

**FACULTE DES SCIENCES ET THECNIQUES DE TANGER**

**DEPARTEMENT DE INFORMATIQUE**

**Rapport** :

Programmation orientee objet en C++

# PFM : POO C++/Raylib

****

**Encadre par** :

Ikram Ben abdel ouahab

**Realise par**:

* Rabih Senhaji Anas
* El Kajdouhi Mohamed Ayman
* Akdi Fouad
* Afazaz Ilyas

# Introduction :

# Ce rapport présente une analyse détaillée du code source d'un jeu de labyrinthe écrit en C++ utilisant la bibliothèque Raylib. Le jeu propose des fonctionnalités variées, notamment un mode solo ou multijoueur, une sélection de difficulté, et une gestion des scores. Le code est structuré autour de plusieurs classes pour une meilleure organisation et modularité.

# Principe de PFM :

Ce projet consiste en un jeu de labyrinthe développé en C++ avec Raylib, où le joueur navigue dans un labyrinthe généré dynamiquement pour atteindre un objectif, avec des niveaux de difficulté variés et un mode multijoueur.

**Structure globale :**

Le programme est composé des éléments suivants :

- **Classes principales** : Maze, Player, Game, et Level.

- **Enumération** : GameState pour gérer les états du jeu (menu, jeu, sélection de difficulté, etc.).

- **Bibliothèques utilisées** : Raylib pour l’interface graphique et audio.

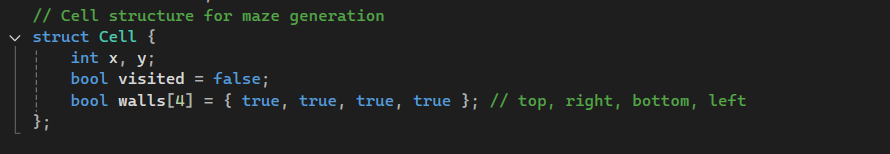
- **Boucle principale** : Contrôle l'état du jeu et appelle les méthodes associées.

1. **Classe Maze (Génération du labyrinthe) :**

**-Rôle de la classe :**

La classe Maze gère la génération et le rendu du labyrinthe. Elle utilise l'algorithme DFS (Depth-First Search) pour créer des chemins complexes. Les cellules du labyrinthe sont modélisées par la structure Cell.

**-Structure Cell :**



Chaque cellule a :

- Ses coordonnées (x, y).

- Un état visited pour suivre si elle a été visitée.

- Des murs représentés par un tableau de booléens.

### **-Attributs  :**

* **grid** : Un vecteur qui stocke toutes les cellules du labyrinthe.
* **wallColor** : Permet de modifier dynamiquement la couleur des murs du labyrinthe.

**-Algorithme DFS :**

L'algorithme DFS est utilisé pour générer un labyrinthe parfait (sans boucles). Il fonctionne en explorant récursivement les cellules voisines non visées.



- **Initialisation** : Chaque cellule est marquée comme non visitée avec des murs intactes.

- **Pile** : Une pile (à l'aide de std::stack) stocke les cellules pour retourner en arrière si nécessaire.

- **Parcours DFS** :

1. Visite une cellule.

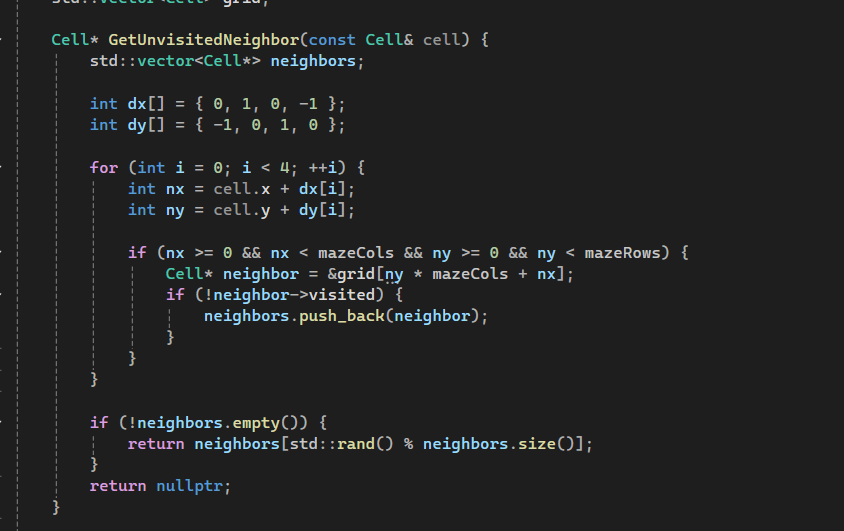
2. Recherche une voisine non visitée (à l'aide de GetUnvisitedNeighbor).

3. Supprime les murs entre la cellule courante et la suivante (à l'aide de RemoveWalls).

4. Retourne en arrière si aucune voisine n'est disponible.

**-Méthode GetUnvisitedNeighbor :**

Cette méthode retourne une cellule voisine non visitée de la cellule donnée. Elle est essentielle pour l'algorithme DFS, car elle détermine les prochaines cellules à visiter.

