

# BÁO CÁO PROJECT MÔN CƠ SỞ LẬP TRÌNH: XÂY DỰNG TRÒ CHƠI MUMMY MAZE

NHÓM We Are One - 25TNT1

Ngày 11 tháng 1 năm 2026

# Mục lục

<b>1 Thông tin nhóm - Phân công nhiệm vụ</b>	<b>3</b>
<b>2 Các thuật toán đã tìm hiểu và triển khai</b>	<b>4</b>
2.1 Thuật toán sinh mê cung ngẫu nhiên . . . . .	4
2.1.1 Ý tưởng cốt lõi . . . . .	4
2.1.2 Mã giả (Pseudocode) . . . . .	5
2.1.3 Ví dụ . . . . .	6
2.2 Thuật toán di chuyển của Xác ướp (Mức độ Trung bình) . . . . .	7
2.2.1 Ý tưởng cốt lõi . . . . .	7
2.2.2 Mã giả (Pseudocode) . . . . .	8
2.2.3 Ví dụ . . . . .	8
2.3 Thuật toán di chuyển của Xác ướp (Mức độ Khó) . . . . .	10
2.3.1 Ý tưởng cốt lõi . . . . .	10
2.3.2 Mã giả (Pseudocode) . . . . .	11
2.3.3 Ví dụ . . . . .	12
2.4 Thuật toán đảm bảo tồn tại ít nhất một cách thăng . . . . .	14
2.4.1 Ý tưởng cốt lõi . . . . .	14
2.4.2 Mã giả (Pseudocode) . . . . .	15
2.4.3 Ví dụ . . . . .	16
<b>3 Kiến trúc Chi tiết của Game</b>	<b>17</b>
3.1 Giao diện Người dùng (User Interface) . . . . .	17
3.2 Cơ chế và Chức năng của Game . . . . .	17
3.3 Cơ chế Nâng cao . . . . .	18
3.3.1 Nâng cao độ khó của trò chơi . . . . .	18
3.3.2 Cải thiện tính thực tế . . . . .	18
3.3.3 Di chuyển nhân vật bằng phím WASD hoặc chuột . . . . .	18
<b>4 Quá trình phát triển game</b>	<b>19</b>
4.1 Khởi tạo hệ thống và Quản lý trạng thái va chạm vật lý . . . . .	19
4.1.1 Thiết kế bản đồ với kỹ thuật Bitmask (File <code>mecung.py</code> ) . . . . .	19
4.1.2 Đối tượng Nhân vật và Cơ chế Di chuyển (File <code>nhanvat.py</code> ) . . . . .	19
4.2 Xây dựng Cấu trúc dữ liệu nâng cao và Thuật toán . . . . .	20
4.2.1 Tự xây dựng Cấu trúc dữ liệu . . . . .	20
4.2.2 Hệ thống Da hình trong Thuật toán (File <code>thuattoan.py</code> ) . . . . .	20
4.2.3 Phát triển Mê cung Mặc định và Hệ thống Tạo Mê cung Ngẫu nhiên	21
4.3 Quá trình phát triển Giao diện Đồ họa và Hệ thống Âm thanh . . . . .	21
4.3.1 Tìm hiểu công nghệ và Yêu cầu . . . . .	21
4.3.2 Tìm kiếm và Xử lý tài nguyên . . . . .	21
4.3.3 Cài đặt và Hiện thực hóa . . . . .	22
<b>5 Hạn chế và định hướng phát triển</b>	<b>23</b>

# 1 Thông tin nhóm - Phân công nhiệm vụ

STT	Thành viên	Nhiệm vụ chi tiết	Tiến độ
1	<b>Nguyễn Mai Khôi 25122025</b>	<ul style="list-style-type: none"><li>- Trưởng nhóm, phân công nhiệm vụ.</li><li>- Xây dựng hệ thống game, cài đặt các đối tượng (OOP) trong file: <i>mecung.py</i>, <i>nhanvat.py</i>, <i>thuattoan.py</i>, <i>dsu.py</i>.</li><li>- Xây dựng logic game: va chạm, trạng thái sống, chìa khoá, bẫy...</li><li>- Thiết kế thuật toán Xác ướp (Khó).</li><li>- Thuật toán sinh mê cung ngẫu nhiên.</li><li>- Viết báo cáo.</li></ul>	100%
2	<b>Nguyễn Anh Huy 25122023</b>	<ul style="list-style-type: none"><li>- Thiết kế giao diện (UI/UX).</li><li>- Thiết kế thuật toán Xác ướp (Dẽ - Trung bình).</li><li>- Tính năng Đăng nhập/Đăng ký.</li><li>- Tính năng Undo và Reset.</li><li>- Tính năng Save và Load game.</li><li>- Quay video demo.</li><li>- Viết báo cáo.</li></ul>	100%
3	<b>Nguyễn Xuân Hoàng 25122021</b>	<ul style="list-style-type: none"><li>- Thiết kế 10 mê cung mặc định (Level Design).</li><li>- Xử lý di chuyển phím: <math>\leftarrow</math>, <math>\rightarrow</math>, <math>\uparrow</math>, <math>\downarrow</math>.</li><li>- Xử lý điều khiển bằng chuột.</li><li>- Quay video demo.</li><li>- Viết báo cáo.</li></ul>	100%
4	<b>Nguyễn Chí Bảo 25122018</b>	<ul style="list-style-type: none"><li>- Thiết kế 10 mê cung mặc định.</li><li>- Thiết kế đồ họa cho game.</li><li>- Xử lý âm thanh: Nhạc nền, tiếng bước chân, game over, victory...</li><li>- Quay video demo.</li><li>- Kiểm tra lại game và các tính năng trong game.</li><li>- Viết báo cáo.</li></ul>	100%

Bảng 1: Bảng phân công nhiệm vụ và kết quả thực hiện

## 2 Các thuật toán đã tìm hiểu và triển khai

### 2.1 Thuật toán sinh mê cung ngẫu nhiên

#### 2.1.1 Ý tưởng cốt lõi

Ta xem mỗi góc của mỗi ô vuông sẽ là 1 đỉnh. Khi đó ta xây dựng mê cung (xây tường) bằng cách nối các cạnh của 2 đỉnh kề nhau (mỗi đỉnh sẽ có nhiều nhất 4 đỉnh kề nhau). Quy tắc để xây dựng 1 mê cung là các đỉnh ấy khi nối lại không tạo thành chu trình, ngoại trừ phần bao bên ngoài của mê cung.

Tại mỗi ô ta sẽ random giá trị từ  $0 \rightarrow 15$  hay  $0 \rightarrow (2^4 - 1)$ , nghĩa là mỗi ô sẽ có giá trị random của 4 bit tương ứng với 4 cạnh của một hình vuông.

