Project Overview

In this project, I designed a simple Snake game utilizing Q-learning, a popular reinforce.The objective of this project is to design and implement a simple Snake game using Q-learning, a widely applied reinforcement learning algorithm. The primary goal is to allow the snake to continuously consume food while avoiding collisions with walls, itself, and randomly generated obstacles. By utilizing a Q-learning agent, the game enables the agent to learn optimal decision-making strategies based on rewards and penalties, thereby enhancing the effectiveness of the gameplay strategy. Through this project I not only practiced the Q-learning algorithm but also gained a deeper understanding of fundamental principles and implementation details in game design.

1.Game Design

1.1 Objective and Rules

The main objective of the game is for the snake to eat food, resulting in its growth. The player controls the snake's movement and must navigate within a defined grid space while avoiding collisions with walls, itself, or randomly generated obstacles. If a collision occurs, the game ends. The rules of the game are straightforward and easy to understand, allowing novice players to quickly get the hang of it. At the start of the game, the snake's length is one segment, and with each food consumed, its length increases, thereby raising the difficulty level.

The core of the game design is to balance challenge with enjoyment. In the initial phases, the food generation location is relatively simple, but as the game progresses, the number and distribution of obstacles increase, requiring players or agents to continuously adjust their strategies to adapt to the changing environment. To enhance the game's fun factor, we also designed multiple colors and shapes for the food and obstacles, making the game visually appealing.

1.2 State Space and Action Space

State Space: The state of the game is defined by the current position of the snake's head, the position of the food, and the positions of any obstacles. Thus, the state can be represented as a tuple containing these three elements: (snake_head_position, food_position, obstacle_positions).

Action Space

The possible actions the agent can take include moving up, down, left, or right. These are represented as coordinate changes:

Up: (0, -GRID_SIZE)

Down: (0, GRID_SIZE)

Left: (-GRID_SIZE, 0)

Right: (GRID_SIZE, 0)

1.3 Reward Function

Designing a reasonable reward function is crucial for reinforcement learning. To incentivize the agent to progress toward the objective, we designed the following reward mechanism:

For desirable actions, positive rewards are given:

Eating food grants a reward of +10, encouraging the snake to consume as much food as possible.

Moving closer to food provides a reward of +1, motivating the agent to approach the food.

Exploring new areas yields a reward of +0.5, rewarding the agent for discovering unknown regions.

For undesirable actions, negative rewards are assigned:

Colliding with walls or obstacles results in -10, penalizing erroneous actions.

Moving away from food incurs a -1 penalty, encouraging the agent to remain near the food.
This design ensures that the agent learns which behaviors are beneficial and which are harmful, thereby enhancing its decision-making capabilities.

1.4 Documentation of Game Design

During the design process, we created several flowcharts that illustrate the game's mechanics, state transitions, and reward distribution. These diagrams helped us better understand the game's logic and served as references during implementation. The main components of the game were modularized, ensuring a clear code structure that simplifies debugging and expansion. In the documentation, we detailed the functionalities, inputs, outputs, and interrelations of each module, providing a solid foundation for future maintenance and upgrades. Additionally, we included some diagrams to showcase the game layout and operational flow.

2. Q-Learning Implementation

2.1 Q-Learning Algorithm

The Q-learning algorithm was implemented in Python, utilizing the Pygame library. The core of the algorithm is a Q-table that maps states to actions with associated rewards. To achieve this, we followed these basic steps:

Initialize the Q-table, setting all Q-values to zero initially.

The agent collects experiences through interactions with the environment and updates the Q-table.

The agent selects actions based on the values in the Q-table and updates the Q-values according to the outcomes of those actions.

2.2 Policy Learning

The agent learns a policy that maximizes cumulative rewards over time through iterative updates to the Q-table. The policy is derived from the Q-values, guiding the agent's actions in the environment.

2.3 Exploration Techniques

An epsilon-greedy strategy was employed to balance exploration and exploitation:

- The agent explores the environment with a probability defined by epsilon, which decays over time to encourage exploitation of learned policies.
- Learning rate and discount factor were adjustable parameters that influenced how quickly the agent learns and how much it values future rewards.

2.4 Documentation and Explanation

Throughout the implementation, we maintained detailed documentation with inline comments explaining each function's purpose, inputs, and outputs. This documentation aids other developers in understanding the code's logic, facilitating future maintenance and expansion. Additionally, we prepared a separate document detailing the Q-learning process, explaining the rationale behind parameter choices and implementation specifics to ensure anyone could quickly grasp our methods. We also recorded experimental results to compare the agent's performance under different parameters for future optimization and adjustments.

3. Game Interaction

3.1 User Interface

A user interface was developed using the Pygame library, allowing the agent to interact with the game environment. The UI includes visual representations of the snake, food, and obstacles.

3.2 Display of Current State

The user interface continuously displays the current state of the game, including:

The position and length of the snake.

The position of the food.

The locations of obstacles.

The actions taken by the agent.

The rewards and penalties received.
This design allows players to stay informed about game progress, enhancing interactivity. By displaying this information, players can better understand the agent's decision-making process, thereby increasing the game's enjoyment and educational value.

3.3 Libraries and Framework Usage

Pygame was utilized effectively to create the game environment, handling graphics rendering and user input. The choice of this library facilitated smooth animation and interaction.

3.4 User Experience

The game provides an intuitive user experience, allowing players (or agents) to easily control the snake's movements. The interface responds quickly, offering immediate feedback, enabling players to rapidly adapt to the game's rhythm. Moreover, the design also considers novice players, allowing them to easily grasp the mechanics and enjoy the game. By gradually guiding players to master the game's mechanics, we enhanced the playability and attractiveness of the game. Additionally, we considered compatibility across different devices, ensuring that the game runs smoothly on various platforms.

4.Presentation

4.1 Video Presentation

A video demonstration was prepared, showcasing the gameplay and explaining the Q-learning algorithm. Through the video, viewers can visually observe the game's operation and the agent's learning process. This visual representation effectively enhances the audience's understanding. In the video, we thoroughly explain the background of the game design, the implementation of Q-learning, and the challenges we faced and solutions we devised, aiming to provide a comprehensive overview of the entire project.