BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC QUỐC GIA TP.HCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

----------※※※----------

BÁO CÁO

Cơ Sở Trí Tuệ Nhân Tạo

Project 01

SEARCH

Giảng viên lý thuyết: Bùi Tiến Lên

Giảng viên hướng dẫn thực hành: Trần Quốc Huy, Phạm Trọng Nghĩa

MỤC LỤC

I. Thông tin nhóm:

STT	MSSV	Thành Viên
1	21127183	Phạm Phú Toàn
2	21127665	Nguyễn Thuận Phát
3	21127047	Nguyễn Trần An Hòa
4	21127638	Tô Khánh Linh
5	21127679	Ngô Quốc Quý

II. Mức độ hoàn thành:

STT	Công việc	Hoàn thành
1	Cài đặt level 1	10%
2	Cài đặt level 2	10%
3	Cài đặt level 3	10%
4	Cài đặt level 4	10%
5	Cài đặt Graphic	10%
6	Tạo 5 maps chơi	10%
7	Viết báo cáo	30%
	Tổng cộng	100%

III. Mô tả hàm:

1. Cấu trúc thư mục:

./project

|_/Document

|_/Report

|_/Input

|_/map1_level1.txt

|_/…

|_/Source

|_/level1_2.py

|_/Level3.py

|_/Level4.py

|_/main.py

*level 1: 2 map

level 2: 2 map

level 3: 2 map

level 4: 3 map

2. Môi trường cài đặt và các thư viện sử dụng

Ngôn ngữ: Python

Thư viện: Pygame, random, copy, sys và các thư viện mặc định khác của Python

3. Cài đặt graphic:

a. Cài đặt class sprite:

- **Đầu vào:** position (vị trí của object)
- **Khởi tạo:**
  - Biến current_position để lấy vị trí hiện tại của đối tượng.
  - Biến surface dùng để xác định khoảng cần xóa và vẽ lại.
  - Hàm changePosition: dùng để cập nhật vị trí hiện tại của đối tượng.
  - Hàm draw: dùng để vẽ ra vị trí hiện tại của đối tượng.

b. Cài đặt class Wall:

- Class Wall được kế thừa từ class sprite và sử dụng được các hàm của sprite.
- **Đầu vào:** position (vị trí của tường).
- **Khởi tạo:**
    - Truyền biến position vào sprite.
    - Gọi hàm draw trong pygame để tạo ra khối hình vuông có kích thước 30 * 30 để tạo thành tường.
    - Gọi hàm draw kế thừa từ sprite và vẽ ra tường.

c. Cài đặt class Ghost:

- Class Ghost được kế thừa từ class sprite và sử dụng được các hàm của sprite.
- **Đầu vào:** position (vị trí của tường).
- **Khởi tạo:**
    - Truyền biến position vào sprite.
    - Gọi hàm draw trong pygame để tạo ra các chấm tròn có bán kính bằng 15 để tạo ra những con ma.
    - Gọi hàm draw kế thừa từ sprite và vẽ ra những con ma với màu là màu đỏ.

d. Cài đặt class Food:

- Class Food được kế thừa từ class sprite và sử dụng được các hàm từ sprite.
- **Đầu vào:** position (vị trí của thức ăn).
- **Khởi tạo:**
    - Truyền biến position vào sprite.
    - Gọi hàm draw trong pygame để tạo ra ra chấm tròn có bán kính bằng 7.5 để tạo ra thức ăn
    - Gọi hàm draw kế thừa từ sprite và vẽ ra thức ăn có màu trắng.

e. Cài đặt class Pacman:

- Class Pacman được kế thừa từ class sprite và sử dụng được các hàm từ sprite.
- **Đầu vào:** position (vị trí của pacman)
- **Khởi tạo:**

- Biến DEAD để kiểm tra xem Pacman đã bị ma ăn hay chưa.
- Truyền biến position vào sprite.
- Gọi hàm draw trong pygame để tạo ra chấm tròn có bán kính bằng 15 để tạo ra hình ảnh Pacman.
- Gọi hàm draw được kế thừa từ sprite và vẽ ra Pacman có màu vàng.

f. Cài đặt class Game:

- **Đầu vào:**
  - Matrix: mảng 2 chiều biểu diễn bản đồ.
  - pacman: vị trí khởi tạo của pacman.
- **Khởi tạo:**
  - List food để lấy danh sách thức ăn.
  - List Ghosts lấy danh sách thức ăn
  - Biến Player để lưu vị trí và cập nhật tình trạng của Pacman.
  - Khởi tạo biến Player là một Pacman và duyệt mảng 2 chiều để lấy các giá trị trong map: 1 là tường, 2 là thức ăn, 3 là ma.
  - Hàm checkGameFinish: để kiểm tra thắng hay thua.
  - Hàm pacmanMove: cập nhật vị trí đi tiếp theo của pacman và tính điểm.
  - Hàm ghostMove: cập nhật vị trí đi theo của con ma.
  - Hàm checkColision: để kiểm tra xem ma và pacman có va chạm hay không.
  - Hàm checkEatFood: để kiểm tra xem thức ăn có được ăn hay không.
  - Hàm clearAnimation: để reset lại màn hình mỗi lần cập nhật di chuyển của các object.

g. Cài đặt hàm drawScore:

- **Đầu vào:** Không có đầu vào
- **Mô tả:**
  - Sử dụng thư viện pygame để vẽ ra điểm

h. Cài đặt hàm menu:

- **Đầu vào:** Không có đầu vào
- **Mô tả:**
    - Hàm menu lấy thông tin được xử lí từ file handle_input, bao gồm kích thước của map, cấu trúc map, vị trí của pacman, level người dùng chọn và tên của map.
    - Hàm menu kiểm tra xem người dùng đã nhập vào level nào sau đó sẽ thực hiện theo yêu cầu của từng level sau đó trả về thông tin được xử lí từ file handle_input sau khi được kiểm tra.

i. Cài đặt hàm drawFinish:

- **Đầu vào:** state (trạng thái của trò chơi: WIN hoặc LOSE)
- **Mô tả:**
    - Sử dụng thư viện pygame để vẽ ra chữ WIN (màu vàng) hoặc LOSE (màu đỏ) ở ngay bên dưới mê cung trò chơi,

j. Cài đặt hàm initGameScreen:

- **Đầu vào:** không có đầu vào
- **Mô tả:**
    - Tạo ra hai thành phần screen và clock trong pygame nhằm hỗ trợ vẽ đồ họa trên màn hình.

k. Cài đặt hàm handle input:

- **Đầu vào:** Không có đầu vào.
- **Mô tả:**
    - Yêu cầu người dùng nhập level trong màn hình console.
    - Yêu cầu người dùng nhập tên map trong thư mục Input.
    - Trả về kích thước của map, map, vị trí pacman, level, và tên map.

l. Cài đặt hàm main:

- **Đầu vào:** Không có đầu vào
- **Mô tả:**
    - Cài đặt kích thước cửa sổ của trò chơi bằng với kích thước của map.
    - Gọi hàm initScreenGame để khởi tạo trò chơi.
    - Thực hiện vòng lặp để trò chơi được bắt đầu.

- Trong vòng lặp sẽ kiểm tra tình trạng của pacman, ma, và thức ăn, sau đó cập nhật và vẽ điểm cho đến khi kết thúc trò chơi (pacman xung đột với ma, hoặc pacman đã ăn hết thức ăn).

4. Cài đặt level:

a. Cài đặt level 1:

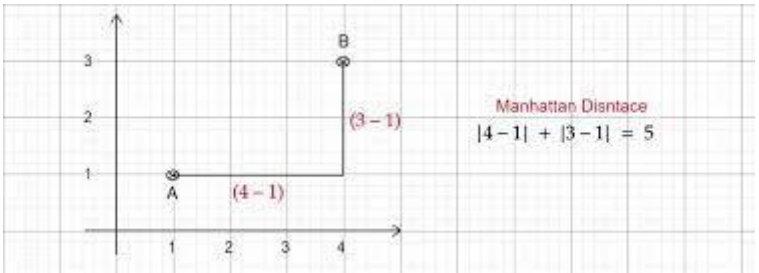
❖ **Thuật toán:** A* Search

❖ **Ý tưởng thực hiện:** Trong level 1, pacman biết được vị trí của thức ăn trên bản đồ và không có sự hiện diện của ma trong map này. Chỉ có một thức ăn trên map. Ta sử dụng thuật toán tìm kiếm A* để giải quyết vấn đề trên. Ta tính chi phí tổng bằng cách tính khoảng cách manhattan cộng với chi phí đường đi của điểm trước cộng 1. Sau đó vị trí tiếp theo sẽ là những vị trí kề với vị trí hiện tại mà có chi phí tổng thấp nhất.

