Project Report: GridWorld Q-Learning Simulation

Project Overview

This project implements a GridWorld environment where two players navigate a grid to learn optimal policies through Q-learning. The environment includes features such as walls, kill zones, and terminal states, with a graphical user interface (GUI) for visualizing the grid and player movements.

Objectives

- To create a grid-based environment where agents can learn to maximize their rewards.
- To implement the Q-learning algorithm to allow agents to learn optimal actions.
- To visualize the learning process and the final paths taken by the agents using a GUI.

Key Components

1. GridWorld Class

The GridWord class encapsulates the grid environment, managing the following functionalities:

- Initialization: Sets up the grid dimensions, initializes player positions, and creates the grid cells.
- Kill zones: Randomly places kill zones on the grid, which penalize players for stepping into them.
- Terminal States: Designates specific cells as terminal states with associated rewards.
- Action Selection: Implements an epsilon-greedy strategy for action selection to balance exploration and exploitation.
- State Transition: Calculates the next state based on the current action and player position.
- Q-value Updates: Updates the Q-values based on the learning algorithm (Q-learning).
- Episode Management: Restarts episodes and tracks cumulative rewards.

2. Q-Learning Algorithm

The q_learning_algorithm method implements the Q-learning algorithm, which includes:

- Action Selection: Chooses actions based on the current Q-values and epsilon-greedy strategy.
- State Updates: Updates the Q-values based on the rewards received and the maximum future reward.
- Episode Tracking: Tracks the number of episodes and adjusts epsilon to reduce exploration over time.

3. GUI Implementation

The GridWorldGUI class uses the Tkinter library to create a user-friendly interface for the GridWorld simulation:

- Grid Visualization: Displays the grid with players, walls, killzones, and terminal states.
- User Input: Allows users to specify grid dimensions and the number of episodes for training.
- Simulation Control: Provides a button to start the simulation and visualize the players' movements in real-time.

4. Helper Functions

The code includes several helper functions to manage grid states, random position selection, and wall placements, ensuring the environment is dynamic and varied for each simulation run.

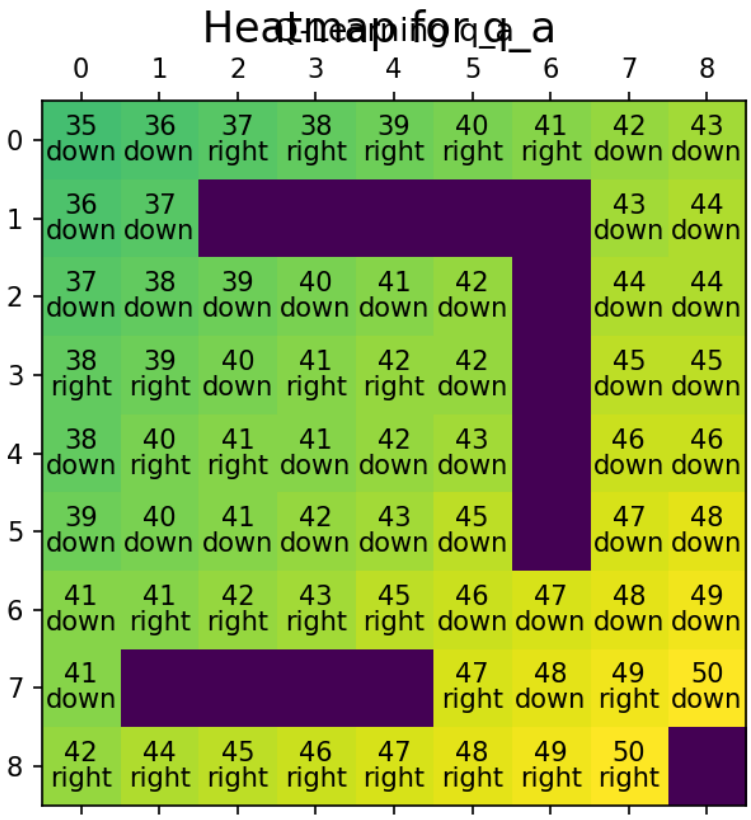
Implementation Details

Code Structure

- Class Definitions:
    - GridWord: Main class for the grid environment.
    - GridWorldGUI: Class for the graphical user interface.
- Methods:
    - __init__: Initializes the grid and player positions.
    - place_killzones: Randomly places killzones on the grid.
    - get_max_q: Determines the best action based on Q-values.
    - set_terminal_state: Configures terminal states in the grid.
    - action_e_greedy: Implements the epsilon-greedy action selection strategy.
    - q_learning_algorithm: Executes the Q-learning algorithm over a specified number of episodes