Để giải quyết vấn đề chu trình, ta sử dụng cấu trúc dữ liệu Disjoint Set Union (DSU). Ở DSU, mỗi đỉnh sẽ lưu lại 1 đỉnh đại diện cho thành phần liên thông chứa nó. Khi tồn tại một cạnh từ  $x$  đến  $y$ , nếu  $x$  và  $y$  khác thành phần liên thông ta sẽ nối 1 cạnh giữa  $x$  và  $y$  và hợp 2 thành phần liên thông lại. Ngược lại, nếu  $x$  và  $y$  đã nằm trong 1 thành phần liên thông, khi nối cạnh lại sẽ tạo ra chu trình nên ta sẽ loại bỏ cạnh đó.

Chú ý nhỏ: Tuy vậy do giới hạn tường không vượt quá 50% diện tích nên mê cung sẽ bị lệch về phía trên khi ta xét theo thứ tự từ trên xuống. Để xử lý được phần nào, ở mỗi hàng ta sẽ chia đôi số cột và mỗi phần có số tường không quá 25% diện tích của mỗi hàng.

## 2.1.2 Mã giả (Pseudocode)

---

**Input:** Kích thước mê cung  $N$   
**Output:** Ma trận mê cung  $M$ , Vị trí cửa *Door*

```

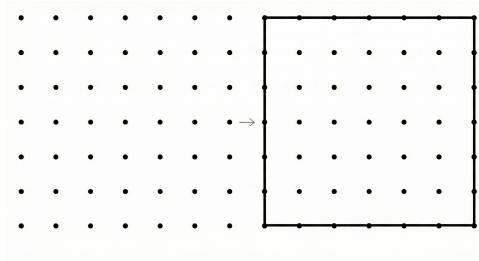
1  $M \leftarrow$  Ma trận  $N \times N$  (tất cả số 0)
2  $DSU \leftarrow$  Khởi tạo cấu trúc Disjoint Set cho các cạnh
   /* 1. Tạo tường bao quanh mê cung (Biên) */ 
3 for  $i \leftarrow 0$  to  $N - 1$  do
4   Thêm tường trên cho  $M[0][i]$  và cập nhật DSU
5   Thêm tường dưới cho  $M[N - 1][i]$  và cập nhật DSU
6   Thêm tường trái cho  $M[i][0]$  và cập nhật DSU
7   Thêm tường phải cho  $M[i][N - 1]$  và cập nhật DSU
8 end
   /* 2. Xử lý logic sinh tường và kiểm tra chu trình */ 
9 foreach ô  $(x, y)$  trong  $M$  do
10  |  $val \leftarrow$  Random(0, 15)                                // Sinh mask ngẫu nhiên
11  |  $limit \leftarrow 0$ 
12  | foreach hướng  $k \in \{0, 1, 2, 3\}$  do
13  |   | if bit  $k$  đang bật trong  $val$  then
14  |   |   |  $u, v \leftarrow$  Hai đỉnh cạnh tương ứng trong DSU
15  |   |   | if  $DSU.Union(u, v) == False$  or  $limit \geq N/4$  then
16  |   |   |   |  $val \leftarrow val - 2^k$                          // Bỏ tường nếu tạo chu trình
17  |   |   |   | else
18  |   |   |   |  $limit \leftarrow limit + 1$ 
19  |   |   | end
20  |   | end
21  | end
22  |  $M[x][y] \leftarrow M[x][y] \vee val$                       // Cập nhật ô hiện tại
23 end
   /* 3. Đồng bộ hóa tường giữa các ô kề nhau */ 
24 foreach ô  $(x, y)$  trong  $M$  do
25  | if  $M[x][y]$  có tường TRÊN ( $x > 0$ ) then
26  |   |  $M[x - 1][y]$  thêm tường DƯỚI
27  | end
28  | if  $M[x][y]$  có tường DƯỚI ( $x < N - 1$ ) then
29  |   |  $M[x + 1][y]$  thêm tường TRÊN
30  | end
31  | if  $M[x][y]$  có tường TRÁI ( $y > 0$ ) then
32  |   |  $M[x][y - 1]$  thêm tường PHẢI
33  | end
34  | if  $M[x][y]$  có tường PHẢI ( $y < N - 1$ ) then
35  |   |  $M[x][y + 1]$  thêm tường TRÁI
36  | end
37 end
   /* 4. Tạo cửa ra ngẫu nhiên ở biên */ 
38  $dir \leftarrow$  Random(0, 3)
39  $pos \leftarrow$  Random(0,  $N - 1$ )
40  $Door \leftarrow$  Xác định tọa độ dựa trên  $dir$  và  $pos$ 
41 return  $M, Door$ 

```

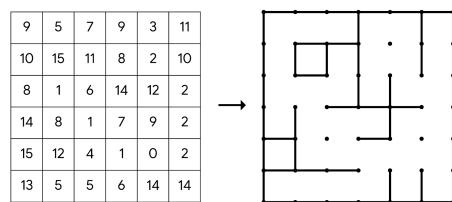
---

### 2.1.3 Ví dụ

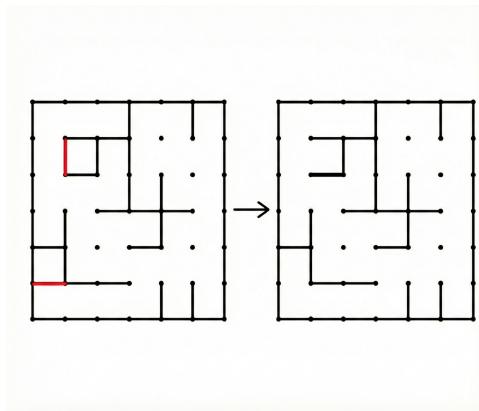
Ví dụ với việc tạo ra một mè cung  $6 \times 6$ , trước tiên là việc nối tất cả các cạnh bên ngoài lại



Tiếp theo ta tạo một số cạnh ngẫu nhiên (random trạng thái của mỗi ô vuông).



Sau đó ta sẽ xét những cạnh tạo ra chu trình và xoá chúng. Như hình trên nếu ta xét cạnh từ trên xuống dưới, từ trái qua phải và các hướng theo chiều kim đồng hồ thì cạnh tạo chu trình sẽ là cạnh bên trái của ô  $(1,1)$  và cạnh bên dưới của ô  $(5,1)$ .



Vậy ta đã được mè cung ngẫu nhiên không tạo ra chu trình.

## 2.2 Thuật toán di chuyển của Xác ướp (Mức độ Trung bình)

### 2.2.1 Ý tưởng cốt lõi

Ở thuật toán này, Xác ướp sẽ di chuyển tham lam theo hướng nào làm cho khoảng cách Manhattan giữa Xác ướp và Nhà thám hiểm là nhỏ nhất.

