

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO
ĐỒ ÁN TÂM VÀ GIA HUY

MÔN: Lập Trình Cho Trí Tuệ Nhân Tạo

Tp. Hồ Chí Minh, tháng 4/2024

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO
ĐỒ ÁN TÂM VÀ GIA HUY

MÔN: Lập Trình Cho Trí Tuệ Nhân Tạo

LỚP: 23TNT1

GIÁO VIÊN HƯỚNG DẪN

Nguyễn Trần Duy Minh

Thành viên Nhóm 1

- Bàng Mỹ Linh - 23122009
- Nguyễn Trần Trung Kiên - 23122038
- Đào Sỹ Duy Minh - 23122041
- Nguyễn Lâm Phú Quý - 23122048

Tp. Hồ Chí Minh, tháng 4/2024

Lời cảm ơn

Lời đầu tiên, xin trân trọng cảm ơn thầy Nguyễn Trần Duy Minh và Thầy Lê Thanh Tùng đã tận tình hướng dẫn chúng em trong quá trình nghiên cứu cũng như hoàn thành báo cáo này để hỗ trợ cho đồ án.

Xin chân thành cảm ơn các Thầy, Cô thuộc khoa Công nghệ Thông tin nói riêng và toàn bộ Thầy, Cô ở trường Đại học Khoa học Tự Nhiên - DHQG TP.HCM nói chung đã tận tình giảng dạy cho chúng em trong suốt thời gian học tập.

Do giới hạn kiến thức và kĩ năng nên chúng em không tránh khỏi có những thiếu sót. Kính mong sự chỉ dẫn và đóng góp của các Thầy, Cô để bản báo cáo và sản phẩm đồ án của chúng em được hoàn thiện hơn. Chúng em xin chân thành cảm ơn!

Thành phố Hồ Chí Minh, ngày 21 tháng 5 năm 2024

Nhóm 1

Mục lục

1	Lời cảm ơn	1
2	Giới thiệu	5
2.1	Thông tin thành viên nhóm	5
2.2	Kế hoạch và phân công	5
2.2.1	Dợt 1	5
2.2.2	Dợt 2	5
2.2.3	Dợt 3	6
2.2.4	Dợt 4	6
3	Tổng quan về đồ án Tâm và Gia Huy	8
4	Cơ chế game	9
4.1	Các thư viện được sử dụng	9
4.2	Các thư viện ngoài cần cài đặt	9
4.3	Class Cell (Ô trong mê cung)	9
4.3.1	Phương thức def __init__(self, x, y):	10
4.3.2	Phương thức draw_mini_map(self, screen, TILE, maze_x, maze_y)	10
4.3.3	Phương thức draw(self, sc, TILE)	10
4.3.4	Phương thức check_cell(self, x, y, cols, rows)	11
4.3.5	Phương thức check_neighbors(self, grid_cells, cols, rows)	11
4.3.6	Phương thức check_neighbors_pass(self, grid_cells, cols, rows)	11
4.4	Class Player	12
4.4.1	Phương thức __init__(self, x, y)	12
4.4.2	Phương thức draw(self, sc, TILE)	12
4.4.3	Phương thức move(self, direction, grid_cells, cols, rows)	12
4.5	Class Player_Animation	13
4.5.1	Phương thức khởi tạo	13
4.5.2	Phương thức get_current_frame	14
4.5.3	Phương thức update	14
4.5.4	Phương thức set_direction	15
4.6	Class MazeDrawer	15
4.6.1	Phương thức khởi tạo	15
4.6.2	Phương thức draw	16
4.6.3	Phương thức draw_cells	16
4.6.4	Kết luận	17
4.7	Class Mouse	17
4.7.1	Phương thức khởi tạo	17
4.7.2	Phương thức move	17
4.7.3	Phương thức draw	18
4.7.4	Kết luận	18
4.8	Class start	18
4.8.1	Phương thức khởi tạo	18
4.8.2	Phương thức draw	19
4.8.3	Kết luận	19
4.9	Tạo mê cung	19
4.9.1	Hàm remove_walls(current, next)	19
4.9.2	Hàm build_walls(current, next)	20
4.9.3	Hàm divide_maze	21
4.9.4	Hàm tạo mê cung generate_maze(cols, rows, TILE)	21
4.10	Chọn điểm đầu và điểm cuối	22
4.10.1	Chọn điểm đầu - điểm cuối bằng cách random	22
4.10.2	Người chơi tự chọn điểm đầu - điểm cuối	24
4.10.3	Cho Người chơi chọn lại ví trí nếu không gặp nhau	25

4.11	Các thuật toán tìm đường đi được sử dụng trong game	26
4.11.1	Thuật toán DFS	26
4.11.2	Thuật toán BFS	28
4.11.3	Thuật Toán A*	29
4.11.4	Thuật Toán Dijkstra	30
4.11.5	So sánh 4 thuật toán tìm đường đi	32
4.12	Gọi ý Đường đi	34
4.12.1	Gọi ý đường đi tại vị trí bất kì	34
4.12.2	Visualize	35
4.13	Máy tự chơi	37
4.13.1	Xử lý sự kiện nhấn phím w	37
4.13.2	Xử lý kết quả tìm đường	37
4.13.3	Cập nhật trạng thái và hiển thị	37
4.13.4	Di chuyển Bot theo đường đi	37
4.13.5	Kết luận	38
4.14	Hàm reset game và hàm new game	39
4.14.1	Hàm reset game	39
4.14.2	Hàm new game	39
4.15	Các biến toàn cục, thư viện và hàm xử lý chính	40
4.15.1	Khởi tạo biến toàn cục và thư viện:	40
5	Database	43
5.1	Tổ Chức Database	43
5.1.1	file database.json	43
5.1.2	file ranking.json	43
5.2	Chức Năng Save_Game	44
5.2.1	Các hàm lưu thông số của trò chơi	44
5.2.2	Hàm lưu trò chơi tổng hợp	45
5.3	Chức Năng load game	46
5.3.1	Hàm load các file chứa các thông số trong trò chơi	46
5.3.2	Hàm load tổng hợp	47
5.3.3	Hàm xử lý sau khi load file	48
6	Giao diện	49
6.1	Design Giao Diện	49
6.2	Login Form	52
6.2.1	Cửa sổ login	52
6.2.2	Cửa sổ sign up	53
6.3	Menu Game	56
6.3.1	draw_menu(current_option)	56
6.3.2	handle_menu_input(current_option)	56
6.3.3	xulymenu()	57
6.3.4	Hàm draw_menu_setting(current_option)	58
6.3.5	Hàm xulymenu_settings()	58
6.4	Class MazeGameLoader(LoadingBar)	60
6.4.1	Giới thiệu	60
6.4.2	Phương thức: __init__()	60
6.4.3	Phương thức: run()	60
6.4.4	Phương thức: draw_progress_bar()	61
6.4.5	Phương thức: render_text()	61
6.4.6	Phương thức: update_loading_tips()	61
6.4.7	Phương thức: draw_wave_animation()	61
6.5	Giao Diện PauseGame	64
6.5.1	Biến Quan Trọng	64
6.5.2	Hàm pause_screen(screen)	64
6.5.3	Hàm continue_game()	64

6.6	Dash Board	65
6.6.1	Giao diện dashboard	65
6.6.2	Cấu trúc và chức năng của hàm <code>draw_dashboard</code>	65
6.6.3	Các hàm khác	66
6.7	Transistion	66
6.7.1	Hàm <code>transition</code>	66
6.7.2	Hàm <code>intro_transition</code>	66
6.7.3	Kết luận	67
6.8	Giao Diện Load Game	67
6.8.1	Hàm <code>load_filegame</code>	67
6.8.2	Hàm <code>draw_mini_maze</code>	67
6.8.3	Hàm <code>draw_load_screen</code>	68
6.8.4	Hàm <code>Load_Game_Display</code>	68
6.8.5	Kết luận	68
6.9	Giao Diện save game	70
6.10	Giao Diện Màn Hình Chiến Thắng	71
6.11	Giao Diện Ranking	73
6.11.1	Khởi tạo và Cài đặt Pygame	73
6.11.2	Hàm <code>get_rank</code>	73
6.11.3	Hàm <code>load_ranking</code>	73
6.11.4	Hàm <code>draw_ranking</code>	73
6.11.5	Hàm <code>xuly_ranking</code>	74
6.11.6	Kết luận	74
6.12	Giao Diện About	75
7	Âm Thanh/Hình Nền	76
7.1	Âm Thanh	76

Giới thiệu

2.1 Thông tin thành viên nhóm

MSSV	Họ và Tên	Email
23122009	Bàng Mỹ Linh	23122009@student.hcmus.edu.vn
23122038	Nguyễn Trần Trung Kiên	23122038@student.hcmus.edu.vn
23122041	Dào Sỹ Duy Minh	23122041@student.hcmus.edu.vn
23122048	Nguyễn Lâm Phú Quý	23122048@student.hcmus.edu.vn

2.2 Kế hoạch và phân công

2.2.1 Đợt 1

Thành Viên	Công việc	Mức độ hoàn thành
Nguyễn Lâm Phú Quý	- Tạo mê cung và thử nghiệm, cài đặt thuật toán BFS, DFS, A*	100%
Dào Sỹ Duy Minh	- Tạo trước player đơn giản với đồ họa ô vuông - Tạo menu đơn giản	100%
Bàng Mỹ Linh	- Thử nghiệm và cài đặt thuật toán Dijkstra	100%
Nguyễn Trần Trung Kiên	- Âm thanh	100%

2.2.2 Đợt 2

Thành Viên	Công việc	Mức độ hoàn thành
Nguyễn Lâm Phú Quý	- Xây dựng file lưu trữ thông tin người chơi - Tạo login/register form - Save và load game Tạo điểm bắt đầu và đích ngẫu nhiên	100%
Dào Sỹ Duy Minh	- Chọn điểm bắt đầu và điểm kết thúc - Xử lý việc đổi thuật toán - Xử lý việc chọn độ khó mê cung - Pause và restart	100%
Bàng Mỹ Linh	- Xử lý thời gian chạy trong game - Xây dựng dashboard trong game để người dùng tùy chọn.	100%
Nguyễn Trần Trung Kiên	- Tạo login/ register form - Save và load game	100%

2.2.3 Đợt 3

Thành Viên	Công việc	Mức độ hoàn thành
Nguyễn Lâm Phú Quý	<ul style="list-style-type: none">- Thiết kế lại mê cung để có nhiều hơn 1 đường đi- Giao diện lưu file	100%
Dào Sỹ Duy Minh	<ul style="list-style-type: none">- Tạo animation cho Tâm và Gia Huy- Giao diện xóa file, load game- Chỉnh sửa gợi ý đường đi- Xử lý và tạo giao diện khi điểm đầu và đích không thể đến được nhau- Tạo giao diện hiển thị leaderboard	100%
Bàng Mỹ Linh	<ul style="list-style-type: none">- Chỉnh sửa và trang trí lại login/ register form- Lựa chọn phông chữ cho game	100%
Nguyễn Trần Trung Kiên	<ul style="list-style-type: none">- Chỉnh sửa bật tắt và thêm các chức năng trong phần âm thanh- Thêm chức năng cho login/ register form như gợi ý mật khẩu, ẩn mật khẩu, nhấn enter...	100%

2.2.4 Đợt 4

Thành Viên	Công việc	Mức độ hoàn thành
Nguyễn Lâm Phú Quý	<ul style="list-style-type: none"> - Trực quan hóa quá trình chạy các thuật toán tìm kiếm - Tinh chỉnh các thông số của mê cung như vị trí, kích cỡ - Lưu trữ và tính toán bảng xếp hạng của người chơi - Tạo màn hình chiến thắng và thua cuộc. - Tạo hướng dẫn cho người chơi - Sửa lỗi load game - Cho phép người chơi bật/ tắt nhạc khi nhấn nút 	100%
Đào Sỹ Duy Minh	<ul style="list-style-type: none"> - Tạo hình ảnh cho mê cung - Tạo animation khi chiến thắng của Tâm - Tạo animation cho màn hình chiến thắng - Chỉnh sửa quá trình trực quan hóa thuật toán - Xử lý việc nhấp chuột vào nút cho người chơi - Kiểm tra toàn bộ source code, sửa lỗi - Trang trí màn hình load game - Tạo màn hình giới thiệu - Lên kịch bản cho video demo 	100%
Bàng Mỹ Linh	<ul style="list-style-type: none"> - Thiết kế lại hình ảnh mê cung - Thiết kế background trong game và menu - Thiết kế các nút - Thiết kế màn hình giới thiệu - Thiết kế màn hình hướng dẫn - Sửa lỗi load file - Sửa lỗi chạy thời gian. - Xử lý sau khi người chơi thua (chơi quá thời gian quy định) 	100%
Nguyễn Trần Trung Kiên	<ul style="list-style-type: none"> - Chỉnh sửa âm thanh - Thêm âm thanh - Chỉnh sửa các lỗi trong phần login/ register form - Edit video demo 	100%

Tổng quan về đồ án Tâm và Gia Huy

Đồ án game Tâm Và Gia Huy giúp nhóm vận dụng các kiến thức cơ bản của lập trình để tạo nên tựa game đơn giản này. Với mục tiêu tạo cơ hội để mỗi thành viên trong nhóm phát triển, nâng cao khả năng của bản thân qua hoạt động nhóm, đồng thời học hỏi kiến thức để hoàn thiện sản phẩm.

Cơ chế game

4.1 Các thư viện được sử dụng

- Thư viện numbers
- Thư viện re
- Thư viện time
- Thư viện turtle
- Thư viện pygame
- Thư viện sys
- Thư viện os
- Thư viện threading
- thư viện queue
- Thư viện random
- Thư viện pyautogui
- Thư viện Tkinter

4.2 Các thư viện ngoài cần cài đặt

- Thư viện pygame
- Thư viện pyautogui
- Thư viện Tkinter

4.3 Class Cell (Ô trong mê cung)

Đối tượng cell trong mê cung là đối tượng dùng để xác định một ô trong mê cung, nó gồm các thuộc tính sau:

- self.x, self.y = x, y: **Tọa độ ngang dọc** với trục x là trục ngang, y là trục dọc
- self.walls = 'top': True, 'right': True, 'bottom': True, 'left': True
Đây là thuộc tính "bức tường" cho các ô, ban đầu ta định nghĩa mỗi ô là một căn phòng có 4 bức tường, sau đó ta sẽ dần xóa các bức tường đi bằng cách random trong việc tạo mê cung.
- self.visited = False: Thuộc tính để hỏi ô đã được "ghé thăm" hay chưa, là một thuộc tính quan trọng để tạo ra mê cung bằng thuật toán DFS
- self.passed = False: Cũng là thuộc tính để hỏi ô đã được thăm hay chưa, nhưng dùng trong việc tìm đường đi trong chế độ gợi ý và tự chơi
- self.path = False: Thuộc tính để truy vết đường đi, nếu path = True nghĩa là đây là một trong những ô đi tới đích
- self.seen = False: Thuộc tính này giúp chúng ta có thể hiển thị quá trình thăm các đỉnh trong thuật toán tìm đường đi

4.3.1 Phương thức def __ init__(self, x, y):

Phương thức này khởi tạo các thuộc tính của ô đã được liệt kê ở trên

4.3.2 Phương thức draw_mini_map(self, screen, TILE, maze_x, maze_y)

Phương thức này giúp ta vẽ các mè cung nhỏ hiển thị trong quá trình load game

```
1  function draw_mini_map(screen, TILE, maze_x, maze_y):
2      // Calculate the coordinates for the current cell
3      x = maze_x + self.x * TILE
4      y = maze_y + self.y * TILE
5
6      // Draw the top wall if it exists
7      if self.walls['top'] is true:
8          draw_rectangle(screen, color='green', rectangle=(x, y, TILE, self.thickness))
9
10     // Draw the right wall if it exists
11     if self.walls['right'] is true:
12         draw_rectangle(screen, color='green', rectangle=(x + TILE - self.thickness, y,
13 self.thickness, TILE))
14
15     // Draw the bottom wall if it exists
16     if self.walls['bottom'] is true:
17         draw_rectangle(screen, color='green', rectangle=(x, y + TILE - self.thickness,
18 TILE, self.thickness))
19
20     // Draw the left wall if it exists
21     if self.walls['left'] is true:
22         draw_rectangle(screen, color='green', rectangle=(x, y, self.thickness, TILE))
```

4.3.3 Phương thức draw(self, sc, TILE)

Dây là phương thức giúp chúng ta vẽ được việc truy vết thuật toán tìm đường chạy như thế nào cũng như tạo ra con đường dẫn đến đích mắt

```
1  function draw(self, sc, TILE):
2      // Calculate the coordinates for the current cell
3      a = self.x * TILE + DELTA_X
4      b = self.y * TILE + DELTA_Y
5
6      // If the cell is part of the path
7      if self.path is true:
8          // Load the path image
9          path_image = load_image("path1.png")
10         // Scale the image to fit within the tile with some padding
11         path_image = scale_image(path_image, TILE - TILE / 3, TILE - TILE / 3)
12         // Draw the path image on the screen with some offset
13         blit_image(sc, path_image, a + TILE / 10, b + TILE / 10)
14
15     // If the cell has been seen
16     if self.seen is true:
17         // Load the seen path image
18         path_image1 = load_image("path2.png")
19         // Scale the image to fit within the tile with some padding
20         path_image1 = scale_image(path_image1, TILE - TILE / 3, TILE - TILE / 3)
21         // Draw the seen path image on the screen with some offset
22         blit_image(sc, path_image1, a + TILE / 5, b + TILE / 5)
```

Listing 4.1: Mã giả cho phương thức draw

- Trong đoạn code trên, a, b được gán là tọa độ ô trong mè cung khi hiển thị trên màn hình, bằng cách nhân tọa độ của nó trong mè cung cộng với TILE (khoảng cách của một ô) và một lượng DELTA_X, DELTA_Y.
- path_image và path_image1 là hai ảnh biểu thị mè vết đi trên mè cung, một cái hiển thị quá trình visualize các thuật toán tìm đường đi, một cái giúp hiển thị đường đi đến đích.

4.3.4 Phương thức check_cell(self, x, y, cols, rows)

```
1     function check_cell(x, y, cols, rows)
2         find_index = lambda(x, y)
3             return x + y * cols
4         end
5
6         if x < 0 or x > cols - 1 or y < 0 or y > rows - 1 then
7             return false
8         end
9         return self.grid_cells[find_index(x, y)]
10    end function
```

Listing 4.2: Mã giả cho phương thức check_cell

- Tham số truyền vào của phương thức này là ô đang xét, số cột (cols), hàng (rows) của mê cung, và tọa độ x, y của ô đang xét
- Trong mã trên, ta định nghĩa một lambda là biểu thức trả về vị trí của ô đang xét trong ma trận. Nếu x và y nằm ngoài đoạn [0, cols) hay [0, rows) thì nghĩa là ô đó đã nằm ngoài mê cung, ta không xét đến và trả về False. Phương thức này cần thiết để kiểm tra các ô xung quanh của một ô trong mê cung chỉ dựa vào tọa độ x, y của ô đó đồng thời trả về vị trí của ô đó trong mê cung.

4.3.5 Phương thức check_neighbors(self, grid_cells, cols, rows)

```
1     function check_neighbors(self, grid_cells, cols, rows)
2         neighbors = []
3         top = check_cell(self.x, self.y - 1, cols, rows)
4         right = check_cell(self.x + 1, self.y, cols, rows)
5         bottom = check_cell(self.x, self.y + 1, cols, rows)
6         left = check_cell(self.x - 1, self.y, cols, rows)
7
8         if top and not top.visited then
9             append top to neighbors
10        end if
11        if right and not right.visited then
12            append right to neighbors
13        end if
14        if bottom and not bottom.visited then
15            append bottom to neighbors
16        end if
17        if left and not left.visited then
18            append left to neighbors
19        end if
20
21        return choose random element from neighbors if neighbors not empty, otherwise
22        return False
end function
```

Listing 4.3: Mã giả cho phương thức check_neighbors

- Phương thức này được truyền vào các tham số: ô đang xét, grid_cells (tất cả các ô trong mê cung), số hàng và số cột của mê cung
- Bằng việc gọi hàm check_cell được định nghĩa trước đó, ta có thể kiểm tra các ô xung quanh có nằm trong mê cung hay không. Sau đó, nếu ô đó chưa được "thăm" (self.visited = False) thì ta "thăm" ô đó, việc này để thuật toán dfs không làm việc với một ô hai lần. Tiếp theo, ta thêm ô đó vào danh sách neighbors, và cuối cùng, chọn ngẫu nhiên một ô xung quanh bằng cách sử dụng phương thức choice trong module random, việc này là để tiếp theo ta có thể xóa đi "bức tường" giữa các ô

4.3.6 Phương thức check_neighbors_pass(self, grid_cells, cols, rows)

```

1     function check_neighbors_pass(self, grid_cells, cols, rows)
2         neighbors = []
3         top = check_cell(self.x, self.y - 1, cols, rows)
4         right = check_cell(self.x + 1, self.y, cols, rows)
5         bottom = check_cell(self.x, self.y + 1, cols, rows)
6         left = check_cell(self.x - 1, self.y, cols, rows)
7
8         if top and not self.walls['top'] then
9             append top to neighbors
10        end if
11        if right and not self.walls['right'] then
12            append right to neighbors
13        end if
14        if bottom and not self.walls['bottom'] then
15            append bottom to neighbors
16        end if
17        if left and not self.walls['left'] then
18            append left to neighbors
19        end if
20    return neighbors
21 end function

```

Listing 4.4: Mã giả cho phương thức `check_neighbors_pass`

- Các tham số truyền vào là: ô đang xét, mảng ô hàng, số hàng và số cột
- Bằng việc kiểm tra các ô xung quanh có nằm trong mê cung hay không, đồng thời kiểm tra giữa hai ô có phải bức tường hay không (`not self.walls[""]`), ta có thể kiểm tra xem liệu từ ô này có thể sang được ô kia hay không, nếu được, ta thêm ô xung quanh đó vào danh sách các ô có thể qua được. Nhờ vậy, ta có thể sử dụng thuật toán tìm đường đi từ ô đầu sang ô đích.

4.4 Class Player

4.4.1 Phương thức `__init__(self, x, y)`

4.4.2 Phương thức `draw(self, sc, TILE)`

Phương thức này được sử dụng để vẽ người chơi trên màn hình với màu xanh dương (thay đổi sau). Đối số `sc` là màn hình pygame, và `TILE` là kích thước của mỗi ô trên lưới.

4.4.3 Phương thức `move(self, direction, grid_cells, cols, rows)`

Phương thức này được sử dụng để di chuyển người chơi trong trò chơi dựa trên hướng di chuyển được chỉ định và tình trạng của các ô trong lưới.

- Nếu hướng di chuyển là 'up' (di chuyển lên trên) và người chơi không gặp tường ở phía trên, thì tọa độ y sẽ giảm đi 1 đơn vị.
- Nếu hướng di chuyển là 'down' (di chuyển xuống dưới) và người chơi không gặp tường ở phía dưới, thì tọa độ y sẽ tăng lên 1 đơn vị.
- Nếu hướng di chuyển là 'left' (di chuyển qua trái) và người chơi không gặp tường ở phía trái, thì tọa độ x sẽ giảm đi 1 đơn vị.
- Nếu hướng di chuyển là 'right' (di chuyển qua phải) và người chơi không gặp tường ở phía phải, thì tọa độ x sẽ tăng lên 1 đơn vị.
- Lưu ý một đơn vị ở đây là 1 ô được định nghĩa trong maze.

```

1 BEGIN Player Class
2 FUNCTION Initialize(x, y)
3     SET self.x TO x
4     SET self.y TO y

```

```

5     END FUNCTION
6
7     FUNCTION Draw(sc, TILE)
8         Pygame.Draw.Rect(sc, Pygame.Color('blue'), (self.x * TILE + 4, self.y * TILE + 4,
9             TILE - 8, TILE - 8))
10    END FUNCTION
11
12    FUNCTION Move(direction, grid_cells, cols, rows)
13        IF direction EQUALS 'up' AND self.y > 0 THEN
14            IF NOT grid_cells[self.x + (self.y - 1) * cols].walls['bottom'] THEN
15                SET self.y TO self.y - 1
16            END IF
17        ELSE IF direction EQUALS 'down' AND self.y < rows - 1 THEN
18            IF NOT grid_cells[self.x + (self.y + 1) * cols].walls['top'] THEN
19                SET self.y TO self.y + 1
20            END IF
21        ELSE IF direction EQUALS 'left' AND self.x > 0 THEN
22            IF NOT grid_cells[self.x - 1 + self.y * cols].walls['right'] THEN
23                SET self.x TO self.x - 1
24            END IF
25        ELSE IF direction EQUALS 'right' AND self.x < cols - 1 THEN
26            IF NOT grid_cells[self.x + 1 + self.y * cols].walls['left'] THEN
27                SET self.x TO self.x + 1
28            END IF
29        END IF
30    END FUNCTION
31 END Player Class

```

Listing 4.5: Mã giả cho Class Player

4.5 Class Player_Animation

4.5.1 Phương thức khởi tạo

```

1 BEGIN PlayerAnimation Class
2     FUNCTION Initialize(animation_frames, sprite_sheet_path, cols, rows, cell_size,
3         frame_rate=60)
4         SET self.sprite_sheet TO pygame.image.load(sprite_sheet_path)
5         SET self.cols TO cols
6         SET self.rows TO rows
7         SET self.cell_size TO cell_size
8         SET self.frame_width TO self.sprite_sheet.get_width() DIV cols
9         SET self.frame_height TO self.sprite_sheet.get_height() DIV rows
10        SET self.frame_count TO cols * rows
11        SET self.current_animation TO FIRST_KEY(animation_frames) // Default direction
12        SET self.current_frame TO 0
13        SET self.frame_rate TO frame_rate // Desired frame rate in frames per second
14        SET self.animation_frames TO animation_frames
15        SET self.frame_duration TO 2 * (1000 DIV frame_rate) // Duration of each frame in
16        milliseconds
16    END FUNCTION

```

Listing 4.6: Mã giả cho phương thức khai tạo

Phương thức khai tạo của lớp PlayerAnimation có nhiệm vụ thiết lập các thông số ban đầu cho đối tượng hoạt hình. Các thông số này bao gồm:

- **animation_frames**: Một từ điển chứa danh sách các chỉ số khung hình tương ứng với mỗi hướng di chuyển. Mỗi hướng di chuyển sẽ có một danh sách các chỉ số này.
- **sprite_sheet_path**: Đường dẫn tới tệp ảnh sprite sheet. Sprite sheet là một tập hợp các hình ảnh nhỏ, mỗi hình ảnh đại diện cho một khung hình trong hoạt hình.
- **cols, rows**: Số cột và số hàng trong sprite sheet, xác định số lượng khung hình trên mỗi dòng và mỗi cột.

