

université abdelmalek essaâdi

Faculté des Sciences et Techniques de Tanger

---

# PROGRAMMATION ORIENTÉE OBJET EN C++

## Rapport Projet RAYLIB

Réalisé Par :

El Mehdi El Khaldi

Ibrahim El Hafdaoui

Oussama Allouch

Encadrée Par : Ikram Ben abdel ouahab

## I. INTRODUCTION :

- Dans le cadre du module de Programmation Orientée Objet (POO) en C++, ce projet vise à concevoir et développer un jeu de casse-tête interactif de type labyrinthe à l'aide de la bibliothèque graphique Raylib. Ce jeu propose une expérience immersive où les joueurs doivent naviguer à travers des labyrinthes générés de manière aléatoire pour atteindre une sortie.
- L'objectif est de fournir une expérience ludique tout en renforçant les concepts de POO grâce à des fonctionnalités avancées telles que la génération procédurale, la gestion des niveaux de difficulté en utilisant des boutons de niveau (EASY, MEDIUM, HARD) avec Chaque partie débutera avec la création d'un nouveau labyrinthe, et les joueurs pourront choisir parmi trois niveaux de difficulté, influençant la taille et la complexité des labyrinthes. Chaque labyrinthe possède un chrono qui calcule le temps émis pour atteindre l'objectif, après avoir atteint l'objectif il y a un affichage d'un message de victoire (YOU WIN en vert) et le chrono à gauche du score de temps, avec la possibilité de revenir rejouer une autre fois en cliquant sur entrer, ce jeu possède quatre interfaces graphiques intuitives : la première correspond à une page d'accueil, après la deuxième possède les niveaux de difficulté, la troisième correspond au labyrinthe et la dernière qui affiche le message après la victoire.
- Le projet met en avant des aspects techniques importants, notamment l'algorithme de génération de labyrinthes, la gestion des collisions, et la conception d'une interface utilisateur fluide et ergonomique.

## Que veut dire Raylib :

- Raylib est une bibliothèque graphique open-source, conçue pour simplifier le développement de jeux vidéo en C et C++. Elle offre des outils puissants pour créer des graphismes 2D et 3D, gérer les entrées utilisateur via clavier, souris ou manettes, et intégrer des sons et musiques de manière intuitive. Compatible avec plusieurs plateformes comme Windows, macOS, Linux, et Android, Raylib se distingue par sa simplicité d'utilisation, idéale pour les débutants. Avec une documentation complète, de nombreux exemples pratiques et une communauté active, elle permet de développer rapidement des jeux avec des fonctionnalités modernes tout en réduisant la complexité technique.

## ➤ LES CLASS :

### ● CLASS BUTTON :

- La classe button définit un bouton interactif avec une image de texture, une position et un facteur de mise à l'échelle. Lors de l'initialisation, elle charge l'image, redimensionne la texture et la prépare pour l'affichage. Elle inclut des méthodes pour dessiner le bouton, détecter si le bouton est pressé par l'utilisateur, et gérer l'état de survol du bouton en fonction de la position de la souris.

```
1 class button {
2 private:
3     Texture2D texture;
4     Vector2 position;
5     float scale;
6     bool isHovered;
7
8 public:
9     button(const char* imagepath, Vector2 imageposition, float scaleFactor) {
10         Image image = LoadImage(imagepath);
11         int originalWidth = image.width;
12         int originalHeight = image.height;
13         int newWidth = static_cast<int>(originalWidth * scaleFactor);
14         int newHeight = static_cast<int>(originalHeight * scaleFactor);
15         ImageResize(&image, newWidth, newHeight);
16         texture = LoadTextureFromImage(image);
17         UnloadImage(image);
18         position = imageposition;
19         scale = scaleFactor;
20         isHovered = false;
21
22         if (texture.id == 0) {
23             std::cerr << "Error loading texture: " << imagepath << std::endl;
24         }
25     }
26
27     ~button() {
28         UnloadTexture(texture);
29     }
30
31     void draw() {
32         Color tintColor = isHovered ? RED : WHITE;
33         DrawTextureEx(texture, position, 0.0f, 1.0f, tintColor);
34     }
35
36     bool ispressed(Vector2 mousePos, bool mousePressed) {
37         Rectangle rect = { position.x, position.y, static_cast<float>(texture.width), static_cast<float>(texture.height) };
38         return CheckCollisionPointRec(mousePos, rect) && mousePressed;
39     }
40
41     void updateHoverState(Vector2 mousePos) {
42         Rectangle rect = { position.x, position.y, static_cast<float>(texture.width), static_cast<float>(texture.height) };
43         isHovered = CheckCollisionPointRec(mousePos, rect);
44     }
45 };
46
```

