

# Reflection Report

## 1. What of OO concepts/strategies/tools did you use in this assignment?

OOP concepts:

Abstraction: The snake, food and environment are abstracted into independent classes. Each class only exposes necessary interfaces, hiding internal complexity.

Encapsulation: In the Snake class, the snake's body coordinates (self.body) and movement logic (move(), turn()) were encapsulated. External interaction was only allowed through public methods, ensuring data integrity.

Inheritance: Our SnakeEnv class in snake\_env.py inherited from gymnasium.Env.

Design tools: Github

## 2. Explain your implementation in detail

- **SnakeEnv:**

- **State abstraction (狀態空間抽象):** In order to make the Q-learning adapt the 20\*20 map, the observation space was abstracted into a low-dimensional vector, which includes local hazard information around the snake's head (immediate danger ahead, left, and right) and the relative direction of the food (X/Y axis).
- **Reward function (獎勵函數):** We adopted the Reward Shaping strategy. The total reward comprised three components: Goal Reward (eating food +10), Survival Penalty (dying -10), and Efficiency Reward (moving only -0.1)

- **Core Q-Learning Design:**

- **Discount Factor (Gamma):** Set initially to 0.1. This allowed us to learn quickly in the early stages.
- **Exploration Rate (Epsilon):** Set very high at 0.99 intentionally. Since Snake is a long-term goal game, this high gamma forced the Agent to consider distant future rewards, preventing it from getting stuck in dead ends by prioritizing immediate gain.

- **Innovation: Flood Fill (BFS)**

Although the Q-Table learning trained well, we discovered that sometimes the agent would run into an un-escapable small space (causing dead end), which may be because it was misled by local optima.

To solve this problem, we integrated Flood Fill into the `get_action` function:

1. The agent still calculates Q-values for all actions and sorted them from highest to lower.
2. Before the agent executes the highest Q-value action, it first executes the function `_check_flood_fill`, which uses the BFS algorithms. It checks if, after taking the proposed step, the agent could find sufficient free space starting from the next grid cell. (We set limit = `len(body_set) * 2`).
3. The final action will be the one which has a high Q-value and was passed the flood fill safety check. If all high-Q-value actions were deemed unsafe, it defaulted to the highest Q-value action (accepting the risk).

- **Final Evaluation and Comparison**

- **Test A: Basic Q-Learning**
- **Test B: Flood Fill Enhanced**

We compared three key metrics: Success Rate, Average Length, and Efficiency (Steps per Food). The comparison chart clearly demonstrated that with the Flood Fill algorithms, the Agent achieved a significant boost in avoiding self-inflicted dead ends.

### 3. What you learned from this assignment?

Instead of studying the OO concepts in books, implementing them by writing code and make up a project is the best way for us to learn it. We understand the usage of encapsulation, polymorphism through this project, and got to know about Q-learning and the flood fill strategy.

### 4. Do you use AI tool in this assignment?

Yes, we did use AI tool (Gemini) .

## **5. How you used an AI tool assignment?**

We use it when we are not sure if we coded correctly or not, and when we met a problem that couldn't be solved by us (issues like Git or API errors). We also use it to quickly understand the core concept of Q-learning state abstraction, and the rewarding shape mathematical formula, etc.

## **6. Evaluation of the tool's usefulness?**

It's helpful when pointing out the main problem we made but couldn't see, and somehow gave us the direction and some opinions to improve the overall program. But sometimes its suggestions are not suitable for our situation, we might need to slightly adjust it.