- **cell_size**: Kích thước của mỗi khung hình sau khi được cắt ra từ sprite sheet. Thường là một cặp giá trị (width, height).
- **frame_rate**: Tốc độ mong muốn của hoạt hình, tính bằng số lượng khung hình xuất hiện trong một giây. Giá trị mặc định là 60 khung hình mỗi giây.

Trong phương thức khởi tạo, đầu tiên, ảnh sprite sheet được tải vào bộ nhớ từ đường dẫn đã cho. Tiếp theo, kích thước của mỗi khung hình được tính toán bằng cách chia kích thước của sprite sheet cho số cột và hàng. Số lượng khung hình được xác định bằng tích của số cột và hàng. Hướng di chuyển mặc định và chỉ số của khung hình hiện tại được thiết lập ban đầu. Cuối cùng, thời gian của mỗi khung hình được tính toán dựa trên tốc độ khung hình và thời gian hiện tại được lưu lại để theo dõi việc cập nhật khung hình.

4.5.2 Phương thức get_current_frame

```

1 FUNCTION get_current_frame()
2     SET animation_index TO self.animation_frames[self.current_animation][self.
3         current_frame]
4     SET col TO animation_index MOD self.cols
5     SET row TO animation_index DIV self.cols
6     SET x TO col * self.frame_width
7     SET y TO row * self.frame_height
8     SET x TO MIN(MAX(0, x), self.sprite_sheet.get_width() - self.frame_width)
9     SET y TO MIN(MAX(0, y), self.sprite_sheet.get_height() - self.frame_height)
10    SET frame TO self.sprite_sheet.subsurface(pygame.Rect(x, y, self.frame_width, self.
11        frame_height))
10    RETURN pygame.transform.scale(frame, (self.cell_size, self.cell_size))
11 END FUNCTION

```

Listing 4.7: Mã giả cho phương thức get current frame thuộc class Animation

Phương thức này có nhiệm vụ trả về khung hình hiện tại của hoạt hình. Quá trình xử lý diễn ra như sau:

- Từ danh sách chỉ số khung hình tương ứng với hướng di chuyển hiện tại và chỉ số khung hình hiện tại, tính toán chỉ số của khung hình trong sprite sheet.
- Xác định vị trí của khung hình trong sprite sheet dựa trên chỉ số tính toán được.
- Cắt ra khung hình từ sprite sheet sử dụng các vị trí đã xác định.
- Thay đổi kích thước của khung hình theo kích thước cell và trả về.

4.5.3 Phương thức update

```

1 FUNCTION update()
2     SET current_time TO pygame.time.get_ticks()
3     IF current_time - self.last_frame_time >= self.frame_duration THEN
4         SET self.current_frame TO (self.current_frame + 1) MOD self.cols
5         SET self.last_frame_time TO current_time
6     END IF
7 END FUNCTION

```

Listing 4.8: Mã giả cho phương thức update thuộc class Animation

Phương thức update kiểm tra xem đã đến lúc cập nhật khung hình mới hay chưa. Nếu thời gian đã trôi qua từ lần cập nhật cuối cùng lớn hơn hoặc bằng thời gian của mỗi khung hình (**frame_duration**), chỉ số khung hình hiện tại sẽ được cập nhật. Cụ thể:

- Lấy thời gian hiện tại.
- So sánh thời gian hiện tại với thời gian của lần cập nhật cuối cùng.
- Nếu thời gian đã trôi qua đủ để cập nhật khung hình mới, chỉ số khung hình hiện tại được tăng lên một đơn vị và thời gian của lần cập nhật mới được lưu lại.

4.5.4 Phương thức set_direction

Phương thức `set_direction` cho phép thay đổi hướng di chuyển của hoạt hình. Khi gọi phương thức này và truyền vào một hướng mới, hoạt hình sẽ chuyển sang sử dụng danh sách khung hình tương ứng với hướng đó.

```
1 FUNCTION set_direction(direction)
2     SET self.current_animation TO direction
3 END FUNCTION
4 END PlayerAnimation Class
```

Listing 4.9: Mã giả cho phương thức set direction thuộc class Animation

4.6 Class MazeDrawer

4.6.1 Phương thức khởi tạo

```
1 BEGIN MazeDrawer Class
2 GLOBAL sc
3 GLOBAL CELLSIZE
4 GLOBAL DELTA_X
5 GLOBAL DELTA_Y
6
7 FUNCTION Initialize(cols, rows, cells, interactive)
8     SET TILE TO Bottom_right_x DIV cols
9     SET self.tilesheet TO pygame.Surface((TILE, TILE*4))
10    SET self.cols TO cols
11    SET self.rows TO rows
12    SET self.tile TO TILE
13    SET self.sc TO sc
14    SET self.cells TO cells
15    SET self.cell_images TO EMPTY LIST
16    FOR y FROM 0 TO 3 DO
17        FOR x FROM 0 TO 3 DO
18            SET rect TO (128 * x, 128 * y, CELLSIZE, CELLSIZE)
19            SET image TO pygame.image.load("MazeTilesheet.png").convert_alpha().subsurface(
rect)
20            SET image TO pygame.transform.scale(image, (self.tile, self.tile))
21            APPEND image TO self.cell_images
22            SET self.wizard_image TO pygame.Surface((self.tile, self.tile))
23            self.wizard_image.fill((255, 255, 0)) // Yellow
24            SET self.maze_offset_x TO DELTA_X
25            SET self.maze_offset_y TO DELTA_Y
26            SET self.visited_cells TO EMPTY LIST
27            SET self.highlight_surface TO pygame.Surface((self.tile, self.tile))
28            self.highlight_surface.set_alpha(50)
29            self.highlight_surface.fill((255, 255, 0)) // Yellow
30            SET self.darken_surface TO pygame.Surface((self.tile, self.tile))
31            self.darken_surface.set_alpha(100)
32            self.darken_surface.fill((0, 0, 0)) // Black
33            SET self.interactive TO interactive
34 END FUNCTION
```

Phương thức khởi tạo của lớp `MazeDrawer` nhận vào các tham số sau:

- `cols, rows`: Số lượng cột và hàng của mê cung.
- `cells`: Danh sách các ô của mê cung, mỗi ô chứa thông tin về các bức tường của nó.
- `interactive`: Biến boolean xác định liệu mê cung có tính tương tác hay không.

Trong phương thức khởi tạo, các biến toàn cục `sc`, `CELLSIZE`, `DELTA_X`, và `DELTA_Y` được khai báo. Giá trị `TILE` được tính toán dựa trên kích thước của cửa sổ và số cột của mê cung. Một bề mặt `tilesheet` được tạo để lưu trữ hình ảnh của các ô. Hình ảnh từ tệp `MazeTilesheet.png` được tải lên, cắt ra thành các ô nhỏ và thay đổi kích thước để phù hợp với kích thước `TILE`. Các hình ảnh này được lưu trữ trong danh sách `cell_images`.

Một hình ảnh đại diện cho nhân vật chính (wizard) được tạo và tô màu vàng. Vị trí của mê cung trên màn hình được xác định bằng cách sử dụng các giá trị `DELTA_X` và `DELTA_Y`. Các bề mặt `highlight_surface` và `darken_surface` được tạo để đánh dấu các ô đã thăm và làm mờ các ô chưa thăm.

4.6.2 Phương thức draw

Phương thức `draw` gọi phương thức `draw_cells` để vẽ các ô của mê cung. Nó có thể nhận một ô được chọn (`select_cell`) và một biến boolean (`visit`) để xác định liệu ô này có được đánh dấu là đã thăm hay không.

```
1 FUNCTION draw(select_cell=None, visit=True)
2     CALL self.draw_cells(select_cell, visit)
3 END FUNCTION
```

4.6.3 Phương thức draw_cells

```
1 FUNCTION draw_cells(select_cell=None, visit=True)
2     FOR index, cell IN ENUMERATE(self.cells) DO
3         SET row_index TO index DIV self.cols
4         SET col_index TO index MOD self.rows
5         SET hasWallTop TO cell.walls['top']
6         SET hasWallRight TO cell.walls['right']
7         SET hasWallBottom TO cell.walls['bottom']
8         SET hasWallLeft TO cell.walls['left']
9
10        SET x TO col_index * self.tile
11        SET y TO row_index * self.tile
12
13        SET cell_index TO 0
14        IF hasWallRight THEN
15            cell_index += 1
16        IF hasWallLeft THEN
17            cell_index += 2
18        IF hasWallBottom THEN
19            cell_index += 4
20        IF hasWallTop THEN
21            cell_index += 8
22        self.sc.blit(self.cell_images[cell_index], (x + self.maze_offset_x, y + self.
maze_offset_y))
23
24        IF cell EQUALS select_cell THEN
25            IF visit THEN
26                APPEND cell TO self.visited_cells
27                self.sc.blit(self.highlight_surface, (x + self.maze_offset_x, y + self.
maze_offset_y))
28                self.sc.blit(self.wizard_image, (x + self.maze_offset_x + self.tile DIV 2, y
+ self.maze_offset_y + self.tile DIV 2))
29
30            IF cell NOT IN self.visited_cells AND self.interactive THEN
31                self.sc.blit(self.darken_surface, (x + self.maze_offset_x, y + self.
maze_offset_y))
32 END FUNCTION
```

Phương thức này vẽ từng ô của mê cung trên màn hình. Quá trình vẽ diễn ra như sau:

- Duyệt qua danh sách các ô và xác định vị trí của mỗi ô trong mê cung dựa trên chỉ số của nó.
- Kiểm tra các bức tường của ô để xác định chỉ số hình ảnh tương ứng trong danh sách `cell_images`.
- Vẽ hình ảnh ô lên màn hình tại vị trí tương ứng, có tính đến các giá trị `maze_offset_x` và `maze_offset_y`.
- Nếu ô hiện tại là ô được chọn, đánh dấu ô đó và vẽ hình ảnh nhân vật chính lên.
- Nếu ô chưa được thăm và mê cung có tính tương tác, làm mờ ô đó.

4.6.4 Kết luận

Đoạn mã Python trên định nghĩa một lớp quản lý việc vẽ mê cung trên màn hình bằng thư viện pygame. Lớp này giúp tải và xử lý hình ảnh từ tileset, vẽ các ô của mê cung và quản lý các yếu tố tương tác như các ô đã thăm và ô hiện tại. Đây là một ví dụ điển hình về việc sử dụng pygame để tạo giao diện đồ họa và tương tác trong một trò chơi hoặc ứng dụng mô phỏng mê cung.

4.7 Class Mouse

4.7.1 Phương thức khởi tạo

```
1   FUNCTION Initialize(x, y, sprite_sheet_path, cell_size)
2     SET self.x TO x
3     SET self.y TO y
4     SET self.sprite_sheet_path TO sprite_sheet_path
5     SET self.cell_size TO cell_size
6     SET self.direction TO "None"
7     SET tmpBB TO {"None": [i FOR i IN range(5)]}
8     SET tmp_die TO {"Die": [i FOR i IN range(8)]}
9     SET self.animation TO PlayerAnimation(tmpBB, "Jerry_animationBB.png", 5, 1, cell_size,
10      3)
11    SET self.animation_die TO PlayerAnimation(tmp_die, "mouse_die.png", 8, 1, cell_size,
12      3)
11 END FUNCTION
```

Listing 4.10: phương thức khởi tạo của Class Mouse

Phương thức khởi tạo của lớp Mouse nhận vào các tham số sau:

- **x, y:** Tọa độ ban đầu của đối tượng chuột trên lưới (grid).
- **sprite_sheet_path:** Đường dẫn tới tệp sprite sheet chứa các khung hình hoạt hình của chuột.
- **cell_size:** Kích thước của mỗi ô trong lưới (grid).

Trong phương thức khởi tạo, các biến đối tượng được thiết lập bao gồm tọa độ x và y, đường dẫn sprite sheet, kích thước ô, và hướng di chuyển ban đầu là "None". Hai từ điển tạm thời tmpBB và tmp_die được tạo ra để chứa các chỉ số khung hình cho các trạng thái khác nhau của chuột (di chuyển và chết). Các đối tượng PlayerAnimation được khởi tạo cho các trạng thái này, với các tham số tương ứng.

4.7.2 Phương thức move

```
1   FUNCTION move(direction, grid_cells, Daisize, rows)
2     SET self.direction TO direction
3     self.animation.set_direction(direction)
4     self.animation_die.set_direction(direction)
5 END FUNCTION
```

Listing 4.11: phương thức move của class Mouse

Phương thức move thay đổi hướng di chuyển của đối tượng chuột. Nó nhận vào các tham số:

- **direction:** Hướng di chuyển mới của chuột.
- **grid_cells:** Danh sách các ô trong lưới (không sử dụng trong đoạn mã này).
- **Daisize:** Kích thước của lưới (không sử dụng trong đoạn mã này).
- **rows:** Số hàng của lưới (không sử dụng trong đoạn mã này).

Phương thức này thiết lập hướng di chuyển mới cho chuột và cập nhật hoạt hình tương ứng bằng cách gọi phương thức set_direction của đối tượng animation.

4.7.3 Phương thức draw

Phương thức `draw` vẽ khung hình hiện tại của chuột lên màn hình. Nó nhận vào các tham số:

- `screen`: Bề mặt (surface) của pygame nơi đối tượng chuột sẽ được vẽ lên.
- `TILE`: Kích thước của một ô trong lưới.

Phương thức này kiểm tra hướng di chuyển hiện tại của chuột và vẽ khung hình tương ứng lên màn hình. Quá trình vẽ diễn ra như sau:

- Gọi phương thức `set_direction` để đảm bảo hoạt hình đúng hướng.
- Nếu hướng là "None", lấy khung hình hiện tại từ `animation` và vẽ lên màn hình tại tọa độ tương ứng với tọa độ của chuột và kích thước ô `TILE`. Sau đó, cập nhật hoạt hình.
- Nếu hướng là "Die", lấy khung hình hiện tại từ `animation_die` và vẽ lên màn hình. Sau đó, cập nhật hoạt hình.

4.7.4 Kết luận

Đoạn mã Python trên định nghĩa một lớp quản lý đối tượng chuột trong game, sử dụng thư viện pygame để xử lý hoạt hình và vẽ đối tượng. Lớp này giúp khởi tạo đối tượng chuột, quản lý và cập nhật hoạt hình khi chuột di chuyển hoặc chết, và vẽ đối tượng lên màn hình. Việc hiểu rõ và sử dụng hiệu quả lớp này có thể giúp tối ưu hóa hiệu suất và chất lượng hoạt hình trong game.

```
1
2
3
4
5 FUNCTION draw(screen, TILE)
6     // Draw current frame on screen
7     self.animation.set_direction(self.direction)
8     IF self.direction EQUALS "None" THEN
9         SET current_frame TO self.animation.get_current_frame()
10        screen.blit(current_frame, (self.x * TILE + DELTA_X, self.y * TILE + DELTA_Y))
11        self.animation.update()
12    ELSE IF self.direction EQUALS "Die" THEN
13        SET current_frame TO self.animation_die.get_current_frame()
14        screen.blit(current_frame, (self.x * TILE + DELTA_X, self.y * TILE + DELTA_Y))
15        self.animation_die.update()
16    END IF
17 END FUNCTION
18 END Mouse Class
```

Listing 4.12: Mã giả cho class Player_Animation

4.8 Class start

4.8.1 Phương thức khởi tạo

Phương thức khởi tạo của lớp Start nhận vào các tham số sau:

- `x, y`: Tọa độ ban đầu của đối tượng bắt đầu trên lưới (grid).
- `sprite_sheet_path`: Đường dẫn tới tệp sprite sheet chứa các khung hình hoạt hình của đối tượng.
- `cell_size`: Kích thước của mỗi ô trong lưới (grid).

Trong phương thức khởi tạo, các biến đối tượng được thiết lập bao gồm tọa độ `x` và `y`, đường dẫn sprite sheet, kích thước ô, và hướng di chuyển ban đầu là "None". Một từ điển tạm thời `tmpBB` được tạo ra để chứa các chỉ số khung hình cho trạng thái mặc định của đối tượng (6 khung hình). Đối tượng `PlayerAnimation` được khởi tạo cho trạng thái này với các tham số tương ứng.

4.8.2 Phương thức draw

Phương thức `draw` vẽ khung hình hiện tại của đối tượng bắt đầu lên màn hình. Nó nhận vào các tham số:

- `screen`: Bề mặt (surface) của `pygame` nơi đối tượng sẽ được vẽ lên.
- `TILE`: Kích thước của một ô trong lưới.

Phương thức này kiểm tra hướng di chuyển hiện tại của đối tượng và vẽ khung hình tương ứng lên màn hình. Quá trình vẽ diễn ra như sau:

- Gọi phương thức `set_direction` để đảm bảo hoạt hình đúng hướng.
- Nếu hướng là "None", lấy khung hình hiện tại từ `animation` và vẽ lên màn hình tại tọa độ tương ứng với tọa độ của đối tượng và kích thước ô `TILE`. Sau đó, cập nhật hoạt hình.

4.8.3 Kết luận

Doạn mã Python trên định nghĩa một lớp quản lý đối tượng bắt đầu trong game, sử dụng thư viện `pygame` để xử lý hoạt hình và vẽ đối tượng. Lớp này giúp khởi tạo đối tượng, quản lý và cập nhật hoạt hình, và vẽ đối tượng lên màn hình. Việc hiểu rõ và sử dụng hiệu quả lớp này có thể giúp tối ưu hóa hiệu suất và chất lượng hoạt hình trong game.

```
1 BEGIN Start Class
2 FUNCTION Initialize(x, y, sprite_sheet_path, cell_size)
3     SET self.x TO x
4     SET self.y TO y
5     SET self.sprite_sheet_path TO sprite_sheet_path
6     SET self.cell_size TO cell_size
7     SET self.direction TO "None"
8     SET tmpBB TO {"None": [i for i in range(6)]}
9     SET self.animation TO PlayerAnimation(tmpBB, "Start_animation.png", 6, 1, cell_size,
10    10)
11 END FUNCTION
12 FUNCTION draw(screen, TILE)
13     // Draw current frame on screen
14     self.animation.set_direction(self.direction)
15     IF self.direction EQUALS "None" THEN
16         SET current_frame TO self.animation.get_current_frame()
17         screen.blit(current_frame, (self.x * TILE + DELTA_X, self.y * TILE + DELTA_Y))
18         self.animation.update()
19     END IF
20 END FUNCTION
21 END Start Class
```

Listing 4.13: Mã giả cho class Player_Animation

4.9 Tạo mê cung

Ở phần này, nhóm đã sử dụng thuật toán DFS để tạo mê cung ngẫu nhiên cho từng ván chơi, kết hợp với việc "xây" một vài bức tường để đảm bảo mê cung có độ khó nhất định.

4.9.1 Hàm `remove_walls(current, next)`

```
1 function remove_walls(current, next, grid_cells, cols, rows, flag):
2     dx = current.x - next.x
3     random_value = random_number()
4     if dx == 1:
5         if current.x < cols - 1 and random_value > 0.4 and flag is true:
6             current.walls['right'] = false
7             adjacent_cell = grid_cells[current.x + current.y * cols + 1]
```

```

8         adjacent_cell.walls['left'] = false
9         current.walls['left'] = false
10        next.walls['right'] = false
11    elif dx == -1:
12        if current.x > 0 and random_value > 0.4 and flag is true:
13            current.walls['left'] = false
14            adjacent_cell = grid_cells[current.x + current.y * cols - 1]
15            adjacent_cell.walls['right'] = false
16            current.walls['right'] = false
17            next.walls['left'] = false
18    dy = current.y - next.y
19    if dy == 1:
20        current.walls['top'] = false
21        next.walls['bottom'] = false
22
23    elif dy == -1:
24        current.walls['bottom'] = false
25        next.walls['top'] = false

```

Listing 4.14: Mã giả cho hàm remove_walls(current, next, flag)

- Đây là hàm quan trọng trong công đoạn tạo mê cung. Như ở phần Class cell ở trên, ta đã khởi tạo mỗi ô đều có 4 bức tường, do đó để tạo ra mê cung ta cần xóa một số bức tường.
- Tham số truyền vào trong 2 hàm này là ô đang xét hiện tại (current) và ô bên cạnh nó (next).
- Bằng cách xét vị trí tương đối của hai ô dựa trên hoành độ và tung độ của chúng, ta có thể xóa đi các bức tường giữa 2 ô.
- Một khác, để khiến mê cung có thể chứa nhiều đường đi giữa 2 điểm (Vì đặc điểm của thuật toán DFS khi tạo mê cung là chỉ thăm qua mỗi ô một lần duy nhất, do đó nó không tạo thành chu trình). Vì vậy, ta cần xóa bỏ ngẫu nhiên thêm một vài bức tường nữa để tạo ra được nhiều đường đi hơn.
- Ở đây, nhóm truyền một biến kiểu bool là flag để "bật cờ" cho phép xóa thêm một số bức tường dựa trên tỉ lệ được random. Nếu tỉ lệ random > 0.4, ta sẽ xóa thêm một số bức tường khác nếu các ô xung quanh nó vẫn còn nằm trong mê cung

4.9.2 Hàm build_walls(current, next)

```

1 function build_walls(current, next):
2     dx = current.x - next.x
3     if dx == 1:
4         current.walls['left'] = true
5         next.walls['right'] = true
6     if dx == -1:
7         current.walls['right'] = true
8         next.walls['left'] = true
9     dy = current.y - next.y
10    if dy == 1:
11        current.walls['top'] = true
12        next.walls['bottom'] = true
13    if dy == -1:
14        current.walls['bottom'] = true
15        next.walls['top'] = true

```

Listing 4.15: Mã giả cho hàm bulid_walls(current, next)

- Đây là hàm dùng để vẽ thêm các bức tường sau khi tạo ra mê cung bằng thuật toán DFS.
- Cũng bằng cách truyền vào hàm hai ô current và next tương tự như trong hàm remove_walls, ta có thể tạo thêm một số bức tường cho mê cung bằng cách gán các giá trị của "bức tường" trở thành True.

