



## RAPPORT DE PROJET

# DEVELOPPEMENT D'UN JEU LABYRINTHE (MAZE) EN C++ AVEC RAYLIB

---

Réalisé(e) par:

EL GHORRIM Oumaima

EL KHATAB Sohaib

BOUFANZI Mohamed said

HAFDAN Abderrahmane

Encadrée par:

Pr. BENABDELOUAHAB Ikram

**2024/2025**

# Sommaire

<b>INTRODUCTION</b>	1
<b>DIAGRAMME DE CLASSE</b>	2
<b>CODE SOURCE</b>	3
<b>DÉFINITION DES BIBLIOTHÈQUES</b>	
<b>CONFIGURATION DE L'ÉCRAN</b>	
<b>CLASSE CELL</b>	
<b>CLASSE MAZE</b>	
<b>CLASSE PLAYER</b>	
<b>CLASSE LEVEL</b>	
<b>ÉNUMÉRATION GAMESTATE</b>	
<b>VARIABLES GLOBALES</b>	
<b>FONCTIONS DE DESSIN</b>	
<b>CONCLUSION</b>	4



# Introduction

1

Le jeu **Star Wars Maze** est un jeu de labyrinthe développé en utilisant la bibliothèque Raylib. Le jeu permet aux joueurs de choisir un personnage et un niveau de difficulté, puis de naviguer à travers un labyrinthe généré aléatoirement pour atteindre la sortie. Ce rapport détaille les différentes parties du code, y compris les classes et les fonctions utilisées.



# diagramme de classe

2

## classDiagram

```
class Cell {
    - int x
    - int y
    - bool walls[4]
    + Cell(int x, int y)
    + void Draw(int cellSize, Color wallColor, int offsetX, int offsetY)
    + void RemoveWall(int direction)
    + bool HasWall(int direction) const
}

class Maze {
    - vector<vector<Cell>> grid
    - int width
    - int height
    - int cellSize
    - vector<Vector2> solution
    - Texture2D startTexture
    - Texture2D endTexture
    + Maze(int width, int height, int cellSize)
    - void GenerateMaze()
    - void FindSolution()
    + void Draw(Color wallColor, int offsetX, int offsetY)
    + void DrawSolution(int offsetX, int offsetY)
    + bool CanMove(int x, int y, int direction) const
    + int GetWidth() const
    + int GetHeight() const
    + int GetCellSize() const
}

class Player {
    - float x
    - float y
    - Texture2D texture
    + Player(int startX, int startY, Texture2D playerTexture)
    + void Draw(int cellSize, int offsetX, int offsetY)
    + void Move(float dx, float dy)
    + int GetX() const
    + int GetY() const
}

class Level {
    - int difficulty
    - int mazeSize
    + Level(int difficulty)
    + int GetMazeSize() const
}
```

# Code Source

3

## Définition des Bibliothèques

### #include <raylib.h>

Raylib est une bibliothèque simple et facile à utiliser pour le développement de jeux et d'applications multimédia. Elle fournit des fonctionnalités pour le rendu graphique, la gestion des entrées, le chargement des textures, et bien plus encore.

### #include <vector>

La bibliothèque <vector> fait partie de la bibliothèque standard C++ (STL) et fournit une classe de conteneur dynamique appelée std::vector. Un std::vector est un tableau dynamique qui peut changer de taille automatiquement.

### #include <stack>

La bibliothèque <stack> fait partie de la bibliothèque standard C++ (STL) et fournit une classe de conteneur adaptateur appelée std::stack. Un std::stack est une structure de données LIFO (Last In, First Out).

### #include <queue>

La bibliothèque <queue> fait partie de la bibliothèque standard C++ (STL) et fournit une classe de conteneur adaptateur appelée std::queue. Un std::queue est une structure de données FIFO (First In, First Out).

# Code Source

3

## Définition des Bibliothèques

### #include <cstdlib>

La bibliothèque `<cstdlib>` fait partie de la bibliothèque standard C++ et fournit des fonctions de gestion de la mémoire, des conversions de types, et des fonctions utilitaires comme `rand()` et `srand()`.

### #include <ctime>

La bibliothèque `<ctime>` fait partie de la bibliothèque standard C++ et fournit des fonctions pour manipuler le temps et la date, comme `time()` et `localtime()`.

