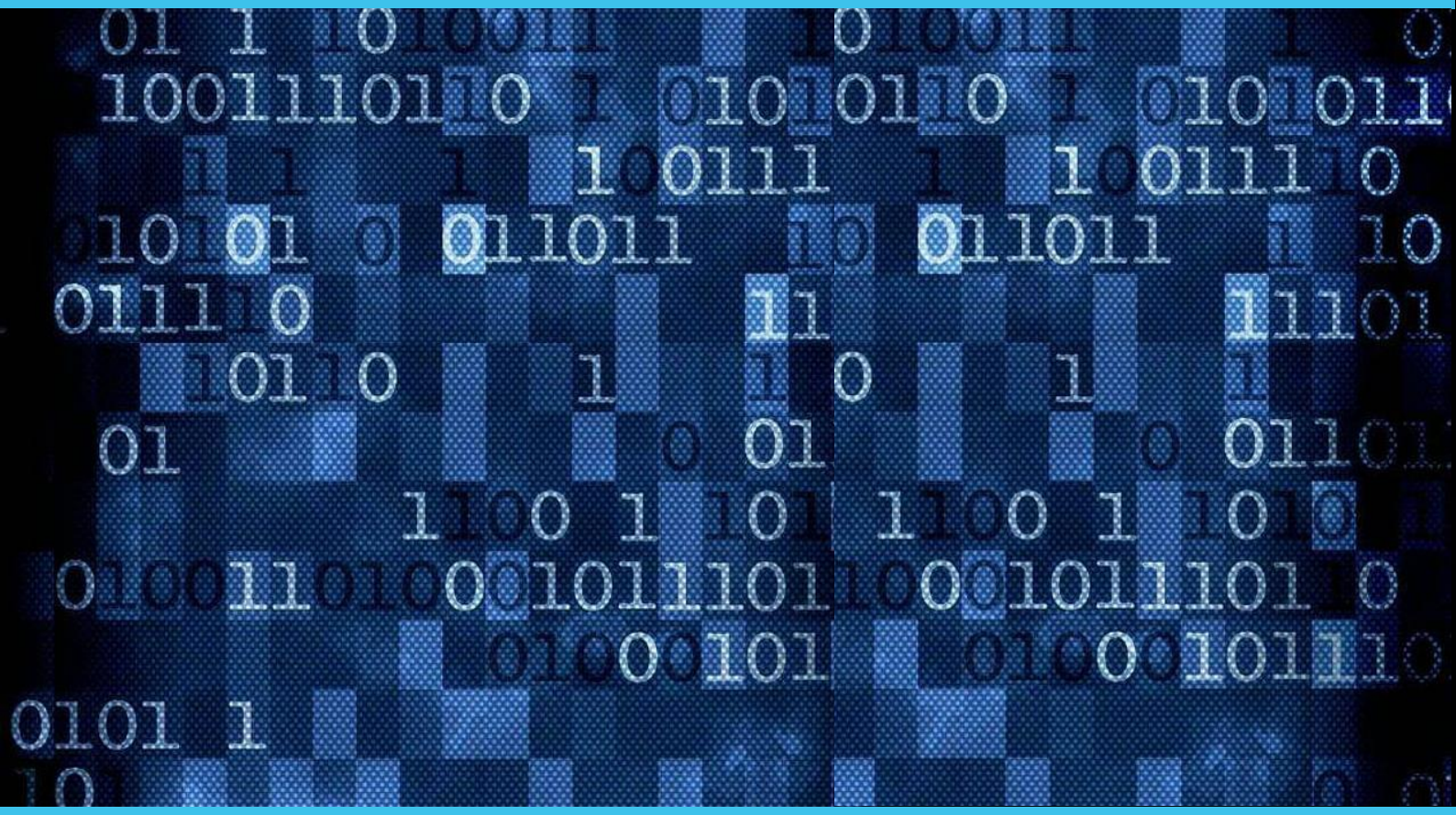
Livrable 2

Projet : Le Jeu De La Vie



Rayane OULDALI et Thaïs VAINES

Sommaire

Préambule

Rappel du contexte

Au cours de ce projet nous devons implémenter un algorithme du jeu de la vie.

Le jeu de la vie est un automate cellulaire (Le joueur n'intervient pas dans le jeu mais celui-ci est coder de manière à suivre son cours jusqu'à l'extinction du programme). Ce jeu issu d'un modèle mathématique proposé par John Conway en 1970 représente des cellules, blanches si elles sont vivantes et noires si elles sont mortes. A chaque itération du programme, les cellules sont analysées afin de savoir si elles doivent vivre, mourir ou rester dans leur état actuel.

Rappel des attentes du client

Règles du jeu :

- Une cellule morte possédant exactement trois cellules voisines vivantes devient vivante (elle naît) ;
- Une cellule vivante ne possédant pas exactement deux ou trois cellules voisines vivantes meurt.

Objectifs du livrable 2

Souhaite désormais l'implémentation du jeu de la vie

Nous devons alors l'implémenter en suivant la logique des diagrammes de classe tout en prenant des libertés si nécessaire afin de produire la version console du jeu de la vie.

Nous devons implémenter une interface graphique.

Nous devons faire des tests unitaires afin de pouvoir valider notre projet.

Rappel des diagrammes

Tous ces diagrammes sont des diagrammes type UML. Les digrammes UML permette de schématiser de manière simple et sous plusieurs formes les différentes fonctionnalité de notre code.

