Solving a 3D Gridworld Problem using MDP & RL

A project Report

of

CS F407 Artificial Intelligence

Department of CSIS, Pilani Campus

October, 2025

Submitted by :

Abhinand P (2024PHXP0497, CSIS)
Ashin Babu (2024PHXP0495P, CSIS)
Priyanka Jain (2024PHXP0498, CSIS)

Submitted To:

Prof. Navneet Goyal
CSIS

Part A — MDP & RL Formulation

- **S (states):** The state space S, is the set of all the possible locations the agent can be in. Specifically its every cell (x,y,z) within the H x W x D grid, excluding the cells designated as obstacles.
- **A (actions):** The action space A is the set of six possibles moves the agent can attempt from any state. +x (east), -x (west), +y (north), -y (south), +z (up), -z (down).
- **P (transition probability):** The transition probability function P(s'—s,a), defines the probability of moving to the state s' from the state s after taking action a. The "slip" is encoded in this function by making the environment stochastic.
  - Intended outcome: With a probability of P, the agent moves in the intended direction of the chosen action. For example, if the agent chooses +y(north), it will move to the cell directly to the north with probability P.
  - Slip outcome: With the probability 1-P, the agent "slips" and moves in the direction perpendicular to its intended one. The 1-P is uniformly distributed among the four perpendicular directions. For the above example of +y(north), the agent will move in each of the four perpendicular directions with a probability of (1-P)/4
  - Blocked moves: If an intended or slip move would cause agent to hit a boundary or an obstacle, the agent stays in the current state
- **R(rewards):** The reward function R gives the numerical values for the transition.
  - Reaching the goal state gives a reward of +50
  - Falling into a pit gives a reward of -50, or a penalty of 50
  - All other moves have a step cost (reward) of -1 to encourage the agent to find a solution efficiently.
- **γ(Discount factor):** The discount factor γ is set to 0.95. This value makes the agent prioritize immediate rewards, but is still patient enough to plan for long-term gains.

Part B - Environment Implementation(refer jupyter notebook file)

The Gridworld3D class is developed to serve as a complete and interactive simulation of the problem environment. This class encapsulates all the specified rules, states, and dynamics, providing a robust foundation for training a reinforcement learning agent.

Class Architecture and Initialization

The environment is configured within the class constructor, `__init__`. This method initializes the grid's static properties based on the problem definition:

- **Grid and states:** The environment is a 6 x 6 x 6 cube. A state s id represented by a tuple of coordinates (x,y,z). The state space S includes all such coordinates except those designated as obstacles.
- **Terminals and Obstacles:** The absorbing terminal states are defined with a goal at (5,5,5) and a pit (2,2,2). Approximately 10-15% of the remaining cells are randomly selected as obstacles. To ensure that the start and goal states remain reachable and that the experiments are repeatable, a fixed random seed is used for obstacle generation.
- **Actions:** The action space A consists of six discrete actions: +x (east), -x (west), +y (north), -y (south), +z (up), -z (down). THese are mapped to integer indices within the code for computational convenience.

Modeling Stochastic Dynamics

The core of the environment's dynamic behavior is implemented in the step(action) method. This function takes an action from the agent and computes the resulting state transtion and reward, accurately modeling the stochastic nature of the world.

- **Transition Probability (P) :** The model incorporates a "slip" probability. For a chosen action, the agent moves in the intended direction and the probability p. With probability 1-p, the agent "slip" and moves to one of the four directions perpendicular to the intended axis, with each of these slip outcomes being equally likely. If an attempted move, whether intended or a slip would result in hitting a boundary or an obstacle, the agent remains in its current state.
- **Reward Calculation (R):** After determining the next state, the method assigns a reward. A large positive reward of +50 is given for reaching the goal, and a large negative reward of -50 is given for falling into a pit. All other transitions result in a step cost of -1 to encourage the agent to find the most efficient path.
- **Standard RL Interface:** The method returns a tuple (next_state, reward, done), which is a standard interface for reinforcement learning environments. next_state is the agent's new coordinate, reward is the numerical feedback, and done is a boolean flag that becomes True when the agent enters an absorbing terminal state, signaling the end of an episode.

