ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN 01

SEARCH

Nhóm

Trần Huy Vũ	18127257
Trần Bùi Tài Nhân	18127168
Phan Nhật Minh	18127153
Tô Đông Phát	18127176

Môn học: Cơ sở trí tuệ nhân tạo

Thành phố Hồ Chí Minh – 2020

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN 01

| Đề tài |

SEARCH PACMAN

| Giáo viên hướng dẫn |

ThS. Lê Ngọc Thành

Môn học: Cơ sở trí tuệ nhân tạo

Thành phố Hồ Chí Minh – 2020

LỜI CẢM ƠN

Để hoàn thành được báo cáo này nhóm em xin gởi lời cảm ơn đến quý thầy cô khoa Công Nghệ Thông Tin trường đại học Khoa Học Tự Nhiên.
Đặc biệt, nhóm xin gởi đến thầy Lê Ngọc Thành, người đã tận tình giúp đỡ chúng em hoàn thành đồ án này, lời cảm ơn chân thành và sâu sắc nhất. Nếu không có những lời hướng dẫn của thầy cô, có lẽ chúng em không thể hoàn thành đồ án tốt như vậy.
Trong quá trình học tập và làm việc, không thể tránh những sai sót, rất mong các thầy cô bỏ qua. Đồng thời, vì lượng kiến thức và kinh nghiệm thực tiễn còn hạn chế nên báo cáo có nhiều thiếu sót, nhóm em rất mong nhận được những góp ý của thầy cô để hoàn thiện hơn.
Chúng em xin chân thành cảm ơn!

MỤC LỤC

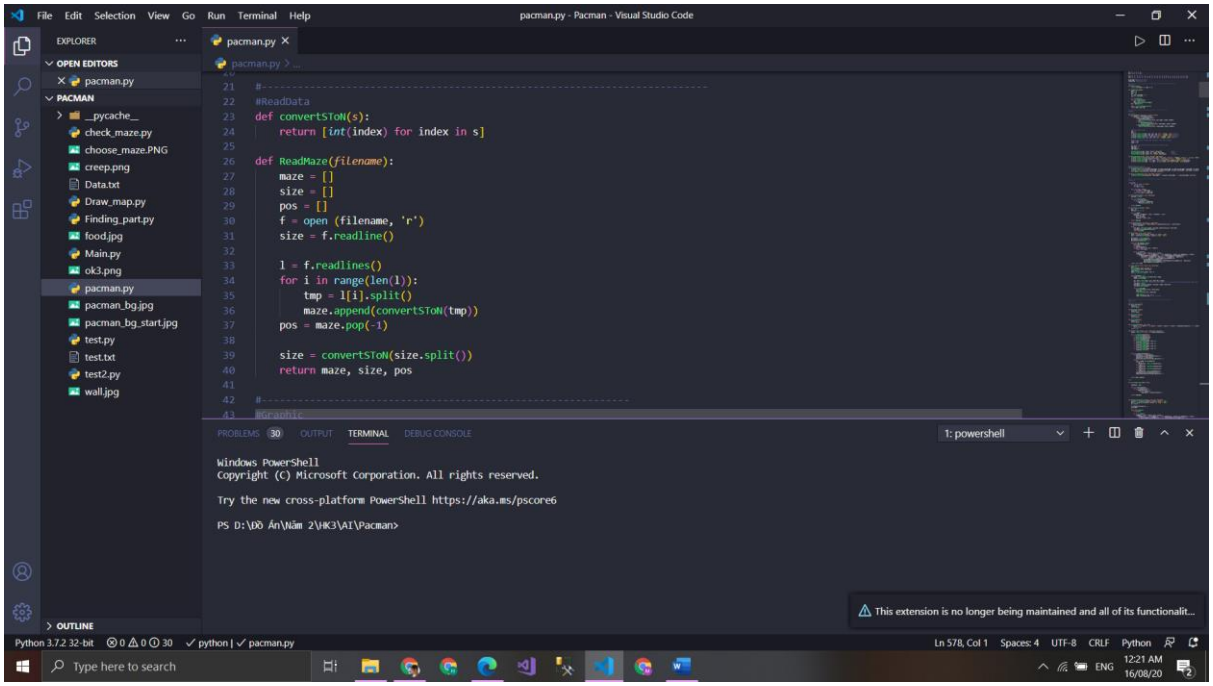
DANH MỤC HÌNH

KẾ HOẠCH PHÂN CÔNG

Nhiệm vụ	Trần Huy Vũ	Trần Bùi Tài Nhân	Phan Nhật Minh	Tô Đông Phát
Tạo file maze	0%	0%	0%	10%
Đồ họa	Menu, buttons, credit, vẽ tường, thức ăn, quái, pacman, pacman di chuyển 10%	0%	0%	0%
Level 1	0%	0%	Giúp đỡ hoàn thành thuật toán và code 5%	Thuật toán A* 10%
Level 2	Thuật toán A* 10%	Giúp đỡ hoàn thành thuật toán và code 5%		0%
Level 3	0%	0%	Thuật toán A* 10%	0%
Level 4	0%	Thuật toán A* 10%	0%	0%
Báo cáo	5% viết ý tưởng level 2, viết hướng dẫn sử dụng	10% viết ý tưởng level 3, chi tiết các hàm, hoàn thành báo cáo	10% viết ý tưởng level 4, chi tiết các hàm, hoàn thành báo cáo	5% viết ý tưởng level 1

MÔI TRƯỜNG LẬP TRÌNH VÀ THƯ VIỆN

Nhóm em quyết định sử dụng VS code là môi trường lập trình chính



Hình 1. môi trường lập trình

Một số thư viện bên ngoài mà nhóm sử dụng:

- Tkinter
- Turtle
- PIL
- numpy
- queue

Ý tưởng thuật toán

1. Đồ họa

Đầu tiên khi vào chương trình sẽ in ra hình ảnh đại diện cho trò chơi (PACMAN), chia ra 3 mục nhỏ để lựa chọn gồm maze, credit và exit. Chọn maze để xem một số mê cung có thể sử dụng trong chương trình. Sau đó nhập tên maze mà mình muốn, sẽ in ra backgroud khác với các button lựa chọn level (1, 2, 3, 4). Nhấn chọn level và được chuyển vào game với level và maze tương ứng.

Khi vẽ map sẽ tạo ra map với kích thước được tính toán bằng số ô của tường (chiều dài, chiều rộng) cộng thêm 1 khoảng trống để hiển thị điểm số, thời gian và chiều dài.

