كلية العلوم و التقنيات بطنجة
ⵜⴰⵖⵡⵍⵉⵜ ⵏ ⵜⵓⵙⵙⵔⵉⵜ ⴰ ⵜⵓⵣⵔⴼⵜⵉ ⵏ ⵟⴰⵏⵊⴰ
Faculté des Sciences et Techniques de Tanger

جامعة عبد المالك السعدي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵄⴰⴱⴷⵍⵎⴰⵍⵉⴽ ⵙⵙⴰⵄⴷⵉ
Université Abdelmalek Essaâdi

MST : Sécurité It et Big Data

Algorithmique avancée et programmation

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

Réalisé par : Siham ELHISSANI

Marouane CHERKAOUI

Encadré par : Pr Ikram BENABDELOUAHAB

1. Introduction

Ce projet consiste à développer un jeu de labyrinthe généré de manière procédurale. L'objectif est de permettre au joueur de se déplacer dans un labyrinthe pour atteindre la sortie tout en respectant un chronomètre. Ce jeu a été réalisé dans le cadre de l'apprentissage de la programmation orientée objet en C++.

L'objectif principal est d'appliquer les concepts de programmation orientée objet (POO) en créant un jeu interactif avec plusieurs niveaux de difficulté. Le défi réside dans la génération dynamique de labyrinthes, l'optimisation des performances et la gestion des interactions avec le joueur.

2. Outils et technologies utilisées

- **Langage de programmation :** C++
- **Bibliothèques utilisées :** Raylib (pour le rendu graphique 2D)
- **Environnement de développement :** Visual Studio Code

3. Etude technique

3.1. Classes clés

3.1.1. Labyrinthe :

La classe Labyrinthe représente un labyrinthe généré procéduralement pour un jeu. Elle utilise des structures de données et des algorithmes pour créer une grille 2D qui définit le labyrinthe avec des murs, des passages, et une sortie.

Voici le code (fichier Labyrinthe.h) :

MST : Sécurité It et Big Data

Algorithmique avancée et programmation

Labyrinthe.h

Projet C++ Raylib > src > Labyrinthe.h > ...

```h
#ifndef LABYRINTHE_H
#define LABYRINTHE_H

#include <vector>
#include <stack>
#include <cstdlib>
#include <ctime>
#include <utility>

class Labyrinthe {
public:
    int largeur, hauteur;
    std::vector<std::vector<int>> grille;
    int finishX, finishY;

    Labyr  void Labyrinthe::genererLabyrinthe()
    void genererLabyrinthe();
    void setFinishPoint(int x, int y);
};

#endif // LABYRINTHE_H
```

Voici le code (fichier Labyrinthe.cpp) :

Labyrinthe.cpp

Projet C++ Raylib > src > Labyrinthe.cpp > ...

```cpp
#include "Labyrinthe.h"

Labyrinthe::Labyrinthe(int l, int h) : largeur(l), hauteur(h) {
    srand(time(0));
    grille.resize(hauteur, std::vector<int>(largeur, 1));
    genererLabyrinthe();
}

void Labyrinthe::genererLabyrinthe() {
    int x = 1, y = 1;
    grille[y][x] = 0;

    std::stack<std::pair<int, int>> pile;
    pile.push({x, y});

    std::vector<std::pair<int, int>> directions = {{2, 0}, {0, 2}, {-2, 0}, {0, -2}};
```

```cpp
    while (!pile.empty()) {
        auto current = pile.top();
        int cx = current.first;
        int cy = current.second;
        std::vector<std::pair<int, int>> voisins;

        for (auto& dir : directions) {
            int nx = cx + dir.first;
            int ny = cy + dir.second;
            if (nx > 0 && ny > 0 && nx < largeur - 1 && ny < hauteur - 1 && grille[ny][nx] == 1) {
                voisins.push_back({nx, ny});
            }
        }

        if (!voisins.empty()) {
            auto next = voisins[rand() % voisins.size()];
            int nx = next.first;
            int ny = next.second;
            grille[cy + (ny - cy) / 2][cx + (nx - cx) / 2] = 0;
            grille[ny][nx] = 0;
            pile.push(next);
        } else {
            pile.pop();
        }
    }

    setFinishPoint(largeur - 2, hauteur - 2);
}
```

```
void Labyrinthe::setFinishPoint(int x, int y) {
    finishX = x;
    finishY = y;
    grille[y][x] = 2;
}
```

Fonctionnement :

La classe Labyrinthe représente un labyrinthe généré procéduralement. Elle utilise une grille 2D pour modéliser les murs, les passages, et la sortie. Chaque cellule de cette grille est représentée par un entier, où les murs sont marqués par 1, les passages libres par 0, et la sortie par 2. Les dimensions du labyrinthe sont définies par les attributs largeur et hauteur, tandis que les coordonnées de la sortie sont stockées dans **finishX** et **finishY.**

