CLUET Arthur
COUDREAU Thomas
TD L

Problème scientifique informatique

Structures de données

Classe Img

La classe Img est la classe qui permet d'effectuer des manipulations sur les images. Elle comporte :

- Des champs qui contiennent les données à mettre dans l'en-tête (nombre de plans, bits par pixel, etc...)
- Des champs statiques qui contiennent les indices auxquels placer les données dans l'en-tête des fichiers Bitmap. Ces indices servent lorsque l'on remet les données de l'en-tête dans un tableau d'octets, lors de l'exportation de l'image.
- Un champ « pixels » qui contient les données de l'image. Il s'agit d'une matrice de Pixel (classe comprenant 3 champs R,G,B). Chaque instance de Pixel représente un pixel de l'image. Dans la matrice, les coordonnées (0,0) correspondent au pixel en haut à gauche de l'image.
- Les images peuvent être crées à partir d'un fichier, ou à partir d'une matrice d'entiers (voir partie suivante) et peuvent être exportées au format Bitmap.

Classe QRGenerator & Fractales

La classe QRGenerator permet de créer des codes QR à partir d'une chaîne de caractères alphanumériques. Elle comprend des champs statiques qui permettent de stocker certaines données indispensables à la création de codes QR (les finder pattern et pattern d'alignement sont stockées comme ça, sous forme de matrice).

Les codes QR sont générés sous forme de matrice d'entiers, remplie de 0 et de 1.

Certaines informations sont nécessaires pour créer des codes QR de version 1 à 40. Nous avons stocké ces informations dans un fichier (./versions_info/table.txt) que nous avons rempli grâce aux informations trouvées sur internet. Il est lu dans le constructeur de QRGenerator, classe qui doit donc être instanciée pour fonctionner.

Les fractales sont aussi générées sous formes de matrice d'entiers avec int[] Fractale.From(..) et la conversion en image est faite avec un constructeur de Img.

Innovations

- Nos classe permet de créer des codes QR de version 1 à 40 et accepte donc des chaînes jusqu'à 4296 caractères.
- Les codes QR de version 1 à 40 utilisant le masque 0, le niveau de correction L et le mode alphanumérique peuvent être lus.
- Les codes QR peuvent être lus à partir d'une image Bitmap qui a été redimensionnée. Les images sont acceptées si elles sont agrandies, si elles ont subi une déformation en rectangle (sans rotation !), ou si des bords blancs sont ajoutées autour.

Exemple d'image lisible par notre programme :

(fichier présent dans le dossier Debug)

- Nous avons fait une application WPF très basique pour générer des codes QR.
- La génération de fractales se fait à partir d'un paramètre de type Complex et elles peuvent être créées de très grande taille. Ainsi, nous avons ajouté une barre de progression pour permettre à l'utilisateur d'estimer son temps d'attente.

Problèmes rencontrés

- Notre lecteur de code QR acceptant les images agrandies, il n'était pas possible d'utiliser la taille de l'image pour connaître le nombre de modules du code QR, et donc la version utilisée. Il a fallu donc chercher la taille d'un module en parcourant l'image.
- La création de codes QR de toutes les tailles a nécessité des données trouvées sur internet, notamment les données de ce tableau :
  https://www.thonky.com/qr-code-tutorial/error-correction-table
  Nous les avons donc placées dans un fichier texte dont le contenu est expliqué dans un autre fichier readme.txt.
  La nécessité de lire ces données dans un fichier implique que la classe QRGenerator soit instanciée pour pouvoir générer des codes QR.
  Le fichier est lu dans le constructeur et les données sont placées dans un tableau de type QRVersion. Cette classe permet de stocker les informations de chaque version facilement, et permet de les dénommer plutôt que tout stocker dans un tableau. Exemple : Pour obtenir la capacité de la version 40, il suffit de faire `versions[40].Capacity`.
- *Nous avons également passé beaucoup de temps pour comprendre pourquoi notre code QR n'était pas lisible par un lecteur. Le cahier des charges ne précisait pas ce que signifiait « Masque 0 » et les modifications que ça impliquait pour les bits de données.*

Auto-critique et pistes d'amélioration

- Le lecteur de code QR n'utilise pas du tout les mots de corrections d'erreur alors qu'ils occupent une part importante des données dans le code QR.
- La fonction qui permet de redimensionner les images ne permet pas de gagner en qualité. L'agrandissement n'est donc pas utile, sauf pour les codes QR où ça fonctionne très bien. Nous avons essayé d'appliquer des filtres de convolution après un agrandissement mais les résultats n'étaient pas satisfaisants.
- Le lecteur de code QR à partir d'un fichier Bitmap fonctionne grâce à des règles bien précises. Si des pixels colorés se trouvent autour du code QR ou par-dessus, des problèmes de lecture risquent de se produire.