Các đối tượng thức ăn, tường, quái vật, sẽ được hiển thị dựa trên việc đọc file txt với 1 cho tường, 2 cho thức ăn và 3 cho quái vật. Còn 0 là đường mà pacman hay quái vật có thể di chuyển.

Sử dụng công cụ chính là tkiner, ngoài ra còn có PIL để có thể đọc những file hình ảnh png hoặc jpg vào chương trình.

Chương trình hiển thị trên Canvas – 1 vùng hình chữ nhật dùng để vẽ và sắp xếp bố cục cho nhữn thành phần trên nó.

Đồ họa được tạo ra bằng cách cập nhật lại hình ảnh dựa trên sự thay đổi của mảng maze được đọc từ file txt như trên (quái, đồ) và sự thay đổi vị trí của pacman.

Sau mỗi bước, chương trình sẽ cập nhập lại mảng maze dựa theo thuật toán của chương trình, sau đó mảng maze được đưa vào các hàm đồ họa. Vị trí của Pacman cũng được cập nhập lại, rồi đưa vào hàm đồ họa dưới dạng 1 list có 2 phần tử là tọa độ của Pacman [x,y]

Hàm đồ họa sau khi nhận được mảng maze mới được cập nhập sẽ xóa toàn bộ các thành phần trên trên Canvas hiện tại, rồi vẽ lại các thành phần dựa trên mảng maze.

Ví dụ:

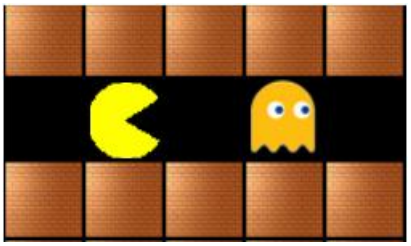
Pacman_pos = [1,1]

```
1 1 1 1 1
0 2 0 0 3
1 1 1 1 1
```

Hình 2. ví dụ 1

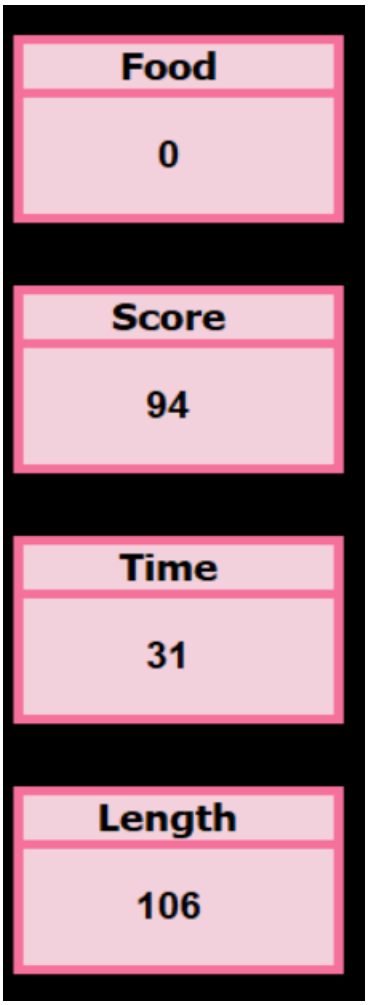
Sau khi update:
Pacman_pos = [2, 1]

```
1 1 1 1 1
0 0 0 3 0
1 1 1 1 1
```



Hình 3. ví dụ 2

Điểm số, thời gian di chuyển, đoạn đường di chuyển và số food còn lại phải ăn sẽ được hiện thị bên phải màn hình.



Hình 4. giao diện thông số

2. **Level 1**

   Pac-man thấy được cả bản đồ và foods, không có monsters. Ý tưởng cho level này là đầu tiên, Pac-man sẽ tìm food ở gần mình nhất bằng cách so sánh khoảng cách Manhattan từ nó đến các foods, sau đó Pac-man sử dụng thuật toán A* để tìm đường đi ngắn nhất từ nó đến food vừa tìm được đến đó để ăn foods. Sau khi ăn xong food, nó tiếp tục tìm và ăn các foods còn lại cho đến khi hết foods.

3. **Level 2**

   Ở level này, bản đồ bắt đầu xuất hiện monsters, nhưng monster này chỉ đứng yên và không di chuyển, vì thế ta có thể coi nó như tường và thực hiện ý tưởng giống như level 1.

4. **Level 3**

   Pac-man có tầm nhìn 3 bước xung quanh nó, và nó sẽ không thấy foods nếu ở ngoài tầm nhìn. Ở level 3, monsters sẽ chỉ di chuyển xung quanh vị trí gốc của nó 1 bước. Mỗi bước pacman đi tương đương với 1 bước di chuyển của monsters.

   Nhiệm vụ của Pac-man là đi khám phá bản đồ để tìm thức ăn và né các mosters.

   Ý tưởng thuật toán:

   a) Pac-man:

      Vì Pac-man biết được vị trí các monsters, nên nó sẽ chủ động tránh né các các ô xung quanh vị trí gốc của monsters.

      Đầu tiên Pac-man sẽ tìm kiếm trong tầm nhìn xem có foods hay không, nếu có, nó sẽ sử dụng thuật toán A* để tìm đường đi ngắn nhất để đến ăn foods.

      Nếu không có foods, Pac-man sẽ tìm và đi đến ô gần nhất chưa đến ngoài tầm nhìn bằng thuật toán BFS.

      Pac-man sẽ tiếp tục khám phá bản đồ và ăn foods, đến khi ăn hết foods thì thắng, hoặc đụng monsters thì thua.

   b) Monsters:

      Monsters sẽ random những vị trí xung quanh vị trí gốc 1 ô để di chuyển.

5. **Level 4**

   Ở level 4, Pac-man sẽ ở trong bản đồ đóng. Nghĩa là Pac-man sẽ không thấy foods, monsters và bản đồ. Monsters sẽ đi săn Pac-man. Nhiệm vụ của nó là đi khám phá bản đồ, ăn nhiều foods nhất có thể và tránh bị các monsters đụng. Mỗi bước Pac-man đi tương đương với 1 bước di chuyển của monsters.

