

PROJET DE PROGRAMMATION C

COMPRESSION D'IMAGES AVEC DES QUADTREES
SDAF | RAVENDRAN

SOMMAIRE :

OBJECTIFS ET ENJEUX

PROJET ARCHITECTURE

MÉTHODES EMPLOYÉES

DIFFICULTÉS RENCONTRÉS

GIT REDMINE

I. OBJECTIFS ET ENJEUX

L'objectif de ce projet est de développer une application graphique pouvant encoder, décoder et compresser des images en utilisant des **quadtrees**. On utilisera les notions de manipulation de fichier, d'image mais aussi des différents pixel d'un image grâce à la bibliothèque graphique **MLV**.

II. PROJET ARCHITECTURE

L'exécutable **quad** se génère en compilant avec le fichier **Makefile** avec la commande **make** puis on l'exécute avec la commande **./quad**.

Pour effacer tout les fichiers indésirables **.o**, faire la commande **make clean**.

Nous avons décidé de découpé notre projet en **7 modules** différents (les **.c** suivi de leur fichiers **.h** respectifs), suivi du **main.c**, puis du **Makefile** avec les inclusion de chaque fichier:

- **menu** : contient toutes les fonctions pour afficher les images et les fichiers et choisir notre lancement du jeu, ce fichier inclut le fichier **quadtree.h**;
- **button**: contient les fonctions pour cliquer sur un sauvegarder notre image en **qtn** ou **qtc**, inclut le fichier **quadtree.h**;
- **image** : contient toutes les fonctions manipulant les images, inclut le fichier **quadtree.h**;
- **queue** : contient les fonctions de création, d'ajout et de suppression d'un élément dans une file, inclut le fichier **quadtree.h**;
- **quadtree** : correspond au fonctions manipulant l'arbre quadratique;
- **compress** : correspond au fonctions de compression d'un arbre donné, inclut les fichiers **queue.h** et **image.h**;
- **game** : contient la fonction qui lance le jeu, inclut les fichiers **menu.h**, **image.h**, **button.h** et **compress.h**;
- **main** : fichier principal pour le lancement du programme, inclut le fichier **game.h**.

III. MÉTHODES EMPLOYÉES

Pour le bon déroulement de ce projet, nous avons eu recours à la documentation de la [bibliothèque graphique MLV](#) :

Nous avons 3 variables définies, dont une pour la hauteur de la fenêtre à 700 pixel, puis pour la largeur de la fenêtre à 512 pixel mais aussi pour la taille d'une image qui sera de 512x512 pixel :

```
#define WIDTH_WIN 512
#define HEIGHT_WIN 700
#define IMAGE_SIZE 512
```

Nous avons utilisées plusieurs structures différentes :

- Module quadtree

```
typedef struct node{
    int red;
    int green;
    int blue;
    int alpha;
    struct node *NO, *NE, *SO, *SE;
} Node, *Tree;
```

Cette structure définit un arbre appelé **Tree** en fonction des ses nuances de couleur de type int: **rouge**, **vert**, **bleu** et **alpha** pour la transparence. Elle définit aussi une liste chaînée contenant les nœuds **NO**, **NE**, **SO** et **SE**.

- Module queue

```
typedef struct element {
    Tree tree;
    struct element* next;
} Element, *Queue;
```

Cette Structure définit une file appelée **Queue** en fonction d'un arbre de type **Tree** et définit une liste chaînée avec un élément suivant appelé **next**.

