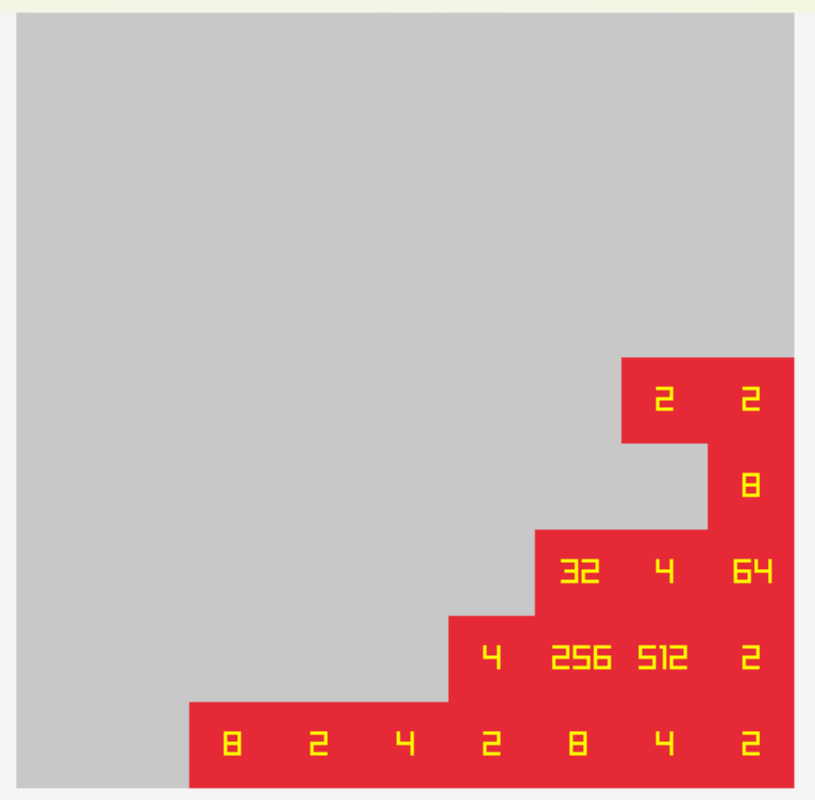
2048 Plus

Projet d'IF2B

FOUICH Noa, SOARES Kenneth



capture d'écran du mode de jeu solo de notre version de 2048 plus

Table des matières

Introduction **3**
Présentation des choix techniques 3
Présentation de la structure du projet 3
Présentation de la librairie choisie pour la partie graphique du projet 3
Présentation du système de compilation 4
Présentation du système de contrôle de version (VCS) 4
Structure détaillée du code **5**
Structure de la partie jeu du code 5
Présentation des fonctions de déplacement 5
Présentation de la fonction de vérification 5
Présentation de la fonction de placement aléatoire d'une nouvelle valeur 6
Présentation de la fonctionnalité permettant de jouer un tour 6
Présentation de la partie extrac_csv du code 6
Présentation de la fonction d'existence d'un fichier 6
Présentation de la fonction de calcul de taille de la grille 6
Présentation de la fonction d'extraction de grille à partir d'un fichier csv 7
Structure détaillée de la partie graphique du code 7
Présentation de la création de la fenêtre 7
Présentation du système d'écrans du jeu 7
Présentation de la fonction d'affichage de la grille 8
Présentation de la partie principale du code 8
Bilan de réalisation du projet **8**
Facilités rencontrées dans la réalisation du projet 8
Connaissance préexistante des outils utilisés 8
Difficultés rencontrées dans la réalisation du projet 9
Communication et organisation 9
Points d'amélioration du projet 9
Utilisation de realloc() dans la fonction main() 9
Gestion de la taille de la fenêtre 9
Mise en place d'une interface graphique plus attrayante visuellement 10
Utilisation des outils à notre disposition pour maximiser la communication et l'organisation 10

Introduction

Dans le cadre de l'UE IF2B, où nous avons pu apprendre à programmer dans le langage C, nous avons réalisé un projet en binôme. Le projet que nous avons choisi de mener est le jeu du 2048 avec une interface graphique et ses 3 modes de jeu.