4.9.3 Hàm divide_maze

```
1  function divide_maze(grid_cells, cols, rows):
2      start = grid_cells[0]
3      rad = (cols^2 + rows^2) / 4
4      list_cell = []
5      for each vertex in grid_cells:
6          if euclid_distance(start, vertex) >= rad and euclid_distance(start, vertex) <= 3
7              * rad:
8                  list_cell.append(vertex)
9
10     while length(DFS_spread(grid_cells, start, cols, rows)) equal to length(grid_cells):
11
12         boundary = random_choice(list_cell)
13         //get the list of available surrounding cells
14         neighbors = boundary.check_neighbors_pass(grid_cells, cols, rows)
15         neighbor = random_choice(neighbors)
16         build_walls(boundary, neighbor)
17         if (boundary.walls['top'] and boundary.walls['bottom'] and boundary.walls['left']
18             and boundary.walls['right']) or (neighbor.walls['top'] and neighbor.walls['bottom']
19             and neighbor.walls['left'] and neighbor.walls['right']):
20             remove_walls(boundary, neighbor, grid_cells, cols, rows, False)
```

Listing 4.16: Mã giả cho hàm divide_maze(grid_cells, cols, rows)

- Hàm divid_maze hoạt động như sau: ta sẽ chọn các ô cách xa khoảng ô đầu tiên trong mê cung một khoảng lớn hơn nửa đường chéo của mê cung và bé hơn căn 3 chia 2 độ dài đường chéo
- Bằng cách lặp ra những ô như thế, ta sẽ tiến hành xây dựng những bức tường để khiến mê cung tồn tại ít nhất hai điểm không thể đến được nhau.
- Bằng cách sử dụng hàm loang DFS, ta có thể biết được liệu rằng có tồn tại 2 ô nào không đến được nhau không (bởi nếu điều kiện len(DFS_spread(grid_cells, start, cols, rows)) == len(grid_cells) sai thì nghĩa là mục tiêu của chúng ta đã thỏa mãn
- Bên trong vòng while, ta sẽ tiến hành xây dựng nên những bức tường nhằm tạo ra 2 "vùng" biệt lập trong mê cung, và điều này cũng được thực hiện nhờ vào hàm choice chọn ra ngẫu nhiên các ô cạnh nhau để xây tường.
- Cuối cùng, để mê cung đẹp hơn, ta đảm bảo không có ô nào bị đóng kín 4 bức tường. Ta sử dụng hàm remove_walls nói ở trên để thực hiện bước này

4.9.4 Hàm tạo mê cung generate_maze(cols, rows, TILE)

```
1  function generate_maze(cols, rows, TILE):
2      // Create a list of grid cells
3      grid_cells = [create_cell(x, y) for y in range(rows) for x in range(cols)]
4      current_cell = grid_cells[0]
5      array = []
6      break_count = 1
7      maze = grid_cells
8
9      // Create a maze drawer object outside the loop
10     drawer = create_maze_drawer(cols, rows, maze, interactive=True)
11
12     // While the maze is not completely generated
13     while break_count is not equal to (cols * rows):
14         // Mark the current cell as visited
15         current_cell.visited = true
16
17         // Check for unvisited neighboring cells
18         next_cell = current_cell.check_neighbors(grid_cells, cols, rows)
19
20         if next_cell is not null:
21             // Mark the next cell as visited
```

```

22     next_cell.visited = true
23     // Increment the break count
24     break_count += 1
25     // Add the current cell to the stack
26     array.append(current_cell)
27     // Remove walls between current and next cell
28     remove_walls(current_cell, next_cell, grid_cells, cols, rows, true)
29     // Move to the next cell
30     current_cell = next_cell
31     else if array is not empty:
32         // Backtrack to the last cell in the stack
33         current_cell = array.pop()
34
35     // Reset the visited status for all cells
36     for each cell in grid_cells:
37         cell.visited = false
38
39     // Further divide the maze if needed
40     divide_maze(grid_cells, cols, rows)
41
42     // Return the generated grid cells
43     return grid_cells

```

Listing 4.17: Mã giả cho hàm `generate_maze(cols, rows, TILE)`

- Hàm tạo mê cung sẽ nhận vào 3 tham số là số cột, số hàng và độ dài cạnh của mỗi ô.
- Đầu tiên, ta khởi tạo một mê cung gồm $cols \times rows$ ô, mỗi ô đều có 4 bức tường bằng dòng lệnh gọi `grid_cells` là một list của các đối tượng cell. Tiếp theo ta khởi tạo một array và biến đếm `break_count` (dùng để đếm số ô đã qua trong mê cung).
- Hàm tạo mê cung này sử dụng quay lui để tạo mê cung. Đầu tiên, tất cả các ô trong mê cung đều là các bức tường kín, bắt đầu từ ô góc trái trên ta sẽ tiến hành xóa đi các bức tường. Bằng cách gán ô đầu tiên là ô hiện tại đang xét (`current_cell`), ta di vào vòng while. Bằng phương thức `check_neighbors` đã nói ở trên, ta chọn ra một ô xung quanh (tất nhiên nằm trong mê cung) của ô đang xét là ô `next_cell`, và gán là ô đó đã đi qua rồi (`self.visited = True`) và tiến hành xóa đi bức tường giữa hai ô đó, và sẽ dựa vào tỉ lệ random để xóa thêm một trong các bức tường xung quanh nữa, đồng thời tăng biến `break_count` lên 1 (Đã qua ô đó rồi).
- Ta cũng đưa ô `current_cell` vào list (có tác dụng như một stack (First In First Out) để sử dụng quay lui (Nếu các ô tiếp theo đến ngõ cụt thì có thể quay lại các ô "còn có thể đi được" để đi sang các hướng khác), việc này giúp ta có thể đi hết tất cả các ô trong mê cung mà không sợ đã đi đến ngõ cụt và xóa các bức tường. Mỗi khi biến gặp biến `next_cell` chưa được thăm.
- Đến khi `break_count` bằng chiều dài của list `grid_cells`, nghĩa là tất cả các ô đã được đi qua, dừng vòng lặp.
- Tiếp theo, hàm `divide_maze` được gọi để đảm bảo vùng sẽ có ô đầu và ô cuối không gặp nhau để làm phần thông báo người dùng chọn lại vị trí xuất phát - đích.

4.10 Chọn điểm đầu và điểm cuối

4.10.1 Chọn điểm đầu - điểm cuối bằng cách random

- Ta sử dụng phương thức `choice` chọn ra một điểm ngẫu nhiên bất kì. Sử dụng thuật toán DFS để "loang" đến tất cả các ô có thể đến được (vì mê cung cũng là một dạng đồ thị liên thông).
- Sau đó dựa vào hàm tính toán khoảng cách, lấy ra các ô trong danh sách các ô có thể đến được đảm bảo một khoảng cách đủ lớn nhất định, tiếp tục sử dụng hàm `choice` để chọn ra ô đích. Nếu danh sách ô rỗng, ta random để chọn lại ô khác mà có các điểm đích thỏa mãn khoảng cách lớn hơn khoảng nhất định cho trước.

Hàm tính toán khoảng cách euclid_distance(cell1, cell2)

```
1 def euclid_distance(cell1, cell2):
2     return (cell1.x - cell2.x)**2 + (cell1.y - cell2.y)**2
```

Listing 4.18: Hàm tính toán trả về bình phương khoảng cách Euclid giữa hai ô

Hàm loang để tìm các ô có thể đến được từ một ô bất kì

```
1 function DFS_spread(grid_cells, start, cols, rows)
2     stack = [start]
3     path = []
4     while stack not empty do
5         vertex = stack.pop()
6         if not vertex.passed then
7             path.append(vertex)
8             vertex.passed = true
9             neighbors = vertex.check_neighbors_pass(grid_cells, cols, rows)
10            for neighbor in neighbors do
11                if not neighbor.passed then
12                    stack.append(neighbor)
13                end if
14            end for
15        end if
16    end while
17    for i = 0 to length(grid_cells) do
18        grid_cells[i].passed = false
19    end for
20    return path
21 end function
```

- Bằng cách sử dụng quay lui nhưng đi hết tất các ô xung quanh "có thể qua" (không bị chắn bởi tường), ta có thể loang từ một ô đến các ô xung quanh nó
- Ban đầu, ta để start là phần tử đầu tiên trong stack, duyệt hết các ô kế cận có thể qua của start, ta tiếp tục loang hết các ô kế cận của các ô đó. Nếu ô nào chưa được thăm, ta gán self.pass = True nghĩa là đã thăm rồi, đồng thời thêm ô đó vào list path (danh sách chưa ô có thể đến)
- Đến ngõ cụt, stack sẽ xóa dần các phần tử của ô ra, duyệt các hướng còn lại, đến khi stack rỗng nghĩa là đã duyệt qua tất cả các ô có thể đến rồi
- Ta đặt lại self.pass = False cho tất cả các ô để có thể gọi các hàm tìm nước đi, và trả về danh sách các ô có thể đến

Hàm chọn ra điểm đích thỏa mãn khoảng cách cho trước select_start_end(grid_cells, cols, rows)

```
1 function select_start_end(grid_cells, cols, rows)
2     list_end = []
3     while not list_end do
4         random_start = random.choice(grid_cells)
5         path = DFS_spread(grid_cells, random_start, cols, rows)
6         rad = (cols^2 + rows^2) / 4
7         for vertex in path do
8             if euclid_distance(vertex, random_start) >= rad then
9                 list_end.append(vertex)
10            end if
11        end for
12    end while
13    random_end = random.choice(list_end)
14    return (random_start, random_end)
15 end function
```

Listing 4.19: Hàm lựa chọn điểm đầu và đích select_start_end(grid_cells, cols, rows)

- Ta tạo một list là các điểm trong danh sách path được trả về từ hàm loang thỏa mãn khoảng cách lớn hơn hoặc bằng độ dài nửa đường chéo của mê cung, nếu list rỗng thì ta lại random điểm khác cho đến khi list_end chưa phần tử.

-
- Sau đó, sử dụng hàm choice để lấy ra phần tử bất kì trong list_end, hàm trả về tuple chứa ô bắt đầu và ô kết thúc.

4.10.2 Người chơi tự chọn điểm đầu - điểm cuối

- Hàm select_start_end_manually nhận các tham số là grid_cells, cols, rows, TILE, và sc, lần lượt là danh sách các ô trong lưới, số cột và số hàng của lưới, kích thước của mỗi ô, và màn hình pygame.
- Các biến start_color, end_color, và bright_color định nghĩa màu sắc cho điểm bắt đầu, điểm kết thúc và màu sáng.
- Hai biến start_selected và end_selected xác định xem điểm bắt đầu và điểm kết thúc đã được chọn chưa.
- Vòng lặp while True liên tục kiểm tra sự kiện chuột và hiển thị các thay đổi trên màn hình pygame cho đến khi cả hai điểm đã được chọn.
- Trong vòng lặp for event in pygame.event.get(), hàm xử lý các sự kiện như nhấn nút thoát và nhấn chuột.
- Khi một sự kiện chuột được nhận, vị trí chuột được xác định, và ô tương ứng trong lưới được xác định dựa trên vị trí đó.
- Nếu chưa có điểm bắt đầu nào được chọn, ô đó sẽ được chọn làm điểm bắt đầu và được vẽ màu đen trên màn hình.
- Nếu đã có điểm bắt đầu được chọn, nhưng chưa có điểm kết thúc, nếu ô được nhấn khác với ô bắt đầu, thì ô đó sẽ được chọn làm điểm kết thúc và được vẽ màu xanh trên màn hình.
- Sau khi cả hai điểm đã được chọn, hàm sẽ trả về các ô tương ứng của điểm bắt đầu và điểm kết thúc.

```

1 BEGIN select_start_end_manually FUNCTION
2     SET start_color TO (0, 255, 0) # Green
3     SET end_color TO (255, 0, 0) # Red
4     SET bright_color TO (255, 255, 0) # Yellow
5
6     SET start_selected TO False
7     SET end_selected TO False
8
9     WHILE True DO
10         FOR EACH event IN Pygame.Get_Events() DO
11             IF event.type EQUALS Pygame.QUIT THEN
12                 Pygame.Quit()
13                 Quit()
14             ELSE IF event.type EQUALS Pygame.MOUSEBUTTONDOWN THEN
15                 SET mouse_pos TO Pygame.Mouse.Get_Pos()
16                 SET col TO Integer_Divide(mouse_pos[0], TILE)
17                 SET row TO Integer_Divide(mouse_pos[1], TILE)
18                 SET index TO col + row * cols
19                 SET cell TO grid_cells[index]
20
21                 IF NOT start_selected THEN
22                     SET start_selected TO True
23                     SET start_cell TO cell
24                     Pygame.Draw.Rect(sc, Black, (start_cell.x * TILE + 4, start_cell.y *
TILE + 4, TILE - 8, TILE - 8))
25                 ELSE IF NOT end_selected THEN
26                     IF cell NOT_EQUALS start_cell THEN
27                         SET end_selected TO True
28                         SET end_cell TO cell
29                         Pygame.Draw.Rect(sc, Green, (end_cell.x * TILE + 4, end_cell.y *
TILE + 4, TILE - 8, TILE - 8))
30                         Pygame.Time.Delay(400)

```

```

31             END IF
32         END IF
33     END IF
34 END FOR_EACH
35
36     Pygame.Display.Update()
37
38     IF start_selected AND end_selected THEN
39         RETURN start_cell, end_cell
40     END IF
41 END WHILE
42 END select_start_end_manually FUNCTION
43
44 SET current_direction TO "None"

```

4.10.3 Cho Người chơi chọn lại ví trí nếu không gặp nhau

Khởi tạo các biến cần thiết

Hàm bắt đầu bằng việc khởi tạo các biến cần thiết như biến `running` để theo dõi trạng thái của vòng lặp, và các biến khác như `screen_width`, `screen_height`, `surface`, `AskBot`, `shadow_offset`, `shadow_color`, `shadow_position`.

Vẽ hộp thoại chọn lại tùy chọn

Hàm sẽ vẽ hộp thoại lên màn hình với hình ảnh `AskBot`, đồng thời cũng vẽ một bóng đổ phía dưới hộp thoại để tạo hiệu ứng 3D.

Hiển thị các tùy chọn và xử lý sự kiện

Dùng một vòng lặp để kiểm tra các sự kiện của người chơi. Người chơi có thể di chuyển qua lại giữa các tùy chọn bằng cách nhấn các phím mũi tên trái và phải. Khi người chơi nhấn phím Enter, hàm sẽ thực hiện một hành động thuộc vào tùy chọn đã chọn.

Cập nhật màn hình

Mọi thay đổi trên màn hình sẽ được cập nhật bằng cách gọi `pygame.display.update()`.

Trả về kết quả

Khi người chơi đã chọn một tùy chọn, hàm sẽ trả về một cặp giá trị là vị trí start và end tương ứng với lựa chọn của họ.

Hàm này được thiết kế để tương tác với người chơi trong quá trình chơi game và cho phép họ chọn lại một tùy chọn nếu cần thiết.

```

1  FUNCTION Ask_Choose_Again(grid_cells, cols, rows, TILE, sc, drawer, background)
2      SET selected_option_choose_again TO GLOBAL VARIABLE
3      SET running TO True
4      WHILE running DO
5          SET screen_width TO pygame.display.Info().current_w
6          SET screen_height TO pygame.display.Info().current_h
7          SET surface TO pygame.Surface([screen_width, screen_height], pygame.SRCALPHA)
8          SET AskBot TO pygame.image.load('Ask_box2.png')
9          SET shadow_offset TO 5
10         SET shadow_color TO (0, 0, 0, 100)
11         SET shadow_position TO (screen_width // 10 + shadow_offset, screen_height // 8 +
12             shadow_offset)
12         CALL screen.blit(surface, (0, 0))
13         CALL pygame.draw.rect(surface, (128, 128, 128, 2), [0, 0, screen_width,
14             screen_height])
14         SET shadow_surface TO AskBot.copy()
15         CALL shadow_surface.fill(shadow_color, special_flags=pygame.BLEND_RGBA_MULT)
16         CALL screen.blit(shadow_surface, shadow_position)
17         CALL screen.blit(AskBot, (screen_width // 10, screen_height // 8))

```

```

18     SET option_font TO pygame.font.Font('8-BIT WONDER.ttf', 12)
19
20     FOR i, option IN ENUMERATE(menu_options) DO
21         SET text_color TO White IF i == selected_option_choose_again ELSE Red
22         SET text TO option_font.render(option, True, text_color)
23         SET text_rect TO text.get_rect(center=(WIDTH // 2 + (i - 0.5) * 130 - 22,
24 HEIGHT // 2 - 5))
25         CALL screen.blit(text, text_rect)
26
27     FOR event IN pygame.event.get() DO
28         IF event.type == pygame.QUIT THEN
29             CALL pygame.quit()
30             CALL sys.exit()
31         ELSE IF event.type == pygame.KEYDOWN THEN
32             IF event.key == pygame.K_LEFT THEN
33                 SET selected_option_choose_again TO (selected_option_choose_again -
34 1) MOD len(menu_options)
35             ELSE IF event.key == pygame.K_RIGHT THEN
36                 SET selected_option_choose_again TO (selected_option_choose_again +
37 1) MOD len(menu_options)
38             ELSE IF event.key == pygame.K_RETURN THEN
39                 IF selected_option_choose_again == 0 THEN
40                     CALL intro_transition()
41                     CALL screen.blit(background, (0, 0))
42                     CALL drawer.draw()
43                     FOR cell IN grid_cells DO
44                         CALL cell.draw(sc, TILE)
45                     CALL pygame.display.flip()
46                     CALL pygame.display.update()
47                     CALL clock.tick(60)
48                     SET start, end TO select_start_end(grid_cells, cols, rows)
49                     PRINT "Random Position selected"
50                     RETURN start, end
51                 ELSE IF selected_option_choose_again == 1 THEN
52                     CALL intro_transition()
53                     CALL screen.blit(background, (0, 0))
54                     CALL drawer.draw()
55                     FOR cell IN grid_cells DO
56                         CALL cell.draw(sc, TILE)
57                     CALL pygame.display.flip()
58                     CALL pygame.display.update()
59                     CALL clock.tick(60)
60                     SET start, end TO select_start_end_manually(grid_cells, cols,
61 rows, TILE, sc)
62                     PRINT "Manual Position selected"
63                     RETURN start, end
64             CALL pygame.display.update()
65     END FUNCTION

```

4.11 Các thuật toán tìm đường đi được sử dụng trong game

4.11.1 Thuật toán DFS

Ý tưởng thuật toán DFS và cài đặt

```

1 function DFS_findPath(grid_cells, start, end, cols, rows, visualize):
2     stack = [start]
3     trace = {start: -1}
4     list_visualize = []
5     while stack is not empty:
6         vertex = stack.pop()
7         if visualize is True:
8             list_visualize.append(vertex)
9         if vertex is end:
10            break
11        if vertex.passed is False:

```

```

12     vertex.passed = True
13     neighbors = vertex.check_neighbors_pass(grid_cells, cols, rows)
14     for each neighbor in neighbors:
15         if neighbor.passed is False:
16             stack.append(neighbor)
17             trace[neighbor] = vertex
18     // Reset passed status for all cells
19     for each cell in grid_cells:
20         cell.passed = False
21     // Trace back the path from end to start
22     u = end
23     path = []
24     path.append(u)
25     u = trace[u]
26     while u is not -1:
27         path.append(u)
28         u = trace[u]
29     if visualize is True:
30         return [list_visualize, reverse(path)]
31     return reverse(path)

```

Listing 4.20: Mã giả cho thuật toán DFS

- DFS (Depth First Search) là thuật toán tìm đường đi theo chiều sâu. Bằng cách sử dụng quay lui, khi xét đến một ô, ta sẽ xét các ô bên cạnh nó và đưa các ô đó vào stack (Cấu trúc dữ liệu First In Last Out, ở đây vì list trong python cũng có thể được dùng như stack nên ta dùng list).
- Cứ như vậy, ta sẽ xét đến các ô bên tiếp theo của ô được xét hiện tại, nếu ô đó không đi được nữa thì trở lại ô trước đó rồi duyệt theo các hướng khác, cho đến khi stack rỗng thì dừng thuật toán.
- Trong đoạn code trên, thuật toán được cài đặt bằng cách duyệt vòng while với điều kiện stack chưa rỗng (Còn hướng để đi tiếp). Khi đó, ta lần lượt lấy ra các ô trên đỉnh của stack, rồi duyệt qua các ô xung quanh "có thể qua được" từ ô đó, và bỏ các ô chưa được thăm vào stack để duyệt tiếp cho các vòng lặp tiếp theo. Vòng lặp sẽ bị break nếu gặp được ô end.
- Dictionary trace có tác dụng giúp ta truy vết đường đi. Với value là ô đang xét, còn key là ô có thể đến được thông qua ô value. Bằng việc tổ chức như vậy, ta có thể truy vết lại đường đi đến ô end bằng cách gọi ra value của ô end trong dictionary (bằng lệnh `u = trace[u]`) và lui dần về cho đến khi gặp start). Các ô trên được trả về list path, việc đảo ngược đảm bảo ô start ở đầu danh sách.
- Ở đoạn cuối của hàm, do ta đã sử dụng thuộc tính `self.pass` để gán bằng `True` (đã được thăm) trong quá trình chạy thuật toán DFS, ta phải gán lại thuộc tính `self.pass = False` cho tất cả các ô nhằm đảm bảo có thể gọi tiếp các hàm tìm đường đi ở những lần khác trong trò chơi.
- Danh sách list `_visualize` dùng để lưu các ô mà thuật toán đã đi qua một cách tuần tự, giúp ta vẽ lên được qua trình chạy thuật toán. Biến bool `visualize` được truyền vào giúp ta điều chỉnh trả về `visualize` hay không.

Đánh giá độ phức tạp thuật toán

:

- **Độ phức tạp thời gian:** Trong thuật toán DFS, độ phức tạp là $O(V + E)$ với V là số đỉnh và E là số cạnh trên đồ thị, bởi mỗi đỉnh ta chỉ duyệt qua 1 lần và mỗi cạnh ta cũng chỉ duyệt qua 2 lần, tối giản thì độ phức tạp sẽ trở thành $O(V + E)$. Đối với mô hình mê cung, mỗi đỉnh là một ô trong mê cung, và số "lối đi" giữa hai ô chung cạnh là cạnh, do mỗi ô có tối đa 4 lối đi sang ô khác nên số cạnh là $4 \times \text{cols} \times \text{rows}$ còn số đỉnh là $\text{cols} \times \text{rows}$, do đó độ phức tạp của thuật toán DFS khi tìm đường đi trong mê cung là $O(\text{cols} \times \text{rows})$. Nếu cạnh mê cung có cạnh là N thì độ phức tạp lúc này là $O(N^2)$
- **Độ phức tạp không gian:** Vì mê cung chứa N^2 ô nên độ phức tạp là $O(N^2)$

4.11.2 Thuật toán BFS

Ý tưởng cho thuật toán BFS và cài đặt

```
1  function BFS_findPath(grid_cells, start, end, cols, rows, visualize):
2      queue = new Queue()
3      queue.put(start)
4      start.passed = True
5      trace = {start: -1}
6      visual = []
7
8      while queue is not empty:
9          vertex = queue.get()
10
11         if visualize is True:
12             visual.append(vertex)
13
14         if vertex is end:
15             break
16
17         neighbors = vertex.check_neighbors_pass(grid_cells, cols, rows)
18
19         for each neighbor in neighbors:
20             if neighbor.passed is False:
21                 queue.put(neighbor)
22                 trace[neighbor] = vertex
23                 neighbor.passed = True
24
25         // Reset passed status for all cells
26         for each cell in grid_cells:
27             cell.passed = False
28
29         // Trace back the path from end to start
30         u = end
31         path = []
32         path.append(u)
33         u = trace[u]
34
35         while u is not -1:
36             path.append(u)
37             u = trace[u]
38
39         if visualize is True:
40             return [visual, reverse(path)]
41
42     return reverse(path)
```

Listing 4.21: Mã giả cho thuật toán BFS

- Thuật toán BFS (Breadth First Search) là thuật toán tìm kiếm theo chiều rộng. Thay vì sử dụng quay lui để di tiếp các ô xung quanh của ô đang xét cho đến khi không thể di tiếp nữa, ta sử dụng hàng đợi Queue trong module Queue (Cấu trúc dữ liệu First In First Out).
- Bằng cách duyệt hết các ô đã được thêm vào hàng đợi, rồi mới duyệt tiếp các ô xung quanh của ô trong hàng đợi, ta có thể tìm được đường đi ngắn nhất trong đồ thị (đường đi nào đến đích nhanh nhất là đường đi ngắn nhất).
- Trong đoạn code trên, bằng việc xét điều kiện hàng đợi không rỗng trong vòng while (nghĩa là còn ô để di tiếp). Ta sẽ xét các ô xung quanh ô đang xét bằng check_neighbors_pass, và bỏ các ô chưa được thăm vào hàng đợi. Việc này giúp ta "loang dần" các điểm được thăm từ điểm start, đến khi gặp end thì ta break và kết thúc vòng lặp.
- Việc hiển thị visualize và truy vết lại đường đi ngắn nhất cũng được thực hiện tương tự thuật toán DFS.

Đánh giá độ phức tạp của thuật toán

:

- **Độ phức tạp thời gian:** Tương tự thuật toán DFS, việc lấy một phần tử từ queue có độ phức tạp là $O(1)$. Do vậy, thuật toán BFS khi áp dụng cho mê cung cũng có độ phức tạp là $O(N^2)$
- **Độ phức tạp không gian:** $O(N^2)$

4.11.3 Thuật Toán A*

Ý tưởng và cài đặt của thuật toán A*

```

1   function AStar_findPath(grid_cells, start, end, cols, rows, visualize):
2       cor_start = (start.x, start.y)
3       cor_end = (end.x, end.y)
4       // Initialize g_score and f_score
5       g_score = {cell: infinity for each cell in grid_cells}
6       g_score[start] = 0
7       f_score = {cell: infinity for each cell in grid_cells}
8       f_score[start] = distance(cor_start, cor_end)
9       visual = []
10      trace = {start: -1}
11      find_index = lambda x, y: x + y * cols
12      res = new PriorityQueue()
13      res.put((distance(cor_start, cor_end), distance(cor_start, cor_end), (start.x, start.y)))
14      while res is not empty:
15          vertex_cor = res.get()[2]
16          vertex = grid_cells[find_index(vertex_cor[0], vertex_cor[1])]
17          if visualize is True:
18              visual.append(vertex)
19          if vertex is end:
20              break
21          neighbors = vertex.check_neighbors_pass(grid_cells, cols, rows)
22          for each neighbor in neighbors:
23              cor_neighbor = (neighbor.x, neighbor.y)
24              temp_g_score = g_score[vertex] + 1
25              temp_f_score = temp_g_score + distance(cor_neighbor, cor_end)
26              if temp_f_score < f_score[neighbor]:
27                  g_score[neighbor] = temp_g_score
28                  f_score[neighbor] = temp_f_score
29                  res.put((f_score[neighbor], distance(cor_neighbor, cor_end), cor_neighbor))
30          trace[neighbor] = vertex
31      u = end
32      path = []
33      path.append(u)
34      u = trace[u]
35      while u is not -1:
36          path.append(u)
37          u = trace[u]
38      if visualize is True:
39          return [visual, reverse(path)]
40      return reverse(path)

```

- Thuật toán Astar hoạt động bằng cách tối ưu hóa chi phí khoảng cách. Bằng cách xét hàm F là tổng của hai hàm: hàm ước lượng (heuristic function - gọi tắt là hàm H) và số bước bé nhất từ ô start để đến được ô đang xét (gọi tắt là hàm G). Hàm F sẽ tối ưu nếu hàm H và hàm G tối ưu. Thuật toán A star sẽ cố gắng đi theo hướng khiến cho hai hàm này tối ưu
- Sử dụng hàng đợi ưu tiên để tìm ra những ô tối ưu hàm F nhất trong hàng đợi, ta sẽ có thể định hướng được nên đi hướng nào mà không cần phải loang từ từ như BFS và DFS, do đó có thể xác định được đi ngắn nhất mà không cần phải đi qua quá nhiều điểm như BFS hay DFS để dò đường.
- Trong trò chơi này, nhóm cài đặt thuật toán A* bằng việc xây dựng hàm heuristic H là khoảng cách Manhattan từ ô đang xét đến ô kết thúc.
- Ban đầu, ta sẽ khởi tạo giá trị F, G, H ở mỗi ô là vô cực (infinity), ngoại trừ ô đầu tiên sẽ được thiết lập giá trị G là 0.