❖ ***Cài đặt:***

Hàm 1: *manhattan_dis()*

- **Input:** start_x là vị trí trục x của vị trí hiện tại, start_y là vị trí của trục y của vị trí hiện tại, des_x là vị trí trục x của mục tiêu, des_y là vị trí của trục y của mục tiêu.
- **Output:** abs(des_x-start_x)+abs(des_y-start_y)
- **Mô tả hàm:**
  - Trả về khoảng cách giữa hai điểm bằng cách sử dụng công thức manhattan:



Hàm 2: *detec_food()*

- **Input:** MAP là map chơi của game,size_x là độ rộng của MAP ,size_y là độ dài của MAP

- **Output:** (i,j) với i là vị trí trục y của thức ăn, j là vị trí trục x của thức ăn.
- **Mô tả hàm:**
    - Dò cả mảng MAP và trả về tọa độ của x và y của thức ăn trong map khi tìm được nó.

Hàm 3: *level_1()*

- **Input:** MAP là map chơi của game ,pos là vị trí của pacman ,size_x là độ rộng của MAP, size_y là độ dài của MAP
- **Output:** path là đường đi đến thức ăn của pacman
- **Mô tả hàm:**
    - Tìm vị trí của thức ăn để tính chi phí tổng cho toàn bộ thuật toán
    - Tính chi phí tổng bằng cách tính khoảng cách manhattan từ điểm bắt đầu đến thức ăn cộng với chi phí đường đi=0 (vì là điểm bắt đầu), cho khoảng cách đó vào queue (dạng priority queue để chi phí thấp nhất lên đầu)
    - Cho hàm vào vòng lặp đến khi queue trống (không có đến thức ăn):
        - Bước 1: Lấy phần tử đầu tiên (v) của hàng đợi queue. Đó là vị trí hiện tại ta xét
        - Bước 2: Kiểm tra vị trí hiện tại của pacman có phải là đang ở vị trí thức ăn không thì làm tiếp bước 3, nếu đúng thì:
            - ➢ Bước 2.1: Thêm vị trí hiện tại vào mảng path.
            - ➢ Bước 2.2: Lấy vị trí ta xét trước đó của vị trí hiện tại trong temp_path. Đồng thời đổi vị trí hiện tại thành ta xét vị trí trước đó của vị trí hiện tại, thêm vị trí hiện tại mới vào path.
            - ➢ Bước 2.3: Lặp bước 2.2 đến khi ta được vị trí hiện tại là vị trí đầu tiên. Sau đó Đảo ngược và trả về đường đến thức ăn (path).
        - Bước 3: Tìm ra vị trí kề xung quanh của vị trí hiện tại (trên, dưới, trái, phải) tránh những điểm kề là tường.

- Bước 4: Xét từng điểm kề của vị trí hiện tại, chỉ xét những vị trí kề không trong những đỉnh đã ghé thăm. Với từng điểm kề ta cộng thêm 1 vào chi phí đường đi của điểm hiện tại, sau đó tính chi phí tổng bằng cách cộng chi phí đường đi đó với khoảng cách manhattan của điểm đó với thức ăn (gọi hàm manhattan_dis()) thêm giá trị đã tính và điểm đó vào queue. Lưu vị trí trước đó của cạnh kề là vị trí hiện tại v (temp_path[cạnh kề v]=v)
- Tiếp tục lại bước 1. Cho đến khi queue trống.

b. Cài đặt level 2:

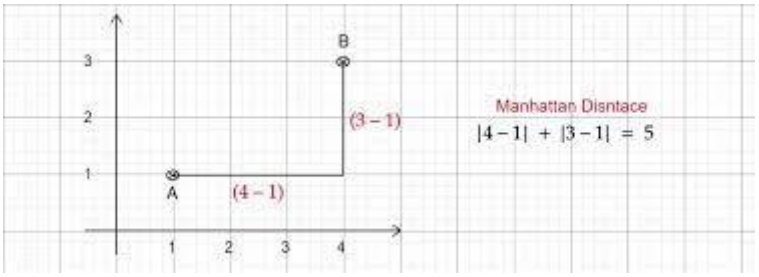
❖ **Thuật toán:** Greedy Best First Search

❖ **Ý tưởng thực hiện:** Trong level 2, pacman biết được vị trí của thức ăn trên bản đồ và có sự hiện diện của nhiều ma trong map này và ma không di chuyển. Chỉ có một thức ăn trên map . Ta sử dụng thuật toán Greedy Best First Search.Tính chi phí tổng bằng cách tính khoảng cách manhattan từ vị trí hiện tại đến thức ăn. Sau đó vị trí tiếp theo sẽ là những vị trí kề với vị trí hiện tại mà có chi phí tổng thấp nhất.

❖ **Cài đặt:**

Hàm 1: *manhattan_dis()*

- **Input:** start_x là vị trí trục x của vị trí hiện tại, start_y là vị trí của trục y của vị trí hiện tại, des_x là vị trí trục x của mục tiêu, des_y là vị trí của trục y của mục tiêu.
- **Output:** abs(des_x-start_x)+abs(des_y-start_y)
- **Mô tả hàm:**
  - Trả về khoảng cách giữa hai điểm bằng cách sử dụng công thức manhattan:



Hàm 2: *detec_food()*

- **Input:** MAP là map chơi của game,size_x là độ rộng của MAP ,size_y là độ dài của MAP
- **Output:** (i,j) với i là vị trí trục y của thức ăn, j là vị trí trục x của thức ăn.
- **Mô tả hàm:**
    - Dò cả mảng MAP và trả về tọa độ của x và y của thức ăn trong map khi tìm được nó.

Hàm 3: *level_2()*

- **Input:** MAP là map chơi của game ,pos là vị trí của pacman ,size_x là độ rộng của MAP, size_y là độ dài của MAP
- **Output:** path là đường đi đến thức ăn của pacman
- **Mô tả hàm:**
    - Tìm vị trí của thức ăn để tính chi phí tổng cho toàn bộ thuật toán
    - Tính chi phí tổng bằng cách tính khoảng cách manhattan từ điểm bắt đầu đến thức ăn), cho khoảng cách đó vào queue (dạng priority queue để chi phí thấp nhất lên đầu)
    - Cho hàm vào vòng lặp đến khi queue trống (không có đến thức ăn):
        - Bước 1: Lấy phần tử đầu tiên (v) của hàng đợi queue. Đó là vị trí hiện tại ta xét
        - Bước 2: Kiểm tra vị trí hiện tại của pacman
        - có phải là đang ở vị trí thức ăn không thì làm tiếp bước 3, nếu đúng thì:
            - ➢ Bước 2.1: Thêm vị trí hiện tại vào mảng path.

➢ Bước 2.2: Lấy vị trí ta xét trước đó của vị trí hiện tại trong temp_path. Đồng thời đổi vị trí hiện tại thành ta xét vị trí trước đó của vị trí hiện tại, thêm vị trí hiện tại mới vào path.

➢ Bước 2.3: Lặp bước 2.2 đến khi ta được vị trí hiện tại là vị trí đầu tiên. Đảo ngược và trả về đường đến thức ăn (path).

- Bước 3: Tìm ra vị trí kề xung quanh của vị trí hiện tại (trên, dưới, trái, phải) tránh những điểm kề là tường.
- Bước 4: Xét từng điểm kề của vị trí hiện tại, chỉ xét những vị trí kề không trong những đỉnh đã ghé thăm. Với từng điểm kề ta tính chi phí tổng bằng cách tính khoảng cách manhattan của điểm đó với thức ăn (gọi hàm manhattan_dis()) thêm giá trị đã tính và điểm đó vào queue. Lưu vị trí trước đó của cạnh kề là vị trí hiện tại v (temp_path[cạnh kề v]=v)
- Tiếp tục lại bước 1. Cho đến khi queue trống.

c. Cài đặt level 3:

❖ **Thuật toán:** Heuristic Local Search

❖ **Ý tưởng thực hiện:** Trong level 3, pacman bị giới hạn khả năng quan sát, với một khung 7x7 với pacman là vị trí trung tâm. Chia khung 7x7 thành 3 vòng quanh pacman với (3x3, 5x5, 7x7).Mỗi khu vực đó được gán giá trị riêng như sau

- 3x3:
  - Gặp thức ăn: +35
  - Gặp ma: -∞
- 5x5:
  - Gặp thức ăn: +10
  - Gặp ma: -∞ (với các vị trí cột hoặc hàng của chúng bằng vị trí của pacman); nếu không thì =-50
- 7x7:
  - Gặp thức ăn: +5
  - Gặp ma: -100

- Sau khi tính các giá trị heuristic tương ứng với mỗi hướng đi, xem xét trừ đi số lần pacman đã đi qua ô đó trong các hướng pacman đã di chuyển, riêng với ô pacman vừa mới đi qua trước đó thì sẽ được -1000 (với mục đích không muốn pacman đi qua đi lại hai ô duy nhất nên không cho phép pacman quay đầu). Tuy nhiên, nếu pacman ở quá gần ma thì sẽ cho phép đi lại ô cũ (-1000<-∞). Trong các heuristic vừa tính được, chọn ra cái lớn nhất.
- Đối với ma chỉ được đi trong vòng 4 ô quanh vị trí ban đầu của nó. Và sau mỗi bước đi của pacman thì tất cả các ma sẽ được đi một ô bất kì.