Part C - Q-learning Implementation

To solve the 3D gridworld problem, the Q-learning algorithm was implemented. Q-learning is a model-free, temporal-difference learning algorithm designed to find the optimal action-selection policy. It achieves this by learning a state-action value function, Q(s,a), which estimates the expected return of taking action a from state s.

Data Structure and Initialization

The core of the implementation is the Q-table, which stores the learned values. A Python dictionary was chosen for this purpose, mapping each state tuple (x, y, z) to a NumPy array containing the Q-values for all six possible actions. This structure is memory-efficient for a large and sparse state space where not all grid cells are reachable. Initially, all Q-values in the table are set to zero, representing the agent's complete lack of prior knowledge about the environment.

Learning Process

The agent learns through an iterative process over a set number of episodes. Each episode consists of the agent moving from a starting position until it reaches a terminal state (the goal or a pit).

- **Exploration v/s Exploitation:** To ensure the agent discovers optimal paths instead of settling for suboptimal ones, an ϵ-greedy exploration strategy was used. At each step, the agent chooses a random action with a probability ϵ (exploration) or the action with the highest current Q-value with probability 1-ϵ (exploitation). The value of ϵ is initially set high (1.0) and is gradually decayed after each episode, shifting the agent's behavior from exploration to exploitation as it learns more about the environment.
- **Q-Value update rule:** After each action, the Q-table is updated using the Bellman equation, as specified in the assignment. The formula for the update is:

  $$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma max_{a'} Q(s', a') - Q(s, a))$$

  Here, α is the learning rate, which controls how much the new information overrides old information. The term $r + \gamma max_{a'} Q(s', a') - Q(s, a)$ serves as a more accurate estimate of the value of the current state-action pair, incorporating the immediate reward r and the discounted maximum Q-value of the next state s'.

Through this repeated process of action, reward, and update over 5,000 episodes, the Q-values iteratively converge toward the optimal values, allowing the agent to learn the best policy for navigating the gridworld.

Part D - Policy Evaluation & Comparison

After the training phase, the agent's learned knowledge was evaluated to quantify its performance and effectiveness. This evaluation was conducted by extracting the final policy from the Q-table and comparing its performance against a simple baseline.

Extracting the Greedy Policy

Once the Q-table has converged after 5,000 training episodes, the optimal policy π(s) is deterministic and can be directly extracted. For any given state s, the optimal action is the one with the highest Q-value. This is known as the greedy policy, formulated as:

$\pi(s) = \arg\max_a Q(s, a)$
This policy represents the agent's best-learned strategy for navigating the gridworld.

Evaluation and Baseline Comparison

To assess the quality of this learned policy, it was evaluated over 100 test episodes. During this evaluation, exploration was disabled (ϵ=0) to ensure that the agent was purely exploiting its learned knowledge.

The performance metric used was the average total return (the sum of rewards) per episode. This result was then compared against a baseline random policy, where the agent selects one of the six actions with uniform probability at each step, representing an unintelligent approach.

Results

The evaluation yielded a clear distinction in performance between the two policies:

- Learned Q-learning Policy Average Return: 32.38
- Baseline Random Policy Average Return: -249.61

The results demonstrate that the Q-learning agent successfully learned an effective policy. The positive average return indicates that the agent consistently found the goal while minimizing step costs. In contrast, the large negative return of the random policy shows that, without learning, an agent is highly likely to accumulate significant penalties from wandering aimlessly or falling into the pit. This stark contrast validates the effectiveness of the Q-learning algorithm in solving the 3D gridworld problem.

Part E: Experiments & Analysis

To understand the sensitivity and behavior of the Q-learning agent, a series of experiments was conducted by varying key parameters of the MDP. The parameters analyzed were the **discount factor (γ)**, the environment's **slip probability**, and the **step cost**. For each experiment, one parameter was changed while the others were held at their baseline values to isolate its effect on the agent's learning curve and final policy.

Effect of Varying the Discount Factor (γ)

The discount factor γ controls the agent's foresight. The baseline $\gamma = 0.95$ was compared against a "short-sighted" agent ($\gamma = 0.7$) and a very "patient" agent ($\gamma = 0.99$). As shown in Figure 1, a lower γ led to faster initial learning but convergence to a suboptimal policy, as the agent was less willing to endure a long sequence of negative step costs for the distant goal reward. Conversely, a higher γ resulted in a more optimal final policy that took safer paths, achieving a higher average return.

