PROJET

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

Encadré par :

Pr. Ikram BENABDELOUAHAB

Nom d'etudiants :

Wail CHAIRI MAHJOR

Yassir AHAROUN

Introduction

Le projet consiste à concevoir un jeu interactif de labyrinthe en utilisant **C++** et la bibliothèque graphique **Raylib**. Ce jeu met en œuvre les principes de la programmation orientée objet (**POO**) pour assurer une architecture modulaire et extensible.
L'objectif principal est de naviguer dans un labyrinthe généré aléatoirement pour atteindre la sortie, tout en proposant différents niveaux de difficulté. Ce rapport détaille les étapes de développement, les fonctionnalités implémentées, les résultats obtenus, et nos réflexions sur le projet.

Objectifs

1. Générer des labyrinthes aléatoires garantissant une solution valide.
2. Intégrer une interface graphique visuelle et intuitive via **Raylib**.
3. Proposer plusieurs niveaux de difficulté **(facile, moyen, difficile)** qui influencent la taille et la complexité des labyrinthes.
4. Suivre le temps pris pour compléter chaque labyrinthe.

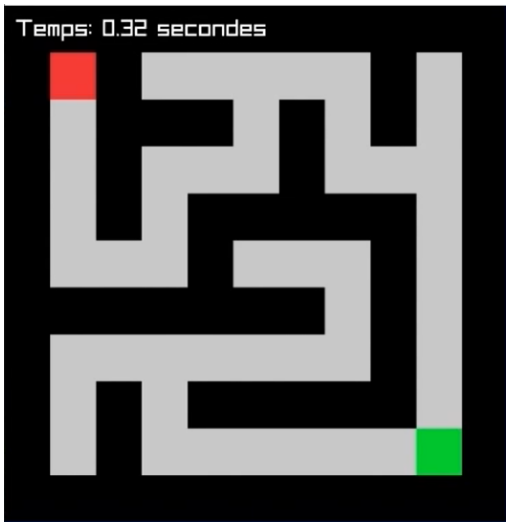
1. Initialisation et Configuration de Raylib

- **Étape 1 :** Mise en place de l'environnement de développement sous Windows avec Visual Studio.
- **Étape 2 :** Installation de **Raylib** .
- **Étape 3 :** Création d'un projet **C++** de base, et vérification que la fenêtre **Raylib** fonctionne correctement.

2. Génération d'un Labyrinthe Aléatoire

- **Étape 4 :** Conception de la classe Labyrinthe pour gérer la génération de la grille à l'aide de l'algorithme de backtracking, garantissant une solution toujours valide.
- **Étape 5 :** Mise en place des couleurs pour les murs (noir), les chemins (gris clair), et la sortie (vert). Test de l'affichage visuel

  ,Voir (Figure 1).



(Figure 1)

3. Gestion des Déplacements du Joueur

- **Étape 6 :** Création de la classe Joueur, avec les coordonnées x et y pour suivre la position dans la grille.
- **Étape 7 :** Implémentation de la méthode move pour gérer les déplacements en vérifiant que le joueur ne traverse pas les murs.

4. Ajout des Niveaux de Difficulté

- **Étape 8 :** Conception de la classe Niveau pour ajuster la taille de la grille (rows, cols) et la complexité selon le niveau choisi.
- **Étape 9 :** Intégration d'un menu interactif pour sélectionner le niveau de difficulté ,(figure 2)

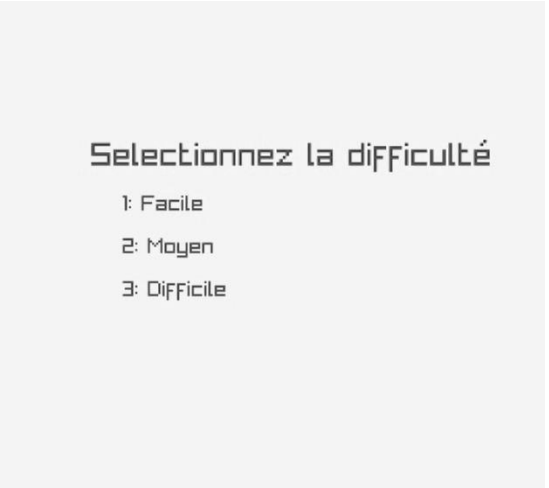


figure 2

5. Gestion de l'État du Jeu

- **Étape 10 :** Création de la classe Jeu pour orchestrer le déroulement du programme :
  - Affichage du menu principal.
  - Gestion des déplacements du joueur et de la condition de victoire.
  - Redémarrage du jeu avec une nouvelle grille lorsque le joueur gagne.

6. Finalisation et Tests

- **Étape 11 :** Ajout d'un chronomètre pour mesurer le temps pris par le joueur à résoudre le labyrinthe ,(figure 3).

