ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

BÁO CÁO MÔN HỌC

CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

IT003.L21.KHCL

Đề tài: TẠO VÀ GIẢI MÃ MÊ CUNG

(ĐÃ BÁO CÁO)

-PHANDONGTEAM-

GIẢNG VIÊN HƯỚNG DẪN: HUỲNH THỊ THANH THƯƠNG

TP. HỒ CHÍ MINH, 7/2021

MỤC LỤC

Thành Viên Nhóm

Họ và Tên	Mã số sinh viên
Phạm Phước An	20520375
Nguyễn Vũ Dương	20520465
Hoàng Công Danh	20520431
Bùi Hữu Đức	20520449

Phân chia công việc

Thành viên	Công việc	Mức độ đóng góp
Nguyễn Vũ Dương	- Code chính - Viết báo cáo chính - Tìm kiếm tài liệu - Phụ PowerPoint	**100%**
Phạm Phước An	- Code phụ - Quay và edit video - Phân công công việc	**90%**
Hoàng Công Danh	- Viết báo cáo - Soạn slide A* - Quản lý thời gian công việc	**80%**
Bùi Hữu Đức	- Viết báo cáo - Soạn slide BackTracking - Tìm kiếm thông tin	**80%**

PHẦN 1. GIỚI THIỆU ĐỀ TÀI SEMINAR

1. Mê cung

-Nguồn gốc : Mê cung có nguồn gốc lâu đời . Theo thần thoại Hy Lạp thì mê cung đầu tiên được xây dựng trên đảo Crete, một đảo lớn thuộc Địa Trung Hải.

-Mê cung thường được hiểu là một hệ thống đường ngầm gồm nhiều nhánh ngang dọc nối chằng chịt với nhau, dùng để chỉ những công trình gồm rất nhiều hành lang, lối đi được bao bọc xung quanh với vô vàn ngã ba, ngã tư, ngõ cụt.

-Ứng dụng của mê cung trong đời sống VD : game , Xây dựng kinh thành (Thành Cổ Loa) , …

2. Trò chơi mê cung – Maze

-Maze được ứng dụng trong trò chơi được phổ biến nhất hiện nay , Game maze đầu tiên được làm ra là vào năm 1959 nó có tên lả Mouse in the Maze được làm bởi nhóm sinh viên của trưởng MIT(Massachusetts Institute of Technology) được làm trên máy TX-0

-Luật chơi của game maze thì tùy theo người lập trình thiết lập thì nó sẽ có cách chơi khác nhau . Nhưng luật chơi chung của nó là một mê cung được tạo ngẫu nhiên và cho người chơi giải mê cung bằng cách tìm đường đi từ một ô bắt đầu đến một ô kết thúc có trong maze

PHẦN 2. Ý TƯỞNG GIẢI QUYẾT

1. Lên phương án

Nhóm em không muốn tạo ra game mê cung mà em muốn tìm hiểu xem cách tạo mê cung và cách tìm đường đi cho mê cung đó . Do đó luật mê cung của em sẽ hơi khác do tụi em chỉ muốn tìm hiểu cách làm ra mê cung :

+ Tụi em sẽ cho người xem cách tạo ra mê cung.
+ Khi tạo xong thỉ cho người chơi có thể thích thêm tường hoặc không.
+ Sau đó máy tính sẽ tìm đường đi từ ô bắt đầu đến ô kết thúc (2 ô này có thể được chỉnh sửa).

2. Phương pháp giải quyết

+ Để giải quyết cách tạo ra mê cung 1 cách ngẫu nhiên thì nhóm em xài thuật toán **Randomized depth-first search**.
+ Và để giải mê cung thì nhóm em sẽ xài thuật toán **A*** để tìm đường đi từ ô bắt đầu đến ô kết thúc.
+ Để hình dung Maze 1 cách tốt nhất em sẽ sử dụng thư viện độ họa là **SFML**.

PHẦN 3. THIẾT KẾ CẤU TRÚC DỮ LIỆU

1. Dữ liệu có trong bài

+ Số lượng hàng và cột lớn nhất có thể có.
+ Vị trí ô bắt đầu tạo maze mặc định.
+ Vị trí ô bắt đầu tìm đường mặc định.
+ Vị trí ô kết thúc tìm đường mặc định.
+ Số lượng hàng và cột do người nhập thiết lập.
+ Ô kết thúc và ô bắt đầu do người nhập thiết lập.
+ Ô tạo maze bắt đầu do người nhập quy định.

2. Tổ chức dữ liệu

- Ta cũng sẽ tạo các enum để cho ta có thể hiểu được trạng thái hiện tại của maze là gì.
- Ta sẽ tạo ra 1 class Spot để dễ dàng cho việc quản lý.
- Trong Spot ta cần thiết lập các yếu tố cần có của 1 ô trong mê cung :
+ Tường , màu sắc của tường và màu sắc của ô.
+Tạo ra các hàm trong public: để ta có thể chỉnh sửa tường và màu sắc ô. Để liên kết các Spot với nhau thì ta sẽ dùng vector để lưu trữ nó để dễ dàng phân tích và quản lý nó một cách dễ dàng.

PHẦN 4. THIẾT KẾ GIẢI THUẬT

Đối với thuật toán này ta có phải giải quyết như sau, vì ta có rất nhiều trạng thái:

- Chọn ô bắt đầu tạo mê cung.
- Tạo mê cung.
- Chọn các ô bị chặn trong mê cung.
- Chọn các ô bị chặn.
- Chọn ô bắt đầu và kết thúc.
- Giải mê cung (Sử dụng A*).

1. Tạo ra enum để là các trạng thái của mê cung

Bước 1 : Ta cần khai báo 1 state (trạng thái) là chưa khởi tạo maze.

Bước 2 : Tạo ra các enum đây là các trường hợp có thể có của state.

2. Hàm check input để kiểm tra người dùng muốn làm gì

Bước 1: Tạo ra các biến để kiểm tra xem ô người nhập muốn chọn.

Bước 2: Kiểm tra các biến đó đồng thời kiểm tra trạng thái hiện tại.

Bước 3 Nếu bước 2 được thực thi thì chuyển state thành trạng thái tương ứng.

3. Hàm thay đổi ô bắt đầu tạo mê cung

Bước 1: Kiểm tra trạng thái hiện tại của maze .

Bước 2: Nếu đúng là trạng thái chưa khởi tạo maze thì ta sẽ lấy thông tin ô mà người dùng muốn chọn bằng cách nhấn chuột.

Bước 3: Để chọn ô bắt đầu ta nhấn chuột trái (Tạo ra hàm kiểm tra có đúng là nhấn chuột trái không) đồng thời thực thi hàm tính toán để xác định đúng ô đó trong màn hình.

Bước 4 : Chọn xong chuyển trạng thái của maze.

4. Tạo mê cung

Bước 1: Đặt ô bắt đầu (ô tạo maze) thành ô hiện tại và đánh dấu nó đã thăm.

Bước 2: Trong khi có các ô chưa được truy cập.

- Nếu ô hiện tại có bất kì hàng xóm nào chưa được truy cập:

    + Chọn ngẫu nhiên 1 hàng xóm chưa được thăm.

    + Đẩy ô hiện tại vào stack.

    + loại bỏ tường giữa ô hiện tại và ô đã chọn.

    + Đặt ô đã chọn là ô hiện tại và đánh dấu nó đã thăm.

- Nếu stack không rỗng:

    + Lấy 1 ô ra khỏi stack.

    + Đặt nó thành ô hiện tại.

- Trường hợp khác:

    + Chọn 1 ô chưa được truy cập và chọn một cách ngẫu nhiên và đặt nó thành ô hiện tại và đánh dấu nó đã được thăm.

5. Thuật toán A*

Bước 1, Khởi tạo openList.

Bước 2, Khởi tạo closedList.

- Đặt ô bắt đầu openList (có thể đặt giá trị f = 0).

Bước 3, Trong khi openList không rỗng.

a, Tìm ô có giá trị f nhỏ nhất trong openList và gọi nó là 'q'.

b, Đưa q ra khỏi openList.

c, Tạo ra các ô kế thừa của ô q và thiết lập cha mẹ đến q.

d, Đối với mỗi ô kế thừa.

- Nếu ô kế thừa là đích , thì ngừng tìm kiếm , với:

+ g = q.g+ Khoảng cách giữa ô kế thừa và q.

+ h = khoảng cách từ đích đến ô kế thừa (Được sử dụng bằng nhiều cách vd: Heuristic , Manhattan).

+ f = g + h.

- Nếu một ô trùng với vị trí ô kế thừa nằm trong openList và có giá trị f thấp hơn ô kế thừa bỏ qua ô kế thừa này.

- Nếu 1 ô có cùng vị trí với ô kế thừa trong closedList và có f thấp. hơn so với ô kế thừa , bỏ qua ô kế thừa này và thêm ô này vào openList.

e, Thêm q vào closedList.

6. Tạo tường theo yêu cầu

Bước 1 : Kiểm tra state.

Bước 2 : Nếu đúng state thì thực thi tiếp bước 3 không thì bỏ qua.

Bước 3 : Để chọn ô thì ta nhấn chuột trái(cũng phải check xem có đúng người dung nhấn chuột trái không) đồng thời ta cũng phải tính toán vị trí ô đó trên màn hình.

Bước 4 : Khi chọn ô đấy xong đưa ô đấy vào stack là ta có thể return về trạng thái ban đầu bằng việc nhấn nút escape.

Bước 5 : Thực thi xong thì state sang trạng thái khác.

7. Thay đổi ô bắt đầu và kết thúc

Bước 1 : Kiểm tra trạng thái hiện tại của state.

Bước 2 : Nếu đúng trạng thái thì chuyển sang bước 3 nếu không thì nếu thúc.

Bước 3 : Kiểm tra người dung có nhấn chuột trái không nếu có thì vị trí trên màn hình của chuột chính là ô bắt đầu còn nếu người dùng nhấn chuột phải thì vị trí ô đó trên màn hình là chuột phải (xây dựng các hàm xác định tọa độ ô).

Bước 4 : Chọn xong thì thay đổi state.

PHẦN 5 : LẬP TRÌNH CÀI ĐẶT

1. Thông tin về chương trình

-Với project này nhóm em sử dụng **ngôn ngữ C++.**

-Vì liên quan đến độ họa nên em đã sử dụng thêm **thư viện SFML**

+ **SFML**(là viết tắt của Simple and Fast Multimedia Library) là một thư viện đa phương tiện viết từ C++. Nó được viết theo hướng OOP.

+ **SFML** bao gồm 5 modules: **Audio**, **Graphics**, **Network**, **System**, **Window**.

.**System**: gồm các class liên quan với hệ thống như làm thời gian, xử lý unicode

.**Window**: liên quan tới việc tạo, đóng và xử lý sự kiện cửa số.

.**Graphics**: bao gồm các class về việc render đồ họa.

.**Audio**: bao gồm các class về xử lý âm thanh, ta có thể dùng để phát một file nhạc hoặc ghi âm cho máy tính và lưu thành file.

+ Ngoài ra, **SFML** còn có thể chạy trên nhiều hệ điều hành khác nhau như Windows, Linux, MacOS.

- Trong project này em sử dụng lập trình theo hướng dối tượng.
- Code trong project là nhóm em tự code từ những thuật toán tụi em tìm hiểu được và cải tiến nó theo ý tưởng của riêng mình.

