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 s Pygame

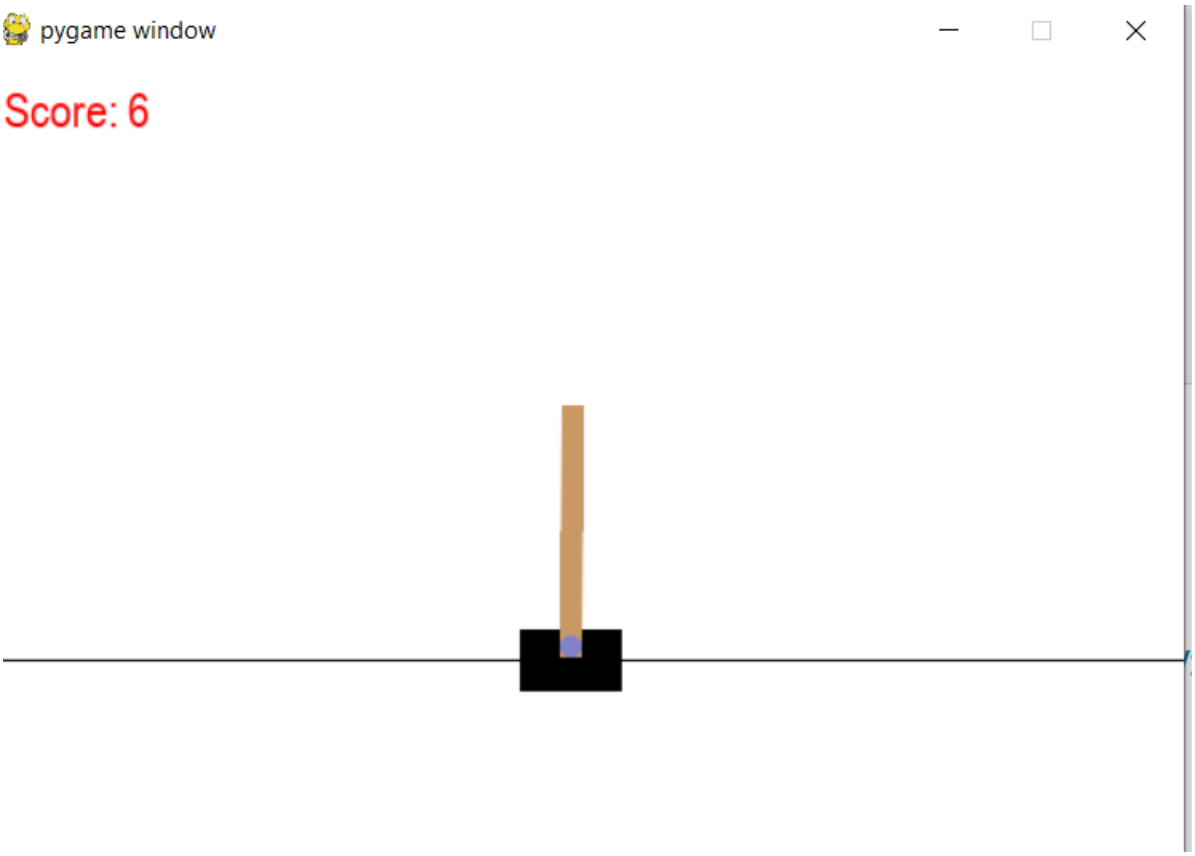
☛ Lab Objectives

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

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:

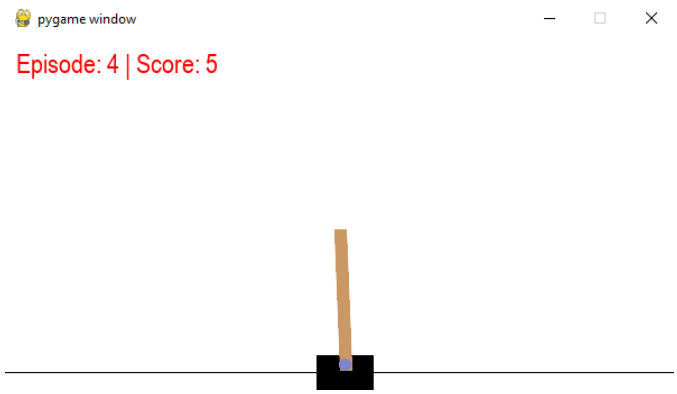
- The score **changes a lot from episode to episode**.
- Some episodes have **low scores** (around 6–15).
- Some episodes have **higher scores** (around 40–58).
- There is **no fixed pattern** in the scores