### #include <algorithm>

La bibliothèque `<algorithm>` fait partie de la bibliothèque standard C++ (STL) et fournit un ensemble d'algorithmes génériques pour manipuler des conteneurs, comme `std::sort`, `std::find`, et `std::random_shuffle`.

### #include <string>

La bibliothèque `<string>` fait partie de la bibliothèque standard C++ et fournit la classe `std::string` pour manipuler des chaînes de caractères.

# Code Source

3

## Configuration de l'Écran

```
const int SCREEN_WIDTH = 1536;
```

```
const int SCREEN_HEIGHT = 864;
```

Ces constantes définissent la largeur et la hauteur de la fenêtre de jeu.

```
const int SCREEN_WIDTH = 1536;
const int SCREEN_HEIGHT = 864;
```

## Classe Cell

La classe Cell représente une cellule individuelle dans le labyrinthe. Chaque cellule a des murs sur ses quatre côtés (haut, droite, bas, gauche).

```
int highestScore = 0;

class Cell {
private:
    int x, y;
    bool walls[4];

public:
    Cell(int x = 0, int y = 0) : x(x), y(y) {
        for (int i = 0; i < 4; i++) {
            walls[i] = true;
        }
    }

    void Draw(int cellSize, Color wallColor, int offsetX, int offsetY) {
        int screenX = x * cellSize + offsetX;
        int screenY = y * cellSize + offsetY;

        if (walls[0]) DrawLine(screenX, screenY, screenX + cellSize, screenY, wallColor);
        if (walls[1]) DrawLine(screenX + cellSize, screenY, screenX + cellSize, screenY + cellSize, wallColor);
        if (walls[2]) DrawLine(screenX, screenY + cellSize, screenX + cellSize, screenY + cellSize, wallColor);
        if (walls[3]) DrawLine(screenX, screenY, screenX, screenY + cellSize, wallColor);
    }

    void RemoveWall(int direction) {
        walls[direction] = false;
    }
}
```

# Code Source

3

## Attributs

- **x, y** : Coordonnées de la cellule.
- **walls** : Tableau de booléens indiquant la présence de murs sur chaque côté.

## Méthodes

- **Cell(int x = 0, int y = 0)** : Constructeur initialisant les coordonnées et les murs.
- **void Draw(int cellSize, Color wallColor, int offsetX, int offsetY)** : Dessine les murs de la cellule.
- **void RemoveWall(int direction)** : Enlève un mur dans une direction donnée.
- **bool HasWall(int direction) const** : Vérifie si un mur existe dans une direction donnée.

## Classe Maze

```
class Maze {
private:
    std::vector<std::vector<Cell>> grid;
    int width, height;
    int cellSize;
    std::vector<Vector2> solution;
    Texture2D startTexture;
    Texture2D endTexture;

    void GenerateMaze() {
        std::stack<std::pair<int, int>> stack;
        std::vector<std::vector<bool>> visited(height, std::vector<bool>(width, false));

        stack.push({0, 0});
        visited[0][0] = true;
```

# Code Source

3

La classe Maze représente le labyrinthe complet, composé de plusieurs cellules.

## Attributs

- **grid** : Grille de cellules.
- **width, height** : Dimensions du labyrinthe.
- **cellSize** : Taille de chaque cellule.
- **solution** : Chemin de la solution.
- **startTexture, endTexture** : Textures pour le début et la fin du labyrinthe.

## Méthodes

- **Maze(int width, int height, int cellSize)** : Constructeur initialisant le labyrinthe.
- **~Maze()** : Destructeur libérant les textures.
- **void GenerateMaze()** : Génère le labyrinthe en utilisant un algorithme de backtracking.
- **void FindSolution()** : Trouve le chemin de la solution en utilisant un algorithme de parcours en largeur (BFS).
- **void Draw(Color wallColor, int offsetX, int offsetY)** : Dessine le labyrinthe.
- **void DrawSolution(int offsetX, int offsetY)** : Dessine le chemin de la solution.
- **bool CanMove(int x, int y, int direction) const** : Vérifie si un mouvement est possible dans une direction donnée.

# Code Source

3

- **int GetWidth() const, int GetHeight() const, int GetCellSize() const :** Accesseurs pour les dimensions et la taille des cellules.

