Royaume du Maroc
Universite Abdelmalek Essaadi
Faculté des Sciences et Techniques Tanger
Département :Génie Informatique
Semestre 5:IDAI

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

Réalisé par :

- Salma EL BANNOUDI
- Meryem AIT TIMCHIRINE
- Essafia HAJAJ ELAAROUD
- Hala EL HMAIDI

Encadré par :

Pr. Ikram BENABDELOUAHAB

Table des matières:

I. Introduction : 1

II. Matériels et Méthodes 2

II.1. Definition de POO en C++: 2

II.2. Definition de Raylib: 2

II.3. Fonctionnalités: 3

II.4. Architecture du Jeu 4

1.Game : 4

2.Labyrinthe : 4

3.Joueur : 5

4.Niveau : 6

5.jeu : 7

6.Obstacle : 9

III- Résultat 9

IV-Conclusion: 10

I. Introduction :

Les jeux vidéo de labyrinthes, également connus sous le nom de **"maze games",** occupent une place emblématique dans l'histoire du divertissement interactif, ils stimulent notre capacite à résoudre des problèmes et à penser de manière logique. En général, un jeu de labyrinthe consiste à naviguer à travers un réseau complexe de chemins et d'obstacles, où l'objectif principal est de trouver une sortie ou d'atteindre un but spécifique.

Ce projet présente le développent d'un jeu vidéo de type labyrinthe en C++, utilisant la bibliothèque Raylib. Le projet a pour objectif de créer une expérience de jeu immersive ou le joueur doit naviguer dans des labyrinthes génères aléatoirement. Les fonctionnalités clés incluent la génération procédurale de labyrinthes, trois niveaux de difficulté, une interface graphique et un système de chronométrage. Ce projet détaille les choix techniques, les défis rencontres et les résultats obtenus.

II. Matériels et Méthodes

II.1. Definition de POO en C++:

C++ est un langage de programmation généraliste connu pour sa puissance et sa flexibilité. Il est particulièrement adapté au développement de logiciels nécessitant des performances élevées, comme les jeux vidéo, grâce à ses fonctionnalités avancées telle que et la programmation orientée objet.

La programmation orientée objet (POO) est le terme utilisé pour décrire une approche de programmation basée sur des objets et des classes. Le paradigme orienté objet nous permet d'organiser le logiciel comme une collection d'objets constitués à la fois de données et de comportements.

II.2. Definition de Raylib:

Raylib est une bibliothèque de développement de logiciels open source et multiplateforme, principalement utilisée pour créer des jeux vidéo et des applications d'interface graphique. Elle a été lancée initialement en novembre 2013. Raylib est écrite en C et est compatible avec plusieurs plateformes, y compris Windows, Linux, macOS, Android,...

La bibliothèque est conçue pour être simple et facile à utiliser, ce qui la rend idéale pour les débutants en programmation de jeux. Elle prend en charge le rendu 2D et 3D, la gestion des entrées, le chargement d'images et de textures, ainsi que la gestion de l'audio.

II.3. Fonctionnalités:

Le jeu se distingue par les fonctionnalités suivantes :

1.Génération automatique de labyrinthes :

- ✓ A chaque partie, un nouveau labyrinthe est généré de manière aléatoire.
- ✓ Les labyrinthes respectent des règles de connexité pour garantir qu'une solution existe.

2.Troix niveaux de difficultés :

- ✓ Facile : petit labyrinthe avec peu d'obstacles.
- ✓ Moyen : labyrinthe de taille moyenne, plus complexe avec plus d'obstacles.
- ✓ Difficile : grand labyrinthe avec des chemins trompeurs et des obstacles nombreux.

3.Interface graphique en 2D :

- ✓ Le jeu est présente de manière graphique grâce à Raylib.
- ✓ Les labyrinthes sont représentés par des blocs(murs) et un personnage contrôlable.

4.Mecanisme de jeu :

- ✓ Le joueur commence à l'entrée du labyrinthe et doit atteindre la sortie.
- ✓ Un chronomètre affiche le temps pris pour résoudre le labyrinthe.
- ✓ Possibilité de réinitialiser la partie à, pour générer un nouveau labyrinthe.
- ✓ Possibilité de mettre le jeu en pause et de le reprendre à tout moment.
- ✓ Option "Suivant" pour passer au niveau suivant après avoir terminé un labyrinthe.
- ✓ Option "Quitter" pour fermer le jeu.

