MICHON Raphaël

MAURICE Edouard

TD K

Rapport Projet Scientifique Informatique

1. Innovation

Nous avons créé deux innovations pour notre projet.

1. Corrompre Image

Nous avons fait une fonction qui permet de corrompre l'image, plus particulièrement les couleurs de celle-ci. Certaines formes restent visibles mais toutes les couleurs deviennent "corrompues".

L'idée nous est venue en codant la sténographie (pour cacher puis révéler une image au sein d'une autre image). Pour arriver à ce résultat, nous échangeons les 4 premiers et derniers bits des bytes codant les couleurs. Ça permet de garder une logique dans l'image tout en modifiant fortement son contenu.



(Coco après avoir utilisé la fonction corrompre)

1. QR Code Version 3

Nous avons rajouté la possibilité d’encoder en QR Code des phrases faisant jusqu’à 77 charactères. Cet ajout peut se voir dans tout le processus de création de QR Code de notre programme. En effet, si une chaîne de charactère est assez longues, le nombre d’octet de correction généré par Reed-Solomon augmente et la taille nécessaire de la chaîne d’octet encodé prend pour valeur 55. Enfin, si cela augmente juste le nombre de boucles pour les méthodes d’encodage des données, nous faisons tout de même attention d’ajouter le *Reminder Bit* à la fin. Cependant, c’est dans la création de la structure du QR Code que le plus gros changement s’effectue. Mise à part pour des parties qui ne bouge jamais de coordonnées comme le *Finder Pattern* en bas à gauche, on change la position des autres fonctions, car la taille de la matrice est plus grande. Aussi, si notre entier version est différent de 0 (et donc qu’il ne s’agit pas de Version 0), alors on trace avec les autres fonctions l’*Alignement Pattern.* On s’assure d’ajuster avec un entier ses coordonnées selon la taille de la matrice, et donc de la version. Enfin, nous n’avons plus qu’à laisser les autres méthodes tourner correctement, elles ne dépendent pas de valeurs fixe et donc peuvent s’adapter à des matrices de toutes tailles.



(Un QR Code Version 3) 

1. Classes et attribues

Nous avons utilisé 2 classes dans notre programme.

1. Pixel :

La première classe est Pixel et prend comme attribue byte R, byte G et byte B.   
C'est simplement une classe composée de 3 bytes qui représentent un pixel de couleur pour simplifier le reste du code.

1. MyImage :

La seconde classe est MyImage et prend comme attribut string file.

 Elle ouvre une image en .bmp et la décompose en plusieurs variables ; les variables composant le header et un tableau de Pixels pour le corps de l'image.

Une image contenant texte

Description générée automatiquement

C'est aussi possible de créer un MyImage à partir d'un autre MyImage, dans ce cas les variables du header seront copiées et un nouveau tableau de Pixel vierge créé.

La classe contient des fonctions pour enregistrer l'image sur le PC, convertir int en endian, et mettre à jour les variables liées à la taille de l'image.

1. Problèmes rencontrés et auto-critique :
2. Problèmes rencontrés :

Les principaux problèmes rencontrés ont été de devoir se renseigner et comprendre comment fonctionnait les bitmaps et l’algorithme ReedSolomon. Également, pour la lecture de QR Code et la rotation, même si nous avions trouvé la théorie derrière le code, nous n’avons pas réussi à les mettre en place.

1. Auto critique :

Même s’il manque quelques fonctionnalités, nous avons réussi à implémenter un WPF qui permet de facilement utiliser ce que nous avons fait, notamment 3 versions de QR Code, ainsi que de facilement voir les résultats des modifications apportées aux images et de les combiner.