Rapport de projet 2048

Universite Paris Saclay

Date: Décembre 2024/25

Auteurs

- VUCENIC, Elena, elena.vucenic@etu-upsaclay.fr, MI-3
- OANCEA, Bianca, bianca.oancea@etu-upsaclay.fr, MI-3

Introduction

Le projet visait à recréer le jeu 2048 en mode console en suivant une approche progressive. Ensuite, le projet a évolué vers une version avec des graphismes, en utilisant la bibliothèque SFML..

Développement

Niveau 0 : réalisée, documentée, testée

Fichier modele.hpp

Explication des bibliothèques utilisées

1. Voici les bibliothèques utilisées dans ce fichier:

<iostream> : Permet la gestion de l'entrée et de la sortie via la console.

<cstdlib> : Contient des fonctions pour générer des nombres aléatoires rand().

Exemple : int case1_l = rand() % 4;

<ctime> : Fournit la fonction time() pour initialiser le générateur de nombres aléatoires.

Exemple : srand(time(0));

Cette ligne garantit que les nombres générés sont différents à chaque exécution.

<iomanip> : Permet de formater l'affichage des nombres, assurant un alignement correct des colonnes sur le plateau.

Exemple : Pour fonction: *string dessine(Plateau brd)*

<vector> : Permet l'utilisation de tableaux dynamiques pour gérer la structure du plateau.

Exemple : using Plateau = vector<vector<int>>;

<string> : Fournit des outils pour manipuler facilement des chaînes de caractères.

Exemple : string aux = to_string(nraux);

Cette commande permet de convertir le nombre *nraux* en chaîne de caractères (string) afin qu'il puisse être affiché à l'écran.

Après l'inclusion des bibliothèques nécessaires, le code utilise deux lignes importantes :

using namespace std;

Cette ligne permet d'utiliser les éléments du namespace standard std sans avoir à les préfixer avec std::.

using Plateau = vector<vector<int>>;

Cette ligne définit un alias pour le type vector<vector<int>>, ce qui permet d'utiliser un nom plus court, Plateau, pour désigner ce type de données.

Première partie du code

Dans cette première partie du code, plusieurs fonctions sont utilisées pour initialiser le jeu et gérer l'apparition des tuiles sur le plateau.

int tireDeuxOuQuatre()

Cette fonction génère aléatoirement un 2 ou un 4, avec une probabilité de 90 % pour un 2 et 10 % pour un 4.

Plateau plateauVide()

La fonction plateauVide() crée un plateau de jeu vide, représenté par une matrice 4x4 où toutes les cases contiennent des 0. Cette fonction est utilisée pour initialiser un plateau vide, ce qui est essentiel pour commencer une partie.

Plateau plateauInitial()

Cette fonction est utilisée pour générer le plateau de départ. Après avoir créé un plateau vide, elle place deux tuiles (un 2 ou un 4) dans des positions aléatoires sur le plateau.

Plateau Nouvelletuille(Plateau tab)

Cette fonction permet d'ajouter une nouvelle tuile sur le plateau. Elle cherche une case vide sur le plateau et y place un 2 ou un 4. Cela se produit après chaque mouvement du joueur, afin de continuer à ajouter des tuiles et de faire évoluer le jeu.

Deuxième partie du code

Dans cette partie du code, plusieurs fonctions sont responsables de la gestion des déplacements et des combinaisons des tuiles sur le plateau de jeu.

Plateau déplacementGauche(Plateau plateau, int& score)

Cette fonction déplace toutes les tuiles vers la gauche et les combine lorsque cela est possible. Si deux tuiles de même valeur se trouvent côte à côte, elles sont combinées (multipliées par deux), et le score du joueur est mis à jour. Après chaque déplacement, la fonction retourne le nouveau plateau avec les tuiles déplacées et combinées.

Plateau déplacementDroite(Plateau plateau, int& score)

Semblable à la fonction précédente, mais cette fois-ci, elle déplace les tuiles vers la droite. Les tuiles se déplacent et se combinent dans cette direction, avec un mécanisme identique au déplacement vers la gauche. Le score est mis à jour chaque fois que des tuiles sont combinées.

