Indian Institute of Engineering Science & Technology, Shibpur
Department of Computer Science & Technology
8th Semester Artificial Intelligence Laboratory 2025
CS 4271
ASSIGNMENT – 5

1. Introduction and Background

This lab assignment challenges you to simulate a robotic agent learning to navigate a 2D environment using both Q-learning and Deep Q-learning approaches. You will be required to model the environment, train both types of agents, compare their performance, and visualize the learned policies. The task is designed to mimic practical AI applications while reinforcing your understanding of reinforcement-based decision-making.

---

2. Problem Statement

You are required to simulate a grid-based environment that represents a bounded two-dimensional space in which a robotic agent is placed. The goal of the robot is to autonomously learn how to move from a predefined **start position** to a **goal location** while avoiding obstacles scattered throughout the grid. The environment should be initialized as a discrete `n x n` grid (recommended size: 10×10 or higher), where each cell can represent one of the following:

- **Start cell (S)**: The initial position of the robot
- **Goal cell (G)**: The target location the robot must reach
- **Obstacle (X)**: Cells that are non-traversable
- **Free space (.)**: Open cells the robot can move to

The agent is allowed to perform the following four actions: **UP**, **DOWN**, **LEFT**, and **RIGHT**, as long as these actions do not result in moving out of the grid or into an obstacle.

You must implement two separate learning agents:

1. **A Q-learning agent** that uses a traditional tabular approach with a state-action value table.
2. **A Deep Q-learning agent (DQN)** that approximates Q-values using a neural network.

Each agent should learn to maximize cumulative reward over multiple episodes of interaction with the environment.

To simulate real-world constraints, at least **20% of the grid cells should be randomly assigned as obstacles**, and **dynamic changes** to the environment (e.g., adding or removing obstacles during training) may optionally be introduced as a bonus task to test adaptability.

---

3. Reward Model

The reward mechanism should incentivize goal-oriented behavior and penalize undesirable actions. The following reward scheme is suggested:

- **+100**: When the agent successfully reaches the goal
- **-10**: If the agent attempts to move towards an obstacle
- **-1**: For each step taken that does not result in reaching to the goal state (to discourage often random wandering)

- Choose the next state using the epsilon greedy algorithm.

The episode must terminate under either of the following conditions:

- The agent reaches the goal
- The agent exceeds a maximum number of allowed steps (e.g., 200 steps)

---

4. Implementation Guidelines

You are expected to complete the assignment in **Python**, making use of standard libraries and tools such as `NumPy`, `Matplotlib`, `TensorFlow` or `PyTorch`.

The implementation must include:

(a) Environment Design

- Programmatically generate the grid world with randomly placed obstacles.
- Clearly indicate the start and goal cells.
- Display the environment graphically (using `matplotlib` or `pygame`).

(b) Q-learning Agent

- Implement a discrete Q-table where each state-action pair is updated using the Q-learning formula.
- Utilize ε-greedy policy for action selection.
- Train the agent over multiple episodes and visualize the final learned path.

(c) Deep Q-learning Agent

- Replace the Q-table with a neural network model.
- Include the use of **experience replay** and **target networks** for stability.
- Normalize state inputs and define suitable loss and optimization functions.
- Train over multiple episodes and visualize the policy after training.

(d) Performance Comparison

- Plot and compare **reward per episode**, **success rate**, and **convergence time** between the two approaches.
- Discuss the trade-offs in terms of **training time**, **generalization**, and **scalability**.

(e) (Optional/Bonus):

- Implement **a GUI-based control panel** for adjusting grid size and obstacle density.

---

5. Sample Grid World Illustration

Below is a sample `10x10` grid environment:

```
S . . . X . . . . .
. X X . . . X . X .
. . . . . X . . . .
X . X X . . . X . X
. . . . . X . . . .
. X . . . . X . X .
X . . X . . . . . .
. X . X . X X X . .
. . . . X . . . X .
. X . . . . . . . G
```

- The agent (S) must learn to reach the goal (G) avoiding all obstacle cells (X).
- The optimal path will be determined by the agent's training using Q-learning and DQN respectively.

---

6. Submission Guidelines

Each student must submit the following:

1. **Code Files**: Well-documented `.py` or `.ipynb` files for both Q-learning and DQN implementations.
2. **Technical Report (PDF)**:
    - Description of approach and algorithms
    - Environment design and setup
    - Hyperparameters used
    - Learning curve plots
    - Screenshots of trained agent paths
    - Comparative performance summary
    - Reflections on challenges, limitations, and scope for future improvements

All files must be submitted via Google Classroom.