Heatmap for Q-values

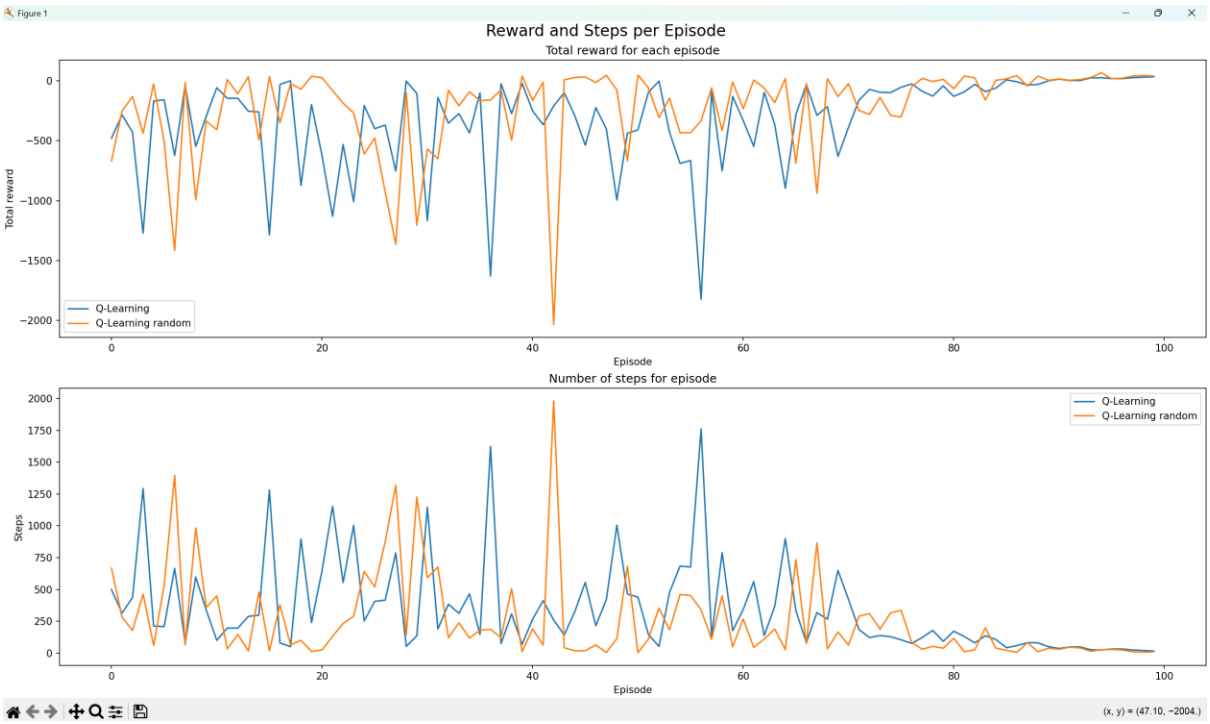
The heatmap provides a visualization of the learned Q-values for each state-action pair within the GridWorld environment. The colour intensity indicates the magnitude of the Q-value, with higher values suggesting optimal actions that yield greater rewards over time. Each cell displays the action (e.g., "up," "down," "right," "left") that the agent considers most favourable based on the learned Q-values. This representation allows us to observe the convergence of Q-learning as the model identifies the optimal policy across the grid, avoiding kill zones and seeking terminal states.



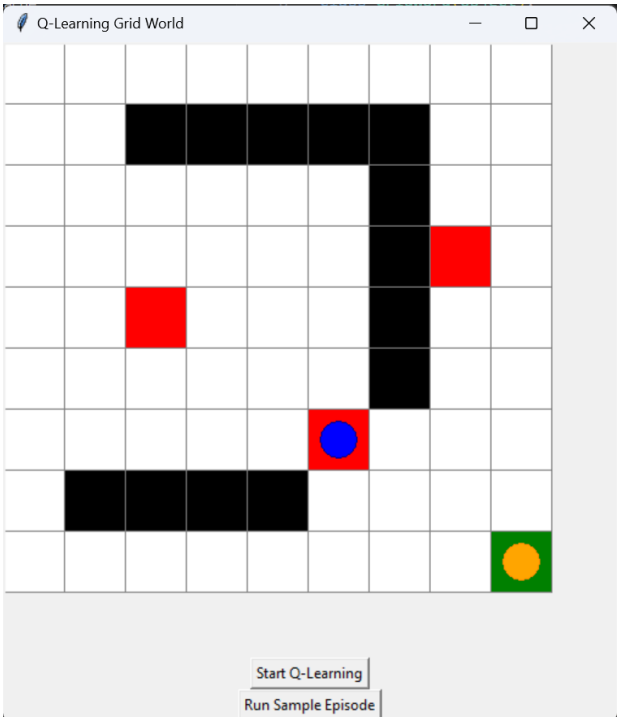
Reward and Steps per Episode

The line graphs show the cumulative reward and the number of steps taken by the agent per episode throughout the training process. The first graph illustrates the total reward for each episode, comparing the Q-learning algorithm's performance to a baseline random strategy. Over time, the cumulative reward increases as the agent learns to avoid negative states and reach terminal states, indicating improved performance.

The second graph displays the number of steps taken per episode, with a decreasing trend that suggests the agent's growing efficiency in navigating the grid. As learning progresses, the agent requires fewer steps to complete episodes, further highlighting the effectiveness of the Q-learning model in optimizing its movements within the environment.



Demonstration of Project



Conclusion

This project successfully implements a GridWorld environment where two players learn to navigate through Q-learning. The GUI enhances user interaction, allowing for real-time observation of the learning process. The combination of algorithmic implementation and visual feedback provides a comprehensive understanding of reinforcement learning in a grid-based context.

Future Enhancements

- Randomizing the placement of walls and more dynamic killzones.
- Incorporating enemy position as a decision making factor.
- Implement additional reinforcement learning algorithms for comparison.