Figure 1: University of Arkansas Logo

CSCE 46103/56103: Introduction to Artificial Intelligence

Homework #3: Game Tree and MDP, Reinforcement Learning”

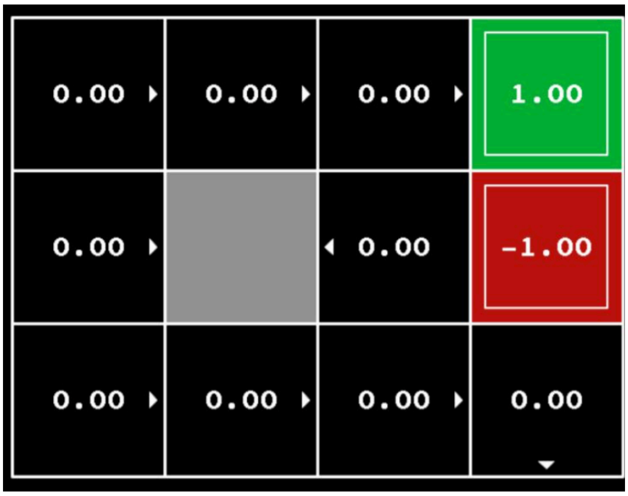
Submission Deadline: 11:59 PM, Octoberr 17^{th}, 2025

Instructions

- **Written Format & Template:** Students can use either Google Doc or LaTeXfor writing the report.
- Write your full name, email address, and student ID in the report.
- Submission through BlackBoard.
- **Submissions:** Your submission should be a zip file containing your report with the screenshots of your outputs, and the source code including your implementation.
- **Name the zip file as `lastname_studentID.zip`**
- submission should be made via black board.
- **Policy:** Review the late days policy and include the total number of “Late Days” used in your report.

I. Problem 1: Value Iteration[20pts]

Below are utilities/values from selected iterations of Value Iteration for a 4×3 GridWorld.



You are given a **4×3 GridWorld** where the agent can move in four directions: **up, down, left,** and **right**. The movement is **stochastic**, meaning that the agent moves in the intended direction with **60% probability**, but with a **20% chance it moves to the left** and a **20% chance it moves to the right** of the intended direction.

The **discount factor** is $\gamma = 0.9$. The terminal states are:

- **(4,3)** with a reward of **+1**
- **(4,2)** with a reward of **−1**

The gray cell in the grid is a **wall** (inaccessible), and all other non-terminal states give a reward of **0** at each time step.

Your Task

Begin your calculations from the state **(3,3)**. At each iteration, update the value using the **Bellman update equation**:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s') \cdot [R(s,a,s') + \gamma \cdot V_k(s')]$$

Update the values of all non-terminal, non-wall states at the first and second iterations (Iteration #1, Iteration #2). Knowing that value of all states are initialized to **0** in Iteration #0.

Note: You are required to show your intermediate calculations.

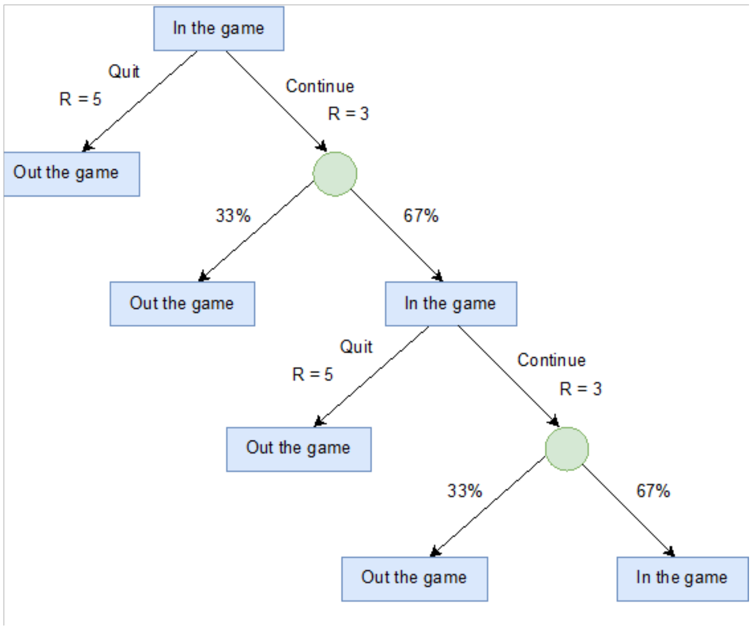


Figure 2: Game Tree for the Decision Process

II. Problem:2 [Game Tree for the Decision Process] (20)

You are given a decision process where the agent can be in one of two states: **In the game** or **Out of the game**. From the **In the game** state, the agent can take one of two actions: **Quit** or **Continue**.

The transitions are as follows:

- If the agent chooses **Quit**, it will be **100% Out of the game**.
- If the agent chooses **Continue**, there is a **33% chance** the agent will end up **Out of the game**, and a **67% chance** it will remain **In the game**.

The rewards are defined as:

- Entering the **Out of the game** state gives a reward of **5**.
- Remaining **In the game** gives a reward of **3**.

There is no discount factor.

Questions:

(a) **First Iteration:**

Compute the value of the state **"In the game"** after the first iteration.

(b) **Second Iteration:**

Compute the updated value of **"In the game"** for iteration 2.

