Encadré par : Teo Sanchez

Auteurs

- Ben-Zine, Asma, asma.ben-zine@universite-paris-saclay.fr, MI4
- HAMMAS, Redwane , redwane.hammas@universite-paris-saclay.fr, MI2

Résumé du travail effectué

Algorithme

Source de l'algorithme utilisé: https://www.baeldung.com/ (https://www.baeldung.com/)

Niveau 0

[réalisé] Les fichiers de projet compilent.

Pour Compiler et exécuter

avec Makefile

```
make && main_game.exe
```

sur Windows

```
g++ -std=c++11 main_game.cpp -o main_game modele.cpp color.cpp && m
ain_game.exe
```

sur Linux

```
g++ -std=c++11 main_game.cpp -o main_game modele.cpp color.cpp &&
 ./main_game
```

[réalisé] Le jeu est fonctionnel.

[réalisé] Le jeu respecte les règles de 2048.

[réalisé] Le score est mis à jour à chaque mouvement.

- Plateau est définit comme une `class` qui hérite de `vector<vector<int>>` , après chaque mouvement, ce score est mis à jour au niveau des fonctions `sumX` et `sumY` qui stockent la somme des cases identiques dans une variable (score).

[réalisé] L'implantation respecte la structure imposée.

[réalisé] Toutes les fonctions sont spécifiées, documentées et testées.

- Le fichier `test_game.cpp` comporte les tests des fonctions du jeu.

Pour Compiler et effectuer les tests

Compilation > `g++ -std=c++11 test_game.cpp -o test_game modele.cpp color.cpp`

Exécution > `test_game.exe` (windows)

[réalisé] Lors de l'affichage, les colonnes sont bien alignées, quelles que soient les valeurs de chaque tuile.

- Utilisation de la méthode setw() ainsi de la bibliothèque center_helper(copié du site stackoverflow)

[réalisé] Le rapport est bien rédigé et comprend : Une présentation du jeu && Une documentation de votre application.

[réalisé] La soutenance est soignée et inclut une brève démonstration de votre jeu.

Niveau 1

[réalisé] Rajouter un peu de couleur à la console.

- Utilisation de "ANSI escape codes" pour rajouter les couleurs, création des fichiers color.cpp et color.h qui définissent une classe de couleurs `Color` et les fonctions qui associent à chaque tuile deux `Colors : foreground && background` .
- Cette méthode permet de rajouter de la couleur à la console **sans utiliser** `ncurses` .
- Cette méthode est testée sur le `cmd` de Windows 10, le terminal de `kali` Linux installé comme sous-système ainsi que les machines dans les salles de TP - l'affichage n'est pas parfait sur Windows 11.

Explication et exemple

- Une déclaration de couleur `ANSI` se fait comme suit : `\033[` + `format` (int : 1 = Bold, 2 = Normal, 3 = Italic) + `;` + `color_code` (int : code de `background`) + `;` + `color_code` (int : code de `foreground`) + `m` texte à afficher + `\033[0m`

Exemple

```
std::cout << "\033[2;43;34mCe texte est bleu sur jaune\033[0m";
```

- Les codes de couleur sont déclarés par la fonction `declare_colors()` dans `color.cpp`

[réalisé] Jouer en utilisant directement les flèches (haut, bas, gauche et droite) du clavier, sans avoir à appuyer sur entrée à chaque fois.

- Utilisation de la fonction `getch()` de la bibliothèque `<conio.h>` qui n'est importée que si la compilation s'effectue sur Windows.

[réalisé] L'affichage d'un plateau se fait à la place du plateau précédent

- Utilisation de `refreshConsole` (définie dans `main_game.cpp`).
- **[fonctionnement]** utilisation des `preprocessor directives` pour déterminer le système d'exploitation puis, communication de la bonne commande au console (`cls` pour Windows et `clear` pour Linux et macOS).

[réalisé] Calculer le score du jeu non pas avec une variable globale, mais avec une structure de données qui associe un plateau Plateau à un score.

- Création : une classe plateau qui a 2 attributs le plateau et le score, donc après chaque mouvement, le plateau et le score sont mis à jour - Création de la classe `Plateau` qui prend le score comme attribut. - Pour éviter la modification du code, le plateau n'est **pas représenté par une attribut**. En effet, la classe `Plateau` hérite de la classe `vector<vector<int>>`

[résolu] Les constructeurs ne sont pas hérités par défaut dans `C++ >` ajout de `using vector::vector` - Nous avons posé la question (https://stackoverflow.com/questions/70098843/adding-a-member-to-stdvectorstdvectorint-class-in-c) sur `stackoverflow`

Niveau 2

[réalisé] Créer un Makefile simple pour la compilation.

[réalisé] Utiliser un \"version control system\" comme git.

Création d'une repo sur github (https://github.com/redyummybread/game_2048) pour gérer les versions du projet et travailler en binôme.

[note] La repo sera privée au moment de soumission du projet.

Les commandes GIT utilisées

> Mise à jour des fichiers locaux : `git pull`
>
> Sauvegarde des modifications : `git commit -a -m \"un message des modifications\"`
>
> Téléverser les fichiers vers Github : `git push`
>
> Annuler les modifications : `git stash`
>
> Annuler un commit : `git revert $commit`

Niveau 3

[réalisé] Proposer une version de 2048 avec une interface graphique, en utilisant la SFML.