Plateau déplacementHaut(Plateau plateau, int& score)

Cette fonction permet de déplacer les tuiles vers le haut du plateau. Elle fonctionne de la même manière que les précédentes, mais les tuiles sont déplacées dans la colonne plutôt que dans la ligne. Chaque fois qu'une combinaison se produit, le score du joueur est ajouté.

Plateau déplacementBas(Plateau plateau, int& score)

Cette fonction déplace les tuiles vers le bas, tout en combinant celles qui sont égales, tout comme les autres fonctions de déplacement. Le score est aussi mis à jour chaque fois que des tuiles sont combinées.

Plateau déplacement(Plateau plateau, int direction, int &score)

Cette fonction centrale permet de gérer tous les déplacements dans le jeu 2048. Elle prend en compte la direction du déplacement, qui est représentée par un entier (0 pour gauche, 1 pour droite, 2 pour haut, et 3 pour bas). En fonction de cette direction, elle appelle la fonction correspondante (déplacementGauche, déplacementDroite, déplacementHaut, ou déplacementBas).

Troisième partie du code

Cette partie du code se concentre sur deux aspects importants du jeu 2048 : l'affichage du plateau de jeu et la vérification des conditions de fin de partie. Elle comprend une fonction pour dessiner le plateau et deux fonctions pour déterminer si la partie est terminée ou si le joueur a gagné.

string dessine(Plateau brd)

Cette fonction est responsable de l'affichage du plateau de jeu dans un format lisible pour l'utilisateur. Elle prend un plateau (de type Plateau, c'est-à-dire un tableau 4x4 d'entiers) et retourne une chaîne de caractères qui représente le plateau sous forme textuelle. Chaque case du plateau est affichée avec des espaces supplémentaires en fonction de sa valeur (pour l'alignement).

bool estTerminé(Plateau plateau, Plateau plateau1)

Cette fonction permet de vérifier si la partie est terminée, c'est-à-dire si aucun mouvement valide n'est possible. Elle compare l'état actuel du plateau avec un plateau précédent (plateau1). Si ces deux plateaux sont identiques, cela signifie qu'aucun déplacement n'a eu lieu depuis le dernier mouvement. Ensuite, la fonction vérifie si le plateau peut être déplacé dans une direction (gauche, droite, haut ou bas). Si aucun déplacement n'est possible dans aucune direction, la partie est terminée.

bool estGagnant(Plateau plateau)

La condition de victoire est simple : si une case du plateau contient le nombre 2048, le joueur a gagné. La fonction parcourt chaque case du plateau pour rechercher ce nombre et retourne true si elle le trouve, sinon false.

Fichier: ncurses_min.cpp

Explication des bibliothèques utilisées

Voici les bibliothèques utilisées dans ce fichier:

#include "modele.hpp"

Cette directive inclut un fichier d'en-tête personnalisé, **modele.hpp**, dans le fichier source C++ actuel. Il contient la définition de la logique du jeu et d'autres structures de données.

#include <curses.h>

Cette bibliothèque **curses** est une bibliothèque de gestion d'affichage pour des interfaces utilisateur en mode texte (terminal).

Exemples d'utilisation :

- **initscr()** : Initialise l'écran et démarre l'interface avec curses.
- **printw()** : Permet d'écrire du texte dans la fenêtre du terminal.
- **getch()** : Lit un caractère du clavier.
- **endwin()** : Ferme proprement la fenêtre curses à la fin de l'application.

Fonction main():

La fonction main() commence par initialiser l'environnement nécessaire au jeu à l'aide de la bibliothèque curses. La fonction **initscr()** initialise l'écran pour l'affichage, puis **cbreak()** permet de désactiver la mise en mémoire tampon des entrées (pour que les entrées soient traitées immédiatement). **keypad(stdscr, TRUE)** permet d'activer la détection des touches spéciales comme les flèches directionnelles.

Ensuite, une fenêtre de taille 20x20 est créée avec la fonction **newwin()**, et des couleurs sont activées à l'aide de **start_color()** et **init_pair(1, COLOR_WHITE, COLOR_BLUE)**, définissant un schéma de couleur pour l'affichage du jeu. La couleur est ensuite appliquée avec **attron(COLOR_PAIR(1))**.