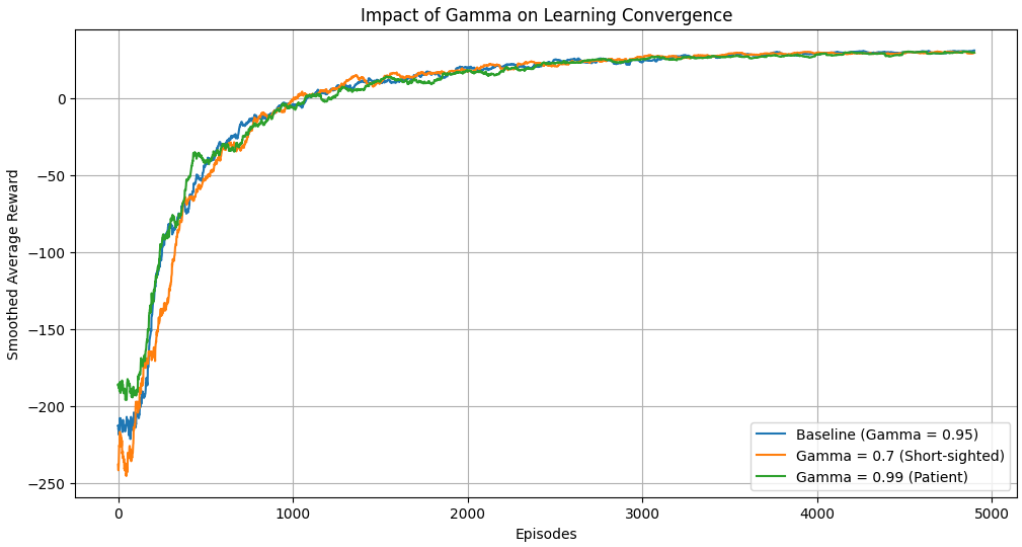

Figure 1: Comparison of learning curves for different discount factors (γ).

Effect of Varying Slip Probability

This experiment tested how environmental stochasticity affects learning. In the near-deterministic case (low slip), the agent learned very quickly, as seen in Figure 2. The resulting policy was highly efficient. With high slip probability, the learning problem became significantly harder. The learning curve was noisy and converged much more slowly to a lower average return, and the final policy was notably more cautious, avoiding cells from which a slip could be catastrophic.

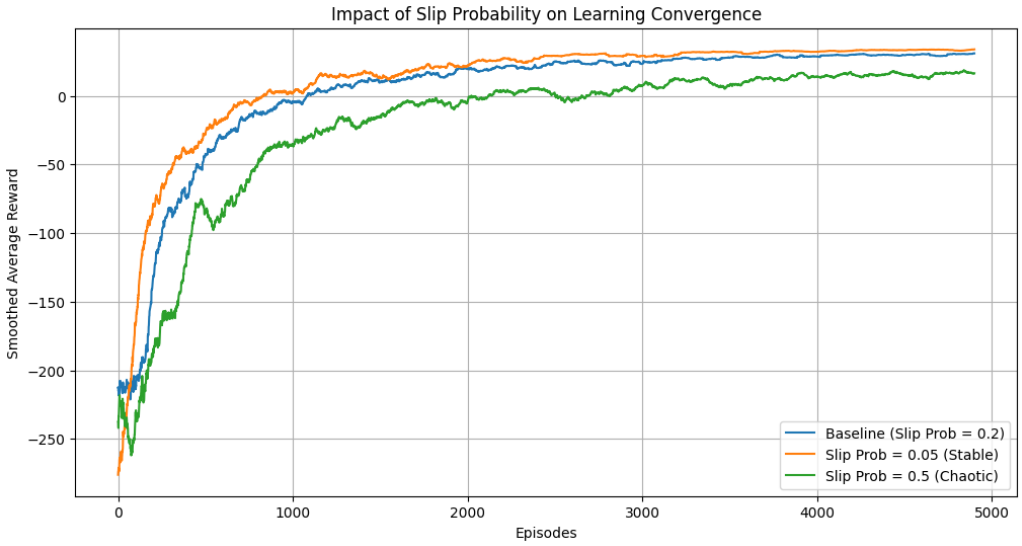

Figure 2: Comparison of learning curves for different slip probabilities.

