**Experiment No.01**

PART A

(PART A: TO BE REFFERED BY STUDENTS)

**A.1 Aim: Introduction to RL Environment and Terminologies using OpenAI Gym**

1. Explore various environments in Open AI Gym
2. Relate RL terminologies and elements with the examples in Gym Environment
3. Implement any Gym Environment from classic control
4. Customize the above gym environment parameters
5. Demonstrate how to interpret state transitions, rewards, and termination conditions with reference to the above example

**A.2 Prerequisite:**

Concept of RL, RL terminologies and elements, Python Programming

**A.3 Learning Outcome:**

After completing this experiment you will be able to-

* Comprehend the fundamental concept of RL terminologies and elements
* Implement basic agent-environment interactions, such as taking actions and receiving rewards
* Implement an agent that interacts with the environment by selecting actions based on states
* Understand how to interpret state transitions and rewards within the context of a specific RL problem

**A.4 Theory:**

**A.4.1 Reinforcement Learning and its key concepts**

Reinforcement Learning (RL) is a branch of machine learning where an agent learns to make decisions by interacting with an environment. Unlike supervised learning, where the agent learns from labeled examples, or unsupervised learning, where the agent discovers patterns in data without specific guidance, RL learns through trial-and-error feedback, known as rewards and punishments.

RL process flow is given below-

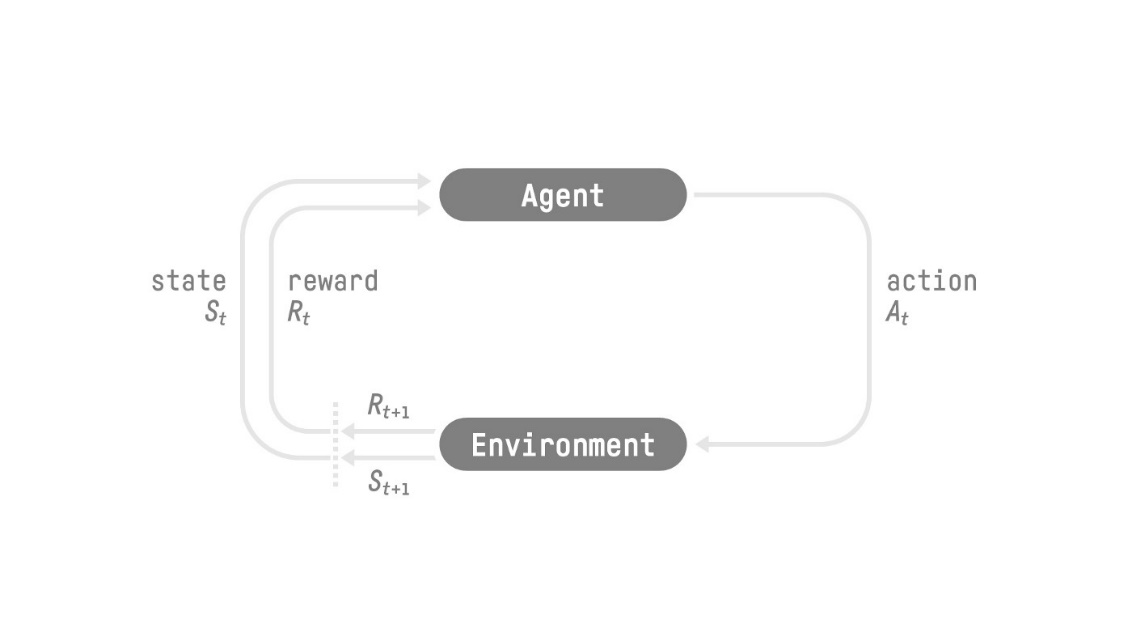


Figure 1RL Process Flow

**Agent**:

* The learner or decision-maker that interacts with the environment. It perceives the environment's state and takes actions to achieve specific goals.

**Environment**:

* The external system with which the agent interacts. It provides feedback to the agent in the form of rewards based on the actions taken.