Khoảng cách Manhattan của 2 toạ độ  $(x_1, y_1)$  và  $(x_2, y_2)$  là  $|x_1 - x_2| + |y_1 - y_2|$ .

## 2.2.2 Mã giả (Pseudocode)

---

**Input:** Vị trí Nhà thám hiểm ( $P_x, P_y$ ), Vị trí Xác ướp ( $M_x, M_y$ )  
**Output:** Hướng di chuyển tốt nhất  $dir \in \{0, 1, 2, 3\}$

/\* Hàm phụ: Tính khoảng cách Manhattan dự kiến \*/

```

1 Function Distance(Mummyy, dir):
2     (x, y)  $\leftarrow$  Toạ độ hiện tại của Mummyy
3     switch dir do
4         case 0 do
5             | x  $\leftarrow$  x - 1 (Lên)
6             end
7             case 1 do
8                 | y  $\leftarrow$  y + 1 (Phải)
9                 end
10            case 2 do
11                | x  $\leftarrow$  x + 1 (Xuống)
12                end
13            case 3 do
14                | y  $\leftarrow$  y - 1 (Trái)
15                end
16        end
17        return  $|x - P_x| + |y - P_y|$ 
18    /* Hàm chính: Tìm nước đi tối ưu (Greedy) */
```

18 **Function** FindBestMove(*Mummyy*):

```

19     best_dir  $\leftarrow$  -1
20     min_dist  $\leftarrow \infty$ 
21     foreach dir  $\in \{0, 1, 2, 3\} do
22         if Mummyy có thể di chuyển hướng dir then
23             | d  $\leftarrow$  Distance(Mummyy, dir)
24             if d < min_dist then
25                 | min_dist  $\leftarrow$  d
26                 | best_dir  $\leftarrow$  dir
27             end
28         end
29     end
30     return best_dir
31    /* Triển khai thực tế */$ 
```

31 **if** Số lượng xác ướp == 1 **then**

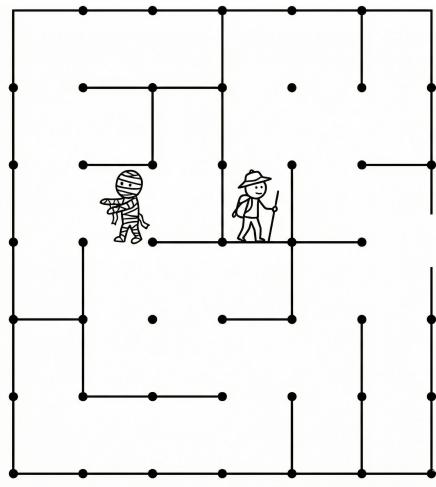
```

32     | return FindBestMove(Xác ướp 1)
33 else
34     | h1  $\leftarrow$  FindBestMove(Xác ướp 1)
35     | h2  $\leftarrow$  FindBestMove(Xác ướp 2)
36     | return (h1, h2)
37 end
```

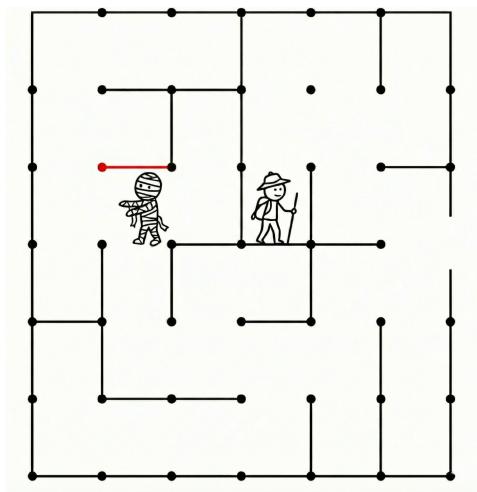
---

## 2.2.3 Ví dụ

Ta sẽ xem xét bản đồ sau:

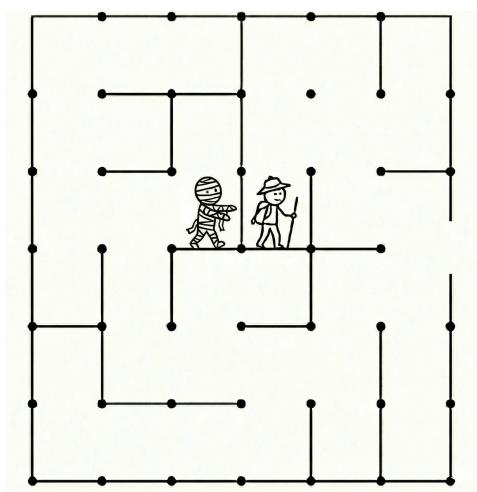


Nếu Xác ướp di chuyển theo hướng 0 (đi lên trên):

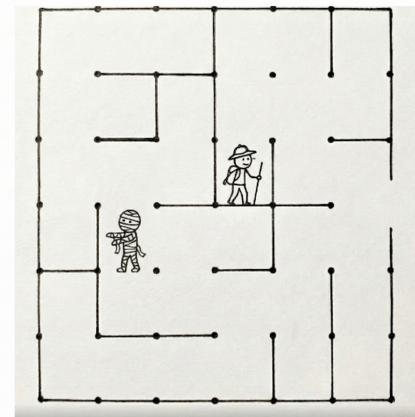


Di chuyển không hợp lệ vì đã có tường chắn.

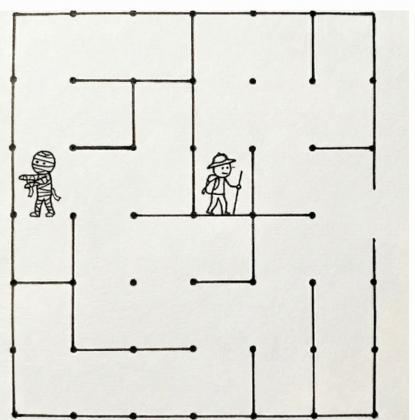
Nếu Xác ướp di chuyển theo hướng 1 (đi sang phải):



Nếu Xác ướp di chuyển theo hướng 2 (đi xuống dưới):



Nếu Xác ướp di chuyển theo hướng 3 (đi sang trái):



Tại đây ta thấy ở bước đi sang phải sẽ tối ưu nhất.