❖ Cài đặt:

Hàm 1: *plusPadding()*

- **Input:** maze là map chơi của game. Là một tập list gồm các list khác với mỗi list là một hàng của map (trong đó chứa các giá trị 0, 1, 2).
- **Output:** maze (Sau khi được cập nhật)
- **Mô tả hàm:**
  - Tạo một list các giá trị 1 với độ dài bằng chiều rộng của map.
  - Chèn list vừa tạo ra 2 lần ở đầu mảng và 2 lần ở cuối mảng.
  - Chèn thêm giá trị 1 2 lần ở đầu mỗi list con trong maze và 2 lần ở cuối chúng.
  - Tạo ra được một maze mới với kích thước là bằng chiều dài +4 và chiều rộng +4 so với ban đầu.
  - Trả về lại maze vừa được tạo.

Hàm 2: *createNewBoard()*

- **Input:** pacman : tọa độ của pacman hiện tại dưới dạng [x,y]
- **Output:** tileFull: vùng ô 7x7 xung quanh pacman với pacman là vị trí trung tâm. Dưới dạng một ma trận (7,7,2)
- **Mô tả hàm:**
  - Từ vị trí của pacman (pacman là trung tâm) : chạy vòng lặp từ -3 -> 4, trong đó vị trí tọa độ x sẽ là x + i (i =(-3;4)); các giá trị y sẽ là +-3; +-2; +-1; +0.
  - Mối lần lặp như vậy sẽ được lưu dưới dạng: [x+i,y-3],[x+i,y-2],[x+i,y-1],[x+i,y],[x+i,y+1],[x+i,y+2],[x+i,y+3]]

và được lưu vào một list chung.

- Trả về list (7,7,2)

Hàm 3: *createPacmanTile()*

- **Input:** pacman: tọa độ của pacman hiện tại dưới dạng [x,y]
- **Output:** maze: tọa độ 4 ô trên dưới trái phải của pacman.
- **Mô tả hàm:**
    - Từ vị trí của pacman, ta tìm 4 ô xung quanh của pacman.
    - Trả về list vị trí của pacman

Hàm 4: *availableTilePacman()*

- **Input:** board: là map chơi của game; là một tập list gồm các list khác với mỗi list là một hàng của map. maze: là list tọa độ vị trí xung quanh của pacman.
- **Output:** available: là các tọa độ vị trí mà pacman có thể đi được. direction là các hướng tương ứng với các tọa độ đó.
- **Mô tả hàm:**
    - Từ list tọa độ vị trí xung quanh maze, nếu tại vị trí đó trên board là khác 1 thì pacman có thể đi được. Lưu vị trí đó vào list available và lưu hướng đi của đó trên dưới trái phải trong direction.
    - Trả về list vị trí đi được và hướng đi của nó.

Hàm 5: heuristicValue()

- **Input:** tilePacman là khung 7x7 tầm nhìn của pacman; maze là map chơi của gamem, direction là list các hướng di chuyển có thể của pacman (list các string: [“trai”, “phai”, “tren”, “duoi”])
- **Output:** tập hợp các giá trị heuristic của các hướng.
- **Mô tả hàm:**
    - Lần lượt xét qua các hướng đi: tính heuristic tương ứng với mỗi hướng. Xét theo từng khung ô như sau:
        - Nếu gặp thức ăn: 3x3 ô xung quanh pacman: +35; 5x5 ô xung quanh pacman: +10; 7x7 ô xung quanh pacman: +5
        - Nếu gặp ma: 3x3 ô xung quanh pacman: -∞; 5x5 ô xung quanh pacman, trừ các ô không nằm cùng hàng hoặc cùng cột pacman (-50) thì -∞; 7x7 ô xung quanh pacman: -100

- Cộng các giá trị tương ứng mỗi hướng lại.
- Trả về tập hợp giá trị của mỗi hướng.

Hàm 6: *localsearch()*

- **Input:** board: là map chơi game đã được cập nhật; pacman: là vị trí hiện tại của pacman; remembered: là lưu giữ tọa độ của ô vừa đi qua trước đó; visited là vị trí tọa độ của các ô đã từng đi qua.
- **Output:** available[index] : với available là list các ô mà pacman có thể đi, index là vị trí trong list available của ô được chọn -> Ô mà pacman sẽ đi ở bước tiếp theo; remembered: là ô trước đó nó mới đi qua.
- **Mô tả hàm:**
  - Gọi hàm createPacmanTile để lấy được 4 hướng xung quanh pacman.
  - Tạo khung 7x7 tầm nhìn quan sát của pacman với pacman trung tâm bằng hàm createNewBoard.
  - Chọn ra những hướng đi, tọa độ của các ô mà pacman có thể đi được trong 4 hướng ở trên (availableTilePacman)
  - Tính các giá trị heuristic của các hướng có thể đi đó (heuristic value).
  - Tính số lần mà trong 4 ô đó đã được đi qua xuyên suốt game trước đó, nếu như là ô vừa mới đi qua ngay trước đó thì sẽ là +1000.(count)
  - Tương ứng với mỗi hướng đi , lấy giá trị heuristic tìm được - cho giá trị count vừa tính.
  - Tìm giá trị max nhất trong các giá trị heuristic.
  - Tìm ra hướng đi có giá trị max tương ứng đó.
  - Trả lại vị trí tọa độ của bước đi tiếp theo và ô vừa đi qua (chính là cái ô pacman hiện tại ở đầu vào).

Hàm 7: *checkStateGame()*

- **Input:** numfood: số lượng thức ăn; pacman: tọa độ hiện tại của pacman; board: map chơi của game.
- **Output:** 1 hoặc 2
- **Mô tả hàm:**

- Nếu số lượng numfood =0 (nghĩa là đã hết thức ăn) thì sẽ trả về 1. (thắng game)
- Nếu tại vị trí tọa độ pacman đang đứng trên board =3 thì pacman gặp ma trả về 2 (pacman bị ăn nên thua game)

Hàm 8: monsterMove()

- **Input:** currghost: list gồm vị trí hiện tại của tập hợp tất cả các ma; initialGhost: list gồm vị trí ban đầu của các ma; board: là map chơi của game.
- **Output:** newpos: là list gồm các vị trí tiếp theo mà tất cả các ma sẽ đi. Oldpos là list vị trí cũ của các ma.
- **Mô tả hàm:**
  - Oldpos chính là bằng vị trí hiện tại của ma ở đầu vào.
  - Duyệt qua từng con ma, nếu vị trí hiện tại của ma bằng với vị trí gốc của ma: thì ma sẽ có các lựa chọn là đi về 4 hướng bất kì; nếu vị trí của ma hiện giờ khác với vị trí gốc ban đầu thì ma chỉ có một lựa chọn là đi về lại vị trí ban đầu. (Ma chỉ có thể đi trong vòng 5 ô).
  - Xem xét trong các ô đó, nếu các ô đó ma có thể đi được (trên board có giá trị !=1) thì ma sẽ đưa vào list hướng đi ma sẽ đi. Nếu list này có số phần tử lớn 1 thì sẽ chọn random một trong các phần tử đó. Nếu không mặc định là 0 (nghĩa là vị trí của phần tử đầu tiên).
  - Trả về list gồm các vị trí của tất cả các ma , và các ô ma vừa đi ngay trước đó.

Hàm 9: *ingame()*

- **Input:** pacman: là vị trí ban đầu gốc của pacman; board: map của game; currghost: các vị trí ma hiện tại của game; initialGhost: vị trí gốc ban đầu của ma; numfood: số lượng thức ăn.
- **Output:** trả về list gồm chuỗi hoạt động của pacman (actionsForPacman), list gồm chuỗi hoạt động của tất cả các ma (actionsForGhost).
- **Mô tả hàm:**
  - Tạo ra các biến để lưu lại các trị list chuỗi hoạt động của pacman: actionsForGhost, list chuỗi hoạt động của ma:

actionsForGhost, actionPacman: là hoạt động tại 1 thời điểm của pacman; visited là list chuỗi các ô mà pacman đã đi qua; remembered là ô mà pacman vừa mới đi qua trước đó (gán ngay là vị trí ban đầu của pacman).