Le jeu commence par la génération du plateau avec deux tuiles aléatoires grâce à la fonction **plateauInitial()**. Le score est affiché au début à l'aide de **printw()**, puis le plateau est dessiné avec la fonction **dessine()**, qui affiche visuellement l'état du jeu dans le terminal.

Dans la boucle principale du jeu, le programme attend que l'utilisateur appuie sur une touche, grâce à **getch()**. En fonction de la touche (flèche gauche, droite, haut ou bas), le plateau est déplacé dans la direction correspondante à l'aide de la fonction **déplacement()**. Après chaque mouvement, une nouvelle tuile est ajoutée avec **Nouvelletuille()**, et le jeu est redessiné avec le score mis à jour. À ce moment, **refresh()** est utilisé pour actualiser l'affichage du jeu, garantissant que les changements effectués (comme l'ajout d'une tuile ou le déplacement d'une tuile) soient immédiatement visibles à l'écran.

La boucle se termine lorsque le jeu est soit gagné (un 2048 est atteint) avec la fonction **estGagnant()**, soit lorsque plus aucun mouvement n'est possible (plateau bloqué) avec **estTerminé()**. Enfin, le programme nettoie l'affichage avec **clear()**, puis rafraîchit l'écran avec **refresh()** pour afficher le nouveau plateau et le score.

Après la fin du jeu, la fenêtre est supprimée avec **delwin()**, et **endwin()** termine l'utilisation de la bibliothèque curses, réinitialisant l'affichage du terminal à son état normal. Cela permet d'offrir une expérience fluide et interactive pour le joueur dans le terminal.

Niveau 1 : réalisée, documentée, testée

1. Ajouter de la couleur à la console

Dans le code, on utilise la bibliothèque curses pour ajouter de la couleur à l'affichage. Avec **start_color()**, on active les couleurs dans le terminal. Ensuite, on définit une paire de couleurs avec **init_pair(1, COLOR_WHITE, COLOR_BLUE)**, où le texte est blanc et le fond est bleu. Cette couleur est appliquée à l'affichage grâce à **attron(COLOR_PAIR(1))**.

2. Utiliser les flèches directement

Grâce à **keypad(stdscr, TRUE)**, on peut détecter directement les touches des flèches directionnelles (haut, bas, gauche, droite).

3. Rafraîchir l'affichage au lieu d'ajouter des lignes

Au lieu d'ajouter de nouvelles lignes à chaque mouvement, on utilise **clear()** pour effacer l'écran et redessiner le plateau à l'endroit exact où il était. Cela améliore l'affichage, car l'état du jeu est mis à jour à chaque mouvement sans encombrer l'écran.

4. Calculer le score sans variables globales

On calcule le score localement en utilisant une structure qui contient à la fois le plateau et le score.

Dans le code, on utilise le symbole **&s** qui fait référence à l'utilisation de l'opérateur **&** devant une variable, ce qui signifie qu'on passe la variable par **référence,** cela signifie que la fonction travaille directement sur la variable d'origine, plutôt que de recevoir une copie de celle-ci.

Niveau 2 : réalisée, documentée, testée

Fichier Makefile

Ce fichier permet de simplifier la compilation en évitant de devoir appeler manuellement le compilateur à chaque fois.

Tout d'abord, il définit les variables CPPFLAGS, LDFLAGS, SRC et EXEC.

CPPFLAGS inclut les options de compilation, comme -Wall pour activer tous les avertissements et -g pour permettre le débogage. LDFLAGS spécifie les bibliothèques à lier, dans ce cas, la bibliothèque ncurses avec l'option -lncurses. SRC définit le fichier source à compiler, ici ncurses_min.cpp, et EXEC indique le nom du fichier exécutable résultant, qui est ncurses_min.

Le bloc all est la cible par défaut qui compile le programme. Elle utilise la commande clang++ pour compiler le fichier source en un exécutable, en intégrant les options de compilation et de liaison définies précédemment.