## 2.3 Thuật toán di chuyển của Xác ướp (Mức độ Khó)

### 2.3.1 Ý tưởng cốt lõi

- Ở thuật toán này, Xác ướp sẽ di chuyển tham lam theo hướng nào làm cho đường đi ngắn nhất giữa Xác ướp và Nhà thám hiểm là nhỏ nhất.
- Để giải quyết vấn đề đường đi ngắn nhất ta sử dụng thuật toán BFS được triển khai như sau:

1. Tạo 1 Queue được cài đặt bằng Linked List.
2. Tạo 1 Set để lưu lại những tọa độ nào đã từng được xét.
3. Đưa vị trí của Nhà thám hiểm vào Queue đã tạo.

4. Lấy phần tử đầu tiên của Queue ra. Xét các toạ độ kề cạnh với phần tử đó. Nếu toạ độ đó là một trong những toạ độ kề cạnh với Xác ướp và Xác ướp có thể đi được theo hướng đó thì đó là hướng mà Xác ướp sẽ đi. Nếu toạ độ đó chưa thuộc Set thì ta sẽ thêm vào Queue và Set.

5. Quay lại bước 4 đến khi Queue rỗng hoặc đã tìm được hướng mà Xác ướp nên đi.

### 2.3.2 Mã giả (Pseudocode)

---

**Input:** Trạng thái Nhà thám hiểm  $P$ , Xác ướp  $M_1$  (và  $M_2$  nếu có)

**Output:** Hướng di chuyển tối ưu cho Xác ướp

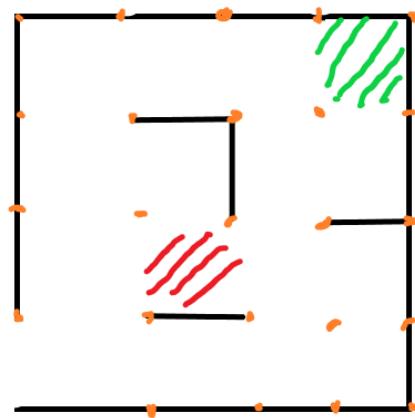
```

1 Queue  $\leftarrow \{P\}$  // Hàng đợi chứa trạng thái Nhà thám hiểm
2 Visited  $\leftarrow \{P.\text{pos}\}$ 
3 Dir1  $\leftarrow -1$ , Dir2  $\leftarrow -1$ 
4 while Queue  $\neq \emptyset$  do
5   u  $\leftarrow \text{Queue.pop}()$  // Lấy trạng thái Nhà thám hiểm hiện tại
6   foreach hướng k  $\in \{0, 1, 2, 3\}$  do
7     /* Kiểm tra nếu Xác ướp có thể chặn đầu tại vị trí u */
8     if  $M_1$  đi được hướng k and  $M_1.\text{next\_pos}(k) == u.\text{pos}$  then
9       if Dir1 == -1 then
10      | Dir1  $\leftarrow k$ 
11    end
12  end
13  if Có  $M_2$  and  $M_2$  đi được hướng k and  $M_2.\text{next\_pos}(k) == u.\text{pos}$  then
14    if Dir2 == -1 then
15      | Dir2  $\leftarrow k$ 
16    end
17  /* Loang BFS tiếp các bước đi của Nhà thám hiểm */
18  if u đi được hướng k and u.next_pos(k)  $\notin$  Visited then
19    | v  $\leftarrow$  Trạng thái mới của Nhà thám hiểm sau khi đi hướng k
20    | Queue.push(v)
21    | Visited.add(v.pos)
22  end
23  if ( $M_1$  có hướng hoặc xong) and ( $M_2$  có hướng hoặc xong) then
24    | break // Đã tìm thấy điểm chặn đầu gần nhất
25  end
26 end
27 if Số lượng xác ướp == 1 then
28   | return Dir1
29 else
30   | return (Dir1, Dir2)
31 end
```

---

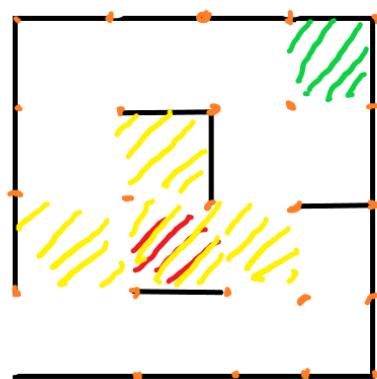
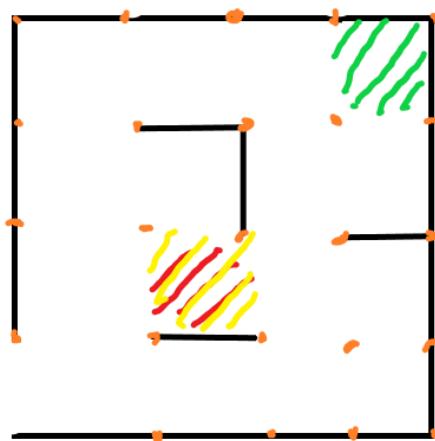
### 2.3.3 Ví dụ

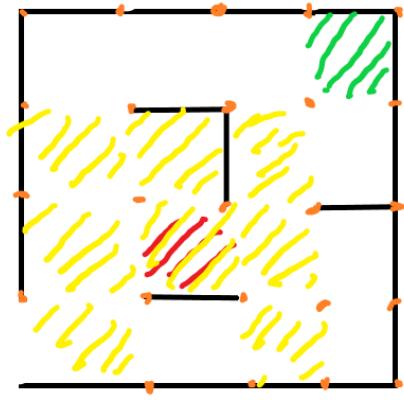
Ta xét một mê cung  $4 \times 4$  như sau:



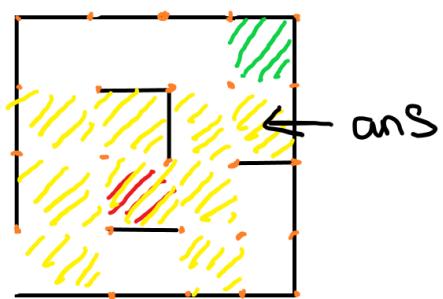
Hình 1: Ô xanh là Xác ướp, đỏ là Nhà thám hiểm

Ta bắt đầu loang từ Nhà thám hiểm ( $\hat{O}$  đỏ) theo các lớp như sau:





Khi loang đến trạng thái này ta nhận thấy ô (2,4) là ô đầu tiên liền kề với Xác ướp mà không có tường chắn nên Xác ướp đi xuống sẽ tối ưu nhất.



## 2.4 Thuật toán đắm bảo tồn tại ít nhất một cách thắng

### 2.4.1 Ý tưởng cốt lõi

Thuật toán này xoay quanh Quy Hoạch Động. Ta sẽ lưu 5 trạng thái  $x, y, x_1, y_1, step$ ; trong đó  $(x, y)$  là toạ độ của Nhà thám hiểm,  $(x_1, y_1)$  là toạ độ của Xác ướp,  $step$  sẽ lưu 3 trạng thái với 0 là trạng thái khi đến lượt Nhà thám hiểm thực hiện di chuyển còn 1 và 2 là trạng thái đến lượt Xác ướp thực hiện di chuyển.