2. Giải thích sơ lược về code

- Code chính :

```
// Liệt kê các trạng thái có thể có của maze

enum states

{

   PREMAZE,
```

```
    Change_start_create_maze,
    complete_change_create_maze,
    GENMAZE,
    PRESOLVE,
    add_wall,
    complete_wall,
    Change_start_end,
    complete_change,
    add_start,
    SOLVING,
    SOLVED
};
```

// Thiết lập trạng thái của maze là màn hình trắng

```
int state = PREMAZE;
```

// Tạo ra 1 class Spot được kế thừa từ class Drawable trong thư viện sfml

// Việc tạo class này để dễ vẽ và quản lý các ô trong màn hình độ họa

```
class Spot : public sf::Drawable
```

```
{
private:
    sf::Color col = sf::Color::White; // Mặc định các ô có  màu trắng
public:
    bool visited = false;  // kiểm tra được thăm không

// f ,g ,h hỗ trợ cho việc giải maze
    float f = 0;
    float g = 0;
    float h = 0;

// i và j là tọa độ của ô trong màn hình
    float i = 0;
    float j = 0;

// Thiết lập mặc định là ô này là ô khép kín
    std::vector<bool> walls = { true, true, true, true };

    std::vector<Spot*> neighbors;
    Spot* previous = 0;

// Thêm hàng xóm cho ô (*This) khi maze chưa vẽ
    void addNeighbors(std::vector<std::vector<Spot>>& cells)
```

```
    {
        neighbors.clear();
        // Phải kiểm tra tọa độ của ô để tránh các trường hợp tọa độ rác
        if (i < *cols - 1) {
            if (!cells[i + 1][j].visited)
                neighbors.push_back(&cells[i + 1][j]);
        }
        if (i > 0) {
            if (!cells[i - 1][j].visited)
                neighbors.push_back(&cells[i - 1][j]);
        }
        if (j < *rows - 1) {
            if (!cells[i][j + 1].visited)
                neighbors.push_back(&cells[i][j + 1]);
        }
        if (j > 0) {
            if (!cells[i][j - 1].visited)
                neighbors.push_back(&cells[i][j - 1]);
        }
    }
```

// Hàm hỗ trợ thêm hàng xóm cho maze đã được vẽ

```
    void addMazeNeighbors(std::vector<std::vector<Spot>>& cells)
```

```
  {
      neighbors.clear();
      if (i < *cols - 1 && !walls[1])
          neighbors.push_back(&cells[i + 1][j]); //Neighbor phía đông
      if (i > 0 && !walls[3])
          neighbors.push_back(&cells[i - 1][j]); // Neighbors phía tây
      if (j < *rows - 1 && !walls[2])
          neighbors.push_back(&cells[i][j + 1]); // Neighbors phía nam
      if (j > 0 && !walls[0])
          neighbors.push_back(&cells[i][j - 1]); // Neighbors phía bắc
  }

// Hàm lấy tọa độ của ô
  void setij(int x, int y) {
      i = x;
      j = y;
  }

// Thiết lập màu sắc cho ô đó
  void setCol(sf::Color new_col) {
      col = new_col;
  }
```

```
// Hàm này đơn giản là vẽ các ô đấy trong đồ họa
   virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
  {
      // Thiết lập kích thước một ô
         int cell_w = SCR_WIDTH / *cols;
      int cell_h = SCR_HEIGHT / *rows;
      sf::RectangleShape shape(sf::Vector2f(cell_w, cell_h)); // Dùng để vẽ ô đó trên của sổ
      shape.move(sf::Vector2f(i * cell_w, j * cell_h));  // Di chuyển ô ấy đến đúng tọa độ ô ấy trên màn hình
      shape.setFillColor(col);
      target.draw(shape);

      if (walls[0])
     { // Vẽ tường trên
         sf::VertexArray wall(sf::Lines, 2);
         wall[0].position = sf::Vector2f(i * cell_w, j * cell_h);
         wall[0].color = sf::Color::Black;
         wall[1].position = sf::Vector2f((i + 1) * cell_w, j * cell_h);
         wall[1].color = sf::Color::Black;
         target.draw(wall);
      }
      if (walls[1])
```

```
{ // Vẽ tường bên phải
    sf::VertexArray wall(sf::Lines, 2);
    wall[0].position = sf::Vector2f((i + 1) * cell_w, j * cell_h);
    wall[0].color = sf::Color::Black;
    wall[1].position = sf::Vector2f((i + 1) * cell_w, (j + 1) * cell_h);
    wall[1].color = sf::Color::Black;
    target.draw(wall);
}
if (walls[2])
{ // Vẽ tường dưới
    sf::VertexArray wall(sf::Lines, 2);
    wall[0].position = sf::Vector2f(i * cell_w, (j + 1) * cell_h);
    wall[0].color = sf::Color::Black;
    wall[1].position = sf::Vector2f((i + 1) * cell_w, (j + 1) * cell_h);
    wall[1].color = sf::Color::Black;
    target.draw(wall);
}
if (walls[3])
{ // Vẽ tường trái
    sf::VertexArray wall(sf::Lines, 2);
    wall[0].position = sf::Vector2f(i * cell_w, j * cell_h);
    wall[0].color = sf::Color::Black;
    wall[1].position = sf::Vector2f(i * cell_w, (j + 1) * cell_h);
```

```
            wall[1].color = sf::Color::Black;
            target.draw(wall);
        }
    }
};
// Khai báo 1 cái lưới dùng để chứa các ô để dễ xử lý
std::vector<Spot> tmp(*rows);
std::vector<std::vector<Spot>> grid(*cols, tmp);

std::vector<Spot*> openSet;
std::vector<Spot*> closedSet;

std::vector<Spot*> cell_stack;
int start_x = 0;
int start_y = 0;
Spot* start = &grid[start_x][start_y]; // Ô tọa độ bắt đầu dùng để giải maze

// Để thêm tất cả các ô trong lưới vào để giải quyết
void addToOpenSet(Spot* cell)
{
    openSet.push_back(cell);
    cell->setCol(sf::Color(180, 255, 180));
}
```

```
// Thêm ô đã được ghé thăm
void addToClosedSet(Spot* cell)
{
    closedSet.push_back(cell);
    cell->setCol(sf::Color::Red);
}
// Xóa ô đó ra khỏi mảng
void removeFromArray(Spot* cell, std::vector<Spot*>& vect)
{
    for (int i = 0; i < (int)vect.size(); i++) {
        if (vect[i] == cell) {
            vect.erase(vect.begin() + i);
        }
    }
}
// Kiểm tra xem các ô có trong mảng  còn hay ko
bool arrayContainsElem(Spot* cell, std::vector<Spot*>& vect)
{
    for (int i = 0; i < (int)vect.size(); i++) {
        if (vect[i] == cell) {
            return true;
        }
    }
```

```
    return false;
}
// Xóa tường giữa 2 ô ( là nơi giúp cho việc quan trọng cho việc tạo maze )
void removeWall(Spot* a, Spot* b)
{
    int x = a->i - b->i;
    int y = a->j - b->j;
    if (x == 1) {
        a->walls[3] = false;
        b->walls[1] = false;
    }
    if (x == -1) {
        a->walls[1] = false;
        b->walls[3] = false;
    }
    if (y == 1) {
        a->walls[0] = false;
        b->walls[2] = false;
    }
    if (y == -1) {
        a->walls[2] = false;
        b->walls[0] = false;
    }
```

```
}


// Hàm kiểm tra tường để tìm đường đi
bool checkWall(Spot* a, Spot* b)
{
   int x = a->i - b->i;
   int y = a->j - b->j;
   if (x == 1) {
      return a->walls[3];
   }
   if (x == -1) {
      return a->walls[1];
   }
   if (y == 1) {
      return a->walls[0];
   }
   return a->walls[2];
}




// Hàm này chỉ để ước lượng khoảng cách giữa 2 ô trong maze
```

```
float heuristic(Spot* a, Spot* b)
{
   return std::sqrt((a->i - b->i) * (a->i - b->i) + (a->j - b->j) * (a->j - b->j));    //trả lại khoảng cách euclid giữa hai điểm

}



void checkInput()
{
// Khai báo các biến này để kiểm tra người dùng có nhấn nó hay không

   bool p_current = sf::Keyboard::isKeyPressed(sf::Keyboard::P); // Dùng cho việc chỉnh sửa nới bắt đầu tạo maze
   bool m_current = sf::Keyboard::isKeyPressed(sf::Keyboard::M); //  Tạo Maze
   bool c_current = sf::Keyboard::isKeyPressed(sf::Keyboard::C); // Thêm tường sau khi tạo maze
   bool f_current = sf::Keyboard::isKeyPressed(sf::Keyboard::F); // Thay đổi vị trí start và end là 2 vị trí để thuất toán giải maze
   bool s_current = sf::Keyboard::isKeyPressed(sf::Keyboard::S);  // Giải maze

// Tùy thuộc vào nút bấm và trạng thái hiện tại thì chương trình  sẽ thực hiện
```

```
if (p_current)
{
    if (state == PREMAZE)
        state = Change_start_create_maze;
}
if (!m_previous && m_current) {
    // M key was pressed, generate maze
    if (state == PREMAZE || state == complete_change_create_maze)
        state = GENMAZE;
}
if (c_current)
{
    if (state == PRESOLVE)
    {
        state = add_wall;
    }
}

if (f_current)
{
    if (state == complete_wall || state == PRESOLVE)
        state = Change_start_end;
```

```
    }
    if (!s_previous && s_current) {
        // S key was pressed, solve maze
        if (state == PRESOLVE || state == complete_wall || state ==
complete_change) {


            state = add_start;
        }
}


// Dùng để có thể quay lại ô trước ( dùng cho lúc thêm tường)
std::stack<Spot>  cell_pre;
int x_cools = 0;
int x_row = 0;
int start_change_x = 0;
int start_change_y = 0;
int support_pre_cell = 0;


int end_x = *cols - 1;
int end_y = *rows - 1;
//Hai cái này là giúp chuyển đổi các ô bắt đầu và kết thúc
int change_start_time = 0;
int change_end_time = 0;
```

```
void playMaze()
{


   sf::RenderWindow window(sf::VideoMode(SCR_WIDTH, SCR_HEIGHT),
"maze solver"); // Biến này dùng để tạo ra 1 cửa sổ window có kích thước là
SCR_WIDTH*SCR_HEIGHT và có tên là maze solver
   window.setFramerateLimit(15);  //  Set khung hình xuống thấp dễ theo dõi



   std::srand(std::time(nullptr));
// set tọa độ  ô trong lưới (cho dễ quản lý)
   for (int i = 0; i < *cols; i++) {
      for (int j = 0; j < *rows; j++) {
         grid[i][j].setij(i, j);
      }
   }
   Spot* current = &grid[start_change_x][start_change_y]; // Dùng để tạo mê
cung
   Spot* next = nullptr;// Khai báo để tránh rác
   Spot* end = &grid[end_x][end_y]; // Vị trí kết thúc (cái này là dùng để giải
maze)
   int flagzz = 0;
   int flag_start = 0;
```

```
    while (window.isOpen())
    {
        if (flagzz == 0)
        {
            start_x = 0;
            start_y = 0;
            end_x = *cols - 1;
            end_y = *rows - 1;
            flagzz = 1;
        }
        sf::Event event;
        while (window.pollEvent(event))
        {
           // Khi đống của sổ thì ta phải reset lại như ban đầu để tránh lỗi
            if (event.type == sf::Event::Closed ||
(sf::Keyboard::isKeyPressed(sf::Keyboard::Key::Escape)))
            {
                window.close();
                std::vector<Spot> tmp1(MAX_ROWS);
                std::vector<std::vector<Spot>> grid_r(MAX_COLS, tmp1);
                /*std::vector<Spot*> openSet1;
                std::vector<Spot*> closedSet1;*/
                openSet.clear();
```

```
cell_stack.clear();
closedSet.clear();


//std::vector<Spot*> cell_stack1;
//Spot* start1 = &grid[0][0];
grid = grid_r;
/*openSet = openSet1;
closedSet = closedSet1;
cell_stack = cell_stack1;*/
while (!cell_pre.empty())
{
   cell_pre.pop();
}
/*std::stack<Spot>  cell_pre1;
cell_pre = cell_pre1;*/
//start = start1;
state = PREMAZE;
start_x = 0;
start_y = 0;
end_x = *cols - 1;
end_y = *rows - 1;
start_change_x = start_change_y = 0;
flagzz = 0;
```

```
        }
    }
    window.clear();

//*****************************************************************
*
    start = &grid[start_x][start_y];
    end = &grid[end_x][end_y];


    checkInput();
    //flagx = 1;
    if (state == add_start) //Cho start  vào để giải maze
    {
        addToOpenSet(start);
        if ((end->walls[0] == true) && (end->walls[1] == true) && (end-
>walls[2] == true) && (end->walls[3] == true))
        {
            end_x = *cols;
            end_y = *rows;
        }
        state = SOLVING; // chuyển state sang solving
    }
```

```
    if (state == Change_start_create_maze) // Lấy tọa độ ô bắt đầu tạo maze từ người nhập bằng cách click chuột trái vào ô bất kì trong của sổ và chọn xong nhấn enter
    {
        current->setCol(sf::Color::Blue);
        if (event.type == sf::Event::MouseButtonPressed)
        {

            if (event.mouseButton.button == sf::Mouse::Left)
            {
                current->setCol(sf::Color::White);
                start_change_x = event.mouseButton.x / (SCR_WIDTH / *cols);
                start_change_y = event.mouseButton.y / (SCR_HEIGHT / *rows);

                current = &grid[start_change_x][start_change_y];

                current->setCol(sf::Color::Blue);



            }
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::Enter))
        {
```

```
            state = complete_change_create_maze;
        }
    }
      // Bắt đầu giải maze
    if (state == GENMAZE) {

        if (flag_start == 0)
        {
            current->visited = true; // set ô bắt đầu tạo maze đã dc thăm
            flag_start = 1;
        }
        current->addNeighbors(grid);
       // Kiểm tra các hàng xóm của maze và chọn 1 ô ngẫu nhiên làm đường đi
        if (current->neighbors.size() > 0) {
            float r = static_cast <float> (std::rand()) / static_cast <float>
(RAND_MAX);
            int i = std::floor(r * current->neighbors.size());
            next = current->neighbors[i];
            current->neighbors.erase(current->neighbors.begin() + i);
        }
       // Kiểm tra next nếu next khác NULL thì ta là giống như current
        if (next != nullptr) {
            next->visited = true;
```

```
        cell_stack.push_back(current);


        removeWall(current, next);
        current = next;
        next = nullptr;
    }
    else {
        if (cell_stack.size() == 0) {
            // Đã tạo xong maze
            state = PRESOLVE;
            for (int i = 0; i < *cols; i++) {
                for (int j = 0; j < *rows; j++) {
                    grid[i][j].addMazeNeighbors(grid);
                }
            }
        }
        else {
             // nếu còn ô chưa được thăm thì quay trở lại thăm vã tạo maze
            current = cell_stack[cell_stack.size() - 1];
            cell_stack.pop_back();
        }
    }
```

```
}
  // Trạng thái giải maze
if (state == SOLVING) {
   if (openSet.size() > 0) {
      // Nếu có phần tử thì làm
      int winner = 0;
      for (int i = 0; i < (int)openSet.size(); i++) {
         if (openSet[i]->f < openSet[winner]->f) {
            winner = i;
         }
      }

      current = openSet[winner];

      if (openSet[winner] == end) {
         std::cout << "Done!" << std::endl; // Cái này là tìm được đường đi
         state = SOLVED;
      }

      //Loại bỏ  current ra khỏi open set
      removeFromArray(current, openSet);
      addToClosedSet(current);
```

```
            for (int i = 0; i < (int)current->neighbors.size(); i++) {
// lặp lại qua tất cả các hàng xóm của current
                if (!arrayContainsElem(current->neighbors[i], closedSet) &&
!checkWall(current, current->neighbors[i])/* && !current->neighbors[i]->wall
*/) {     // nếu hàng xóm hiện tại không nằm trong closedset và hàng xóm ko phải
là tường
                    float tempG = current->g + 1;


                    if (arrayContainsElem(current->neighbors[i], openSet)) {    // if
hàng xóng hiện tại  đang ở  the open set (đang có giá trị G)
                        if (tempG < current->neighbors[i]->g) {
                            current->neighbors[i]->g = tempG;                // nếu điểm G
dự kiến tốt hơn hiện tại, hãy đặt giá trị G mới
                            current->neighbors[i]->previous = current;
                        }
                    }
                    else {
                        current->neighbors[i]->g = tempG;
// Nếu hàng xóm hiện tại ko ở  open set, thêm vào  open set và thiết lập giá trị G
mới
                        addToOpenSet(current->neighbors[i]);
                        current->neighbors[i]->h = heuristic(current->neighbors[i],
end);
                        current->neighbors[i]->previous = current;
```

```
                }
                current->neighbors[i]->f = current->neighbors[i]->g + current->neighbors[i]->h;
            }
        }

    }
    else {
        // no solution
        std::cout << "no solution" << std::endl; // Không có đường đi xuất ra no solution
        state = SOLVED;
    }

    for (int i = 0; i < (int)closedSet.size(); i++) {
        closedSet[i]->setCol(sf::Color(255, 180, 180)); // Thiết lập rằng màu này ko phải đường đi
    }
}

if (state == SOLVING || state == SOLVED) {
    while (current->previous != 0) {
```

```
            current->setCol(sf::Color(180, 180, 255));// Thiết lập rằng màu  này  là đường đi
            current = current->previous;
        }
        current->setCol(sf::Color(180, 180, 255));
    }

// Hỗ trợ việc vẽ các ô trong maze
    for (int i = 0; i < *cols; i++) {
        for (int j = 0; j < *rows; j++) {
            if (state == GENMAZE) {
                if (grid[i][j].visited)
                    //grid[i][j].setCol(sf::Color::Magenta);
                    grid[i][j].setCol(sf::Color(255, 180, 255));
                current->setCol(sf::Color::Blue);
            }
            else if (state == PRESOLVE) {
                grid[i][j].setCol(sf::Color::White);
            }
            window.draw(grid[i][j]);
        }
    }
```

```
if (state == add_wall) // Kiểm tra trạng thái
{
   // Thêm tường vào ô bất kì
   sf::Vector2i MousePosn = sf::Mouse::getPosition(window);


   if (event.type == sf::Event::MouseButtonPressed)
   {
      int flag = 0;
          //Thêm tường bằng cách nhấn chuột trái
      if (event.mouseButton.button == sf::Mouse::Left)
      {
         x_cools = event.mouseButton.x / (SCR_WIDTH / *cols);
         x_row = event.mouseButton.y / (SCR_HEIGHT / *rows);



         if ((grid[x_cools][x_row].walls[0] == false) ||
(grid[x_cools][x_row].walls[1] == false) || (grid[x_cools][x_row].walls[2] ==
false) || (grid[x_cools][x_row].walls[3] == false))
         {
            flag = 1;
            cell_pre.push(grid[x_cools][x_row]);
            grid[x_cools][x_row].walls = { true,true,true,true };
            grid[x_cools][x_row].setCol(sf::Color::White);
```

```
            }
        }
        if (flag == 1)
        {
            window.draw(grid[x_cools][x_row]);
        }


    }
      // Muốn quay trở lại các ô thì dùng phím backspace
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::BackSpace))
    {
        if (!cell_pre.empty())
        {
            if (support_pre_cell == 0)
            {
                Spot cell_pre_stack = cell_pre.top();
                cell_pre.pop();
                grid[cell_pre_stack.i][cell_pre_stack.j] = cell_pre_stack;


                support_pre_cell = 1;
            }
            else
                support_pre_cell = 0;
```

```
            }


        }
                // Hoàn thành xong việc tạo tường thì nhấn enter để chuyển sang trạng thái khác
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::Enter))
        {
            state = complete_wall;


        }
    }
     // Thay đổi ô bắt đầu và kết thúc ( dùng cho giải maze )
    if (state == Change_start_end)
    {
        start->setCol(sf::Color::Yellow); // Set ô bắt đầu là vàng
        end->setCol(sf::Color(255, 127, 80)); // Set ô kết thúc là màu cam
        change_end_time = change_start_time = 1;
          // Để thay đổi start thì ta nhấn chuột trái còn end thì chuột phải
        if (event.type == sf::Event::MouseButtonPressed)
        {


            if (event.mouseButton.button == sf::Mouse::Left)
            {
```

```
    if (change_start_time == 1)
    {
        grid[start_x][start_y].setCol(sf::Color::White);
        change_start_time = 0;
    }
    start_x = event.mouseButton.x / (SCR_WIDTH / *cols);
    start_y = event.mouseButton.y / (SCR_HEIGHT / *rows);

    grid[start_x][start_y].setCol(sf::Color::Yellow);
    change_start_time = 1;
}
if (event.mouseButton.button == sf::Mouse::Right)
{
    if (change_end_time == 1)
    {
        grid[end_x][end_y].setCol(sf::Color::White);
        change_end_time = 0;
    }
    end_x = event.mouseButton.x / (SCR_WIDTH / *cols);
    end_y = event.mouseButton.y / (SCR_HEIGHT / *rows);
    grid[end_x][end_y].setCol(sf::Color(255, 127, 80));
    change_end_time = 1;
}
```

```
        }


          // Thay đổi xong nhấn enter thay đổi trạng  thái
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::Enter))
        {
            state = complete_change;
        }
    }


        window.display();
}
```