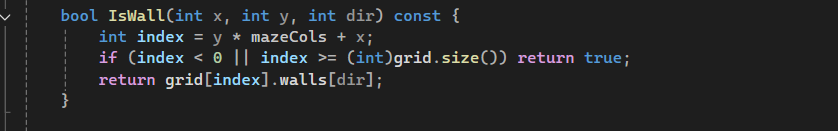


**Détails :**

* Parcourt les quatre directions cardinales (haut, droite, bas, gauche).
* Vérifie si une cellule voisine existe dans les limites du labyrinthe et n'a pas été visitée.
* Retourne une cellule voisine aléatoire

**-Méthode IsWall :**

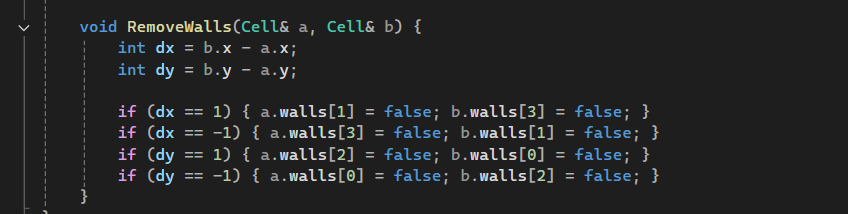
Permet de vérifier si un mur existe dans une direction donnée à partir des coordonnées d'une cellule.



**Détails :**

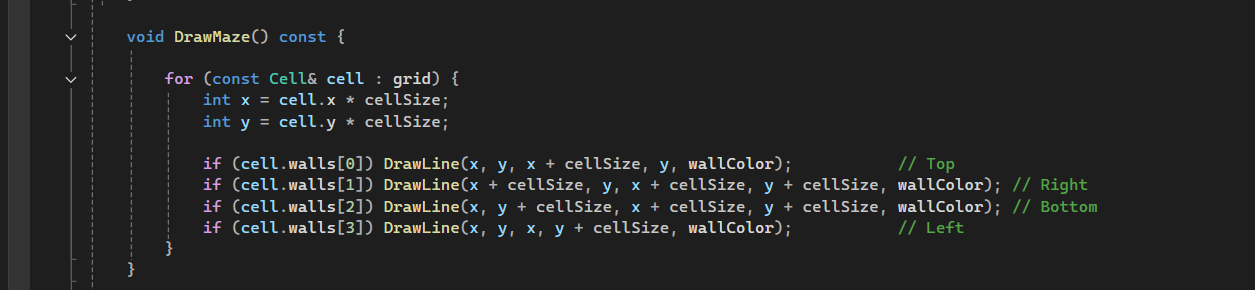
* Prend les coordonnées (x, y) et une direction (dir) comme arguments.
* Retourne true si un mur bloque la direction spécifiée

**-Suppression des murs :**



Cette méthode met à jour les murs entre deux cellules voisines selon leur direction relative.

**-Dessin du labyrinthe :**



Chaque mur est dessiné avec DrawLine si son état est true.

**2) Classe Player (Gestion du joueur):**

**-Rôle de la classe :**

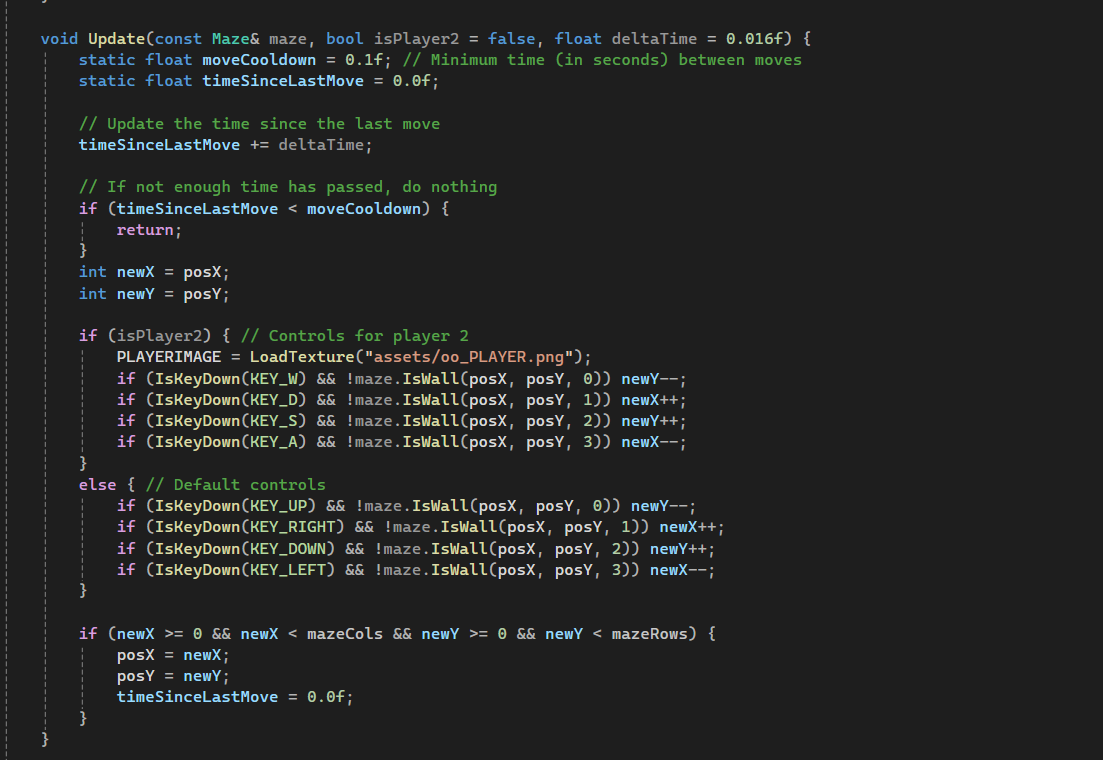
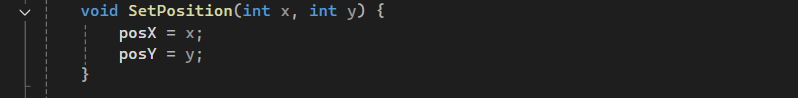
La classe Player représente le joueur dans le labyrinthe. Elle gère sa position, ses mouvements, et l'affichage.