Effect of Varying Step Cost

The step cost encourages efficiency. As illustrated in Figure 3, a high penalty (-5) forced the agent to prioritize path length, resulting in a highly optimized, short-path policy. In contrast, a low penalty (-0.1) reduced the agent's urgency, and while it still learned to reach the goal, the final policy was often longer and less efficient.

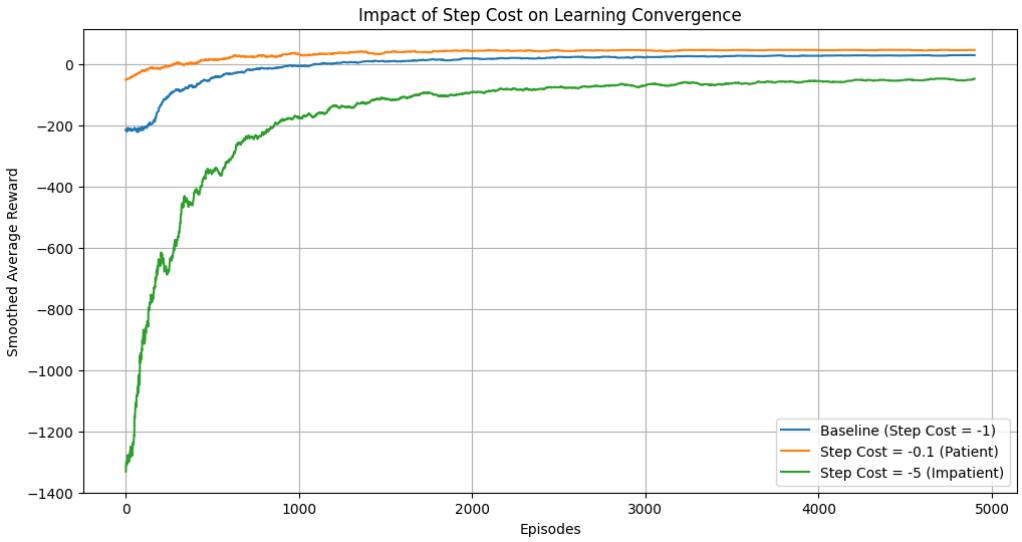

Figure 3: Comparison of learning curves for different step costs.

Part F: Visualization

To interpret the agent's learned behavior and verify the effectiveness of its final policy, two key visualizations were generated. These visualizations provide an intuitive understanding of both the agent's decision-making process at each state and its overall path-finding strategy.

Value Function and Policy Heatmaps

The learned state-value function, $V(s) = \max_a Q(s, a)$, was visualized to show the agent's valuation of each state in the grid. Since the environment is three-dimensional, the value function was plotted as a series of 2D heatmaps, with one slice for each selected z-level (Figure 4). The color intensity of each cell corresponds to its value, with brighter colors indicating states from which the agent expects higher future rewards.

Overlaid on these heatmaps are arrows representing the learned greedy policy for each state, showing the optimal action to take. This combined view clearly illustrates the agent's strategy, such as its tendency to move toward high-value regions and navigate around obstacles and the pit.

