**Rapport détaillé sur le code du jeu Labyrinthe**

Ce document explique en détail chaque élément du code, en couvrant les classes, les attributs, les méthodes, et les interactions entre les composants du programme. Le jeu est développé en **C++** en utilisant la bibliothèque **Raylib** pour la gestion graphique.

**Structure Générale du Programme**

Le code est organisé autour de **quatre classes principales** :

1. **Labyrinthe** : Génère et gère le labyrinthe.
2. **Joueur** : Représente le joueur et ses déplacements.
3. **Niveau** : Définit la difficulté du jeu en fonction du choix de l'utilisateur.
4. **Jeu** : Gère les états du jeu, le menu, le chronomètre et la logique de victoire.

Chaque classe a des responsabilités bien définies et interagit avec les autres pour créer une expérience de jeu cohérente.

**Classe Labyrinthe : Génération et Gestion d'un Labyrinthe Dynamique**

**1.1. Structure interne : Cell**

La classe Labyrinthe repose sur une structure interne appelée **Cell**, qui représente l'unité fondamentale de construction du labyrinthe.

**Attributs :**

* **Coordonnées spatiales** : Chaque cellule possède des attributs **x** et **y** permettant de la localiser précisément dans la grille.
* **État de visite** : Un booléen **visited** pour suivre l'exploration des cellules lors de la génération du labyrinthe.
* **Gestion des murs** : Quatre booléens (**topWall, rightWall, bottomWall, leftWall**) représentant l'état des murs.
  + true indique la présence d'un mur.
  + false signifie un passage ouvert.

**Rôle** :  
La structure **Cell** est un élément fondamental. Chaque cellule a des murs initiaux (tous activés). Pendant la génération du labyrinthe, certains murs sont supprimés pour créer des chemins.

**1.2. Attributs de la classe Labyrinthe**

* **rows, cols** : Nombre de lignes et colonnes du labyrinthe. Définit la taille totale.
* **cellSize** : Taille d’une cellule en pixels, calculée dynamiquement en fonction de la largeur de l'écran.
* **grille** : Une matrice 2D (type vector<vector<Cell>>) contenant toutes les cellules.

**1.3. Constructeur et Initialisation**

**Labyrinthe(int r, int c)**

* Rôle : Constructeur qui initialise le labyrinthe avec ses dimensions.
* **Paramètres d'entrée** : Prend le nombre de lignes et de colonnes comme arguments.
* **Calcul de la taille des cellules** : Déterminée dynamiquement en fonction de la largeur de l'écran.
* **Création de la grille** : Initialise une grille 2D de cellules avec leurs coordonnées respectives.

**Méthode GenererLabyrinthe()**

La méthode de génération utilise l'algorithme **backtracking** (parcours en profondeur, **DFS**).

**Étapes détaillées :**

1. **Initialisation** :
   * La cellule en haut à gauche est marquée comme visitée.
   * Une pile (**stack**) suit les cellules à explorer.
2. **Exploration** :
   * Tant que la pile n'est pas vide :
     + Récupère la cellule courante.
     + Identifie ses voisins non visités.
     + Si des voisins existent :
       - En choisit un au hasard.
       - Supprime les murs entre la cellule courante et le voisin.
       - Ajoute le voisin à la pile et le marque comme visité.
     + Sinon, effectue un retour en arrière (retire la cellule de la pile).

**Avantages** :

* Garantie d’un labyrinthe totalement connexe.
* Génération unique et aléatoire à chaque exécution.

**Méthode DessinerLabyrinthe()**

* **Parcours de la grille** : Analyse chaque cellule.
* **Dessin des murs** : Utilise **Raylib** pour dessiner des lignes blanches avec **DrawLine()**.
* **Représentation visuelle** : Respect exact de la topologie générée.

**Méthode VerifierSolution()**

* Rôle : Vérifie si le joueur a atteint la cellule de sortie.
* **Paramètres d'entrée** : Position actuelle du joueur (**playerX**, **playerY**).
* **Condition de victoire** : Coin inférieur droit du labyrinthe.
* **Retour** : true si la condition est remplie.

**Classe Joueur : Gestion des Déplacements et de l’Interaction avec le Labyrinthe**

**Attributs principaux :**

* **x, y** : Coordonnées actuelles du joueur dans la grille.
* **couleur** : Couleur graphique du joueur (par défaut : bleu).
* **taille** : Rayon du cercle représentant le joueur, proportionnel à la taille des cellules.

**Constructeur et Initialisation**

* **Joueur()** : Initialise la position (0,0) et la couleur par défaut (bleue).
* **Initialiser(int cellSize)** : Définit la taille dynamique du joueur.

**Méthode Afficher()**

* **Calcul des coordonnées graphiques** : Conversion des indices (x, y) en pixels.
* **Affichage** : Utilise **DrawCircle()** pour représenter un cercle coloré.

**Méthode Deplacer()**

* **Entrées utilisateur** : Capture les touches directionnelles.
* **Validation** :
  + Vérifie les limites de la grille.
  + Contrôle l’absence de murs dans la direction souhaitée.
* **Mise à jour** : Modifie la position (x, y) si les conditions sont remplies.

**Classe Niveau : Sélection de la Difficulté**

**Attributs :**

* **rows, cols** : Dimensions ajustées selon le niveau sélectionné.

**Méthode ChoisirNiveau(int niveau)**

* **Niveau 1** : 10x10.
* **Niveau 2** : 15x15.
* **Niveau 3** : 20x20.
* **Par défaut** : Niveau moyen si l’entrée est invalide.

**Classe Jeu : Gestion Globale**

**Attributs :**

* **startTime, elapsedTime** : Suivi du chronomètre.
* **EtatJeu** : MENU, EN\_COURS, TERMINE.
* **tempsFinal** : Temps total après victoire.

**Méthodes**

* **DemarrerChrono()** : Démarre le chronomètre.
* **AfficherMenu()** : Affiche le menu principal.
* **AfficherChrono()** : Met à jour et affiche le temps.
* **AfficherVictoire()** : Affiche un message de victoire.

**Interconnexion des Classes**

1. **Choix du niveau** : Détermine les dimensions du labyrinthe.
2. **Génération** : Création dynamique d’un labyrinthe unique.
3. **Déplacement** : Interaction fluide entre le joueur et le labyrinthe.
4. **Victoire** : La classe Jeu gère la fin de la partie.

**Conclusion** : Le jeu **Labyrinthe** repose sur une conception claire et modulaire. Chaque classe contribue à une partie spécifique de la logique du jeu, assurant flexibilité, lisibilité, et robustesse de l'ensemble du programme.