### **-Attributs supplémentaires :**

* **PLAYERIMAGE** : Texture utilisée pour représenter visuellement le joueur dans le labyrinthe.
* **goalTexture** : Texture utilisée pour représenter l’objectif final.
* **posX, posY** : Coordonnées de la position actuelle du joueur.

**-Code principal :**

**-Positionnement et mise à jour :**



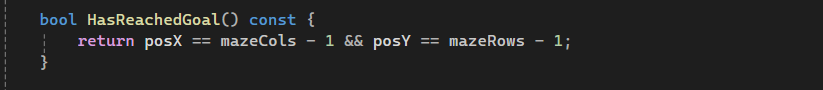
Gère les mouvements du joueur en fonction des touches pressées et vérifie si le chemin est libre.

**Détails :**

* Implémente un délai minimum entre les déplacements pour éviter des mouvements trop rapides.
* Différencie les commandes selon qu'il s'agit du joueur 1 ou 2 (mode multijoueur).
* Le joueur peut se déplacer si le chemin n’est pas bloqué par un mur.

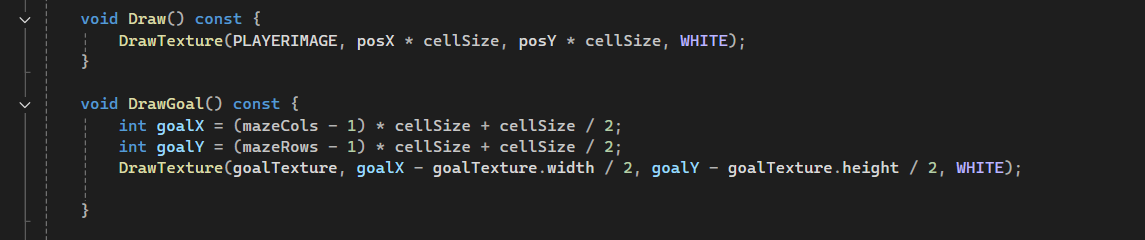
#### **-Méthode HasReachedGoal :**

Permet de déterminer si le joueur a atteint l'objectif du labyrinthe.



Vérifie si les coordonnées du joueur correspondent à celles de l'objectif (coin inférieur droit du labyrinthe)

**-Affichage :**



Le joueur et l'objectif sont affichés à l'aide de textures préchargées.

**3) Classe game(Logique principale du jeu) :**

**-Rôle de la classe :**

La classe Game gère la logique principale du jeu. Elle contrôle l'état du jeu, suit le temps, et orchestre les interactions entre les joueurs et le labyrinthe.

