Faculté des Sciences et Techniques de Tanger

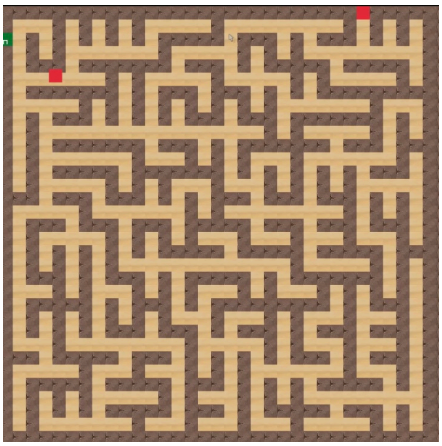
Rapport :

Encadré par :

- Ikram Ben Abdel ouahab

Réalisé par :

- Imad Boulyakine
- Siham Amirech
- Yasmine Zahnoun



Introduction au Code

Le code est un jeu développé avec raylib, une bibliothèque C pour la création de jeux. Le jeu centré sur un personnage qui doit naviguer dans un labyrinthe généré de manière procédurale. Il y a plusieurs éléments dans le jeu :

- **Le labyrinthe généré aléatoirement.**
- **Des obstacles comme des murs et des zones d'eau.**
- **Un personnage joueur qui interagit avec l'environnement.**
- **Des ennemis qui poursuivent le joueur.**
- **Un système de scores et de sauvegarde des meilleurs temps.**
- **Des textures et des sons pour améliorer l'expérience.**

1. Structure du Code

Le code est structuré autour de plusieurs classes et fonctions principales. Voici une description des principaux composants du jeu :

a. Chargement des Ressources

- **Textures : Le jeu charge plusieurs textures pour différents éléments de l'environnement (sable, roches, vers, et joueur) à l'aide de la fonction LoadTextures(). Cela inclut également des textures spécifiques pour les murs et l'eau.**
- **Audio : Le jeu inclut un fichier musical de fond et un son pour la victoire, chargés via les fonctions LoadAudio() et déchargés avec UnloadAudio().**
- **Shader : Un shader est utilisé, mais il n'est pas clairement intégré dans le gameplay. Il est potentiellement destiné à des effets spéciaux (glow) non encore complètement détaillés dans ce code.**

b. Génération de Labyrinthe

La classe Maze est responsable de la génération du labyrinthe, en utilisant un algorithme de backtracking. La fonction Generate() crée le labyrinthe en supprimant les murs entre les cellules pour créer un chemin valide. Des cellules d'eau sont aussi générées aléatoirement dans le labyrinthe. Le labyrinthe est ensuite affiché sur un RenderTexture2D, ce qui permet un rendu optimisé du jeu.

- **Cellules de Labyrinthe : Chaque cellule est représentée par la classe Cell, qui peut être un mur, de l'eau, ou un espace libre. La fonction render() de cette classe dessine chaque cellule à l'écran en fonction de son état.**

c. Gestion du Joueur

Le joueur est représenté par la classe Player, qui gère la position, le mouvement, et l'état de la soif (ressource). Les mouvements du joueur sont traités par les fonctions moveUp(), moveDown(), moveLeft(), et moveRight(). Si le joueur entre dans une cellule d'eau, il peut se désaltérer, ce qui augmente son niveau de soif. La classe Player gère également les collisions avec les murs du labyrinthe.

- **Thirst : Le joueur a une ressource de soif, et lorsqu'il boit de l'eau, son niveau de soif augmente. Si la soif atteint un niveau critique, le joueur pourrait être en danger (bien que ce mécanisme ne semble pas encore complet dans ce code).**

d. Ennemi (Worm)

La classe Worm représente un ennemi qui poursuit le joueur. Le ver suit le joueur en détectant sa position grâce à un rayon de vision (visionRadius). Le ver se déplace vers le joueur si celui-ci est à portée. La fonction moveTowardsPlayer() ajuste la position du ver en fonction de la position du joueur.

e. Gestion des Scores

Les scores sont stockés dans un fichier texte (highscores.txt). Le joueur peut ajouter un score à la liste des meilleurs scores, et ces scores sont affichés avec la fonction ShowHighScores(). Cela inclut l'enregistrement du nom du joueur et du temps qu'il a mis pour finir le jeu.

2. Mécanismes de Jeu

Le jeu repose sur plusieurs mécanismes de gameplay intéressants :

a. Naviguer dans le Labyrinthe

Le joueur doit naviguer dans un labyrinthe généré aléatoirement en évitant les murs. Le joueur commence à une position aléatoire sur le labyrinthe et doit atteindre la sortie. Le labyrinthe est généré à chaque partie avec un chemin unique entre le point de départ et d'arrivée.

b. Gestion de la Soif

Une caractéristique importante du jeu est la gestion de la soif du joueur. L'eau est représentée par des cellules spécifiques dans le labyrinthe. Chaque fois que le joueur entre dans une cellule d'eau, son niveau de soif augmente, et l'eau est supprimée de la cellule. Cela ajoute une dimension de survie, car le joueur doit gérer sa soif en plus de sa navigation.

c. Interaction avec les Ennemis

Les ennemis (vers) se déplacent de manière simple en suivant le joueur lorsqu'ils sont à portée. Cela représente un danger constant pour le joueur, qui doit éviter d'être attrapé tout en essayant d'atteindre la sortie du labyrinthe.

d. Système de Scores

Le jeu propose un système de scores où le temps du joueur est enregistré et comparé aux meilleurs scores. Ces scores sont stockés dans un fichier texte et peuvent être affichés à l'écran.

3. Points Forts

- **Génération aléatoire du labyrinthe : L'algorithme utilisé permet de créer des labyrinthes uniques à chaque partie.**
- **Mécanisme de soif : Le système de soif est un ajout intéressant qui oblige le joueur à gérer ses ressources en plus de simplement naviguer dans le labyrinthe.**
- **Gestion des ennemis : Le système de poursuite des vers ajoute un élément de tension au gameplay, rendant chaque déplacement plus risqué.**

Conclusion

Ce code présente une base solide pour un jeu de labyrinthe avec des mécanismes de survie intéressants. La gestion de la soif et des ennemis, ainsi que la génération aléatoire des labyrinthes, apportent de la diversité et de la tension au gameplay.