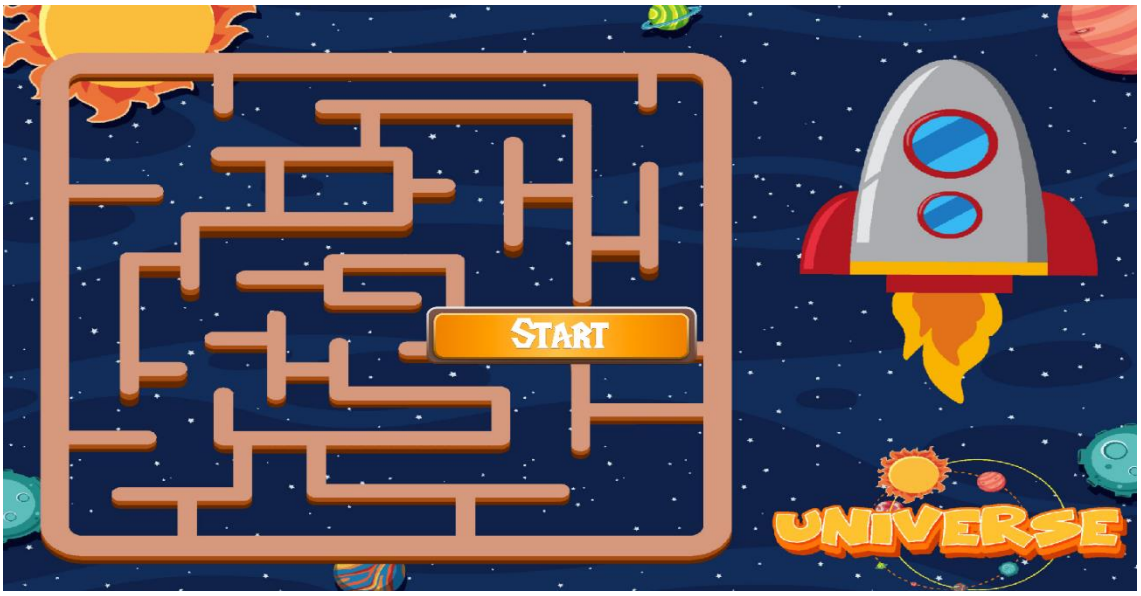
5.Système de Sauvegarde et Scores :

- ✓ Sauvegarde du temps pris pour chaque partie et affichage des meilleurs scores.

II.4. Architecture du Jeu

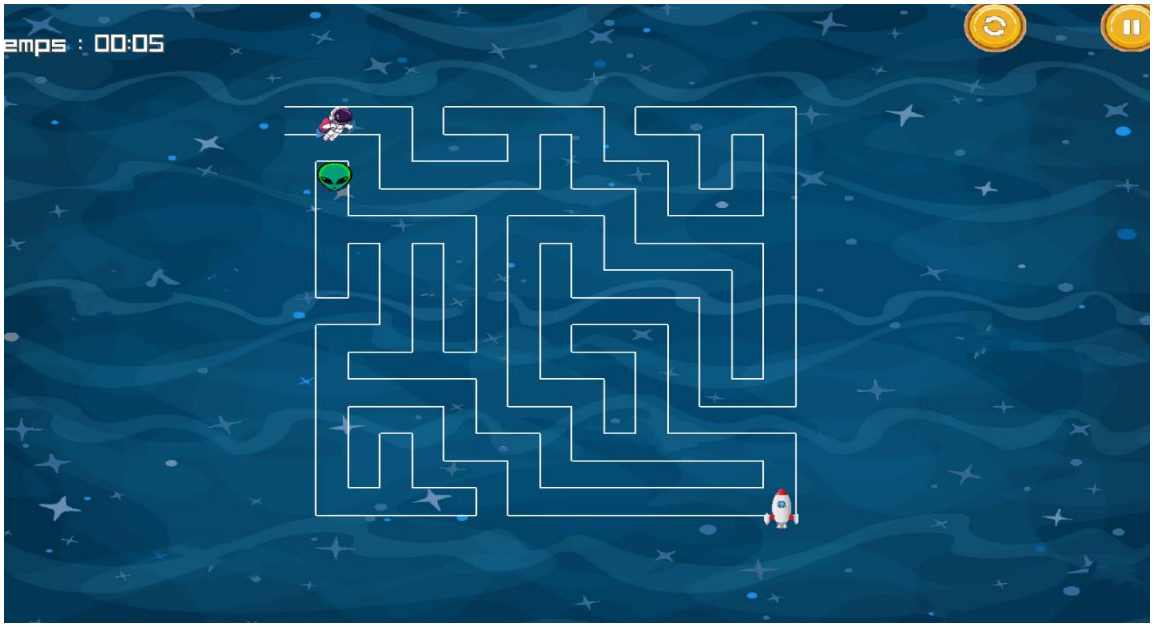
Le jeu a été conçu selon les principes de la POO, avec quatre classes principales :