Lorsqu'un objet Labyrinthe est créé, la grille est initialisée avec des murs partout, et la méthode **genererLabyrinthe** est appelée pour générer un labyrinthe aléatoire en utilisant un algorithme de **backtracking**. Cet algorithme part d'une cellule de départ et creuse des passages en avançant vers des voisins disponibles, tout en supprimant les murs entre les cellules. Une pile est utilisée pour revenir en arrière lorsqu'aucune avancée n'est possible, garantissant ainsi que le labyrinthe est entièrement connecté. La méthode **setFinishPoint** permet de définir la sortie du labyrinthe à une position spécifique, généralement proche de l'extrémité opposée à l'entrée.

3.1.2. Joueur :

La classe Joueur représente un personnage contrôlé par le joueur dans un jeu de labyrinthe. Elle gère la position du joueur dans une grille 2D, permet les déplacements en fonction des entrées clavier et des contraintes du labyrinthe, et affiche le personnage à l'écran.

Voici le code (fichier Joueur.h) :

Joueur.h

```
src > C Joueur.h
#ifndef JOUEUR_H
#define JOUEUR_H

#include "Labyrinthe.h"
#include "game.h"

class Joueur {
public:
    Joueur(int startX, int startY, float speed) : x(startX), y(startY), speed(speed) {}
    int x, y;
    float speed;
    static constexpr float initSpeed = 1.0f;
    Joueur(int startX, int startY) : x(startX), y(startY), speed(initSpeed) {}
    GameScreen Update(Labyrinthe& labyrinthe);
    void deplacer(char direction, Labyrinthe& labyrinthe);
void Draw(float mazeOffsetX, float mazeOffsetY);
};
```

Voici le code (fichier Joueur.cpp) :

```cpp
// Joueur.cpp
#include "raylib.h"
#include "Joueur.h"

GameScreen Joueur::Update(Labyrinthe& labyrinthe) {
    const int speedMultiplier = 2;


    if (IsKeyDown(KEY_RIGHT)) {
        for (int i = 0; i < speedMultiplier; i++) {
            deplacer('R', labyrinthe);
        }
    }
    if (IsKeyDown(KEY_LEFT)) {
        for (int i = 0; i < speedMultiplier; i++) {
            deplacer('L', labyrinthe);
        }
    }
    if (IsKeyDown(KEY_DOWN)) {
        for (int i = 0; i < speedMultiplier; i++) {
            deplacer('D', labyrinthe);
        }
    }
    if (IsKeyDown(KEY_UP)) {
        for (int i = 0; i < speedMultiplier; i++) {
            deplacer('U', labyrinthe);
        }
    }
    return SOLOCONTROLS;
}
```

```cpp
void Joueur::deplacer(char direction, Labyrinthe& labyrinthe) {
    if (direction == 'R' && labyrinthe.grille[y][x + 1] == 0) x++;
    if (direction == 'L' && labyrinthe.grille[y][x - 1] == 0) x--;
    if (direction == 'D' && labyrinthe.grille[y + 1][x] == 0) y++;
    if (direction == 'U' && labyrinthe.grille[y - 1][x] == 0) y--;
}


void Joueur::Draw(float mazeOffsetX, float mazeOffsetY) {
    const int cellSize = 20;
    const Color playerColor = BLUE;
    DrawRectangle(x * cellSize + mazeOffsetX, y * cellSize + mazeOffsetY, cellSize, cellSize, playerColor);
}
```

Fonctionnement :

La classe Joueur représente le personnage contrôlé par le joueur dans le labyrinthe. Elle gère sa position actuelle, définie par les attributs x et y, ainsi que sa vitesse de déplacement, stockée dans l'attribut **speed**. Deux constructeurs permettent d'initialiser ces valeurs, avec la possibilité d'utiliser une vitesse par défaut.

Le joueur interagit avec le labyrinthe grâce à la méthode **Update**, qui détecte les touches directionnelles appuyées par l'utilisateur et ajuste la position en conséquence. Les déplacements sont gérés par la méthode **deplacer**, qui vérifie si la case cible dans le labyrinthe est libre avant de permettre le mouvement. Enfin, la méthode **Draw** affiche le joueur à l'écran en dessinant un rectangle coloré à sa position actuelle, rendant son emplacement visible dans l'interface graphique.

3.1.3. Niveau (Difficulty)

La classe Difficulty représente l'écran de sélection de la difficulté dans un jeu. Elle permet au joueur de choisir entre différents niveaux de difficulté (facile, moyen, difficile) ou de revenir au menu principal. L'interface est composée de boutons interactifs affichés à l'écran, et les choix du joueur sont détectés via des clics de souris. Cette classe gère l'affichage des boutons, leur positionnement, et la logique pour naviguer vers d'autres parties du jeu en fonction de la sélection effectuée.

