Osorio Thomas-TDH Rapport Problème Scientifique Informatique

I/Structure de données

Le Programme est constitué de 4 classes principales (sans prendre en compte l'innovation) : MyImage, Pixel, QRcode, Histogramme. Une cinquième est une classe de type statique nommée Tools. Chaque classe a été construite pour favoriser au maximum la programmation orienté objet.

Tout d'abord parlons de la classe MyImage. C'est la classe centrale et principale du projet. En effet c'est dans celle-ci que l'on créer une image au format BMP, on la traite ou encore on l'affiche. Elle possède 6 attributs. bfSize est le champs correspondant à la taille de notre fichier au format bmp. Les deux attributs suivants sont les caractéristiques de dimensionnement de l'image, biHeight et biWidth(respectivement la largeur et la longueur de l'image). D'après la donnée du cahier des charges, nos pixels sont codés sur 24 bits, ce pourquoi la variable nbBitcount est défini comme static car commune à toutes les instances de type MyImage. Nous terminons notre liste d'attribut par notre matrice de type de Pixel, qui est en réalité notre Image traitée. Cette classe possède 4 constructeurs. L'un d'entre eux permet de lire une image déjà existent et de l'instancier au type MyImage. Ainsi il possède une entrée de type string qui sera le nom du fichier au format BMP à lire. Un deuxième constructeur permet de créer une fractale de MandelBrot aux dimensions souhaitées de type MyImage. De même pour créer un Histogramme d'un objet de type MyImage, il était nécessaire d'avoir un autre constructeur qui possède une entrée MyImage. Pour finir le dernier permet de créer des images de type QRcode. La taille de ce-dernier différentes méthodes de remplissage des patterns notamment a été créer, ainsi les étapes principales de la création de cet objet sont définies et lisibles. A des fins de lisibilité j'utilise aussi des const pour éviter les magic number. La méthode From_Image_To_File() permet d'écrire les données d'un objet de typer MyImage au format BMP dans le dossier DEBUG. Elle prend donc en entrée le nom du fichier et est de type void. Cette méthode et le constructeur MyImage à partir d'une image existante font tous les deux appels aux méthodes : Convert_Int_To_Endian et Convert_Endian_To_Int qui sont donc nécessaires pour convertir les tableaux d'octets en int et vice-versa en respectant la lecture du little-endian.

Les méthodes qui ont nécessité le plus de travail sont l'agrandissement, la rotation, la stéganographie et La convolution. Pour l'agrandissement, j'ai fait le choix de l'interpolation bilinéaire. Une méthode qui pour les petites Images peut paraitre flou et inutile. Elle se révèle nécessaires pour des images suffisamment grandes. En effet contrairement à la méthode du plus proche voisin par exemple, elle permet d'effectuer une moyenne des pixels voisins et de construire ces nouveaux pixels de manière plus approprié. Son unique entrée est le coefficient d'agrandissement et elle modifie directement les données de l'image, ce pourquoi elle est de type void. Le rétrécissement n'est que la réciproque de l'aggrandissement. Pour plus de possibilité il est possible de rentrer des nombres et demi. Ma rotation est des plus simples, je n'ai pas réussi à en effectuer une sans rognage. Ainsi cette rotation possède une perte de donnée et prends en entrée l'angle de rotation. Comme pour l'agrandissement elle est de type void. Une autre tâche du cahier des charges est la stéganographie, cette partie possède 2 méthodes. Une permet de cacher une image dans une autre. Pour cela on prend les bits de poids forts de l'image principale et les bits de poids faible de l'autre. Ces derniers deviennent les bits de poids faibles de l'image final tandis que les premiers restent des poids forts. Pour reconstruire les deux images on effectue l'inverse. Néanmoins les bits de poids faibles des deux pixels reconstruits seront nuls, on les initialisera donc à 0. On observe donc que la reconstruction n'est pas

parfaite mais on reconnait toutes les formes. Les entrées pour cacher une image est donc un objet MyImage et pour la décoder le nom du fichier à décoder (noter que dans le wpf vous ne pouvez décoder que des images melanger dans l'application). Une autre méthode intéressante est celle de Convolution. Les consignes ont été respecter quant à la généricité du code. Toutes les opérations de filtrage sont réalisées par cette unique méthodes, ce pourquoi elle prend en entrée une matrice de int, qui correspond en réalité au filtre appliqué. Tous les pixels vont être filtre à l'aide d'une méthode de filtrage contenu dans la classe Pixel, cette dernière va effectuer les opérations sur le pixel cible. Pour le traitement des bords, le pixel le plus près est étendu. Les pixels en coin eux sont étendu à 90°. Les méthodes concernant le QRcode sont juste des méthodes pour remplir une image au format QR. A propos de l'effet miroir, j'ai longtemps hésité entre doubler l'image ou un effet miroir classique, cependant je me suis référé aux logiciels que je connaissais qui inverse le sens de l'image.