Diagramme Use Case

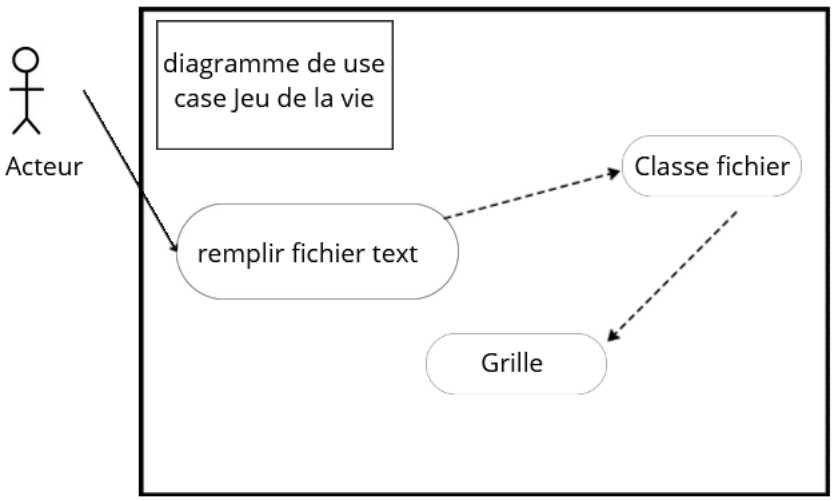


Diagramme de Séquence

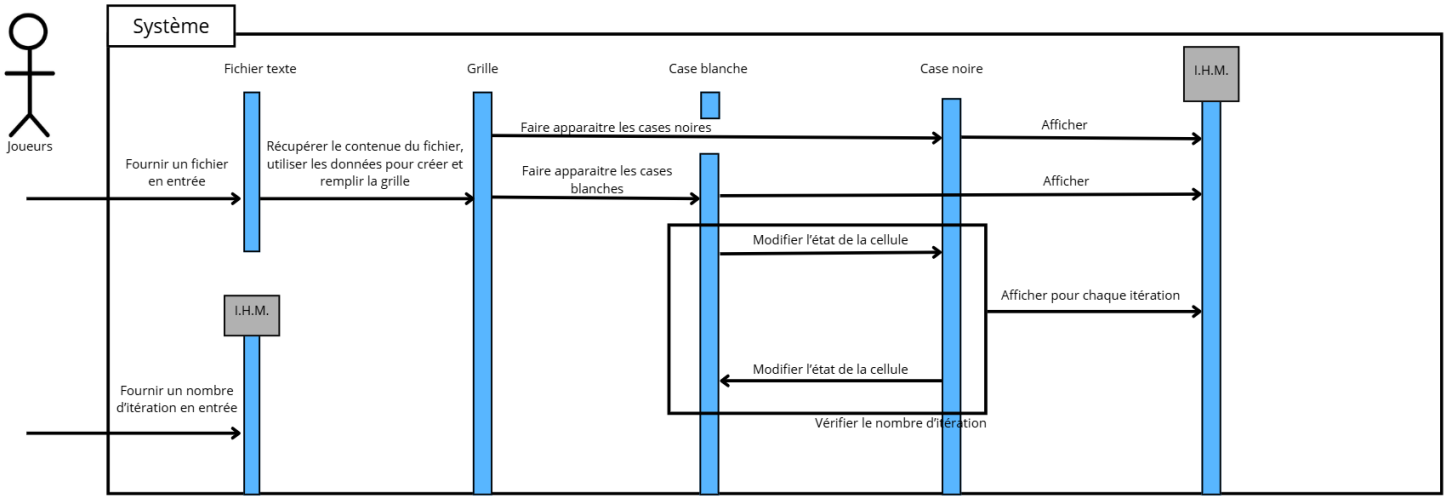


Diagramme d’Activité

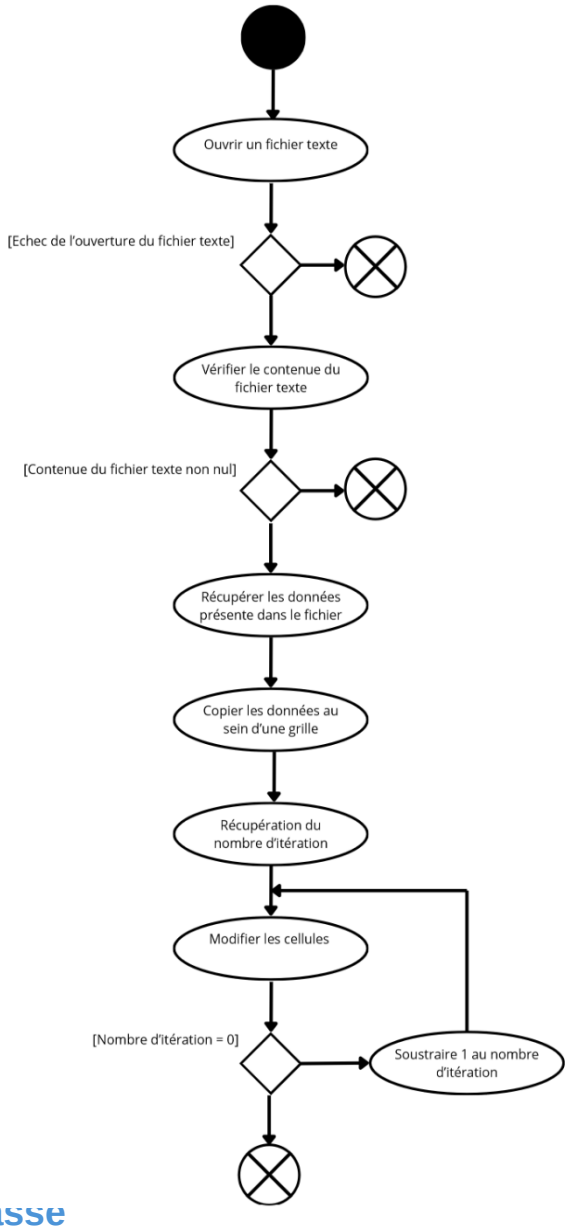
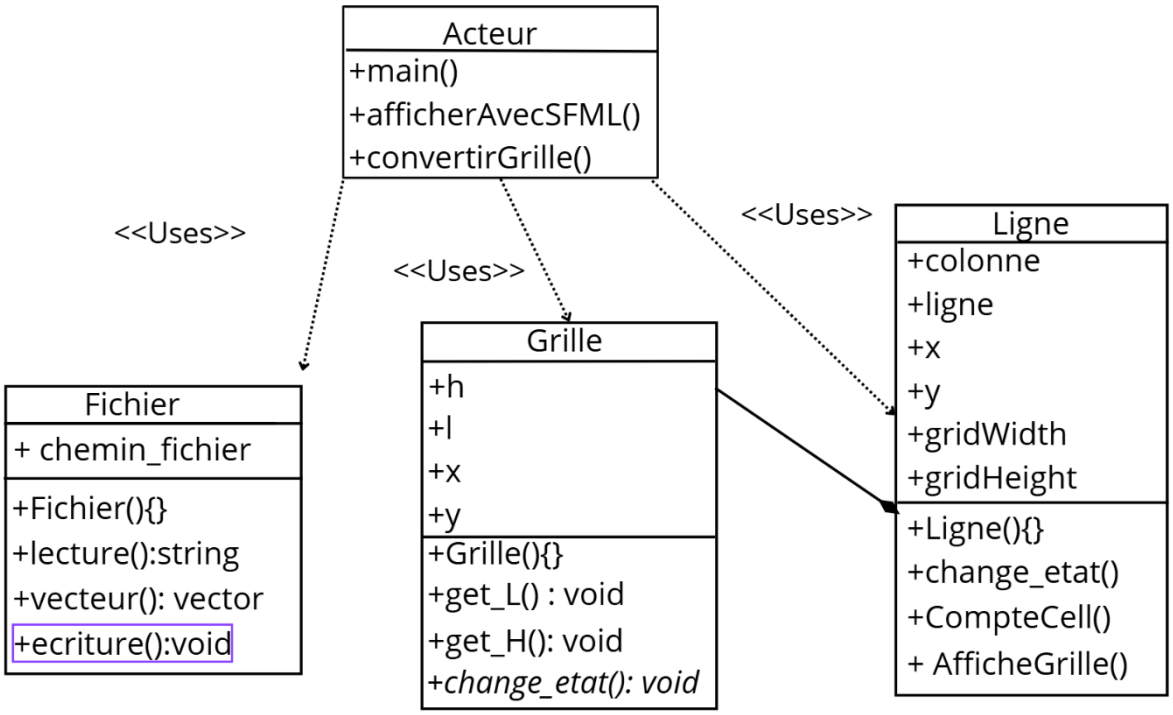


Diagramme de classe

Nous avons réalisé des ajustements sur le diagramme de classes afin de contourner les problématiques techniques que nous avons rencontré. Nous avons ajouté 2 méthodes dans le main afin de faire l’affichage de la grille. Nous avons

également ajouté 2 méthodes dans la classe ligne afin de gérer les modifications nécessaires à apporter sur le vecteur retourné par la classe fichier. Les attributs ont aussi été modifiés afin que les classes puissent fonctionner correctement.



Spécificité du code

Classe Fichier

Dans la classe fichier, on retrouve 3 méthodes :

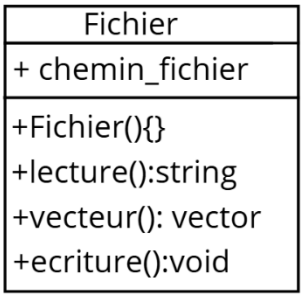
```
class Fichier {

private:
    string chemin_fichier;

public:
    Fichier() {} // constructeur

    string lecture();
    vector<string> vecteur(const string& txt);
    void ecriture(const string& nomFichier, const vector<string>& grille);

};
```



Lecture