Mỗi trạng thái ta sẽ lưu lại kết quả tại lượt chơi của nhân vật đó nếu nhân vật đó chơi tối ưu thì ai sẽ là người chiến thắng (1: Nhà thám hiểm thắng, 2: Xác ướp thắng).

Tuy nhiên để thực hiện được Quy Hoạch Động ta cần phải có thứ tự lần lượt để cập nhật các trạng thái. Và ta nhận ra thứ tự trạng thái ở vấn đề này là thứ tự BFS (loang) nên ta sẽ kết hợp Quy Hoạch Động (DP) và Loang (BFS).

Thuật toán được triển khai như sau:

1. Tạo mảng 4 chiều  $dp[x][y][x_1][y_1][step]$ , lưu lại như đã nói ở trên. Ban đầu tất cả giá trị của mảng dp sẽ là 0 (nghĩa là chưa xét).
2. Ta khởi tạo trạng thái ban đầu: Khi vị trí Nhà thám hiểm và Xác ướp trùng nhau thì chắc chắn Xác ướp sẽ thắng. Khi Nhà thám hiểm đứng kế bên cửa và đến lượt di chuyển của Nhà thám hiểm thì Nhà thám hiểm sẽ thắng.
3. Ta tao Queue được cài đặt bằng Linked List để BFS các trạng thái và bỏ các trạng thái ban đầu vào Queue.
4. Lấy trạng thái đầu tiên bỏ ra Queue.
5. Ta xét trạng thái vừa lấy ra. Xét lượt đi trước step đó  $((step + 2)\%3)$ . Ta sẽ cập nhật trạng thái quy hoạch động của lượt đi này (cập nhật lại vị trí của Nhà thám hiểm hoặc Xác ướp tùy thuộc vào lượt đi của ai). Với lượt đi của Nhà thám hiểm ta sẽ cố gắng đạt giá trị 1, còn với Xác ướp ta sẽ cố gắng để giá trị đạt 2.
6. Nếu giá trị ta vừa cập nhật có sự thay đổi về giá trị dp thì ta sẽ bỏ vô Queue.
7. Quay lại bước 4 và tiếp tục đến khi Queue rỗng.

## 2.4.2 Mã giả (Pseudocode)

---

**Input:** Kích thước  $N$ , Vị trí cửa  $Door$

**Output:** Bảng  $DP$  chứa trạng thái thăng/thua của mọi trường hợp

```

/* 1. Khởi tạo và Thiết lập trạng thái cơ sở */
```

- 1  $DP[N][N][N][N][3] \leftarrow$  Toàn bộ bằng 0 (Chưa xác định)
- 2  $Queue \leftarrow \emptyset$
- 3 **foreach** ô  $(x, y)$  trong Mê cung **do**
- 4 | **if**  $(x, y) \neq Door$  **then**
- 5 | |  $DP[Door_x][Door_y][x][y][0] \leftarrow 1$  // Nhà thám hiểm thăng (tại cửa)
- 6 | |  $Queue.push(Door_x, Door_y, x, y, 0)$
- 7 | **end**
- 8 | **foreach** bước  $k \in \{0, 1, 2\}$  **do**
- 9 | |  $DP[x][y][x][y][k] \leftarrow 2$  // Xác ướp thăng (bắt được)
- 10 | |  $Queue.push(x, y, x, y, k)$
- 11 | **end**
- 12 **end**

```

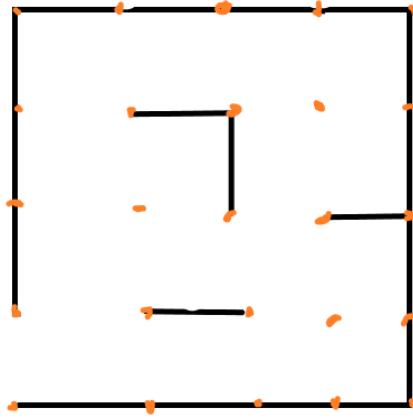
/* 2. Duyệt BFS ngược từ trạng thái kết thúc */
```

- 13 **while**  $Queue \neq \emptyset$  **do**
- 14 |  $(U_x, U_y, M_x, M_y, step) \leftarrow Queue.pop()$
- 15 |  $prev\_step \leftarrow (step + 2) \pmod 3$  // Lùi lại 1 lượt chơi
- 16 | **if**  $prev\_step == 0$  (Lượt Nhà thám hiểm) **then**
- 17 | | **foreach** hướng  $d \in \{0..3\}$  **do**
- 18 | |  $(U'_x, U'_y) \leftarrow$  Vị trí trước đó của Nhà thám hiểm theo hướng  $d$
- 19 | | **if**  $D_i$  chuyển hợp lệ **and** Cần cập nhật  $DP$  **then**
- 20 | |  $DP[U'_x][U'_y][M_x][M_y][prev\_step] \leftarrow$  Cập nhật trạng thái thăng/thua
- 21 | |  $Queue.push(U'_x, U'_y, M_x, M_y, prev\_step)$
- 22 | | **end**
- 23 | | **end**
- 24 | **else**
- 25 | | /\* Lượt của Xác ướp (Step 1 hoặc 2) \*/
- 26 | | **foreach** hướng  $d \in \{0..3\}$  **do**
- 27 | |  $(M'_x, M'_y) \leftarrow$  Vị trí trước đó của Xác ướp theo hướng  $d$
- 28 | | **if**  $D_i$  chuyển hợp lệ **and** Cần cập nhật  $DP$  **then**
- 29 | |  $DP[U_x][U_y][M'_x][M'_y][prev\_step] \leftarrow$  Cập nhật trạng thái thăng/thua
- 30 | |  $Queue.push(U_x, U_y, M'_x, M'_y, prev\_step)$
- 31 | | **end**
- 32 | | **end**
- 33 | **end**
- 34 **return**  $DP$

---

### 2.4.3 Ví dụ

Ta xét mê cung  $4 \times 4$  như sau:



Giá trị khởi tạo ban đầu sẽ là:

- $dp[3][0][x][y][0] = 1, \forall x, y : x \neq 3, y \neq 0$
- $dp[x][y][x][y][z] = 2, \forall x, y, z : z < 3$

Ta bắt đầu loang trạng thái quy hoạch động, lấy một ví dụ để minh họa:

- Khi ta lấy ra khỏi queue trạng thái  $dp[1][1][0][0][1] = val$ . Khi đó step ta xét sẽ là  $(1 + 2)\%3 = 0$  (Lùi về lượt Nhà thám hiểm), nghĩa là ta sẽ lấy giá trị của  $val$  cập nhật cho trạng thái  $dp[x_{new}][y_{new}][0][0][0]$  với  $(x_{new}, y_{new})$  là toạ độ có thể trước đó của Nhà thám hiểm. Trong mê cung này sẽ là ô  $(1,0)$  và  $(2,1)$ .
- Khi giá trị của  $dp[x_{new}][y_{new}][0][0][0] == 1$  ta sẽ không làm gì vì tại step 0 đã đạt giá trị 1 (mục tiêu của step 0), nhưng nếu nó là giá trị 0 thì ta buộc phải gán  $dp[x_{new}][y_{new}][0][0][0] = val$ . Nếu giá trị của  $dp[x_{new}][y_{new}][0][0][0]$  thay đổi so với giá trị ban đầu ta sẽ đầy vô Queue và thực hiện tiếp tục.

