Bài tập 2: A* với Pacman

I. Mô tả bài toán:

- Trong bài toán Pac-Man, một nhân vật trò chơi nổi tiếng, di chuyển qua mê cung để thu thập những viên thức ăn rải rác. Mục tiêu cụ thể là xác định một đường đi (hoặc chuỗi đường đi) có tổng số bước di chuyển tối thiểu, cho phép Pac-Man "hoàn thành nhiệm vụ" bằng cách thu thập tất cả các viên thức ăn trong mê cung.
- Trạng thái đích: Bài toán Pac-Man kết thúc khi Pac-Man ở vị trí mục tiêu hoặc khi không còn mục tiêu nào trên bản đồ.
- Quy luật di chuyển:

  + Danh sách các hành động của Pac-Man : Bắc, Đông, Tây, Nam, Dừng
  + Không đi xuyên tường.
  + Có thể đi qua thức ăn (ăn thức ăn khi đi vào ô đó).
  + Có thể đi xuyên tường trong 5 bước sau khi ăn bánh ma thuật
  + Nếu pacman đến góc mê cung, nó sẽ tự động dịch chuyển đến góc đối diện.

- Yêu cầu của bài tập:

  + Mô hình hóa bài toán dưới dạng không gian trạng thái, xác định các thành phần liên quan.
  + Triển khai thuật toán A* để giải quyết vấn đề, thảo luận về tính admissible và consistent của heuristic được chọn.

II. Định hướng bài làm

1. Xây dựng không gian trạng thái:

- Mỗi trạng thái cần lưu trữ các thông tin sau:

  + Vị trí Pacman: Tọa độ (x, y) trên bản đồ.
  + Danh sách thức ăn còn lại: Vị trí các thức ăn chưa được ăn
  + steps_left: Số bước còn lại có thể đi xuyên tường

2. Thuật toán A*

- Thuật toán A* được triển khai để tìm đường đi tối ưu từ trạng thái ban đầu đến một trong trạng thái đích.
- sử dụng **Manhattan Distance** làm hàm heuristic h(n) để ước lượng khoảng cách từ vị trí hiện tại đến mục tiêu (thức ăn).

- Tính chất của heuristic:

  + Admissibility: Đảm bảo không đánh giá quá cao chi phí thực tế đến đích.
  + Consistency: Đáp ứng điều kiện nhất quán, tức là chi phí ước lượng từ trạng thái hiện tại không vượt quá tổng chi phí thực tế và chi phí ước lượng từ trạng thái tiếp theo.

III. Chi tiết chương trình

1) Lớp Maze

- Lớp Maze được thiết kế để lưu trữ cấu trúc mê cung, hỗ trợ tìm kiếm các vị trí quan trọng và xử lý các chức năng như kiểm tra tường, tìm Pacman, thức ăn, cổng dịch chuyển
- Thuộc tính:
  + **self.grid**: Ma trận (danh sách 2D) lưu trạng thái mê cung.
  + **self.food_positions:** Danh sách tọa độ chứa thức ăn (.).
  + **self.wall_positions**: Danh sách tọa độ chứa tường (%).
- Phương thức:
  + **find_pacman(self):** Duyệt toàn bộ mê cung để tìm vị trí P (Pacman).
  + **find_food(self):** Trả về danh sách tọa độ của thức ăn.
  + **find_walls(self):** Trả về danh sách tọa độ của tường.
  + **is_wall(self, x, y):** Kiểm tra xem vị trí (x, y) có phải là tường hay không.
  + **find_teleports(self):** Định nghĩa các cổng dịch chuyển dựa vào góc của mê cung:
    - (1,1) -> (x_max, y_max)
    - (1, y_max) -> (x_max, 1)

- (x_max, 1) -> (1, y_max)
- (x_max, y_max) -> (1,1)

+ **(self, x, y):** Kiểm tra xem tại tọa độ (x, y) có bánh thần (O) hay không.
+ **x_max(self) & y_max(self):** Xác định giới hạn tọa độ lớn nhất trong mê cung (trừ viền).

2) Lớp Pacman

- Lớp Pacman được thiết kế để biểu diễn trạng thái của Pacman trong mê cung, hỗ trợ tìm kiếm đường đi bằng thuật toán tìm kiếm A*. Mỗi trạng thái của Pacman bao gồm vị trí hiện tại, thức ăn còn lại, số bước có thể xuyên tường, và thông tin về trạng thái cha để truy vết.
- **Thuộc tính**
  + **self.x:** Vị trí hàng (row) của Pacman.
  + **self.y:** Vị trí cột (column) của Pacman.
  + **self.remaining_food:** Tập hợp thức ăn còn lại trong mê cung**.**
  + **self.pie_steps:** Số bước có thể đi xuyên tường (nếu đã ăn bánh thần).
  + **self.g:** Chi phí di chuyển từ trạng thái ban đầu.
  + **self.parent:** Trạng thái cha để truy vết đường đi.
  + **self.maze:** Tham chiếu đến lớp Maze để kiểm tra trạng thái mê cung.
  + **self.state:** Trạng thái duy nhất của Pacman, gồm (x, y, remaining_food, pie_steps).
  + **self.h:** Giá trị heuristic
  + **self.f:** Tổng giá trị đánh giá trạng thái f = g + h để sử dụng trong A*.
- **Phương thức**
  + **is_valid(self, x, y):**Kiểm tra xem vị trí (x, y) có nằm trong phạm vi hợp lệ của mê cung hay không.
  + **is_wall(self, x, y):**Kiểm tra xem vị trí (x, y) có phải là tường (%) hay không.
  + **get_successors(self):**
    - Trả về danh sách các trạng thái kế tiếp của Pacman từ vị trí hiện tại.

