Programming Assignment 2

DA5007: Special Topics in ML (Reinforcement Learning)

1 Problem Statement

In this assignment, you will implement and compare two popular Temporal Difference Learning (TD-Learning) algorithms, i.e. **SARASA** and **Q-Learning**.

You will implement both algorithms across various versions of the Grid World environment and explore how on-policy (SARSA) and off-policy (Q-Learning) TD control methods learn optimal policies through interaction with the environment.

Environment

A classic Grid World environment is commonly used in Reinforcement Learning (RL) to study and compare algorithms. We will use a few variants of the Grid World environment to explore the behaviour of two TD-Learning algorithms, i.e. **SARSA** and **Q-Learning**.

The variants of the grid world environment used in this assignment consist of several types of states, each serving a distinct purpose. These states are:

- *Start state:* The agent begins here.
- *Goal states:* The agent's objective is to reach these states, and the episode will terminate here.
- *Obstructed states:* These are walls that block entry; attempting to move into them results in no movement.
- *Bad states:* Entering these states results in a higher penalty than usual.
- *Restart states:* Entering these incurs a severe penalty and causes the agent to teleport back to the start state on the next step without ending the episode.
- *Normal states:* Any other states that impose a small penalty upon entry.

The agent can take one of four actions: **UP**, **DOWN**, **LEFT**, or **RIGHT**. The agent moves to the next state based on the direction of the selected action with a probability $p \in [0, 1]$ (specified in the environment code as the parameter ***transition_prob***). Additionally, a bias parameter $b \in [0, 1]$ is defined (set to $b = 0.5$ for this assignment).

Consider the direction of the action chosen as the agent's "*North*". For example, if the action is '*left*', it is the agent's North, and the agent's East would be the direction of the action '*up*'. Figure 1 provides an illustration of the same. The agent transitions to the state West of the chosen action with probability $(1 - p) \times b$, and to the East of the chosen action with probability $(1 - p) \times (1 - b)$.

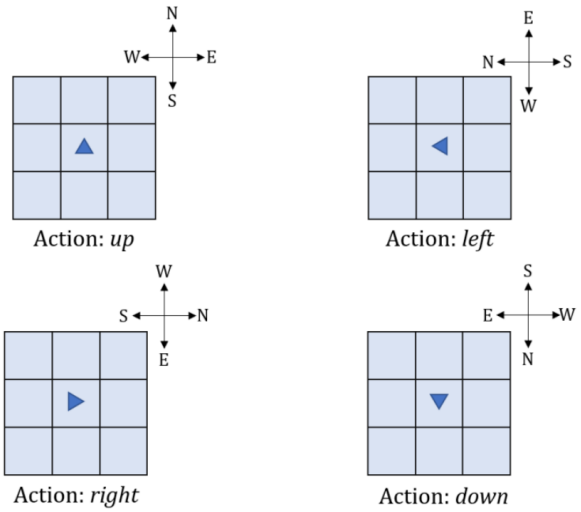

Figure 1:

The environment may also have a wind blowing that can push the agent one additional cell to the right after transitioning to the new state with a probability of 0.4. An episode is terminated either when a goal is reached or when the timesteps exceed 100. Transitions that take you off the grid will not result in any change in state.

Reward Structure:

State Type	**Reward**
Normal States	-1
Goal States	10
Bad States	-6
Restart States	-100

10×10 Grid World

The environment consists of a **10×10 Grid World**, where an **agent** learns to navigate from a **fixed start state** to one of several **goal states** (in this assignment, there are **three goal states**). Along its path, the agent must **avoid hazardous states**, such as **bad states**, **restart states**, and **obstructed cells**, each of which introduces penalties or special transitions within the environment.

This Grid World can operate under both **deterministic** and **stochastic** dynamics, controlled by the parameter ***transition_prob*** during initialization. Additionally, **wind effects** can be enabled using the ***wind*** parameter, which adds randomness to the agent's movement, simulating environmental disturbances. The agent's **starting position** can also be modified through the ***start_state*** parameter to study how different initial positions influence learning behavior.

In this assignment, you will use these configuration parameters to conduct multiple experiments with **SARSA** and **Q-Learning** algorithms. Your goal is to compare their learning performance under varying environmental conditions.

Refer to Figure 2 for a visualisation of the environment.

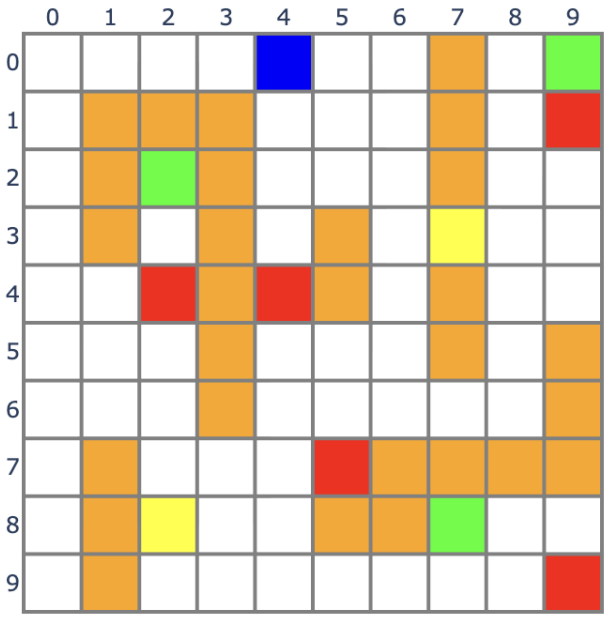

Figure 2: 10×10 **Grid World**: The *Blue* cell indicates the Start state, *Green* cells denote the Goal states, *Red* represents the Bad state, *Yellow* corresponds to the Restart states, and *Orange* highlights the Obstructed regions