**State (S)**:

* A representation of the current situation or configuration of the environment. It captures all relevant information needed for decision-making at a particular time step.

**Action (A)**:

* The decision or move that the agent selects from a set of possible actions available in the current state. Actions lead to transitions to new states and impact the environment.

**Reward (R)**:

* Feedback from the environment indicating the consequences of an agent's action. The goal of the agent is typically to maximize cumulative reward over time.

**Policy**:

* A strategy or rule that the agent uses to determine its actions based on the current state. It maps states to actions or specifies a distribution over actions.

**Value Function**:

* Estimates the expected return (cumulative reward) of being in a particular state under a specific policy. It helps the agent evaluate the desirability of different states.

**Q-Value (Action-Value Function)**:

* Estimates the expected return of taking a specific action in a given state and following a specific policy thereafter. It assists in selecting actions based on their long-term outcomes.

**A.4.2 Reinforcement Learning Work Flow**

**Initialization**: The agent starts in an initial state and initializes parameters such as policy or value function.

**Action Selection**: Based on the current state and policy, the agent selects an action to take.

**State Transition**: The environment transitions to a new state based on the chosen action.

**Reward Observation**: The agent receives a reward from the environment based on the action taken and the new state.

**Learning**: The agent updates its policy or value function based on the observed reward and state transition.

**Repeat**: The process repeats over multiple episodes or time steps, gradually improving the agent's decision-making ability.

**A.4.3 About Open AI Gym**

Open AI gym is a tootlkit for developing and testing reinforcement learning algorithms. This python library gives us a huge number of test environments to work on our RL agent’s algorithms with **shared interfaces** for writing general algorithms and testing them.

**A.4.4 Understanding RL Terminologies with OpenAI Gym**

* **Agent**: The learner or decision-maker.
  + In OpenAI Gym: Represented by Python code that interacts with the environment.
* **Environment**: The external system with which the agent interacts.
  + In OpenAI Gym: Defined by various simulation environments (e.g., CartPole, MountainCar).
* **State**: The current situation or configuration.
  + In OpenAI Gym: Accessed via env.reset() to initialize and env.step(action) to observe the next state after taking an action.
* **Action**: Decision or move taken by the agent.
  + In OpenAI Gym: Actions are taken using env.step(action) where action is typically an integer or discrete choice.
* **Reward**: Feedback from the environment indicating success or failure.
  + In OpenAI Gym: Received after each action via step() function, typically a scalar value indicating the goodness of the action.
* **Policy**: Strategy or decision-making rule.
  + In OpenAI Gym: Algorithms or strategies implemented by the agent to select actions based on states.
* **Value Function**: Expected return or utility of being in a particular state.
  + In OpenAI Gym: Algorithms like Q-learning or SARSA update value functions based on observed rewards and transitions.
* **Q-Value (Action-Value Function)**: Expected utility of taking a specific action in a given state.
  + In OpenAI Gym: Often used in algorithms that estimate the value of taking actions in states (e.g., Q-learning).

**A.5 Task to be completed:**

1. Explore any three classic control gym environment
2. Customize the above three environments and demonstrate your observations
3. For the above examples, identify Agent, Environment, State, Action, Reward, Policy, Value

**References**

1. <https://gymnasium.farama.org/>
2. <https://machinelearningmastery.com/principles-of-reinforcement-learning-an-introduction-with-python/?utm_source=drip&utm_medium=email&utm_campaign=MLM+Newsletter+July+13%2C+2024&utm_content=Getting+Started+with+Deep+Learning+%E2%80%A2+Principles+of+Reinforcement+Learning+with+Python>
3. <https://blog.paperspace.com/getting-started-with-openai-gym/>
4. <https://www.youtube.com/playlist?list=PLgNJO2hghbmjlE6cuKMws2ejC54BTAaWV>
5. <https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2>
6. <https://wandb.ai/mukilan/intro_to_gym/reports/A-Gentle-Introduction-to-OpenAI-Gym--VmlldzozMjg5MTA3>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