- Trong vòng lặp, ta duyệt đến khi nào hàng đợi ưu tiên rỗng hoặc gặp được ô end. Ở mỗi vòng lặp, hàng đợi ưu tiên sẽ được ưu tiên lấy ra đỉnh có hàm F bé nhất, nếu giữa các ô có F bằng nhau thì chọn ra ô có hàm H bé nhất(như một sự đảm bảo cho việc gần về đến đích). Ta lại duyệt tiếp các ô kề(tất nhiên không bị ngăn cách bởi tường với ô đó), tính toán lại chi phí F, G, H ở ô đó và chỉ gán lại khi các chi phí này bé hơn chi phí trước đó, rồi đưa các ô này vào hàng đợi ưu tiên. Cứ như vậy, thuật toán sẽ cố gắng đi theo hướng đi mà giá trị F và G là tốt nhất để đi đến đích (đường đi ngắn nhất).
- Tuy nhiên trong một số trường hợp nhỏ, việc tìm đường đi bằng thuật toán A* lại không hiệu quả bằng BFS nếu hàm heuristic không phù hợp với map của mê cung.
- Cơ chế visualize cũng như truy vết ở hàm này cũng tương tự ở hai hàm BFS_findPath và DFS_findPath.

Đánh giá độ phức tạp của thuật toán A*

:

- Độ phức tạp thời gian:** Do mỗi ô chỉ được update giá trị một lần (việc tối ưu hàm F ở mỗi ô đảm bảo không có ô nào được gán giá trị và thăm hai lần), do đó mỗi ô chỉ được duyệt qua tối đa một lần. Hơn nữa, việc thêm phần tử vào hàng đợi ưu tiên có chi phí là $\log(N)$ với N là số phần tử hiện có trong hàng đợi. Mà mỗi ô có tối đa 4 đỉnh kề, do vậy độ phức tạp của thuật toán A* là $O(E(\log V))$ nếu áp dụng cho đồ thị tổng quát. Tuy nhiên xét trong trường hợp mê cung, mỗi ô có tối đa 4 ô kề nhau, và nếu cạnh bằng N thì có tất cả là N^2 đỉnh. Vì vậy, độ phức tạp thời gian sẽ là $O(N^2 \log(4N^2))$, rút gọn là $O(N^2 \log(N))$
- Độ phức tạp về mặt không gian:** $O(N^2)$

4.11.4 Thuật Toán Dijkstra

Giới thiệu thuật toán Dijkstra

- Thuật toán Dijkstra hoạt động với mục tiêu tìm đường đi có tổng trọng số nhỏ nhất xuất phát từ điểm cho trước đến một điểm khác trên bản đồ.
- Trọng số của mỗi ô trên bản đồ là không âm.

```

1 Function Dijkstra_FindPath(grid_cells, start, end, cols, rows, visualize):
2     Initialize priority queue pq
3     Set INF to a large value (e.g., 10^6)
4     Create dist array with size (cols * rows) and set all values to INF
5     Create trace array with size (cols * rows) and set all values to -1
6     Create visited array with size (cols * rows) and set all values to False
7
8     Calculate start_index from start.x and start.y
9     Calculate end_index from end.x and end.y
10    Set dist[start_index] to 0
11    Initialize visual as an empty list
12    Add (0, start_index) to pq
13
14    While pq is not empty:
15        Pop the element with the smallest distance from pq and assign it to u
16        If visualize is True:
17            Append grid_cells[u] to visual
18        If u equals end_index:
19            Break the loop
20        Mark u as visited
21        Calculate u_x and u_y from u using cols
22        Get neighbors of grid_cells[u] using check_neighbors_pass method
23
24        For each neighbor in neighbors:
25            Calculate v as the one-dimensional index of neighbor
26            Calculate weight as the Euclidean distance between u and neighbor
27            If v is not visited and dist[v] > dist[u] + weight:

```

```

28             Update dist[v] to dist[u] + weight
29             Add (dist[v], v) to pq
30             Update trace[v] to u
31
32     Initialize path as an empty list
33     Set u to end_index
34     While u is not -1:
35         Append grid_cells[u] to path
36         Update u to trace[u]
37
38     If visualize is True:
39         Return [visual, reverse(path)]
40     Else:
41         Return reverse(path)

```

Ý tưởng của thuật toán Dijkstra trong giải mê cung

- Gọi ô xuất phát là start và ô kết thúc là end.
- Gọi $\text{dist}[u]$ là khoảng cách ngắn nhất từ ô start đến ô u trong mê cung.
- Gọi $\text{trace}[u]$ là ô liền trước u trong đường đi ngắn nhất từ start đến u .
- Gọi pq là một hàng đợi ưu tiên chứa các cặp (distance, cell), với distance là độ dài đường đi ngắn nhất từ start đến cell, hàng đợi ưu tiên độ dài nhỏ nhất.
- Gọi neighbors là một danh sách chứa các ô nối liền với ô đang xét trong mê cung, đường đi trực tiếp từ ô đang xét đến các ô trong neighbors có trọng số bằng nhau và bằng 1.
- Khởi tạo $\text{dist}[u] = \text{INF}$ với mọi ô u thuộc mê cung, INF là một số dương rất lớn.
- Gán $\text{dist}[\text{start_index}] = 0$, với start_index là chỉ số của ô start trong mảng một chiều tương ứng của mê cung, sau đó đưa cặp $(0, \text{start})$ vào hàng đợi ưu tiên.
- Với mỗi cặp (distance, cell) lấy ra từ hàng đợi ưu tiên:
 - Duyệt mọi ô v trong neighbors nối liền với ô đang xét, giả sử ô đang xét là u , nếu $\text{dist}[v] > \text{dist}[u] + 1$ thì cập nhật $\text{dist}[v] = \text{dist}[u] + 1$, đưa cặp $(\text{dist}[v], v)$ vào hàng đợi ưu tiên, sau đó cập nhật $\text{trace}[v] = u$.
 - Thực hiện đến khi ô đang xét trùng với end hoặc hàng đợi ưu tiên rỗng (là trường hợp end là điểm cuối mê cung).
- Với giải thuật trên, toàn bộ các ô có khả năng dẫn đến vị trí mục tiêu end trên mê cung đều được duyệt qua và lưu lại đường đi ngắn nhất. Sau đó, đường đi ngắn nhất từ start đến end được cập nhật thông qua việc truy vết mảng trace .

Phân tích và đánh giá độ phức tạp thuật toán

- Gọi $N \times N$ là kích thước của mê cung.
- Vòng lặp chính của thuật toán là while pq is not empty, do đó trong trường hợp xấu nhất có độ phức tạp $O(N^2)$.
- Bên trong vòng lặp chính, thao tác lấy phần tử có độ ưu tiên cao nhất của hàng đợi có độ phức tạp $O(\log(N^2))$.
- Vòng lặp con duyệt neighbors có tối đa 4 lần lặp (do ứng với mỗi ô sẽ có tối đa 4 ô liền kề có thể đi qua).
- Bên trong vòng lặp con, thao tác thêm $(\text{dist}[v], v)$ vào hàng đợi ưu tiên có độ phức tạp $O(\log(N^2))$.
- Suy ra, độ phức tạp thời gian của thuật toán là:

$$O(N^2)(O(\log(N^2)) + 4O(\log(N^2))) = O(10N^2 \log N) = O(N^2 \log N)$$

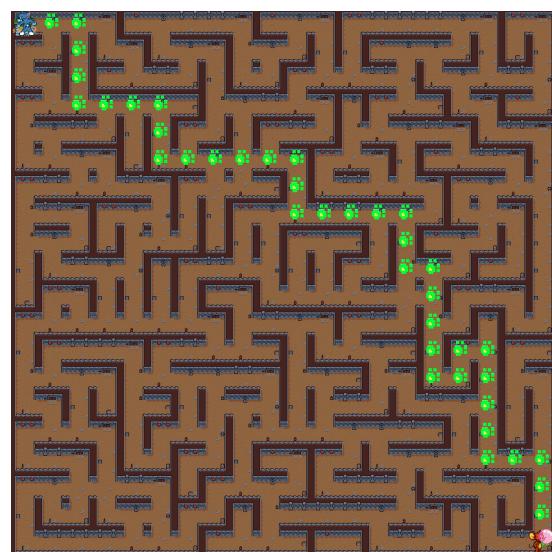
- **Độ phức tạp không gian:** Các mảng trace, dist và hàng đợi ưu tiên có độ phức tạp $O(N^2)$. Do đó độ phức tạp không gian của thuật toán là $O(N^2)$.
- Như vậy, với kích cỡ mê cung tối đa là 100×100 , thuật toán Dijkstra hoạt động tương đối nhanh và hiệu quả cao. Tuy nhiên, việc sử dụng thuật toán Dijkstra để giải mê cung chưa phải là phương án tối ưu vì trọng số giữa các ô liền kề trên mê cung đều bằng nhau, do đó không phát huy được hết thế mạnh của thuật toán.

4.11.5 So sánh 4 thuật toán tìm đường đi

Độ dài đường đi của mỗi thuật toán



Hình 4.1: Đường đi tìm được bởi thuật toán DFS



Hình 4.2: Đường đi tìm được bởi thuật toán BFS



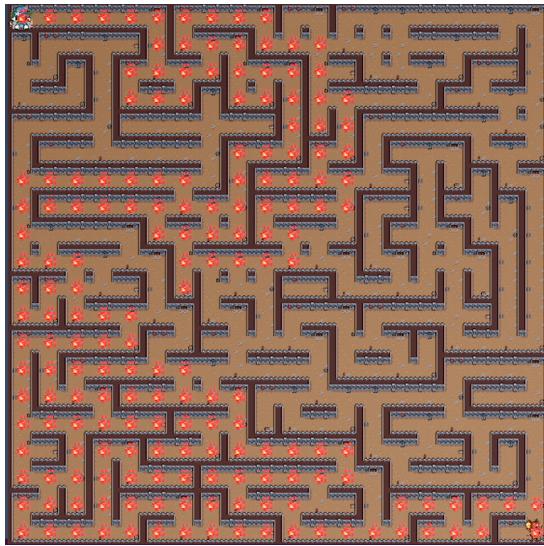
Hình 4.3: Đường đi tìm được bởi thuật toán A*



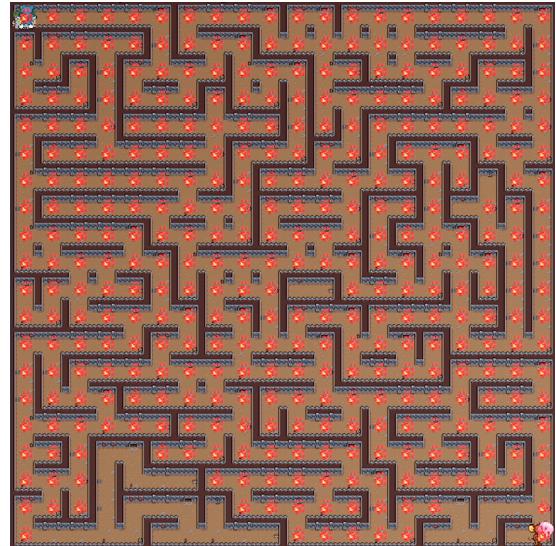
Hình 4.4: Đường đi tìm được bởi thuật toán Dijktra

- Ta có thể thấy, so về đường đi ngắn nhất, thuật toán DFS kém hiệu quả hơn so với 3 thuật toán còn lại bởi chỉ đi theo ngẫu nhiên một hướng cho đến khi gặp ngõ cụt hoặc khi xung quanh chỉ toàn là ô đã thăm rồi. Do đó, mặc dù rằng nó có thể tìm được đường đi về đích, tuy nhiên có thể đường đi đó không hiệu quả vì hướng di chuyển là ngẫu nhiên.
- Ở thuật toán BFS, để đến được ô end, thuật toán sẽ loang dần ra mọi phía từ ô start chứ không đi theo một hướng nhất định, vì vậy, ngay khi gặp được ô end (Đường nào cho ra ô end sớm nhất là đường ngắn nhất), vì vậy thuật toán sẽ tối ưu vì không phải chọn theo hướng ngẫu nhiên.
- Ở thuật toán A*, nhờ việc tính toán chi phí, do đó thuật toán sẽ định hướng được cần đi hướng nào. Mỗi ô khi lấy ra khỏi hàng đợi ưu tiên đều là ô được kì vọng bằng hàm tính toán chi phí là ô nằm trên đường đi tối ưu nhất. Nếu một hướng mà dẫn đến ô tiếp theo đó có chi phí không tối ưu nhất chắc chắn sẽ bị thay đổi sang hướng khác, vì vậy hướng đi theo thuật toán A* là ngắn nhất
- Ở thuật toán Dijkstra, đường đi ngắn nhất từ ô bắt đầu (start) đến ô kết thúc (end) được tìm thông qua quá trình liên tục cập nhật khoảng cách ngắn nhất từ ô bắt đầu đến các ô khác và sử dụng hàng đợi ưu tiên để luôn chọn ô có khoảng cách ngắn nhất hiện tại để mở rộng. Vì vậy, tại mỗi ô có khả năng đến được end đều được lưu lại đường đi ngắn nhất. Sau khi hàng đợi ưu tiên trống hoặc ô kết thúc được lấy ra từ hàng đợi, đường đi ngắn nhất từ ô bắt đầu đến ô kết thúc được tìm thấy qua việc truy vết mảng trace.

So sánh số ô cần qua để tìm được đường đi

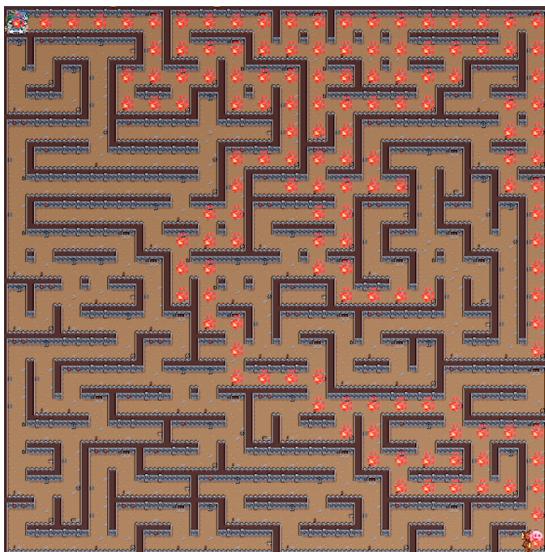


Hình 4.5: Số ô đi qua bởi thuật toán DFS

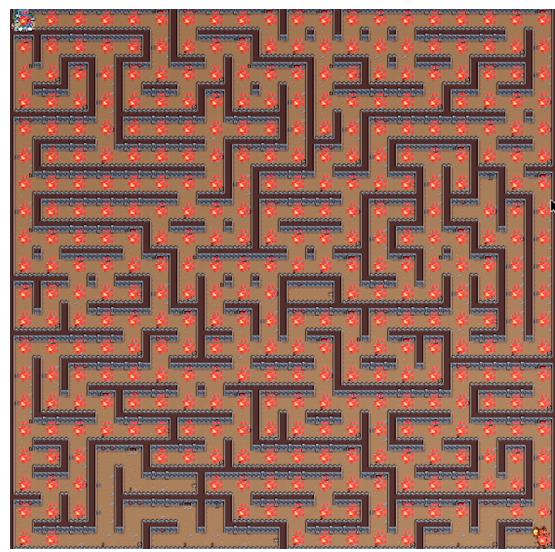


Hình 4.6: Số ô đi qua bởi thuật toán BFS

- Tuy thuật toán DFS không tìm được đường đi ngắn nhất, nhưng số ô đi qua bởi nó ít hơn đối với thuật toán BFS, đặc biệt là trong trường hợp khoảng cách của hai ô xa nhau, bởi tính chất quay lui và khi tìm được ô đích thì nó dừng thuật toán. Còn thuật toán BFS sẽ loang dần khắp các đỉnh kề của ô đang xét cho tới khi ghi gặp ô đích, vì vậy nó sẽ vét cạn cho đến khi nào tìm được ô đích và điều này cũng
- Tương tự với thuật toán Dijkstra bởi thuật toán Dijkstra trong mê cung không hiệu quả do trọng số của các đỉnh kề nhau là bằng nhau. Vì vậy, quá trình chạy thuật toán cũng tương tự việc vét cạn trong thuật toán BFS.
- Tuy nhiên, thuật toán A* lại đáp ứng việc tối ưu khía cạnh này bởi nó có chút tương đồng với thuật toán DFS vì đi theo hướng mà hàm tính toán cho là tối ưu nhất chứ không đơn thuần lan rộng như thuật toán BFS hay Dijkstra (Dijkstra trong trường hợp này cũng là loang dần như BFS bởi các trọng số trên đồ thị đều bằng nhau).



Hình 4.7: Số ô đi qua bởi thuật toán A*



Hình 4.8: Số ô đi qua bởi thuật toán Dijktrsa

Độ phức tạp về thời gian

- Như đã trình bày ở trên, độ phức tạp cho thuật toán DFS và BFS là $O(V+E)$, còn với A* và Dijkstra là $O(V\log(E))$. Khi đó về mặt trực quan thì BFS và DFS sẽ chạy nhanh hơn A* và Dijkstra.

4.12 Gợi ý Đường đi

4.12.1 Gợi ý đường đi tại vị trí bắt kì

Ý tưởng

- Do các hàm thuật toán đã trả về path của ô bắt đầu đến ô kết thúc
- Vậy mỗi lần gọi ý đường đi chúng ta chỉ cần thay vị trí ô bắt đầu là vị trí hiện tại của người chơi còn vị trí điểm đến và vị trí của

Đưa Vô Game

- Dầu tiên mặc định là mỗi khi người chơi bấm nút space thì thực hiện gọi hàm gợi ý đường đi
- Tiếp theo xác định thuật toán bằng if else . Sau đó nếu có đường thì ta sẽ đặt cờ flagpath bằng False .
- Tiếp theo ta cho vô vòng lặp duyệt qua các cell trong path mà thuật toán đã trả về .Tiếp tục cho một vòng for lấy sự kiện trong pygame.Nếu người chơi bấm space 1 lần ngược thì gợi ý đường đi tự động tắt. Cập nhật lại trạng thái của toàn bộ ô của gợi ý đường đi để nó không tô màu ô đó nữa .
- Lưu ý trong vòng for này chúng ta phải liên tục cập nhật lại map hay animation của người chơi thì mọi hình vẽ không bị đè lên nhau(do xử lý animation)

```

1 If the SPACE key is pressed:
2     If the position list is empty:
3         If Mode_Findpath is 0:
4             Find path using DFS algorithm
5         Else if Mode_Findpath is 1:
6             Find path using BFS algorithm
7         Else if Mode_Findpath is 2:
8             Find path using A* algorithm

```

```

9     Else if Mode_Findpath is 3:
10        Find path using Dijkstra's algorithm
11    If a path is found:
12      Set flagpath to False
13      For each cell in the path:
14        If flagpath is True, exit loop
15        Draw the maze
16        Mark the cell as part of the path
17      For each event1 in pygame events:
18        If QUIT event, quit game
19        If KEYDOWN event and SPACE key is pressed:
20          Reset path display and append target_cell to position list
21          Redraw game elements on screen
22          Set flagpath to True
23        If MOUSEBUTTONDOWN event:
24          Simulate pressing SPACE key
25          Update animations and draw game elements
26        Redraw game elements on screen
27        Append target_cell to position list
28        Redraw game elements on screen
29        Delay for smooth rendering
30
31    If the last position in the position list is the same as target_cell:
32      If a path exists:
33        Reset path display
34        Clear the position list
35
36    If neither condition is met:
37      If a path exists:
38        Reset path display
39        Clear the position list

```

Listing 4.22: Mã giả cho đoạn code xử lý gọi ý đường đi tại vị trí bất kỳ

4.12.2 Visualize

Bằng list visualize chứa tuần tự các ô được "ghé thăm" trong thuật toán, kết hợp với đoạn code sau ta có thể trực quan hóa quá trình chạy thuật toán nhờ

```

1  Allow = False
2  IF event.key == pygame.K_w:
3  IF Game_mode == "Bot" OR Game_mode == "Both":
4    IF Mode_Findpath == 0:
5      (route, path) = DFS_findPath(grid_cells, target_cell, end, cols, rows, True)
6    ELSE IF Mode_Findpath == 1:
7      (route, path) = BFS_findPath(grid_cells, target_cell, end, cols, rows, True)
8    ELSE IF Mode_Findpath == 2:
9      (route, path) = AStar_findPath(grid_cells, target_cell, end, cols, rows, True)
10   )
11   ELSE IF Mode_Findpath == 3:
12     (route, path) = Dijkstra_FindPath(grid_cells, target_cell, end, cols, rows,
True)
13
14   Mode_game_bot = 2
15   tmp = []
16
17   IF path IS NOT EMPTY:
18     IF route IS NOT EMPTY:
19       FOR rou IN route:
20         rou.seen = True
21         tmp.append(rou)
22         rou.draw(sc, TILE)
23         pygame.display.flip()
24         pygame.display.update()
25         playSound(4)
26
27       FOR events IN pygame.event.get():
28         IF events.type == pygame.KEYDOWN AND events.key == pygame.K_w:
              Mode_game_bot = 1

```

```

29             Allow = False
30         IF Mode_game_bot == 1:
31             FOR cells IN grid_cells:
32                 cells.path = False
33                 cells.seen = False
34             BREAK
35
36         IF len(route) == len(tmp):
37             Allow = True
38             FOR vertex IN tmp:
39                 vertex.seen = False
40             FOR vertex IN path:
41                 vertex.path = True
42
43         IF Allow:
44             FOR cell IN path:
45                 FOR events IN pygame.event.get():
46                     IF events.type == pygame.KEYDOWN AND events.key == pygame.K_w
47                         :
48                         Mode_game_bot = 1
49                         Allow = False
50                         BREAK
51
52         IF Mode_game_bot == 1:
53             FOR cells IN grid_cells:
54                 cells.path = False
55                 cells.seen = False
56             BREAK
57
58         IF (cell.x - player.x, cell.y - player.y) == (0, 1):
59             player.move('down', grid_cells, cols, rows)
60         ELSE IF (cell.x - player.x, cell.y - player.y) == (0, -1):
61             player.move('up', grid_cells, cols, rows)
62         ELSE IF (cell.x - player.x, cell.y - player.y) == (1, 0):
63             player.move('right', grid_cells, cols, rows)
64         ELSE IF (cell.x - player.x, cell.y - player.y) == (-1, 0):
65             player.move('left', grid_cells, cols, rows)
66
67         player.animation.update()
68         step += 1
69         screen.blit(background, (0, 0))
70         drawer.draw()
71         [cell.draw(sc, TILE) FOR cell IN grid_cells]
72         start_real.draw(sc, TILE)
73         mouse.draw(sc, TILE)
74
75         draw_dashboard(step, optionsettings[0], optionsettings[1],
76         optionsettings[2])
77         player.draw(sc, TILE)
78         pygame.display.update()
79         playSound(4)
80
81     Allow = False

```

Listing 4.23: Mã giả cho đoạn code xử lý visualize thực toán trong hàm xử lý chính của trò chơi

- Ở đoạn code trên, mỗi khi nhấn phím W hay nhấn nút trên giao diện, hàm sẽ trả về một list tất cả các điểm di qua trong thuật toán và đường đi tìm được bởi thuật toán đó. Tiếp theo, sử dụng biến Allow như cờ để cho biết quá trình visualize đã xong chưa và cho phép bot di chuyển.
- Với mỗi cell trong list route (list chứa tất cả ô đã đi qua), ta tuân tự cho gán các thuộc tính cell.seen = True để vẽ ra vết di của ô đó. Đồng thời, thêm lại ô đó vào list khác. Như vậy, ta có thể vẽ lên truy vết và đồng thời khi thuật toán chạy xong, xóa các vết đó và vẽ lên màu khác cho các ô nằm trên đường đi được tìm ra.
- Sau quá trình visualize thuật toán, biến Allow được bật thành True, cho phép bot di chuyển.

4.13 Máy tự chơi

4.13.1 Xử lý sự kiện nhấn phím w

Khi người dùng nhấn phím w, đoạn mã thực hiện các bước sau:

- Kiểm tra Game_mode để xác định chế độ chơi là Bot hoặc Both.
- Dựa vào giá trị của Mode_Findpath, chọn thuật toán tìm đường:
 - DFS_findPath
 - BFS_findPath
 - AStar_findPath
 - Dijkstra_FindPath
- Đặt Mode_game_bot thành 2, đánh dấu Bot đang tìm đường.