   Ý tưởng thuật toán: Ưu tiên hàng đầu của Pac-man là sống sót, vì vậy, đầu tiên, Pac-man sẽ kiểm tra trong tầm nhìn xem có monsters hay không, nếu có, PACMAN sẽ tiên đoán đường đi của monsters bằng cách sử dụng thuật toán A* để tìm đường đi ngắn nhất từ monsters đến nó, sau đó nó sẽ né những ô ở bước đi tiếp theo - đường mà nó đã tiên đoán monsters sẽ đi. Tiếp theo, Pac-man sẽ kiểm tra trong tầm nhìn xem có foods hay không, nếu có, nó sẽ sử dụng thuật toán A* để tìm đường đi ngắn nhất đến foods và không đi qua những ô đã chủ động né monsters. Nếu không có foods, Pac-man sẽ tìm và đi đến ô gần nhất chưa đến ngoài tầm nhìn bằng thuật toán BFS để khám phá bản đồ. Nếu như PACMAN không tìm được đường đi khác khi né những ô đó đi thì PACMAN sẽ mở rộng dần những ô cần né đó và tìm đường, vẫn không được thì có nghĩa là PACMAN hết đường sống. Các monsters được trang

bị thuật toán A* để tìm đường đi ngắn nhất từ nó đến Pac-man và sẽ chạy theo đường đó để đi săn Pac-man. Cứ mỗi bước di chuyển, Pac-man và quái sẽ phải chạy lại thuật toán để tiếp tục thực hiện nhiệm vụ của mình vì lúc này vị trí của cả 2 đã thay đổi.

Một số hàm nổi bật

1. Các hàm đọc dữ liệu và đồ họa

def draw_map(maze, maze_size, cv_first, level)	maze: mảng mê cung maze_size:list kích thước của mảng [x, y] cv_first: biến quản lý Canvas level: level đang chọn ➔ Vẽ map gồm tường, thức ăn, quái
def draw_pacman(cv_first, cur_pos)	cv_first: biến quản lý Canvas Cur_pos: list vi trí hiện tại của Pacman [x, y] ➔ Vẽ Pacman ở vị trí [x, y]
def Draw_of_something(cv_first, tmp_image, tmp_pos)	cv_first: biến quản lý Canvas tmp_image: hình ảnh muốn vẽ tmp_pos: vị trí muốn vẽ hình ảnh [x, y] ➔ Vẽ thành phần dựa trên vị trí [x, y]

```
def convertSToN(s):
    return [int(index) for index in s]

def ReadMaze(filename):
    maze = []
    size = []
    pos = []
    f = open (filename, 'r')
    size = f.readline()

    l = f.readlines()
    for i in range(len(l)):
        tmp = l[i].split()
        maze.append(convertSToN(tmp))
    pos = maze.pop(-1)

    size = convertSToN(size.split())
    return maze, size, pos
```

Hình 5. hàm đọc file

Hàm đọc file, đọc lần lượt từng giá trị của file đó, gán vào 1 list được khởi tạo là maze. Sau đó trả về các giá trị gồm maze, size (độ dài của file), pos (vị trí bắt đầu của pacman)

```
def draw_map(maze, maze_size, cv_first, level):
    for height in range(maze_size[0]):
        for width in range(maze_size[1]):
            if maze[height][width] == 1:
                Draw_of_something(cv_first, wall_image, (width, height))
            if level < 3:
                if maze[height][width] == 2:
                    Draw_of_something(cv_first, food_image, (width, height))
            if maze[height][width] == 3:
                Draw_of_something(cv_first, creep_image, (width, height))
```

Hình 6. hàm vẽ bản đồ

```
def Draw_of_something(cv_first, tmp_image, tmp_pos):
    cv_first.create_image(tmp_pos[0] * grid_edge + 1, tmp_pos[1]*grid_edge + 1, image=tmp_image, anchor=NW)
```

Hình 7. hàm vẽ theo yêu cầu

2 Hàm dùng để vẽ thức ăn, tường, quái vật khi đọc trong file được các giá trị tương ứng. Hàm truy xuất mảng maze, xét từng phần tử trong mảng, rồi truyền vào hình ảnh và vị trí tương ứng tới hàm Draw_of_something để vẽ hình ảnh tương ứng tại vị trí đó

```
def draw_pacman(cv_first, cur_pos):
    cv_first.create_oval(cur_pos[1]*grid_edge, cur_pos[0]*grid_edge, cur_pos[1]*grid_edge + grid_edge, cur_pos[0]*grid_edge + grid_edge, fill='yellow')
    cv_first.create_arc(cur_pos[1]*grid_edge, cur_pos[0]*grid_edge, cur_pos[1]*grid_edge + grid_edge, cur_pos[0]*grid_edge + grid_edge,
    fill='black', style=PIESLICE, start=330, extent=60)
```

Hình 8. hàm vẽ pacman

Hàm dùng để vẽ pacman, với việc vẽ 1 hình tròn màu vàng và hình tam giác màu đen cho cái miệng ở vị trí [x, y]

2. Hàm tìm kiếm

```
107    def Find_Something(maze, size, Something):
108        Position = list()
109
110        for i in range(size[0]):
111            for j in range(size[1]):
112                if maze[i][j] == Something:
113                    Position.append([i, j])
114
115        return Position
```

Hình 9. hàm tìm kiếm

Hàm này tìm kiếm và trả về vị trí của thứ mình muốn tìm trên bản đồ. Với foods thì cần đưa vào ‘Something’ với giá trị là 2 hoặc monsters thì giá trị sẽ là 3.

```python
def Find_Nearest_Food(Pos, Foods):
    Min = 1e9
    Food_pos = [ -1, -1]

    for i in Foods:
        distance = abs(Pos[0] - i[0]) + abs(Pos[1] - i[1])
        if distance < Min:
            Min = distance
            Food_pos = [i[0], i[1]]

    return Food_pos
```

Hình 10. hàm tìm kiếm thức ăn

Hàm tìm kiếm foods gần nhất so với Pac-man và trả về vị trí của nó.

```python
def AStart(maze, size, Start, Goal, type):
    path = (-1)*np.ones((size[0], size[1], 2), dtype = 'int')
    dist = 100000*np.ones((size[0], size[1]), dtype = 'int')

    pq_frontier = PriorityQueue()
    pq_frontier.put(Node(Start, 0))
    dist[Start[0]][Start[1]] = 0

    while not pq_frontier.empty():
        top = pq_frontier.get()
        u = top.id
        w = dist[u[0]][u[1]]
        if u[0] == Goal[0] and u[1] == Goal[1]:
            return True, path

        for i in range(4):
            neighbor = [u[0] + dr1[i], u[1] + dc1[i]]
            if neighbor[0] > -1 and neighbor[1] > -1 and neighbor[0] < size[0] and neighbor[1] < size[1]:
                if maze[neighbor[0]][neighbor[1]] == 0 or (maze[neighbor[0]][neighbor[1]] == 2 and type == 0) or (maze[neighbor[0]][neig
                    heuristic = abs(neighbor[0] - Goal[0]) + abs(neighbor[1] - Goal[1])
                    if w + 1 + heuristic < dist[neighbor[0]][neighbor[1]]:
                        dist[neighbor[0]][neighbor[1]] = w + 1
                        pq_frontier.put(Node(neighbor, dist[neighbor[0]][neighbor[1]] + heuristic))
                        path[neighbor[0]][neighbor[1]] = u
    return False, path
```