3. Giao diện

+ Có Menu nơi để người nhập chọn như :

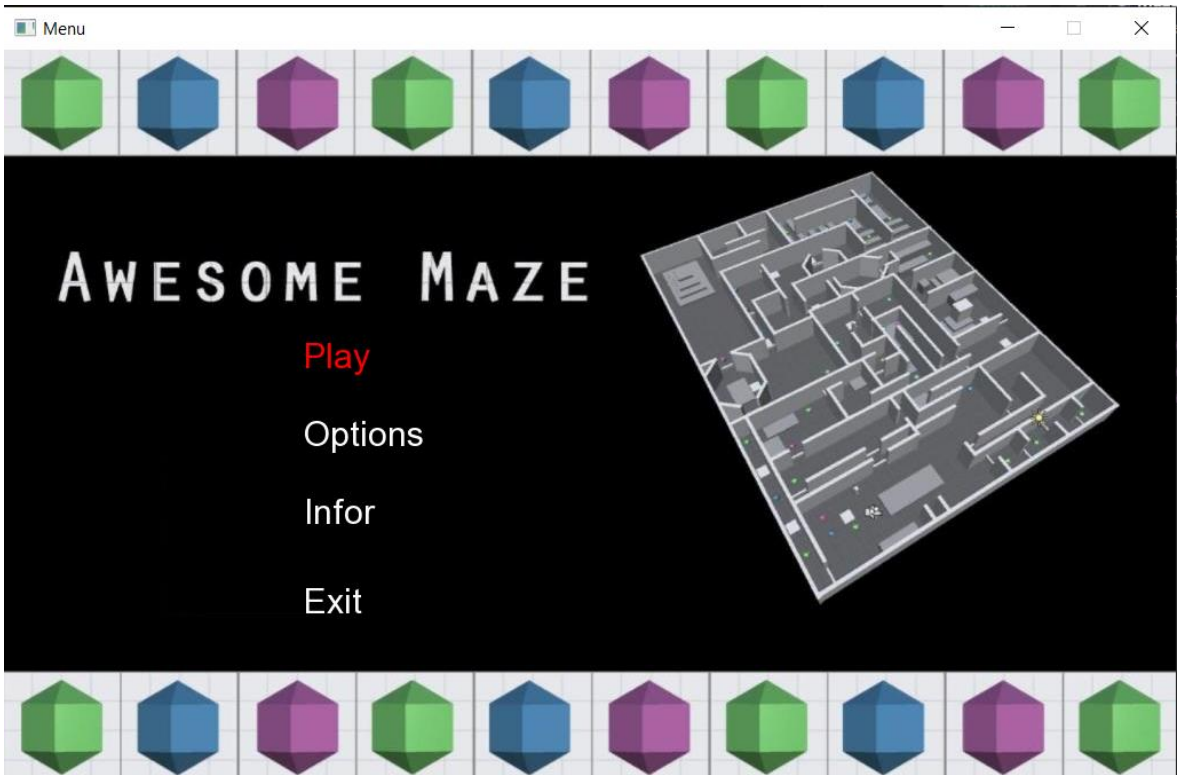
1) Play (dùng để tạo maze)

2) Option dùng thay đổi số lượng hàng và cột trong maze

3) Infor nơi dùng để để lưu thông tin tụi em

4) Exit dùng để thoát

Giao diện Menu như phái dưới



Giao diện trong option:

- Chức năng chương trình :

+ Tạo maze 1 cách ngẫu nhiên từ bất kì ô nào do người chơi chỉ đỉnh (Nếu không chỉ định mặc định là ô (0,0))