Cette méthode qui est de type string a pour rôle l'ouverture d'un fichier texte dans lequel on retrouve sur une première ligne les dimensions de la grille suivi de la grille composée de 0 et de 1. Elle récupère ensuite ligne par ligne le contenue et le stocke dans une grande chaine de caractère.

```
string Fichier::lecture() {
    string full;
    string contenu;
    string const nomFichier(s:"C:/Users/thais/Documents Local/CPI A2/Projet 2/Livrable/Projet-2/Fichier.txt");
    ifstream monFlux(s:nomFichier.c_str());
    while (getline([&]monFlux, [&]contenu)) {
        if(monFlux)  //Teste si l'ouverture a fonctionné
        {
            full +=  contenu + "\n";
        }
        else
            cout << "Impossible d'ouvrir le fichier !" << endl;
    }
    monFlux.close();
    return full;
}
```

Vecteur

Cette méthode qui retourne un vecteur n'a qu'un seul but : séparer la longue chaine de caractère composé du contenue du fichier en un vecteur composée de plusieurs sous vecteurs (matrice)afin que cette matrice soit analysable.

```cpp
vector<string> Fichier::vecteur(const string& txt) {
    vector<string> liste;
    stringstream ss(txt);
    string ligne;
    while (getline([&]ss,[&]ligne)) {
        liste.push_back(ligne);
    }
    return liste;
}
```