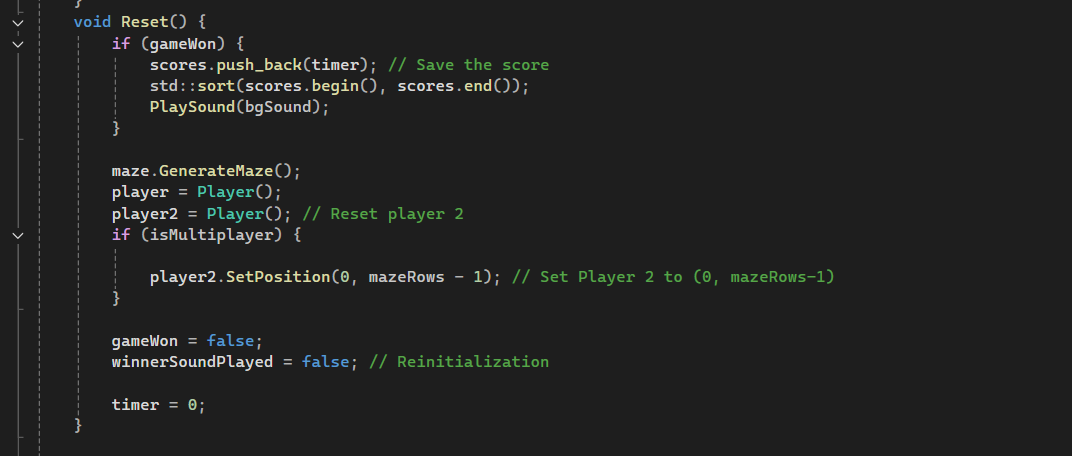
### **-Attributs supplémentaires :**

* **maze** : Instance de la classe Maze pour gérer le labyrinthe actuel.
* **player, player2** : Instances de la classe Player représentant les joueurs.
* **scores** : Liste des meilleurs temps enregistrés.
* **gameWon** : Indique si le joueur a gagné.
* **timer** : Chronomètre pour mesurer le temps de jeu.
* **showScores** : Indicateur pour afficher ou masquer le tableau des scores.

**-Code principal :**

**-Réinitialisation et gestion des niveaux :**

Réinitialise l'état du jeu en recréant un nouveau labyrinthe, en repositionnant les joueurs, et en réinitialisant les variables de score.

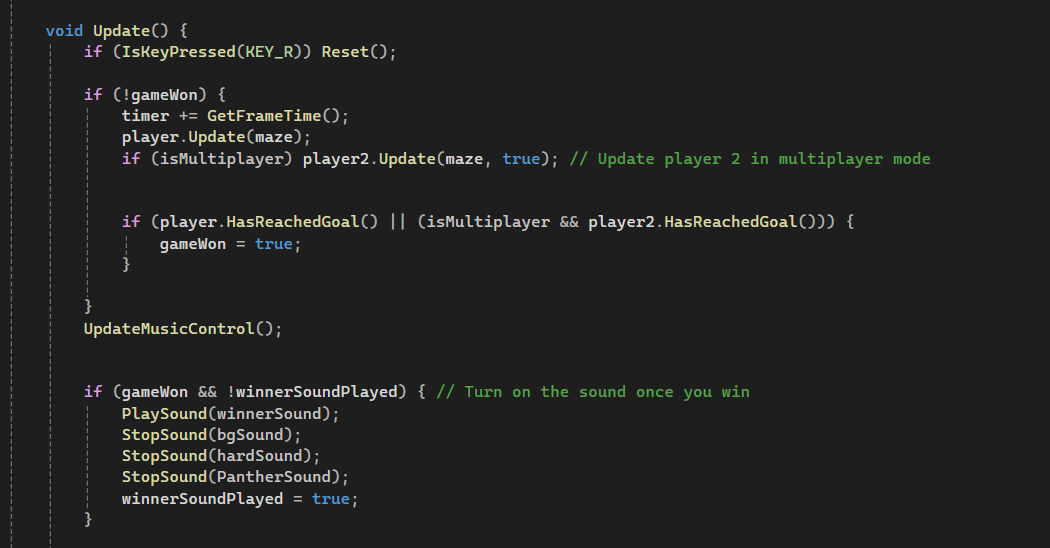


**Détails :**

* Ajoute le temps actuel au tableau des scores si le joueur a gagné.
* Regénère le labyrinthe en appelant Maze::GenerateMaze.

**-Mise à jour du jeu :**

Met à jour les événements du jeu, tels que les mouvements des joueurs, la gestion du chronomètre, et la vérification de la condition de victoire.