### Explication du code :

- **button(const char\* imagepath, Vector2 imageposition, float scaleFactor)** : Le constructeur de la classe qui charge l'image, la redimensionne, et initialise la texture et d'autres variables.
- **~button()** : Le destructeur de la classe qui libère la mémoire de la texture.
- **LoadImage(const char\* filename)** : Charge une image depuis un fichier spécifié.
- **ImageResize(Image\* image, int newWidth, int newHeight)** : Redimensionne l'image à une nouvelle largeur et hauteur.
- **LoadTextureFromImage(Image image)** : Convertit une image en texture utilisable pour l'affichage.
- **UnloadImage(Image image)** : Libère la mémoire utilisée par l'image, une fois la texture chargée.
- **UnloadTexture(Texture2D texture)** : Libère la mémoire utilisée par une texture.
- **DrawTextureEx(Texture2D texture, Vector2 position, float rotation, float scale, Color tint)** : Dessine une texture à l'écran avec une position, une rotation, une mise à l'échelle et une teinte de couleur.
- **CheckCollisionPointRec(Vector2 point, Rectangle rec)** : Vérifie si un point (ici la position de la souris) se trouve à l'intérieur d'un rectangle (zone du bouton).

- **draw()** : Fonction qui dessine le bouton à l'écran avec un changement de couleur si la souris le survole.
- **ispressed(Vector2 mousePos, bool mousePressed)** : Vérifie si le bouton a été pressé en fonction de la position de la souris et de l'état du clic.
- **updateHoverState(Vector2 mousePos)** : Met à jour l'état de survol du bouton en fonction de la position de la souris.

## ➤ CLASS MAZE

### ● Explication du code

- **Maze()** : Constructeur par défaut qui initialise un labyrinthe vide.
- **Maze(int w, int h, int size)** : Constructeur qui initialise un labyrinthe avec des dimensions impaires et génère le labyrinthe.
- **initializeMaze()** : Initialise le labyrinthe en remplissant toutes les cellules de murs (1).
- **generateMaze(int seed)** : Cette fonction génère un labyrinthe en utilisant un algorithme de type **Depth-First Search** (DFS) avec une approche de backtracking. Elle commence à partir d'une cellule impaire et creuse des chemins en alternant les directions (haut, bas, gauche, droite) et en s'assurant que les murs sont supprimés pour créer des chemins. Elle utilise un stack pour suivre les cellules visitées et effectuer le backtracking si nécessaire. La fonction vérifie également que l'entrée et la sortie du labyrinthe sont accessibles et recrée le labyrinthe si la sortie est inaccessible. □ **isPath(int x, int y) const** : Vérifie si une cellule spécifique est un chemin (représenté par 0), en vérifiant les limites.
- **draw(Texture2D wallTexture, Texture2D pathTexture) const** : Dessine le labyrinthe à l'écran, en affichant les murs et les chemins avec les textures spécifiées.
- **drawGoal(Texture2D goalTexture) const** : Dessine l'objectif (la sortie) du labyrinthe.
- **getWidth() const** et **getHeight() const** : Retourne la largeur et la hauteur du labyrinthe.

## CLASSE PLAYER

```

1 class Player {
2 private:
3     int x, y;
4     Texture2D texture;
5     int cellSize;
6
7     void adjustPosition() {
8         if (x < 0) x = 0;
9         if (y < 0) y = 0;
10    }
11
12 public:
13     Player(int startX, int startY, Texture2D tex, int size) : x(startX), y(startY), texture(tex), cellSize(size) {}
14
15     void draw() const {
16         DrawTexture(texture, x * cellSize, y * cellSize, WHITE);
17     }
18
19     void move(const Maze& maze) {
20         if (IsKeyPressed(KEY_UP) && maze.isPath(x, y - 1)) y--;
21         if (IsKeyPressed(KEY_DOWN) && maze.isPath(x, y + 1)) y++;
22         if (IsKeyPressed(KEY_LEFT) && maze.isPath(x - 1, y)) x--;
23         if (IsKeyPressed(KEY_RIGHT) && maze.isPath(x + 1, y)) x++;
24         adjustPosition();
25     }
26
27     bool hasWon(const Maze& maze) const {
28         return x == maze.getWidth() - 2 && y == maze.getHeight() - 2;
29     }
30
31     void resetPosition() {
32         x = 1;
33         y = 1;
34     }
35 };

```

### Explication du code :

- x, y : Position du joueur dans le labyrinthe.
- texture : Texture du joueur pour l'affichage.
- cellSize : Taille des cellules pour le calcul de la position du joueur sur l'écran.

**Constructeur** : Le constructeur initialise la position du joueur (x, y), la texture du joueur, et la taille des cellules du labyrinthe.