4.13.2 Xử lý kết quả tìm đường

Nếu tìm thấy đường đi (path không rỗng), đoạn mã sẽ:

- Lặp qua từng điểm trong route và đánh dấu chúng đã được thấy (seen).
- Vẽ lại các điểm trên màn hình.
- Kiểm tra sự kiện để cho phép người dùng dừng quá trình di chuyển của Bot.

4.13.3 Cập nhật trạng thái và hiển thị

Nếu Mode_game_bot chuyển thành 1 (điều khiển Bot), đoạn mã sẽ:

- Đặt lại trạng thái các ô (cells) trong lưới (grid_cells).
- Nếu tất cả các điểm trong route đã được thấy, cho phép Bot di chuyển theo path.

4.13.4 Di chuyển Bot theo đường đi

Đoạn mã sẽ lặp qua từng ô trong path và di chuyển Bot:

- Di chuyển Bot theo hướng thích hợp (lên, xuống, trái, phải) dựa trên sự khác biệt về tọa độ giữa Bot và ô tiếp theo.
- Cập nhật hoạt ảnh của Bot và số bước di chuyển (step).
- Vẽ lại toàn bộ màn hình, bao gồm:
 - Nền
 - Mê cung
 - Điểm bắt đầu (start_real)
 - Con trỏ chuột (mouse)
 - Bảng điều khiển (dashboard)
 - Bot
- Cập nhật thời gian đã trôi qua (elapsed_time) và hiển thị số bước di chuyển và thời gian.
- Đồng bộ hóa tốc độ khung hình với clock.tick(120).

4.13.5 Kết luận

Đoạn mã này minh họa cách triển khai các thuật toán tìm đường để điều khiển Bot trong trò chơi sử dụng Pygame. Các thuật toán tìm đường như DFS, BFS, A*, và Dijkstra được áp dụng để tìm đường đi từ vị trí hiện tại của Bot đến đích. Đoạn mã cũng cho phép người dùng dừng quá trình di chuyển của Bot bất cứ lúc nào bằng cách nhấn phím w, đồng thời cập nhật trạng thái và hiển thị trên màn hình theo thời gian thực.

```
1 IF event.key == pygame.K_w THEN
2     IF Game_mode == "Bot" OR Game_mode == "Both" THEN
3         IF Mode_Findpath == 0 THEN
4             route, path = CALL DFS_findPath(grid_cells, target_cell, end, cols, rows,
5             True)
6             ELSE IF Mode_Findpath == 1 THEN
7                 route, path = CALL BFS_findPath(grid_cells, target_cell, end, cols, rows,
8                 True)
9                 ELSE IF Mode_Findpath == 2 THEN
10                     route, path = CALL AStar_findPath(grid_cells, target_cell, end, cols, rows,
11                     True)
12                     ELSE IF Mode_Findpath == 3 THEN
13                         route, path = CALL Dijkstra_FindPath(grid_cells, target_cell, end, cols, rows
14                         , True)
15
16             SET Mode_game_bot TO 2
17             SET tmp TO empty list
18
19             IF path IS NOT EMPTY THEN
20                 IF route IS NOT EMPTY THEN
21                     FOR each rou IN route DO
22                         SET rou.seen TO True
23                         ADD rou TO tmp
24                         CALL rou.draw(sc, TILE)
25                         CALL pygame.display.flip()
26                         CALL pygame.display.update()
27
28                     FOR each events IN pygame.event.get() DO
29                         IF events.type == pygame.KEYDOWN AND events.key == pygame.K_w
30                         THEN
31                             SET Mode_game_bot TO 1
32                             SET Allow TO False
33
34                             IF Mode_game_bot == 1 THEN
35                                 FOR each cells IN grid_cells DO
36                                     SET cells.path TO False
37                                     SET cells.seen TO False
38                                     BREAK
39
40                             IF length of route IS EQUAL TO length of tmp THEN
41                                 SET Allow TO True
42                                 FOR each vertex IN tmp DO
43                                     SET vertex.seen TO False
44                                     FOR each vertex IN path DO
45                                         SET vertex.path TO True
46
47                             IF Allow THEN
48                                 FOR each cell IN path DO
49                                     FOR each events IN pygame.event.get() DO
50                                         IF events.type == pygame.KEYDOWN AND events.key == pygame.K_w
51                                         THEN
52                                             SET Mode_game_bot TO 1
53                                             SET Allow TO False
54                                             BREAK
55
56                             IF Mode_game_bot == 1 THEN
57                                 FOR each cells IN grid_cells DO
58                                     SET cells.path TO False
59                                     SET cells.seen TO False
60                                     BREAK
61
62                             IF (cell.x - player.x, cell.y - player.y) == (0, 1) THEN
```

```

57             CALL player.move('down', grid_cells, cols, rows)
58     ELSE IF (cell.x - player.x, cell.y - player.y) == (0, -1) THEN
59         CALL player.move('up', grid_cells, cols, rows)
60     ELSE IF (cell.x - player.x, cell.y - player.y) == (1, 0) THEN
61         CALL player.move('right', grid_cells, cols, rows)
62     ELSE IF (cell.x - player.x, cell.y - player.y) == (-1, 0) THEN
63         CALL player.move('left', grid_cells, cols, rows)
64
65     CALL player.animation.update()
66     INCREMENT step BY 1
67     CALL screen.blit(background, (0, 0))
68     CALL drawer.draw()
69     FOR each cell IN grid_cells DO
70         CALL cell.draw(sc, TILE)
71     CALL start_real.draw(sc, TILE)
72     CALL mouse.draw(sc, TILE)
73     SET current_time TO CALL pygame.time.get_ticks()
74     SET elapsed_time TO current_time - start_time
75     CALL draw_dashboard(step, elapsed_time, optionsettings[0],
optionsettings[1], optionsettings[2])
76         CALL player.draw(sc, TILE)
77         CALL pygame.display.flip()
78         CALL pygame.display.update()
79         CALL clock.tick(120)
80
81     SET Allow TO False

```

Listing 4.24: Mã giả Class MazeLoading

4.14 Hàm reset game và hàm new game

4.14.1 Hàm reset game

```

1      function reset_game(grid_cells, TILE, drawer, start, end):
2          set global variable paused to False
3          set global variable COLS
4          set global variable ROWS
5          set global variable ChosingStartendMode
6          fill screen with Black color
7          background = load_image("ingame.png")
8          background = scale_image(background, WIDTH, HEIGHT)
9
10         blit_image(screen, background, 0, 0)
11         drawer = create_MazeDrawer(COLS, ROWS, grid_cells, True)
12         drawer.draw()
13
14         update_display()
15         flip_display()
16
17         for each cell in grid_cells:
18             cell.path = False
19
20     return grid_cells, drawer, start, end

```

- Hàm reset game có chức năng reset hết game về lại ban đầu, bao gồm vị trí của người chơi cũng như vẽ lại bản đồ mê cung. Sau đó hàm sẽ trả về mê cung và các điểm đầu cuối để set lại cho người dùng.

4.14.2 Hàm new game

```

1      def new_game(grid_cells, TILE, drawer):
2          global paused
3          paused = False
4          global COLS
5          global ROWS

```

```

6     global ChosingStartendMode
7     screen.fill(Black)
8     background = pygame.image.load("ingame.png")
9     background = pygame.transform.scale(background, (WIDTH, HEIGHT))
10
11    screen.blit(background, (0, 0))
12    grid_cells = generate_maze(COLS, ROWS, TILE)
13    drawer = MazeDrawer(COLS, ROWS, grid_cells, True)
14    drawer.draw()
15    #[cell.draw(sc, TILE) for cell in grid_cells]
16    pygame.display.update()
17    pygame.display.flip()
18    if ChosingStartendMode == 1:
19        start, end = select_start_end_manually(grid_cells, COLS, ROWS, TILE, sc)
20        path = DFS_spread(grid_cells, start, COLS, ROWS)
21        while not end in path:
22            start, end = Ask_Choose_Again(grid_cells, COLS, ROWS, TILE, sc, drawer,
background)
23            path = DFS_spread(grid_cells, start, COLS, ROWS)
24        else:
25            start, end = select_start_end(grid_cells, COLS, ROWS)
26
27        for cell in grid_cells:
28            cell.path = False
29        return grid_cells, drawer, start, end

```

- Hàm new game là hàm khởi tạo lại cả mê cung và cả điểm bắt đầu và điểm đích của game.
- Tùy theo mỗi chế độ, ta sẽ cho phép người dùng chọn lại điểm đầu và điểm cuối.
- Trong chế độ người chơi tự chọn điểm đầu cuối, ta sẽ xét xem liệu điểm đầu có đến được điểm cuối không bằng hàm DFS_spread được trình bày ở trên. Vòng while sẽ dừng khi nào điểm đầu và điểm cuối có thể gặp nhau.

4.15 Các biến toàn cục, thư viện và hàm xử lý chính

- Hàm main() là hàm chính của chương trình, nhận các tham số khởi đầu như kích thước ô vuông TILE, số bước di chuyển step, số cột và hàng của mê cung cols và rows, thời gian đã trôi qua elapsetime, danh sách các ô gridcells, vị trí của người chơi player, và vị trí bắt đầu và kết thúc của mê cung start và end.

4.15.1 Khởi tạo biến toàn cục và thư viện:

Biến toàn cục

Có một số biến toàn cục được sử dụng trong chương trình như Mode_Findpath, Mode_game_bot, Game_mode, paused, COLS, ROWS, ChosingStartendMode, và current_direction.

Khởi tạo và cài đặt màn hình

Chương trình khởi tạo cửa sổ Pygame và đồng hồ. Nếu không có kích thước ô vuông được chỉ định, thì nó được tính toán dựa trên kích thước của màn hình và số cột của mê cung. Một hình nền được tải và vẽ lên màn hình.

Tạo mê cung

- Nếu danh sách các ô grid chưa được tạo, thì một mê cung mới sẽ được tạo bằng cách sử dụng hàm generate_maze(). Một đối tượng MazeDrawer được tạo để vẽ mê cung lên màn hình.
- Sau khi đã tạo xong mê cung ta tiếp tục tạo các đối tượng khác như là người chơi Player, Mouse, Start. Đây là các đối tượng chính của game và người chơi sẽ liên tục tương tác với nó

Chọn vị trí bắt đầu và kết thúc

- Nếu vị trí bắt đầu và kết thúc của mê cung không được chỉ định, thì sẽ sử dụng ô bắt đầu tiên và cuối cùng của mê cung hoặc chọn bằng tay hoặc tự động tùy thuộc vào chế độ. Nếu đã được chỉ định, sẽ sử dụng chúng.
- Nếu vị trí được chọn tay nhưng nếu 2 điểm chọn của người chơi không được chọn thì sẽ gọi hàm `AskChooseAgain` để hiển thị màn hình cho người chơi chọn lại
- Sau khi cho người chơi chọn vị trí xong thì ta sẽ biểu thị các đối tượng bằng cách gán tọa độ do người chơi chọn cho các đối tượng và sử dụng phương thức `draw` để hiển thị người chơi

Vòng lặp chính

- Chương trình sẽ tiếp tục chạy trong một vòng lặp vô hạn, xử lý sự kiện của người dùng và cập nhật trạng thái của trò chơi. Nếu người chơi đến được ô kết thúc hoặc hết thời gian, trò chơi sẽ kết thúc.
- Ta sẽ lấy sự kiện bằng vòng `for event in pygame.event.get()`, ta lần lượt xem xét các sự kiện trong game lần lượt là `KeyDown` hoặc `MOUSEBUTTONDOWN`. Sau đó xem xét xử lý từng tương tác của người chơi đến game. Tương tự thì xử lý chuột sẽ tùy vào tọa độ của chuột mà người chơi khi click để thực hiện hoặc gọi hàm tương ứng
- Những tương tác quan trọng có thể nói tới là `savegame setting onoff sound` và các nút liên quan tới game hoặc thuật toán

```
1 Procedure (TILE, step, cols, rows, elapse_time, gridcells, player, start, end)
2 Declare global variables: ModeFindpath, Modegamebot, Gamemode, paused, COLS, ROWS,
3 ChosingStartendMode, currentdirection
4 Initialize variables
5 Set sound.inMenu to False and toggle sound if previously in menu
6 If TILE is None, set TILE based on Bottomrightx and COLS
7 Set up pygame display and clock
8 Fill screen and set background image
9 If cols and rows are None, set them to global COLS and ROWS
10 If gridcells is None, generate maze
11 Create MazeDrawer instance and draw maze
12 Display initial maze on screen
13
14 If start and end are None
15     Set default start and end cells
16     If ChosingStartendMode is 1
17         Select start and end manually
18         Start game clock
19         Draw dashboard
20         Spread path using DFS
21         While end not in path
22             Ask to choose start and end again
23             Spread path using DFS
24         Else if ChosingStartendMode is 2
25             Select start and end automatically
26             Start game clock
27             Create start and mouse objects
28             Draw start and mouse on screen with delay
29     Else
30         Set start and end based on given coordinates
31
32     If player is None, create player instance
33     Else, create player instance based on given coordinates
34     Create start and mouse objects
35     Draw initial maze, player, and mouse on screen
36     Initialize variables for path, Allow, time, and step
37     Set starttime using pygame
38
39     While True
40         Update player index and target_cell
41         If target_cell is end
```

```

41     Play win sound
42     Get elapsed time
43     Get rank and display winning sequence
44     Update screen during animation
45     Show winning screen and check for option_after_game
46     If option_after_game, reset game
47     Else, exit to menu
48
49     Check if game time exceeded 15 minutes
50         If true, play game over sound and show game over screen
51         If option_after_game, reset game
52         Else, exit to menu
53
54     Handle pygame events
55         If QUIT event, exit game
56         If USEREVENT, decrement time
57         If KEYDOWN event
58             If K_x, toggle sound
59             If K_e, open settings menu and restart
60             If Game_mode is Player or Both, handle arrow keys for player movement
61             If K_w, handle bot mode pathfinding
62                 Set route and path based on Mode_Findpath
63                 Set Mode_game_bot to 2
64                 If path is found, draw route and path on screen
65                 Handle in-game events during path drawing
66                 If path complete, allow movement and update player position
67                 If K_q, cycle Mode_Findpath and update settings
68                 If K_SPACE, handle manual pathfinding
69                     Toggle path display and reset path if necessary
70                 Handle pause, reset, continue, and save keys
71
72         If MOUSEBUTTONDOWN event
73             If paused, show pause menu and handle button clicks
74             Else, handle game interactions based on mouse position
75
76     If paused, show pause menu and continue loop
77
78     Handle player movement based on current direction
79     Update dashboard and draw all game elements on screen
80     Update animations and delay for smooth rendering
81
82 End Procedure

```

Database

5.1 Tổ Chức Database

5.1.1 file database.json

- Đây là file dùng để lưu trữ các thông tin của người dùng. Được tổ chức như sau bằng file json.
- Tài khoản của người dùng được lưu trữ trong một list, và mỗi tài khoản là một dictionary độc lập với nhau .
- Key đầu tiên của dictionary là tên đăng nhập của người dùng, với value là mật khẩu người dùng. Sau tên đăng nhập là các bộ file người dùng. Mỗi khi người chơi nhập file để lưu, hệ thống sẽ mã hóa lại tên file bằng cách gắn vào đuôi TAIL nhằm khiến cho tên file độc lập với tên đăng nhập, tránh sai sót khi lưu file và load file. Mỗi value của key tên file là một list các chuỗi chứa các tên file của file thành phần như thông số mê cung, vị trí người chơi, vị trí bắt đầu và kết thúc, thời gian tính tới hiện tại và số bước đã đi.



```
[{"xyz": {"abcd_name_of_file_for_saving_and_loading": ["abcd_maze.csv", "abcd_player.csv", "abcd_start_end.csv", "abcd_steps_time.csv"], "15_name_of_file_for_saving_and_loading": ["15_maze.csv", "15_player.csv", "15_start_end.csv", "15_steps_time.csv"]}, "abc_name_of_file_for_saving_and_loading": [{"abc_maze.csv", "abc_player.csv", "abc_start_end.csv", "abc_steps_time.csv"}], "abcde_name_of_file_for_saving_and_loading": [{"abcde_maze.csv", "abcde_player.csv", "abcde_start_end.csv", "abcde_steps_time.csv"}], "xyz_name_of_file_for_saving_and_loading": [{"xyz_maze.csv", "xyz_player.csv", "xyz_start_end.csv", "xyz_steps_time.csv"}], "12_name_of_file_for_saving_and_loading": [{"12_maze.csv", "12_player.csv", "12_start_end.csv", "12_steps_time.csv"}]}]
```

Hình 5.1: Minh họa cách tổ chức file database.json

5.1.2 file ranking.json

- File ranking được tổ chức là một list chứa 3 dictionary cho 3 mức độ chơi. Nếu người chơi nào phá kỉ lục trước đó của bản thân sẽ được ghi nhận lại vào file ranking này.

```

1   [
2     {
3       "xyz": [
4         1155,
5         1
6       ],
7       "duyminh": [
8         1929,
9         3
10      ],
11      "b": [
12        5131,
13        1
14      ],
15      "a": [
16        10428,
17        11
18      ]
19    },
20    {
21      "xyz": [
22        23016,
23        12
24      ],
25      "mylinh": [
26        1534979,
27        45
28      ],
29    },
30    {
31      "duyminh": [
32        7214,
33        17
34      ],
35      "xyz": [
36        20718,
37        24
38      ],
39      "a": [
40        148550,
41        138
42      ]
43    }
44 ]

```

Hình 5.2: Minh họa cách tổ chức file ranking.json

5.2 Chức Năng Save_Game

Các thông số liên quan đến trò chơi sẽ được lưu hết bằng file csv.

5.2.1 Các hàm lưu thông số của trò chơi

```

1   function save_maze_to_csv(grid_cells, cols, rows, filename):
2     open file with name 'filename' in write mode
3
4     create a CSV writer object
5
6     // Write maze dimensions
7     write [cols, rows] to the file
8
9     // Write maze cells
10    for each cell in grid_cells:
11      write the following attributes of cell to the file:
12        - cell.x
13        - cell.y
14        - cell.walls['top'] (converted to integer)
15        - cell.walls['right'] (converted to integer)
16        - cell.walls['bottom'] (converted to integer)
17        - cell.walls['left'] (converted to integer)
18        - cell.visited (converted to integer)
19        - cell.passed (converted to integer)
20        - cell.path (converted to integer)
21
22    close the file

```

Listing 5.1: Hàm lưu mē cung

```
1     function save_player_position_to_csv(player, filename):
2         open file with name 'filename' in write mode
3
4         create a CSV writer object
5
6         write player's position to the file:
7             - player.x
8             - player.y
9
10        close the file
```

Listing 5.2: Mã giả cho hàm lưu vị trí người chơi

```
1    unction save_end_start_to_csv(start, end, filename):
2         open file with name 'filename' in write mode
3
4         create a CSV writer object
5
6         write start and end positions to the file:
7             - start.x
8             - start.y
9             - end.x
10            - end.y
11
12         close the file
```

Listing 5.3: Mã giả cho hàm lưu vị trí điểm xuất phát và điểm đích

```
1     function save_time_and_steps_to_csv(steps, elapsed_time, filename):
2         open file with name 'filename' in write mode
3
4         create a CSV writer object
5
6         write elapsed time and steps to the file:
7             - elapsed_time
8             - steps
9
10        close the file
```

Listing 5.4: Mã giả cho hàm lưu tổng thời gian chơi và số bước đi

- Với mỗi ô trong mẻ cung, ta sẽ lưu lại tọa độ, các bức tường của nó, cũng như các thuộc tính phụ như cell.visited, cell.passed, cell.path.
 - Hàm thứ hai giúp lưu lại vị trí hiện tại của người chơi bằng cách lưu lại tọa độ x và y.
 - Ta cũng lưu lại vị trí xuất phát và vị trí đích, lưu lại vị trí xuất phát giúp người chơi có thể reset game nếu muốn.
 - Lưu lại thời gian và số bước đi nhằm cung cấp các thông tin phụ cho người chơi và vẽ lại dashboard.

5.2.2 Hàm lưu trò chơi tổng hợp

```
1 function save_game(file_name, username, grid_cells, cols, rows, player, start, end,
2 steps, elapsed_time):
3     tail1 = '_maze.csv'
4     tail2 = '_player.csv'
5     tail3 = '_start_end.csv'
6     tail4 = '_steps_time.csv'
7
8     file_maze = concatenate(file_name, tail1) // file for saving maze
9     file_player = concatenate(file_name, tail2) // file for saving player's position
10    file_start_end = concatenate(file_name, tail3) // file for saving start and end
11    points
12    file_steps_time = concatenate(file_name, tail4) // file for saving steps and time
13    file_name = concatenate(file_name, TAIL) // append TAIL to file_name to distinguish
14    it from username key
```

```

13 // Save maze data to CSV
14 save_maze_to_csv(grid_cells, cols, rows, file_maze)
15
16 // Save player position to CSV
17 save_player_position_to_csv(player, file_player)
18
19 // Save start and end points to CSV
20 save_end_start_to_csv(start, end, file_start_end)
21
22 // Save steps and time to CSV
23 save_time_and_steps_to_csv(steps, elapsed_time, file_steps_time)
24
25 // Open 'database.json' in read mode
26 open file 'database.json' in read mode
27 read data from file into variable data
28 close the file
29
30 // Iterate through data to find the username
31 for each element in data:
32     if username exists in element:
33         // Add a new key for file_name with an empty list
34         element[file_name] = []
35         // Append the file paths to the list
36         element[file_name].append(file_maze) // element 0: maze file
37         element[file_name].append(file_player) // element 1: player position file
38         element[file_name].append(file_start_end) // element 2: start and end points
39         file
40             element[file_name].append(file_steps_time) // element 3: steps and time file
41
42 // Open 'database.json' in write mode
43 open file 'database.json' in write mode
44 write data back to the file in JSON format with indentation
45 close the file

```

Listing 5.5: Hàm lưu game tổng hợp

- Khi người dùng chọn chức năng lưu game và nhập tên file, hàm save_game được gọi và truyền vào các tham số cần thiết là tên file, tên đăng nhập, mê cung, số hàng, số cột của mê cung, class người chơi, điểm xuất phát và điểm đích, tổng thời gian chơi và số bước đã đi.
- Đầu tiên, ta sẽ định nghĩa các đuôi của các file nhỏ để định danh các file theo chức năng của nó, ví dụ đuôi tail1 là '_maze.csv' tên file dùng để lưu mê cung. Sau đó, ta tạo tên của các file nhỏ này bằng việc gắn file_name người dùng truyền vào với đuôi.
- Để phân biệt giữa key user_name và key tên file lưu trò chơi, ta sẽ gắn file_name với đuôi TAIL đã được định nghĩa trước đó là '_name_of_file_for_saving_and_loading'
- Sau đó lần lượt gọi các hàm lưu đã được định nghĩa ở trên để lưu lại thông số trò chơi
- Cuối cùng, ta tạo ra key file_name với value là một list các tên file lưu các thông số khác nhau. Ta cần duyệt hết database để tìm đến tài khoản có user name tương ứng và lưu trò chơi vào tài khoản đó.

5.3 Chức Năng load game

Chương trình thông qua các hàm load game để truy xuất thông tin trò chơi từ những file csv đã lưu.

5.3.1 Hàm load các file chứa các thông số trong trò chơi

```

1 function load_maze_from_csv(filename):
2     Open the file named 'filename' in read mode
3
4     Initialize a list to store grid cells
5     Read the grid size (number of columns and rows) from the file
6

```

```

7     Read each line in the file:
8         - Parse the line to extract information of each grid cell (x, y, top, right,
bottom, left, visited, passed, path)
9         - Create a grid cell object with the corresponding information
10        - Append the grid cell to the list
11
12    Return the list of grid cells and the grid size

```

Listing 5.6: Mã giả cho hàm truy xuất mê cung

```

1 function load_player_position_from_csv(filename):
2     Open the file named 'filename' in read mode
3
4     Read the player's coordinates from the file
5
6     Return the player's coordinates

```

Listing 5.7: Mã giả cho hàm truy xuất vị trí người chơi

```

1 function load_start_end_from_csv(filename):
2     Open the file named 'filename' in read mode
3
4     Read the coordinates of the start and end points from the file
5
6     Return the coordinates of the start and end points

```

Listing 5.8: Mã giả cho hàm truy xuất vị trí điểm xuất phát và điểm đích

```

1 function load_time_and_steps_from_csv(filename):
2     Open the file named 'filename' in read mode
3
4     Read the elapsed time and number of steps from the file
5
6     Return the elapsed time and number of steps

```

Listing 5.9: Mã giả cho hàm truy xuất tổng thời gian chơi và số bước đi

- Với mỗi ô trong mê cung, ta sẽ truy xuất lại tọa độ, các bức tường của nó, cũng như các thuộc tính phụ như cell.visited, cell.passed, cell.path.
- Hàm thứ hai giúp truy xuất lại vị trí hiện tại của người chơi bằng cách truy xuất lại tọa độ x và y.
- Ta cũng truy xuất lại vị trí xuất phát và vị trí đích, truy xuất lại vị trí xuất phát giúp người chơi có thể reset game nếu muốn.
- Truy xuất lại thời gian và số bước đi nhằm cung cấp các thông tin phụ cho người chơi và vẽ lại dashboard.