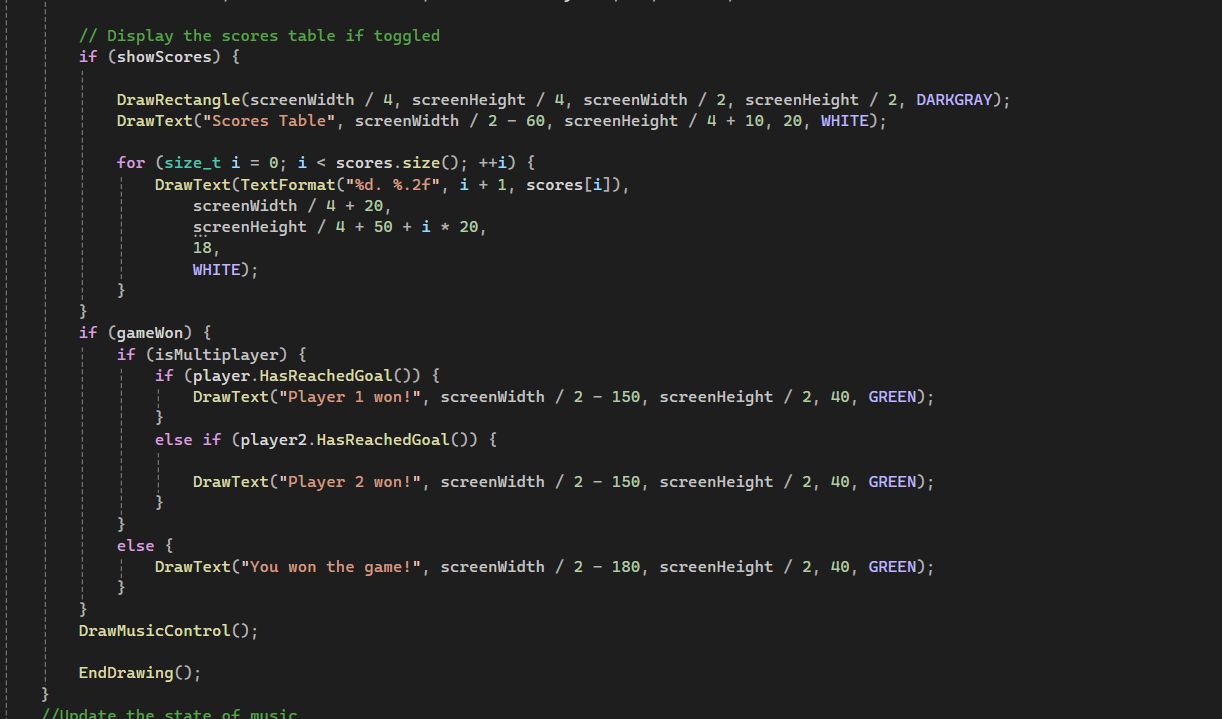
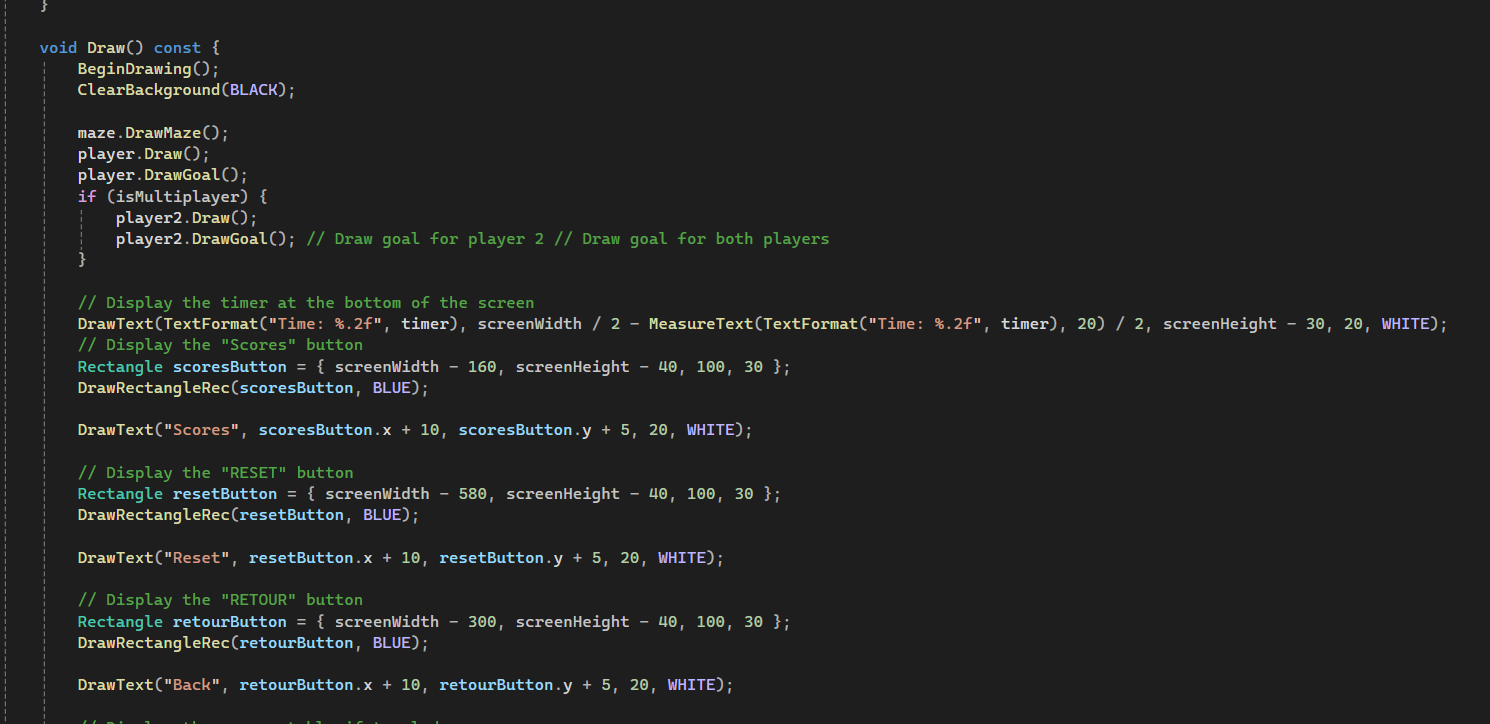


**Détails :**

* Gère les interactions des joueurs avec le labyrinthe.
* Contrôle les boutons interactifs, tels que "Reset" et "Scores"

**-Affichage du jeu :**

Dessine les éléments visuels du jeu, y compris le labyrinthe, les joueurs, les scores, et les boutons.



