Rapport de Projet de Fin de Module

*Développement d'un Jeu de Casse-Tête de Style Labyrinthe en **C++** avec **Raylib***

Module : Programmation Orientée Objet (POO)

Année : 2024-2025

Rédigé par :

- El Akhal Aya
- Edrissi Noussaiba
- Zeroual Hiba
- El-Taalabi Soufiene

Encadré par :

Prof. Ikram Ben Abdel Ouaheb

Table De Matière

I. Introduction :

Ce projet s'inscrit dans le cadre du module de Programmation Orientée Objet en C++, avec pour objectif principal la création d'un jeu interactif de labyrinthe utilisant la bibliothèque graphique **Raylib**. Ce jeu propose une expérience immersive où le joueur doit résoudre un labyrinthe généré aléatoirement, tout en naviguant entre différents niveaux de difficulté adaptés à ses compétences.

L'idée de ce projet répond à deux aspirations : d'une part, consolider les concepts clés de la programmation orientée objet tels que la modularité, la réutilisabilité et la structuration du code ; d'autre part, explorer des outils modernes comme **Raylib** pour créer une interface graphique intuitive et attractive. En associant ces deux dimensions, le projet permet de relier théorie et pratique dans un cadre stimulant.

Le développement d'un jeu de labyrinthe repose sur des défis techniques variés, notamment la génération procédurale de labyrinthes garantissant une solution unique, la gestion des

déplacements du joueur, et l'intégration fluide de mécanismes comme le chronomètre ou les sauvegardes. Ces aspects enrichissent le projet et offrent une opportunité d'apprendre à gérer un développement structuré, à la fois graphique et algorithmique.

Ce rapport a pour vocation de présenter en détail les étapes de conception, de développement et de validation du jeu, ainsi que les solutions mises en œuvre pour relever les défis techniques rencontrés. Ce projet représente une occasion unique d'allier créativité, rigueur technique et apprentissage pratique dans un cadre motivant et stimulant.

II. Cahier De Charge :

1. Exigences Fonctionnelles :

1.1. Génération automatique de Labyrinthes :

- ➔ Création procédurale de labyrinthes respectant les règles de connexité.
- ➔ Réinitialisation automatique pour générer un nouveau labyrinthe à chaque partie.

1.2. Niveaux et difficultés :

- ➔ Facile : petit labyrinthe.
- ➔ Moyen : labyrinthe de taille moyenne avec des chemins plus complexes.
- ➔ Difficile : grand labyrinthe avec des chemins trompeurs.

1.3. Interface graphique :

- ➔ Représentation des labyrinthes en 2D (murs, chemins, et joueurs).
- ➔ Déplacement du joueur à l'aide des touches directionnelles du clavier.

1.4. Mécanique de jeu :

- ➔ Atteindre la sortie du labyrinthe à partir de l'entrée.
- ➔ Chronomètres affichant le temps pris pour terminer le labyrinthe.
- ➔ Option de réinitialisation de la partie.

1.5. Sauvegarde et score (optionnel) :

- ➔ Enregistrement des meilleurs temps.
- ➔ Affichage d'un tableau des scores.

2. Exigences Techniques :

- ➔ Langage de programmation : **c++**.
- ➔ Bibliothèque graphique : **Raylib**.
- ➔ Paradigme : **Programmation Orientée Obje**t.
- ➔ Architecture technique :
    - **Classe labyrinthe.**
    - **Classe joueur.**
    - **Classe niveau**
    - **Classe jeu.**

3. Contraintes :

- ➔ ***Performance :*** temps de réponse rapide pour la génération des labyrinthes et les interactions.
- ➔ ***Stabilité :*** gestion des erreurs pour éviter les crashs ou comportements imprévus.
- ➔ ***Accessibilité :*** interface simple et facile à comprendre pour les utilisateurs.