*Hình 11. thuật toán A**

Hàm thuật toán A* tìm kiếm đường đi ngắn nhất giữa 2 điểm. Áp dụng thuật toán loang dầu để đưa các ô xung quanh vị trí khởi đầu theo độ ưu tiên trên, phải, dưới, trái vào Priority Queue. Priority Queue sẽ sắp xếp lại các ô đó dựa vào khoảng cách Manhattan cộng chi phí thực (khoảng cách ô này qua ô khác được tính là 1), chúng ta chỉ cần lấy vị trí đầu tiên là sẽ được ô gần đỉnh khởi đầu nhất (ô ngắn nhất) trong Priority Queue. Tiếp tục xét các ô xung quanh ô đó bỏ vào Priority Queue, cho đến phần tử đầu tiên lấy ra được là nơi muốn tìm đến.

```
def Nearest_Unexplored_Cell(maze, size, Pos, Explored):
    visited = (-1)*np.ones((size[0], size[1]), dtype = 'int')
    path = (-1)*np.ones((size[0], size[1], 2), dtype = 'int')
    q = Queue()

    visited[Pos[0]][Pos[1]] = 1
    q.put(Pos)

    while not q.empty():
        u = q.get()

        for i in range(4):
            neighbor = [u[0] + dr1[i], u[1] + dc1[i]]
            if neighbor[0] > -1 and neighbor[1] > -1 and neighbor[0] < size[0] and neighbor[1] < size[1]:
                if neighbor not in Explored:
                        path[neighbor[0]][neighbor[1]] = u
                        return True, neighbor, path
                if maze[neighbor[0]][neighbor[1]] == 0 or maze[neighbor[0]][neighbor[1]] == 2:
                    if visited[neighbor[0]][neighbor[1]] == -1:
                        visited[neighbor[0]][neighbor[1]] = 1
                        path[neighbor[0]][neighbor[1]] = u
                        q.put(neighbor)
    return False, Pos, path
```

Hình 12. Ô chưa khám phá

Dùng thuật toán BFS để tìm ô chưa khám phá gần nhất. Cũng dựa vào thuật toán dầu loang để đưa các ô xung quanh vị trí khởi đầu theo độ ưu tiên trên, phải, dưới, trái vào Queue. Đỉnh tiếp theo được lấy ra xét sẽ là đỉnh được đưa vào sớm nhất trong Queue theo nguyên tắc First Come, First Out. Cứ lấy các đỉnh ra xét cho đến khi tìm thấy ô chưa được khai phá.

3. Hàm di chuyển monster

Level 3:

```
240  def monsters_Move_3(maze, size, origin_pos, monsters):
241      select = [[1, 2, 3, 4] for _ in range(len(monsters))]
242
243      for i in range(len(monsters)):
244          move = []
245          for j in range(4):
246              move.append([origin_pos[i][0] + dr1[j], origin_pos[i][1] + dc1[j]])
247          for j in range(4):
248              if not Is_Valid_Move(maze, size, move[j]):
249                  select[i].remove(j+1)
250
251
252      for i in range(len(monsters)):
253          if monsters[i] != origin_pos[i]:
254              maze[monsters[i][0]][monsters[i][1]] = 4
255              monsters[i] = origin_pos[i]
256              maze[origin_pos[i][0]][origin_pos[i][1]] = 3
257          else:
258              tmp = random.choice(select[i])
259              if tmp == 1:
260                  monsters[i] = Move_Left(monsters[i])
261              elif tmp == 2:
262                  monsters[i] = Move_Down(monsters[i])
263              elif tmp == 3:
264                  monsters[i] = Move_Up(monsters[i])
265              elif tmp == 4:
266                  monsters[i] = Move_Right(monsters[i])
267              maze[monsters[i][0]][monsters[i][1]] = 3
268              maze[origin_pos[i][0]][origin_pos[i][1]] = 4
269
270
271      return maze, monsters
```

Hình 13. hàm di chuyển monster lv3

Đầu tiên, hàm này tìm những bước đi hợp lệ của monsters quanh vị trí gốc của nó. Những bước đi không hợp lệ sẽ được loại ra khỏi sự lựa chọn bước đi tiếp theo của monsters. Nếu monster đang ở vị trí gốc, nó sẽ chọn ngẫu nhiên 1 trong các bước đi hợp lệ xung quanh nó để đi, ngược lại nếu monsters đang không ở vị trí gốc, nó chỉ có 1 lựa chọn là đi về vị trí gốc.

Level 4:

```
def monsters_Move_4(maze, size, monsters, Goal):

    for i in range(len(monsters)):
        ans, path = AStart(maze, size, monsters[i], Goal, 1)
        if not ans:
            for j in range(4):
                step = [monsters[i][0] + dr1[j], monsters[i][1] + dc1[j]]
                if step[0] > -1 and step[1] > -1 and step[0] < size[0] and step[1] < size[1]:
                    if maze[step[0]][step[1]] == 0:
                        maze[monsters[i][0]][monsters[i][1]] = 0
                        maze[step[0]][step[1]] = 3
                        monsters[i] = step
                        break;
        else:
            list_path = list()
            list_path = Path_Return(path, list_path, Goal, monsters[i])
            list_path.append(Goal)

            maze[monsters[i][0]][monsters[i][1]] = 0
            maze[list_path[0][0]][list_path[0][1]] = 3
            monsters[i][0] = list_path[0][0]
            monsters[i][1] = list_path[0][1]

    return maze, monsters
```

Hình 14. hàm di chuyển monster lv4

Đầu tiền, monster sẽ dung thuật toán A* để tìm đường đi ngắn nhất từ nó đến Pac-man, nếu không tìm thấy, nó sẽ xét xem các ô xung quanh, ô nào đi được theo thứ tự trên, trái, phải dưới và đi đến ô đầu tiên có thể đi được. Ngược lại nếu tìm thấy đường đến PACMAN, nó sẽ đi đến ô đầu tiên trên đường vừa tìm được. Hàm trả về vị trí mới của monsters và maze vừa mới thay đổi.