**Détails :**

* Utilise Maze::DrawMaze pour dessiner le labyrinthe.
* Affiche les joueurs à l'aide de Player::Draw.
* Affiche un message de victoire lorsque la partie est terminée.

### **-Gestion des scores :**

Les scores sont enregistrés dans un vecteur trié. L'interface utilisateur permet de basculer l'affichage des scores avec un bouton.

**4) Classe Level :**

**-Rôle de la classe :**

La classe Level définit les paramètres spécifiques à chaque niveau de difficulté, tels que les dimensions du labyrinthe, la taille des cellules, et la couleur des murs.

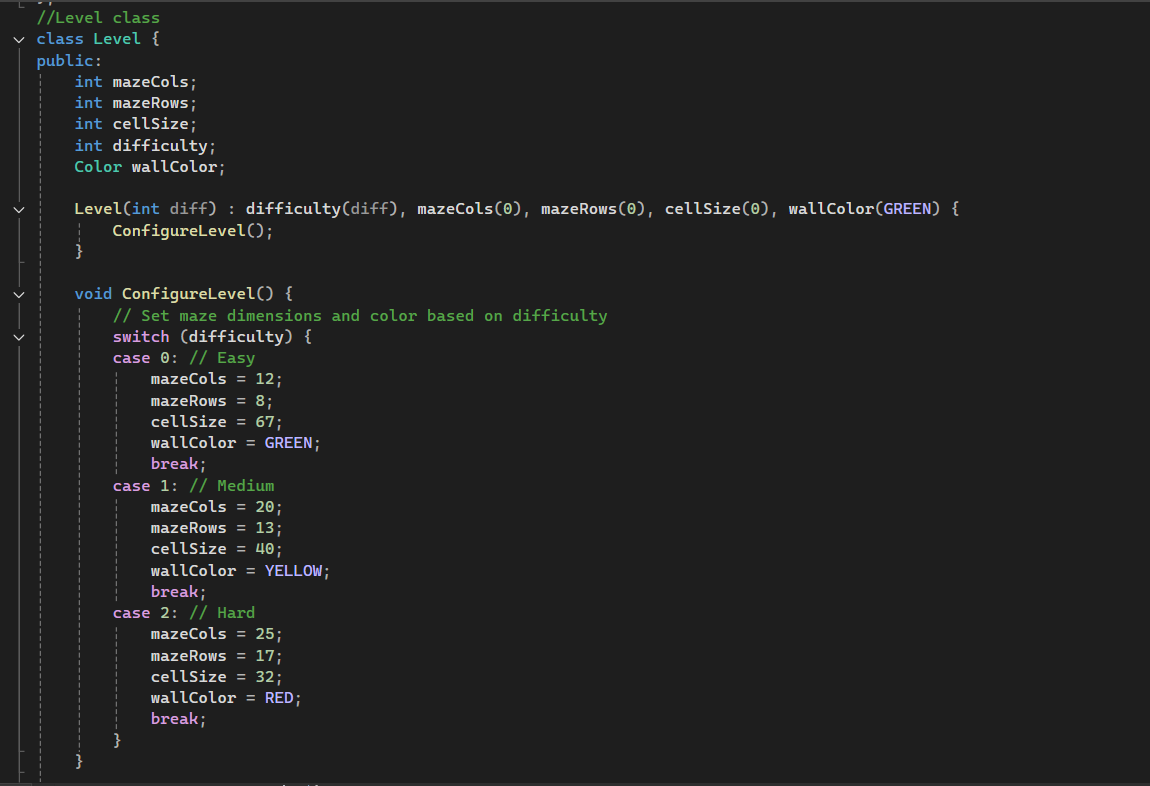
### **-Attributs :**

1. **mazeCols** : Nombre de colonnes dans le labyrinthe pour ce niveau.
2. **mazeRows** : Nombre de lignes dans le labyrinthe pour ce niveau.
3. **cellSize** : Taille des cellules du labyrinthe (en pixels).
4. **difficulty** : Niveau de difficulté sélectionné (0 pour facile, 1 pour moyen, 2 pour difficile).
5. **wallColor** : Couleur des murs du labyrinthe, qui change selon la difficulté.

### **-Configuration des niveaux :**

Les niveaux sont divisés en trois catégories : facile, moyen, et difficile. Chaque niveau ajuste dynamiquement les propriétés du labyrinthe.