Ecriture

Au sein du cahier des charge, on retrouve le stockage des grilles en mode console dans des fichiers textes pour chaque itération. Ainsi, on retrouve dans la classe fichier la méthode Ecriture qui a pour rôle de retranscrire les vecteurs affichés à l'écran lors du mode console dans des fichiers au sein d'un dossier.

```cpp
void Fichier::ecriture(const string& nomFichier, const vector<string>& grille) {
    ofstream fichier(nomFichier);  // Ouvre le fichier en mode écriture
    if (!fichier.is_open()) {
        cerr << "Erreur : Impossible d'ouvrir le fichier " << nomFichier << endl;
        return;
    }

    // Écrire la grille dans le fichier
    for (const auto& ligne :const string& : grille) {
        fichier << ligne << '\n';
    }

    fichier.close();
    cout << "État écrit dans : " << nomFichier << endl;
}
```

Classe Grille

Au sein de cette classe on ne retrouve que les getters. En effet la troisième fonction déclarée était une fonction virtuelle pure elle ne doit donc pas être codée dans cette classe là mais seulement dans la classe enfant. La fonction changement d'état ne peut pas être appliquée sur une grille dans sa totalité mais sur les lignes une par une, de ce fait elle est déclarée dans la fonction grille mais n'est utilisé que dans la classe ligne.

Grille
+h +l +x +y
+Grille(){} +get_L() : void +get_H(): void *+change_etat(): void*

La présence d'une fonction virtuelle pure dans la classe Grille la rend par conséquent abstraite et la classe ne peux pas être instanciée.

Classe Ligne

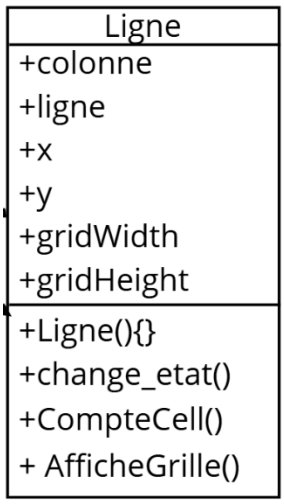
```
class Ligne :public Grille{

public:
    vector<string>grille;
    int colonne;
    int ligne;
    int x;
    int y;
    int gridWidth;
    int gridHeight;

    Ligne(vector<string> grille);// consturcteur

    int get_h() const;
    int get_l()const;

    void change_etat()override;
    int CompteCell(int x,int y,vector<string> grille);
    void AfficheGrille();
};
```



Nous avons ici la déclaration de la classe ligne dans l'entête .h nous voyons les attributs de la classe ainsi que la déclaration des méthodes de la classe Ligne.

```
Ligne::Ligne(vector<string> grille): grille(grille), cologne(grille[0].size()), ligne(grille.size()), x(0), y(0), gridWidth(0), gridHeight(0) {}
```

Le constructeur dans le fichier Ligne.cpp est un constructeur personnalisé qui permet de vérifier les entrées dans les méthodes de la classe

CompteCell

La méthode CompteCell(x,y,grille) ou x et y sont une position dans la grille cette méthode déclarer en tant que Int renvoi le nombre de cellule vivante à proximité d'une autre cellule qui elle est positionner en x,y cette fonction est très utile pour la génération de nouvelle cellule ou la mort de certaine cellule.

Cette méthode est très utile dans la méthode change_etat () .

```cpp
int Ligne::CompteCell(int x,int y,vector<string> grille) {
    int compteur = 0;
    for(int i=-1; i<=1; i++) {
        for(int j=-1; j<=1; j++) {
            if(i == 0 && j == 0) continue;
            int nx = x+i;
            int ny = y+j;
            if (nx >= 0 && nx<ligne && ny>= 0 && ny < colonne) {
                compteur += grille[nx][ny] =='1';
            }
        }
    }
    return compteur;
}
```

Change_etat

En parlant de la méthode change_etat(), la voici. Cette méthode permet d'implémenter la « vie » au sein de la grille. Comme vous le voyer ligne 27 la méthode CompteCell() est utiliser dans la déclaration de la variable voisin .

Le fonctionnement de change état est très simple. On parcourt toute les cellule de la grille à l'aide des deux boucle fort ligne 25,26. A chaque itération on compte le nombre de cellule voisine vivante et on applique alors les conditions du jeu de la vie.

```cpp
void Ligne::change_etat() {
    vector<string> nvlle = grille;

    for (int i = 0; i < ligne; ++i) {
        for (int j = 0; j < colonne; ++j) {
            int voisin = CompteCell(x:i, y:j, grille);
            if (grille[i][j] == '1') {
                nvlle[i][j] =(voisin == 2 || voisin ==  3) ? '1' : '0';
            }else {
                nvlle[i][j] = (voisin ==3) ? '1' : '0';
            }
        }
    }
    grille = nvlle;
}
```

AfficheGrille

La fonction AfficheGrille permet de faire un affichage dans la console de chaque ligne de la grille. Cette fonction est appelée après le changement d'état en mode console seulement.

```
void Ligne::AfficheGrille() {
    for (const auto& l :const string& : grille) {
        cout << l << '\n';
    }
    cout << '\n';
}
```

Main

Le main est le socle du code, on y réalise les appels de méthode, les principales entrées / sorties, ainsi que 2 fonctions qui n'appartiennent à aucune classe et qui réalisent la fenêtre pop-up de l'affichage graphique.

