NFA032 – Compte Rendu et Mode d'Emploi

TinyMage, lecteur d'image PPM

Structure du projet :

Le programme se lance via la classe **TinyMage** qui contient la fonction main.
Celle-ci instancie la fenêtre principale du programme : **TinyMageCore**.

TinyMageCore étend le **Collector**, une classe abstraite qui joue un rôle de cloud, en collectant toutes les images générées, ce à fin des fonctions d'incrustation/superposition d'image.

Le **Collector** a pour cela sa propre fenêtre, le Mixer.

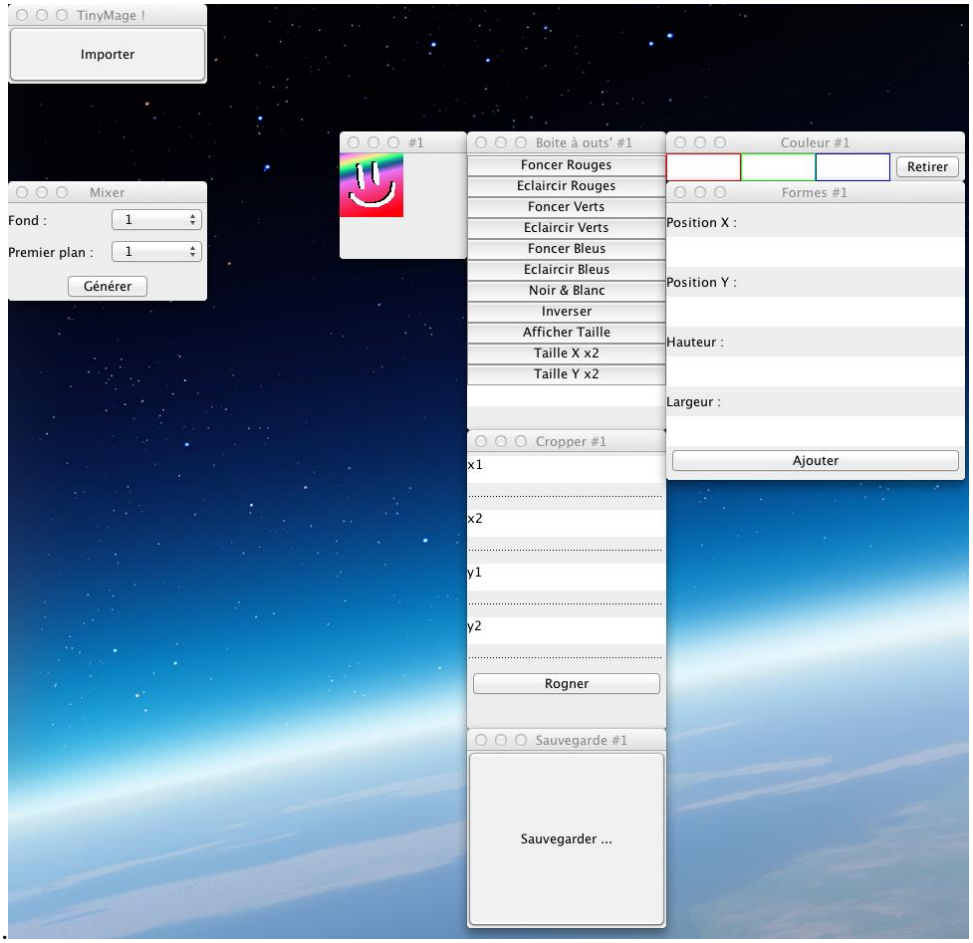
TinyMageCore n'a qu'une seule fenêtre, qui sert à l'importation des fichiers et à créer une image aléatoire *(attention, cette fonction a été rajoutée tardivement et n'a pas été testée)*.

En cliquant sur importer, et après avoir choisi un fichier, **TinyMageCore** s'occuper de parser le fichier choisi, et d'en générer des **Segments**. De ces segments seront générés un tableau de **Pixels** donnant naissance à une **PPMImage**. **Attention :** l'importation d'une image est une opération longue surtout avec de grands fichiers. Vous trouverez de petites images en 64x64 avec le projet qui conviennent pour un test rapide.

PPMImage est une sous-classe de la classe abstraite **AbstractImage**, tout comme **CompositeImage**, qui est la classe d'image générée par incrustation par le Mixer que nous verrons plus loin.
*Depuis une très récente mise à jour, il y a aussi **RandomImage**, qui permet de générer des images aléatoires, mais qui n'a pas encore été proprement testée.*

Lors de la création d'une **AbstractImage**, une **Toolbox** est automatiquement associée. La Toolbox est une boite à outils qui contient la quasi-totalité des fonctions demandées pour ce programme.