## 3 Kiến trúc Chi tiết của Game

### 3.1 Giao diện Người dùng (User Interface)

Giao diện được thiết kế theo tiêu chí tối giản và thân thiện, giúp người chơi dễ dàng tiếp cận ngay từ lần đầu trải nghiệm.

- **Hệ thống Đăng nhập:**

- **Log in:** Dành cho người dùng đã có tài khoản, yêu cầu nhập chính xác thông tin để truy cập dữ liệu cũ.
- **Sign in/Register:** Cho phép người chơi mới khởi tạo tài khoản cá nhân.

- **Menu Chính:** Sau khi đăng nhập thành công, hệ thống điều hướng người chơi đến hai phân khu chức năng chính:

- **Play:** Bắt đầu cuộc thám hiểm.
- **Guide:** Hướng dẫn chi tiết về cách điều khiển và luật chơi.

- **Chế độ Chơi (Game Modes):**

- **Chế độ Chiến dịch (Campaign):** Gồm 20 cấp độ thử thách. Người chơi phải hoàn thành màn chơi hiện tại để mở khóa (unlock) màn tiếp theo.
- **Chế độ Mê cung Ngẫu nhiên (Sandbox):** Người chơi tự do khởi tạo môi trường bằng cách nhập kích thước  $N \times N$  nhưng chỉ giới hạn trong  $4 \leq N \leq 50$ . Ví dụ: Nếu nhập 10, mê cung tạo ra sẽ có kích thước  $10 \times 10$ .

### 3.2 Cơ chế và Chức năng của Game

#### Hệ thống Nhân vật và Kẻ địch:

- **Nhà thám hiểm (Explorer):** Nhân vật chính do người chơi điều khiển với mục tiêu tìm lối thoát.
- **Mummy (Xác ướp):** Kẻ địch AI có nhiệm vụ truy đuổi người chơi. Số lượng Mummy sẽ tỉ lệ thuận với độ khó của bản đồ.
- **Vật phẩm và Cạm bẫy:**

- **Key (Chìa khóa):** Vật phẩm bắt buộc để kích hoạt cửa thoát hiểm.
- **Chướng ngại vật:** Hệ thống hố bẫy và bom được bố trí ngẫu nhiên để tăng tính thử thách.

#### Chức năng Hệ thống:

- **Undo & Reset:** Hệ thống hỗ trợ tính năng **Undo (Hoàn tác)** và **Reset (Thiết lập lại)** nhằm tối ưu hóa trải nghiệm người dùng. Việc cho phép quay lại vị trí trước đó hoặc làm mới màn chơi giúp người chơi chủ động hơn trong việc thử sai, từ đó linh hoạt tìm kiếm lối đi tối ưu nhất trong các mê cung phức tạp.
- **Save & Load:** Hệ thống tự động lưu trữ trạng thái (vị trí, chỉ số, tiến trình) khi người chơi tạm dừng hoặc kết thúc phiên chơi, đảm bảo tính liên tục của trải nghiệm.
- **Quản lý Âm thanh:** Tính năng bật/tắt âm lượng tích hợp trong menu cài đặt, giúp người chơi tùy chỉnh theo môi trường và nhu cầu tập trung.

### 3.3 Cơ chế Nâng cao

#### 3.3.1 Nâng cao độ khó của trò chơi

Để tăng tính thử thách, độ khó sẽ được điều chỉnh tăng dần qua từng màn chơi:

- **Số lượng kẻ địch:** Số lượng Mummy tăng tỉ lệ thuận với cấp độ màn chơi.
- **Trí tuệ nhân tạo (AI):** Thuật toán tìm đường của Mummy được tối ưu hóa ở các màn cao, khiến chúng truy đuổi thông minh và khó lường hơn.

#### 3.3.2 Cải thiện tính thực tế

Trò chơi tích hợp hệ thống âm thanh đa dạng để tăng tính nhập vai:

- **Nhạc nền kiểu Ai Cập:** Âm thanh đặc trưng Ai Cập tăng tính huyền bí của mê cung.
- **Âm thanh môi trường và hành động:** Tiếng bước chân của nhà thám hiểm, tiếng gầm gừ của Mummy khi lại gần.
- **Âm thanh phản hồi:** Hiệu ứng âm thanh thông báo khi **Victory (Chiến thắng)** hoặc **Defeat (Thất bại)**.

#### 3.3.3 Di chuyển nhân vật bằng phím WASD hoặc chuột

Trong trò chơi, người chơi có thể di chuyển nhân vật chính bằng cách sử dụng các phím WASD hoặc click chuột vào các ô ngay cạnh nhân vật mà không bị chặn bởi tường.

## 4 Quá trình phát triển game

### 4.1 Khởi tạo hệ thống và Quản lý trạng thái va chạm vật lý

Trong giai đoạn đầu, nhóm tập trung thiết kế các đối tượng cơ bản cấu thành nên trò chơi.

#### 4.1.1 Thiết kế bản đồ với kỹ thuật Bitmask (File `mecung.py`)

Thay vì sử dụng các cấu trúc dữ liệu cồng kềnh để lưu trữ thông tin về 4 bức tường của một ô vuông, nhóm đã áp dụng kỹ thuật **\*\*Bitmask\*\*** (Thao tác trên bit) để thuận tiện trong việc lưu trữ

- **Cơ chế lưu trữ:** Ma trận mè cung `self.matrix` kích thước  $N \times N$  chỉ chứa các số nguyên từ 0 đến 15. Mỗi số nguyên này đại diện cho trạng thái của một ô vuông, trong đó 4 bit đầu tiên ( $2^0, 2^1, 2^2, 2^3$ ) tương ứng với 4 hướng: Trên, Phải, Dưới, Trái.