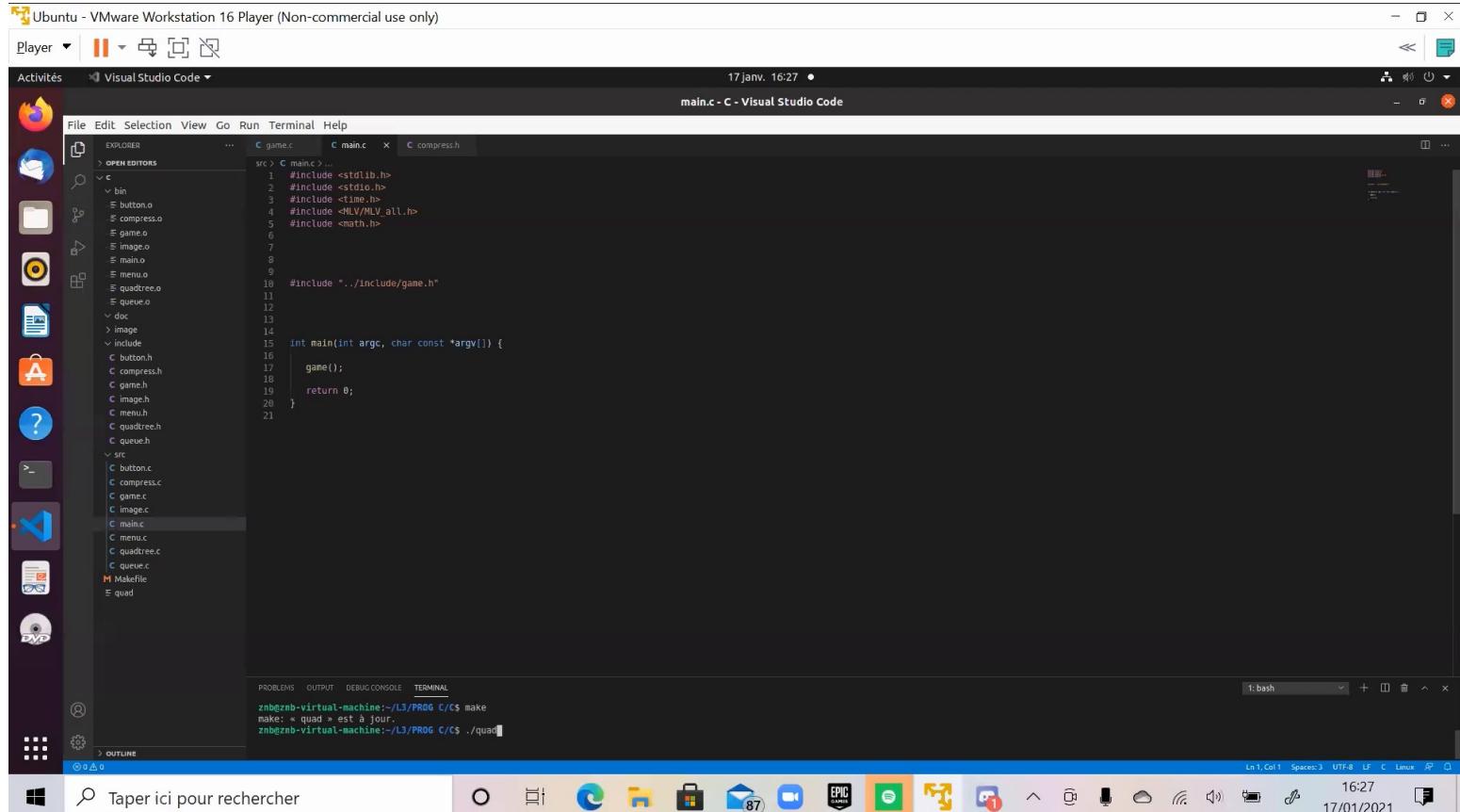
Une fois le programme lancé sur le terminal, on sera à présent devant le menu de notre jeu :

Nous avons alors le choix entre compresser un arbre à partir d'une image ou à partir d'un fichier quadratique **.qtn** en noir et blanc (binaire) ou **.qtc** en couleur.

Après avoir fait notre choix, l'image est affiché en fonction de l'arbre et le programme attend un clique de l'utilisateur pour commencer la compression.

A la fin de la compression, le programme attend un clique pour faire la sauvegarde en **qtn** ou en **qtc** de l'image compressé.

Voici une démonstration vidéo de notre projet :



IV. AMÉLIORATION

- Vitesse d'affichage

On peut changer le taux de rafraîchissement (la vitesse de l'affichage) de l'image générée petit à petit en fonction de la variable définie **TAUX_R** dans le fichier **compress.h**.

- Affichage banque d'images

Au début de notre programme on a l'affichage de toutes les images contenu dans notre dossier, ce qui rend notre projet assez sympathique.

V. DIFFICULTÉS RENCONTRÉS

Nous avons mis du temps à comprendre le sujet car il a fallu nous renseigner sur les **quadTree**, mais aussi comprendre les explications donné dans le sujet. Une fois que nous avons compris comment cela marchait, nous avions pris du temps à visualiser le comportement de la compression pour faire une parcours en largeur au lieu de faire un parcours en profondeur et de compresser plusieurs fois un même arbre. Nous avons pas fini le projet notamment la partie de la minimisation.

VI. GIT REDMINE

Nous avons eu du mal à créer notre projet dans l'interface **Redmine** et nous avons pas vu l'intérêt de cet usage car nous avons toujours coder ensemble en faisant des partage d'écran.

Nous avons ensuite décidé de faire un commit chacun notre tour dans le **Redmine**, même si parfois on ne pensait pas à faire un commit. On a aussi eu du mal à gérer les **commit**, **add**, **push** et les **pull** au début de l'utilisation.