AIL 722: Assignment 2
Semester I, 2025-26

Due: October 6, 2025 12:00 PM

Instructions:

- This assignment has two parts.
- You should submit all your code (including any pre-processing scripts you wrote) and any graphs you might plot.
- Include **a write-up (pdf) file**, which includes a brief description for each question explaining what you did. Include any observations and/or plots required by the question in this single write-up file.
- Please use Python for implementation using only standard python libraries. Do not use any third-party libraries/implementations for algorithms.
- Your code should have appropriate documentation for readability.
- You will be graded based on what you have submitted as well as your ability to explain your code.
- Refer to the course website for assignment submission instructions.
- This assignment is supposed to be done individually. You should carry out all the implementation by yourself.
- We plan to run Moss on the submissions. We will also include submissions from previous years since some of the questions may be repeated. Any cheating will result in a zero on the assignment, a penalty of -10 points and possibly much stricter penalties (including a **fail grade** and/or a **DISCO**).
- Starter code is available at this link.

1 Temporal Difference [50 points]

1.1 Cliff Environment

For this part, you will be working with a custom grid-world environment called the Multi-Goal Cliff. This environment is designed to test an agent's ability to navigate complex scenarios involving risk, reward, and conditional objectives. Runtime for each experiment in this section should not take more than 15 minutes.

The world is a 10×20 grid. The agent begins at the bottom-left corner and must navigate to a goal state.

1.1.1 Key Features of the Environment

- **The Cliff**: The final row of the grid is a dangerous cliff. Stepping on any cliff cell (except for the start and goals i.e only red cells) will result in a significant penalty and immediately end the episode.
- **Stochasticity**: The world is not entirely deterministic.
  - The row just above the cliff (light blue) is a **slippery slope**. While on this row, there is a 20% chance that the agent will slip and take a random action instead of the intended one.
  - Certain cells on the map contain **Wine**. Landing on a wine cell incurs a small penalty, and there is a 50% chance the agent's next action will be randomized.

- **Goals and Checkpoints**: There are two treasure goals, but they can only be accessed if specific conditions are met.
  - **Checkpoints**: Two checkpoints, light blue and light green circles, are located on the map. The agent must visit them to "unlock" the goals.
  - **The Safe Goal**: Located midway along the cliff edge. It offers a moderate reward, but the agent must have visited **Light Blue Checkpoint** to claim it.
  - **The Risky Goal**: Located at the far-right corner of the cliff edge. It offers a very high reward, but the agent must have visited **both Checkpoints** to succeed.

The agent receives a small negative reward for every step it takes to encourage efficiency. Your task is to develop learning algorithms that can discover policies to successfully navigate this environment. The MDP is: Action Space (UP, DOWN, LEFT, RIGHT), transitions and reward are handled by environment, state space is (ROWS * COLS * isFirstCheckpointVisited * isSecondCheckpointVisited).