- **adjustPosition()** : Cette fonction ajuste la position du joueur pour s'assurer qu'il ne sort pas des limites du labyrinthe, en fixant x et y à zéro si nécessaire.
- **draw()** : Affiche le joueur à l'écran à la position correspondante, multipliée par la taille des cellules pour ajuster l'échelle.
- **move(const Maze& maze)** : Déplace le joueur en fonction des touches directionnelles (KEY\_UP, KEY\_DOWN, KEY\_LEFT, KEY\_RIGHT) et vérifie si le mouvement est valide (si le prochain emplacement est un chemin dans le labyrinthe via la méthode isPath() de la classe Maze). Après chaque mouvement, il ajuste la position pour qu'elle reste dans les limites du labyrinthe.
- **hasWon(const Maze& maze)** : Vérifie si le joueur a atteint la sortie du labyrinthe (position avant le bord droit et bas du labyrinthe).
- **resetPosition()** : Réinitialise la position du joueur à la position de départ (en haut à gauche du labyrinthe).

## ➤ INTERFACE GRAPHIQUE :



### • Page d'accueil :

Cette interface possède trois photos parmi lesquelles il y a la première d'un dessin d'un maze, et deux boutons PLAY AND QUIT :

## ➤ CODE DU FONCTIONNEMENT DE CES TACHES :

### WINDOWS CODE :

```
1  const int WIDTH_DEFAULT = 23;
2      const int HEIGHT_DEFAULT = 23;
3      const int CELL_SIZE_DEFAULT = 40;
4      const int RIGHT_PADDING = 200;
5      int WIDTH = WIDTH_DEFAULT;
6      int HEIGHT = HEIGHT_DEFAULT;
7      int CELL_SIZE = CELL_SIZE_DEFAULT;
8
9      InitWindow(WIDTH * CELL_SIZE + RIGHT_PADDING, HEIGHT * CELL_SIZE, "Maze Game");
```

Ce code initialise les dimensions et la fenêtre graphique d'un jeu appelé "Maze Game" en utilisant la bibliothèque Raylib.

Les constantes `WIDTH_DEFAULT` et `HEIGHT_DEFAULT` définissent la largeur et la hauteur par défaut du labyrinthe en termes de nombre de cellules, ici 23x23, tandis que `CELL_SIZE_DEFAULT` fixe la taille de chaque cellule à 40 pixels.

Une constante supplémentaire, `RIGHT_PADDING`, ajoute un espace de 200 pixels sur la droite de la fenêtre ,pour afficher des informations ou des éléments comme un chronomètre.

La fonction `InitWindow()` , fournie par Raylib, est ensuite utilisée pour créer une fenêtre graphique avec une largeur de 1120 pixels (calculée comme `WIDTH * CELL_SIZE + RIGHT_PADDING`) et une hauteur de 920 pixels (calculée comme `HEIGHT * CELL_SIZE`). Le titre de la fenêtre est défini comme "Maze Game", qui sera affiché dans la barre de titre :

## ➤ Pour Charger les images de l'interface et Button :

```
1  // Charger les images pour les interfaces
2      Texture2D interfaceImg = LoadTexture("img/interface_vrai.png");
3      // Charger les textures des boutons
4      button playButton("img/play.png", {300, 749}, 0.25f);
5      button quitButton("img/quit.png", {600, 749}, 0.25f);
```

- Ce code initialise des éléments graphiques pour une interface de jeu en chargeant des images et textures à l'aide de Raylib.
- La première ligne charge une image appelée `interface_vrai.png` située dans le dossier `img` et l'associe à une variable de type `Texture2D` nommée `interfaceImg`.
- Ensuite, deux boutons sont créés en utilisant une classe `button`.

- Le bouton "PLAY", associé à l'image play.png : **PLAY**
- est positionné aux coordonnées (300, 749) et redimensionné avec un facteur de 0.25 (25 % de sa taille d'origine) .
- De même, le bouton "QUIT" utilise l'image quit.png **QUIT** est positionné à (600, 749) et redimensionné de la même manière.
- Pour que ces boutons fonctionnent il faut lui appliquer des actions qui donnent l'accès qu'on click sur les boutons : voici le mécanisme qu'on a utilisé

```

1  while (!WindowShouldClose()) {
2
3      UpdateMusicStream(music);
4
5      if (IsKeyPressed(KEY_P)) PauseMusicStream(music);
6      if (IsKeyPressed(KEY_R)) ResumeMusicStream(music);
7
8      Vector2 mousePos = GetMousePosition();
9      bool mousePressed = IsMouseButtonPressed(MOUSE_BUTTON_LEFT);
10
11     // Si on est dans le menu principal
12     if (inMenu) {
13         // Si le bouton "Jouer" est pressé
14         playButton.updateHoverState(mousePos);
15         if (playButton.isPressed(mousePos, mousePressed)) {
16             inMenu = false; // Quitter le menu principal
17             inLevelSelection = true; // Passer à l'interface de sélection du niveau
18         }
19
20         // Si le bouton "Quitter" est pressé
21         quitButton.updateHoverState(mousePos);
22         if (quitButton.isPressed(mousePos, mousePressed)) {
23             CloseWindow(); // Fermer le jeu
24             return 0;
25         }
26
27         // Dessiner le menu principal
28         BeginDrawing();
29         ClearBackground(RAYWHITE);
30         DrawTexture(interfaceImg, 0, 0, WHITE);
31         playButton.draw();
32         quitButton.draw();
33         EndDrawing();
34     }

```

### ➤ EXPLICATION DU CODE :

- **while (!WindowShouldClose()) :**

- Cette fonction est utilisée pour démarrer la boucle principale du jeu. Elle vérifie si la fenêtre doit être fermée (en cliquant sur la croix ou en appuyant sur des touches de fermeture). Tant que cette condition est fautive, le jeu continue d'exécuter la boucle.