La cible clean est utilisée pour supprimer l'exécutable généré, ce qui permet de repartir de zéro si nécessaire. Enfin, la cible rebuild combine les deux étapes, d'abord en nettoyant les anciens fichiers, puis en recompilant le projet.

GitHub

Le lien pour le repository Game2048 sur GitHub:

https://github.com/elenavucenic/Game2048/tree/main

https://github.com/biancaoan/2048_project.git

Niveau 3 : réalisée, documentée, testée

Pour le niveau 3, Bianca a travaillé sur Visual Studio en utilisant la bibliothèque SFML.

Le processus a été divisé en 3 étapes :

1. Apprendre les bases de SFML
2. Modifier la fonction « dessine » et la fonction main du code original.
3. La dernière étape a été la partie esthétique, en jouant avec les couleurs, la taille, le positionnement et les polices.

1. Les bases de SFML et les instructions utilisées
   Cette étape a commencé par des recherches sur le site Web de SFML, depuis comment le télécharger et l'utiliser dans Visual Studio jusqu'à l'apprentissage des instructions ci-dessous.
   Exemples:
   - sf::RenderWindow window(sf::VideoMode(4 * 160, 5 * 160), "2048 - par Bianca et Elena");
     Crée une fenêtre de taille 640x800 pixels (4 colonnes et 5 lignes de tuiles, chacune de 160x160), avec le titre « 2048 - par Bianca et Elena »
   - window.clear();
     Efface la fenêtre pour préparer le rendu de la prochaine image
   - dessine(window, newgamet, s);
     Appelle la fonction personnalisée dessine pour dessiner le tableau de jeu ()
   - window.display();
     Affiche fenêtre

- sf::Event event;
  Déclare un objet d'événement pour gérer les entrées utilisateur (Haut, Bas, Gauche, Droite).
- window.pollEvent(event)
  Vérifie s'il y a des événements.
- event.key.code == sf::Keyboard::Left
- Vérifie si l'utilisateur a appuyé sur la touche fléchée gauche
- sf::Font font;
  Définir la police utilisée pour l'objet texte
- sf::RectangleShape tile(sf::Vector2f(160, 160));
  Crée une forme rectangulaire de taille 160x160 pixels, représentant une seule tuile du tableau de jeu.
- tile.setFillColor(sf::Color(255, 250, 250));
  Définit la couleur de la tuile en (RGB : 255, 250, 250) – (Pale Pink).

2. En utilisant SFML, la fonction « dessine » est passée de retourner le tableau sous forme string à dessiner le tableau de jeu et le score dans la « window ».
   La fonction main a été modifiée pour ouvrir une fenêtre au lieu d'afficher dans la console. À la fin, bien que le langage soit différent, la structure reste la même.

   - void dessine(sf::RenderWindow& window, Plateau brd, int &scoreText)

     Structure:

     1. Dessiner le rectangle au-dessus du tableau de jeu, où le score est écrit.
     2. Dessiner 16 rectangles (tuiles), chacun ayant un numéro (le plateau[i][j] après chaque round)
     3. Dessiner les lignes qui séparent les tuiles et le score du plateau.

La première difficulté que j'ai rencontrée était l'utilisation de la police. D'après le tutoriel SFML, j'avais cette ligne de code :

```
if (!font.loadFromFile("arial.ttf"))
{
   // erreur...
}
```

Ce code ne fonctionnait pas pour moi, donc j'ai rencontré cette erreur car je ne savais pas comment accéder au dossier où la police était stockée. J'ai résolu le problème en recherchant et en découvrant qu'il existe un dossier par défaut dans le répertoire Windows : « C:\Windows\Fonts\arial.ttf ».

Bien que la fonction principale garde la même structure que celle utilisée dans l'affichage console, j'ai rencontré des problèmes avec les entrées de l'utilisateur, donc pour résoudre cela, j'ai utilisé ce code.

```
if (dircurrent != -1)
{
… moved = True;
}
if (event.type == sf::Event::KeyReleased)
{
   moved = false;
}
```

Cette condition vérifie si une touche a été relâchée. Cela signifie que l'utilisateur a cessé de maintenir une touche enfoncée.

3. Pour la partie esthétique, je me suis occupée du choix des couleurs, de la taille des numéros et du score, ainsi que de la position des rectangles et des lignes afin que tout soit bien organisé.