III. Développement :

1. Approche adoptée pour coder chaque fonctionnalité

1.1. Organisation du projet

L'implémentation du **Maze Game** est conçue de manière modulaire afin d'assurer une organisation claire, une lisibilité accrue et une maintenabilité optimale du code. Voici les principaux fichiers du projet :

- ***config. H*** *:*
  - Définit les variables globales telles que la taille des cellules et les dimensions de la grille.
  - Utilise des déclarations externes pour éviter les redéfinitions multiples lors de l'inclusion des en-têtes.
  - Avantages :
    - Assure une déclaration unique des variables, accessibles dans tout le programme.
    - Évite les erreurs liées aux redéfinitions.

- ***Game. H et game.cpp :***
  - Contient la logique principale du jeu : *initialisation*, *boucle principale* et *gestion des états*.
  - Les fonctions pour chaque état, comme *menu* ou gameplay, sont organisées séparément pour une gestion simplifiée.
- ***Maze. H et maze.cpp :***
  - Gère la génération du labyrinthe à l'aide de l'algorithme **DFS** (***Depth-First Search***).
  - Définit une structure pour représenter chaque cellule du labyrinthe.
- ***Player. H et player.cpp :***
  - Gère les actions du joueur : déplacements, positionnement et affichage.
- ***Goal. H et goal.cpp :***
  - Définit la position et l'affichage de l'objectif (le but à atteindre).
- ***Menu. H et menu.cpp :***
  - Contient la gestion des menus principaux, pause et fin de jeu.
- ***Scoreboard. H et scoreboard.cpp :***
  - S'occupe de la gestion des meilleurs scores : chargement, sauvegarde et mise à jour.

1.2. *Gestion des états du jeu*

Pour organiser la logique principale, un système d'états a été mis en place (***Game State***). Chaque état est géré de manière distincte, avec une méthode dédiée pour les événements et l'affichage.

Les états principaux incluent :

- *Menu principal :* Permet au joueur de démarrer ou de quitter le jeu.
- *Menu de difficulté :* Permet de sélectionner le niveau de difficulté.
- *Gameplay :* Contrôle les déplacements du joueur et vérifie les conditions de victoire.
- *Pause :* Met le jeu en pause.
- *Game Over :* Affiche les scores et offre des options pour redémarrer ou quitter.

Une méthode principale assure la transition entre les états grâce à un système clair basé sur des conditions.

1.3. *Génération du labyrinthe avec DFS*

La génération procédurale du labyrinthe utilise l'algorithme *Depth-First Search* (**DFS**) :

1. Initialisation : Chaque cellule du labyrinthe est marquée comme fermée par des murs et possède un drapeau pour vérifier si elle a été explorée.
2. Exploration des voisins : Les directions sont aléatoirement mélangées pour produire des labyrinthes imprévisibles.
3. Suppression des murs : Si une cellule voisine n'a pas encore été visitée, les murs entre la cellule actuelle et la voisine sont retirés.
4. Récursion : L'algorithme est appelé récursivement jusqu'à ce que toutes les cellules soient visitées.

1.4. *Gestion du joueur*

La classe **Player** est chargée de gérer les actions du joueur, comme :

- *Déplacements conditionnés :* Le joueur ne peut avancer que si les murs correspondants ont été détruits.

- *Limitation des déplacements rapides :* Un délai est appliqué entre chaque déplacement pour éviter les mouvements trop rapides.
- *Affichage dynamique :* La texture du joueur est redimensionnée en fonction de la difficulté, afin de s'adapter à la taille des cellules.

1.5. *Gestion des scores*

Les scores sont gérés par une classe dédiée :

- *Chargement des scores :* Les meilleurs temps sont lus à partir d'un fichier lors de l'initialisation.
- *Mise à jour des scores :* Si un joueur réalise un temps meilleur, celui-ci est intégré au classement.
- *Sauvegarde des scores :* Les nouveaux meilleurs temps sont écrits dans le fichier pour assurer une persistance entre les sessions.