## Classe Player

La classe Player représente le joueur dans le labyrinthe.

```
class Player {  
private:  
    float x, y;  
    Texture2D texture;  
  
public:  
    Player(int startX, int startY, Texture2D playerTexture) : x(startX), y(startY), texture(playerTexture) {}  
  
    void Draw(int cellSize, int offsetX, int offsetY) {  
        float scale = (float)cellSize / std::max(texture.width, texture.height);  
        DrawTextureEx(texture, {x * cellSize + offsetX, y * cellSize + offsetY}, 0, scale, WHITE);  
    }  
  
    void Move(float dx, float dy) {  
        x += dx;  
        y += dy;  
    }  
  
    int GetX() const { return static_cast<int>(x); }  
    int GetY() const { return static_cast<int>(y); }  
};
```

## Attributs

- **x, y :** Coordonnées du joueur.
- **texture :** Texture du joueur.

## Méthodes

- **Player(int startX, int startY, Texture2D playerTexture) :** Constructeur initialisant les coordonnées et la texture.
- **void Draw(int cellSize, int offsetX, int offsetY) :** Dessine le joueur.
- **void Move(float dx, float dy) :** Déplace le joueur.
- **int GetX() const, int GetY() const :** Accesseurs pour les coordonnées.

# Code Source

3

## Classe Level

- La classe Level représente un niveau de difficulté du jeu.

```
class Level {  
private:  
    int difficulty;  
    int mazeSize;  
  
public:  
    Level(int difficulty) : difficulty(difficulty) {  
        switch (difficulty) {  
            case 1: mazeSize = 10; break;  
            case 2: mazeSize = 15; break;  
            case 3: mazeSize = 20; break;  
            default: mazeSize = 10; break;  
        }  
    }  
  
    int GetMazeSize() const { return mazeSize; }  
};
```

## Attributs

- **difficulty** : Niveau de difficulté.
- **mazeSize** : Taille du labyrinthe en fonction de la difficulté.

## Méthodes

- **Level(int difficulty)** : Constructeur initialisant la difficulté et la taille du labyrinthe.
- **int GetMazeSize() const** : Accesseur pour la taille du labyrinthe.

## Énumération GameState

L'énumération GameState représente les différents états du jeu.

# Code Source

3

```
enum class GameState {  
    FIRST_SCREEN,  
    CHARACTER_SELECTION,  
    LEVEL_SELECTION,  
    PLAYING,  
    GAME_OVER,  
    VICTORY  
};
```

## Variables Globales

Ces variables globales gèrent l'état du jeu, les textures, et les objets principaux.

```
GameState currentState = GameState::FIRST_SCREEN;  
int selectedCharacter = 0;  
int selectedLevel = 0;  
float gameTimer = 0.0f;  
int lastScore = 0;  
bool showSolution = false;  
  
Maze* maze = nullptr;  
Player* player = nullptr;  
Level* level = nullptr;  
  
Texture2D player3Texture;  
Texture2D player1Texture;  
Texture2D player2Texture;  
Texture2D starWarsBackground;  
  
Music spaceMusic;
```

## Fonctions de Dessin

# Code Source

3

ces fonctions gèrent le rendu des différents écrans du jeu.

## DrawFirstScreen()

Dessine l'écran d'accueil avec le titre, le sous-titre, les boutons de démarrage et de sortie, et le score le plus élevé.

```
void DrawFirstScreen() {
    DrawTexture(starWarsBackground, 0, 0, WHITE);

    const char* titleText = "Star Wars Maze";
    int titleFontSize = 70;
    int titleWidth = MeasureText(titleText, titleFontSize);
    int titleX = (SCREEN_WIDTH - titleWidth) / 2;

    DrawText(titleText, titleX, 100, titleFontSize, GOLD);

    const char* subtitleText = "Navigate through the maze to win!";
    int subtitleFontSize = 30;
    int subtitleWidth = MeasureText(subtitleText, subtitleFontSize);
    int subtitleX = (SCREEN_WIDTH - subtitleWidth) / 2;

    DrawText(subtitleText, subtitleX, 200, subtitleFontSize, RAYWHITE);

    int buttonWidth = 200, buttonHeight = 60;
    int buttonX = (SCREEN_WIDTH - buttonWidth) / 2;
    Rectangle startButton = { (float)buttonX, 400, (float)buttonWidth, (float)buttonHeight };
    DrawRectangleRounded(startButton, 0.2f, 10, DARKGREEN);
    DrawText("Start", buttonX + (buttonWidth - MeasureText("Start", 30)) / 2, 415, 30, WHITE);

    Rectangle exitButton = { (float)buttonX, 500, (float)buttonWidth, (float)buttonHeight };
    DrawRectangleRounded(exitButton, 0.2f, 10, MAROON);
    DrawText("Exit", buttonX + (buttonWidth - MeasureText("Exit", 30)) / 2, 515, 30, WHITE);
```