Présentation des choix techniques

Présentation de la structure du projet

Pour réaliser le projet de 2048 nous avons découpé notre code en 4 fichiers sources .c principaux, le fichier game.c gère toutes les fonctions liées aux déplacements, à l'initialisation d'une grille vide en début de partie, à la vérification de défaite, à la détection d'appui de touche et de jeu de tour, à l'ajout de valeurs aléatoires dans les cases vides de la grille. Le fichier graphics.c gère toute la partie graphique du projet avec la gestion du menu principal et du système d'écrans, l'affichage des matrices simples et doubles avec prise en charge de la contrainte du bloc X dans le mode de jeu puzzle et la fonction de création de la fenêtre. Le fichier extrac_csv.c gère la partie du projet lié au calcul de la taille de la grille à charger dans le fichier csv et l'extraction de la grille du fichiers csv vers un tableau à 2 dimensions. Enfin, le fichier main.c gère l'appel et a toutes les autres fonctions des autres fichiers et contient notamment le point d'entrée du programme.

Présentation de la librairie choisie pour la partie graphique du projet

Ayant choisi de réaliser le projet de 2048 avec une interface graphique nous avons dû choisir un module permettant de réaliser une interface graphique. Afin de respecter les contraintes de temps imposées, nous avons écarté les choix offrant un plus grand contrôle sur l'aspect performance tels que les API bas-niveau d'OpenGL comme GLAD et GFLW. De même, nous avons écarté SDL car les avantages que cette librairie offrait étaient négligeables en comparaison avec la redondance syntaxique imposée par celle-ci. Enfin, nous avons déjà utilisé Raylib pour un projet personnel en C. Nous avons donc choisi de baser l'interface graphique de notre projet dessus.
Raylib est une librairie très simple d'utilisation, basée sur OpenGL elle ne nécessite aucune dépendance supplémentaire ce qui facilite le processus de compilation que nous allons aborder par la suite. Avec Raylib nous pouvons nous concentrer sur le contenu à afficher sur l'écran et non sur l'écriture de shader, programmes permettant de donner des informations à la carte graphique sur la façon d'afficher nos images etc. Elle prend également en compte la gestion d'acquisition des touches au clavier et un module pour les sons que nous n'avons pas utilisé dans notre projet. Enfin il est important de noter que Raylib, en plus d'être un module facile d'utilisation et performant, possède également une bonne documentation qui peut être synthétisée à l'aide d'une seule page du site, la cheat sheet.

Il peut être utile de noter que nous utilisons la version 4.5 de Raylib dans notre projet.

logo de Raylib

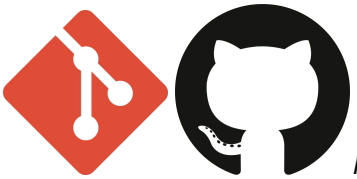
Présentation du système de compilation

Nous avons écrit l'entièreté du code de ce projet sous Linux, plus précisément la dernière version de PopOs! Nous avons donc pris soin d'éviter d'utiliser des librairies non cross-plateformes telles que GNU readline. De plus, pour faciliter la compilation d'un système d'exploitation à un autre, nous avons choisi un système de compilation cross-plateforme appelé CMake. CMake génère des Makefile, fichiers de compilations pouvant s'adapter au compilateur présent sur notre machine (GCC, clang, etc.). L'avantage de CMake réside dans la flexibilité de la syntaxe du langage de rédaction de fichier de compilation. En effet, on a pu faire en sorte de récupérer une archive de Raylib 4.5 dans notre fichier CMakeLists.txt et de la lier dans le Makefile généré. Cela signifie que tout utilisateur souhaitant compiler le jeu a partir du code source n'aura pas besoin d'avoir Raylib d'installé globalement sur sa machine, ni aura besoin de récupérer l'archive par lui-même.


logo de CMake

Présentation du système de contrôle de version (VCS)

Nous avons choisi d’utiliser Github pour partager nos modifications sur le code et ainsi toujours travailler sur une version à jour de celui-ci. Nous avons donc utilisé la ligne de commande pour une bonne partie de notre travail afin de récupérer (pull) les changements apportés au code et mettre à jour (commit) la version modifiée du code. Il y avait d’autres alternatives telles que gitlab, mais nous avons choisi Github car nous sommes tous les deux familiers avec ce site. Une majeur partie des sites de VCS est utilisable avec le logiciel en ligne de commande git que nous avons utilité


logo de git (gauche) et Github (droite)

Structure détaillée du code

Structure de la partie jeu du code

Pour prendre en compte la contrainte du mode de jeu puzzle et faire des fonctions de déplacements adaptables, nous avons défini un enum VALEUR permettant de définir la valeur X comme étant égale à -1.