Démonstration

L'implementation du jeu 2048 était très délicate. on a créé deux versions de projet :

> `game_console` : version console `game_sfml` : version sfml (avec l'interface graphique)

Les fichiers du projet

`game_console`

`main_game.cpp` : le script principal (main) qui va s'exécuter

`modele.cpp` : le fichier qui regroupe les fonctions principales du jeu

`modele.h` : le header file pour les déclarations

`test_game.cpp` : les tests unitaires de `modele.cpp` et `color.cpp`

`color.cpp` : une bibliothèque de couleur que nous avons écrit pour afficher de la couleur directement au console du système d'exploitation.

`color.h` : le header file qui contient toutes les fonctions implantées dans color.cpp

`center_class.cpp` une bibliothèque permettant de centrer le texte après utilisation de `setw()`

`makefile` pour compiler le code

`game_sfml`

`graphx.cpp` regroupe les fonctions de rendering de l'SFML

`graphx.h` le header file pour les déclarations

`setup.h` regroupe les différentes dimensions du jeu - modifiable.

NOTE : Si le jeu compile est l'interface est trop large -> Modifier la variable `SCALE`

Compilation et exécution

> windows - console : `make && main_game.exe`
>
> windows - sfml : `make && main.exe`

Organisation du travail

- L'implémentation du jeu 2048 a été faite en binôme. On travaillait dans la bibliothèque universitaire pour discuter l'avancement du projet.
- Au cas où un des deux membres a avancé dans la recherche, résolution des problèmes ou implémentation du code, on utilisait discord pour revoir et améliorer le code.
- L'implémentation du code en relation avec les notions vues durant le semestre, autrement dit, toutes les fonctions qui implémentent la première version du jeu 2048 sans graphique ou couleur a été implémentée en binôme, au cas où on avait des problèmes, on chercher une solution sur internet, poser des questions sur `stackoverflow` ou directement au chargé de TD.
- La partie qui nécessite une recherche documentaire restait ouverte, de manière que chacun faisait des recherches de son côté puis l'implémentatoin des meuilleures solutions se faisait en binôme.

Prise de recul

Fonction rand()

> La fonction `rand()` ne marchait pas au début à cause de des appels multiples de la fonction `timeInit()` , qui initialise le sub-seed à plusieurs reprises > Ce qui en résulte l'initilisation du même nombre aléatoire.

La solution était de faire l'appel à la fonction `timeInit()` une seule fois dans la fonction `main()` .

Le score

> Pour calculer et mettre à jour du score sans utilisation d'une variable globale. Nous avons proposé de créer une classe `Plateau` qui hérite toutes les méthodes de `vector<vector<int>>` et nous l'avons attribuée une variable `score` . C'est une sorte de généralisation du `typedef` proposé par le prof. qui permet de maintenir la même syntaxe.

[question] Notre question sur stackoverflow : lien (https://stackoverflow.com/questions/70098843/adding-a-member-to-stdvectorstdvectorint-class-in-c)

Affichage avec les couleurs dans le terminal

> Pour afficher de la couleur au console sans utiliser `ncurses` nous avons proposé l'utilisation des échappements de couleurs ANSI.

En effet, pour le réaliser il fallait écrire toute une bibliothèque de couleurs afin de definir une classe des couleurs contenant les attributs `background` et `foreground` ainsi que des diffèrentes fonctions;

`declare_colors()` > attribue à chaque couleur (foreground & background) son code (un entier).

`get_color_of()` : qui retourne un string échappenent ANSI représentant la couleur attribuée à value.

`format_color()` > renvoie un string échappement ANSI représentant les couleurs.

Affichage de la console dans le terminal

> Pour que l'affichage soit bien aligné, il fallait utiliser la fonction `setw()` de la bibliothèque `<iomanip>` .
>
> Ceci pose un autre problème : Les valeurs de tuiles sont alignées à gauche. Pour résoudre ceci, nous avons rajouter une nouvelle bibliothèque : `center-class.cpp` trouvé sur stackoverflow afin d'utiliser la fonction `centered()` qui résout ceci.

Utilisation des flèches pour les déplacements

> Pour jouer avec les flèches directement, on a utilisé la bibliothèque `<conio.h>` , grâce à la fonction `getch()` qui permet de stocker le code des flèches dans une variable userinput pour définir le déplacement sui sera éxécuter.

Éviter les overlaps posés par #include

> Utilisation de `#pragma once` qui permet de copier le code une seule fois.

Utilisation des makefiles

Il fallait chercher tout sur les makefiles, comprendre la syntaxe et la respecter. en effet, la commande 'make' marche seulement sur linux, pour l'utiliser dans le terminal de windows, il fallait l'installer et rajouter son chemin au PATH.

SFML et interface graphique

Pour rendre hommage aux créateurs, nous avons décidés de maintenir le thème classique du jeu. pour cela nous avons debugger le jeu original afin d'éxtraire le couleurs et les dimensions précises utilisées dans celui-ci, et ceci à l'aide du Code source, des fichiers CSS et JSON.

Lors de l'affichage des valeurs des tuiles, tout s'affichait toujours à gauche; Après une recherche dans la documentation de l'SFML, nous avons exploité la fonction `floatrect` qui permet de definir les bornes du texte (un margin rajouté par défaut) selon 4 paramétres (margin left and top, height and width) qui permet de le positionner le texte au milieu avec addition et soustraction de la longueur, margin etc.