Cette méthode configure les paramètres du niveau en fonction de la difficulté choisie :

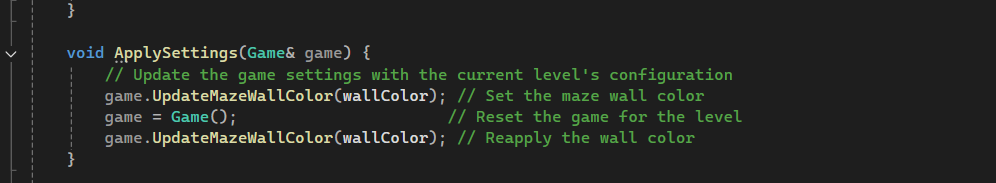


**Détails :**

* Détermine les dimensions et la taille des cellules du labyrinthe.
* Définit la couleur des murs :
  + **Vert** pour facile.
  + **Jaune** pour moyen.
  + **Rouge** pour difficile.

**-Application des paramètres :**

Les paramètres du niveau sont appliqués à l’aide de la méthode ApplySettings, qui met à jour les attributs de la classe Game, notamment la couleur des murs et la configuration du labyrinthe.

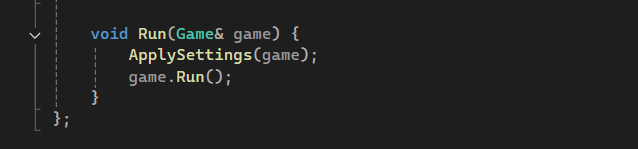


**Détails :**

* Met à jour la couleur des murs du labyrinthe dans la classe Maze via Game::UpdateMazeWallColor.
* Réinitialise la partie en recréant l'objet Game avec les nouveaux paramètres

#### **-Méthode Run :**

Exécute le jeu avec les paramètres configurés.



**Détails :**

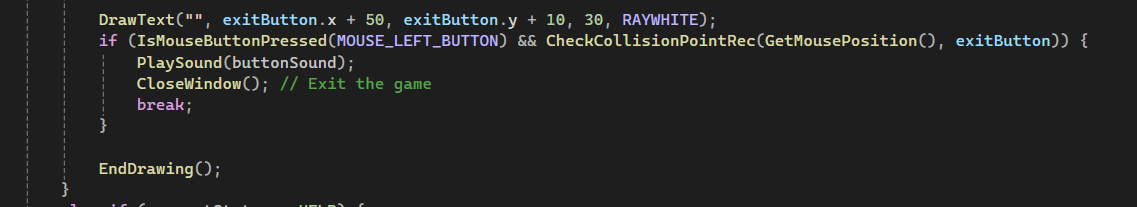
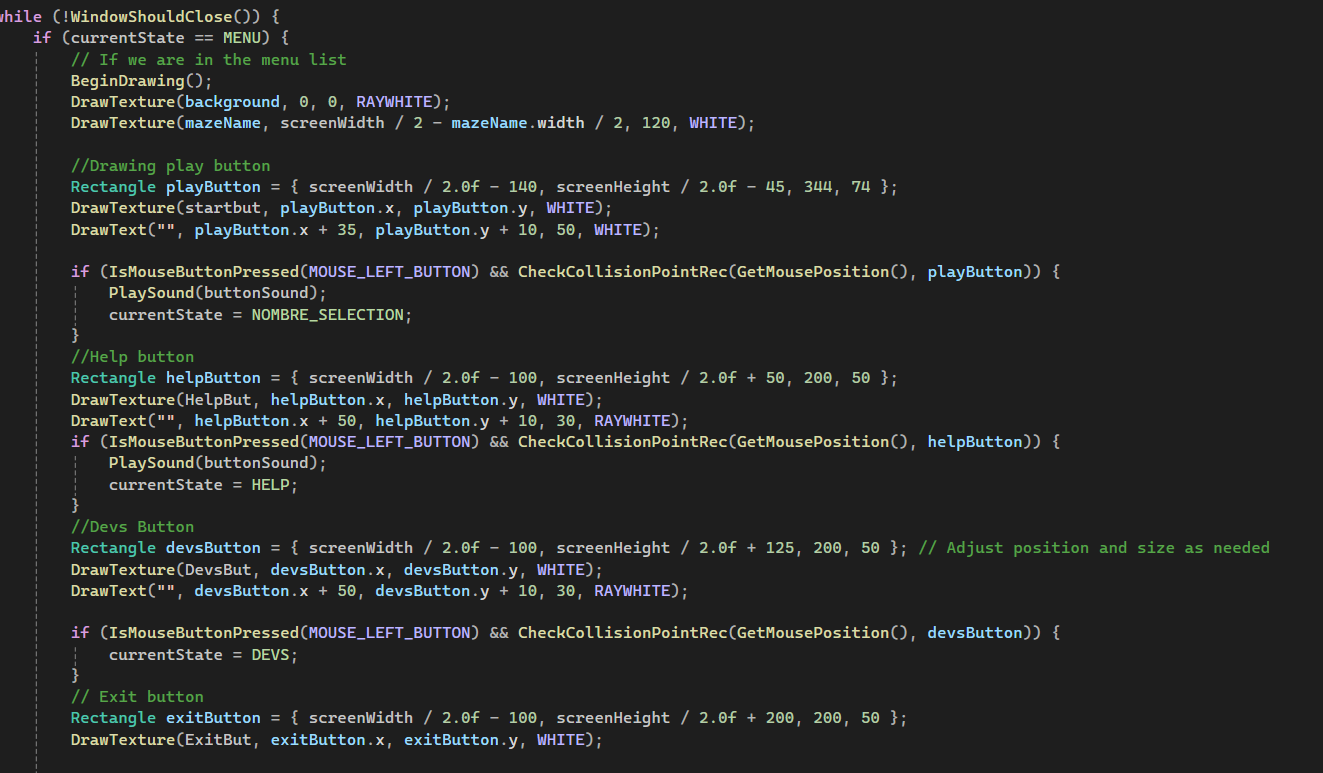
* Appelle ApplySettings pour configurer le jeu.
* Lance la boucle principale du jeu avec Game::Run