+ Cho thêm tường vào mê cung lúc đã tạo xong

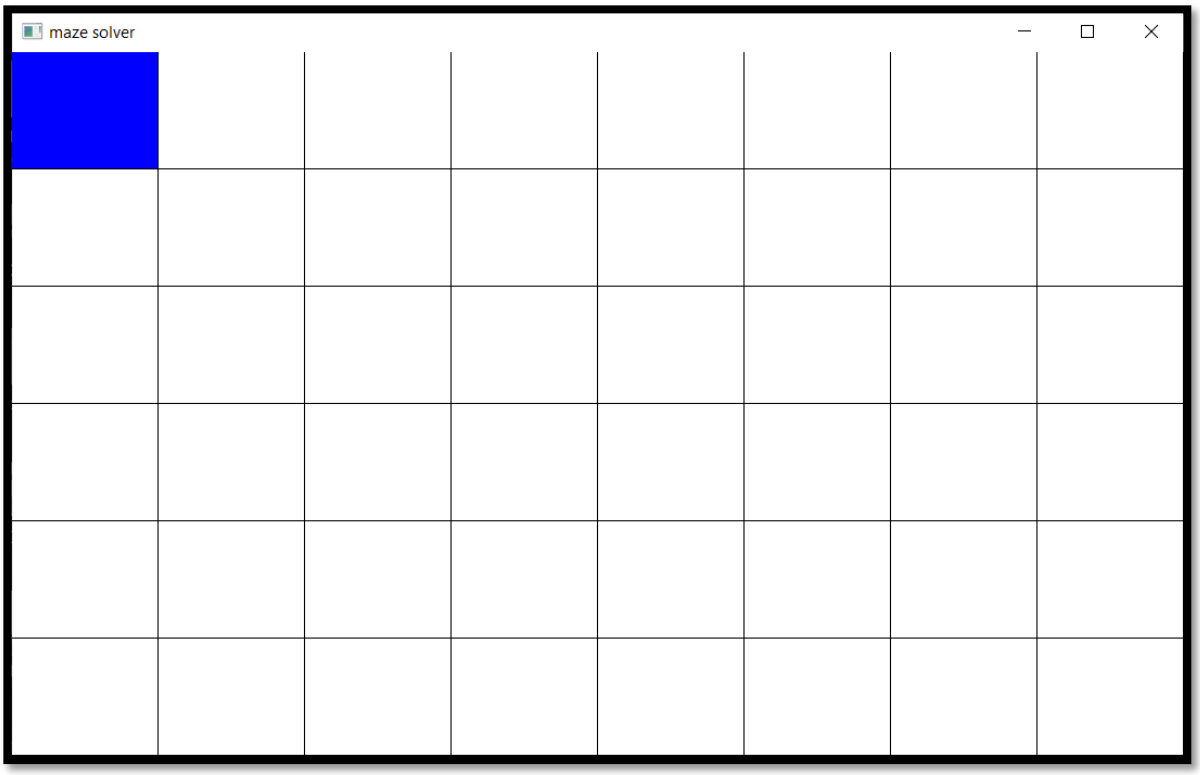
+ Thay đổi được ô bắt đầu và xuất phát

+ Giải mê cung

PHẦN 6 : KIỂM THỬ VÀ KẾT LUẬN

1. Tạo mê cung một cách ngẫu nhiên

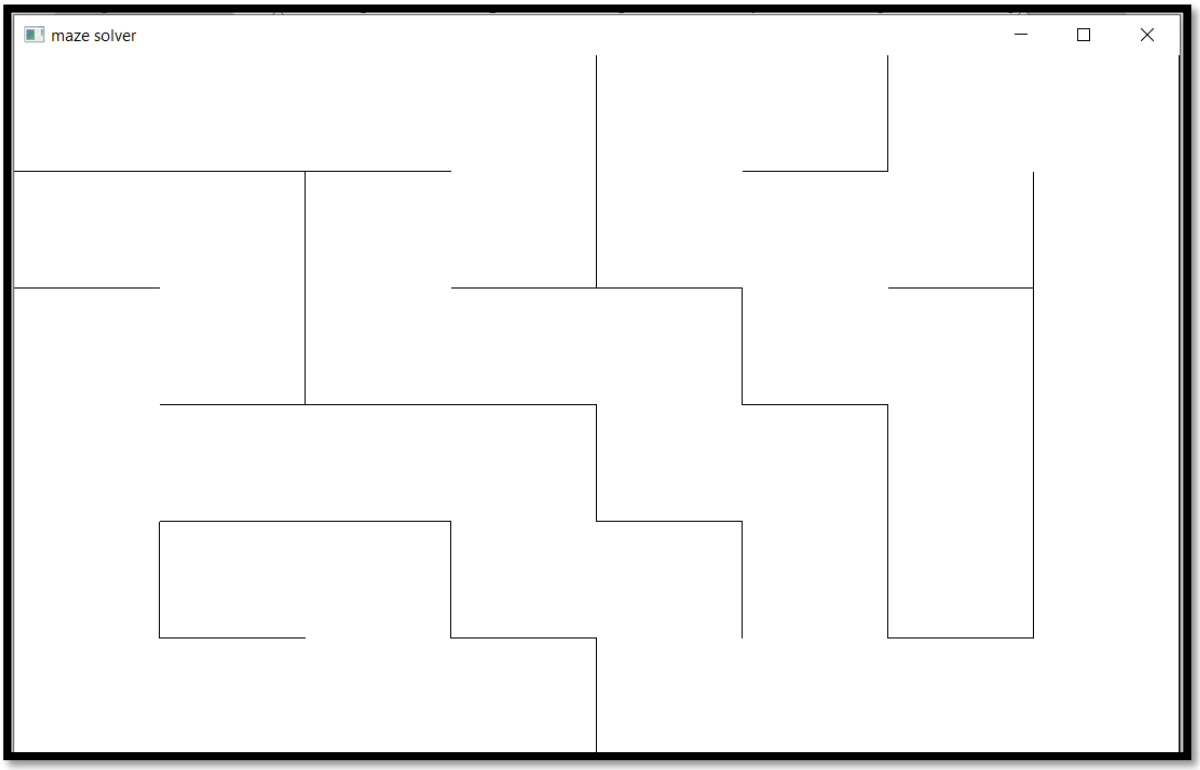
- Đối với việc tạo maze thì ta cần phải chọn một nơi bắt đầu tạo maze
  + Test case đầu tiên thì tụi em sẽ tạo ra mê cung có kích cỡ là (8x6) và chọn ô bắt đầu như hình sau:



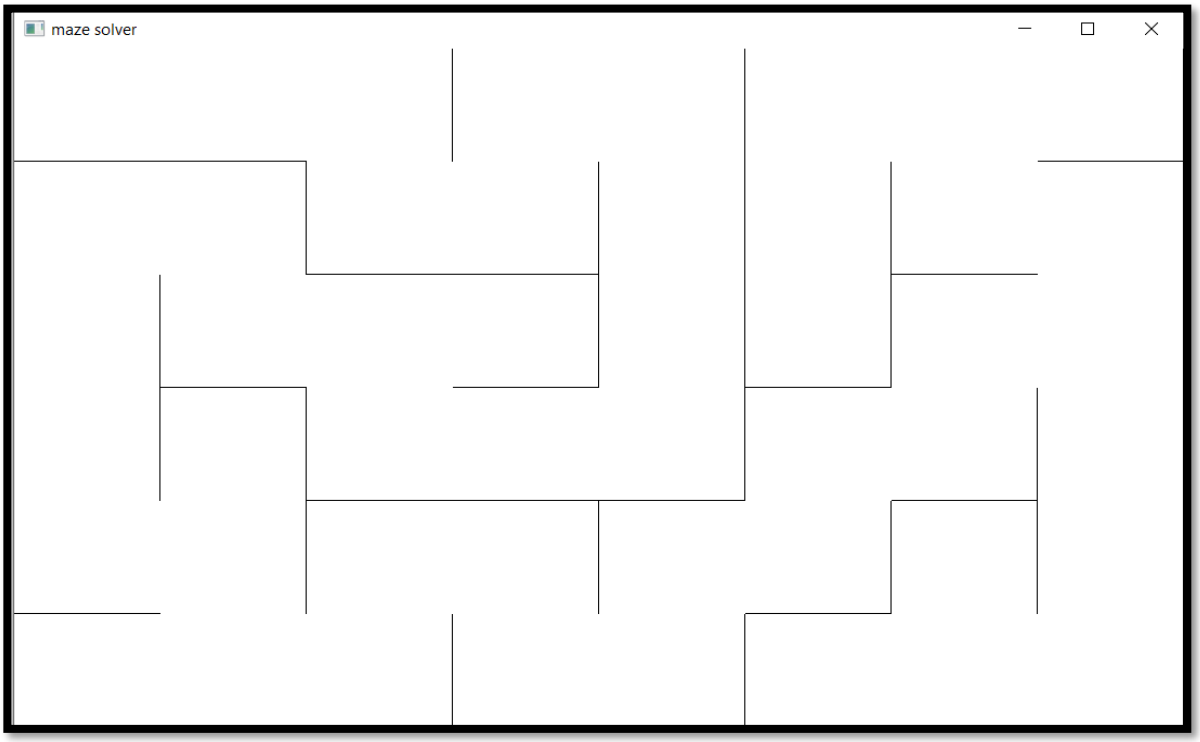
Ô bắt đầu là ô có màu xanh dương

+ Kết quả sau khi chạy chương trình 3 lần ta sẽ có các mê cung như sau

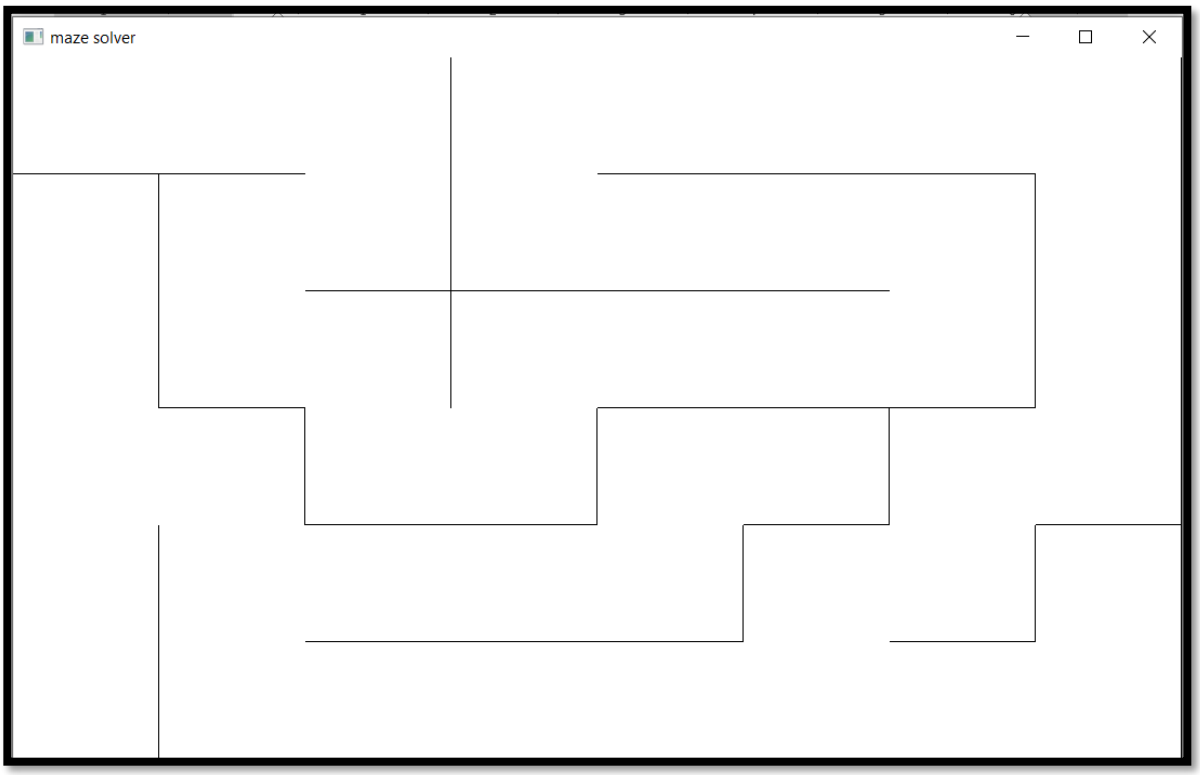
Lần 1 :



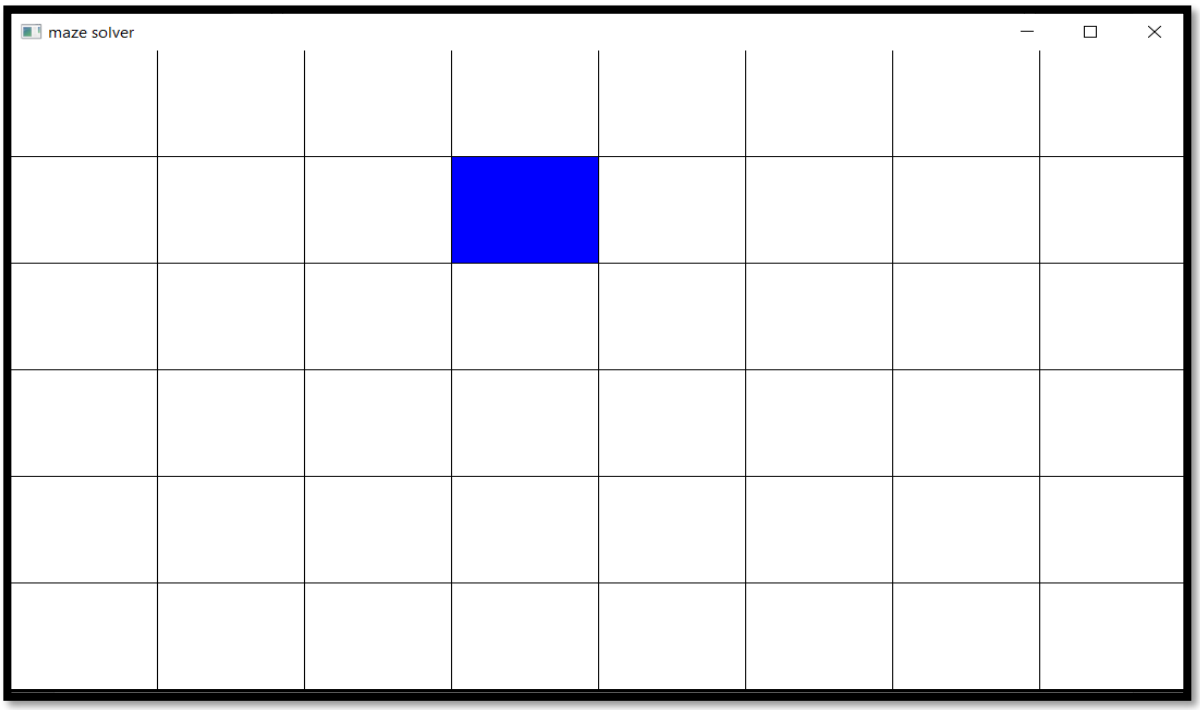
Lần 2:



Lần 3:

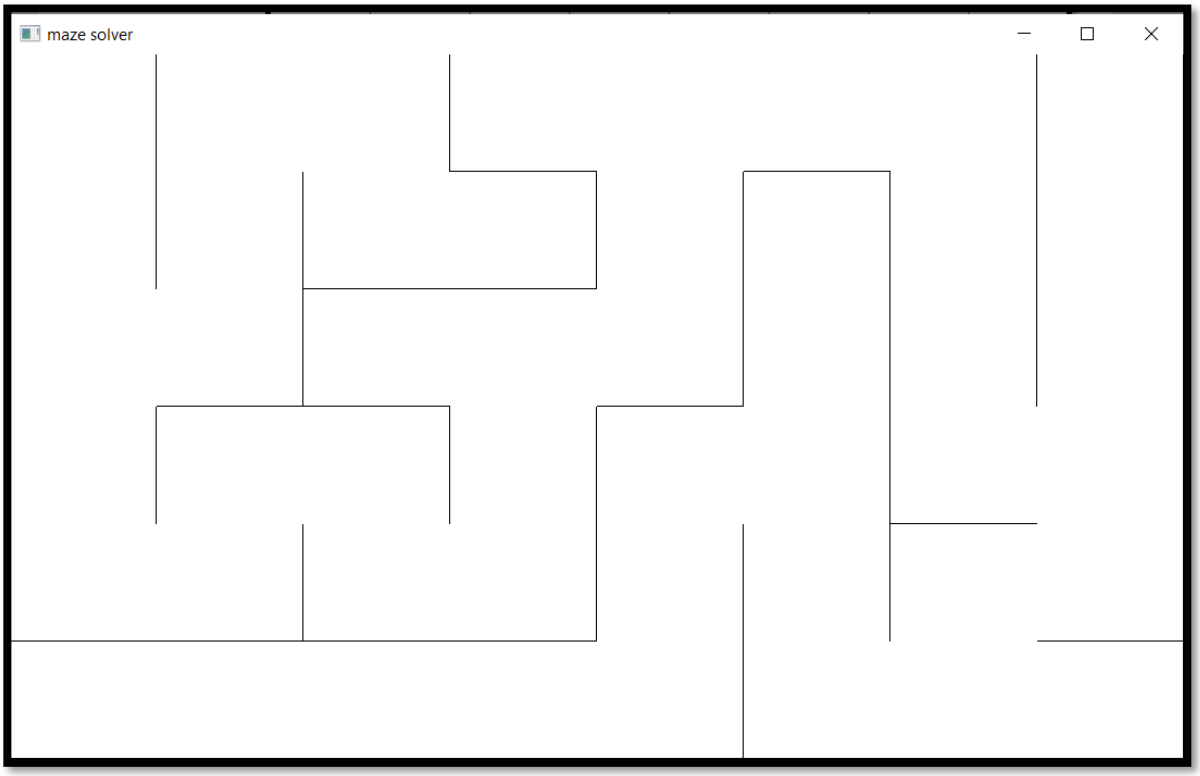


Ta có thể thấy sau 3 lần tạo ra 1 mê cung (8x6) từ ô bắt đầu như trên thì ta tạo được 1 cách ngẫu nhiên mê cung để chứng minh lời khẳng định của mình là đúng thì em sẽ cho ô bắt đầu tại ô như hình sau:

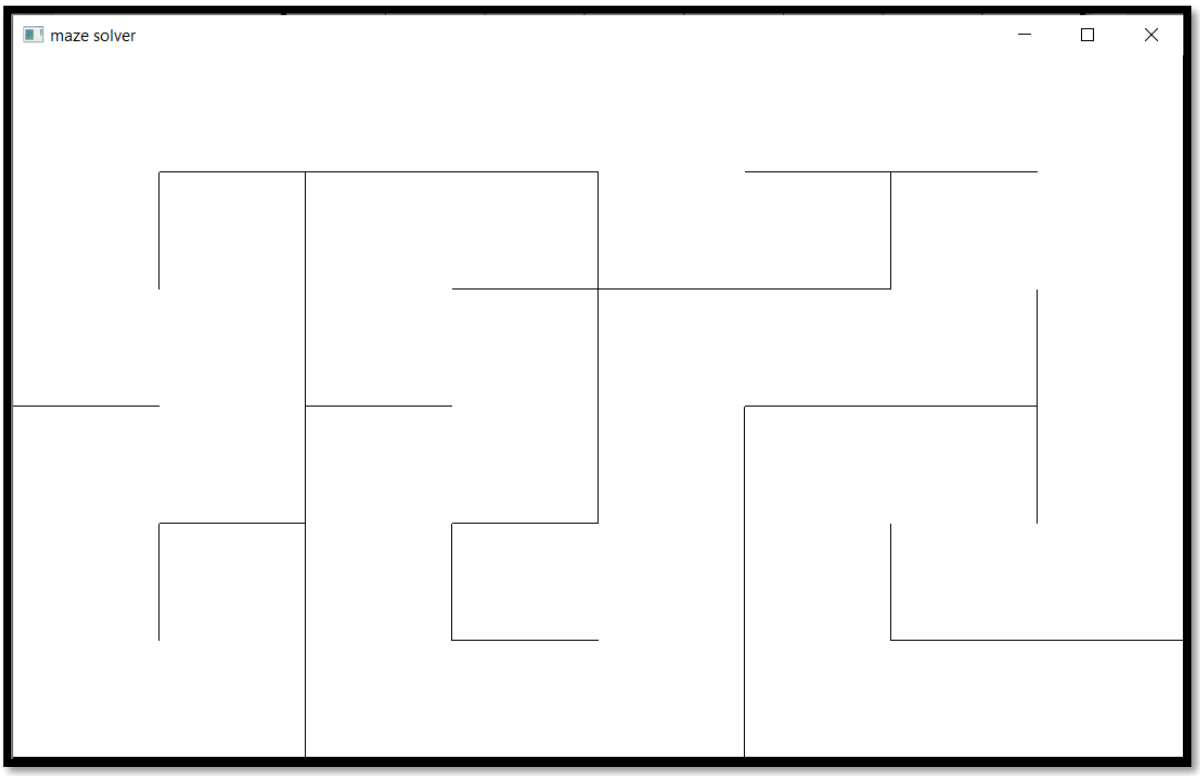


Kết quả sau khi ta tạo maze bắt đầu tại ô trên là

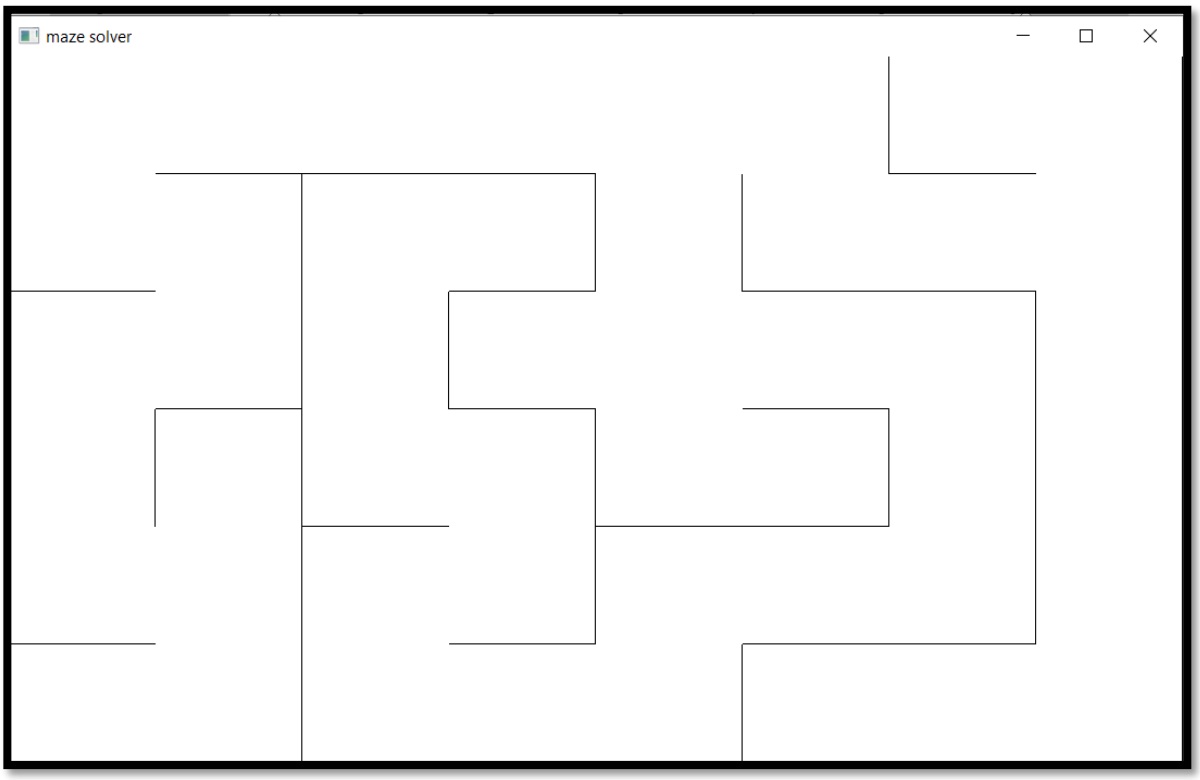
Lần 1:



Lần 2 :



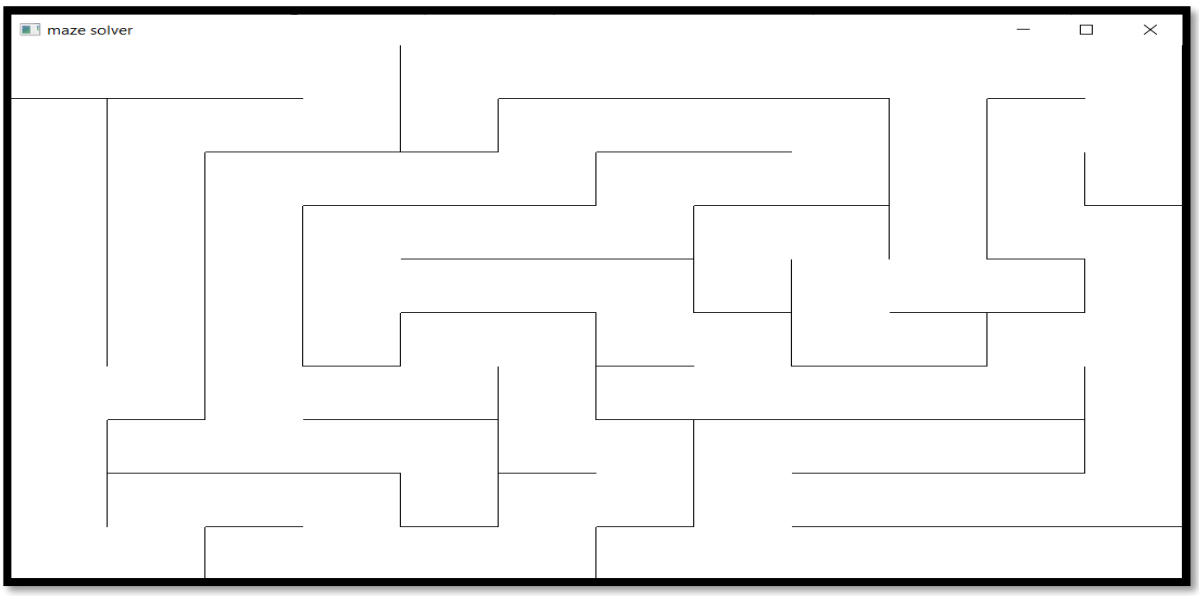
Lần 3 :