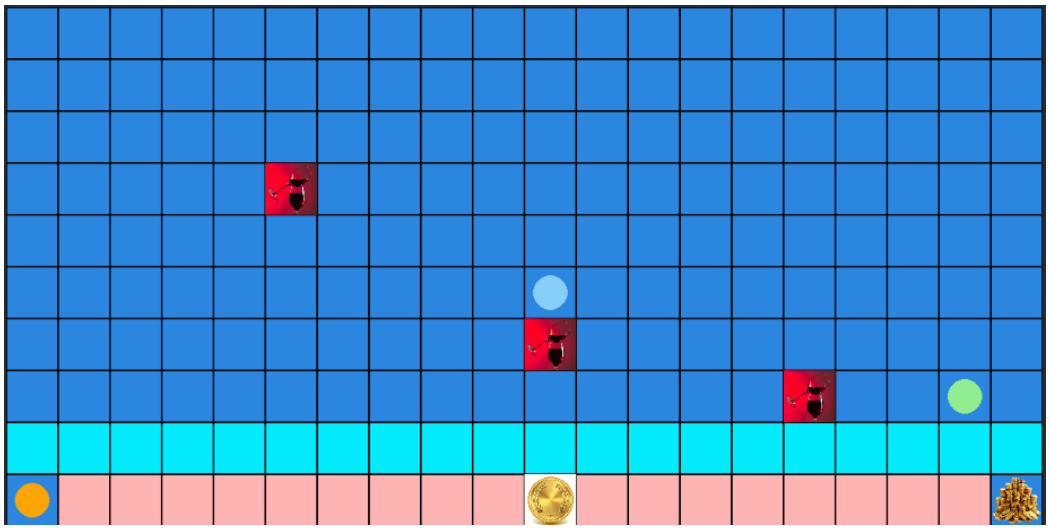

Figure 1: A visual representation of the Multi-Goal Cliff environment. The agent (golden circle) must navigate from the start to one of the goals while managing checkpoints and avoiding the cliff.

1.1.2 Assignment Tasks

You are required to implement and compare three reinforcement learning algorithms on this environment: QLearning, Sarsa, Expected Sarsa. You are expected to modify the agents.py file to implement your algorithms, do not change the function signature and return types as it will be auto graded. You may create separate files for analysis and plotting. Also, make sure to use the maximum episode length as 1000. For each of the three algorithms, you must perform the following tasks:

1. **Train over Multiple Seeds**: Train each agent across **10 different random seeds**. (i.e., 10 separate training runs per algorithm).
2. **Plot Average Learning Curves**: For each algorithm, average the episodic rewards across the 10 seeds i.e say you run algorithm for X episodes, then at the end of the training you will have rewards of shape (10, X) where each row represents episodic rewards for single seed, then average across all the seeds resulting in (X,) array. Plot these averaged rewards of all three algorithms. This will show the average learning progress over time. You must save it in the following names in the plots folder given in the Starter Code
   - *qlearning_plot.png*
   - *sarsa_plot.png*
   - *expected_sarsa_plot.png*

Algorithm 1 Expected Sarsa

Initialize $Q(s, a)$ arbitrarily for all s,a
loop {over episodes}
  Initialize s
  repeat {for each step in the episode}
    choose a from s using policy π derived from Q
    take action a, observe r and s'
    $V_{s'} = \sum_a \pi(s', a) \cdot Q(s', a)$
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma V_{s'} - Q(s, a) \right]$
    $s \leftarrow s'$
  until s is terminal
end loop

Figure 2: Expected Sarsa Algorithm, Reference

3. **Plot Average Goal Visits**: Create a grouped bar chart for all the algorithms in single plot that shows the average number of times the agents reached the "Safe Goal" versus the "Risky Goal" during training, averaged across 10 seeds. This will illustrate the different strategies learned by each algorithm. Again store it in the same plots folder. You must save it as a single image named
   - *average_goal_visits.png*
4. **Perform Evaluation**: Take the "best Q-table" for each of the three algorithms and run a evaluation for 100 episodes. Ensure you reset the environment with a new seed for each episode (e.g., `env.reset(seed=i)` where `i` is the evaluation episode number). Since you will be training across 10 seeds for each algorithm, you need to come up with your definition of "best".
5. **Report Results**: Report the final performance in a json file *cliff_evaluation_results.json* present in the starter code, and add **mean and standard deviation** of the rewards obtained during the 100-episode evaluation for each of the three best policies.
6. **Generate GIFs**: Create one GIF for each of the three best policies, visualizing the agent's path from start to finish in the environment. You must save gifs in the following format in the gifs folder in Starter Code
   - *expected_sarsa.gif*
   - *qlearning.gif*
   - *sarsa.gif*

1.2 Frozen Lake Variant

For this part, you will implement and compare two reinforcement learning algorithms on a custom Frozen Lake environment.

1.2.1 Markov Decision Process (MDP) Definition

The environment can be formally described as an MDP, defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$.