Présentation des fonctions de déplacement

Dans nos fonctions de déplacements, nous avons défini à chaque fois des variables curseur et cible afin de savoir quelle valeur nous traitons et avec quelle valeur nous essayons de la fusionner. Pour chaque déplacement nous avons donc défini un cas ou la valeur que nous traitons est égale à X, dans ce cas la, nous décalons notre curseur et la cible d'un emplacement à gauche, à droite, en haut ou en bas selon la fonction de déplacement en question. Dans le cas où la valeur de la case où on souhaite effectuer le déplacement est nulle, on effectue simplement le déplacement en remplaçant sa valeur par celle de la case que l'on traite. Dans le cas où la valeur de la case que l'on traite et la valeur de destination sont égales, on additionne les deux valeurs dans la case de destination et on annule la valeur de la case d'origine et on décale d'un emplacement. Dans le cas où la valeur d'origine et la valeur de destination sont différentes, on déplace les valeurs des deux cases sans les fusionner. La fonction contient également une variable d'état "changement" qui est mise à 1 à chaque fois qu'un déplacement est faisable et effectué et retournée à la fin de l'exécution de la fonction.

Cette variable d'état sera utile pour la fonction de vérification qui va nous permettre de savoir si la partie est terminée où non, cette fonction sera abordée plus tard dans le rapport.

À savoir que ces fonctions de déplacements sont les plus importantes du jeu et ont été pensées de façon à éviter les doubles boucles sur tableau à 2 dimensions limitant considérablement la fluidité du programme.

Présentation de la fonction de vérification

La fonction de vérification permet de savoir si le joueur a perdu ou s' il peut continuer à jouer. Pour ce faire, on réalise une copie de la grille et on teste si l'un des 4 déplacements peut être effectué à l'aide de leurs valeurs de retour. Si l'un des déplacements peut être effectué, la fonction vérification retournera 1, elle retourne 0 dans le cas échéant.

Présentation de la fonction de placement aléatoire d'une nouvelle valeur

Dans un premier temps, la fonction itère dans chaque case de la grille pour compter le nombre de cases à la valeur 0. Si le nombre de cases vides est égal à 0, la fonction retourne alors 0.
On utilise ensuite la fonction *rand()* pour déterminer la valeur à insérer, soit 2 dans 80% des cas, soit 4 dans 20% des cas.
On initialise une variable de distance_indice qui permet de savoir quelle case vide on va cibler pour ajouter notre valeur aléatoire.
on itère une dernière fois dans notre grille en décrémentant distance_indice à chaque case vide rencontrée et lorsque la distance_indice est égale à 0, on place notre valeur aléatoire à la case à laquelle on se trouve.

La fonction initialisation du tableau remplit le tableau de case vides et fait appel à la fonction de placement aléatoire d'une nouvelle valeur.

Présentation de la fonctionnalité permettant de jouer un tour

La fonctionnalité permettant de jouer un tour est décomposée en 2 fonctions:

la fonction *jouer_entree()*, permet d'effectuer un des 4 déplacements si un des 4 flèches est pressée et retourne 1 si le déplacement à été effectué, sinon 0.

la fonction *jouer()*, vérifie si un tour à été joué, si c'est le cas, elle vérifie que le joueur n'a pas encore perdu, et si il n'a pas encore perdu, un appel à la fonction permettant d'ajouter une valeur aléatoire est effectué. La fonction retourne 1 si un tour à été joué et 0 dans le cas contraire.

Présentation de la partie extrac_csv du code

Présentation de la fonction d'existence d'un fichier

La fonction d'existence d'un fichier permet de s'assurer que l'utilisateur ne demande pas au programme de charger une grille dans un fichier csv qui n'existe pas.
elle teste la fonction *fopen()*, et renvoie 1 si le fichier existe, 0 si le fichier n'existe pas.

Présentation de la fonction de calcul de taille de la grille

La fonction de calcul de la taille de la matrice lit la première ligne du fichier avec la fonction *fgets()*, on récupère le premier caractère de la ligne (la taille de la matrice), et on convertit en char[] pour ensuite utiliser *atoi()* et récupérer un int qu'on va renvoyer.

Présentation de la fonction d'extraction de grille à partir d'un fichier csv

La fonction d'extraction lit chaque ligne du fichier à l'aide de la fonction *fgets()*, et itère entre chaque valeur à l'aide de la fonction *strtok()*, pour chaque 'x' rencontré dans le fichier, matrice[i][j] est affecté à X (de l'enum valeur avec X=-1). pour chaque autre valeur, 0 ou puissance de 2, matrice[i][j] est affectée.