5.3.2 Hàm load tổng hợp

```

1 function load_game(file_name):
2     Append 'TAIL' to the file_name
3
4     Open the 'database.json' file in read mode
5     Load data from the JSON file
6
7     Loop through each account in the data:
8         If the login_forms.User_name exists in the account:
9             If the account has more than one file associated with it:
10                 If the file_name exists in the account:
11                     Retrieve file paths for grid, player position, start-end points,
12                     elapsed time, and steps
13                     Load maze grid cells from CSV file using 'load_maze_from_csv'
14                     function
15                         Load player position from CSV file using ,
16                         'load_player_position_from_csv' function
17                         Load start and end points from CSV file using ,
18                         'load_start_end_from_csv' function

```

```

15         Load elapsed time and steps from CSV file using '
16     load_time_and_steps_from_csv' function
17         Return grid cells, grid dimensions, player position, start and end
18     points, steps, and elapsed time

```

Listing 5.10: Hàm truy xuất game tổng hợp

- Khi người dùng chọn chức năng load game và chọn maze đã lưu thông qua hình ảnh tinh gọn trạng thái của maze đó cùng với tên file, hàm load_game được gọi và truyền vào tham số duy nhất là tên file.
- Đầu tiên, chúng ta sẽ thêm đuôi TAIL vào tên file để xác định loại file cần tải.
- Tiếp theo, mở file JSON chứa thông tin game và tải dữ liệu vào biến data. Sau đó, ta duyệt qua từng tài khoản trong data. Nếu tên đăng nhập của tài khoản tương ứng với người chơi, và tài khoản đó có nhiều hơn một file liên quan, ta kiểm tra xem tên file cần tải có trong danh sách các file của tài khoản đó không.
- Nếu có, ta trích xuất đường dẫn của các file mê cung, vị trí người chơi, điểm xuất phát và điểm đích, thời gian đã trôi qua và số bước đã đi. Sau đó, ta sử dụng các hàm tải dữ liệu từ file CSV đã được định nghĩa trước để tải thông tin từ các file này.
- Cuối cùng, ta trả về các thông tin về mê cung, kích thước mê cung, vị trí người chơi, điểm xuất phát và điểm đích, số bước đã đi và thời gian đã trôi qua. Nếu không tìm thấy tài khoản hoặc file, ta trả về False để thông báo là không tìm thấy trò chơi đã lưu.

5.3.3 Hàm xử lý sau khi load file

```

1  function continue_game(file_name):
2      res = load_game(file_name)
3      if res is not null:
4          grid_cells, cols, rows, player, start, end, step, elapsed_time = res
5
6          TILE = Bottom_right_x divided by cols
7
8          (TILE, step, cols, rows, elapsed_time, grid_cells, player, start, end)
9      else:
10         return False

```

- Ta tận dụng hàm để làm điều này. Hàm được truyền các tham số mặc định def (TILE = None, step = None, cols = None, rows = None, elapsed_time = None, grid_cells = None, player = None, start = None, end = None)
- Khi ta không truyền các tham số này vào, hàm được hiểu là tạo ra trò chơi mới. Nhưng khi ta truyền các tham số này vào, hàm sẽ tạo lại trò chơi theo các thông số đó.
- Hàm continue_game sẽ gọi hàm load_file tổng hợp để tải lên các thông số, nếu tên file hợp lệ, hàm trả về một tuple chứa các thông số và ta truyền các thông số này cho hàm là trò chơi sẽ được load lên. Ngược lại hàm sẽ trả về False nếu file không hợp lệ.

Giao diện

6.1 Design Giao Diện

Giao diện của game được nhóm tự design qua công cụ Canva và phần mềm vẽ pixel Aseprite.



Hình 6.1: Giao diện Menu Game



Hình 6.2: Giao diện Setting



(a) Chế độ Player



(b) Chế độ Bot

Hình 6.3: Giao diện In Game



Hình 6.4: Màn hình Load Game



Hình 6.5: Màn hình Leader Board



NAME	ID	ROLE
BANG MY LINH	♀ 28122009	GAME DESIGNER
DAO SY DUY MINH	♂ 28122041	FULL-STACK DEV
NGUYEN LAM PHU QUY	♂ 28122048	TECHNICAL DEV
NGUYEN TRAN TRUNG KIEN	♂ 28122038	GAME/SOUND DEV

BY AI SALVE

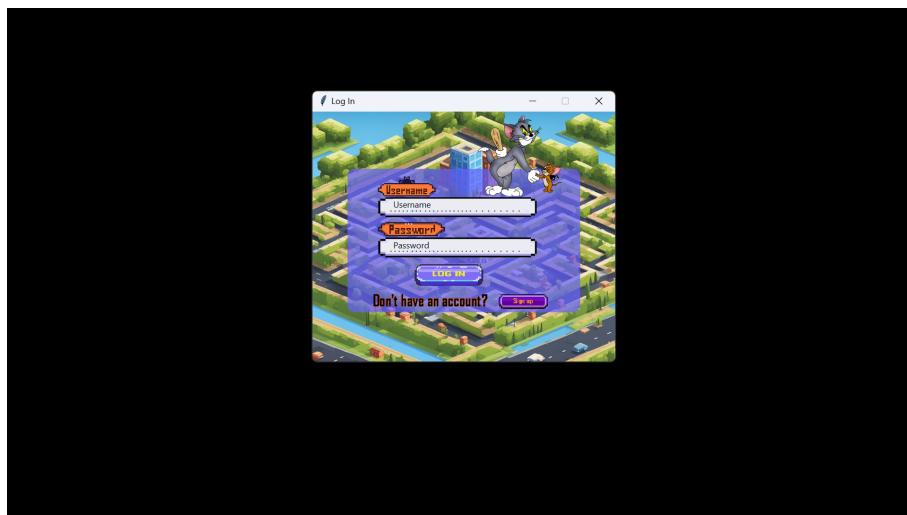
Hình 6.6: Màn hình About



Hình 6.7: Màn hình Guide - Hướng dẫn cách chơi

6.2 Login Form

- Login Form của game được tạo bằng thư viện Tkinter.
- Thư viện Tkinter là một trong những thư viện GUI (Giao diện người dùng đồ họa) phổ biến nhất cho Python. Tkinter cung cấp nhiều widget (như button, label, entry, canvas,...) để xây dựng các giao diện người dùng phong phú và trực quan. Cho phép người dùng tùy chỉnh các thuộc tính của widget như màu sắc, phông chữ, kích thước,... giúp tạo ra giao diện đẹp mắt và phù hợp với nhu cầu của ứng dụng.



Hình 6.8: Giao diện login form

6.2.1 Cửa sổ login

- Cửa sổ chính (root) được tạo bằng Tkinter với kích thước 640x528 với tiêu đề "Log In".
- Hàm confirm_close được thiết lập để xác nhận việc đóng cửa sổ khi người dùng nhấn nút đóng.
- Hàm load_image_login được gọi để tải các hình ảnh và biểu tượng cần thiết cho giao diện đăng nhập từ thư mục assets/frame0, bao gồm: hình ảnh nút đăng nhập, nút chuyển đến đăng ký và các hình ảnh trang trí.
- Tạo một Canvas kích thước 640x528 để chứa các thành phần giao diện như hình ảnh, các trường nhập liệu và các nút.
- Tạo các nút đặt tại các tọa độ cụ thể trên màn hình, sử dụng hình ảnh đã tải lên từ thư mục frame0
- Tạo các trường nhập liệu với kích thước và vị trí phù hợp, sử dụng hình nền đã được tải lên từ thư mục assets.
- Hàm sign_in được gọi khi người dùng nhấn nút đăng nhập.

```
1     function sign_in():
2         flag = false
3         username = get the value from the username entry field
4         password = get the value from the password entry field
5
6         if length of username > 15:
7             show error message "Username should not exceed 15 characters"
8             return
9
```

```

10     open 'database.json' file in read mode as data:
11         accounts = parse JSON data from the file
12
13     for each account in accounts:
14         if username exists in account and account[username] == password:
15             show info message "Sign in successfully"
16             set global User_name to username
17             flag = true
18             set global logged_in to true
19             close the login window (root.destroy())
20             break
21
22     if flag is still false:
23         show error message "The username or password is invalid, try again"

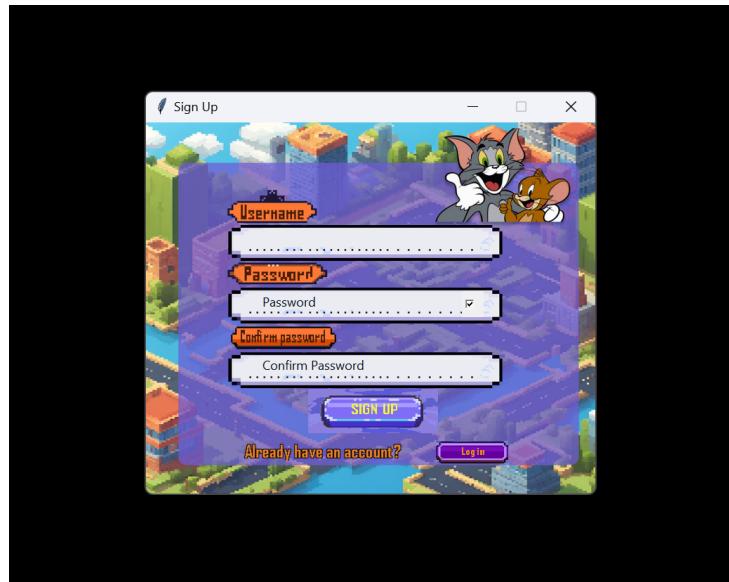
```

Listing 6.1: Mã giả của hàm sign_in

Cách thức hoạt động của hàm sign_in:

- Lấy tên người dùng và mật khẩu từ các trường nhập liệu (user và code). Kiểm tra độ dài của tên người dùng (không vượt quá 15 ký tự).
- Mở tệp database.json để đọc dữ liệu tài khoản đã lưu.
- Duyệt qua các tài khoản trong tệp để kiểm tra tên người dùng và mật khẩu. Nếu thông tin đăng nhập đúng, gửi message box thông báo đăng nhập thành công và đóng cửa sổ đăng nhập. Ngược lại, gửi message box báo lỗi.

6.2.2 Cửa sổ sign up



Hình 6.9: Cửa sổ sign up

- Cửa sổ sign up được tạo và trang trí tương tự cửa sổ login.
- Hàm `sign_up_command` là hàm xử lý quá trình đăng ký tài khoản vào game, được gọi khi người dùng nhấn nút sign up.

Các hàm trong sign_up_command

- Hàm kiểm tra độ mạnh mật khẩu (`check_password_strength`):

```

1     function check_password_strength(password):
2         if length(password) < 8:
3             return False, "Password must be at least 8 characters long"
4
5         if not any character in password is a digit:
6             return False, "Password must contain at least one digit"
7
8         if not any character in password is a lowercase letter:
9             return False, "Password must contain at least one lowercase
letter"
10
11        if not any character in password is an uppercase letter:
12            return False, "Password must contain at least one uppercase
letter"
13
14        return True, ""
15

```

Listing 6.2: Mã giả của hàm `check_password_strength`

1. Kiểm tra độ dài của mật khẩu, nếu dưới 8 ký tự, trả về False và thông báo lỗi.
2. Kiểm tra xem mật khẩu có chứa ít nhất một chữ số, một chữ cái thường và một chữ cái hoa không. Nếu không, trả về False và thông báo lỗi tương ứng.
3. Nếu tất cả các điều kiện đều được thỏa mãn, trả về True và chuỗi rỗng (không có thông báo lỗi).

- Hàm xử lý đăng ký `sign_up`

```

1  function signup():
2      username = get_value_from_input(user)
3      password = get_value_from_input(code)
4      confirm_password = get_value_from_input(confirm_pw)
5
6      strength = check_password_strength(password)
7      if strength != "Strong password.":
8          show_warning_message("Password Strength", strength)
9          bring_window_to_front(window)
10         return
11
12     if length(username) > 15:
13         show_error_message('Invalid', 'Username should not exceed 15
characters')
14         bring_window_to_front(window)
15         return
16
17     if password == confirm_password:
18
19         data = open_file('database.json', 'r')
20         accounts = parse_json(data)
21         close_file(data)
22
23         is_new_user = True
24
25         for each account in accounts:
26             if username exists in account:
27                 show_error_message('Invalid', 'This username has existed, try
again')
28                 bring_window_to_front(window)
29                 is_new_user = False
30                 break
31
32         if is_new_user:
33             new_account = create_new_account(username, password)
34             append_to_list(accounts, new_account)
35             show_info_message('Valid', 'Sign up successfully')
36
37             save_file = open_file('database.json', 'w')
38             save_json(accounts, save_file, indent=4)

```

```
39         close_file(save_file)
40
41         destroy_window(window)
42     else:
43         show_error_message('Invalid', 'Both passwords should match')
44         bring_window_to_front(window)
45
```

Listing 6.3: Mã giả của hàm xử lý đăng ký

1. Lấy giá trị user_name, password, confirm_password từ các trường nhập liệu.
 2. Kiểm tra độ dài của tên người dùng: nếu độ dài của user_name vượt quá 15 ký tự thì hiển thị thông báo lỗi và kết thúc hàm bằng return.
 3. Ngược lại, kiểm tra xem mật khẩu và xác nhận mật khẩu có khớp không: Nếu không khớp, hiển thị thông báo lỗi và kết thúc hàm bằng return. Nếu khớp, gọi hàm check_password_strength để kiểm tra độ mạnh mật khẩu.
 4. Tạo đối tượng người dùng mới: Tạo một dictionary new_user chứa user_name và password, sau đó thêm dữ liệu người dùng mới vào tệp database.json.
 5. Gửi message box thông báo đăng ký thành công và đóng cửa sổ đăng ký.
- Ngoài hai hàm xử lý chính trên, còn có hàm toggle_password_visibility chuyển đổi trường mật khẩu từ dạng văn bản sang sang dấu '*' và ngược lại.

6.3 Menu Game

Dây Có Thể nói là màn hình quan trọng nhất của game , vì đó là nơi kết nối giữa các màn hình , cũng như là nơi điều phối mọi hoạt động của game khi tương tác với người chơi.

MenuGame 5 phần chính bao gồm :

- Start Game
 - Đây là nơi người chơi sẽ vào màn hình game chính , bắt đầu trò chơi
- Settings
 - Như cái tên nó sẽ cho người chơi lựa chọn những cái đặt ,thay đổi các tính năng mặc định của game
- Ranking
 - Ranking thì sẽ có ranking cho từng chế độ chơi easy,medium,hard
- About
 - Giới Thiệu Thành Viên Nhóm
- Quit
 - Thoát Game

6.3.1 draw_menu(current_option)

Hàm này được sử dụng để vẽ menu trên màn hình game. Nó nhận vào một tham số là `current_option`, đại diện cho lựa chọn hiện tại của người dùng trong menu.

Các bước thực hiện trong hàm bao gồm:

1. Xác định chiều rộng (`screen_width`) và chiều cao (`screen_height`) của màn hình game.
2. Tải hình ảnh nền cho menu từ tệp 'menubackground.jpg', sau đó chuyển đổi kích thước của nó để phù hợp với kích thước màn hình.
3. Xóa màn hình bằng màu đen (`Black`) và vẽ hình ảnh nền của menu lên màn hình.
4. Thiết lập font chữ cho tiêu đề và các tùy chọn trong menu.
5. Vẽ tiêu đề "MAZE GAME" ở giữa màn hình.
6. Vẽ các tùy chọn trong menu, với màu xanh (`Green`) cho tùy chọn hiện tại và màu vàng (`Yellow`) cho các tùy chọn còn lại.

6.3.2 handle_menu_input(current_option)

Hàm này được sử dụng để xử lý đầu vào từ người dùng khi họ tương tác với menu. Nó nhận vào một tham số là `current_option`, đại diện cho lựa chọn hiện tại của người dùng trong menu.

Các bước thực hiện trong hàm bao gồm:

1. Lắp qua tất cả các sự kiện trong hàng đợi của pygame.
2. Nếu người dùng nhấn nút thoát (`pygame.QUIT`), thì thoát khỏi trò chơi.
3. Nếu người dùng nhấn một phím (`pygame.KEYDOWN`), kiểm tra xem phím nào đã được nhấn.
4. Nếu người dùng nhấn phím mũi tên lên (`pygame.K_UP`), cập nhật lựa chọn hiện tại lên tùy chọn trước đó trong menu.
5. Nếu người dùng nhấn phím mũi tên xuống (`pygame.K_DOWN`), cập nhật lựa chọn hiện tại xuống tùy chọn tiếp theo trong menu.
6. Nếu người dùng nhấn phím enter (`pygame.K_RETURN`), trả về lựa chọn hiện tại và một cờ để cho biết rằng người dùng đã chọn một tùy chọn trong menu.

6.3.3 xulymenu()

Hàm này là trung tâm của việc xử lý menu trong trò chơi. Nó không nhận tham số và hoạt động trong một vòng lặp vô hạn cho đến khi người dùng thoát khỏi menu hoặc chọn một tùy chọn cụ thể.

Các bước thực hiện trong hàm bao gồm:

- Thiết lập các biến `menu_option` và `in_menu` để theo dõi lựa chọn hiện tại và trạng thái của menu.
- Trong vòng lặp, lặp qua các sự kiện từ người dùng và vẽ menu lên màn hình.
- Nếu người dùng chọn một tùy chọn trong menu, kiểm tra tùy chọn đó và thực hiện các hành động tương ứng, chẳng hạn như thoát khỏi menu hoặc chuyển đến cài đặt.
- Nếu người dùng chọn "Start Game", thì thoát khỏi menu và bắt đầu trò chơi.

```
1 IMPORT class_file
2 IMPORT global_variable
3 IMPORT sound
4
5 DEFINE soluongoption = 5
6
7 FUNCTION draw_menu(current_option):
8     SET screen_width TO pygame.display.Info().current_w
9     SET screen_height TO pygame.display.Info().current_h
10
11    SET ImMenu TO pygame.image.load('menubackground.jpg')
12    ImMenu = pygame.transform.scale(ImMenu, (screen_width, screen_height))
13
14    screen.fill(Black)
15    screen.blit(ImMenu, (0, 0))
16
17    SET title_font TO pygame.font.Font('8-BIT WONDER.ttf', 60)
18    SET option_font TO pygame.font.Font('8-BIT WONDER.ttf', 32)
19
20    SET title_text TO title_font.render("MAZE GAME", True, Purple)
21    screen.blit(title_text, (WIDTH / 2 - title_text.get_width() / 2, 150))
22
23    SET options TO ["Start Game", "Settings", "Ranking", "About", "Exit"]
24    FOR i FROM 0 TO LENGTH(options) - 1:
25        IF i EQUALS current_option THEN
26            SET rendered_text TO option_font.render(options[i], True, Green)
27        ELSE:
28            SET rendered_text TO option_font.render(options[i], True, Yellow)
29
30        screen.blit(rendered_text, (WIDTH / 2 - rendered_text.get_width() / 2, 300 + 50 * i))
31
32 FUNCTION handle_menu_input(current_option):
33    FOR EACH event IN pygame.event.get():
34        IF event.type EQUALS pygame.QUIT THEN
35            pygame.quit()
36            sys.exit()
37        RETURN current_option, False
38        IF event.type EQUALS pygame.KEYDOWN THEN
39            IF event.key EQUALS pygame.K_UP THEN
40                SET current_option TO (current_option - 1) MOD soluongoption
41            ELSE IF event.key EQUALS pygame.K_DOWN THEN
42                SET current_option TO (current_option + 1) MOD soluongoption
43            ELSE IF event.key EQUALS pygame.K_RETURN THEN
44                RETURN current_option, True
45
46    RETURN current_option, False
47
48 FUNCTION xulymenu():
49    SET menu_option TO 0
50    SET in_menu TO True
51
52    WHILE in_menu:
53        menu_option, selected = handle_menu_input(menu_option)
```

```

54     draw_menu(menu_option)
55     pygame.display.update()
56
57     IF selected THEN
58         IF menu_option EQUALS 0 THEN
59             SET in_menu TO False
60         ELSE IF menu_option EQUALS 1 THEN
61             xulymenu_settings()
62         ELSE IF menu_option EQUALS 4 THEN
63             pygame.quit()
64             sys.exit()
65
66     IF menu_option EQUALS 0 THEN
67         mixer.music.stop()
68         transition.play()
69         sleep(1)
70         mixer.music.load('background.wav')
71         mixer.music.play(-1)
72     SET maze_game_loader TO MazeGameLoader()
73     maze_game_loader.run()
74

```

Listing 6.4: Mã giả MenuGame

6.3.4 Hàm draw_menu_setting(current_option)

Hàm có chức năng tương tự như là các hàm xử lý menugame

- **Mục đích:** Vẽ menu cài đặt trò chơi lên màn hình.
- **Tham số đầu vào:**
 - current_option: Chỉ số của tùy chọn hiện tại được chọn trong menu.
- **Công việc thực hiện:**
 1. Lấy kích thước màn hình và tải hình nền menu.
 2. Thực hiện các cài đặt liên quan đến giao diện như tạo font, tạo văn bản.
 3. Vẽ tiêu đề trò chơi và các tùy chọn cài đặt lên màn hình, sử dụng màu sắc phù hợp cho mỗi tùy chọn.
- **Kết quả đầu ra:** Giao diện menu cài đặt được vẽ lên màn hình.

6.3.5 Hàm xulymenu_settings()

- **Mục đích:** Xử lý sự kiện khi người dùng tương tác với menu cài đặt.
- **Công việc thực hiện:**
 1. Khởi tạo biến và danh sách các tùy chọn cài đặt.
 2. Bắt đầu vòng lặp vô hạn để theo dõi sự kiện từ bàn phím và vẽ lại menu cài đặt.
 3. Xử lý các sự kiện từ bàn phím:
 - Di chuyển lên/xuống để chọn tùy chọn.
 - Di chuyển qua trái/phải để thay đổi giá trị của tùy chọn.
 - Nhấn phím ESC để quay lại menu chính.
- **Kết quả đầu ra:** Cập nhật các tùy chọn cài đặt và giao diện menu khi người dùng tương tác.

```

1   optionsettings = ["Difficult: 20x20","GameMode: Player","FindPathMode: DFS",
2   ChooseStartEnd: Manual"]
3
4   function draw_menu_setting(current_option):
5       screen_width = pygame.display.Info().current_w
6       screen_height = pygame.display.Info().current_h
7       ImMenu = load_image('menubackground.jpg')
8       ImMenu = scale_image(ImMenu, (screen_width, screen_height))
9       screen.fill(Black)
10      screen.blit(ImMenu, (0, 0))
11      title_font = load_font('8-BIT WONDER.ttf', 60)
12      option_font = load_font('8-BIT WONDER.ttf', 32)
13      title_text = render_text("MAZE GAME", True, Purple)
14      screen.blit(title_text, (WIDTH / 2 - title_text.width / 2, 150))
15
16      for i, option in enumerate(optionsettings):
17          if i == current_option:
18              rendered_text = render_text(option, True, Green)
19          else:
20              rendered_text = render_text(option, True, White)
21          screen.blit(rendered_text, (WIDTH / 2 - rendered_text.width / 2, 300 + 50 * i
22      ))
23
24      function xulymenu_settings():
25          Dif_op = 0
26          Game_op = 0
27          Path_op = 0
28          Start_op = 0
29          menu_option = 0
30          in_menu = True
31          difficulty =["Difficult : 20x20","Difficult : 40x40","Difficult : 100x100"]
32          difficultyVa =[ (20,20) ,(30,30) ,(40,40) ,(100,100)]
33          game_modes = ["GameMode:Player","GameMode:Bot"]
34          game_modesVa =[True, False]
35          path_modes = ["FindPath: DFS","FindPath: BFS","FindPath :A*star"]
36          path_modesVa = [1,2,3]
37          start_modes =[ "ChooseStartEnd: Manual", "ChooseStartEnd: Automated"]
38          start_modesVa = [1,2]
39
40          while True:
41              soluongoptionse = 4
42              draw_menu_setting(menu_option)
43              pygame.display.update()
44              for event in pygame.event.get():
45                  if event.type == pygame.KEYDOWN :
46                      if event.type == pygame.QUIT:
47                          quit_game()
48                      if event.key == pygame.K_UP:
49                          menu_option = (menu_option - 1) % soluongoptionse
50                      else if event.key == pygame.K_DOWN:
51                          menu_option = (menu_option + 1) % soluongoptionse
52                      else if event.key == pygame.K_RIGHT:
53                          if menu_option == 0:
54                              Dif_op = (Dif_op + 1)%3
55                              optionsettings[menu_option] = difficulty[Dif_op]
56                          else if menu_option == 1:
57                              Game_op = (Game_op+1)%2
58                              optionsettings[menu_option] = game_modes[Game_op]
59                          else if menu_option == 2:
60                              Path_op = (Path_op+1)%3
61                              optionsettings[menu_option] = path_modes[Path_op]
62                          else if menu_option == 3 :
63                              Start_op = (Start_op + 1)%2
64                              optionsettings[menu_option] = start_modes[Start_op]
65                          else if event.key == pygame.K_LEFT:
66                              if menu_option == 0:
67                                  Dif_op = (Dif_op - 1)%3
68                                  optionsettings[menu_option] = difficulty[Dif_op]
69                              else if menu_option == 1:
70                                  Game_op = (Game_op-1)%2

```

```

69             optionsettings[menu_option] = game_modes[Game_op]
70     else if menu_option == 2:
71         Path_op = (Path_op-1)%3
72         optionsettings[menu_option] = path_modes[Path_op]
73     else if menu_option == 3 :
74         Start_op = (Start_op -1)%2
75         optionsettings[menu_option] = start_modes[Start_op]
76     else if event.key == pygame.K_ESCAPE :
77         xulymenu()

```

Listing 6.5: Menu_settings

6.4 Class MazeGameLoader(LoadingBar)

6.4.1 Giới thiệu

Lớp `MazeGameLoader` là một phần của trò chơi "Maze Game" và chịu trách nhiệm cho việc tải trò chơi. Trong báo cáo này, chúng tôi sẽ phân tích từng phương thức của lớp để hiểu cách nó hoạt động và vai trò của nó trong quá trình tải trò chơi.