1.Game :



La classe **game** représente un jeu avec une interface utilisateur graphique, incluant un écran d´accueil. Dans le constructeur, elle charge les textures nécessaires pour l´image de fond et le bouton ¨Démarrer/Start. La méthode **display Interface ()** est responsable de l´affichage de l´interface, en dessinant l´image de fond et le bouton centre a l´écran. La méthode **detectStartClick()** détecte si l`utilisateur clique sur le bouton ¨Démarrer¨.

2.Labyrinthe :

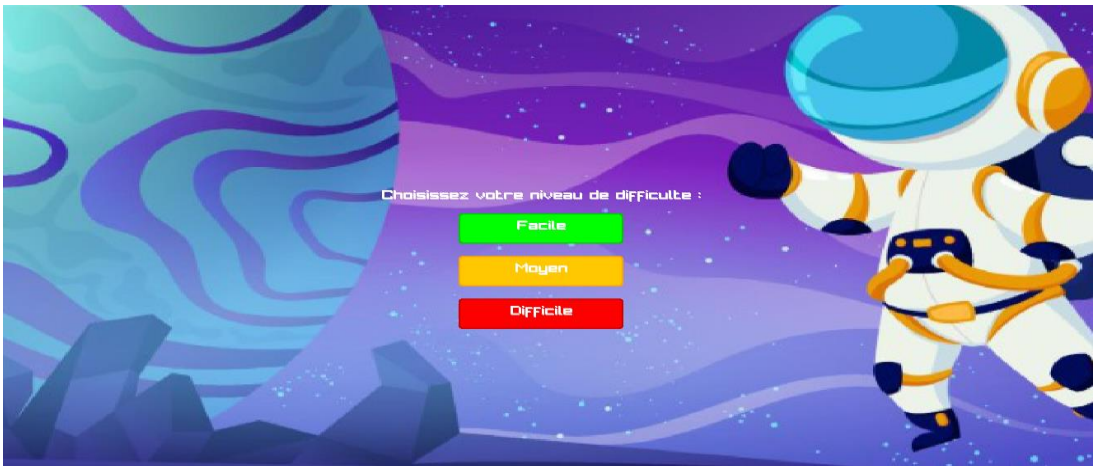


La classe **labyrinthe** est responsable de la gestion de la grille du labyrinthe ainsi que de sa génération procédurale. Elle est initialisée avec des dimension spécifique (largeur et hauteur) et crée une grille 2D. Cette class contient 3 méthodes, la méthode **generer()** utilise un algorithme de recherche en profondeur aléatoire pour créer des chemins dans le labyrinthe ,en s'assurant que chaque cellule est accessible es qu'une solution existe. La classe définit également un point de sortie, qui est détermine par la méthode **arriver (),** cherchant un espace ouvert dans la grille pour y placer la sortie .la méthode **afficher ()** est utilisée pour dessiner le labyrinthe à l'écran, en utilisant des lignes pour représenter les murs et une texture pour la sortie. Des méthodes getter permettent d'accéder aux dimensions du labyrinthe et à la position de la sortie, ainsi qu'à la valeur d'une case spécifique dans la grille. Cette classe constitue un élément central du jeu, car elle gère la structure et l'affichage du labyrinthe, offrant ainsi une base solide pour l'interaction du joueur.