Structure détaillée de la partie graphique du code

Présentation de la création de la fenêtre

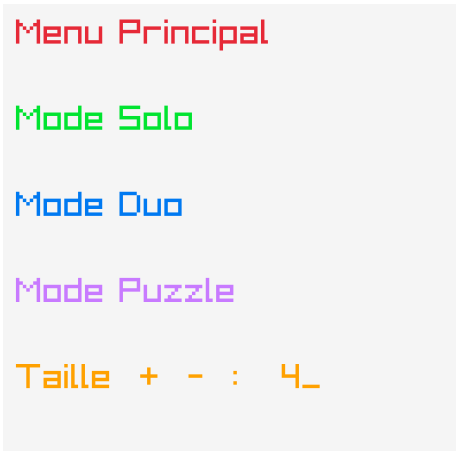
La librairie Raylib étant structurée sur l'élimination de la syntaxe redondante, la fonction de création de la fenêtre est très simple puisqu'elle n'est constitué que d'un appel aux fonction *Init_window()* et *SetTargetFPS()* permettant de créer une fenêtre aux dimensions passées en paramètre et au nombre d'image par seconde spécifié également en paramètre, 60 dans notre cas, bien que dans le cas d'un 2048, bloquer le nombre d'images par seconde ne soit pas forcément indispensable.

Présentation du système d'écrans du jeu

L'une des contraintes importantes du projet était la présence de plusieurs modes de jeu. Cela signifie que pour la partie graphique, nous avons dû inclure dans notre code une façon de choisir le mode de jeu à jouer. Nous avons donc opté pour un menu principal graphique avec un système d'écrans assez simple. Le système d'écran est simplement basé sur une énumération avec différents écrans: MENU, SOLO, DUO et PUZZLE pour chaque mode de jeu ainsi que le menu. Dans notre fonction *dessin_fenetre()*, appelée après la fonction *creation_fenetre()*, nous avons donc simplement eu à utiliser une instruction switch pour chaque écran afin d'appeler les fonctions correspondantes.

Il est important de noter que nous avons également implémenté le choix de la taille de la grille ainsi que le champ d'entrée du nom du fichiers CSV à charger dans le mode de jeu puzzle, directement dans le menu principal afin d'éviter de rajouter un écran intermédiaire et faciliter la distinction entre l'écran de jeu et le menu principal.

Enfin en cas de défaite, le retour au menu se fera aussi dans la fonction *dessin_fenetre()*.



capture d'écran du menu principal, modifié dans la version finale du projet pour accueillir le champs d'entrée du fichier .csv à charger pour le mode puzzle

Présentation de la fonction d'affichage de la grille

La fonction d'affichage de la grille fonctionne en itérant sur chaque case de la grille et en ayant un cas par type de valeur. Si la valeur est nulle, on dessine une case grise vide, si la valeur est X, alors on dessine une case magenta avec un caractère "X" en violet. On rappelle que X fait partie de l'enum VALEUR et vaut -1, il faut donc bien faire la distinction entre le type de "X"et de X présent dans la grille (VALEUR). Dans le cas d'une valeur autre que X ou 0, on convertit la valeur en char* à l'aide de la fonction *sprintf()*, on dessine un carré rouge et on affiche le nombre en jaune par dessus. Sachant que le nombre peut être constitué de plusieurs chiffres, il convient d'adapter la taille de la police au nombre de chiffres dans la valeur. Pour ce faire, on utilise la formule suivante:

$$n = \lfloor \frac{ln(N)}{ln(B)} + 1 \rfloor$$

avec n le nombre de chiffres de N en base B.

Pour le mode de jeu duo, nous avons réalisé une fonction simple *affichage_double_matrices()* qui appelle la fonction *affichage_matrice()* pour deux matrices différentes de taille donnée.

Présentation de la partie principale du code

Le fichier main.c est la partie principale du code puisqu'elle contient pour seule fonction le point d'entrée du programme où une taille par défaut de 4 est définie et 2 tableaux a 2 dimensions sont créés et remplis avec une valeur aléatoire. On crée également la fenêtre du programme dans cette fonction. Si la taille à changé, on libère la mémoire de nos grilles et on leur alloue la mémoire nécessaire en fonction de la nouvelle taille. Si le mode de jeu est solo ou puzzle, on appelle la fonction de jeu de la partie game.c pour une seule des deux grilles, sinon, si le mode de jeu choisi est duo on appelle la fonction pour chacune des deux grilles. À la fin de l'exécution des fonctions de jeu, on libère la mémoire allouée aux deux grilles et on ferme la fenêtre.