Four Room Grid World

The **Four-Room Environment** is a classic **Grid World** setup that simulates a two-dimensional environment divided into four interconnected rooms. The grid contains **walls (obstructions)**, **goal states**, and special **terminal** or **restart states** that significantly influence the agent's learning process.

The environment is defined on a **9×9** grid, where each cell represents a unique state. The agent begins from a predefined **start state**. The grid is partitioned into four rooms by a set of fixed obstructions (walls) that the agent cannot cross. These walls create a maze-like layout, requiring the agent to navigate through doorways to reach a goal state.

In this assignment, the environment is configured to be **deterministic** by setting the parameter ***transition_prob = 1*** during initialization, and the **wind effect** is disabled. This ensures that the agent's actions produce consistent transitions without stochastic disturbances.

The environment also supports a **dynamic goal feature**, where the goal location changes randomly after each episode. This design encourages the agent to continuously adapt its policy instead of converging to a fixed goal. The behavior can be activated by setting the argument ***goal_change = True*** when initializing the environment.

In this assignment, you will utilize the ***goal_change*** configuration to conduct a series of experiments using the **SARSA** and **Q-Learning** algorithms. The objective is to analyze and compare how both algorithms adapt their policies when the goal location changes dynamically across episodes.

Refer to Figure 3 for a visualisation of the environment.

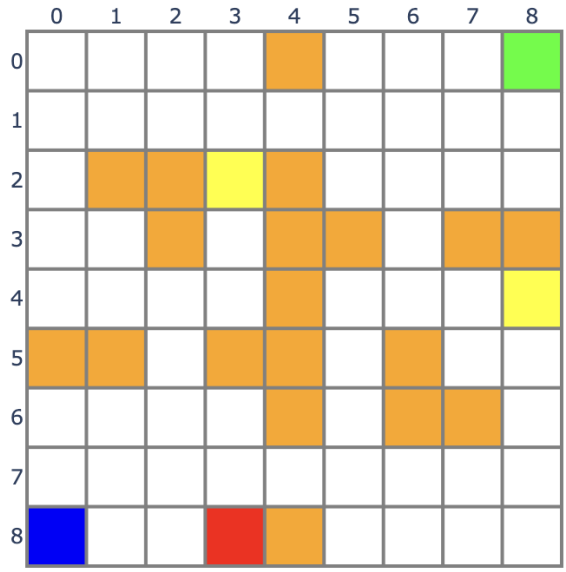

Figure 3: **Four Room Grid World:** The *Blue* cell indicates the Start state, *Green* cells denote the Goal states, *Red* represents the Bad state, *Yellow* corresponds to the Restart states, and *Orange* highlights the Obstructed regions

Environment Source Code

Click here to access the starter code, which can be used to instantiate the environment.

Use the functions ***create_standard_grid*** and ***create_four_room*** from the *env.py* file to create the respective environments. An example of initialising and creating the environments is provided below:

```
from env import create_standard_grid, create_four_room

# Create a standard grid environment
standard_grid_env = create_standard_grid(
                        start_state=np.array([[0,4]]),
                        wind=False
)
# Refer to the source code for all available arguments that can be
    passed.

# Create a four-room environment
four_room_env = create_four_room()
# Additional arguments can also be provided to this function as needed
    .
```

The *env.step*() function takes as arguments the current state and action, and returns the reward and next state. The appropriate termination conditions have to be specified by the student in the code. *env.reset*() function resets the environment.

```
from env import create_standard_grid, create_four_room

# Helper code to understand the env.step() and env.reset() functions
```

```
#Initiate the environment
standard_grid_env = create_standard_grid()

#Now since action allowed are Up, Down, Left, Right.
#These action are represented as 0,1,2,3 in the environment

#To get the start state
standard_grid_env.start_state_seq  # Outputs start state
standard_grid_env.goal_states_seq # Outputs list of goal states
standard_grid_env.num_states #Outputs total number of states
standard_grid_env.num_actions #Outputs total number of actions

#Now let's use step function in environment class
standard_grid_env.step(standard_grid_env.start_state_seq, 3)
#This is how step() function is used.
#Experiment with source code to understand what the action corresponds
    to action 3.

#Once episode is over, you can reset the environment to its initial
   condition with reset() function

standared_grid_env.reset()
```

Please don't modify the environment code without prior permission from the instructor. If you are unable to access the link directly, try copying and pasting it into your browser's address bar. https://github.com/rohitrk06/Grid-World-Environment

2 Tasks

Implement the SARSA and Q-Learning Algorithm (25 Marks)

You will highlight the major difference among the two algorithms in your report. Along with the snapshot of your implementation.