(c) **Third Iteration:**

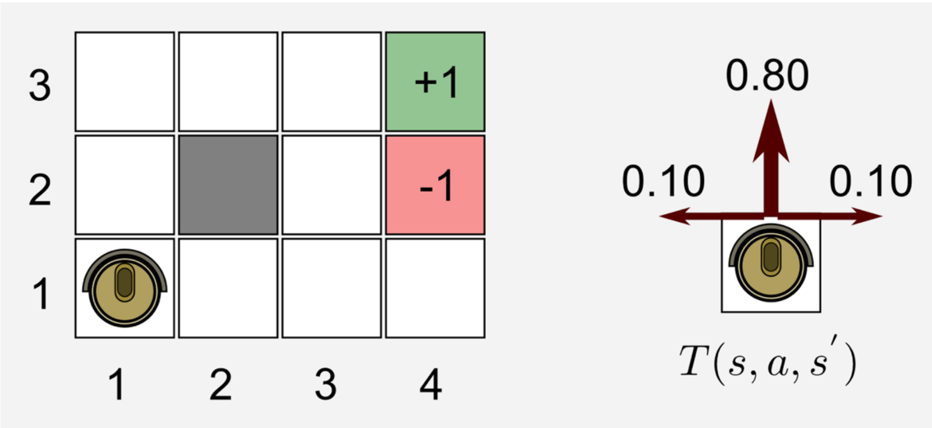
Compute the value of **"In the game"**for the third iteration.

(d) **Policy Discussion:**

Why might the agent prefer to **Quit** instead of choosing **Continue**?

Support your reasoning based on the utility value/values computed in the above iterations.

III. Problem: 3: Reinforcement Learning [20pts]



filename

Figure 3: GridWorld Setup and Transition Probabilities

You are given a 4x3 GridWorld environment. The agent starts at position **(1,1)**. The movement model is stochastic: when the agent attempts to move in a certain direction:

- It moves in the intended direction with probability **0.80**.
- It moves 90 degrees to the left with probability **0.10**.
- It moves 90 degrees to the right with probability **0.10**.

Terminal states:

- **(4,3)** has a reward of $+\mathbf{1}$.
- **(4,2)** has a reward of $-\mathbf{1}$.

All other non-terminal cells give a reward of **0**. The gray cell is a wall and is inaccessible. There is no discount.

Your task

Compute the Q-table and V-table after the first iteration.

IV. Problem 4: Q-Learning with FrozenLake (40 pts)

In this section, you will implement and evaluate the **Q-learning algorithm** to train an agent to navigate the `FrozenLake` environment. The agent should learn a policy to reach the goal while avoiding holes, under both deterministic and stochastic settings.

Environment Setup

This assignment uses a **custom Gym-compatible FrozenLake environment** with GUI support.

Before running the code, install the following dependencies:

```
pip install gymnasium pygame
```

Files Provided

Filename	**Purpose**
`q_learning_agent.py`	Q-learning agent implementation (scaffolded)
`Main.py`	Main script for launching training and evaluation
`frozenlake_env.py`	Custom FrozenLake environment with GUI and keyboard controls

a. Q-Learning Agent Implementation (25 points)

You are provided with a scaffolded `train()` method inside the `QLearningAgent` class. Your task is to complete the missing Q-learning logic based on the comments marked `# TODO:`.

Complete the following components inside the training loop:

(a) Epsilon-Greedy Action Selection

(b) Environment Step

(c) Episode Termination Logic

(d) Temporal-Difference (TD) Target Calculation

(e) Q-table Update

Code Snippet (Inside `train()` Loop):

```
while not done and steps < self.max_steps:
    # TODO: Choose action using epsilon-greedy policy
    # if np.random.rand() < self.epsilon:
    #     action = ...
    # else:
    #     action = ...

    # TODO: Take a step in the environment
    # new_state, reward, terminated, truncated, _ = ...

    # TODO: Compute whether the episode has ended
    # done = ...
```

```
    # TODO: Calculate the TD target
    # if not terminated:
    #     best_next_action = ...
    #     td_target = ...
    # else:
    #     td_target = ...

    # TODO: Update the Q-table
    # td_error = ...
    # self.q_table[state, action] += ...
```

Constraints:

- Do not change any method/function signatures.
- Use only `NumPy` and standard Python logic.
- Use keyboard shortcuts (e.g., `=` to increase FPS) for better GUI performance during training.

b. Training & Testing (15 points)

Evaluate your agent in the following two configurations:

```
\textbf{Deterministic Setup (4x4 grid, 2000 episodes):}
python main.py --mode q --map 4x4 --episodes 2000 --render
\vspace{0.5em}
\noindent
\textbf{Stochastic Setup (8x8 grid, 4000 episodes):}
\begin{verbatim}
python main.py --mode q --map 8x8 --episodes 4000 --render --slippery
```

Language=python

For each configuration, submit the following:

- A screenshot of the console output showing final success rate.
- A brief explanation covering:
  - (i) How the agent's policy evolved over time.
  - (ii) Whether the learned policy appears optimal or suboptimal.
  - (iii) Key differences in learning behavior between deterministic (non-slippery) and stochastic (slippery) environments.

Expected Success Rate / Reward Range

Map	Episodes	Expected Avg Reward (or Success Rate)
4x4	2000	~0.8 and above (well-trained agent)
8x8	4000	~0.2 to 0.5 (realistic target)

In the 8x8 slippery setting, a success rate of **0.2 to 0.4** is considered a solid achievement for Q-learning for this assignment.