**(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Portal/MS Teams assignment link at the end of the practical)**

|  |  |
| --- | --- |
| Roll No. C052 | Name: Drumil Kotecha |
| Program: BTI | Division: B |
| Batch: B2 | Date of Experiment: 02/01/2025 |
| Date of Submission: 03/01/25 | Grade : |

**B.1 Tasks given in PART A to be completed here**

*(****Students must write the answers of the task(s) given in the PART A /Students must copy the code, output screenshots here based on the task(s) given in section A.5)***

1. **CartPole Environment**: The agent must balance a pole on a moving cart. We observed the reward structure designed to encourage longer balancing durations and the termination conditions based on angle thresholds or episode steps.
   * *State*: Includes cart position, velocity, pole angle, and angular velocity.
   * *Action*: Discrete (push left or right).
   * *Reward*: +1 for every time step.
2. **MountainCar Environment:** The agent must build momentum to reach the flag at the top of a hill. It was evident that rewards encourage reaching the goal with fewer steps.
   * *State*: Position and velocity of the car.
   * *Action*: Discrete (acceleration left, right, or no acceleration).
   * *Reward*: -1 for each step to encourage faster solutions.
3. **Acrobot Environment**: A two-link pendulum must swing up to reach a goal height. Observing this highlighted the complexity of achieving balance and control in higher-dimensional environments.
   * *State*: Angular positions and velocities of the two links.
   * *Action*: Discrete (apply torque to either joint).
   * *Reward*: -1 for every time step until the goal is reached.

**Terminologies:**

* 1. **Agent**

Definition: The decision-maker or learner in an RL setup. The agent interacts with the environment to achieve specific goals.

Relation to Gymnasium: The agent is represented by Python code that uses Gym’s API to interact with the environment by taking actions and receiving observations and rewards. For example, the code implementing logic like action = env.action\_space.sample() defines the agent's behavior.

* 1. **Environment**

Definition: The system or scenario within which the agent operates. It provides feedback in the form of state transitions and rewards.

Relation to Gymnasium: Gymnasium provides pre-built environments like CartPole-v1, MountainCar-v0, or Acrobot-v1. These environments define the rules, states, actions, and reward structures.

* 1. **State (S)**

Definition: A representation of the environment's current situation. It provides the necessary information for decision-making.

Relation to Gymnasium: The state is accessed using methods like env.reset() (to initialize) or env.step(action) (to observe the next state after an action). For instance, in the CartPole environment, the state includes values like pole angle and cart velocity.

* 1. **Action (A)**

Definition: The decision or move chosen by the agent at any given time step. It leads to a state transition in the environment.

Relation to Gymnasium: Actions are specified using env.step(action), where the action is typically an integer corresponding to a discrete or continuous space defined by the environment’s action\_space. For example, in MountainCar, actions might be ‘accelerate left’, ‘accelerate right’, or ‘no acceleration’.

* 1. **Reward (R)**

Definition: Feedback provided by the environment indicating the quality of the agent’s action. The agent's objective is to maximize cumulative rewards.

Relation to Gymnasium: The reward is received as part of the tuple from the env.step(action) function. For example, in the CartPole environment, a reward of +1 is given for each time step the pole remains balanced.

* 1. **Policy**

Definition: A strategy that maps states to actions. It guides the agent’s decision-making process.

Relation to Gymnasium: The policy is implemented as part of the agent's logic, which could be random (action\_space.sample()) or learned through algorithms like Q-learning.

* 1. **Value Function**

Definition: Estimates the expected return (cumulative future rewards) from a state under a specific policy.

Relation to Gymnasium: While Gymnasium does not directly implement value functions, RL algorithms applied to Gym environments often estimate value functions to optimize the agent’s performance.

* 1. **Q-Value (Action-Value Function)**

Definition: Estimates the expected return of taking a specific action in a given state under a specific policy.

Relation to Gymnasium: Q-values are estimated using algorithms like Q-learning or Deep Q-Networks (DQN) applied to Gym environments. These values help in determining the optimal action for each state.