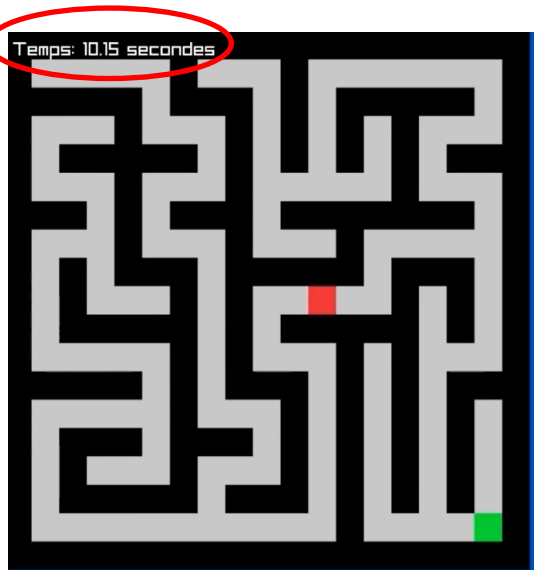


figure 3

- **Étape 12 :** Résolution des erreurs, comme les problèmes de dimensions de la grille et la compatibilité entre les classes.
- **Étape 13 :** Tests approfondis pour garantir une expérience fluide.

Architecture du Code

1. Classe Labyrinthe

- **Responsabilité :** Générer, afficher, et valider la logique du labyrinthe.
- **Méthodes principales :**
  - generateMaze() : Génère un labyrinthe aléatoire avec une solution valide.
  - drawMaze() : Affiche le labyrinthe.
  - canMoveTo() : Vérifie si une cellule est accessible.

2. Classe Joueur

- **Responsabilité :** Gérer les déplacements du joueur.
- **Méthodes principales :**
  - move(dx, dy, maze) : Déplace le joueur en respectant les murs.
  - reset() : Réinitialise la position du joueur.

3. Classe Niveau

- **Responsabilité :** Gérer la taille et la complexité du labyrinthe selon la difficulté.
- **Méthodes principales :**
  - setDifficulty() : Définit les paramètres en fonction de la difficulté.

4. Classe Jeu

- **Responsabilité :** Orchestrer les différentes étapes du jeu.
- **Méthodes principales :**
  - run() : Boucle principale.
  - showDifficultySelection() : Menu pour sélectionner la difficulté.
  - playGame() : Gère les interactions du joueur avec le labyrinthe.

Fonctionnalités Implémentées

1. **Labyrinthe procédural :** Généré à l'aide d'un algorithme de backtracking.
2. **Niveaux de difficulté :**
   - Facile : Grille de 11x11 cellules.
   - Moyen : Grille de 19x19 cellules.
   - Difficile : Grille de 31x31 cellules.
3. **Interface graphique :**
   - Couleurs intuitives pour les murs, les chemins, et la sortie.
   - Le joueur est représenté par un rectangle rouge.
4. **Chronomètre :** Temps affiché en haut de l'écran.
5. **Redémarrage :** Génération d'un nouveau labyrinthe à tout moment.

Résultats Obtenus

- **Interface fonctionnelle :** Navigation fluide dans le labyrinthe.
- **Labyrinthes aléatoires :** Respect des règles de connexité pour garantir une solution.
- **Chronomètre précis :** Mesure du temps pris pour compléter le labyrinthe.
- **Gestion des erreurs :** Résolution des problèmes liés à la logique de mouvement et à la taille de la grille.

Défis Rencontrés

1. **Gestion des classes :** Conception d'une architecture modulaire avec une séparation claire des responsabilités.
2. **Génération du labyrinthe :** Implémentation correcte de l’algorithme de backtracking.
3. **Interaction joueur-labyrinthe :** Validation des mouvements pour éviter les bugs de collision.
4. **Intégration de Raylib :** Adaptation à l'environnement graphique et gestion des événements clavier.

Notre Parcours et Réflexions

Au cours de ce projet, nous avons travaillé étape par étape :

1. **Exploration :** Nous avons commencé par des tests simples pour comprendre le fonctionnement de **Raylib**.
2. **Conception incrémentale :** Chaque fonctionnalité (labyrinthe, joueur, niveaux, état du jeu) a été ajoutée individuellement avant d’être intégrée.
3. **Itérations et corrections :** Plusieurs cycles de débogage ont permis de résoudre des problèmes comme l’absence de murs sur certaines bordures ou la redondance de certaines méthodes.
4. **Travail collaboratif :** La discussion constante sur les choix de conception nous a aidés à comprendre les principes clés de la **POO** et des algorithmes procéduraux.

Améliorations Futures

1. **Tableau des scores :** Ajouter un classement pour enregistrer les meilleures performances.
2. **Obstacles dynamiques :** Ajouter des pièges ou des ennemis pour augmenter la difficulté.
3. **Thèmes graphiques :** Permettre de personnaliser l'apparence du labyrinthe.
4. **Sauvegarde de progression :** Enregistrer l’état du jeu pour reprendre plus tard.

Conclusion

Ce projet a été une excellente opportunité pour appliquer les concepts de POO et explorer les algorithmes de génération procédurale. La bibliothèque Raylib a permis de créer une interface visuelle engageante et facile à utiliser.
Grâce à une approche structurée, les objectifs principaux ont été atteints, tout en laissant de la place pour des améliorations futures.