- **Quy ước Bit:**

- Bit 0 (Giá trị 1): Tường phía Trên.
- Bit 1 (Giá trị 2): Tường phía Phải.
- Bit 2 (Giá trị 4): Tường phía Dưới.
- Bit 3 (Giá trị 8): Tường phía Trái.

Ví dụ: Một ô có giá trị 9 ( $1001_2$ ) nghĩa là ô đó có tường ở phía Trái (8) và phía Trên (1).

- **Truy xuất trạng thái:** Phương thức `trang_thai(x, y, id)` sử dụng phép dịch bit (`>>`) và phép toán AND (`&`) để kiểm tra tường:

$$\text{is\_wall} = (\text{value} \gg \text{id}) \& 1$$

Nếu kết quả trả về 1, hướng đó có tường chắn; ngược lại là đường đi thông thoáng. Cách tiếp cận này giúp giảm độ phức tạp không gian xuống thấp nhất và tăng tốc độ kiểm tra va chạm lên mức tối đa ( $O(1)$ ).

#### 4.1.2 Đối tượng Nhân vật và Cơ chế Di chuyển (File `nhanvat.py`)

Class `Char` được thiết kế để đại diện chung cho mọi đối tượng trong game (Nhà thám hiểm và Xác ướp).

- **Định danh đối tượng:** Mỗi nhân vật sở hữu tọa độ thực ( $x, y$ ) và các cờ trạng thái như `song` (còn sống hay không), `have_key` (đã nhặt chìa khoá chưa).
- **Kiểm tra va chạm (Collision Detection):** Phương thức `di_chuyen(huong)` không chỉ đơn thuần thay đổi tọa độ, mà nó đóng vai trò là bộ lọc hợp lệ. Nó gọi lại phương thức `trang_thai` của đối tượng `Mecung` để xác nhận xem hướng di chuyển có bị chặn bởi tường bitmask hay không. Nếu hợp lệ, phương thức trả về 1, ngược lại trả về 0.

## 4.2 Xây dựng Cấu trúc dữ liệu nâng cao và Thuật toán

Sau khi hệ thống vật lý hoàn thiện, giai đoạn 2 tập trung vào việc tăng tính nổi bật cho game thông qua các thuật toán sinh ngẫu nhiên và thuật toán làm khó người chơi.

### 4.2.1 Tự xây dựng Cấu trúc dữ liệu

Thay vì phụ thuộc hoàn toàn vào thư viện có sẵn, nhóm đã tự cài đặt các cấu trúc dữ liệu cốt lõi để tối ưu hóa cho bài toán cụ thể:

- **Queue bằng Linked List (File linklist.py):** Nhóm xây dựng cấu trúc hàng đợi (Queue) dựa trên danh sách liên kết đơn (Node). Mỗi node chứa giá trị và con trỏ next. Việc cài đặt thủ công này giúp nhóm kiểm soát và hiểu rõ cơ chế FIFO (First-In-First-Out) phục vụ cho thuật toán loang (BFS).
- **Disjoint Set Union - DSU (File dsu.py):** Để sinh ra một mê cung hoàn hảo (mọi điểm đều liên thông và không có chu trình thừa), nhóm sử dụng cấu trúc DSU.
  - Mã hoá tọa độ 2D ( $x, y$ ) sang chỉ số 1D để quản lý các tập hợp đỉnh.
  - Phương thức `find(u)` sử dụng kỹ thuật nén đường đi để tìm gốc của tập hợp hay còn gọi là đỉnh đại diện cho thành phần liên thông đó.
  - Phương thức `union(u, v)` giúp gộp 2 thành phần liên thông chứa 2 đỉnh lại, hỗ trợ cho việc nối cạnh.

### 4.2.2 Hệ thống Đa hình trong Thuật toán (File thuattoan.py)

Thuật toán được thiết kế theo mô hình Hướng đối tượng (OOP) với tính Đa hình (Polymorphism), cho phép mở rộng độ khó dễ dàng:

- **Lớp cha Thuat\_toan:** Đóng vai trò là bộ điều khiển trung tâm. Nó quản lý các tương tác như: Nhà thám hiểm nhặt chìa khoá, Xác ướp bắt được người chơi, hoặc va chạm với bom.
- **AI Cấp độ Dễ (Thuat\_toan\_easy):** Sử dụng thuật toán ngẫu nhiên. Xác ướp di chuyển không có mục đích cụ thể, chỉ đơn giản là chọn một hướng đi hợp lệ bất kỳ trong 4 hướng.
- **AI Cấp độ Trung bình (Thuat\_toan\_medium):** Áp dụng thuật toán Tham lam dựa trên khoảng cách Manhattan.

$$D = |x_{mummy} - x_{player}| + |y_{mummy} - y_{player}|$$

Tại mỗi bước, Xác ướp sẽ tính toán khoảng cách  $D$  giả định cho 4 hướng đi. Hướng nào giúp giảm  $D$  nhiều nhất sẽ được lựa chọn. Ở thuật toán này sẽ giữ được tính ngẫu thơ của trò chơi giúp người chơi sẽ cảm thấy giải trí hơn

- **AI Cấp độ Khó (Thuat\_toan\_hard):** Sử dụng thuật toán BFS .
  - Sử dụng hàng đợi `dequeue` để duyệt qua các trạng thái.
  - Kết quả giúp Xác ướp không chỉ đuổi theo mà còn có khả năng chặn đầu hoặc tìm đường ngắn nhất xuyên qua mê cung phức tạp để bắt người chơi.

### 4.2.3 Phát triển Mê cung Mặc định và Hệ thống Tạo Mê cung Ngẫu nhiên

Dây là một thành phần cốt lõi của trò chơi. Trong giai đoạn này, nhóm đã tập trung thiết kế các màn chơi mặc định và cài đặt thuật toán sinh mê cung ngẫu nhiên.

Mê cung được mô hình hóa dưới dạng một ma trận 2 chiều. Mỗi ô trong ma trận chứa một giá trị nguyên từ 0 đến 15. Giá trị này đại diện cho 4 bit trạng thái của 4 bức tường bao quanh ô đó như các quy ước đã nói trên.

Để đảm bảo trải nghiệm người chơi tốt nhất, hệ thống mê cung tuân thủ các nguyên tắc:

- **Mật độ tường:** Số lượng tường không vượt quá 50% tổng diện tích mê cung.
- **Luôn có cách thắng:** Luôn đảm bảo tồn tại ít nhất một đường đi hợp lệ dẫn đến chiến thắng cho người chơi.