Acteur
+main()
+afficherAvecSFML()
+convertirGrille()

AfficherAvecSFML

```
void afficherAvecSFML(const vector<vector<int>>& grid, int cellSize) {
    if (grid.empty()) return; // Vérifier si la grille est vide
    sf::RenderWindow window(sf::VideoMode( modeWidth: grid[0].size() * cellSize, modeHeight: grid.size() * cellSize), title: "Game of Life");
    sf::Clock clock;
    while (window.isOpen()) {// Dessiner la grille
        sf::Event event;
        while (window.pollEvent([&]event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }
            if (clock.getElapsedTime().asSeconds() > 2) {// Ferme la fenêtre après n secondes
                window.close();
            }
        }
        window.clear();
```

La méthode afficherAvecSFML utilise la bibliothèque SFML qui permet de créer des interfaces graphiques avec python.

Cette première partie de AfficheAvecSFML() initialise la grille

```
            sf::RectangleShape cell( size: sf::Vector2f( X: cellSize - 1,  Y: cellSize - 1));
            for (size_t y = 0; y < grid.size(); ++y) {
                for (size_t x = 0; x < grid[y].size(); ++x) {
                    if (grid[y][x] == 1) {
                        cell.setPosition(x * cellSize, y * cellSize);
                        cell.setFillColor(sf::Color::White); // Cellules vivantes en vert
                        window.draw(cell);
                    }
                }
            }
            window.display();
            sf::sleep( duration: sf::milliseconds( amount: 500)); // Attente entre les affichages
        }
    }
```

ConvertirGrille

Cette fonction convertit une grille représentée sous forme de vecteur de chaînes de caractères (vector<string>) en une grille numérique (vector<vector<int>>).

```
// Fonction pour convertir la grille fournie par le méthode fichier.vecteur
vector<vector<int>> convertirGrille(const vector<string>& grille) {
    vector<vector<int>> grid;
    for (const auto& ligne :const string& : grille) {
        vector<int> row;
        for (char c : ligne) {
            if (c == '0' || c == '1') {
                row.push_back( x: c - '0');
            }
        }
        grid.push_back(row);
    }
    return grid;
```

Test unitaire

Afin de vérifier le fonctionnement de chaque méthode ou classe, on peut réaliser des tests unitaires sur chacune des fonctions. Nous allons donc les mettre en place sur notre code afin que les vérifications soient effectuées. Pour cela, on créer des fonctions qui appellent et vérifient la sortie des méthodes pour une entrée prédéfinie.

```
// Test de la fonction `Fichier::vecteur`
TEST(FichierTest, ConversionVecteur) {
    Fichier fichier;
    string contenu = "5 5\n10101\n01010\n10101\n01010\n";
    vector<string> vecteur = fichier.vecteur(contenu);

    EXPECT_EQ(vecteur.size(), 5) << "Le vecteur doit contenir 5 lignes.";
    EXPECT_EQ(vecteur[1], "10101") << "La deuxième ligne doit être '10101'.";
}
```

Dans cette fonction, on appelle la méthode ConversionVecteur pour un string contenue fixé. Une fois l'appel de la méthode effectué, la fonction analyse le résultat : ici on vérifie la taille du vecteur ainsi que la première ligne qui doivent correspondre aux résultat attendu. Si tel est le cas, le test retourne vrai sinon il renvoie faux et un message expliquent la fonctionnalité n'a pas marché comme « Le vecteur doit contenir 5 lignes. » si la fonction retourne un vecteur une taille de 6 à la place du 5 escompté.

Une fois que tous les tests sont réalisés on ajoute l'appel de ces tests dans le main afin de tous les faire tourner des le début du programme.

```
cout<<"Initialisation des tests unitaires...\n"<<endl;
::testing::InitGoogleTest(&argc, argv);

if (argc > 1 && string(argv[1]) == "--run-sfml") {
    cout<<"Lancement de la fenêtre SFML...\n"<<endl;
}

cout<<"Exécution des tests unitaires...\n"<<endl;
int testResult = RUN_ALL_TESTS();

cout<<"Tous les tests unitaires sont terminés.\n"<<endl;
cout<<"Résultat des tests : " << (testResult == 0 ? "Tous validés !" : "Échec.") << "\n"<<endl;
```

Dans notre cas, tous les tests sont validés donc la sortie est positive. Si cela n'avait pas été le cas nous aurions eu un échec du programme ou le programme n'aurait pas abouti au résultat voulu et nous aurions eu l'affichage de la fonctionnalité n'ayant pas marché comme pour le vecteur.

```
Initialisation des tests unitaires...

Exécution des tests unitaires...

Tous les tests unitaires sont terminés.

Résultat des tests : Tous validés !
```

Conclusion :

Notre jeu de la vie est à présent terminé.

Nous avons mis en place la base de notre projet en créant plusieurs diagrammes UML (Use case, séquence, activité, de classe) afin de ne rien oublier de prendre en compte et en définissant un environnement de travail (Logiciel et outils, Langage et libraire, Organisation du code).

- Le diagramme Use Case donne un aperçu des différentes fonctionnalités dont notre programme va disposer.
- Le diagramme de Séquence met en avant les différentes étapes du système nécessaires à son bon fonctionnement. On peut y voir les entrées et sortie au travers de l'Interface Homme-Machine (I.H.M.) mais aussi les étapes au sein même du programme, comme la création de la grille sur la base du fichier pris en entrée.
- Le diagramme d'Activité précise les étapes données précédemment par le diagramme de Séquence, il précise le début et la fin du programme ainsi que les boucles.
- Le diagramme de Classe quant à lui propose une architecture simplifiée du code. On y retrouve les classes : composées de leurs méthodes et attributs, mais aussi le lien entre celles-ci.

Nous avons réalisé le code complet et fonctionnel du jeu de la vie. Celui-ci respecte les attendus : un mode console et un mode graphique, changement d'état des cellules, modification du nombre d'itération etc.