- **UpdateMusicStream(music) :**

- Cette fonction met à jour le flux de musique en continu. Elle est appelée à chaque itération de la boucle principale pour garantir que la musique se joue correctement pendant le jeu.

- **IsKeyPressed(KEY\_P) et IsKeyPressed(KEY\_R) :**



- Ces fonctions vérifient si les touches définies par KEY\_P (pause) et KEY\_R (reprendre) sont pressées. Si KEY\_P est pressé, la musique est mise en pause avec `PauseMusicStream(music)`. Si KEY\_R est pressé, la musique reprend avec `ResumeMusicStream(music)`.
- **GetMousePosition()** :
  - Cette fonction retourne la position actuelle de la souris sous forme de `Vector2` (une structure qui contient les coordonnées x et y de la souris).
- **IsMouseButtonPressed(MOUSE\_BUTTON\_LEFT)** :
  - Cette fonction vérifie si le bouton gauche de la souris est pressé.
- **if (inMenu)** :
  - Cette condition vérifie si le jeu est actuellement dans le menu principal. Si `inMenu` est true, les éléments suivants se produisent.

### Interaction avec les boutons :

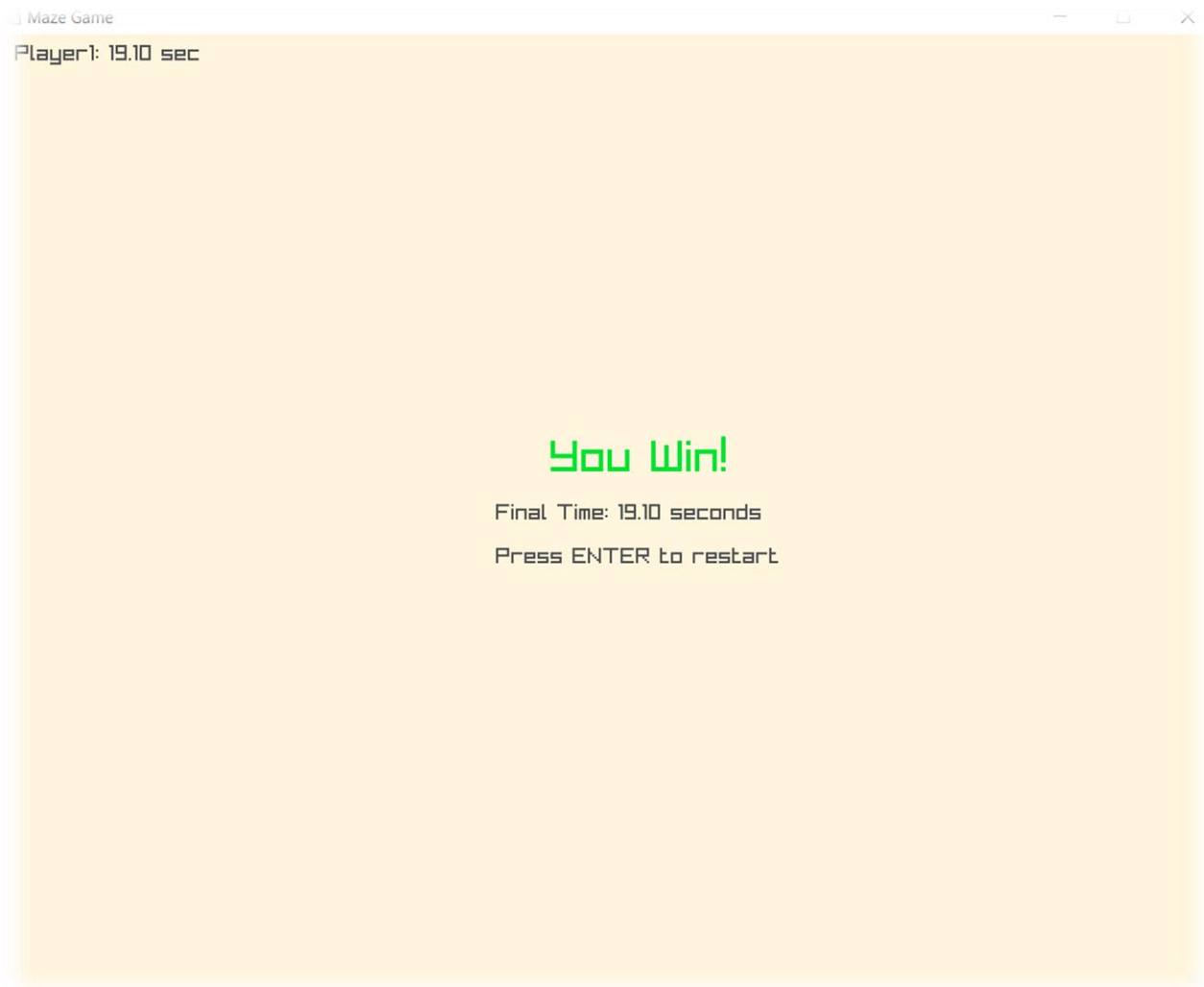
- **playButton.updateHoverState(mousePos)** : Cette fonction met à jour l'état de survol du bouton "Jouer" en fonction de la position de la souris.
- **if (playButton.ispressed(mousePos, mousePressed))** : Cette fonction vérifie si le bouton "Jouer" a été pressé en fonction de la position de la souris et de l'état du bouton gauche de la souris. Si le bouton est pressé, le jeu quitte le menu principal et passe à l'écran de sélection de niveau.
- **quitButton.updateHoverState(mousePos)** et **if (quitButton.ispressed(mousePos, mousePressed))** : Ces lignes vérifient de manière similaire si le bouton "Quitter" a été pressé. Si c'est le cas, la fonction `CloseWindow()` est appelée pour fermer la fenêtre du jeu.