Elle est pour cela séparée en 5 fenêtres.



La **boite à outs'** contient toutes les fonctions sur les couleurs et la taille.
Le **Cropper** contient 4 champs (startX, endX, startY, endY) pour **redimensionner** l'image.

Sauvegarde permet de sauvegarder le fichier.

Couleur permet, en cliquant sur un pixel de l'image ou en entrant la valeur RGB suivi d'un clic sur le bouton **Retirer**, de marquer les pixels à ne pas représenter dans **l'incrustation** d'une image.
Formes permet de créer des **rectangles** sur l'image. Si aucune couleur n'est sélectionnée ou entrée, le rectangle sera rouge.

Certaines opérations créent une nouvelle image qui est ajoutée au **Collector**, d'autres se contentent de modifier l'image déjà créée.

Utilisation du Mixer pour les fonctions d'incrustation :

Importez une première image. Une fois l'image affichée, avec la souris, s**électionnez la couleur à retirer de l'image en cliquant dessus et en cliquant sur le bouton Retirer.**

Importez une deuxième image. **Sélectionnez** le numéro de l'image correspondant dans les menus déroulants correspondants du Mixer. L'image dont vous avez retiré des couleurs devrait être en premier plan, et votre deuxième image importée en fond.

Cliquez sur **Générer**.
Patientez. Et voilà !

Vous trouverez **deux fichiers pour réaliser cet exemple** dans le dossier du projet : tinymageIncrust.ppm pour le premier plan et tinymage.ppm pour le fond.

Attention : toutes les fonctions de la boite à outils ne sont pas disponibles pour les images Composite. Vous pouvez cependant les sauvegarder et les réimporter en image PPM et accéder à toutes les fonctions.

Historique et notes d'observation:

Merci à vous de prendre le temps de corriger ce travail.

Comme pour NFA035, ce projet a été réalisé en même temps que deux autres : TinyXL, un petit tableur que je vous ai déjà rendu, et MonsterMayhem, un jeu vidéo réalisé sans framework pour NFA019.

TinyMage a été le plus dur des trois et a subi de nombreux rebondissements.
Je n'avais pas saisi que nous n'avions pas le droit d'utiliser les Collections, ce qui m'a amené à recommencer le projet après avoir installé la plupart des fonctions principales, qui étaient le plus souvent implémentées à l'aide d'ArrayList. J'avais notamment, comme je vous l'avais dit, utilisé LinkedList pensant tout simplement que votre souhait de liste chaînée correspondait à cette Collection.
J'ai donc recommencé le projet avec une vraie liste chainée de Segments, même si celle-ci devient assez vite un tableau de Pixels, beaucoup plus pratiques pour la manipulation des images (à cause des occurrences). J'ai cependant réalisé tous les exercices sans problème lorsque nous étudions le sujet.

Vous ne trouverez pas de fonctions récursives dans ce projet, pour la simple et bonne raison que sur la plupart des endroits où cela me paraissait utile, je me retrouvais avec des StackOverflowError. Ceux-ci disparaissant avec l'utilisation du debugger, je ne pense pas qu'ils résultaient d'une mauvaise écriture, mais plutôt d'une réalisation excessive de fonctions simples, sur de gros fichiers, l'exécution se faisant trop vite et faisant planter la JVM.
Vous trouverez cependant une fonction récursive dans le projet de NFA035, TinyXL (Cell.resolveFormula) qui devrait montrer que cela ne me pose pas de problème.

Mise à jour : *vous trouverez une fonction récursive dans AbstractImage : recursiveWriteToString()*

Elle est utilisée dans la classe de test correspondante.

La partie la plus dure à écrire dans mon souvenir est le parseur, sans utiliser de tableau. Les fichiers exemples n'étant pas tous écrits de la même manière, il a fallu envisager plein de cas différents et au final lire les fichiers caractère par caractère. Cela a donné naissance à une classe abstraite **Value** qui consiste en un entier et une booléenne, qui repère quand la valeur est remplie et peut passer à la

suivante. Trois classes l'étendent (**Red, Green, Blue**) ce qui ne m'a pas servi à grand-chose à part pour le débuggage et me repérer dans l'écriture du parseur.

Voilà je vous souhaite un bon été et à bientôt peut-être !