Pour finir, nous avons implémenter des tests unitaires dans notre code afin de vérifier le bon déroulé de celui-ci au travers de toutes les méthodes. Nous sommes donc maintenant capables de présenter un code complet et fonctionnel du jeu de la vie.

Annexe

Lien vers le document partagé GitHub du projet : https://github.com/CESi-CPIA2-2425/Projet-2-Le-Jeu-De-La-Vie.git

```cpp
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
/////////////////////////////////////////// LE JEU DE LA VIE ///////////////////////////////////////////////////////////////.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
//////////////////Inclusion des fichier, bibliothèques, namespace et instances de classes//////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.

//Bibliothèques externes (SFML -> Rendu graphique, GTEST -> Tests unitaires)
#include <SFML/Graphics.hpp>
#include <gtest/gtest.h>
//Bibliothèque internes (entrées/sorties, gestiosn de fichiers, vecteurs...)
#include <iostream>
#include <vector>
#include <ctime>
#include <stdio.h>
#include <string>
#include <filesystem>
//Inclusions des classes
#include "Fichier.h"
#include "Ligne.h"
//Utilisation de cette notation pour les cout,endl, vecteurs ...
using namespace std;
//instance de la classe Fichier dans le main
Fichier fichier;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
////////////////////////////Initialisation des valeurs (par défaut)//////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.

const int cellSize = 50;  //taille des cellules
const int gridWidth = 10;  //nombre de cellules en largeur (valeur présentes dans la première ligen du fichier texte)
const int gridHeight = 10;  //nombre de cellules en hauteur (valeur présentes dans la première ligen du fichier texte)

vector<vector<int> > grid( n: gridWidth, value: vector<int>( n: gridHeight));

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
////////////////////////////Appel de l'ouverture du fichier//////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.

string txt = fichier.lecture();
vector<string> liste = fichier.vecteur(txt);
vector<string> liste = fichier.vecteur(txt);
Ligne jeu( liste);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
////////////////////////////Affichage de la grille à l'écran/////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.

// Fonction pour convertir la grille fournie par le méthode fichier.vecteur
vector<vector<int>> convertirGrille(const vector<string>& grille) {
    vector<vector<int>> grid;
    for (const auto& ligne :const string& : grille) {
        vector<int> row;
        for (char c : ligne) {
            if (c == '0' || c == '1') {
                row.push_back( x: c - '0');
            }
        }
        grid.push_back(row);
    }
    return grid;
}
```

```cpp

// Fonction pour afficher la grille avec SFML
void afficherAvecSFML(const vector<vector<int>>& grid, int cellSize) {
    if (grid.empty()) return; // Vérifier si la grille est vide
    sf::RenderWindow window(sf::VideoMode( modeWidth: grid[0].size() * cellSize,  modeHeight: grid.size() * cellSize),  title: "Game of Life");
    sf::Clock clock;
    while (window.isOpen()) {// Dessiner la grille
        sf::Event event;
        while (window.pollEvent( [&] event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }
            if (clock.getElapsedTime().asSeconds() > 2) {// Ferme la fenêtre après n secondes
                window.close();
            }
        }
        window.clear();
        sf::RectangleShape cell( size: sf::Vector2f( X: cellSize - 1,  Y: cellSize - 1));
        for (size_t y = 0; y < grid.size(); ++y) {
            for (size_t x = 0; x < grid[y].size(); ++x) {
            for (size_t x = 0; x < grid[y].size(); ++x) {
                if (grid[y][x] == 1) {
                    cell.setPosition(x * cellSize, y * cellSize);
                    cell.setFillColor(sf::Color::White); // Cellules vivantes en vert
                    window.draw(cell);
                }
            }
        }
        window.display();
        sf::sleep( duration: sf::milliseconds( amount: 500)); // Attente entre les affichages
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
//////////////////////////////////////Test unitaires////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.

// Test de la fonction `Fichier::lecture`
TEST(FichierTest, LectureFichier) {
    Fichier fichier;
    string contenu = fichier.lecture();
    EXPECT_FALSE(contenu.empty()) << "Le contenu du fichier ne doit pas être vide.";
}

// Test de la fonction `Fichier::vecteur`
TEST(FichierTest, ConversionVecteur) {
    Fichier fichier;
    string contenu = "5 5\n10101\n01010\n10101\n01010\n";
    vector<string> vecteur = fichier.vecteur(contenu);

    EXPECT_EQ(vecteur.size(), 5) << "Le vecteur doit contenir 5 lignes.";
    EXPECT_EQ(vecteur[1], "10101") << "La deuxième ligne doit être '10101'.";
}

// Test du constructeur de la classe `Ligne`(pas  le constucteur par défaut)
TEST(LigneTest, ConstructionLigne) {
    vector<string> grille = {"10101", "01010", "10101", "01010"};
    Ligne jeu(grille);

    EXPECT_EQ(jeu.grille.size(), 4) << "La grille doit contenir 4 lignes.";
    EXPECT_EQ(jeu.grille[0], "10101") << "La première ligne de la grille doit être '10101'.";
}

// Test de la fonction `Ligne::CompteCell`
TEST(LigneTest, CompteCell) {
    vector<string> grille = {"101", "010", "101"};
    Ligne jeu(grille);

    int voisins = jeu.CompteCell( x: 1,  y: 1, grille);
    EXPECT_EQ(voisins, 4) << "La cellule (1,1) doit avoir 4 voisins vivants.";
}

// Test de la fonction `Ligne::Generation`
TEST(LigneTest, Generation) {
    vector<string> grille = {"000", "111", "000"};
    Ligne jeu(grille);

    jeu.change_etat();
    EXPECT_EQ(jeu.grille[1], "010") << "Après une génération, la ligne du milieu doit être '010'.";
}