### Dessiner l'interface :

- **BeginDrawing() et EndDrawing()** : Ces fonctions indiquent le début et la fin de la session de dessin sur la fenêtre.
- **ClearBackground(RAYWHITE)** : Cette fonction vide l'écran et le remplit avec la couleur blanche (utilisée ici pour réinitialiser l'arrière-plan avant de dessiner de nouveaux éléments).
- **DrawTexture(interfaceImg, 0, 0, WHITE)** : Cette fonction dessine une image (l'interface du menu) à la position (0, 0) avec la couleur blanche.
- **playButton.draw() et quitButton.draw()** : Ces fonctions dessinent les boutons "Jouer" et "Quitter" sur l'écran.



## ➤ INTERFACE DE VICTOIRE



- Voici la dernier interface de victoire qui affiche message en Vert **You Win !** avec le score du chrono.
- Aussi le message “ Press ENTER to restart ” pour rejouer la partie .

```
1  if (gameEnded) {
2      ClearBackground(FFF6DD);
3      DrawText("You Win!", GetScreenWidth() / 2 - 50, GetScreenHeight() / 2 - 60, 40, GREEN);
4      DrawText(TextFormat("Final Time: %.2f seconds", endTime - startTime), GetScreenWidth() / 2 - 100, GetScreenHeight() / 2, 20, DARKGRAY);
5      DrawText("Press ENTER to restart", GetScreenWidth() / 2 - 100, GetScreenHeight() / 2 + 40, 20, DARKGRAY);
6
7      // Si ENTER est pressé, réinitialiser le jeu
8      if (IsKeyPressed(KEY_ENTER)) {
9          gameStarted = true;
10         gameEnded = false;
11         startTime = GetTime();
12         maze = Maze(WIDTH, HEIGHT, CELL_SIZE);
13         player.resetPosition();
14     }
15 }
```

➤ Explication du code :

- Si **gameEnded** est vrai (indiquant que le jeu est terminé), plusieurs actions sont effectuées pour afficher un message de victoire.

Affichage des informations de fin de jeu :

- **ClearBackground(FFF6DD)** : Efface l'écran avec une couleur de fond beige clair.
- **DrawText("You Win!", ...)** : Affiche le message "You Win!" au centre de l'écran, en vert.
- **DrawText(TextFormat("Final Time: %.2f seconds", endTime - startTime), ...)** : Affiche le temps écoulé entre le début et la fin du jeu.
- **DrawText("Press ENTER to restart", ...)** : Affiche un message indiquant à l'utilisateur de presser ENTER pour recommencer.

Redémarrage du jeu :

- Si **ENTER** est pressé (**IsKeyPressed(KEY\_ENTER)**), les éléments suivants sont réinitialisés :
  - **gameStarted** est défini sur true, ce qui signifie que le jeu recommence.
  - **gameEnded** est défini sur false, pour signaler que le jeu est en cours.
  - **startTime** est réinitialisé avec l'heure actuelle pour redémarrer le chronomètre.
  - **maze = Maze(WIDTH, HEIGHT, CELL\_SIZE)** génère un nouveau labyrinthe avec les dimensions spécifiées.
  - **player.resetPosition()** réinitialise la position du joueur au début du labyrinthe.