Experimentation and Hyperparameter tuning (65 Marks)

After implementing the **SARSA** and **Q-Learning** algorithms, you will conduct a series of experiments across multiple environment configurations.

Configurations

- 10×10 **Grid World**
  1. Q-learning:
     - ***transition_prob*** $= 0.7$ or 1.0
     - ***start_state*** $= (0, 4)$ or $(3, 6)$
     - ***exploration strategy*** $= \epsilon$-greedy or Softmax
     - ***wind*** $=$ False

     Total Configuration: 8
  2. SARSA:

- ***wind*** = True or False
- ***start_state*** = $(0, 4)$ or $(3, 6)$
- ***exploration strategy*** = ϵ-greedy or Softmax
- ***transition_prob*** = 1.0

Total Configuration: 8

- **Four Room Grid World**

1. Q-learning:
   - ***goal_change*** = True or False
   - Keep other configurations as default configurations as defined in the source code.

   Total Configuration: 2
2. SARSA:
   - ***goal_change*** = True or False
   - Keep other configurations as default configurations as defined in the source code.

   Total Configuration: 2

You would run experiments for total 20 configurations.

PART A

For each configuration, you are required to perform **hyperparameter tuning** to identify the optimal set of parameters, including the exploration rate (ϵ) in ϵ-greedy exploration, the temperature (τ) in softmax exploration, the learning rate (α), and the discount factor (γ). *Note: To obtain reliable results and mitigate the impact of randomness, each algorithm should be executed with at least five different random seed values. The best hyperparameters should be selected based on the averaged performance across these runs.*

For each algorithm and configuration of the Grid World environment, experiment with the following values for the key hyperparameters:

Learning Rate (α):

$$\alpha \in \{0.001, 0.01, 0.1, 1.0\}$$

Discount Factor (γ):

$$\gamma \in \{0.7, 0.8, 0.9, 1.0\}$$

Exploration Strategy: Use one of the following:

- **ϵ-greedy**:

$$\epsilon \in \{0.001, 0.01, 0.05, 0.1\}$$

- **Softmax Exploration**:

$$\tau \in \{0.01, 0.1, 1, 2\}$$

You are encouraged to be **creative** in how you visualize and present your hyperparameter tuning results. Rather than simply reporting numerical values, consider using meaningful plots and comparative visualizations to justify your hyperparameter choices.

To facilitate this, you may explore tools such as ***Weights & Biases (wandb)*** or ***MLflow***. Both of these tools provide powerful experiment tracking, visualization, and hyperparameter

optimization capabilities. They allow you to log training metrics (e.g., episode rewards, losses, convergence curves) and automatically generate dashboards to compare the performance of different runs or hyperparameter configurations.

It is strongly recommended that you use one of these tools for your experiments and include the generated plots or tables in your report to support your analysis and choice of hyperparameters. Clear, well-justified visualizations will be given additional credit.

You can easily find tutorials and documentation on how to use these tools online, which will help you integrate them into your assignment effectively.

PART B

Using the best-performing hyperparameters, conduct the following analyses and visualizations:

1. **Training Curves (averaged over 100 runs):**
   - Plot the **average reward per episode** over time.
   - Plot the **average number of steps to reach the goal** per episode.
2. **State Visit Heatmap:**
   - Generate a heatmap of the grid showing the **average number of visits to each state**, aggregated over all runs during training.
3. **Q-Value Heatmap and Optimal Policy:**
   - After training, visualize a heatmap showing the **average maximum Q-value** for each state across runs.
   - Overlay the **optimal action (policy)** for each state, derived from the learned Q-values.

Make sure to clearly annotate all plots and explain the trends observed. These visualizations should help assess both the learning efficiency and exploration strategy of the agent.

Report Presentation(10 marks)

This section assesses the overall presentation quality of your report, including clarity, structure, coherence, and visual appeal. A well-presented report should not only summarize your experimental work but also clearly communicate the key insights, patterns, and conclusions derived from your analysis.

Note: LLM generated insights without strong grounding in plots will be penalized.

3 Submission Instructions

- You are required to create a comprehensive PDF report containing key code snippets, observations, insights, and plots.

- Your submission may include either `.ipynb` (Jupyter Notebook) files or `.py` (Python script) files. Ensure that your code is **well-documented** and easy to understand. Each submitted file should contain sufficient explanations, comments, and instructions so that anyone can reproduce your experiments and results without additional clarification. **We will run your code, which must generate the same plots and charts as shown in the report.**
- In your report, include only the important code snippets rather than every line of code.
- Zip your report and code files together and submit the zip file through the portal.
- Submission of both the report and code files is mandatory. Incomplete submissions will not be evaluated and will receive 0 marks.
- Use clear and meaningful code comments for readability, and follow all problem specifications carefully.
- It is recommended to include a brief description at the beginning of your code or notebook explaining the overall structure, dependencies, and execution steps.