Rapport

Joyce LAPILUS – ESILV A2 TD C

Structure des données

Expliquer comment on a raisonné pour résoudre le projet…

J'ai décidé de séparer toutes les parties qui nécessitait un apport de travail assez conséquent en plusieurs classes. Au fur et à mesure, je me suis donc retrouvée avec **5 classes** :

- **MyImage** : celle autour de laquelle se centre le sujet, on peut y retrouver « grosso-modo » les mêmes fonctionnalités que la classe Bitmap
- **HeaderInfo** : qui s'occupe de toutes les manipulations à faire sur le header d'une image au format Bitmap
- **Complexe** : principalement utilisée lors de la création de fractales, cette classe gère les opérations des nombres complexes (multiplication, module, etc.)
- **QRCode** : gère les manipulations et opérations afin de générer des QR codes
- **Pixel** : classe qui gère les manipulations des pixels

Explication de l'innovation

Négatif

Figure 2 : Image "Coco" sans modification

Figure 1 : Image "Coco" avec l'effet "Négatif"

C'est une modification d'images qui n'a pas été demandée, mais que j'ai tout de même trouvé intéressant d'essayer, sachant que c'est un effet qui ne prend pas beaucoup de temps à comprendre ainsi qu'à appliquer.

Miroirs

Mon projet propose plusieurs façons de miroiter une image : selon l'axe X, l'axe Y ou les deux.

Le sujet n'explicitant pas quel type d'effet miroir était voulu, j'ai pensé à simplement tous les intégrer dans le projet.

Figure 5 : "Lena" miroitée selon l'axe X

Figure 4 : "Lena" miroitée selon l'axe Y

Figure 3 : "Lena" miroitée selon les deux axes

Plusieurs fractales

Lors du TD5, nous devions créer une image décrivant une fractale. Or, lors de mes recherches, j'ai découvert qu'il y avait une pléthore de fractales différentes, que ça soit par rapport aux formules utilisées ou à l'apparence finale. J'ai donc décidé de me focaliser sur les **fractales de Julia**, particulièrement l'**originale** (*Julia Fractal*) et la **Julia cubique** (*Cubic Julia Fractal*), que j'ai trouvé spécialement agréables à regarder.

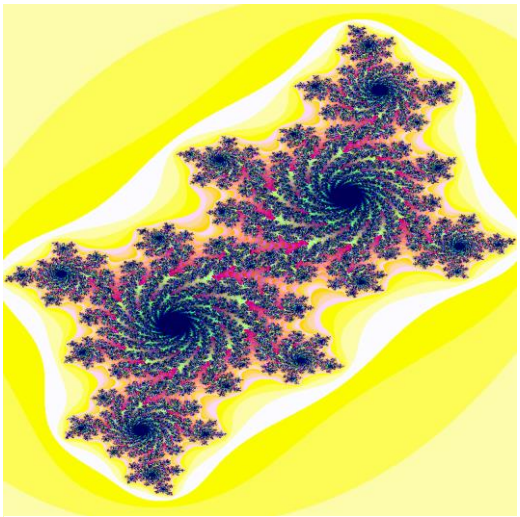

Figure 6 : Julia Fractal (c = - 0.4 - 0.59i)

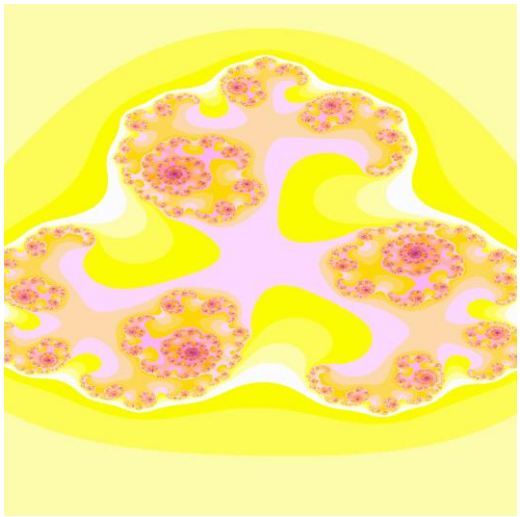

Figure 7 : Cubic Julia Fractal (c = - 0.5 - 0.05i)

De plus, cette innovation a été plus facilement réalisable grâce à mon initiative de créer une interface graphique pour accompagner mon projet.

Interface graphique

Après qu'on a eu notre CMo sur le QR Code et l'interface graphique, j'ai décidé d'entamer la création de la mienne car j'avais assez d'avance sur les TDs, donc j'avais le temps de me renseigner et de me consacrer à sa création sans prendre de retard sur le reste.

Cependant, l'interface reste assez simple ; il n'y a pas de page d'accueil, de séparation de fenêtres/d'onglets. Toutes les fonctionnalités se trouvent sur la même fenêtre, et il y a des fenêtres pop-up pour la génération de QR Code, l'encodage & le décodage d'image, ainsi que la génération d'histogrammes et de fractales.