Task 2: Display Episode Number on Screen

Modify the code to show:

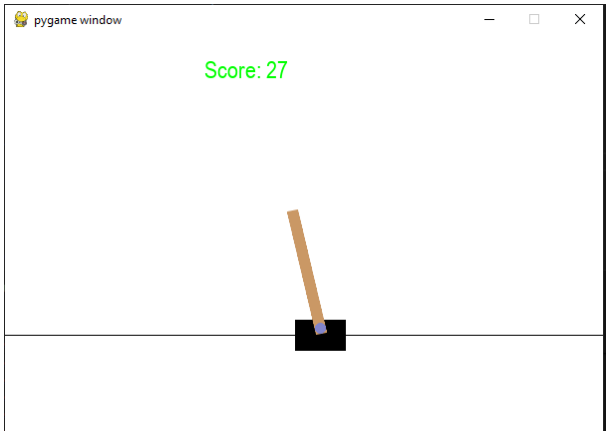
Episode: X | Score: Y

on the CartPole window.



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**

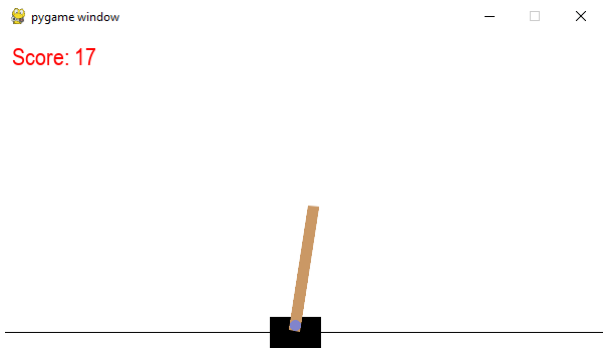
```
Maximum Score: 59.0
(venv) PS D:\AI>
```

Task 5: Slow Down the Environment

Insert a small delay using:

pygame.time.delay(20)

Observe the effect on visualization.

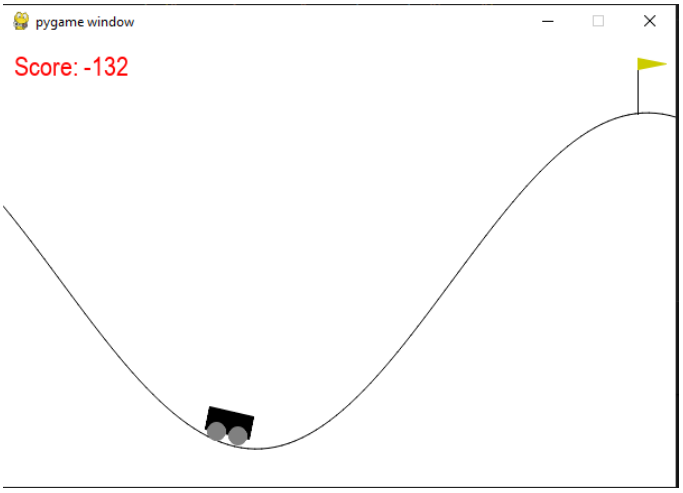


Task 6: Replace CartPole with MountainCar

Change the environment to:

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

- Reward behavior
- Episode termination condition



Task 7: Identify State Variables

Print the state vector and answer:

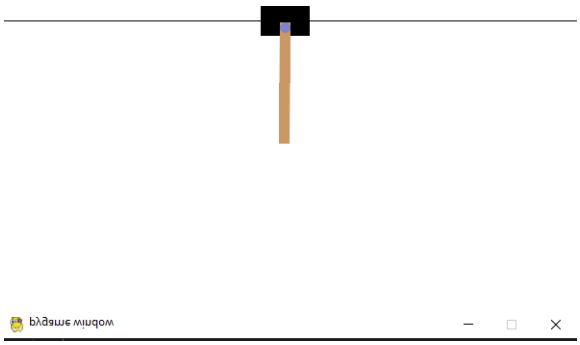
- How many state variables are there?
- What does each variable represent?

```
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 s

Pygame

Lab Objectives

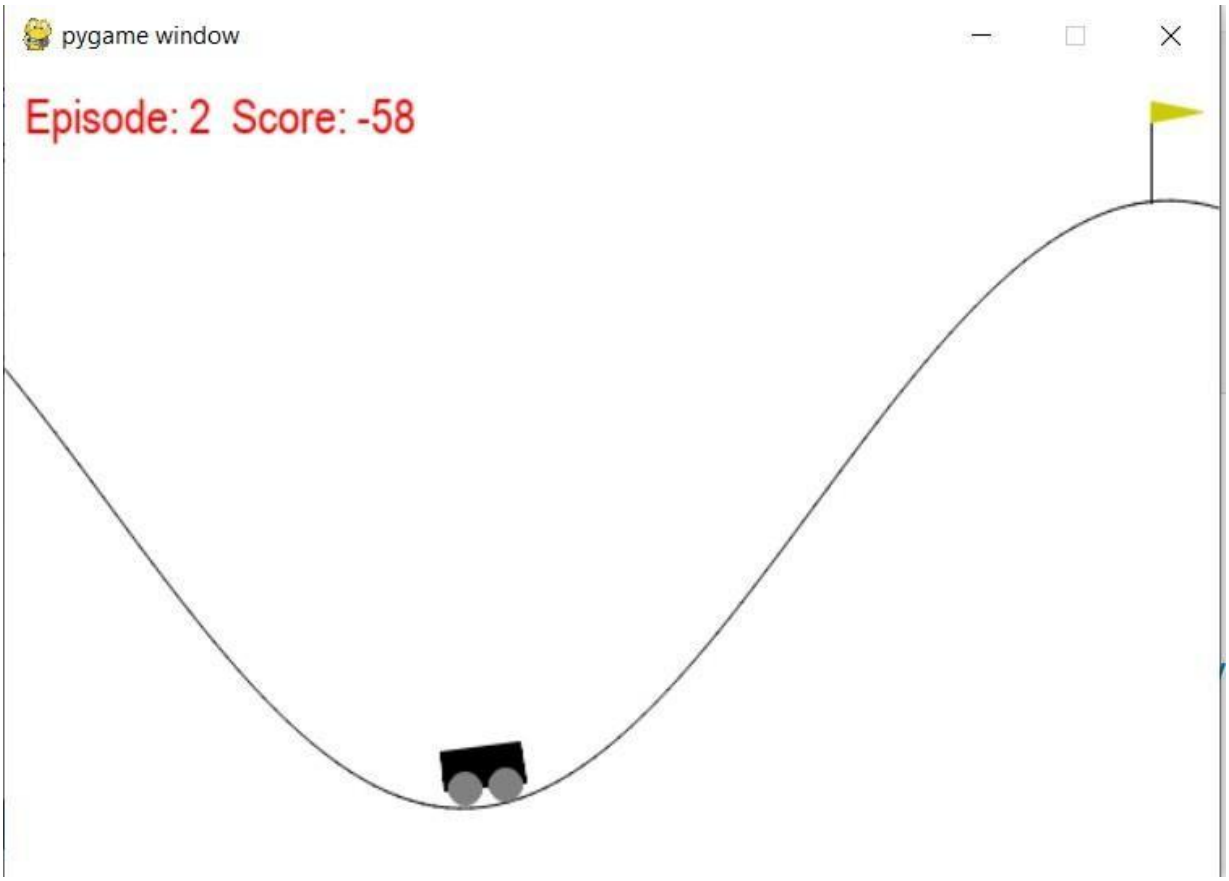
After completing this lab, students will be able to:

- 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

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 goalis 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

Lab Assessment:

Student Name		**LAB Rubrics**	CLO3 , P5, PLO5
		Total Marks	10
Registration No		**Obtained Marks**	
		Teacher Name	Dr. Syed M Hamedoon
Date		**Signature**	

Laboratory Work Assessment Rubrics

Sr. No.	Performance Indicator	Excellent (5)	Good (4)	Average (3)	Fair (2)	Poor (1)
1	**Theoretical knowledge 10%**	Student knows all the related concepts about the theoretical background of the experiment and rephrase those concepts in written and oral assessments	Student knows most of the related concepts about the theoretical background of the experiment and partially rephrase those concepts in written and oral assessments	Student knows few of the related concepts about the theoretical background of the experiment and partially rephrase those concepts in written and oral assessments	Student knows very little about the related concepts about the theoretical background of the experiment and poorly rephrase those concepts in written and oral assessments	Student has poor understanding of the related concepts about the theoretical background of the experiment and unable to rephrase those concepts in written and oral assessments
2	**Application Functionality 10%**	Application runs smoothly and operation of the application runs efficiently	Application compiles with no warnings. Robust operation of the application, with good recovery.	Application compiles with few or no warnings. Consideration given to unusual conditions with reasonable	Application compiles and runs without crashing. Some attempt at detecting and correcting errors.	Application does not compile or compiles but crashes. Confusing. Little or no error detection or correction.
3	**Specifications 10%**	The program works very efficiently and meets all of the required specifications.	The program works and meets some of the specifications.	The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results but does not display them correctly.	The program is producing incorrect results.
4	**Level of understanding of the learned skill 10%**	Provide complete and logical answers based upon accurate technical content to the questions asked by examiner	Provide complete and logical answers based upon accurate technical content to the questions asked by examiner with few errors	Provide partially correct and logical answers based upon minimum technical content to the questions asked by examiner	Provide very few and illogical answers to the questions asked by examiner.	Provide no answer to the questions asked by examiner.
