Lab Assignment: Exploring OpenAI Gym with MountainCar-v0

1. Installation of Necessary Libraries

To start, install the `gymnasium` library, which provides access to the MountainCar environment. If you haven't installed it yet, you can do so as follows:

```
!pip install gymnasium matplotlib
```

```
Collecting gymnasium
  Downloading gymnasium-1.0.0-py3-none-any.whl.metadata (9.5 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (1.26.4)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (3.1.0)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (4.12.2)
Collecting farama-notifications>=0.0.1 (from gymnasium)
  Downloading Farama_Notifications-0.0.4-py3-none-any.whl.metadata (558 bytes)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Downloading gymnasium-1.0.0-py3-none-any.whl (958 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 958.1/958.1 kB 7.3 MB/s eta 0:00:00
Downloading Farama_Notifications-0.0.4-py3-none-any.whl (2.5 kB)
Installing collected packages: farama-notifications, gymnasium
Successfully installed farama-notifications-0.0.4 gymnasium-1.0.0
```

2. Import Libraries

```
import gymnasium as gym  # OpenAI Gym environment
import numpy as np       # Numerical operations
from IPython.display import display, clear_output  # Display tools
import matplotlib.pyplot as plt  # Plotting tools
from PIL import Image  # For image processing of environment frames
import time  # For introducing delays in rendering
import pandas as pd  # For tabular representation
```

3. Set Up the Environment with IPython Visualization

1. **Initialize the Environment**: Create the MountainCar environment with visual output.
2. **Define Policy**: Use a simple policy to guide the car.
3. **Display Frames**: Capture each frame, render, and show it inline in the Colab notebook.

```
# Set up the MountainCar-v0 environment
env = gym.make("MountainCar-v0", render_mode="rgb_array")
```

Policy: move left if velocity is negative, else move right.

```
# Define a simple policy
def simple_policy(state):
    _, velocity = state
    return 2 if velocity >= 0 else 0  # Right if moving right, else left
```

Capture a frame from the environment and display it.

```
# Define the display function for Jupyter
def display_frame(env):
    frame = env.render()  # Render the environment as an RGB image
    img = Image.fromarray(frame)  # Convert frame to PIL image format
    display(img)  # Display frame
    clear_output(wait=True)  # Clear the previous output for the next frame
```

```
# Run episodes and display frames
num_episodes = 5
episode_rewards = []  # Track rewards for each episode
episode_steps = []    # Track steps taken in each episode

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

    while not done:
        # Take action based on the simple policy
        action = simple_policy(state)
        state, reward, done, truncated, info = env.step(action)

        total_reward += reward
        step_count += 1

        # Display the current frame
        display_frame(env)
        time.sleep(0.05)  # Optional delay for better visualization

        if done:
            print(f"Episode {episode + 1} finished in {step_count} steps with total reward {total_reward}")
            episode_rewards.append(total_reward)
            episode_steps.append(step_count)
            break

env.close()  # Close the environment after finishing all episodes
```

```
Episode 5 finished in 116 steps with total reward -116.0
```

4. Display Rewards

After running all episodes, we can visualize the total rewards per episode to evaluate how well the policy performs.

```
# Create a DataFrame for the rewards and steps per episode
reward_data = pd.DataFrame({
    "Episode": range(1, num_episodes + 1),
    "Total Reward": episode_rewards,
    "Steps Taken": episode_steps
})

# Display the table
display(reward_data)
```

	Episode	Total Reward	Steps Taken
0	1	-116.0	116
1	2	-115.0	115
2	3	-115.0	115
3	4	-122.0	122
4	5	-116.0	116

Next steps: Generate code with `reward_data` View recommended plots New interactive sheet

Explanation of Key Parts

- **display_frame()**: This function captures and displays each frame in the Jupyter notebook by converting the RGB array to a PIL image.
- **clear_output()**: Clears the previous frame to give a smoother visualization of the agent's movement.
- **Episode Loop**: Runs each episode and tracks the agent's performance using a basic momentum-based policy.

Key Insights

1. **Policy Effectiveness**: The basic momentum-based policy (moving right if velocity is positive, otherwise left) provided a straightforward approach to control the car. However, it lacks the sophistication needed to consistently reach the goal due to the limitations of hard-coded actions.
2. **Visualization of the Environment**: Utilizing the `display_frame()` function with IPython allowed for clear, step-by-step visualization within Jupyter. The frame clearing after each step created smooth transitions, effectively showing the agent's progression toward the goal.

3. **Tracking Performance**: Tabular tracking of rewards and steps for each episode helped visualize the agent's performance, enabling easy analysis of how many steps were taken and the rewards achieved in each episode. This data can be useful for optimizing the policy or implementing a learning algorithm.
4. **Agent's Momentum Concept**: This experiment highlighted the significance of momentum in controlling the car effectively, especially in environments that require building up momentum to overcome obstacles, like the MountainCar.

Conclusion

The MountainCar-v0 environment provides a valuable introduction to continuous control and reinforcement learning concepts. By using a simple, rule-based policy, we demonstrated how to visualize and monitor an agent's performance in OpenAI Gym using Python. Although the policy was rudimentary, the exercise illustrated the importance of more advanced algorithms in reinforcement learning for achieving complex goals. The insights gained from tracking steps and rewards suggest that a more adaptive policy could significantly improve performance. This lab serves as a foundation for exploring more advanced control strategies in future studies.