4. Hàm mở rộng tầm nhìn của Pac-man

```
314   def Explored_Sight_3(maze, size, cv_first, PACMAN_Pos, Sight, food, target):
315       Inside_Foods = list()
316
317       for i in range(24):
318               neighbor = [PACMAN_Pos[0] + dr3[i], PACMAN_Pos[1] + dc3[i]]
319               if neighbor[0] > -1 and neighbor[1] > -1 and neighbor[0] < size[0] and neighbor[1] < size[1]:
320                   if maze[neighbor[0]][neighbor[1]] == 2:
321                       Draw_of_something(cv_first, food_image, (neighbor[1], neighbor[0]))
322                       Inside_Foods.append(neighbor)
323                       if neighbor == target:
324                           continue
325                       if neighbor not in food:
326                           food.append(neighbor)
327                   if neighbor not in Sight:
328                       Sight.append(neighbor)
329
330       return Sight, Inside_Foods
```

Hình 15. hàm mở rộng tầm nhìn lv3

```python
def Explored_Sight_4(maze, size, cv_first, PACMAN_Pos, Sight, food):

    for i in range(24):
            neighbor = [PACMAN_Pos[0] + dr3[i], PACMAN_Pos[1] + dc3[i]]
            if neighbor[0] > -1 and neighbor[1] > -1 and neighbor[0] < size[0] and neighbor[1] < size[1]:
                if neighbor not in Sight:
                    if maze[neighbor[0]][neighbor[1]] == 1:
                        Draw_of_something(cv_first, wall_image, (neighbor[1], neighbor[0]))
                    if maze[neighbor[0]][neighbor[1]] == 2:
                        Draw_of_something(cv_first, food_image, (neighbor[1], neighbor[0]))
                        food.append(neighbor)
                    Sight.append(neighbor)

    return Sight, food
```

Hình 16. hàm mở rộng tầm nhìn lv 4

Hàm này sẽ mở rộng tầm nhìn của Pac-man ở những ô thuộc 3 bước đi xung quanh nó, ở lv3, foods chỉ hiện thị được trong tầm nhìn 3 bước, còn ở lv4, Pac-man sẽ nhìn được ở những vị trí mà nó đã khám phá, nên những foods nào thuộc trong vùng khám phá đều thấy được.

5. Hàm di chuyển của Pac-man

```python
def Block_monster_path(maze, size, monsters_insight, PACMAN_Pos, n_unblock_cell):
    max = 0

    temp_maze = list()
    for i in range(size[0]):
        temp = maze[i].copy()
        temp_maze.append(temp)

    for i in monsters_insight:
        ans, path = AStart(maze, size, i, PACMAN_Pos, 1)
        if ans == True:
            list_path = list()
            list_path = Path_Return(path, list_path, PACMAN_Pos, i)
            if len(list_path) > max:
                max = len(list_path)

            if max - len(list_path) >= n_unblock_cell:
                for j in range(len(list_path)):
                    if j < len(list_path) - n_unblock_cell:
                        temp_maze[list_path[j][0]][list_path[j][1]] = 4

    return temp_maze, max
```

Hình 17. Hàm di chuyển cho pacman lv3

Hàm này tiên đoán đường đi của monsters bằng thuật toán A*, Pac-man sẽ dựa vào đây để tránh các đường nguy hiểm. Tùy theo n_unblock_cell là bao nhiêu mà block các vị trí mà quái sẽ tìm tới PACMAN.

VD: Khi n_unblock_cell = 0 thì sẽ block các ô từ quái tới PACMAN.

Khi n_unblock_cell = 3 thì sẽ block (tổng - 3) ô theo hướng quái về PACMAN. Tức là 3 ô gần PACMAN nhất thuộc đường đi sẽ không bị block.

```python
def PACMAN_Move_4(maze, size, cv_first, monsters_insight, PACMAN_Pos, sight, food):
    max = 1
    i = 0
    while i < max:
        temp_maze, max = Block_monster_path(maze, size, monsters_insight, PACMAN_Pos, i)
        i+=1

        temp_food = food.copy()
        while len(temp_food) != 0:
            target = Find_Nearest_Food(PACMAN_Pos, temp_food)
            temp_food.remove(target)
            ans, path = AStart(temp_maze, size, PACMAN_Pos, target, 0)
            if ans == True:
                list_path = list()
                list_path = Path_Return(path, list_path, target, PACMAN_Pos)
                list_path.append(target)

                return [list_path[0][0],list_path[0][1]]

        ans, target, path = Nearest_Unexplored_Cell(temp_maze, size, PACMAN_Pos, sight)
        if ans == True:
            list_path = list()
            list_path = Path_Return(path, list_path, target, PACMAN_Pos)

            return [list_path[0][0], list_path[0][1]]

    #He will die soon : (
    for i in range(4):
        step = [PACMAN_Pos[0] + dr1[i], PACMAN_Pos[1] + dc1[i]]
        if step[0] > -1 and step[1] > -1 and step[0] < size[0] and step[1] < size[1]:
            if maze[step[0]][step[1]] == 0:
                return step

    return PACMAN_Pos
```

Hình 18. hàm di chuyển cho pacman lv4

Giống như lúc nêu ý tưởng, PACMAN sẽ ưu tiên đặt mạng sống lên đầu. Nên bước đầu tiên trong thuật toán. Dùng hàm Block_monster_path để đánh dấu đường đi của Monsters. Sau đó xét xem trong tầm nhìn có foods nào không. Nếu có ưu tiên tìm foods bằng hàm A* và nếu không thì sẽ tìm đường đi ngắn nhất tới ô chưa khám phá tầm nhìn. Nếu như không tìm được đường nào thì sẽ gọi lại hàm Block_monster_path với i đã tăng 1 (tức đã giảm đi 1 ô bị block gần PACMAN nhất), vòng lặp sẽ chạy như vậy cho tới khi tìm đường hoặc unblock hết tất cả các ô từ monsters tới PACMAN mà vẫn không tìm được đường đi. Nếu vậy PACMAN sẽ tìm đại 1 ô nào đó để đi và chờ chết.