- Trong vòng lặp while (true):
  - Chèn actionPacman vào list visited để ghi nhớ ô pacman đã đi qua.
  - Sử dụng hàm localsearch để tìm ra hoạt động tiếp theo của pacman.
  - Gắn actionPacman vừa tìm được vào actionsForPacman để đánh dấu lại.
  - Đối với ma, gọi hàm monsterMove để tìm ra đường đi của ma.
  - Trong những đường đi vừa tìm được đó, cập nhật lại vị trí của ma trên board, những ô ma vừa đi vào thì sẽ được gán là 3 (actionGhost); còn đối với các ô cũ trước đó thì sẽ được gán bằng 0 (oldpos).
  - Xét nếu tại vị trí hiện giờ pacman đang đứng là thức ăn (=2) thì số lượng thức ăn sẽ trừ đi 1 (numfood-=1) và cập nhật lại trên board là =0.
  - Sử dụng hàm checkStateGame để kiểm tra xem trò chơi đã kết thúc chưa, nếu trả về 1 thì pacman thắng (Đã ăn hết food); nếu trả về 2 thì ma thắng (pacman bị ma ăn).
  - Trả về lại chuỗi hoạt động của pacman và ma (actionsForGhost, actionsForPacman).

d. Cài đặt Level 4:

❖ **Thuật toán:** Minimax

❖ **Ý tưởng thực hiện:**

➢ Max-value: với mỗi bước đi, hàm max sẽ tìm tọa độ những ô lân cận không phải tường, sau đó gọi hàm min để xem nếu chọn bước đi kế tiếp là ô đó thì những con ma sẽ di chuyển như thế nào. Hàm này sẽ trả về bước đi ngắn nhất để ăn thêm 1 food.

➢Min-value: hàm min sẽ cho quái vật di chuyển hướng về vị trí của pacman, sau đó sẽ gọi hàm max để xem pacman sẽ di chuyển kế tiếp như thế nào.

➢Hàm decision: chạy một vòng lặp sẽ ngưng khi pacman ăn hết food, chạm con ma, hoặc hết đường, trả về đường đi pacman và những con ma đã đi qua.

❖ ***Cài đặt:***

Hàm 1: monsterMove()

- **Input:** map: mảng 2 chiều chứa map của game; monsterPos: vị trí hiện tại của con ma,pacman: vị trí của pacman
- **Output:** vị trí con ma nên đi để đuổi theo pacman
- **Mô tả hàm:**
  - Kiểm tra vị trí trí hiện tại, nếu trùng vị trí với pacman thì đã đụng nha, trả lại vị trí hiện tại
  - Thêm các ô lân cận không phải tường vào một mảng, tính khoảng cách từ các ô đó đến pacman(khoảng cách euclide) lưu vào một mảng, tìm khoảng cách nhỏ nhất trong mảng đó, tìm ô tương ứng khoảng cách ngắn nhất đó để trả về

Hàm 2: isCollide()

- **Input:** pacman:vị trí hiện tại của pacman; monsters: mảng lưu tất cả vị trí của những con ma
- **Output:** True nếu pacman có chạm nhau ma, false nếu không chạm
- **Mô tả hàm:** duyệt mảng monsters kiểm tra xem pacman có chạm nhau với những con ma hay không(tọa độ trùng nhau).

Hàm 3: pacmanMove_max()

- **Input:** map: mảng 2 chiều chứa map của game; currentPos: vị trí hiện tại của pacman; lastPos:vị trí trước đó của pacman; montsers: mảng lưu vị trí của những con ma; numOfFood: số thức ăn còn lại; score: số thức ăn đã ăn; trace hàm lưu lại tọa độ pacman đã đi qua.
- **Output:** một tuple, phần tử đầu tiên là số thức ăn đã ăn, phần tử thứ 2 là tọa độ của những ô trên đường đã đi, phần tử thứ 3 là lý do trả về

- **Mô tả hàm:** là hàm max-value của minimax. Thực hiện theo thứ tự như sau
    - Kiểm tra xem đã đụng ma chưa hoặc quãng đường đã đi có quá xa(tránh max recursion depth), nếu có thì trả về với nguyên do là "collide"
    - Kiểm tra vị trí hiện tại có thức ăn hay không, nếu có thì "ăn" bằng cách gán lại tọa độ hiện tại trong mảng map = 0 rồi trả về với nguyên do là "found 1 food"(mục đích của hàm là tìm đường ngắn nhất để ăn thêm 1 thức ăn)
    - Kiểm tra có hết thức ăn chưa(numOfFood = 0), nếu có thì trả về với nguyên do là "out of food"
    - Thêm các ô lân cận không phải tường vào một mảng option, sau đó kiểm tra xem có con ma nào đang đứng trên ô đó không, nếu có thì bỏ ô đó ra khỏi mảng.
    - Nếu không có ô nào có thể đi được thì trả về với nguyên do là "no option"
    - Gọi hàm pacmanMove_min truyền vào những ô trong mảng option là tham số currentPos, và vị trí hiện tại là lastPos và lựa chọn ô nên đi để quãng đường để ăn thêm 1 food là ngắn nhất trong những kết quả trả về.

Hàm 4: pacmanMove_min()

- **Input:** map: mảng 2 chiều chứa map của game; currentPos: vị trí hiện tại của pacman; lastPos:vị trí trước đó của pacman; montsers: mảng lưu vị trí của những con ma; numOfFood: số thức ăn còn lại; score: số thức ăn đã ăn; trace hàm lưu lại tọa độ pacman đã đi qua.
- **Output:** một tuple, phần tử đầu tiên là số thức ăn đã ăn, phần tử thứ 2 là tọa độ của những ô trên đường đã đi, phần tử thứ 3 là lý do trả về(vốn là kết quả trả về của hàm pacmanMove_max() được gọi bên trong)
- **Mô tả hàm:** Tương đương min-value của minimax. duyệt mảng monsters, gọi hàm monsterMove truyền vào các phần tử trong mảng monster để cập nhật vị trí mới cho những con ma(đuổi theo pacman). Sau đó gọi lại hàm pacmanMove_max() truyền vào mảng

monster chứa giá trị mới rồi trả về kết quả mà hàm pacmanMove_max() trả về.

Hàm 5: level4()

- **Input:**map: mảng 2 chiều chứa map của game;montsers: mảng lưu vị trí của những con ma;pacman : vị trí ban đầu của pacman.
- **Output:** một tuple với phần tử đầu là số lượng thức ăn pacman đã ăn được đến khi hết trò chơi, phần tử thứ 2 là tọa độ của các ô trên quãng đường mà pacman đã đi qua, phần tử thứ 3 là mảng 2 chiều, mỗi 1 dòng của mảng 2 chiều này là tọa độ của các ô trên quãng đường mà 1 con ma đi qua, phần tử thứ tư là nguyên nhân khiến game kết thúc
- **Mô tả hàm:** Tương đương hàm decision của minimax. thực hiện theo thứ tự sau
    - khởi tạo các biến để lưu kết quả trả về: monstersMoveList để lưu bước đi của những con ma(phần tử thứ 3 trong tuple trả về), pacmanMoveList để lưu bước đi của pacman(phần tử thứ hai của tuple trả về), numEaten số lượng thức ăn đã ăn(phần tử thứ nhất của tuple trả về)
    - Với biến monstersMoveList append thêm n dòng(n là số lượng ma), mỗi dòng chứa tọa độ ban đầu của 1 con ma. Với biến pacmanMoveList, append thêm tọa độ ban đầu của pacman vào.
    - Chạy vòng lặp while khi mà numOfFood vẫn còn lớn hơn 0.
    - với mỗi bước lặp, gọi hàm pacmanMove_max() và lưu kết quả vào biến output, trừ numOfFood đi một lượng bằng lượng food mà pacman đã ăn sau khi chạy hàm pacmanMove_max(), đồng thời cộng numEaten một lượng tương đương. Thêm các ô mà pacman đã đi qua sau khi chạy hàm pacmanMove_max() vào mảng pacmanMoveList(bỏ đi tọa độ đầu vì trùng với tọa độ cuối trong mảng hiện tại). Dựa vào các ô mà pacman đã đi qua thêm các ô mà con ma sẽ đi để đuổi theo pacman vào mảng monstersMoveList bằng cách gọi hàm monsterMove() và truyền vào lần lượt những tọa độ mà pacman đã đi qua rồi thêm vào từng dòng.

- Nếu pacmanMove_max trả về kết quả là va chạm với ma "collide" trước khi ăn hết thức ăn thì break vòng while, kết thúc game. Tương tự với "no option"
- Sau khi kết thúc vòng while thì trả về kết quả như đã nói ở trên.