Quy trình tạo mê cung động được chia làm hai giai đoạn nhỏ:

- **Giai đoạn 1: Xây dựng tường.** Xây dựng mê cung với tường ngẫu nhiên bằng cách sử dụng cấu trúc dữ liệu Disjoint Set Union (DSU).
- **Giai đoạn 2: Đảm bảo tồn tại đường đi chiến thắng.** Để đảm bảo luôn tồn tại cách thắng, nhóm đã sử dụng kĩ thuật quy hoạch động kết hợp BFS như đã trình bày.

## 4.3 Quá trình phát triển Giao diện Đồ họa và Hệ thống Âm thanh

Sau khi hoàn thiện phần logic cốt lõi và thuật toán, nhóm tiến hành giao diện, thêm âm thanh cho trò chơi. Mục tiêu là chuyển đổi các khối vuông đơn điệu thành hình ảnh trực quan, sinh động và tích hợp âm thanh để tăng trải nghiệm người dùng. Quá trình này được chia thành 3 giai đoạn chính:

### 4.3.1 Tìm hiểu công nghệ và Yêu cầu

Trước khi bắt tay vào thực hiện, nhóm đã tiến hành phân tích các yêu cầu kỹ thuật dựa trên thư viện pygame:

- **Về hình ảnh (Graphics):** pygame hỗ trợ load các định dạng ảnh phổ biến như PNG, JPG. Tuy nhiên, để tạo hiệu ứng chuyển động (animation) cho nhân vật, ta không thể chỉ dùng ảnh tĩnh. Cần tìm hiểu kỹ thuật **Sprite Sheet** (một tấm ảnh lớn chứa nhiều khung hình nhỏ) và sử dụng phương thức **subsurface** để cắt ảnh theo thời gian thực [2].
- **Về âm thanh (Audio):** Cần phân biệt giữa *Nhạc nền* (Background Music - phát lặp lại) và *Hiệu ứng âm thanh* (Sound Effects - phát 1 lần khi có sự kiện). Thư viện pygame.mixer cung cấp **music** cho nhạc nền và **Sound** cho hiệu ứng ngắn.

### 4.3.2 Tìm kiếm và Xử lý tài nguyên

Để đảm bảo tính thẩm mỹ và đồng bộ với phong cách Ai Cập cổ đại của trò chơi gốc Mummy Maze, nhóm đã quyết định tìm kiếm nguồn tài nguyên mã nguồn mở.

- **Nguồn tài nguyên:** Một phần assets (hình ảnh nhân vật, tường, vật phẩm và các file âm thanh) được tham khảo và chọn lọc từ kho lưu trữ mã nguồn mở của dự án Mummy Maze trên GitHub [1].

- **Xử lý tiền kỳ:** Các hình ảnh gốc có kích thước cố định, tuy nhiên game của nhóm cho phép thay đổi kích thước mê cung (từ  $4 \times 4$  đến  $50 \times 50$ ). Do đó, nhóm đã xây dựng cơ chế **Dynamic Scaling** (tự động co giãn) hình ảnh dựa trên kích thước ô (**cell\_size**) được tính toán lúc runtime thay vì fix cứng kích thước ảnh.

### 4.3.3 Cài đặt và Hiện thực hóa

Giai đoạn này tập trung vào việc chuyển đổi các ý tưởng và tài nguyên đã chuẩn bị thành code thực tế trong file `main.py`.

#### a. Xử lý Đồ họa và Animation nhân vật

Việc vẽ nhân vật không chỉ đơn thuần là hiển thị một bức ảnh tĩnh. Để nhân vật "bước đi" mượt mà, nhóm đã cài đặt cơ chế **Sprite Sheet Animation**.

Khi người chơi di chuyển xuống dưới, hệ thống sẽ lần lượt cắt và hiển thị 5 hình ảnh con từ trái sang phải. Tương tự đối với nhân vật phản diện (Mummy), một Sprite Sheet riêng biệt cũng được áp dụng.

Ngoài ra, nhóm đã sử dụng kỹ thuật **Memoization** (ghi nhớ) bằng cách lưu trữ các ảnh đã scale vào biến toàn cục để tối ưu hóa hiệu năng.

#### b. Hệ thống Âm thanh tương tác

Hệ thống âm thanh được thiết kế để phản hồi lại hành động của người chơi theo thời gian thực:

- **Âm thanh môi trường:** Nhạc nền được phát lặp lại (loop).
- **Âm thanh sự kiện:** Tiếng bước chân, tiếng thắng/thua.
- **Cảnh báo nguy hiểm (Proximity Sound):** Nếu khoảng cách giữa người chơi và Xác ướp < 3 ô, âm thanh cảnh báo dồn dập sẽ kích hoạt.

## 5 Hạn chế và định hướng phát triển

Mummy Maze đã được nhóm triển khai thành công, có thể chạy được và chơi được dựa trên các thuật toán đã tìm hiểu, tuy nhiên vẫn còn đó một số vấn đề:

- Giao diện chưa đẹp và chưa hoàn chỉnh. Theo video demo, có thể thấy có một số các thông báo của game chưa có các định dạng hợp theo chủ đề trò chơi mà chỉ là các dòng chữ hiển thị thẳng lên giao diện, cũng như một số ô nhập dữ liệu không khớp với hình ảnh.
- Trò chơi chưa thực sự đa dạng về lối chơi, chỉ tăng độ khó bằng việc điều chỉnh mê cung, tăng số lượng mummy hoặc yêu cầu key để thắng game.

Định hướng phát triển trong tương lai:

- Cải thiện đồ họa giao diện trò chơi, tự vẽ hoặc kiểm các thiết kế mã nguồn mở để thay thế một số hình ảnh hiện tại.
- Tìm hiểu cách để cho các nút bấm được phân biệt rõ ràng hơn, cũng như sửa các ô nhập liệu trong trò chơi cho khớp với giao diện.
- Thêm một số các chương ngại, phản diện khác như bọ cạp, ma, ... và cũng như một số các tính năng khác để thay đổi lối chơi
- Tìm hiểu và thử ứng dụng một số thuật toán khác trong việc tạo mê cung, tìm lối đi cho nhân vật và cách mummy di chuyển

## Tài liệu

- [1] Osddeitf, “Mummy Maze Assets Repository,” *GitHub*, 2016. [Trực tuyến]. Địa chỉ: <https://github.com/osddeitf/mummy-maze>. [Truy cập: 10/01/2026].
- [2] Pygame Community, “Pygame Documentation - Surface and Mixer module,” *Pygame.org*. [Trực tuyến]. Địa chỉ: <https://www.pygame.org/docs/>. [Truy cập: 10/01/2026].
- [3] Python Software Foundation, “Python JSON Library Docs,” *Python.org*. [Trực tuyến]. Địa chỉ: <https://docs.python.org/3/library/json.html>. [Truy cập: 10/01/2026].