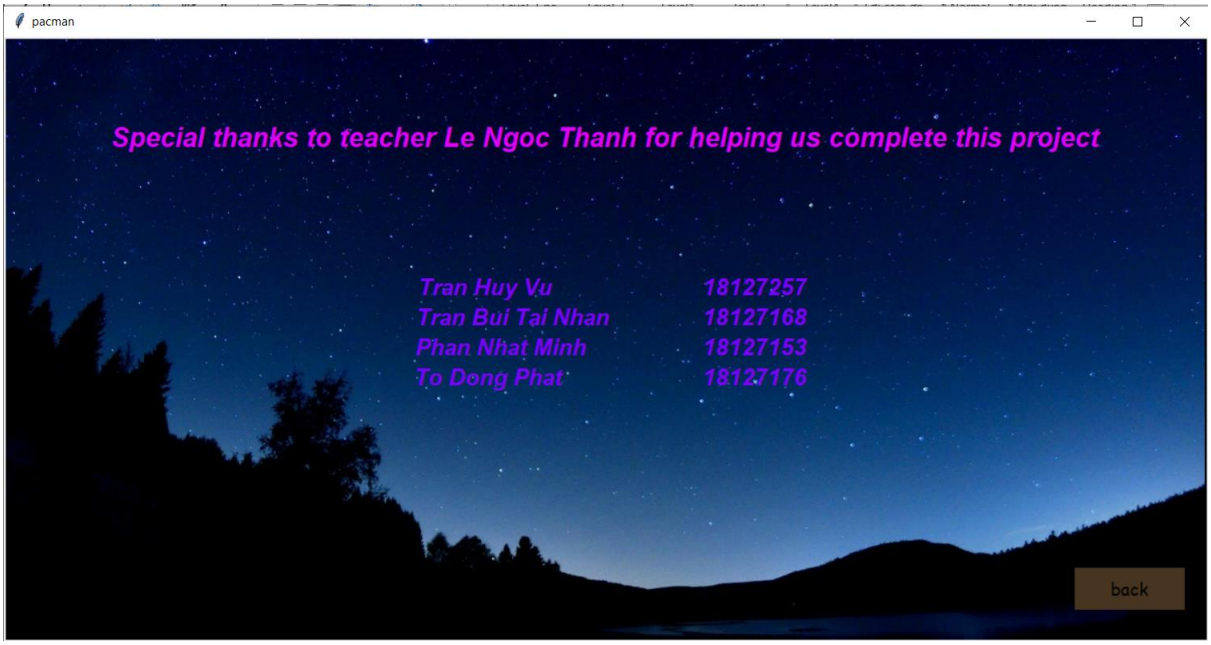
HƯỚNG DẪN SỬ DỤNG

Đầu tiên, nhập file thực thi python: "python pacman.py" để khởi chạy chương trình. Chương trình sẽ lên hình như sau:

Hình 19. giao diện người dùng 1

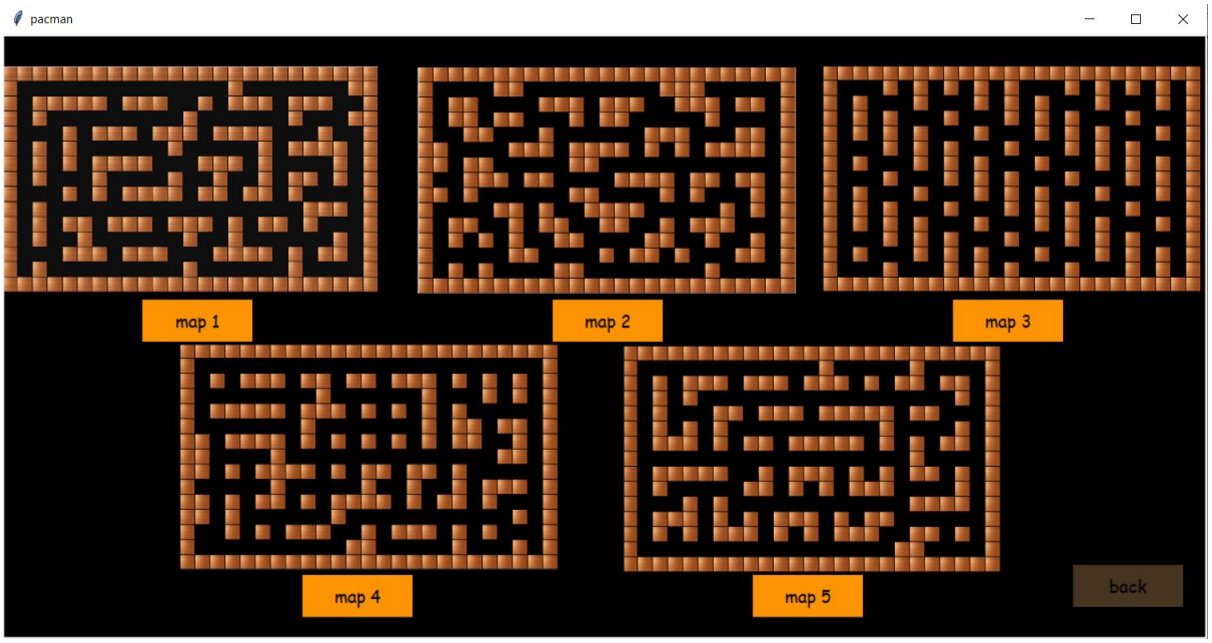
Ở đây ta có 3 mục lựa chọn với exit là thoát chương trình, credit để xem thông tin nguồn về sản phẩm và maze.

Khi chọn credit:



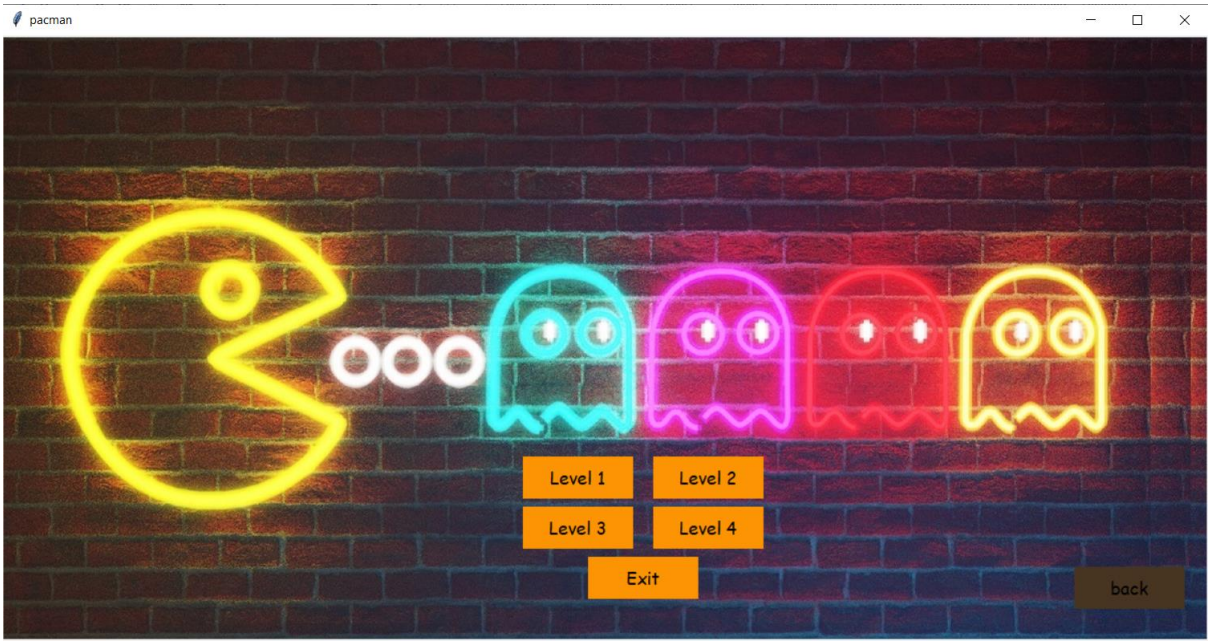
Hình 20. gioa diện credit

Chọn maze ta được



Hình 21. giao diệc chọn maze

Lúc này chúng ta có thể lựa chọn 1 trong 5 maze sau, tùy theo sở thích của bạn, lưu ý, ở góc phải màn hình, có nút back, khi nhấn vào có thể quay lại màn hình trước đó.
Ví dụ ta chọn map 1



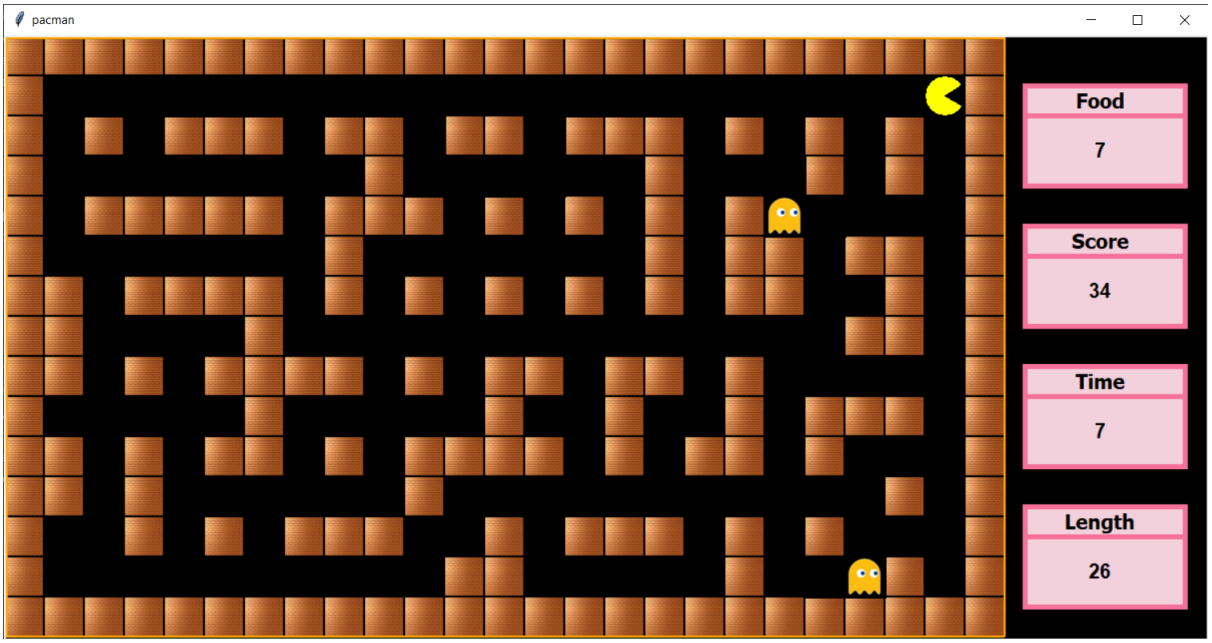
Hình 22. giao diện chọn level

Tiếp đó chọn level bạn muốn chơi với các level được phân cấp như sau

- Level 1: chỉ có pacman tìm food
- Level 2: pacman tìm food nhưng lúc này đã có monster, tuy nhiên monster không thể di chuyển

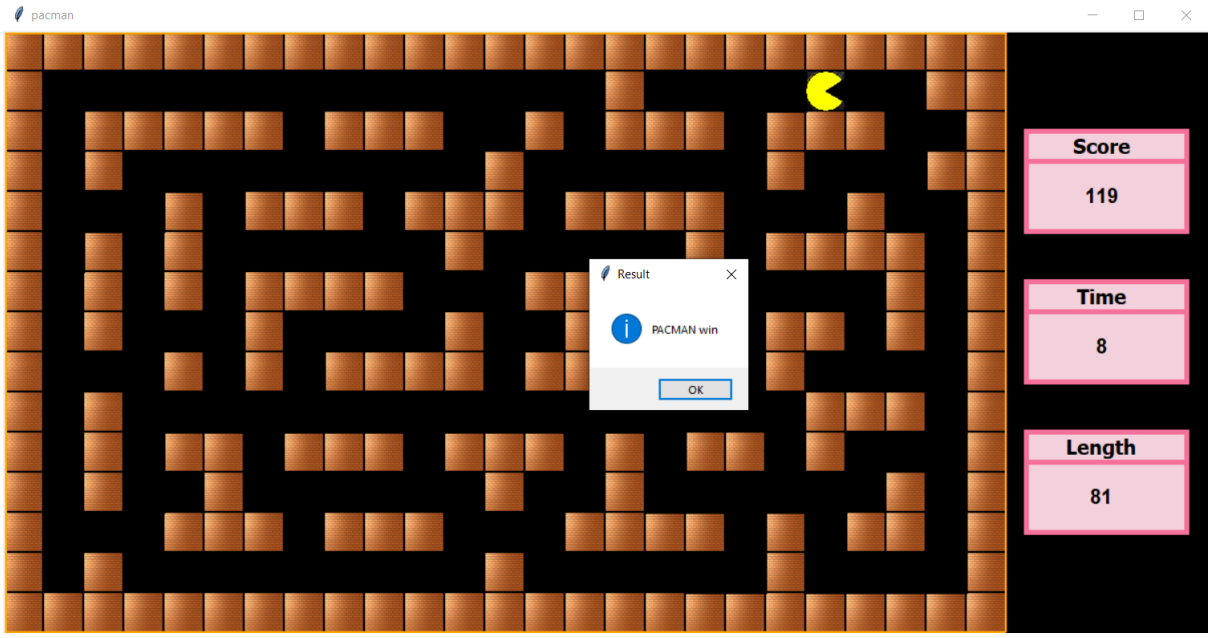
- Level 3: pacman tìm food nhưng lúc này có nhiều monster. Các monster sẽ di chuyển xung quanh vị trí của nó.
- Level 4: tương tự như level 3 nhưng ở đây, monster di chuyển đuổi theo pacman.