Bilan de réalisation du projet

Facilités rencontrées dans la réalisation du projet

Connaissance préexistante des outils utilisés

Compte tenu de notre connaissance préexistante du langage C ainsi que de la librairie Raylib et des outils de compilation tels que CMake, le projet n'était pas trop exigeant d'un

point de vue technique pour notre binôme. Nos connaissances respectives ont permis de résoudre les points de blocages de l'un et de l'autre dans la réalisation de certaines des tâches plus complexes par exemple dans les fonctions de déplacements qui ont été modifiées plusieurs fois afin d'atteindre leur état actuel ou encore la fonction de calcul de taille d'une grille.

L'utilisation de git et de la ligne de commande nous à ouvert un champs de possibilité pour la gestion du code bien plus étendu que si nous ne connaissions pas ces outils.

L'uniformisation de nos environnements de développement à également été un choix qui nous à été grandement bénéfique dans le partage de bouts de codes test via nos canaux de communication.

Difficultés rencontrées dans la réalisation du projet

Communication et organisation

Le plus gros point de difficulté rencontré durant l'élaboration de ce jeu de 2048 à été sans aucun doute l'organisation. plusieurs fonctions ont dû être entièrement réécrites dû aux manques de communication vis à vis de l'organisation des fonctions que nous avons écrites.

Nous aborderons dans la dernière partie du rapport, la manière dont nous pourrions nous améliorer de ce point de vue.

Points d'amélioration du projet

Utilisation de realloc() dans la fonction main()

Dans la fonction point d'entrée du programme, nous allouons de facto de la mémoire pour les deux grilles pour une taille par défaut de 4, puis nous libérons la mémoire et la allouons pour une autre taille, si la taille à changé. À la place de libérer la mémoire et de la réallouer nous aurions pu utiliser la fonction realloc() qui permet de gérer ce changement de taille sans complètement libérer la mémoire des grilles.

Gestion de la taille de la fenêtre

Dans l'entièreté de notre jeu nous avons défini dans notre code les dimensions de notre fenêtre comme des constantes (1280x720, HD en 16:9). Il aurait pu être intéressant de rajouter un menu nous permettant de modifier la taille de la fenêtre en plus de modifier potentiellement d'autres paramètres de jeu comme d'éventuels type de contrôle (Z,Q,S,D au lieu des flèches, etc)

Mise en place d'une interface graphique plus attrayante visuellement

Nous n'avons pas utilisé d'images dans notre projet pour faciliter le processus d'écriture du code. Cependant cela à déteint sur l'aspect global de notre jeu, un peu trop simpliste et du coup moins attrayant visuellement. Bien qu'il ne s'agisse pas de l'aspect le plus important d'un jeu vidéo de ce type, il est important de garder en tête l'importance de visuels intéressants dans des projets à but commercial.

Utilisation des outils à notre disposition pour maximiser la communication et l'organisation

Comme nous l'avons abordé précédemment, l'organisation et la communication ont été un gros point de blocage dans notre groupe notamment au niveau de l'établissement de conventions ou de partage d'information. Nous aurions pû utiliser les outils à notre disposition pour palier partiellement à ces problèmes.

Premièrement en utilisant le fichier README.md, nous aurions pû instaurer des règles au niveau des conventions de nommage (pas de franglais, standard de nommage de variable etc.) mais également des pratiques à respecter comme le fait de push sur github tout changement ou encore de mettre un message de commit descriptif.
Nous aurions également pû utiliser Github Issues pour savoir quelles tâches ou bugs étaient toujours d'actualité ou encore Trello pour savoir comment répartir correctement le travail et éviter les dupliqués.

Enfin nous aurions pu utiliser un calendrier en ligne afin de nous fixer des deadlines et limiter les périodes à efficacité réduite qui nous ont retiré du temps d'optimisation et d'amélioration graphique.

En complément, il aurait pu être utile de rajouter dans le fichier README.md une forme de documentation succinte du code, en plus des commentaires déjà présents dans celui-ci, pour comprendre certaines décisions moins intuitives mais plus optimales.

logo de la licence GPL 3.0