# Boutons dans le deuxième interface:

## 1. Bouton Facile:

- **Rôle :** Configure le labyrinthe avec une petite taille pour un niveau facile.
- **Photo utiliser:**

**EASY**

- **Fonctionnement :**
  - Définit les dimensions du labyrinthe (WIDTH et HEIGHT) sur une valeur plus petite.
  - Génère un labyrinthe et positionne le joueur au début.
  - Lance le jeu.
- **Code associé :**

```
1  button facileButton("img/easy.png", {410, 300}, 0.33f);
2      facileButton.updateHoverState(mousePos);
3          // Pour le niveau facile
4      if (facileButton.ispressed(mousePos, mousePressed)) {
5          WIDTH = 11; HEIGHT = 11; CELL_SIZE = 40;
6          maze = Maze(WIDTH, HEIGHT, CELL_SIZE);
7          player = Player(1, 1, playerTexture, CELL_SIZE);
8          gameStarted = true;
9          startTime = GetTime();
10         inLevelSelection = false;
11     }
12     facileButton.draw();
```

- **Explication du code :**

Le code du bouton **Facile** permet d'interagir avec l'utilisateur en utilisant plusieurs fonctions de Raylib. Il commence par détecter la position actuelle de la souris à l'aide de la fonction **GetMousePosition()**, essentielle pour suivre les mouvements de l'utilisateur. Ensuite, il utilise **IsMouseButtonPressed(MOUSE\_BUTTON\_LEFT)** pour vérifier si le bouton gauche de la souris a été cliqué. Pour déterminer si la souris survole le bouton, la fonction **updateHoverState** est appelée, ce qui peut déclencher un changement d'apparence du bouton pour indiquer son état survolé. Si le bouton est cliqué, la fonction **ispressed** est activée pour confirmer que l'utilisateur a appuyé sur ce bouton. Lorsque le clic est confirmé, l'état du jeu est modifié pour passer au mode de difficulté "Facile", et le chronomètre est initialisé avec **GetTime()** pour enregistrer le temps écoulé. Ce fonctionnement rend l'interface utilisateur réactive et assure une transition intuitive vers le mode de jeu sélectionné.

## 2. Bouton Moyen:

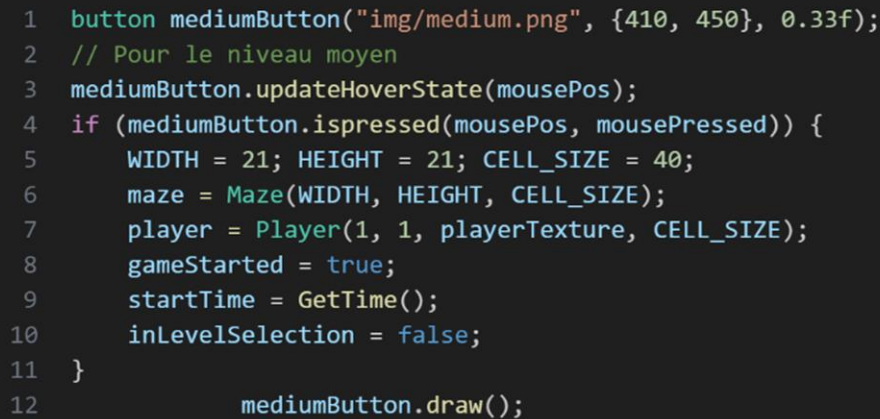
- **Rôle :** Configure le labyrinthe avec une taille moyenne pour un niveau intermédiaire.
- **Photo utiliser:**

**MEDIUM**

- **Fonctionnement :**

- Définit des dimensions plus grandes que pour le mode facile.
- Génère un labyrinthe de taille moyenne et positionne le joueur.

- **Code associé :**



```
1  button mediumButton("img/medium.png", {410, 450}, 0.33f);
2  // Pour le niveau moyen
3  mediumButton.updateHoverState(mousePos);
4  if (mediumButton.ispressed(mousePos, mousePressed)) {
5      WIDTH = 21; HEIGHT = 21; CELL_SIZE = 40;
6      maze = Maze(WIDTH, HEIGHT, CELL_SIZE);
7      player = Player(1, 1, playerTexture, CELL_SIZE);
8      gameStarted = true;
9      startTime = GetTime();
10     inLevelSelection = false;
11 }
12     mediumButton.draw();
```

- **Explication du code :**

Le code du bouton **Moyen** utilise plusieurs fonctions pour détecter les interactions de l'utilisateur et déclencher des actions spécifiques lorsqu'il est cliqué. La fonction principale commence par récupérer la position de la souris grâce à **GetMousePosition()** et détecte si le bouton gauche de la souris a été pressé via **IsMouseButtonPressed(MOUSE\_BUTTON\_LEFT)**. Ensuite, elle utilise la méthode **updateHoverState** pour déterminer si la souris est actuellement au-dessus du bouton, ce qui peut provoquer un changement visuel pour indiquer un survol. Si l'utilisateur clique sur le bouton, la méthode **ispressed** est appelée pour confirmer que le bouton a été activé. Enfin, lorsque le bouton "Moyen" est cliqué, il initialise le chronomètre avec la fonction **GetTime()** pour suivre le temps écoulé et change l'état du jeu pour passer à une nouvelle scène ou mode de difficulté. Ces interactions permettent une transition fluide et intuitive pour le joueur, rendant le jeu interactif et réactif.