## Explication :

- Dessine l'écran d'accueil avec le titre, le sous-titre, les boutons de démarrage et de sortie, et le score le plus élevé.
- Utilise DrawTexture pour afficher l'arrière-plan.
- Utilise DrawText pour afficher le titre et le sous-titre.
- Dessine les boutons "Start" et "Exit" avec DrawRectangleRounded et DrawText.

# Code Source

3

- Gère les interactions avec les boutons pour changer l'état du jeu ou fermer la fenêtre.

## DrawCharacterSelection()

```
void DrawCharacterSelection() {
    DrawTexture(starWarsBackground, 0, 0, WHITE);

    const char* titleText = "Choose Your Character";
    int titleFontSize = 50;
    int titleWidth = MeasureText(titleText, titleFontSize);
    int titleX = (SCREEN_WIDTH - titleWidth) / 2;

    DrawText(titleText, titleX, 100, titleFontSize, GOLD);

    int buttonWidth = 100, buttonHeight = 100;
    int imageSize = 40;
    int spacing = 50;
    int totalWidth = 3 * buttonWidth + 2 * spacing;
    int startX = (SCREEN_WIDTH - totalWidth) / 2;
```

- Dessine l'écran de sélection de personnage avec les boutons pour choisir un personnage.
- Utilise DrawTexture pour afficher l'arrière-plan.
- Utilise DrawText pour afficher le titre.
- Dessine les boutons pour chaque personnage avec DrawRectangleRounded et DrawTextureEx.
- Gère les interactions avec les boutons pour changer l'état du jeu et sélectionner un personnage.

# Code Source

3

## DrawLevelSelection()

```
void DrawLevelSelection() {
    DrawTexture(starWarsBackground, 0, 0, WHITE);

    const char* titleText = "Choose Difficulty";
    int titleFontSize = 50;
    int titleWidth = MeasureText(titleText, titleFontSize);
    int titleX = (SCREEN_WIDTH - titleWidth) / 2;

    DrawText(titleText, titleX, 100, titleFontSize, GOLD);

    int buttonWidth = 200, buttonHeight = 60;
    int buttonX = (SCREEN_WIDTH - buttonWidth) / 2;
```

- Dessine l'écran de sélection de niveau avec les boutons pour choisir la difficulté.
- Utilise DrawTexture pour afficher l'arrière-plan.
- Utilise DrawText pour afficher le titre.
- Dessine les boutons pour chaque niveau de difficulté avec DrawRectangleRounded et DrawText.
- Gère les interactions avec les boutons pour changer l'état du jeu et sélectionner un niveau de difficulté.
- Initialise le labyrinthe, le joueur, et les variables de jeu lorsque l'état passe à PLAYING.

# Code Source

3

## DrawGameScreen()

```
void DrawGameScreen() {
    DrawTexture(starWarsBackground, 0, 0, WHITE);

    if (maze && player) {
        gameTimer += GetFrameTime();

        int offsetX = (SCREEN_WIDTH - maze->GetWidth() * maze->GetCellSize()) / 2;
        int offsetY = (SCREEN_HEIGHT - maze->GetHeight() * maze->GetCellSize()) / 2;

        maze->Draw(LIGHTGRAY, offsetX, offsetY);
        if (showSolution) {
            maze->DrawSolution(offsetX, offsetY);
        }
        player->Draw(maze->GetCellSize(), offsetX, offsetY);

        DrawRectangle(0, 0, SCREEN_WIDTH, 50, Fade(BLACK, 0.5f));
        DrawText(TextFormat("Time: %.2f", gameTimer), 10, 10, 30, WHITE);
    }
}
```