MST : Sécurité It et Big Data

Algorithmique avancée et programmation

Voici le code (fichier Difficulty.h) :

```h
src > C difficulty.h > ...
#ifndef DIFFICULTY_H
#include "raylib.h"
#include "game.h"


class Difficulty {
public:
    Difficulty();
    ~Difficulty();

    GameScreen Update();
    void Draw();

private:
    Rectangle easyButton;
    Rectangle mediumButton;
    Rectangle hardButton;
    Rectangle returnButton;

    Texture2D easyTexture;
    Texture2D mediumTexture;
    Texture2D hardTexture;
    Texture2D returnTexture;
};

#endif // DIFFICULTY_H
```

Voici le code (fichier Difficulty.cpp) :

```cpp
src > difficulty.cpp > Difficulty()
Difficulty::Difficulty() {
    easyTexture = LoadTexture("buttons/Hover.png");
    mediumTexture = LoadTexture("buttons/Hover.png");
    hardTexture = LoadTexture("buttons/Hover.png");
    returnTexture = LoadTexture("buttons/Hover.png");


    float buttonWidth = easyTexture.width;
    float buttonHeight = easyTexture.height;
    float spacing = 20;

    easyButton = { (GetScreenWidth() - buttonWidth) / 2, GetScreenHeight() - (buttonHeight * 4 + spacing * 3) - 100, buttonWidth, buttonHeight };
    mediumButton = { (GetScreenWidth() - buttonWidth) / 2, easyButton.y + buttonHeight + spacing, buttonWidth, buttonHeight };
    hardButton = { (GetScreenWidth() - buttonWidth) / 2, mediumButton.y + buttonHeight + spacing, buttonWidth, buttonHeight };
    returnButton = { (GetScreenWidth() - buttonWidth) / 2, hardButton.y + buttonHeight + spacing, buttonWidth, buttonHeight };
}

Difficulty::~Difficulty() {
    UnloadTexture(easyTexture);
    UnloadTexture(mediumTexture);
    UnloadTexture(hardTexture);
    UnloadTexture(returnTexture);
}
```

```cpp
GameScreen Difficulty::Update() {
    if (IsMouseButtonPressed(MOUSE_BUTTON_LEFT)) {
        Vector2 mousePos = GetMousePosition();

        if (CheckCollisionPointRec(mousePos, easyButton)) return EASYSOLOMODE;
        if (CheckCollisionPointRec(mousePos, mediumButton)) return MEDIUMSOLOMODE;
        if (CheckCollisionPointRec(mousePos, hardButton)) return HARDSOLOMODE;
        if (CheckCollisionPointRec(mousePos, returnButton)) return MENU;
    }

    return DIFFICULTY;
}

void Difficulty::Draw() {
    DrawTextureRec(easyTexture, { 0, 0, (float)easyTexture.width, (float)easyTexture.height }, { easyButton.x, easyButton.y }, WHITE);
    DrawTextureRec(mediumTexture, { 0, 0, (float)mediumTexture.width, (float)mediumTexture.height }, { mediumButton.x, mediumButton.y }, WHITE);
    DrawTextureRec(hardTexture, { 0, 0, (float)hardTexture.width, (float)hardTexture.height }, { hardButton.x, hardButton.y }, WHITE);
    DrawTextureRec(returnTexture, { 0, 0, (float)returnTexture.width, (float)returnTexture.height }, { returnButton.x, returnButton.y }, WHITE);
}
```

Fonctionnement :

La classe Difficulty gère l'écran de sélection de la difficulté dans un jeu. Elle propose quatre options interactives sous forme de boutons : **Easy**, **Medium**, **Hard**, et **Return**. Ces options permettent au joueur de choisir une difficulté ou de revenir au menu principal.

Le constructeur initialise les textures associées à chaque bouton et calcule leurs positions en fonction des dimensions de l'écran. Chaque bouton est représenté par un rectangle (Rectangle) dont les coordonnées et dimensions sont configurées pour un alignement centré verticalement et horizontalement avec un espacement constant entre eux.

Le destructeur de la classe libère les ressources des textures chargées en mémoire pour éviter les fuites de mémoire.

La méthode Update détecte les clics de souris sur les boutons. Lorsqu'un bouton est cliqué, elle retourne une valeur spécifique (**EASYSOLOMODE, MEDIUMSOLOMODE, HARDSOLOMODE**, ou **MENU**) correspondant à l'action ou au mode de jeu sélectionné. En l'absence d'interaction, la méthode retourne un état neutre (**DIFFICULTY**).

Enfin, la méthode **Draw** affiche les boutons à l'écran en dessinant leurs textures à leurs positions respectives. Cela permet de représenter visuellement les options disponibles dans l'interface utilisateur.