2. Utilisation de Raylib : Description des éléments graphiques et leur intégration

Raylib, une bibliothèque de développement de jeux vidéo, a été utilisée pour simplifier la gestion des éléments graphiques.

2.1. *Création de la fenêtre et configuration initiale*

- La fenêtre est configurée avec des dimensions fixes pour assurer une compatibilité sur différentes résolutions d'écran.
- Le jeu est limité à 60 images par seconde pour garantir une fluidité optimale.

2.2. *Affichage du labyrinthe*

Le labyrinthe est dessiné à l'écran en représentant les murs de chaque cellule à l'aide de lignes simples. Les couleurs et l'épaisseur des murs sont ajustées en fonction de la difficulté choisie par le joueur.

2.3. *Affichage du joueur et du but*

- *Joueur (Player) :* Représenté par une texture qui est redimensionnée et centrée dans sa cellule.

- **But (Goal)** : Une texture similaire est utilisée pour marquer l'objectif, elle aussi ajustée selon la difficulté.

2.4. *Menus et textes*

Les menus et instructions sont affichés avec des textes centrés et ajustés en fonction de la résolution. Des couleurs différentes sont utilisées pour améliorer la lisibilité.

2.5. *Gestion des thèmes visuels*

Des thèmes visuels sont appliqués en fonction du niveau de difficulté : couleurs de fond et des murs, offrant une expérience visuelle immersive.

3. Tests unitaires et validation

3.1. *Tests des déplacements du joueur*

Les déplacements ont été testés pour garantir que le joueur respecte les murs et ne les traverse pas.

3.2. *Tests de génération du labyrinthe*

L'algorithme de génération a été testé pour s'assurer que chaque cellule du labyrinthe est accessible et correctement connectée.

3.3. *Tests des états du jeu*

Les transitions entre les différents états (*MAIN_MENU, DIFFICULTY_MENU,PAUSE, GAME_OVER)* ont été testées pour assurer une navigation fluide.

3.4. *Tests de persistance des scores*

Les fonctions de sauvegarde et de chargement des scores ont été testées pour garantir que les meilleurs temps sont correctement enregistrés et restaurés.

3.5. *Tests d'affichage graphique*

L'alignement des éléments visuels, le redimensionnement des textures et l'affichage des textes ont été vérifiés pour une présentation fluide et cohérente.

IV. Résultat Et Analyse :

1. Analyse des performances :

- *Fluidité du jeu* : Le jeu a été conçu pour garantir une expérience utilisateur fluide, avec des déplacements du joueur sans latence notable. Grâce à l'optimisation des boucles principales et à l'utilisation efficace de Raylib, les taux de rafraîchissement de l'écran ont été maintenus au-dessus de 60 FPS sur une machine standard.

- *Temps de génération des labyrinthes* : L'algorithme utilisé (DFS) a permis de générer des labyrinthes rapidement, avec un temps moyen de génération de moins de 100 ms pour un labyrinthe de taille moyenne (20x20). Pour les labyrinthes de grande taille (50x50), le temps de génération reste sous les 500 ms.

2. Difficultés Rencontrées :

- *Problème de connexité des labyrinthes :*

Lors des premiers tests, certains labyrinthes générés n'étaient pas solvables en raison d'erreurs dans l'algorithme de génération.
Solution : Une vérification supplémentaire a été ajoutée à l'algorithme pour s'assurer que chaque labyrinthe généré possède au moins une solution viable.

- *Gestion des collisions :*

Les collisions entre le joueur et les murs du labyrinthe n'étaient pas toujours correctement détectés, ce qui permettait parfois au joueur de traverser les murs.
Solution : Une validation des déplacements a été intégrée pour vérifier que la case cible est accessible avant d'effectuer le mouvement.