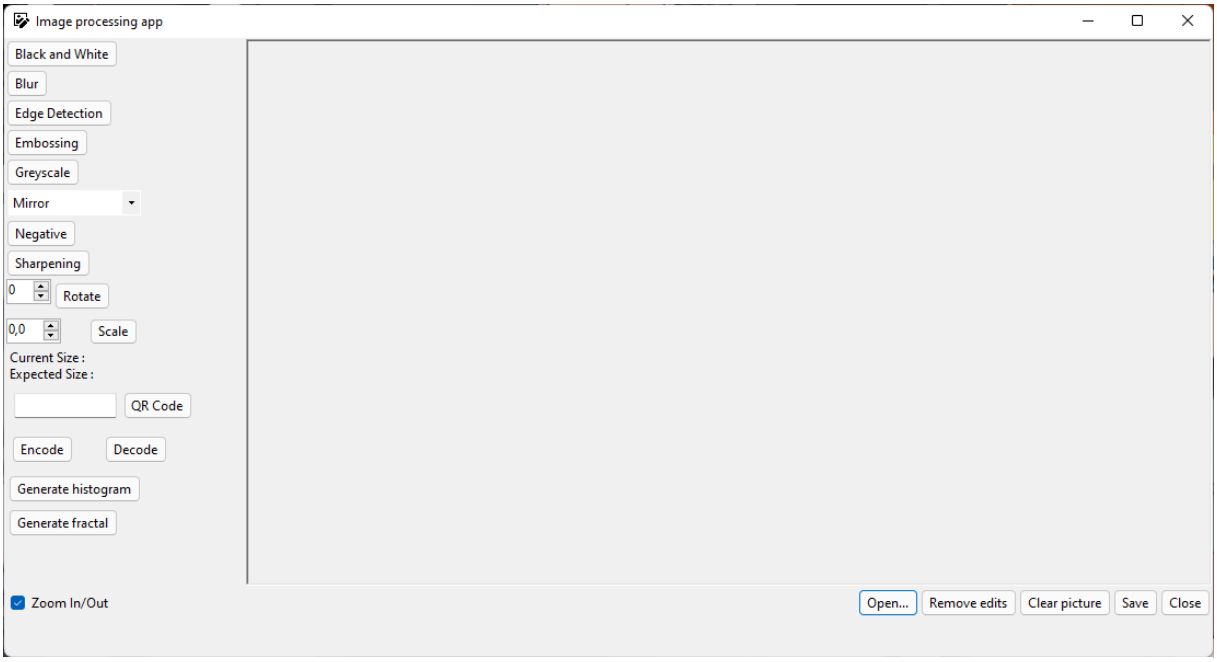

Figure 8 : Image de l'interface graphique

Après coup, j'ai essayé de rendre l'interface graphique aussi interactive que possible, en prenant en compte notre niveau débutant en programmation C#. *(Voir exemple ci-contre)*

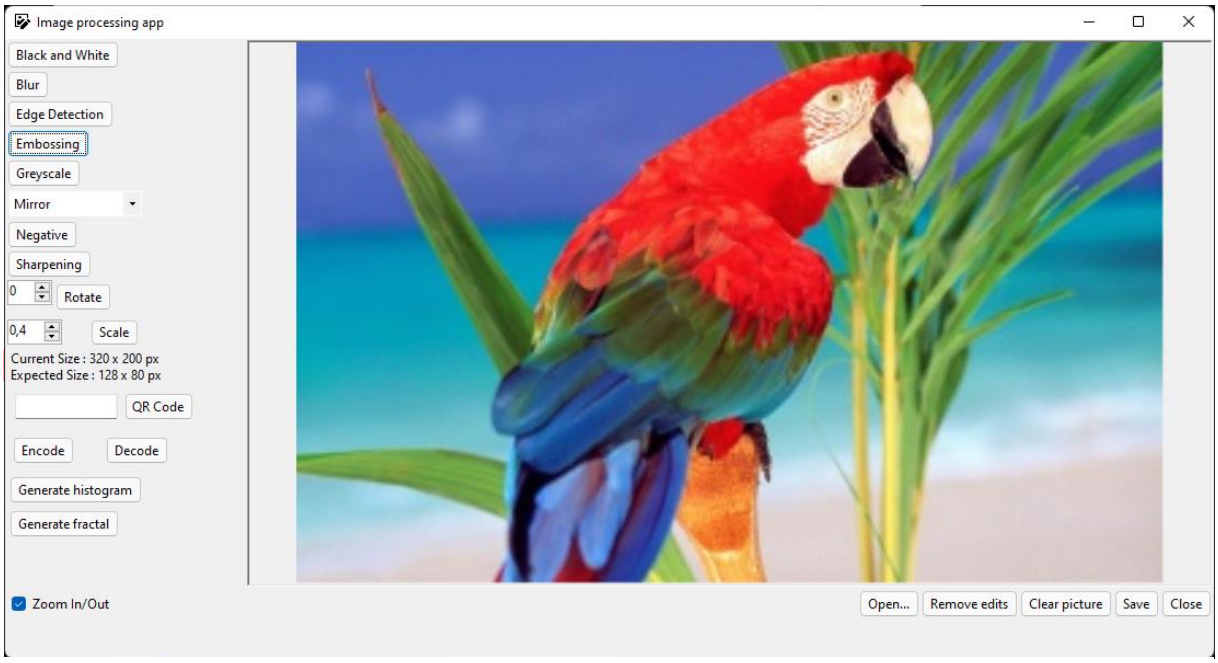

Figure 9 : Image de l'interface graphique (2)

La taille attendue après une modification (rotation ou agrandissement/rétrécissement de l'image) est écrite et change au fur et à mesure que l'on choisit une valeur, sans pour autant appliquer la modification voulue.

Après avoir essayé d'implémenter un bouton « précédent » pour annuler la modification la plus récente comme dans un logiciel de traitement d'image lambda, on peut finalement annuler toutes les modifications faites sur une image grâce au bouton « *Remove edits* » situé en bas à droite.

La seule fonctionnalité que je n'ai pas réussi à implémenter dans mon projet c'est la **lecture de QR Code** car je pensais avoir compris la méthode. Mais lors de l'application avec les codes ça ne fonctionnait pas, et je manquais de temps donc j'ai fini par ne pas faire le lecteur de QR Code.

FIN