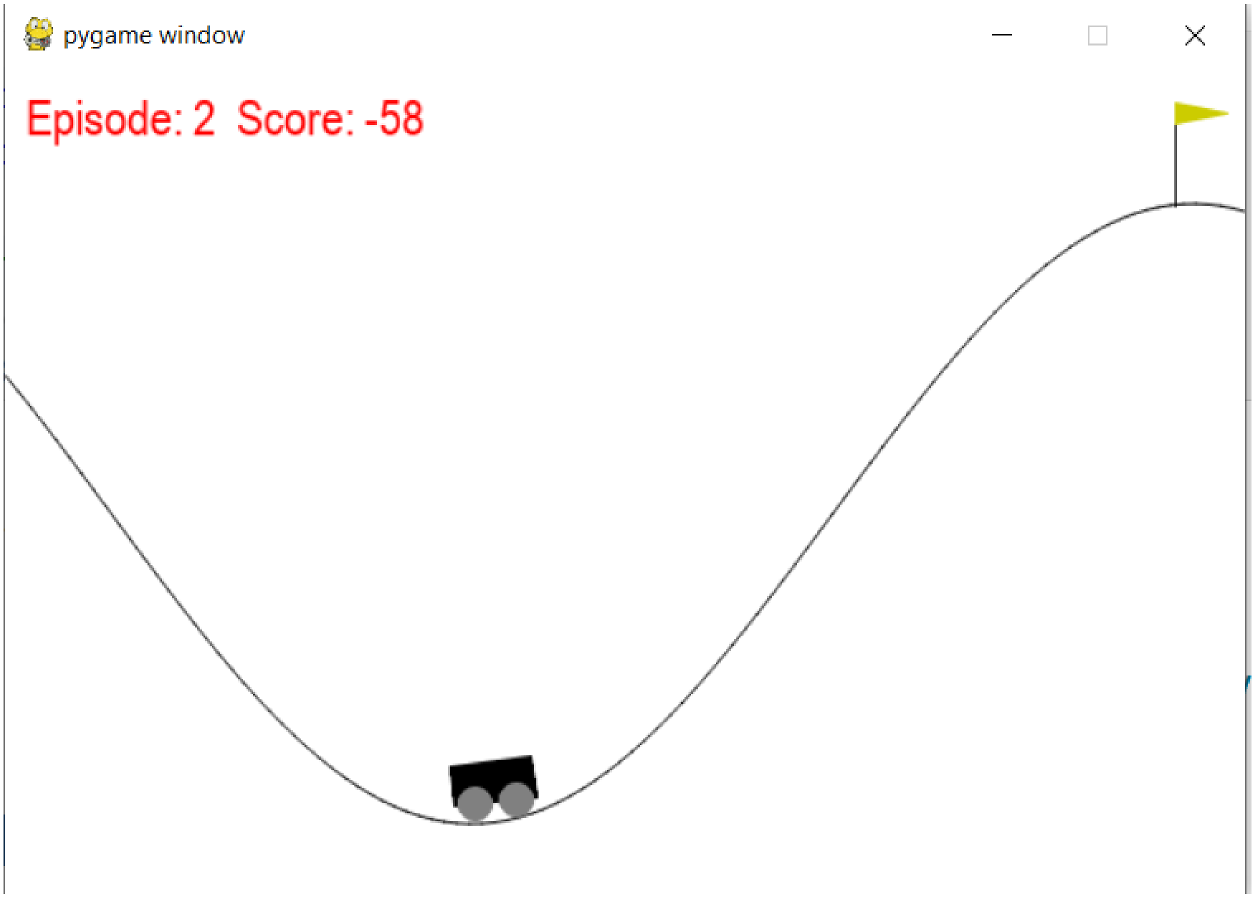
After completing this lab, students will be able to:

- 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}")
```



Episode: 2 Score: -58

Lab Questions (Conceptual Understanding)

Q1.

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

Answer:

Reinforcement Learning (RL) is a learning method where an agent learns by interacting with an environment and receiving rewards.

In MountainCar code:

- **Agent:** The car controller (our program)
- **Environment:** MountainCar-v0
- **State:** [position, velocity]
- **Action:** Push left (0), no push (1), push right (2)
- **Reward:** -1 at every step until goal is reached

Q2.

Explain the purpose of the following statement:

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

Answer:

- Creates the MountainCar environment
- render_mode="human" displays the environment visually on the screen

---

Q3.

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

Answer:

- **Position (state[0])**
  → Horizontal position of the car on the hill

- **Velocity (state[1])**
  → Speed and direction of the car's movement

Q4.

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

Answer:

Action	Meaning
0	Push car left
1	No push
2	Push car right

Q5.

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

Answer:

- The agent receives **-1 reward at every step**
- The goal is to **reach the hilltop in fewer steps**

Reason for negative reward:
To encourage the agent to reach the goal as quickly as possible.

Q6.

What is the difference between:

Terminated and truncated in this environment?

Answer:

- **terminated:** Episode ends because the goal is reached
- **truncated:** Episode ends because maximum step limit is reached

---

Q7.

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

Answer:

- Random actions do not build momentum
- The car cannot climb the hill without coordinated left-right movement
- Therefore, the agent fails to reach the goal

Q8.

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

Answer:

- The car must first move **away from the goal** to gain speed
- Momentum helps the car climb the steep hill
- Without momentum, the car cannot reach the top
