RAPPORT DE PROJET

Projet préparé : CLERVIL Adonaï

DIALLO Thierno Djenabou

Encadré : Ikram Ben Abdel Ouahab

Décembre 2024

1. **Introduction**

Le projet a pour objectif de développer un jeu de casse-tête de type labyrinthe en utilisant le paradigme orienté objet en C++ et la bibliothèque Raylib. Le jeu propose une expérience immersive où le joueur devra naviguer dans le labyrinthe qui lui sera généré aléatoirement à chaque partie. Le jeu aura trois niveaux de difficulté (facile, moyen, difficile) qui affectera la taille et la complexité de la navigation.

2. **Présentation théorique**
    a. **Définition**

Un labyrinthe est un tracé sinueux, muni ou non d'embranchements, d'impasses et de fausses pistes, destiné à égarer ou à ralentir celui qui cherche à se déplacer. Dans ce projet, le joueur est immergé dans le labyrinthe et cherche à retrouver le chemin.

    b. **Algorithme utilisé**

L'algorithme utilisé est l'algorithme de Recherche en Profondeur, *Depth-First Search* (DFS) en anglais. Il consiste, à partir d'une cellule initiale, à visiter aléatoirement les cellules voisines en les ajoutant dans une pile. Cette pile sera ensuite consultée lorsque l'algorithme rencontre une impasse, ce qui permet de visiter toutes les cellules. Cet algorithme respecte les règles de connexité, ce qui assure l'existence d'une solution. De plus, puisque chaque cellule est visitée une seule fois, cela garantit l'unicité de la solution.

3. **Conception et Développement**
    a. **Objectifs de la conception**

Le projet était de concevoir un jeu de labyrinthe en C++. Les principaux objectifs sont les suivants :

1. Génération automatique de labyrinthes
2. Ajouter trois Niveaux de Difficulté
3. Avoir une interface Graphique en utilisant la bibliothèque Raylib
4. Gestion des options de jeux

    b. **Structure du projet**

Le jeu est structuré autour de plusieurs classes clés :

1. Labyrinthe :
    • Représente la grille du labyrinthe et contient la méthode de génération aléatoire.
    • Propriétés : lignes, colonnes, vecteur de type "cell".
    • Méthodes : génération procédurale, affichage.
2. Joueur :
    • Représente le personnage contrôlé par le joueur.
    • Propriétés : position x, position y, déplacements.
    • Méthodes : déplacement
3. Gestionnaire de Jeu :
    • Gestion de la boucle principale du jeu.
    • Propriétés : état du jeu, chronomètre, interface utilisateur (menu, fin de partie).
    • Méthodes : gestion des événements, démarrage et réinitialisation des parties.

Développement

a. **Génération du labyrinthe**

Comme mentionné l'algorithme utilisé est l'algorithme de Recherche en Profondeur dont voici ses étapes :

1. Initialisation

Pour faciliter la manipulation, nous avons choisi d'utiliser un vecteur de type "cell", qui est une structure représentant les cellules. Ce vecteur stocke toutes les cellules, et le nombre de cellules sera déterminé en fonction du nombre de colonnes et de lignes. Afin de maintenir la cohérence, étant donné qu'une structure de deux dimensions est stockée dans un vecteur à une seule dimension, une méthode appelée "index_cellule()" a été déclarée. Cette méthode permet de retourner l'index de la cellule dont les coordonnées sont passées en argument.

2. Parcours récursif
   Partant d'une cellule passée en argument, la méthode "trouver_les_cellules_non_visitees()" cherche de manière aléatoire les cellules voisines non visitées. Elle retourne la cellule trouvée, qui sera ensuite stockée dans une pile après l'appel de la méthode dans "initialisation_du_labyrinthe()". Par la suite, les murs correspondant à la direction des cellules visitées sont supprimés, puis on fait appel à la méthode "tracer_du_labyrinthe()" qui représente visuellement cette modification en traçant les murs non cassés.

3. Une fois cela fait, nous traçons les murs extérieurs qui bordent le labyrinthe, ce qui complète l'affichage du labyrinthe.

b. **Gestionnaire du joueur**

1. Initialisation
   Toutes les ressources liées au joueur sont chargées dans le constructeur de la classe Joueur, et sa position est initialisée à la première cellule.

2. Déplacement
   Les déplacements du joueur sont contrôlés par les touches de direction du clavier. Le joueur ne peut se déplacer que s'il n'y a pas de mur dans la direction choisie, afin de maintenir la cohérence et de gérer les collisions.

3. Détection de victoire
   Lorsque le joueur atteint la dernière cellule (celle en bas à droite), la méthode "gagner()" de la classe retourne "true".

c. **Gestionnaire de Jeux**

La classe **"GestionnaireDeJeux"** est responsable de l'affichage des options, des menus, des paramètres, ainsi que de toute la logique du jeu. Elle gère les interactions entre les différents composants du programme et l'état actuel du jeu. À cet effet, elle contiendra plusieurs méthodes.

Défis techniques

Durant le développement de ce jeu, j'ai rencontré plusieurs défis techniques liés à la bibliothèque utilisée, qui ne supporte pas nativement certaines fonctionnalités. Voici une analyse de ces défis et des solutions que j'ai mises en place pour les surmonter.

1. Agrandir la fenêtre
   Étant donné que j'utilise des images pour représenter le joueur, je devais m'assurer que la taille de chaque cellule correspondait à celle du joueur, afin d'éviter que la taille de la fenêtre soit insuffisante.

Solution : Pour garantir que toutes les cellules soient affichées, peu importe la taille du labyrinthe, j'ai déclaré une fonction qui agrandira la taille de la fenêtre au début du jeu, car Raylib ne prend pas en charge cette fonctionnalité nativement.

2. Simulation de bouton
   Tout au long du jeu, le joueur devra effectuer des clics, et ceux-ci devront être gérés en fonction de leur contexte. Malheureusement, Raylib ne contient pas d'utilitaires, c'est-à-dire d'objets qui joueraient le rôle de boutons et qui déclencheraient un événement que l'on pourrait traiter.

   Solution : Pour pouvoir gérer les clics du joueur, nous avons dû simuler des boutons en entourant la zone utile par un rectangle. Ensuite, à chaque clic du joueur, nous vérifions la zone où il a cliqué. Si c'est dans le rectangle, nous appelons la fonction qui gère ce cas.

3. Simulation de vidéo
   Pour l'animation du début, je devais utiliser une vidéo, au-dessus de laquelle j'allais animer mon texte. Cependant, Raylib ne prend pas en charge la lecture de vidéos.

   Solution : Pour pouvoir utiliser la vidéo, je l'ai convertie en une image animée de type GIF, qui est prise en charge par Raylib. Ensuite, j'ai géré l'animation, les frames et les délais de manière à simuler une vidéo.