```

```cpp
// Test de la conversion de grille pour SFML
TEST(ConversionGrilleTest, ConvertirGrille) {
    vector<string> grille = {"101", "010", "101"};
    vector<vector<int>> grid = convertirGrille(grille);

    EXPECT_EQ(grid.size(), 3) << "La grille convertie doit avoir 3 lignes.";
    EXPECT_EQ(grid[0][1], 0) << "La deuxième cellule de la première ligne doit être 0.";
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
////////////////////////////////////Main (appel de méthodes)////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.

int main(int argc, char** argv) {
    ////////////////////////////////Lancement des tests unitaires/////////////////////////////////////////////////////////
    cout<<"Initialisation des tests unitaires...\n"<<endl;
    ::testing::InitGoogleTest(&argc, argv);

    if (argc > 1 && string(argv[1]) == "--run-sfml") {
        cout<<"Lancement de la fenêtre SFML...\n"<<endl;
    }

    cout<<"Exécution des tests unitaires...\n"<<endl;
    int testResult = RUN_ALL_TESTS();

    cout<<"Tous les tests unitaires sont terminés.\n"<<endl;
    cout<<"Résultat des tests : " << (testResult == 0 ? "Tous validés !" : "Échec.") << "\n"<<endl;
    ////////////////////////////////Lancement du programme du jeu de la vie///////////////////////////////////////////////
    cout<<"Combien d'itérations voulez-vous réaliser ? \t" <<endl;
    int iteration = 5;
    cout<<iteration<<endl;

    cout<<"Comment voulez-vous obtenir le résultat ? \t 1. mode console \t 2. mode graphique"<<endl;
    int mode = 2;
    cout<<mode<<endl;

    while (iteration > 0) {
        if (mode == 1) {
            string fichierEntree = "grille.txt";
            string txt = fichier.lecture();
            string dossierSortie = fichierEntree + "_out";
            filesystem::create_directory(dossierSortie);
            //string txt = fichier.lecture();
            vector<string> liste = fichier.vecteur(txt);
            for (const auto& ligne :const string& : liste) {
                cout<<ligne<<endl;
            }
            liste.erase(position: liste.begin());
            Ligne jeu(liste);
            for (int i = 0; i < 5; ++i) {
                cout<<"Iteration : "<< i + 1 << "\n"<<endl;
                jeu.change_etat();
                jeu.AfficheGrille();
                string fichierIteration = dossierSortie + "/iteration_" + to_string(val:i + 1) + ".txt";
                fichier.ecriture(fichierIteration, jeu.grille);
            }
        }

        if (mode == 2) {
            Fichier fichier;
            string txt = fichier.lecture();
            vector<string> liste = fichier.vecteur(txt);

            for (const auto& ligne :const string& : liste) {
                cout<<ligne<<endl;
            }

            liste.erase(position: liste.begin());

            Ligne jeu(liste);

            for (int i = 0; i < 5; ++i) {
                jeu.change_etat();
                vector<vector<int>> grid = convertirGrille(jeu.grille);
                afficherAvecSFML(grid, cellSize); // Afficher la grille avec SFML
            }
```

```
            return 0;
        }
    }
    return testResult;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////// LE JEU DE LA VIE ///////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////// Classe Grille //////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////Inclusion des fichier, bibliothèques, namespace et instances de classes///////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "Grille.h"

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////Méthodes de la classe Grille//////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//Get de la hauteur de la grille
int Grille::get_h() const {
    return x;
}

//Get de la largeur de la grille
int Grille::get_l() const {
    return y;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////// LE JEU DE LA VIE ///////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////// Classe Grille //////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#ifndef GRILLE_H
#define GRILLE_H

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////Déclaration de la classe//////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

class Grille {  //Classe abstraite

private:
    int h;
    int l;
    int x;
    int y;

public:
    Grille(){}  //Constructeur
    virtual ~Grille() = default;  //Destructeur

    int get_h() const;
    int get_l() const;

    virtual void change_etat()=0;  //Méthode virtuelle pure
};

#endif //GRILLE_H

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```cpp
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
/////////////////////////////////////////////// LE JEU DE LA VIE ///////////////////////////////////////////////////////.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
/////////////////////////////////////////////////// Classe Ligne ///////////////////////////////////////////////////////.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
////////////////////Inclusion des fichier, bibliothèques, namespace et instances de classes/////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.

#include "Ligne.h"
#include "Fichier.h"
#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
/////////////////////////////////////////////Méthodes de la classe Ligne/////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.

Ligne::Ligne(vector<string> grille): grille(grille), colonne(grille[0].size()), ligne(grille.size()), x(0), y(0), gridWidth(0), gridHeight(0) {}

//Changement de l'éta '0' ou '1' en fonction des 8 cases environnantes
void Ligne::change_etat() {
    vector<string> nvlle = grille;

    for (int i = 0; i < ligne; ++i) {
        for (int j = 0; j < colonne; ++j) {
            int voisin = CompteCell(i, j, grille);
            if (grille[i][j] == '1') {
                nvlle[i][j] =(voisin == 2 || voisin ==  3) ? '1' : '0';
            }else {
                nvlle[i][j] = (voisin ==3) ? '1' : '0';
            }
        }
    }
    grille = nvlle;
}

//Compte le nombre de cellules qui composent la grille
int Ligne::CompteCell(int x,int y,vector<string> grille) {
    int compteur = 0;
    for(int i=-1; i<=1; i++) {