IV. Quá trình thực thi:

1. Cách sử dụng:

1: Bật console cùng cấp với thư mục main.py

2: Gọi hàm main,py

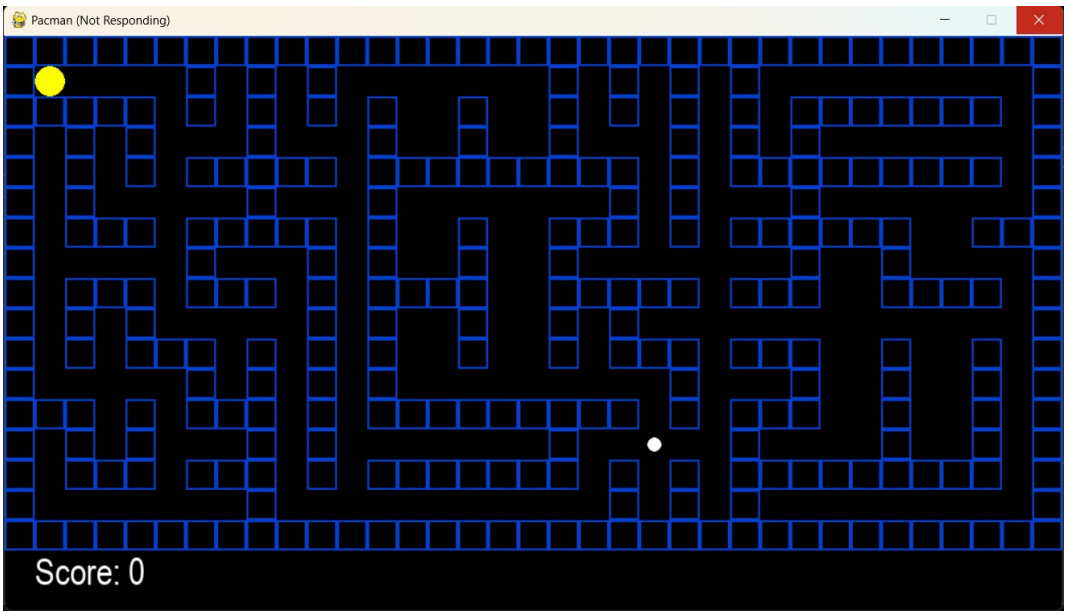
3: Người dùng nhập level trong màn hình console.

4: Nhập tên map trong thư mục Input: (ví dụ: 'map1_level1.txt' - dùng cho level 1)

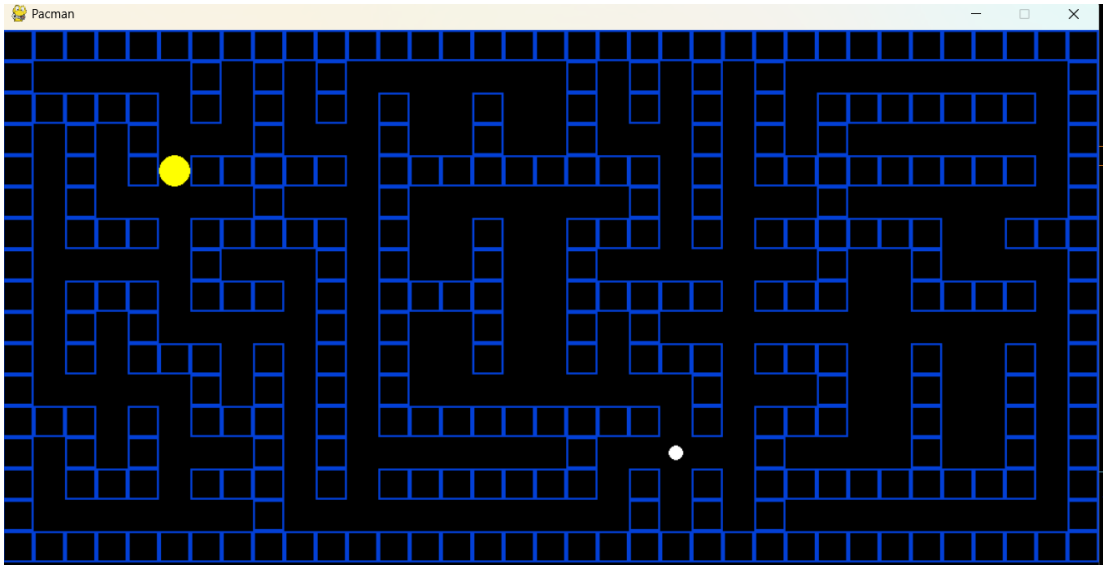
3: Thuật toán sẽ chạy và hiển thị quá trình lên trên màn hình (với pacman màu vàng, ghost là màu đỏ, thức ăn là màu trắng)

2. Quá trình chạy:

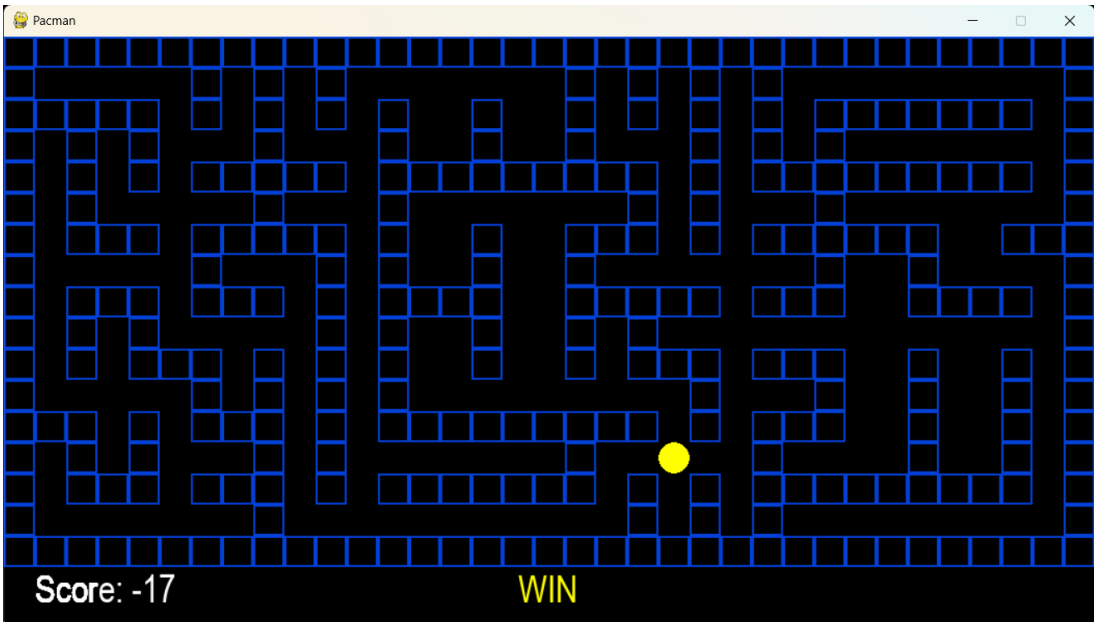
Level 1: (map1_level1.txt)
Ban đầu:



Đang chạy:

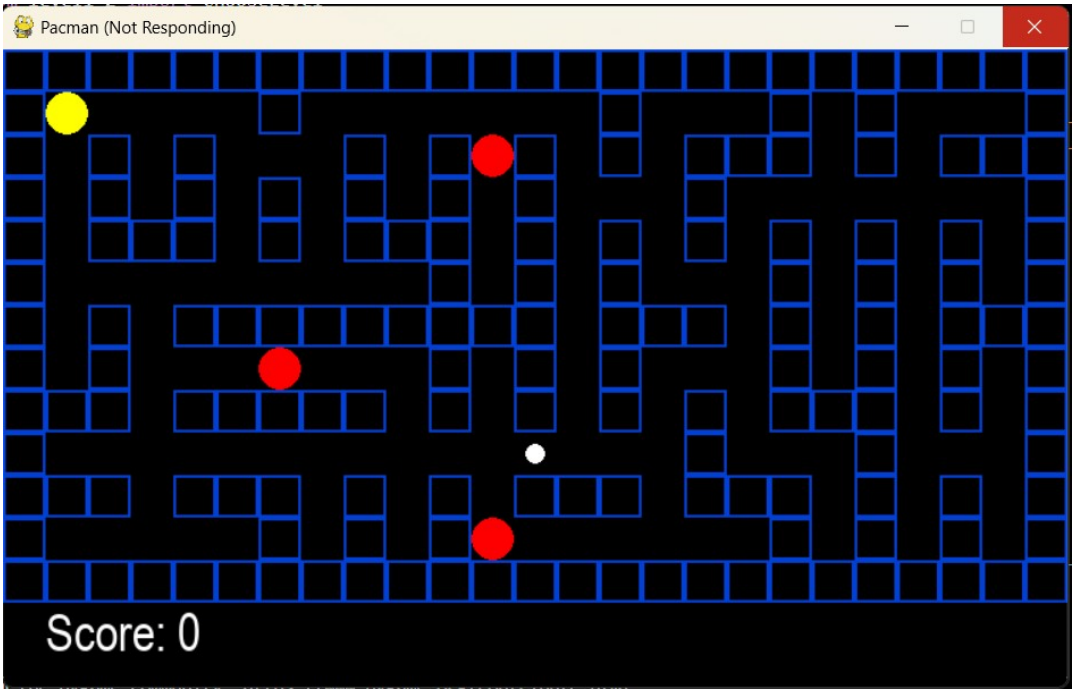


Kết thúc:

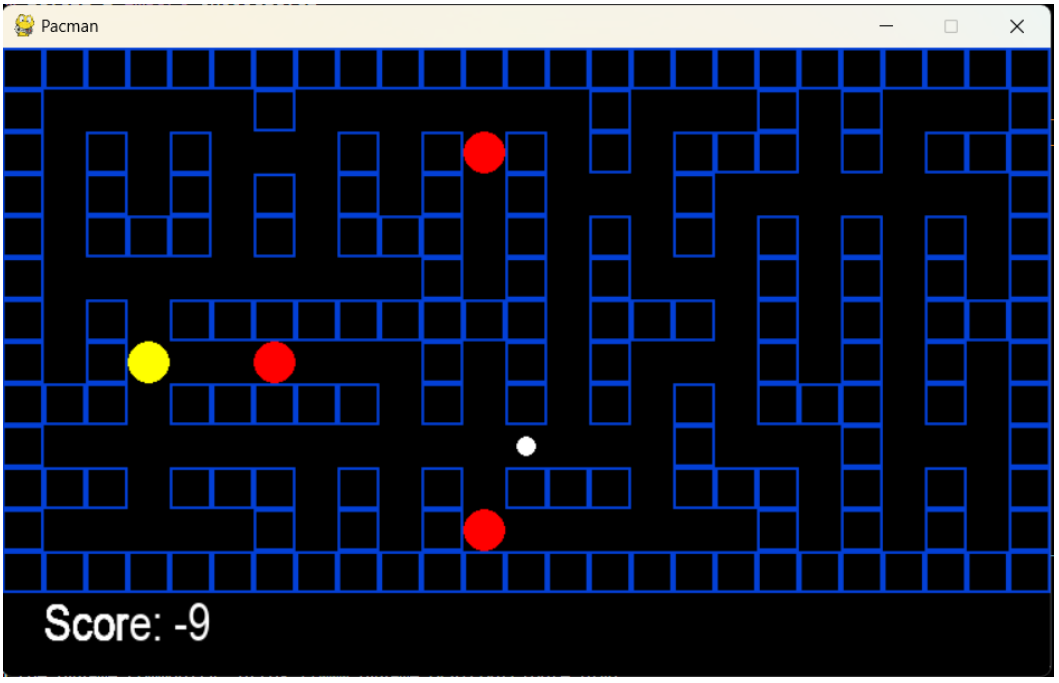


Level 2: (map1_level2.txt)

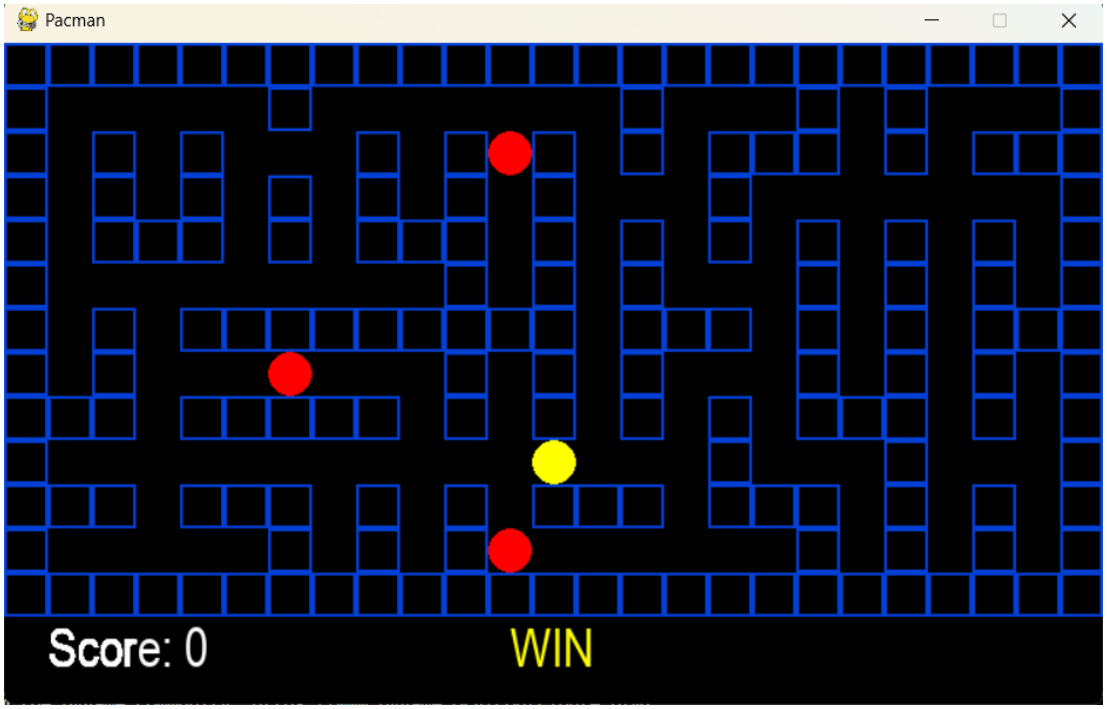
Ban đầu:



Đang chạy:

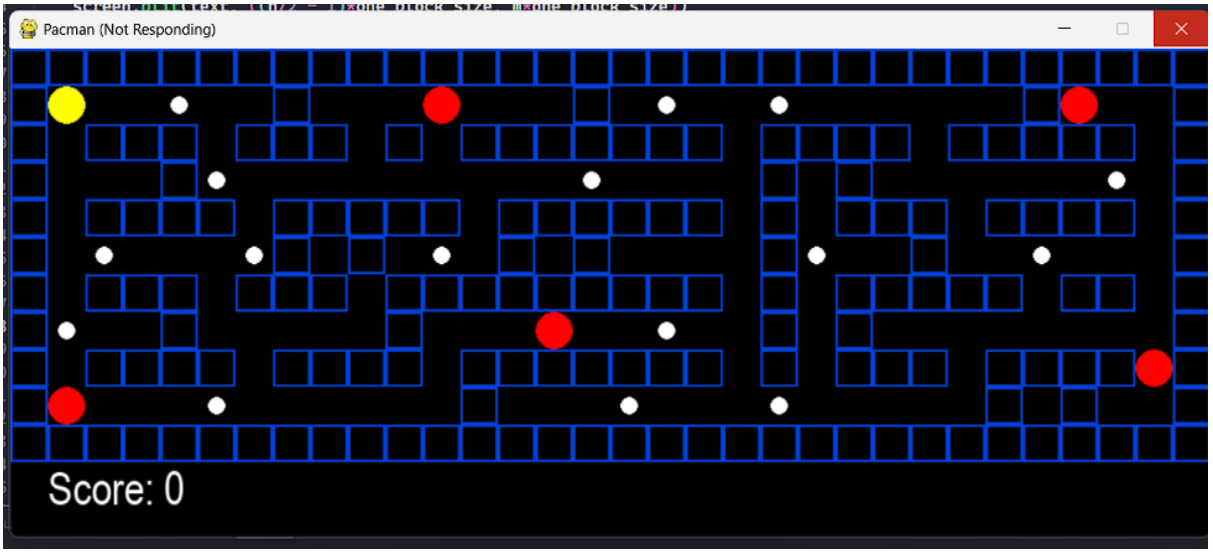


Kết thúc

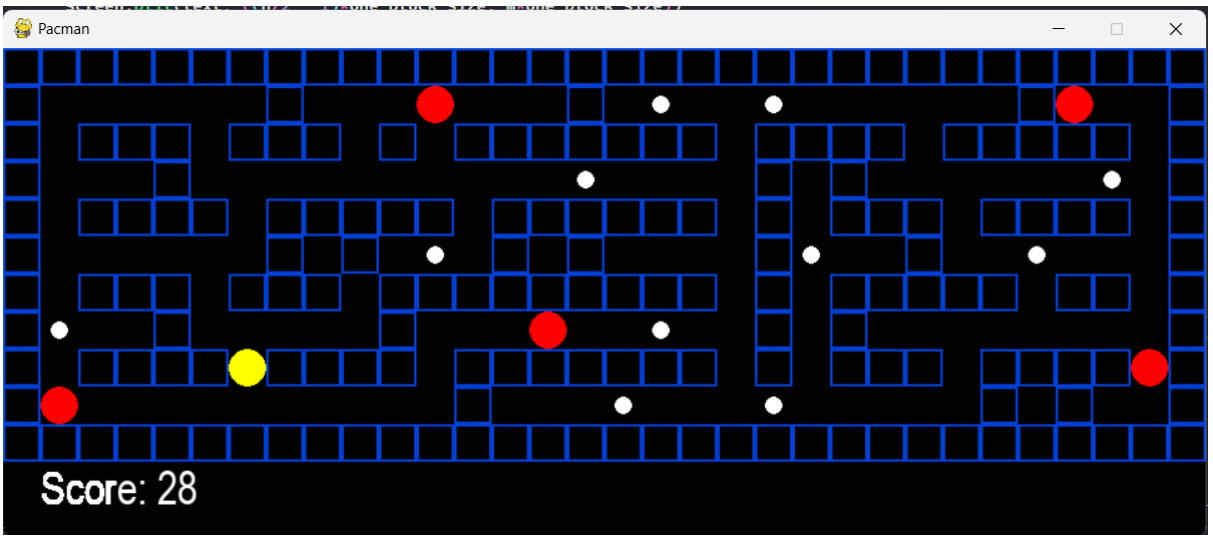


Level 3: (map1_level3.txt)