Sau khi đã chọn level, bạn có thể tận hưởng cuộc chơi. Khi pacman ăn được hết food hoặc bị quái bắt được, trò chơi sẽ kết thúc



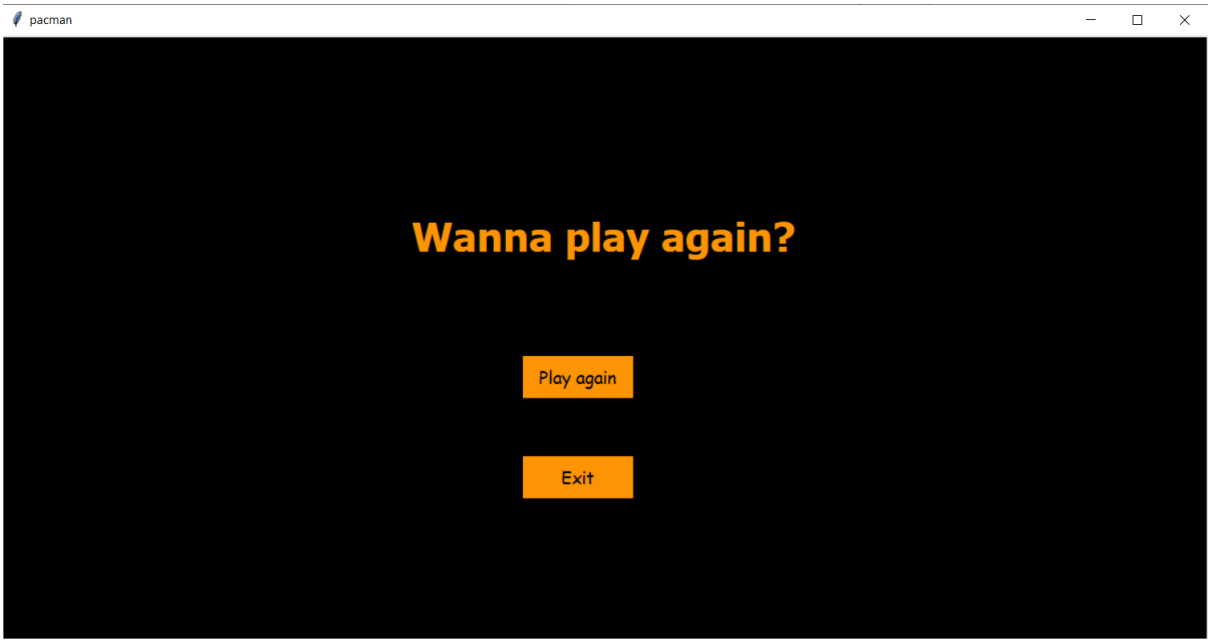
Hình 23. một màn chơi nào đó

Khi kết thúc trò chơi, hiện ra hộp thoại bạn đã thắng hay thua



Hình 24. thắng hay thua

Nháy ok bạn sẽ có quyền quyết định chơi lại hay thoát khỏi trò chơi



Hình 25. Chơi lại hay thoát

MỨC ĐỘ HOÀN THÀNH VÀ ĐÁNH GIÁ CHO TỪNG YÊU CẦU

1. Đồ họa

   Ưu điểm:

   - Giao diện phù hợp và gây ấn tượng với người dùng
   - Các button dễ sử dụng
   - Hình ảnh lên đầy đủ, đúng với file txt

   Khuyết điểm:

   - Chưa có tính tùy biến hình ảnh cao, khi các hình ảnh đồ ăn, quái,.. là gán cứng

   Đánh giá: 90%

   Ưu điểm:

   - Thuật toán A* luôn tìm thấy đường đi ngắn nhất tới foods

   Khuyết điểm:

   - Do Heuristics được tính bằng Manhattan nên đôi khi hàm tìm food gần nhất tìm thấy food không phải là food thực sự gần nhất. Nếu dùng thuật toán để tìm food thực sự gần nhất thì phải chạy thuật toán tìm kiếm cho tất cả các food cho mỗi lần muốn đi đến 1 food, như vậy sẽ tốn chi phí rất lớn
2. Level 2

   Có những ưu khuyết điểm của level 1.
3. Level 3

   Có những ưu khuyết điểm của level 1.

   Khuyết điểm:

   Do thuật toán tìm kiếm ở level 3 sẽ né đường monsters ra nên sẽ không tìm được foods ở vị trí đặt biệt.
4. Level 4

   Ưu điểm:

   - Thuật toán A* luôn tìm thấy đường đi ngắn nhất tới foods
   - Dùng thuật Toán A* tìm và block đường đi của monsters cho PACMAN không đi, nếu không tìm được mở block dần nên PACMAN tìm đường đi tốt và lối thoát ổn.

   Khuyết điểm:

   - Chỉ tiên đóan đường đi hiện tại của monsters nên nhiều khi PACMAN đi, monsters sẽ tìm đường khác ngắn hơn khác với PACMAN đã tính. Dẫn đến PACMAN đi vào ngõ cục, hoặc PACMAN và quái nhảy qua lại vị trí cũ.
5. Data

   Maze đa dạng, với 5 level khác nhau, nhiều đường cho PACMAN chạy, nhiều góc cho 2 monsters ép góc PACMAN.

Đánh giá chung:

Mức độ hoàn thành: 95%

Đáp ứng đầy đủ các yêu cầu của đồ án, phát triển thêm nhiều về giao diện và thuật toán

TÀI LIỆU THAM KHẢO

https://bom.to/AYimE9
https://bom.to/1rKOPk
https://bom.to/qQFreg