        for(int j=-1; j<=1; j++) {
            if(i == 0 && j == 0) continue;
            int nx = x+i;
            int ny = y+j;
            if (nx >= 0 && nx<ligne && ny>= 0 && ny < colonne) {
                compteur += grille[nx][ny] =='1';
            }
        }
    }
    return compteur;
}

//Affichage de la Grille de '0' et '1' dans la console de l'ordinateur
void Ligne::AfficheGrille() {
    for (const auto& l :const string& : grille) {
        cout << l << '\n';
    }
    cout << '\n';
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////// LE JEU DE LA VIE ////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////// Classe Ligne ////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#ifndef LIGNE_H
#define LIGNE_H

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////Inclusion des fichier, bibliothèques, namespace et instances de classes///////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <vector>
#include "Fichier.h"
#include "Grille.h"

using namespace std;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////Déclaration de la classe///////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


class Ligne :public Grille{

public:
    vector<string>grille;
    int colonne;
    int ligne;
    int x;
    int y;
    int gridWidth;
    int gridHeight;

    Ligne(vector<string> grille);// consturcteur

    int get_h() const;
    int get_l()const;

    void change_etat()override;
    int CompteCell(int x,int y,vector<string> grille);
    void AfficheGrille();
};
#endif //LIGNE_H

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////// LE JEU DE LA VIE ////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////// Classe Fichier //////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////Inclusion des fichier, bibliothèques, namespace et instances de classes///////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "Fichier.h"
#include <string>
#include <vector>
#include <iostream>//entrée/sortie
#include <fstream>//fichier
#include <sstream>

using namespace std;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////Méthodes de la classe Fichier//////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//Ouverture et recupération des données présentes dans le fichier texte (sous forme de string)
```

```cpp
string Fichier::lecture() {
    string full;
    string contenu;
    string const nomFichier(s:"C:/Users/thais/Documents Local/CPI A2/Projet 2/Livrable/Projet-2/Fichier.txt");
    ifstream monFlux(s:nomFichier.c_str());
    while (getline([&]monFlux, [&]contenu)) {
        if(monFlux)  //Teste si l'ouverture a fonctionné
        {
            full +=  contenu + "\n";
        }
        else
            cout << "Impossible d'ouvrir le fichier !" << endl;
    }
    monFlux.close();
    return full;
}

//transformation de la string du fichier en une serie de vecteurs (matrice en 2D)
vector<string> Fichier::vecteur(const string& txt) {
    vector<string> liste;
    stringstream ss(txt);
    string ligne;
    while (getline([&]ss,[&]ligne)) {
        liste.push_back(ligne);
    }
    return liste;
}

void Fichier::ecriture(const string& nomFichier, const vector<string>& grille) {
    ofstream fichier(nomFichier);  // Ouvre le fichier en mode écriture
    if (!fichier.is_open()) {
        cerr << "Erreur : Impossible d'ouvrir le fichier " << nomFichier << endl;
        return;
    }

    // Écrire la grille dans le fichier
    for (const auto& ligne:const string& : grille) {
        fichier << ligne << '\n';
    }

    fichier.close();
    cout << "État écrit dans : " << nomFichier << endl;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```cpp
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////// LE JEU DE LA VIE ////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////// Classe Fichier ///////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#ifndef FICHIER_H
#define FICHIER_H

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////Inclusion des fichier, bibliothèques, namespace et instances de classes//////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <string>
#include <vector>

using namespace std;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////Déclaration de la classe///////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

class Fichier {

private:
    string chemin_fichier;

public:
    Fichier() {} // constructeur

    string lecture();
    vector<string> vecteur(const string& txt);
    void ecriture(const string& nomFichier, const vector<string>& grille);

};

#endif
```

```
};
#endif //FICHIER_H

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////.
```

```
cmake_minimum_required(VERSION 3.15)
project(Projet_2 LANGUAGES CXX)

# Définir les options de compilation
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_FLAGS "-g -Wall -Werror")

# Spécifier les chemins vers SFML
set(SFML_INCLUDE_DIR "C:/Users/thais/Bibli_SFML_c++/SFML-2.6.2-windows-gcc-13.1.0-mingw-64-bit/SFML-2.6.2/include")
set(SFML_LIBRARY_DIR "C:/Users/thais/Bibli_SFML_c++/SFML-2.6.2-windows-gcc-13.1.0-mingw-64-bit/SFML-2.6.2/lib")

# Inclure les fichiers d'en-tête et de bibliothèque SFML
include_directories(${SFML_INCLUDE_DIR})
link_directories(${SFML_LIBRARY_DIR})

# Ajouter les fichiers source
file(GLOB_RECURSE SRCS "Projet_2.cpp")

# Ajouter l'exécutable principal
add_executable(SFMLTest ${SRCS} Projet_2.cpp)

# Chemin vers le répertoire Google Test
set(GTEST_DIR "C:/Users/thais/Documents Local/CPI A2/Projet 2/Livrable/Projet-2/cmake-build-debug/googletest/googletest-main")

# Ajouter Google Test comme sous-dossier
add_subdirectory(${GTEST_DIR} googletest)

# Inclure les répertoires de Google Test
include_directories(${GTEST_DIR}/googletest/include)
include_directories(${GTEST_DIR}/googlemock/include)

# Ajouter un exécutable pour les tests unitaires
add_executable(UnitTests Projet_2.cpp
        Fichier.cpp
        Ligne.cpp
        Grille.cpp)

# Lier les bibliothèques SFML
target_link_libraries(SFMLTest sfml-graphics sfml-window sfml-audio sfml-network sfml-system)

# Lier Google Test ET SFML aux tests unitaires
target_link_libraries(UnitTests gtest gtest_main sfml-graphics sfml-window sfml-audio sfml-network sfml-system)
```