6.4.2 Phương thức: `__init__()`

- Mô tả: Phương thức khởi tạo của lớp `MazeGameLoader` được gọi khi một đối tượng mới của lớp được tạo ra. Nó khởi tạo các thuộc tính cần thiết cho quá trình tải trò chơi.
- Các tham số:
 - `width` (mặc định là 800): Chiều rộng mặc định của cửa sổ trò chơi.
 - `height` (mặc định là 600): Chiều cao mặc định của cửa sổ trò chơi.
- Các bước thực hiện:
 1. Gán giá trị của `width` và `height` vào các thuộc tính `WIDTH` và `HEIGHT` của đối tượng.
 2. Khởi tạo biến `progress` với giá trị ban đầu là 0.
 3. Xác định kích thước và vị trí của thanh tiến trình trên màn hình.
 4. Khởi tạo danh sách các gợi ý `loading_tips`.
 5. Thiết lập các tham số cho việc hiển thị các gợi ý.
 6. Khởi tạo các biến liên quan đến hoạt ảnh sóng.
 7. Khởi tạo thư viện `pygame` và thiết lập cấu hình cửa sổ trò chơi.

6.4.3 Phương thức: `run()`

- Mô tả: Phương thức này chịu trách nhiệm chạy vòng lặp chính của quá trình tải trò chơi.
- Các bước thực hiện:
 1. Tải hình nền từ tệp 'backgroundload.jpg'.
 2. Trong vòng lặp:
 - Kiểm tra các sự kiện như thoát khỏi cửa sổ.
 - Vẽ nền và thanh tiến trình trên màn hình.
 - Hiển thị và cập nhật các gợi ý tải.
 - Vẽ hoạt ảnh sóng.
 - Cập nhật tiến độ và kiểm tra nếu quá trình tải đã hoàn thành.
 - Hoãn vòng lặp trong một khoảng thời gian ngắn.

6.4.4 Phương thức: draw_progress_bar()

- Mô tả: Phương thức này được sử dụng để vẽ thanh tiến trình trên màn hình.
- Các bước thực hiện:
 1. Vẽ ra ngoại vi của thanh tiến trình.
 2. Xác định chiều dài của thanh tiến trình dựa trên tiến độ.
 3. Áp dụng một hiệu ứng gradient màu sắc dựa trên tiến độ.

6.4.5 Phương thức: render_text()

- Mô tả: Phương thức này được sử dụng để hiển thị văn bản trên màn hình.
- Các bước thực hiện:
 1. Hiển thị tiến độ tải và thông báo "Loading Maze Game...".
 2. Hiển thị các gợi ý trong danh sách `current_tips`.

6.4.6 Phương thức: update_loading_tips()

- Mô tả: Phương thức này cập nhật các gợi ý hiển thị dựa trên tiến độ của quá trình tải.
- Các bước thực hiện:
 1. Xác định thời điểm hiện tại.
 2. Nếu đã đủ thời gian cho một gợi ý mới, cập nhật danh sách `current_tips`.

6.4.7 Phương thức: draw_wave_animation()

- Mô tả: Phương thức này được sử dụng để vẽ hoạt ảnh sóng trên thanh tiến trình.
- Các bước thực hiện:
 1. Tính toán độ lệch của sóng dựa trên tham số `angle`.
 2. Vẽ ra sóng trên thanh tiến trình để tạo ra hiệu ứng di động.

```
1 BEGIN MazeLoading Class
2 FUNCTION Initialize(width, height)
3     SET self.WIDTH TO width
4     SET self.HEIGHT TO height
5     SET self.progress TO 0
6     SET self.bar_width TO 600
7     SET self.bar_height TO 30
8     SET self.bar_x TO (self.WIDTH - self.bar_width) DIV 2
9     SET self.bar_y TO (self.HEIGHT - self.bar_height) DIV 2 + 50
10    SET self.loading_tips TO [
11        "Tip 1: Don't forget to explore every corner of the maze!",
12        "Tip 2: Some paths might seem blocked, but they could be shortcuts!",
13        "Tip 3: Keep an eye out for hidden passages behind walls!",
14        "Tip 4: If you're stuck, try retracing your steps to find a new route!",
15        "Tip 5: Don't rush through the maze - take your time to solve it!"
16    ]
17    SET self.tip_timer TO 0
18    SET self.tip_delay TO 2000
19    SET self.current_tips TO []
20    SET self.angle TO 0
21    SET self.wave_amplitude TO 5
22    SET self.wave_frequency TO 0.1
23    CALL Pygame.Initialize()
24    SET screen_width TO Pygame.GetDisplayInfo().current_w
25    SET screen_height TO Pygame.GetDisplayInfo().current_h
```

```

26     SET self.screen TO Pygame.SetMode((self.WIDTH, self.HEIGHT))
27     CALL Pygame.SetCaption("Maze Game Loading")
28     SET self.font TO Pygame.CreateFont(None, 36)
29 END FUNCTION
30
31 FUNCTION Run()
32     SET running TO True
33     SET backgroundLoading TO Pygame.LoadImage('backgroundload.jpg')
34     SET screen_width TO Pygame.GetDisplayInfo().current_w
35     SET screen_height TO Pygame.GetDisplayInfo().current_h
36     SET backgroundLoading TO Pygame.Transform.Scale(backgroundLoading, (screen_width,
37     screen_height))
38     WHILE running DO
39         FOR EACH event IN Pygame.Event.Get()
40             IF event.type EQUALS Pygame.QUIT THEN
41                 SET running TO False
42             END FOR
43
44             Pygame.Fill(self.screen, Black)
45             Pygame.Blit(self.screen, backgroundLoading, (0, 0))
46             CALL DrawProgressBar()
47             CALL RenderText()
48             CALL UpdateLoadingTips()
49             CALL DrawWaveAnimation()
50             Pygame.Display.Flip()
51             SET self.progress TO self.progress + 0.2
52             IF self.progress >= 100 THEN
53                 SET running TO False
54             END IF
55             Pygame.Time.Delay(10)
56         END WHILE
57     END FUNCTION
58
59 FUNCTION DrawProgressBar()
60     SET outer_border_rect TO Pygame.Rect(self.bar_x - 2, self.bar_y - 2, self.
61     bar_width + 4, self.bar_height + 4)
62     Pygame.Draw.Rect(self.screen, (255, 255, 255), outer_border_rect, 2)
63     SET progress_width TO INT(self.progress / 100 * self.bar_width)
64     SET color_gradient TO (MIN(255, INT(255 * (100 - self.progress) / 100)),
65     MIN(255, INT(255 * self.progress / 100)), 200)
66     Pygame.Draw.Rect(self.screen, color_gradient, (self.bar_x, self.bar_y,
67     progress_width, self.bar_height))
68 END FUNCTION
69
70 FUNCTION RenderText()
71     SET percentage_text TO Pygame.RenderFont(f"{{INT(self.progress)}}%", True, Black)
72     SET text_rect TO percentage_text.GetRect(center=(self.WIDTH DIV 2, self.HEIGHT
73     DIV 2 + 50))
74     Pygame.Blit(self.screen, percentage_text, text_rect)
75     SET dynamic_text TO Pygame.RenderFont("Loading Maze Game...", True, (255, 255,
76     255))
77     SET dynamic_text_rect TO dynamic_text.GetRect(center=(self.WIDTH DIV 2, self.
78     HEIGHT DIV 2 - 50))
79     Pygame.Blit(self.screen, dynamic_text, dynamic_text_rect)
80     FOR i FROM 0 TO LENGTH(self.current_tips) - 1
81         SET tip_text TO Pygame.RenderFont(self.current_tips[i], True, (255, 255, 255)
82     )
83         SET tip_text_rect TO tip_text.GetRect(center=(self.WIDTH DIV 2, self.HEIGHT
84     DIV 2 + 100 + i * 40))
85         Pygame.Blit(self.screen, tip_text, tip_text_rect)
86     END FOR
87 END FUNCTION
88
89 FUNCTION UpdateLoadingTips()
90     IF Pygame.Time.GetTicks() - self.tip_timer > self.tip_delay THEN
91         SET self.tip_timer TO Pygame.Time.GetTicks()
92         SET self.current_tips TO SUBSTRING(self.loading_tips, 0, INT(self.progress /
93     20) + 1)
94     END IF
95 END FUNCTION

```

```
87     FUNCTION DrawWaveAnimation()
88         SET wave_offset TO self.wave_amplitude * SIN(self.angle)
89         SET progress_width TO INT(self.progress / 100 * self.bar_width)
90         Pygame.Draw.Rect(self.screen, (255, 255, 255),
91                         (self.bar_x + progress_width - 5, self.bar_y - 5 + wave_offset,
92                          5, self.bar_height + 10))
93         SET self.angle TO self.angle + self.wave_frequency
94     END FUNCTION
95 END MazeLoading Class
```

Listing 6.6: Mă giả Class MazeLoading

6.5 Giao Diện PauseGame

6.5.1 Biến Quan Trọng

- **paused:** Một biến boolean dùng để xác định trạng thái của trò chơi. Nếu paused là True, trò chơi được tạm dừng; nếu là False, trò chơi đang chạy. Có nghĩa ta sẽ cho if paused thì continue trong hàm nếu nó paused thì nó sẽ trong vòng lặp vô hạn và không thực hiện các dòng các có chức năng tương tác với người dùng sau nó .
- **surface:** Đối tượng pygame.Surface được sử dụng để vẽ các thành phần của màn hình tạm dừng, cụ thể nó sẽ là một màn hình có màu đơn sắc có độ trong suốt để lén cái màn hình screen ban đầu để có hiệu ứng dừng lại.
- **DarkGray, White, Black:** Các biến được sử dụng để định nghĩa màu sắc trong trò chơi, cụ thể là màu xám đậm, màu trắng và màu đen.

6.5.2 Hàm pause_screen(screen)

Hàm này được sử dụng để hiển thị màn hình tạm dừng trò chơi. Nó vẽ các hình chữ nhật và văn bản trên surface và sau đó vẽ surface lên màn hình chính (screen).

6.5.3 Hàm continue_game()

Hàm này xử lý sự kiện khi người chơi muốn tiếp tục hoặc tạm dừng trò chơi. Nếu phím ESCAPE được nhấn, trò chơi sẽ tạm dừng hoặc tiếp tục tùy thuộc vào trạng thái hiện tại của biến paused. Nếu phím R được nhấn, trò chơi sẽ được thiết lập lại về trạng thái ban đầu bằng cách set lại các biến bao gồm :

- Vị trí của người chơi về vị trí ban đầu (hoặc có thể chọn vị trí start, end mới)
- Các ô hướng dẫn sẽ được quay lại màu ô ban đầu
- Nhạc nền cũng sẽ được phát lại
- Mọi thông số bao gồm cả điểm , số bước đi đều sẽ được set về defualt

. Nếu phím M được nhấn, chức năng xử lý menu sẽ được gọi.

```
1   SET paused TO False
2   SET surface TO Pygame.Surface([WIDTH, HEIGHT], Pygame.SRCALPHA)
3   SET DarkGray TO (128, 128, 128)
4   SET White TO (255, 255, 255)
5   SET Black TO (0, 0, 0)
6
7 FUNCTION pause_screen(screen)
8     # Display the pause screen
9     Pygame.Draw.Rect(surface, (128, 128, 128, 2), [0, 0, WIDTH, HEIGHT])
10    Pygame.Draw.Rect(surface, DarkGray, [200, 150, 600, 50], 0, 10)
11    SET reset TO Pygame.Draw.Rect(surface, White, [200, 220, 280, 50], 0, 10)
12    SET save TO Pygame.Draw.Rect(surface, White, [520, 220, 280, 50], 0, 10)
13    surface.Blit(font.Render('Game Paused: Escape to Resume', True, Black), (220, 160))
14    surface.Blit(font.Render('Restart', True, Black), (220, 230))
15    surface.Blit(font.Render('Save', True, Black), (540, 230))
16    screen.Blit(surface, (0, 0))
17    Pygame.Display.Update()
18
19
20 FUNCTION continue_game()
21   # Continue the game
22   SET global paused TO False
23   FOR EACH event IN Pygame.Event.Get()
24     IF event.type EQUALS Pygame.KEYDOWN THEN
25       IF event.key EQUALS Pygame.K_ESCAPE THEN
26         Pygame.Mixer.Pause()
```

```

27     SET paused TO NOT paused
28 ELSE IF event.key EQUALS Pygame.K_r THEN
29     SET grid_cells, start, end TO reset_game(grid_cells, TILE)
30     SET player.x, player.y TO start.x, start.y
31     SET paused TO False
32 ELSE IF event.key EQUALS Pygame.K_m THEN
33     SET paused TO NOT paused
34     xulymenu()
35 IF paused THEN
36     pause_screen(sc)
37 CONTINUE # Skip the reing part of the loop if paused

```

Listing 6.7: Mã Giả cho chức năng Pause/Continue/Resart

6.6 Dash Board

6.6.1 Giao diện dashboard

- Dashboard trong game hiển thị thông tin về thời gian chơi, số bước đi, chế độ chơi, cấp độ hiện tại, thuật toán được sử dụng, và các nút chức năng khác nhau cho người chơi.
- Giao diện này được vẽ bởi hàm `draw_dashboard()`.



Hình 6.10: Màn hình dashboard

- Giao diện này cung cấp cho người chơi cái nhìn tổng quan về trạng thái hiện tại của trò chơi và cho phép họ tương tác với trò chơi thông qua các nút bấm.

6.6.2 Cấu trúc và chức năng của hàm `draw_dashboard`

1. Các biến toàn cục

- hours, minutes, seconds: Các biến lưu trữ thời gian đã trôi qua được chuyển đổi sang định dạng giờ:phút:giây.
- `elapsed_time` là biến lưu thời gian đã trôi qua kể từ khi bắt đầu trò chơi (kể từ khi người chơi nhấn chọn vị trí của Tom và Jerry), được gán bằng `game_clock.get_elapsed_time()` để lấy thời gian đã trôi qua từ đối tượng `game_clock`. Thời gian này sau đó được chuyển đổi thành giờ, phút và giây để hiển thị trên màn hình.
- Các biến lưu trữ tọa độ và hình ảnh các nút được khai báo trước ở ngoài hàm.

Vẽ thông tin và các nút chức năng lên Dashboard

- Lệnh `time_text = d_font.render(f"hours:02:minutes:02:seconds:02", True, "red")` tạo đối tượng văn bản hiển thị thời gian với định dạng hh:mm:ss.
- Lệnh `screen.blit(timer_img, time_coord1)`: Vẽ biểu tượng thời gian (ảnh timer_img) ở tọa độ time_coord1 của màn hình.
- Tương tự, các lệnh `screen.blit` tiếp theo vẽ các văn bản về số bước đi (steps), chế độ chơi (mode_game), cấp độ hiện tại (level), và thuật toán được sử dụng (algo) lên màn hình tại các tọa độ đã xác định.
- Các nút chức năng bao gồm: nút tạm dừng, nút di chuyển (lên, xuống, trái, phải), nút đổi thuật toán, nút quay lại cài đặt, nút gợi ý đường đi nếu chế độ chơi là "Player" và nút Play Bot nếu chế độ chơi là Bot, được vẽ lên màn hình tại các tọa độ xác định trước để người chơi nhấn và tương tác.

6.6.3 Các hàm khác

- Các hàm `is_mouse_over_button()`, `handle_suggest_button_click()`, `handle_change_algorithm_button_click()`, `handle_pause_button_click()`: chuyển các nút đã được vẽ lên màn hình thành nút tương tác chuột.
- Các hàm `send_key_q()`, `send_key_space()`, `send_key_escape()`, `send_key_right()`: mã hóa 1 số phím tắt trên bàn phím thành thành các nút chức năng giống với các nút trên.

6.7 Transistion

6.7.1 Hàm transition

Hàm `transition` tạo một bề mặt chuyển cảnh với hiệu ứng vòng tròn giãn nở. Cách hoạt động của hàm như sau:

- Tao một bề mặt mới với kích thước toàn màn hình bằng cách gọi `pygame.Surface((WIDTH, HEIGHT))`.
- Tính toán bán kính của vòng tròn dựa trên giá trị của `transition_value`. Bán kính này thay đổi để tạo hiệu ứng giãn nở và thu nhỏ.
- Vẽ một vòng tròn màu trắng ở giữa màn hình với bán kính đã tính.
- Trả về bề mặt chuyển cảnh đã tạo.

6.7.2 Hàm intro_transition

Hàm `intro_transition` điều khiển hiệu ứng chuyển cảnh từ từ. Cách hoạt động của hàm như sau:

- Khởi tạo giá trị `transition_value` và tốc độ chuyển cảnh `transition_speed`.
- Sử dụng một vòng lặp để cập nhật và hiển thị hiệu ứng chuyển cảnh cho đến khi `transition_value` đạt đến giá trị 30.
- Trong vòng lặp, xử lý các sự kiện của Pygame, bao gồm việc thoát chương trình khi người dùng đóng cửa sổ.
- Tăng giá trị `transition_value` theo `transition_speed`.
- Gọi hàm `transition` với `transition_value` hiện tại để tạo bề mặt chuyển cảnh.
- Thiết lập màu trắng làm màu trong suốt trên bề mặt chuyển cảnh bằng cách sử dụng `set_colorkey((255, 255, 255))`.

- Vẽ bìa mặt chuyển cảnh lên màn hình chính.
- Cập nhật màn hình bằng cách gọi `pygame.display.flip()` và đồng bộ hóa tốc độ khung hình bằng `clock.tick(FPS)`.

6.7.3 Kết luận

Doạn mã này cung cấp một cách hiệu quả để tạo và hiển thị hiệu ứng chuyển cảnh sử dụng Pygame. Hiệu ứng này làm cho việc chuyển đổi giữa các cảnh trong trò chơi trở nên mượt mà và hấp dẫn hơn. Việc sử dụng vòng lặp để cập nhật và hiển thị hiệu ứng đảm bảo rằng người dùng sẽ thấy được hiệu ứng chuyển cảnh diễn ra từ từ, tạo ra trải nghiệm thị giác tốt hơn.

```

1 FUNCTION transition(transition_value)
2     CREATE transition_surf AS new Surface of size (WIDTH, HEIGHT)
3     SET radius TO int((30 - abs(transition_value)) * 8)
4     DRAW circle on transition_surf with color (255, 255, 255) at (WIDTH//2, HEIGHT//2)
5         with radius
6     RETURN transition_surf
7 END FUNCTION
8
9 FUNCTION intro_transition()
10    SET transition_value TO 0
11    SET transition_speed TO 1 # Adjust this value to control the speed of the transition
12    WHILE transition_value < 30
13        FOR event IN pygame.event.get()
14            IF event.type == pygame.QUIT THEN
15                CALL pygame.quit()
16                CALL sys.exit()
17            # Update transition value
18            INCREMENT transition_value BY transition_speed
19            # Draw the transition effect
20            SET transition_surf TO transition(transition_value)
21            CALL transition_surf.set_colorkey((255, 255, 255)) # Set the color key to white
22            for transparency
23                CALL screen.blit(transition_surf, (0, 0))
24            CALL pygame.display.flip()
25            CALL clock.tick(FPS)
26    END FUNCTION

```

Listing 6.8: Mã giả cho các hàm xử lý Hiệu Ứng Chuyển Cảnh

6.8 Giao Diện Load Game

6.8.1 Hàm load_filegame

Hàm `load_filegame` có chức năng đọc thông tin từ tệp JSON `database.json` và trả về danh sách các tên tệp game đã lưu của người dùng hiện tại. Cụ thể:

- Hàm mở tệp `database.json` và đọc dữ liệu.
- Duyệt qua từng tài khoản trong dữ liệu để tìm tên người dùng khớp với người dùng hiện tại.
- Thu thập các tên tệp trò chơi đã lưu, bỏ qua tên người dùng.
- Trả về danh sách các tên tệp trò chơi.

6.8.2 Hàm draw_mini_maze

Hàm `draw_mini_maze` chịu trách nhiệm vẽ một bản đồ thu nhỏ của mê cung lên màn hình. Các tham số đầu vào bao gồm màn hình, danh sách các ô mê cung, số cột, số hàng, vị trí người chơi, vị trí bắt đầu và kết thúc, tọa độ x và y, và kích thước của mê cung. Cách hoạt động của hàm như sau:

- Tính toán kích thước của mỗi ô dựa trên kích thước mê cung và số cột, số hàng.

- Duyệt qua từng ô trong danh sách và gọi phương thức `draw_mini_map` của lớp `Cell` để vẽ ô đó.
- Vẽ người chơi, vị trí bắt đầu, và vị trí kết thúc bằng các vòng tròn với màu sắc khác nhau.

6.8.3 Hàm draw_load_screen

Hàm `draw_load_screen` có nhiệm vụ vẽ giao diện màn hình tải game, hiển thị các bản đồ thu nhỏ của các trò chơi đã lưu và tên của chúng. Cách hoạt động của hàm:

- Xóa màn hình bằng cách đổ màu trắng lên toàn bộ màn hình.
- Tính toán số lượng mê cung có thể hiển thị trên một dòng và khoảng cách giữa chúng.
- Vẽ từng mê cung và tên tệp lên màn hình theo các vị trí đã tính toán.
- Kiểm tra vị trí con trỏ chuột để làm nổi bật mê cung đang được trỏ tới.
- Cập nhật màn hình để hiển thị các thay đổi.

6.8.4 Hàm Load_Game_Display

Hàm `Load_Game_Display` là hàm chính để hiển thị giao diện tải game và xử lý các sự kiện liên quan đến nó. Các bước thực hiện bao gồm:

- Tải danh sách các tệp game đã lưu bằng hàm `load_filegame`.
- Tính toán số lượng mê cung có thể hiển thị trên một dòng và khoảng cách giữa chúng.
- Sử dụng vòng lặp sự kiện của Pygame để xử lý các sự kiện như nhấn phím và nhấn chuột.
- Khi người dùng nhấn chuột vào một mê cung, hàm gọi hộp thoại xác nhận để tải trò chơi tương ứng.
- Vẽ giao diện load game bằng cách gọi hàm `draw_load_screen`.