3.Joueur :

La classe **Joueur** gère le personnage contrôle par le joueur dans le labyrinthe .Elle est initialisée avec des coordonnées de départ (x,y) et une taille fixe pour le personnage ,qui est représenté par une texture (une image d'astronaute).La méthode **afficher()** est responsable du rendu graphique du joueur à l'écran ,en utilisant la fonction **DrawTexturePro** pour dessiner la texture a la position appropriée ,avec des dimensions spécifiques .La méthode **deplacer()** permet de déplacer le joueur dans le labyrinthe en vérifiant si la nouvelle position est un chemin valide .Si le mouvement est valide ,les coordonnées du joueur sont mises à jour .La méthode **aGagne()** vérifie si le joueur a atteint la sortie du labyrinthe en comparant sa position actuelle avec celle de la sortie .Des méthodes getter, , **getX()** et **getY()**, permettent d'accéder aux coordonnées du joueur .Cette classe est essentielle pour l'interaction du joueur avec le labyrinthe ,car elle gère les mouvement et les conditions de victoire , tout en assurant un rendu visuel approprié du personnage.

4.Niveau :

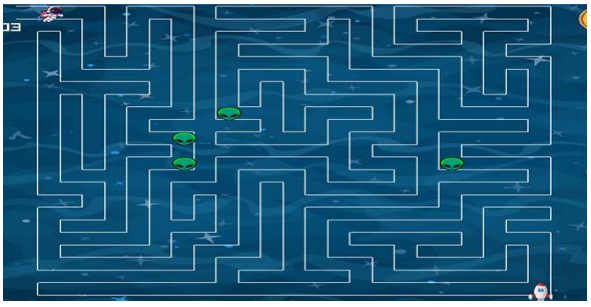


La classe **Niveau** est conçu pour gérer les différents niveaux de difficulté du jeu, en définissant la taille du labyrinthe et le nombre d'obstacles en fonction de la difficulté choisie par le joueur. Lors de l'initialisation, le constructeur prend en paramètre une chaîne de caractères représentant le niveau de difficulté (soit "Facile", "Moyen", ou "Difficile") et appelle la méthode **definirTaille()** pour établir la taille du labyrinthe correspondante. La méthode **getNombreObstacles()** retourne le nombre d'obstacles à placer dans le labyrinthe, qui varie selon le niveau de difficulté :

1 obstacle pour le niveau facile

3 pour le niveau moyen

4 pour le niveau difficile



La classe inclut également une méthode **augmenterDifficulte()**, qui permet d'augmenter la taille du labyrinthe jusqu'à une limite maximale de 50, offrant ainsi une flexibilité pour des niveaux de difficulté supplémentaires si nécessaire. Enfin, des méthodes getter telles que **getTailleLabyrinthe()** et **getDifficulte()** permettent d'accéder aux propriétés de la classe. Cette classe joue un rôle crucial dans l'adaptation de l'expérience de jeu en fonction des compétences du joueur, rendant le jeu accessible tout en offrant des défis appropriés.

5.jeu :

La classe **Jeu** représente la logique principale d'un jeu, intégrant des fonctionnalités essentielles telles que la gestion des scores, des obstacles, et des interactions utilisateur. Elle stocke les meilleurs scores pour trois niveaux de difficulté (facile, moyen, difficile) et gère le score actuel et le temps écoulé. Le constructeur initialise les textures pour les boutons de pause, de reprise et de réinitialisation, et crée des obstacles en fonction du niveau de difficulté, en s'assurant qu'ils ne se trouvent pas sur des cases occupées dans le labyrinthe. La classe inclut également des méthodes :

void afficherTemps(int x, int y) Affiche le temps écoulé depuis le début du jeu à une position donnée.

void afficherMenuPause() Affiche le menu de pause avec les boutons appropriés .