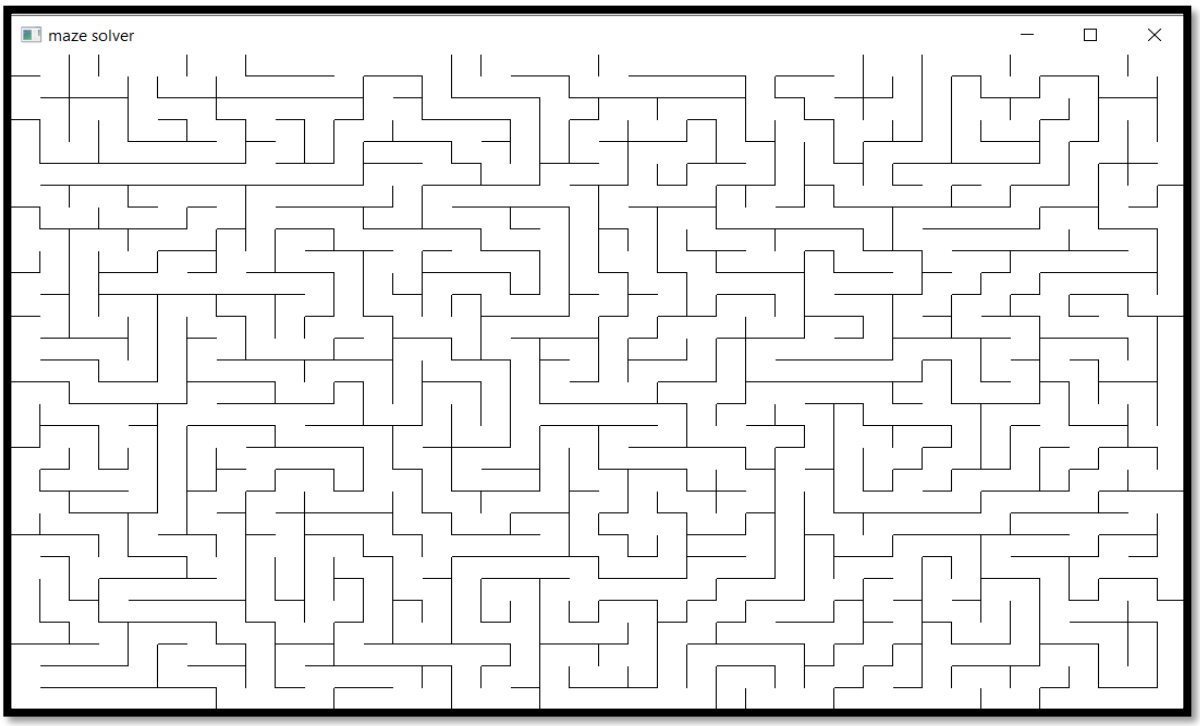
+ Ta có thể thấy được rằng tại bất kì ô nào trên cửa sổ ta sẽ dễ dàng tạo ra 1 mê cung 1 cách ngẫu nhiên

+ Đối với các mê cung khác có kích cỡ (m*n) thì ta cũng dễ dàng tạo bất kì một mê cung

VD: Mê cung kích cỡ 12*10

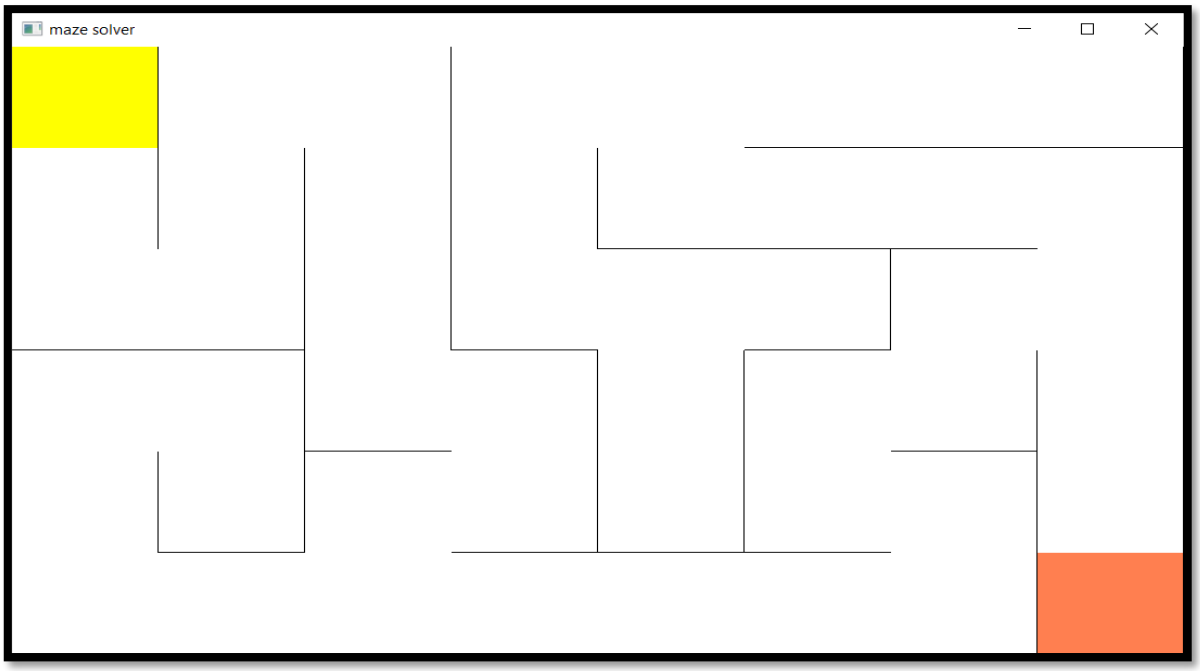


VD : Mê cung có kích cỡ (40x30)



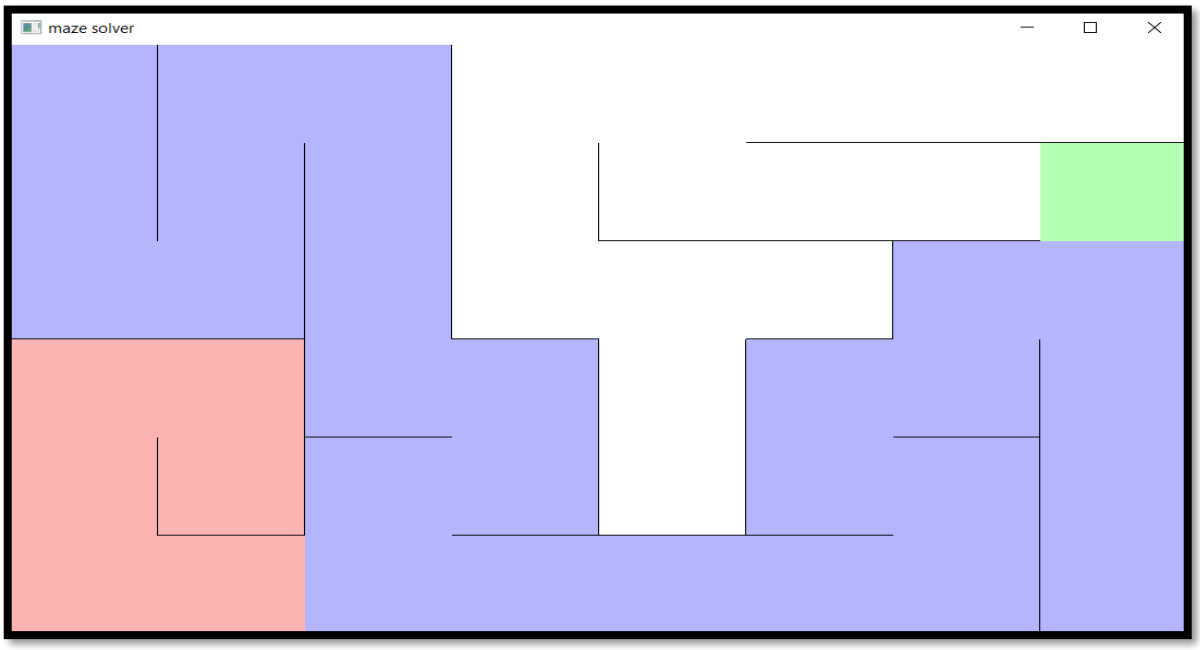
2. Giải mê cung

Test case đầu tiên : em sẽ cho giải mê cung ô bắt đầu và kết thúc như hình sau:



+ Ô bắt đầu là ô màu vàng , Ô kết thúc là ô màu cam

Kết quả sau khi giải mê cung như hình sau:

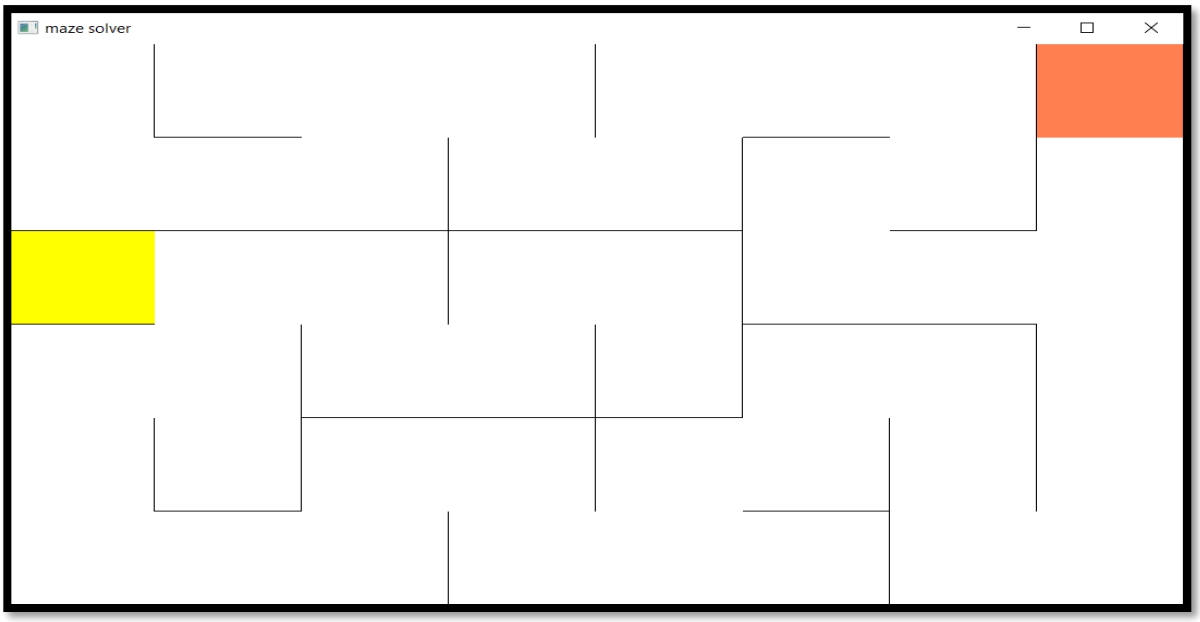


+ Đường đi sẽ là các ô màu tím

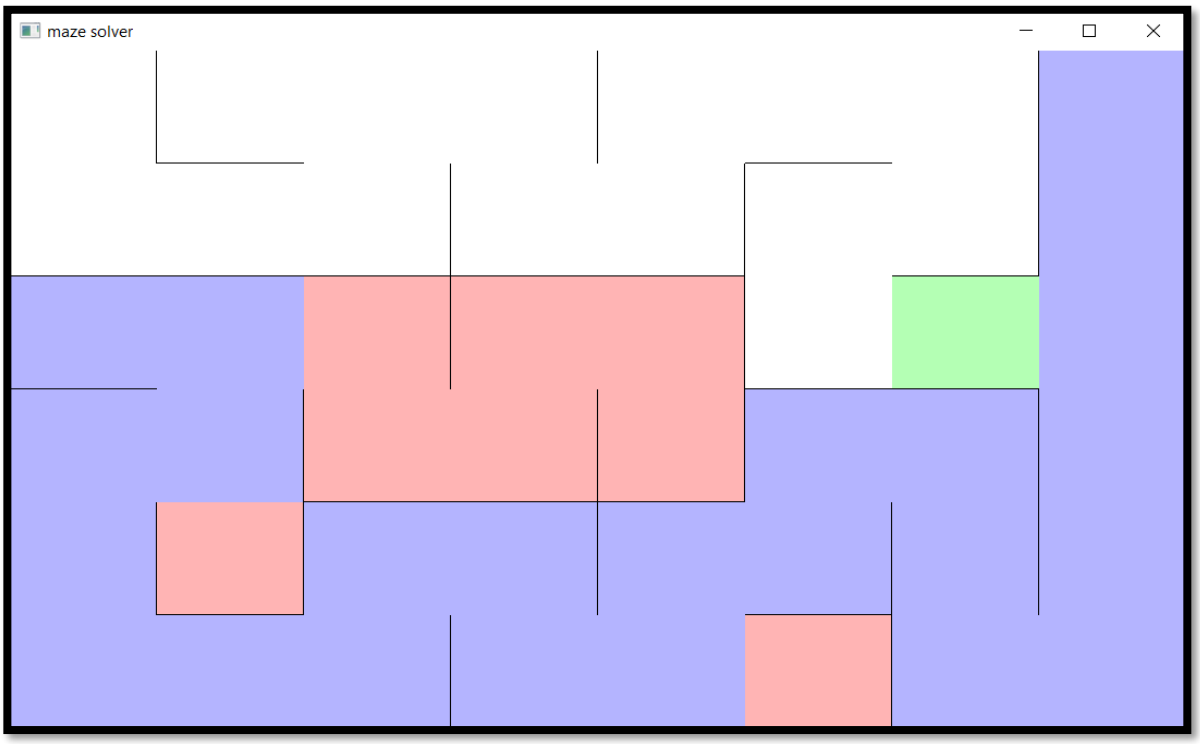
+ Ô màu đỏ hồng sẽ là ô không đi qua

+ Ô màu xanh lá là dung cho bộ trợ tìm đường đi

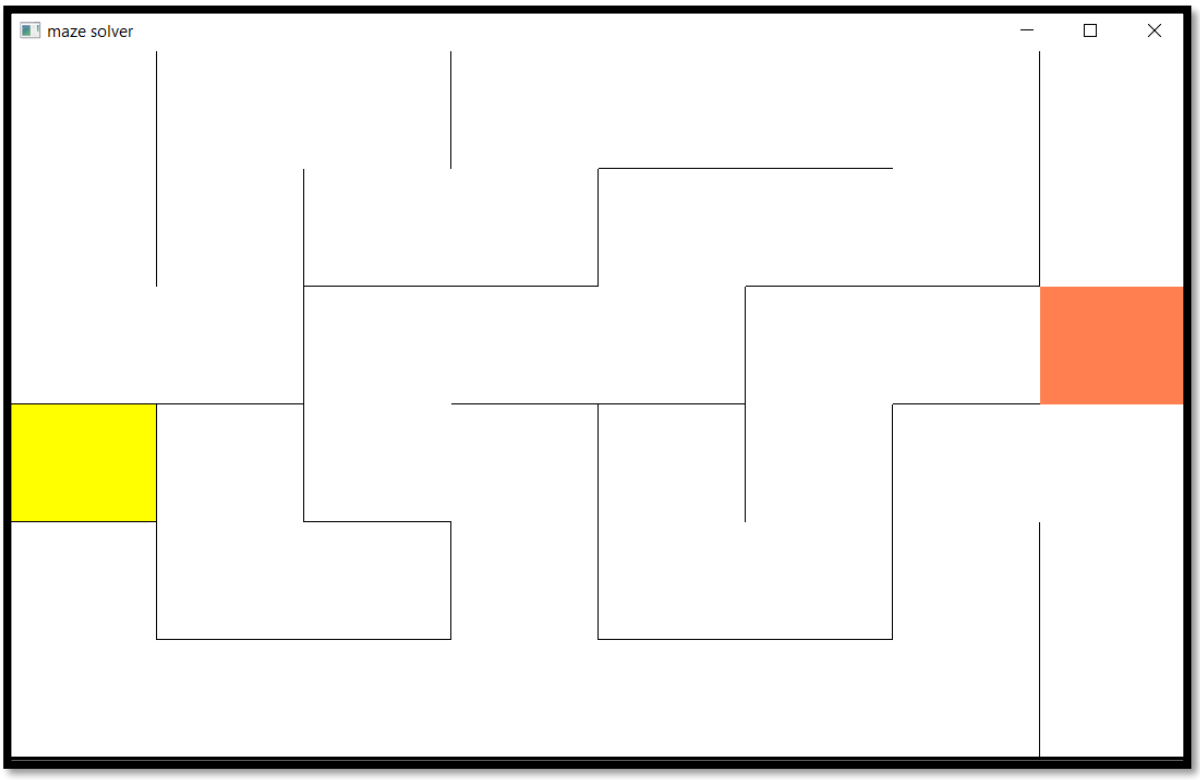
- Test thứ 2 : Thì em sẽ thay đổi ô bắt đầu và ô kết thúc như hình dưới :



+ Hình ảnh sau khi giải mê cung là :

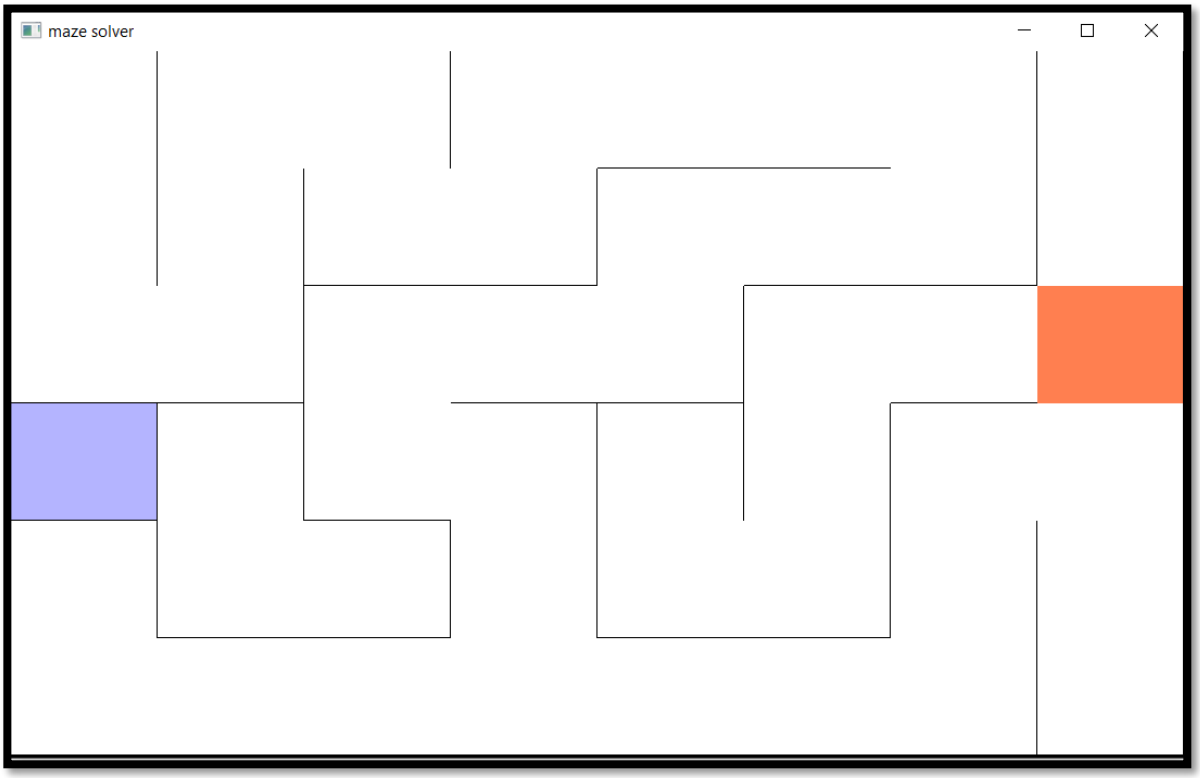


+ Test case thứ 3 : là chỉ cần 1 trong 2 ô bắt đầu và ô kết thúc là đều được bao quanh bởi tường
VD: Như hình sau :



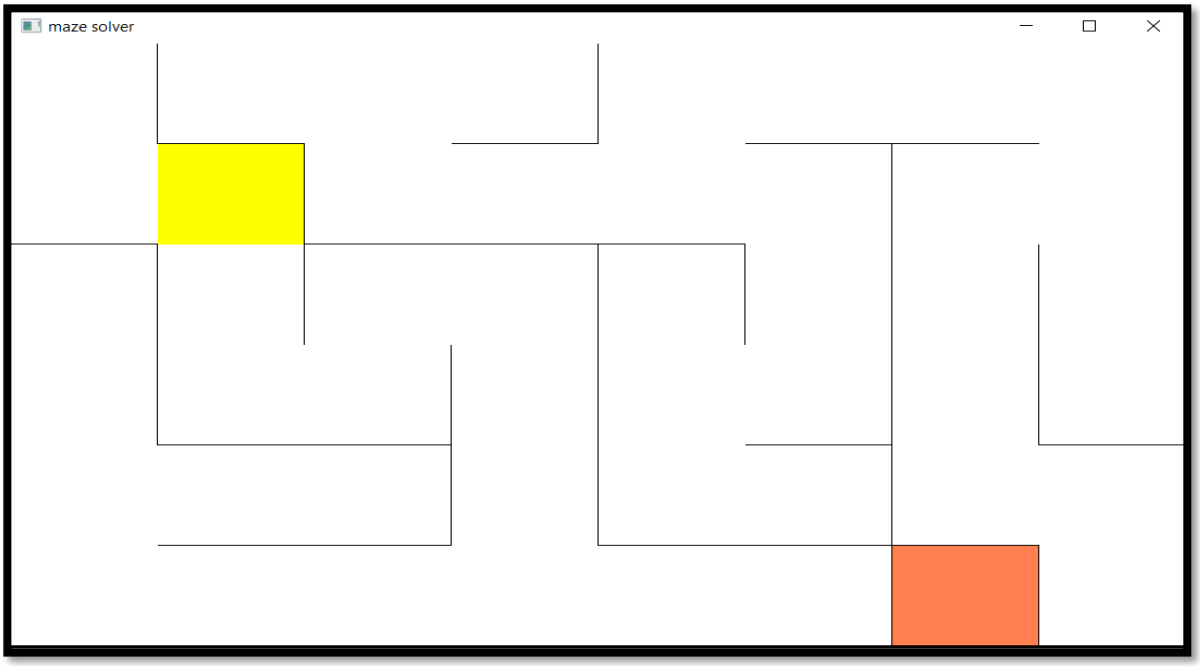
+ Ta có thể thấy ô bắt đầu đều được bao quanh bởi tường

Kết quả sẽ như sau:

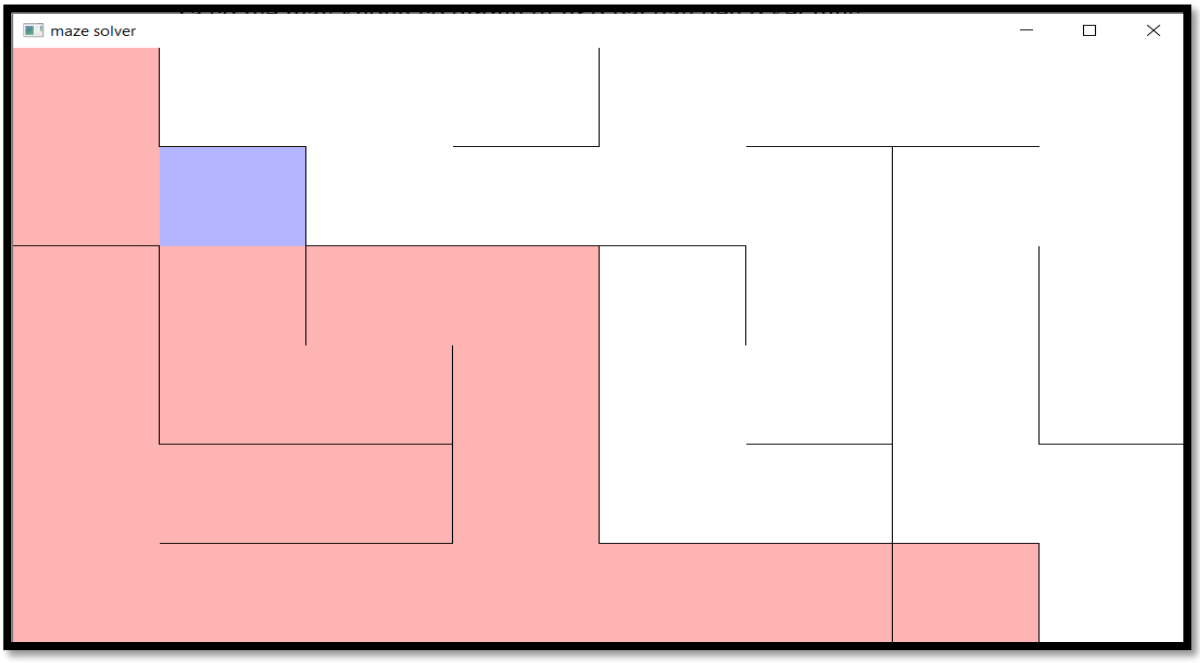


Ta có thể thấy không có đường đi từ ô bắt đầu đến ô kết thúc

VD : Nếu ô bị chặn là ô kết thúc



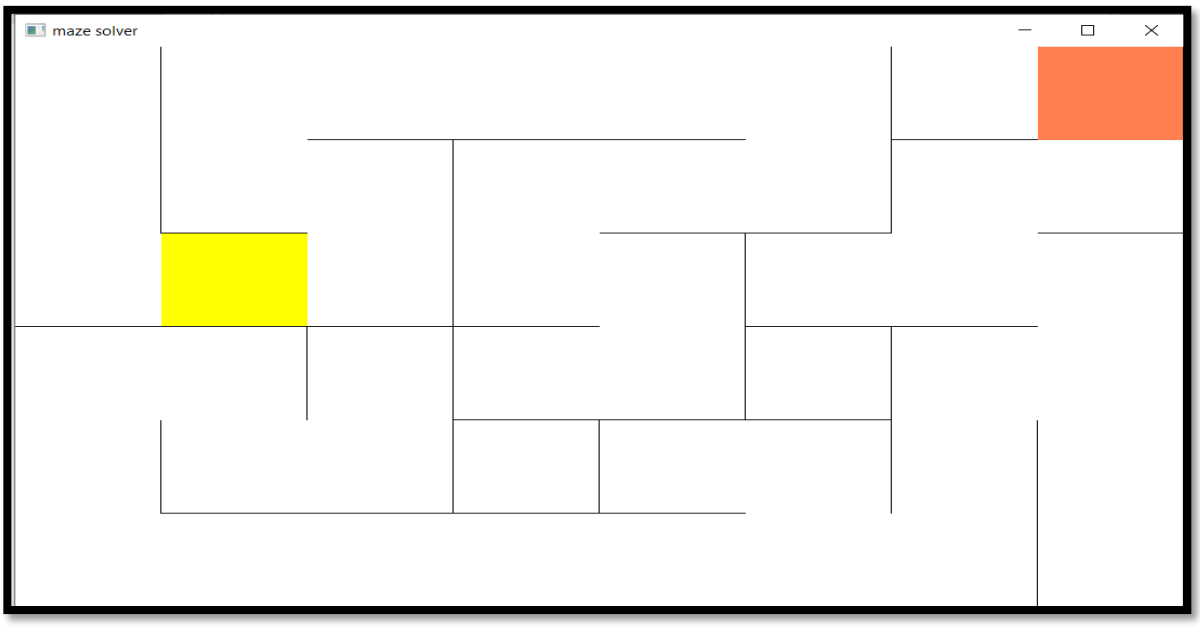
Kết quả :



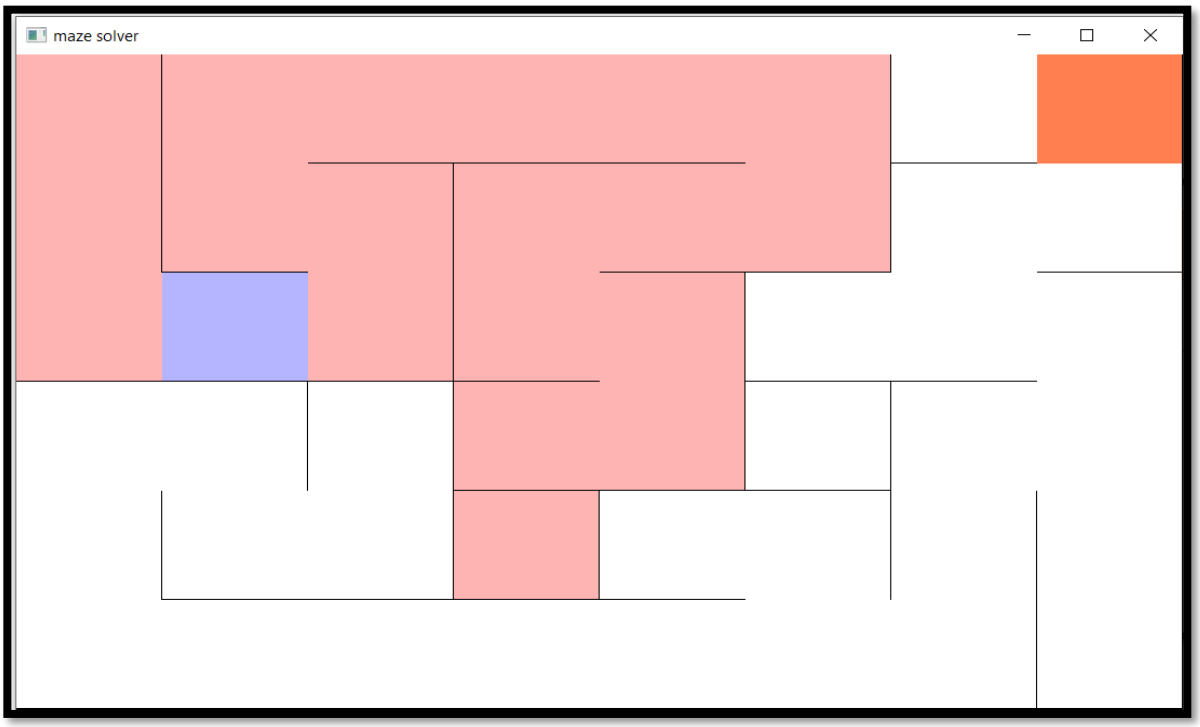
Dễ dàng thấy thì không có đường đi từ ô bắt đầu đến kết thúc

Test case thứ 4 : Là giữa đường đi của 2 ô có các vật cản làm ko thể tìm được đường đi

VD : Như hình dưới đi :

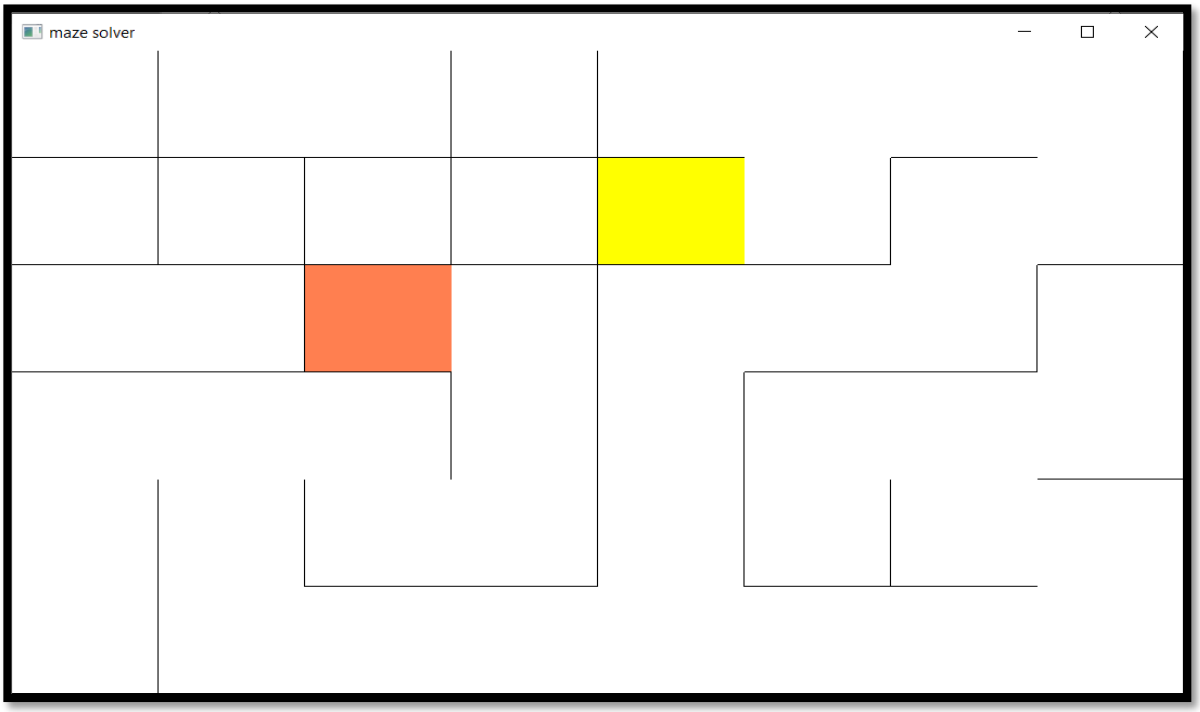


= Ta dễ dàng thấy được rằng trên đường đi từ ô bắt đầu và kết thúc ta gặp phải các bức tường kết quả sẽ như sau :

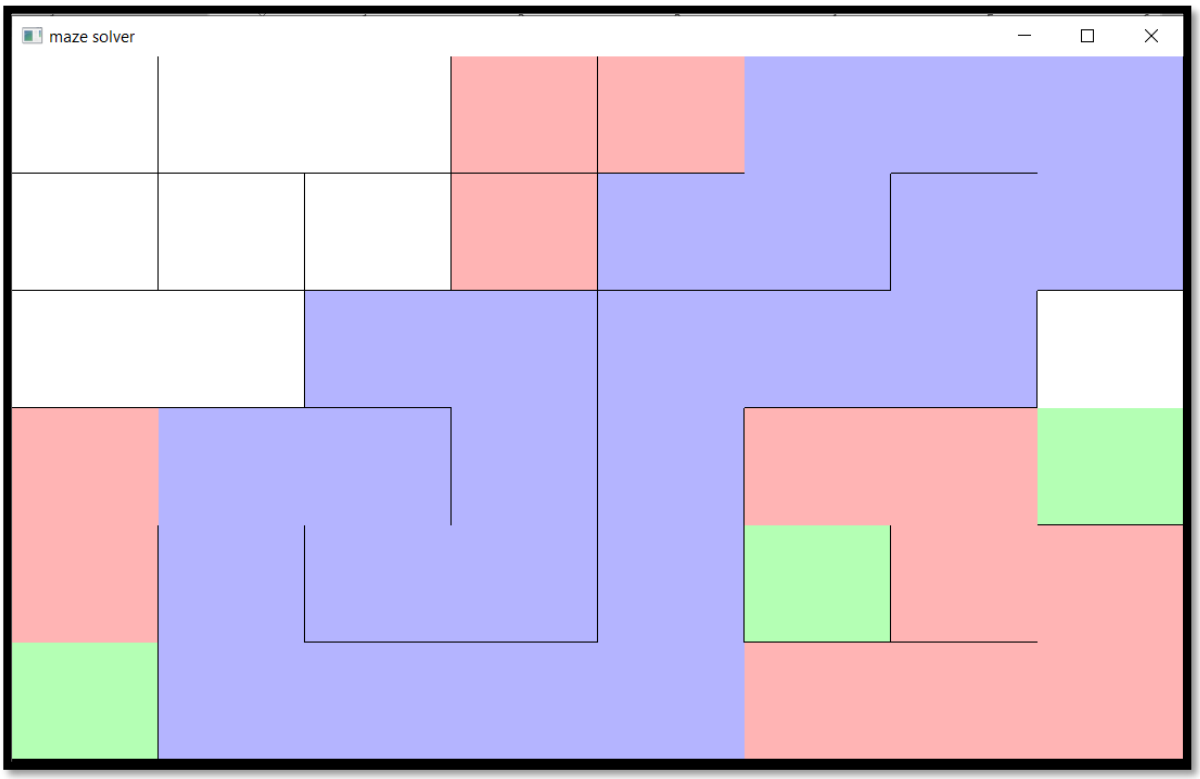


Ta sẽ không thể tìm được đường đi

VD : Trong trường hợp các bức tường không ảnh hưởng đến đường đi từ ô bắt đầu đến ô kết thúc



+ Kết quả như sau :



3. Kết luận và Hướng phát triển

- Với tất cả các trường hợp có thể xảy ra mà nhóm em có thể nghĩ trong việc tạo và giải mê cung thì tất cả các trường hợp thì hai thuật toán nhóm em chọn đều hỗ trợ rất tốt và đưa ra đáp án mà tụi em cần.
- Với hướng phát triển : Thì em đang nghĩ đến là biến chương trình thành một tựa game mê cung và để cho nó hấp dẫn thì em sẽ đặt một con bot để bám đuổi người chơi và trên đường đi thì em sẽ tạo ngẫu nhiên xu để người chơi chạy qua để tang điểm số để làm cho chơi mê cung không nhằm chán.

PHẦN 7: TÀI LIỆU THAM KHẢO

1. Các trang web tham khảo

[1] Thuật toán tạo maze, https://tinyurl.com/cfaxt9ak

[2] Thuật toán A*, https://tinyurl.com/nmfhfehf

[3] Thư viện sfml , https://www.sfml-dev.org/index.php

2. Tài Liệu

- SFML Game Development By Example