Comme expliqué précédemment, l'attribut de l'image est un tableau multidimensionnel de type Pixel. Chaque Pixel est codé avec 3 couleurs, le Rouge, le Vert, le Bleu (RGB). Ce pourquoi les pixels possèdent 3 attributs de type entier R, G, B. Un quatrième attribut nous dit si c'est un pixel qui est en cache un autre ou non, Cache est donc de type booléen. La classe Pixel possède 2 constructeurs. Un constructeur construit à partir de nombres entiers de RGB et qui initialise cache à false. Enfin le deuxième correspond à un constructeur de copie qui possède donc en entrée un autre Pixel. Les getters et setters ont été créer pour tous les attributs. Les deux méthodes présentes sont celle permettant de réaliser une interpolation sur des pixels ainsi que du filtrage et on était décrite précédemment.

La troisième classe traitée est la classe Histogramme. Cette classe permet d'obtenir l'histogramme d'une image BMP. Ainsi on peut avoir plus facilement les Histogrammes de la couleur Rouge, Vert, Bleu ainsi que celui de la moyenne (en noir). Ces trois valeurs sont des tableaux de int. La classe possède un constructeur qui permet d'instancier un Histogramme à partir d'une matrice de Pixel (une Image). Les propriétés permettent d'accéder à ces tableaux. Son unique méthode permet d'obtenir un entier représentant la valeur maximale stocker dans les 3 tableaux.

Dans la dernière partie du projet, la classe QRcode a été nécessaires pour pouvoir créer une image de type QRcode. Comme vu dans le cahier des charges, le QRcode possèdent différents bits de données pour pouvoir être affiché. Le premier attribut est logiquement la phrase alphanumérique en string. On trouve aussi des informations sur sa version, la correction utilisée (Au début je comptais faire pour les 4 corrections mais je n'ai pas eu le temps, cependant certaines méthodes pour définir les nombres de bits de données sont présents). Sizebit représente la taille totale des données (sans la correction d'erreurs). Pour éviter les Magic Numbers le nombres de caractères pour une version, le nombre de bits pour coder une paire de lettres ou encore le nombre de bits pour coder la taille de la chaine de caractères sont de type const. Les deux tableaux de bool TOFILL1 et TOFILL2 sont les terminaisons pour remplir les données, ce pourquoi ils sont constants. De même chaque tableau de bool correspond à une partie du QRcode et permet de séparer correctement les données. L'addition de tous ces tableaux ainsi que la correction d'erreur permet d'avoir une List de booléen, correspondant aux données. En d'autres mots les bits sont codés par des booléens, false étant 0 et true 1. Le projet demandait un lecteur et un générateur de QRcode, ainsi la classe possède deux constructeurs. Le premier est donc un générateur. On peut le couper en 9 étapes distincte. Les 4 premières étant le codage des informations sur la chaîne, taille, version, mode(alphanumérique). Arrivé à la $5^{ème}$ étape on lit les lettres par pairs. On distingue donc les chaînes paires et impaires (ce dernier cas traitera une lettre seule qui sera codé sur 6 bits au lieu de 11). Ensuite on remplit la chaine par une terminaison de 0 allant d'une taille null à 4. L'étape 7 permet d'additionner des 0 jusqu'à obtenir un nombre de données multiple de 8(ainsi nous obtenons une chaine de 19 octets pour la version 1/L par exemple).

La dernière étape est la correction d'erreur faisant appel à la méthode Encode de la classe ReedSolomonAlgorithm (fournit par le professeur). Celle-ci génère le message de correction d'erreur qui viendra compléter nos données. Le deuxième constructeur est un lecteur de QRcode qui n'est juste que la réciproque de celui-ci-dessus. (Dans WPF pour lire un QRcode mask 0, il faut qu'il soit dans le fichier debug et le Qr code crée est enregistré dans le fichier Monqrcode.bmp)

Les méthodes de cette classe QR sont essentiellement là pour alléger le code des constructeurs, donc toutes des voids. On y trouve 2 « bibliothèques » qui permettent de faire le lien entre le caractère lu et sa valeur numérique. Une autre renvoie le masque 0 (minimum demandé). NbBitQR permet de définir le nombre d'octet de données et de correction d'erreur. Elle n'est certes pas très utile dans le projet pour la version L mais dans une optique plus généraliste elle aurait été nécessaire (si on avait codé tous les types de correction). La méthode suivante permet de calculer la valeur d'une paire. La dernière permet de réaliser la correction d'erreur correctement en séparant notre chaine de bits en octet et réaliser l'appel de ReedSolomon. Pour finir on peut noter que l'on peut rentrer des textes en minuscules qui seront automatiquement retranscrit en majuscule.