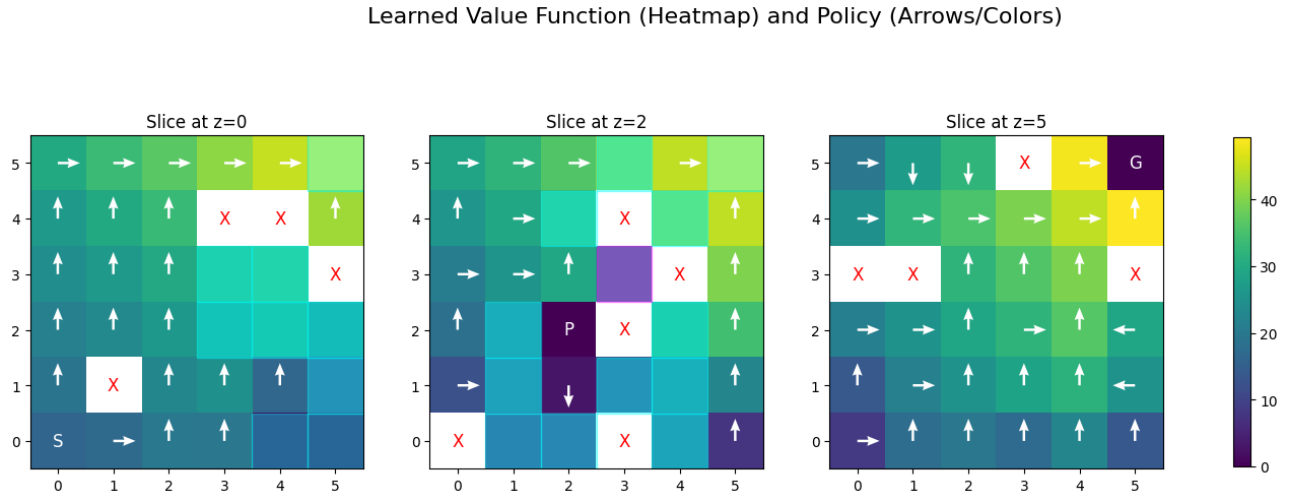

Figure 4: Heatmap of the learned value function $V(s)$ for z-levels 0, 2, and 5. Color intensity represents the value of a state (brighter is better), and arrows indicate the optimal policy.

3D Optimal Path Trajectory

To visualize the policy in action, a sample trajectory was plotted in 3D (Figure 5). This plot shows the complete path taken by the agent when following the final greedy policy from the start state to the goal state. The visualization clearly shows how the agent successfully navigates the 3D space, avoiding obstacles and the pit to reach the goal. This provides concrete evidence of a successful and coherent learned policy.

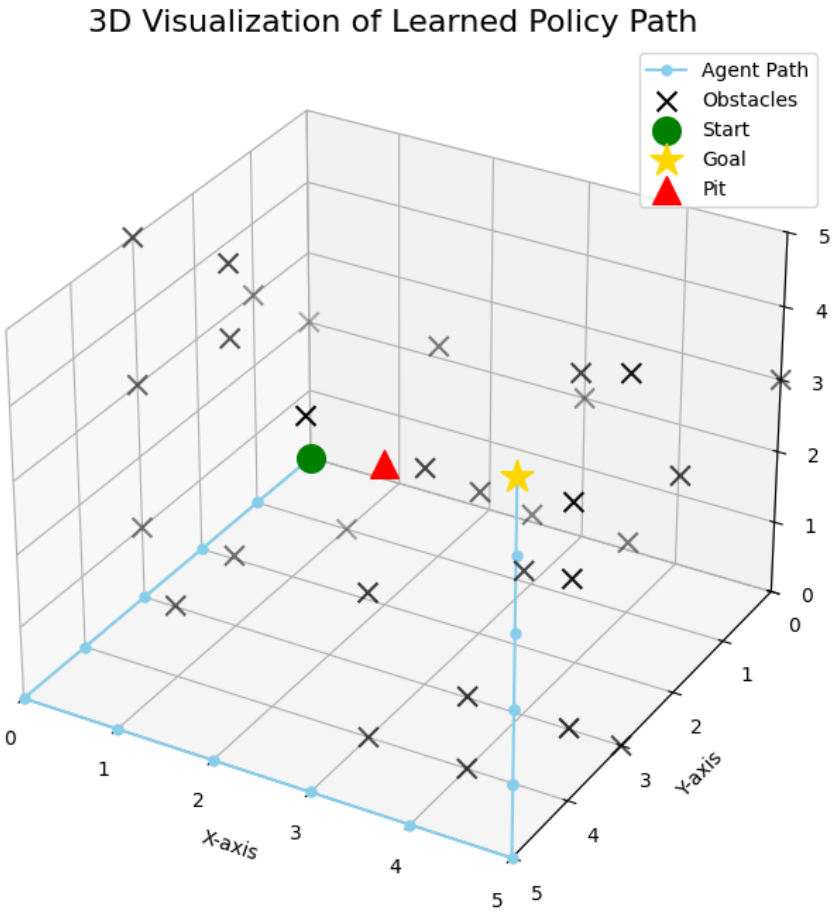

Figure 5: A sample trajectory of the agent following the final learned policy from the start state (green) to the goal state (gold), successfully navigating around obstacles (black).