- Duyệt qua các hướng di chuyển (Bắc, Đông, Tây, Nam, Dừng)
- Kiểm tra nếu di chuyển hợp lệ và xử lý các trường hợp đặc biệt:
  - Nếu gặp tường % và không có pie_steps, bỏ qua bước đi này.
  - Nếu gặp thức ăn loại bỏ vị trí đó khỏi remaining_food.
  - Nếu gặp bánh thần O, đặt pie_steps = 5.
  - Nếu gặp cổng dịch chuyển, thực hiện dịch chuyển đến vị trí tương ứng.

+ **is_goal(self):** Kiểm tra xem Pacman đã ăn hết thức ăn chưa (điều kiện chiến thắng).

3) Lớp Heuristic

- Lớp Heuristic được thiết kế để cung cấp các hàm heuristic hỗ trợ thuật toán A* trong bài toán Pac-Man. Các hàm này ước lượng chi phí từ trạng thái hiện tại đến trạng thái đích(vị trí thức ăn), giúp định hướng tìm kiếm theo hướng tối ưu. Lớp này được xây dựng theo mô hình lập trình hướng đối tượng (OOP), đảm bảo tính module hóa và dễ bảo trì.
- **Chức năng**: Tính tổng khoảng cách Manhattan từ vị trí hiện tại của mỗi ô đến vị trí đúng của nó trong trạng thái đích gần nhất.
- **Cách hoạt động**: đo khoảng cách theo trục dọc (y) và ngang (x) giữa hai điểm, bỏ qua chướng ngại vật như tường.
- **Công thức toán học**

$$h(n)=|x1-x2|+|y1-y2|$$

Trong đó:

- h(n): Giá trị heuristic, là ước lượng khoảng cách từ Pac-Man đến viên thức ăn.
- x1,y1x_1, y_1x1,y1: Tọa độ hiện tại của Pac-Man trên lưới.
- x2,y2x_2, y_2x2,y2: Tọa độ của viên thức ăn.
- |x1−x2||x_1 - x_2||x1−x2|: Khoảng cách theo trục hoành (trục x) giữa Pac-Man và viên thức ăn.

- |y1−y2||y_1 - y_2||y1−y2|: Khoảng cách theo trục tung (trục y) giữa Pac-Man và viên thức ăn.

4) Lớp A*

- Lớp AStar được thiết kế để tìm đường đi tối ưu cho Pacman bằng thuật toán A*.Thuật toán sử dụng hàng đợi ưu tiên (priority queue) với heapq để lựa chọn trạng thái có chi phí thấp nhất f = g + h. h là giá trị heuristic được chọn từ lớp Heuristic.
- **Thuộc tính:**
  + **self.initial_node:** Trạng thái ban đầu của Pacman.
  + **self.heuristic_name:** Tên của hàm heuristic sẽ sử dụng.
  + **self.nodes_explored:** Danh sách các trạng thái đã được mở rộng trong quá trình tìm kiếm.

- **Phương thức:**
  + **solve(self):** Triển khai thuật toán A* để tìm đường đi tối ưu.
  + **reconstruct_path(self, node):** Truy vết đường đi từ trạng thái cuối cùng về trạng thái ban đầu.

- **Mã giả:**

```
FUNCTION ASTAR
SOLVE(initial_state, heuristic, goalStates) RETURNS path or failure

 initial_node ← NODE(initial_state)
 frontier ← PRIORITY_QUEUE()
 h_value ← COMPUTE_H(initial_node.STATE, goalStates, heuristic
)
 INSERT_IN_QUEUE(frontier, initial_node.g + h_value, initial_nod
e)
 g_score ← MAP()
 g_score[initial_node.id] ← 0
 closed_set ← set()


 WHILE frontier IS NOT EMPTY DO
```

```
        _, current ← EXTRACT_MIN(frontier)
        MARK_AS_EXPLORED(current)

        IF current IS_GOAL(goalStates) THEN
            RETURN RECONSTRUCT_PATH(current)

        FOR EACH neighbor IN current.GET_SUCCESSORS() DO
            h_value ← COMPUTE_H(neighbor.STATE, goalStates, heurist
ic)
            f_value ← neighbor.g + h_value

            IF g_score.get(neighbor.id, INFINITY) > neighbor.g THEN
                g_score[neighbor.id] ← neighbor.g
                INSERT_IN_QUEUE(frontier, f_value, neighbor)

    RETURN failure

    FUNCTION RECONSTRUCT_PATH(node) RETURNS list of states
     from start to goal
        path ← EMPTY_LIST()
        WHILE node ≠ NULL DO
            PREPEND_TO_LIST(path, node.STATE)
            node ← node.PARENT
        RETURN path
```

IV. Ưu điểm và nhược điểm

- **Ưu điểm**
  - + Đơn giản, dễ cài đặt.
  - + Tìm được đường đi tối ưu nếu không có tường chắn.
  - + Dễ triển khai
- **Nhược điểm**
  - + Manhattan Distance không tính đến tường chắn.
  - + Có thể chậm nếu bản đồ lớn và phức tạp (nhiều vật cản).
  - + Không hoạt động tốt nếu có địa hình phức tạp.