- *Accordement des codes entre modules :*

L'intégration des différentes parties du projet (génération des labyrinthes, interface utilisateur, gestion des événements) a nécessité des ajustements pour assurer une communication fluide entre les classes.
Solution : Une révision globale des interactions entre classes a été effectuée, avec l'ajout de tests unitaires pour détecter rapidement les incohérences.

V. Extension Et Amélioration :

1. Fonctionnalités Ajoutées :

➔ *Score en fonction du temps passé :* Le score est calculé en fonction du temps que le joueur met pour terminer un niveau. Plus le joueur est rapide, plus il obtient un score élevé, ce qui motive à jouer efficacement. Ce mécanisme ajoute un aspect compétitif au jeu et encourage les joueurs à rejouer pour améliorer leurs performances.

➔ *Changement de couleur à chaque niveau :* Chaque niveau possède un thème distinct basé sur une palette de couleurs spécifique. Ce changement visuel permet de différencier les niveaux et offre une expérience plus variée et immersive. Par exemple, un niveau pourrait être en tons bleus, et le suivant en tons orange, pour marquer une progression visuelle.

2. Perspectives d'amélioration :

➔ *Amélioration du système de score :* Ajouter des bonus ou des pénalités basées sur des actions spécifiques, comme trouver un objet caché ou entrer dans une impasse.
Implémenter un tableau des scores pour comparer les performances des joueurs entre différentes sessions ou avec d'autres joueurs.

➔ *Obstacles dynamiques :* Ajouter des pièges ou des blocs en mouvement dans les labyrinthes pour augmenter la difficulté. Ces obstacles pourraient être évités en fonction du timing ou de la réflexion.

➔ *Obstacles dynamiques :* Ajouter des pièges ou des blocs en mouvement dans les labyrinthes pour augmenter la difficulté. Ces obstacles pourraient être évités en fonction du timing ou de la réflexion.

➔ *Modes de jeu supplémentaires :*

- ***Mode contre-la-montre :*** Finir un labyrinthe dans un temps limité pour ajouter de la pression et du challenge.
- ***Mode survie :*** Les labyrinthes deviennent de plus en plus difficiles, et le joueur doit survivre le plus longtemps possible.

➔ *Effets visuels et sonores :* Intégrer des animations pour les transitions de niveaux ou des effets sonores pour les interactions (déplacements, score obtenu, etc.) pour enrichir l'expérience.

VI. Conclusion :

Le projet de jeu de labyrinthe développé avec ***Raylib*** en ***C++*** a permis de concevoir un jeu fonctionnel répondant aux objectifs définis au départ. Les fonctionnalités principales, comme la génération de labyrinthes, la gestion des niveaux, et l'interaction du joueur avec l'environnement, ont été implémentées avec succès. De plus, l'ajout d'un système de score basé sur le temps et d'un changement de couleur par niveau a enrichi l'expérience utilisateur.

Ce projet a également été une occasion précieuse pour approfondir nos compétences en programmation orientée objet et en utilisation de bibliothèques graphiques comme ***Raylib***. Nous avons acquis une meilleure compréhension de la gestion des projets en équipe, notamment dans la répartition des tâches, la coordination, et la résolution collaborative des défis techniques.
En conclusion, ce projet ne se limite pas à une simple réalisation technique, mais représente aussi un apprentissage enrichissant sur le plan académique et personnel. Il illustre l'importance de la planification, de la collaboration, et de la persévérance dans le développement de solutions logicielles complexes. Avec les améliorations envisagées, ce jeu pourrait évoluer vers une version plus complète et immersive, présentant un potentiel pour être utilisé comme projet de référence ou base pour des développements futurs.

VII. Annex :

https://www.raylib.com/cheatsheet/cheatsheet.html

https://learn.raylib.com/

https://youtu.be/HyK_Q5rrcr4?si=krMd21yj6xHjeIgj

https://youtu.be/ioUl1M77hww?si=DFGou9SvTHFJcTvd

https://youtu.be/RGzj-PF7D74?si=PiWmJbKZ1AzJf2dJ