   L'ordre dans lequel les différentes parties (cf. 2) sont dessinées est important, car le programme les affiche dans cet ordre dans la fenêtre. Ainsi, les lignes doivent être dessinées en dernier, sinon les rectangles (qu'il s'agisse des tuiles ou du score) seront dessinés par-dessus, rendant les lignes invisibles. De plus, les numéros doivent être dessinés par-dessus les tuiles.

   Pour les couleurs, j'ai utilisé un site web qui proposait quelques couleurs de base en RGB et j'ai joué avec les nuances (Exemple : le score est coloré dans une teinte légèrement plus foncée que les lignes, donc j'ai simplement soustrait environ 20 à chaque code : (119, 136, 153) - (100, 117, 134)). La dernière tuile est dorée, ce qui signifie que l'utilisateur a gagné !

   Pour l'écriture, j'ai utilisé les lignes de code suivantes :

   ```
   sf::FloatRect textBounds = text.getLocalBounds();
   text.setOrigin(textBounds.width / 2, textBounds.height / 2 + 10);
   text.setPosition(j * 160 + 160 / 2, i * 160 + 160 / 2 + 160);
   ```

   Pour centrer les numéros dans les tuiles, même lorsqu'ils augmentent en taille, j'ai utilisé cette ligne de code : « score.setPosition(); » afin de m'assurer que tout soit bien positionné. (Exemple : pour que le score soit affiché au-dessus du tableau de jeu, j'ai utilisé score.setPosition(0, 0);).

Organisation du travail

Nous avons organisé notre travail sur le projet en tenant régulièrement des réunions en présentiel. Ces rencontres ont rendu la communication et la collaboration plus faciles, ce qui nous a permis de partager nos idées, de résoudre les problèmes rapidement et de suivre l'avancement du projet à chaque étape.

Concernant la répartition des tâches, Elena a principalement travaillé avec l'environnement **Jupyter**, et Bianca a utilisé **Visual Studio** pour développer les fonctions et l'interface graphique.

Pour garantir la qualité et la cohérence de notre code, nous avons choisi de travailler de manière collaborative : d'abord, nous avons réparti les fonctions à coder, puis nous avons échangé nos parties pour que chacun puisse tester et vérifier le travail de l'autre.

Cette organisation et l'utilisation des ressources disponibles nous ont permis de travailler efficacement, de réussir notre projet et d'améliorer nos compétences techniques.

Prise de recul

Pendant ce projet, nous avons appris à utiliser des outils que nous n'avions jamais utilisés auparavant. Par exemple, nous avons appris à travailler avec la bibliothèque **curses** pour créer des interfaces en mode texte. Cela nous a permis de comprendre comment gérer l'affichage et l'interaction avec l'utilisateur dans un environnement sans interface graphique. Nous avons aussi utilisé **SFML** dans **Visual Studio** pour la partie graphique, ce qui nous a aidées à mieux comprendre comment gérer les fenêtres, les événements et les objets visuels dans un projet C++. Au début, nous avons rencontré des difficultés pour fusionner notre travail sur un seul grand projet, mais cela a été une bonne expérience d'apprentissage.

Ce projet nous a permis d'apprendre à travailler ensemble sur un projet de programmation plus grand que ce à quoi nous étions habitués. Nous avons dû gérer le travail, diviser les tâches en fonction des compétences de chacun et discuter de nos préoccupations concernant l'efficacité et l'aspect du projet. Cela nous a vraiment aidés à mieux comprendre comment collaborer et communiquer efficacement en équipe. Ce projet a été une expérience enrichissante et il nous a aidés à mieux comprendre les différentes approches et points de vue de chacun.

Nous avons travaillé sur ce projet pendant environ 60 heures et bien que nous ayons réparti le travail, nous avons vérifié et testé les fonctions des uns et des autres.

De plus, nous avons découvert **GitHub**. Nous avons appris à utiliser les branches, à résoudre les conflits et à suivre les changements dans le code.

Ce projet nous a permis d’acquérir des compétences techniques que nous utiliserons sûrement dans nos futurs travaux.