- **States ($\mathcal{S}$)**: The state space is the set of all grid cells on the $N \times N$ map, where N is the 'map_size'. For a default map size of 16, there are 256 states. The agent starts from some cell and aims to reach the bottom-right.
- **Actions ($\mathcal{A}$)**: From any state, the agent can choose from a discrete set of 3 actions:

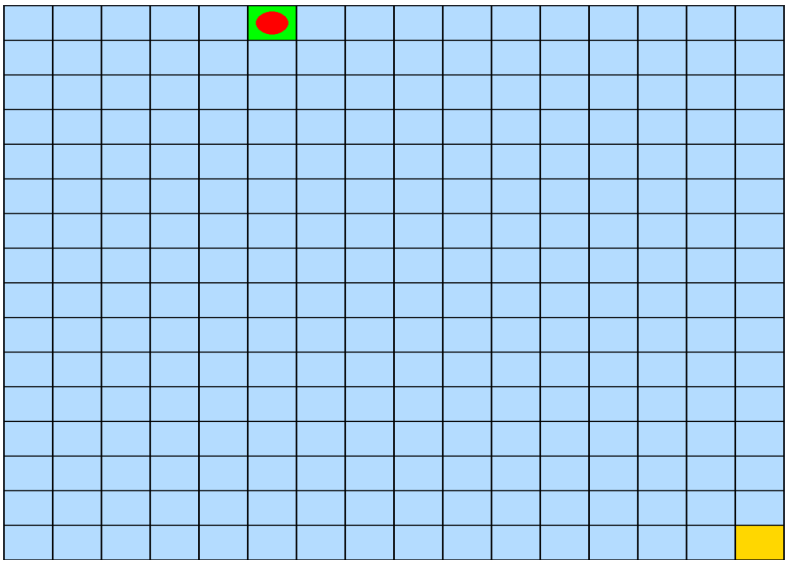

Figure 3: Expected Sarsa Algorithm

  - 0: **Down**
  - 1: **Down-Left**
  - 2: **Down-Right**

  The agent cannot move up, meaning the episodes are guaranteed to terminate.

- **Transition Dynamics ($\mathcal{P}$)**: The transitions are deterministic.
- **Rewards ($\mathcal{R}$)**: The reward function is defined as follows:
  - $R(s, a, s') = +1.0$ if s' is the Goal state (yellow square) 'G'.
  - $R(s, a, s') = -1.0$ if the agent moves into the final row without reaching the Goal.
  - $R(s, a, s') = -0.1/\sqrt{N}$ is an additional penalty applied if the agent moves down-right. This discourages a policy that simply moves towards the goal.
  - $R(s, a, s') = 0$ for all other transitions.
- **Episode Termination**: An episode ends if the agent reaches the Goal state or if it lands on any cell in the final row.

1.2.2 Assignment Tasks

You are required to implement the following things:

1. **Implementation:** Implement and compare two reinforcement learning algorithms on this environment: Monte Carlo On-policy and QLearning. You are expected to modify the agents.py file to implement your algorithms, do not change the function signature and return types as it will be auto graded. You may create additional files for plots and analysis.

2. **Training:** For both algorithms, you should run the training for a single seed and cap the number of steps per episode at 100. After training, plot the episodic rewards for both the algorithms with the following names and them in the plots folder.

   (a) $frozenlake\_mc\_(START\_STATE).png$, e.g $frozenlake\_mc\_(0, 3).png$

   (b) $frozenlake\_qlearning\_(START\_STATE).png$

3. **Evalutation** For each algorithm, run an evaluation for 100 episodes (again, use different seed for different episodes) and report mean and std of rewards in the file $frozenlake\_variant\_evaluation\_results.json$.

4. **Generate GIF** Finally, create one gif for both the algorithms. You must output the gif in the following format. Also, ($START\_STATE$) is just the placeholder, you need to replace with actual start states you will be working with. Add the gifs in the starter code, gifs folder.

   (a) $frozenlake\_mc\_(START\_STATE).gif$

   (b) $frozenlake\_qlearning\_(START\_STATE).gif$

You need to perform above steps for 2 starting positions of agents: (0, 3) and (0, 5)

2 Importance Sampling [50 points]

2.1 Task Overview

This problem explores off-policy reinforcement learning using two different importance sampling approaches:

- Off-policy TD[0] control with **importance sampling**
- Off-Policy Monte Carlo Control with **weighted Importance Sampling**

The task is to implement **both** algorithms in a **'gym_gridworlds'** environment and analyze their performance through visualizations. Runtime for each experiment in this section should not take more than 15 minutes.