3.1.4. Jeu (Main)

La classe Jeu représente le cœur d'un jeu, où l'ensemble du jeu est orchestré. Elle gère l'affichage de l'interface, la gestion des écrans (menu principal, paramètres, difficulté, modes de jeu), ainsi que la gestion des ressources multimédia comme la musique et les arrière-plans. Le programme commence en initialisant la fenêtre du jeu, les sons et la musique, puis il passe à la gestion des différents écrans et interactions avec le joueur. Le jeu se déroule dans une boucle principale où l'état actuel du jeu est constamment mis à jour, les actions de l'utilisateur sont traitées, et l'écran est redessiné en fonction de l'état du jeu. Enfin, lorsque le jeu est terminé, les ressources sont libérées et la fenêtre est fermée.

Voici le code du main.cpp :

MST : Sécurité It et Big Data

Algorithmique avancée et programmation

src > main.cpp > ...

```cpp
#include "raylib.h"
#include "menu.h"
#include "settings.h"
#include "difficulty.h"
#include "solocontrols.h"
#include "easysolomode.h"
#include "mediumsolomode.h"
#include "hardsolomode.h"
#include "volume.h"
#include "resolution.h"

int main() {

    const int screenWidth = 1920;
    const int screenHeight = 1080;
    InitWindow(screenWidth, screenHeight, "Maze");
    InitAudioDevice();
    SetTargetFPS(240);

    Texture2D background = LoadTexture("images/Background.jpg");
    if (background.width == 0 || background.height == 0) {
        return -1;
    }

    Music music = LoadMusicStream("Sounds/music.mp3");
    if (music.stream.buffer == nullptr) {
        UnloadTexture(background);
        CloseAudioDevice();
        CloseWindow();
        return -1;
    }
    PlayMusicStream(music);
    SetMusicVolume(music, 0.5f);

    Font customFont = LoadFont("Font/monogram.ttf");

    GameScreen currentScreen = MENU;
    Menu menu;
    Settings settings;
    Difficulty difficulty;
    Solocontrols solocontrols;
    Easysolomode easysolomode;
    Mediumsolomode mediumsolomode;
    Hardsolomode hardsolomode;
    Volume volume;
    Resolution resolution;

    while (!WindowShouldClose()) {


        UpdateMusicStream(music);

        if (GetMusicTimePlayed(music) >= GetMusicTimeLength(music)) {
            StopMusicStream(music);
            PlayMusicStream(music);
        }

        switch (currentScreen) {
            case MENU:
                currentScreen = menu.Update();
                break;
            case SETTINGS:
                currentScreen = settings.Update();
                break;
            case DIFFICULTY:
                currentScreen = difficulty.Update();
                break;
            case SOLOCONTROLS:
                currentScreen = solocontrols.Update();
                break;
            case EASYSOLOMODE:
                currentScreen = easysolomode.Update();
                break;
            case MEDIUMSOLOMODE:
                currentScreen = mediumsolomode.Update();
                break;
            case HARDSOLOMODE:
                currentScreen = hardsolomode.Update();
                break;
            case VOLUME:
                currentScreen = volume.Update(music);
                break;
            case RESOLUTION:
                currentScreen = resolution.Update();
                break;
        }
```

```
        BeginDrawing();
        ClearBackground(RAYWHITE);
        DrawTexture(background, 0, 0, WHITE);
        switch (currentScreen) {
            case MENU:
                menu.Draw();
                break;
            case SETTINGS:
                settings.Draw();
                break;
            case DIFFICULTY:
                difficulty.Draw();
                break;
            case SOLOCONTROLS:
                solocontrols.Draw();
                break;
            case EASYSOLOMODE:
                easysolomode.Draw();
                break;
            case MEDIUMSOLOMODE:
                mediumsolomode.Draw();
                break;
            case HARDSOLOMODE:
                hardsolomode.Draw();
                break;
            case VOLUME:
                volume.Draw();
                break;
            case RESOLUTION:
                resolution.Draw();
                break;
        }

        EndDrawing();
    }
    UnloadTexture(background);
    UnloadMusicStream(music);
    UnloadFont(customFont);
    CloseWindow();
    return 0;
}
```

4. Conclusion

Le programme principal du jeu gère l'ensemble du flux de jeu, en permettant une interaction fluide entre différents écrans (menu, paramètres, modes de jeu, etc.) et en assurant la gestion continue des ressources multimédia comme la musique et les images. À travers une boucle principale, il gère l'état du jeu, met à jour l'affichage en temps réel, et répond aux actions de l'utilisateur. Cette structure modulaire permet de séparer clairement les différentes fonctionnalités du jeu, facilitant ainsi l'extension et la maintenance du code. Finalement, lorsque le joueur termine sa session, le programme se ferme proprement en libérant toutes les ressources utilisées.

MST : Sécurité It et Big Data

Algorithmique avancée et programmation