**-Interface utilisateur (UI) :**

### **-Gestion des états du jeu :**

Le programme utilise l’énumération GameState pour naviguer entre plusieurs états, comme le menu principal, la sélection de difficulté, l’aide, et l’écran des développeurs. Chaque état a une interface distincte.

**-Exemple : Menu principal :**



### **-Boutons interactifs :**

Chaque écran contient des boutons interactifs, tels que les boutons "Scores", "Reset", et "Back". Les clics sur ces boutons déclenchent des actions spécifiques, comme le retour au menu principal ou la réinitialisation du jeu.

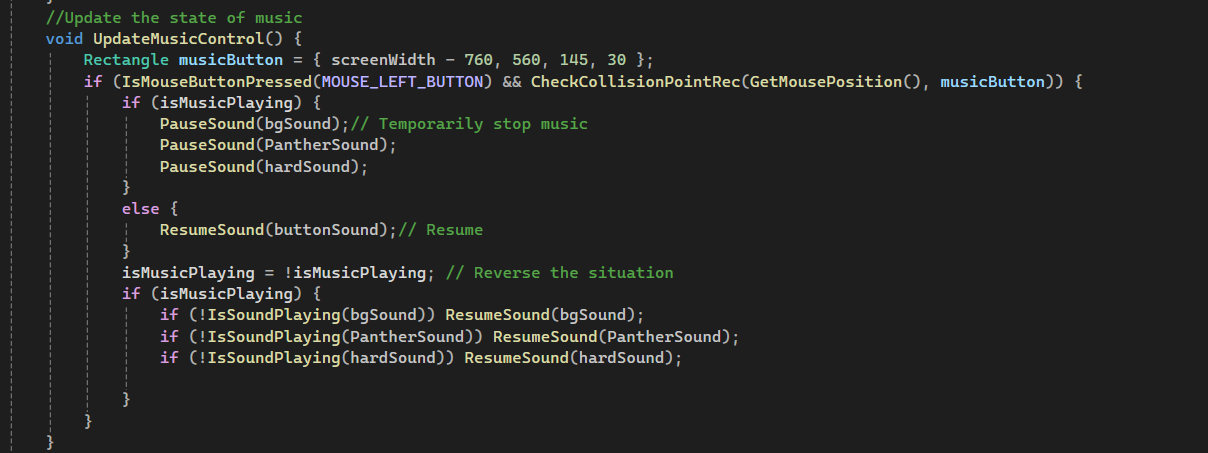
## Fonctionnalités audio et graphiques :

### **-Effets sonores :**

Le programme utilise plusieurs sons pour enrichir l’expérience utilisateur :

* **buttonSound** : Son joué lors des clics sur les boutons.
* **bgSound** : Musique de fond.
* **winnerSound** : Son joué lors de la victoire.

**-Exemple d’implémentation :**



### **-Textures et images :**

Les textures sont utilisées pour les joueurs, le but, et les arrière-plans. Elles sont chargées au début de l’exécution et libérées à la fin.

## -Mode multijoueur :

Le mode multijoueur permet à deux joueurs de naviguer dans le labyrinthe en simultané. Les commandes sont divisées comme suit :

* Joueur 1 : Touches fléchées.
* Joueur 2 : Touches **W/A/S/D**.

## Conclusion :

Ce projet de jeu de labyrinthe illustre l’utilisation efficace de C++ avec la bibliothèque Raylib pour créer une application interactive et visuellement attrayante. Les fonctionnalités telles que les niveaux, l’interface utilisateur intuitive, les effets sonores immersifs, et le mode multijoueur rendent ce projet complet et adapté à un large public.