Grid World Random Agent Simulation

Introduction

This project involves the design and implementation of a Grid World simulation using the Python Pygame library. The aim of the project is to model the interaction between an autonomous agent and a discrete environment and to observe the effectiveness of different movement strategies(adaptive and non-adaptive) when navigating towards a fixed goal. The simulation provides a visual and programmatic representation of an agent operating under strict environmental constraints, allowing performance to be evaluated across multiple trials.

The project draws inspiration from classical artificial intelligence environments that are often used as an introduction to agent based systems and reinforcement learning. In the non-adaptive implementation, the agent does not learn from experience and instead operates according to a purely random policy. In the adaptive implementation, the agent(through Q learning) learns from its environment.Using these two different implementations allows the behaviour of an agent to be examined in a controlled setting.

Environment Design

The environment is implemented as a two dimensional grid consisting of five rows and five columns. Each grid cell has fixed dimensions of one hundred by one hundred pixels. The grid is rendered within a Pygame window.

The agent begins each episode in the top left cell of the grid, while the goal state is located in the bottom right cell. The goal is visually represented by a blue square, making it clearly distinguishable from other grid cells. Two obstacle cells are placed at predefined positions within the grid and are displayed in green. These obstacles represent impassable terrain and constrain the possible movements of the agent.

The Grid class is responsible for drawing the grid, obstacles and goal state. It also handles redrawing grid lines after the agent moves, ensuring that the grid remains visually consistent throughout the simulation. In addition, the class provides a method to determine the agent's current position by examining pixel colours at the centre of each grid cell.

Agent Design

The agent is implemented in a separate class, ensuring a clear separation between the environment and the entity operating within it. The agent is represented as a red square and occupies exactly one grid cell at any given time. Movement is restricted to one cell per action, reflecting the discrete nature of the environment.

At each step, the agent selects a direction at random from the set of possible actions. These actions correspond to movement in the horizontal or vertical directions. Before completing a move, the agent verifies that the resulting position lies within the boundaries of the grid and does not coincide with an obstacle. If the selected action results in an invalid position, the move is reversed and a new direction is chosen.

This approach ensures that the agent always remains in a valid state and that illegal moves do not disrupt the simulation.

Episode Execution

The simulation is structured around the concept of episodes. Each episode represents an independent attempt by the agent to reach the goal from the starting position. A fixed number of episodes are executed, and each episode is subject to a maximum step limit to prevent infinite execution.

During an episode, the agent performs a sequence of random moves. After each move, the display is updated to reflect the agent's new position, and a short delay is applied to make the movement visually observable. If the agent reaches the goal state, the episode terminates early and the environment is reset for the next trial.

The number of steps taken to reach the goal is recorded for each successful episode. The simulation keeps track of the smallest number of steps observed so far, allowing performance across episodes to be compared.

State Detection and Evaluation

Rather than maintaining a separate logical representation of the grid state, the program determines the agent's position by analysing the rendered display. The Grid class scans each cell and checks the colour of a pixel located at the centre of that cell. When a red pixel is detected, the corresponding grid coordinates are returned as the agent's current position.

This method allows the simulation to infer state information directly from the visual output. While this approach is functional for a small grid, it would become inefficient for larger environments and is therefore best suited for demonstration purposes.

Performance evaluation is based solely on the number of steps required to reach the goal. This metric provides a simple but effective measure of how inefficient a random strategy can be in a constrained environment.

Limitations and Future Work

The primary limitation of the system is the absence of learning. The agent does not modify its behaviour based on previous outcomes and therefore cannot improve its performance over time. As a result, success is largely dependent on chance, and many episodes fail to reach the goal within the step limit.

Further improvements could include the introduction of a learning algorithm that allows the agent to associate actions with outcomes. Maintaining an internal representation of the grid state would also improve efficiency and scalability. Additional features such as variable grid sizes or dynamic obstacle placement could further extend the complexity of the simulation.

Conclusion

This project demonstrates a complete implementation of a Grid World environment and a random agent using Pygame. It successfully illustrates the interaction between an agent and a constrained environment and provides a clear example of how performance can be evaluated across multiple episodes. Although the agent’s behaviour is intentionally simple, the project establishes a solid foundation for future extensions involving learning based approaches and more sophisticated decision making.