5	**Readability and Reusability 10%**	The code is exceptionally well organized and very easy to follow and reused	The code is fairly easy to read. The code could be reused as a whole or each class could be reused.	Most of the code could be reused in other programs.	Some parts of the code require change before they could be reused in other programs.	The code is poorly organized and very difficult to read and not organized for reusability.

6	**AI System Design 10%**	Well-designed AI models. Code is highly maintainable	Good designed AI models and Little code duplications	Some attempt to make AI models. Code can be maintained with significant effort	Little attempt to design AI models and less understanding of code	Very poor attempt to design AI models and its code
7	**Responsiveness to Questions/ Accuracy 10%**	1. Responds well, quick and very accurate all the time. 2. Effectively uses eye contact, speaks clearly, effectively and confidently using suitable volume	1. Generally Responsive and accurate most of the times. 2. Maintains eye contact, speaks clearly with suitable volume and pace.	1. Generally Responsive and accurate few times. 2. Some eye contact, speaks clearly and unclearly in different portions.	1. Not much Responsive and accurate most of the times. 2. Uses eye contact ineffectively and fails to speak clearly and audibly	. 1. Non Responsive and inaccurate all the times. 2. No eye contact and unable to speak 3. Dresses inappropriately
8	**Efficiency 10%**	The code is extremely efficient without sacrificing readability and understanding	The code is fairly efficient without sacrificing readability and understanding	Some part of the code is efficient and other part of the code is not understandable and work properly	The code is brute force and unnecessarily long	The code is huge and appears to be patched together
G	**Delivery 10%**	The program was delivered in time during lab.	The program was delivered in Lab before the end time.	The program was delivered within the due date.	The code was delivered within a day after the due date.	The code was delivered more than 2 days overdue.
10	**Awarenessof Safety Guidelines 10%**	Student has sufficient knowledge of the laboratory safety SOPsand protocol and is fully compliant to the guidelines	Student has sufficient knowledge of the laboratory safety SOPs and protocol and is Partially compliant to the guidelines	Student has little knowledge of the laboratory safety SOPs and protocol and is Partially compliant to the guidelines	Student has little knowledge of the laboratory safety SOPs and protocol and is non-compliant to the guidelines	Student has no knowledge of the laboratory safety SOPs and protocol and isnon-compliant to the guidelines