La classe Complexe permet quant à elle de réaliser des opérations sur les complexes lors de la création des fractales de MandelBrot.

La dernière classe créer dans la partie « classique du projet » est la classe static Tools qui, comme son nom l'indique permet l'accès à une base d'outils, c'est-à-dire des opérations nécessaires dans différentes classes. On y retrouve principalement des opérations de conversion ou encore de puissances. Les deux méthodes int_to_bit et bool_to_bit permettent d'écrire des nombres entiers en binaire. On rentre donc soit la chaine, soit l'entier ainsi que le nombre de bits sur lequel on veut que ce chiffre soit codé. La fonction puissance permet quant à elle de calculer une puissance et ne pas passer par la bibliothèque Math.

II/ Innovation

Mon innovation se porte autour du sujet de la compression JPEG. J'ai voulu essayer une compression complète d'un fichier JPEG, cependant par manque de temps je n'y suis pas arrivé.

Comme discuté lors de la revue de code, j'ai réussi à mettre en place un algorithme de Huffman. Pour résumer lorsque l'on rentre un mot, on obtient sa suite binaire. Mon algorithme se base en 3 étapes principales. La première consiste à trier dans l'ordre croissant des fréquences de chaque symbole. Dans un second temps, on prend les deux premiers éléments de la liste donc ceux qui sont les moins grandes fréquences, on additionne leurs fréquences et on « assemble » leurs symboles. On attributs à cette ensemble une valeur de gauche (bits 0) qui sera la valeur du premier et une de droite qui sera la valeur du deuxième (bits 1). On réalise cet algorithme jusqu'à prendre en compte toutes les valeurs. Pour coder la chaine de bits, on réaliser donc l'inverse en remontant les branches de l'arbre d'Huffman pour chaque caractère du texte entré. Ce pourquoi on utilise un algorithme récursif. Il va lire la branche gauche et la droite pour chaque symbole et va remonter au fur et à mesure en prenant le coté droite ou gauche et quand les deux côtés seront terminés la fin de la récursivité en renvoyant le code final du caractère visé. Notons que chaque symbole parent ou nœud de la branche contient lui-même 2 symboles soit un autre nœud, soit un symbole recherché.

Concernant le JPEG, j'ai laissé une classe inactive du même nom qui montre le début de la construction. En premier je réalise un sous echantillonage 4 :2 :2. Par la suite je traite mon Image par bloque de 8 pixels. A chaque itération, je réalise une opération de DCT puis une quantification à l'aide

de la matrice de quantification de la classe. L'un des problèmes rencontrés venait de la lecture en zig-zag de ma matrice que je savais fausse. Pour finir la compression il me manque donc l'implémentation du code de Huffman qui consiste juste à rentrer ma chaine de int lu en RLE au format string et la dernière étape, l'écriture de mon fichier JPEG.

III/ Auto-Critique :

Pour conclure, je pense rendre un projet assez complet. En effet la majorité des tâches demandées ont été réalisées. Le côté négatif et ma méthode la moins efficace est probablement la rotation d'Image. J'ai passé du temps à essayer d'éliminer le problème du rognage sans jamais vraiment réussir. Je n'ai pas attaché une grande importance à cette méthode car j'ai préféré avancer sur d'autres sujets. Un autre aspect que j'aurai pu améliorer, est l'affichage du QR Code, agrandir l'image aurait été bien, même si lorsque je l'importe sur une application portable, le code est lu correctement. Ne pas avoir terminer la compression JPEG est aussi une déception car je pense qu'avec légèrement plus de temps j'aurai pu terminer. Cependant, travailler sur ce sujet m'a permis d'apprendre énormément de chose en mathématique avec la DCT mais aussi en algorithmie avec l'algorithme de Huffman.

Les points positifs de mon code sont la généricité de la méthode de convolution ainsi que l'interpolation bilinéaire qui m'a demandé beaucoup de travail. J'ai aussi consacré une grande partie du temps sur le QRCODE. Le plus difficile a été l'affichage de ce dernier, notamment le lien avec le masque que je n'avais pas bien compris au départ. De plus le lecteur QRCode est réussi. La programmation objet a été appliqué de manière assez correcte. Le travail réalisé sur l'algorithme de Huffman est selon moi bien fait et a nécessité un certain travail. Je ressors de ce projet avec de nouvelles compétences en programmation et en mathématiques. L'interpolation bilinéaires, la convolution sont des traitements que je ne connaissais pas et qui m'ont beaucoup intéressé. J'ai aussi appris quelques rudiments des interfaces graphiques WPF en C#. Cette application graphique est assez ludique comparé à d'autres, même si j'aurai souhaité afficher mes images de manière dynamique dans ma fenêtre. En conclusion, je suis assez content de mon travail.