- Dessine l'écran de jeu principal, y compris le labyrinthe, le joueur, le chronomètre, et gère les mouvements du joueur.
- Utilise DrawTexture pour afficher l'arrière-plan.
- Dessine le labyrinthe avec maze->Draw.
- Dessine la solution du labyrinthe si showSolution est vrai avec maze->DrawSolution.
- Dessine le joueur avec player->Draw.
- Affiche le chronomètre avec DrawText.
- Gère les mouvements du joueur avec les touches fléchées et vérifie les collisions avec les murs du labyrinthe.
- Affiche ou masque la solution du labyrinthe avec la touche espace.
- Vérifie si le joueur a atteint la sortie pour passer à l'état de victoire et met à jour le score.

# Code Source

3

## DrawVictoryScreen()

```
void DrawVictoryScreen() {
    DrawTexture(starWarsBackground, 0, 0, WHITE);

    const char* victoryText = "Congratulations! You Won!";
    int victoryFontSize = 60;
    int victoryWidth = MeasureText(victoryText, victoryFontSize);
    int victoryX = (SCREEN_WIDTH - victoryWidth) / 2;

    DrawText(victoryText, victoryX, 100, victoryFontSize, GOLD);

    const char* scoreText = TextFormat("Your Score: %d", lastScore);
    int scoreFontSize = 40;
    int scoreWidth = MeasureText(scoreText, scoreFontSize);
    int scoreX = (SCREEN_WIDTH - scoreWidth) / 2;

    DrawText(scoreText, scoreX, 200, scoreFontSize, WHITE);

    int buttonWidth = 250, buttonHeight = 60;
    int buttonX = (SCREEN_WIDTH - buttonWidth) / 2;
```

- Dessine l'écran de victoire avec le score du joueur et les boutons pour rejouer ou retourner au menu principal.
- Utilise DrawTexture pour afficher l'arrière-plan.
- Utilise DrawText pour afficher le message de victoire et le score du joueur.
- Dessine les boutons "Play Again" et "Main Menu" avec DrawRectangleRounded et DrawText.
- Gère les interactions avec les boutons pour changer l'état du jeu.

# Code Source

3

## Fonction main()

La fonction main() initialise la fenêtre de jeu, charge les textures et la musique, et gère la boucle principale du jeu.

```
int main() {
    InitWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "Star Wars Maze");
    SetTargetFPS(60);

    player3Texture = LoadTexture("src/player3.png");
    player1Texture = LoadTexture("src/player1.png");
    player2Texture = LoadTexture("src/player2.png");
    starWarsBackground = LoadTexture("src/star_wars.png");

    InitAudioDevice();
    spaceMusic = LoadMusicStream("src/music.mp3");
    PlayMusicStream(spaceMusic);

    while (!WindowShouldClose()) {
        UpdateMusicStream(spaceMusic);

        BeginDrawing();
        ClearBackground(RAYWHITE);

        switch (currentState) {
            case GameState::FIRST_SCREEN:
                DrawFirstScreen();
                break;
            case GameState::CHARACTER_SELECTION:
                DrawCharacterSelection();
                break;
            case GameState::LEVEL_SELECTION:
                DrawLevelSelection();
                break;
            case GameState::PLAYING:
                DrawGameScreen();
                break;
            case GameState::VICTORY:
                DrawVictoryScreen();
                break;
        }
    }
}
```

# Code Source

3

```
    EndDrawing () ;  
}  
  
UnloadTexture (player3Texture) ;  
UnloadTexture (player1Texture) ;  
UnloadTexture (player2Texture) ;  
UnloadTexture (starWarsBackground) ;  
UnloadMusicStream (spaceMusic) ;  
CloseAudioDevice () ;  
  
CloseWindow () ;  
return 0;  
}
```

# Conclusion

4

Le jeu "Star Wars Maze" est une application interactive qui permet aux joueurs de naviguer à travers un labyrinthe généré aléatoirement. Le code est bien structuré avec des classes pour les cellules, le labyrinthe, le joueur, et les niveaux, ainsi que des fonctions pour gérer les différents états du jeu et le rendu graphique.