**Code:**

**CartPole**

import gymnasium as gym

environment = gym.make("CartPole-v1", render\_mode="human")

state, \_ = environment.reset()

total\_steps = 100

for step in range(total\_steps):

    environment.render()  # Display the environment

    random\_action = environment.action\_space.sample()  # Select a random action

    next\_state, reward, done, truncated, info = environment.step(random\_action)  # Apply the action

    if done or truncated:

        print(f"Episode ended at step {step + 1}. Resetting environment.")

        state, \_ = environment.reset()  # Reset the environment

environment.close()

**A number of steps with black text

Description automatically generated with medium confidence**

**MountainCar**

car\_env = gym.make("MountainCar-v0", render\_mode="human")

current\_state, \_ = car\_env.reset()

max\_steps = 200

for step in range(max\_steps):

car\_env.render() # Display the environment

action\_taken = car\_env.action\_space.sample()

next\_state, reward, done, truncated, info = car\_env.step(action\_taken)

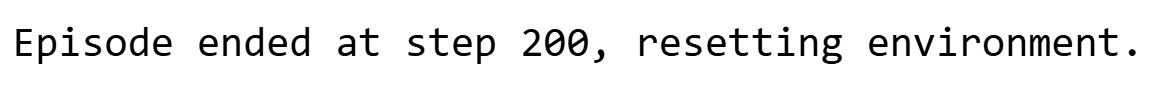
if done or truncated:

print(f"Episode ended at step {step + 1}, resetting environment.")

current\_state, \_ = car\_env.reset()

# Clean up and close the environment window

car\_env.close()

****

**A black and white line with a black and white line

Description automatically generated with medium confidence**

**Acrobot**

acrobot\_env = gym.make("Acrobot-v1", render\_mode="human")

initial\_state, \_ = acrobot\_env.reset()

total\_steps = 200

for step in range(total\_steps):

acrobot\_env.render() # Display the environment

random\_action = acrobot\_env.action\_space.sample()

next\_state, reward, done, truncated, info = acrobot\_env.step(random\_action)

if done or truncated:

print(f"Episode finished at step {step + 1}, resetting environment.")

initial\_state, \_ = acrobot\_env.reset()

# Close the environment and clean up

acrobot\_env.close()

**A screenshot of a computer

Description automatically generated**

**Modified Parameters of CartPole:**

cartpole\_env = gym.make(

"CartPole-v1",

render\_mode="human",

max\_episode\_steps=500, # Limit steps per episode

)

# Reset the environment and get the initial state

current\_state, \_ = cartpole\_env.reset()

# Run for a fixed number of steps (e.g., 300)

for step\_num in range(300):

cartpole\_env.render() # Display the environment

# Extract the pole angle from the current state

angle\_of\_pole = current\_state[2]

# Decide the action based on the pole angle

selected\_action = 1 if angle\_of\_pole > 0 else 0

# Execute the action and get feedback from the environment

current\_state, reward, is\_done, is\_truncated, info = cartpole\_env.step(selected\_action)

# Print the current step, state, reward, and termination status

print(f"Step: {step\_num}, State: {current\_state}, Reward: {reward}, Done: {is\_done}, Truncated: {is\_truncated}")

# Check if the episode is over and reset if necessary

if is\_done or is\_truncated:

print("Episode finished, resetting environment.")

current\_state, \_ = cartpole\_env.reset()

# Close the environment when done

cartpole\_env.close()

**A table of numbers and letters

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated**

**B.2 Observations and Learning:**

*(****Students must write the observations and learning based on their understanding built about the subject matter and inferences drawn)***

I am now able to comprehend the fundamental concept of RL terminologies and elements, implement basic agent-environment interactions, such as taking actions and receiving rewards, implement an agent that interacts with the environment by selecting actions based on states and understand how to interpret state transitions and rewards within the context of a specific RL problem.

**B.3 Conclusion:**

*(****Students must write the conclusive statements as per the actual attainment of individual outcomes listed above and learning/observation noted in section B.2)***

## Clear understanding of RL terminologies, can implement basic agent-environment interactions, and interpret state transitions and rewards in the context of RL problems.