### 3. Bouton Difficile:

- **Rôle :** Configure le labyrinthe avec une grande taille pour un niveau difficile.
- **Photo utiliser:**



**HARD**

- **Fonctionnement :**

- Définit les dimensions maximales pour un défi plus important.
- Génère un grand labyrinthe et place le joueur au départ.

- **Code associé :**



```
1  button hardButton("img/hard.png", {410, 600}, 0.33f);
2      hardButton.updateHoverState(mousePos);
3      if (hardButton.ispressed(mousePos, mousePressed)) {
4          WIDTH = 23; HEIGHT = 23; CELL_SIZE = 40; // Grand labyrinthe pour Difficile
5          maze = Maze(WIDTH, HEIGHT, CELL_SIZE);
6          player.resetPosition();
7          gameStarted = true;
8          startTime = 0.0;
9          startTime = GetTime();
10         inLevelSelection = false;
11     }
12         hardButton.draw();
```

- **Explication du code :**

Le code du bouton **Difficulté** permet de gérer les interactions utilisateur pour sélectionner le mode de difficulté du jeu. Il commence par capturer la position de la souris avec la fonction **GetMousePosition()**, ce qui permet de suivre les mouvements du curseur en temps réel. Ensuite, il utilise la fonction

**IsMouseButtonPressed(MOUSE\_BUTTON\_LEFT)** pour détecter si l'utilisateur a cliqué avec le bouton gauche de la souris. Pour savoir si la souris est positionnée au-dessus du bouton, la fonction **updateHoverState** est utilisée, ce qui permet de modifier l'apparence du bouton lorsqu'il est survolé pour indiquer visuellement son état. Si un clic est détecté sur ce bouton, la fonction **ispressed** est déclenchée pour valider l'interaction. Une fois le clic confirmé, l'état du jeu est mis à jour pour activer le mode "Difficulté". Simultanément, le chronomètre est initialisé avec **GetTime()**, permettant de mesurer le temps de jeu. Ce code garantit une transition fluide et interactive entre les différents niveaux de difficulté en réponse aux actions de l'utilisateur.

# Labyrinth interface:

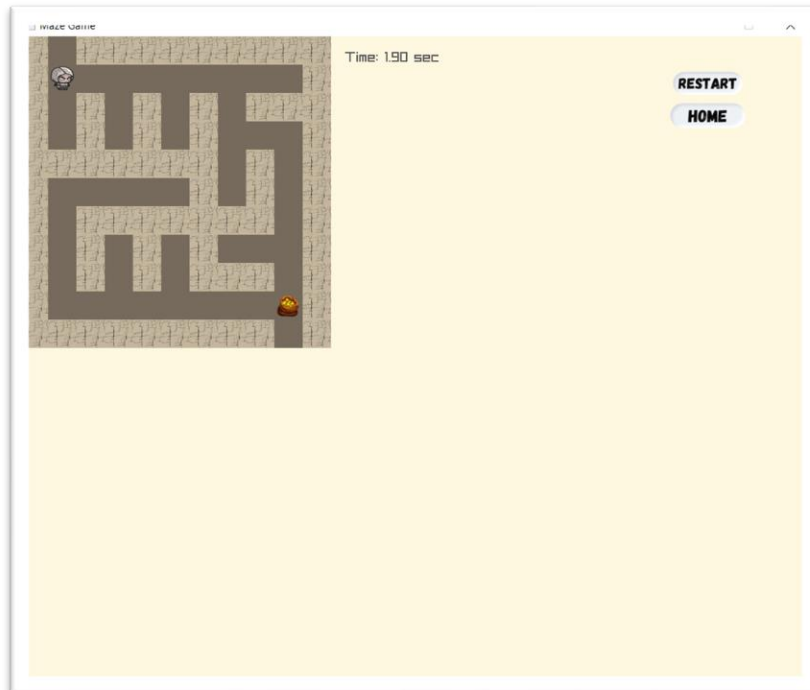
## 1. Niveau facile:

- **Génération du labyrinthe facile :**

Le constructeur de la classe Maze génère un labyrinthe en utilisant l'algorithme de parcours en profondeur en partant de la case (1, 1).

Cela définit un labyrinthe de 11x11 cases, chaque case ayant une taille de 40 pixels.

Le joueur est initialisé à la position (1, 1).



- **Affichage des murs et des chemins :**

La méthode **draw()** parcourt chaque cellule de la matrice maze :

- Si la valeur est 1, la texture de mur est affichée.
- Si la valeur est 0, la texture de chemin est affichée.