6.8.5 Kết luận

Các hàm này cung cấp một cách hiệu quả để tải và hiển thị các trò chơi đã lưu trong Pygame. Chúng sử dụng các phương pháp đồ họa của Pygame để tạo ra một giao diện người dùng trực quan và thân thiện, cho phép người dùng dễ dàng chọn và tải lại các trò chơi đã lưu. Điều này cải thiện trải nghiệm người dùng và giúp quản lý trò chơi đã lưu một cách thuận tiện.

```

1 IMPORT pygame
2 IMPORT global_variables, save_load_game, class_file
3 IMPORT json
4
5
6 "Definite draw_mini_map function for Cell class"
7 FUNCTION draw_mini_map(self, screen, TILE, maze_x, maze_y)
8     SET x, y TO maze_x + self.x * TILE, maze_y + self.y * TILE
9     IF self.walls['top'] THEN
10        CALL pygame.draw.rect(screen, pygame.Color('green'), pygame.Rect(x, y, TILE, self.thickness))
11    IF self.walls['right'] THEN
12        CALL pygame.draw.rect(screen, pygame.Color('green'), pygame.Rect(x + TILE - self.thickness, y, self.thickness, TILE))
13    IF self.walls['bottom'] THEN
14        CALL pygame.draw.rect(screen, pygame.Color('green'), pygame.Rect(x, y + TILE - self.thickness, TILE, self.thickness))
15    IF self.walls['left'] THEN
16        CALL pygame.draw.rect(screen, pygame.Color('green'), pygame.Rect(x, y, self.thickness, TILE))
17    IF self.path THEN
18        CALL pygame.draw.rect(screen, pygame.Color('red'), (x + 10, y + 10, TILE - 20, TILE - 20))

```

```

19 END FUNCTION
20
21 "Load game file names"
22 FUNCTION load_filegame()
23     SET file_names TO EMPTY LIST
24     OPEN 'database.json' AS f
25         SET data TO json.load(f)
26         FOR account IN data
27             IF login_forms.User_name IN account THEN
28                 IF len(account) > 1 THEN
29                     FOR key IN account
30                         IF key != login_forms.User_name THEN
31                             SET tmp TO key.split("_")
32                             APPEND tmp[0] TO file_names
33                             PRINT tmp[0]
34             RETURN file_names
35 END FUNCTION
36
37
38 FUNCTION draw_mini_maze(screen, grid_cells, cols, rows, player, start, end, x, y, size)
39     SET cell_size TO size / max(cols, rows)
40     FOR cell IN grid_cells
41         CALL cell.draw_mini_map(screen, cell_size, x, y)
42     SET start, end TO grid_cells[start[0] + start[1] * cols], grid_cells[end[0] + end[1] * cols]
43     SET player TO Player(player[0], player[1], "Tom_animation.png", 4, 4, cell_size)
44     CALL pygame.draw.circle(screen, pygame.Color('yellow'), (x + player.x * cell_size + cell_size // 2, y + player.y * cell_size + cell_size // 2), cell_size // 4)
45     CALL pygame.draw.circle(screen, pygame.Color('red'), (x + start.x * cell_size + cell_size // 2, y + start.y * cell_size + cell_size // 2), cell_size // 4)
46     CALL pygame.draw.circle(screen, pygame.Color('blue'), (x + end.x * cell_size + cell_size // 2, y + end.y * cell_size + cell_size // 2), cell_size // 4)
47 END FUNCTION
48
49
50 FUNCTION draw_load_screen(screen, file_names, selected_maze_index)
51     CALL screen.fill((255, 255, 255))
52     SET font TO pygame.font.Font(None, 36)
53     SET text_y TO MARGIN
54     SET num_mazes_per_row TO min(len(file_names), NUM_MAZES)
55     IF len(file_names) == 0 THEN
56         SET num_mazes_per_row TO 1
57     SET maze_WIDTH_mini_maze TO 250
58     SET maze_HEIGHT_mini_maze TO 250
59     SET x_spacing TO (WIDTH - num_mazes_per_row * maze_WIDTH_mini_maze - (num_mazes_per_row - 1) * MARGIN) / 2
60     SET y_spacing TO (HEIGHT - len(file_names) / num_mazes_per_row * (maze_HEIGHT_mini_maze + MARGIN)) / 2
61     FOR i, file_name IN ENUMERATE(file_names)
62         SET row TO i // num_mazes_per_row
63         SET col TO i % num_mazes_per_row
64         SET x TO x_spacing + col * (maze_WIDTH_mini_maze + MARGIN)
65         SET y TO y_spacing + row * (maze_HEIGHT_mini_maze + MARGIN)
66         SET maze_rect TO pygame.Rect(x, y, maze_WIDTH_mini_maze, maze_HEIGHT_mini_maze)
67         SET grid_cells, cols, rows, player, start, end, steps, elapsed_time TO load_game(
file_name)
68         CALL draw_mini_maze(screen, grid_cells, cols, rows, player, start, end, x, y,
maze_WIDTH_mini_maze)
69         SET file_text TO font.render(file_name, True, (0, 0, 0))
70         SET file_rect TO file_text.get_rect()
71         SET file_rect.midtop TO (x + maze_WIDTH_mini_maze // 2, y + maze_HEIGHT_mini_maze
+ 10)
72         CALL screen.blit(file_text, file_rect)
73         SET mouse_pos TO pygame.mouse.get_pos()
74         IF maze_rect.collidepoint(mouse_pos) THEN
75             SET selected_maze_index TO i
76         IF selected_maze_index IS NOT None THEN
77             SET selected_x TO x_spacing + (selected_maze_index % num_mazes_per_row) * (
maze_WIDTH_mini_maze + MARGIN)

```

```

78     SET selected_y TO y_spacing + (selected_maze_index // num_mazes_per_row) * (
79         maze_HEIGHT_mini_maze + MARGIN)
80     SET selected_rect TO pygame.Rect(selected_x, selected_y, maze_WIDTH_mini_maze,
81         maze_HEIGHT_mini_maze)
82     CALL pygame.draw.rect(screen, pygame.Color('black'), selected_rect, 3)
83     CALL pygame.display.flip()
84 END FUNCTION
85
84 FUNCTION Load_Game_Display(screen, file_names)
85     SET num_mazes_per_row TO min(len(file_names), NUM_MAZES)
86     IF len(file_names) == 0 THEN
87         SET num_mazes_per_row TO 1
88     SET maze_WIDTH_mini_maze TO 250
89     SET maze_HEIGHT_mini_maze TO 250
90     SET x_spacing TO (WIDTH - num_mazes_per_row * maze_WIDTH_mini_maze - (
91         num_mazes_per_row - 1) * MARGIN) / 2
92     SET y_spacing TO (HEIGHT - len(file_names)) / num_mazes_per_row * (
93         maze_HEIGHT_mini_maze + MARGIN)) / 2
94     SET file_names TO load_filegame()
95     SET selected_maze_index TO None
96     SET running TO True
97     WHILE running DO
98         FOR event IN pygame.event.get() DO
99             IF event.type == pygame.QUIT THEN
100                 SET running TO False
101             ELSE IF event.type == pygame.KEYDOWN THEN
102                 IF event.key == pygame.K_ESCAPE THEN
103                     RETURN
104                 ELSE IF event.type == pygame.MOUSEBUTTONDOWN THEN
105                     IF event.button == 1 THEN
106                         SET mouse_pos TO pygame.mouse.get_pos()
107                         SET backbutton TO pygame.Rect(0, 0, rendered_text.get_width(),
108                                         rendered_text.get_height())
109                         IF backbutton.collidepoint(mouse_pos) THEN
110                             RETURN
111                         SET mouse_pos TO pygame.mouse.get_pos()
112                         FOR i, file_name IN ENUMERATE(file_names)
113                             SET row TO i // num_mazes_per_row
114                             SET col TO i % num_mazes_per_row
115                             SET x TO x_spacing + col * (maze_WIDTH_mini_maze + MARGIN)
116                             SET y TO y_spacing + row * (maze_HEIGHT_mini_maze + MARGIN)
117                             SET maze_rect TO pygame.Rect(x, y, maze_WIDTH_mini_maze,
118                                         maze_HEIGHT_mini_maze)
119                             IF maze_rect.collidepoint(mouse_pos) THEN
120                                 CALL confirm_dialog(screen, file_name, file_names, WIDTH, HEIGHT)
121                                 PRINT file_name
122                                 CALL draw_load_screen(screen, file_names, selected_maze_index)
123                                 CALL pygame.quit()
124 END FUNCTION

```

Listing 6.9: Mã giả cho các hàm xử lý giao diện Load_Game

6.9 Giao Diện save game

```

1 function save_screen(screen):
2     file_name = ""
3     msg = ""
4     paused = True
5
6     while paused:
7         for event in pygame.get_events():
8             if event.type == QUIT:
9                 quit_program()
10
11         open 'database.json' as file:
12             data = parse_json(file)
13             for account in data:
14                 if User_name in account:

```

```

15         if length of account > 6:
16             msg = 'You have reached the limit of file'
17         else:
18             if event.type == KEYDOWN:
19                 if event.key == BACKSPACE:
20                     file_name = remove_last_character(file_name)
21                 elif event.key == RETURN:
22                     if file_name.strip() is empty:
23                         msg = "Please enter a valid file name"
24                     elif file_name + TAIL exists in account:
25                         msg = "This name has existed"
26                     else:
27                         msg = 'Save successfully!'
28                         display_message_on_surface(msg)
29                         update_display()
30                         delay(1000)
31                         clear_message_on_surface()
32                         return file_name
33                 elif event.key == ESCAPE:
34                     return None
35                 elif event_unicode is not empty:
36                     if length of file_name < 10:
37                         file_name += event_unicode
38
39             # Display save screen
40             draw_rect(surface, (128, 128, 128, 2), [0, 0, WIDTH, HEIGHT])
41             draw_rect(screen, DarkGray, [pause_x + 100, pause_y, pause_width, pause_height],
42             0, 10)
43             draw_rect(screen, White, [pause_x + 420, pause_y + 70, 280, 50], 0, 10)
44             draw_text_on_surface('Save', (pause_x + 530, pause_y + 80))
45             draw_text_on_surface('Enter file name: ' + file_name, (pause_x + 120, pause_y +
46             20))
47
48             if msg is not empty:
49                 draw_text_on_surface(msg, (pause_x + 220, pause_y + 220))
50                 update_display()
51                 delay(1000)
52             return
53
54             update_display()

```

- Đây là hàm có chức năng hiển thị giao diện save game cho người chơi.
- Khi người chơi nhấn phím S hoặc vào dừng trò chơi, ta cần xét lại biến toàn cục paused = True
- Trong vòng lặp vô tận, ta cần kiểm tra tài khoản của người chơi hiện tại liệu đã đạt đến số file giới hạn chưa, nếu rồi, ta sẽ thông báo đã đạt giới hạn file và thoát ra.
- Ngược lại, ta tiếp tục xét xem tên file người chơi đặt có trùng với tên file nào khác chưa, nếu có, ta thông báo đã lưu và thoát game.
- Bằng việc kiểm soát số lượng ký tự được nhập (10 ký tự), người chơi sẽ không nhập tràn ô nhập file.
- Nếu tên file nhập vào hợp lệ, ta thông báo cho người chơi và return về tên file.

6.10 Giao Diện Màn Hình Chiến Thắng

```

1     function winning_screen(elapsed_time, steps):
2         running = True
3         winning_box_y = get_display_height()
4         transition_speed = 70 //speed of winning_box
5         background = load_image("winning_background.png")
6         background = scale_image(background, WIDTH, HEIGHT)
7
8         font = create_font(None, 50)

```

```

10
11     while running:
12         screen_width = get_display_width()
13         screen_height = get_display_height()
14         restart_button = load_image('restart_button.png')
15         restart_button = scale_image(restart_button, restart_button.width / 2,
16         restart_button.height / 2)
17         back_button = load_image('back_button.png')
18         back_button = scale_image(back_button, back_button.width / 2, back_button.height
19         / 2)
20         surface = create_surface(screen_width, screen_height, SRCALPHA)
21         winning_box = load_image('winning_box.png')
22         winning_box = scale_image(winning_box, winning_box.width * 2, winning_box.height
23         * 2)
24
25         blit_image(screen, background, 0, 0)
26
27         winning_box_x = (screen_width - winning_box.width) // 2
28         restart_button_x = winning_box_x + winning_box.width // 2 - restart_button.width
29         // 2 + 25 * WIDTH / 96
30         restart_button_y = winning_box_y + winning_box.height // 2 - restart_button.
31         height // 2 + 20 * HEIGHT / 72
32         back_button_x = winning_box_x + winning_box.width // 2 - back_button.width // 2 -
33         25 * WIDTH / 256
34         back_button_y = winning_box_y + winning_box.height // 2 - back_button.height // 2
35         + 83 * HEIGHT / 288
36
37         restart_button_rect = get_rect(restart_button, center=(restart_button_x,
38         restart_button_y))
39         back_button_rect = get_rect(back_button, center=(back_button_x, back_button_y))
40
41         blit_image(screen, winning_box, winning_box_x, winning_box_y)
42         blit_image(screen, restart_button, restart_button_rect)
43         blit_image(screen, back_button, back_button_rect)
44
45         steps_text = render_text(font, f"Number of steps: {steps}", True, (0, 0, 0))
46         time_text = render_text(font, f"Time: {elapsed_time}", True, (0, 0, 0))
47
48         steps_text_rect = get_rect(steps_text, center=(winning_box_x + winning_box.width
49         // 2, winning_box_y + 170 * HEIGHT / 432))
50         time_text_rect = get_rect(time_text, center=(winning_box_x + winning_box.width //
51         2, winning_box_y + 220 * HEIGHT / 432))
52
53         blit_image(screen, steps_text, steps_text_rect)
54         blit_image(screen, time_text, time_text_rect)
55
56         if winning_box_y > (screen_height - winning_box.height) // 2:
57             winning_box_y -= transition_speed
58
59         update_display()
60
61         for each event in get_events():
62             if event.type == QUIT:
63                 quit_program()
64                 exit_program()
65             elif event.type == MOUSEBUTTONDOWN:
66                 x, y = event.position
67                 if restart_button_rect.contains(x, y):
68                     return True
69                 elif back_button_rect.contains(x, y):
70                     return False

```

Listing 6.10: Hàm tạo giao diện màn hình chiến thắng

- Hàm này sẽ tạo ra màn hình chiến thắng gồm một background và một hộp thoại tương tác với người dùng.
- Hàm nhận vào các thông số là số bước đã đi và thời gian chơi để hiển thị cho người dùng

-
- Đầu tiên, ta tải lên các ảnh cần thiết đã được chuẩn bị sẵn trong thư mục source code lên, sau đó, đặt mọi thứ trong một vòng lặp vô hạn while running.
 - Sau đó, tính toán các tỉ lệ theo thông số của desktop để đặt vị trí các box và nút. Hàm cũng đồng thời tạo hiệu ứng di chuyển box thông báo chiến thắng từ dưới lên bằng cách thay đổi tọa độ winning_box_y trong vòng lặp.
 - Thực hiện việc lấy tọa độ vùng các nút bằng hàm get_rect, sau đó, xử lý sự kiện nhất chuột nếu người chơi chọn vào 1 trong hai ô. Nếu người chơi chọn nút back, hàm trả về False để xử lý tiếp trong hàm và dẫn về menu. Ngược lại, hàm trả về true, ta gọi hàm new game để xử lý tiếp trò chơi mới.

6.11 Giao Diện Ranking

6.11.1 Khởi tạo và Cài đặt Pygame

Mã nguồn bắt đầu bằng việc nhập các module cần thiết và khởi tạo Pygame:

- Các module cần thiết như login_forms, menu, và các biến toàn cục được nhập vào.
- Thư viện Pygame và các thư viện hệ thống (sys) được nhập vào để xử lý đồ họa và sự kiện.
- Khởi tạo hệ thống hiển thị và font chữ của Pygame bằng cách gọi pygame.display.init() và pygame.font.init().
- Định nghĩa hai font chữ sử dụng trong trò chơi: FONT2 và FONT.

6.11.2 Hàm get_rank

Hàm get_rank có nhiệm vụ cập nhật bảng xếp hạng dựa trên thời gian hoàn thành và số bước của người chơi. Cách hoạt động của hàm như sau:

- Xác định kích thước mê cung (20x20, 40x40, hoặc 100x100) và ánh xạ nó tới chỉ số tương ứng (0, 1, 2).
- Mở và đọc tệp ranking.json để lấy bảng xếp hạng hiện tại.
- Kiểm tra xem người chơi hiện tại đã có mặt trong bảng xếp hạng hay chưa và cập nhật thông tin nếu thời gian mới tốt hơn.
- Sắp xếp lại bảng xếp hạng theo thời gian hoàn thành.
- Ghi lại bảng xếp hạng mới vào tệp ranking.json.

6.11.3 Hàm load_ranking

Hàm load_ranking có chức năng tải bảng xếp hạng từ tệp ranking.json và trả về ba bảng xếp hạng tương ứng với ba kích thước mê cung. Cách hoạt động:

- Mở và đọc tệp ranking.json.
- Trả về ba bảng xếp hạng dưới dạng tuple: rank[0], rank[1], và rank[2].

6.11.4 Hàm draw_ranking

Hàm draw_ranking chịu trách nhiệm vẽ bảng xếp hạng lên màn hình. Cách hoạt động:

- Đặt vị trí y ban đầu để bắt đầu vẽ bảng xếp hạng.
- Duyệt qua từng mục trong bảng xếp hạng và vẽ tên người chơi, thời gian hoàn thành, và số bước.
- Mỗi mục được vẽ trên một dòng mới, với khoảng cách y tăng dần.

6.11.5 Hàm xuly_ranking

Hàm xuly_ranking là hàm chính để xử lý giao diện và sự kiện liên quan đến bảng xếp hạng. Cách hoạt động:

- Tải bảng xếp hạng bằng cách gọi load_ranking.
- Hiển thị nền bảng xếp hạng bằng cách tải và vẽ hình ảnh nền.
- Sử dụng vòng lặp sự kiện của Pygame để xử lý các sự kiện như nhấn phím và nhấn chuột.
- Khi người dùng nhấn phím trái hoặc phải, lựa chọn bảng xếp hạng tương ứng với kích thước mè cung sẽ được thay đổi.
- Khi người dùng nhấn phím ESC hoặc nhấn chuột vào nút quay lại, hàm sẽ thoát và trở về màn hình trước đó.
- Cập nhật và hiển thị lại bảng xếp hạng mỗi khi có sự thay đổi.

6.11.6 Kết luận

Đoạn mã này cung cấp các hàm và phương thức cần thiết để xử lý và hiển thị bảng xếp hạng trong trò chơi. Sử dụng Pygame, đoạn mã tạo ra một giao diện trực quan và thân thiện, cho phép người chơi dễ dàng xem và cập nhật bảng xếp hạng. Điều này giúp cải thiện trải nghiệm người dùng và tạo động lực cho người chơi để cải thiện kết quả của họ.

```
1 IMPORT login_forms
2 IMPORT menu
3 IMPORT global_variable
4 IMPORT json
5 IMPORT pygame
6 IMPORT sys
7
8 CALL pygame.display.init()
9 CALL pygame.font.init()
10 SET FONT2 TO pygame.font.Font('8-BIT WONDER.ttf', 32)
11 SET FONT TO pygame.font.SysFont(None, 30)
12
13 FUNCTION get_rank(elapsed_time, steps, size)
14     SET index TO 0
15     IF size == 20 THEN
16         SET size TO 0
17     ELSE IF size == 40 THEN
18         SET size TO 1
19     ELSE
20         SET size TO 2
21     OPEN 'ranking.json' AS f
22     SET rank TO json.load(f)
23     IF login_forms.User_name IN rank[size] AND elapsed_time <= rank[size][login_forms
     .User_name][0] THEN
24         SET rank[size][login_forms.User_name] TO [elapsed_time, steps]
25         SORT rank[size] BY elapsed_time ASCENDING
26     ELSE IF login_forms.User_name NOT IN rank[size] THEN
27         SET rank[size][login_forms.User_name] TO [elapsed_time, steps]
28         SORT rank[size] BY elapsed_time ASCENDING
29     OPEN 'ranking.json' AS out FOR WRITE
30     CALL json.dump(rank, out, indent=4)
31 END FUNCTION
32
33 FUNCTION load_ranking()
34     OPEN 'ranking.json' AS f
35     SET rank TO json.load(f)
36     RETURN (rank[0].copy(), rank[1].copy(), rank[2].copy())
37 END FUNCTION
38
39 FUNCTION draw_ranking(screen, ranking)
40     SET y_offset TO HEIGHT // 3
41     FOR i, (username, data) IN ENUMERATE(ranking.items())
```

```

42     SET text_surface TO FONT.render(f"{i+1}. {username}: {data[0]}s {data[1]} steps",
43     True, White)
44     CALL screen.blit(text_surface, (WIDTH // 2 - text_surface.get_width() // 2,
45     y_offset))
46     SET y_offset TO y_offset + 40
47 END FUNCTION
48
47 FUNCTION xuly_ranking()
48   GLOBAL screen
49   SET levelsRanking TO ["20x20", "40x40", "100x100"]
50   SET soluongoptionRanking TO 3
51   SET current_option_ranking TO 0
52   SET clock TO pygame.time.Clock()
53   SET running TO True
54   SET ranking TO load_ranking()
55   SET learboard TO pygame.image.load("ranking.png")
56   SET learboard TO pygame.transform.scale(learboard, (WIDTH, HEIGHT))
57   SET idx TO 0
58   WHILE running
59     CALL screen.fill(White)
60     CALL screen.blit(learboard, (0, 0))
61     CALL draw_ranking(screen, ranking[current_option_ranking])
62     SET text_surface TO FONT2.render(levelsRanking[current_option_ranking], True,
63     Black)
63     IF current_option_ranking == 2 THEN
64       SET idx TO 0
65     ELSE
66       SET idx TO 10
67     CALL screen.blit(text_surface, (WIDTH // 2 - 90 + idx, HEIGHT - 80))
68     CALL pygame.display.flip()
69     CALL pygame.display.update()
70     FOR event IN pygame.event.get()
71       IF event.type == pygame.KEYDOWN THEN
72         IF event.type == pygame.QUIT THEN
73           CALL pygame.quit()
74           CALL sys.exit()
75         ELSE IF event.key == pygame.K_RIGHT THEN
76           SET current_option_ranking TO (current_option_ranking + 1) %
77           soluongoptionRanking
77         ELSE IF event.key == pygame.K_LEFT THEN
78           SET current_option_ranking TO (current_option_ranking - 1) %
78           soluongoptionRanking
79         ELSE IF event.key == pygame.K_ESCAPE THEN
80           SET running TO False
81           PRINT "return"
82           RETURN
83         ELSE IF event.type == pygame.MOUSEBUTTONDOWN THEN
84           IF event.button == 1 THEN
85             SET mouse_pos TO pygame.mouse.get_pos()
86             SET backbutton TO pygame.Rect(0, 0, rendered_text.get_width() + 100,
86               rendered_text.get_height() + 100)
87             IF backbutton.collidepoint(mouse_pos) THEN
88               RETURN
89 END FUNCTION

```

Listing 6.11: Mã giả cho các hàm xử lý Ranking

6.12 Giao Diện About

Âm Thanh/Hình Nền

7.1 Âm Thanh

Toàn bộ các chức năng và hàm liên quan đến sound được xử lý trong file sound.py

```
1 # Import necessary modules and files
2 from class_file import *
3 from global_variable import *
4
5 # Initialize and set up sound
6 pygame.mixer.init()
7 mixer.music.load('Menu_sound.mp3')
8 mixer.music.play(-1)
9
10 # Define sound effects
11 Win_sound = mixer.Sound('WIN.wav')
12 transition_sound = mixer.Sound('transition.wav')
13
14 # Start of the main game loop
15 # Initialize global variables for sound status and menu state
16 sound_enabled = True
17 inMenu = True
```

Listing 7.1: Mã giả "Import các module cần thiết, khởi tạo và cài đặt âm thanh"

- Import các định nghĩa từ các file class_file.py và global_variable.py để sử dụng trong file sound.py.
- Pygame.mixer.init() được gọi để khởi tạo module mixer của Pygame để sử dụng âm thanh.
- mixer.music.load('Menu_sound.mp3') nạp tệp 'Menu_sound.mp3' vào trình phát nhạc của Pygame.
- mixer.music.play(-1) phát lại âm thanh 'Menu_sound.mp3' lặp lại vô hạn.
- Win_sound và transition_sound được gán bằng các đối tượng âm thanh từ các tệp âm thanh tương ứng.
- Biến sound_enabled và inMenu được sử dụng để lưu trữ trạng thái của âm thanh và trạng thái hiện tại của trò chơi (đang ở trong menu hay không).

```
1 Function toggle_sound():
2     Global sound_enabled
3     sound_enabled = not sound_enabled
4     Print(sound_enabled)
5     If sound_enabled Then
6         optionsettings[4] = "SoundOn"
7         If inMenu Then
8             pygame.mixer.unpause() # Enable sound
9             mixer.music.load('Menu_sound.mp3')
10            mixer.music.play(-1)
11        Else
12            pygame.mixer.unpause() # Enable sound
13            mixer.music.load('Background.ogg')
14            mixer.music.play(-1)
15        Else
16            optionsettings[4] = "Soundoff"
17            mixer.music.stop()
18            pygame.mixer.pause()
19
20 Function playSound(n: int):
21     sound_files = ["Background.ogg", "Menu_sound.mp3", "Intro.mp3", "WIN.wav", "move
22 .mp3", "lose.mp3"]
23     sound = pygame.mixer.Sound(sound_files[n])
24     Global sound_enabled
25     If sound_enabled Then
26         If n != 3 And n != 5 Then
```

```

26         sound.play()
27     Else
28
29         mixer.music.stop()
30         sound.play()

```

Listing 7.2: Mã giả "Hai hàm sound chính"

Hàm `toggle_sound()`:

- Biến global và trạng thái âm thanh:
 - Biến global `sound_enabled` được sử dụng để lưu trữ trạng thái hiện tại của âm thanh (đã bật hay đã tắt).
 - Ban đầu, biến `sound_enabled` được thiết lập thành `True`, cho biết rằng âm thanh đang được bật.
- Đảo ngược trạng thái âm thanh:
 - Dòng mã `sound_enabled = not sound_enabled` đảo ngược trạng thái hiện tại của biến `sound_enabled`.
 - Ví dụ: Nếu âm thanh đang được bật (`sound_enabled = True`), sau dòng này nó sẽ được chuyển thành tắt (`sound_enabled = False`).
- Cập nhật trạng thái và cài đặt âm thanh:
 - Sau khi đảo ngược trạng thái, mã kiểm tra trạng thái mới của `sound_enabled`:
 - * Nếu âm thanh được bật, các cài đặt âm thanh được thiết lập để phát lại âm thanh menu hoặc nền tùy thuộc vào trạng thái hiện tại của `inMenu`.
 - * Nếu âm thanh đã tắt, phát lại âm thanh sẽ bị ngừng lại và `mixer` sẽ được tạm dừng.

Hàm `playSound(n: int)`:

- Chọn âm thanh từ danh sách:
 - Hàm này nhận vào một chỉ số `n`, đại diện cho âm thanh cần phát từ danh sách `sound_files`.
- Tạo đối tượng âm thanh và phát lại:
 - Dòng mã `sound = pygame.mixer.Sound(sound_files[n])` tạo một đối tượng âm thanh từ tệp âm thanh tương ứng với chỉ số `n`.
 - Nếu âm thanh đã được bật (qua kiểm tra của biến `sound_enabled`), âm thanh được phát lại:
 - * Nếu âm thanh không phải là WIN hoặc lose (chỉ số 3 và 5), sử dụng `sound.play()` để phát lại âm thanh.
 - * Nếu âm thanh là WIN hoặc lose, `mixer` sẽ dừng phát lại âm thanh hiện tại (nếu có) trước khi phát âm thanh mới. Điều này đảm bảo rằng chỉ có một âm thanh được phát vào một thời điểm nhất định.