Void afficherBoutonReinitialiser() Dessine le bouton de réinitialisation à une position spécifique sur l'écran. **bool detecterClicReinitialiser()** vérifie si l'utilisateur a cliqué sur le bouton de réinitialisation en comparant la position de la souris avec celle du bouton. .

bool detecterClicRejouer(int buttonsY, int buttonHeight) Vérifie si le bouton "Rejouer" a été cliqué.

bool detecterClicQuitter(int buttonsY, int buttonHeight) vérifie si le bouton "Quitter" a été cliqué.

bool detecterClicSuivant(int buttonsY, int buttonHeight) vérifie si le bouton "Suivant" a été cliqué, si applicable.

void reinitialiserObstacles(const Niveau &niveau, const Labyrinthe &labyrinthe) Vide la liste des obstacles et en crée de nouveaux en fonction du niveau, en s'assurant qu'ils sont placés sur des cases vides.

void mettreAJourObstacles(const Labyrinthe &labyrinthe, float deltaTime) met à jour la position de chaque obstacle en fonction du temps écoulé.

bool detecterCollisionAvecObstacles(const Joueur &joueur)Vérifie si le joueur entre en collision avec un obstacle en comparant leurs positions .

void mettreAJourMeilleurScore(const std::string& difficulte, float temps) Met à jour le meilleur score en fonction de la difficulté et du temps écoulé.

void afficherMessagePerte(int &buttonsY, int buttonHeight) Affiche un message de perte et un bouton "Rejouer" lorsque le joueur perd.

void afficherFinJeu(float tempsEcoule, const Niveau &niveau, Jeu &jeu, int &buttonsY, int buttonHeight) Affiche un message de victoire, le temps écoulé, et le meilleur score à la fin du jeu.

6.Obstacle :

La classe **Obstacle** représente un obstacle dans un jeu, avec des attributs pour sa position (x, y), sa taille, sa vitesse de déplacement, sa direction, et le temps écoulé depuis son dernier mouvement. Le constructeur initialise ces attributs, en attribuant une direction aléatoire et en chargeant une texture pour l'affichage. La méthode déplacer gère le mouvement de l'obstacle en fonction du temps écoulé et de sa direction, tout en vérifiant la validité du mouvement par rapport à un labyrinthe donné.

III- Résultat

Les résultats obtenus après la mise en œuvre de ce projet sont les suivants :

- ✓ **Fonctionnement fluide de l'interface graphique :**

L'utilisation de Raylib a permis de concevoir une interface 2D esthétique, incluant la représentation visuelle des murs, du joueur et des obstacles.

- ✓ **Génération procédurale des labyrinthes :**

Le jeu réussit à générer des labyrinthes aléatoires qui respectent les règles de connexité et possèdent toujours une solution viable.

- ✓ **Différenciation claire des niveaux de difficulté :**

Les labyrinthes générés pour chaque niveau (facile, moyen, difficile) diffèrent significativement en taille et complexité, offrant une expérience de jeu adaptée à différents types de joueurs.

- ✓ **Chronométrage précis et système de sauvegarde :**

Le jeu intègre un chronomètre fonctionnel et affiche le meilleur score, permettant au joueur de suivre ses performances et de se comparer avec d'autres sessions.

- ✓ **Interaction intuitive avec le labyrinthe :**

Les contrôles de déplacement du joueur sont réactifs, et les collisions avec les murs et obstacles sont correctement gérées.

IV-Conclusion:

La réalisation de ce projet de jeu de casse-tête de type labyrinthe en C++ a permis d'appliquer nos connaissances en programmation orientée objet. Grâce à l'utilisation de la bibliothèque Raylib, nous avons pu créer une interface graphique et dynamique, facilitant l'interaction du joueur avec le labyrinthe. Le jeu propose une interface avec un menu de démarrage, une sélection de difficulté et un système de génération de labyrinthes dynamique qui s'adapte au niveau de difficulté choisi intègre aussi des mécaniques de jeu essentielles telles que le mouvement du joueur, la détection de collisions et le système de score, offrant une expérience engageante pour les utilisateurs. L'inclusion d'une fonctionnalité de pause et la possibilité de redémarrer le jeu enrichissent encore l'expérience utilisateur, garantissant que les joueurs peuvent profiter du jeu à leur propre rythme.

En conclusion, ce projet a été une occasion précieuse de mettre en pratique des compétences en développement de jeux, tout en nous confrontant à des défis techniques qui ont enrichi notre apprentissage.