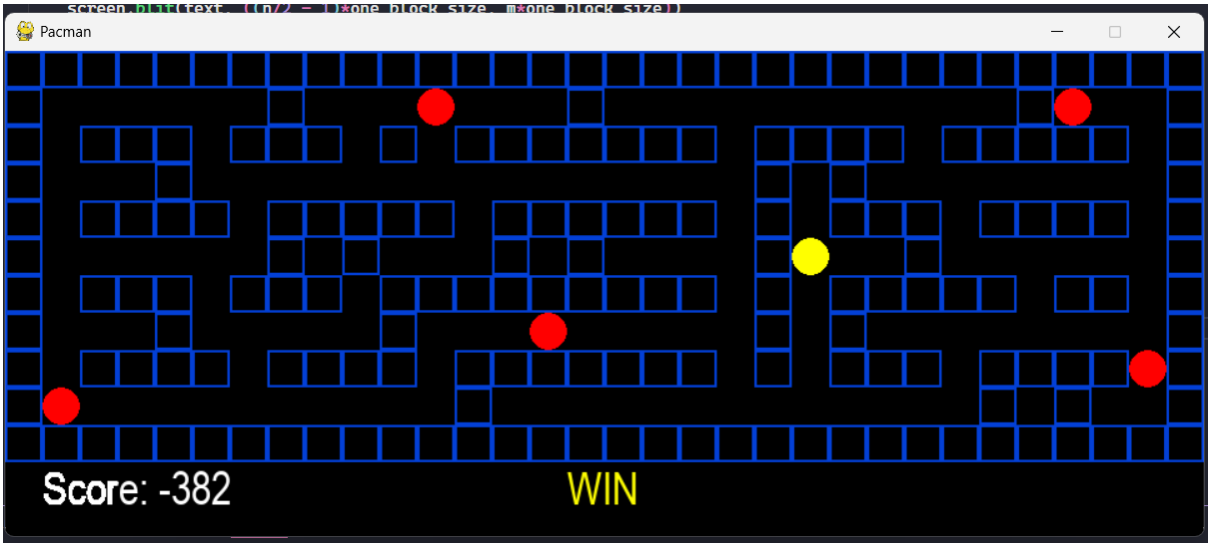
Ban đầu:



Đang chạy:

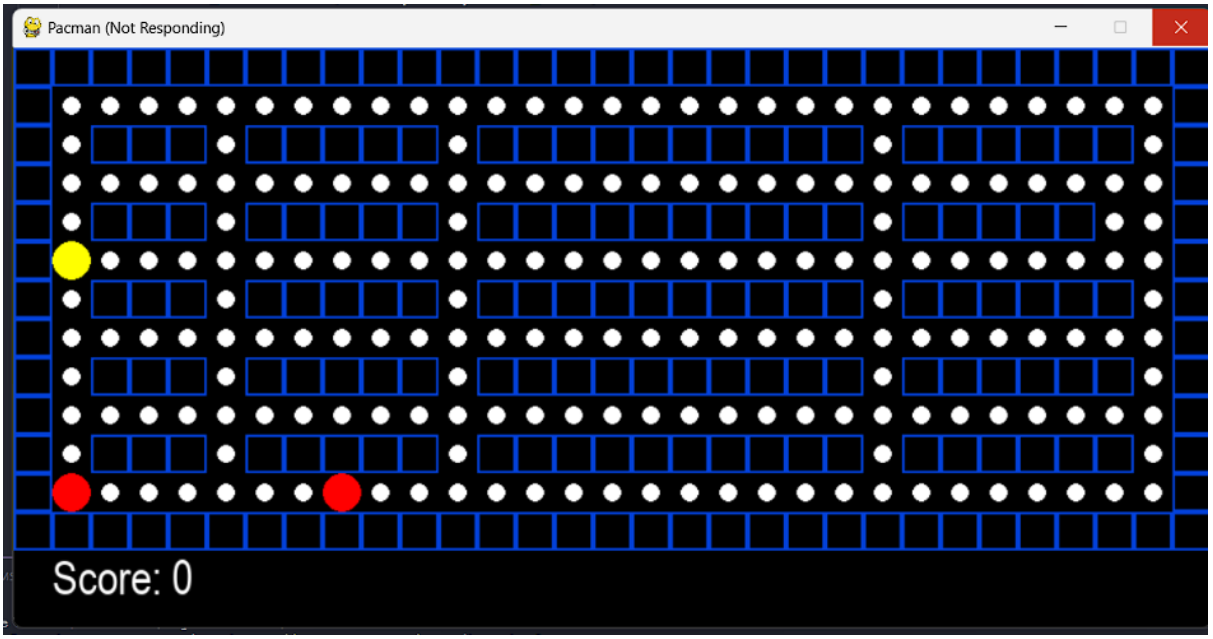


Kết thúc:

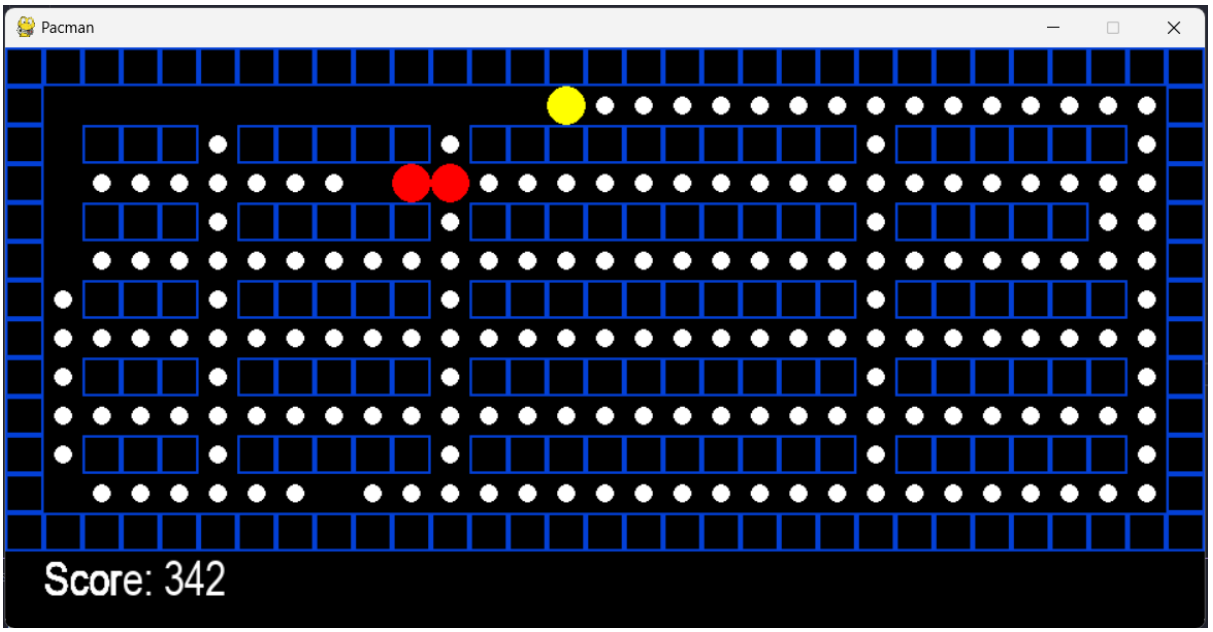


Level 4: (map1_level4.txt)

Ban đầu:



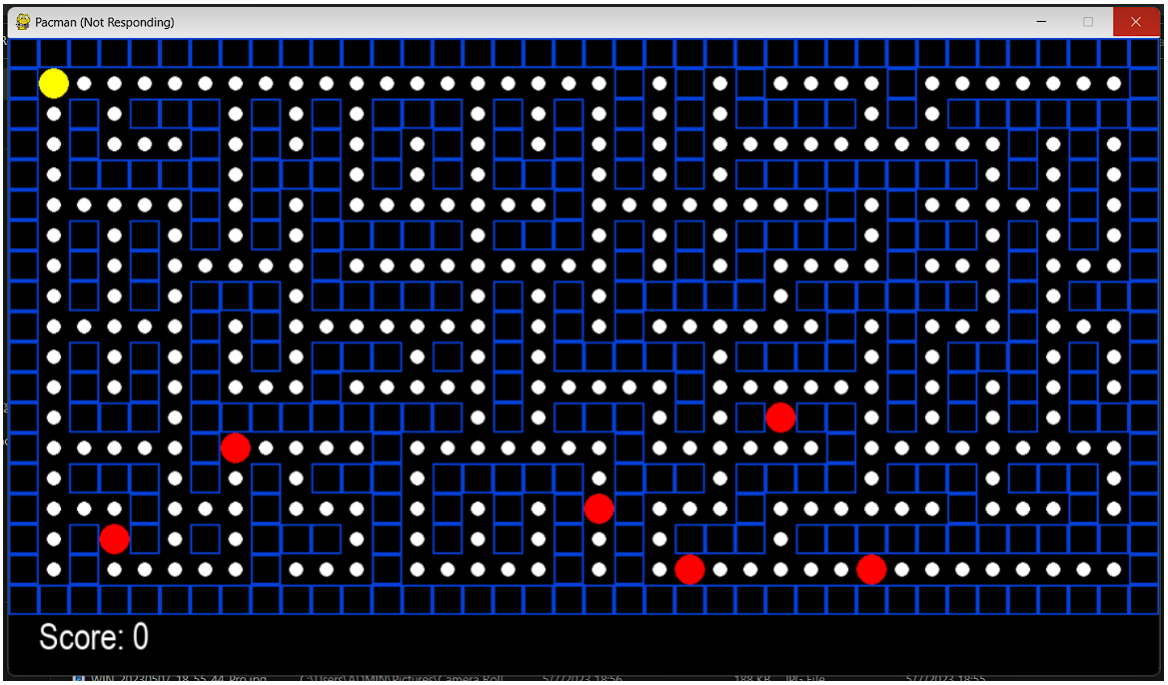
Đang chạy:



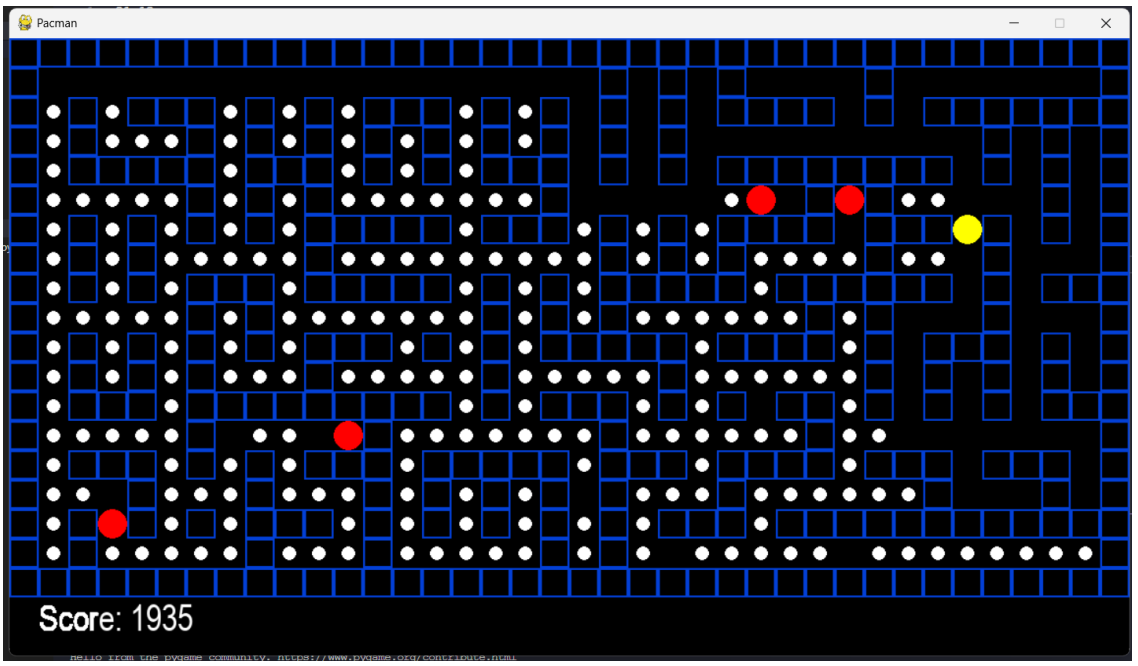
Kết thúc

Level 4: (map2_level4.txt)

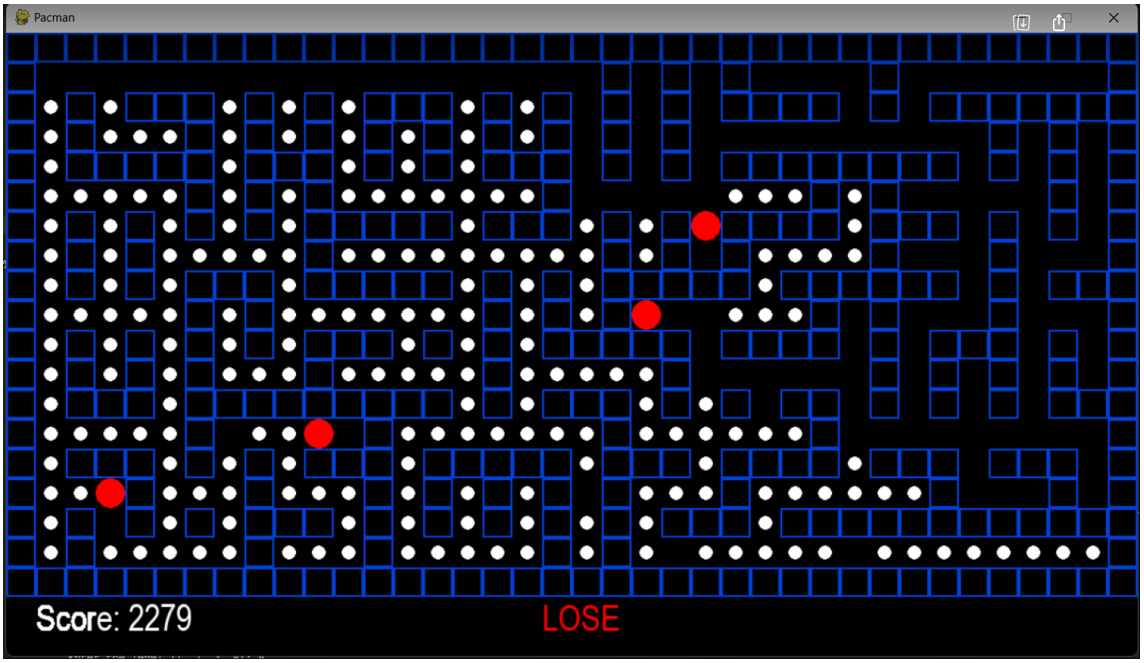
Ban đầu:



Đang chạy:



Kết thúc



V. Đánh giá thuật toán:

- Level 1 sử dụng thuật toán A* (tối ưu).
- Level 2 sử dụng thuật toán Greedy Best First Search:
  - Chắc chắn tìm được path nhưng chưa tối ưu.
- Level 3 sử dụng hàm heuristic để đánh giá bước đi (GBFS):
  - Path tìm được thay đổi sau mỗi lần đi, có thể đưa ra path khiến Pacman đi vào ngõ cụt, lặp nhiều lần và thua game (chết) khi chưa ăn hết được food.

    - Path tìm được là không tối ưu do tính chất của thuật toán.
    - Hàm heuristic dùng để đánh giá chưa tốt và cần cải thiện.
- Level 4 sử dụng thuật toán minimax:
    - Chạy khá tốt, tuy nhiên vẫn còn hạn chế khi pacman đi vào những ngõ cụt dẫn đến ghost có thể ăn được pacman.
    - Các đối tượng di chuyển không thật sự giống game khi ghost chỉ đuổi theo pacman.

VI. Các nguồn tài liệu tham khảo:

Graphic: Các Yếu Tố Lập Trình Game Cơ Bản Với Pygame (codelearn.io)

Level 3: ai-pacman/Source/HeuristicLocalSearch.py at master · kieuconghau/ai-pacman (github.com)

Level 1, 2 & 4: slides, learning materials, moodle