- **Affichage de l'objectif (Goal) :**

La méthode **drawGoal()** place la texture de l'objectif dans la case de sortie (**width - 2, height - 2**).

- **Affichage des Boutons Restart et Home:**

### Bouton Restart :

- La texture est chargée depuis "img/restart.png".
- La position est définie en haut à droite.
- La taille est réduite de 15%.



```
1 button restartButton("img/restart.png", {WIDTH * CELL_SIZE - 15, HEIGHT * CELL_SIZE / 20}, 0.15);  
2
```

## Bouton Home :

- Positionné juste en dessous du bouton Restart.



```
1 button homeButton("img/home.png", {WIDTH * CELL_SIZE - 15, HEIGHT * CELL_SIZE / 10}, 0.15);  
2
```

- **Affichage des boutons :**



```
1 restartButton.draw();  
2     homeButton.draw();
```

- **Interactions avec les boutons :**

## Bouton Restart :



```
1 if (restartButton.ispressed(mousePos, mousePressed)) {  
2     maze = Maze(WIDTH, HEIGHT, CELL_SIZE);  
3     player.resetPosition();  
4     startTime = GetTime();  
5  
6     gameEnded = false;  
7     gameStarted = true;  
8 }
```

- Lorsque le bouton est pressé, un nouveau labyrinthe est généré.
- La position du joueur est réinitialisée.
- Le chronomètre redémarre.



## Bouton Home :

```
1 if (homeButton.isPressed(mousePos, mousePressed)) {
2     gameStarted = false;
3     gameEnded = false;
4     inMenu = true;
5     showMainMenu = true;
6 }
```

- Ramène le joueur au menu principal.

## Chronomètre:

Le temps de début est enregistré lorsque le joueur choisit un niveau ou redémarre le jeu.

- Affichage du temps écoulé :
  - Le temps écoulé est calculé en soustrayant **startTime** au temps actuel (**GetTime()**).
- Enregistrement du score:

```
1 DrawText(TextFormat("Final Time: %.2f seconds", endTime - startTime), GetScreenWidth() / 2 - 100, GetScreenHeight() / 2, 20, DARKGRAY);
2
```

- Une fois le jeu terminé, le temps final est sauvegardé avec le nom du joueur.

```
1 saveScore(playerName, endTime - startTime, scores);
2
```

## 2. Niveau Moyen:

- **Génération du labyrinthe Moyen:**

La taille du labyrinthe est réglée sur 21x21, et Chaque cellule du labyrinthe mesure 40 pixels, ce qui détermine la taille visuelle des cases.

Le joueur est initialisé à la position (1, 1).

- **Affichage du labyrinthe:**

Dans la boucle de rendu principale, **le labyrinthe de niveau moyen est dessiné à l'écran à l'aide des textures.**

**La méthode maze.draw() effectue cette tâche.**

**maze.draw(...)** : Affiche les murs et les chemins du labyrinthe en parcourant chaque cellule et en utilisant les textures appropriées.

**maze.drawGoal(...)** : Dessine l'objectif (par exemple, le fromage ou un autre marqueur) dans la dernière cellule du labyrinthe.



### 3. Niveau Difficile:

- **Génération du labyrinthe Difficile:**

La taille du labyrinthe est réglée sur 31x31, et Chaque cellule du labyrinthe mesure 30 pixels, ce qui détermine la taille visuelle des cases.

Le joueur est initialisé à la position (1, 1).

- **Affichage du labyrinthe:**

Dans la boucle de rendu principale, le labyrinthe de niveau Difficile est dessiné à l'écran à l'aide des textures. La méthode `maze.draw()` effectue cette tâche.

**`maze.draw(...)`** : Affiche les murs et les chemins du labyrinthe en parcourant chaque cellule et en utilisant les textures appropriées.

**`maze.drawGoal(...)`** : Dessine l'objectif (par exemple, le fromage ou un autre marqueur) dans la dernière cellule du labyrinthe



#### CONCLUSION :

Ce projet de jeu de labyrinthe développé en C++ avec Raylib représente une réalisation technique remarquable qui illustre parfaitement la synergie entre programmation orientée objet et développement de jeux vidéo. L'équipe a réussi à créer une expérience de jeu complète et engageante tout en démontrant une maîtrise approfondie des concepts techniques fondamentaux. Ce jeu représente une excellente synthèse entre théorie et pratique, entre programmation et expérience utilisateur. Il démontre la capacité de l'équipe à mener à bien un projet de développement complet, de sa conception à sa réalisation finale. Le résultat est un produit non seulement fonctionnel et bien structuré sur le plan technique, mais aussi divertissant et engageant pour les utilisateurs finaux. Cette réalisation servira sans doute d'exemple inspirant pour les futurs projets de développement de jeux en milieu académique.