2.2 Environment

The environment consists of a blue agent in a grid of square colored tiles. Black tiles are empty and the agent can traverse through them unhindered. Green tiles have positive reward. Higher intensity green (top right corner) indicates higher positive reward. For this problem, the green tile with highest intensity is the goal state. Red tiles give negative rewards. Yellow tiles are quicksands, where executed action will fail with a probability of 0.9 and agent will land up in random cell. Black tiles with arrows are tiles from which the agent can transition only in the direction of the arrow.

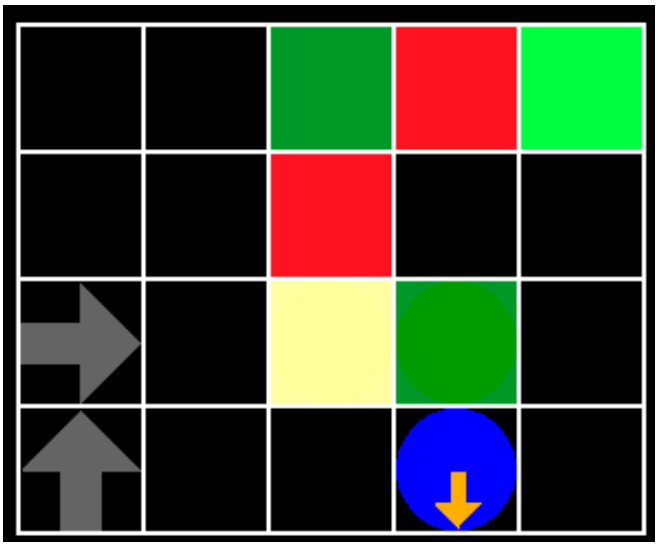

Figure 4: Gym-Gridworlds (Full 4x5 v0)

The observation is discrete in the range $\{0, 1, 2, \ldots, (n\_rows \times n\_cols) - 1\}$. Each integer denotes the current location of the agent. The rewards obtained for different action conditions are listed below:

2.3 Assignment Tasks

For both Temporal Difference and Monte Carlo methods:

1. **Algorithm Implementation**: Implement both off-policy algorithms:

Condition	Reward
Stay at Goal (highest intensity green)	+2.0
Stay at Distracting goal (low intensity green)	+0.1
Any action from penalty tiles	−5.0
Any action in small penalty tiles	−0.1

Table 1: Reward structure of the environment

(a) Off-policy TD[0] control with importance sampling

(b) Off-Policy Monte Carlo Control with weighted importance sampling

You must use the behavior policy provided in the starter code. Behavior policy takes a noise parameter as input. Run the following parts for three given noise values: $[0.0, 0.1, 0.01]$. Modify the following files:

- `A2/Q2/gym_gridworlds/gym_gridworlds/monte_carlo_starter.py`
- `A2/Q2/gym_gridworlds/gym_gridworlds/td_starter.py`

2. **Training Phase**: For both 'Off-policy TD[0] with importance sampling' and 'Off Policy Monte Carolo Control with weighted importance sampling', run the training loop for **10 different random seeds**. Then select and return the best learned Q table for each algorithm.

3. **Evaluation Phase**: For your best chosen Q-table run evaluation for 100 different episodes, for both algorithms and report mean and std in the file *importance_sampling_evaluation_results.json*

4. **Learning Analysis**: Plot the **reward curves vs. number of episodes** during training for both algorithms, averaged across the 10 seeds. Add the generated plots in the **starterCode/Q2/plots** folder

   (a) *monte_carlo_reward_curve_*$(NOISE)$*.png*, e.g *monte_carlo_reward_curve_*(0.0)*.png*

   (b) *temporal_differene_reward_curve_*$(NOISE)$*.png*

5. **Policy Demonstration**: Generate a **GIF of the best policy** for each algorithm. Include the best GIFs in your submission for both policies. Add the gifs in the **starterCode/Q2/gifs** folder

   (a) *monte_carlo_gif_*$(NOISE)$*.png*

   (b) *temporal_differene_gif_*$(NOISE)$*.png*