كلية العلوم و التقنيات بطنجة
Faculté des Sciences et Techniques de Tanger

جامعة عبد المالك السعدي
Université Abdelmalek Essaadi

RAPPORT DE MINI PROJET

Par

BEGDOURI TERRAF MARWA

AIT AHMED OABD MARIAM

BENALI MAROUAN

BENBIGA SALMA

DEVELOPPEMENT D'UN JEU DE CASSE-TETE DE STYLE LABYRINTHE (MAZES) EN C++ AVEC RAYLIB

Encadrante : **Ikram Ben abdel ouahab**

Introduction :

Ce projet a pour but de développer un jeu de type casse-tête où le joueur doit naviguer à travers un labyrinthe généré procéduralement pour atteindre une sortie. Le jeu utilise les concepts de **programmation orientée objet (POO)** et exploite la bibliothèque graphique **Raylib** pour offrir une expérience immersive avec une interface graphique 2D.

L'idée principale est d'allier apprentissage de la POO avec des concepts plus pratiques, tels que les algorithmes de génération de labyrinthes, la gestion des déplacements, et l'interaction graphique. Le projet met également en valeur l'utilisation de textures, sons et systèmes de niveaux pour enrichir le gameplay.

Objectifs :

1. **Apprentissage et application de la POO en C++** : Organisation du code en classes pour modéliser différents aspects du jeu.
2. **Création d'un gameplay captivant** : Implémenter un jeu amusant avec une difficulté progressive.
3. **Génération procédurale de labyrinthes** : Utilisation d'algorithmes pour générer automatiquement des labyrinthes résolvables.
4. **Interface utilisateur intuitive** : Gestion graphique avec Raylib, incluant des textures, sons et chronomètres.
5. **Multiplateforme** : Exploiter la portabilité de Raylib pour rendre le jeu fonctionnel sur plusieurs plateformes.

Fonctionnalités :

1. **Génération aléatoire de labyrinthes** :
    - Un labyrinthe est généré à chaque partie grâce à un algorithme procédural (DFS - *Depth-First Search*).
    - Tous les labyrinthes sont garantis résolvables.
2. **Niveaux de difficulté** :
    - **Facile** : Petits labyrinthes avec peu d'obstacles.
    - **Moyen** : Labyrinthes de taille moyenne avec une navigation plus complexe.
    - **Difficile** : Grands labyrinthes avec plusieurs chemins trompeurs et obstacles.
3. **Interface graphique 2D** :
    - Représentation visuelle du labyrinthe avec des textures pour le joueur (souris) et la sortie (fromage).
    - Navigation au clavier.
4. **Mécanique de jeu immersive** :
    - Chronomètre pour mesurer le temps pris pour terminer chaque niveau.
    - Option de réinitialisation d'une partie pour générer un nouveau labyrinthe.
5. **Système de progression** :
    - Transition automatique entre les niveaux.

- Changements dynamiques des textures et musiques selon le niveau.

6. **Options supplémentaires** (optionnel) :
   - Possibilité de désactiver ou activer la musique en temps réel.
   - Sauvegarde des meilleurs temps.

Organisation du Code :

1. Classe Cell :

- Modélise une cellule du labyrinthe.
- **Attributs :**
  - **isWall :** indique si la cellule est un mur.
  - **visited :** indique si la cellule a été visitée lors de la génération.
  - **isButton (optionnel) :** peut être utilisé pour des mécaniques supplémentaires.

2. Classe Maze :

- Gère la génération du labyrinthe et son affichage.
- **Méthodes importantes :**
  - **generateMazeRecursive** : Génération procédurale via DFS.
  - **isWall :** Vérifie si une position donnée est un mur.
  - **draw :** Affiche graphiquement le labyrinthe.

3. **Classe Player** :

- Représente le joueur contrôlé par l'utilisateur.
- **Attributs :**
  - x et y : position du joueur dans le labyrinthe.
- **Méthodes :**
  - move : Permet au joueur de se déplacer en évitant les murs.
  - draw : Affiche graphiquement le joueur.

4. **Boucle principale (main)** :

- Gère la logique globale :
  - Sélection du niveau et de la difficulté.
  - Gestion des événements (déplacement, progression, musique).
  - Transition entre les niveaux et affichage des chronomètres.

Algorithmes Utilisés

1. Génération procédurale du labyrinthe :

- Utilisation de l'algorithme DFS (*Depth-First Search*) :
  1. Partir d'une cellule de départ.
  2. Marquer la cellule comme visitée.
  3. Explorer aléatoirement les voisins non visités en cassant les murs entre les deux cellules.

4. Retourner sur ses pas si aucune option n'est possible.
    - Cet algorithme garantit des labyrinthes avec une seule solution.

2. **Gestion des déplacements du joueur** :
    - Vérification des collisions : Empêche le joueur de traverser les murs.

Aspect Visuel

- **Textures** :
    - Souris (joueur) : Différentes textures selon la difficulté.
    - Fromage (sortie) : Icône colorée indiquant la destination.
- **Couleurs** :
    - Les labyrinthes changent de couleur selon la difficulté (gris clair ou sombre).
- **Chronomètre** :
    - Temps écoulé affiché en haut de l'écran.

Défis Techniques

1. **Génération des labyrinthes** :
    - Implémenter un algorithme DFS optimisé pour éviter les surcharges.

2. **Interface graphique fluide** :
    - Utiliser efficacement Raylib pour afficher et mettre à jour les éléments.
3. **Gestion des niveaux** :
    - Ajustement dynamique des tailles de labyrinthes, textures, et musique.

Résultats Obtenus

Le jeu est entièrement fonctionnel, avec :

- Une génération automatique de labyrinthes garantissant une solution.
- Une navigation intuitive et fluide à l'aide des touches directionnelles.
- Des transitions entre niveaux avec un ajustement des textures, musiques et tailles de labyrinthes.
- Une visualisation claire du temps pris pour chaque niveau.

Conclusion

Ce projet a permis de renforcer les compétences en programmation orientée objet et en développement graphique tout en intégrant des mécaniques ludiques captivantes. Grâce à Raylib, l'aspect visuel est à la fois simple et efficace. Les résultats démontrent la faisabilité de créer un jeu multiplateforme engageant tout en respectant les concepts fondamentaux de la POO.